IMS
バージョン 13

データベース管理

SA88-5485-04

IMS
バージョン **13**

データベース管理

SA88-5485-04

お願い

本書および本書で紹介する製品をご使用になる前に、 973 ページの『特記事項』に記載されている情報をお読みください。

本書は、IMS バージョン 13 (プログラム番号 5635-A04)、IMS Database Value Unit Edition V13.1 (プログラム番号 5655-DSM)、IMS Transaction Manager Value Unit Edition V13.1 (プログラム番号 5655-TM2)、および新しい版で明記されていない限り、以降のすべてのリソースおよびモディフィケーションに適用されます。

お客様の環境によっては、資料中の円記号がバックスラッシュと表示されたり、バックスラッシュが円記号と表示されたりする場合があります。

原典： SC19-3652-04
IMS
Version 13
Database Administration
(November 6, 2017 edition)

発行： 日本アイ・ビー・エム株式会社

担当： トランスレーション・サービス・センター

目次

本書について

これらのトピックでは、IMS™ データベースのタイプと概念について説明し、 IMS データベースの設計、実装、保守、変更、バックアップ、およびリカバリーの方法についても説明しています。

この情報は、IBM® Knowledge Center で参照できます。

前提知識

本書をご使用になる際には、z/OS® と IMS の基本概念、およびインストールされている IMS システムについて理解しておく必要があります。IMS は、DB バッチ、DCCTL、 TM バッチ、DB/DC、DBCTL の各環境で稼働できます。ご使用のシステムに適用される環境についても知っておく必要があります。ここで説明されている IMS の概念は、IMS データベースの管理に関連するものだけです。DL/I 呼び出しおよびアセンブラー、COBOL、PL/I、C などの言語の使用法も知っている必要があります。

z/OS の詳細については、IBM Knowledge Center の「z/OS basic skills」トピックを参照してください。

IMS の基本概念を理解するには、「*An Introduction to IMS*」(IBM Press 出版) をお読みになると役立ちます。

IBM では、IMS の学習に役立つような講習会や自習講座を数多く提供しています。利用可能な講習の詳しいリストについては、IBM Skills Gateway にアクセスして、IMS を検索してください。

本書で使用される**IMS** 機能名

本書では、「HALDB オンライン再編成」という用語は、特に断りがない限り、IMS バージョン 13 の一部として組み込まれた HALDB オンライン再編成機能を指しています。

新規および変更された情報の識別方法

IMS ライブラリーの PDF 資料のほとんどの新規および変更された情報は、左マージン内の文字 (改訂マーカー) によって示されています。「リリース計画」、ならびに「*Program Directory*」および「*Licensed Program Specifications*」の第 1 版 (-00) には、改訂マーカーは含まれていません。

改訂マーカーは、以下の一般的な規則に従っています。

- 技術的な変更のみにマークが付けられています。形式上の変更や文法的な変更には、マークは付けられていません。

- 段落、構文図、リスト項目、操作手順、または図などの要素の一部が変更された場合、その要素の一部だけの変更であっても、要素全体に改訂マーカーが付けられています。
- トピックの変更が 50% を超えた場合には、そのトピック全体に改訂マーカーが付けられています (そのため、新規トピックではなくても、新規トピックのように見えることがあります)。

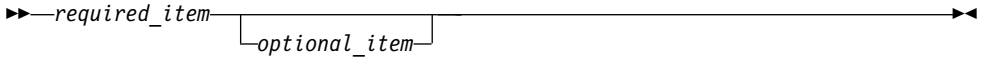
改訂マーカーは情報に加えられたすべての変更を示しているとは限りません。削除されたテキストとグラフィックスには、改訂マーカーでマークを付けることはできないためです。

構文図の読み方

本書で使用されている構文図には、以下の規則が適用されています。

- 構文図は、経路を示す線に沿って、左から右、上から下に読み取ります。以下の規則が使用されます。
  - >>--- 記号は、構文図の始まりを示します。
  - ---> 記号は、構文図が次の行に続くことを示します。
  - >--- 記号は、この構文図が直前の行から続いていることを示します。
  - --->< 記号は、構文図の終わりを示します。
- 必須項目は、水平線 (メインパス) 上に表示されます。

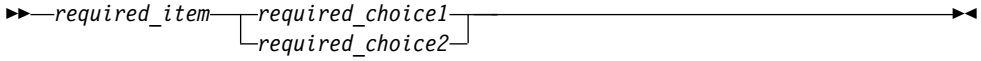
- オプション項目は、メインパスより下に示されます。



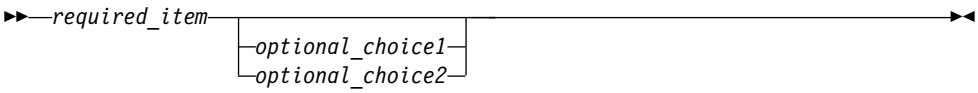
メインパスより上にオプショナル項目が示されている場合は、その項目が構文エレメントの実行に影響することはなく、読みやすくするためのみの表記です。

- 複数の項目から選択できる場合は、縦方向に並べて (スタック) 示されます。

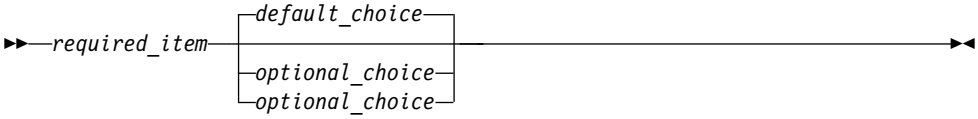
  それらの項目の中から 1 つを選択する必要がある 場合は、スタックの中の 1 つの項目がメインパス上に表示されます。



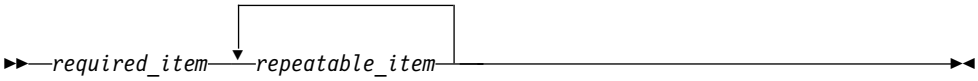
それらの項目から 1 つを選択することがオプションである場合は、スタック全体がメインパスの下に表示されます。



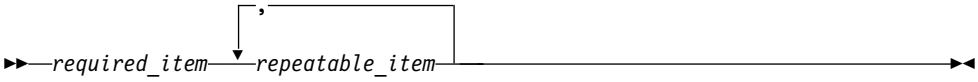
デフォルト項目が含まれている場合、その項目はメインパスより上に示され、他の選択項目はメインパスより下に示されます。



- メインパスの上方にある左に戻る矢印線は、項目が反復可能であることを示します。

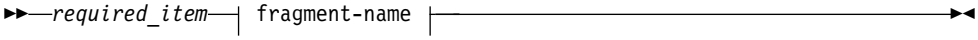


反復矢印線にコンマが含まれている場合は、反復項目をコンマで区切る必要があります。



スタック上方の反復矢印線は、スタック内の項目を反復できることを示しています。

- 1 つの構文図を複数のフラグメントに分割しなければならない場合もあります。構文フラグメントはメインの構文図とは別に示されますが、フラグメントの内容は、図のメインパス上にあるものとして読む必要があります。



fragment-name:

- IMS では、b 記号は、該当位置にブランクが 1 つあることを示します。
- キーワード、および該当する場合はキーワードの最小の省略語は、大文字で表されます。これらは、示されているとおりに入力する必要があります。変数は、すべて小文字のイタリック文字で示されます (例えば、*column-name*)。これらは、ユーザーが指定する名前または値を表します。
- キーワードとパラメーターは、構文図で間に句読点が示されていない場合は、少なくとも 1 つのスペースで分離します。
- 句読記号、括弧、算術演算子、およびその他の記号は、構文図で示されたとおりに入力します。

- 脚注は、例えば (1) のように、数字を括弧で囲んで示してあります。

IMS バージョン 13 のアクセシビリティー機能

アクセシビリティー機能は、運動障害または視覚障害など身体に障害を持つユーザーが情報技術製品を快適に使用できるようにサポートします。

アクセシビリティー機能

以下のリストは、 IMS バージョン 13 を含む z/OS 製品の主なアクセシビリティー機能を示しています。これらの機能は、以下をサポートしています。

- キーボードのみの操作。
- スクリーン・リーダー (読み上げソフトウェア) およびスクリーン拡大鏡によって通常使用されるインターフェース。
- 色、コントラスト、フォント・サイズなど表示属性のカスタマイズ。

キーボード・ナビゲーション

IMS バージョン 13 ISPF パネル機能には、キーボードまたはキーボード・ショートカット・キーを使用してアクセスできます。

TSO/E または ISPF を使用して IMS バージョン 13 ISPF パネルをナビゲートする詳細については、「*z/OS TSO/E* 入門」、「*z/OS TSO/E* ユーザーズ・ガイド」、および「*z/OS* 対話式システム生産性向上機能 *(ISPF)* ユーザーズ・ガイド 第 *1* 巻」を参照してください。上記の資料には、キーボード・ショートカットまたはファンクション・キー (PF キー) の使用方法を含む、各インターフェースのナビゲート方法が記載されています。それぞれの資料では、PF キーのデフォルトの設定値とそれらの機能の変更方法についても説明しています。

関連のアクセシビリティー情報

IMS バージョン 13 のオンライン資料は、IBM Knowledge Center で参照できます。

IBM におけるアクセシビリティー

IBM のアクセシビリティーに対する取り組みについて詳しくは、*IBM Human Ability and Accessibility Center* (www.ibm.com/able) を参照してください。

第 1 部 データベースの一般的な概念、標準、および手順

以下のトピックでは、IMS Database Managerを使用したデータベースの管理の概要 (IMS データベースについての一般的な説明、IMS データベース処理時に使用される基本的な標準と手順、基本的な設計とレビューのプロセス、およびデータベース・セキュリティーを含む) を示します。

第 1 章 IMS データベースの概要

IMS データベースの概要では、データベース管理の作業について述べ、IMS Database Manager を管理する際に使用される主要な概念と用語について説明します。

データベース管理の概要

データベース管理の作業とは、データベースの設計、インプリメント、および保守を行うことです。

本書では、Information Management System Database Manager (IMS DB) の管理に関係する作業について説明します。IMS は、IMS Database Manager と IMS Transaction Manager の 2 つの部分で構成されています。 IMS Database Manager は、データベース内のレコードの物理ストレージを管理します。 IMS Transaction Manager は、端末ネットワーク、メッセージの入出力、およびオンライン・システム・リソースを管理します。 IMS Transaction Manager の管理については、「*IMS V13* システム管理」で説明しています。IMS のネットワーキングについては、「*IMS V13* コミュニケーションおよびコネクション」で説明しています。

本書では、データベース管理作業について、それらの作業を通常実行する順序で説明します。データベース構築過程では、ある特定の順序で行う作業があります。その他の作業は継続して行われます。作業がどのようなものであるかを把握するだけでなく、個々の作業がどのように相互に関連しているかも把握することが大切です。

本書の第 1 部では、データベース管理プロセス全体の重要な概念と手順を説明します。第 2 部には、データベース管理の個々の作業に対応する章があります。

関連概念:

5 ページの『データベース管理作業』

DL/I

データ言語 /I (DL/I) は、IMS データ操作言語で、ユーザー・アプリケーションと IMS との間の共通高水準インターフェースです。

DL/I 呼び出しは、PL/I、 COBOL、VS Pascal、C、Ada などの言語で書かれたアプリケーション・プログラムから呼び出されます。また、サブルーチン呼び出しによって、アセンブラー言語アプリケーション・プログラムから呼び出すこともできます。IMS を使用すると、ユーザーはデータ構造の定義、構造とアプリケーションとの関連付け、構造のロード、および構造の再編成を行うことができます。

関連概念:

IMS DB のアプリケーション・プログラミング (アプリケーション・プログラミング)

関連資料:

データベース管理 (アプリケーション・プログラミング API)

CICS

顧客情報管理システム (CICS®) は、データベース・リソース・アダプター (DRA) を介して IMS データベースにアクセスします。

CICS または他のトランザクション管理サブシステム (IMS トランザクション・マネージャーを除く) は、DB/DC 環境または DBCTL 環境で DRA を介して、IMS の全機能データベースと高速処理データベース (DEDB) にアクセスすることができます。

顧客情報管理システム CICS のユーザー向けに、作業が異なっている場合、その違いについて簡単に説明を加えています。

DBCTL および DCCTL

データベース制御 (DBCTL) は、非メッセージ・ドリブン・バッチ・メッセージ処理 (BMP) プログラムをサポートします。データ通信管理 (DCCTL) は、全機能 DEDB および MSDB (主記憶データベース) はサポートしないが、BMP 領域で GSAM データベースをサポートするトランザクション管理サブシステムです。

DBCTL は、独自のログを持っていて、データベースのリカバリーに参加します。ロックは、IMS プログラム分離機能 (PI) または内部リソース・ロック・マネージャー (IRLM) によって提供されます。

DCCTL 環境でデータベースにアクセスするには、DCCTL はデータベースをサポートする外部サブシステムに接続する必要があります。

オープン・データベース・アクセス (ODBA)

z/OS アドレス・スペース内で稼働しているどのプログラムも、オープン・データベース・アクセス (ODBA) 呼び出し可能インターフェースを通じて、IMS DB にアクセスすることができます。

z/OS リソース・リカバリー・サービス (RRS) によって管理されている z/OS アドレス・スペースで稼働中の z/OS アプリケーション・プログラムはいずれも、IMS の全機能データベースおよび高速処理データベース (DEDB) にアクセスできます。ODBA インターフェースを使用する z/OS アプリケーション・プログラムは、ODBA アプリケーションと呼ばれます。

IMS の観点からすれば、関係する z/OS アドレス・スペースは、z/OS アプリケーション領域と呼ばれる別の領域に見えます。

ODBA インターフェースを呼び出せるプログラムのタイプには、次のものがあります。

- Db2® for z/OS ストアード・プロシージャー (COBOL、PL/I、および Java™ プロシージャーを含む)
- WebSphere® Application Server for z/OS で稼動する Enterprise Java Bean

- その他の z/OS アプリケーション

関連タスク:

ODBA インターフェースを介した IMS データベースへのアクセス (コミュニケーションおよびコネクション)

データベース管理作業

このトピックでは、IMS データベースに関するデータベース管理作業を列挙します。

設計のレビューへの参加
: 設計のレビューとは、データベース管理者と関係者が一堂に会して、データベースの設計とインプリメンテーションについて検討する一連の正式会議です。設計のレビューは、データベース・システムの設計とインプリメンテーションの段階で、継続して行われます。新しいアプリケーションを既存のシステムに追加するときにも、設計のレビューが行われます。

データ要件の分析
: ご使用のシステムのユーザーが各データ処理要件を確認し終わった後で、データベース管理者がデータ構造を構成します。データ構造は、どのようなデータがデータベースに保管されるか、また、そのデータがどのように編成されるかを示します。この作業は、データベースの実際の設計の前準備です。

データベースの設計
: データ構造が決まると、次の段階はデータベースの設計です。データベース設計には、以下の作業が含まれます。

  - データの物理的な編成方法の選定
  - 使用する必要のある IMS 処理オプションの決定
  - データベースの性能と使用可能スペースの使用効率を左右する設計上の一連の決定

テスト・データベースの構築
: データベースを使用するアプリケーションを本稼働に移す前に、そのアプリケーションをテストすべきです。既存のデータの形に応じて、 IMS データベース設計エイドの中から 1 つ以上のプログラムを使用して、自分のテスト・データベースを設計し、作成し、ロードし、テストすることができます。

データベース設計のインプリメント
: データベースの設計が終わった後で、このデータベースの特性とアプリケーション・プログラムがこのデータベースをどのように使用するかを、IMS に対して記述することによりこの設計をインプリメントします。

  この作業は、データベース記述 (DBD) とプログラム仕様ブロック (PSB) をコーディングすることから成り立ちます。DBD と PSB は一連のマクロ・ステートメントです。 また、データベース設計のインプリメントでは、データベースのアプリケーション制御ブロック (ACB) を事前に作成するのかあるいは動的に作成するのかも決定します。

データベースのロード
: データベースの特性を定義した後で、初期ロード・プログラムを作成し、ユ

ーザーのデータをデータベースにロードします。データベースをロードすると、このデータベースを対象とするアプリケーション・プログラムを実行することができます。

データベースのモニター
: データベースが稼働しているときに、このデータベースのパフォーマンスを定常的にモニターしてください。 IMS システムをモニターするための多様なツールが用意されています。

データベースのチューニング
: パフォーマンスが低下する場合、あるいは外部ストレージの使用率が最適でないときには、データベースのチューニングをしてください。定常的にモニターを行うと、システムのチューニングが必要な時期や必要なチューニングのタイプを判別するのに役立ちます。モニターと同様に、データベースのチューニングも継続的な作業です。

データベースの変更
: 新しいアプリケーションが開発されたり、ユーザーのニーズが変化したりするにつれて、ユーザーのデータベースに変更を加える必要が生じることがあります。例えば、データベース編成、データベース階層 (またはデータベース階層内のセグメントやフィールド) を変更したり、1 つ以上の区画を追加あるいは削除することができます。モニターやチューニングと同様、データベースの変更も継続的な作業です。

データベースのリカバリー
: データベースのリカバリーとは、なんらかの障害でデータベースが無効になったあと、このデータベースを元の状態に復元することです。リカバリー手順を開発し、リカバリーを実行する作業は重要な作業です。しかし、データのリカバリーとシステムのリカバリーを切り離すのは難しいので、リカバリー作業については「*IMS V13* オペレーションおよびオートメーション」で別途説明しています。

  データベース・リカバリー管理機能 (DBRC) を使用して、データベースのリカバリーをサポートすることができます。データベースが RECON データ・セットに登録されていると、以下の IMS ユーティリティーを実行している間は、DBRC が制御を獲得します。

  - データベース・イメージ・コピー
  - オンライン・データベース・イメージ・コピー
  - データベース・イメージ・コピー 2
  - 変更累積
  - データベース・リカバリー
  - ログのリカバリー
  - ログのアーカイブ
  - DEDB エリア・データ・セットの作成
  - HD および HISAM の再編成のアンロードと再ロード
  - HALDB 索引/ILDS 再作成

  すべてのデータベース・リカバリーでは、旧リリースのユーティリティーではなく、必ず現行 IMS のユーティリティーを使用するようにしてください。

セキュリティーの確立
: プログラム連絡ブロック (PCB) を使用することにより、許可されていない人がデータベースの中のデータにアクセスするのを防止することができます。 PCB により、データベース管理者は、ある一人のユーザーがこのデータベースのどれだけの部分を見ることができるかを指定することができ、またこのデータに対してどのような操作を行えるかを指定することができます。さらに、非 IMS プログラムがデータベースにアクセスするのを防止するためのステップを実行することができます。

標準と手順の設定
: アプリケーションおよびデータベース開発のための標準と手順を設定することは大切です。このことは、複数アプリケーションの環境においては特に当てはまります。指針や標準が決められていると、アプリケーション開発において時間が節約され、命名規則やプログラミング標準の不整合のような問題が後に生じないようにすることができます。

関連概念:

3 ページの『データベース管理の概要』

関連資料:

- ログ保存ユーティリティー (DFSUARC0) (システム・ユーティリティー)
- ログ・リカバリー・ユーティリティー (DFSULTR0) (システム・ユーティリティー)

データベース概念と用語

このトピックでは、IMS データベース管理作業を実行する際に理解しておく必要のある概念と用語について説明します。

このトピックを理解するためには、DL/I 呼び出しとは何か、およびそれをコーディングする方法について知る必要があります。 DL/I 呼び出しにおける機能コードとセグメント検索引数 (SSA) を理解するとともに、修飾呼び出しまたは非修飾呼び出し (*IMS V13* アプリケーション・プログラミング に説明がある) としての呼び出しの意味を知っておく必要があります。

データベースへのデータの保管方法

データベースの中のデータは、一連のデータベース・レコード に分けられています。各データベース・レコードは、セグメント と呼ばれる小さいデータ群から構成されています。セグメントは、IMS が保管できる最小のデータ単位です。セグメントは、1 つ以上のフィールド で構成されています。

以下の図は学校用データベース・レコードを示しています。長方形の枠は、それぞれ 1 つのセグメント、すなわちデータベース・レコードの中で分離されたデータのグループです。このデータベース・レコードの中のセグメントには、以下の情報が入っています。

COURSE
: 科目名

INSTR
その科目の担当教師名

REPORT
その科目の終了時に教師が要求する報告書

STUDENT
その科目を選択した学生名

GRADE
学生の科目成績

PLACE
その科目を教えるのに使用される教室

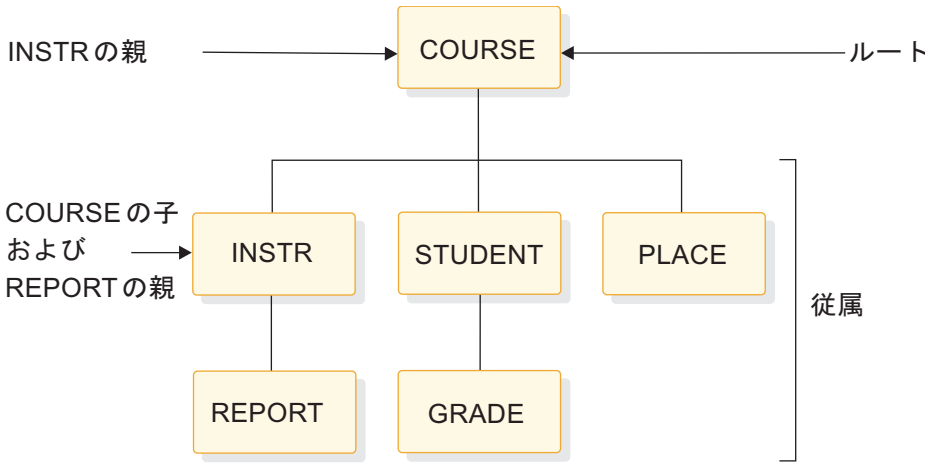


図 *1.* 学校用データベース・レコードの中のセグメント・タイプ

データベース・レコード内のセグメントは、いわゆる階層 の中に存在しています。階層とは、セグメントが並べられている順序です。この順序は何らかの意味を持っています。このデータベースにおいては、講習科目についてのデータが保管されています。 COURSE セグメントはこの階層の最上位にあります。 COURSE が存在しなければ、このデータベース・レコード内のセグメントの中の他のタイプのデータは無意味となります。

ルート・セグメント

1 つのデータベース・レコード中に存在できるルート・セグメント は 1 つだけです。データベース・レコード内のその他のセグメントは、従属セグメント と呼ばれます。

7 ページの『データベースへのデータの保管方法』に示す例では、COURSE セグメントがルート・セグメントとなっています。セグメント INSTR、REPORT、STUDENT、GRADE、および PLACE は従属セグメントです。従属セグメントの存在は、ルート・セグメントの存在に依存しています。例えば、ルート・セグメントの COURSE が存在しなければ、どの教室で科目が教えられるのかを示している PLACE セグメントが存在する理由はありません。

3 番目のレベルの従属セグメント REPORT と GRADE は、2 番目のレベルのセグメント INSTR および STUDENT の存在に依存しています。例えば、2 番目のレベルのセグメント STUDENT がなければ、この学生のこの科目の成績を示す GRADE セグメントの存在理由はありません。

親セグメントと子セグメント

階層においてセグメントが相互にどのような関係にあるのかを示すのに用いられるもう 1 組の用語が親セグメント と子セグメント です。親セグメントとは、階層においてすぐ下に従属セグメントを持つ任意のセグメントです。

7 ページの『データベースへのデータの保管方法』に示す図では、COURSE が INSTR の親であり、INSTR が REPORT の親となっています。子セグメントとは、階層においてそれより上にある他のセグメントに従属している任意のセグメントです。 REPORT は INSTR の子であり、 INSTR は COURSE の子になります。 INSTR は、REPORT との関係においては親セグメントであり、COURSE との関係では子セグメントである点に注意してください。

セグメント・タイプとセグメント・オカレンス

セグメント・タイプ およびセグメント・オカレンス という用語は、データベース内のセグメントのタイプと特定のセグメント・インスタンスを区別します。

これは、セグメント間の関係 を表す用語 (ルート、従属、親、および子) とは異なります。

7 ページの『データベースへのデータの保管方法』に示すデータベースは、実際にはデータベースの設計です。これは、このデータベースのセグメント・タイプを示しています。 『セグメント間の関係』 は、実際のデータベース・レコードとセグメント・オカレンスを示しています。

セグメント・オカレンスは、単一の、ある特定のセグメントです。 Math は COURSE セグメント・タイプの 1 つのオカレンスです。 Baker と Coe は、STUDENT というセグメント・タイプを持つ複数のオカレンスです。

セグメント間の関係

セグメントについて説明する最後の用語の 1 つが、兄弟セグメント です。兄弟とは、(ルート、従属、親、および子と同様)、セグメントとセグメントの間の関係を表します。兄弟セグメントとは、ある 1 つの親の下にある同一のセグメント・タイプの複数のオカレンスです。

以下の図では、セグメント Baker と Coe が兄弟です。これらは、同じ親 (Math) を持ち、同じセグメント・タイプ (STUDENT) です。 Pass と Inc は兄弟ではありません。 Pass と Inc は同じセグメント・タイプ (GRADE) ですが、親が同じではないからです。 Pass は Baker の子セグメントであり、Inc は Coe の子セグメントです。

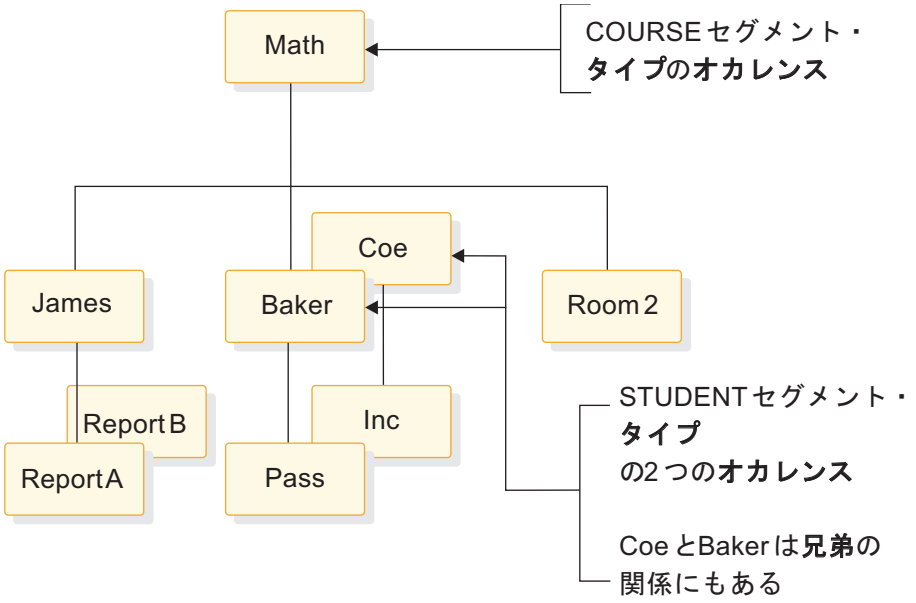


図 2. 学校用データベース・レコードの中のセグメント・オカレンス

次のトピックでは、階層についてさらに詳細に説明します。後続のトピックでは、データベースの中のオブジェクトがどのようなものから構成されており、どのような規則がオブジェクトの存在や使用の際に適用されるかを説明します。これらのオブジェクトについて説明します。

- データベース・レコード
- データベース・レコード内のセグメント
- セグメントの中のフィールド

データベース・レコード内の階層

これまでに見てきたのは、データベースは一連のデータベース・レコードより成り立っていること、データベース・レコードにはセグメントが入っていること、およびデータベース・レコードの中でのセグメントの配置は階層を構成していることです。

階層内での番号付け順序 - 上から下へ

データベース・レコードがデータベースに保管される場合、このデータベース・レコードでのセグメントの階層配置は、セグメントが保管される順序です。

セグメントがデータベースの中に保管される順序は、データベース・レコードの最上位 (ルート・セグメント) から始まり、以下の図の番号で示されている順序です。

この順序は階層の頂上から始まり、階層の最初の (一番左の) パス、すなわち、脚 (縦列) の一番下まで続きます。データベースの一番下に達すると、この順序は左から右となります。階層のこのパスに属するすべてのセグメントが保管されてしまうと、この順序は右隣のパスに続き、ここでも頂上から最下位へ、次いで左から右へと進みます。 (その階層の 2 番目の脚では、右に進むべきものは何もありません。) セグメントが保管される順序は、大まかに、「上から下へ、左から右へ」と呼ばれています。

以下の図は、 7 ページの『データベースへのデータの保管方法』 の学校用データベースのセグメント・タイプの順序付けを示しています。一連のセグメント・タイプは以下の順序で保管されます。

1. COURSE (上から下へ)
2. INSTR
3. REPORT
4. STUDENT (左から右へ)
5. GRADE (上から下へ)
6. PLACE (左から右へ)

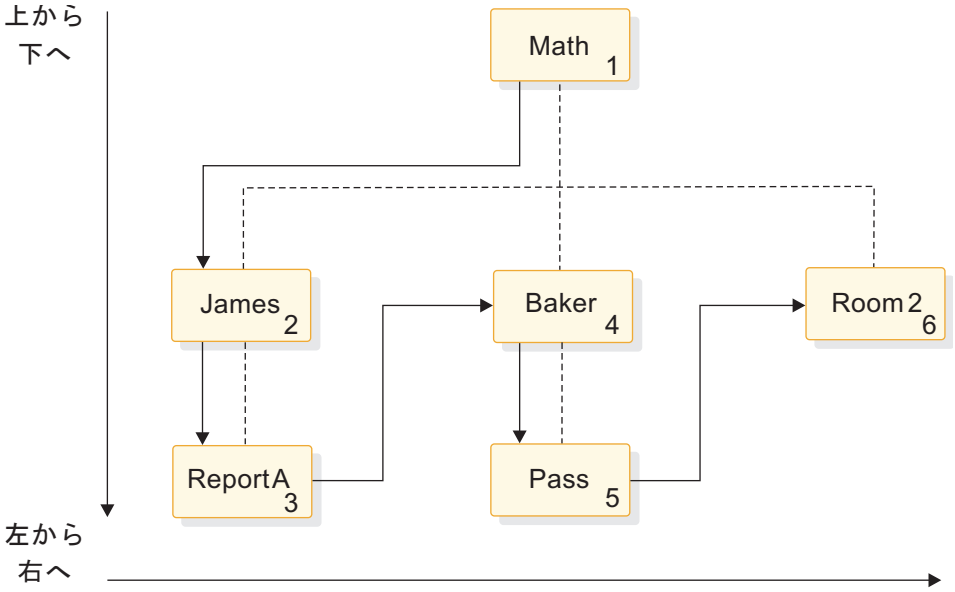


図 3. 学校用データベースのセグメント・タイプの階層順

以下の図は、 9 ページの『セグメント間の関係』 の学校用データベース・レコードのセグメント・オカレンスを示しています。セグメント・タイプの複数のオカレンスがあるため、セグメントは「上から下へ、左から右へ」に加えて「前から後ろへ」読み取られます。学校用データベースのセグメント・オカレンスは以下の順序で保管されます。

1. Math (上から下へ)
2. James
3. ReportA
4. ReportB (前から後ろへ)
5. Baker (左から右へ)
6. Pass (上から下へ)
7. Coe (前から後ろへ)
8. Inc (上から下へ)
9. Room2 (左から右へ)

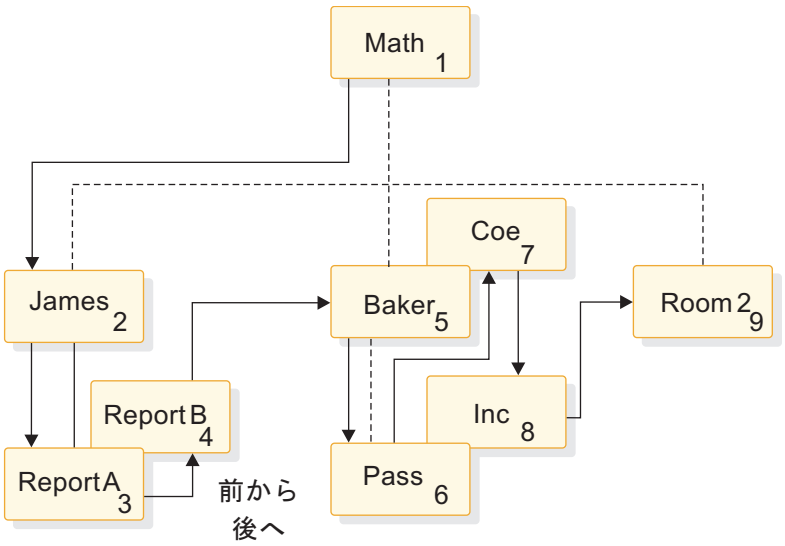


図 4. 学校用データベースのセグメント・オカレンスの階層順

番号付けの順序が、最初は依然として上から下へという順序になっていることに注意してください。しかし、この階層の最下位には REPORT セグメントの 2 つのオカレンスがあることに注目してください。

これは、この階層の最下位なので、これら 2 つのセグメント・オカレンスを取り上げた後で、この階層のこのパスの右側に移ります。この両方の報告書は教師セグメント James と関係をもつものであり、したがって、この 2 つを一緒にデータベースの中に保管することには意味があります。この階層の 2 番目のパスにも 2 つのセグメント・オカレンスがありますが、今度はこの 2 つは学生セグメントのオカレンスです。しかし成績セグメント Pass に達するまではこの階層パスの最下位に着きません。したがって、学生セグメント Baker と Coe という 2 つのオカレンスによって順序付けが「中断」されることはありません。このことは意味のあることです。というのは、学生 Baker と成績 Pass を一緒にしておこうとするからです。

学生 Coe の下の成績 Inc は、Baker の下の別のオカレンスであるとは見なされないという点に注意してください。 Coe と Inc はこの階層の中では別個の 1 つのパスとなります。階層パスの最下位に達したときにのみ、「上から下、左から右」の順序付けが中断され、複数のセグメント・オカレンスが取り上げられます。階層の中の順序付けは実際には「上から下、前から後ろ、左から右」であるとすることができますが、階層の最下位においてのみ「前から後ろへ」の順序付けが発生します。これ以外のレベルにおけるある 1 つのセグメントの複数オカレンスの順序は、階層の中の別個のパスとしての順序になります。

前述したように、セグメントのこの番号付けは、セグメントがデータベースに保管される順序を表しています。アプリケーション・プログラムにおいて、あるデータベース・レコードの中のすべてのセグメントを階層順に要求した場合、すなわち Get-Next (GN) 呼び出しを出した場合、この順序に従ってこれらのセグメントがこのアプリケーション・プログラムに提供されることになります。

階層内での番号付け順序 - 移動と位置

移動と位置という用語が使用されるのは、アプリケーション・プログラムから呼び出しが出されたとき、どのようにしてセグメントにアクセスするかについて話をするような場合などです。これらの用語は、階層での番号付け順序を表すために使用されます。

階層の中の移動について話すときは、いつでもこれは番号付けの方式によって表される順序での移動を意味しています。移動には順方向への移動と逆方向への移動とがあります。階層の中の位置について言っているときには、ある特定のセグメントのところにある (位置している) ことを述べているのです。

セグメントは、IMS が保管できる最小のデータ単位です。あるアプリケーション・プログラムから学生セグメント BAKER を対象とする Get-Unique (GU) 呼び出しが出されると (12 ページの図 4 参照)、現在の位置は BAKER セグメント・オカレンスのすぐ後ろになります。このあと、アプリケーション・プログラムから非修飾 GN 呼び出しが出されると、IMS はデータベース内を順方向に移動して、PASS セグメント・オカレンスを戻します。

階層内での番号付け順序 - レベル

階層でのレベル とは、ルート・セグメントを基準とした階層でのセグメントの位置です。ルート・セグメントは、常にレベル 1 です。

以下の図は、 9 ページの『セグメント間の関係』 のデータベース・レコードのレベルを図示したものです。

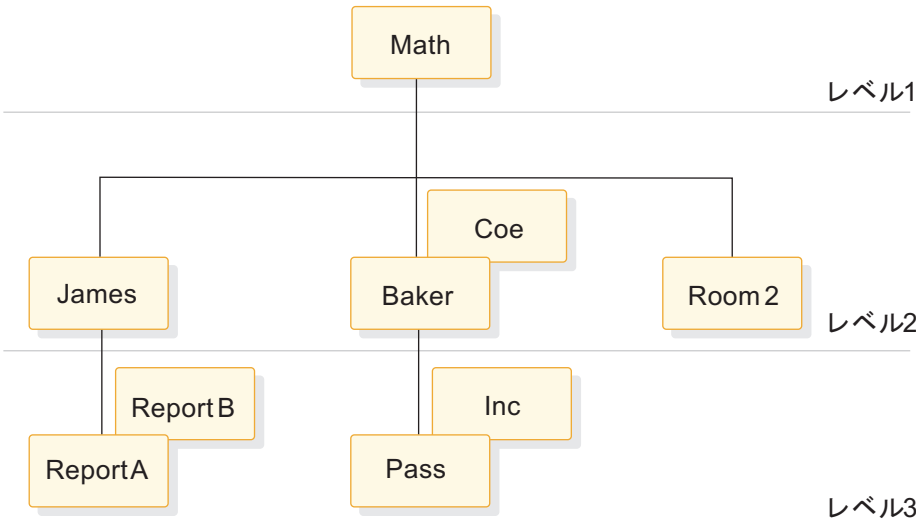


図 5. データベース内でのレベル

IMS データベース・タイプ

IMS では、さまざまな種類のデータベースを定義することができます。アプリケーションの処理要件に最も適合したデータベース・タイプを選んで定義します。

IMS データベースはそれぞれ独自のアクセス方式を持っていますが、これは、IMS が z/OS オペレーティング・システムの制御のもとで稼働するためであることを知っておく必要があります。このオペレーティング・システムはセグメントが何であるか知りません。このオペレーティング・システムは論理レコードを処理するのであり、セグメントを処理するわけではないためです。したがって、IMS アクセス方式がデータベース・レコードの中のセグメントを操作します。論理レコードを読み取る必要があるときは、オペレーティング・システムのアクセス方式 (または IMS) が使用されます。

以下の表は、定義できる IMS データベース・タイプ、IMS データベースで使用される IMS アクセス方式、およびこのアクセス方式と併用できるオペレーティング・システム・アクセス方式を列挙したものです。データベース・タイプは、アクセス方式が少し異なりますが、それらはすべてデータベース・レコードを使用します。

表 *1. IMS* データベース・タイプと *z/OS* アクセス方式

IMS データベースの種類	データベース・タイプのフルネーム	使用可能な **IMS** またはオペレーティング・システムのアクセス方式
DEDB [1]	高速処理データベース	メディア・マネージャー
GSAM	汎用順次アクセス方式	QSAM/BSAM または VSAM
HDAM	階層直接アクセス方式	VSAM または OSAM
HIDAM	階層索引直接アクセス方式	VSAM または OSAM
HISAM	階層索引順次アクセス方式	VSAM
HSAM	階層順次アクセス方式	BSAM または QSAM
MSDB [2]	主記憶データベース	N/A
PHDAM	区分階層直接アクセス方式	VSAM または OSAM
PHIDAM	区分階層索引直接アクセス方式	VSAM または OSAM
PSINDEX	区分副次索引	VSAM
SHSAM	単純階層順次アクセス方式	BSAM または QSAM
SHISAM	単純階層索引順次アクセス方式	VSAM

表の注記**:**

1. DBCTL の場合、BMP のみ使用可能
2. DBCTL には適用できない

上記の表にリストされているデータベースは、全機能データベース・タイプと高速機能データベース・タイプの 2 種類に分けられます。高速機能データベース・タイプは、DEDB と MSDB の 2 つのみです。上記の表のそれ以外のデータベースは、いずれも全機能データベース・タイプと見なされます。

関連概念:

117 ページの『第 3 部 データベースのタイプと機能』
119 ページの『第 12 章 IMS データベースのタイプと機能の要約』

データベース・レコード

データベースは一連のデータベース・レコードから成り立っており、データベース・レコードは一連のセグメントから成り立っています。

ここでもう 1 つ理解しておくべきことは、特定の 1 つのデータベースは 1 種類のデータベース・レコードしか収容できないということです。 例えば、この学校用データベースには、必要な数だけの学校用レコードを保管することができます。しかし、異なる種類のデータベース・レコード、例えば以下の図に示されているような医療用データベース・レコードを作成して、これを学校用データベースに保管することはできません。

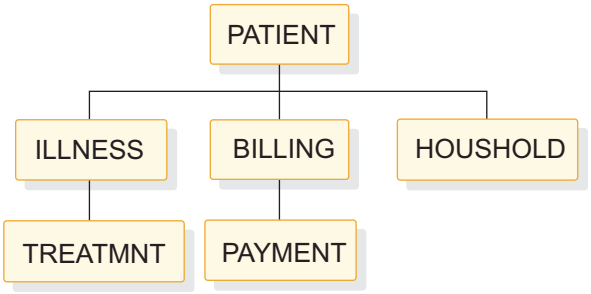


図 6. 医療用データベース・レコードの例

最後に理解しておくべきことは、特定のデータベース・レコードをデータベースに保管するとき、このデータベース・レコードは必ずしも最初に設計したすべてのセグメント・タイプを含んでいる必要はないということです。 データベース・レコードがデータベースの中に存在するためには、データベース・レコードはルート・セグメントのオカレンスだけを含んでいればいいのです。先に見た学校用データベースの中には、以下の図の 4 つのレコードのすべてを保管することができます。

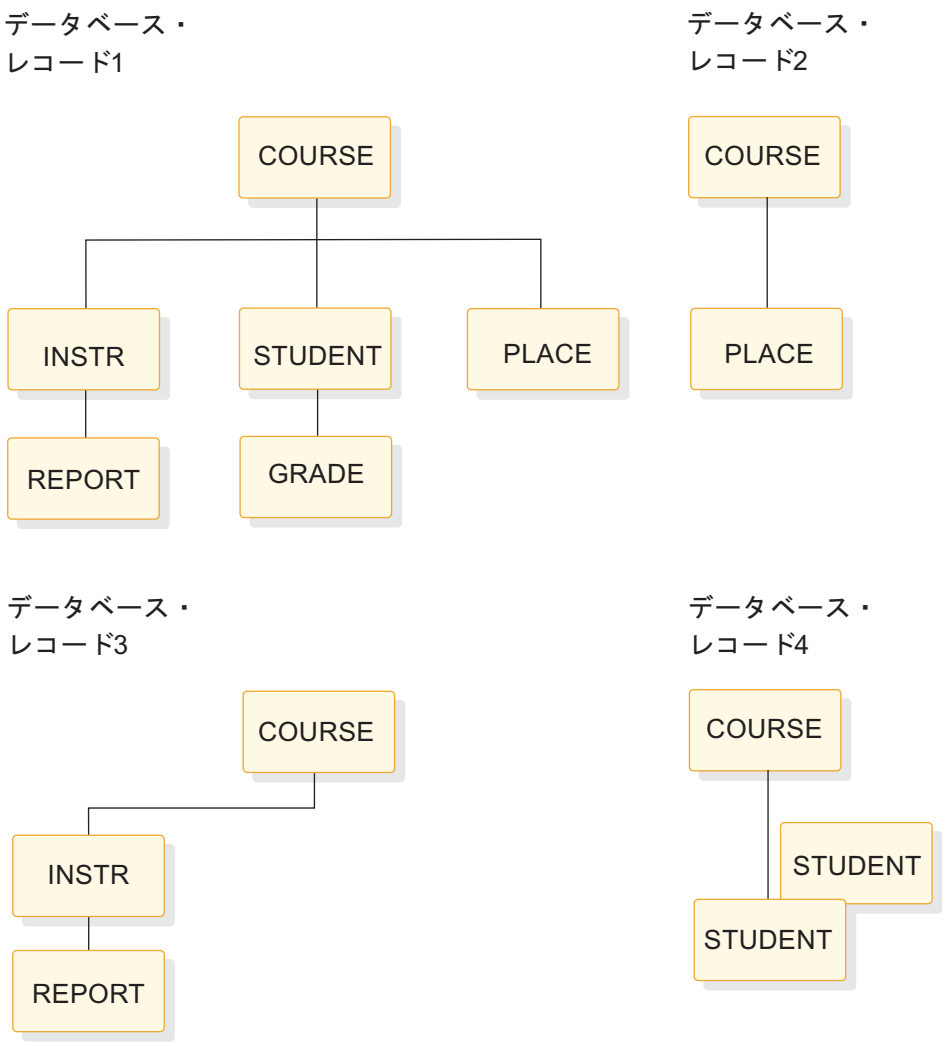


図 7. 学校用データベースに保管することのできるレコードの例

しかし、親も保管しないかぎりセグメントを保管することはできません。例えば、以下の図のレコードを保管することはできません。

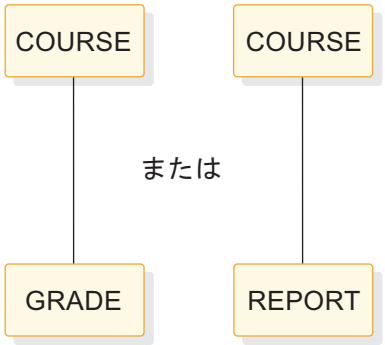


図 8. 学校用データベースに保管することのできないレコード

どのセグメント・タイプのオカレンスも、あとでデータベースに追加したりデータベースから削除したりすることができます。

セグメント

データベース・レコードは、1 つ以上のセグメントから構成され、セグメントは、IMS が保管できる最小のデータ単位です。

セグメントについてさらに知っておく必要のあることを以下に示します。

- 1 つのデータベース・レコードには、最大 255 のセグメント・タイプを入れることができます。データベースに割り振られているスペースがセグメント・オカレンスの数を制限します。
- セグメントの長さを決めるのはデータベース管理者です。ただし、セグメントの長さは、セグメントが保管される装置の物理レコード長より大きくはできません。
- セグメントの長さはセグメント・タイプによって指定されます。セグメント・タイプは、可変長または固定長にすることができます。

セグメントは 2 つの部分 (接頭部とデータ) から成り立っています。ただし、SHSAM データベースまたは SHISAM データベースを使用する場合には、これはあてはまりません。 SHSAM データベースおよび SHISAM データベースでは、セグメントはデータだけから成り立っています。 GSAM データベースにはセグメントは存在しません。

以下の図は固定長セグメントのフォーマットを示しており、

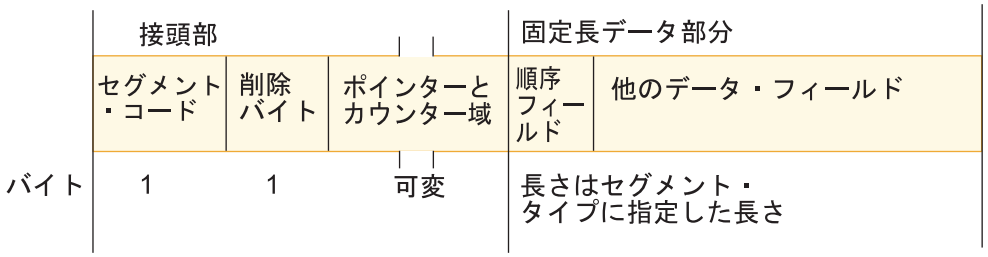


図 *9.* 固定長セグメントのフォーマット

以下の図に示すのは可変長セグメントのフォーマットです。

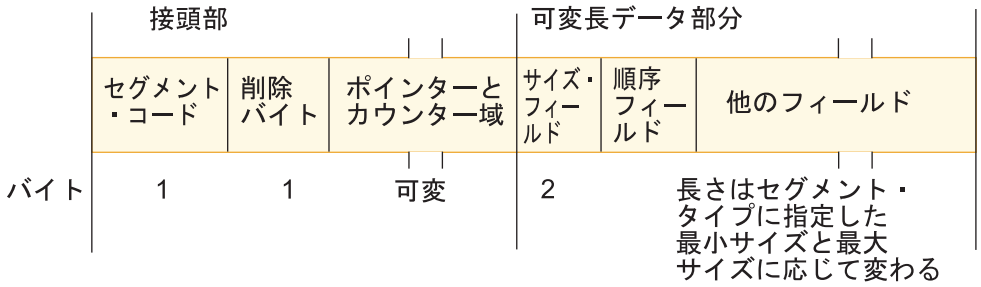


図 *10.* 可変長セグメントのフォーマット

IMS はセグメントの接頭部の部分を使用して、セグメントを「管理」します。セグメントの接頭部はセグメント・コードと削除バイトから成り立ち、そしてある種のデータベースでは、ポインターとカウンター域も含まれます。アプリケーション・

プログラムは、セグメントの接頭部の部分を「参照」しません。ユーザーのデータは 1 つ以上のフィールドに配置されて、セグメントのデータ部分に入っています。

関連概念:

146 ページの『SHSAM、SHISAM、および GSAM データベース』

234 ページの『主記憶データベース (MSDB)』

211 ページの『高速処理データベース』

セグメント・コード

IMS には、データベースに保管される各セグメント・タイプを識別する手段が必要です。 IMS は、そのためにセグメント・コード・フィールドを使用します。

セグメント・タイプをロードするとき、IMS は固有の ID (整数 1 から 255) をこのセグメント・タイプに割り当てます。 IMS は番号を昇順に割り当て、ルート・セグメント・タイプ (番号 1) から始まり、階層順にすべての従属セグメント・タイプにそれを続けていきます。

削除バイト

アプリケーション・プログラムがデータベースからあるセグメントを削除した場合、このセグメントによって占められていたスペースはただちに再使用が可能になる場合もあり、そうでない場合もあります。

セグメントの削除については、それぞれのデータベース・タイプについての記述の中で説明します。ここで知っておくことは、IMS が接頭部の中のこのバイトを用いて、削除されたセグメントの状況を追跡しているということです。

関連資料:

354 ページの『削除バイトの中のビット』

ポインターとカウンター域

ポインターとカウンター域は、HDAM、PHDAM、HIDAM、および PHIDAM データベース、および特別な状況の場合には HISAM データベースの中に存在します。

ポインターとカウンター域には、次の 2 種類の情報を入れることができます。

- ポインター情報は、セグメントが指す、1 つ以上のセグメント・アドレスから成り立ちます。
- カウンター情報は、IMS のオプションの機能の 1 つである論理関係を定義するときに使用されます。

ポインターとカウンター域の長さは、セグメントに含まれるアドレスの数、および論理関係を使用するかどうかによって変わります。これらのトピックについては、本書で後からさらに詳しく説明します。

データ部分

セグメントのデータ部分には、1 つ以上のデータ・エレメントが収容されています。これはアプリケーション・プログラムが処理するデータであり、セグメントの接頭部の部分とは異なり、アプリケーション・プログラムはこのデータを見ることができます。

アプリケーション・プログラムは、セグメント・タイプの名前を用いてデータベースの中のセグメントにアクセスします。アプリケーション・プログラムがある 1 つのセグメントの一部分を参照しようとしている場合には、参照しようとしているセグメントのその部分を示すフィールド名を IMS に対して定義することができます。フィールド名は、セグメント検索引数 (SSA) において呼び出しを修飾するのに使用されます。アプリケーション・プログラムは、データをたとえフィールドとして定義しなくてもこのデータを見ることができます。しかし、データをフィールドとして定義しないと、アプリケーション・プログラムはこのデータを対象とする SSA を修飾することはできません。

ある 1 つのセグメント・タイプに対して定義できるフィールドの最大数は 255 です。ある 1 つのデータベースに対して定義することのできるフィールドの最大数は 1000 です。ここで注意すべきことは、1000 というのはデータベースの中のフィールドのタイプの数を意味し、オカレンスの数ではないという点です。データベース内のフィールドのオカレンスの数は、データベース用に定義しているストレージの容量によってのみ制限されます。

データ部分の 3 つのフィールド・タイプ

セグメントのデータ部分では、3 種類のフィールド・タイプを定義することができます。シーケンス・フィールド、データ・フィールド、および可変長セグメントの場合にセグメントの長さを示しているサイズ・フィールドです。

最初の 2 種類のフィールド・タイプには、ユーザー・データを収容します。アプリケーション・プログラムはこの両方のフィールドを使用して呼び出しを修飾することができます。なお、シーケンス・フィールドには、データを収容すること以外の用途があります。

シーケンス・フィールド (しばしば、キーと呼ばれる) を使用して、1 つのセグメント・タイプの複数のオカレンスを特定の親のもとにキー・シーケンスで保管することができます。例えば、以下の図のデータベース・レコードでは、STUDENT セグメントのセグメント・オカレンスが 3 つあり、STUDENT セグメントにはデータ・エレメントが 3 つあります。

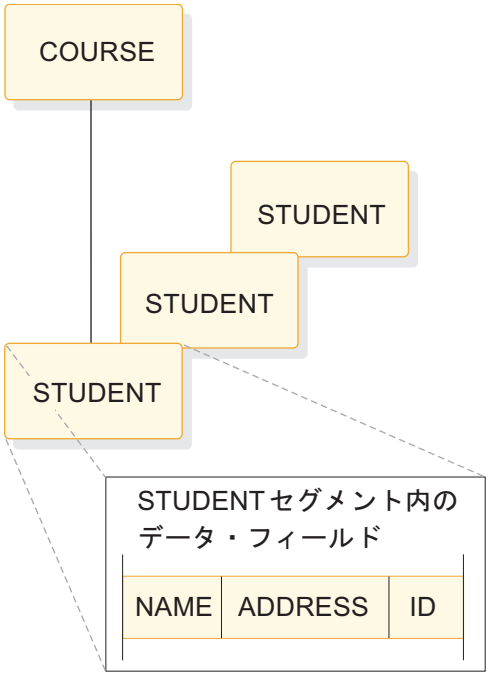


図 *11. STUDENT* セグメントの *3* つのセグメント・オカレンスと *3* つのデータ・エレメント

STUDENT セグメントをデータベースの中に保管する際に、学生名をアルファベット順に並べたいと考えているものとします。 NAME データのフィールドを固有のシーケンス・フィールドとして定義すると、IMS は STUDENT セグメント・オカレンスをアルファベット順に保管します。以下の図は、アルファベット順に保管された STUDENT セグメントの 3 つのオカレンスを示しています。

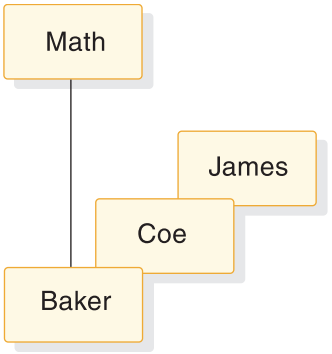


図 *12.* アルファベット順に保管された *STUDENT* セグメントの例

HISAM、HDAM、PHDAM、HIDAM、または PHIDAM データベースのルート・セグメントにシーケンス・フィールドを定義すると、アプリケーション・プログラムはそれを使用して特定のルート・セグメントに、したがって特定のデータベース・レコードにアクセスすることができます。シーケンス・フィールドを使用することにより、アプリケーション・プログラムは特定のデータベース・レコードの検出のためにデータベースを順次に検索する必要がなくなりますが、順次にレコードを検索することもできます (HISAM、HIDAM、および PHIDAM の場合)。

IMS のオプションの機能である論理関係または副次索引を使用する場合には、他の方法でシーケンス・フィールドを使用することもできます。これらの他の使用法については、本書で後から詳しく説明します。

シーケンス・フィールドについてここで知っておくべき重要なことを、次に列挙しておきます。

- 必ずしも常にシーケンス・フィールドを定義する必要はありません。本書では、シーケンス・フィールドが必要な場合について説明します。
- シーケンス・フィールドの値は、固有または非固有として定義することができます。
- シーケンス・フィールドの中のデータまたは値は、セグメントの「キー」と呼ばれます。

オプション・データベース機能の概説

IMS には、データベースのために使用できるオプションの機能がいくつかあります。

機能には、次のものが含まれます。

論理関係

論理関係。この機能を使用すると、アプリケーション・プログラムが論理データベース・レコードにアクセスできるようになります。 1 つの論理データベース・レコードは 1 つ以上の物理データベース・レコードの中のセグメントによって構成することができます。物理データベース・レコードは、1 つ以上のデータベースに保管することができます。したがって、論理データベース・レコードを使用することにより、物理データベース構造とは異なるデータベース構造をアプリケーション・プログラムが見ることができます。

例えば、ある 1 つの論理データ構造の中に 2 つの異なる物理データベースからのセグメントが入っている場合には、次のような 2 つの異なるパスから 1 つのセグメントにアクセスすることができます。

- 最も頻繁に使用され、最も緊急な応答時間が必要なパスに、このセグメントを物理的に保管することができます。
- このセグメントの場所を指しているポインターを、別のアプリケーション・プログラムが必要とする代替パスに、物理的に保管することができます。

副次索引

副次索引。あるデータベースの中のセグメントを、シーケンス・フィールドで定義している順序とは異なる順序でアクセスするのに使用できます。

可変長セグメント

可変長セグメント。セグメント・タイプのデータ部分の長さを可変長にすることができます。ある 1 つのセグメント・タイプのデータ部分のサイズがセグメント・オカレンスごとに大きくばらついている場合には、可変長セグメントを使用してください。可変長セグメントに対しては、セグメント・タイプの最小の長さと最大の長さを定義します。最小と最大の長さの両方を定

義することによって、セグメントの長さが最大の長さより短いときにはいつでもデータベースの中のスペースが節約されます。

フィールド・レベル・センシティビティー
フィールド・レベル・センシティビティー は、次の目的で使用できる機能です。

- セキュリティーの目的で、セグメントの中のフィールドで、選択済みのフィールドにはアプリケーション・プログラムのアクセスをリジェクトするようにします。
- アプリケーション・プログラムがある 1 つのセグメントを構成するフィールドのサブセットを使用する (使用しないフィールドを処理しない) ことができるようにするか、またはアプリケーション・プログラムがある 1 つのセグメントの中のフィールドを違った順序で使用することができるようにします。アプリケーション・プログラムからのさまざまなニーズに対応するため、フィールド・レベル・センシティビティーは、このような方法で使用してください。

セグメント編集/圧縮
セグメント編集/圧縮。セグメントに対して次の 3 つのことができます。

- セグメント・データが装置上にあるとき、このセグメント・データをエンコード、すなわち、「暗号化」し、このセグメントへのアクセス権をもつアプリケーション・プログラムにだけデコード (復号) された形のデータを与えることができるようにします。
- データを編集し、保管されているのとは異なるフォーマットでアプリケーション・プログラムがそのデータを受け取ることができるようにします。
- セグメントを装置に書き込むとき、データを圧縮し、直接アクセス・ストレージ・デバイス (DASD) のストレージの使用効率を高めます。

データ・キャプチャー出口ルーチン
データ・キャプチャー出口ルーチン。挿入、置き換え、または削除呼び出しで、アプリケーション・プログラムが IMS データベースを更新するときに、セグメント・データを収集するために使用されます。これは、作業単位またはアプリケーションの更新内で実行される同期的なアクティビティーです。収集されたデータは、Db2 for z/OS データベースへのデータの伝搬に使用します。データ・キャプチャー出口ルーチンは、データ伝搬以外の操作を実行する場合にも使用できます。

非同期データ・キャプチャー
非同期データ・キャプチャー。挿入、置き換え、または削除呼び出しで、アプリケーション・プログラムが IMS データベースを更新するときに、セグメント・データを収集する機能です。これは、作業単位またはアプリケーションの更新の外で実行される非同期のアクティビティーです。収集されたデータは、Db2 for z/OS データベースにデータを非同期に伝搬する場合に使用します。非同期データ・キャプチャー機能は、データ伝搬以外の操作にも使用できます。

IMS DataPropagator によって、変更済みのデータを IMS と Db2 for z/OS の間で同期または非同期に伝搬することができます。

関連資料: IMS DataPropagator の詳細については、「*IMS DataPropagator for z/OS: An Introduction*」を参照してください。

複数データ・セット・グループ

複数データ・セット・グループ。 1 つのデータベース・レコードの中のいくつかのセグメントを 1 次データ・セット以外のデータ・セットの中に保管する機能です。これは、データベース・レコードの中のセグメントの階層順を破壊することなく、実行することができます。

複数データ・セット・グループを使用する理由の 1 つは、さまざまなアプリケーションのいろいろなニーズに応えるためです。複数データ・セット・グループを使用することによって、アプリケーション・プログラムが、関心のあるセグメントへの高速アクセスを行うことができます。アプリケーション・プログラムは、不要なセグメントを収容しているデータ・セットを単に迂回するだけです。複数データ・セット・グループを使用するもう 1 つの理由は、例えば、頻繁に使用されるセグメントとあまり使用されないセグメントとを分けることにより、パフォーマンスの向上を図ることです。あるいは、複数データ・セット・グループを使用して、サイズが平均からかけ離れているセグメント・タイプを別個のデータ・セット・グループに保管することにより、スペースを節約することもできます。

関連概念:

423 ページの『第 18 章 オプション・データベース機能』

IMS に対してデータベースを定義する方法

DBD を生成するマクロをコーディングして、データベースのほとんどの特性を IMS に対して定義します。

*DBD (*データベース記述子*)* は、データベースの編成とアクセス方式、データベース・レコードの中のセグメントとフィールド、およびセグメント・タイプ相互間の関係について記述する一連のマクロ命令です。

高可用性ラージ・データベース (HALDB) と総称される、IMS 区分階層直接データベースなどの一部のデータベースに対しては、RECON データ・セットで追加のデータベース特性を定義する必要があります。

IBM DB/DC (データベース/データ・コミュニケーション) データ・ディクショナリーが備わっている場合には、これを用いて (DEDB と MSDB 以外の) データベースを定義することができます。この DB/DC データ・ディクショナリーには、DBD を作成するのに必要なすべての情報を入れておくことができます。

アプリケーション・プログラムからのデータベースの見方

データベース管理者は、アプリケーション・プログラムがデータベースをどう見るかを制御できます。

アプリケーション・プログラムが、データベース・レコードの中のすべてのセグメントまたはフィールドを使用する必要があるとは限りません。セキュリティーまたは保全のために、あるアプリケーション・プログラムが特定のセグメントにアクセスする必要がないこともあります。さらに、あるアプリケーション・プログラムが

一部のセグメントまたはフィールドに対してある種の操作を行うのを必要としないこともあります。例えば、あるアプリケーション・プログラムが SALARY セグメントに対する読み取りアクセスを必要とするが、更新アクセスは必要ない場合があります。

PSB (プログラム仕様ブロック) をコーディングし、生成することにより、あるアプリケーションでどのセグメントとフィールドを表示することができ、あるセグメントに対してどのような操作を行うことができるかを制御します。

PSB は、データベースの中のセグメントに対するあるアプリケーション・プログラムのアクセスがどうなっているか記述している一連のマクロ命令です。 PSB は 1 つ以上のプログラム連絡ブロック (PCB) により成り立っており、各 PCB は、当該アプリケーション・プログラムがデータベースを読み取りおよび使用できる能力について記述しています。例えば、ある 1 つのアプリケーション・プログラムが同一のデータベースに異なる複数の視点と用法を持つことができます。 また、1 つのアプリケーション・プログラムが異なるいくつかのデータベースにアクセスすることができ、そのためにその PSB の中に複数の PCB を持つことができます。

IBM DB/DC データ・ディクショナリーが備わっている場合には、これを用いてアプリケーション・プログラムによるデータベースへのアクセスを定義することができます。このデータ・ディクショナリーには、プログラム・ビューを作成するのに必要なすべての情報を収容することができます。

第 2 章 IMS データベースのための標準、手順、および命名規則

入念に計画された標準と手順があり、IMS の規則が十分に理解されていれば、管理者、オペレーター、およびプログラマーはそれらに従って、使用するシステムの信頼性と効率を高めることができます。

データベース・システムのための標準と手順

使用するデータベース・システムのための標準と手順を作成する必要があります。

適切な標準と手順を使用することで、以下の点が向上します。

- アプリケーション・システムの品質 (標準と手順を設けてそれに従うことで、アプリケーション開発プロセス全体をより綿密に管理できるため)
- アプリケーションおよびデータベースの設計における生産性 (設計上の決定を行うための指針があるため)
- アプリケーションのコーディングの生産性 (コーディングの標準と手順があるため)
- データベース管理者とアプリケーション開発者間のコミュニケーション (双方の責任分担が明確に定められるため)
- 操作における信頼性とリカバリー可能性 (明確で理解しやすい操作手順があるため)

データベース設計、アプリケーション開発、アプリケーション・プログラムによるデータベースの使用、アプリケーション設計、およびバッチ操作に関する手順および標準を設定し、これをテストしてみる必要があります。これらの標準は、インストール要件が変わればそれに従って変わることのある指針です。

次に示すデータベース設計の各側面に対して標準的な慣例を確立することができます。

- データベース構造とセグメント化
  - データベース内のセグメントの数
  - セグメントの配置
  - セグメントのサイズ
  - 可変長セグメントの使用
  - セグメント編集/圧縮機能を使用する場合
  - 2 次データ・セット・グループを使用する場合
  - アプリケーション内のデータベースの数
  - フィールド・レベル・センシティビティーを使用する場合とその使用法
  - データベース・サイズ
- アクセス方式
  - HISAM を使用する場合
  - HISAM のレコード・サイズの選定

VSAM を使用する HISAM 編成
GSAM を使用する場合
物理子/物理兄弟ポインターの使用
兄弟逆方向ポインターの使用
最終子ポインターの使用
VSAM を使用する HIDAM または PHIDAM 索引編成
ルート・セグメント・レベルでの HIDAM または PHIDAM ポインター・オプション
兄弟チェーンの順序付け
HD フリー・スペースの使用
HDAM または PHDAM を使用する場合
HDAM または PHDAM データベースの順次処理
HDAM または PHDAM の「バイト限界カウント」の使用
HDAM または PHDAM ルート・セグメントの兄弟逆方向ポインターの使用
HDAM または PHDAM でのフリー・スペースの使用
DEDB を使用する場合
DEDB の順次処理
DEDB パラメーターの使用
サブセット・ポインターの使用
多重エリア・データ・セットの使用

- 副次索引
  順次処理のための副次索引の使用
  発揮性セグメントでの副次索引の使用
  HISAM データベースにおける副次索引の使用
  固有の副次索引の使用
  疎索引付けの使用
  別個のデータベースとしての副次索引の処理
- 論理関係
  直接ポインターとシンボリック・ポインターの比較
  長い論理兄弟チェーンの回避
  論理兄弟チェーンの順序付け
  実論理子セグメントの配置

また、アプリケーション・プログラムでのデータベースの使用方法に関する次のような標準を確立することもできます。

- 更新機能と読み取り機能を別々のプログラムに入れることを必須とする
- アプリケーション・プログラム当たりの許されるトランザクション・タイプの数
- アプリケーションが計画的な異常終了を出すタイミング、およびアプリケーションで許容する異常終了コードの範囲
- アプリケーション・プログラムがマスター端末にメッセージを出すのを許すかどうか

- IOAREA におけるデータの参照方式、および IMS 変数 (PCB や SSA など) の参照方式
- アプリケーションによる事前定義構造 (PCB マスク、SSA、データベース・セグメントのフォーマットなど) の使用
- メッセージ・キューに対する GU 呼び出しの使用
- MPP プログラムおよび BMP プログラム再使用の可能性
- 修飾された呼び出しおよび SSA の使用
- パス呼び出しの使用
- CHANGE 呼び出しの使用
- システム呼び出しの使用: PURG、LOG、STAT、SNAP、GCMD、および CMD

次に示すアプリケーション設計の各側面を制御するための手順を確立します。

- データベース管理者とアプリケーション設計担当者との間の対話
- データ・エレメントとデータ構造のためのディクショナリー、COPY ライブラリー、または STRUCTURE ライブラリーの使用
- 設計のレビューと検査の要求

操作については、以下の作成を検討してください。

- コンピューター機能へのアクセスを限定するための手順
- 次の事項を確認するための制御点
  - ジョブに、完全でかつ適切な提出用文書化が入っている
  - ジョブがスケジュールどおり正常に実行される
  - 正しい入出力ボリュームが使用され、出力が正しく配布される
  - 決められたテスト計画に従ってのみテスト・プログラムが実行される
  - 問題がすべて記録され、責任部門当事者に報告されるようにするために、事故報告書が保持される
- 通常の操作手順 (操作日程、コールド・スタートの手順、ウォーム・スタートの手順、シャットダウンの手順、およびバッチ・プログラムのスケジューリングと実行)
- 非常事態のための手順。非常時には、環境は緊迫しています。手順を文書化しておけば、このような事態を 1 つずつ解決するための指針になります。これには、緊急時再始動、データベース・バックアウト、データベース・リカバリー、ログ・リカバリー、およびバッチ・プログラム再始動の手順が含まれます。
- マスター端末オペレーターのためのインストールの手引き。この手引きは、「*IMS V13* オペレーションおよびオートメーション」を補足するものです。
- マスター操作ログ。このログに入れることのできるものは、システム可用性、障害の時刻と種類、障害の原因、とられたリカバリー・ステップ、および正常な終了の場合のシステム終了の種類についての記録です。
- システム保守ログ。このログに入れることのできるものは、すべてのリリース・レベルと修正レベル、リリース従属関係、適用されたプログラム一時修正 (PTF)、APAR の状況と提出日付、および迂回解決についての記録です。

IMS データベースのための一般的な命名規則

命名規則は、ユーザーが IMS システム内で多数のリソースを識別して管理するために役立ちます。一部の命名規則は IMS で定義されていますが、それ以外に多数の命名規則をユーザーが定義できます。

命名規則の設定のための一般的な規則

データ処理プロジェクト、特に複数アプリケーション環境においては、適切な命名規則が必要です。

適切な命名規則には、以下のような一般規則が含まれます。

- それぞれの名前を固有のものにする必要があります。名前が固有でない場合、予測不能なエラーが発生することがあります。
- それぞれの名前に意味を持たせ、すべての人にその名前が付いたエレメントのリソース・タイプが分かるようにする必要があります。

以下の表に、基本的な命名規則の例を示します。これらの規則は単に一例であり、ユーザーは独自の命名規則を設定できます。

表 2. 基本的な命名規則の例：

リソース・タイプ	規則
SYSTEM	先頭の文字 S
JOB	先頭の文字 J
PROGRAM	先頭の文字 P IMS プログラムである場合 (PSB に合わせるため) 先頭の文字 G その他の場合
MODULE	先頭の文字 M
COPY	先頭の文字 C セグメント構造が入っているメンバーの場合先頭の A セグメントのすべての SSA が入っているメンバーの場合他のメンバーはセグメント名と同じにする必要がある。
TRANSACTION	先頭の文字 T
PSB	先頭の文字 P
PCB	PSB と同じ注: PCB オカレンス番号は、PSB における PCB の位置を示す。

表 2. 基本的な命名規則の例 (続き) :

リソース・タイプ	規則		
DATABASE	先頭の文字 D 後続の文字でデータベースのタイプおよび他のデータベースとの関係を識別。例えば D*taaann* では、文字 *taaann* が以下を示す。		
	文字	意味	
	t	データベース・タイプ。データベースは次に挙げる種類のいずれかに属します。	
		P	物理
		L	論理
		X	1 次索引
		Y	副次索引
	aaa	同一の物理データベースに基づいているすべての論理データベースと索引データベースに共通している固有のデータベース ID	
	nn	固有 ID。複数の論理データベースまたは副次索引データベースがある場合に、名前を固有なものにするための ID	
SEGMENT	先頭の文字 S、R、または O 後続の文字でセグメントのタイプおよびそのデータベースとの関係を識別。R は非 DL/I ファイル・レコード定義である「セグメント」を示す。O はその他のデータ域 (端末入出力域、制御ブロック、報告書行など) を示す。例えば S*aaabbbb* では、文字 *aaabbbb* が以下を示す。		
	文字	意味	
	aaa	固有のデータベース ID。セグメントが属する物理データベースと同じ **注:** 連結セグメントの aaa の値は、論理子セグメントの aaa と同じものにすべきである。	
	bbbb	ユーザー名のための ID	
エレメント	先頭の文字 E		

HALDB 区画、DD 名、およびデータ・セットのための命名規則

区画、DD 名、およびデータ・セット名の HALDB のための命名規則により、HALDB の PHDAM、PHIDAM、および PSINDEX データベース内の多数の区画およびデータ・セットの管理が単純化されます。

関連概念:

747 ページの『HALDB オンライン再編成のデータ・セット命名規則』

関連タスク:

963 ページの『論理的に関連したデータベース・データ・セットの割り振り』

954 ページの『索引付きデータベース・データ・セットの割り振り』

941 ページの『データベース・データ・セットの割り振り』

HALDB 区画のための命名規則

各区画に名前を割り当てます。区画名は長さが 1 から 7 バイトです。

区画名は、RECON データ・セットに登録されるデータベース名、区画名、および高速機能域名の中で固有でなければなりません。区画内のデータを記述するために区画名を使用できますが、そのような名前は注意深く選択してください。区画を追加または削除すると、またはその境界を変更すると、データは 1 つの区画から別の区画に移動します。この移動は、意味のある名前の割り当てを難しくするおそれがあります。既存の区画の名前は、その区画を削除して、新しい区画として再定義しなければ、変更することはできません。

HALDB データ定義名 (DD 名) のための命名規則

IMS は HALDB データ定義名 (DD 名) を、区画名に 1 バイトの接尾部を付加することにより定義します。接尾部はデータ・セットのタイプを示し、複数データ・セット・グループを使用する場合は、グループ内のデータ・セットを区別します。

以下の表に、HALDB データ・セット・タイプと、対応する DD 名接尾部を示します。

表 3. データ・セット・タイプ別の *HALDB DD* 名のための接尾部

データ・セット・タイプ	**DD** 名の接尾部	**HALDB** オンライン再編成が使用される場合の追加接尾部
データベース・データ・セット	A–J	M–V
1 次索引 (PHIDAM のみ)	X	Y
間接リスト・データ・セット (PHDAM および PHIDAM のみ)	L	L (ILDS 用の接尾部は変更されない)

複数データ・セット・グループを使用する場合、A から J の接尾部は、SEGM ステートメントの DSGROUP パラメーターに指定する値です。文字 A は最初のデータベース・データ・セット (DBDS) を示し、文字 B は 2 番目のデータベース・データ・セットを示し、それ以降は文字 J まで可能です。複数データ・セット・グループを使用しない場合は、DSGROUP パラメーターを指定しません。その場合、レコード・セグメントを含む単一のデータ・セットの DD 名に接尾部 A が付加されます。

接尾部 M から V および Y は、IMS の統合 HALDB オンライン再編成機能用に自動的に作成されます。これらを DBD に指定する必要はありません。ある区画を再編成するために HALDB オンライン再編成機能を使用したことがない場合、その区画で接尾部 M から V および Y は使用されません。

PSINDEX データベースでは、個々の区画にはデータ・セットが 1 つだけ入れられます。そのデータ・セットに対応する DD 名には、接尾部 A が使用されます。

例えば、PART1 という名前の PHIDAM データベース区画は、その最初の DBDS の DD 名は PART1A、2 番目の DBDS の DD 名は PART1B、10 番目の DBDS の DD 名は最後の PART1J ということになります。区画 PART1 の間接リスト・データ・セット (ILDS) および 1 次索引の DD 名は、それぞれ、PART1L および PART1X になります。そして、PARSI という名前の PSINDEX データベース区画は、そのデータ・セットの DD 名は PARSIA になります。

区画の再編成時、IMS の統合 HALDB オンライン再編成機能は、オンライン再編成プロセスを開始する前にアクティブな各データ・セット用に追加データ・セットを使用します。例えば、PART1M という DD 名が作成されて、これは、アクティブ・データ・セット PART1A に対応しています。 PART1B には PART1N というふうに作成され、PART1J が存在していれば、PART1J には最後の PART1V が作成されます。

DD 名は、RECON データ・セットに登録されるデータベース名、区画名、および高速機能域名の中で固有でなければなりません。

HALDB データ・セット名のための命名規則

HALDB データ・セット名は、その一部分をユーザーが定義し、残りの部分を IMS が作成します。

区画を定義するとき、その区画用に最大 37 文字のデータ・セット名接頭部を定義します。データ・セット名接頭部は、他のいかなる HALDB データベースのデータ・セット名接頭部とも重複してはなりませんが、単一の HALDB データベース内であれば、重複していてもかまいません。区画 ID は固有であるため、データ・セット名接頭部に IMS が付加する接尾部によって、HALDB データベース内のさまざまな区画のデータ・セット名が固有の名前になります。区画名とそのデータ・セットの名前との間に、必要な相関関係はありません。

最低レベルの修飾子を作成するために、IMS は接頭部に 6 文字の接尾部を付加してデータ・セット名を形成します。IMS が提供する接尾部の先頭文字は英字であって、A から J、L、および X、または M から V、L、および Y です。6 文字の接尾部は、先行するデータ・セット名修飾子からピリオドによって分離されます。

データ・セット名接尾部の先頭文字は、DD 名の接尾部として使用される文字と一致します。接尾部の残りの 5 桁は区画 ID 番号を表します。これは DBRC によって割り当てられ、変更することはできません。例えば、次のようになります。

- A00001 という接尾部は、区画 ID 1 の区画の中の最初または唯一の DBDS を表します。
- J00004 という接尾部は、区画 ID 4 の区画の中の 10 番目の DBDS を表します。
- L00007 という接尾部は、区画 ID 7 の区画の中の ILDS を表します。
- X00011 という接尾部は、区画 ID 11 の PHIDAM 区画の中の 1 次索引を表します。

第 3 章 データベース開発のレビュー・プロセス

データベース設計が適切に行われ、次いでこれが効果的にインプリメントされるようにする最善の方法の 1 つは、開発のさまざまな段階での設計をよくレビューすることです。

データベース・システムの開発中によく行われるレビューのタイプについて、以下のトピックで説明します。

第 1、2、3、4 回の設計レビュー

第 1、2 回のコード検査

セキュリティー検査

インプリメンテーション後レビュー

設計のレビュー

設計のレビューで確認すべきことは、開発されている機能が適切なものであること、パフォーマンスが満足なレベルにあること、ご使用のシステムの規準が満たされていること、およびこのプロジェクトが理解されていて十分な管理下にあることです。

レビューは、初期データベース・システムの構築時に何回か実施し、その後もこのデータベース・システムを対象として実行される 1 つのプログラムまたはプログラム群を開発するたびに実施します。

設計レビューでのデータベース管理者の役割

レビュー・プロセスでの管理者の役割は、データベースの設計が適切に行われ、次いでこれが効果的にインプリメントされるようにすることです。この役割は継続するものであり、他のデータベース管理作業を支える枠組みとなります。

レビュー過程においては、データベース管理の役割は重要です。通常は、データベース管理スタッフの一員で、開発されている特定のシステムとつながりのない人が、このレビューのモデレーターとなります。モデレーターの役割は、単なるミーティングの開催より大きなものです。モデレーターは、このシステムの開発が既存のシステムまたは将来のシステムにどのような影響を及ぼすかについても調べます。本書の読者は、このシステムの開発を担当しているデータベース管理者ですから、すべてのレビューに参加する必要があります。

レビューについての一般情報

通常、システム開発中は、ほとんどの開発プロジェクトに共通の一連のレビューが、開発グループによって実施されます。

話を簡単にするために、ここではレビューの対象を「システム」と呼びます。実際には、「システム」は、1 つのプログラムであったり、一連のプログラムであったり、あるいはデータベース・システム全体であったりします。レビューの回数、参加者数、およびレビューにおける参加者の具体的な役割は、ご使用のシステムごと

に多少異なることがあります。理解しておくべきことは、これらのレビューの重要性およびここで実行しなければならない作業です。次に、レビューについての一般情報をいくつか挙げておきます。

- すべてのレビューに参加する人は (データベース管理者のほかに)、レビュー・チームやシステム設計者などです。レビュー・チームは通常、システムの開発に何の責任も持っていません。レビュー・チームは少人数のグループで、目的はあるレビューから次のレビューへと継続性と客観性を確認することです。システム設計者は、最初の機能仕様を書く人です。
- 各レビューの終わりに、そのレビューの過程で出てきた問題点の一覧表を作成します。これらの問題点は、通常、変更の要件となります。各問題点の解決には、それぞれに特定の人を割り当てて、解決の期限を設定しておきます。問題点を解決するためにシステムに大きな変更を加える必要がある場合には、主要な問題点がすべて解消されるまで、何回でもレビューを追加してスケジュールします。
- データ・ディクショナリーがある場合は、これを更新して各レビューの終わりに決定した事項を反映します。このディクショナリーは、情報を最新でしかも使用可能な状態に維持するための重要な補助手段です。特に設計上の決定が行われる最初の 4 回のレビューでは、ディクショナリーは重要です。

第 1 回の設計レビュー

第 1 回の設計レビューの目的は、ユーザーの要件がすべて出そろっていることを確認すると共に、設計上の前提が目標と首尾一貫していることを確かめることです。

システムの最初の機能仕様が完了したあと、最初の設計レビューを行います。この時点では、システムの詳細設計はできあがっていないし、できあがっている必要もありません。この機能仕様をレビューした結果として、この計画をより詳細設計へと進めてもよいか否かが決まります。第 1 回の設計レビューが正常に終了すると、その出力として、承認された一連の最初の機能 仕様ができあがります。

第 1 回の設計レビューに参加する人の中には、通常の参加者のほかに、要件を提出した組織に属する人、および詳細設計の開発に携わる人が含まれます。データベース管理者がこのレビューに参加するのは、主として、情報収集のためです。次のような事柄についても調べます。

データ・エレメント相互間の関係

必要とされるデータの中に、すでに存在するものはないか

第 2 回の設計レビュー

第 2 回の設計レビューでのデータベース管理者の主な役割は、情報収集です。

システムの最終的な機能 仕様が完了したあと、第 2 回目の設計レビューを行います。システムの最終的な機能仕様ができあがっているということは、このシステムのすべてのプログラムの全体的な論理が定義されていること、ならびにプログラム相互間のインターフェースおよび対話が定義されていることを意味します。このレビューの時点で、監査要件およびセキュリティー要件が定義されます。さらにデータ要件も大部分が定義されます。第 2 回の設計レビューが正常に終了すると、その出力として、承認された一連の最終機能仕様ができあがります。

第 1 回の設計レビューに参加した人はすべて第 2 回の設計レビューに参加すべきです。テストおよび保守の担当者の中からもオブザーバーとして参加して、テスト・ケース設計および保守のための情報収集を始めます。監査とセキュリティーに携わっている人も参加することができます。

このレビューにおけるデータベース管理者の主な役割は、依然として情報収集です。次のような事柄についても調べます。

- 仕様がユーザーの要件を満たしているか
- データ項目相互間の関係は正しいか
- 必要とされているデータの中に、すでに存在するものはないか
- 監査要件とセキュリティー要件がユーザーの要件と首尾一貫しているか
- 監査要件とセキュリティー要件はインプリメントできるものであるか

第 3 回の設計レビュー

第 3 回の設計レビューでのデータベース管理者の役割は、トランザクションのフローが、作成中のデータベース設計と矛盾しないようにすることです。

システムの最初の論理 仕様が完了したあと、第 3 回の設計レビューを行います。この時点で、高水準の疑似コードまたはフローチャートが完成します。ロジックにおける主要な決定点が定義されており、外部データとモジュールに対する呼び出しまたは参照が定義されており、しかもロジックの流れの大筋が分かっていれば、高水準疑似コードまたはフローチャートが完成しているといえます。すべてのモジュールおよび外部インターフェースがこの時点で定義され、データ要件の定義が完了し、さらにデータベースとデータ・ファイルの設計が行われます。最初のテスト計画とリカバリー計画が用意されます。しかしコードはまだ書かれません。第 3 回の設計レビューが正常に終了すると、その出力として、承認された一連の最初の論理仕様ができあがります。

第 2 回の設計レビューに参加した人はすべて第 3 回の設計レビューに参加すべきです。このプロジェクトが大規模なものである場合、詳細設計の開発に携わる人は、プロジェクトの各担当部分がレビューの対象となるときだけ参加すれば十分です。

このレビューの結果、論理仕様が使用可能になるはずです。

この時点での設計レビュー・プロセスでは、階層を設計し、データベースの設計を開始しています。

関連概念:

475 ページの『第 20 章 データ要件の分析』
121 ページの『第 13 章 全機能データベース・タイプ』
423 ページの『第 18 章 オプション・データベース機能』
487 ページの『第 21 章 全機能データベースの設計』

第 4 回の設計レビュー

第 4 回の設計レビューの主な目標は、システム・パフォーマンスが満足できるものであることを確認することにあります。

第 3 回の設計レビューが完了し、システム設計が本質的に完全であるという点ですべての関係者が納得したら、第 4 回の設計レビューを行います。このレビューでは特別の文書がレビューされることはありませんが、最終的な機能仕様と最初の論理仕様ないし最終的な論理仕様ができあがっています。

開発過程のこの時点では、設計に対して必要な調整を行うのに十分なだけの柔軟性がまだ残っています。それは、詳細設計は完了しているもののコードは存在しないからです。いったんコーディングが始まっても、何らかの設計変更が生じるのは確かでしょうが、その変更がこのシステム全体に影響を及ぼしてはなりません。この時点ではコードは存在しませんが、設計されたデータベースが期待どおりの結果をもたらすことを確認するためのテストを行うことができ、またそのようにすべきです。

第 4 回の設計レビューが正常に終了すると、データベース設計は完了したと見なされます。

すべての設計レビューに参加する人 (モデレーター、レビュー・チーム、データベース管理者、およびシステム設計者) は、第 4 回の設計レビューに参加すべきです。他の人たちは、特定の細部が必要な場合だけ参加します。

レビューの過程におけるこの時点では、データベース管理者は、設計およびテストと共にデータベース管理の作業を完了しています。

関連概念:

475 ページの『第 20 章 データ要件の分析』

121 ページの『第 13 章 全機能データベース・タイプ』

595 ページの『第 24 章 テスト・データベースの構築』

第 1 回コード検査

第 1 回コード検査の目標は、これまでに開発してきた論理が機能仕様を正しく解釈したことを確認することです。また、第 1 回コード検査は、パフォーマンス上の考慮点または問題点について確かめるために論理フローをレビューする機会となります。

システムの最終論理仕様が完了したあと、最初のコードの検査が行われます。

この時点ではコードは書かれていませんが、最終的な機能仕様の解釈は済んでいます。疑似コードとフローチャートには書くべきアセンブラー言語コードの 5 ないし 25 行ごとに、あるいは COBOL か PL/I コードの 5 ないし 15 行ごとに、1 つのステートメントあるいは論理枠が書かれています。さらに、モジュール・プロローグが書かれ、入り口と出口の論理およびすべてのデータ域が定義されています。

第 1 回コード検査が正常に終了すると、その出力として、承認された一連の最終論理仕様ができあがります。

第 1 回コード検査の参加者

第 1 回コード検査に参加するのは、主としてコーディング担当者です。すべての設計レビューに参加する人 (モデレーター、レビュー・チーム、データベース管理者、およびシステム設計者) も第 1 回コード検査に参加します。テスト担当者が参加して、コードの妥当性検査をするのに用いられるテスト・ケースを提示し、保守担当者が参加してデータベースの保守容易性についての情報収集および評価を行います。

この時点におけるデータベース管理者の役割は、これまでのように積極的なものではありません。データベース管理者が参加するのは、これまでのレビューにおいて定義されたデータの用法およびアクセス順序が守られていることを確認するためです。

レビュー・プロセスのこの時点では、データベース設計のインプリメンテーション、テスト・データベースの開発、およびデータベースのロードを始めています。

関連概念:

557 ページの『第 23 章 データベース設計のインプリメント』

595 ページの『第 24 章 テスト・データベースの構築』

603 ページの『第 25 章 データベースのロード』

第 2 回コード検査

第 2 回コード検査の目的は、モジュールの論理が疑似コードまたはフローチャートと合致していることを確認することです。インターフェースとレジスターの使用規則、およびコードの全般的な質が検査されます。コードの文書化と保守容易性が評価されます。

コーディングが完了したあと、テスト部門によるテストが始まる前に、第 2 回コード検査が行われます。

第 1 回コード検査に参加した人すべてが、第 2 回コード検査に参加すべきです。

このときのデータベース管理者の役割は、第 1 回コード検査での役割と同じです。

レビュー過程のこの時点においては、テスト・データベースの構築、データベース設計のインプリメンテーション、およびデータベースのロードというデータベース管理作業は終了しつつあります。

データベースのテストでは、DB モニターを実行して、各自で設定した期待されるパフォーマンスをデータベースが依然として満たしていることを確認する必要があります。

関連概念:

699 ページの『第 28 章 データベースのモニター』

セキュリティー検査

セキュリティー検査レビューの目的は、システム・インターフェース、保護されたデータベース、テーブル、または他の危険性の高い項目のセキュリティーに違反するコードがないか調べることです。

セキュリティーの検査を行うか否かはオプションですが、セキュリティーが重要な関心事である場合には、ぜひこれを行うようお勧めします。セキュリティー検査は、システム開発過程の適切な時点でいつでも実行できます。セキュリティーの方針は初期に定義して、設計レビューの間にそのインプリメンテーション状況を検査してください。ここで取り上げられているセキュリティー検査は、すべての単位テストと統合テストが完了した後に行われます。

このセキュリティー検査のレビューに参加する人の中には、モデレーター、システム設計者、セキュリティー担当者として指定された人、およびデータベース管理者が含まれます。データベース管理者はデータベースのセキュリティーをインプリメントし、これをモニターすることに責任を負っていますから、実際のところ、セキュリティー担当者として指定されることがあるかもしれません。セキュリティーが重要な関心事である場合には、レビュー・チームの人々にはこの検査に参加してほしくない場合もあります。

このセキュリティー検査や他のセキュリティー検査の間は、データベース管理者は、データベース管理作業、すなわちセキュリティーの確立に従事しています。

関連概念:

39 ページの『第 4 章 データベース・セキュリティー』

インプリメンテーション後レビュー

インプリメンテーション後レビューは、通常データベース・システムが稼働状態に入ってから約 6 カ月後に行われます。この目的は、このシステムがユーザーの要件を満たしていることを確認することです。

推奨事項: インプリメンテーション後レビューを実施してください。

該当のデータベース・システムの設計およびインプリメンテーションに関与したすべての人が、インプリメンテーション後レビューに参加すべきです。システムがユーザーの要件を満たしていない場合には、このレビューの出力として、ユーザーの要件を満たすように、設計またはパフォーマンス上の問題を訂正するための計画が作成されることになるはずです。

第 4 章 データベース・セキュリティー

データベース・セキュリティーには、ユーザー検査とユーザー権限の 2 つの側面があります。

ユーザー検査 とは、オンライン・データベースを使用する人が、実際に許可された人であることを確認する方法を意味します。

ユーザー権限 とは、ユーザーの ID を検査した後に、ユーザーが表示できる内容、およびその内容に対してユーザーが実行できる操作を制御する方法を意味します。

以下のトピックでは、主として後者の側面を取り上げます。すなわち、ユーザーのデータ視点、およびデータに関するユーザーのアクションをどのようにして制御することができるか説明します。

関連資料: CICS を使用する場合は、セキュリティーの確立について「*CICS Transaction Server for z/OS RACF Security Guide*」を参照してください。

関連概念:

38 ページの『セキュリティー検査』

データ・アクセスの範囲の制限

アプリケーション・プログラム用に定義したデータベースのビューを限定することにより、データベースのエレメントに対するユーザーのアクセス (さらにはエレメントの存在の認識) を制限できます。

PCB は、データベースのプログラムの視点 (したがって、ユーザーの視点) を定義します。 PCB は、DBD によって定義されているデータ構造を覆う「マスク」と考えることができます。PCB マスクによって、データ構造の特定の部分を隠すことができます。

40 ページの『処理権限の制限』の最初の図は、データベース管理者から見た、DBD で定義されているとおりの PAYROLL データベースの階層構造を示しています。一部のアプリケーションにとっては、SALARY セグメントにアクセスする必要はない (望ましくない) ことがあります。 SALARY セグメントの DB PCB で SENSEG ステートメントを省略すれば、このセグメントが単に存在しないかのようにすることができます。こうすることによって、そのセグメントへの無認可ユーザーのアクセスをリジェクトし、ユーザーがそのセグメントの存在自体を知るのをリジェクトしました。

この方法が成功するためには、マスクで隠されるセグメントは、アクセスされるセグメントの検索パスにあってはなりません。検索パスにあると、アプリケーションは少なくとも、「隠される」セグメントのキーを知ることになります。

フィールド・レベル・センシティビティーを用いれば、フィールド・レベルで同じマスキング効果を実現することができます。 SALARY と NAME が同じセグメン

トの中にあっても、このセグメントの中の他のフィールドへのアクセスはリジェクトせずに、SALARY フィールドへのアクセスを制限することができます。

処理権限の制限

ある一人のユーザーがアクセスできるデータの範囲を規制したあと、この範囲の中で権限を制御することもできます。

権限の制御は、任意のユーザーが、与えられたデータに対してどのような処理アクションを行うのかを決めることができます。例えば、あるアプリケーション・プログラムにはデータベースの中のセグメントを読み取る権限だけを与え、他のプログラムにはセグメントを更新したり削除したりする権限を与えることができます。

PROCOPT パラメーターによる処理権限の制御

プログラム・ビュー (PSB) を定義するときに、PROCOPT パラメーターを使用してユーザーの処理アクションを制御できます。 PROCOPT パラメーターは、データベースに対してどのようなアクションをとることを許すかを IMS に伝えます。プログラムは、PROCOPT で宣言されている事柄を実行することができます。

PROCOPT パラメーターは、PSB 生成ユーティリティーのマクロ命令をコーディングするときに SENSEG ステートメントと PCB ステートメントに指定できます。

必要なセグメントのみに限定するプログラム・センシティビティーの制限

アクセスと権限を制限するほかに、アプリケーション・プログラムがセンシティブなセグメントも制限できます。センシティブ・セグメントの数と指定された処理オプションがデータ可用性に影響を与える場合があります。データ可用性を最大限にするには、該当 PSB は必要なセグメントに対してだけセンシティブにし、処理オプションはできる限り制限の強いものにしなければなりません。

例えば、DBD 生成ユーティリティーの以下のマクロ命令内のデータベース定義は、従業員の名前、住所、職位、および給与を保管する給与データベースを記述しています。コードの後の図に、データベース・レコードの階層構造を示しています。

図 *13.* 給与データベースのデータベース定義の例

```
DBD   NAME=PAYROLL,...
DATASET ...
SEGM  NAME=NAME,PARENT=0...
FIELD NAME=
SEGM  NAME=ADDRESS,PARENT=NAME,...
FIELD NAME=
SEGM  NAME=POSITION,PARENT=NAME,...
FIELD NAME=
SEGM  NAME=SALARY,PARENT=NAME,...
FIELD NAME=
⋮
```

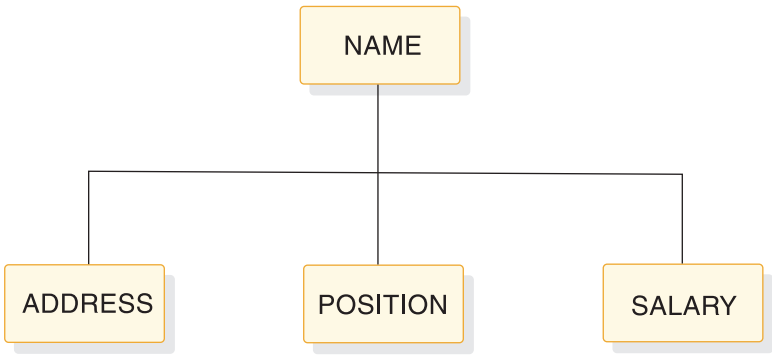


図 *14.* マスクのない給与データベース・レコード

アプリケーションが従業員の名前、住所、および職位にアクセスする必要があるが、給与にアクセスする必要はない場合、PSB 生成ユーティリティーの DB PCB マクロ命令内の SENSEG ステートメントを使用してアプリケーションを名前、住所、および職位のセグメントに対してのみセンシティブにすることができます。DB PCB の SENSEG ステートメントは、セグメントを隠す (アプリケーションから見えないようにする) ための、データベース・レコードを覆うマスクを作成します。以下のコードは、給与データベースの SALARY セグメントをマスクする DB PCB を示しています。

図 *15.* 給与データベースの *PCB* の例

```
PCB TYPE=DB,DBDNAME=PAYROLL,...
SENSEG NAME=NAME,PARENT=0,...
SENSEG NAME=ADDRESS,PARENT=NAME,...
SENSEG NAME=POSITION,PARENT=NAME,...
⋮
```

以下の図は、上記のプログラム・ビュー定義に基づいて給与データベース・レコードがアプリケーションから見てどのようになるかを示しています。これは、SALARY セグメントが隠されている点を除いて、前の図のデータベース・レコードと見かけは変わりません。

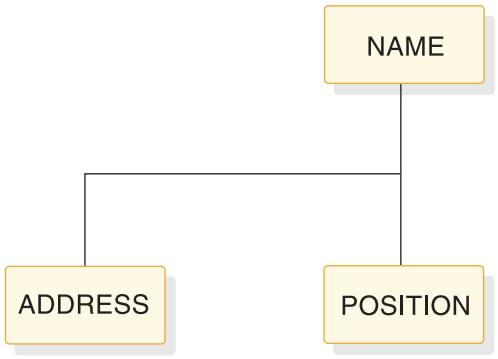


図 *16. SALARY* セグメントがマスクされた給与データベース・レコード

非 IMS プログラムによるアクセスの制限

機密漏れの可能性の 1 つは、非 IMS プログラムによって IMS データ・セットにアクセスを試みる人によります。このような機密漏れに対する 2 つの保護手段として、データ・セットのパスワードによる保護とデータベースの暗号化があります。

VSAM パスワードによるデータの保護

IMS データベースが保管されている VSAM データ・セットを非 IMS プログラムが読み取るのを防止するために、VSAM パスワードによる保護を利用することができます。

VSAM パスワードを使用してデータを保護するには、DBD ステートメントで PASSWD=YES をコーディングして、VSAM データ・セットに対するパスワード保護を有効にします。こうすれば、IMS は、DBD 名をパスワードとして渡します。DBD ステートメントで PASSWD=NO と指定すると、このデータ・セットがオープンされるたびに、VSAM にパスワードを与えるようにという指示がコンソールのオペレーターに出されます。

この方法はバッチ環境においてのみ役に立ち、また、オンライン・システムにおいては、VSAM パスワードの検査はすべて迂回されます。 (RACF® をインストールしている場合には、これを用いて VSAM データ・セットを保護することができます。)

関連資料:

DBD ステートメント (システム・ユーティリティー)

データベースの暗号化

DL/I データベースを暗号化して、非 IMS プログラムから読み取られないようにすることができます。

セグメント編集/圧縮出口から入るユーザー独自の暗号化ルーチンを用いて、DL/I セグメントを暗号化することができます。セグメントがデータベースに書き込まれる前に、IMS は制御をユーザー・ルーチンに渡し、このルーチンがセグメントを暗号化します。このあと、セグメントが取り出されるたびに、セグメントがアプリケーション・プログラムに渡される前にユーザー・ルーチンによってセグメントの暗号解読が行われます。

索引データベースまたは HISAM データベースのルート・セグメントにおいては、キーまたはキー・フィールドの位置は変更してはなりません。

関連概念:

427 ページの『セグメント編集/圧縮出口ルーチン』

セキュリティーの確立のためのディクショナリーの援用

ディクショナリー (IBM DB/DC データ・ディクショナリーなど) は、コンピューティング環境におけるエンティティー間の関係 (どのプログラムがどのデータ・エレメントを使用するかなど) をモニターするので、セキュリティーを管理するための理想的なツールとなります。

ディクショナリーを用いれば、使用許可マトリックスを定義することができます。拡張機能を利用すると、端末、プログラム、ユーザー、データ、およびこれら相互間の関係を定義することができます。このようにして、危険な傾向について示した報告書を作成することができます。誰がどの端末から何を使用するか、どのユーザーがどのデータにアクセスするかなどです。各ユーザーごとに、ディクショナリーを用いて、次のような情報をリストすることができます。

- 使用できるプログラム
- 入力できるトランザクションの種類
- 読み取ることのできるデータ・セット
- 変更できるデータ・セット
- ある 1 つのデータ・セット内で読み取ることのできるデータの種類
- 変更できるデータの種類

第 2 部 IMS カタログ

以下のトピックでは、IMS カタログを保守するための IMS カタログ・データベース・タスクおよび管理用タスクの目的と内容について説明します。

第 5 章 IMS カタログの概要

IMS カタログには、IMS に対して定義されている IMS データベースとアプリケーション・プログラム・ビューの信頼できるメタデータと定義が格納されます。

IMS カタログ自体は、HALDB PHIDAM データベースです。IMS に対して定義された各データベースおよびアプリケーション・プログラムは、IMS カタログ内の別個のレコードに保管されます。各レコードで、ルート・ヘッダー・セグメントは、含まれているリソースのタイプを識別します。リソースのタイプは、データベース定義 (DBD) またはプログラム・ビュー (PSB) のいずれかです。

階層内では、ヘッダー・セグメントの後に DBD セグメントまたは PSB セグメントが続きます。 DBD セグメントまたは PSB セグメントと、その従属セグメントには、データベースまたはプログラム・ビューの定義およびメタデータが保管されます。

その後のデータベースあるいはプログラム・ビューの定義は、DBD または PSB セグメントおよびその従属の追加インスタンスを挿入することで、前のデータベースあるいはプログラム・ビューと同じレコードに保管されます。データベースあるいはプログラム・ビューの異なるインスタンスは、タイム・スタンプによって区別されます。

レコード内のデータベースおよびプログラム・ビューのインスタンスには、オンライン IMS システムによって使用されているアクティブ・インスタンス、以前に定義されたインスタンス、さらに、これまでアクティブになったことがないドラフト・インスタンスが含まれます。

データベースのバージョン管理が使用されている場合、以前のバージョンのデータベースにアプリケーション・プログラムがアクセスできるようにするには、以前のバージョンのデータベースの DBD セグメントを、IMS カタログ内の DBD レコードに保持する必要があります。

IMS システムの ACBLIB には、そのシステム内のデータベースとアプリケーションの DBD と PSB が含まれています。しかし、IMS 管理者またはアプリケーション・プログラマーは、それらの情報に直接アクセスすることはできません。ACBGEN のとき、それらの情報を IMS カタログ・データベースに自動的に複製できます。 IMS カタログ・データベースに対する照会は、標準 DL/I 処理、Universal DL/I ドライバーを介した DL/I 処理、および Universal JDBC ドライバーを介した DL/I 処理を使用して行うことができます。IMS カタログ・データベース・レコードの更新、置換、挿入、および削除は、提供されたユーティリティーを使用した場合以外、行うことはできません。

一部のデータ・タイプは、情報が ACBLIB に保管されていないために、ACBLIB から生成されません。IMS はそれらの情報を、GSAM データベースおよび論理データベースの定義も含め、ACBGEN プロセスのときに DBDLIB および PSBLIB

から生成します。GSAM データベースは、GSAM と非 GSAM の両方のデータベースを含んでいる混合 PSB で参照されている場合にのみ、カタログに含まれています。

重要: IMS カタログが IMS システムの DFSDFxxx IMS PROCLIB メンバーで使用可能に設定されていない場合、ACBGENのときにカタログ情報は作成されません。

データは、16 進数 (タイプ X) として (数値の場合)、または Cp1047 EBCDIC 文字データとして IMS カタログ・データベースに保管されます。場合によっては、スペースを節約するために、値に切り捨てが行われます。フィールドにブランクが含まれている場合 (文字データの場合)、それは当該のフィールドがデータベースまたはプログラム仕様ブロックに適用されないことを示しています。個々のフィールド定義では、ブランク・フィールドに別の意味がある場合もあります。

IMS は、カタログ・データベースにアクセスするための基幹 COBOL コピーブックおよび基幹 PL/I プログラムを提供します。 COBOL コピーブックは DFS3DCBL という名前で、IMS.ADFSISRC にあります。PL/I サンプル・アプリケーションは DFS3DPL1 という名前で、IMS サンプル・ライブラリー IMS.ADFSSMPL にあります。

カタログが IMS.PROCLIB データ・セットの DFSDF*xxx* メンバーで使用可能に設定されている場合、IMS は IMS カタログの PCB リストを、実行時にそれぞれのユーザー PSB に自動的に追加します。

IMS は IMS カタログの 2 つの DBD を提供します。1 つはメイン IMS カタログ・データベース用で、もう 1 つは副次索引用です。IMS は、さまざまな目的とアプリケーション・プログラム・タイプのために、IMS カタログの複数の PSB を提供します。IMS カタログの DBD と PSB は、常駐として定義されます。

カタログ・データベース・セグメント・タイプは、そのセグメント・タイプがデータベース照会でアクセスされる頻度に基づいて、4 つの異なるデータ・セット・グループ (A から D) にグループ化されます。ルート・セグメント・タイプ (HEADER) と DBD および PSB セグメント・タイプは、データ・セット・グループ A にあります。ユーザー注釈など、アクセスされる頻度が最も小さいセグメント・タイプは、データ・セット D にグループ化されます。

HALDB PHIDAM データベースと同様に、IMS カタログは、標準の DL/I 処理、Universal DL/I ドライバーを介した DL/I 処理、および Universal JDBC ドライバーを介した SQL を使用して、照会することができます。 IMS カタログ・データベース・レコードの更新、置換、挿入、および削除は、提供されたユーティリティーを使用した場合以外、行うことはできません。

関連概念:

IMS カタログの定義と調整 (システム定義)

関連タスク:

IMS カタログの DBD および PSB のインストール (システム定義)

関連資料:

IMS カタログ・データ・セット・グループ (システム定義)

- IMS カタログ・ユーティリティー (システム・ユーティリティー)
- IMS カタログ Redpaper

第 6 章 IMS カタログのバックアップとリカバリー

IMS カタログは HALDB データベースであるため、標準の HALDB バックアップおよびリカバリー手順を使用して、IMS カタログのバックアップおよびリカバリーを行うことができます。ただし、リカバリーした IMS カタログ内のレコードが、IMS システム内でアクティブなアプリケーション制御ブロック (ACB) に一致することを確認する必要があります。

データベースのバージョン管理など、特定の IMS 機能では、バックアップ・コピーから IMS カタログをリカバリーする機能が必要です。これらの機能を使用しない場合、IMS カタログを ACB、DBD、および PSB のライブラリーから再作成するデータ追加ユーティリティーのいずれかを使用して、リカバリーすることができます。

リカバリーにバックアップ・コピーが必要な機能の 1 つは、データベースのバージョン管理です。データベース・バージョン管理が使用される場合、以前のバージョンのデータベースの DBD は、IMS カタログ内にのみ存在し、ACB ライブラリーから再作成することはできません。アクティブとして指定されているデータベース定義 (DBD) のバージョンのみが、アクティブ ACB ライブラリーから再作成することができます。

IMS カタログ・データ・セット

IMS カタログは、HALDB 区分 HIDAM (PHIDAM) データベースで、1 次索引データ・セット、間接リスト・データ・セット (ILDS) を備えています。 IMS カタログには、HALDB 区分副次索引 (PSINDEX) データベースが含まれます。

アクティブ ACB を使用した IMS カタログ・リカバリーの調整

バックアップ・イメージ・コピーから IMS カタログをリカバリーする場合は、リカバリーした IMS カタログをアクティブ ACB ライブラリーと同期させる必要があります。IMS カタログ内のレコードのタイム・スタンプは、アクティブ ACB ライブラリー内の対応する ACB メンバーのタイム・スタンプと一致する必要があります。

DBRC、IMS リカバリー・ユーティリティー、および IMS カタログ

IMS で提供されるイメージ・コピー・ユーティリティーやデータベース・リカバリー・ユーティリティーを含め、IMS のすべての標準ユーティリティーは、カタログ・データ・セットに対して実行できます。

IMS カタログが DBRC によって管理されている場合、IMS で提供されるユーティリティーは IMS カタログが更新されたときにログ・データ・セット内にリカバリー情報を作成します。DBRC は、IMS カタログのリカバリーに必要なログ、イメージ・コピー、および JCL を管理します。お客様は、IMS カタログ・パーティションの完全なデータベース・リカバリーを実行するか、ポイント・イン・タイム・リカバリーを実行できます。

IMS カタログに DBRC を使用しない場合でも、他の HALDB データベースに使用される標準のバックアップ・プロセスとリカバリー・プロセスを使用して、IMS カタログをリカバリーすることができます。ただし、ログ、イメージ・コピー、JCL などを管理するためのプロセスを正しく配置しておく必要があります。

再作成によるリカバリー

インストール済み環境でデータベースのバージョン管理を使用しない場合、イメージ・コピーおよびログ・レコードから IMS カタログをリカバリーする代わりに、IMS Catalog Populate ユーティリティー (DFS3PU00) または ACB Generation and Catalog Populate ユーティリティー (DFS3UACB) を使用して IMS を再ロードすることで、ACB、DBD、および PSB ライブラリーから IMS カタログを再作成することができます。

いずれかのデータ追加ユーティリティーを使用して ACB、DBD、および PSB のライブラリーから IMS カタログを再作成する場合、ライブラリー内にある IMS カタログのレコード・セグメントのみがリストアされます。この方法は、DBD および PSB の過去のインスタンスのレコード・セグメント、およびそれらのセグメント・インスタンスに含まれる履歴メタデータがいずれも必要ない場合にのみ使用してください。

いずれかのデータ追加ユーティリティーによるカタログの初期ロードでは、ログ・データ・セット内にデータベース・リカバリー情報は作成されません。 IMS カタログをロードした直後にイメージ・コピーを作成して、イメージ・コピーとアクティブ ACB ライブラリーを必ず整合させてください。

関連概念:

646 ページの『データベース・バックアップ・コピーの作成』

661 ページの『データベースのリカバリー』

639 ページの『第 26 章 データベースのバックアップおよびリカバリー』

関連資料:

IMS カタログ・ユーティリティー (システム・ユーティリティー)

IMS カタログのバックアップ方式

IMS カタログは HALDB PHIDAM データベースなので、IMS イメージ・コピー・ユーティリティーを使用して、メイン・データベースおよび副次索引の区分データ・セットをバックアップすることができます。

他の HALDB データベースのリカバリーと同様に、IMS カタログのリカバリーは、1 次索引データ・セットおよび間接リスト・データ・セット (ILDS) のいずれのイメージ・コピーも使用しません。代わりに、これらのデータ・セットは、HALDB Index/ILDS Rebuild ユーティリティー (DFSPREC0) を使用して再作成することでリカバリーされます。

IMS イメージ・コピー・ユーティリティーが使用され、IMS カタログが DBRC によって管理されている場合、イメージ・コピーおよび必要なログ・データ・セットは DBRC によって管理されます。

IMS カタログが IMS システム間で共用されている場合、IMS カタログを変更累積 (CA) グループに追加して、すべての IMS システムからのログをリカバリーの前にマージできるようにする必要があります。

関連概念:

646 ページの『データベース・バックアップ・コピーの作成』

639 ページの『第 26 章 データベースのバックアップおよびリカバリー』

関連資料:

- バックアップ・ユーティリティー (データベース・ユーティリティー)
- IMS カタログ・ユーティリティー (システム・ユーティリティー)

関連情報:

第 7 章 IMS カタログの保守

IMS カタログに多くの保守は必要ありませんが、IMS カタログは IMS HALDB 全機能データベースであるため、他の HALDB 全機能データベースに適用される保守上の考慮事項が IMS カタログにも適用されます。

IMS カタログを再編成あるいは保守する頻度は、IMS 環境内の新規あるいは変更されたデータベースおよびアプリケーション・プログラムについて、インストール済み環境で IMS カタログ内の PSB レコードおよび DBD レコードを追加あるいは更新する頻度によって異なります。

IMS カタログを再編成するには、HALDB Online Reorganization (OLR) を使用して、IMS カタログがオフラインにならないようにします。 IMS カタログは標準的な HALDB データベースより小さいため、IMS カタログで OLR を実行しても、パフォーマンスやリソースに大きな影響はありません。

IMS カタログをオフラインにすることができる場合は、HD Reorganization Unload ユーティリティー (DFSURGU0) および HD Reorganization Reload ユーティリティー (DFSURGL0) を使用して IMS カタログを再編成することができます。

不要になった PSB および DBD のインスタンスを IMS カタログから削除するには、IMS Catalog Record Purge ユーティリティー (DFS3PU10) を使用することができます。

関連概念:

57 ページの『第 8 章 IMS カタログからの DBD インスタンスおよび PSB インスタンスの削除』

関連資料:

HD 再編成アンロード・ユーティリティー (DFSURGU0) (データベース・ユーティリティー)

HD 再編成再ロード・ユーティリティー (DFSURGL0) (データベース・ユーティリティー)

IMS Catalog Record Purge ユーティリティー (DFS3PU10) (システム・ユーティリティー)

第 8 章 IMS カタログからの DBD インスタンスおよび PSB インスタンスの削除

IMS Catalog Record Purge ユーティリティー (DFS3PU10) を使用して、個別の DBD インスタンスおよび PSB インスタンスを表すセグメントを、IMS カタログ内の DBD レコードおよび PSB レコードから削除することができます。

また、IMS Catalog Record Purge ユーティリティーを使用して、特定の DBD バージョンのすべての DBD インスタンスを IMS カタログ内の DBD レコードから削除することができます。DBD バージョンを削除すると、その DBD バージョンのすべての DBD インスタンスが DBD レコードから削除されます。削除された後、その DBD のバージョンは、もう IMS カタログに存在しません。

まだ必要な DBD セグメントまたは PSB セグメントを誤って削除しないようにするために、IMS カタログ内の DBD レコードおよび PSB レコードの保存基準を定義することができます。IMS Catalog Record Purge ユーティリティーの分析機能は、カタログ内の各レコードの有効な保存基準に基づいて、削除することができる DBD または PSB セグメント・インスタンスを特定し、DELETE ステートメントを作成します。

DBD レコードまたは PSB レコードに固有の保存基準は、IMS Catalog Record Purge ユーティリティーの UPDATE 制御ステートメントによって設定され、レコードの HEADER セグメントに格納されます。特定のレコードに保存基準が指定されていない場合、カタログ・レコードは、IMS.PROCLIB データ・セットの DFSDF*xxx* メンバーの CATALOG セクション内にある RETENTION ステートメントで設定されたデフォルトの保存基準に従います。

保存基準には、IMS が DBD レコードまたは PSB レコードに保存する必要がある最小セグメント・インスタンス数と、セグメント・インスタンスが削除可能になるまでの最小保存期間があります。デフォルトで、IMS は、レコードに最小インスタンス数の DBD および PSB を保存します。インスタンスを保存する必要がある期間についてのデフォルト設定はありません。この期間は、日数で設定します。

例えば、レコード内の DBD セグメントまたは PSB セグメントのインスタンス数が、そのレコードに設定されている保存インスタンス数以下である場合、インスタンスを削除することはできません。DBD または PSB インスタンスが IMS カタログ内にある日数が、それらが含まれている DBD または PSB レコードに定義されている保存期間と同じかまたはそれより短い場合は、DBD または PSB インスタンスを削除することはできません。

DBD および PSB レコード内のセグメント・インスタンスの保存基準を定義する際には、インスタンスが増えると、IMS カタログに必要となるストレージの量も増えることに注意してください。

関連資料:

61 ページの『HEADER セグメントのフォーマット』

IMS Catalog Record Purge ユーティリティー (DFS3PU10) (システム・ユーティリティー)

DFSDFxxx メンバーの CATALOG および CATALOGxxxx セクション (システム定義)

第 9 章 未登録の IMS カタログを用いた HALDB ユーティリティーの使用

すべての HALDB ユーティリティーは、RECON データ・セットに登録されており、DBRC によって管理されている IMS カタログ・データベースでサポートされています。未登録のカタログには、いくつかの制約事項があります。

IMS Catalog Partition Definition Data Set ユーティリティーを使用して、DBRC によって管理されていない IMS カタログ・データベースを構成することができます。一部の標準 HALDB ユーティリティーは、特定の制約事項がありますが、未登録のカタログ・データベースに使用できます。

データベース・イメージ・コピー・ユーティリティー **(DFSUDMP0)**
: このユーティリティーを使用して、未登録の IMS カタログ・データベースのバッチ・イメージ・コピーを作成できます。未登録の IMS カタログ・データベースでは、並行イメージ・コピーはサポートされません。また、未登録のカタログ・データベースでは動的データ・セット割り振りがサポートされないため、Datain DD ステートメントを指定する必要があります。

  注**:** IMS カタログ・データベースのリカバリーについて詳しくは、 51 ページの『第 6 章 IMS カタログのバックアップとリカバリー』を参照してください。

バッチ・バックアウト・ユーティリティー **(DFSBBO00)**
データベース・リカバリー・ユーティリティー **(DFSURDB0)**
HALDB 索引**/ILE** データ・セット再作成ユーティリティー **(DFSPREC0)**
HD 再編成アンロード・ユーティリティー **(DFSURGU0)**
HD 再編成再ロード・ユーティリティー **(DFSURGL0)**
: これらのユーティリティーは未登録の IMS カタログ・データベースにも使用できますが、ユーティリティーの EXEC ステートメントに DFSDF= パラメーターを組み込む必要があります。 DFSDF パラメーターは、未登録の IMS カタログ・データベースを指定する IMS.PROCLIB データ・セットの DFSDF*xxx* メンバーに付加する 3 文字の接尾部を指定します。DFSDF*xxx* メンバーは、DATABASE ステートメントの UNREGCATLG パラメーターで未登録の IMS カタログ・データベース名を指定します。

HALDB 区画データ・セット初期設定ユーティリティー **(DFSUPNT0)**
: このユーティリティーは、未登録の IMS カタログ・データベースとは互換性がありません。Catalog Populate ユーティリティー (DFS3PU00) は、類似したサポートを登録済みと未登録の IMS カタログ・データベースに対して提供します。

関連資料:

HALDB 索引/ILDS 再作成ユーティリティー (DFSPREC0) (データベース・ユーティリティー)

HD 再編成アンロード・ユーティリティー (DFSURGU0) (データベース・ユーティリティー)

- HD 再編成再ロード・ユーティリティー (DFSURGL0) (データベース・ユーティリティー)
- データベース・リカバリー・ユーティリティー (DFSURDB0) (データベース・ユーティリティー)
- データベース・イメージ・コピー・ユーティリティー (DFSUDMP0) (データベース・ユーティリティー)
- IMS カタログ・ユーティリティー (システム・ユーティリティー)

第 10 章 IMS カタログ・データベース内のレコードのフォーマット

IMS カタログ・データベースには、ACB 生成時に定義された各 PSB および DBD に固有のレコードが含まれています。各レコード・タイプおよびレコード内の各セグメント・タイプには、フォーマットが事前定義されています。

カタログ・データベース・セグメント・タイプは、そのセグメント・タイプがデータベース照会でアクセスされる頻度に基づいて、4 つの異なるデータ・セット・グループ (A から D) にグループ化されます。ルート・セグメント・タイプ (HEADER) など、最も頻繁にアクセスされるセグメント・タイプは、データ・セット・グループ A に配置されます。ユーザー注釈セグメントなど、アクセスされる頻度が最も小さいセグメント・タイプは、データ・セット D にグループ化されます。

DBDHXXX など、カタログ・データベース内の一部のセグメント・タイプは現時点では使用されておらず、将来の開発用に予約されています。セグメント定義は、カタログ・データベースのアンロードや再ロードを行わなくても将来のサービスと開発の拡張を実装できるよう、カタログに組み込まれています。

関連資料:

IMS カタログ・データ・セット・グループ (システム定義)

HEADER セグメントのフォーマット

IMS カタログの HEADER セグメント・タイプ (リソース・ヘッダーとも呼ばれます) は、IMS カタログ・データベースのルート・セグメント・タイプです。

IMS カタログ・データベース・レコードのリソース・ヘッダーは、そのレコードに保管されているメタデータのタイプに関する情報を含んでいます。リソース・ヘッダーは、特定の IMS カタログ・レコードが DBD または PSB メタデータを含んでいるかどうか、およびカタログ・レコードが記述しているリソースの IMS 名および別名を含んでいるかどうかを示します。

カタログ・レコードのルート・キーは、このセグメント内の RHDRSEQ フィールドの値です。このキー値は、IMS Catalog Populate ユーティリティー (DFS3PU00) または ACB Generation and Catalog Populate ユーティリティー (DFS3UACB) によって生成されます。値は、リソースのレコード・タイプと IMS メンバー名を連結して作成されます。レコード・タイプは長さが 8 文字で、右側にブランク文字が埋め込まれます。IMS メンバー名は、常に長さが 8 文字です。

例えば、ACF12000 という名前の DBD レコードのルート・キーは、以下のとおりです。

```
DBD     ACF12000
```

MXG88888 という名前の PSB レコードのルート・キーは、以下のとおりです。

```
PSB     MXG88888
```

また、ルート・キー値は、カタログ・データベースが複数の区画から構成されている場合に、カタログ・レコードをデータベース区画にソートするためにも使用されます。データベース内の最後の区画の区画ハイ・キーは、カタログ内の最高のハイ・キー・レコードを含めるのに十分な値であることが必要です。

セグメント名
HEADER

親名　適用外

順序フィールド
RHDRSEQ

セグメント長
56 バイト

表 *4. HEADER* セグメント・マップ :

オフセット (バイト)		長さ (バイト)	データ・タイプ	フィールド名	説明	固有キー・フィールド
1		2	X	LEN	このリソース・ヘッダー・セグメントの長さ	
3		2	X	CTL	制御フィールド	
5		2	X	SEQNUM	セグメント・シーケンス番号	
9		16	C	RHDRSEQ	シーケンス・フィールド、タイプ = U	X
	9	8	C	TYPE	このカタログ・レコード内のリソース・メタデータのタイプ	
	17	8	C	IMSNAME	このカタログ・レコード内で記述されているリソースの名前	
25		4	X	RETNINST	レコード・セグメントが削除される際にこのレコード内に保持する必要がある、DBD または PSB の最小インスタンス数。この値は、IMS Catalog Record Purge ユーティリティーの UPDATE ステートメントによって変更されます。このフィールドに値を指定しなかった場合、ユーティリティーは IMS.PROCLIB データ・セットの DFSDF*xxx* メンバーの CATALOG セクション内の RETENTION ステートメントで指定された値を使用します。	

表 4. *HEADER* セグメント・マップ (続き) :

オフセット (バイト)	長さ (バイト)	データ・タイプ	フィールド名	説明	固有キー・フィールド
29	4	X	RETNDAYS	このレコードの DBD または PSB の各インスタンスを保持する最小日数。この日数が経過すると、インスタンスを削除できるようになります。この値は、IMS Catalog Record Purge ユーティリティーの UPDATE ステートメントによって変更されます。このフィールドの値を指定しなかった場合、次の 2 つのシナリオが考えられます。 • RETNINST フィールドに 1 以上の値が指定されている場合、IMS Catalog Record Purge ユーティリティーは、この DBD または PSB のインスタンスを経過時間に基づいてパージしません。 • RETINST フィールドに 0 が指定されている場合、IMS Catalog Record Purge ユーティリティーは、DFSDF*xxx* メンバーで指定された値を使用して、このレコードの DBD または PSB インスタンスの保存基準を決定します。	
33	8	C	FILLER1	予約済み	
41	7	C	ACTTS	アクティブ・レコードのタイム・スタンプ。アクティブ・メンバーのタイム・スタンプを識別します。	
48	7	C	PNDTS	保留中レコードのタイム・スタンプ。保留中メンバーのタイム・スタンプを識別します。	
55	2	C	FILLER2	予約済み	

DBD レコード・セグメントのフォーマット

IMS カタログ内の DBD レコード・セグメントは、IMS データベース定義 (DBD) に関する情報を保管するために使用されます。

次の図は、DBD の IMS カタログ・レコードの編成を大まかに示したものです。

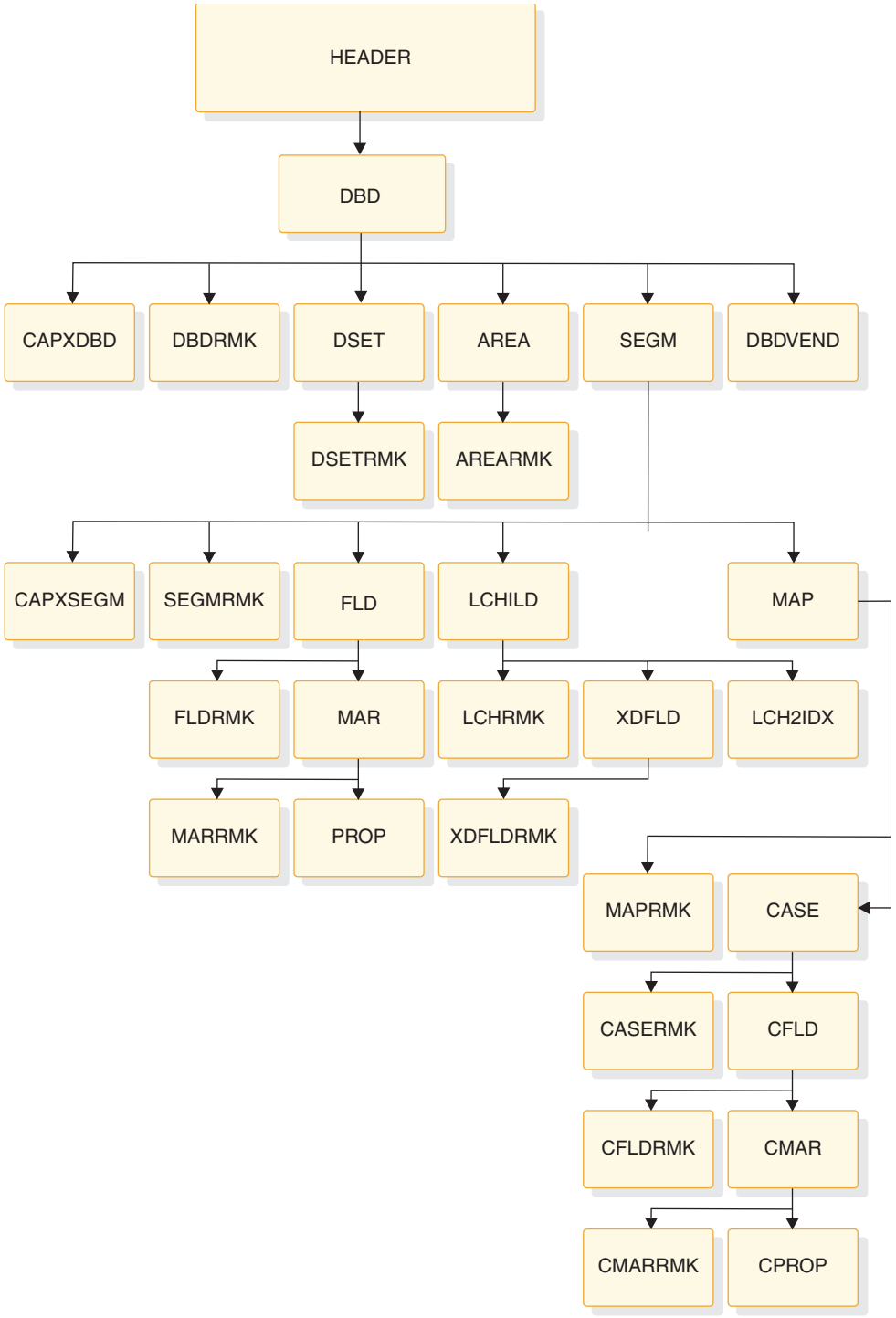


図 17. データベース記述の *IMS* カタログ・レコードのフォーマット

AREA セグメント・タイプのフォーマット

IMS カタログの AREA セグメント・タイプは、高速機能データベース内のデータベース・エリアに関する情報を含んでいます。

このセグメント・タイプは、高速機能データベースの IMS カタログ・レコード内でのみ使用されます。

セグメント名
AREA

親名　DBD

順序フィールド
AREASEQ

セグメント長
40 バイト

表 *5. AREA* セグメント・タイプのフォーマット：

オフセット (バイト)	長さ (バイト)	データ・タイプ	フィールド名	説明	固有キー・フィールド
1	2	X	LEN	このセグメントの長さ。	
3	2	X	CTL	制御フィールド。	
5	2	X	AREASEQ	シーケンス・フィールド、タイプ = U。	X
5	2	X	SEQNUM	シーケンス番号。	
9	8	C	DD1	このデータベース・エリアのデータ・セット名。	
17	2	X	SIZE	このデータベース・エリアの制御インターバルのサイズ (バイト単位)。	
19	2	X	UOW1	このデータベース・エリアの 1 作業単位における制御インターバルの数。	
21	2	X	UOW2	このデータベース・エリアの 1 作業単位のオーバーフロー・セクションにおける制御インターバルの数。	
23	2	X	ROOT1	このデータベース・エリアのルート・アドレス可能セクションに割り振られた合計スペース。この値は、UOW1 フィールドで指定された UOW サイズの作業単位 (UOW) の数で指定されます。	
25	2	X	ROOT2	このデータベース・エリア内で独立オーバーフローに割り振られた合計スペース。この値は、UOW2 フィールドで指定された UOW サイズの作業単位 (UOW) の数で指定されます。	
27	14	C	FILLER	予約済み。	

関連概念:

560 ページの『AREA ステートメントの概説』

関連資料:

AREA ステートメント (システム・ユーティリティー)

AREARMK セグメント・タイプのフォーマット

IMS カタログの AREARMK セグメント・タイプは、高速機能データベースのデータベース・エリアに関するユーザーのコメントを含んでいます。

このセグメントは、コメントの対象である AREA セグメント・インスタンスの直接の子です。

セグメント名
AREARMK

親名　AREA

順序フィールド
ARCMSEQ

セグメント長
264 バイト

表 *6. AREARMK* セグメント・マップ：

オフセット (バイト)	長さ (バイト)	データ・タイプ	フィールド名	説明	固有キー・フィールド
1	2	X	LEN	このセグメントの長さ	
3	2	X	CTL	制御フィールド	
5	2	X	ARCMSEQ	シーケンス・フィールド、タイプ = U	X
5	2	X	SEQNUM	シーケンス番号	
7	2	C	FILLER	予約済み	
9	256	C	REMARKS	このデータベース・エリアに関するユーザーのコメント	

CAPXDBD セグメント・タイプのフォーマット

IMS カタログの CAPXDBD セグメントは、DBD によって使用されるデータ・キャプチャー出口ルーチンに関する情報を含んでいます。

このセグメント内のメタデータは、出口ルーチンの名前と処理オプションを含んでいます。複数の DBD が単一のデータ・キャプチャー出口を参照でき、単一の DBD が複数のデータ・キャプチャー出口を参照できます。後者の場合、単一の親 DBD セグメント・インスタンスに対して、このセグメント・タイプの複数の子インスタンスが存在します。

セグメント名
CAPXDBD

親名　DBD

順序フィールド
DDCAPSEQ

セグメント長
32 バイト

表 7. *CAPXDBD* セグメント・マップ：

オフセット (バイト)	長さ (バイト)	データ・タイプ	フィールド名	説明	固有キー・フィールド
1	2	X	LEN	セグメント長	
3	2	X	CTL	制御フィールド	
5	2	X	DDCAPSEQ	シーケンス・フィールド、タイプ = U	X
5	2	X	SEQNUM	シーケンス番号	
9	8	C	EXITNAME	このデータ・キャプチャー出口ルーチンのモジュール名	
17	1	C	LOG	データ・キャプチャー出口ルーチン制御ブロックとデータを IMS システム・ログに書き込むかどうかを示します。	
18	1	C	KEY	データ・キャプチャー出口ルーチンを呼び出すときに、更新されたセグメントの物理連結キーを出口ルーチンに渡すかどうかを示します。	
19	1	C	PATH	物理ルート・セグメントの階層パスに含まれる各セグメントからのデータを出口ルーチンに渡すかどうかを示します。	
20	1	C	DATA	データ・キャプチャー出口ルーチンに物理セグメント・データを渡すかどうかを示します。	
21	1	C	BEFORE	REPL 呼び出しについて書き込まれるタイプ X'99' ログ・レコードに、事前データを含めるかどうかを示します。	
22	1	C	DLET	DLET 呼び出しのタイプ X'99' ログ・レコードを書き込むかどうかを示します。	
23	1	C	CASCADE	DL/I 呼び出しで、親セグメントの削除の結果としてこのセグメントが削除される場合に、このデータ・キャプチャー出口ルーチンを呼び出すかどうかを示します。	
24	1	C	CKEY	カスケード削除操作の結果として実行される呼び出しのときに、物理連結キーを出口ルーチンに渡すかどうかを示します。	
25	1	C	CPATH	カスケード削除操作の結果として実行される呼び出しのときに、物理ルート・セグメントの階層パスに含まれる各セグメントからのデータを出口ルーチンに渡すかどうかを示します。	
26	1	C	CDATA	カスケード削除操作の結果として実行される呼び出しのときに、物理セグメント・データを出口ルーチンに渡すかどうかを示します。	
27	1	C	CBEFORE	DEDB に対する REPL 呼び出しについて書き込まれるタイプ X'99' ログ・レコードに、事前データを含めるかどうかを示します。	
28	1	C	CDLET	DEDB に対する DLET 呼び出しのタイプ X'99' ログ・レコードを書き込むかどうかを示します。	
29	4	C	FILLER	予約済み	

関連タスク:

431 ページの『データ・キャプチャー出口ルーチン用の DBD パラメーター』

関連資料:

DBD ステートメント (システム・ユーティリティー)

CAPXSEGM セグメント・タイプのフォーマット

IMS カタログの CAPXSEGM セグメント・タイプは、データベース・セグメント用に指定されたデータ・キャプチャー出口ルーチンに関する情報を含んでいます。

このセグメント内のメタデータは、出口ルーチンの名前と処理オプションを含んでいます。複数のセグメントが単一のデータ・キャプチャー出口を参照でき、単一のセグメントが複数のデータ・キャプチャー出口を参照できます。後者の場合、単一の親 SEGM セグメント・インスタンスに対して、このセグメント・タイプの複数の子インスタンスが存在します。

セグメント名
: CAPXSEGM

親名
: SEGM

順序フィールド
: SDCAPSEQ

セグメント長
: 32 バイト

表 *8. CAPXSEGM* セグメント・マップ:

オフセット (バイト)	長さ (バイト)	データ・タイプ	フィールド名	説明	固有キー・フィールド
1	2	X	LEN	このセグメントの長さ	
3	2	X	CTL	制御フィールド	
5	2	X	SDCAPSEQ	シーケンス・フィールド、タイプ = U	X
5	2	X	SEQNUM	シーケンス番号	
9	8	C	EXITNAME	この出口ルーチンのモジュール名	
17	1	C	LOG	データ・キャプチャー出口ルーチン制御ブロックとデータを IMS システム・ログに書き込むかどうかを示します。	
18	1	C	KEY	データ・キャプチャー出口ルーチンを呼び出すときに、更新されたセグメントの物理連結キーを出口ルーチンに渡すかどうかを示します。	
19	1	C	PATH	物理ルート・セグメントの階層パスに含まれる各セグメントからのデータを出口ルーチンに渡すかどうかを示します。	
20	1	C	DATA	データ・キャプチャー出口ルーチンに物理セグメント・データを渡すかどうかを示します。	

表 *8. CAPXSEGM* セグメント・マップ *(*続き*)*:

オフセット (バイト)	長さ (バイト)	データ・タイプ	フィールド名	説明	固有キー・フィールド
21	1	C	BEFORE	REPL 呼び出しについて書き込まれるタイプ X'99' ログ・レコードに、事前データを含めるかどうかを示します。	
22	1	C	DLET	DLET 呼び出しのタイプ X'99' ログ・レコードを書き込むかどうかを示します。	
23	1	C	CASCADE	DL/I 呼び出しで、(カスケード削除操作のときに) 親セグメントの削除の結果としてこのセグメントが削除される場合に、このデータ・キャプチャー出口ルーチンを呼び出すかどうかを示します。	
24	1	C	CKEY	カスケード削除操作の結果として実行される呼び出しのときに、物理連結キーを出口ルーチンに渡すかどうかを示します。	
25	1	C	CPATH	カスケード削除操作の結果として実行される呼び出しのときに、物理ルート・セグメントの階層パスに含まれる各セグメントからのデータを出口ルーチンに渡すかどうかを示します。	
26	1	C	CDATA	カスケード削除操作の結果として実行される呼び出しのときに、物理セグメント・データを出口ルーチンに渡すかどうかを示します。	
27	1	C	CBEFORE	DEDB に対する REPL 呼び出しについて書き込まれるタイプ X'99' ログ・レコードに、事前データを含めるかどうかを示します。	
28	1	C	CDLET	DEDB に対する DLET 呼び出しのタイプ X'99' ログ・レコードを書き込むかどうかを示します。	
29	4	C	FILLER	予約済み	

関連タスク:

431 ページの『データ・キャプチャー出口ルーチン用の DBD パラメーター』

関連資料:

SEGM ステートメント (システム・ユーティリティー)

CASE セグメント・タイプのフォーマット

IMS カタログの CASE セグメント・タイプは、IMS データベース・セグメントのマッピングの特定のケースに関する情報を含んでいます。

セグメント名
: CASE

親名 MAP

順序フィールド
: CASESEQ

セグメント長
656 バイト

表 *9. CASE* セグメント・タイプのフォーマット：

オフセット (バイト)	長さ (バイト)	データ・タイプ	フィールド名	説明	固有キー・フィールド
1	2	X	LEN	このセグメントの長さ	
3	2	X	CTL	制御フィールド	
5	2	X	CASESEQ	シーケンス・フィールド、タイプ = U	X
5	2	X	SEQNUM	シーケンス番号	
7	2	C	FILLER	予約済み	
9	1	C	CASETYPE	CASEID フィールドのエンコード・タイプ。このフィールドは、Cp1047 EBCDIC エンコードを表す C、または 16 進バイナリー表現を表す X のいずれかを指定します。	
10	7	C	FILLER01	予約済み	
17	128	C	CASENAME	このケースの名前	
145	128	C	CASEID	このケースの固有 ID。このフィールドは、CASETYPE フィールドの値に基づいて解釈してください。	
273	128	C	MAPNAME	このケースが属するセグメント・タイプ・マッピングの名前	
401	256	C	FILLER02	予約済み	

関連資料:

DFSCASE ステートメント (システム・ユーティリティー)

CASERMK セグメント・タイプのフォーマット

IMS カタログの CASERMK セグメント・タイプは、セグメント・タイプ・マッピングのケースに関するユーザー指定のコメントを含んでいます。

このセグメントは、コメントの対象である CASE セグメント・インスタンスの直接の子です。

セグメント名
CASERMK

親名 CASE

順序フィールド
CASCMSEQ

セグメント長
264 バイト

表 *10. CASERMK* セグメント・マップ：

オフセット (バイト)	長さ (バイト)	データ・タイプ	フィールド名	説明	固有キー・フィールド
1	2	X	LEN	このセグメントの長さ	
3	2	X	CTL	制御フィールド	
5	2	X	CASCMSEQ	シーケンス・フィールド、タイプ = U	X
5	2	X	SEQNUM	シーケンス番号	
7	2	C	FILLER	予約済み	
9	256	C	REMARKS	親 CASE セグメントによって記述されたケース定義に関するユーザーのコメント	

CFLD セグメント・タイプのフォーマット

IMS カタログの CFLD セグメント・タイプは、特定のセグメント・タイプ・フォーマット・ケース内のフィールドに関する情報を含んでいます。

CFLD セグメントの各インスタンスは、あるセグメント・タイプ・フォーマット内の 1 つのケースの 1 つのフィールドを記述しています。CFLD セグメント・インスタンス内の情報は、ユーザー・データベース・セグメントが親 CASE セグメント・インスタンス内で定義されているマッピング・ケースを使用してマップされている場合にのみ、そのユーザー・データベース・セグメントに対して有効です。

セグメント名
: CFLD

親名 CASE

順序フィールド
: FIELDSEQ

セグメント長
: 904 バイト

表 *11. CFLD* セグメント・タイプのフォーマット：

オフセット (バイト)	長さ (バイト)	データ・タイプ	フィールド名	説明	固有キー・フィールド
1	2	X	LEN	このセグメントの長さ	
3	2	X	CTL	制御フィールド	
5	2	X	FIELDSEQ	シーケンス・フィールド、タイプ = U	X
5	2	X	SEQNUM	シーケンス番号	
9	8	C	IMSNAME	このフィールドの 8 文字の IMS 名	
17	3	C	NAMESEQ	このフィールドがシーケンス・フィールドであるかどうかを示します。	

表 *11. CFLD* セグメント・タイプのフォーマット *(*続き*)*:

オフセット (バイト)	長さ (バイト)	データ・タイプ	フィールド名	説明	固有キー・フィールド
20	1	C	SEQUM	このフィールドが固有シーケンス・フィールド (U) であるか、非固有シーケンス・フィールド (M) であるかを示します。	
21	2	X	BYTES	このフィールドの長さ (バイト単位)	
23	2	X	START	セグメントの先頭から数えた、このフィールドの開始オフセット (バイト単位)。このフィールドにデータが入っている場合、STARTAFT フィールドは使用されません。	
25	1	C	TYPE	このフィールドの空きスペースに埋め込むために IMS が使用するバイナリー・データのタイプを示します。 **X** 左に X'00' を埋め込み。 **P** 左に X'00' を埋め込み。 **C** 右に X'40' を埋め込み。 **F** 2 進フルワード・データ。MSDB 専用。 **H** 2 進ハーフワード・データ。MSDB 専用。	
26	15	C	FILLER01	予約済み	
41	9	C	DATATYPE	フィールドの外部 (非 IMS) データ・タイプ	
50	3	C	FILLER02	予約済み	
53	2	X	PRECISN	10 進データ・タイプのフィールドの精度	
55	2	X	SCALE	10 進データ・タイプのフィールドの位取り	
57	4	X	MINOCCUR	DATATYPE=ARRAY フィールド内のエレメントの最小数	
61	4	X	MAXOCCUR	DATATYPE=ARRAY フィールド内のエレメントの最大数	
65	4	X	MAXBYTES	配列を格納する DATATYPE=ARRAY フィールドまたは DATATYPE=STRUCT フィールドの最大バイト数	
69	4	X	RELSTART	フィールドの相対開始位置 (バイト単位)	
73	128	C	NAME	このフィールドの外部エイリアス名	
201	128	C	PARENT	このフィールドがネストされている別のフィールドの外部エイリアス名	
329	128	C	REDEFINE	このフィールドを再定義できる別のフィールドの外部エイリアス名。CFLD セグメント・タイプのこのインスタンスによって定義されたフィールド、および REDEFINE フィールドで指定された名前を持つフィールドは、COBOL アプリケーションの REDEFINES ステートメントで処理できます。	

表 *11. CFLD* セグメント・タイプのフォーマット *(*続き*)*:

オフセット (バイト)	長さ (バイト)	データ・タイプ	フィールド名	説明	固有キー・フィールド
457	128	C	DEPENDON	CFLD セグメント・タイプのこのインスタンスによって定義されたフィールドが従属するマッピング・セレクター・フィールド	
585	128	C	CASENAME	このフィールドが属するマッピング・ケースの名前	
713	128	C	STARTAFT	セグメント内で、このフィールドの直前にあるフィールドの外部エイリアス名。このフィールドにデータが入っている場合、START フィールドにデータは入っていません。	
841	64	C	FILLER03	予約済み	

関連資料:

- FIELD ステートメント (システム・ユーティリティー)
- DFSMAP ステートメント (システム・ユーティリティー)

CFLDRMK セグメント・タイプのフォーマット

IMS カタログの CFLDRMK セグメント・タイプは、特定のセグメント・タイプ・フォーマット・ケースに含まれているデータベース・フィールドについてのユーザーのコメントを含んでいます。

このセグメントは、コメントの対象である CFLD セグメント・インスタンスの直接の子です。

セグメント名
: CFLDRMK

親名 CFLD

順序フィールド
: CFLDCSEQ

セグメント長
: 264 バイト

表 *12. CFLDRMK* セグメント・マップ:

オフセット (バイト)	長さ (バイト)	データ・タイプ	フィールド名	説明	固有キー・フィールド
1	2	X	LEN	このセグメントの長さ	
3	2	X	CTL	制御フィールド	
5	2	X	CFLDCSEQ	シーケンス・フィールド、タイプ = U	X
5	2	X	SEQNUM	セグメント・コード	
7	2	C	FILLER	予約済み	

表 12. *CFLDRMK* セグメント・マップ (続き) :

オフセット (バイト)	長さ (バイト)	データ・タイプ	フィールド名	説明	固有キー・フィールド
9	256	C	REMARKS	親 CFLD セグメントに関するユーザーのコメント	

CMAR セグメント・タイプのフォーマット

IMS カタログの CMAR セグメント・タイプは、特定のケースのセグメント・タイプ・フォーマットにのみ適用される、IMS データベース内のフィールド・マーシャラー定義に関する情報を含んでいます。

それぞれの CASEFLD セグメントは、フィールドのデータ・マーシャル・プロパティーを含んでいる CMAR 子セグメントを持つことができます。

セグメント名
CMAR

親名 CFLD

順序フィールド
MARSHSEQ

セグメント長
704 バイト

表 13. *CMAR* セグメント・タイプのフォーマット :

オフセット (バイト)	長さ (バイト)	データ・タイプ	フィールド名	説明	固有キー・フィールド
1	2	X	LEN	このセグメントの長さ	
3	2	X	CTL	制御フィールド	
5	2	X	MARSHSEQ	シーケンス・フィールド、タイプ = U	X
5	2	X	SEQNUM	シーケンス番号	
9	8	C	OVERFLOW	予約済み	
17	1	C	SIGN	データ・タイプが DECIMAL (PACKEDDECIMAL または ZONEDDECIMAL のいずれかの内部タイプ・コンバーターを使用するデータ) の場合、このフィールドはデータが符号付き 10 進値であるかどうかを示します。	
18	6	C	FILLER01	予約済み	
24	25	C	ENCODING	親 FLD セグメントによって識別されるフィールドに入っているデータのエンコード・タイプ (コード・ページ) を識別します。	
49	50	C	PATTERN	親 FLD セグメントによって識別されるフィールドに入っているデータを、Java の日付オブジェクトに変換するためのパターン・マスクを識別します。	

表 *13. CMAR* セグメント・タイプのフォーマット *(*続き*)* :

オフセット (バイト)	長さ (バイト)	データ・タイプ	フィールド名	説明	固有キー・フィールド
99	30	C	ITYPCONV	親 FLD セグメントの内部タイプ・コンバーターを識別します。このフィールドにデータが入っている場合、UTYPCONV フィールドにはブランクが入っています。内部タイプ・コンバーターは、IMS データを特定のタイプの Java データ・オブジェクトに変換するために使用されます。	
129	256	C	UTYPCONV	親 FLD セグメントのユーザー・タイプ・コンバーターを識別します。このフィールドにデータが入っている場合、ITYPCONV フィールドにはブランクが入っています。	
385	256	C	URL	予約済み	
641	64	C	FILLER02	予約済み	

関連資料:

- DFSMAP ステートメント (システム・ユーティリティー)
- DFSCASE ステートメント (システム・ユーティリティー)

CMARRMK セグメント・タイプのフォーマット

IMS カタログの CMARRMK セグメント・タイプは、セグメント・タイプ・マッピングの特定のケースにおけるフィールド・マーシャラー定義に関するユーザー指定のコメントを含んでいます。

このセグメントは、コメントの対象である CMAR セグメント・インスタンスの直接の子です。

セグメント名
: CMARRMK

親名 CMAR

順序フィールド
: CMARCSEQ

セグメント長
: 264 バイト

表 *14. CMARRMK* セグメント・マップ :

オフセット (バイト)	長さ (バイト)	データ・タイプ	フィールド名	説明	固有キー・フィールド
1	2	X	LEN	このセグメントの長さ	
3	2	X	CTL	制御フィールド	
5	2	X	CMARCSEQ	シーケンス・フィールド、タイプ = U	X

表 *14. CMARRMK* セグメント・マップ *(*続き*)*:

オフセット (バイト)	長さ (バイト)	データ・タイプ	フィールド名	説明	固有キー・フィールド
5	2	X	SEQNUM	シーケンス番号	
7	2	C	FILLER	予約済み	
9	256	C	REMARKS	親 CMAR セグメントで定義されているマーシャル・プロパティーに関するユーザーのコメント	

CPROP セグメント・タイプのフォーマット

IMS カタログの CPROP セグメント・タイプは、IMS セグメント・タイプ・マッピングの特定のケースに関するユーザー定義のマーシャラー・プロパティーを含んでいます。

セグメント名
: CPROP

親名 CMAR

順序フィールド
: CPROSEQ

セグメント長
: 304 バイト

表 *15. CPROP* セグメント・タイプのフォーマット:

オフセット (バイト)	長さ (バイト)	データ・タイプ	フィールド名	説明	固有キー・フィールド
1	2	X	LEN	このセグメントの長さ	
3	2	X	CTL	制御フィールド	
5	2	X	CPROSEQ	シーケンス・フィールド、タイプ = U	X
5	2	X	SEQNUM	シーケンス番号	
9	128	C	NAME	ユーザー定義のマーシャル・プロパティーの名前	
137	128	C	VALUE	このマーシャル・プロパティーのユーザー定義値	
265	40	C	FILLER	予約済み	

関連資料:

- DFSCASE ステートメント (システム・ユーティリティー)
- DFSMAP ステートメント (システム・ユーティリティー)

DBD セグメント・タイプのフォーマット

IMS カタログの DBD セグメント・タイプは、IMS ユーザー・データベースに関するメタデータを含んでいます。

この情報は、システム定義のときに DBDGEN ユーティリティーにサブミットされたパラメーターから収集されます。

セグメント名
DBD

親名 HEADER

順序フィールド
DBDSEQ

セグメント長
552 バイト

表 *16. DBD* セグメント・タイプのフォーマット：

オフセット (バイト)	長さ (バイト)	データ・タイプ	フィールド名	説明	固有キー・フィールド
1	2	X	LEN	このセグメントの長さ	
3	2	X	CTL	制御フィールド	
5	2	X	SEQNUM	シーケンス番号	
9	17	X	DBDSEQ	シーケンス・フィールド、タイプ = U	X
9	4	X	CATVERS	この DBD セグメントおよびその従属セグメントによって記録されるデータベース定義のバージョン番号。	
13	13	C	TSVERS	このバージョンの ACB 生成タイム・スタンプ (フォーマットは *yyDDDHHmmssff*)	
26	1	C	FILLER	予約済み	
27	2	X	RLVL	ACB 生成ユーティリティーのリリース・レベル	
29	7	C	ACCESS	このデータベースの DL/I データベース・タイプ	
36	4	C	OSACC	このデータベースのアクセス方式	
40	6	C	PROT	副次索引データベースでは、このフィールドは索引ポインター・セグメントに整合性保護が使用されるかどうかを示します。 • このフィールドに PROT が入っている場合、索引ポインター・セグメントを削除する操作ではターゲット・セグメント・ポインターも削除されますが、ソース・セグメントは削除されません。 • このフィールドに NOPROT が入っている場合、アプリケーション・プログラムは、ポインター・セグメントの中の定数フィールド、検索フィールド、およびサブシーケンス制御フィールドを除くすべてのフィールドを置き換えることができます。	

表 *16. DBD* セグメント・タイプのフォーマット *(*続き*)*:

オフセット (バイト)	長さ (バイト)	データ・タイプ	フィールド名	説明	固有キー・フィールド
46	7	C	DOSCOMP	このデータベースが DLI/DOS 索引であることと、このデータベースのセグメントの接頭部に DLI/DOS セグメント・コードが含まれていることを示します。IMS はセグメント処理の間、コードを保存し、セグメントが挿入されるときは新しいコードを提供します。	
53	8	C	PSNAME	このデータベース用の HALDB 区画選択出口ルーチンの名前。	
61	8	C	RMNAME	HDAM データベースか PHDAM データベース、または高速機能の高速処理データベース (DEDB) 用のランダム化出口ルーチンのモジュール名。	
69	4	X	RMRBN	ランダム化出口ルーチンが、この HDAM データベースまたは PHDAM データベース用に生成する最大相対ブロック番号。この値は、データベースのルート・アドレス可能域における制御ブロックまたは制御インターバルの数でもあります。	
73	4	X	RMBYTES	中断のない挿入操作シーケンスによって、このデータベースのルート・アドレス可能域に保管できるユーザー・データの最大バイト数。このサイズを超えるデータベース・レコードは、オーバーフロー域に部分的に保管されます。	
77	2	X	RMANCH	このフィールドの意味は、高速機能 DEDB データベースの場合と全機能 HDAM または PHDAM データベースの場合では異なります。 DEDB データベースの場合は、ランダマイザーのタイプを示します。値 1 は 1 ステージ・ランダマイザーを示します。値 2 は 2 ステージ・ランダマイザーを示します。 HDAM データベースと PHDAM データベースの場合、この値は、HDAM データベースか PHDAM データベースのルート・アドレス可能域における各制御ブロックまたは各制御インターバルのルート・アンカー・ポイントの数を示します。	
79	1	C	RMXCI	この DEDB が、ランダム化出口ルーチンを呼び出すときに拡張呼び出しインターフェース (XCI) を使用するかどうかを示します。	
80	3	C	FILLER01	予約済み	
83	1	C	PASSWD	このデータベースが、IMS 以外のプログラムによる偶発的なデータベース操作を防止するために、デフォルトの VSAM パスワード (DBD 名) を使用するかどうかを示します。	

表 16. DBD セグメント・タイプのフォーマット (続き):

オフセット (バイト)	長さ (バイト)	データ・タイプ	フィールド名	説明	固有キー・フィールド
84	1	C	DATXEXIT	このデータベースが、データ変換ユーザー出口ルーチン DFSDBUX1 を使用するかどうかを示します。	
85	1	C	FPI	このデータベースが、高速機能データベースの副次索引であるかどうかを示します。	
86	255	C	VERSION	このデータベースのユーザー提供バージョン情報	
341	2	C	FILLER02	予約済み	
343	2	X	IDXCNT	副次索引の共用の数	
345	8	C	IDXNM01	共用副次索引名	
353	8	C	IDXNM02	共用副次索引名	
361	8	C	IDXNM03	共用副次索引名	
369	8	C	IDXNM04	共用副次索引名	
377	8	C	IDXNM05	共用副次索引名	
385	8	C	IDXNM06	共用副次索引名	
393	8	C	IDXNM07	共用副次索引名	
401	8	C	IDXNM08	共用副次索引名	
409	8	C	IDXNM09	共用副次索引名	
417	8	C	IDXNM10	共用副次索引名	
425	8	C	IDXNM11	共用副次索引名	
433	8	C	IDXNM12	共用副次索引名	
441	8	C	IDXNM13	共用副次索引名	
449	8	C	IDXNM14	共用副次索引名	
457	8	C	IDXNM15	共用副次索引名	
465	8	C	IDXNM16	共用副次索引名	
473	8	C	CREATEBY	予約済み	
481	25	C	ENCODING	このデータベース内のすべての文字データをエンコードするために使用されるコード・ページ。この値は、個別のセグメント定義およびフィールド定義によってオーバーライドできます。	
506	47	C	FILLER03	予約済み	

DBDRMK セグメント・タイプのフォーマット

IMS カタログの DBDRMK セグメント・タイプは、データベース定義に関するユーザー指定のコメントを含んでいます。

このセグメントは、コメントの対象である DBD セグメント・インスタンスの直接の子です。

セグメント名

DBDRMK

親名　DBD

順序フィールド
DBDCMSEQ

セグメント長
264 バイト

表 *17. DBDRMK* セグメント・マップ :

オフセット (バイト)	長さ (バイト)	データ・タイプ	フィールド名	説明	固有キー・フィールド
1	2	X	LEN	このセグメントの長さ	
3	2	X	CTL	制御フィールド	
5	2	X	DBDCMSEQ	シーケンス・フィールド、タイプ = U	X
5	2	X	SEQNUM	シーケンス番号	
7	2	C	FILLER	予約済み	
9	256	C	REMARKS	この DBD に関するユーザーのコメント	

DBDVEND セグメント・タイプのフォーマット

IMS カタログの DBDVEND セグメント・タイプは、短縮ヘッダーと、それに続く大きなブロックの未フォーマット・スペースを含んでいます。

このセグメント・タイプは、ベンダー提供ツールが使用するために予約されています。

セグメント名
DBDVEND

親名　DBD

順序フィールド
DVNDSEQ

セグメント長
4000 バイト

表 *18. DBDVEND* セグメント・マップ :

オフセット (バイト)	長さ (バイト)	データ・タイプ	フィールド名	説明	固有キー・フィールド
1	2	X	LEN	このセグメントの長さ	
3	2	X	CTL	制御フィールド	
5	2	X	SEQNUM	セグメント・シーケンス番号	
7	2	C	FILLER	予約済み	
9	3992	X	DATA	ベンダー製品の DBD データ	
5	2	X	DVNDSEQ	シーケンス・フィールド、タイプ = U	X

DSET セグメント・タイプのフォーマット

IMS カタログの DSET セグメント・タイプは、IMS データベースのデータ・セット仕様に関するメタデータを含んでいます。

このセグメント内の情報は、DBDGEN ユーティリティーの DATASET ステートメントのパラメーターに基づいて生成されます。すべての DBD カタログ・レコードには、少なくとも 1 つの子 DSET セグメント・インスタンスが存在します。

セグメント名
: DSET

親名 DBD

順序フィールド
: DSETSEQ

セグメント長
: 96 バイト

表 *19. DSET* セグメント・タイプのフォーマット：

オフセット (バイト)	長さ (バイト)	データ・タイプ	フィールド名	説明	固有キー・フィールド
1	2	X	LEN	このセグメントの長さ	
3	2	X	CTL	制御フィールド	
5	2	X	DSETSEQ	シーケンス・フィールド、タイプ = U	X
5	2	X	SEQNUM	シーケンス番号	
9	8	C	DD1	このデータ・セット・グループの最初のデータ・セットの名前。 • HSAM、SHSAM、および GSAM データベースの場合、このフィールドは入力データ・セットの名前です。 • HISAM、SHISAM、および INDEX データベースの場合、このフィールドは 1 次データ・セットの名前です。 • 高速機能データベースの場合、このフィールドは定義されたデータ域の名前です。 MSDB および論理データベースは、このフィールドを使用しません。	
17	8	C	DD2	HSAM、SHSAM、および GSAM データベースの出力データ・セットの名前。GSAM データベースの名前が指定されなかった場合、DD1 が出力データ・セットとして使用されます。	
25	8	C	OVERFLOW	このグループのオーバーフロー・データ・セットの名前。	

表 19. DSET セグメント・タイプのフォーマット (続き):

オフセット (バイト)	長さ (バイト)	データ・タイプ	フィールド名	説明	固有キー・フィールド
33	2	X	BLOCK1	データ・セット・グループのブロック化因数 1。使用法については、DBDGEN ユーティリティーの DATASET ステートメントを参照してください。	
35	2	X	BLOCK2	データ・セット・グループのブロック化因数 2。使用法については、DBDGEN ユーティリティーの DATASET ステートメントを参照してください。	
37	2	X	SIZE1	データ・セット・グループのブロック・サイズ 1。使用法については、DBDGEN ユーティリティーの DATASET ステートメントを参照してください。	
39	2	X	SIZE2	データ・セット・グループのブロック・サイズ 2。使用法については、DBDGEN ユーティリティーの DATASET ステートメントを参照してください。	
41	2	X	RECORD1	データ・セット・グループのレコード・サイズ 1。使用法については、DBDGEN ユーティリティーの DATASET ステートメントを参照してください。	
43	2	X	RECORD2	データ・セット・グループのレコード・サイズ 2。使用法については、DBDGEN ユーティリティーの DATASET ステートメントを参照してください。	
45	2	X	SCAN	セグメント挿入操作のとき、フリー・ストレージがないかどうかスキャンする DASD シリンダーの数。HIDAM データベースおよび HDAM データベースにのみ使用されます。	
47	2	X	SEARCHA	セグメント挿入操作のとき、フリー・ストレージがないかどうか検索するために使用されるアルゴリズムのタイプ。HIDAM データベースおよび HDAM データベースにのみ使用されます。以下のように、さまざまなタイプ・コードと意味があります。 **0** 使用するアルゴリズムは IMS によって選択されます。 **1** IMS は、2 番目に望ましいブロックまたはコントロール・インターバルではスペースを検索しません。 **2** IMS は、2 番目に望ましいブロックまたはコントロール・インターバルでもスペースの検索を実行します。	

表 19. *DSET* セグメント・タイプのフォーマット (続き) :

オフセット (バイト)	長さ (バイト)	データ・タイプ	フィールド名	説明	固有キー・フィールド
49	2	C	RECFM	GSAM データベースの場合、このデータ・セット・グループのレコード・フォーマット。 **F** 固定長 **FB** 固定長およびブロック化 **V** 可変長 **VB** 可変長およびブロック化 **U** 未定義の長さ	
51	2	X	FRSPFBFF	HDAM データベースまたは HIDAM データベースで、1 ブロックのフリー・スペースに割り振られるデータ・ブロック数。	
53	2	X	FRSPFSPF	このデータ・セット・グループにおける各制御ブロックまたはインターバルのフリー・スペースの最小パーセンテージ。HDAM データベースおよび HIDAM データベースにのみ使用されます。	
55	8	C	REL1	MSDB での端末リレーションシップ・タイプおよびセグメント所有権タイプ: **NO** 端末キーなし端末非関連 **TERM** LTERM 名をキーとする端末非関連 **FIXED** LTERM 名をキーとし、セグメントの挿入と削除が使用不可の端末関連 **DYNAMIC** LTERM 名をキーとし、セグメントの挿入と削除が使用可能の端末関連	
63	8	C	REL2	キー付き MSDB の疑似シーケンス・フィールドの名前。セグメント検索引数では、この疑似フィールドの名前と LTERM 名をキー値として使用できます。	
71	26	C	FILLER	予約済み	

DSETRMK セグメント・タイプのフォーマット

IMS カタログの DSETRMK セグメント・タイプは、データ・セット・グループ定義に関するユーザー指定のコメントを含んでいます。

このセグメントは、コメントの対象である DSET セグメント・インスタンスの直接の子です。

セグメント名
DSETRMK

親名 DSET

順序フィールド
DSCMSEQ

セグメント長
264 バイト

表 *20. DSETRMK* セグメント・マップ：

オフセット (バイト)	長さ (バイト)	データ・タイプ	フィールド名	説明	固有キー・フィールド
1	2	X	LEN	このセグメントの長さ	
3	2	X	CTL	制御フィールド	
5	2	X	DSCMSEQ	シーケンス・フィールド、タイプ = U	X
5	2	X	SEQNUM	シーケンス番号	
7	2	C	FILLER	予約済み	
9	256	C	REMARKS	この DSET に関するユーザーのコメント	

FLD セグメント・タイプのフォーマット

IMS カタログの FLD セグメント・タイプは、IMS データベース内のフィールドに関するメタデータを含んでいます。

FLD セグメントの各インスタンスは、あるデータベース内の 1 つのセグメントの 1 つのフィールドを記述しています。この情報は、DBDGEN ユーティリティーの FIELD ステートメントからのシステム生成時に収集されます。

セグメント名
FLD

親名 SEGM

順序フィールド
FLDSEQ

セグメント長
904 バイト

表 *21. FLD* セグメント・タイプのフォーマット：

オフセット (バイト)	長さ (バイト)	データ・タイプ	フィールド名	説明	固有キー・フィールド
1	2	X	LEN	このセグメントの長さ	
3	2	X	CTL	制御フィールド	
5	2	X	FLDSEQ	シーケンス・フィールド、タイプ = U	X
5	2	X	SEQNUM	シーケンス番号	
9	8	C	IMSNAME	このフィールドの 8 文字の IMS 名	
17	3	C	NAMESEQ	このフィールドがシーケンス・フィールドであるかどうかを示します。	

表 *21. FLD* セグメント・タイプのフォーマット *(*続き*)*:

オフセット (バイト)	長さ (バイト)	データ・タイプ	フィールド名	説明	固有キー・フィールド
20	1	C	SEQUM	このフィールドが固有シーケンス・フィールド (U) であるか、非固有シーケンス・フィールド (M) であるかを示します。	
21	2	X	BYTES	このフィールドの長さ (バイト単位)	
23	2	X	START	セグメントの先頭から数えた、このフィールドの開始オフセット (バイト単位)。このフィールドにデータが入っている場合、STARTAFT フィールドは使用されません。	
25	1	C	TYPE	このフィールドの空きスペースに埋め込むために IMS が使用するバイナリー・データのタイプを示します。 **X** 左に X'00' を埋め込み。 **P** 左に X'00' を埋め込み。 **C** 右に X'40' を埋め込み **F** 2 進フルワード・データ。MSDB 専用。 **H** 2 進ハーフワード・データ。MSDB 専用。	
26	15	C	FILLER01	予約済み	
41	9	C	DATATYPE	フィールドの外部データ・タイプ	
50	3	C	FILLER02	予約済み	
53	2	X	PRECISN	10 進データ・タイプのフィールドの精度	
55	2	X	SCALE	10 進データ・タイプのフィールドの位取り	
57	4	X	MINOCCUR	DATATYPE=ARRAY フィールド内のエレメントの最小数	
61	4	X	MAXOCCUR	DATATYPE=ARRAY フィールド内のエレメントの最大数	
65	4	X	MAXBYTES	配列を格納する DATATYPE=ARRAY フィールドまたは DATATYPE=STRUCT フィールドの最大バイト数	
69	4	X	RELSTART	配列または struct が親セグメント内でこのフィールドより前に発生する場合の、動的配列または動的 struct の終わりから数えた相対的な開始オフセット (バイト単位)	
73	128	C	NAME	このフィールドの外部エイリアス名	
201	128	C	PARENT	このフィールドがネストされている別のフィールドの外部エイリアス名	

表 21. FLD セグメント・タイプのフォーマット (続き) :

オフセット (バイト)	長さ (バイト)	データ・タイプ	フィールド名	説明	固有キー・フィールド
329	128	C	REDEFINE	このフィールドを再定義できる別のフィールドの外部エイリアス名。このフィールドおよび REDEFINE フィールドで示されているフィールドは、COBOL アプリケーションの REDEFINES ステートメントで処理できます。	
457	128	C	DEPENDON	このフィールドにはブランクが入ります。	
585	128	C	CASENAME	このフィールドにはブランクが入ります。	
713	128	C	STARTAFT	セグメント内で、このフィールドの直前にあるフィールドの外部エイリアス名。このフィールドにデータが入っている場合、START フィールドにデータは入っていません。	
841	64	C	FILLER03	予約済み	

FLDRMK セグメント・タイプのフォーマット

IMS カタログの FLDRMK セグメント・タイプは、データベース・フィールドに関するユーザーのコメントを含んでいます。

このセグメントは、コメントの対象である FLD セグメント・インスタンスの直接の子です。

セグメント名
: FLDRMK

親名 FLD

順序フィールド
: FLDCMSEQ

セグメント長
: 264 バイト

表 22. FLDRMK セグメント・マップ :

オフセット (バイト)	長さ (バイト)	データ・タイプ	フィールド名	説明	固有キー・フィールド
1	2	X	LEN	このセグメントの長さ	
3	2	X	CTL	制御フィールド	
5	2	X	FLDCMSEQ	シーケンス・フィールド、タイプ = U	X
5	2	X	SEQNUM	シーケンス番号	
7	2	C	FILLER	予約済み	
9	256	C	REMARKS	親 FLD に関するユーザーのコメント	

LCH2IDX セグメント・タイプのフォーマット

IMS カタログの LCH2IDX セグメント・タイプには、親 LCHILD セグメントで定義された LCHILD ステートメントによって指定されている高速機能副次索引に関する情報を含んでいます。

セグメント名
: LCH2IDX

親名　LCHILD

順序フィールド
: LCH2ISEQ

セグメント長
: 24 バイト

表 *23. LCH2IDX* セグメント・マップ :

オフセット (バイト)	長さ (バイト)	データ・タイプ	フィールド名	説明	固有キー・フィールド
1	2	X	LEN	このセグメントの長さ	
3	2	X	CTL	制御フィールド	
5	2	X	LCH2ISEQ	シーケンス・フィールド、タイプ = U	X
5	2	X	SEQNUM	シーケンス番号	
7	2	C	FILLER01	予約済み	
9	8	C	IMSNAME	IMS 名	
17	8	C	DBDNAME	ターゲットの副次索引の名前	

LCHILD セグメント・タイプのフォーマット

IMS カタログの LCHILD セグメント・タイプは、セグメント・タイプ間の関係に関する情報を含んでいます。

セグメント名
: LCHILD

親名　SEGM

順序フィールド
: LCHLDSEQ

セグメント長
: 72 バイト

表 *24. LCHILD* セグメント・マップ :

オフセット (バイト)	長さ (バイト)	データ・タイプ	フィールド名	説明	固有キー・フィールド
1	2	X	LEN	このセグメントの長さ	
3	2	X	CTL	制御フィールド	
5	2	X	LCHILDSEQ	シーケンス・フィールド、タイプ = U	X
5	2	X	SEQNUM	シーケンス番号	
9	8	C	IMSNAME	この論理関係の親セグメント・タイプに関連付けられているセグメント・タイプの名前。	

表 *24. LCHILD* セグメント・マップ *(*続き*)*:

オフセット (バイト)	長さ (バイト)	データ・タイプ	フィールド名	説明	固有キー・フィールド
17	8	C	DBNAME	IMSNAME フィールドによって識別されたセグメントを含んでいるデータベースの名前。	
25	4	C	PTR	この論理関係で使用されるポインターのタイプを示します。 **SNGL** 論理子の最初のポインター・フィールドが親セグメント・タイプの中で予約済みであることを示します。このフィールドは、以下の 3 つの方法のいずれかで使用されます。 • 論理親関係の場合、このポインター・フィールドには論理子セグメント・タイプの最初のオカレンスを指す直接アドレス・ポインターが入ります。 • HIDAM 1 次索引データベース関係の場合、このポインター・フィールドには HIDAM ルート・データベース・セグメントを指す直接アドレス・ポインターが入ります。 • 副次索引関係の場合、このポインター・フィールドには索引ターゲット・セグメントを指す直接アドレス・ポインターが入ります。 **DBLE** 論理親セグメント・タイプの中で 2 つの 4 バイト・ポインター・フィールドが予約済みであることを示します。最初のポインター・フィールドには論理子セグメント・タイプの最初のオカレンスが入り、2 番目のポインター・フィールドには論理子セグメント・タイプの最後のオカレンスのアドレスが入ります。 **NONE** 論理親セグメント・タイプの中で予約済みのポインター・フィールドはありません。論理親と論理子の間の関係は、実装されていないか、物理的に対になっているセグメントを使用して保守されています。 **INDX** HIDAM データベース内の最初の論理子関係の場合、この値は、親セグメントが HIDAM データベース内のルート・セグメント・タイプであり、ターゲット・セグメントがそのデータベースの 1 次索引のルート・セグメントであることを示します。HIDAM データベースのそれ以後の論理子関係の場合、または他のデータベースの場合、この値は、ターゲット・セグメント・タイプがそのデータベースの副次索引ターゲットであることを示します。 **SYMB** この値は、1 次データベース内のポインター・フィールドに副次索引データベース内のターゲット・セグメントを指す直接ターゲット・アドレスが入っていないことを示します。代わりに、ポインター・フィールドにはターゲット・セグメントの連結キーが入っています。副次索引データベースでは、この値は索引ポインター・セグメント内にターゲット・セグメントのアドレス用のスペースが予約済みでないことを示します。	
29	8	C	PAIR	双方向論理関係において、IMSNAME フィールドで示されたセグメントと対になっているセグメントの名前。	
37	8	C	INDEX	このデータベースが 1 次索引である HIDAM データベースのルート・セグメント・タイプのシーケンス・フィールドの名前。このフィールドにデータが入るのは、このデータベースが HIDAM データベースの 1 次索引である場合だけです。	

表 *24. LCHILD* セグメント・マップ *(*続き*)*:

オフセット (バイト)	長さ (バイト)	データ・タイプ	フィールド名	説明	固有キー・フィールド
45	5	C	RULES	仮想論理子セグメントがシーケンス・フィールドを含んでいないか非固有シーケンス・フィールドを使用している場合に、DL/I 挿入操作のときにそれらのセグメントを順序付ける方法を示します。 **FIRST** シーケンス・フィールドが存在しない場合、論理子の最初の既存インスタンスの前に新しいセグメント・インスタンスが挿入されます。非固有シーケンス・フィールドが存在する場合は、新しいインスタンスと同じキー値を持つすべての既存インスタンスの前に、新しいセグメント・インスタンスが挿入されます。 **LAST** シーケンス・フィールドが存在しない場合、論理子の最後の既存インスタンスの後ろに新しいセグメント・インスタンスが挿入されます。非固有シーケンス・フィールドが存在する場合は、新しいインスタンスと同じキー値を持つすべての既存インスタンスの後ろに、新しいセグメント・インスタンスが挿入されます。 **HERE** 新しいインスタンスは、前回の DL/I 呼び出し後のカーソル位置に挿入されます。現在位置がない場合は、FIRST が代わりに使用されます。	
50	1	C	MULTI	LCHILD ステートメントが複数副次索引セグメント・グループのメンバーであるかどうかを示します。	
51	2	C	FILLER01	予約済み	
53	4	X	RKSIZE	ターゲット・データベースのルート・キー・サイズ。このフィールドは、区分副次索引データベースにのみ使用されます。	
57	16	C	FILLER02	予約済み	

関連概念:

268 ページの『論理関係の種類』

285 ページの『論理子セグメント』

関連資料:

LCHILD ステートメント (システム・ユーティリティー)

LCHRMK セグメント・タイプのフォーマット

IMS カタログの LCHRMK セグメント・タイプは、IMS データベース内の論理子関係に関するユーザー指定のコメントを含んでいます。

このセグメントは、コメントの対象である LCHILD セグメントの直接の子です。

セグメント名
: LCHRMK

親名 LCHILD

順序フィールド
: LCHCMSEQ

セグメント長
: 264 バイト

表 25. *LCHRMK* セグメント・マップ:

オフセット (バイト)	長さ (バイト)	データ・タイプ	フィールド名	説明	固有キー・フィールド
1	2	X	LEN	このセグメントの長さ	
3	2	X	CTL	制御フィールド	
5	2	X	LCHCMSEQ	シーケンス・フィールド、タイプ = U	X
5	2	X	SEQNUM	シーケンス番号	
7	2	C	FILLER	予約済み	
9	256	C	REMARKS	この論理関係に関するユーザーのコメント	

MAP セグメント・タイプのフォーマット

IMS カタログの MAP セグメント・タイプは、IMS データベース・セグメント内のセグメント・タイプ・マッピングに関する情報を含んでいます。

セグメント名
MAP

親名 SEGM

順序フィールド
MAPSEQ

セグメント長
520 バイト

表 26. *MAP* セグメント・タイプのフォーマット:

オフセット (バイト)	長さ (バイト)	データ・タイプ	フィールド名	説明	固有キー・フィールド
1	2	X	LEN	このセグメントの長さ	
3	2	X	CTL	制御フィールド	
5	2	X	MAPSEQ	シーケンス・フィールド、タイプ = U	X
5	2	X	SEQNUM	シーケンス番号	
7	2	C	FILLER	予約済み	
9	128	C	NAME	このマップの名前	
137	128	C	DEPENDON	このセグメント内の制御フィールドの外部名 (FLD カタログ・レコードの NAME フィールドにあります)。これは、ユーザー・データベース内のマップされた各セグメント・インスタンスに使用されるマップ・ケースを決定します。	
265	256	C	FILLER	予約済み	

関連資料:

DFSMAP ステートメント (システム・ユーティリティー)

MAPRMK セグメント・タイプのフォーマット

IMS カタログの MAPRMK セグメント・タイプは、セグメント・タイプ・マッピング定義に関するユーザーのコメントを含んでいます。

セグメント名
: MAPRMK

親名　MAP

順序フィールド
: MAPCMSEQ

セグメント長
: 264 バイト

表 *27. MAPRMK* セグメント・マップ :

オフセット (バイト)	長さ (バイト)	データ・タイプ	フィールド名	説明	固有キー・フィールド
1	2	X	LEN	このセグメントの長さ	
3	2	X	CTL	制御フィールド	
5	2	X	MAPCMSEQ	シーケンス・フィールド、タイプ = U	X
5	2	X	SEQNUM	シーケンス番号	
7	2	C	FILLER	予約済み	
9	256	C	REMARKS	親 MAP セグメントに関するユーザーのコメント	

MAR セグメント・タイプのフォーマット

IMS カタログの MAR セグメント・タイプは、IMS データベース内のフィールド・マーシャラー定義に関する情報を含んでいます。

それぞれの FLD セグメントは、フィールドのデータ・マーシャル・プロパティーを含んでいる MAR 子セグメントを持つことができます。このセグメント・タイプ内の情報は、DBDGEN ユーティリティーの DFSMARSH ステートメントの入力パラメーターから生成されます。

セグメント名
: MAR

親名　FLD

順序フィールド
: MARSEQ

セグメント長
: 704 バイト

表 28. MAR セグメント・タイプのフォーマット：

オフセット (バイト)	長さ (バイト)	データ・タイプ	フィールド名	説明	固有キー・フィールド
1	2	X	LEN	このセグメントの長さ	
3	2	X	CTL	制御フィールド	
5	2	X	MARSEQ	シーケンス・フィールド、タイプ = U	X
5	2	X	SEQNUM	シーケンス番号	
7	2	C	FILLER	予約済み	
9	8	C	OVERFLOW	予約済み	
17	1	C	SIGN	データ・タイプが DECIMAL (PACKEDDECIMAL または ZONEDDECIMAL のいずれかの内部タイプ・コンバーターを使用するデータ) の場合、このフィールドはデータが符号付き 10 進値であるかどうかを示します。	
18	6	C	FILLER01	予約済み	
24	25	C	ENCODING	親 FLD セグメントによって識別されるフィールドに入っているデータのエンコード・タイプ (コード・ページ) を識別します。	
49	50	C	PATTERN	親 FLD セグメントによって識別されるフィールドに入っているデータを、Java の日付オブジェクトに変換するためのパターン・マスクを識別します。	
99	30	C	ITYPCONV	親 FLD セグメントの内部タイプ・コンバーターを識別します。このフィールドにデータが入っている場合、UTYPCONV フィールドにはブランクが入っています。内部タイプ・コンバーターは、IMS データを特定のタイプの Java データ・オブジェクトに変換するために使用されます。	
129	256	C	UTYPCONV	親 FLD セグメントのユーザー・タイプ・コンバーターを識別します。このフィールドにデータが入っている場合、ITYPCONV フィールドにはブランクが入っています。	
385	256	C	URL	予約済み	
641	64	C	FILLER02	予約済み	

MARRMK セグメント・タイプのフォーマット

IMS カタログの MARRMK セグメント・タイプは、フィールド・マーシャラー定義に関するユーザーのコメントを含んでいます。

セグメント名
MARRMK

親名 MAR

順序フィールド
MARCMSEQ

セグメント長
264 バイト

表 *29. MARRMK* セグメント・マップ:

オフセット (バイト)	長さ (バイト)	データ・タイプ	フィールド名	説明	固有キー・フィールド
1	2	X	LEN	このセグメントの長さ	
3	2	X	CTL	制御フィールド	
5	2	X	MARCMSEQ	シーケンス・フィールド、タイプ = U	X
5	2	X	SEQNUM	シーケンス番号	
7	2	C	FILLER	予約済み	
9	256	C	REMARKS	親 MAR セグメントに関するユーザーのコメント	

PROP セグメント・タイプのフォーマット

IMS カタログの PROP セグメント・タイプは、ユーザー定義マーシャラー・プロパティー定義を含んでいます。

セグメント名
PROP

親名 MAR

順序フィールド
PROPSEQ

セグメント長
304 バイト

表 *30. PROP* セグメント・タイプのフォーマット:

オフセット (バイト)	長さ (バイト)	データ・タイプ	フィールド名	説明	固有キー・フィールド
1	2	X	LEN	このセグメントの長さ	
3	2	X	CTL	制御フィールド	
5	2	X	PROPSEQ	シーケンス・フィールド、タイプ = U	X
5	2	X	SEQNUM	シーケンス番号	
9	128	C	NAME	このユーザー定義マーシャラー・プロパティーの名前	
137	128	C	VALUE	ユーザー定義マーシャラー・プロパティー情報	
265	40	C	FILLER	予約済み	

関連資料:

DFSMAP ステートメント (システム・ユーティリティー)

SEGM セグメント・タイプのフォーマット

IMS カタログの SEGM セグメント・タイプは、IMS データベース・セグメントに関するメタデータを含んでいます。

この情報は、DBDGEN ユーティリティーの SEGM ステートメントのパラメーターに基づいて生成されます。

セグメント名
SEGM

親名 DBD

順序フィールド
SEGMSEQ

セグメント長
376 バイト

表 *31. SEGM* セグメント・タイプのフォーマット：

オフセット (バイト)	長さ (バイト)	データ・タイプ	フィールド名	説明	固有キー・フィールド
1	2	X	LEN	このセグメントの長さ。	
3	2	X	CTL	制御フィールド。	
5	2	X	SEGMSEQ	シーケンス・フィールド、タイプ = U。	X
5	2	X	SEQNUM	シーケンス番号。	
9	8	C	IMSNAME	このセグメントの名前。	
17	8	C	PARPHY	このセグメントの論理親の名前。	
25	4	C	PARTYPE	このセグメントの物理親のすべてのオカレンスに含まれる物理子ポインター (SNGL または DBLE) のタイプ。	
29	8	C	PARLOG	このセグメントの論理親の名前。	
37	8	C	PARCHK	論理親の連結キーが仮想か物理かを示します。	
45	8	C	DBNAME	このセグメントの論理親が定義されているデータベースの名前。	
53	4	X	BYTE1	可変長セグメントの最大長 (バイト単位)、または固定長セグメントのデータ域のバイト数。	
57	4	X	BYTE2	可変長セグメントの最小長 (バイト単位)。このフィールドにデータが入っている場合、このセグメント・タイプは可変長です。	
61	4	X	FREQ	このセグメントが物理親の各インスタンスについて発生する回数の見積もり。この値は、このセグメント・タイプを含んでいるデータベース内のデータ・セット・グループの論理レコード長と物理ストレージ・ブロック・サイズを決定するために IMS によって使用されます。	

表 *31. SEGM* セグメント・タイプのフォーマット *(*続き*)*:

オフセット (バイト)	長さ (バイト)	データ・タイプ	フィールド名	説明	固有キー・フィールド
65	3	C	RULE1	このセグメント・タイプのインスタンスを挿入、削除、または置換するために使用する必要があるパス・タイプ。このフィールドには 3 文字が入ります。最初の文字は挿入操作用のパス・タイプで、2 文字目は削除操作用のパス・タイプ、3 文字目は置換操作用のパス・タイプです。	
68	5	C	RULE2	固有のシーケンス・フィールドがないセグメント・タイプの新しいインスタンスを追加するときに、IMS が使用する規則。	
73	8	C	SRCSEG1	仮想論理子セグメント・タイプのカタログ・レコード内では、これは、この仮想論理子に対応する実論理子の名前です。 論理データベース内のセグメント・タイプのカタログ・レコード内では、これは論理セグメント・タイプとして定義されようとしている物理データベース内のソース・セグメントの名前です。	
81	4	C	SRCFBK1	仮想論理子セグメント・タイプのカタログ・レコード内で、このフィールドは、このセグメント・タイプをユーザー入出力域に構成するために実論理子のキーだけが使用されるのか、それともキーとデータの両方の部分が使用されるのかを示します。 このフィールドは、論理データベース内のセグメントにのみ使用されます。	
85	8	C	SRCDBN1	SRCSEG1 フィールドで識別されたセグメント・タイプを含んでいるデータベースの名前。	
93	8	C	SRCSEG2	連結された仮想論理子セグメント・タイプのカタログ・レコード内では、これは実論理子セグメントの物理親の名前です。 論理データベース内のセグメントのカタログ・レコード内では、これは、この論理連結セグメントの宛先親セクションを構成するために使用される、物理データベース内の論理または物理親セグメントの名前です。 このフィールドにデータが入っている場合、このセグメントは論理連結セグメントです。	

表 *31. SEGM* セグメント・タイプのフォーマット (続き) :

オフセット (バイト)	長さ (バイト)	データ・タイプ	フィールド名	説明	固有キー・フィールド
101	4	C	SRCFBK2	連結仮想論理子セグメント・タイプのカタログ・レコード内で、このフィールドは、このセグメント・タイプをユーザー入出力域に構成するために実論理子のキーだけが使用されるのか、それともキーとデータの両方の部分が使用されるのかを示します。 連結セグメントのキー値は、論理子がどのパスからアクセスされるかによって、物理兄弟シーケンス・フィールドまたは論理兄弟シーケンス・フィールドのいずれかの値になります。 このフィールドは、論理データベース内の連結セグメントにのみ使用されます。	
105	8	C	SRCDBN2	SRCSEG2 フィールドで識別されたセグメント・タイプを含んでいるデータベースの名前。	
113	8	C	COMPRTN	このセグメントに使用されるセグメント編集/圧縮出口ルーチンの名前。	
121	4	C	COMPDATA	このセグメント用のセグメント編集/圧縮出口ルーチンが、データ・フィールドだけを対象として圧縮または変更を行い、シーケンス・フィールドは対象としないことを示します。	
125	4	C	COMPINIT	COMPRTN フィールドで識別されたセグメント編集/圧縮出口ルーチンに初期設定および終了処理の制御が必要であるかどうかを示します。	
129	4	X	COMPMAX	このセグメントが COMPRTN フィールドで識別されたセグメント編集/圧縮出口ルーチンで変更される場合、このセグメントの最大解凍サイズ (バイト単位) を示します。	
133	3	C	COMPPAD	セグメント・インスタンスがセグメント編集/圧縮出口ルーチンによって圧縮され、COMPMAX フィールドで指定されたサイズより小さくなった場合に、そのサイズまで埋め込みを行うかどうかを示します。	
136	1	C	FILLER01	予約済み。	

表 *31. SEGM* セグメント・タイプのフォーマット *(*続き*)*:

オフセット (バイト)	長さ (バイト)	データ・タイプ	フィールド名	説明	固有キー・フィールド
137	7	C	PTR1	ポインター・フィールドが、セグメント接頭部で HIER (階層順方向ポインター)、HIERBWD (階層順方向および逆方向ポインター)、TWIN (兄弟順方向ポインター)、TWINBWD (兄弟順方向および逆方向ポインター)、NOTWIN (物理兄弟ポインターの予約フィールドなし) のいずれかの関係用に予約されているかどうかを示します。これらの値について詳しくは、DBDGEN ユーティリティーの SEGM ステートメントの POINTER= パラメーターを参照してください。	
144	8	C	PTR2	ポインター・フィールドが、セグメント接頭部で LTWIN (論理兄弟順方向ポインター) 関係または LTWINBWD (論理兄弟順方向および逆方向ポインター) 関係用に予約されているかどうかを示します。	
152	6	C	PTR3	ポインター・フィールドがセグメント接頭部で LPARNT (論理親セグメントを指すポインター) 関係用に予約されているかどうかを示します。	
158	3	C	PTR4	4 バイト・フィールドがセグメント接頭部で論理関係カウンター用に予約されているかどうかを示します。	
161	6	C	PTR5	このセグメント・タイプが双方向論理関係に含まれているかどうかを示します。	
167	2	X	SSPTR	サブセット・ポインターの数。このフィールドの値が 0 の場合、このセグメント・タイプではサブセット・ポインターは使用されません。	
169	3	C	TYPE	このセグメント・タイプの DEDB 従属セグメントのタイプ (順次または従属) を示します。	
172	1	C	DSGRP	このセグメントのデータ・セット・グループ ID。このフィールドは、HALDB 内のセグメントにのみ使用されます。	
173	2	X	DSGHAL	内部使用のために予約済みです。	
175	12	C	FILLER02	予約済み。	
187	128	C	NAME	このセグメントの外部エイリアス名。	
315	25	C	ENCODING	このセグメントのすべての文字データをエンコードするために使用されるコード・ページ。	
340	37	C	FILLER03	予約済み。	

関連概念:

560 ページの『SEGM ステートメントの概説』

関連資料:

SEGM ステートメント (システム・ユーティリティー)

SEGMRMK セグメント・タイプのフォーマット

IMS カタログの SEGMRMK セグメント・タイプは、データベース・セグメントに関するユーザーのコメントを含んでいます。

セグメント名
SEGMRMK

親名　SEGM

順序フィールド
SGMCMSEQ

セグメント長
264 バイト

表 *32. SEGMRMK* セグメント・マップ：

オフセット (バイト)	長さ (バイト)	データ・タイプ	フィールド名	説明	固有キー・フィールド
1	2	X	LEN	このセグメントの長さ	
3	2	X	CTL	制御フィールド	
5	2	X	SGMCMSEQ	シーケンス・フィールド、タイプ = U	X
5	2	X	SEQNUM	シーケンス番号	
7	2	C	FILLER	予約済み	
9	256	C	REMARKS	親 SEGM に関するユーザーのコメント	

XDFLD セグメント・タイプのフォーマット

IMS カタログの XDFLD セグメント・タイプは、副次索引関係の索引付きフィールドに関するメタデータを含んでいます。

各 XDFLD セグメント・インスタンスは、副次索引関係を定義している LCHILD セグメント・インスタンスの直接の子です。

セグメント名
XDFLD

親名　LCHILD

順序フィールド
XDFLDSEQ

セグメント長
200 バイト

表 *33. XDFLD* セグメント・タイプのフォーマット：

オフセット (バイト)	長さ (バイト)	データ・タイプ	フィールド名	説明	固有キー・フィールド
1	2	X	LEN	このセグメントの長さ。	
3	2	X	CTL	制御フィールド。	

表 33. *XDFLD* セグメント・タイプのフォーマット (続き):

オフセット (バイト)	長さ (バイト)	データ・タイプ	フィールド名	説明	固有キー・フィールド
5	2	X	XDFLDSEQ	シーケンス・フィールド、タイプ = U。	X
5	2	X	SEQNUM	シーケンス番号。	
9	8	C	NAME	このフィールドの名前。	
17	8	C	SEGMENT	副次索引関係の索引ソース・セグメント・タイプの名前を示します。	
25	8	C	SRCH1	副次索引で検索フィールドとして使用できる索引ソース・セグメント内の最大 5 つのフィールドのうちの 1 つを示します。	
33	8	C	SRCH2	副次索引で検索フィールドとして使用できる索引ソース・セグメント内の最大 5 つのフィールドのうちの 1 つを示します。	
41	8	C	SRCH3	副次索引で検索フィールドとして使用できる索引ソース・セグメント内の最大 5 つのフィールドのうちの 1 つを示します。	
49	8	C	SRCH4	副次索引で検索フィールドとして使用できる索引ソース・セグメント内の最大 5 つのフィールドのうちの 1 つを示します。	
57	8	C	SRCH5	副次索引で検索フィールドとして使用できる索引ソース・セグメント内の最大 5 つのフィールドのうちの 1 つを示します。	
65	8	C	SUBSEQ1	副次索引のサブシーケンス・フィールドとして使用される索引ソース・セグメント内の最大 5 つのフィールドのうちの 1 つを示します。	
73	8	C	SUBSEQ2	副次索引のサブシーケンス・フィールドとして使用される索引ソース・セグメント内の最大 5 つのフィールドのうちの 1 つを示します。	
81	8	C	SUBSEQ3	副次索引のサブシーケンス・フィールドとして使用される索引ソース・セグメント内の最大 5 つのフィールドのうちの 1 つを示します。	
89	8	C	SUBSEQ4	副次索引のサブシーケンス・フィールドとして使用される索引ソース・セグメント内の最大 5 つのフィールドのうちの 1 つを示します。	
97	8	C	SUBSEQ5	副次索引のサブシーケンス・フィールドとして使用される索引ソース・セグメント内の最大 5 つのフィールドのうちの 1 つを示します。	
105	8	C	DDATA1	副次索引の重複データ・フィールドとして使用される索引ソース・セグメント内の最大 5 つのフィールドのうちの 1 つを示します。	
113	8	C	DDATA2	副次索引の重複データ・フィールドとして使用される索引ソース・セグメント内の最大 5 つのフィールドのうちの 1 つを示します。	

表 33. *XDFLD* セグメント・タイプのフォーマット (続き) :

オフセット (バイト)	長さ (バイト)	データ・タイプ	フィールド名	説明	固有キー・フィールド
121	8	C	DDATA3	副次索引の重複データ・フィールドとして使用される索引ソース・セグメント内の最大 5 つのフィールドのうちの 1 つを示します。	
129	8	C	DDATA4	副次索引の重複データ・フィールドとして使用される索引ソース・セグメント内の最大 5 つのフィールドのうちの 1 つを示します。	
137	8	C	DDATA5	副次索引の重複データ・フィールドとして使用される索引ソース・セグメント内の最大 5 つのフィールドのうちの 1 つを示します。	
145	8	C	EXITRTN	この副次索引関係用にユーザーが提供する副次索引データベース保守出口ルーチンの名前。	
153	8	C	PSELRTN	高速機能 1 次データベースの副次索引として定義されたこの HISAM データベースまたは SHISAM データベースに対してユーザー区分化が要求されたときに使用される、ユーザー提供の区画選択出口ルーチンの名前。	
161	1	C	PSELOPT	最初の区画の終わりを過ぎても処理できる GN 呼び出し用に、ユーザー区画グループ内の区画データベースを論理グループ化する方法を以下のように示します。 **M (複数グループ化)** 選択されたユーザー区画および後続の区分データベースは、1 次 DEDB カタログ・レコード内の LCHILD セグメント・インスタンスの NAME フィールドで定義されているとおりにグループに組み込まれます。 **S (単一グループ化)** 選択されたユーザー区画データベースだけが使用されます。	
162	3	C	FILLER01	予約済み。	
165	5	C	CONSTANT	特定の副次索引内にあるすべての索引ポインターを識別する文字を示します。この値は、同じデータベースに保管されているさまざまな副次索引のポインターを区別します。	
170	5	X	NULLVAL	索引検索フィールドのポインター抑止値。この値が索引ソース・セグメントのすべての SRCH フィールドに入っていると、索引ポインターは作成されません。	
175	26	C	FILLER	予約済み。	
175	26	C	FILLER02	この XDFLD の外部エイリアス名。	

関連概念:

363 ページの『第 16 章 副次索引の作成』

関連資料:

- XDFLD ステートメント (システム・ユーティリティー)
- 副次索引データベース保守出口ルーチン (出口ルーチン)

XDFLDRMK セグメント・タイプのフォーマット

IMS カタログの XDFLDRMK セグメント・タイプには、親 XDFLD セグメント・インスタンスで定義されている XDFLD セグメント・タイプに関するユーザー指定のコメントが含まれます。

セグメント名
: XDFLDRMK

親名 XDFLD

順序フィールド
: XDFRMSEQ

セグメント長
: 264 バイト

表 *34. XDFLDRMK* セグメント・マップ :

オフセット (バイト)	長さ (バイト)	データ・タイプ	フィールド名	説明	固有キー・フィールド
1	2	X	LEN	このセグメントの長さ	
3	2	X	CTL	制御フィールド	
5	2	X	XDFRMSEQ	シーケンス・フィールド、タイプ = U	X
5	2	X	SEQNUM	シーケンス番号	
7	2	X	RMKLEN	注釈の長さ	
9	256	C	REMARKS	親 XDFLD ステートメント定義に関するユーザー指定のコメント	

PSB レコード・セグメントのフォーマット

IMS カタログ内の PSB レコード・セグメントは、IMS データベースのアプリケーション・プログラム・ビューとスキーマに関する情報を保管するために使用されます。これらは、プログラム仕様ブロック (PSB) およびプログラム連絡ブロック (PCB) とも呼ばれます。

次の図は、PSB の IMS カタログ・レコードの編成を大まかに示したものです。

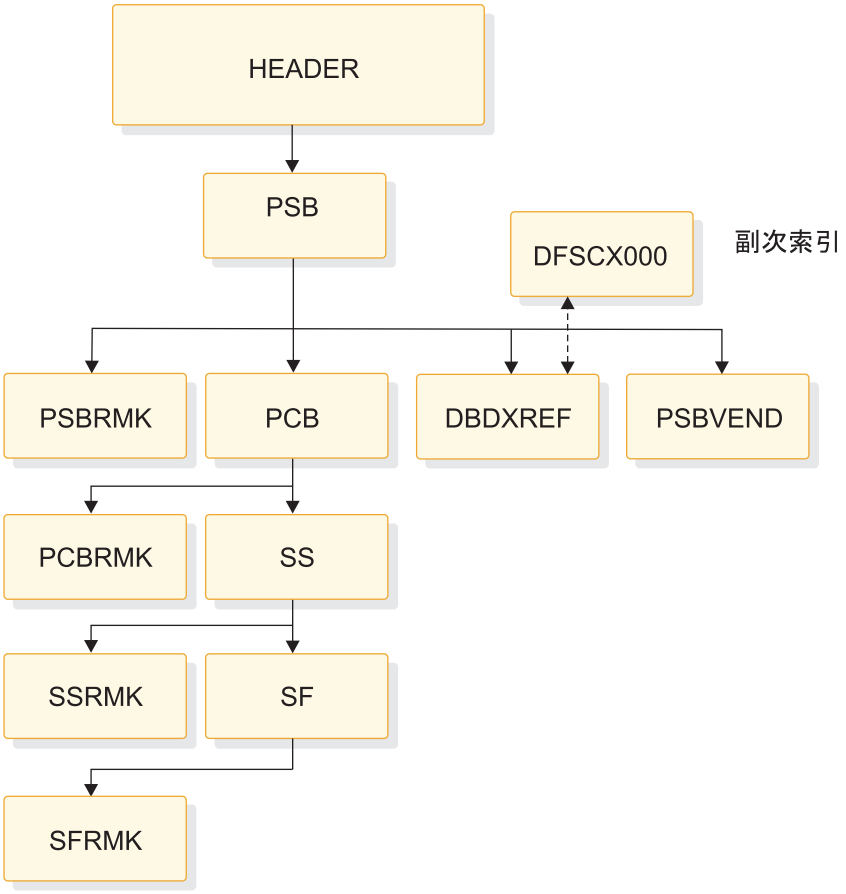


図 *18.* プログラム仕様ブロックの *IMS* カタログ・レコードのフォーマット

DBDXREF セグメント・タイプのフォーマット

IMS カタログの DBDXREF セグメント・タイプは、プログラム仕様ブロック (PSB) のインテント・リスト内の DBD に関するメタデータを含んでいます。

セグメント名
: DBDXREF

親名
: PSB

順序フィールド
: DBDXSEQ

セグメント長
: 48 バイト

表 *35. DBDXREF* セグメント・マップ :

オフセット (バイト)	長さ (バイト)	データ・タイプ	フィールド名	説明	固有キー・フィールド
1	2	X	LEN	長さ	
3	2	X	CTL	制御フィールド	

表 *35. DBDXREF* セグメント・マップ *(*続き*)* :

オフセット (バイト)	長さ (バイト)	データ・タイプ	フィールド名	説明	固有キー・フィールド
5	2	X	DBDXSEQ	シーケンス・フィールド、タイプ = U	X
5	2	X	SEQNUM	シーケンス番号	
9	4	X	CATVERS	カタログ・バージョン番号	
13	13	C	TSVERS	このバージョンの ACB 生成タイム・スタンプ (フォーマットは *yyDDDHHmmssff*)	
26	3	C	FILLER	予約済み	
29	8	C	IMSNAME	DBD の名前	
37	8	C	PSBNAME	PSB インテント・リスト内の IMSNAME フィールドで指名されている DBD を含んでいる PSB の名前	

関連概念:

115 ページの『第 11 章 IMS カタログの副次索引』

PCB セグメント・タイプのフォーマット

IMS カタログの PCB セグメント・タイプは、プログラム制御ブロック定義に関するメタデータを含んでいます。

このセグメント・タイプ内の情報は、PSBGEN ユーティリティーの PCB ステートメントのパラメーターに基づいて生成されます。

セグメント名
: PCB

親名 PSB

順序フィールド
: PCBSEQ

セグメント長
: 288 バイト

表 *36. PCB* セグメント・タイプのフォーマット :

オフセット (バイト)	長さ (バイト)	データ・タイプ	フィールド名	説明	固有キー・フィールド
1	2	X	LEN	このセグメントの長さ	
3	2	X	CTL	制御フィールド	
5	2	X	PCBSEQ	シーケンス・フィールド、タイプ = U	X
5	2	X	SEQNUM	シーケンス番号	

表 *36. PCB* セグメント・タイプのフォーマット *(*続き*)*:

オフセット (バイト)	長さ (バイト)	データ・タイプ	フィールド名	説明	固有キー・フィールド
9	8	C	IMSNAME	この PCB のデータベース・セグメントの 1 次ソースである物理または論理 DBD の名前。副次索引および相互参照を使用して、この PCB の論理データ構造に他のデータベースが追加される場合があります。 代替 PCB の中では、このフィールドは出力メッセージの送信先の LTERM 名です。	
17	8	C	PCBNAME	PCB ステートメントの PCBNAME パラメーターまたは LABEL パラメーターに基づいた、この PCB の 8 文字の IMS 名。	
25	8	C	LABEL	PCB ラベル・パラメーターに基づいた、この PCB の 8 文字の IMS 名。このフィールドにデータが入っている場合、PCBNAME フィールドにはブランクが入っています。	
33	4	C	TYPE	この PCB が標準 PCB か代替 PCB かを識別します。標準 PCB は、出力メッセージを入力メッセージのソースに返します。代替 PCB は、端末キューやトランザクション・キューなど、別の宛先にメッセージを返します。 **DB** 標準 PCB **TP** 代替 PCB	
37	4	C	PROCOPT	この PCB の処理オプションを識別します。処理オプションは、PCB を使用するアプリケーション・プログラムで実行できる操作のタイプを定義します。1 つの PCB に最大 4 つのオプションを使用できます。	
41	8	C	PROCSEQ	IMSNAME フィールドで識別されたデータベースの副次索引の名前。この PCB を使用するアプリケーション・プログラムは、1 次データベースでなく副次索引の処理シーケンスを使用します。	
49	8	C	PROCSEQD	IMSNAME フィールドで識別された高速機能データベースの副次索引の名前。この PCB を使用するアプリケーション・プログラムは、1 次高速機能データベースでなく副次索引の処理シーケンスを使用します。	
57	2	X	KEYLEN	この PCB によってアクセスされるデータ構造内で使用されているセンシティブ・セグメントの階層パスの最も長い連結キーのバイト数。	
59	2	X	COPIES	この PCB 用に存在するランタイム・コピーの数。この値は XQUERY 処理に使用されます。	
61	4	C	VIEW	この高速機能データベース用の PCB が、DEDB コミット・ビューと MSDB コミット・ビューのどちらを使用するかを示します。	

表 *36. PCB* セグメント・タイプのフォーマット *(*続き*)*:

オフセット (バイト)	長さ (バイト)	データ・タイプ	フィールド名	説明	固有キー・フィールド
65	1	C	ALTRESP	応答モード、会話モード、または排他モードで、端末応答メッセージに標準入出力 PCB の代わりにこの代替 PCB を使用できるかどうかを示します。	
66	1	C	EXPRESS	この代替 PCB を使用するアプリケーション・プログラムが異常終了したときに、キューに入っているメッセージを送信するのか (Y) それともバックアウトするのか (N) を示します。	
67	1	C	MODIFY	この代替 PCB の宛先名を動的に変更できるかどうかを指定します。	
68	1	C	SAMETRM	この代替 PCB の宛先論理端末が入力メッセージを送信した論理端末と同じものであることを IMS が検査するかどうかを示します。	
69	1	C	SB	この PCB を、可能であれば順次バッファリングでバッファーに入れるかどうかを示します。	
70	1	C	POS	この PCB がターゲット・データ構造内で単一位置付け (S) と複数位置付け (M) のどちらを使用するかを識別します。	
71	1	C	LIST	アプリケーション・プログラムに制御が渡されるとき、アプリケーション・プログラムに渡される PCB リストにこの PCB を含めるかどうかを示します。	
72	1	C	PSELOPT	この PCB が、SSA 処理を使用しない修飾 GN 呼び出し用のユーザー区画データベースを、ユーザー区画データベース内でデータの終わりに到達する前にどのように論理グループ化するかを示します。 **M** 選択されたユーザー区画データベースおよびユーザー・データ区画内の後続のユーザー区画データベースは、1 次高速機能データベース DBD の LCHILD 定義の NAME フィールドで物理的に定義されているとおりにグループ化されます。 **S** 選択されたユーザー区画データベースだけが PCB によって使用されます。後続のユーザー区画データベースは、論理グループに追加されません。 このフィールドは、高速機能 2 次データベースにのみ使用されます。	
73	1	C	FILLER01	予約済み	

表 *36. PCB* セグメント・タイプのフォーマット *(*続き*)*:

オフセット (バイト)	長さ (バイト)	データ・タイプ	フィールド名	説明	固有キー・フィールド
74	7	C	ACCESS	この PCB が、ターゲット・データベースへのアクセスに通常の副次索引を使用するのか、それとも別個の論理データベースを使用するのかを示します。	
81	128	C	NAME	この PCB の外部エイリアス名	
209	13	C	DBDTS	DBDGEN タイム・スタンプ	
222	4	X	DBVER	要求される DBD のバージョン	
226	63	N/A	N/A	予約済みバイト	

関連概念:

565 ページの『PSBGEN ユーティリティーへの入力としてのプログラム仕様ブロックのコーディング』

関連資料:

全機能または高速機能データベースの PCB ステートメント (システム・ユーティリティー)

代替 PCB ステートメント (システム・ユーティリティー)

PCBRMK セグメント・タイプのフォーマット

IMS カタログの PCBRMK セグメント・タイプは、IMS プログラム制御ブロックに関するユーザー指定のコメントを含んでいます。

このセグメントは、コメントの対象である PCB セグメント・インスタンスの直接の子です。

セグメント名
: PCBRMK

親名 PCB

順序フィールド
: PCBCMSEQ

セグメント長
: 264 バイト

表 *37. PCBRMK* セグメント・マップ:

オフセット (バイト)	長さ (バイト)	データ・タイプ	フィールド名	説明	固有キー・フィールド
1	2	X	LEN	このセグメントの長さ	
3	2	X	CTL	制御フィールド	
5	2	X	PCBCMSEQ	シーケンス・フィールド、タイプ = U	X

表 *37. PCBRMK* セグメント・マップ *(*続き*)*:

オフセット (バイト)	長さ (バイト)	データ・タイプ	フィールド名	説明	固有キー・フィールド
5	2	X	SEQNUM	シーケンス番号	
7	2	C	FILLER	予約済み	
9	256	C	REMARKS	この PCB に関するユーザーのコメント	

PSB セグメント・タイプのフォーマット

IMS カタログの PSB セグメント・タイプは、IMS プログラム仕様ブロックに関するメタデータを含んでいます。

セグメント名
: PSB

親名 HEADER

順序フィールド
: PSBSEQ

セグメント長
: 88 バイト

表 *38. PSB* セグメント・タイプのフォーマット:

オフセット (バイト)	長さ (バイト)	データ・タイプ	フィールド名	説明	固有キー・フィールド
1	2	X	LEN	このセグメントの長さ	
3	2	X	CTL	制御フィールド	
5	2	X	SEQNUM	シーケンス番号	
9	17	X	PSBSEQ	シーケンス・フィールド、タイプ = U	X
9	4	X	CATVERS	カタログ・バージョン番号	
13	13	C	TSVERS	このバージョンの ACB 生成タイム・スタンプ (フォーマットは *yyDDDHHmmssff*)	
26	1	C	FILLER	予約済み	
27	2	X	RLVL	ACB 生成ユーティリティーのリリース・レベル	
37	6	C	LANG	このアプリケーションで使用するメッセージ処理プログラムまたはバッチ処理プログラムのコンパイラー言語	

表 38. PSB セグメント・タイプのフォーマット (続き):

オフセット (バイト)	長さ (バイト)	データ・タイプ	フィールド名	説明	固有キー・フィールド
43	2	X	IOERCC	IMS が正常に終了し、この PSB を使用したプログラムの実行中にいずれかのデータベースに 1 つ以上のエラーが発生したときに、オペレーティング・システムに返される条件コード。この値が 451 で IMS 異常終了コードが U451 の場合、IMS は条件コードを発行せずに U451 異常終了で終了します。IMS 異常終了コードが U451 でない場合、IMS は DFS0426I メッセージも発行します。	
45	4	C	IOERWTOR	データベース・エラーの後、IMS が終了する前に、WTOR DFS0451A エラー・メッセージを発行してオペレーターが ABEND コマンドで応答するのを待つかどうかを示します。	
49	2	X	MAXQ	この PSB を使用して発行できる、Qx コマンド・コードによる同期点間のデータベース呼び出しの最大数。	
51	2	X	LOCKMAX	この PSB を使用して、アプリケーション・プログラムが一度に取得できるロックの最大数。値は、1000 ロック単位です。値 0 は、アプリケーション・プログラムがこの PSB を使用して取得できるロックの数に制限がないことを示します。	
53	1	C	CMPAT	PSB がバッチ DL/I で実行されている場合でも、常にその PSB に入出力 PCBがあるかのように扱うかどうかを示します。	
54	1	C	OLIC	この PSB のユーザーがオンライン・データベース・イメージ・コピー・ユーティリティーまたは調査ユーティリティーを実行できるかどうかを示します。	
55	1	C	GSROLBOK	以下の条件が満たされた場合に、非 GSAM データベースについて内部 ROLB 呼び出し (Y) とタイプ 777 ユーザー異常終了 (N) のどちらを発行するかを示します。 • アプリケーションが非メッセージ・ドリブン BMP である • PSB に GSAM PCB が含まれている • スレッド作成時または SQL 呼び出し時に DB2® for z/OS がデッドロックを報告した	
56	1	C	DBLEVEL	要求されるデフォルトの DBD のバージョン	
57	8	C	FILLER01	予約済み	
65	8	C	CREATEBY	予約済み	
73	16	C	FILLER03	予約済み	

関連概念:

565 ページの『PSBGEN ユーティリティーへの入力としてのプログラム仕様ブロックのコーディング』

関連資料:

プログラム仕様ブロック (PSB) 生成ユーティリティー (システム・ユーティリティー)

PSBVEND セグメント・タイプのフォーマット

IMS カタログの PSBVEND セグメント・タイプは、短縮ヘッダーと、それに続く大きなブロックの未フォーマット・スペースを含んでいます。

このセグメント・タイプは、ベンダー提供ツールが使用するために予約されています。

セグメント名
PSBVEND

親名 PSB

順序フィールド
PVNDSEQ

セグメント長
4000 バイト

表 *39. PSBVEND* セグメント・マップ

オフセット (バイト)	長さ (バイト)	データ・タイプ	フィールド名	説明	固有キー・フィールド
1	2	X	LEN	このセグメントの長さ	
3	2	X	CTL	制御フィールド	
5	2	X	SEQNUM	セグメント・シーケンス番号	
7	2	C	FILLER	予約済み	
9	3992	X	DATA	ベンダー製品の PSB データ	
5	2	X	PVNDSEQ	シーケンス・フィールド、タイプ = U	X

PSBRMK セグメント・タイプのフォーマット

IMS カタログの PSBRMK セグメント・タイプは、IMS プログラム仕様ブロックに関するユーザー指定のコメントを含んでいます。

このセグメントは、コメントの対象である PSB セグメント・インスタンスの直接の子です。

セグメント名
PSBRMK

親名 PSB

順序フィールド
PSBCMSEQ

セグメント長
264 バイト

表 *40. PSBRMK* セグメント・マップ :

オフセット (バイト)	長さ (バイト)	データ・タイプ	フィールド名	説明	固有キー・フィールド
1	2	X	LEN	このセグメントの長さ	
3	2	X	CTL	制御フィールド	
5	2	X	PSBCMSEQ	シーケンス・フィールド、タイプ = U	X
5	2	X	SEQNUM	シーケンス番号	
7	2	C	FILLER	予約済み	
9	256	C	REMARKS	この PSB に関するユーザーのコメント	

SF セグメント・タイプのフォーマット

IMS カタログの SF セグメント・タイプは、プログラム制御ブロック (PCB) 内のセンシティブ・セグメントのセンシティブ・フィールド定義に関する情報を含んでいます。

セグメント名
SF

親名 SS

順序フィールド
SENFLSEQ

セグメント長
40 バイト

表 *41. SF* セグメント・タイプのフォーマット :

オフセット (バイト)	長さ (バイト)	データ・タイプ	フィールド名	説明	固有キー・フィールド
1	2	X	LEN	このセグメントの長さ	
3	2	X	CTL	制御フィールド	
5	2	X	SENFLSEQ	シーケンス・フィールド、タイプ = U	X
5	2	X	SEQNUM	シーケンス番号	
9	8	C	IMSNAME	FLD カタログ・レコード内で定義されたとおりのフィールドの IMS 名	
17	2	X	START	ユーザー入出力域に返されたとおりの、セグメントの先頭から数えたフィールドのオフセット	
19	1	C	REPL	置換呼び出しでフィールドを変更できるかどうかを示します。	
20	21	C	FILLER	予約済み	

SFRMK セグメント・タイプのフォーマット

IMS カタログの SFRMK セグメント・タイプは、センシティブ・フィールド定義に関するユーザーのコメントを含んでいます。

このセグメントは、コメントの対象である SF セグメント・インスタンスの直接の子です。

セグメント名
SFRMK

親名 SF

順序フィールド
SENFLSEQ

セグメント長
264 バイト

表 *42. SFRMK* セグメント・タイプのフォーマット：

オフセット (バイト)	長さ (バイト)	データ・タイプ	フィールド名	説明	固有キー・フィールド
1	2	X	LEN	このセグメントの長さ	
3	2	X	CTL	制御フィールド	
5	2	X	SENFLSEQ	シーケンス・フィールド、タイプ = U	X
5	2	X	SEQNUM	シーケンス番号	
7	2	C	FILLER	予約済み	
9	256	C	REMARKS	親 SEGM に関するユーザーのコメント	

SS セグメント・タイプのフォーマット

IMS カタログの SS セグメント・タイプは、プログラム制御ブロック (PCB) のセンシティブ・セグメント定義に関する情報を含んでいます。

セグメント名
SS

親名 PCB

順序フィールド
SENSGSEQ

セグメント長
328 バイト

表 *43. SS* セグメント・タイプのフォーマット：

オフセット (バイト)	長さ (バイト)	データ・タイプ	フィールド名	説明	固有キー・フィールド
1	2	X	LEN	このセグメントの長さ。	
3	2	X	CTL	制御フィールド。	

表 *43. SS* セグメント・タイプのフォーマット (続き) :

オフセット (バイト)	長さ (バイト)	データ・タイプ	フィールド名	説明	固有キー・フィールド
5	2	X	SENSGSEQ	シーケンス・フィールド、タイプ = U。	X
5	2	X	SEQNUM	シーケンス番号。	
9	8	C	IMSNAME	センシティブ・セグメント・タイプの名前。	
17	8	C	PARENT	センシティブ・セグメント・タイプの直接の親の名前。このフィールドの値が 0 の場合、これはセンシティブ・ルート・セグメント・タイプです。	
25	4	C	PROCOPT	このセンシティブ・セグメントに使用するのに有効な処理オプション。	
29	2	X	IDXCNT	このセンシティブ・セグメント・タイプへの有効なパスを持つ副次索引の数。	
31	2	X	SSPTRCNT	このセンシティブ・セグメントのサブセット・ポインターの数。	
33	256	C	INDICES	このセンシティブ・セグメント・タイプへの有効なパスを持つ副次索引データベースの最大 32 個の DBD 名からなるリスト。	
289	2	X	SSPNUM01	このサブセット・ポインターが PSB ソース内で指定された順序。	
291	1	C	SSPSEN01	このサブセット・ポインターのセンシティビティー・タイプ: 読み取り (R) または更新 (U) センシティビティー。	
292	1	C	SSPFILL1	予約済み。	
293	2	X	SSPNUM02	このサブセット・ポインターが PSB ソース内で指定された順序。	
295	1	C	SSPSEN02	このサブセット・ポインターのセンシティビティー・タイプ: 読み取り (R) または更新 (U) センシティビティー。	
296	1	C	SSPFILL2	予約済み。	
297	2	X	SSPNUM03	このサブセット・ポインターが PSB ソース内で指定された順序。	
299	1	C	SSPSEN03	このサブセット・ポインターのセンシティビティー・タイプ: 読み取り (R) または更新 (U) センシティビティー。	
300	1	C	SSPFILL3	予約済み。	
301	2	X	SSPNUM04	このサブセット・ポインターが PSB ソース内で指定された順序。	
303	1	C	SSPSEN04	このサブセット・ポインターのセンシティビティー・タイプ: 読み取り (R) または更新 (U) センシティビティー。	
304	1	C	SSPFILL4	予約済み。	
305	2	X	SSPNUM05	このサブセット・ポインターが PSB ソース内で指定された順序。	

表 43. SS セグメント・タイプのフォーマット (続き):

オフセット (バイト)	長さ (バイト)	データ・タイプ	フィールド名	説明	固有キー・フィールド
307	1	C	SSPSEN05	このサブセット・ポインターのセンシティビティー・タイプ: 読み取り (R) または更新 (U) センシティビティー。	
308	1	C	SSPFILL5	予約済み。	
309	2	X	SSPNUM06	このサブセット・ポインターが PSB ソース内で指定された順序。	
311	1	C	SSPSEN06	このサブセット・ポインターのセンシティビティー・タイプ: 読み取り (R) または更新 (U) センシティビティー。	
312	1	C	SSPFILL6	予約済み。	
313	2	X	SSPNUM07	このサブセット・ポインターが PSB ソース内で指定された順序。	
315	1	C	SSPSEN07	このサブセット・ポインターのセンシティビティー・タイプ: 読み取り (R) または更新 (U) センシティビティー。	
316	1	C	SSPFILL7	予約済み。	
317	2	X	SSPNUM08	このサブセット・ポインターが PSB ソース内で指定された順序。	
319	1	C	SSPSEN08	このサブセット・ポインターのセンシティビティー・タイプ: 読み取り (R) または更新 (U) センシティビティー。	
320	1	C	SSPFILL8	予約済み。	
321	8	C	FILLER	予約済み。	

関連概念:

567 ページの『SENSEG ステートメント』

関連資料:

SENSEG ステートメント (システム・ユーティリティー)

SSRMK セグメント・タイプのフォーマット

IMS カタログの SSRMK セグメント・タイプは、センシティブ・セグメント定義に関するユーザー指定のコメントを含んでいます。

このセグメントは、コメントの対象である SS セグメント・インスタンスの直接の子です。

セグメント名
SSRMK

親名 SS

順序フィールド
SENSGSEQ

セグメント長
264 バイト

表 *44. SSRMK* セグメント・タイプのフォーマット：

オフセット (バイト)	長さ (バイト)	データ・タイプ	フィールド名	説明	固有キー・フィールド
1	2	X	LEN	このセグメントの長さ	
3	2	X	CTL	制御フィールド	
5	2	X	SENSGSEQ	シーケンス・フィールド、タイプ = U	X
5	2	X	SEQNUM	シーケンス番号	
7	2	C	FILLER	予約済み	
9	256	C	REMARKS	このセンシティブ・セグメント定義に関するユーザーのコメント	

第 11 章 IMS カタログの副次索引

IMS カタログ副次索引は、特定の DBD を参照している PSB を判別するための短い処理パスを提供します。

概要

IMS アプリケーション環境では、1 つまたは多数の IMS PSB を介して互いにアクセスする任意の数のデータベースを使用できます。IMS データベースを安全に変更または削除するために、データベース管理者は、どのアプリケーションがデータベースに依存しているかを前もって知っておく必要があります。IMS カタログ副次索引は、特定の DBD に依存している PSB を判別するための高速の処理パスを提供します。

IMS カタログ副次索引 (DFSCX000) のルート・セグメント・タイプ DBDPSB は、IMS カタログ・データベース内で DBDXREF セグメント・タイプに論理リンクされています。*DFSC* の接頭部は、カタログ別名接頭部が IMS に対して定義されていると、その接頭部に置き換えられます。

DBDXREF セグメント・タイプの IMSNAME フィールドは、副次索引関係用に DBD2PSB XDFLD として索引付けされています。IMSNAME フィールドには、カタログ・レコードによって記述された PSB が参照する DBD の 8 文字の IMS 名が入っています。DBDXREF セグメントは、IMSNAME (DBD 名) フィールド、PSBNAME フィールド、または TSVERS フィールドで検索できます。

使用法

IMS カタログ副次索引を使用して、次のような方法で IMS カタログ・メタデータを処理できます。

- DFSCATSX PCB を使用して、PROCSEQ=DFSCX000 で 1 次カタログ・データベース (DFSCD000) を処理します。DBDXREF セグメント・タイプは、この PCB 内でセンシティブ・セグメントとして定義されている唯一のセグメント・タイプです。
- DFSCATX0 PCB を使用して、カタログ副次索引を直接処理します。副次索引ルート・セグメント・タイプ (DBDPSB) は、副次索引内の唯一のセグメント・タイプです。

どちらの PCB も IMS カタログ PSB のDFSCP*xxx* に含まれています。これらの PSB は、IMS カタログがアクティブのときにユーザー PSB に動的に接続されます。

関連概念:

363 ページの『第 16 章 副次索引の作成』

IMS カタログを使用したアプリケーション・プログラミング (アプリケーション・プログラミング)

関連資料:

102 ページの『DBDXREF セグメント・タイプのフォーマット』

第 3 部 データベースのタイプと機能

IMS データベースには、全機能と高速機能の 2 つの一般的クラスがあります。各クラスにはさまざまなタイプのデータベースがあり、各データベース・タイプにはさまざまな機能と特性があります。

関連概念:

13 ページの『IMS データベース・タイプ』

第 12 章 IMS データベースのタイプと機能の要約

次の表は、さまざまなタイプの IMS データベースの特性、機能、およびオプションを要約したものです。

表 *45.* 各種のデータベース・タイプのデータベース特性とオプションの要約

特性	HSAM	HISAM	HDAM	PHDAM	HIDAM	PHIDAM	DEDB	MSDB
階層構造	Y	Y	Y	Y	Y	Y	Y	N
直接アクセス・ストレージ	Y	Y	Y	Y	Y	Y	Y	N
複数データ・セット・グループ	N	N	Y	Y	Y	Y	N	N
論理関係	N	Y	Y	Y	Y	Y	N	N
可変長セグメント	N	Y	Y	Y	Y	Y	Y	N
セグメント編集/圧縮	N	Y	Y	Y	Y	Y	Y	N
データ・キャプチャー出口ルーチン	N	Y	Y	Y	Y	Y	Y	N
フィールド・レベル・センシティビティー	Y	Y	Y	Y	Y	Y	N	N
1 次索引	N	Y	N	N	Y	Y	N	N
副次索引	N	Y	Y	Y	Y	Y	Y	N
ロギング、リカバリー、オフライン再編成	N	Y	Y	Y	Y	Y	Y	Y
VSAM	N	Y	Y	Y	Y	Y	Y	N/A
OSAM	N	N	Y	Y	Y	Y	N	N/A
QSAM/BSAM	Y	N	N	N	N	N	N	N/A
ブール演算子	Y	Y	Y	Y	Y	Y	Y	N
コマンド・コード	Y	Y	Y	Y	Y	Y	Y	N
サブセット・ポインター	N	N	N	N	N	N	Y	N
主記憶域の使用	N	N	N	N	N	N	N	Y
高度の並行機能 (フィールド呼び出し)	N	N	N	N	N	N	N	Y
圧縮	Y	Y	Y	Y	Y	Y	Y	N
DBRC サポート	Y	Y	Y	必須	Y	必須[1]	Y	N/A
区分化サポート	N	N	N	Y	N	Y	Y	N
データ共用	Y	Y	Y	Y	Y	Y	Y	N
区画の共用	N	N	N	Y	N	Y	Y	N
ブロック・レベル共用	Y	Y	Y	Y	Y	Y	Y	N
エリアの共用	N/A	N/A	N/A	N/A	N/A	N/A	Y	N/A
レコードの非活動化	N	N	N	N	N	N	Y	N/A
データベース・サイズ	中	中	中	大	中	大	大	小

表 45. 各種のデータベース・タイプのデータベース特性とオプションの要約 (続き)

特性	HSAM	HISAM	HDAM	PHDAM	HIDAM	PHIDAM	DEDB	MSDB
オンライン・ユーティリティー	N	N	N	N	N	N	Y	N
オンライン再編成	N	N	N	Y	N	Y	Y	N
バッチ	Y	Y	Y	Y	Y	Y	N	N

表の注記:

1. IMS カタログは PHIDAM データベースです。他の HALDB データベースと異なり、IMS カタログの PHIDAM データベースは DBRC サポートを必要としません。しかし、テスト環境と開発環境を除いて、DBRC サポートが強く推奨されます。

関連概念:

13 ページの『IMS データベース・タイプ』

211 ページの『第 14 章 高速機能データベース・タイプ』

121 ページの『第 13 章 全機能データベース・タイプ』

123 ページの『パフォーマンス考慮点の概要』

第 13 章 全機能データベース・タイプ

IMS 全機能データベースは、DL/I 呼び出しによってアクセスされる階層構造のデータベースです。IMS は、アプリケーション・プログラムが IMS データベースへの、セグメントの検索、置き換え、削除、追加を行えるようにします。

IMS では、12 種類のデータベースを定義することができます。これらの各種類はそれぞれ異なる編成または処理特性をもちます。 DEDB と MSDB を除くすべてのデータベース・タイプをこの章で説明します。

さまざまな種類のデータベース・タイプの違いを理解すれば、アプリケーションの処理要件に最適な種類を選択することができるようになります。

各データベース・タイプは、固有のアクセス方式を持っています。以下の表に、各データベース・タイプとそのアクセス方式を示します。

表 *46.* データベース・タイプとそのアクセス方式

データベースの種類	アクセス方式
HSAM	階層順次アクセス方式
HISAM	階層索引順次アクセス方式
SHSAM	単純階層順次アクセス方式
SHISAM	単純階層索引順次アクセス方式
GSAM	汎用順次アクセス方式
HDAM	階層直接アクセス方式
PHDAM	区分階層直接アクセス方式
HIDAM	階層索引直接アクセス方式
PHIDAM	区分階層索引直接アクセス方式
PSINDEX	区分副次索引データベース
DEDB	高速処理データベース (階層直接アクセス)
MSDB	主記憶データベース (階層直接アクセス)

各種のデータベースは、使用するアクセス方式に応じて、順次記憶方式と直接記憶方式の 2 つのグループに分類することができます。

関連概念:

119 ページの『第 12 章 IMS データベースのタイプと機能の要約』
211 ページの『高速処理データベース』
234 ページの『主記憶データベース (MSDB)』
35 ページの『第 3 回の設計レビュー』
36 ページの『第 4 回の設計レビュー』

順次記憶方式

HSAM、HISAM、SHSAM、SHISAM データベースは、データのアクセスに順次方式を使用します。

この方式では、データベース内のセグメントの階層順は、相互に物理的隣接の保管場所にセグメントを入れることによって維持されます。GSAM データベースもデータのアクセスに順次方式を使用しますが、GSAM データベースには階層、データベース・レコード、またはセグメントという概念はありません。

直接記憶方式

HDAM、PHDAM、HIDAM、DEDB、MSDB、および PHIDAM データベースでは、データのアクセスに直接法が使用されます。この方式では、直接アドレス・ポインターを各セグメントの接頭部に入れることにより、セグメントの階層順が維持されます。

関連概念:

123 ページの『パフォーマンス考慮点の概要』

DBCTL でサポートされるデータベース

IMS のデータベース制御 (DBCTL) 構成は、すべての IMS 全機能データベースをサポートします。

DBCTL でサポートされている全機能データベースには、次のものがあります。

- HSAM
- HISAM
- SHSAM
- SHISAM
- HDAM
- PHDAM
- HIDAM
- PHIDAM
- PSINDEX

IMS BMP 領域からは、独立したトランザクション管理サブシステムからと同じく DBCTL を通してデータベースにアクセスすることができます。 DBCTL はメッセージまたはトランザクションをサポートしないため、バッチ指向 BMP プログラムしかサポートされません。

複数の CICS オンライン・プログラムが、同じ IMS データベースに同時にアクセスすることができますが、IMS バッチ・プログラムはそのデータベースに対して排他的なアクセスが必要です (IMS データの共用を行わない場合)。

IMS データの共用によって IMS データベースへのアクセスを現在行っているバッチ・ジョブがある場合は、これを変換して、DBCTL を通してデータベースに直接

アクセスする BMP として実行すると、パフォーマンスを向上させることができます。さらに、DEDB にアクセスするために現行のバッチ・プログラムを BMP に変換することもできます。

関連概念:

オンラインでのバッチ処理: バッチ指向 BMP (アプリケーション・プログラミング)

関連資料:

EXEC パラメーター (IMS バッチ・メッセージ処理領域) (システム定義)

DCCTL でサポートされるデータベース

IMS の DCCTL 構成では、データベースと従属領域のいくつかの組み合わせがサポートされます。

IMS の DCCTL 構成によってサポートされるデータベースと従属領域の組み合わせには、次のものがあります。

- BMP 領域用の GSAM データベース
- 外部サブシステム接続機能 (ESAF) を使用した BMP 領域、MPP 領域、および IFP 領域用の Db2 for z/OS データベース
- DB2 リカバリー可能リソース・マネージャー・サービス接続機能 (RRSAF) を使用した JMP および JBP 領域用の Db2 for z/OS データベース

制約事項**:** DCCTL は、全機能データベースまたは高速機能データベースをサポートしません。

関連資料**:** RRSAF の詳細については、「*DB2 for z/OS* アプリケーション・プログラミングおよび *SQL* 解説書」を参照してください。

関連概念:

149 ページの『GSAM データベース』

外部サブシステム接続機能 (ESAF) (コミュニケーションおよびコネクション)

関連タスク:

DB2 接続機能 (コミュニケーションおよびコネクション)

関連資料:

IMS システム出口ルーチン (出口ルーチン)

パフォーマンス考慮点の概要

IMS データベースの機能上およびパフォーマンス上の特性は、IMS データベースのタイプによって異なります。自分の目的に最適なデータベース編成のタイプを調べて、判断する必要があります。

以下のリストでは、さまざまな全機能データベース・タイプのパフォーマンス特性を簡単に要約し、階層順次データベース、階層直接データベース、および汎用順次データベースの効率的な点と機能の不足している点を示します。

汎用順次 **(GSAM)**

- DCCTL によってサポートされる。
- 階層なし、データベース・レコードなし、セグメントなし、キーなし。
- DLET なし、REPL なし。
- ISRT はデータ・セットの末尾にレコードを追加する。
- GN および GU はバッチまたは BMP アプリケーションでのみ処理される。
- IMS シンボリック・チェックポイント呼び出しおよびチェックポイントからの再始動が可能 (VSAM がロードされたデータベースを除く)。
- IMS へのデータの変換およびデータの引き渡しに適する。
- MPP および JMP 領域からアクセス不能。
- スペース効率は良い。
- 時間効率は良くない。

VSAM

- 固定長または可変長のレコードが使用可能。
- VSAM ESDS DASD に保管される。
- IMS シンボリック・チェックポイント呼び出し。
- チェックポイントからの再始動はなし。

BSAM/QSAM

- 固定長、可変長、不定長レコードが使用可能。
- BSAM/QSAM DS テープまたは DASD に保管される。
- IMS シンボリック・チェックポイント呼び出しおよびチェックポイントからの再始動が可能。

階層順次

セグメントは物理的な連続性によってリンクされる。

HSAM

- DBCTL によってサポートされる。
- テープまたは DASD に保管されているルート・セグメントおよび従属セグメントへの物理順次アクセス。
- ISRT はデータベースのロード時にのみ可能。
- GU、GN、および GNP が可能。
- データベースのマージと新規データベースの書き込みによるデータベース更新。
- QSAM および BSAM アクセス可能。
- スペース効率は良いが、時間効率は良くない。
- 順次アクセス

HISAM

- DBCTL によってサポートされる。
- ルート・セグメントへの階層索引付きアクセス。
- 従属セグメントへの順次アクセス。
- DASD に保管される。
- VSAM アクセス可能。
- すべての DL/I 呼び出しが可能。
- 索引がルート・セグメントのシーケンス・フィールドにある。
- 更新頻度の低いデータベースに適する。
- 更新頻度が高い場合、スペース効率は良くない。
- SSA 修飾呼び出しにより、時間効率は良い。

SHSAM

- DBCTL によってサポートされる。
- ルート・セグメントのみへの単純階層順次アクセス方式。
- ISRT はデータベースのロード時にのみ可能。
- GU、GN、および GNP が可能。
- データベースの再ロードによるデータベース更新。
- QSAM および BSAM アクセス可能。
- IMS シンボリック・チェックポイント呼び出しおよびチェックポイントからの再始動が可能 (VSAM がロードされたデータベースを除く)。
- IMS へのデータの変換およびデータの引き渡しに適する。
- MPP および JMP 領域からアクセス不能。
- スペース効率は良い。
- 時間効率は良くない。

SHISAM

- DBCTL によってサポートされる。
- ルート・セグメントのみへの単純階層索引順次アクセス。
- DASD に保管される。
- VSAM アクセス可能。
- すべての DL/I 呼び出しが可能。
- IMS へのデータの変換およびデータの引き渡しに適する。
- スペース効率は良くない。
- 時間効率は良い。

階層直接

セグメントはポインターによってリンクされる。

HDAM

- DBCTL によってサポートされる。
- ルート・セグメントへのハッシュ・アクセス。
- 副次索引によるセグメントへの順次アクセス。
- すべての DL/I 呼び出しが可能。

- DASD 上の VSAM ESDS または OSAM データ・セットに保管される。
- レコードへの直接アクセスに適する。
- 階層ポインターが可能。
  - 従属セグメントへの階層順次アクセス。
  - 子/兄弟ポインターの場合より効率が良い。
  - 必要なスペースは子/兄弟ポインターの場合より少ない。
- 子/兄弟ポインターが可能。
  - ポインターへの直接アクセス。
  - 追加の索引 VSAM ESDS データベースにより、多くのスペースが必要。

HIDAM

- DBCTL によってサポートされる。
- ルート・セグメントへの索引付きアクセス。
- 従属セグメントへのポインター・アクセス。
- すべての DL/I 呼び出しが可能。
- DASD 上の VSAM ESDS または OSAM データ・セットに保管される。
- レコードへのランダム・アクセスおよび順次アクセスに適する。
- セグメント・パスへのランダム・アクセスに適する。
- 階層ポインターが可能。
  - 従属セグメントへの階層順次アクセス。
  - 子/兄弟ポインターの場合より効率が良い。
  - 必要なスペースは子/兄弟ポインターの場合より少ない。
- 子/兄弟ポインターが可能。
  - ポインターへの直接アクセス。
  - 追加の索引 VSAM ESDS データベースにより、多くのスペースが必要。

HALDB 区画階層直接

セグメントはポインターによってリンクされます。 HALDB データベースには、1 から 1 001 個の区画があります。ラージ・データベースとしては、HALDB データベースが最良の選択です。

PHDAM

- DBCTL によってサポートされる。
- 最大 1 001 個の区画をサポートする。
- 区画は最大 10 個のデータベース・データ・セットと 1 個の間接リスト・データ・セット (ILDS) をサポートする。
- 最大サイズは、OSAM データ・セットでは 4 GB または 8 GB、VSAM データ・セットでは 4 GB である。

- データベース内の区画は、そのデータベース内の他の区画から独立して割り振り、許可、処理、再編成、およびリカバリーを行うことができる。
- 区画を並列処理すると、再編成時間が短縮される。
- 個々の区画は、個別のルート・アドレス可能域 (RAA) を持つことができる。
- 論理関係および副次索引には間接ポインターが使用される。それによって、
  - データベース再編成後の間接ポインターの自動更新、すなわち、自己回復 が可能になる。
  - 各区画に ILDS が必要になる。
- ルート・セグメントへのハッシュ・アクセス。
- 副次索引によるセグメントへの順次アクセス。
- すべての DL/I 呼び出しが可能。
- DASD 上の VSAM ESDS または OSAM データ・セットに保管される。
- レコードへの直接アクセスに適する。
- 直接ポインターは論理関係の中で使用され、シンボリック・ポインターはサポートされない。
- 階層ポインターはない。
- 子/兄弟ポインターが可能。
  - ポインターへの直接アクセス。
  - 追加の索引 VSAM ESDS データベースにより、多くのスペースが必要。

PHIDAM

- DBCTL によってサポートされる。
- 最大 1 001 個の区画をサポートする。
- 区画は最大 10 個のデータベース・データ・セット、1 個の 1 次索引データ・セット、および 1 個の間接リスト・データ・セット (ILDS) をサポートする。
- 最大サイズは、OSAM データ・セットでは 4 GB または 8 GB、VSAM データ・セットでは 4 GB である。
- データベース内の区画は、そのデータベース内の他の区画から独立して割り振り、許可、処理、再編成、およびリカバリーを行うことができる。
- 区画を並列処理すると、再編成時間が短縮される。
- 論理関係および副次索引には間接ポインターが使用される。それによって、
  - データベース再編成後の間接ポインターの自動更新、すなわち、自己回復 が可能になる。
  - 各区画に ILDS が必要になる。
- ルート・セグメントへの索引付きアクセス。

- 1 次索引はリカバリー不能データベースなので、SLDS への移動時に圧縮される前であっても、データベース更新ログのほうが小さい。
- レコード・キーは各区画内で順番に保管される。区画相互間でレコードのシーケンスが維持されるかどうかは、使用される区画選択方法によって異なる。
- 従属セグメントへのポインター・アクセス。
- すべての DL/I 呼び出しが可能。
- DASD 上の VSAM ESDS または OSAM データ・セットに保管される。
- レコードへのランダム・アクセスおよび順次アクセスに適する。
- セグメント・パスへのランダム・アクセスに適する。
- 直接ポインターは論理関係の中で使用され、シンボリック・ポインターはサポートされない。
- 階層ポインターはない。
- 子/兄弟ポインターが可能。
  - ポインターへの直接アクセス。
  - 追加の索引 VSAM ESDS データベースにより、多くのスペースが必要。

HALDB 区分副次索引

PSINDEX

- DBCTL によってサポートされる。
- 最大 1 001 個の区画をサポートする。
- 区画は単一データ・セットのみをサポートする。
- DASD 上の VSAM KSDS データ・セットに保管される。
- VSAM データ・セットの最大サイズは 4 GB である。
- HALDB 自己回復ポインター処理のため、索引付きデータベースの再編成後に再作成する必要はない。
- 区分副次索引 (PSINDEX) の区画は、そのデータベース内の他の区画から独立して割り振り、許可、処理、再編成、およびリカバリーを行うことができる。
- セグメントの接頭部は、28 バイトの拡張ポインター・セット (EPS) と副次索引ターゲット・セグメントのルート・キーの長さとの両方を入れるために非区分副次索引より大きい。
- 共用副次索引をサポートしない。
- シンボリック・ポインターをサポートしない。
- 副次索引レコード・セグメントは固有キーを持たなければならない。

関連概念:

119 ページの『第 12 章 IMS データベースのタイプと機能の要約』

211 ページの『高速処理データベース』

234 ページの『主記憶データベース (MSDB)』

122 ページの『直接記憶方式』

リカバリー不能の全機能データベース

DBRC コマンドを使用して、RECON データ・セットで全機能データベースをリカバリー不能 として定義することができます。

全機能データベースがリカバリー不能として定義されている場合、IMS は、データベース内のデータが更新されるたびに、更新前の状態のデータのみをログに記録します。 IMS は、更新後の状態のデータをログに記録しません。そのため、リカバリー不能の全機能データベースに対する更新をバックアウトすることはできますが、データベースの以前のイメージ・コピーに更新を再適用してデータベースを回復することはできません。

リカバリー不能の全機能データベースのデータの「更新前」イメージは、タイプ X'50' ログ・レコードに記録されます。

DBRC コマンド INIT.DB または CHANGE.DB のいずれかで NONRECOV キーワードを使用して、データベースをリカバリー不能として定義することができます。

関連タスク:

- データベースのリカバリー可能化またはリカバリー不能化 (オペレーションおよびオートメーション)

関連資料:

- INIT.DB コマンド (コマンド)
- CHANGE.DB コマンド (コマンド)

HSAM データベース

階層順次アクセス方式 (HSAM) データベースは、データのアクセスに順次方式を使用します。 すべてのデータベース・レコードと各データベース・レコード内のすべてのセグメントは、ストレージの中で物理的隣接しています。

HSAM データベースをテープまたは直接アクセス記憶装置に保管することができます。それらは、オペレーティング・システム・アクセス方式として、基本順次アクセス方式 (BSAM) または待機順次アクセス方式 (QSAM) を用いて処理されます。アクセス方式は、PCB の中の PROCOPT= パラメーターに指定します。PROCOPT=GS を指定すると、QSAM が常に使用されます。 PROCOPT=G を指定すると、BSAM が使用されます。

HSAM データ・セットには、(ルート・セグメントのキーが存在する場合には) キーの昇順にルート・セグメントがロードされ、また階層順に従属セグメントがロードされます。ルート・セグメントにはキー・フィールドを定義する必要はありません。しかし、セグメントをロード・プログラムに与える順序は、セグメントをロードしようとしている順序でなければなりません。 HSAM データ・セットでは、固定長、非ブロック化レコード・フォーマット (RECFM=F) が使用されますが、これは論理レコード長が物理ブロック・サイズと等しいことを意味します。

HSAM データベースを更新するには、再度書き込みするしかありません。削除 (DLET) 呼び出しと置き換え (REPL) 呼び出しは許可されていないし、挿入 (ISRT) 呼び出しもデータベースがロードされているときのみに許されるに過ぎません。
HSAM データベースでは、フィールド・レベル・センシティビティーを使用できますが、以下のオプションはいずれも使用できません。

- 複数データ・セット・グループ
- 論理関係
- 副次索引
- 可変長セグメント
- セグメント編集/圧縮出口ルーチン
- データ・キャプチャー出口ルーチン
- 非同期データ・キャプチャー
- ロギング、リカバリー、または再編成

複数位置付けと複数 PCB は、HSAM データベースでは使用できません。

HSAM を使用する場合

HSAM は、順次処理のみを必要とするアプリケーションに使用されます。

HSAM は、その処理特性のために用途が限定されています。一般的には、HSAM は使用頻度の低いファイルに使用します。この種のファイルとしては、例えば、監査証跡や統計報告を収容しているファイルがあり、あるいは主データベースから除去されたヒストリー・データやアーカイブ・データを収容しているファイルがあります。

HSAM レコードの保管方法

HSAM データベースの中のセグメントは、このセグメントをロード・プログラムに与えた順序でロードされます。

ある 1 つのデータベース・レコードに属するすべてのセグメントを階層順に与えるべきです。ルート・セグメントのシーケンス・フィールドが定義されている場合には、ロード・プログラムに対して、ルート・キーの昇順にデータベース・レコードを与えるべきです。

以下の図は HSAM データベースの例を示しています。

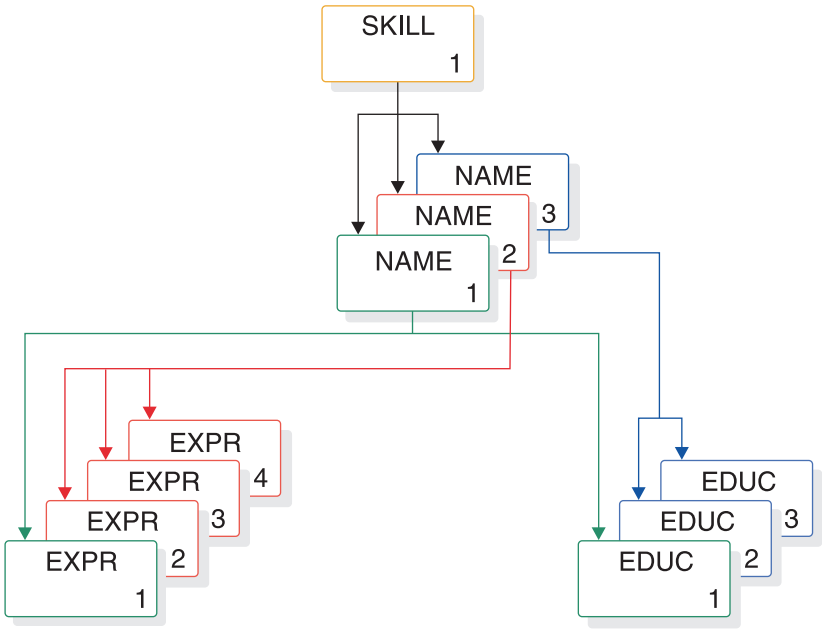


図 *19. HSAM* データベースの例

以下の図は、前の図に示す HSAM データベースの例がどのようにブロックに保管されるかを示しています。

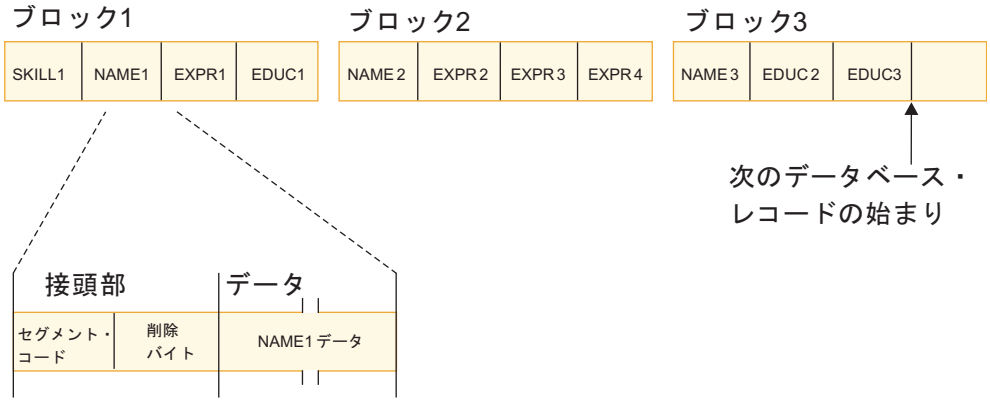


図 *20.* ブロックに保管された *HSAM* データベース

このデータ・セットでは、データベース・レコードは 1 つ以上の連続ブロックに保管されます。ユーザーが、ブロック・サイズの値を定義します。 各ブロックには、データベース・レコードのセグメントが保管されていき、これは次のセグメントを保管するのに十分なスペースが各ブロックに残っていない状態になるまで続きます。十分なスペースが存在しなくなると、各ブロックの残りのスペースにはゼロが埋め込まれ、次のセグメントは次の連続ブロックに保管されます。ある 1 つのデータベース・レコードの最後のセグメントがあるブロックに保管されると、そこに十分なスペースがあれば、その使用されていないスペースに次のデータベース・レコードのセグメントが保管されます。

ストレージにおいては、HSAM セグメントは、2 バイトの接頭部とこれに続くユーザー・データから成り立っています。 接頭部の先頭バイトは、IMS に対してセグ

メントのタイプ を示すセグメント・コードです。この番号は 1 から 255 までの範囲にあります。セグメント・コードは、IMS によってセグメントに昇順で割り当てられます。その順序はルート・セグメントから始まって階層順にすべての従属セグメントに続きます。接頭部の 2 番目のバイトは削除バイトです。 HSAM データベースを対象として DLET 呼び出しを使用することはできないので、このバイトは使用されません。

HSAM データベースを対象とする DL/I 呼び出し

HSAM データベースに最初に入るのは、GU 呼び出しまたは GN 呼び出しを介してです。 最初の呼び出しが出されると、このデータベースの初めから該当するセグメントの検索が始まり、該当セグメントに達するまで、このデータベースに保管されているすべてのセグメントを順次探していきます。

該当セグメントに達すると、その位置がこのデータベースを順方向に処理するこれ以後の呼び出しのための開始位置として使用されます。

HSAM データベースの中の位置が確立された後、GU 呼び出しがどのように処理されるかは、ルート・セグメントにシーケンス・フィールドが定義されているかどうか、およびどのような処理オプションが有効であるかによって異なります。以下の図は、シーケンス・フィールドが定義されているかどうか、またどのような処理オプションが有効かに基づいて、とられるアクションのフローチャートを示したものです。

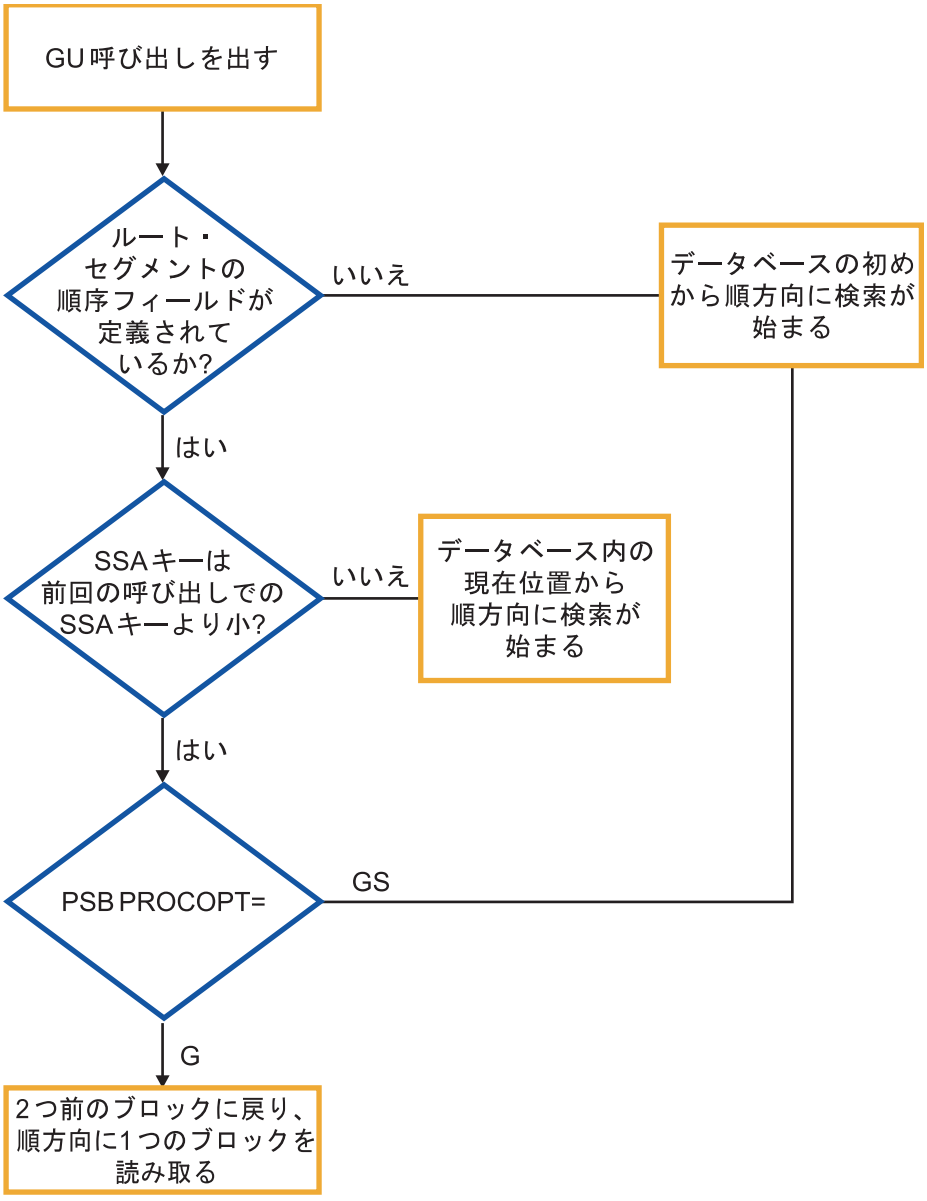


図 *21. HSAM* データベースを対象とする *GU* 呼び出し

定義されていないシーケンス・フィールド

シーケンス・フィールドが定義されていない場合には、GU 呼び出しが出されるたびに、現在の位置とはかかわりなくデータベースの初めから、目的のセグメントに対する検索が始まります。

このため、HSAM データベースを直接処理することができます。しかし、この処理は 1 つのボリュームの処理に限定されます。

定義されたシーケンス・フィールド

シーケンス・フィールドが定義されており、しかも GU 呼び出しがデータベース内の順方向にあるセグメントを検索する場合には、現在の位置から検索が開始され、これが目的のセグメントまで順方向に移動していきます。

目的のセグメントにアクセスするためにデータベース内で逆方向の移動を行わなければならない場合には、(PSBGEN の間に指定される) PROCOPT= パラメーター G または GS によって逆方向への移動の方法が決まります。 PROCOPT=GS と指定する (つまり、QSAM を用いてデータベースを読み取る) 場合には、目的のセグメントに対する検索はデータベースの初めから開始され、順方向に移動します。 PROCOPT=G と指定すると (つまり、BSAM を用いてデータベースを読み取ると)、検索はデータベース内で逆方向に移動します。この検索では、直前に読み取ったブロックとその前のブロックに戻ってから、前のブロックを順方向に読み取って、必要なセグメントを探します。

HSAM データベースにおけるセグメントへのアクセス方式のために、ルート・セグメントに対しては順次にアクセスし、データベース・レコード内では階層順に従属セグメントにアクセスするのが最も実用的です。他のアクセス方式では、後退したり、テープを巻き戻したり、あるいはデータ・セットを初めからスキャンしたりする必要が生じるので、大変時間がかかることがあります。

前述のように、HSAM データベースを対象として DLET 呼び出しと REPL 呼び出しを出すことはできません。 また、ISRT 呼び出しは、データベースをロードしているときにのみ使えます。 HSAM データベースを更新するためには、現在の HSAM データベースと更新データをマージするためのプログラムを書かなければなりません。 更新データは 1 つ以上のファイルに保管することができます。この過程で作成された出力データ・セットが、更新された新しい HSAM データベースになります。以下の図はこの処理を図示したものです。

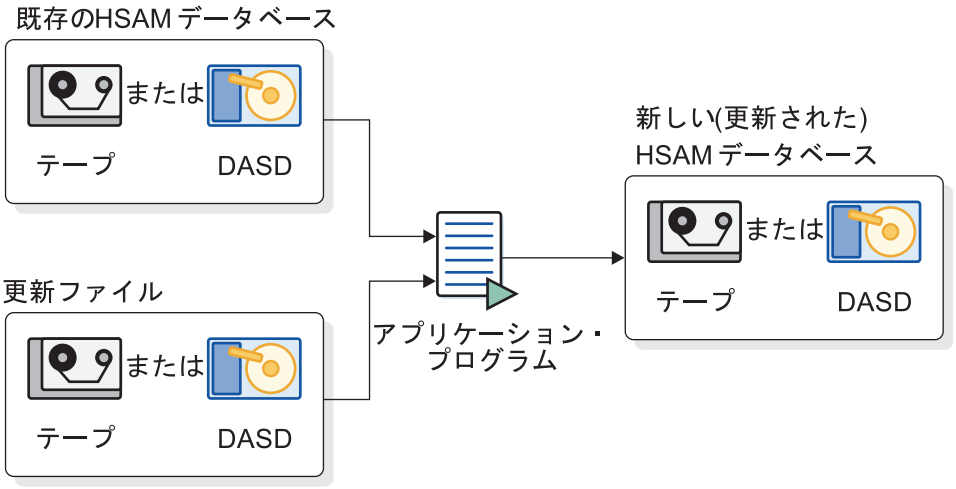


図 22. *HSAM* データベースの更新

HISAM データベース

階層索引順次アクセス方式 (HISAM) データベースでは、 HSAM データベースと同様に、各データベース・レコードに属するセグメントの間の関係は、ストレージ内での物理的な隣接関係によって決まります。

しかし、HSAM とは異なり、各 HISAM データベース・レコードについての索引が作られるので、データベース・レコードへの直接アクセスを行うことができます。 HISAM データベースを定義するにあたっては、各ルート・セグメントに、固有のシーケンス・フィールドを 1 つ定義しなくてはなりません。 このシーケンス・フィールドは、データベース内のルート・セグメント (したがって、データベース・レコード) を対象とする索引を作成するのに使用されます。

HISAM データベースは直接アクセス装置に保管されます。 このデータベースは、仮想記憶アクセス方式 (VSAM) ユーティリティーを用いて処理することができます。 HSAM とは異なり、すべての DL/I 呼び出しは、HISAM データベースを対象として出すことができます。さらに、HISAM データベースを対象として以下のオプションが使用可能です。

- 論理関係
- 副次索引
- 可変長セグメント
- セグメント編集/圧縮出口ルーチン
- データ・キャプチャー出口ルーチン
- フィールド・レベル・センシティビティー
- ロギング、リカバリー、および再編成

HISAM 選択の基準

ルート・セグメントに対して順次アクセスまたは直接アクセスが必要であり、しかもデータベース・レコードの中の従属セグメントの順次処理が必要である場合には、HISAM を使用すべきです。

データベースが、以下の特性をすべてまたはほとんど備えている場合、データ編成として HISAM を使用するのはよい選択であるといえます。

- 各ルート・セグメントの従属セグメントが少ない。

  ルート・セグメントへのアクセスは索引が付いているので高速です。従属セグメントへのアクセスは順次に行われるので、比較的遅くなります。
- データベースに対する削除操作が少ない。

  ルート・セグメントを削除する場合を除き、削除操作はすべてデータベースを再編成するまでは使用不可のスペースを作成します。
- アプリケーションは、狭いキー範囲内に挿入される少量のルート・セグメントに依存している (VSAM)。

  最初のロードの後に挿入されるルート・セグメントは、KSDS の中の該当する CI にルート・キー・シーケンスで挿入されます。多くのルート・セグメントの

キーが狭いキー範囲の中にあれば、多数の CI 分割が起きることになります。これによってパフォーマンスが低下することがあります。

- ほとんどのデータベース・レコードのサイズがほぼ同じである。

  このような特性があれば、論理レコード長と CI サイズを適宜選択することにより、大部分のデータベース・レコードを 1 次データ・セットに収めることができます。データベース・レコードのほとんどが 1 次データ・セットに収まるようにしたいのは、データベース・レコードのうち、オーバーフロー・データ・セットに保管されている部分にアクセスするにはさらに読み取り操作とシーク操作が必要になるからです。余分の読み取りとシークによって、パフォーマンスは低下します。ただし、データベース・レコードに対して行われる処理の大部分が、1 次データ・セットの中のセグメントを対象とするものである場合 (言い換えれば、よく使用されるセグメントが 1 次データ・セットに保管されている場合) には、この考慮事項はそれほど重要でないかもしれません。

  ほとんどのデータベース・レコードのサイズが同じであれば、スペースも節約できます。各データベース・レコードは、論理レコードの先頭から始まります。論理レコードの中のデータベース・レコードによって使用されないスペースはすべて使用不可のスペースです。このことは、1 次データ・セットとオーバーフロー・データ・セットの両方の論理レコードにあてはまります。データベース・レコードのサイズに大きなばらつきがある場合には、多くの論理レコードの末尾に未使用スペースの大きなすき間ができることになります。

HISAM レコードの保管方法

HISAM データベース・レコードは、1 次データ・セットとオーバーフロー・データ・セットの 2 つのデータ・セットに保管されます。

1 次データ・セット には、1 つの論理レコードに収容できる、データベース・レコード内のすべてのセグメントと 1 つの索引が保管されます。この索引を通じて、ルート・セグメントへの (したがって、データベース・レコードへの) 直接アクセスが行えます。*オーバーフロー・データ・セット* には、1 次データ・セットに収容できない、データベース・レコード内のセグメントがすべて保管されます。キー順データ・セット (KSDS) は 1 次データ・セットであり、入力順データ・セット (ESDS) はオーバーフロー・データ・セットです。

HISAM データベース・レコードの保管に関して、次のことを知っている必要があります。

- ユーザーが、1 次データ・セットとオーバーフロー・データ・セットの双方の論理レコード長を定義します (これはこのトピックにリストされている規則に従います)。データ・セットごとに論理レコード長が異なっていてもかまいません。このため、1 次データ・セットにおいては、「平均的」なデータベース・レコード、あるいはデータベース・レコード内で最も頻繁にアクセスされるセグメントを保管するのに十分な大きさの論理レコード長を定義することができます。この後、オーバーフロー・データ・セットの論理レコード長を、(ある種の制約のもとに) データベース・レコードの特性に照らして最も効率のよい長さとして定義することができます。

- 論理レコードは、いくつかの制御インターバル (CI) に分類されます。制御インターバルとは、入出力装置と記憶装置の間で転送されるデータの単位です。ユーザーが CI のサイズを定義します。
- 各データベース・レコードは、1 次データ・セットの中の 1 つの論理レコードの先頭から始まります。 1 つのデータベース・レコードは、1 次データ・セットにおいては 1 つの論理レコードしか占めることができませんが、データベース・レコードのオーバーフロー・セグメントはオーバーフロー・データ・セット内の複数の論理レコードを占めることができます。
- ある 1 つのデータベース・レコードに属するセグメントを分割し、これを 2 つの論理レコードにまたがって保管することはできません。 この理由で、そしてさらに、各データベース・レコードが新しい論理レコードから始まるという理由から、多くの論理レコードの末尾には使用されないスペースが残ります。最初にデータベースがロードされるときに、IMS は最後のルート・セグメントとして、すべての X'FF' をキーとして持つルート・セグメントをデータベースに挿入します。

以下の図は、4 つの HISAM データベース・レコードを示しています。

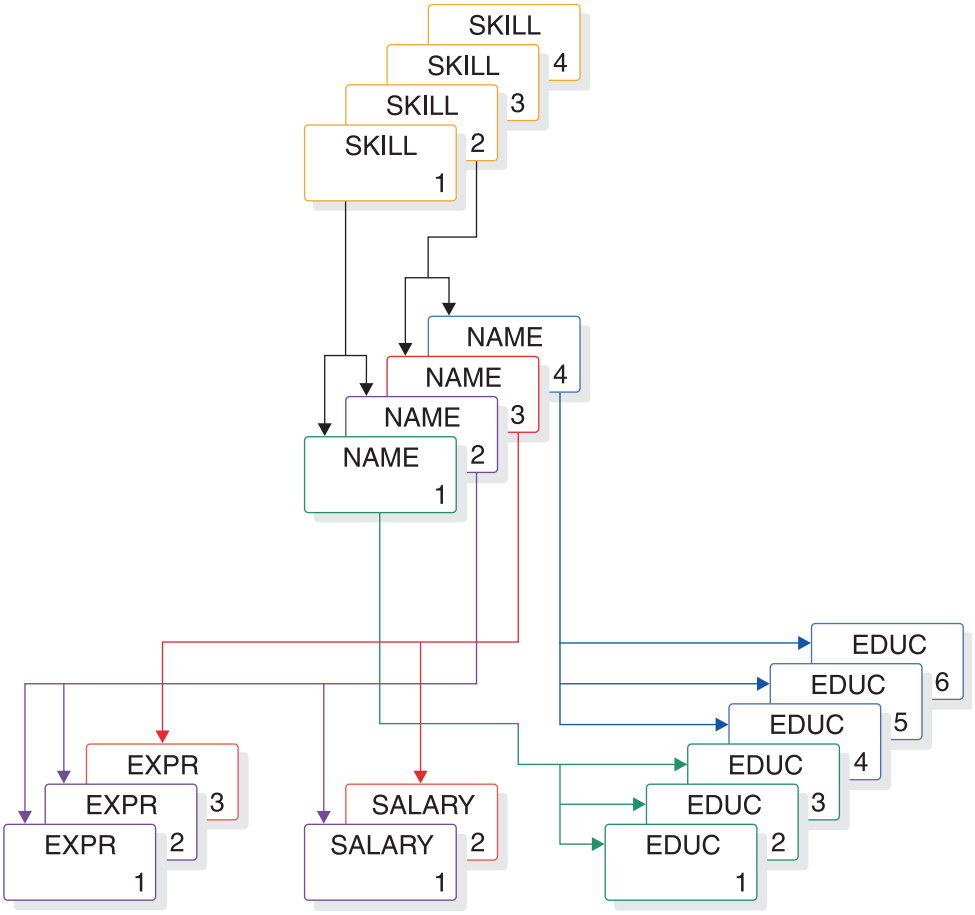


図 23. *HISAM* データベース・レコードの例

以下の図は、前の図の 4 つのレコードが、1 次データ・セットとオーバーフロー・データ・セットに最初に保管されたときの様子を示したものです。ストレージにおいては、HISAM セグメントは、2 バイトの接頭部とこれに続くユーザー・データ

から成り立っています。 接頭部の先頭バイトは、IMS に対してセグメントのタイプ を示すセグメント・コードです。この番号は 1 から 255 までの範囲にあります。セグメント・コードは、IMS によってセグメントに昇順で割り当てられます。その順序はルート・セグメントから始まって階層順にすべての従属セグメントに続きます。接頭部の 2 番目のバイトは削除バイトです。

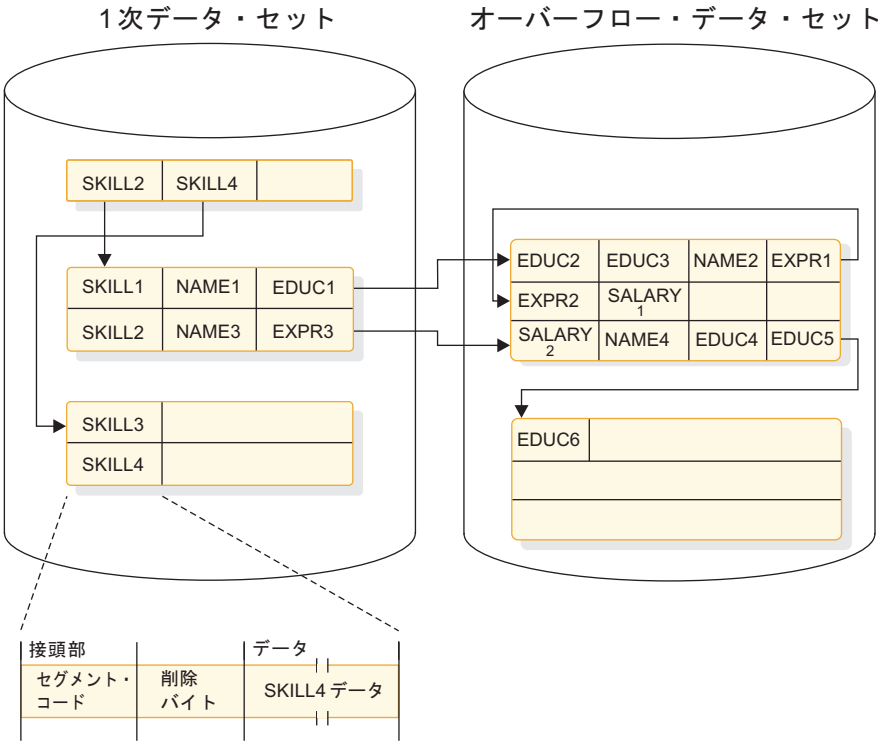


図 24. ストレージの中の *HISAM* データベース・レコードの例

1 次データ・セット内の論理レコードには、ルート・セグメントとルート・セグメントの従属セグメントが (階層順に) スペースの許す限り入ります。データベース・レコードの残りのセグメントは、オーバーフロー・データ・セットに (ここでも階層順に) 入ります。データベース・レコードのこの 2 つの「部分」は、直接アドレス・ポインターによって相互につなぎ合わされます。 前の図の 1 番目と 2 番目のデータベース・レコードのように、あるデータベース・レコードのオーバーフロー・セグメントがオーバーフロー・データ・セットで複数の論理レコードを使用する場合は、これらの論理レコードもまた直接アドレス・ポインターによって相互につなぎ合わされます。この図で注目すべき点は、HISAM 索引にはデータベース内の各ルート・セグメントを指すポインターが含まれていないということです。そうではなしに、この索引は、各ブロックまたは CI の中の一番上のルート・キーを指しています。

以下の図は、HISAM データベースにおける論理レコードの構造を示したものです。

- 論理レコードにおいては、最初の 4 バイトはこのデータベース・レコードに属する次の論理レコードへの直接アドレス・ポインターです。このポインターによって、ある 1 つのデータベース・レコードに属するすべての論理レコードが正しい順序に維持されます。データベース・レコードの最後の論理レコードのこのフィールドにはゼロが含まれているという点に注意してください。

- このポインターの後ろには、このデータベース・レコードの 1 つ以上のセグメントが階層順に続きます。
- このセグメントの後ろには、0 という値をもつ 1 バイトのセグメント・コードがあります。これは、論理レコードの最後のセグメントに達したことを示しています。

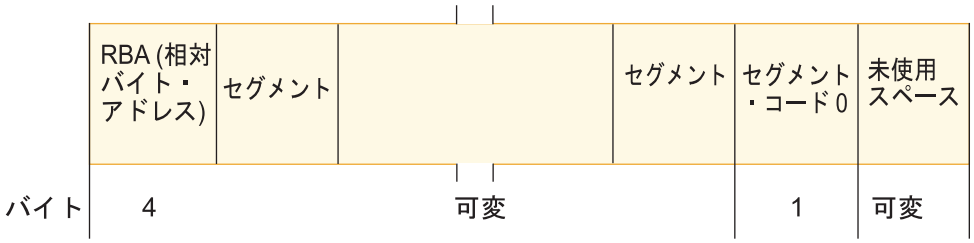


図 *25. HISAM* データベースにおける論理レコードのフォーマット

セグメントへのアクセス

アプリケーション・プログラムは、HISAM データベース内のセグメントにアクセスする際に、一連の検索シーケンスに従います。

HISAM においては、アプリケーション・プログラムからルート・セグメントのキーで修飾されたセグメント検索引数 (SSA) を用いて呼び出しが出されると、次のようにしてこのセグメントが検出されます。

1. 索引を検索して、指定されているルート・キーより大きいか等しい値を持つ最初のポインターを探します (索引は各 CI の中で一番上のルート・セグメント・キーを指しています)。
2. この索引ポインターに従って、該当する CI に到達します。
3. この CI の中で、該当する論理レコードを探します (指定されているルート・セグメント・キーの値が、この CI の中の各ルート・セグメント・キーと比較されます)。
4. 該当する論理レコード (したがって、データベース・レコード) が見つかると、この中をすべて順次検索していき、指定されたセグメントを検出します。

アプリケーション・プログラムが GU 呼び出しを出すときに、あるルート・セグメントに対して修飾されていない SSA を用いる場合、あるいはルート・キー以外で修飾された SSA を用いる場合には、HISAM 索引を使用することはできません。このセグメントの検索は、データベースの先端から始まり、指定されているセグメントが検出されるまで順次続けられます。

VSAM を用いたルート・セグメントの挿入

最初のロードの後に、HISAM データベースに挿入されたルート・セグメントは、キーの昇順に従って 1 次データ・セットに保管されます。

CI には、新しいルート・セグメントを挿入できる空き論理レコードが存在する場合も存在しない場合もあります。両方の状態について、次に説明します。

空き論理レコードが存在する場合

この例は、空き論理レコードが存在する場合の挿入方法を示したものです。

以下の図では、新しいルート・セグメントは、ルート・キーの順序に従って CI に挿入されます。この CI の中に、これより大きいキーを持つルート・セグメントを含む論理レコードが存在する場合は、この論理レコードが「押し下げられて」新しい論理レコードのためのスペースが作られます。

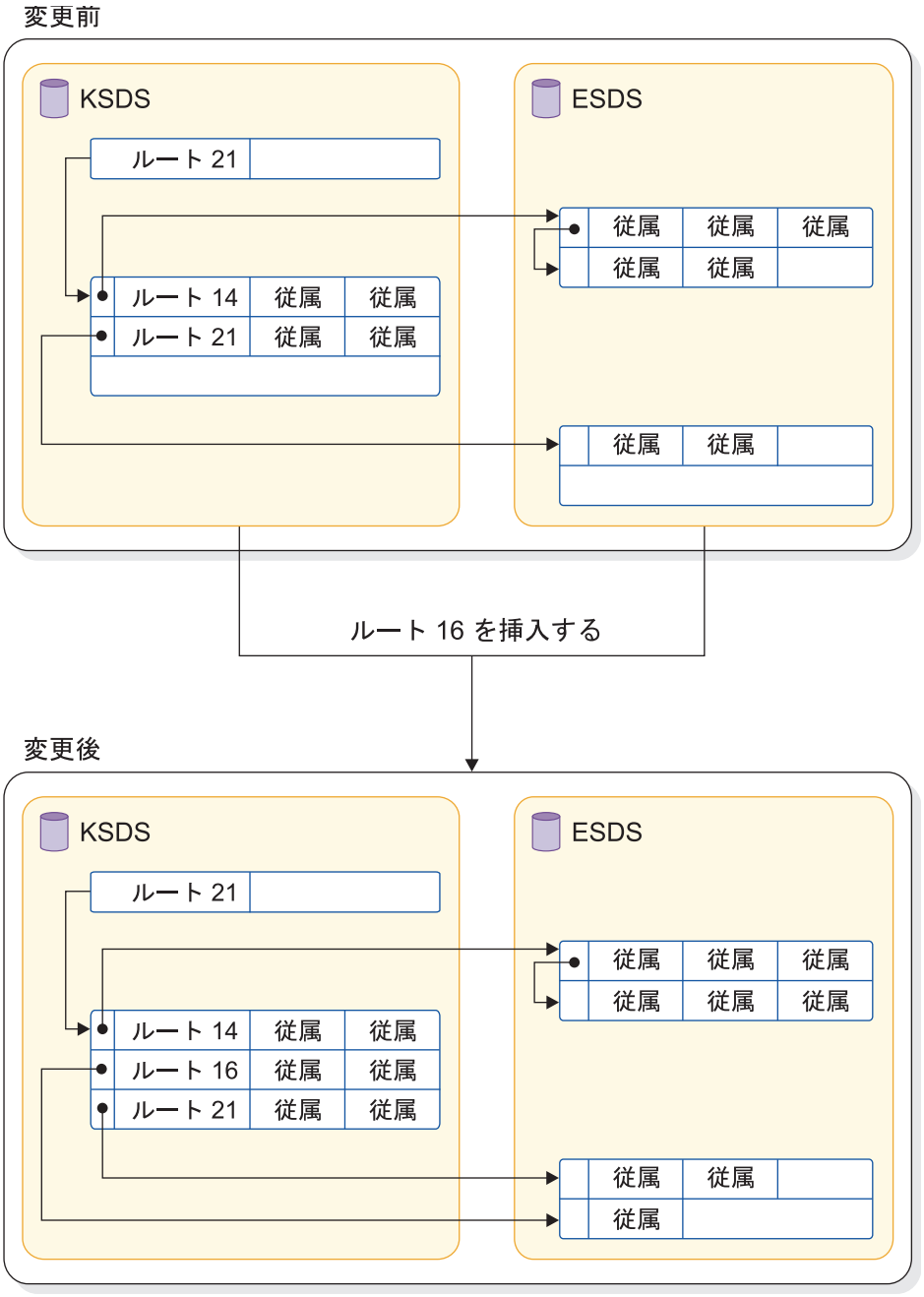


図 *26. HISAM* データベースへのルート・セグメントの挿入 *(CI* の中に空き論理レコードが存在する場合*)*

空き論理レコードが存在しない場合

この例は、空き論理レコードが CI の中に存在しない場合の挿入方法を示したものです。

この場合、CI は分割されて新しい CI が 2 つ作成されます。この 2 つの CI サイズはともに元のサイズと同じです。 CI が分割される場所は、DFSVSAMP データ・セット (バッチ環境の場合) または DFSVSMxx PROCLIB メンバー (オンライン環境の場合) で OPTIONS ステートメントの INSERT= パラメーターに指定した内容によって決まります。

分割が起こり得るポイントは、ルート・セグメントが挿入される場所、または CI の中央です。 CI が分割されたあとは、それぞれの新しい CI の中に空き論理レコードができますから、新しいルート・セグメントはルート・キー・シーケンスに従って、しかるべき CI の中に挿入されます。 140 ページの『空き論理レコードが存在する場合』に示す図のように、新しい CI の中の論理レコードがより上のキーを持つルート・セグメントを収容している場合には、この論理レコードが「押し下げられて」新しい論理レコードのためのスペースが作られます。

HISAM データベースに新しいルート・セグメントを追加する場合には、ルート・セグメントをキーの昇順に追加すると、パフォーマンスを少し向上することができます。

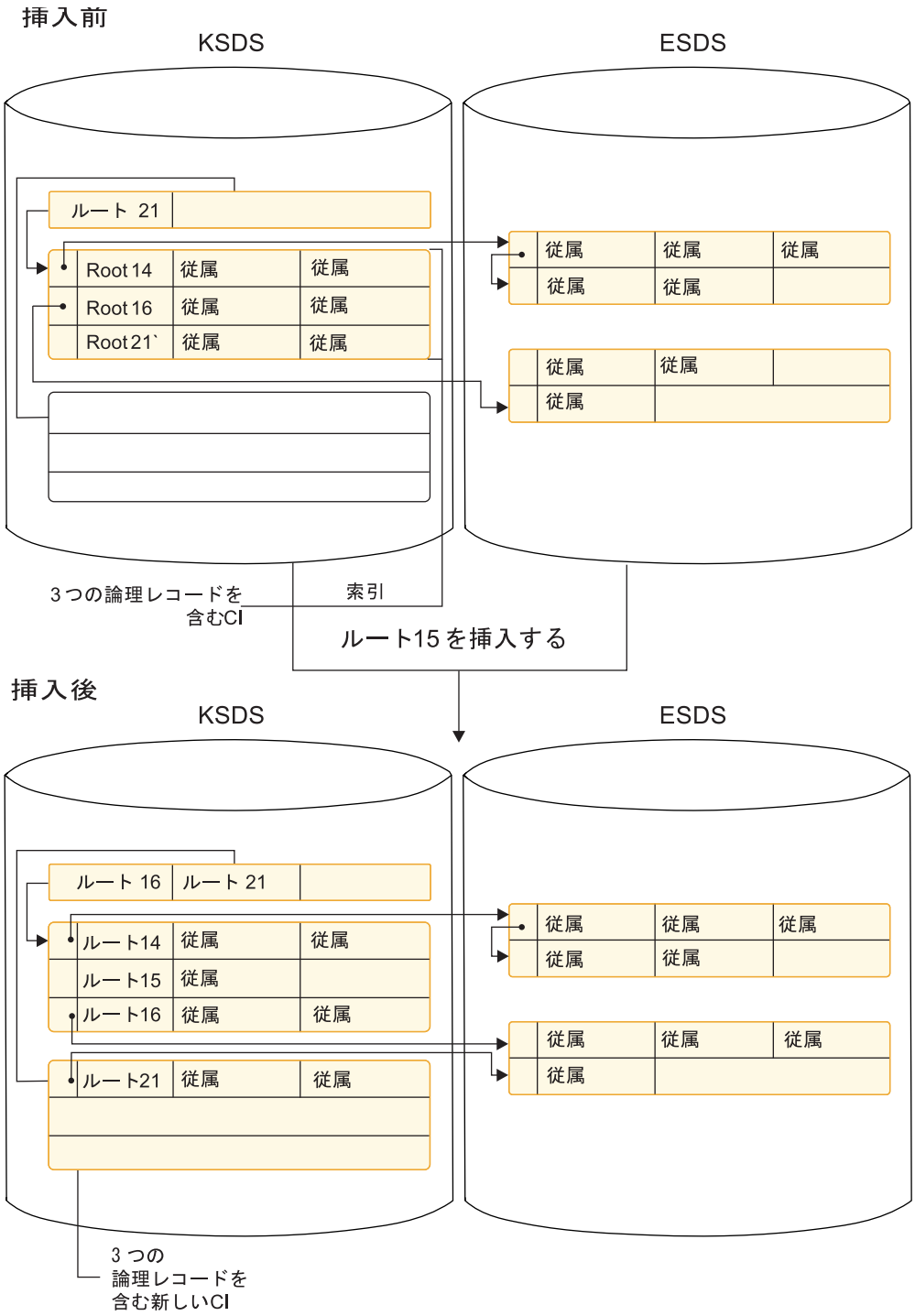


図 27. *HISAM* データベースへのルート・セグメントの挿入 *(CI* の中に空き論理レコードが存在しない場合)

関連資料:

- IMS PROCLIB データ・セットの DFSVSMxx メンバー (システム定義)
- IMS プロシージャーの DD ステートメント (システム定義)

従属セグメントの挿入

最初のロードの後に HISAM データベースに挿入される従属セグメントは、階層順に挿入されます。該当する論理レコードの中のどこに新しい従属セグメントを挿入するかは IMS が決定します。

2 つの状況が考えられます。この論理レコードの中に新しい従属セグメントのためのスペースが十分に存在する場合とそうでない場合です。

以下の図は、論理レコードの中にスペースが十分に存在する場合のセグメントの挿入方法を示したものです。新しい従属セグメントは論理レコードの中のこのセグメントの所定の階層位置に挿入されます。その際、階層の順序でこのセグメントよりも後ろにあるセグメントは、論理レコードの中で右の方に移されます。

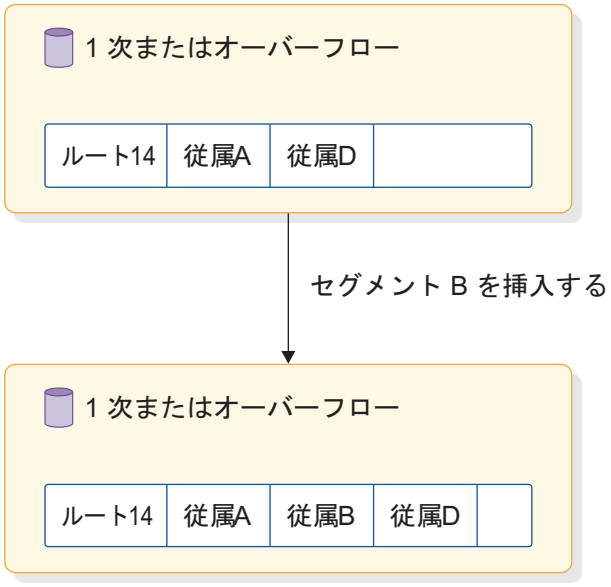


図 *28. HISAM* データベースへの従属セグメントの挿入 (論理レコードの中にスペースが存在する場合)

以下の図は、論理レコードの中にスペースが十分に存在しない場合のセグメントの挿入方法を示したものです。先の例と同様、新しい従属セグメントは常に論理レコードの中のそれぞれの所定の階層順のところに保管されます。しかし、新しいセグメントより右にあるすべてのセグメントは、オーバーフロー・データ・セットの中の空いている最初の論理レコードに移されます。

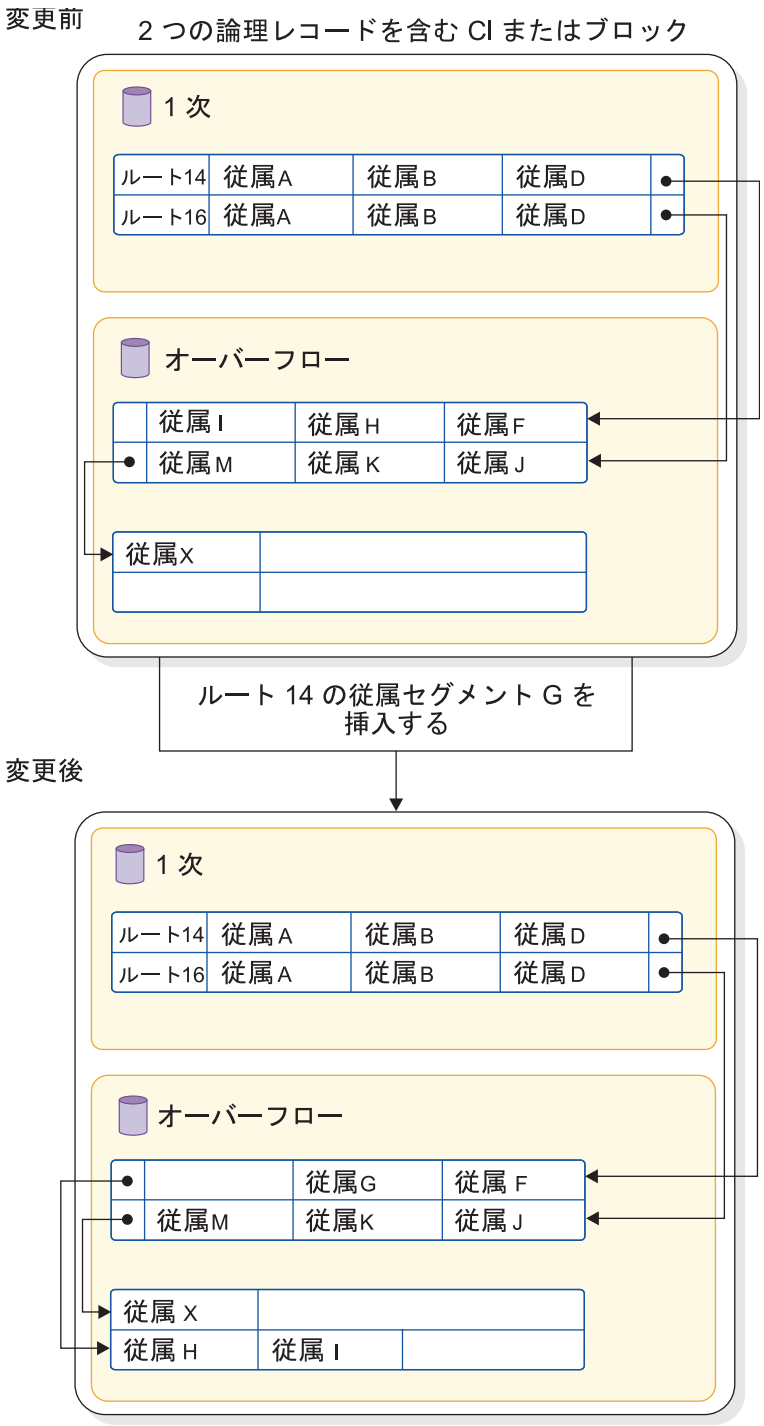


図 *29. HISAM* データベースへの従属セグメントの挿入 (論理レコードの中にスペースが存在しない場合)

セグメントの削除

セグメントを HISAM データベースから削除すると、このセグメントの接頭部の削除バイトの中に削除の印が付けられます。このセグメントが、このデータベースか

ら物理的に取り除かれるわけではありません。これには 1 つの例外があり、後にこのトピックで、この例外については説明します。

削除されたセグメントの従属セグメントには削除の印が付けられていないが、その親に印が付けられているため、この従属セグメントにアクセスすることはできません。このデータベースが再編成されるとき、削除の印が付いていないこのような従属セグメントは (もちろん削除の印が付けられているセグメントも) 削除されます。

ここで注意すべきことが 1 つあります。それは、あるデータベース・レコードにおいて、階層の順序から見て、削除されたセグメントより後ろにあるセグメントにアクセスする場合、削除されたセグメントは依然として「検索」する必要があるということです。この概念を、以下の図に示します。

セグメント B2 がこのデータベース・レコードから削除されています。これは、セグメント B2 とその従属セグメント (C1、C2、C3) は依然このデータベースの中に存在しますが、もはやこれらのセグメントにアクセスできないことを意味します。

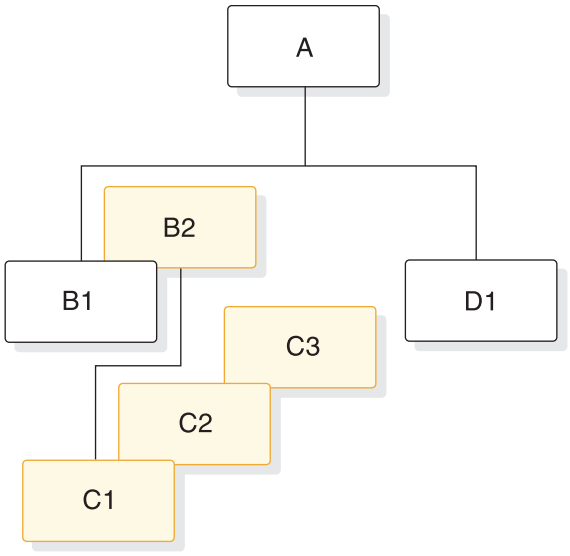


図 *30.* データベース上の階層セグメントのレイアウト

セグメント D1 へのアクセス要求が出されます。セグメント B2、C1、C2、および C3 にはアクセスすることはできませんが、これらは依然としてデータベースの中に存在します。したがって、これらのセグメントはアクセス不能ですが、「1 つ 1 つ検索する」必要があります (以下の図を参照)。

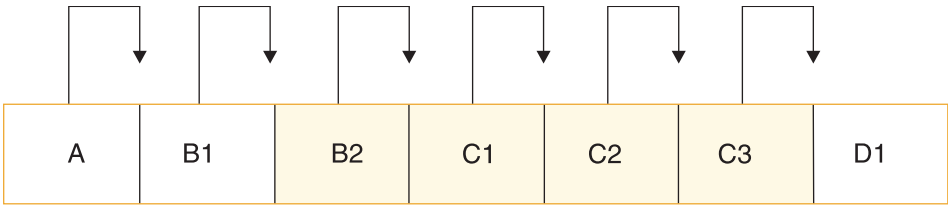


図 *31.* 階層上、削除されたセグメントより後ろにある *HISAM* セグメントのアクセス

削除されたセグメントが物理的にもデータベースから除去される場合が 1 つあります。削除されたセグメントがルート・セグメントである場合、このルート・セグメントとその従属セグメントのいずれも論理関係に関与していないならば、このルート・セグメントが入っている論理レコードが消去されます。デフォルトは ERASE=YES であり、「バッファー変更済みマーク」は付けられません。したがって、PROCOPT=G 読み取りジョブは、別のジョブが削除バイトを設定した後でロックを待つ必要はなく、「セグメントが検出されない」という条件を戻します。他の DB タイプとの整合性を保つためには、ERASE=NO を使用して、読み取りを試行する前に物理的削除を待つようにしてください。 ERASE パラメーターは、DFSVSMxx PROCLIB メンバーの DBD ステートメントで指定します。

この論理レコードが取り除かれた後、そのスペースは再使用が可能です。ただし、このルート・セグメントの従属セグメントを収容しているオーバーフロー論理レコードは再使用に利用できないので注意してください。この特別の場合を除き、削除されたセグメントによって占められていたスペースを再取得するために、HISAM データベースをアンロードし、再ロードする必要があります。

関連概念:

190 ページの『プログラムを分離するロッキング』

セグメントの置き換え

固定長セグメントを使用している限り、HISAM データベースでのセグメントの置き換えは直接的に行われます。つまり、セグメント内のデータは、いったん変更されると、ストレージの中のその元の場所に返されます。セグメント内のキー・フィールドは変更できません。

可変長セグメントが使用される場合は、考慮すべき影響がほかに存在します。

関連概念:

423 ページの『可変長セグメント』

SHSAM、SHISAM、および GSAM データベース

非データベース・システムを IMS に変換するとき、またはアプリケーション・プログラム間でデータを渡すときは、一般に単純階層順次アクセス方式 (SHSAM)、単純階層索引順次アクセス方式 (SHISAM)、および汎用順次アクセス方式 (GSAM) の各データベースを使用します。

非データベース・システムから IMS へ変換する場合は、SHSAM、SHISAM、および GSAM データベースを用いると、z/OS アクセス方式を使用している既存のプログラムは、IMS への変換の間でも依然として使用可能です。これが可能なのは、これらのデータベースのデータ・フォーマットが z/OS データ・セットのデータ・フォーマットと同じためです。

データベース (または非データベース) アプリケーション・プログラムがデータベース (または非データベース) アプリケーション・プログラムへデータを渡す場合、そのアプリケーション・プログラムは、まず渡そうとするデータを SHSAM、SHISAM、または GSAM データベースの中に置くことができます。それが済めば、データベース (または非データベース) アプリケーション・プログラムは、これらのデータベースからそのデータにアクセスします。

IMS データ・セットと z/OS データ・セットの双方にアクセスする必要のあるアプリケーション・プログラムがある場合には、SHSAM、SHISAM、または GSAM を選択することができます。このうちどれを使用するかは、必要とされる機能に応じて決まります。以下の表で、この 3 種類のデータベースのタイプの特性と使用可能な機能を比較します。

表 *47. SHSAM*、*SHISAM*、および *GSAM* データベースの比較

特性と機能	**SHSAM**	**SHISAM**	**GSAM**
階層構造の使用	NO	NO	NO
セグメント接頭部の使用	NO	NO	NO
可変長レコードのサポート	NO	NO	YES
チェックポイント/再始動のサポート	NO	YES[1]	YES[1]
非 IMS データ・セットと互換性があるか	YES	YES	YES
VSAM をオペレーティング・システム・アクセス方式としてサポートできるか	NO	YES	YES
BSAM をオペレーティング・システム・アクセス方式としてサポートできるか	YES	NO	YES
バッチ領域からアクセス可能か	YES	YES	YES
バッチ・メッセージ処理領域からアクセス可能か	YES	YES	YES
メッセージ処理領域からアクセス可能か	YES	YES	NO
ロギングのサポート	NO	YES	NO
GET 呼び出しのサポート	YES	YES	YES
ISRT 呼び出しのサポート	YES[2]	YES	YES[3]
CICS-DBCTL のサポート	YES	YES	NO
DCCTL のサポート	NO	NO	YES

注**:**

1. シンボリック・チェックポイントを使用する
2. データベースのロードのみ
3. データ・セットの終わりでのみ可能

関連概念:

17 ページの『セグメント』

SHSAM データベース

単純 HSAM (SHSAM) データベースは、ただ 1 種類のセグメント、すなわちルート・セグメントが入っている HSAM データベースです。セグメントには接頭部はありません。それは、セグメント・コード (セグメント・タイプは 1 つだけ) も削除バイト (削除はできない) も必要ではないためです。

SHSAM セグメントにはユーザー・データしか入っていない (IMS 接頭部はない) ので、SHSAM データベースには z/OS BSAM と QSAM によってアクセスすることができます。ISRT、 DLET、および REPL 呼び出しは更新には使用できません。しかし、ISRT は SHSAM データベースをロードするのには使用できます。SHSAM データベースを処理するのに有効な呼び出しは GET 呼び出しだけです。これを使えば、データベースからセグメントの取り出しだけを行うことができま

す。 SHSAM データベースを更新するには、このデータベースを再ロードしなくてはなりません。 通常の SHSAM を使用する状況については、このトピックの紹介の部分で説明しています。 SHSAM を使用する前に、GSAM データベースに関するトピックを読んでおいてください。GSAM は SHSAM と同じ機能の多くを備えているためです。しかし、SHSAM とは異なり、メッセージ処理領域から GSAM ファイルにアクセスすることはできません。ただし、GSAM ではチェックポイントをとり、再始動することができます。

SHSAM データベースは、フィールド・レベル・センシティビティー・オプションを使用できますが、以下のオプションは使用できません。

- 論理関係
- 副次索引
- 複数データ・セット・グループ
- 可変長セグメント
- セグメント編集/圧縮出口ルーチン
- データ・キャプチャー出口ルーチン
- ロギング、リカバリー、または再編成

SHISAM データベース

単純 HISAM (SHISAM) データベースは、ただ 1 種類のセグメント、すなわちルート・セグメントだけが入っている HISAM データベースです。

このセグメントには接頭部はありません。それは、セグメント・コード (1 つのセグメント・タイプしかありません) も削除バイト (削除は VSAM 消去操作を使用します) も必要でないという理由からです。 SHISAM データベースは VSAM を介してアクセスされる KSDS でなければなりません。 SHISAM セグメントにはユーザー・データしか含まれていない (IMS 接頭部はない) ため、DL/I 呼び出しだけでなく、VSAM マクロを使用してアクセスすることができます。すべての DL/I 呼び出しは、SHISAM データベースに対して出すことができます。

SHISAM IMS シンボリック・チェックポイント呼び出し

SHISAM は、z/OS データ・セットにアクセスして IMS シンボリック・チェックポイント呼び出しを使用するアプリケーション・プログラムが必要である場合にも役立ちます。

IMS シンボリック・チェックポイント呼び出しを使用すると、z/OS 基本チェックポイント呼び出しを使用するよりも再始動が容易です。アプリケーション・プログラムで使用している z/OS データ・セットを SHISAM データベースの 1 つのデータ・セットに変換する場合、シンボリック・チェックポイント呼び出しを使用することができます。これによって、アプリケーション・プログラムは処理中にチェックポイントをとることができ、のちに 1 つのチェックポイントからアプリケーション・プログラムを再始動することができます。主な利点は、システム障害が起きた場合、アプリケーション・プログラムはそれまでに実行されたすべての処理結果を失うのではなく、あるチェックポイントからリカバリーすることができるというところにあります。これには 1 つの例外があります。VSAM をオペレーティング・システムのアクセス方式として使用するデータベースを最初にロードするアプリケーション・プログラムは、チェックポイントから再始動することはできませ

ん。 GSAM データベースを使用しているアプリケーション・プログラムも、シンボリック・チェックポイント呼び出しを出すことができます。 SHSAM データベースを使用しているアプリケーション・プログラムは、この呼び出しを出すことはできません。

SHISAM を使用するかどうか判断する前に、GSAM データベースに関する次のトピックを読んでおく必要があります。 GSAM は、SHISAM と同じ機能を多く備えています。しかし、SHISAM とは異なり、GSAM ファイルは、メッセージ処理領域からアクセスすることはできません。

SHISAM データベースは、フィールド・レベル・センシティビティーとデータ・キャプチャー出口ルーチンを使用することができます。ただし、以下のオプションは使用できません。

- 論理関係
- 副次索引
- 複数データ・セット・グループ
- 可変長セグメント
- セグメント編集/圧縮出口ルーチン

GSAM データベース

GSAM データベースは、z/OS データ・セットとの互換性を持つように設計された順次編成のデータベースです。

GSAM データベースには、階層、データベース・レコード、セグメント、キーがありません。 GSAM データベースは、以前に作成されたデータ・セットまたは後で z/OS アクセス方式の VSAM または QSAM/BSAM でアクセスされるデータ・セット上に保管することができます。 GSAM データ・セットは、VSAM の使用時は固定長レコードまたは可変長レコードを使用し、QSAM/BSAM の使用時は固定長レコード、可変長レコード、または不定長レコードを使用することができます。

VSAM を使用して GSAM データベースを処理する場合、VSAM データ・セットが入力順であり、DASD 上に保管されていなければなりません。 QSAM/BSAM を使用する場合には、物理順次 (DSORG=PS) データ・セットは DASD 装置またはテープ装置のいずれの上にあってもかまいません。 BSAM を使用する場合は、DD ステートメントで DSNTYPE=LARGE を指定することにより、GSAM データ・セットを z/OS のラージ・フォーマット・データ・セットと定義できます。

GSAM は、VSAM および BSAM の両方について、DFSMS ストライプ拡張フォーマット・データ・セットをサポートします。

GSAM データベース・データ・セットは、拡張アドレス・ボリューム (EAV) の拡張アドレス方式スペース (EAS) に割り振ることができます。

制約事項: GSAM データベースは、CICS アプリケーションでは使用できません。

GSAM データベースは DCCTL 環境でサポートされているため、BMP プログラムを使用して非 IMS 順次データ・セットを処理する必要がある場合は、GSAM データベースを使用することができます。

GSAM データベースは、ロード・プログラムにレコードを与えた順番でロードされます。 GSAM データベースに対して DLET と REPL の呼び出しを出すことはできません。ただし、このデータベースがロードされた後に ISRT 呼び出しを出すことはできますが、このデータ・セットの終わりにレコードを追加するためだけです。レコードは、GSAM データ・セットにランダムには追加されません。

GSAM および SHSAM データベースのランダム処理は可能ですが、GSAM データベースのランダム処理はレコード検索引数 (RSA) によって修飾された GU 呼び出しを使用します。この処理は、主として、一連の GN 呼び出しを出す前にこのデータベースにおける位置を設定するのに役立ちます。

SHSAM および SHISAM データベースは任意の処理領域で処理することができますが、GSAM データベースは、バッチ領域あるいはバッチ・メッセージ処理領域においてだけ処理することができます。

次の IMS オプションはどれも GSAM データベースには該当しません。

- 論理関係
- 副次索引
- セグメント編集/圧縮出口ルーチン
- フィールド・レベル・センシティビティー
- データ・キャプチャー出口ルーチン
- ロギングまたは再編成
- 複数データ・セット・グループ

GSAM データ・セットとアクセス方式の詳細については、GSAM によるストライプ拡張フォーマット・データ・セットの使用に関する情報も含め、「*IMS V13* アプリケーション・プログラミング」で『GSAM データベースの処理』を参照してください。

z/OS データ・セットの詳細については、「*z/OS DFSMS: Using Data Sets*」のほか、 z/OS DFSMShsm、DFSMSdss、および DFSMSdfp のストレージ管理ガイドおよびリファレンスを参照してください。

関連概念:

123 ページの『DCCTL でサポートされるデータベース』

GSAM IMS シンボリック・チェックポイント呼び出し

GSAM は、他の用途に加えて、z/OS データ・セットにアクセスして IMS シンボリック・チェックポイント呼び出しを使用するアプリケーション・プログラムが必要である場合にも役立ちます。

IMS シンボリック・チェックポイント呼び出しを使用すると、z/OS 基本チェックポイント呼び出しを使用するよりも再始動が容易です。この IMS シンボリック・チェックポイント呼び出しによってアプリケーション・プログラムは処理中にチェックポイントをとることができ、これによって、チェックポイントからプログラムを再始動させることができます。チェックポイント呼び出しは、すべての GSAM バッファーに挿入レコードを短ブロックとして書き込みます。チェックポイントをとることの主な利点は、システムに障害が起こった場合、アプリケーション・プログラムがチェックポイントからリカバリー処理を行うため、処理済みのデータが失

われることがない点です。しかし、オペレーティング・システム・アクセス方式として VSAM を使用し、データベースの初期ロードを行うすべてのアプリケーション・プログラムは、チェックポイントから再始動することはできません。

一般に、ジョブ・ステップの最初の実行時に、DISP=MOD を使用していた場合でも、チェックポイントから再始動するときに、GSAM データ・セットに必ず DISP=OLD を使用してください。 DISP=OLD を使用すると、データ・セットは、その初めの位置に位置付けられます。 DISP=MOD を使用した場合は、データ・セットは、その終わりの位置に位置付けられます。

HDAM、PHDAM、HIDAM、および PHIDAM データベース

階層直接 (HD) データベースとは、(セグメントを階層順に物理的に保管するのではなく) セグメントが互いを指し合うようにすることで、データベースのセグメントの階層順を維持するデータベースです。

HD データベースは VSAM ESDS または OSAM データ・セットで直接アクセス装置に保管されます。

ほとんどの場合、HD データベース内の各セグメントは接頭部に 1 つ以上の直接アドレス・ポインターが入っています。直接アドレス・ポインターを使用すると、データベースの中の任意の場所にデータベース・レコードとセグメントを保管することができます。データベースに挿入されたセグメントは、削除されない限り、あるいはデータベースが再編成されるまで、その当初の位置にとどまります。データベース更新アクティビティーの実行中に、セグメントの階層関係を反映するようにポインターが更新されます。

また、HD データベースは、データベース内のスペースを再利用できることから、順次編成データベースとは異なります。ある 1 つのデータベース・レコードの一部分またはすべてを削除した場合、新しいデータベース・レコードまたはセグメントを挿入するとき、この削除スペースを再使用することができます。

HD データベースは、そこに含まれるルート・セグメントに 2 とおりの方法でアクセスします。すなわち、ランダム化モジュールを使用する方法と 1 次索引を使用する方法です。ランダム化モジュールを使用する HD データベースは、階層直接アクセス方式 *(HDAM)* データベースと呼ばれます。 1 次索引を使用する HD データベースは、階層索引直接アクセス方式 *(HIDAM)* データベースと呼ばれます。

HD データベースは区画に分割することもできます。ランダム化モジュールを使ってルート・セグメントにアクセスする区分 HD データベースは、区分 *HDAM (PHDAM)* データベースと呼ばれます。 1 次索引を使ってルート・セグメントにアクセスする区分 HD データベースは、区分 *HIDAM (PHIDAM)* データベースと呼ばれます。 PHDAM および PHIDAM データベースは、区分副次索引 *(PSINDEX)* データベースとともに、まとめて高可用性ラージ・データベース *(HALDB)* タイプ・データベースと呼ばれます。

ランダム化モジュールを使用する HD データベース内のストレージ編成と、1 次索引を使用する HD データベース内のストレージ編成は基本的に同じです。主な相違は、そのルート・セグメントへのアクセス方法です。 HDAM または PHDAM データベースにおいては、ランダム化モジュールはルートのキーを調べて、ルート・

セグメントを指すポインターのアドレスを判別します。 HIDAM または PHIDAM データベースにおいては、各ルート・セグメントの保管場所は、索引を検索することによって見つけられます。 HIDAM データベースにおいては、1 次索引は、IMS がロードして維持するデータベースです。 PHIDAM データベースにおいては、1 次索引は、IMS がロードして維持するデータ・セットです。ランダム化モジュールの有利な点は、索引を検索するために必要な入出力操作が不要であることです。

PHDAM および PHIDAM データベースにおいては、IMS は、ランダム化モジュールまたは 1 次索引を使用する前に、区画の選択 と呼ばれるプロセスを使用することにより、ルート・セグメントがどの区画に保管されているかを判別する必要があります。IMS に区画の選択を実行させるには、区画にルート・キーの範囲を割り当てるか、または区画選択出口ルーチンを使用します。

以下の図では、HDAM データベースの論理ビューと PHDAM データベースの論理ビューとを比較しています。

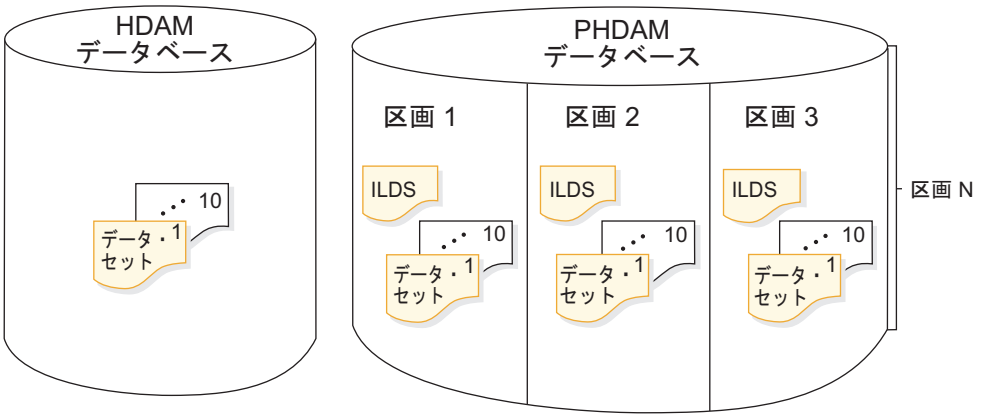


図 32. *HDAM* データベースと *PHDAM* データベースの論理ビューの比較

以下の図では、HIDAM データベースの論理ビューと PHIDAM データベースの論理ビューとを比較しています。

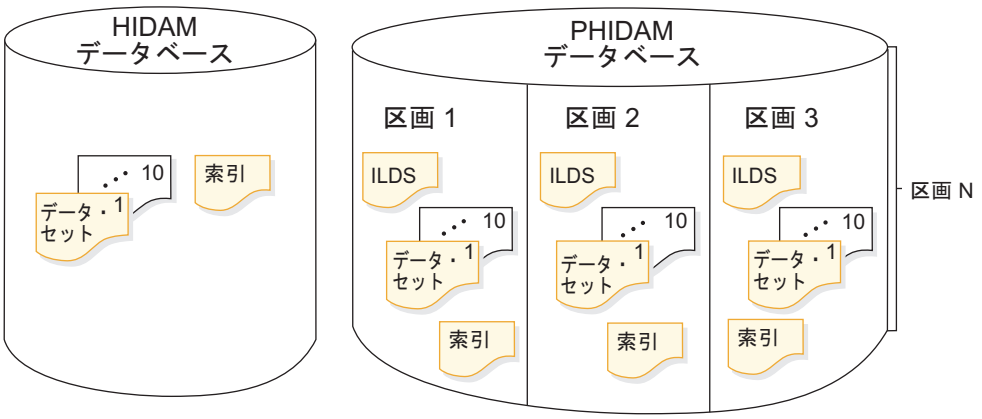


図 33. *HIDAM* データベースと *PHIDAM* データベースの論理ビューの比較

関連概念:

200 ページの『HALDB 区画の選択』

HD データベースの最大サイズ

HDAM、PHDAM、HIDAM、および PHIDAM データベースの許容最大サイズは、データベースが保持できるデータ・セットの数およびデータ・セットのサイズに基づいています。データ・セットの許容最大サイズは、VSAM または OSAM が使用されるかどうか、ならびにデータベースがパーティション化されるかどうかに応じて異なります。

以下の表は、最大データ・セット・サイズ、データ・セットの最大数、ならびに HDAM、PHDAM、HIDAM、および PHIDAM の各データベースの最大データベース・サイズをリストしたものです。

表 *48. HDAM*、*HIDAM*、*PHDAM*、および *PHIDAM* データベースの最大サイズ

データ・セット・タイプ	最大データ・セット・サイズ	データ・セットの最大数	最大データベース・サイズ
OSAM HDAM または HIDAM データベース	8 GB	10 データ・セット	80 GB
VSAM HDAM または HIDAM データベース	4 GB	10 データ・セット	40 GB
OSAM PHDAM または PHIDAM データベース	4 GB または 8 GB[1]	10 010 データ・セット (区画ごとに 10 データ・セット、データベースごとに 1001 区画)	40 040 GB または 80 080 GB
VSAM PHDAM または PHIDAM データベース	4 GB	10 010 データ・セット (区画ごとに 10 データ・セット、データベースごとに 1001 区画)	40 040 GB

注:

1. 最大サイズは HALDB がどのように DBRC に登録されているかによって異なります。デフォルトでは、OSAM データ・セットの最大サイズは 4 GB です。

関連概念:

613 ページの『アクセス方式としての OSAM の使用』

HD データベースを対象として発行可能な DL/I 呼び出し

すべての DL/I 呼び出しは、HD データベースに対して出すことができます。

さらに、次のオプションが使用可能です。

- 複数データ・セット・グループ
- 論理関係
- 副次索引
- 可変長セグメント
- セグメント編集/圧縮出口ルーチン

- データ・キャプチャー出口ルーチン
- フィールド・レベル・センシティビティー
- ロギング、リカバリー、およびオフライン再編成
- HALDB 区画のオンライン再編成

関連概念:

639 ページの『第 26 章 データベースのバックアップおよびリカバリー』

ロギング (システム管理)

738 ページの『HALDB オンライン再編成』

HDAM および PHDAM を使用する場合

HDAM および PHDAM データベースは、通常、データベース・レコードへの直接アクセスを行う場合に使用されます。

ランダム化モジュールは、ルート・セグメント (したがって、データベース・レコード) に高速アクセスを行うことを可能にします。また、HDAM および PHDAM データベースでは、データベース・レコードの DBD の指定に従って、セグメントの経路に高速アクセスを行えるようにします。例えば、以下の図において、物理子ポインターを使用すると、このポインターに従ってセグメント B、C、D、または E に達することができます。データベース・レコード内でのセグメントの階層順の検索は迂回されます。セグメント C、D、または E に到達するためには、セグメント B をアクセスする必要がありません。また、セグメント E に到達するためには、セグメント D をアクセスする必要がありません。セグメント B または C に到達するためには、セグメント A だけをアクセスする必要があります。また、セグメント D または E に到達するためには、セグメント A および C だけをアクセスする必要があります。

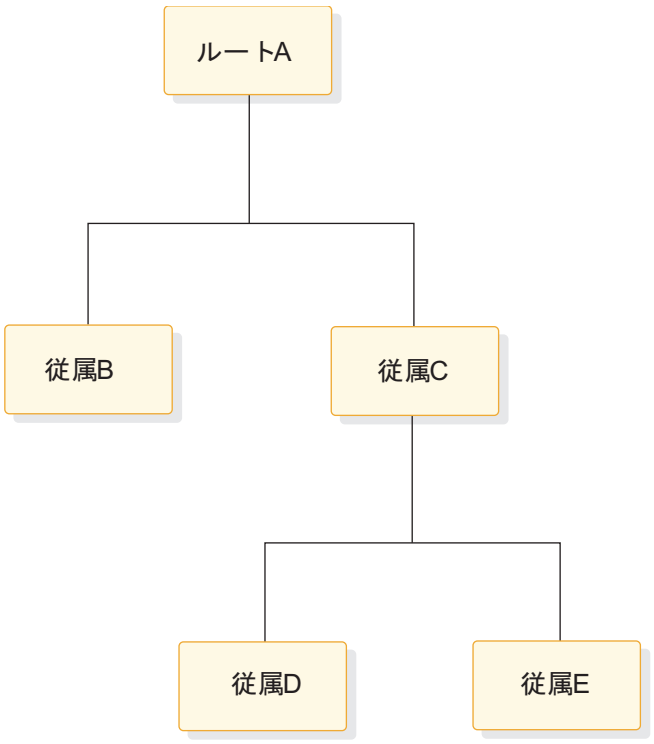


図 34. データベース・レコードの例

HIDAM および PHIDAM を使用する場合

HIDAM および PHIDAM データベースは、通常、データベース・レコードへのランダム・アクセスと順次アクセスが必要であり、しかも、データベース・レコードの中のセグメントの経路へのランダム・アクセスが必要である場合に使用されます。

ルート・セグメントに (したがって、データベース・レコードに) アクセスするのは HDAM (または PHDAM) におけるほど速くはありませんが、それは、HIDAM (または PHIDAM) 索引データベースの中でルート・セグメントのアドレスを検索しなければならないからです。しかし、この索引ではルート・セグメントのアドレスがキー・シーケンスで保管されているため、データベース・レコードを順次処理することができます。

HD データベース内のポインター

HD データベースがどのように保管されて処理されるかを詳しく見る前に、HD データベースで使用されるポインターについて熟知しておく必要があります。

指定することのできるポインターの種類

一方、HD アクセス方式においては、データベース・レコードの中のセグメントは、直接アドレス・ポインターを用いて、階層順に維持されます。

特殊ないくつかの場合を除いて、HD セグメントの各接頭部には、 1 つ以上のポインターが入っています。この各ポインターは、長さが 4 バイトで、各ポインターが

指しているセグメントの相対バイト・アドレスより成り立っています。この場合、「相対」とは、「データ・セットの初めを基準にした」という意味です。

直接アドレス・ポインターにはいくつかの種類があります。このセクションのこの後のトピックで、各ポインターの機能を説明します。しかし、基本的には次の 3 種類です。

- 階層ポインターは、ある 1 つのセグメントから、順方向の階層順または順方向と逆方向の階層順に、次のセグメントを指します。
- 物理子ポインターは、各子セグメント・タイプごとに、ある親からその最初のそれぞれの子、またはある親からそれぞれの最初の子と最後の子を指します。
- 物理兄弟ポインターは、同じ親のもとで、あるセグメント・タイプのあるセグメント・オカレンスから順方向に、あるいは順方向と逆方向に次のセグメント・オカレンスを指します。

データベース・レコードの中のセグメントが通常は階層順に処理される場合には、階層ポインターを使用します。データベース・レコードの中のセグメントが通常はランダムに処理される場合には、物理子ポインターと物理兄弟ポインターを適当に組み合わせて使用します。ポインターに関する記述を読んでいる間、留意すべきことの 1 つは、ある規則のもとに 1 つのデータベース・レコードの中で異なる何種類かのポインターを混ぜて使用することができることです。しかし、ポインターはセグメント・タイプごとに指定されているので、同じセグメント・タイプのオカレンスはすべて同じ種類のポインターを持つことになります。

まず、各ポインターの種類について、このトピックで説明します。このトピックのサブトピックでは、各ポインターの種類を示していて、各図は以下の図のデータベース・レコードに基づいています。

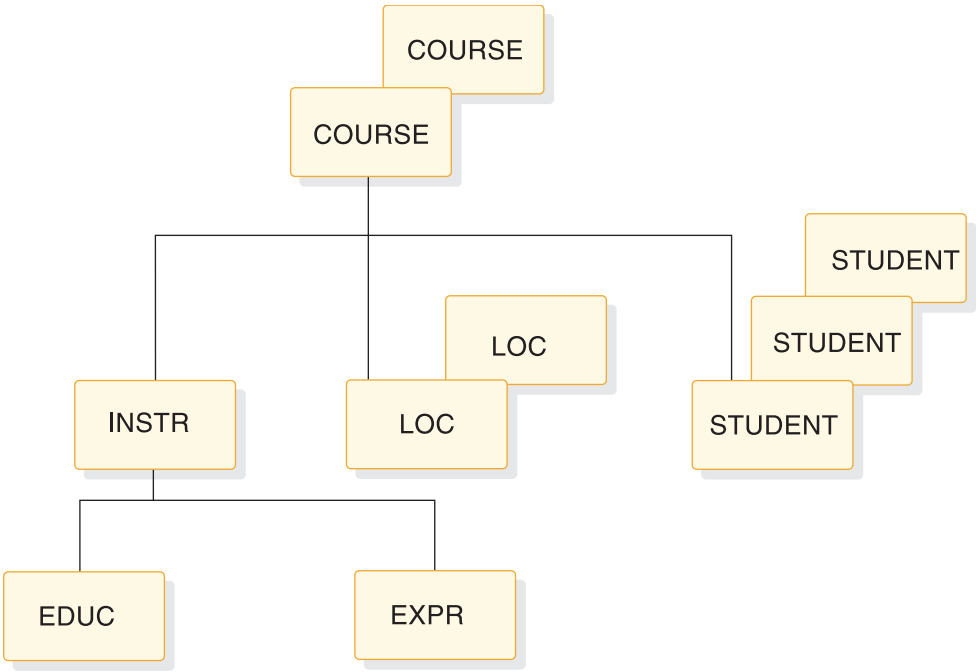


図 *35.* ポインターの図のためのデータベース・レコードの例

関連概念:

166 ページの『各種ポインターの併用』
925 ページの『第 31 章 データベース・タイプの変換』
159 ページの『物理第 1 子ポインター』
160 ページの『物理第 1 子ポインターと物理最終子ポインター』
関連タスク:
929 ページの『データベースの HIDAM から HDAM への変換』

階層順方向ポインター

階層順方向 (HF) ポインターを使用する場合、データベース・レコードの中の各セグメントは、階層の中でそのセグメントより後ろにあるセグメントを指します。

以下の図は、階層順方向ポインターを示したものです。

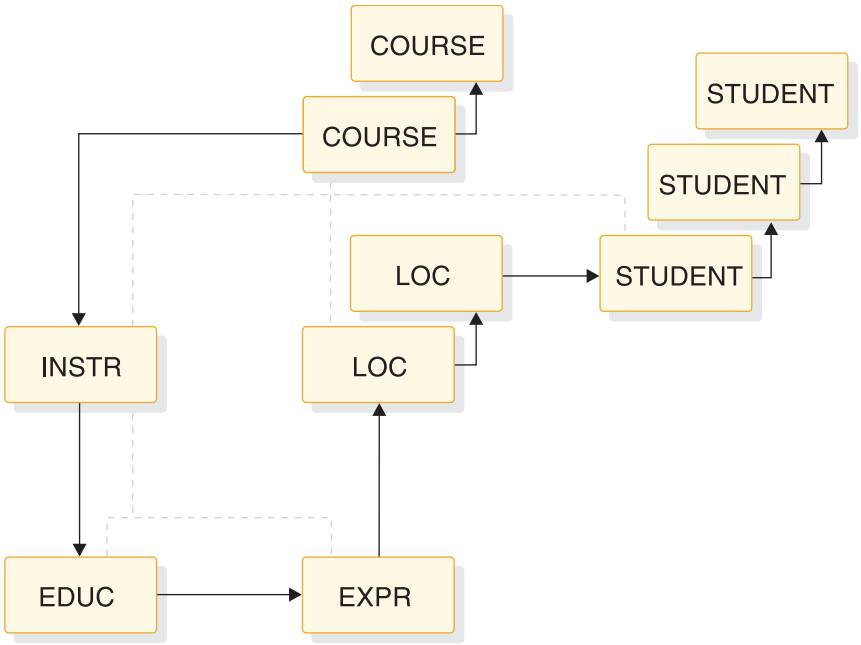


図 *36.* 階層順方向ポインター

アプリケーション・プログラムがセグメントに対する呼び出しを出すと、指定されているセグメントが見つかるまで、HF ポインターに従っていきます。この意味では、HD データベースにおいて HF ポインターを使用するのは、順次編成データベースを使用するのと似ています。いずれの場合にも、ある従属セグメントに達するためには、このデータベース・レコードにおいて階層構造の上でこのセグメントより前にあるすべてのセグメントを調べなければなりません。データベース・レコードの中のセグメントが概して階層順に処理され、しかも、処理にかなり多数の削除操作を必要としない、という場合には、HF ポインターを使用すべきです。多数の削除操作がある場合には、階層順方向ポインターと階層逆方向ポインター (次に説明します) を選択した方が望ましいといえます。

各従属セグメントの接頭部では、HF ポインターに 4 バイトが必要です。ルート・セグメントでは、8 バイト必要です。というのは、ルート・セグメントは次のルート・セグメントを指すだけでなく、データベース・レコードの中の最初の従属セグメントも指すからです。

HF ポインターを指定するには、DBD の中の SEGM ステートメントに PTR=H とコーディングします。

制約事項: HALDB データベースは HF ポインターをサポートしていません。

階層順方向ポインターと階層逆方向ポインター

階層順方向ポインター (HF) と階層逆方向ポインター (HB) を使用する場合、データベース・レコードの中の各セグメントは、階層の中でそのセグメントより後ろにあるセグメントとそれより前にあるセグメントの両方を指します (ただし、従属セグメントはルート・セグメントを指し戻しません)。

HB ポインターだけを使用することはできないので、HF ポインターと HB ポインターは一緒に使用しなければなりません。 以下の図は、HF ポインターと HB ポインターの働きを示したものです。

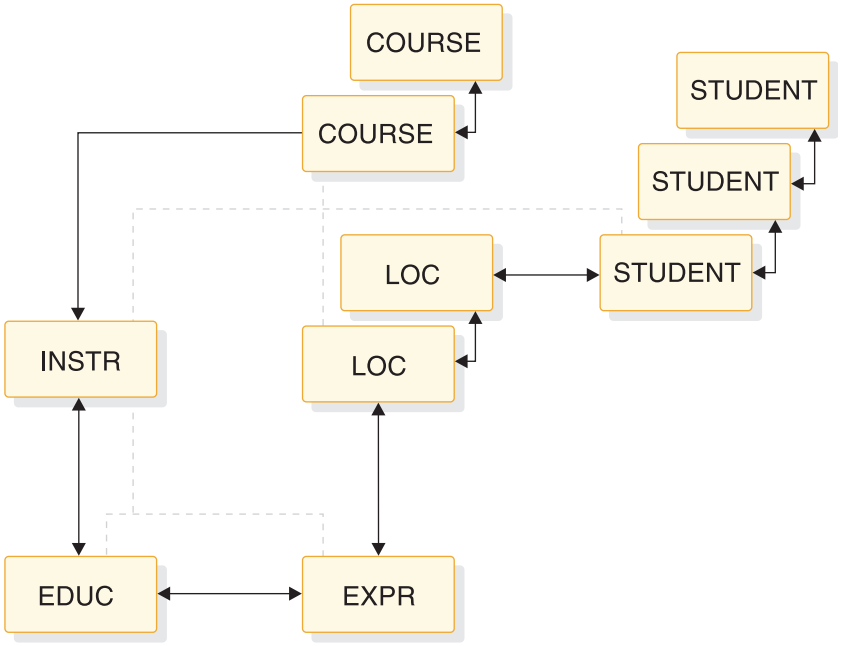


図 37. 階層順方向ポインターと階層逆方向ポインター

HF ポインターの働きは、 157 ページの『階層順方向ポインター』で説明した HF ポインターの働きと同じです。

HB ポインターは、あるセグメントから階層の中のその直前のセグメントを指します。ほとんどすべての場合、削除処理に HB ポインターは必要ではありません。IMS は、チェーン上で検索された前のセグメントの位置を保管し、この情報を削除処理に使用します。チェーン上の前のセグメントがアクセスされていなければ、逆方向ポインターは削除処理に役立ちます。削除されるセグメントが論理関係で入力されている場合に、このことが起こります。

逆方向ポインターは、次のすべての条件が当てはまる場合にだけ役立ちます。

- 論理関係からの直接ポインターまたは副次索引が、削除されるセグメントまたはその従属セグメントの 1 つを指す。
- これらのポインターを用いてセグメントにアクセスする。

- 該当のセグメントが削除される。

各従属セグメントの接頭部では、HF ポインターおよび HB ポインターを収容するのに 8 バイトが必要です。ルート・セグメントでは、12 バイトが必要です。ルート・セグメントは以下のものを指すので多くのバイトを必要とします。

- 順方向に従属セグメントを指します。
- 順方向にデータベースの中の次のルート・セグメントを指します。
- 逆方向にデータベースの中の直前のルート・セグメントを指します。

HF ポインターおよび HB ポインターを指定するには、DBD の中の SEGM ステートメントで PTR=HB とコーディングします。

制約事項: HALDB データベースは HF ポインターおよび HB ポインターをサポートしていません。

物理第 1 子ポインター

物理第 1 子 (PCF) ポインターを使用する場合、データベース・レコードの中の各親セグメントは、その直接の従属子セグメント・タイプそれぞれの最初のオカレンスを指します。

以下の図は、PCF ポインターを示したものです。

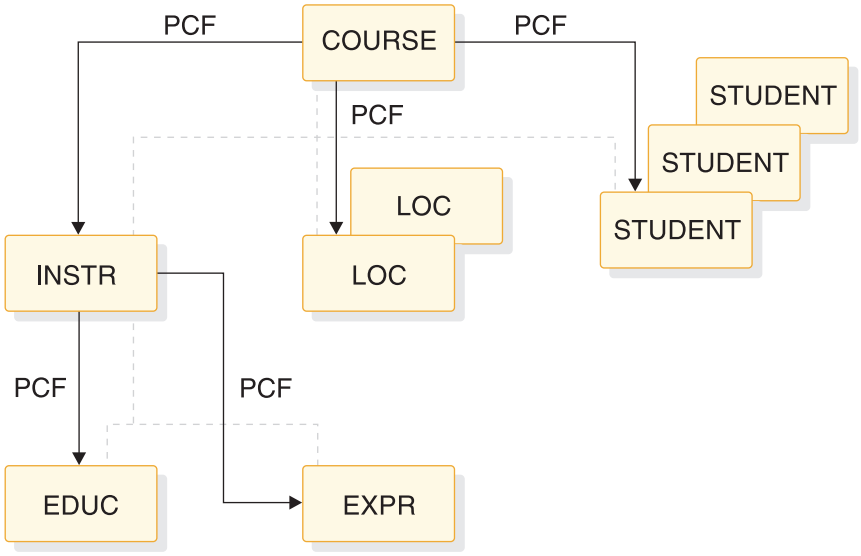


図 *38.* 物理第 *1* 子ポインター

PCF ポインターでは、階層は部分的に接続されるに過ぎません。ある 1 つの親の下にある同一のセグメント・タイプの複数のオカレンスを相互に接続するためのポインターは存在しません。物理兄弟ポインターを使用すると、この接続を構成できます。データベース・レコードの中のセグメントが概してランダムに処理され、しかもセグメント・タイプのシーケンス・フィールドが定義されている場合、またはシーケンス・フィールドが定義されておらず、挿入規則が FIRST または HERE の場合は、PCF ポインターを使用してください。シーケンス・フィールドが定義され

ておらず、また既存のセグメント・オカレンスの最後に新しいセグメントが挿入される場合には、PCF ポインターと物理最終子 (PCL) ポインター (次に説明します) の組み合わせがよい選択になります。

各親セグメントにおいて、各 PCF ポインターごとに 4 バイトずつ必要です。

PCF ポインターを指定するには、DBD の SEGM ステートメントに PARENT=((name,SNGL)) とコーディングします。 これは、このポインターによって指し示される子に対する SEGM ステートメントであり、その親に対する SEGM ステートメントではありません。ただし、注意すべきことは、このポインターはこの親セグメントに保管されるという点です。

関連概念:

155 ページの『指定することのできるポインターの種類』

論理関係がプログラミングに与える影響 (アプリケーション・プログラミング)

関連資料:

ISRT 呼び出し (アプリケーション・プログラミング API)

SEGM ステートメント (システム・ユーティリティー)

関連情報:

物理第 1 子ポインターと物理最終子ポインター

物理第 1 子ポインター (PCF) と物理最終子ポインター (PCL) を使用すると、データベース・レコードの中の各親セグメントは、その直接の従属子セグメント・タイプの最初のオカレンスと最後のオカレンスの両方を指します。

PCL ポインターだけを使用することはできないので、PCF ポインターと PCL ポインターは一緒に使用しなければなりません。 以下の図は、PCF ポインターと PCL ポインターを示したものです。

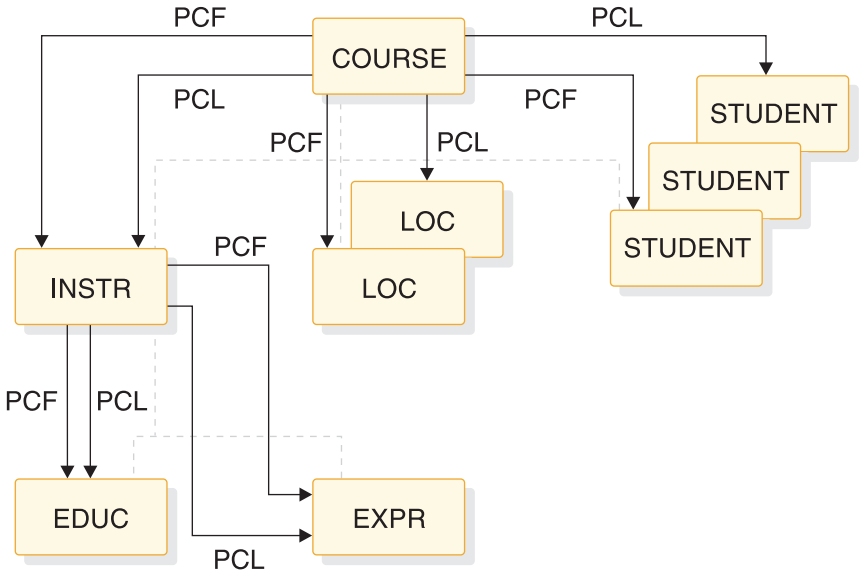


図 *39.* 物理第 *1* 子ポインターと物理最終子ポインター

ある特定の親セグメントの物理子が 1 つしか存在しない場合、PCF ポインターと PCL ポインターが共に同じセグメントを指すことに注意してください。 PCF ポインターと同様に、PCF と PCL ポインターは、階層を部分的に接続するだけで、ある 1 つの親の下にある同一のセグメント・タイプの複数のオカレンスを相互に接続するためのポインターは存在しません。物理兄弟ポインターを使用すると、この接続を構成できます。

(PCF ポインターだけの場合と異なり)、PCF ポインターと PCL ポインターが使用されるのは、一般的に次のような場合です。

- セグメント・タイプのシーケンス・フィールドが定義されていない。
- あるセグメント・タイプの新しいセグメント・オカレンスが、既存のすべてのセグメント・オカレンスの最後に挿入される。

挿入操作にあたって、ISRT 規則として LAST が指定されていれば、セグメントは、このセグメント・タイプの既存のすべてのセグメント・オカレンスの最後に挿入されます。 PCL ポインターを使用すると、セグメントが挿入される位置への高速アクセスが可能です。というのは、最後のオカレンスより前に保管されているすべてのセグメント・オカレンスを順方向にずっと検索する必要がないからです。また、PCL ポインターを使用すると、アプリケーション・プログラムは、一連のセグメント・オカレンスの中の最後のセグメントを速やかに検索することができます。最終セグメントを検索するための呼び出しをアプリケーション・プログラムから出すには、コマンド・コード L を持つ非修飾 SSA を使用します。最終セグメント・オカレンスを入手するために PCL ポインターをたどった場合でも、このデータベースにおけるこれ以降の移動はすべて順方向です。

PCL ポインターを用いて、一連の従属子セグメント・オカレンスの最後のオカレンスから最初のオカレンスへと検索することはできません。

各親セグメントにおいて、各 PCF ポインターと PCL ポインターごとに 4 バイトずつ必要です。

PCF ポインターと PCL ポインターを指定するには、DBD の中の SEGM ステートメントの PARENT= オペランドを、PARENT=((name,DBLE)) のようにコーディングします。 これは、このポインターによって指し示される子に対する SEGM ステートメントであり、その親に対する SEGM ステートメントではありません。ただし、これらのポインターが親セグメントの中に保管されることに注意してください。

もう 1 つ注意すべき点は、ある 1 つの親セグメントで 1 つの直接従属子セグメント・タイプには SNGL と指定し、しかも別の直接従属子セグメント・タイプには DBLE と指定できることです。

以下の DBD ステートメントの例では、PCF ポインターと PCL ポインターを指定しています。

```
DBD
SEGM A
SEGM B  PARENT=((name.SNGL))   (specifies PCF pointer only)
SEGM C  PARENT=((name.DBLE))   (specifies PCF and PCL pointers)
```

以下の図は、データベース定義で PCF ポインターおよび PCL ポインターを指定した結果を示しています。

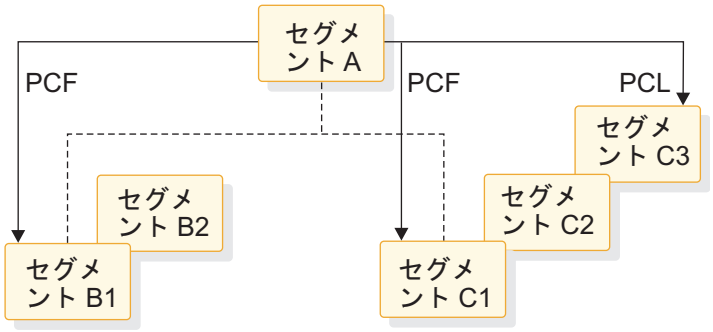


図 *40. PCF* ポインターと *PCL* ポインターの指定

関連概念:

155 ページの『指定することのできるポインターの種類』

論理関係がプログラミングに与える影響 (アプリケーション・プログラミング)

関連資料:

ISRT 呼び出し (アプリケーション・プログラミング API)

SEGM ステートメント (システム・ユーティリティー)

関連情報:

物理兄弟順方向ポインター

物理兄弟順方向 (PTF) ポインターを使用すると、同じ親のもとにある 1 つのセグメント・タイプのそれぞれのセグメント・オカレンスが、順方向に次のセグメント・オカレンスを指します。

PHIDAM データベースの場合を除き、ルート・セグメントに対して PTF ポインターを指定できることに注意してください。 HDAM または PHDAM データベースにおいてこれを指定した場合、このルート・セグメントは、同じルート・アンカー・ポイント (RAP) からチェーニングされている、このデータベースの中の次のルート・セグメントを指します。この RAP からこれ以上ルート・セグメントがチェーニングされていない場合には、PTF ポインターはゼロです。

PTF ポインターを HIDAM データベースのルート・セグメントに対して指定した場合には、このルート・セグメントはデータベースの中の次のルート・セグメントを指しません。

HIDAM データベースの中のルート・セグメントで PTF ポインターを指定する場合は、ルート・セグメントの順次処理すべてにおいて HIDAM 索引を使用しなければなりません。 PTF ポインターだけを使用すると、アクセス時間が増えます。PTF ポインターと物理兄弟逆方向 (PTB) ポインターを指定すれば、このオーバーヘッドを除去することができます。

PHIDAM データベースの中のルート・セグメントに PTF ポインターを使用することはできません。 PHIDAM データベースは、従属セグメントの PTF ポインターのみをサポートします。

PTF ポインターは、階層を部分的に結合するに過ぎません。親セグメントと子セグメントを結合するためのポインターはありません。この結合を行うためには、物理子ポインター (このポインターについては前述しました) を使用することができます。データベース・レコードの中のセグメントが概してランダムに処理され、また、データベース・レコードの順次処理が必要でない、という場合には、PTF ポインターを使用すべきです。

PTF ポインターのためには、1 つのセグメント・タイプの各セグメント・オカレンスに 4 バイト必要です。

PTF ポインターを指定するには、DBD の中の SEGM ステートメントに PTR=T とコーディングします。これは、このポインターを収容するセグメントのための SEGM ステートメントです。

以下の図は、PTF ポインターを示したものです。

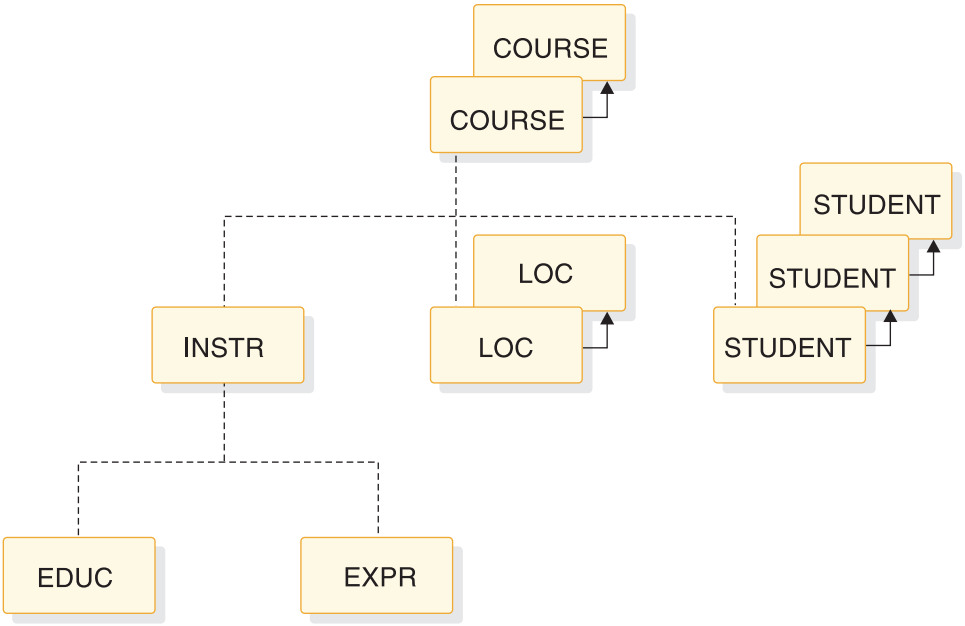


図 *41.* 物理兄弟順方向ポインター

関連概念:

180 ページの『HIDAM データベースにおける RAP の使用』

『物理兄弟順方向ポインターと物理兄弟逆方向ポインター』

168 ページの『HD データベースの一般フォーマットと特殊フィールドの用途』

物理兄弟順方向ポインターと物理兄弟逆方向ポインター

物理兄弟順方向ポインター (PTF) と物理兄弟逆方向ポインター (PTB) を使用する場合、同じ親のもとにある 1 つのセグメント・タイプの各セグメント・オカレンスは、順方向に次のセグメント・オカレンスを指し、しかも、逆方向に直前のセグメント・オカレンスを指します。

PTB ポインターだけを使用することはできないので、 PTF ポインターと PTB ポインターは一緒に使用しなければなりません。 以下の図は、PTF ポインターと PTB ポインターの働きを示したものです。

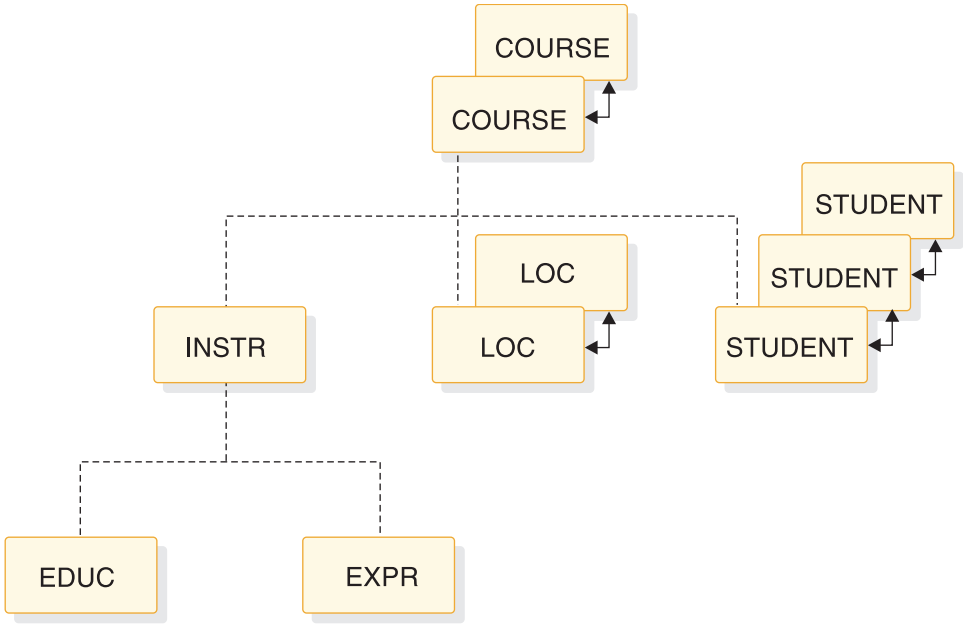


図 42. 物理兄弟順方向ポインターと物理兄弟逆方向ポインター

PTF ポインターと PTB ポインターは、ルート・セグメントに対して指定することができるという点に注意してください。このような指定を行った場合、ルート・セグメントはデータベース内の次のルート・セグメントと直前のルート・セグメントの両方を指し示します。 PTF ポインターと同様に、PTF ポインターと PTB ポインターは、階層を部分的に結合するに過ぎません。親セグメントと子セグメントを結合するためのポインターはありません。この結合を行うためには、物理子ポインターを使用することができます。(このポインターについては前述しました。)

データベース・レコードの高速順次処理を行う必要がある場合は、PTF ポインターだけではなく、PTF ポインターと PTB ポインターを HIDAM または PHIDAM データベースのルート・セグメントで使用してください。ルート・セグメントで PTB ポインターを使用すると、アプリケーション・プログラムは、IMS に HIDAM 索引または PHIDAM 索引へ参照させることなく、データベース・レコードの順次処理を行うことができます。 HIDAM データベースの場合、仮想対になった論理関係によりアクセスされる兄弟チェーン内のセグメントを削除するときに、PTB ポインターを使用するとパフォーマンスが向上します。このような兄弟チェーンのアクセスが起こるのは、論理アクセス・パスからの削除が DASD スペースを解放させる場合です。

PTF ポインターと PTB ポインターのためには、1 つのセグメント・タイプの各セグメント・オカレンスに 8 バイト必要です。

PTF ポインターと PTB ポインターを指定するには、DBD の中の SEGM ステートメントに PTR=TB とコーディングします。

関連概念:

162 ページの『物理兄弟順方向ポインター』

各種ポインターの併用

ポインターはセグメント・タイプごとに指定されるので、1 つのデータベース・レコードにおいて、さまざまな種類のポインターを併用できます。ただし、あるセグメントに対して指定できるのは、階層ポインターか物理的ポインターのいずれかであり、両方を指定することはできません。

1 つのセグメント・タイプに指定できるポインターの種類は次のとおりです。

HF　階層順方向ポインター

HF と **HB**
　階層順方向ポインターと階層逆方向ポインター

PCF　物理第 1 子ポインター

PCF と **PCL**
　物理第 1 子ポインターと物理最終子ポインター

PTF　物理兄弟順方向ポインター

PTF と **PTB**
　物理兄弟順方向ポインターと物理兄弟逆方向ポインター

以下の図は、各種ポインターを併用しているある 1 つのデータベース・レコードを示したものです。例えば、従属セグメント B のように、1 種類のポインターしか指定していない、あるいは指定できなくても、多くのポインターが存在する場合があることに注意してください。また、セグメントが 1 つのチェーンの最後のセグメントである場合、その最後のポインター・フィールドがゼロに設定される (例えば、セグメント E1) という点にも注意してください。各種ポインターの併用規則には、1 つの例外があります。図の凡例で、PTR= または PARENT= オペランドにどのような指定があれば、どのようなポインターが生成されるか説明します。

各種ポインターの併用に関する規則は次のとおりです。

- あるセグメントに対して PTR=H と指定すると、このセグメントからその子を指すための PCF ポインターは存在しません。あるセグメントがその子を指すための PCF ポインターを持つためには、このセグメントに対して PTR=T または TB と指定しなければなりません。
- ルート・セグメントに対して PTR=H または PTR=HB と指定すると、第 1 子によって H ポインターまたは HB ポインターのいずれを使用するかが決まります。その他の子はすべて同じ種類でなければなりません。
- ルート・セグメント以外のセグメントに対して PTR=H と指定すると、その子のいずれに対しても PTR=TB と PTR=HB と指定することはできません。ルート・セグメント以外のセグメントに対して PTR=HB と指定すると、どの子に対しても PTR=T と PTR=H と指定することはできません。

  すなわち、階層ポインターを使用しているセグメントの子には、親セグメントと同じ数のポインター (兄弟ポインターまたは階層ポインター) が入っていなければなりません。
- 直接の親が PTR=H または PTR=HB を使用しているセグメントに対して、PTR=T または TB を指定すると、この兄弟のチェーンの中の最後のセグメントには、ゼロは入りません。代わりに、このセグメントは、このデータベース・レコードの階層中の同じレベルの右側のセグメント・タイプの最初のオカレンスを

指します。兄弟のチェーン が存在せず、単一セグメントに対して PTR=T または TB と指定した場合にも、これが当てはまります (図の従属セグメント B と E2 はこの規則を示しています)。

- 直接の親が PTR=T または TB を使用しているセグメントに対して、PTR=H または HB と指定すると、この兄弟のチェーンの中の最後のセグメントにゼロが入ります (図の従属セグメント C2 はこの規則を示しています)。

以下の図は、データベース・レコードの中で各種ポインターを併用している例を示します。

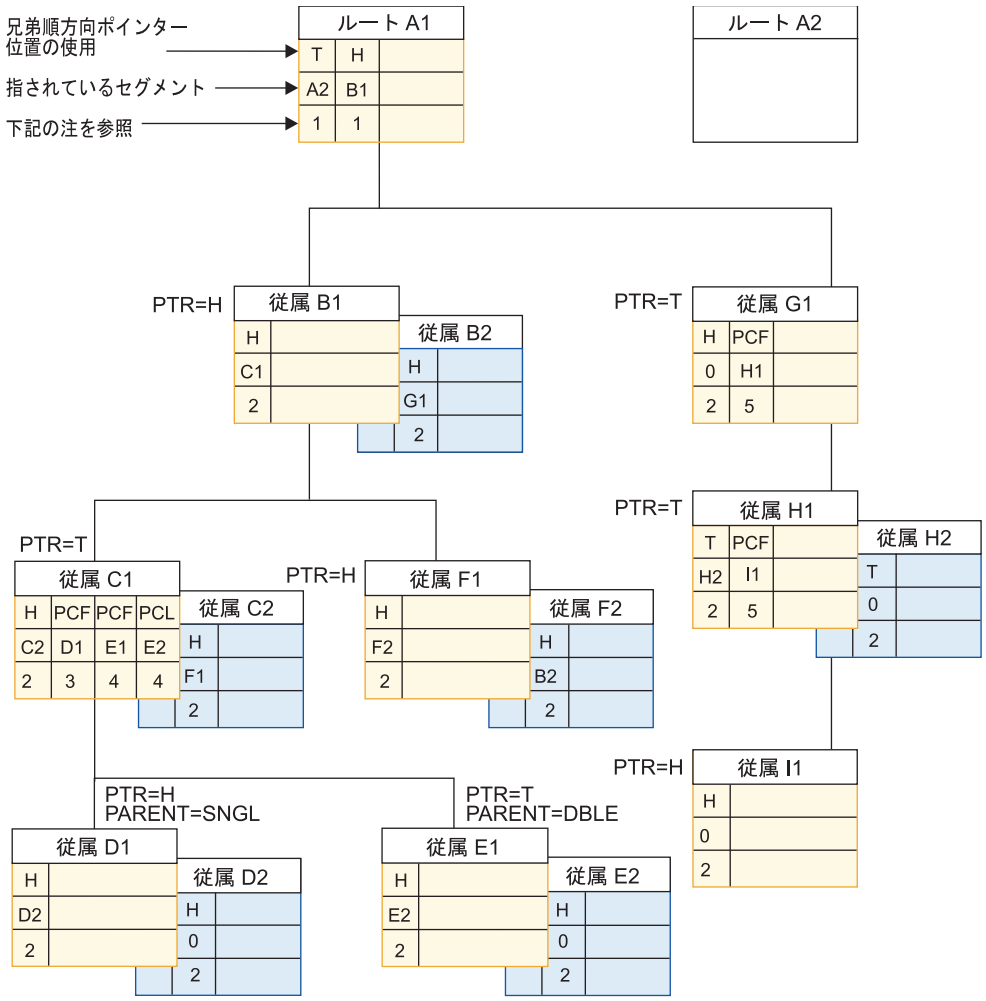


図 *43.* 各種ポインターの併用

図の注**:**

1. これらのポインターは、ルート・セグメントで PTR=H を指定した場合に生成されます。
2. PTR=H を指定した場合は階層 (H) を使用し、その他の場合は兄弟 (T) を使用します。
3. これらのポインターは、セグメント・タイプ C で PTR=T を指定し、セグメント・タイプ D で PARENT=SNGL を指定した場合に生成されます。

4. これらのポインターは、セグメント・タイプ C で PTR=T を指定し、セグメント・タイプ E で PARENT=DBLE を指定した場合に生成されます。
5. これらのポインターは、このセグメント・タイプで PTR=T を指定した場合に生成されます。

関連概念:

155 ページの『指定することのできるポインターの種類』

関連タスク:

604 ページの『セグメント・サイズの決定』

セグメントの接頭部におけるポインターの順序

セグメントに 2 種類以上のポインターが入っている場合は、ポインターが特定の順序でセグメントの接頭部に配置されます。

ポインターは、セグメントの接頭部に以下の順序で配置されます。

1. HF
2. HB

または

1. PTF
2. PTB
3. PCF
4. PCL

HD データベースの一般フォーマットと特殊フィールドの用途

HD データベースの編成方法は特に複雑であるというわけではありませんが、このデータベースには、スペース管理などのために使用される特殊フィールドがあるため、順次編成のデータベースとはまったく異なっているように見えることがあります。

ここで説明しているデータベースは、HDAM または PHDAM データベース、および、HIDAM または PHIDAM データベースです。HIDAM と PHIDAM には、それぞれ追加のデータベース、すなわち 1 次索引データベースがあり、それに対してデータ・セットを割り振る必要があります。HIDAM データベースの 1 次索引には、独自の DBD ステートメントのセットが必要です。PHIDAM データベースの 1 次索引は、独自の DBD ステートメントのセットを必要としません。どちらについても、IMS™ が索引を維持します。このトピックでは、HIDAM レコードのストレージについて述べるときに、この索引データベースについて説明します。以下の図は、HD データベースの一般フォーマットとその中で使用される一部の特殊フィールドを示したものです。

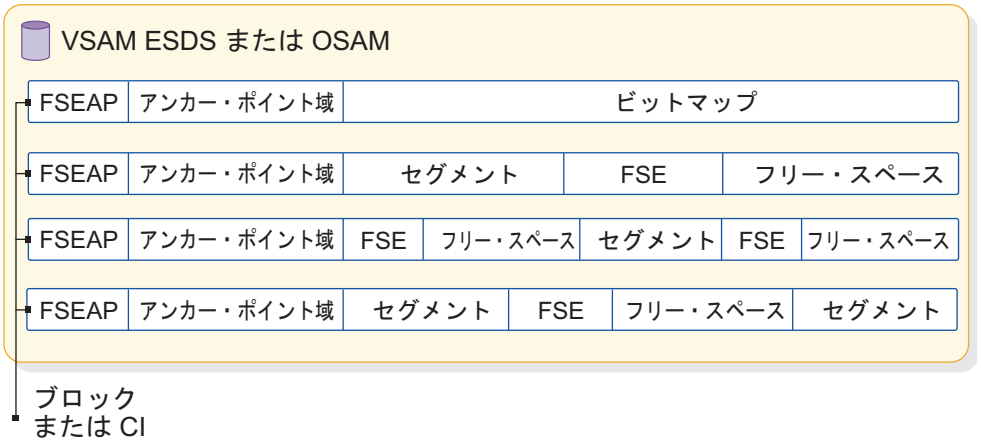


図 *44. HD* データベースのフォーマットとその中の特殊フィールド

HD データベースは 1 つのデータ・セットを使用します。これは、 VSAM ESDS または OSAM データ・セットです。このデータ・セットには、1 つ以上の CI (VSAM ESDS) またはブロック (OSAM) が入っています。このデータ・セットの中のデータベース・レコードは非ブロック化フォーマットです。 OSAM を使用する場合には、論理レコード長がブロック・サイズと等しくなります。 VSAM を使用する場合には、論理レコード長は CI サイズより少し小さくなります。 (VSAM では、CI の中に余分な制御情報が必要です。)

ユーザー自身が論理レコード長を指定することもできますし、これをデータベース記述生成 (DBDGEN) ユーティリティーに指定させることもできます。このユーティリティーは、論理レコード長として全トラック・ブロック、その半分、その 3 分の 1 またはその 4 分の 1 に等しい長さを生成します。

HD データベースの中のすべてのセグメントは、ハーフワード境界で始まります。1 つのセグメントの長さの合計が奇数の場合には、HD データベースの中で使用されているスペースは、そのセグメントより 1 バイト長くなります。この余分なバイトを「遊びバイト」と呼びます。

上の図のデータベースには、フリー・スペースの区域がいくつか含まれていることに注意してください。 このフリー・スペースは、このデータ・セットの中のセグメントを対象として行われた削除操作または置き換え操作の結果であることもあります。 HD データベースにおいてはスペースを再使用できることを銘記してください。あるいは、このフリー・スペースは、データベースをロードするときに、意図的に設けられるということもあります。 HD データベースにおいては、フリー・スペースを残しておくことができますが、それには周期的にブロックまたは CI を空けておくよう指定するか、あるいはブロックまたは CI の中のスペースを一定の率だけ空けておくよう指定します。

さて、上の図に示されている 4 つのフィールドについて見ていきます。このうち 3 つのフィールドは、データベースのスペースを管理するために使用されます。残りの 1 つはアンカー・ポイント域ですが、ルート・セグメントのアドレスが入っています。フィールドは次のとおりです。

- ビットマップ
- フリー・スペース・エレメント・アンカー・ポイント

- フリー・スペース・エレメント
- アンカー・ポイント域

関連概念:

162 ページの『物理兄弟順方向ポインター』

関連タスク:

612 ページの『ステップ 5. ビットマップに必要なスペース量の決定』

ビットマップ

ビットマップはいくつかのビットのストリングが入っています。それぞれのビットは、ある特定の CI またはブロックの中で、データ・セット・グループで定義されている最大のセグメント・オカレンスを保持するのに十分なスペースが使用可能であるかどうかを記述しています。

最初のビットは、このビットマップが入っている CI またはブロックにフリー・スペースがあるかどうかを示しています。連続する各ビットは、連続している次の CI またはブロックにフリー・スペースがあるかどうかを示しています。このビット値が 1 であれば、この CI またはブロックには、データ・セット・グループで定義されている最長のセグメント・タイプのオカレンスを保持するのに十分なだけのスペースがあることを意味しています。このビット値がゼロであれば、十分なスペースが使用可能ではありません。

OSAM データ・セットの中の最初のビットマップは、このデータ・セットの最初のエクステントの最初のブロックの中にあります。VSAM データ・セットにおいては、2 番目の CI がビットマップのために使用され、最初の CI は予約されています。データ・セットの中の最初のビットマップには *n* ビットが入っており、これらのビットは、このデータ・セットの中の次の連続している *n-1* 個の CI またはブロックにおけるスペースの使用可能性を示しています。最初のビットマップの後、n 番目の CI またはブロックごとに別のビットマップが保管され、このデータベースの中の次のグループの CI またはブロック内でスペースが使用可能であるかどうかを記述しています。

HALDB 区画の場合、最初のビットマップ・ブロックに区画 ID (2 バイト) および再編成番号 (2 バイト) が保管されます。これらは、ブロックの先頭で FSEAP より前に保管されます。

ビットマップの例を以下の図に示します。

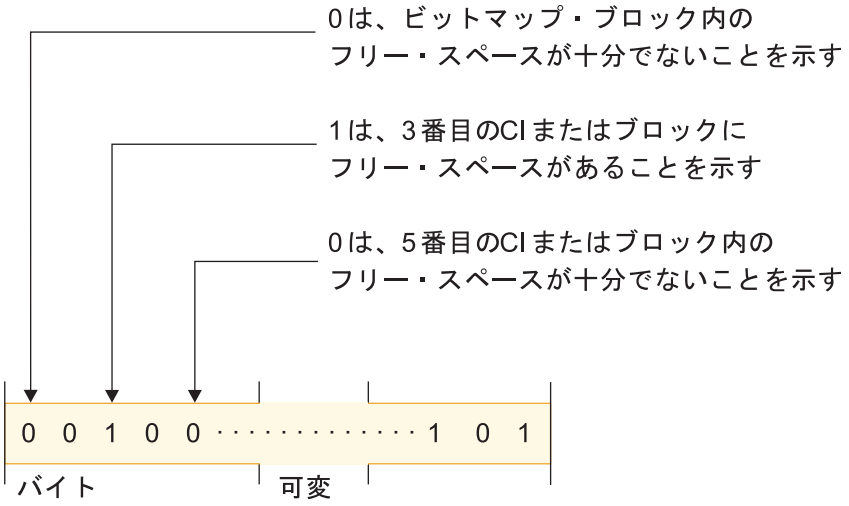


図 *45. HD* データベースのビットマップ

フリー・スペース・エレメント・アンカー・ポイント (FSEAP)

フリー・スペース・エレメント・アンカー・ポイント (FSEAP) は、2 つの 2 バイト・フィールドより成り立っています。

この最初のフィールドにはこの CI またはブロックの最初のフリー・スペース・エレメント (FSE) までのオフセットをバイト単位で示した値が入っています。 FSE とは、ブロックまたは CI のフリー・スペースの区域について記述したものです。2 番目のフィールドは、このブロックまたは CI にビットマップが含まれているかどうかを示しています。このブロックまたは CI にビットマップが入っていない場合には、このフィールドの中にゼロが入っています。データ・セットの中の各 CI またはブロックの先頭に FSEAP が 1 つずつあります。 IMS は、自動的に FSEAP を生成し、これを維持します。

FSEAP を以下の図に示します。

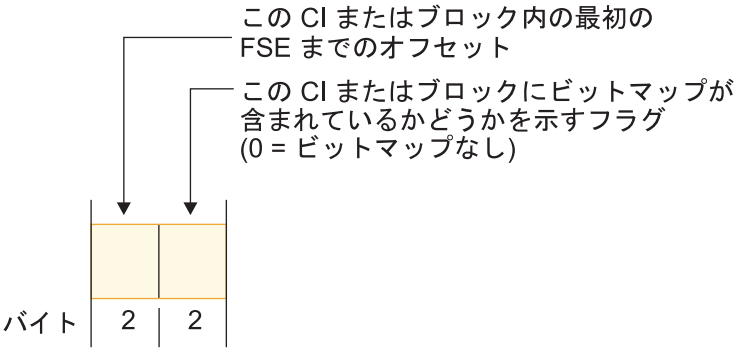


図 *46. FSEAP*

OSAM データ・セットにおける最初のビットマップ・ブロックの中の FSEAP には特殊な用途があります。これは、データベースのための DBRC 使用標識を入れるた

めに使用されます。この DBRC 使用標識は、データベースのオープン時に、更新処理が適切な DBRC RECON データ・セットが使用されていることを検証するために使われます。

フリー・スペース・エレメント (FSE)

FSE は、CI またはブロックの中にある、長さが 8 バイトまたはそれ以上のフリー・スペースの各区域について記述するためのものです。

IMS が、自動的に FSE を生成しこれを維持します。 FSE はフリー・スペースの区域のその最初の 8 バイトを占めます。 FSE は以下の 3 つのフィールドにより成り立っています。

- フリー・スペース・チェーン・ポインター (CP) フィールド。このフィールドには、この CI またはブロックの初めからこの CI またはブロックの中の次の FSE までのオフセットをバイト単位で表した値が入っています。このフィールドは長さが 2 バイトです。ブロックまたは CI の中の最後の FSE では、CP フィールドがゼロに設定されます。
- 使用可能長さ (AL) フィールド。このフィールドには、この FSE で指しているフリー・スペースの長さをバイト単位で表した値が入っています。このフィールドの中の値には、FSE 自体の長さが含まれています。 AL フィールドは長さが 2 バイトです。
- タスク ID (ID) フィールド。このフィールドには、FSE で示されたスペースを解放したプログラムのタスク ID が入っています。このタスク ID によって、ある 1 つのプログラムは、あるスケジューリングの際に、同じスペースを他のプログラムと競合することなく、これを解放し、再使用することができます。 ID フィールドの長さは 4 バイトです。

FSE を以下の図に示します。

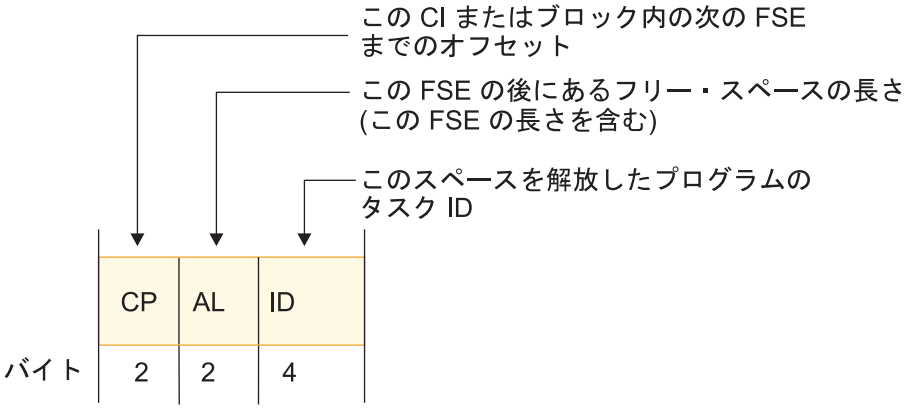


図 *47. FSE*

アンカー・ポイント域

アンカー・ポイント域は、1 つ以上の 4 バイトのルート・アンカー・ポイント (RAP) より成り立っています。

各 RAP には、それぞれ 1 つのルート・セグメントのアドレスが入っています。HDAM の場合、必要な RAP の数を DBD ステートメントの RMNAME パラメーターに指定します。 PHDAM の場合、必要な RAP の数を DBD ステートメントの RMNAME パラメーターに指定するか、または HALDB 区画定義ユーティリティーを使用するか、または DBRC INIT.PART コマンドに指定します。HIDAM の場合 (ただし、PHIDAM を除く)、ルート・セグメントに対して PTR=T か PTR=H を指定することにより、RAP を持つか否かを指定します。ブロックまたは CI 当たり 1 つの RAP しか生成されません。 RAP の使い方は、HDAM、PHDAM、および HIDAM で異なります。

HDAM または PHDAM データベースのアンカー・ポイント域を以下の図に示します。

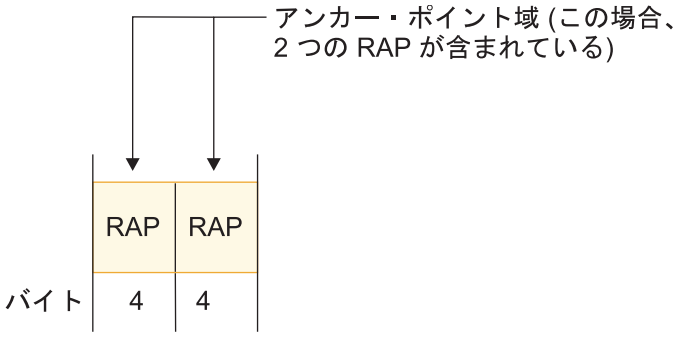


図 *48. HDAM* または *PHDAM* のアンカー・ポイント域

関連概念:

『HDAM および PHDAM レコードの保管方法』

177 ページの『HIDAM および PHIDAM レコードの保管方法』

HDAM および PHDAM レコードの保管方法

HDAM または PHDAM データベースは 2 つの部分から構成されます。ルート・アドレス可能域とオーバーフロー域です。

ルート・アドレス可能域には、すべてのルート・セグメントが入り、データベース・レコードの中の従属セグメントの主要なストレージです。オーバーフロー域は、ルート・アドレス可能域に収まらないセグメントを保管するための場所です。ルート・アドレス可能域のサイズは、DBD ステートメントの中の RMNAME パラメーターの相対ブロック番号 (RBN) オペランドで指定します。PHDAM の場合は、ルート・アドレス可能域のサイズを指定するのに、HALDB 区画定義ユーティリティーを使用することができます。また、ルート・アドレス可能域に保管されるデータベース・レコードの最大バイト数を、DBD ステートメントの中の RMNAME パラメーターの BYTES オペランドを使用して指定します。 PHDAM データベースの場合は、HALDB 区画定義ユーティリティーを使用して、ルート・アドレス可能域の最大バイト数を指定することができます。

次の図は、SKILL データベース・レコードの例を示しています。

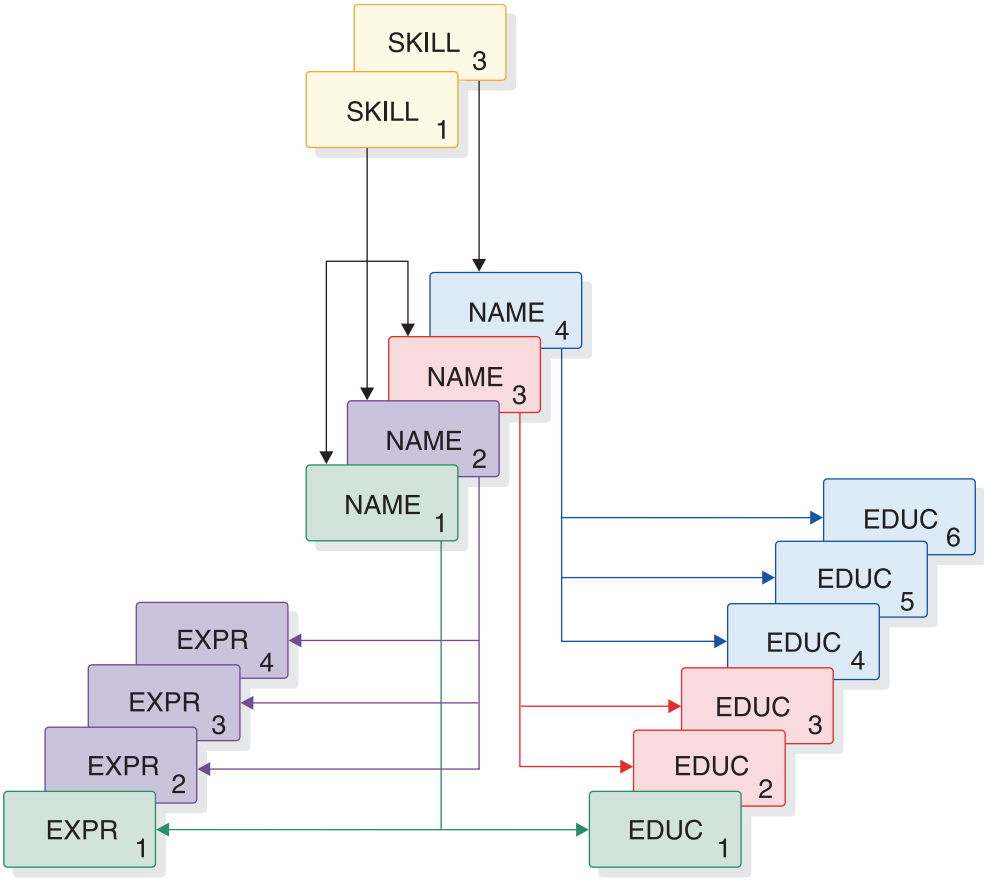


図 *49. HD* データベース内の *SKILL* レコードの *2* つの例

以下の図は、これらのレコードが HDAM または HIDAM データベースにどのように保管されるかを示しています。

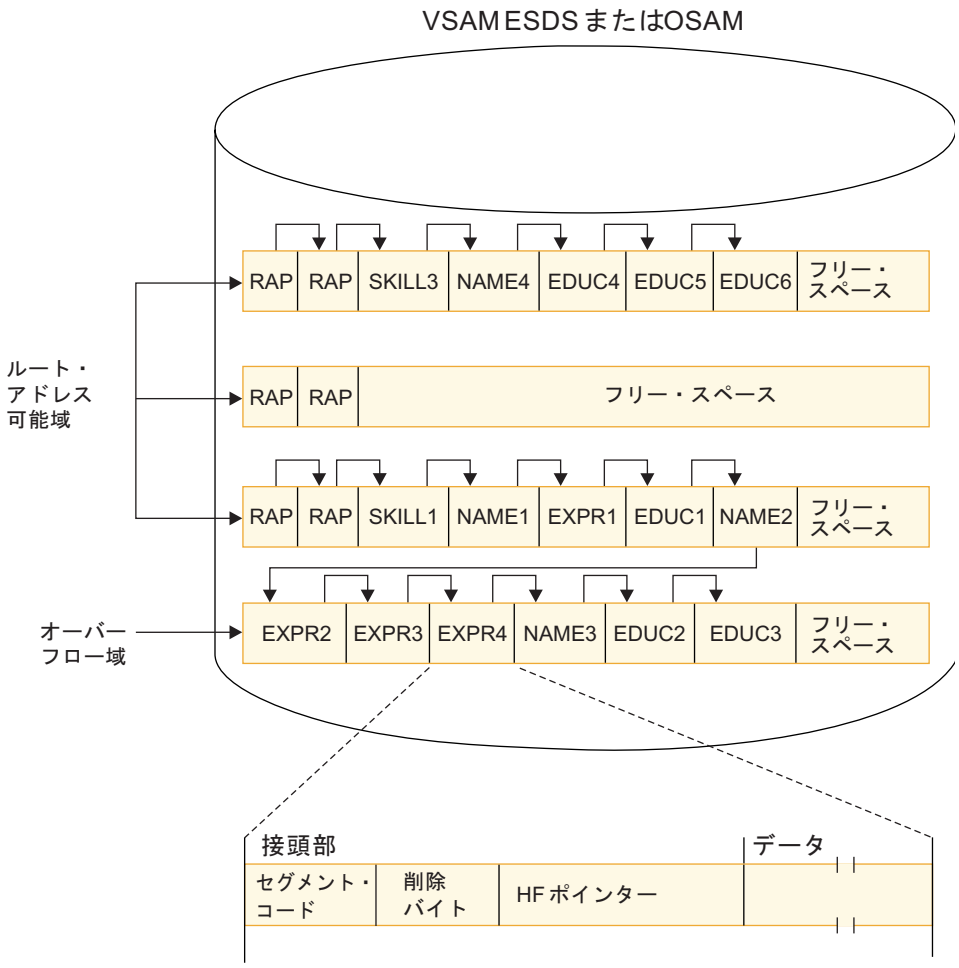


図 *50.* ストレージの中の *HDAM* または *PHDAM* データベース・レコード

データベースを最初にロードするときには、ルート・セグメントと各従属セグメントがルート・アドレス可能域に保管されていきます。この保管が止まるのは、次のセグメントを保管することによって、使用済みのスペースの合計が BYTES オペランドで指定されているスペースの量を超えるときです。この点で、そのデータベース・レコードの中の残りのすべての従属セグメントがオーバーフロー域に保管されます。

HDAM または PHDAM データベースでは、データベース・レコードをロードする順序は問題ではありません。ユーザーのランダム化モジュールによって、各ルート・セグメントをどこに保管するか決められます。しかし、データベースのすべてのタイプで、データベースにロードするときに、1 つのルート・セグメントの従属セグメントはすべて、このルート・セグメントに続けて階層順にロードしなければなりません。

HDAM または PHDAM データベース・レコードを保管するために、ランダム化モジュールは、そのルート・セグメントのキーを取り出し、ハッシュまたは他の何らかの演算手法によって、RBN または CI 番号と、このブロックまたは CI の中での RAP 番号を計算します。このモジュールはこれらの番号を IMS に渡し、IMS はルート・アドレス可能域の中のどこにルート・セグメントを保管するかを決めます。RBN または CI 番号は、IMS に対して、(データ・セットの先頭を基準にして) ど

の CI またはブロックに RAP を保管すべきか知らせます。 RAP 番号は、この CI またはブロックの中のどの RAP にルート・セグメントのアドレスが入るか知らせます。ロードの間、IMS は、ルート・セグメントとその従属セグメントを、(BYTES オペランドに基づいて) 収まる数だけルート・アドレス可能域に保管します。

データベースを最初にロードするときには、IMS は、可能であれば、ルート・セグメントとその従属セグメントを、指定されている CI またはブロックの中の使用可能な最初のスペースに収容します。次に、IMS は、ルート・セグメントの 4 バイト・アドレスを、ランダム化モジュールによって指定された CI またはブロック内の RAP に収容します。 RAP はルート・アドレス可能域の中にのみ存在するという点に注意してください。ルート・セグメントのためのスペースがルート・アドレス可能域の中で使用可能でない場合には、このルート・セグメントは、オーバーフロー域に収容されます。ただし、ルート・セグメントは、ルート・アドレス可能域の RAP からチェーニングされます。

関連概念:

『ルート・セグメントを保管するための十分なストレージがない場合』

172 ページの『アンカー・ポイント域』

183 ページの『HDAM または PHDAM データベースへのルート・セグメントの挿入』

ルート・セグメントを保管するための十分なストレージがない場合

ランダム化モジュールで指定された CI またはブロックに、ルート・セグメントを保管するための十分なスペースがない場合、 IMS は HD スペース検索アルゴリズムを使用してスペースを見つけます。

指定された CI またはブロックの中に、ルート・セグメントを保管するのに十分なスペースが存在しない場合には、このアルゴリズムが、指定された CI またはブロックに一番近い使用可能なスペースを見つけます。スペースが見つかると、ルート・セグメントのアドレスは、やはり、ランダム化モジュールによって生成された元のブロックまたは CI の中の指定された RAP の中に保管されます。

ランダム化モジュールが、複数のルート・セグメントに対して、同じ相対ブロック番号と RAP 番号を指定した場合には、この RAP が 1 つのルート・セグメントを指し、同じ相対ブロック番号と RAP 番号を持つ他のすべてのルート・セグメントが、物理兄弟ポインターを用いて、相互にチェーニングされます。ルート・セグメントは常に、キーの昇順にチェーニングされます。非固有キーが存在する場合には、FIRST、LAST、および HERE という ISRT 規則によって、ルート・セグメントがチェーニングされる順序が決まります。単独のアンカー・ポイント域からこのようにしてチェーニングされているルート・セグメントはすべてシノニム と呼ばれます。

173 ページの『HDAM および PHDAM レコードの保管方法』 は、HDAM または PHDAM データベース・レコード 2 件と、それらの初期ロード後のストレージ内の形を示したものです。この例では、指定されたブロックまたは CI の中に、ルート・セグメントを保管するのに十分なスペースがあり、また各ルート・セグメントごとに他と重複することのない相対ブロック番号と RAP 番号がランダム化モジュールによって生成されています。 BYTES パラメーターは、データベース・レコ

ードの 5 つのセグメントがルート・アドレス可能域に収容できるように十分なスペースを指定しています。残りのセグメントはすべてオーバーフロー域に収容されます。 HDAM または PHDAM データベース・レコードが最初にロードされるときには、ルート・アドレス可能域に収まりきらない従属セグメントは、オーバーフロー域で最初に使用可能なスペースに入れられます。

データベース・レコードの中のセグメントがどのようにして相互にチェーニングされているか注意してください。この場合、物理子ポインターと物理兄弟ポインターの組み合わせではなく、階層ポインターが使用されます。各セグメントは、階層順に次のセグメントを指します。また、CI またはブロック当たり 2 つの RAP が指定されていて、ロードされた各ルート・セグメントがそれぞれ 1 つの RAP によって指されている点に注意してください。 173 ページの『HDAM および PHDAM レコードの保管方法』 では、話を簡単にするために、各種スペース管理フィールドは図示されていません。

ストレージ内では、HDAM または PHDAM セグメントは接頭部とこれに続くユーザー・データで構成されています。接頭部の先頭バイトは、IMS に対してセグメントのタイプを示すセグメント・コードです。この番号は 1 から 255 までの範囲にあります。セグメント・コードは、IMS によってセグメントのタイプに昇順で割り当てられますが、その順序はルート・セグメントから始まって階層順にすべての従属セグメントに続きます。接頭部の 2 番目のバイトは削除バイトです。接頭部の 3 番目のフィールドには、このセグメントが指している 1 つ以上のセグメントのアドレスが入っています。この例では、階層順方向ポインターが使用されています。したがって、EXPR4 セグメントには、1 つのアドレス、すなわち NAME3 セグメントのアドレスだけが入っています。

関連概念:

188 ページの『HD スペース検索アルゴリズムの動作』

173 ページの『HDAM および PHDAM レコードの保管方法』

関連資料:

ISRT 呼び出し (アプリケーション・プログラミング API)

HIDAM および PHIDAM レコードの保管方法

HIDAM データベースは、実際には 2 つのデータベースから構成されています。一方のデータベースにはデータベース・レコードが入り、他方のデータベースは HIDAM 索引が入ります。 HIDAM では、HDAM および PHDAM が使用するルート・アンカー・ポイントではなく、特定のルート・セグメントに達する索引を使用しています。

関連概念:

172 ページの『アンカー・ポイント域』

HIDAM または PHIDAM データベースのロード方法

HIDAM または PHIDAM データベースの中のルート・セグメントには、他と重複していないキー・フィールドがなければなりません。ルート・セグメントのキーに基づいて各ルート・セグメントごとに索引項目が 1 つずつ存在するからです。

HIDAM または PHIDAM データベースを最初にロードするときには、すべてのルート・セグメントをキーの昇順に、さらに各ルート・セグメントごとのすべての従

属セグメントをルート・セグメントに階層順に続けてロード・プログラムに渡す必要があります。以下の図は、 174 ページの図 49 の 2 つの Skill データベース・レコードが初期ロード後にストレージでどのように現れるかを示したものです。HDAM と PHDAM とは異なり、HIDAM と PHIDAM にはルート・アドレス可能域もオーバーフロー域もなく、ただ一連のブロックまたは CI があるだけということに注意してください。

制約事項: PHIDAM データベース用のロード・プログラムは、DLI 領域タイプで実行する必要があります。HIDAM データベース用のロード・プログラムには、この制約事項はありません。

データベース・レコードを最初にロードするときには、データベース・レコードはロード・プログラムに与えられる順序で次々にロードされるにすぎません。以下の図で、各ブロックまたは CI の終わりにあるスペースは、このデータベースをロードするとき指定されたフリー・スペースです。この例では、ブロックまたは CI 当たり 30% のフリー・スペースと指定されています。

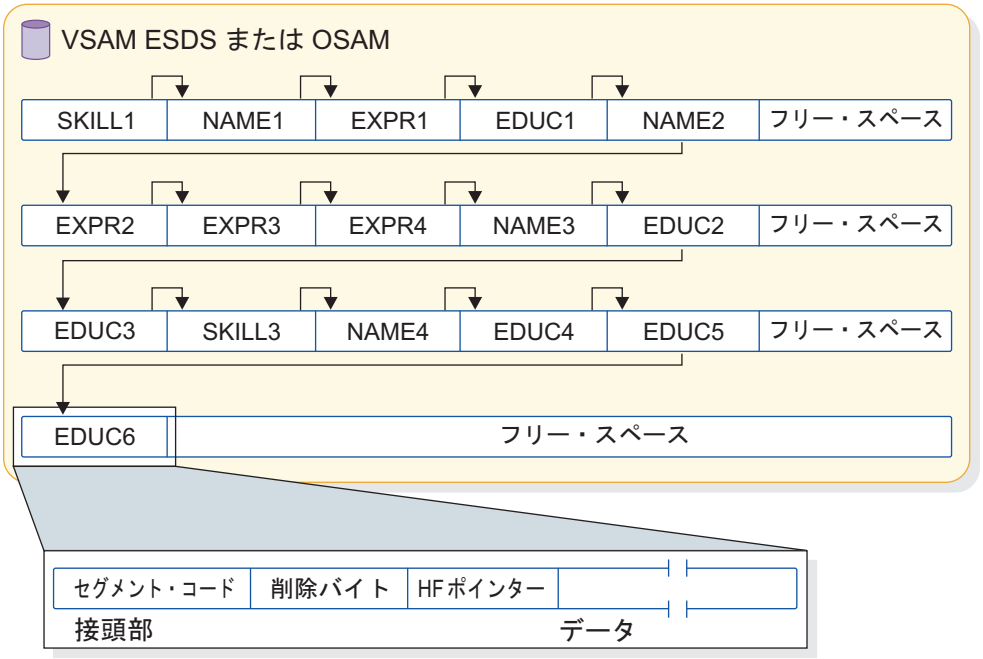


図 *51.* ストレージの中の *HIDAM* データベース・レコード

1 つのデータベース・レコードの中のセグメントがどのようにして相互にチェーニングされているか注意してください。この場合、物理子ポインターと物理兄弟ポインターの組み合わせではなく、階層ポインターが使用されます。各セグメントは、階層順に次のセグメントを指します。上の図には RAP は存在しません。 HIDAM データベースは RAP を持つことができますが、通常は、RAP を使用する必要はありません。

ストレージ内では、HIDAM または PHIDAM セグメントは接頭部とこれに続くユーザー・データで構成されています。接頭部の先頭バイトは、IMS に対してセグメントのタイプを示すセグメント・コードです。この番号は 1 から 255 までの範囲にあります。セグメント・コードは、IMS によってセグメントに昇順で割り当てられます。その順序はルート・セグメントから始まって階層順にすべての従属セグメ

ントに続きます。接頭部の 2 番目のバイトは削除バイトです。接頭部の 3 番目のフィールドには、このセグメントが指している 1 つ以上のセグメントのアドレスが入っています。この例では、階層順方向ポインターが使用されています。したがって、EDUC6 セグメントにはアドレスが 1 つ入っているだけです。これは、このデータベースの中の次のデータベース・レコードのルート・セグメント (この図にはない) のアドレスです。

関連概念:

180 ページの『HIDAM データベースにおける RAP の使用』

索引セグメントの作成

各ルート・セグメントが HIDAM または PHIDAM データベースに保管されるときに、IMS はそのルート・セグメントのための索引セグメントを 1 つずつ作成し、それを索引データベースまたは索引データ・セットの中に保管します。

索引データベースは 1 つの VSAM KSDS で構成されます。この KSDS には、データベースまたは HALDB 区画の各ルート・セグメントごとに 1 つの索引セグメントが入ります。最初に HIDAM または PHIDAM データベースがロードされるとき、IMS はデータベース内の最後のルート・セグメントとして、キーの値がすべて X'FF' のルート・セグメントをデータベースまたは区画に挿入します。

索引セグメントのフォーマットを以下の図に示します。

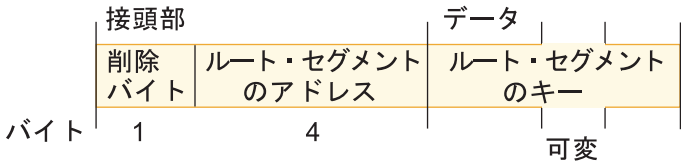


図 52. 索引セグメントのフォーマット

索引セグメントの接頭部には、削除バイトとルート・セグメントのアドレスが入っています。索引セグメントのデータ部には、索引付けされるルート・セグメントのキー・フィールドが入っています。このキー・フィールドは、この索引セグメントがどのルート・セグメント用のものであるかを表し、さらに、HIDAM または PHIDAM データベース内のルート・セグメントがなぜ固有のシーケンス・フィールドをもっていなければならないかの理由ともなっています。各索引セグメントはそれぞれ別個の 1 つの論理レコードです。次の図は、 174 ページの図 49 の 2 つのデータベース・レコードがロードされたときに IMS が生成することになる索引データベースを示しています。

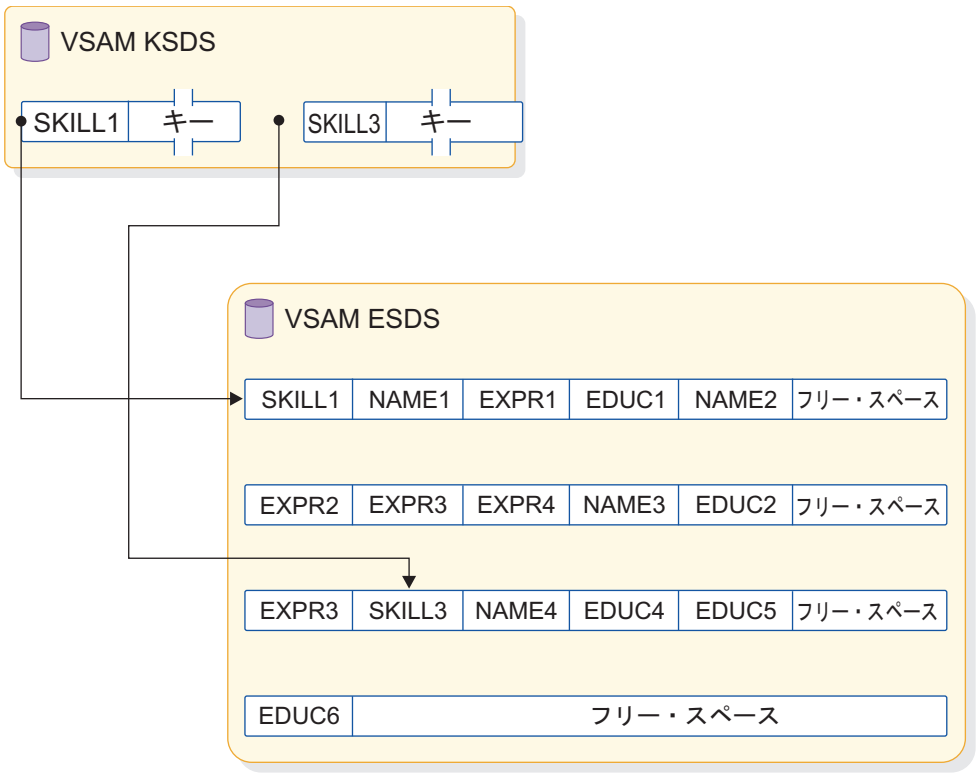


図 *53. HIDAM* 索引データベースまたは *PHIDAM* 索引データベース

HIDAM データベースにおける RAP の使用

HIDAM データベースにおける RAP の使用法は、HDAM または PHDAM データベースにおける RAP の使用法とは異なります。

HDAM または PHDAM では、RAP はルート・セグメントを指すためにあります。しかも、ランダム化モジュールが、同じ相対ブロックと RAP 番号をもつ複数のルート・セグメント (シノニム) を生成した場合には、RAP が 1 つのルート・セグメントを指し、そのルート・セグメントからシノニムが相互チェーニングされていきます。

HIDAM データベースにおいては、ルート・セグメントに対して PTR=T または PTR=H と指定した場合のみ、RAP が生成されます。このどちらかを指定すると、各 CI またはブロックの先頭に RAP が 1 つ生成され、さらにその CI またはブロック内のルート・セグメントは、これらのルート・セグメントを挿入した時点に基づいて、RAP から逆順チェーニングされます。この方式では、RAP はブロックまたは CI に最後に挿入されたルートを指し、そのブロックまたは CI に最初に挿入されたルートの階層順方向ポインターまたは兄弟順方向ポインターがゼロに設定されます。そのブロックの中の他の各ルート・セグメントの階層順方向ポインターまたは兄弟順方向ポインターは、そのブロックにおいて前に挿入されたルート・セグメントを指します。

以下の図は、HIDAM データベースにおいて、ルート・セグメントに対して PTR=T または PTR=H を指定した場合を示しています。図では、以下の略語を使用しています。

FSE フリー・スペース・エレメント

RAP ルート・アンカー・ポイント

SC セグメント・コード

DB 削除バイト

TF 兄弟順方向

H 階層順方向ポインター

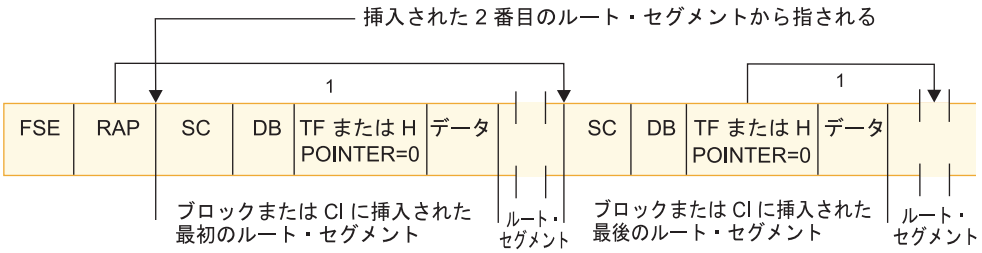


図 *54. HIDAM* データベースのルート・セグメントに *PTR=T* または *PTR=H* を指定した場合

HIDAM ルートに PTR=H を指定すると、階層の最初の従属を指す追加の階層ポインターを使用できることに注意してください。前の図では、「1」がこの追加の階層ポインターが現れる位置を示しています。

PTR=T または PTR=H を指定すると、1 つのルート・セグメントから次のルート・セグメントを指すポインターを使用してルート・セグメントを順次処理することができなくなります。代わりに、あらゆるルート・セグメントの順次処理に、HIDAM 索引を使用しなければならなくなり、このためアクセス時間が増えます。したがって、HIDAM データベースの中のルート・セグメントには PTR=TB または PTR=HB と指定してください。こうすれば、RAP が生成されることなく、ルート・セグメントに対する GN 呼び出しは、通常の物理兄弟順方向チェーンに沿って進められます。 HIDAM ルート・セグメントにポインターを指定しなければ、デフォルトは PTR=T です。

関連概念:

177 ページの『HIDAM または PHIDAM データベースのロード方法』

162 ページの『物理兄弟順方向ポインター』

セグメントへのアクセス

HD データベースの中のセグメントへのアクセス方法は、このセグメントを対象とする DL/I 呼び出しが修飾されているか修飾されていないかによって異なります。

修飾された呼び出し

あるルート・セグメントを対象とする呼び出しが発行され、しかもこの呼び出しがこのルート・セグメントのキーによって修飾されている場合、このセグメントが入

っているデータベース・レコードがどのような方法で見つけ出されるかは、このデータベースが HDAM、PHDAM、HIDAM、または PHIDAM のいずれであるかによって異なります。

HDAM または PHDAM データベースにおいては、ランダム化モジュールがこのルート・セグメント (したがって、そのデータベース・レコード) の位置を生成します。HIDAM または PHIDAM データベースにおいては、このルート・セグメントのキーが入っている索引セグメントが見つかるまで、HIDAM 索引または PHIDAM 索引で検索が行われます。

このルート・セグメントが見つかると、修飾された呼び出しが従属セグメントを対象とするものであるならば、IMS は各従属セグメントの接頭部にあるポインターをたどってこの従属セグメントを探します。この検索が具体的にどのようにして進められるかは、使用しているポインターの種類によって異なります。以下の図は、PCF ポインターと PTF ポインターが使用されている場合にどのようにして従属セグメントが見つけ出されるかを示したものです。

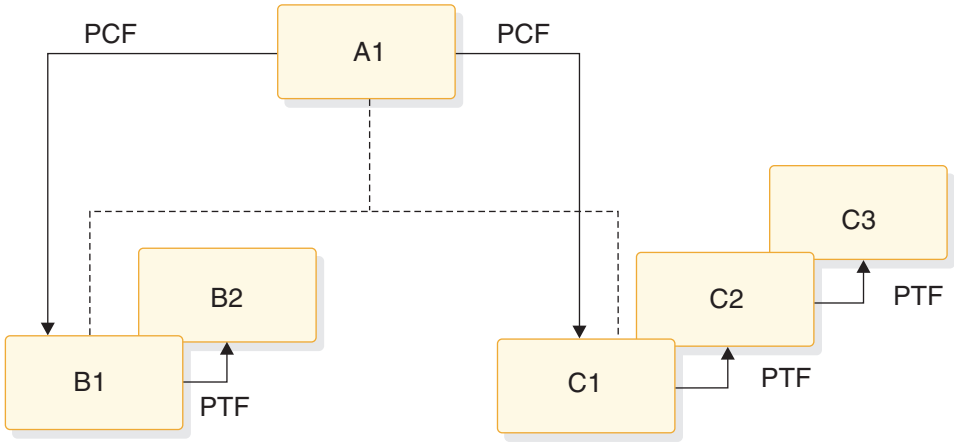


図 *55. PCF* ポインターと *PTF* ポインターを使用した場合の従属セグメントの見つけ方

修飾されていない呼び出し

修飾されていない呼び出しがセグメントに対して出された場合、検索の進み方はいくつかの要因によって左右されます。

要因には、次のものがあります。

- データベースが HDAM、PHDAM、HIDAM、または PHIDAM のいずれであるか
- ルート・セグメントにアクセスするのか従属セグメントにアクセスするのか
- 現在、データベースの中のどこに位置付けられているのか
- どのような種類のポインターを使用しているか
- どのような親子関係が設定されているか (呼び出しが GNP の場合)

可変要因が多いために、セグメントへのアクセス方法については、一般論を展開しても役に立ちません。

ルート・セグメントの挿入

HD データベースの中へのルート・セグメントの挿入方法は、データベースが HDAM、PHDAM、HIDAM、または PHIDAM のいずれかによって異なります。

PHDAM または PHIDAM データベースの場合、ルート・セグメントのキーに基づいて、区画選択が最初に実行されます。

HDAM または PHDAM データベースへのルート・セグメントの挿入

初期ロードの後、HDAM または PHDAM データベースの中にルート・セグメントを挿入する方法は、最初のロードのときのルート・セグメントの挿入方法とまったく同じです。

関連概念:

173 ページの『HDAM および PHDAM レコードの保管方法』

HIDAM または PHIDAM データベースへのルート・セグメントの挿入

ルート・セグメントは、ルート・セグメント・キーの昇順に、HIDAM および PHIDAM データベースに挿入されます。

初期ロードの後、ルート・セグメントは次のように HIDAM または PHIDAM データベースに挿入されます。

1. 挿入しようとしているルートのキーよりも大きいルート・キーを持つ索引セグメントを求めて、HIDAM 索引または PHIDAM 索引の中で検索が行われます。
2. 新たな索引セグメントが、ルート・セグメント・キーについて昇順に挿入されます。
3. 索引セグメントが生成されると、ルート・セグメントは、データベースの中の、HD スペース検索アルゴリズムによって指定された場所に保管されます。

以下の図では、 180 ページの図 53 で初めて示されたデータベースに、ルート・セグメント SKILL2 を挿入しています。

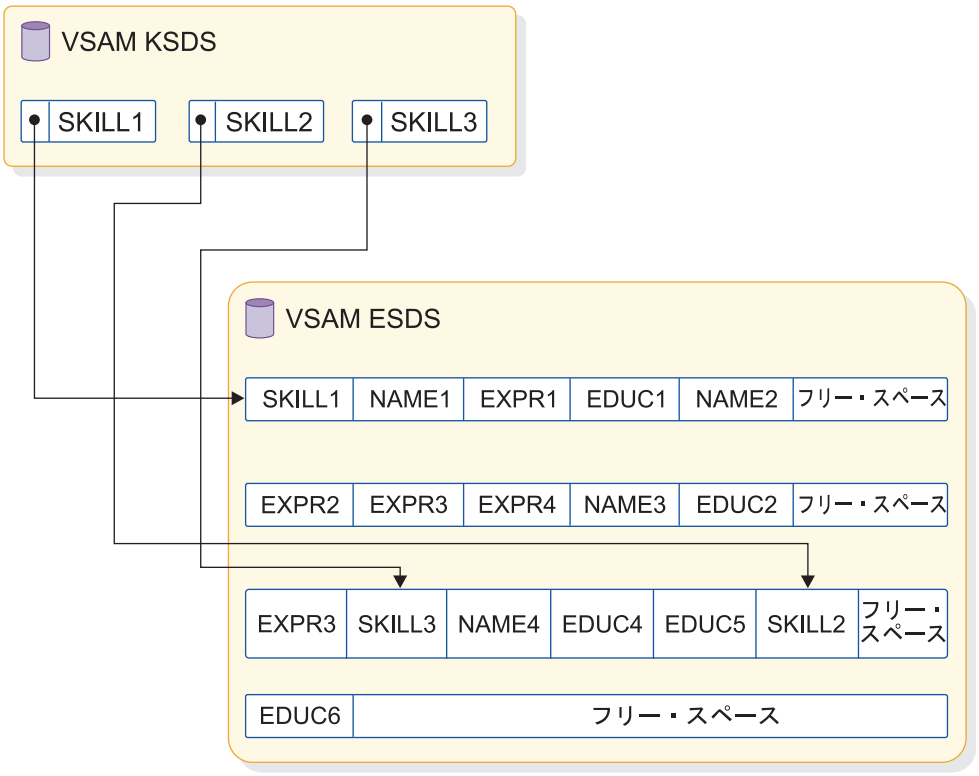


図 *56. HIDAM* または *PHIDAM* データベースへのルート・セグメントの挿入

関連概念:

188 ページの『HD スペース検索アルゴリズムの動作』

ルート・セグメントを挿入する際のスペース管理フィールドの更新

ルート・セグメントを HD データベースに挿入するときには、スペース管理フィールドを更新する必要があります。

以下の図はこの処理を図示したものです。この図においては、実際の値をスペース管理フィールドに入れられるように、いくつかの仮定が立てられています。これらの仮定を以下に列挙します。

- このデータベースは HDAM または PHDAM です (したがって、ルート・アドレス可能域をもちます)。
- VSAM がアクセス方式です。したがって、このデータベースの中には、(ブロックではなく) CI があります。 VSAM が使用されているので、各論理レコードには 7 バイトの制御情報があります。
- 論理レコードの長さは、512 バイトです。
- 各 CI の中には RAP が 1 つあります。
- 挿入されるルート・セグメント (SKILL1) の長さは、32 バイトです。

「挿入前」の図は、ビットマップが入っている CI を示しています (VSAM においては、ビットマップは常にデータベースの 2 番目の CI の中にあります)。このビットマップの 2 番目のビットが 1 に設定されていますが、これは次の CI の中にフリー・スペースがあることを示しています。次の CI (すなわち、CI #3) においては、

- FSEAP は、この CI の先頭から 8 バイトのところに FSE が 1 つあることを示しています (1 つの FSE がフリー・スペースの 1 つの区域について記述します)。
- アンカー・ポイント域 (この場合、RAP を 1 つ含んでいます) にはゼロが入っています。ルート・セグメントが現在この CI には保管されていないからです。
- FSE AL フィールドは、この FSE の先頭からの 497 バイトのフリー・スペースが使用可能であることを示しています。

挿入される SKILL1 ルート・セグメントの長さは 32 バイトです。したがって、CI #3 には SKILL1 を保管するのに十分なスペースがあります。

「挿入後」の図は、SKILL1 が挿入されるとき、CI #3 のスペース管理フィールドがどのようにして更新されるかを示しています。

- FSEAP は、今この CI の先頭から 40 バイトのところに FSE が 1 つあることを示しています。
- RAP は SKILL1 を指しています。 RAP の中のポインター値は、次の数式を用いて得られます。

  ポインター値 = *(CI* サイズ*)*(CI* 番号 *- 1) + CI* ルート・セグメントに関するオフセット

  この場合、ポインター値は 1032 (ポインター値 = 512 x 2 + 8) です。
- FSE はフリー・スペースの残りの区域の先頭に「移されました」。 FSE AL フィールドは、この FSE の先頭から始まる 465 バイト (497 - 32) のフリー・スペースが使用可能であることを示しています。

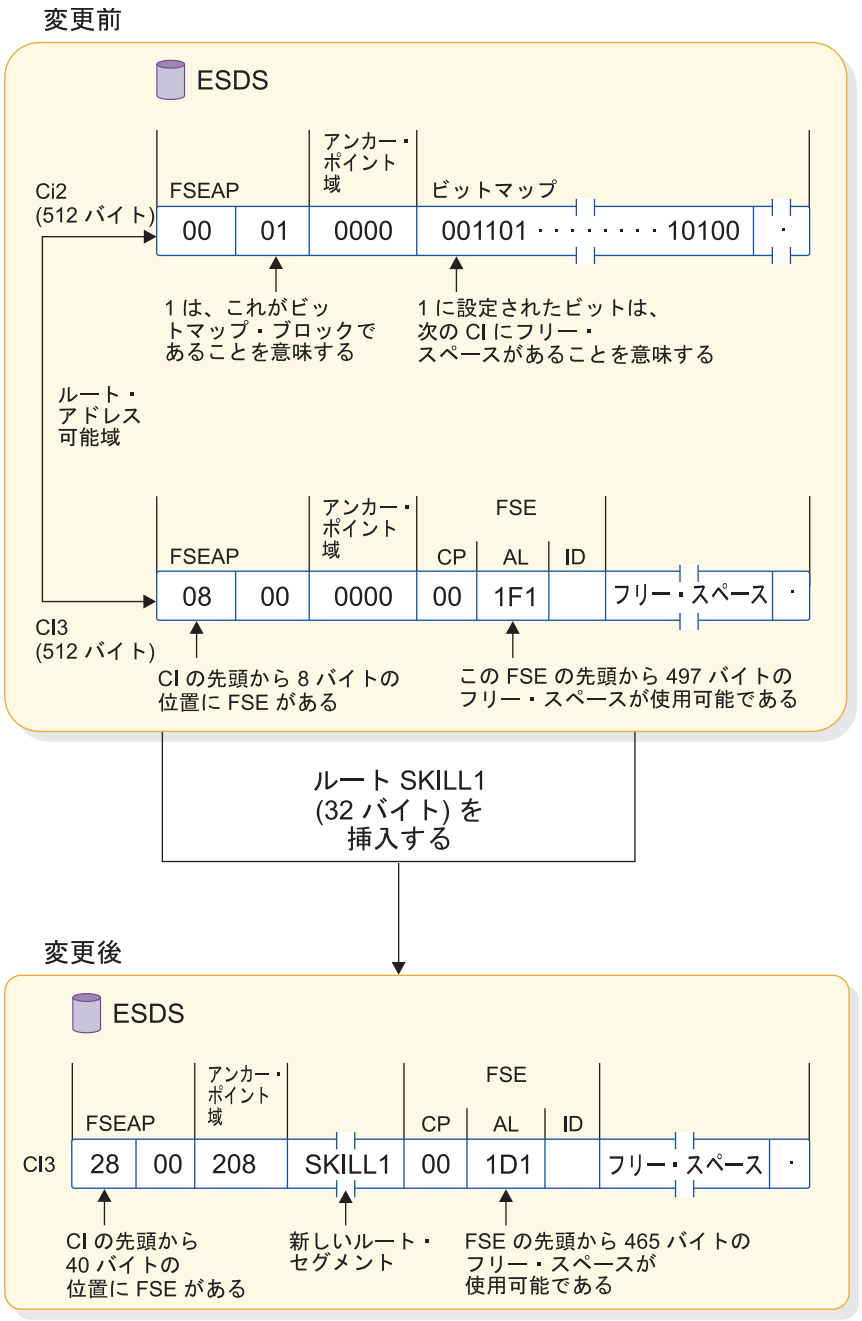


図 57. *HDAM* または *PHDAM* データベースの中のスペース管理フィールドの更新

関連概念:

187 ページの『セグメントの削除』

『従属セグメントの挿入』

従属セグメントの挿入

初期ロードが行われた後に、HD スペース検索アルゴリズムを用いて従属セグメントが HD データベースの中に挿入されます。

HD データベースへのルート・セグメントの挿入と同様に、このデータベース内のさまざまなスペース管理フィールドを更新する必要があります。

関連概念:

188 ページの『HD スペース検索アルゴリズムの動作』

184 ページの『ルート・セグメントを挿入する際のスペース管理フィールドの更新』

セグメントの削除

HD データベースにおいてセグメントを削除すると、このセグメントはデータベースから物理的に除去されます。新しいセグメントを挿入するとき、削除されたセグメントが占めていたスペースを再使用することができます。

HD データベースへのセグメントの挿入と同様、さまざまなスペース管理フィールドを更新する必要があります。

- セグメントが削除されるブロックまたは CI に、今度は 1 つのセグメントを挿入するのに十分なスペースがある場合、ビットマップを更新する必要があります。(ビットマップが示しているのは、定義されている最長のタイプのセグメントを保管するのに十分なスペースがブロックまたは CI の中に存在するかどうかであることを思い出してください。つまり、削除されたセグメントによって解放されたスペースが、定義されている最長のセグメント・タイプに十分な大きさでない場合には、ビットマップは変更されません。)
- ブロックまたは CI において最初の FSE が現在どの位置にあるか示すために、FSEAP を更新する必要があります。
- セグメントを削除するとき、新しい FSE が生成される場合があり、削除されるセグメントのすぐ前にある FSE の中の AL フィールド値を更新する必要が生じる場合もあります。
- 削除されるセグメントが HDAM または PHDAM データベースの中のルート・セグメントである場合には、その PTF ポインターの値が、このセグメントを指していた RAP ポインターまたは PTF ポインターの中に入ります。これによって、RAP からのチェーンが維持され、削除されるセグメントがチェーンから除かれます。

削除されるセグメントが、すでに使用可能なスペース域の隣りにある場合は、この 2 つの区域が結合されて 1 つの区域になります。ただし、この 2 つの区域が、まだ同期点に達していないオンライン・タスクによって作成されたものである場合には、この限りではありません。

関連概念:

184 ページの『ルート・セグメントを挿入する際のスペース管理フィールドの更新』

セグメントの置き換え

固定長セグメントを使用している限り、HD データベースにおけるセグメントの置き換えは直接的に行われます。

セグメント・データは、変更されたらストレージの中の元の場所に戻されるだけです。セグメントの中のキー・フィールドを置き換えることはできません。

可変長セグメントが、それより長いセグメントで置き換えられる場合に、そのセグメントに隣接している、十分なスペースが使用可能であればそのセグメント・データは元の位置に戻されます。隣接しているスペースが使用できない場合には、セグメントの拡張されたデータ部分用にオーバーフロー域よりスペースを入手します。このセグメントは「分離されたデータ・セグメント」と呼ばれます。このセグメントには、1 バイトのセグメント・コードと 1 バイトの削除フラグから成る 2 バイトの接頭部があり、後ろにセグメント・データが続きます。分離されたデータ・セグメントの削除バイトは、X'FF' に設定されており、分離データ・セグメントであることを示します。分離されたデータ部分を示すために、元のセグメントのすぐ後ろにポインターが作成されます。元のセグメントの削除バイトのビット 4 は、このセグメントのデータ部分が分離されたことを示すように ON に設定されます。元のセグメントの残ったスペースは、再使用のために使用可能です。

HD スペース検索アルゴリズムの動作

セグメントを HD データベースの中に挿入する場合の一般的規則は、このセグメント (ルート・セグメントであるか従属セグメントであるかには関係なく) を最も望ましいブロックまたは CI の中に保管するというものです。

関連概念:

176 ページの『ルート・セグメントを保管するための十分なストレージがない場合』

186 ページの『従属セグメントの挿入』

183 ページの『HIDAM または PHIDAM データベースへのルート・セグメントの挿入』

関連タスク:

487 ページの『フリー・スペースの指定 (HDAM、PHDAM、HIDAM、および PHIDAM のみ)』

ルート・セグメント

アクセス方式に応じて、最も望ましいブロックが異なります。

HDAM または PHDAM ルート・セグメントの場合、最も適したブロックは、挿入されるルート・セグメントを指し示す RAP またはルート・セグメントが入っているブロックです。HIDAM または PHIDAM ルート・セグメントの場合、ルート・セグメントに逆方向兄弟ポインターがない場合に、最も望ましいブロックは、すぐ上のキーを持つルート・セグメントを収容しているブロックです。ルート・セグメントに兄弟逆方向ポインターがある場合、最も望ましいブロックは、すぐ下のキーを持つルート・セグメントです。

従属セグメント

最も望ましいブロックは、挿入されるセグメントを指すことになるセグメントを収容しているブロックです。

物理子ポインターと物理兄弟ポインターの両方を使用している場合、最も望ましいブロックは、親または直前の兄弟を収容しているブロックです。階層ポインターを使用している場合に、最も望ましいブロックは、階層内の直前のセグメントを収容しているブロックです。

2 番目に望ましいブロック

1 つ以上のセグメントを最も望ましいブロックに保管できない場合 (例えばスペースがないなどの理由で)、HD スペース検索アルゴリズムによって 2 番目に望ましいブロックまたは CI が検索されます。

このブロックがバッファー・プールの中にあるか、またはビットマップに従ってフリー・スペースが入っている場合のみです。2 番目に望ましいブロックまたは CI とは、データベースのロードまたは再編成時に空いたまま残されるブロックまたは CI です。

n 個ごとにブロックまたは CI を空いたまま残すように指定できます。*n* 個ごとにブロックまたは CI を空いたまま残すように指定しなかった場合、 HD スペース検索アルゴリズムは 2 番目に望ましいブロックまたは CI を検索しません。

HDAM または HIDAM データベースの場合は、DBDGEN ユーティリティーの DATASET マクロで FRSPC= キーワードを使用することによって、フリー・スペースの指定を入力できます。

PHDAM または PHIDAM データベースの場合は、HALDB 区画定義ユーティリティーを使用するか、または DBRC バッチ・コマンド INIT.PART または CHANGE.PART の FBFF(*value*) および FSPF(*value*) パラメーターを使用して、フリー・スペースの指定を区画単位で DBRC RECON データ・セットに入力できます。

HD スペース検索アルゴリズムで定義されている検索範囲は、ステップ 9 および 10 を除いて、最も望ましいブロックが入っている物理エクステントに限られます。最も望ましいブロックがオーバーフロー域にある場合には、ステップ 9 および 10 を除いて、検索範囲はオーバーフロー域に限定されます。

HD スペース検索アルゴリズムのステップは次のようになります。ステップは、実施される順に列挙してあります。ここに列挙されているいずれか 1 つのステップで使用可能なスペースが見つかると、以後の検索は打ち切られ、セグメントが保管されます。

HD スペース検索アルゴリズムは、以下の順序でスペースを探します。

1. 最も望ましいブロックの中で。(このブロックまたは CI はバッファー・プールの中にあります。)
2. 2 番目に望ましいブロックまたは CI の中で。
3. 同じシリンダー上のバッファー・プールの中の任意のブロックまたは CI の中で。
4. ビットマップに照らして、同じトラック上の任意のブロックまたは CI の中で。(ビットマップは、定義されている最長のセグメント・タイプに十分なスペースが使用可能であるかどうかを示しています。)
5. ビットマップに照らして、同じシリンダー上の任意のブロックまたは CI の中で。
6. プラスまたはマイナス *n* シリンダー内のバッファー・プールにおける任意のブロックまたは CI の中で。 *n* は、DBD の DATASET ステートメントの中の SCAN= キーワードで指定されている値です。

HALDB データベースの場合、SCAN= キーワードの値は常に 0 です。

7. ビットマップに照らして、プラスまたはマイナス *n* シリンダー内のバッファー・プールにおける任意のブロックまたは CI の中で。
8. このデータ・セットの終わりのバッファー・プールの中の任意のブロックまたは CI の中で。
9. ビットマップに照らして、このデータ・セットの終わりの任意のブロックまたは CI の中で。このデータ・セットは、次のステップに進む前に可能な限り拡張されます。
10. ビットマップに照らして、このデータ・セットの中のスペースが存在する任意のブロックまたは CI の中で。(HIDAM または PHIDAM データベースにロードするときには、このステップは使用されません。)

ロード・モード処理の上記のステップの中には、とばされるものもあります。

挿入される従属セグメントが 2 次データ・セット・グループの中の最高レベルのものである場合には、スペースを探す場所と方法は、以下のように少し異なっています。

- 最初に、このセグメントが兄弟をもっていない場合には、HD スペース検索アルゴリズムのステップ 1 からステップ 8 までがスキップされます。
- 2 番目に、このセグメントが兄弟チェーンの中でこれに先行する兄弟を持つ場合には、最も望ましいブロックはこの兄弟が入っているブロックです。
- 3 番目に、このセグメントが兄弟チェーンの中でこの後にのみ兄弟を持つ場合には、最も望ましいブロックは、新しいセグメントがチェーニングされる兄弟が入っているブロックです。

ロッキング・プロトコル

IMS はロックを使用して、平行して実行中のプログラムによるデータベースの変更を分離します。

ロックを行うには、プログラム分離 (PI) ロック・マネージャーまたは IRLM を使用します。 PI ロック・マネージャーには 4 種類のロック・レベルのみがあり、IRLM は 11 種類のロック状態をサポートします。

IRLM はまた、要求された「フィードバックのみ」および「テスト」ロックをサポートし、PI ロック・マネージャーから提供されたフィードバックと互換性のあるロック要求についてのフィードバックを提供します。

プログラムを分離するロッキング

すべてのデータベース編成において、ロックされる基本項目は、データベース・レコードです。

データベース・レコードは、その中で位置が最初に取得されたときに、ロックされます。ロックされる項目は、ルート・セグメント、あるいは HDAM または PHDAM の場合にはアンカー・ポイントです。したがって、HDAM または PHDAM の場合、アンカーからチェーニングされるすべてのデータベース・レコードがロックされます。 PCB の処理オプションは、2 つのプログラムが同じデータベース・レコードに同時にアクセスできるかどうか決定します。この処理オプションによって更新が可能になっていると、その他のプログラムは、そのデータベー

ス・レコードに同時にアクセスすることはできません。このデータベース・レコードは、別のデータベースに位置が変更されるか、またはプログラムがコミット・ポイントに達するまで、ロックされています。

あるプログラムが、INSERT、DELETE、または REPLACE 呼び出しを使用して、あるセグメントを更新すると、データベース・レコードではなく、そのセグメントがロックされます。 INSERT または DELETE 呼び出しの場合、他のセグメントが少なくとも 1 つ変更され、ロックされます。

データは常に階層順にアクセスされるので、ルート・セグメント (またはアンカー) についてのロックが取得されると、IMS はいずれかのプログラムが従属セグメントのロックを保持しているかどうかを判断します。どのプログラムも従属セグメントのロックを保持していない場合は、従属セグメントにアクセスする際、これをロックする必要はありません。

このロック・プロトコルにより、IMS は次のような決定を行うことができます。ルート・セグメントが更新されると、ルート・セグメントのロックは、コミットまで、更新レベルで保持されます。従属セグメントが更新されると、これは更新レベルでロックされます。データベース・レコードを終了するときに、ルート・セグメントは読み取りレベルにデモートされます。プログラムがデータベース・レコードに入り、読み取りレベルか更新レベルでロックを入手すると、ロック・マネージャーは、別のプログラムが読み取りレベルでロックを有しているかどうかを示すフィードバックを提供します。これによって、従属セグメントをアクセスするときに、このセグメントをロックするかどうか決まります。 HISAM の場合には、1 次論理レコードがルート・セグメントとして扱われ、オーバーフロー論理レコードが従属セグメントとして扱われます。

これらのロッキング・プロトコルは、PI ロック・マネージャーが使用されるときに適用されます。しかし、IRLM を使用している場合には、従属セグメントを更新するときにロックは取得されません。その代わり、データベース・レコードを終了するときに、ルート・セグメントのロックが単一の更新レベルに保持されます。したがって、従属セグメントの挿入、削除、または置き換えをする場合に、別のロックは必要とされません。

関連概念:

144 ページの『セグメントの削除』

Q コマンド・コードの場合のロッキング

ルート・セグメントか従属セグメントに対して Q コマンド・コードが出されると、そのセグメントについて、共用レベルの Q コマンド・コード・ロックが取得されます。このロックは、同じクラスで DEQ 呼び出しが出されるまで、またはコミット・タイムになるまで、解放されません。

保留状況でルート・セグメントが返されると、データベース・レコードに入るときに取得されたルート・セグメントのロックは、更新能力を有する別のユーザーがそのデータベース・レコードに入るのを防止します。保留状況で従属セグメントが返されると、Q コマンド・コード・テスト・ロックが必要です。あるデータベースに対して Q コマンド・コード・ロックが出されるたびに、標識が付きます。この標識は、そのデータベースに対してスケジュールされた唯一のアプリケーションが終了

すると、必ずリセットされます。標識が設定されていない場合には、Q コマンド・コード・ロックは未解決なものが 1 つもなく、保留状況の従属セグメントを返すテスト・ロックは必要とされません。

ブロック・レベル共用を指定するリソース・ロッキングの考慮事項

リソース・ロッキングは、非シスプレックス環境ではローカルに、あるいはシスプレックス環境ではグローバルに発生する可能性があります。

非シスプレックス環境では、ローカル・ロックが以下の 3 つの方法のいずれかで付与されます。

- 下記のいずれかの理由により、即時に付与されます。

  IMS は必要な IRLM ロックを取得でき、しかもこのリソースに関して他のインタレストがない。

  要求は、他の保有者または待機者のものに対応している。
- 要求が、必要な IRLM ラッチを取得できずに中断されたので、非同期に付与されます。(これは、シスプレックス環境でも発生する可能性があります。) ロックは、ラッチが使用可能になり、しかも下記の 3 つの条件のうちのいずれかが存在する場合に付与されます。

  他に保有者がいない。

  要求は、他の保有者または待機者のものに対応している。

  要求は、他の保有者または待機者のものに対応せず、それらのインタレストが解放された後で付与された。(これは、シスプレックス環境でも発生する可能性があります。)

シスプレックス環境では、グローバル・ロックが以下の 3 つの方法のいずれかで付与されます。

- 下記のいずれかの理由により、**IRLM** によってローカルに付与されます。

  このリソースに対して他のインタレストがない。

  この IRLM にはインタレストしかなく、この要求はこのシステムで保有者または待機者のものに対応しており、さらに XES はすでにこのリソースについて認識している。
- 下記のいずれかの理由により、**XES** 呼び出しと同期して付与されます。

  このリソースに対して、XES は他のインタレストを示さない。

  ハッシュ・クラスに対して、XES は SHARE インタレストのみを示す。
- 下記の 3 つの条件のうちのいずれかの理由により、**XES** 呼び出しと非同期に付与されます。

  IRLM によって XES はハッシュ・クラスに EXCLUSIVE インタレストを示すが、リソース名が一致しない (RMF™ による FALSE CONTENTION)。

  IRLM によって XES はハッシュ・クラスに EXCLUSIVE インタレストを示し、リソース名が一致するが、いずれにしても STATES は共存できるので IRLM CONTENTION 出口により付与される (IRLM FALSE CONTENTION)。

  要求は、他の保有者と共存できず、それらのインタレストが解放された後で CONTENTION 出口により付与される (IRLM REAL CONTENTION)。

ロッキングに対するデータ共用の影響

ブロック・レベルのデータ共用を用いている場合には、IRLM はグローバル・ロックを付与する前に、共用システムが並行できるようにしなければなりません。

ルート・セグメント・ロックはグローバル・ロックですが、従属セグメントはそうではありません。ブロック・レベルのデータ共用を用いる場合には、ロックのために、共用システムは同じバッファーを並行して更新することができません。そのバッファーは更新を行う前にロックされ、そのロックはコミット処理中にバッファーが書き込まれた後まで保持されます。バッファーが読み取られているときには、どのバッファーのロックも取得されません。

いずれかのセグメントに対して Q コマンド・コードが出されると、そのバッファーはロックされます。これによって、共用システムは、Q コマンド・コードのロックが解除されるまで、そのバッファーを更新することができません。

HDAM、PHDAM、HIDAM、および PHIDAM データベースのロッキング

索引を通して、HIDAM または PHIDAM ルート・セグメントにアクセスする場合には、そのルート・セグメントの RBA をリソース名として使用して、その索引上でロックを取得します。その結果、1 つのロック要求が、索引とルート・セグメントとの両方をロックします。

PHIDAM ルート上で NOTWIN ポインターが指定されている場合、データ保全性を提供するために、削除されていない次に高いルート上にロックが必要です。 IMS は、削除されていない索引項目が見つかるまで索引を読み取り、次にルート・セグメントの RBA をリソース名としてロックすることによって追加ロックを取得します。

HDAM または PHDAM データベースにアクセスする場合には、必要なルート・セグメントのアンカーがロックされます。ただしこれは、そのアンカーからチェーニングしているルート・セグメントのいずれかに位置が存在する場合に限ります。したがって、1 つの更新 PCB が、ある HDAM または PHDAM ルート・セグメントに位置を持っていると、他のユーザーはそのアンカーにアクセスできません。あるセグメントが更新されてしまっており、その IRLM が使用されていると、更新しているユーザーがコミットするまで、他のユーザーはそのアンカーにアクセスできません。 PI ロック・マネージャーが使用され、コミットされていない作業単位 (UOW) がアンカーを保留している場合、更新しているユーザーがコミットするまでそのアンカーからチェーニングしているすべてのルート・セグメントおよび従属セグメントにアクセスするためにロックが必要です。

副次索引のロッキング

副次索引に対して挿入、削除、または置き換えが行われる場合、その副次索引はルート・セグメント・ロックでロックされます。

副次索引を使用して、副次索引のターゲットにアクセスするとき、その索引が指しているものによっては、その副次索引をロックする必要がある場合があります。

HIDAM 1 次索引と PHIDAM 1 次索引のバックアップおよびリカバリー

HD 1 次索引のバックアップおよびリカバリーは、データベースが HIDAM データベースであるか、PHIDAM データベースであるかによって異なります。

HIDAM 1 次索引のバックアップおよびリカバリーは、他の多くのデータベース・データ・セットの場合と同じ方法で、イメージ・コピーとログからのデータベース変更レコードを使用することによって行います。1 次索引データ・セットのイメージ・コピーは、索引 HIDAM データベースのデータベース・データ・セットのイメージ・コピーを作成する場合と同じ頻度で作成してください。リカバリー時、イメージ・コピーから 1 次索引を復元した後、データベース・リカバリー・ユーティリティー (DFSURDB0) を使用してログからデータベース変更レコードを適用します。 1 次索引のイメージ・コピーを作成しない場合、唯一の代替リカバリー方法は、IBM IMS Index Builder for z/OS などの別価格の索引ビルダー・ツールで HIDAM 1 次索引を再作成することです。

PHIDAM 1 次索引はバックアップまたはリカバリーしないでください。 PHIDAM 1 次索引は、索引付きデータベース・データ・セットのリカバリー後に、HALDB 索引/ILDS 再作成ユーティリティー (DFSPREC0) を使って再作成します。

関連概念:

639 ページの『第 26 章 データベースのバックアップおよびリカバリー』

関連資料:

- バックアップ・ユーティリティー (データベース・ユーティリティー)
- リカバリー・ユーティリティー (データベース・ユーティリティー)

PHDAM、PHIDAM、および PSINDEX データベース内の区画

HALDB データベースは、それが PHDAM、PHIDAM、または PSINDEX データベースであるかに関係なく、レコードが、区画と呼ばれる管理可能なセクションに分割されている単一データベースとして表示できます。

区画は、HALDB データベースのサブセットです。区画の最大サイズは、区画当たりのデータ・セットの最大数と、データ・セットの最大サイズによって異なります。

多くの場合、個々の区画はデータベース内の他の区画から独立して管理できます。HALDB データベース内で区画をまたがってレコードを分割しても、区画を選択的に処理するのでなければ、その分割はアプリケーション・プログラムには透過です。

HALDB 区画には固有の機能が含まれていて、IMS は、それによって HALDB 区画を管理することができ、ユーザーは HALDB 区画をより容易に使用できるようにもなります。 HALDB 区画の構造も、選択的区画処理のような固有の機能を提供していますが、この機能は、区分化されていないデータベースでは使用できません。

HALDB 区画の名前と番号

各 HALDB 区画は名前、ID 番号、変更バージョン番号、および再編成番号を持っています。ユーザーが区画名を定義し、IMS が番号を割り当てて、管理します。

区画 ID 番号、変更バージョン番号、および再編成番号は、IMS による HALDB 区画の管理および区画に入っているデータの保全性にとって重要なものです。

これらの番号は、RECON データ・セット内の各区画レコードに保管されているだけでなく、HALDB マスター・データベース・レコードにも保管されています。IMS は、区画の管理を支援するため、および区画に入っているデータにアクセスするために HALDB マスター・データベース・レコード内の番号を使用します。

HALDB マスター・データベース・レコードを削除すると、そこに入っていた区画番号は失われ、削除された HALDB マスター・データベース・レコードに関連付けられていた区画は、それ以降、アクセスできなくなります。 HALDB マスター・データベース・レコードを再定義しても、区画番号は複製されません。

HALDB マスター・データベース・レコードを削除すると、論理的に関連していた HALDB データベース内のデータへのアクセスも失われることになります。ターゲット・セグメントの区画 ID および再編成番号が失われると、論理的に関連していたセグメントの拡張ポインター・セット (EPS) が無効になるからです。

このような理由から、HALDB マスター・データベース・レコードは、HALDB データベースとその全データ、およびそのデータベースへの全参照 (そのデータベースと論理的に関連するデータベース内または副次索引データベース内) を永久に削除する場合を除いて、RECON データ・セットから決して削除しないでください。

関連概念:

895 ページの『区画定義制御ブロックと RECON データ・セット内の区画定義』

HALDB 区画名

HALDB 区画名 は、ユーザーが定義して、制御する英数字 7 文字の固有の ID です。

HALDB マスター・データベースまたは区画を削除しない限り、区画名は変更されません。区画名は、その区画に入っているレコードと必ず対応するわけではありません。

ヒント**:** HALDB データベース内の区画名が、各区画に入っているレコード・キーを反映するようにし、さらにデータベースの存続中は区画名をシーケンスどおりに保持したい場合、新規区画名を命名シーケンスを崩さずにデータベースに追加できるように、新規区画用の余地を残す名前を区画に割り当ててください。

例えば、区画名 ABC100、ABC200、ABC300、ABC400 のように定義して、区画に入っているレコードのキー範囲シーケンスに合致するようにします。区画 ABC300 が後になって大きくなりすぎて、そのレコードを新規区画を追加することにより分割しなければならなくなった場合、命名シーケンスを崩さずに新規区画を ABC250 と命名することができます。

多くのコマンドでマスター・データベース名の代わりに区画名を指定することができ、それによって、指定した区画にコマンドを限定することができます。

HALDB 区画 ID 番号

ユーザーが区画を定義すると、IMS はその新規区画に区画 *ID* 番号 を割り当てます。

IMS は、割り当て済みの最後の区画 ID 番号を 1 だけ大きくすることによって新規区画 ID 番号を生成します。ユーザーが区画 ID 番号を制御することはできないので、ユーザーは、データベース内の区画 ID が順番どおりのままになることを想定してはなりません。

例えば、区画 ABC100、ABC200、ABC300 を順に定義した場合、区画 ABC100 は区画 ID 番号 1 になり、区画 ABC300 は区画 ID 番号 3 になります。その後、ユーザーが新規区画 ABC250 を定義すると、IMS はその区画に区画 ID 番号 4 を割り当てます。

HALDB 変更バージョン番号

IMS は、個々の区画と HALDB マスター・データベースに変更バージョン番号 を割り当てます。

IMS は、変更バージョン番号を使用して、内部区画定義制御ブロックが必ず RECON データ・セットに保管されている HALDB 定義と一致するようにします。

DBRC LIST コマンドによって生成されるレポートに変更バージョン番号が表示されることがありますが、その場合でも、ユーザーは変更バージョン番号を制御することはできません。ユーザーが HALDB 区画の定義を変更するたびに、区画と HALDB マスター・データベースの両方について、RECON データ・セットに保管されている変更バージョン番号を DBRC が増分します。 DBRC が番号を増分したことを IMS が検出すると、IMS は HALDB データベースの制御ブロックを、新しい変更を反映するように更新します。

HALDB 区画再編成番号

IMS は、データベース再編成中のデータの保全性を確保するために、各区画の再編成番号 を割り当てて、維持します。

IMS は、また、論理関係または副次索引を使用する HALDB 区画の再編成後の HALDB 自己回復ポインターの処理でも再編成番号を使用します。

再編成番号は、各区画の中の以下の場所に保管されています。

- 各区画の最初のデータベース・データ・セットの最初のブロックの中
- 区画のすべてのセグメントに含まれる間接リスト・キー (ILK) の中
- 個々の副次索引項目および個々の論理子セグメントの拡張ポインター・セットの中
- ILDS の中の、個々の副次索引ターゲット・セグメントおよび個々の論理親セグメント用の各間接リスト項目 (ILE) の中

注意**:** 区画の再編成番号が壊れると、HALDB データベース内の区画を将来再編成または変更すると、重複したセグメント ILK が生成される可能性があり、データは失われます。

再編成番号が壊れる可能性があるのは、HALDB 再編成番号検査機能が使用可能にされておらず、再編成が区画の再編成番号を正しく増分できなかった場合、または EPS に小さい再編成番号が入っているセグメントが区画に移されて、宛先区画の再編成番号が小さくなった場合です。

壊れた再編成番号は検出が困難であり、ユーザーが論理関係または副次索引を使用しないならば、再編成番号が壊れても、直ちに問題が起こることはありません。しかし、後で、壊れた再編成番号を持つ HALDB データベースに論理関係または副次索引を追加すると、データは恐らく失われるでしょう。

区画の再編成番号の整合性を確実にするには、HALDB 再編成番号検査 機能を使用可能にしてください。HALDB 再編成番号検査機能は各区画の再編成番号を RECON データ・セットに記録し、再編成番号が常に適切に増分されるようにします。 HALDB 再編成番号検査機能は、使用可能にされると、RECON データ・セットに記録されているすべての HALDB データベースに適用されます。

HALDB 再編成番号検査機能を使用可能にするには、次のようにします。

- DBRC コマンド `INIT.RECON REORGV` または `CHANGE.RECON REORGV`、あるいはタイプ 1 コマンド `/RMINIT DBRC='RECON REORGV'` または `/RMCHANGE DBRC='RECON REORGV'` を実行します。
- RECON データ・セットに登録されている各 HALDB データベースの区画で、少なくとも 1 つのレコードを更新するプログラムを実行します。

HALDB 再編成番号検査機能を使用可能にすると、RECON データ・セット内のすべての HALDB 区画用の再編成番号がゼロにリセットされます。各区画内のレコードにアクセスすると、RECON データ・セットが、各区画に保管されている現行再編成番号で更新されます。

関連概念:

893 ページの『レコード分散とハイ・キーの区分化』

764 ページの『HALDB 自己回復ポインター処理』

RECON データ・セットの初期化および保守 (システム管理)

関連タスク:

900 ページの『区画のハイ・キーの変更』

関連資料:

/RMxxxxxx コマンド (コマンド)

DBRC コマンド (コマンド)

HALDB 区画の初期設定

区画を定義してそのデータ・セットを割り振った後、区画を初期設定する必要があります。

初期設定処理は区画を使用可能にしますが、区画の中にデータベース・セグメントを入れることはありません。初期設定処理後、区画は空です。

HALDB 区画を初期設定するには、HALDB 区画データ・セット初期設定ユーティリティー (DFSUPNT0) またはデータベース事前再編成ユーティリティー (DFSURPR0) を実行します。

区画の初期設定では、区画 ID 番号と初期再編成番号が PHDAM 区画と PHIDAM 区画に書き込まれます。 HALDB 再編成番号検査が使用可能になっていなければ、初期再編成番号は 1 に設定されます。HALDB 再編成番号検査が使用可能になっている場合は、再編成番号は、RECON データ・セットに保管されている既存の再編成番号から 1 だけ増分されます。

区画 ID 番号および再編成番号は、最初のデータ・セットの最初のブロックの先頭 4 バイトに書き込まれます。最初のブロックはビットマップ・ブロック と呼ばれます。

PHDAM 区画の場合、区画の初期設定では、ダミー・レコードが書き込まれて、削除されます。

PHIDAM 区画の場合、区画の初期設定では、各区画内にすべてが X'FF' のハイ・キー・レコードが書き込まれます。

PSINDEX 区画の場合、区画の初期設定では、ダミー・レコードが書き込まれて削除されますが、それによって使用頻度が高い RBA は非ゼロにされます。

HALDB 区画データ・セット

HALDB データベースには、そのタイプに関係なく、1 から 1001 個の区画を入れることができます。ただし、1 区画当たりのデータ・セットの数は、HALDB データベースのタイプ、および統合 HALDB オンライン再編成機能が使用されているかどうかによって異なります。

HALDB 区画には、以下のタイプのデータ・セットが入っています。

データベース・データ・セット
: データベース・データ・セットには、PHDAM データベースおよび PHIDAM データベース用のセグメント・データが入ります。データベース・データ・セットは、OSAM または VSAM 入力順データ・セット (ESDS) のいずれかです。

索引データ・セット
: 索引データ・セットは、PHIDAM データベースの 1 次索引、または PSINDEX 区画の副次索引データ・セットです。索引データ・セットは、VSAM キー順データ・セット (KSDS) です。

間接リスト・データ・セット **(ILDS)**
: ILDS には、論理関係と PSINDEX が使用される場合にポインターを管理するために使用される、間接リスト項目 (ILE) が含まれています。ILDS は VSAM KSDS です。

HALDB 区画内のデータ・セットの数

1 つの HALDB 区画に入れることができるデータ・セットの最小数と最大数は、HALDB データベースのタイプ、および統合 HALDB オンライン再編成機能を使用するかどうかによって異なります。

IMS は、統合 HALDB オンライン再編成機能向けに、各データベース・データ・セット用の追加データ・セットおよび PHIDAM 1 次索引データ・セットを、再編成される区画内に作成します。追加のデータ・セットは再編成処理で使用され、オンライン再編成が進行中であるかどうか、進行中でない場合は非アクティブなデータ・セットが最後のオンライン再編成の完了後に削除されたかどうかに応じて、アクティブである場合、そうでない場合、あるいは存在しない場合があります。

1 つの HALDB 区画に入れることができるデータ・セットの最小数と最大数を、以下の表にリストします。

表 *49.* 各 *HALDB* 区画のデータ・セットの最小数と最大数

HALDB タイプ	データ・セットの最小数	データ・セットの最大数
PHDAM	2 または 3: データベース・データ・セット用に 1 つの OSAM または VSAM ESDS、ILDS 用に 1 つの KSDS、および統合 HALDB オンライン再編成機能が使用されている場合は 2 番目の OSAM または VSAM ESDS	11 または 21: データベース・データ・セット用に 10 個の OSAM または VSAM ESDS、ILDS 用に 1 つの KSDS、および統合 HALDB オンライン再編成機能が使用されている場合はさらに 10 個の OSAM または VSAM ESDS
PHIDAM	3 または 5: データベース・データ・セット用に 1 つの OSAM または VSAM ESDS、ILDS 用に 1 つの KSDS、1 次索引用に 1 つの KSDS、および統合 HALDB オンライン再編成機能が使用されている場合は 2 番目の OSAM または VSAM ESDS と 1 次索引用の 2 番目の KSDS	12 または 23: データベース・データ・セット用に 10 個の OSAM または VSAM ESDS、ILDS 用に 1 つの KSDS、1 次索引用に 1 つの KSDS、および統合 HALDB オンライン再編成機能が使用されている場合はさらに 10 個の OSAM または VSAM ESDS と 1 次索引用の 2 番目の KSDS
PSINDEX	1: 1 つの KSDS	1: 1 つの KSDS

間接リスト・データ・セットと HALDB 区画

副次索引または論理関係を使用する HALDB のすべての PHDAM 区画および PHIDAM 区画には、間接リスト・データ・セット (ILDS) をそれに割り当てておかなければなりません。

HALDB 自己回復ポインター処理は、データベース再編成後に副次索引ポインターおよび論理関係ポインターを更新するために ILDS を使用します。

バッチ環境では、PHDAM または PHIDAM データベースの中の、副次索引または論理関係を使用しない区画であっても ILDS が割り振られている必要があります。

オンライン環境では、副次索引または論理関係を使用しない区画に対して IMS が ILDS を割り振る必要はありません。

HALDB データベースのすべてのデータ・セットと同じく、ILDS の最大サイズは 4 GB です。ILDS 内の各 ILE は 50 バイトです。したがって ILDS は、単一区画で 8500 万個を超える論理親セグメントまたは副次索引ターゲット・セグメントをサポートできません。ILE の上限に達することはほとんどありませんが、達した場合は、単一区画を複数の区画に分割することができます。

データベースを HALDB に変換するか、区画内のデータベース・データ・セットを再編成するか、または区画内のデータベース・データ・セットのリカバリーを実行する場合、ILDS が更新または再作成されて、ILDS 内の ILE ターゲット・セグメントの物理的な位置の変更が反映されます。 ILDS を更新または再作成できる IMS ユーティリティーは次のとおりです。

- HD 再編成再ロード・ユーティリティー (DFSURGL0)
- HALDB 索引/ILDS 再作成ユーティリティー (DFSPREC0)

これらのユーティリティーは両方とも、VSAM 更新モードまたは VSAM ロード・モードを使用して ILDS を再作成するオプションを提供しています。 VSAM ロード・モードでは、ILDS を定義する VSAM DEFINE ステートメントで要求されたフリー・スペースが追加され、ユーティリティーの現在の実行と、これ以降の再編成およびリカバリーの両方のパフォーマンスが改善されます。

HALDB 区画のデータ・セットおよびリカバリー

HALDB データベースのリカバリーは、各区画をリカバリーすることで実行します。各区画内のデータベース・データ・セットをリカバリーする方法は、非 HALDB データベース内のデータベース・データ・セットの場合と同じです。

区画内のデータベース・データ・セットがリカバリーされたら、1 次索引 (存在する場合) と ILDS を再作成できます。HALDB 1 次索引および ILDS はバックアップまたはリカバリーされません。

HALDB 1 次索引および ILDS を再作成するには、HALDB 索引/ILDS 再作成ユーティリティーを使用してください。これらのユーティリティーは、VSAM 更新モードまたは VSAM ロード・モードを使用して ILDS を作成するオプションを提供しています。 VSAM ロード・モードでは、ILDS を定義する VSAM KSDS DD ステートメントで要求されたフリー・スペースが組み込まれ、ユーティリティーの現在の実行と、これ以降の再編成およびリカバリー両方のパフォーマンスが改善されます。

関連概念:

639 ページの『第 26 章 データベースのバックアップおよびリカバリー』

HALDB 区画の選択

IMS は、データベース・レコードにアクセスするたびに、正しい HALDB 区画を選択しなければなりません。選択処理は、区画の選択 と呼ばれます。

区画の選択では、ルート・セグメントがある区画、および区画が処理される順序が判別されます。

IMS は区画の選択を、区画のハイ・ルート・キーに基づくキー範囲の区分化 を使用するか、またはユーザーが定義した選択基準を使用する区画選択出口ルーチンを使用して行います。

IMS は、ルート・セグメントのキーに基づいて区画にデータベース・レコードを割り当てます。

バッチ・プログラム、BMP プログラム、および JBP プログラムの場合、PCB は、1 つ以上の区画にアクセスするように制限できます。

関連概念:

151 ページの『HDAM、PHDAM、HIDAM、および PHIDAM データベース』

ハイ・キーを使用する区画の選択

ハイ・キー区分化を使用する場合、ハイ・キーは区画の境界を定義し、区画全体でレコードがどのように分散されるかを決定します。

IMS は、個々の区画に定義されたハイ・キーに基づいて区画の選択を行います。区画のハイ・キーは、その区画に含まれるキーの範囲も定義します。IMS は、ルート・セグメントを、ルート・セグメントのキーより大か等しい最小のハイ・キーを持つ区画に割り当てます。例えば、ハイ・キーが 1000、2000、および 3000 の 3 つの区画があるとします。1001 から 2000 のルート・セグメント・キーは、ハイ・キー 2000 の区画に含まれます。

区画のハイ・キーは、HALDB データベースにおける区画の順序も定義します。

ユーザーが出口ルーチンを作成する必要がないので、ハイ・キーの区分化をインプリメントするのがより簡単な方法です。各区画にハイ・キーを割り当てればよいだけです。

ハイ・キーの区分化を使用する PHIDAM および PSINDEX データベースでは、レコードは、HIDAM および非 HALDB 副次索引データベースの中にある場合と同じように、データベース全体にわたってキー・シーケンスで入っています。区画選択出口ルーチンを使用する PHIDAM または PSINDEX データベースでは、レコードは区画内ではキー・シーケンスで入っていますが、区画全体では必ずしもそうではありません。そのため、このようなデータベースは、HIDAM および非 HALDB 副次索引データベースとは整合していません。区画全体にわたってデータベース・レコードがキー・シーケンスになっている必要があるアプリケーション・プログラムは、区画選択出口ルーチンが使用されると、正しく機能しません。

推奨事項: ハイ・キー区分化を使用する場合は、データベース内で最大のハイ・キーを持つ区画にハイ・キー値 (すべて X'FF') を指定します。すべて X'FF' というハイ・キー値により、すべてのキーを確実に区画に割り当てることができます。最後の区画 (最大のキーが指定された区画) にすべて X'FF' 以外のキー値を割り当てた場合、指定されたハイ・キーより大きいキーを持つデータベース・レコードにアクセス、またはそれを挿入しようとすると、その呼び出しに対して FM 状況コードが生成されます。非 HALDB データベース用に作成されたアプリケーション・プログラムが、この FM 状況コードを処理できることはまずありません。

区画選択出口ルーチンを使用する区画の選択

ハイ・キー以外の基準で区画を選択する必要がある場合は、区画選択出口ルーチンを使用できます。

IMS には、ルート・セグメントのキーに基づいて区画にレコードを割り当てる、サンプルの HALDB 区画選択出口ルーチン (DFSPSE00) が用意されています。この出口ルーチンは、順次処理が区画にアクセスする順序も決定します。ユーザーが、ユーザー固有の区画選択出口ルーチンも作成できます。

PHIDAM データベースの場合、区画選択出口ルーチンは、データベース内の他の区画のキー・シーケンスに関してシーケンス外の区画内でレコードをキー・シーケン

スで分散できます。例えば、キーの右端の部分を使用して区画を選択する区画選択出口ルーチンは、データ検索呼び出しで HDAM データベースの特性に従うことができます。区画 PARTA には、レコードが A001、B001、C001、D001 のシーケンスで入っているとします。区画 PARTB には、レコードが A010、B010、C010、D010 のシーケンスで入っているとします。HDAM データベースの場合と同様に、キー C010 を持つセグメントを検索する順次検索呼び出しは、区画 PARTA が選択されると、失敗します。

区画選択出口ルーチンを使用して、一定のデータベース・レコードをその特性で分離することもできます。例えば、PHDAM データベースのほとんどのレコードのサイズが概ね同じで、ごく一部のレコードが相当に大きい場合、その大きなレコードが原因で、区画内のスペース使用状況に問題が生じるおそれがあります。このような大きいレコードのキーが判明していれば、出口ルーチンはそのキーを認識して、異なるスペース特性を持つ区画にそのレコードを配置することができます。その区画では、同じスペース量に分散するレコードの数がずっと少なくなるか、あるいはその区画専用の特殊なランダム化ルーチンを持つことがあります。

IBM IMS HALDB Conversion and Maintenance Aid for z/OS には、このタイプの区画選択を実行できる IHCPSEL0 出口ルーチンが組み込まれています。IHCPSEL0 出口ルーチンを使用する場合、出口ルーチンを作成する必要はありません。使用するキーの一部と各区画用の値を指定すればよいだけです。

IBM IMS HALDB Conversion and Maintenance Aid for z/OS について詳しくは、IBM DB2 および IMS ツールの Web サイト (www.ibm.com/software/data/db2imstools) にアクセスしてください。

アプリケーション・プログラムでの HALDB 区分データベースの処理方法

アプリケーション・プログラムは、その処理が制限されていない限り、区分化された HALDB データベース内のデータを、データが保管されている区画に関係なく、区分化されていない全機能データベースと同じように処理します。

データを順次処理するアプリケーション・プログラムは、HALDB データベース内の区画間を、区画の選択順に進みます。データをランダムに処理するアプリケーション・プログラムは、HALDB データベース内の区画もランダムにアクセスします。アプリケーション・プログラム、および HALDB データベースにアクセスするためにアプリケーション・プログラムが使用する PCB は、アクセスする HALDB データベース内の区画を配慮する必要はありません。

| 注: BMP アプリケーションが変更をコミットせずに持つことができるデータベース
| および HALDB の区画数に制限はありません。

HALDB 区画選択処理

BMP、JBP、およびバッチ・アプリケーション・プログラムを単一の HALDB 区画または HALDB 区画のサブセットに制限することができます。

アプリケーション・プログラムを区画のサブセットに制限すると、アプリケーション・プログラムの複数のインスタンスが、他のアプリケーション・プログラムから

は独立して、HALDB 区画を並列で処理できるようになります。アプリケーション・プログラムが独立して区画を処理することは、ユーティリティーが独立して区画を処理する場合に似ています。

処理を区画のサブセットに制限するには、DFSHALDB DD ステートメントで、区画名と、データベース PCB のラベル名またはデータベース PCB の *n* 番目の位置を指定することにより、データベース PCB の使用を区画に制限します。

DFSHALDB DD ステートメントについて詳しくは、「*IMS V13* システム定義」を参照してください。

論理関係と選択区画処理**:**

BMP、JBP、およびバッチ処理アプリケーションは、論理関係を持つ 1 つ以上の連続する区画のサブセットを選択的に処理できます。

アプリケーション・プログラムの処理が制限されている区画に論理子があり、その論理親がアプリケーションがアクセスできない別の区画にある場合でも、アプリケーションは論理親を処理できます。論理関係があるため、制限付きアクセスを持つアプリケーションが、直接アクセスできない区画を処理することができます。

副次索引と選択区画処理**:**

BMP、JBP、およびバッチ処理アプリケーション・プログラムを HALDB 区分副次索引 (PSINDEX) の 1 つ以上の連続した区画のサブセットに制限することができます。

処理対象の選択区画を指定するには、DFSHALDB ステートメントで PSINDEX 区画の名前を指定します。

アプリケーション・プログラムが PSINDEX を独立型データベースとして処理するのか、PHDAM または PHIDAM データベースの代替処理シーケンスとして使用するのかに関係なく、PSINDEX の区画を選択的に処理できます。

PSINDEX 内の区画は、PHDAM または PHIDAM データベース内の、PSINDEX が示す区画と対応していません。したがって、PSINDEX に対して選択区画処理を指定すると、選択区画処理は PSINDEX 区画のみに適用され、PHDAM または PHIDAM データベースの区画には適用されません。ターゲット・セグメントは、索引付き PHDAM または PHIDAM データベースのどの区画にあってもかまいません。

同様に、PHDAM または PHIDAM データベースに対して選択区画処理を指定すると、選択区画処理では、関連付けられた PSINDEX 内の区画へのアクセスが制限されなくなります。

選択区画処理を PSINDEX に対して使用するのか、索引付き PHDAM または PHIDAM データベースに対して使用するのかに関係なく、選択区画処理は、ターゲット・セグメントが更新されるときに、IMS による副次索引の内部更新に影響を与えることはありません。 例えば、アプリケーション・プログラムが PSINDEX の単一区画に制限されている場合にセグメントを索引付き PHDAM または PHIDAM データベースに挿入すると、対応する新規索引項目は、PSINDEX のどの区画にでも挿入できます。

処理が単一区画に制限されている場合の区画の選択**:**

ハイ・キー区分化を使用する場合、IMS は、DL/I 呼び出しで使用されたルート・キーおよび区画に対して定義されたハイ・キーを使用して区画を選択します。アクセスが単一区画に制限されている場合に、ルート・キーが区画のキー範囲の外側にあると、IMS は FM または GE 状況コードを戻します。

区画を選択するために区画選択出口ルーチンを使用する場合、DL/I 呼び出しで特定のルート・キーを指定すると、このルーチンが呼び出されます。この出口ルーチンは、指定されたルート・キーに基づいて区画を選択します。選択された区画がアプリケーションのアクセスできる区画と異なる場合、IMS は FM または GE 状況コードを戻します。

アクセスが単一区画に制限されている場合、最初の区画は常に、アクセスが制限されている対象である区画であり、次の区画は存在しません。この出口ルーチンは、最初の区画または次の区画の選択のためには呼び出されません。

推奨事項**:** 処理を単一区画に制限する場合、SSA には、その区画のキー範囲内にあるルート・キーのみを含めてください。

単一区画処理の例*:*

以下の各例は、処理対象が単一の区画に制限されているアプリケーション・プログラムに、FM、GE、および GB の各状況コードが返される状況を示しています。

すべての例において、DB PCB の使用は、ルート・キー 201 から 400 を持つレコードを含む HALDB 区画に制限されています。

GU rootkey=110
: ルート・キー 110 はこの区画のルート・キーの範囲の外側にあります。IMS は FM 状況コードを戻します。

GU rootkey=240 GN rootkey=110
: 処理は、110 に等しいキーを見つけるためにルート・キー 240 から順方向に移動します。110 は 240 より小さいため、IMS は GE 状況コードを戻します。

GU rootkey=240 GN rootkey>=110
: 処理は、110 より大か等しいキーを見つけるためにルート・キー 240 から順方向に移動します。区画の終わりに到達するまでにキーが見つからなかった場合、IMS は GB 状況コードを戻します。

GN rootkey>=110
: 処理は、キー 110 から検索を開始しようとします。キーがこの区画のルート・キー範囲の外側にあるため、IMS は FM 状況コードを戻します。

PSINDEX の単一区画処理の例*:*

以下の各例は、処理対象が HALDB 区分副次索引 (PSINDEX) の単一区画に制限されているアプリケーション・プログラムに、FM 、GE、および GB の各状況コードが返される状況を示しています。

すべての例において、DB PCB の使用は、副次索引キー 201 から 400 を持つレコードを含む区画に制限されています。PSINDEX の区画 2 は、索引付き HALDB データベース内の複数の区画を参照しています。

GU xdfldkey=110
: ルート・キー 110 はこの区画のルート・キーの範囲の外側にあります。IMS は FM 状況コードを戻します。

GU xdfldkey=240 GN xdfldkey=110
: 処理は、110 に等しいキーを見つけるためにルート・キー 240 から順方向に移動します。110 は 240 より小さいため、IMS は GE 状況コードを戻します。

GU xdfldkey=240 GN xdfldkey>=110
: 処理は、110 より大か等しいキーを見つけるためにルート・キー 240 から順方向に移動します。区画の終わりに到達するまでにキーが見つからなかった場合、IMS は GB 状況コードを戻します。

GN xdfldkey>=110
: 処理は、キー 110 から検索を開始しようとします。キーがこの区画のルート・キー範囲の外側にあるため、IMS は FM 状況コードを戻します。

処理が区画の範囲に制限されている場合の区画の選択**:**

区画を選択するには、DL/I 呼び出しのルート・キーおよび区画に対して定義されたハイ・キーを使用します。

アクセスが連続した区画の範囲に制限されている場合に、ルート・キーがそれらの区画のいずれかのキー範囲の外側にあると、状況コード FM または GE が戻されます。

区画選択出口ルーチンを使用する場合、DL/I 呼び出しで特定のルート・キーを指定すると、このルーチンが呼び出されます。この出口ルーチンは、与えられたルート・キーに基づいて区画を選択します。選択された区画が、アプリケーションがアクセスできる区画ではない場合、状況コード FM または GE が戻されます。この出口ルーチンは、最初の区画または次の区画の選択のためには呼び出されません。

アクセスが区画の範囲に制限されている場合、最初の区画は、常に、DFSHALDB ステートメントで指定された区画であり、次に選択される区画は、区画ハイ・キーまたは区画選択出口ルーチンで定義された区画選択順序によって異なります。

推奨事項**:** 処理を区画の範囲に制限する場合、SSA には、その区画のキー範囲内にあるルート・キーのみを含めてください。

区画の範囲の選択処理の例*:*

以下の例では、DB PCB の使用が 3 つの HALDB 区画 A、B、および C に制限されています。DFSHALDB ステートメントは、区画 A および NUM=3 を指定しています。

各区画のルート・キー範囲は以下のとおりです。

- 区画 A には、ルート・キー 201 から 400 のレコードが入っています。
- 区画 B には、ルート・キー 401 から 600 のレコードが入っています。

- 区画 C には、ルート・キー 601 から 800 のレコードが入っています。

GU rootkey=110
ルート・キー 110 は、上記区画のルート・キーの範囲外にあります。 IMS は FM 状況コードを戻します。

GU rootkey=240 GN rootkey=110
処理は、110 に等しいキーを見つけるためにルート・キー 240 から順方向に移動します。110 は 240 より小さいため、IMS は GE 状況コードを戻します。

GU rootkey=240 GN rootkey>=110
処理は、110 より大か等しいキーを見つけるためにルート・キー 240 から順方向に移動します。区画の終わりに到達するまでにキーが見つからなかった場合、IMS は GB 状況コードを戻します。

GN rootkey>=110
処理は、キー 110 から検索を開始しようとします。キーがこの区画のルート・キー範囲の外側にあるため、IMS は FM 状況コードを戻します。

GU rootkey=810
ルート・キー 810 は、区画の範囲のルート・キーの範囲の外側にあります。 IMS は FM 状況コードを戻します。

GU rootkey=440 GN rootkey>=110
処理は、110 より大か等しいキーを見つけるためにルート・キー 440 から順方向に移動します。区画の終わりに到達するまでにキーが見つからなかった場合、IMS は GB 状況コードを戻します。

並列区画処理**:**

選択区画処理を使用すると、バッチ環境およびオンライン環境両方で、アプリケーション・プログラムの異なるインスタンスがデータベースの異なる区画を並列で処理できます。

DBRC は、アプリケーション・プログラムが HALDB データベース全体ではなく、個々の区画を処理することを許可します。区画の並列処理は、その処理に必要な時間を短縮できます。

バッチ環境では、バッチ・アプリケーション・プログラムは、HALDB データベース全体ではなく、個々の区画を一時点で 1 つずつ処理することが許可されます。IRLM は不要です。

オンライン環境では、複数の従属領域が、同じ区画内、または異なる区画内のレコードを処理できます。データ共用は不要です。

ブロック・レベルのデータ共用を使用する場合、複数のサブシステムに関して異なる区画を並列で容易に処理できます。サブシステムは、オンライン・システムでも、バッチ・ジョブでもかまいません。ブロック・レベルのデータ共用を使用するには、IRLM を使用し、ブロック・レベルのデータ共用を許可するときに DBRC にデータベースを登録する必要があります。ブロック・レベルのデータ共用について詳しくは、「*IMS V13* システム管理」の DBRC に関する情報とデータ共用に関する情報を参照してください。

アプリケーション・プログラムの複数のインスタンスが区画を並列で処理を使用可能にするには、DFSHALDB DD ステートメントで、区画名と、データベース PCB のラベル名またはデータベース PCB の *n* 番目の位置を指定することにより、各インスタンスのデータベース PCB を区画に制限します。

アプリケーション・プログラムの入力、出力、または処理ロジックに対して、以下の 1 つ以上の変更が必要になる場合があります。

- アプリケーション・プログラムの複数のインスタンスに供給するためにアプリケーション・プログラムへの入力を分割する。
- アプリケーション・プログラムの複数のインスタンスからの出力を統合する。
- アプリケーション・プログラムを変更して、アプリケーション・プログラムがアクセスできない区画が使用不能であることに正しく対応できるようにする。

DFSHALDB DD ステートメントについて詳しくは、「*IMS V13* システム定義」を参照してください。

HALDB でサポートされている IMS ユーティリティー

IMS には、HALDB 区分データベース専用に開発されたユーティリティーがいくつかあります。HALDB 区分データベースは、ほかにも他の全機能データベース・タイプがサポートするユーティリティーの多くをサポートしています。

HALDB データベースに対してどのユーティリティーを実行する場合も、データベース・リカバリー管理 (DBRC) が必須です。各ユーティリティーは、DBRC があるかを検査します。 DBRC がないと、ユーティリティーはエラー・メッセージを出して、終了します。

イメージ・コピー・ユーティリティーは、HALDB ILDS または PHIDAM 1 次索引データ・セットをイメージ・コピーする試みをリジェクトします。リカバリー・ユーティリティーは、HALDB ILDS または PHIDAM 1 次索引データ・セットをリカバリーする試みをリジェクトします。 イメージ・コピー・ユーティリティーもリカバリー・ユーティリティーも、HALDB 区画の特定のデータ・セットに対してだけ実行させることができます。

以下の表は、HALDB データベースで使用できるデータベース・ユーティリティーの全リストです。

表 *50. HALDB* データベースに対して実行できるユーティリティー

ユーティリティー	説明	コメント
DFSMAID0	HALDB マイグレーション・エイド	
DFSPREC0	HALDB 索引/ILDS 再作成	
DFSUPNT0	HALDB 区画データ・セット初期設定	
%DFSHALDB	HALDB 区画定義	c-list によるユーティリティーの呼び出し。 %DFSHALDB は、モジュール DSPXPDDU の TSO 呼び出し。
DFSURUL0	HISAM 再編成アンロード	
DFSURRL0	HISAM 再編成再ロード	

表 *50. HALDB* データベースに対して実行できるユーティリティー *(*続き*)*

ユーティリティー	説明	コメント
DFSURGU0	HD 再編成アンロード	PHDAM、PHIDAM、および PSINDEX に適用
DFSURGL0	HD 再編成再ロード	PHDAM、PHIDAM、および PSINDEX に適用
DFSURPR0	事前再編成	
DFSUDMP0	イメージ・コピー	
DFSUICP0	オンライン・イメージ・コピー	
DFSUDMT0	データベース・イメージ・コピー 2	
DFSUCUM0	変更累積	
DFSURDB0	データベース・リカバリー	
DFSBBO00	バッチ・バックアウト	

データベース入出力エラーの管理

データベース入出力エラーが発生した場合、IMS は仮想バッファーにエラー・ブロック制御インターバル (CI) のバッファー内容をコピーします。続いて DL/I 要求によりエラー・ブロック /CI は、バッファー・プールに読み戻されます。

書き込みエラー情報とバッファーは、都合のよい時期までリカバリー操作を延ばせるように、いくつかの再始動にわたって維持されます。入出力エラーの再試行は、データベースの終了時に自動的に実行されます。再試行操作が成功すると、エラー状態はそれ以上続かず、リカバリー操作も不要となります。

シスプレックス環境でデータベース入出力エラーが発生する場合、ローカル・システムはバッファーを維持し、データベースに登録済みインタレストをもつデータ共用グループの全メンバーに、その CI が使用不能であることを通知します。その CI に対するその後の DL/I 要求は、入出力エラーが残っている限り、障害を示す戻りコードを受け取ります。

エラー処理を動作させるためにデータベースを DBRC に登録する必要はありませんが、HALDB データベースについては登録が必要であり、その他のすべての全機能データベースについても登録が強く推奨されます。

統合 HALDB オンライン再編成機能は、共用システムでの HALDB 入出力エラーの解消に役立ちます。書き込みエラー EEQE を持つシステムでオンライン再編成を開始すると、オンライン再編成機能によってバッファーのローカル・コピーが作成され、出力データ・セットに書き込まれます。バッファーが出力データ・セットに書き込まれると、バッファー内の更新をすべての共用システムが再び使用できるようになります。

重要**:** DBRC を用いて登録されたデータベースでエラーが発生してシステムが停止した場合、システムを再始動して、データベースにアクセスする前に /DBR コマンドを発行しないと、データベースに損傷を与える場合があります。再始動によりエ

ラー・バッファーの内容がシステム停止時の状態に復元されるからです。バッチを実行している間、同じブロックが更新されていると、バッチの更新はオーバーレイされてしまいます。

第 14 章 高速機能データベース・タイプ

高速機能データベースには、高速処理データベース (DEDB) および主記憶データベース (MSDB) が含まれます。 DEDB は、大量データの効率的な保管と、大量データへの効率的なアクセスを提供します。 DEDB はまた、大量データの高水準の可用性も提供します。MSDB は、ご使用のシステムで最も頻繁に使用されるデータを保管し、そのデータへのアクセスができるようにします。

DEDB と MSDB は両方とも、データを保管するのに直接法を使用します。直接法では、直接アドレス・ポインターを各セグメントの接頭部に入れることにより、セグメントの階層順が維持されます。

各 IMS 環境は、次のように高速機能データベースをサポートしています。

- DB/DC は DEDB と MSDB の両方をサポートします。
- DBCTL は DEDB をサポートしますが、MSDB をサポートしません。
- DCCTL は DEDB も MSDB もサポートしません。

関連概念:

119 ページの『第 12 章 IMS データベースのタイプと機能の要約』

高速処理データベース

高速処理データベース (DEDB) は、大量データの効率的な保管と、大量データへの効率的なアクセスを提供します。 DEDB はまた、データの高水準の可用性も提供します。

DEDB のいくつかの特性により、詳細情報や要約情報を収集する必要がある場合にも DEDB が有用です。これらの特性とは、次のようなものです。

エリアのフォーマット

エリア・データ・セットの複製

レコードの非活動化

非リカバリー・オプション

DEDB は、最大 127 のセグメント・タイプが入っている階層データベースです。セグメント・タイプとして、ルート・セグメント、オプションの順次従属セグメント、および 0 から 126 までの直接従属セグメントがあります。オプションの順次従属セグメントが定義されている場合、直接従属セグメントは 125 までしか定義できません。 DEDB 構造は、最大 15 の階層レベルをもつことができます。 1 つのエリアに対する順次従属セグメントのインスタンスは、どのルート・セグメントに従属しているかには関係なく、発生順に保管されます。直接従属セグメントは、迅速な取り出しが可能になるように、階層の形で保管されます。

推奨事項: ETO 端末は端末関連 MSDB にアクセスできないため、新規の高速機能データベースは MSDB ではなく DEDB として構築するようにしてください。また、非端末関連キーを持つ非端末関連 MSDB がすでに存在する場合、それを VSO

DEDB に変換することを検討してください。それには、MSDB - DEDB 間変換ユーティリティーを使用することができます。

関連概念:

121 ページの『第 13 章 全機能データベース・タイプ』

123 ページの『パフォーマンス考慮点の概要』

17 ページの『セグメント』

関連資料:

MSDB から DEDB への変換ユーティリティー (DBFUCDB0) (データベース・ユーティリティー)

DEDB の機能

DEDB と MSDB には、多数の類似機能があります。

共通の機能には次のものがあります。

- 仮想記憶域
- フィールド (FLD) 呼び出し
- 固定長セグメント
- MSDB または DEDB コミット・ビュー

さらに、DEDB には以下の機能およびサポートがあります。

- DBRC の完全なサポート
- 使用可能エリアのブロック・レベル共用
  - DBCTL
  - LU 6.2 アプリケーション
  - DB/DC アプリケーション
- RSR トラッキング
- HSSP サポート
- DEDB ユーティリティー
- オンラインのデータベース保守
- INSERT 呼び出しと DELETE 呼び出しのサポートを含む全階層モデル
- ランダマイザー検索技法
- 副次索引サポート

DEDB エリア

DEDB は、エリアと呼ばれる 1 つ以上のデータ・セットに編成することができます。エリアは、DEDB の効率、容量、および柔軟性を向上させます。このトピックでは、DEDB エリアおよびその取り扱い方法を説明します。

エリアと DEDB フォーマット

DEDB は、エリアと呼ばれるデータ・セットを複数使用することができ、このそれぞれのエリアにはデータ構造全体が保管されます。

DEDB の物理的なフォーマットは、データがより簡単に使用可能になるようにしています。階層構造をもち、エリアを使用しない IMS データベースでは、論理データ構造がデータベース全体にわたって広がっています。複数データ・セットが使用されている場合には、データ構造はセグメント単位に分けられています。

DEDB の中の各エリアが、それぞれ 1 つの VSAM データ・セットです。 1 つの DEDB レコード (1 つのルート・セグメントとその従属セグメント) が、複数のエリアにまたがることはありません。 1 つの DEDB を、最高 2048 のこのようなエリアに分割することができます。このような編成はアプリケーション・プログラムからは見えません。

DEDB エリアの最大サイズは 4 GB です。データベースごとのエリアの最大数は 2048 であり、したがって、DEDB データベースの最大サイズは 8 796 093 020 160 バイト (8 TB) です。

IMS では、複数の DEDB データベースによって同時にオープンできるエリア・データ・セットの数に制限はありません。ただし、使用する IMS 構成と、ご使用のシステムで稼動している可能性があるその他の z/OS® サブシステム (例えば、Db2 for z/OS など) の両方によって、オープンできるエリア・データ・セットの数が潜在的に制限される場合があります。

エリア・データ・セットの場合、オープンするデータ・セットの数が非常に多いと制限を受ける可能性があるリソースの 1 つは、拡張共通サービス域 (ECSA) および拡張専用ストレージ (EPVT) のストレージです。

ランダム化モジュールを用いて、どのレコードがどのエリアに置かれるか決められます。エリアについてのこのような考え方により、大きいデータベースは、単一の VSAM データ・セットの上限である 2^{32} バイトを超えることができます。各エリアはそれぞれ独自のスペース管理パラメーターを持つことができます。エリアごとにメッセージの量が異なる場合があるので、メッセージ・ボリュームに応じてスペース管理パラメーターを選択することができます。 DEDB エリアを、複数の異なるタイプのボリュームに割り振ることができます。

DEDB の初期設定、再編成、およびリカバリーは、エリア単位で行われます。リソース割り振りは、制御インターバル (CI) のレベルで行われます。複数のプログラムを、オプションで 1 つのオンライン・ユーティリティーと共に使用するとき、各プログラムが異なる CI を使用している限り、すべてのプログラムがデータベース内の 1 つのエリアに同時にアクセスすることができます。 CI サイズとしては 512 バイト、1 K、2 K、4 K、そして 4 K の増分で 28 K までが可能です。メディア・マネージャーとデータ機能記憶管理サブシステム (DFSMS) の統合カタログ機能についてのカタログが必要です。

関連概念:

228 ページの『セグメント CI のエンキュー・レベル』

DEDB エリアのオープンと事前オープン

デフォルトでは、IMS は適格なアプリケーションがエリアにアクセスするまで、DEDB エリアをオープンしません。

これによって、始動時に不要なエリアがオープンされることはなくなりますが、DEDB エリアに最初にアクセスするアプリケーションには、付加的な処理オーバー

ヘッドがかかります。始動処理の直後に続く複数エリアに対する複数呼び出しは、この負担を大幅に増やすことがあります。

DEDB エリアを事前オープンすることにより、エリアをオープンするオーバーヘッドを制限することができます。また、アプリケーションが最も使用するエリアだけを事前オープンし、他のエリアはすべてアプリケーションが最初にアクセスするまでクローズしたままにしておくことで、このオーバーヘッドを始動処理とオンライン操作の間で分散させることもできます。

エリアの事前オープン状況は、DBRC コマンド INIT.DBDS または CHANGE.DBDS の PREOPEN および NOPREO パラメーターを使用して指定します。

デフォルトでは、IMS は始動処理時に事前オープン状況を割り当てられたすべての DEDB エリアを事前オープンします。ただし、始動処理時に多数の DEDB エリアを事前オープンすると、データ処理に遅延が発生することがあります。このような遅延を避けるために、始動処理の後でアプリケーション・プログラムの実行と非同期に、IMS に DEDB エリアを事前オープンさせることができます。この場合、アプリケーション・プログラムがある DEDB エリアにアクセスしようとした時点で IMS がまだそのエリアを事前オープンしていないと、IMS はその時点でその DEDB エリアをオープンします。この振る舞いは、IMS および DBC 始動プロシージャーの FPOPN= キーワードを使用して指定することができます。特に FPOPN=P を指定すると、IMS は始動後にアプリケーション・プログラムの実行と非同期で DEDB エリアを事前オープンします。

FPOPN= キーワードは、IMS が正常再始動 (/NRE) および緊急時再始動 (/ERE) の両方の場合にどのように DEDB エリアを再オープンするかを決定します。

DEDB エリアは、OPTION(OPEN) キーワードを指定した以下のタイプ 2 コマンドのいずれかを実行する方法でもオープンできます。

- UPDATE AREA NAME(*areaname*) START(ACCESS) OPTION(OPEN)
- UPDATE DB NAME(*dedbname*) AREA(*) START(ACCESS) OPTION(OPEN)

注**:** OPTION(OPEN) プロセスは、UPDATE AREA コマンドまたは UPDATE DB コマンドのログには記録されません。このオプションを使用した後で IMS を再始動した場合、IMS はこれらの UPDATE コマンドで以前にオープンされた DEDB エリアを再オープンしません。

関連資料:

IMS プロシージャーのパラメーターの説明 (システム定義)

緊急時再始動時の **DEDB** エリアの再オープン**:**

緊急時再始動時に IMS がどのように DEDB エリアを再オープンするかは、IMS プロシージャーまたは DBC プロシージャーの FPOPN= キーワードを使用して指定することができます。

次のリストは、FPOPN= キーワードが緊急時再始動時の DEDB エリアの再オープンにどのように影響するかを示したものです。

FPOPN=N
始動処理中に、IMS は事前オープン状況をもつエリアのみをオープンします。これはデフォルトです。

FPOPN=P
始動処理の完了後、アプリケーション処理の再開と非同期で、IMS は事前オープン状況をもつエリアのみをオープンします。

FPOPN=R
始動処理の完了後、アプリケーション処理の再開と非同期で、IMS は異常終了以前にオープンしていたエリアのみをオープンします。異常終了の発生時にクローズしていたすべての DEDB エリアは、事前オープン状況をもつ DEDB エリアも含め、IMS の再始動時にクローズされたままになります。

FPOPN=D
事前オープン処理を抑止します。事前オープン状況をもつ DEDB エリアは、アプリケーション・プログラムによって最初にアクセスされるまで、または /START AREA コマンドを使用して手動でオープンされるまで、事前オープンされずにクローズされたままになります。

FPOPN=D は、DBRC コマンド INIT.DBDS および CHANGE.DBDS の PREOPEN パラメーターによって設定された DEDB エリアの事前オープン状況をオーバーライドしますが、変更することはありません。

関連概念:

260 ページの『緊急時再始動処理』

関連資料:

- IMS プロシージャーのパラメーターの説明 (システム定義)
- DBC プロシージャー (システム定義)
- IMS プロシージャー (システム定義)

DEDB および DEDB エリアの停止

適切なコマンドを実行することにより、DEDB へのアクセスや DEDB に対するアプリケーション・プログラムのスケジューリングを、データベース・レベルまたはエリア・レベルで停止できます。

データベース・レベルのコマンドには、タイプ 1 コマンド /STOP DB と /DBRECOVERY DB、およびタイプ 2 コマンド UPDATE DB STOP(ACCESS|SCHD) があります。

エリア・レベルのコマンドには、タイプ 1 コマンド /STOP AREA と /DBRECOVERY AREA、およびタイプ 2 コマンド UPDATE AREA STOP(ACCESS|SCHD) があります。

タイプ 1 コマンド /STOP DB とタイプ 2 コマンド UPDATE DB STOP(SCHD) には、DEDB に対する新しいアプリケーション・プログラムのスケジューリングを停止するという同等の効果があります。コマンド /DBRECOVERY DB と UPDATE DB STOP(ACCESS) は、DEDB に対するアクセスを全面的に停止します。エリア・レベルのタイプ 1 コマンドとタイプ 2 コマンドにも、同じような同等の効果があります。

DEDB および DEDB エリアの開始

DEDB へのアクセスまたは DEDB に対するアプリケーション・プログラムのスケジューリングは、データベース・レベルまたはエリア・レベルで開始することができます。

データベース・レベルのコマンドには、タイプ 1 コマンド /START DB およびタイプ 2 コマンド UPDATE DB START(ACCESS) があります。

エリア・レベルのコマンドには、タイプ 1 コマンド /START AREA およびタイプ 2 コマンド UPDATE AREA START(ACCESS) があります。 /START AREA コマンドでは、PREOPEN または PRELOAD エリアとして指定しない限り、エリアがオープンされません。

タイプ 2 コマンド UPDATE DB START(ACCESS) の AREA(*) パラメーターを使用すると、DEDB の全エリアを一度に開始できます。タイプ 1 コマンド /DBRECOVERY DB またはタイプ 2 コマンド UPDATE DB STOP(ACCESS) を実行して DEDB へのアクセスをデータベース・レベルで停止した場合は、AREA(*) パラメーターが便利です。エリア名を指定しても (例えば AREA(*area_name*)) 無効となるので注意してください。

AREA(*) パラメーターをタイプ 2 コマンド UPDATE DB START(ACCESS) の SET(ACCESS) パラメーターと併用して、全エリアを一斉に開始し、DEDB のアクセス・タイプを同時に変更することができます。

DEDB エリアを開始するときにそのエリアをオープンすることもできます。そのためには、タイプ 2 コマンド UPDATE AREA START(ACCESS) または UPDATE DB START(ACCESS) で OPTION(OPEN) キーワードを指定します。

IRLM 障害後のエリアの再始動と再オープン

内部リソース・ロック・マネージャー (IRLM) は、データ共用環境におけるデータベースの保全性を確立します。

IRLM 障害が発生すると、DEDB エリア内のデータの保全性が損なわれるのを防ぐために、障害の発生した IRLM の制御下にあるすべての DEDB エリアが停止します。ユーザーは IRLM の障害を正して IMS システムに再接続した後、その IRLM が制御している DEDB エリアを再始動し、再オープンしなければなりません。

IMS および DBC プロシージャーの FPRLM= キーワードを使用して、IRLM の再接続後に IMS がどのように DEDB エリアを再始動および再オープンするかを、指定することができます。

関連概念:

- データベース・レベル共用における IRLM の使用 (システム管理)
- IRLM を伴うリカバリー (システム管理)
- IMS の障害後の再始動 (システム管理)
- IRLM 障害 (オペレーションおよびオートメーション)

関連資料:

- DBC プロシージャー (システム定義)
- IMS プロシージャーのパラメーターの説明 (システム定義)
- IMS プロシージャー (システム定義)

DEDB エリアの読み取りエラーと書き込みエラー

このトピックでは、DEDB エリアで発生する読み取りエラーと書き込みエラーを IMS がどのように処理するかについて説明します。

読み取りエラー:

あるエリアにおいて読み取りエラーが検出されると、当該アプリケーション・プログラムは AO 状況コードを受け取ります。

エラー・キュー・エレメント (EQE) は作成されますが、2 番目の CI に書き込まれることも、データ共用環境の共用システムに送られることもありません。アプリケーション・プログラムはそのエリアへのアクセスを続けることができ、エラーのある CI へのアクセスだけができなくなります。 4 つの異なる CI の読み取りエラーの後では、エリア・データ・セット (ADS) は停止します。読み取りエラーは連続していなければなりません。つまり、もし書き込みエラーが介在していると、読み取り EQE カウントはクリアされます。このような読み取りエラー処理は、多重エリア・データ・セット (MADS) 環境にのみ適用されます。

書き込みエラー:

あるエリアにおいて書き込みエラーが検出されると、EQE が作成され、アプリケーション・プログラムはその EQE の数が 11 になるまで該当エリアにアクセスすることができます。

データベースの一部が使用可能でない場合でも (1 つ以上のエリアが停止している場合)、このデータベースは論理的には依然として使用可能であり、したがって、このデータベースを使用するトランザクションはスケジュールされます。複数のデータ・セットがエリアを構成している場合には、データの 1 つのコピーが常時使用可能になっている可能性が高くなります。

DEDB がリカバリー不能の場合は、書き込みエラーは、リカバリー可能 DEDB の場合とは異なる方法で処理されます。エリアに書き込みエラーがあると、EQE が作成されます。エリア当たり 10 個の EQE があると、DBRC はそれに「リカバリーが必要」のマークを付け、IMS はそのエリアを停止します。エリアが共用されている場合は、共用グループ内のすべての IMS システムに通知され、これらのシステムもそのエリアを停止します。 DEDB に「リカバリーが必要」のマークが付けられているときは、その DEDB を復元する (例えば、イメージ・コピーから) 必要があります。このリカバリー手順を運用手順に組み込んでください。

MADS を使用している DEDB に書き込みエラーが発生すると、書き込みエラーのある ADS に対して EQE が作成されます。この環境では、EQE の数が最大の 10 に達したときに、その ADS は停止されます。

単一の ADS を使用しているリカバリー可能 DEDB エリアに書き込みエラーが発生すると、IMS は入出力許容 (IOT) 処理を呼び出します。 IMS は ECSA に仮想

バッファーを割り振り、エラーのある制御インターバルを高速機能共通バッファーから仮想バッファーにコピーします。 IMS は、仮想バッファーの作成を、X'26' ログ・レコードによって記録します。データベースが DBRC に登録されている場合、Extended Error Queue Element (EEQE) が作成されて DBRC に登録されます。この EEQE は、その制御インターバルがエラーであることを示します。 IRLM を使用するデータ共用環境では、すべての共用パートナーに EEQE の作成が通知されます。

許容されているデータは、EEQE を作成した IMS システムで使用可能です。共用パートナーがその CI を要求すると、データは使用不能のため、'AO' 状況コードを受け取ります。許容されている制御インターバルに要求が出されると、データは仮想バッファーから共通バッファーにコピーされます。そのデータが更新されると、そのデータは仮想バッファーへコピーされて戻されます。その更新について、標準の X'5950' ログ・レコードが生成されます。

書き込みエラーはすべて、エリアごとに EEQE によって表わされます。 EEQE は DBRC によって更新され、そして IMS ログに X'26' ログ・レコードとして記録されます。1 つのエリアに存在可能な EEQE の数に、論理的な制限はありません。保守可能な EEQE の数に、DBRC や ECSA での物理的なストレージの制限はあります。この制限はご使用のシステムに依存します。 DBRC または ECSA を使用し過ぎないことを確実にするために、DEDB には、限られた数の EEQE しか認められません。制限値は 100 です。 1 つのエリアについて 100 個の EEQE が作成された後で、そのエリアは停止されます。

システム・チェックポイント、/STO、および /VUN コマンドの間、IMS はエラーのある CI を書き戻そうと試みます。書き込みが成功すれば、EEQE は除去されます。書き込みが失敗した場合、EEQE は残ります。

関連概念:

219 ページの『非リカバリー・オプション』

レコードの非活動化

アプリケーション・プログラムが DEDB を更新している間にエラーが起きた場合には、データベースあるいはエリアさえも停止する必要はありません。 IMS は、アプリケーション・プログラムが続けてそのエリアにアクセスできるようにします。

IMS は、エラーのある制御インターバル (CI) に対して EQE を作成することで、アプリケーション・プログラムがそのエラー CI にアクセスできないようにするだけです。エリアの複数コピーがある場合には、そのデータのコピーの 1 つは常に使用可能になっている可能性が高いといえます。同じ制御インターバルでエリアのすべてのコピーにエラーが発生する可能性はまずありません。エリア・データ・セットのエラー数が 11 に達すると、IMS は自動的にそのエリア・データ・セットを使用不能にします。

レコードの非活動化によって次のような方法でデータベース障害の影響を最小限にとどめ、データへのエラーを少なくします。

- あるエリア・データ・セットの複数のコピーを使用している場合には、アプリケーション・プログラムが該当エリアを更新している間にエラーが起きても、エラーをただちに訂正する必要はありません。このとき他のアプリケーション・プロ

グラムは、該当エリアの他の使用可能なコピーを使用してそのエリア内のデータに引き続きアクセスすることができます。

- あるエリアのコピーに複数の入出力エラーがある場合には、DEDB エリア・データ・セット作成ユーティリティーを使用してそのエリアの既存コピーから新しいコピーを作成することができます。この後で、エラーのあるコピーを破棄することができます。

非リカバリー・オプション

VSO または非 VSO の DEDB をリカバリー不能として指定すれば、オンライン・パフォーマンスを向上させ、DEDB のデータベース変更ロギングを削減することができます。

IMS は、リカバリー不能 DEDB からの変更をログに記録することも、DBRC RECON データ・セット内の更新を保持することもしません。リカバリー不能 DEDB では、すべてのエリアがリカバリー不能です。

全機能のリカバリー不能データベースはバックアウトをサポートしますが、これとは異なり、リカバリー不能 DEDB は本当にリカバリー不能であり、再始動時または XRF テークオーバー時に REDO (再実行) できません。 IMS はエリアごとに 1 回ずつ同期点で単一ログ・レコード X'5951' を書き込んで、リカバリー不能抑止が行われたことを示します。

緊急時再始動時または XRF テークオーバー時には、X'5951' ログと DMAC フラグによってエリアの保全性が判断されます。再始動時または XRF テークオーバー時に、リカバリー不能な DEDB 書き込みエラーが発生する可能性があります。XRF テークオーバー時または緊急時再始動時にリカバリー不能 DEDB でエラーが検出された場合、メッセージ DFS3711W が出され、DEDB は停止されません。

リカバリー不能 DEDB は DBRC に登録しなければなりません。 DEDB をリカバリー不能として定義するには、コマンド `INIT.DB DBD() TYPEFP NONRECOV` を使用してください。デフォルトは、リカバリー可能 DEDB を表す RECOVABL です。

DEDB のリカバリー可能性を変更する前に、`/STOP DB`、`/STO AREA`、または `/DBR DB` コマンドを出します。リカバリー可能 DEDB をリカバリー不能 DEDB に変更するには、DBRC コマンド `CHANGE.DB DBD() NONRECOV` を使用します。リカバリー不能 DEDB をリカバリー可能 DEDB に変更するには、コマンド `CHANGE.DB DBD() RECOVABL` を使用します。

リカバリー不能 DEDB を復元するには、`GENJCL.RECOV RESTORE` コマンドを使用します。リカバリー・ユーティリティーはデータベースを最後のイメージ・コピーに復元します。 DEDB がリカバリー可能 DEDB からリカバリー不能 DEDB に変更されていた場合、リカバリー・ユーティリティーは、この変更が行われた時点までのログからのすべての更新を適用します (リカバリー不能への変更の後でイメージ・コピーが作成されなかった場合)。

関連概念:

217 ページの『書き込みエラー』

702 ページの『高速機能ログの縮小』

エリア・データ・セットの複製

データ・セットは最高 7 回までコピー、すなわち複製できるので、アプリケーション・プログラムの可用性は高くなります。

DEDB エリア・データ・セット作成ユーティリティー (DBFUMRI0) によって、エリアを停止することなくそのエリアのデータ・セットのコピーを 1 つ以上作成できます。あるエリアのデータ・セットのコピーはすべて同一の CI サイズとスペースをもっていることが必要ですが、別々の装置に置くことができます。このユーティリティーは現在のコピーをすべて使用して新規データ・セットを完成させますが、特定のレコードに対して入出力エラーを検出すると別のコピーに進みます。このようにして、使用可能なデータの集合から、変更のないコピーを構成することができます。新規データ・セットへの現時点での更新がただちに有効になります。

作成ユーティリティーは、ジョブ制御言語 (JCL) で指定されたとおりに、別の装置上に新しいコピーを作成することができます。ご使用のシステムでデータを別のストレージ・デバイスにマイグレーションしたい場合には、オンライン・システムを実行中にこの処理を実行し、そのデータを現行の状態に保つことができます。

DEDB のコピーすべてを同一にするため、IMS は、1 つのコピーにしか変更が加えられていないときでも、すべてのコピーを更新します。

ADS が、複数データ・セット (MADS) を持つ DEDB の通常のオープン処理中にオープンに失敗した場合、ADS のコピーの割り振りができないため、そのエリアは停止します。しかし、緊急時再始動している間にオープンの障害が発生すると、失敗した ADS だけが割り振られず停止します。 ADS の他のコピーは、使用可能な状態になっています。

DEDB とデータ共用

DEDB には異なるレベルのデータ共用を指定することができます。 1 つの DEDB に対する指定は、その DEDB 内のすべてのエリアに適用されます。

DEDB でデータ共用を許可しないよう指定すると、同時に 1 つの IMS システムのみが 1 つの DEDB エリアにアクセスすることができます。ただしその場合も、他の IMS システムは、その DEDB 内の他のエリアにアクセスすることができます。

DEDB でデータ共用を許可するよう指定すると、同時に複数の IMS システムが同じ DEDB エリアにアクセスすることができます。単一 DEDB エリアの共用は、全機能データベースのブロック・レベル共用と同等です。

DEDB で許可するデータ共用のレベルは、DBRC コマンド `INIT.DB` および `CHANGE.DB` の SHARELVL パラメーターを使用して指定することができます。すでにそのデータベースを許可している IMS がある場合には、SHARELVL を変更してもそのデータベース・レコードは変更されません。 SHARELVL パラメーターは、DEDB の中のすべてのエリアに適用されます。

DEDB エリアの共用は、DASD から直接行うことも、仮想記憶オプション (VSO) を使用してカップリング・ファシリティー構造から行うこともできます。

関連概念:

242 ページの『高速機能仮想記憶オプション』

247 ページの『VSO DEDB エリアの共用』

IMS 環境でのデータ共用 (システム管理)

関連資料:

DBRC コマンド (コマンド)

DEDB での固定長セグメントと可変長セグメント

DEDB は、固定長セグメントをサポートします。したがって、DEDB には固定長セグメントも可変長セグメントも定義できます。このサポートにより、MSDB アプリケーションを DEDB 用として使用することができます。

固定長セグメントを定義するには、DBDGEN 時に SEGM マクロの BYTES= パラメーターに値を 1 つ指定します。可変長セグメントを定義するには、DBDGEN 時に SEGM マクロの BYTES= パラメーターに値を 2 つ指定します。

固定長セグメント DEDB (MSDB と同様) 用のアプリケーション・プログラムは、各セグメントの先頭にある長さ (LL) フィールドを調べません。可変長セグメントの DEDB 用のアプリケーション・プログラムは、各セグメントの先頭にある長さ (LL) フィールドを調べるので、セグメントを適切に処理するにはこのフィールドを使用する必要があります。

REPL 呼び出しと ISRT 呼び出しを使用する固定長セグメントのアプリケーション・プログラムでは、長さ (LL) フィールドを省略できます。

セグメントの定義例

以下の例では、BYTES= パラメーターを使用して可変長セグメントまたは固定長セグメントを定義する例を示しています。

可変長セグメントの定義

```
ROOTSEG  SEGM  NAME=ROOTSEG1,                                  C
               PARENT=0,                                       C
               BYTES=(390,20)
```

固定長セグメントの定義

```
ROOTSEG  SEGM  NAME=ROOTSEG1,                                  C
               PARENT=0,                                       C
               BYTES=(320)
```

DEDB エリアの部分

DEDB エリアは、以下の 3 つの部から構成されます。

各部分は次のとおりです。

- ルート・アドレス可能部
- 独立オーバーフロー部
- 順次従属部

以下の図は、DEDB エリアのこれらの部を示しています。

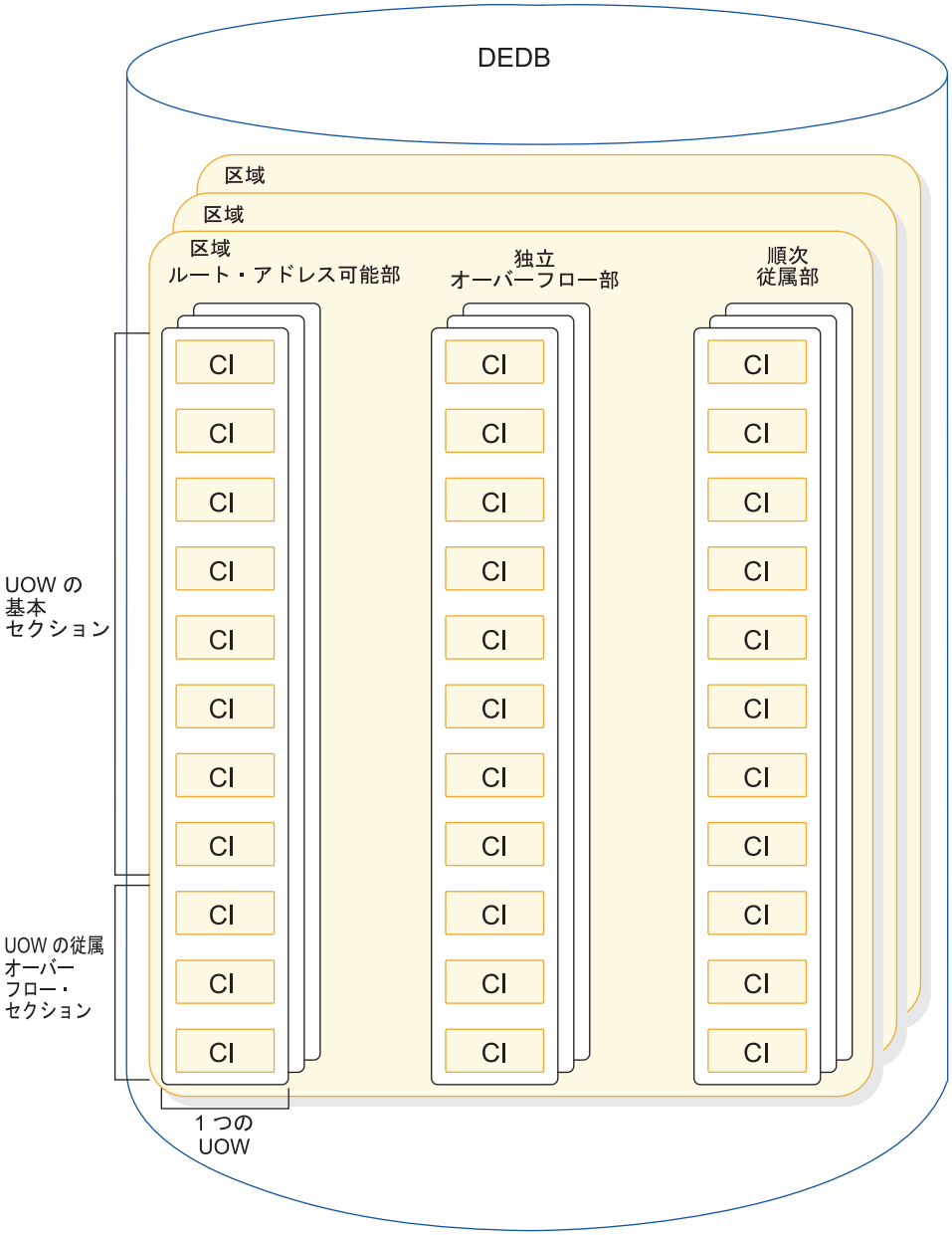


図 *58.* ストレージでの *DEDB* エリアの各部

DEDB データ・セットが DEDB 初期設定ユーティリティー (DBFUMIN0) によって初期設定されると、内部用に追加の CI が作成されます。そのため、DEDB エリアには実際には前の図よりも多くの CI が格納されます。このような余分な CI は、再編成 UOW と呼ばれます。IMS はこれらの余分な CI を使用しませんが、DBFUMIN0 は互換性を保つ目的でそれらを作成します。

ルート・アドレス可能部

ルート・アドレス可能部は、作業単位 (UOW) に分割されます。 UOW は、スペース割り振りの基本エレメントです。

UOW は、ユーザーによって指定された数の物理的隣接した CI で構成されています。

ルート・アドレス可能部の中の各 UOW は、さらに 1 つの基本セクションと 1 つのオーバーフロー・セクションに分割されています。基本セクションには、UOW の諸 CI のうち、ランダム化モジュールによってアドレスされる CI が入っており、これに対して、UOW のオーバーフロー・セクションは、この UOW の中の CI の論理的な拡張部分として使用されます。

ルート・セグメントと直接従属セグメントは、基本セクションに保管されます。基本セクション満杯状態が存在する場合には、この両方のセグメントをオーバーフロー・セクションに保管することができます。

独立オーバーフロー部

独立オーバーフロー部には、そのエリアの中のどの UOW でも使用できる空 (から) の CI が入っています。

いったんある UOW が独立オーバーフロー部からある CI を獲得すると、この CI を使用できるのはこの UOW だけです。独立オーバーフロー部の中の CI は、これがある UOW に割り振られると、ただちにルート・アドレス可能部の中のオーバーフロー・セクションの延長部分と見なすことができます。独立オーバーフロー部の中の CI はある特定の UOW に割り振られたままの状態が続き、再編成の後、もはやこの CI が必要でなくなったときに初めて、この CI は解放されます。

順次従属部

順次従属部には、このエリアのすべての UOW の中のルート・セグメントに従属する順次従属セグメントが保持されます。

順次従属セグメントは、ルート・セグメントあるいはルート・セグメントが入っている UOW と関係なく、発生順に保管されます。順次従属部が満杯になると、これは先頭から再使用されます。しかし、順次従属部を再使用できるようにするためには、その前に DEDB 順次従属削除ユーティリティー (DBFUMDL0) を使用して、隣接部分、すなわち順次従属部の中のすべての順次従属セグメントを削除しておく必要があります。

CI とセグメントのフォーマット

DEDB 制御インターバル (CI) およびセグメントのフォーマットを、以下の表と図に示します。

このトピックには診断、変更、およびチューニングに関する情報が含まれています。

以下の一連の図は、次のフォーマットを示しています。

- CI のフォーマット
- ルート・セグメントのフォーマット
- 順次従属セグメントのフォーマット
- 直接従属セグメントのフォーマット

各図に続く表では、CI およびセグメントのセクションを、図に示されている順に説明します。

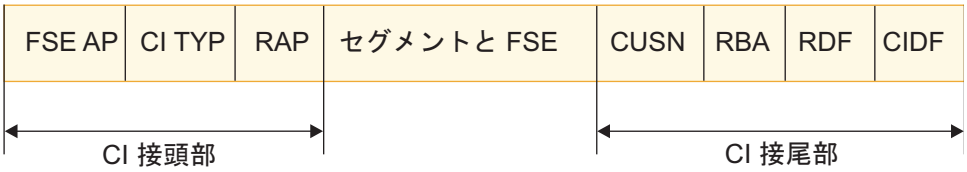


図 *59. CI* のフォーマット

表 *51. CI* のフォーマット

CI セクション	バイト数	説明
FSE AP	2 バイト	最初のフリー・スペース・エレメントまでのオフセット。順次従属部の中に CI がある場合は、この 2 バイトは使用されません。
CI TYP	2 バイト	この CI の使用と次の 4 バイトの意味について説明します。
RAP	4 バイト	この CI がルート・アドレス可能域の基本セクションに属する場合のルート・アンカー・ポイント。ランダム化の結果、この CI にランダム化されるルート・セグメントはすべて、この RAP からキーの昇順にチェーニングされます。 RAP は CI 当たり 1 つしかありません。 重要: 従属オーバーフロー部と独立オーバーフロー部においては、この 4 バイトは、高速機能制御情報によって使用されます。順次従属部の CI に RAP はありません。
CUSN	2 バイト	CI 更新シーケンス番号 (CUSN)。各 CI の中で維持されているシーケンス番号。この番号は、同期化処理中に、特定の CI の更新ごとに増加します。
RBA	4 バイト	この CI の相対バイト・アドレス。
RDF	3 バイト	レコード定義フィールド (VSAM 制御情報が入っている)。
CIDF	4 バイト	CI 定義フィールド (VSAM 制御情報が入っている)。

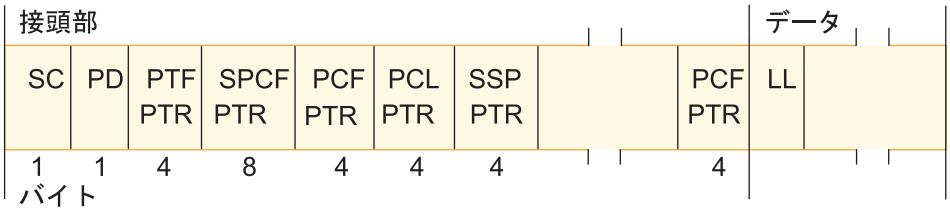


図 *60.* ルート・セグメントのフォーマット *(*サブセット・ポインターがある順次従属セグメントと直接従属セグメントを持つ*)*

表 *52.* ルート・セグメントのフォーマット

セグメント・セクション	バイト数	説明
SC	1 バイト	セグメント・コード。
PD	1 バイト	接頭部記述子。

表 52. ルート・セグメントのフォーマット (続き)

セグメント・セクション	バイト数	説明
PTF	4 バイト	物理兄弟順方向ポインター。キー・シーケンスで次のルート・セグメントの RBA が入っています。
SPCF	8 バイト	順次第 1 物理子ポインター。このルート・セグメントの下に最後に挿入された順次依存従属セグメントのサイクル・カウントと RBA が入っています。順次従属セグメントが定義されていない場合は、このポインターはありません。
PCF	4 バイト	物理第 1 子ポインター。 PCF は、直接従属セグメント・タイプの最初のオカレンスを指します。 PCF ポインターの数は最高 126 であり、順次従属セグメントがあれば、PCF ポインターは 125 です。直接従属セグメントが定義されていない場合には、この PCF ポインターはありません。
PCL	4 バイト	物理最終子ポインター。 PCL はセグメント・タイプの最後の物理子を指しているオプション・ポインターです。 直接従属セグメントが定義されていない場合は、このポインターはありません。
SSP	4 バイト	サブセット・ポインター。親の子タイプごとに、オプションのサブセット・ポインターは最高 8 つまで存在できます。
LL	2 バイト	このセグメントの可変長。

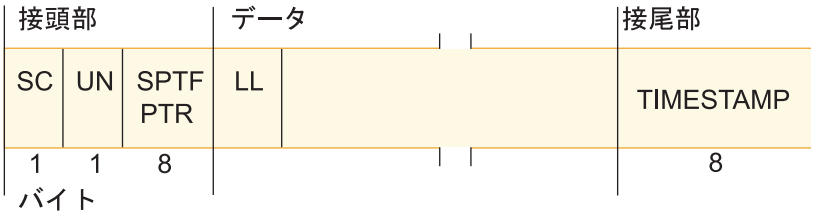


図 61. 順次従属セグメントのフォーマット

表 53. 順次従属セグメントのフォーマット

セグメント・セクション	バイト数	説明
SC	1 バイト	セグメント・コード。
UN	1 バイト	接頭部記述子。
SPTF	8 バイト	順次物理兄弟順方向ポインター。 同じルート・セグメントの下にある直前の順次従属セグメントのサイクル・カウントと RBA が入っています。
LL	2 バイト	このセグメントの可変長。

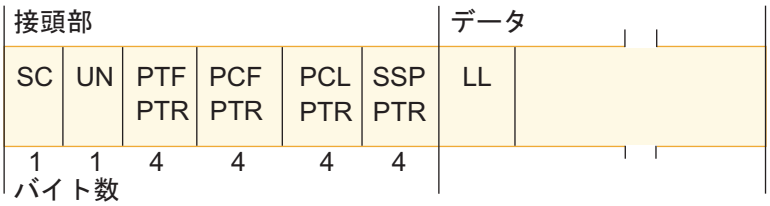


図 *62.* 直接従属セグメントのフォーマット

表 *54.* 直接従属セグメントのフォーマット

セグメント・セクション	バイト数	説明
SC	1 バイト	セグメント・コード。
UN	1 バイト	使用されません。
PTF	4 バイト	物理兄弟順方向ポインター。この直接従属セグメント・タイプの次のオカレンスの RBA が入っています。
PCF	4 バイト	物理第 1 子ポインター。 PCF は、直接従属セグメント・タイプの最初のオカレンスを指します。直接従属セグメントにおける PCF ポインターの数は最高 125 であり、順次従属セグメントがあれば、PCF ポインターは最高 124 です。直接従属セグメントが定義されていない場合には、この PCF ポインターはありません。
PCL	4 バイト	物理最終子ポインター。 PCL はセグメント・タイプの最後の物理子を指しているオプション・ポインターです。 直接従属セグメントが定義されていない場合は、このポインターはありません。
SSP	4 バイト	サブセット・ポインター。親の子タイプごとに、オプションのサブセット・ポインターは最高 8 つまで存在できます。
LL	2 バイト	このセグメントの可変長。

関連概念:

230 ページの『DEDB 挿入アルゴリズム』

ルート・セグメントの保管

DEDB ルート・セグメントは、ランダム化ルーチンの規定に従って保管され、各アンカー・ポイントからキーの昇順にチェーニングされます。

エリア内の UOW の基本セクションの中の各 CI はそれぞれ 1 つのアンカー・ポイントを有しています。 GN 呼び出しを使用する順次処理は、ルート・セグメントを以下の順序で処理します。

1. エリア番号の昇順
2. UOW の昇順
3. 各アンカー・ポイント・チェーンでのキーの昇順

各ルート・セグメントには、キーの昇順に、次のルート・セグメントの RBA が入っている PTF ポインターが収容されています。

関連資料:

高速処理データベース・ランダム化ルーチン (DBFHDC40 / DBFHDC20 DBFHDC44 / DBFHDC24 DBFHDC2S) のサンプル (出口ルーチン)

直接従属セグメントの保管

DEDB は、直接従属セグメント・タイプによって階層物理構造をサポートすると同時に、処理効率を維持します。

最大 127 のセグメント・タイプがサポートされます (直接従属セグメント・タイプは 126 まで、ただし順次従属セグメントがある場合には、125)。

直接従属 (DDEP) セグメント・タイプは、階層構造を活用して効率よく取り出すことができ、ユーザーは、セグメントに対する完全なオンライン処理の制御を行えます。サポートされる処理オプションは、挿入、取得、削除、および置き換えです。ユーザーは、置き換え機能を用いて、セグメントの長さを変えることがきます。DEDB スペース管理論理は、挿入される直接従属セグメントを、そのルート・セグメントが入っている CI と同じ CI に保管しようとします。その CI に十分な使用可能スペースがない場合には、このエリアのルート・アドレス可能オーバーフロー部の中を探し、次に独立オーバーフロー部の中を探します。

DDEP セグメントを定義するとき、固有のシーケンス・フィールドをこのセグメントに与えることもできるし、これを与えなくてもかまいません。直接従属セグメントは、キーの昇順に保管されます。

直接従属セグメントの物理的なチェーニングは、定義されているそれぞれの従属セグメント・タイプごとに 1 つずつ親セグメントの中にある物理第 1 子ポインター (PCF) と、各従属セグメントの中に 1 つずつある物理兄弟順方向ポインター (PTF) で構成されています。

DEDB により、物理最終子 (PCL) ポインターを使用できるようになります。このポインターによって、セグメント・タイプの物理親から直接、物理最終子ポインターにアクセスすることができます。挿入規則の LAST により、長くなる可能性のある物理子ポインター・チェーンに従う必要はなくなります。

サブセット・ポインターは、同じ親の下で発生したセグメントのチェーンを複数のグループのサブセットに分ける方法です。任意のセグメント・タイプに対して最高 8 つのサブセット・ポインターを定義することができ、そのチェーンを最高 9 つのサブセットに分けることができます。 各サブセット・ポインターは、新しいサブセットの開始を指しています。

関連概念:

サブセット・ポインター・コマンド・コードを使用した高速機能 DEDB の処理 (アプリケーション・プログラミング)

サブセット・ポインター・オプションを使用した高速機能 DEDB の処理 (アプリケーション・プログラミング)

順次従属セグメントの保管

DEDB の順次従属 (SDEP) セグメントは、エリアの順次従属部に入力順に保管されます。

1 つのエリアに属する異なったルート・セグメントからチェーニングされている複数の SDEP セグメントは、どのルート・セグメントが親であるかには関係なく、このエリアの順次従属部の中に混じり合っています。 SDEP セグメントは、迅速な挿入ができるようにするために特に設計されているものです。しかし、オンライン検索はそれほど効率はよくありませんが、これは入力操作が増えることもあるからです。

すべての SDEP 従属セグメントが 1 つのルート・セグメントからチェーニングされている場合には、「親における後続セグメントの取り出し」呼び出しによる処理を行うと、逆方向順次の順序になります。 (アプリケーションによっては、この処理方法を使用することができるものがあります。) 通常 SDEP セグメントは、DEDB 順次従属スキャン・ユーティリティー (DBFUMSC0) の使用時に限り順次に検索されます。このユーティリティーについては、「*IMS V13* データベース・ユーティリティー」で説明しています。次に、SDEP セグメントは、オフライン・ジョブによって処理されます。

SDEP セグメントはデータ収集アプリケーション、ジャーナリング・アプリケーション、および監査アプリケーションのために使用されます。

セグメント CI のエンキュー・レベル

CI の割り振りには、異なる 3 つのエンキュー・レベルがあります。

各エンキュー・レベルについて以下に説明します。

- NO ENQ レベル。SDEP CI では典型的なものです。 SDEP セグメントを更新することはできません。したがって、すべての領域がそれらに同時にアクセスし、それらを共用することができます。
- SHARED レベル。更新を行わない複数のトランザクションの間で CI を共用することができることを意味しています。 SHARED レベルにある CI は、更新を行うトランザクションからの要求を遅らせます。
- EXCLUSIVE レベル。競合しているものが同じリソースを取得するのを防止します。

ROOT セグメントと SDEP セグメントの CI に当初どのエンキューのレベルが取得されるのかは、トランザクションの意図によって異なります。意図が更新であるならば、取得される CI は (SDEP CI を除き) すべて、EXCLUSIVE レベルに保持されます。意図が更新でない場合には、デッドロックの可能性があっても、それらの CI は SHARED レベルに保持されます。

バッファー・スチール機能が呼び出されるたびに、前述したように、エンキューのレベルが再レビューされます。このバッファー・スチール機能は、すでに割り振られた各バッファー (と CI) を検査し、このエンキューのレベルを更新します。

エンキューされている他の CI はすべて解放されるので、他の領域はこれらの CI を割り振ることができます。

以下の図は、DEDB 構造についての例を示しています。

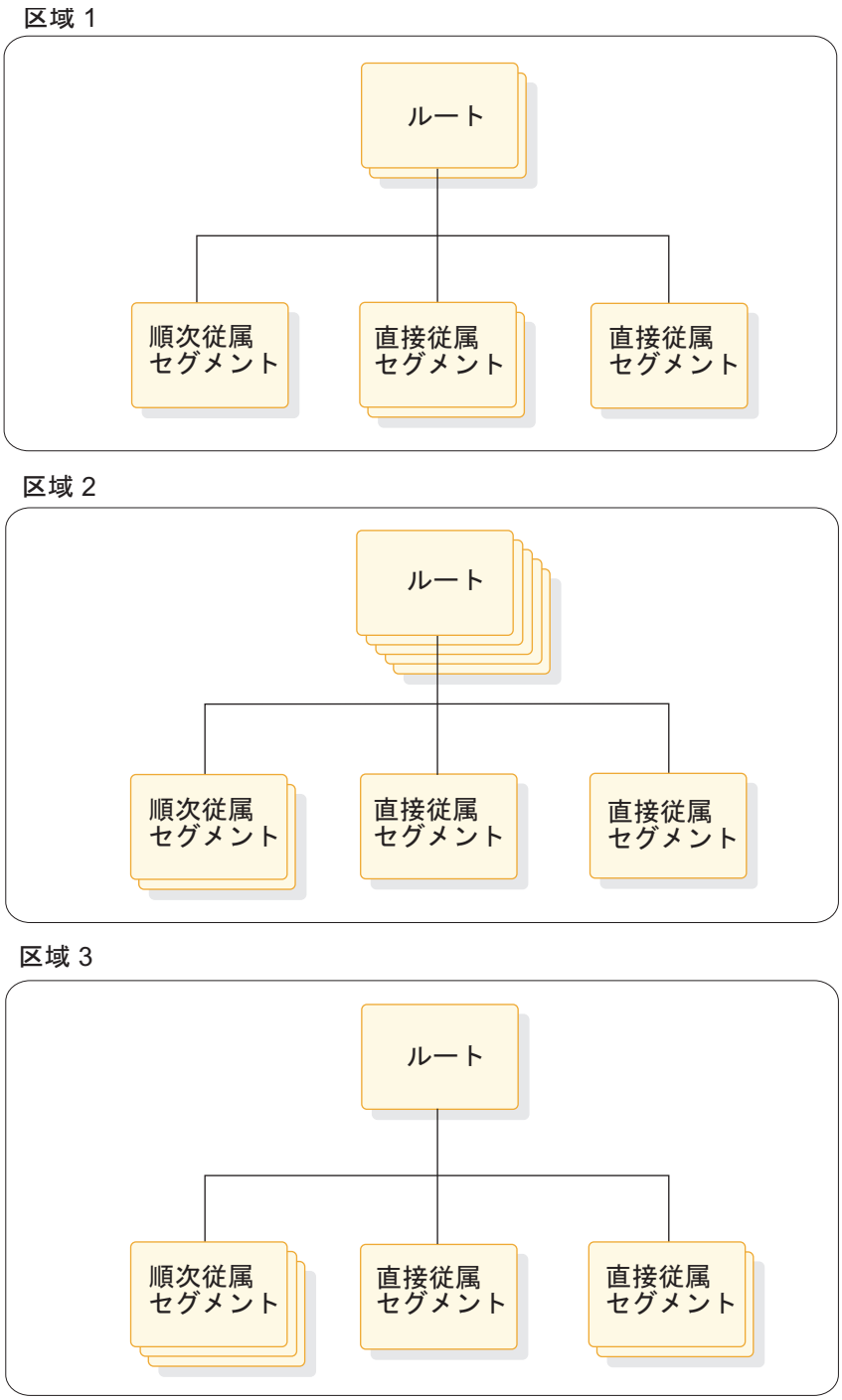


図 *63. DEDB* 構造の例

関連概念:

546 ページの『高速機能バッファー割り振りアルゴリズム』

212 ページの『エリアと DEDB フォーマット』

DEDB スペース検索アルゴリズム

DEDB スペース検索アルゴリズムは、多数の RAP およびオーバーフロー CI にまたがってデータベース・レコードのセグメントを分散させるのではなく、最小の量の CI にデータを保管しようとします。

このトピックには診断、変更、およびチューニングに関する情報が含まれています。

トレードオフは、将来のデータベース・レコードのアクセスの改良されたパフォーマンスと、最適なスペースの使用効率との間の比較考量です。

セグメントを DEDB に挿入する一般規則は、HD データベースに挿入する場合と同じです。この規則では、そのセグメント (ルート・セグメントと直接従属セグメント) を、最も望ましいブロックに保管します。

ルート・セグメントの場合、最も望ましいブロックは、RAP CI です。直接従属セグメントの場合は、最も望ましいブロックはルート・セグメント CI です。ルート・セグメントあるいは直接従属セグメントの保管用のスペースが最も望ましいブロック中で使用可能でない場合には、DEDB の挿入アルゴリズム (次に説明) が追加のスペースを検索します。セグメントを保管するスペースが存在できるのは次の 2 つの場所です。

- 従属オーバーフロー部
- この UOW が現在所有している独立オーバーフロー CI

必要に応じて、追加の独立オーバーフロー CI が割り振られます。

関連資料:

高速処理データベース・ランダム化ルーチン (DBFHDC40 / DBFHDC20 DBFHDC44 / DBFHDC24 DBFHDC2S) のサンプル (出口ルーチン)

DEDB 挿入アルゴリズム

DEDB の挿入アルゴリズムは、最も望ましいブロックに使用可能なスペースがない場合には、余分なスペースを検索します。

このトピックには診断、変更、およびチューニングに関する情報が含まれています。

ルート・セグメントの場合、RAP CI にレコード全体を保持するだけのスペースがなければ、この RAP CI はルート・セグメントとできるだけ多くの直接従属セグメントを収容します。ランダマイザーの対象とならない基本 CI は使用されません。このアルゴリズムは、次に、この UOW に応じた最初の従属オーバーフロー CI の中でスペースを探します。第 1 従属オーバーフロー CI のヘッダーから、その CI にスペースが存在するかどうかの判断が行われます。

現行オーバーフロー・ポインターが指している CI に十分なスペースがなければ、次の従属オーバーフロー CI でスペースを探します (そのような CI がある場合に限ります)。現行オーバーフロー・ポインターは、この従属オーバーフロー CI を指すよう更新されます。使用可能な従属オーバーフロー CI がなくなれば、アルゴリズムは独立オーバーフロー部のスペースを検索します。

データを保管するために 1 つの独立オーバーフロー CI が選ばれたときは、それは、それを要求した UOW に対するオーバーフロー部の論理的な延長部分と見なされます。

以下の図は、1 つの UOW がどのように独立オーバーフローに拡張されるかを示しています。この UOW には CI が 10 個と定義され、8 つの基本 CI と 2 つの従属オーバーフロー CI が組み込まれています。この UOW にランダム化されるデータベース・レコードを保管するために、さらにスペースが必要です。独立オーバーフロー CI が 2 つ取得されているので、この UOW のサイズは 12 の CI に拡張されます。第 1 従属オーバーフロー CI には、第 2 独立オーバーフロー CI を指すポインターがあり、次にスペースを探す場所がその CI であることを示しています。

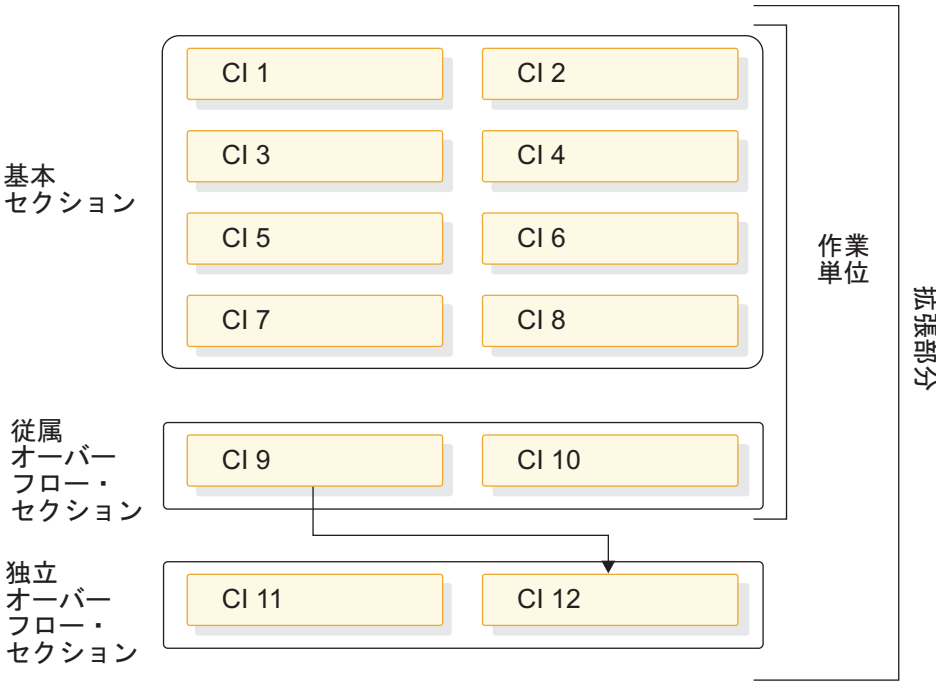


図 *64.* 独立オーバーフローを使用する *UOW* の拡張

関連資料:

223 ページの『CI とセグメントのフォーマット』

DEDB 制御インターバル (CI) 問題支援エイド (診断)

DEDB フリー・スペース・アルゴリズム

DEDB フリー・スペース・アルゴリズムは、従属オーバーフロー CI と独立オーバーフロー CI を解放するのに使用します。

このトピックには診断、変更、およびチューニングに関する情報が含まれています。

従属オーバーフロー CI が完全に空 (から) になると、第 1 従属オーバーフロー CI の現行オーバーフロー・ポインターがこの CI を指し、最も望ましいブロックが満杯になったときにオーバーフロー・スペースとして使用する最初のオーバーフロー

CI であることを示します。独立オーバーフロー CI は、その中に保管されたすべてのセグメントが取り除かれるまで、これが割り振られている UOW に所有されます。独立オーバーフロー CI の中の最後のセグメントが削除されると、空 (から) になったこの CI は使用可能となり、再使用されます。従属オーバーフロー CI の中の最後のセグメントが削除されると、このトピックの冒頭で説明したように、再使用できるようになります。

従属オーバーフロー CI や独立オーバーフロー CI は、再編成またはセグメントの削除によって解放できます。

再編成

オンラインでの再編成中は、UOW の中のセグメントが GN の順序で読み取られ、再編成ユーティリティー専用バッファー・セットに書き込まれます。この処理によって、セグメントが再編成ユーティリティー専用バッファー・セットに挿入され、組み込まれていたフリー・スペースが除去されます。

すべてのセグメントが再編成ユーティリティー専用バッファー・セット (RAP CI に従属オーバーフロー CI を加えたもの) に収まらない場合は、必要に応じて新しい独立オーバーフロー CI が割り振られます。再編成ユーティリティー専用バッファー・セット内のデータが正しい位置にコピーし直されると、次のイベントが発生します。

- 新たに取得された独立オーバーフロー CI が保持される
- 古いセグメントが削除される
- 前に割り振られた独立オーバーフロー CI が解放される

セグメントの削除

セグメントの削除は、アプリケーションの DLET 呼び出しによって行われるか、REPL によってセグメントが異なる長さで置き換えられたため行われます。

セグメントの置き換え (REPL) によって、セグメントが移動することがあります。全機能は、DEDB とは異なる方法でセグメント長の増加を処理します。全機能では、セグメント長が増加したため使用可能なフリー・スペースに収まらない場合、セグメントは分割されて、データは接頭部から離れた位置に挿入されます。 DEDB の場合、置き換えられるセグメントが変更される場合、そのセグメントはまず削除され、次に再挿入されます。この挿入プロセスは、通常のスペース割り振り規則に従います。

最後のセグメントが CI から削除されると、REPL 呼び出しによって、その従属オーバーフロー CI または独立オーバーフロー CI を解放できます。

IMS ツールによる使用不可スペースの管理

あるエリアに関してスペース切れの状態が起きないようにするために DEDB のスペースは綿密にモニターしてください。

IMS ハイパフォーマンス (HP) ポインター・チェッカー (階層データベース (HD) チューニング・エイドおよびスペース・モニターの各ツールを含む) などの製品を使用することにより、RAP CI、従属オーバーフロー CI、および独立オーバーフロー CI の中のフリー・スペースの割合の違いを示すことができます。 RAP CI また

は従属オーバーフロー CI に大量のフリー・スペースがあり、それと同時に独立オーバーフロー CI の使用率が高い場合には、再編成によってデータをルート・アドレス可能部に保管することによって他の UOW が独立オーバーフロー CI を使用できるように解放することができます。 IMS HP ポインター・チェッカーおよびそれに含まれる各ツールは、データを分散させるのが妥当かどうかを判断するのに役立ちます。

関連概念:

790 ページの『高速機能システムのチューニング』

DEDB を対象とする DL/I 呼び出し

DEDB 処理においては、DL/I 処理の場合と同じ呼び出しインターフェースが使用されます。したがって、DEDB を対象として DL/I 呼び出しまたは DL/I 呼び出しシーケンスを実行した場合には、論理的に HDAM または PHDAM データベースを対象として実行した場合と同じ結果になります。

このトピックには診断、変更、およびチューニングに関する情報が含まれています。

DEDB に対する SSA 規則には、以下の制限があります。

- DEDB には、Q コマンド・コードは使用できません。
- IMS は、順次従属セグメントについて使用されたコマンド・コードを無視します。
- DEDB に対する呼び出しで D コマンド・コードを使用する場合には、プログラムの PCB で P 処理オプションを指定する必要はありません。 P 処理オプションは、DEDB では、DL/I データベースの場合とは別の意味を持っています。

関連概念:

DEDB の処理 (IMS、および DBCTL を使用する CICS) (アプリケーション・プログラミング)

混合モードの処理

IMS アプリケーション・プログラムはメッセージ処理プログラム (MPP)、バッチ・メッセージ処理プログラム (BMP)、および高速機能プログラム (IFP) として実行できます。

IFP は全機能データベースをアクセスすることができます。同様に、MPP と BMP は DEDB と MSDB をアクセスすることができます。

同期点処理は多様なので、データベースの更新がコミットされる方法はさまざまです。全機能リソースを要求する IFP、あるいは DEDB (または MSDB) リソースを要求する MPP (または BMP) は「混合モード」で作動します。

関連概念:

261 ページの『高速機能同期点』

主記憶データベース (MSDB)

MSDB 構造は、固定長のルート・セグメントのみで構成されます。ルート・セグメントの長さは、MSDB ごとに異なっていてもかまいません。

セグメントの長さの上限は 32,000 バイトであり、キーの長さの上限は 240 バイトです。このほかに接頭部データが加わるために、合計レコード・サイズの上限は 32,258 バイトに拡張されます。

以下のオプションは、MSDB には使用できません。

- 複数データ・セット・グループ
- 論理関係
- 副次索引
- 可変長セグメント
- フィールド・レベル・センシティビティー

MSDB データベースには、以下の 4 種類があります。

- 端末関連固定 データベース
- 端末関連動的 データベース
- 端末キーあり 端末非関連データベース
- 端末キーなし 端末非関連データベース

推奨事項: 新規の高速機能データベースを開発する場合は、MSDB の代わりに DEDB を使用します。端末関連 MSDB および端末関連鍵を使用する非端末関連 MSDB は、サポートされなくなりました。非端末関連鍵を使用する非端末関連 MSDB は引き続きサポートされていますが、既存の MSDB はすべて DEDB に変換することを検討してください。 MSDB-to-DEDB 変換ユーティリティーを使用することができます。

MSDB は、DBD ステートメントで `ACCESS=MSDB` をコーディングすることによって、他の IMS データベースと同様に DBD の中で定義されます。 DATASET ステートメントの REL キーワードは、MSDB の 4 種類のタイプのうち 1 つを選択します。

動的および固定の端末関連 MSDB は共に以下の特性を備えています。

- レコードの更新は、そのレコードを所有する LTERM から出されたメッセージの処理によってのみ行うことができます。しかし、レコードの読み取りは、任意の LTERM からのメッセージによっても行えます。
- セグメントを所有する LTERM の名前がそのセグメントのキーです。 1 つの LTERM は、どの MSDB においても複数のセグメントを所有することはできません。
- キーは保管されているセグメントの中にはありません。
- ある 1 つの固定の端末関連 MSDB の中の各セグメントは、それぞれ異なる 1 つの LTERM に割り当てられ、それぞれが割り当てられた LTERM により所有されます。

端末関連 MSDB に ETO 端末からアクセスすることはできません。

非端末関連 MSDB は、以下の特性を備えています。

- セグメントの所有権はありません。
- 挿入呼び出しまたは削除呼び出しは許されていません。
- セグメントのキーは LTERM 名とすることもセグメント内の 1 つのフィールドとすることもできます。端末関連 MSDB の場合と同様に、キーが LTERM 名である場合には、そのキーはセグメントの中にはありません。キーが LTERM 名でない場合には、そのキーはセグメントのシーケンス・フィールドの中にあります。キーがセグメントの中にある場合には、セグメントをキー・シーケンスでロードしなければなりません。修飾された SSA がキー・フィールドに出された場合、バイナリー・サーチが開始されるからです。

関連概念:

121 ページの『第 13 章 全機能データベース・タイプ』

123 ページの『パフォーマンス考慮点の概要』

17 ページの『セグメント』

MSDB を使用する場合

MSDB は、ご使用のシステムで最も頻繁に使用されるデータを保管し、そのデータへのアクセスができるようにします。MSDB のデータはセグメントに保管され、各セグメントは 1 つの端末でのみ使用可能とすることができ、1 つあるいはすべての端末に使用可能として割り当てることもできます。

MSDB には高度の並列処理機能があり、銀行の総勘定元帳などのアプリケーションに適しています。高速アクセスを提供し、このデータを頻繁に更新できるようにするため、MSDB は実行時に仮想記憶域に常駐します。

端末関連固定 MSDB の 1 つの用途は、各セグメントにそれぞれ 1 つの論理端末に関連しているデータが入るようなアプリケーションです。この種のアプリケーションでは、アプリケーション・プログラムが (おそらく、一般的な報告のために) データを読み取ることはできますが、このデータを更新することはできません。

(端末関連キーなしの) 非端末関連 MSDB を使用する一般的なアプリケーションは、多数の人々が高いトランザクション率でデータを更新する必要があるようなアプリケーションです。このようなアプリケーションの例としては、多数のキャッシュ・レジスターから在庫減少の通知が入るリアルタイム在庫管理アプリケーションがあります。

MSDB の保管

MSDB 保守ユーティリティー (DBFDBMA0) は、MSDBINIT 順次データ・セットを物理的な昇順で作成します。

コールド・スタート時、または通常のウォーム・スタート時のオペレーターの要求により、順次データ・セットの MSDBINIT が読み取られ、MSDB が仮想記憶域に作成されます。以下の図を参照してください。

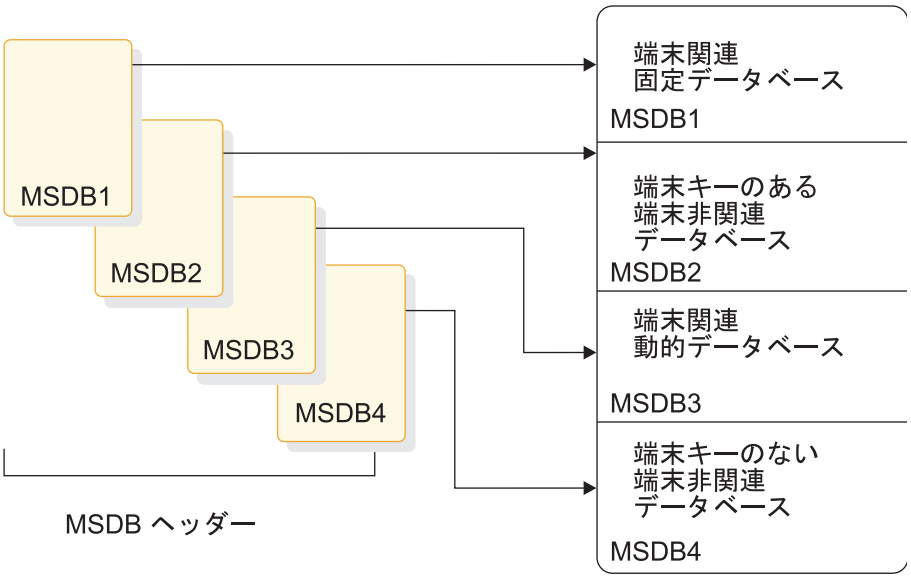


図 *65. MSDB* ポインター

ウォーム・スタート時に、制御プログラムは現行チェックポイント・データ・セットを使用して初期設定を行います。 MSDB 保守ユーティリティーは、古い MSDBINIT データ・セットの内容を変更することもできます。ウォーム・スタートの場合、マスター端末オペレーターはチェックポイント・データ・セットではなしに、IMS.MSDBINIT を使用するよう要求することができます。

以下の図は MSDBINIT レコードのフォーマットを示しています。レコードの各部分については、図の後の表で説明します。

	LL	00	DBDname	Count	Type	KL	KEY	MSDB Segment
バイト	2	2	8	4	1	1	可変	可変 (MAX 32,000)

図 *66. MSDBINIT* レコード形式

表 *55. MSDBINIT* レコード形式

レコード部分	**Bytes**	説明
LL	2	レコード長 (最大 32,258)
X'00'	2	常に 16 進数のゼロ
DBDname	8	DBD 名
Count	4	セグメント数
タイプ	1	MSDB の種類 • X'11' 非関連 • X'31' 非関連 (端末キーあり) • X'33' 固定関連 • X'37' 動的関連
KL	1	キー長 (最大 240)
キー	可変	キーまたは端末名
MSDB segment	可変	MSDB セグメント (最大 32,000)

関連タスク:

633 ページの『MSDB のロード』

MSDB レコードの保管

MSDB レコードに入っているポインターは、端末関連動的データベースの中の空きセグメント・レコードを接続する順方向チェーン・ポインター (FCP) のみです。

このトピックには診断、変更、およびチューニングに関する情報が含まれています。

以下の図は、MSDB が優先順位に従って配置される様子を全体像として見たものです。

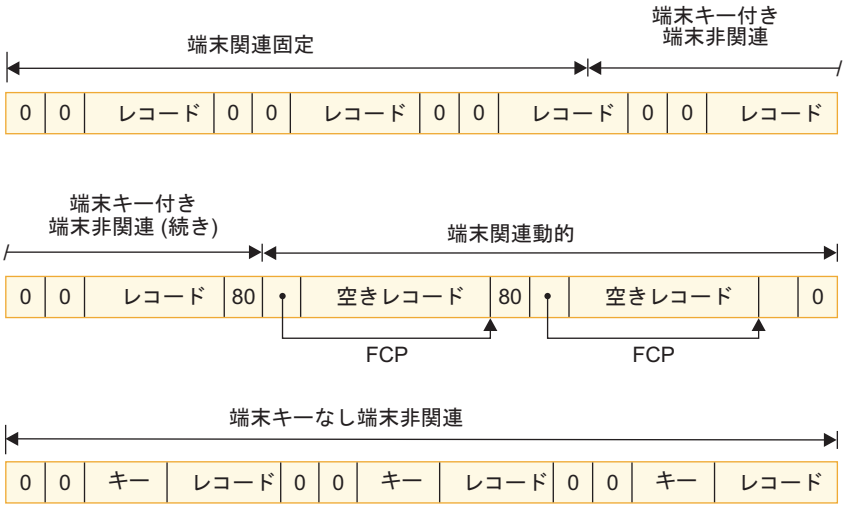


図 67. 4 種類の *MSDB* 編成の順序

再始動用の MSDB の保管

システム・チェックポイントで、MSDB のコピーはすべて、MSDB のチェックポイント・データ・セット (MSDBCP1 または MSDBCP2) の 1 つに交互に書き込まれます。

再始動時に、MSDB は MSDBCP1 または MSDBCP2 の最新のコピーから再ロードされます。緊急時再始動の場合には、ログを使用して MSDB を更新します。正常再始動の場合は、オペレーターが再始動コマンドの MSDBLOAD パラメーターを使用して、MSDBINIT から再ロードすることができます。

コールド・スタート (/ERE CHKPT 0 を含む) の場合、MSDB は MSDBINIT データ・セットからロードされます。

MSDB を対象とする DL/I 呼び出し

DL/I データベース呼び出しは、「親の中のもの」を指定するものを除き、すべて、MSDB に対して有効です。

MSDB はルート・セグメントのみのデータベースであるので、「親の中のもの」の呼び出しは無意味です。このほかに、すべての MSDB に適用できる DL/I 呼び出し、すなわち FLD があります。 FLD 呼び出しを出すことにより、アプリケーション・プログラムは、MSDB セグメントの中の 1 つのフィールドを調べこれを変更することができます。

SSA の使用規則

MSDB の処理においては、SSA (セグメント検索引数) の使用に以下の制限が課せられます。

ブール演算子は使用できない。

コマンド・コードは使用できない。

上記のような制限はありますが、データベースに対する、SSA なしの呼び出し、非修飾 SSA による呼び出し、または修飾された SSA による呼び出しの結果は、全機能データベースに対する呼び出しの結果と同じです。例えば、SSA なしの検索呼び出しが出されると、作業時の環境に応じて、MSDB データベースまたは全機能データベースの最初のレコードが返されます。次のリストは、修飾された SSA に対して使用される比較手法または検索手法の種類を示したものです。

比較の種類

- シーケンス・フィールド: 論理
- シーケンス・フィールド以外の算術フィールド: 算術
- シーケンス・フィールド以外かつ算術フィールド以外: 論理

検索の種類

- シーケンス・フィールド: 演算子が = または >= の場合はバイナリー、それ以外の場合は順次
- シーケンス・フィールド以外の算術フィールド: 順次
- シーケンス・フィールド以外かつ算術フィールド以外: 順次

セグメントの挿入と削除

端末関連動的 MSDB データベースは、ISRT 呼び出しと DLET 呼び出しを受け入れます。しかし、他の MSDB データベースはこれらの呼び出しを受け入れません。

動的データベースでは、実際に、セグメントが物理的に挿入されたり削除されたりするわけではありません。そうではなしに、ISRT 呼び出しは、1 つのセグメントを空きセグメント・プールから 1 つの LTERM に割り当てます。また DLET 呼び出しは、このセグメントを解放し、空きセグメント・プールに戻します。

239 ページの『バイナリー・サーチと直接アクセス方式の組み合わせ』の図は、4 種類の MSDB のレイアウトとこれらの MSDB にアクセスするのに必要な制御ブロックとテーブルを示したものです。拡張通信ノード・テーブル (ECNT) の所在は、拡張システム内容ディレクトリー (ESCD) の中のポインターによって突き止められ、ESCD の所在は、順繰りに、システム目録ディレクトリー (SCD) の中のポインターによって突き止められます。 ESCD には、MSDB ヘッダー・キューを示す最初と最後のヘッダー・ポインターが入っています。 MSDB の各ヘッダーには、それぞれのデータベースのエリアの始まりを指すポインターが入っています。

バイナリー・サーチと直接アクセス方式の組み合わせ

ある 1 つの DL/I 呼び出しで、MSDB に対して複合アクセス手法が活用されます。このアクセス手法はバイナリー・サーチと直接アクセス方式を組み合わせたものです。

ECNT テーブルでのバイナリー・サーチによって、テーブル内の LTERM 名と、要求を出した端末の LTERM 名とを突き合わせようとします。両者が一致すると、アプリケーション・プログラムは、ECNT テーブルの中の直接ポインターを用いて、望みのデータベースのセグメントにアクセスします。端末非関連データベースのセグメントへ端末キーなしでアクセスするときは、ECNT は使用されず、バイナリー・サーチだけが使用されます。

以下の図は、ECNT と MSDB のストレージ・レイアウトを示しています。

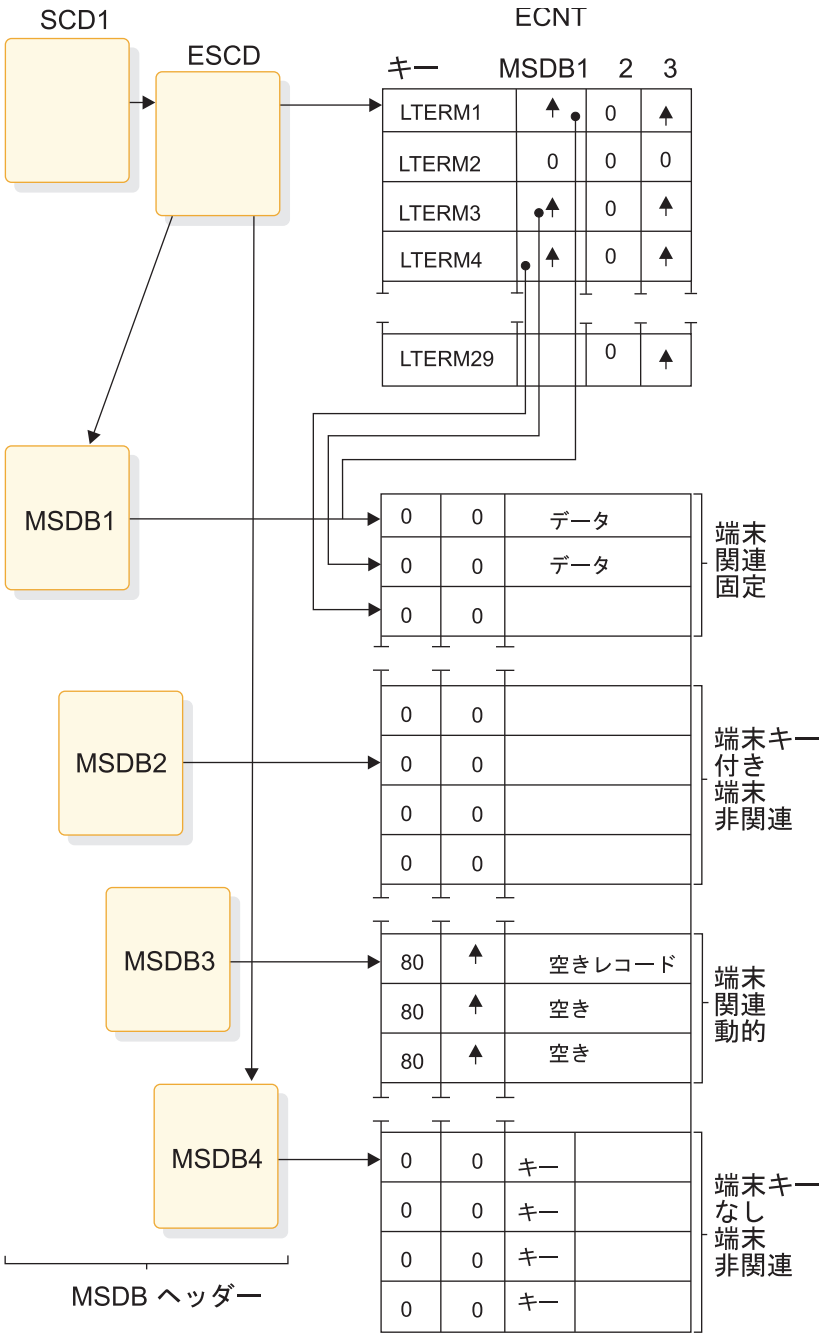


図 *68. ECNT* と *MSDB* のストレージ・レイアウト

MSDB における位置

DL/I 呼び出しが出されると、位置ポインターが現在のセグメントの上に固定されます。「次のセグメント」の意味は、MSDB のキーによって異なります。

LTERM キーなしの端末非関連データベースにおける現在のセグメントは、出された呼び出しが対象としている物理セグメントです。この場合、次のセグメントとは、現在のセグメントに続く物理的隣接セグメントです。他の 3 種類のデータベースでは、LTERM 名がキーとして使用され、現在のポインターが ECNT テーブルの

中のある位置に固定されます。このテーブルの中の各項目は、それぞれ 1 つの LTERM 名と、この LTERM が作業の対象とするすべての MSDB への各セグメント・ポインターとを表しています。ゼロの項目は、LTERM と MSDB セグメントとの間に関連がないことを示しています。この項目がゼロでなければ、次のセグメントとは、このテーブルの中の次の項目です。ゼロ項目は、ゼロでない項目が見つかるまでとばされます。

フィールド呼び出し

MSDB と DEDB で使用可能な DL/I FLD 呼び出しを用いると、1 つのセグメント全体ではなく 1 つのフィールドを対象として操作を行うことができます。

さらに、DL/I FLD 呼び出しでは 1 つのフィールドに対して条件付きの操作を行うこともできます。

変更を行うには、CHANGE の形の FLD 呼び出しを使用します。 VERIFY の形の FLD 呼び出しを使用すれば、フィールドの値をテストすることができます。これらの形の呼び出しを用いることにより、アプリケーション・プログラムは、変更を加える前にフィールドの値をテストすることができます。 VERIFY が失敗に終われば、同じ FLD 呼び出しの中のすべての CHANGE 要求がリジェクトされます。この呼び出しは、「*IMS V13* アプリケーション・プログラミング *API*」で説明しています。

呼び出しシーケンスの結果

MSDB と他の IMS データベースに同じ呼び出しシーケンスを与えても、結果が異なることがあります。

MSDB データに並列にアクセスする場合、MSDB レコードの更新は同期点処理の間に行われます。この同期点が完了するまでは、変更はこのレコードには反映されません。例えば、同じデータベース・レコードに対して呼び出しシーケンス GHU (Get-Hold-Unique)、REPL (Replace)、GU (Get-Unique) を与えた場合、GU 呼び出しの結果として入出力域にもたらされる情報は、GHU 呼び出しで戻されるものと同じです。

データベース・レコードの更新がコミット時点まで延期されることは、FLD/CHANGE 呼び出しにも当てはまり、FLD/VERIFY 呼び出しに影響を及ぼします。同じセグメントの同じフィールドを対象とする、複数の FLD/VERIFY 呼び出しと FLD/CHANGE 呼び出しには注意してください。呼び出しがこのように並んでいると、再処理が行われるため、パフォーマンスが低下することがあります。

端末関連動的 MSDB に対して次のような呼び出しシーケンスを出した場合、その結果は他の IMS データベースまたは DEDB の場合と同じではありません。

- ISRT の次に GHU を出すと、「セグメントが見つからない」という状況コードを受け取ります。
- DLET の後で ISRT を出すと、「セグメントがすでに存在する」という状況コードを受け取ります。
- 1 つのトランザクションの処理については、それぞれの MSDB ごとに 1 つの ISRT または DLET しか許されません。

トランザクションが全機能 DL/I データと MSDB データの両方を更新したり参照したりする場合には、上記の相違がより重大になります。全機能 DL/I データベースと DEDB に対する更新はただちに使用可能になりますが、MSDB に対する変更はそうではありません。例えば、MSDB の 1 つのセグメントに対して GHU と REPL を出し、次に同じコミット・インターバルの同一セグメントに対して別の GET 呼び出しを出す場合に、IMS がユーザーに戻すセグメントは「古い」値で、更新された値ではありません。

処理が単一モードでない場合、この相違は増大することがあります。複数モード処理の場合には、トランザクションごとに同期点処理が呼び出されるわけではありません。これは、MSDB データを更新しようとしているときには、単一モード処理を要求することによっても解決できます。

MSDB の処理について考慮すべきもう 1 つの事項は、端末関連 MSDB セグメントを更新する場合、このセグメントの所有者、すなわち、LTERM から出された初めのトランザクションによってのみ、このセグメントを更新することができるという点です。プログラムが非トランザクション・ドリブン BMP である場合、このプログラムは、端末非関連として宣言されている MSDB しか更新することができません。

高速機能仮想記憶オプション

高速機能仮想記憶オプション (VSO) を使用すると、データを仮想記憶域またはカップリング・ファシリティー構造にマップすることができます。

1 つ以上の DEDB エリアを、VSO エリアとして定義することによって、仮想記憶域またはカップリング・ファシリティー構造にマップすることができます。

DEDB を使用する高性能なアプリケーションでは、DEDB エリアを VSO として定義すると、次のような性能の向上を実現することができます。

- 読み取り入出力が削減される

  仮想記憶域に IMS および VSAM 制御インターバル (CI) を取り込むと、その後のすべての入出力要求では DASD からではなく、仮想記憶域からデータを読み取ります。
- ロッキングの競合が減少する

  VSO の DEDB では、次の両方の場合にロックが解放されます。
  - アプリケーションの同期 (コミット) 点の第 2 フェーズのロギングが完了している場合
  - データが仮想記憶域に移動した場合

  非 VSO の DEDB では、ロックは VSAM CI レベルで行われ、更新されたデータが DASD に書き込まれた後初めてロックが解放されます。
- エリア・データ・セットへの書き込みが少なくなる

  更新されたデータ・バッファーは、ただちに DASD に書き込まれるわけではなく、データ・スペースに保持されて、システム・チェックポイントの発生時、またはしきい値に達したときに DASD に書き込まれます。

この他のすべての点では、VSO DEDB は非 VSO の DEDB と同じです。したがって、VSO DEDB エリアは、他の IMS DB または IMS TM アプリケーションだけでなく、IMS DBCTL と LU 6.2 アプリケーションでも使用できます。 VSO DEDB エリアを定義するには、DBRC コマンド INIT.DBDS および CHANGE.DBDS を使用してください。

VSO DEDB エリアの仮想記憶域は、そのエリアに割り当てられている共用レベルに応じて、異なった場所に収められます。共用レベル 0 および 1 の VSO DEDB エリアは、z/OS データ・スペースにロードされます。共用レベル 2 および 3 の VSO DEDB エリアは、カップリング・ファシリティー・キャッシュ構造にロードされます。

カップリング・ファシリティー・キャッシュ構造はシステム管理者によって定義され、単一の DEDB エリアまたは複数の DEDB エリアを収容することができます。複数の DEDB エリアをサポートするキャッシュ構造は、多重エリア構造 と呼ばれます。多重エリア構造について詳しくは、「*IMS V13* システム管理」を参照してください。

推奨: 端末関連 MSDB と端末関連キーを使用する非端末関連 MSDB は、サポートされていません。端末関連キーを使用しない非端末関連 MSDB は、サポートされています。したがって、既存の MSDB すべてを VSO DEDB または非 VSO の DEDB に変換することを考慮する必要があります。

関連概念:

220 ページの『DEDB とデータ共用』

VSO DEDB エリアの使用時の制約事項

VSO DEDB エリアには、その使用に関するいくつかの制約事項があります。

制約事項は次のとおりです。

- VSO DEDB エリアは、DBRC に登録しなければならない。
- ローカル VSO DEDB エリアの場合、z/OS データ・スペースには 2 GB (2 147 483 648 バイト) という最大サイズの制限があり、これは DASD 上の単一または多重エリア・データ・セットの最大サイズが 4 GB であっても変わらない。ローカル VSO DEDB エリアが 2 GB より大きい場合、そのエリアは開けません。

  ローカル VSO DEDB エリアがプリロード用に開かれるとき、IMS は、そのエリアがデータ・スペース内に収まるかを検査して確認します。そのエリアが収まらない場合、オープンは失敗します。

  z/OS データ・スペースでローカル VSO DEDB エリアに使用可能な実際のサイズは、最大サイズ (2 GB) から z/OS で使用される容量 (0 から 4 KB) および IMS 高速機能で使用される容量 (約 100 KB) を引いた容量になります。

  VSO DEDB エリアのサイズ、使用量、その他の統計を見るには、/DISPLAY FPV コマンドを入力してください。
- 共用 VSO DEDB エリアの場合、z/OS カップリング・ファシリティー・キャッシュ構造のエリア・データ・セットの最大サイズに制限はありません。カップリング・ファシリティー・キャッシュ構造の最大サイズは 2 GB です。共用 VSO

DEDB エリアが 2 GB より大きい場合、IMS は、カップリング・ファシリティー・キャッシュ構造に最大の 2 GB の CI をロードします。

IMS は、共用 VSO DEDB エリアがプリロードされるか要求時にのみロードされるかに関係なく、その共用 VSO DEDB エリアがカップリング・ファシリティー・キャッシュ構造に収まるかどうかを検査しません。

- DEDB エリア・データ・セット比較ユーティリティー (DBFUMMH0) は、VSO DEDB エリアはサポートしていません。

関連概念:

253 ページの『データ・スペースのアクセス』

関連資料:

/DISPLAY FPV コマンド (コマンド)

VSO DEDB エリアの定義

DEDB の仮想記憶オプション (VSO) に関する情報はすべて、RECON データ・セットに記録されます。

VSO DEDB エリアを定義するには、DBRC INIT.DBDS および CHANGE.DBDS コマンドの以下のパラメーターを使用してください。

VSO エリアを VSO エリアとして定義します。

CI の最初の読み取り時に、z/OS データ・スペースまたはカップリング・ファシリティー構造にコピーされます。データは共通バッファーに読み込まれた後、データ・スペースまたは構造にコピーされます。その後データにアクセスすると、DASD からではなく、データ・スペースまたは構造から検索されます。

読み取られない CI は、データ・スペースまたは構造にコピーされません。

そのデータに対して更新を行うと、その内容はすべて再度データ・スペースまたは構造にコピーされ、保持されていたロックは解放されます。更新された CI は、定期的に DASD に戻され、書き込まれます。

NOVSO

エリアを非 VSO エリアとして定義します。これはデフォルトです。

DEDB を非 VSO として定義する場合、または特定のエリアに対して VSO オプションをオフにする場合、NOVSO を使用することができます。エリアを NOVSO として再定義したときにそのエリアが仮想記憶域内にあった場合、変更内容を有効にするため、そのエリアを停止 (/STOP AREA または /DBR AREA) するか、または仮想記憶域から除去する (/VUNLOAD) 必要があります。

PRELOAD

VSO エリアについては、VSO エリアのオープン時にエリアをデータ・スペースまたはカップリング・ファシリティー構造にプリロードします。このキーワードは、PREOPEN キーワードを暗黙指定するので、PRELOAD を指定した後 PREOPEN を指定する必要はありません。

制御領域の初期設定時または /START AREA 処理中に、エリアのルート・アドレス可能部と独立オーバーフロー部がデータ・スペースまたはカップリン

グ・ファシリティー構造にロードされます。その後データは、データ・スペースまたはカップリング・ファシリティー構造から共通バッファーに読み取られます。更新は再度データ・スペースまたはカップリング・ファシリティー構造にコピーされ、ロックはすべて解放されます。更新された CI は、定期的に DASD に戻され、書き込まれます。

NOPREL
エリアを要求時ロードとして定義します。 VSO DEDB エリアについては、CI はデータ・セットから読み取られると、データ・スペースまたはカップリング・ファシリティー構造にコピーされます。これはデフォルトです。

NOPREL を使用してエリアを定義すると、プリロード処理を非活動化する能力が与えられます。このエリアは次回のオープン時にデータ・スペースまたはカップリング・ファシリティー構造にプリロードされません。

NOPREL を指定し、エリアを事前オープンするには、そのエリアに対して別に PREOPEN を指定する必要があります。

CFSTR1
1 次カップリング・ファシリティーのキャッシュ構造の名前を定義します。キャッシュ構造名は、z/OS カップリング・ファシリティーの命名規則に従わなければなりません。CFSTR1 はデフォルトとして DEDB エリアの名前を使用します。このパラメーターは、SHARELVL(2|3) を使用して定義した VSO DEDB エリアについてのみ有効です。

CFSTR2
IMS によって管理される構造の二重化を使用する場合に、2 次カップリング・ファシリティー・キャッシュ構造名を定義します。キャッシュ構造名は、z/OS カップリング・ファシリティーの命名規則に従わなければなりません。CFSTR2 はデフォルトの名前を提供しません。このパラメーターは、SHARELVL(2|3) を使用して定義し単一エリア構造を持つ DEDB の VSO エリアについてのみ有効です。このパラメーターは多重エリア構造には使用できません。多重エリア構造はシステム管理の二重化を使用します。

MAS VSO DEDB エリアを、単一エリア構造ではなく、多重エリア構造を使用するものとして定義します。

NOMAS
VSO DEDB エリアを、多重エリア構造ではなく、単一エリア・キャッシュ構造を使用するものとして定義します。 NOMAS はデフォルトです。

LKASID
NOLKASID
このエリアに対する読み取り要求で、バッファー・ルックアサイドを実行するかどうかを指定します。

単一エリア構造を使用する VSO DEDB エリアについては、RECON データ・セットまたはオプションで DFSVSMxx PROCLIB メンバーで、LKASID または NOLKASID を指定することができます。RECON データ・セットでの LKASID または NOLKASID の指定は、DFSVSMxx PROCLIB メンバーでの LKASID または NOLKASID の指定に優先します。

多重エリア構造を使用する VSO DEDB エリアについては、DFSVSMxx PROCLIB メンバーを使用して LKASID または NOLKASID を指定する必要があります。RECON データ・セット内の指定は無視されます。

FULLSEG
NOFULLSG

置き換え (REPL) 呼び出しによってセグメントが更新された場合、X'5950' ログ・レコードに完全なセグメント・イメージを記録するかどうかを指定する、互いに排他的なオプション・キーワード。これらのキーワードは、高速機能 DEDB の場合にのみ有効です。

FULLSEG は、完全なセグメント・イメージをログに記録するよう指示します。

NOFULLSG は、セグメントの更新された部分だけをログに記録するよう指示します。

どちらのキーワードも指定しなかった場合は、DEDB のデータベース・レコードで設定されているデフォルト値が使用されます。

関連概念:

250 ページの『カップリング・ファシリティー構造の命名規則』

252 ページの『DFSVSMxx IMS.PROCLIB メンバーを使用する専用バッファー・プールの定義』

776 ページの『動的データベース・バッファー・プールの概要』

シスプレックス・データ共用の概念と用語 (システム管理)

関連タスク:

251 ページの『DBRC でのキャッシュ構造名の登録』

関連資料:

INIT.DBDS コマンド (コマンド)

CHANGE.DBDS コマンド (コマンド)

DBRC コマンド (コマンド)

IMS PROCLIB データ・セットの DFSVSMxx メンバー (システム定義)

VSO DEDB エリアと PREOPEN および NOPREO キーワード

DBRC の INIT.DBDS コマンドと CHANGE.DBDS コマンドの PREOPEN および NOPREO キーワードは、VSO DEDB エリアと非 VSO DEDB エリアの両方に使用できます。

NOPREO エリアが共用レベル 2 または 3 で共用 VSO としても定義されている場合には、/START AREA コマンドでエリアをオープンすることができます。これにより、エリアが VSO 構造に接続されます。

DBRC コマンドは、VSO DEDB エリアを定義するためにいつでも使用することができます。IMS がアクティブな状態である必要はありません。これらの DBRC コマンドで指定したキーワードは、高速機能処理の次の 2 つの異なる時点で有効になります。

- 制御領域の始動時

  制御領域の初期設定後の最初のチェックポイントの後、DBRC はいずれかの VSO オプション (VSO、NOVSO、PRELOAD、NOPREL) または PREOPEN オプションか NOPREO オプションのどちらかを指定したエリアのリストを提供します。これらのオプションは、その後 IMS 高速機能によって保持されます。
- コマンドの処理時

  /START AREA コマンドを使用すると、DBRC はそのエリアに VSO オプションまたは PREOPEN|NOPREO オプションを提供します。エリアを事前オープンまたはプリロードする必要がある場合、この時点で行います。

  /STOP AREA コマンドを使用すると、必要な VSO 処理が実行されます。

  関連資料: /START および /STOP コマンド処理の詳細については、「*IMS V13* コマンド 第 *2* 巻*: IMS* コマンド *N-V*」を参照してください。

VSO DEDB エリアの共用

VSO DEDB エリアを共用すると、複数の IMS システムが同じ VSO DEDB エリアを並行して読み取り、更新することができます。 3 つの主な参加者は、カップリング・ファシリティー・ハードウェア、カップリング・ファシリティー方針ソフトウェア、および XES サービスと z/OS サービスです。

カップリング・ファシリティー・ハードウェアは、シスプレックス環境で IMS システムがデータを共用できる、高性能で、ランダム・アクセスの共用ストレージを提供します。カップリング・ファシリティー内の共用ストレージのエリアは、構造というセクションに分割されます。 VSO DEDB データの場合、使用される構造タイプは、リスト構造またはロック構造ではなく、キャッシュ構造 と呼ばれます。キャッシュ構造は、高性能読み取り参照再使用と変更済みデータの据え置き書き出し用に設計されています。カップリング・ファシリティーと構造は、共通の z/OS データ・セットである結合データ・セット (COUPLExx) に定義されます。

カップリング・ファシリティーの方針ソフトウェアとそのキャッシュ構造サービスは、共用ストレージでの VSO DEDB データの共用を可能にするインターフェースとサービスを z/OS に提供します。共用ストレージは VSO DEDB の読み取りと書き込みを制御します。

- VSO CI の読み取りによって CI が DASD からカップリング・ファシリティーにもち込まれる。
- 更新された VSO CI の書き込みは、CI を主記憶域からカップリング・ファシリティーにコピーし、それに変更のマークを付ける。
- 変更された CI データは、定期的に DASD に書き戻される。

XES サービスと z/OS サービスは、キャッシュ構造内のデータを操作する方法が提供されます。これらのサービスは、複数の IMS システムがデータを共用するための高いパフォーマンス、データ保全性、および整合性を提供します。

関連概念:

220 ページの『DEDB とデータ共用』

カップリング・ファシリティーと共用ストレージ

カップリング・ファシリティー共用ストレージでは、1 つのキャッシュ構造が 1 つ以上の VSO エリアを表すことができます。ただし、どの VSO DEDB エリアも、1 つのキャッシュ構造によってのみ表されます。

キャッシュ構造に持続性はありません。すなわち、これらの構造は、カップリング・ファシリティーからの最後の IMS システム切断後に削除されます。

二重化構造

二重化構造は、同じエリアの重複構造です。

二重化を行うと VSO DEDB エリアの二重構造がサポートされ、データの可用性とリカバリー可能性の確立に役立ちます。

構造の二重化は、IMS で管理するか、システムで管理することができます。IMS で管理される二重化の場合、DBRC および z/OS カップリング・ファシリティー・リソース管理 (CFRM) ポリシーで、1 次および 2 次の両方の構造を定義しなければなりません。システムで管理される二重化の場合には、基本次構造のみを定義する必要があります。二重動作はユーザーには透過ですが、ユーザーは CFRM ポリシー内で二重化モードを要求し、2 次構造インスタンス用の追加リソースを割り振ることだけが必要です。

VSO 多重エリア構造では、システムで管理される二重化を使用する必要があります。

関連タスク:

CQS における z/OS 構造の二重化 (システム管理)

構造サイズの自動変更

z/OS は、ストレージ・スペースを必要とする場合、カップリング・ファシリティー内の VSO 構造のサイズを自動的に拡大または縮小することができます。

プリロード済み VSO DEDB でこの機能を使用可能にすると、無駄なスペースの発生を避けることができます。ただし、VSO DEDB が要求時ロードされるときは、この機能の使用に関して注意が必要です。

推奨事項: 共用 VSO DEDB エリアをプリロードする場合、自動変更機能は使用しないでください。自動変更機能は、キャッシュ構造内の未変更のデータを再利用することがあります。

確実に正しいサイズ変更を行い、確実にプリロード済みの共用 VSO DEDB エリアに対して自動変更機能を使用不可にするためには、エリアの CFRM ポリシーに以下のパラメーターを指定してください。

INITSIZE(*x*)
: カップリング・ファシリティー内の構造に割り振る初期スペース量を指定します。このサイズは、IBM System z® CFSizer (Coupling Facility Structure Sizer Tool) を使用して見積もることができます。CFSizer は、www.ibm.com/servers/eserver/zseries/cfsizer/ の Web サイトで入手できます。あるいは、IBM Web サイト (www.ibm.com) で「CFSizer」を検索してください。

SIZE(*y*)
カップリング・ファシリティー内の構造に割り振る最大スペース量を指定します。

ALLOWAUTOALT(NO)
この構造に対してシステム開始の変更 (自動変更) を許可するかどうかを指定します。プリロード済み共用 VSO DEDB エリアの場合は、ALLOWAUTOALT(NO) を指定してください。

INITSIZE によって指定されたサイズが小さすぎる場合、IMS は構造のサイズを、SIZE パラメーターで指定されたサイズまでの、IXLCSP によって計算された必要な値に変更します。

関連資料: 構造の自動変更を使用可能にするための CFRM パラメーターについては、「*z/OS MVS*™ シスプレックスのセットアップ」を参照してください。

関連概念:

IMS 環境でのデータ共用 (システム管理)

システム管理の再作成

すべての VSO 構造をオンラインにしている場合、構造を別のカップリング・ファシリティーにコピーすることにより、カップリング・ファシリティーを再構成することができます。 VSO 定義に加える変更はありません。

関連概念:

シスプレックス・データ共用の概念と用語 (システム管理)

専用バッファー・プール

IMS には、共用 VSO エリアのための特別な専用バッファー・プールがあります。各プールは、エリア、DBD、またはエリアの特定グループに関連付けることができます。

これらの専用バッファー・プールは、共用 VSO データ専用です。これらの専用バッファー・プールを使用すれば、データのバッファー・ルックアサイドを要求することができます。キーワード LKASID または NOLKASID を DBRC コマンド INIT.DBDS または CHANGE.DBDS で指定すると、このルックアサイド機能を使用するかどうかを示すことができます。

DEDB VSO 構造への接続の許可

共用 DEDB VSO 構造へのアクセスを管理するには、キャッシュ構造へのアクセス権限を付与するセキュリティー・プロファイルを、許可された IMS システムのみに定義します。

RACF を使用する場合、RACF セキュリティー管理者は FACILITY クラス内にセキュリティー・プロファイルを定義します。

DEDB VSO 構造に接続する前に、IMS システムは IMS 制御領域のジョブ名をユーザー ID として使用して、RACROUTE REQUEST=AUTH 呼び出しを発行します。この構造にアクセスするには、ユーザー ID がセキュリティー・プロファイル内に少なくとも UPDATE 権限を持っている必要があります。

セキュリティー・プロファイルの名前には、VSOSTR.*structure_name* のフォーマットを使用する必要があります。ここで、*structure_name* は保護対象の VSO 構造の名前です。この構造名は、構造の RECON データ・セット内で定義されている構造名に一致する必要があります。

以下の例は、VSO 構造の RACF セキュリティー・プロファイルを定義する RACF コマンドと、IMS システムに更新権限を付与する RACF コマンドの両方を示しています。保護対象の VSO 構造の名前は DB21AR1@STRUCT@1 です。更新権限を受け取る IMS システムは VSOB06A1 です。

```
ADDUSER VSOB06A1
      RDEFINE FACILITY (VSOSTR.DB21AR1@STRUCT@1) UACC(NONE)
      PERMIT VSOSTR.DB21AR1@STRUCT@1 CLASS(FACILITY) ID(VSOB06A1)
      ACCESS(UPDATE)
      SETROPTS CLASSACT(FACILITY)
```

関連タスク:

251 ページの『DBRC でのキャッシュ構造名の登録』

VSO DEDB キャッシュ構造名の定義

システム・プログラマーは、VSO キャッシュ構造を含むすべてのカップリング・ファシリティー構造を、CFRM ポリシー定義内で定義します。

このポリシー定義では、VSO 構造を、リスト構造 (共用キューによって使用される) またはロック構造 (IRLM によって使用される) ではなく、キャッシュ構造として定義します。

カップリング・ファシリティー構造の命名規則

カップリング・ファシリティー構造名の長さは 16 文字で、必要に応じて右側がブランクで埋め込まれます。

名前には以下のいずれの文字も使用できますが、先頭は大文字の英字でなければなりません。

- 大文字の英字
- 数字
- 特殊文字 ($、@、および #)
- 下線 (_)

IBM 名は、以下の文字から始まります。

- SYS
- 文字 A から I (大文字)
- IBM コンポーネントの接頭部

関連概念:

244 ページの『VSO DEDB エリアの定義』

カップリング・ファシリティー構造の定義例

以下の JCL は、別個のカップリング・ファシリティー内に 2 つの構造を定義する方法を示しています。

```
//UPDATE EXEC PGM=IXCL2FDA
//SYSPRINT DD SYSOUT=A
//*
//*  THE FOLLOWING SYSIN WILL UPDATE THE POLICY DATA IN THE COUPLE
//*  DATASET FOR CFRM (COUPLING FACILITY RESOURCE MANAGEMENT)
//*
//SYSIN  DD *
 UPDATE DSN(IMS.DSHR.PRIME.FUNC) VOLSER(DSHR03)

  DEFINE POLICY(POLICY1)

    DEFINE CF(FACIL01)
           ND(123456)
           SIDE(0)
           ID(01)
           DUMPSPACE(2000)

    DEFINE CF(FACIL02)
           ND(123456)
           SIDE(1)
           ID(02)
           DUMPSPACE(2000)

    DEFINE STR(LIST01)
           SIZE(1000)
           PREFLIST(FACIL01,FACIL02)
           EXCLLIST(CACHE01)

    DEFINE STR(CACHE01)
           SIZE(1000)
           PREFLIST(FACIL02,FACIL01)
           EXCLLIST(LIST01)
/*
```

この例では、プログラマーは 1 つのリスト構造 (LIST01) と 1 つのキャッシュ構造 (CACHE01) を定義しています。

注意: キャッシュ構造を DBRC に対して定義する場合、名前が CFRM ポリシーで使用されている名前と同一になるようにしてください。

関連タスク:

『DBRC でのキャッシュ構造名の登録』

DBRC でのキャッシュ構造名の登録

DBRC に対して DEDB エリアを定義する場合、CFRM ポリシーで定義されているものと同じ構造名を使用して、各 DEDB エリアが使用する構造を指定してください。

DEDB エリアの定義および対応する構造名が、その後 RECON データ・セットに保管されます。構造名は、INIT.DBDS または CHANGE.DBDS コマンドの CFSTR1 または CFSTR2 パラメーターのいずれかで入力します。

制約事項: CFSTR2 パラメーターは多重エリア構造ではサポートされていません。INIT.DBDS で CFSTR2 と MAS の両方を指定するか、CHANGE.DBDS を使用してすでに MAS によって定義されている DEDB エリアに CFSTR2 を適用すると、IMS は DSP0141I または DSP0144I エラー・メッセージを出して DBRC コマンドをリジェクトします。

以下に示す INIT.DBDS コマンドの例では、構造名 TSTDEDBAR1 を登録しています。

関連概念:

244 ページの『VSO DEDB エリアの定義』

250 ページの『カップリング・ファシリティー構造の定義例』

関連タスク:

249 ページの『DEDB VSO 構造への接続の許可』

DFSVSMxx IMS.PROCLIB メンバーを使用する専用バッファー・プールの定義

専用バッファー・プールを定義するには、IMS.PROCLIB データ・セットの DFSVSMxx メンバーで DEDB ステートメントを指定します。

例えば、次の 2 つのステートメントは、2 つの専用バッファー・プールを定義します。

```
DEDB=(POOL1,512,400,50,800,LKASID)
DEDB=(POOL2,8196,100,20,400,NOLKASID)
```

1 番目のステートメントが定義しているプールは、バッファー・サイズが 512 で、最初に 400 個のバッファーを割り振り、必要に応じて 50 個ずつバッファーを増やし、最大 800 個まで割り振ります。このプールはローカル・キャッシュとして使用され、バッファー・ルックアサイドが、このプールを共用するエリアごとに実行されます。

2 番目のステートメントが定義しているプールは、バッファー・サイズが 8 K で、最初に 100 個のバッファーを割り振り、必要に応じて 20 個ずつバッファーを増やし、最大 400 個まで割り振ります。このプールは共通バッファー・プールと同様に使用されます。ルックアサイドは実行されません。

専用バッファー・プールを定義しない場合は、デフォルトのパラメーター値が次のステートメントによって記述されます。

```
DEDB=(poolname,XXX,64,16,512)
```

このステートメントの内容は次のとおりです。

- XXX は、オープンするエリアの CI サイズです。
- 初期バッファー割り振りは 64 です。
- 2 次割り振りは 16 です。
- このプールに許可されるバッファーの最大数は 512 です。
- LKASID オプションは、このエリアの DBRC に指定されている場合、指定されます。

関連概念:

244 ページの『VSO DEDB エリアの定義』

関連資料:

単一エリア構造のための高速機能 DEDB バッファー・プールの定義 (システム定義)

多重エリア構造用の専用バッファー・プールの定義

DFSVSMxx PROCLIB メンバーの DEDBMAS= キーワードを使用して、多重エリア構造用の専用バッファー・プールを定義することができます。

cisize と *strname* の 2 つの追加パラメーターは例外として、DEDBMAS キーワードのパラメーターは DFSVSMxx PROCLIB メンバーの DEDB= キーワードの場合と同じです。

cisize パラメーターは、エリアの制御インターバルのサイズを指定します。*strname* パラメーターは、1 次カップリング・ファシリティー構造の名前を指定します。この構造は、カップリング・ファシリティー・リソース管理 (CFRM) 管理ポリシーで定義する必要があります。

関連資料:

多重エリア構造のための高速機能 DEDB バッファー・プールの定義 (システム定義)

VSO DEDB エリアのためのデータ・スペースの獲得とアクセス

IMS は、VSO DEDB エリアを収めるためのデータ・スペースを割り振ります。

VSO DEDB エリア CI は、プリロード時または最初の読み取り時に、データ・スペース (またはカップリング・ファシリティー構造) にコピーされます。そのデータにアクセスすると、DASD ではなく、そのデータ・スペースからデータが検索されます。

データ・スペースの獲得

IMS は、VSO エリアが最初にオープンする時点で VSO エリアのデータ・スペースを獲得し、それ以前に獲得しておくことはありません。

VSO エリア・データ・スペースの最大サイズは、2 ギガバイトです。プリロード済み VSO エリアのデータ・スペースは、z/OS DREF (参照を使用不可にする) オプションを使用します。プリロードされていない VSO エリアのデータ・スペースは、DREF オプションを使用しません。

DREF データ・スペースは中央ストレージを使用しますが、補助ストレージは使用しません。 DREF オプションのないデータ・スペースは、中央ストレージと、補助ストレージが使用可能であれば補助ストレージを使用します。

IMS は VSO エリアのために、必要に応じて DREF オプションのあるものとないものの両方をあわせた追加データ・スペースを獲得します。

データ・スペースのアクセス

IMS は、「first fit (最初の適合)」アルゴリズムによってエリアをデータ・スペースに割り当てます。

エリアのルート・アドレス可能部全体 (独立オーバーフロー部を含む) がデータ・スペースに存在します。順次従属部は、データ・スペースには存在しません。

データ・スペース内のエリアに必要なスペースの量は、*(CI* サイズ*)* × *(UOW* 当たりの *CI* の数*)* × *((*ルート・アドレス可能部の *UOW* の数*)* + *(*独立オーバーフロー部の *UOW* の数*))* となります。値は 4 KB 単位に丸められます。

この式を DBDGEN AREA ステートメントのパラメーターの用語で表すと、 *(SIZE* パラメーターの値*)* × *(UOW* パラメーターの値*)* × *(ROOT* パラメーターの値*)* となります。値は 4 KB 単位に丸められます。

1 つ以上のエリアで使用できるデータ・スペース内の実際のスペースの容量は、2 ギガバイト (524,288 ブロック、各 4 KB) から z/OS が予約している容量 (0 から 4 KB) を引き、さらに IMS 高速機能で使用される容量 (約 100 KB) を引いたものです。/DISPLAY FPVIRTUAL コマンドを使用して、特定のエリアの実際のストレージの使用量を判別することができます。

IMS 制御領域の初期設定中に、IMS は DBRC を呼び出して VSO として定義されている全エリアのリストを要求します。このリストには、各 VSO エリアの PREOPEN または PRELOAD 状況を含みます。 VSO エリアが存在すると、IMS は適切なデータ・スペースを獲得します。その後、IMS は PREOPEN を使用して定義されているすべてのエリアをオープンし、PRELOAD を使用して定義されているエリアをオープンおよびロードします。 IMS プロシージャーの FPOPN パラメーターの指定を変更した場合には、正常または緊急時再始動の間、エリアのオープンとロードは制御領域初期設定の後で行われることがあります。

関連概念:

243 ページの『VSO DEDB エリアの使用時の制約事項』

リソースの制御とロッキング

VSO を使用すると、セグメント・レベルで DEDB リソース要求を管理し、更新セグメントがデータ・スペースに返されるまでロッキングすることによって、DEDB リソースのロッキングの競合数を減らし、所要時間を短縮することができます。

セグメント・レベルのリソース制御とロッキングは、Get 呼び出しと Replace 呼び出し (置き換え呼び出し) にしか適用されません。

VSO を使用しない場合、VSAM CI (物理ブロック) は、DEDB リソース要求管理とロッキングに使用できる最小のリソースです。 CI の一部に更新が行われた場合、その CI 全体が DASD に再書き込みされるまでロックは保持されます。最初のリクエスターのロックが解放されるまで、他のリクエスターは CI のどの部分にもアクセスできません。

VSO では、DEDB リソース要求管理とロッキングに使用できる最小のリソースはデータベース・セグメントです。セグメント・レベルでロッキングできるのは、ルート・セグメント専用構造を持つ DEDB のルート・セグメント部分だけであり、さらにそのルート・セグメントが固定長セグメントである場合だけです。呼び出し PCB に処理オプション R または G を指定した場合、IMS は DEDB リソース要求を管理/制御し、セグメント・レベルで変更を逐次化することができます。他の処理オプションでは、IMS は VSAM CI ロックを保持します。セグメント・ロックが保持されるのは、データ・スペース内の CI にセグメントの更新が適用されるまでです。同一の CI 内の異なるセグメントに対する他の複数のリクエスターは、同時にアクセスすることが許されています。

セグメントに対する VSO DEDB リソース要求を行うと、CI 全体が共通バッファーにコピーされます。 VSO は、セグメントに対する要求とそのアクセスの意図と一致した制御レベルで、セグメント要求を管理します。また、VSO はそのセグメントが入っている CI に対するアクセスを管理しますが、その管理はすべての場合に共用レベルで行われます。同じ CI の中のセグメントに対して、それ以降別のユーザーから要求が出されると、すでにバッファーに入っている CI のイメージがアクセスされます。

データの更新は、更新時にバッファー内の CI に直接適用されます。セグメント・レベルのリソース制御とシリアライゼーションによって、複数のリクエスターの間での保全性が確保されます。更新されたセグメントがコミットされ、データ・スペース内の CI のコピーに適用された後、他のリクエスターがバッファー内の CI のコピーから更新済みセグメントにアクセスできるようになります。

セグメントが変更された後、リクエスターによる更新が何らかの理由でコミットされなかった場合、VSO はセグメントの未変更のイメージをデータ・スペースからバッファー内の CI へコピーします。 VSO は、コミットされていない、取り消された更新を除去する処理を完了してから初めて、他のリクエスターがそのセグメントにアクセスするのを許可します。セグメント・レベルでのロッキングは、共用 VSO エリアではサポートされません。 CI ロッキングのみがサポートされます。

圧縮ルーチンが、ルート専用構造をもつ DEDB のルート・セグメントに定義されているとき、しかもそのルート・セグメントが固定長セグメントであるときは、圧縮後のその長さが変数になります。そのため、圧縮済みセグメントを置き換えるには削除と挿入が必要です。この場合、セグメント・レベルでの制御とロッキングは利用不能です。

データ共用環境での事前オープン・エリアと VSO エリア

VSO は、さまざまな共用レベルに登録することができます。

VSO は、以下の共用レベルのいずれかを使用して登録することができます。

SHARELVL(0)

排他的アクセス: データ共用環境では、制御領域の初期設定を完了する最初の IMS システムが、PREOPEN オプション (VSO PREOPEN と VSO PRELOAD を含む) を指定した SHARELVL(0) エリアをオープンします。IMS は、他の IMS ではエリアを事前オープンしません。

SHARELVL(1)

1 人による更新、多数による読み取り: データ共用環境では、PREOPEN オプションを指定した SHARELVL(1) エリアが、共用するすべての IMS システムによって事前にオープンされます。制御領域の初期設定を完了する最初の IMS システムは更新許可を持ち、その他はすべて読み取り許可を持ちます。

SHARELVL(1) エリアが VSO エリアの場合、このエリアをオープンする IMS がこのエリアをデータ・スペースに割り振ります。エリアが VSO PREOPEN または VSO PRELOAD で定義されている場合は、共用するすべての IMS システムがこのエリアをデータ・スペースに割り振ります。

エリアが VSO NOPREO NOPREL として定義されている場合は、すべての IMS システムが (それぞれがこのエリアをオープンするときに) このエリアをデータ・スペースに割り振ります。このエリアにアクセスする最初の IMS は更新許可を持ち、その他はすべて読み取り許可を持ちます。

SHARELVL(2)
ブロック・レベル共用: 少なくとも 1 つのカップリング・ファシリティー構造名 (CFSTR1) が定義されている SHARELVL(2) エリアは、単一の IRLM の範囲内で、ブロック・レベルまたは制御インターバル (CI) レベルで共用されます。複数の IMS システムが同じ IRLM を使用している場合、それらの複数の IMS システムは更新あるいは読み取りの処理を許可されることができます。

SHARELVL(3)
ブロック・レベル共用: 少なくとも 1 つのカップリング・ファシリティー構造名 (CFSTR1) が定義されている SHARELVL(3) エリアは、複数の IRLM の範囲内で、ブロック・レベルまたは制御インターバル (CI) レベルで共用されます。複数の IMS システムに非排他アクセスを許可することができます。

注意**:** VSO エリアを SHARELVL(1) として登録する場合は注意が必要です。読み取り専用許可を持ったシステムは、読み取り/書き込みシステムが行った更新を認識できません。これは、読み取りはすべて最終的に更新が書き込まれる DASD ではなく、データ・スペースから行われるためです。

VSO を使用した入出力処理

仮想記憶オプション (VSO) が使用された場合、読み取り要求および更新要求に対する応答として IMS でバッファー、データ・スペース、および DASD が使用される方法は、VSO が使用されない場合と異なることがあります。

以下のトピックで詳しく説明します。

入力処理

アプリケーション・プログラムが VSO エリアに対して読み取り要求を出すときに、IMS はそのデータがデータ・スペースにあるかどうかを検査します。

データがデータ・スペースにあれば、データ・スペースから共通バッファーにコピーされ、アプリケーションに渡されます。データがデータ・スペースになかった場合、IMS は DASD 上のエリア・データ・セットから CI を読み取って、共通バッファーに入れ、そのデータをデータ・スペースにコピーして、アプリケーションに渡します。

SHARELVL(2|3) VSO エリアの場合、高速機能は専用バッファー・プールを使用します。バッファー・ルックアサイドは、このようなバッファー・プールに関するオプションです。ルックアサイド・プールを使用する SHARELVL(2|3) VSO エリアに対して読み取り要求が出されると、要求されたデータがそのプールに存在するかどうかの検査が行われます。データがそのプールにある場合、XES に対する妥当性検査が行われます。データが有効であれば、そのデータはローカル・バッファーからアプリケーションに渡されます。データがローカル・バッファー・プールにないか、または XES によってプール内のデータが無効であることが示された場合、デ

ータはカップリング・ファシリティー構造から読み取られて、アプリケーションに渡されます。バッファー・プールが非ルックアサイド・オプションを指定している場合、データに対するすべての要求は、カップリング・ファシリティーに向けられます。

VSO オプションと NOPREL オプションを使用して、要求時ロードで定義されているエリアでは、CI へのアクセスは最初に DASD から行われます。データはデータ・スペースにコピーされ、それから、この CI の後続の読み取りは、DASD ではなく、データ・スペースからデータを取り出します。 VSO オプションと PRELOAD オプションを使用して定義されているエリアでは、CI のアクセスはすべてデータ・スペースから行われます。

データが DASD から取り出されるかデータ・スペースから取り出されるかは、アプリケーション・プログラムが行う処理には意識されません。

出力処理

同期点処理のフェーズ 1 では、VSO データは非 VSO のデータと同じように処理されます。 VSO の使用は、ロギングには意識されません。

同期点処理のフェーズ 2 では、VSO データと非 VSO のデータの取り扱いは異なります。 VSO データでは、更新されたデータがデータ・スペースにコピーされると、ロックが解除されて、使用可能なキューにバッファーが戻されます。更新済みの CI の相対バイト・アドレス (RBA) は、ビットマップとして保持されます。RBA が前回の更新時からすでにビットマップの形である場合は、RBA のコピーが 1 つだけ保持されます。更新済み CI は、インターバル・タイマーの時間で DASD に書き込まれます。複数の更新をバッチ処理することによって、頻繁に更新される CI の出力処理量を減らすことができます。データの書き込み前にデータ・スペース内にデータのコピーが作成されるため、DASD への更新データの書き込み中も、アプリケーション・プログラムからこのデータを読み取ったり、更新したりすることができます。

SHARELVL(2|3) VSO エリアの場合、更新済みの CI をカップリング・ファシリティー構造に書き込むために、出力スレッド処理が使用されます。書き込みが完了すると、ロックが解除されます。 XES は、CI に関するディレクトリー項目内のデータの更新済み状況を保持します。

PRELOAD オプション

1 つのエリアのロードは、他のエリアのロードと非同期に行われます。 1 つのエリアのロードは、そのエリアへのアプリケーション・プログラムのアクセスと並行して行われ (あるいは行うことができ) ます。

アプリケーション・プログラムが要求する CI がデータ・スペースにロードされている場合、CI はデータ・スペースから取り出されます。要求された CI がデータ・スペースにロードされていない場合は、CI は DASD から取得され、UOW ロッキングによってデータ保全性が維持されます。

SHARELVL(2|3) VSO エリアのプリロード・プロセスは、SHARELVL(0|1) の場合の処理と似ています。複数のプリロードは、並行して実行することができ、また、アプリケーション処理と並行して実行することもできます。しかし、ロッキングは異なります。カップリング・ファシリティー構造にロードされた SHARELVL(2|3)

エリアは、UOW ロッキングの代わりに、CI ロッキングを使用します。カップリング・ファシリティーへのロード・プロセスは、一度に 1 つの CI ごとに行われます。

プリロード中に読み取りエラーが発生すると、エラー・メッセージ・フラグによってエラーが通知されますが、プリロード・プロセスは継続します。後続のアプリケーション・プログラムの呼び出しが、読み取りエラーのためデータ・スペースにロードされなかった CI にアクセスした場合、CI の要求は DASD に送られます。読み取りエラーが再び発生した場合、アプリケーション・プログラムは非 VSO のアプリケーションの場合と同様、「A0」状況コードを受け取ります。このときに DASD へのアクセスが正常に行われた場合は、CI がデータ・スペースにロードされます。

関連概念:

『読み取りエラー』

入出力エラー処理

VSO を使用すると、書き込みエラーの発生時にデータの可用性が向上します。

VSO エリアの CI が一度データ・スペースに入れられると、CI への更新を DASD へ書き込む間に書き込みエラーが発生しても、IMS がアクティブな状態である間はそのデータ・スペースから CI にアクセスすることができます。

書き込みエラー:

書き込みエラーが発生すると、エラーのある CI に対して IMS がエラー・キュー・エレメント (EQE) を作成します。

VSO エリアの場合、データ・スペースからデータを読み取ることによって、すべての読み取り要求が満たされます。したがって、そのエリアがデータ・スペース内に存在している間は、書き込みエラーのある CI は引き続き使用可能です。エリアがデータ・スペースから除去されると、CI は使用できなくなり、その CI に対して要求を出すと、「A0」状況コードを受け取ります。

読み取りエラー:

VSO エリアの場合、CI に最初にアクセスすると、CI が DASD から読み取られ、データ・スペースにコピーされます。それ以降のすべての読み取り要求は、データ・スペースから満たされます。データ・スペースで読み取りエラーが発生すると、z/OS は異常終了します。

PRELOAD オプションで定義されている VSO エリアの場合、データは事前にデータ・スペースにロードされます。したがって、すべての読み取り要求はデータ・スペースから満たされます。

可用性を向上するために、SHARELVL(2|3) VSO エリアは、エリアごとに複数の構造をサポートします。 1 つの構造からの読み取りエラーが発生した場合、2 番目の構造からの読み取りが試みられます。構造が 1 つしか定義されていない場合に、読み取りエラーが発生すると、AO 状況コードがアプリケーションに戻されます。

1 つの構造からは、最大 3 つの読み取りエラーが許されます。最大値に到達し、構造が 1 つしか定義されていない場合、そのエリアは停止され、構造は切断されます。

最大値に到達し、構造が 2 つ定義されている場合、エラーが発生した構造は切断されます。残りのもう 1 つの構造が使用されます。構造への書き込みエラーが発生すると、エラーの CI が構造から削除され、DASD に書き込まれます。 CI の削除は、共用パートナーから実行されます。いずれの共用パートナーも構造から CI を削除できない場合、EQE が生成され、CI が非活動化されます。 1 つの構造に対して最大 3 つの書き込みエラーが許されます。 2 つの構造が定義されていて、そのうちの 1 つがエラーの最大値に到達した場合、その構造は切断されます。

関連概念:

257 ページの『PRELOAD オプション』

VSO エリアの CI のキャストアウトしきい値

VSO エリアの CI のキャストアウトしきい値はタイマーです。タイマーが満了すると、CI は VSO エリアからキャストアウトされます。

VSO エリアの CI のキャストアウトしきい値は、CI が VSO エリアからキャストアウトされるまでのタイマーです。 IMS PROCLIB データ・セットの DFSDFxxx メンバーの <SECTION=FASTPATH> セクションで以下のパラメーターを指定することで、さまざまな数の CI を持つ VSO エリアに対してしきい値を構成することができます。

VSO1THLD=
800 以下の制御インターバル (CI) を含む VSO エリアのキャストアウトしきい値時間を秒単位で指定します。有効な値の範囲は、1 から 300 です。デフォルト値は 300 です。

VSO2THLD=
801 から 3500 までの CI を含む VSO エリアのキャストアウトしきい値時間を秒単位で指定します。有効な値の範囲は、1 から 300 です。デフォルト値は 240 です。

VSO3THLD=
3500 を超える CI を含む VSO エリアのキャストアウトしきい値時間を秒単位で指定します。有効な値の範囲は、1 から 300 です。デフォルト値は 180 です。

関連資料:

DFSDFxxx メンバーの FASTPATH セクション (システム定義)

チェックポイント処理

システム・チェックポイントの進行中に、データ・スペースにある VSO エリアの更新のすべてが DASD に書き込まれます。 CF 構造内の更新されているすべての CI も DASD に書き込まれます。

更新されている CI だけが書き込まれます。また、進行中の更新すべてを完了させた後で、チェックポイント処理を継続することもできます。

IMS 再始動後の VSO オプション

XRF のテークオーバー (コールド・スタート、ウォーム・スタート、緊急時再始動、COLDBASE、COLDCOMM、COLDSYS 緊急時再始動) を除く、すべての種類の IMS 再始動で、再始動の後に有効な VSO オプションは、DBRC に定義されているオプションです。

XRF がテークオーバーする場合、テークオーバーの後有効な VSO オプションは、XRF のテークオーバーの原因となった障害の発生前に、アクティブ IMS に対して有効であったオプションと同じです。

緊急時再始動処理

IMS 障害または z/OS 障害での VSO エリアのリカバリーは、VSO エリア以外のリカバリー処理と同様です。IMS は、直前のシステム・チェックポイントからログの終わりまでログ・レコードを調べて、コミットされた更新データのうち、障害の発生前に DASD に書き込まれていないものがあるかどうかを判別します。

このようなコミット済み更新データがあった場合、IMS はその更新を REDO 処理で再実行します (その更新を CI に適用し、更新された CI を DASD に書き込みます)。複数の VSO の更新が、通常の処理時にバッチ処理されるため、VSO エリアに必要な REDO 処理は非 VSO のエリアより多くなる可能性があります。

緊急時再始動のログ処理中、IMS は VSO エリアの共用レベルに応じて異なった方法で VSO エリア更新を追跡します。共用レベル 0 および 1 の場合、IMS はデータ・スペースを使用して VSO エリア更新を追跡します。共用レベル 2 および 3 の場合、IMS はメモリー内のバッファーを使用して VSO エリア更新を追跡します。

IMS は、再始動の終了時に解放される DREF 以外のデータ・スペースも 1 つ取得します。再始動ログ処理で VSO の REDO 処理の実行に必要なデータ・スペースや主記憶域のリソースを取得できない場合、このエリアは停止し、このエリアに「リカバリーが必要である」ことを示すマークが付きます。

デフォルトでは、緊急時再始動の終了時に IMS は PREOPEN オプションまたは PRELOAD オプションで定義されているエリアをオープンします。次に IMS は、PRELOAD オプションで定義されているエリアをデータ・スペースまたはカップリング・ファシリティー構造にロードします。この振る舞いは IMS プロシージャーの FPOPN キーワードを使用して変更することができ、IMS がすべての VSO DEDB エリアを障害発生時のオープンまたはクローズ状態に復元するよう設定することができます。

PREOPEN オプションまたは PRELOAD オプションが指定されていない VSO エリアは、緊急時再始動後の最初のアクセス時にデータ・スペースに割り当てられます。

緊急時再始動の後、エリアに対して有効な VSO オプションと PREOPEN|NOPREO オプションは DBRC に定義されているオプションであり、障害の発生時に有効なオプションとは一致しない場合があります。例えば、障害発生前に /VUNLOAD コマンドによって仮想記憶域から除去された非共用 VSO エリア

は、緊急時再始動後にデータ・スペースに復元されます。共用 VSO エリアの場合は、次の /STA AREA コマンドがエリアに対して出されるまで、エリアはアンロードされたままです。

関連概念:

214 ページの『緊急時再始動時の DEDB エリアの再オープン』

『XRF での VSO オプション』

XRF での VSO オプション

代替 IMS でのトラッキング・フェーズとテークオーバー・フェーズの間、ログ・レコードはアクティブ IMS の緊急時再始動時 (直前の有効なシステム・チェックポイントからログの終わりまで) と同じように処理されます。

代替 IMS は、ログ・レコードを使用して、コミット済みの更新のうち、アクティブ IMS の障害発生前に DASD に書き込まれていないものがどのエリアにあるかを判別します。このようなコミット済みの更新が見つかった場合、代替サブシステムは、アクティブ IMS の緊急時再始動と同じ処理を行った後、REDO によってその更新を再実行します。

代替サブシステムは、トラッキング・フェーズ中にデータ・スペースを使用して、VSO エリアの更新をトラッキングします。代替サブシステムは、VSO エリアで使用されるデータ・スペース・リソースの他に、テークオーバーの終了時に解放される DREF 以外のデータ・スペースを 1 つ獲得します。 XRF のトラッキングとテークオーバーで VSO の REDO 処理の実行に必要なデータ・スペースや主記憶域のリソースを取得できない場合、このエリアは停止し、このエリアに「リカバリーが必要である」ことを示す印が付きます。

XRF のテークオーバー後もオープンしていたエリア、またはデータ・スペースの中にあるエリアは、オープンした状態またはデータ・スペースに入った状態のままです。テークオーバー前にアクティブ IMS に対して有効であった VSO オプションと PREOPEN|NOPREO オプションは、テークオーバー後も代替 IMS (新しくアクティブになったもの) で有効です。これらのオプションは、DBRC に定義されているものと一致しない場合があります。例えば、テークオーバー前に /VUNLOAD コマンドによって仮想記憶域から除去された VSO エリアは、テークオーバー後、データ・スペースに復元されません。

プリロード・オプションで定義されている VSO エリアは、 XRF のテークオーバーの終了時にプリロードされます。ほとんどの場合、従属領域はプリロードが開始する前にエリアにアクセスすることができますが、プリロードが完了するまでいくつかのエリア読み取り要求を DASD から検索しなければならないことがあります。

関連概念:

260 ページの『緊急時再始動処理』

高速機能同期点

MSDB と DEDB は、アプリケーション・プログラムの処理中は更新されませんが、同期点まで更新がバッファーの中に保持されます。出力メッセージは、そのメッセージの応答がログに記録されるまで送られません。

高速機能同期点は、メッセージ・ドリブン・プログラムについては、次の GU 呼び出し、高速機能を使用した BMP については SYNC 呼び出しまたは CHKP 呼び出しとして定義されます。

同期点処理は 2 つの段階で発生します。

関連概念:

233 ページの『混合モードの処理』

段階 1 - ログ・レコードの構築

DEDB の更新と検証された MSDB レコードがシステム・ログ・レコードの中に書き込まれます。現行同期点に対するすべての DEDB の更新は、一連のログ・レコードとしてチェーニングされます。ここで、リソースの競合、デッドロック、スペース不足の状態、および MSDB 検査障害が発見されます。

段階 2 - システム・ログへのレコードの書き込み

データベース・レコードとメッセージ・レコードが、IMS システム・ログに書き込まれます。ロギングの後で、MSDB レコードは更新され、DEDB の更新は開始され、メッセージは端末に送られます。

DEDB の更新は、出力スレッドと呼ばれる一種の非同期処理によって適用されます。 DEDB の変更が行われるまで、書き込まれていないセグメントにアクセスしようとするプログラムはすべて待ち状態に置かれます。

アプリケーション処理中に、高速機能プログラムが MSDB か DEDB 以外のデータベースまたは代替 PCB に対して呼び出しを出すと、この処理は、全機能のイベントと逐次化されます。このことは、高速機能プログラムのパフォーマンスに影響を与えることがあります。 BMP または MPP が高速機能データベースに対して呼び出しを出す場合、高速機能リソースが保持され、これらのリソースを必要とする高速機能プログラムに対するスループットが影響を受けることがあります。

入出力エラーと長時間待ちの管理

データベース書き込み入出力エラーが単一エリア・データ・セット (ADS) で発生した場合、IMS はエラー制御インターバル (CI) のバッファー内容を仮想バッファーにコピーします。その後 DL/I 要求があると、エラーの CI は、バッファー・プールに読み戻されます。

書き込みエラー情報とバッファーは、都合のよい時期までリカバリー操作を延ばせるように、複数の再始動間で維持されます。入出力エラーの再試行は、データベースのクローズ時およびシステム・チェックポイント時に自動的に実行されます。再試行が成功すると、エラー状態はそれ以上続かず、リカバリー操作も不要となります。

データベース読み取り入出力エラーが発生すると、IMS は、エラーの RBA を記録するエリア・データ・セット (ADS) に関連した非永続 EQE を作成します。他の ADS が使用可能になっている場合、IMS は、他の ADS を使用して読み取りを再試行します。単一 ADS のみがある場合、またはすべての ADS について読み取りが失敗する場合、アプリケーション・プログラムは「AO」状況コードを受け取りま

す。 EQE があると、この ADS 内の同じ CI へのその後のアクセスは阻止されます。この CI にアクセスしようとすると、即時入出力エラー表示を受け取ります。

MADS の場合、入出力は別の ADS に対して試みられます。 ADS ごとに最大 3 個の別個の入出力エラーを記録できます。 4 番目のエラーが発生すると、IMS は内部的に ADS を停止します。これがそのエリア用の唯一の ADS である場合、そのエリアは停止されます。

EQE は一時的なものであって、IMS の再始動後、またはエリアをオープンしてクローズした後には残っていません。 EQE は DBRC、または DASD 上の DMAC には記録されません。書き込みエラーが発生すると、読み取り EQE は除去され、カウンターはリセットされます。

読み取り EQE を作成する処理では、DASD から DMAC (2 番目の CI) も読み取ります。 DMAC 読み取りが失敗した場合 (これは、障害が装置レベルの場合に起こる可能性があります)、IMS は ADS を内部的に停止します。ADS 停止処理はエリアの物理的なクローズを伴い、それにより、DMAC (2 番目の CI) の書き込みが行われます。この処理が失敗すると、クローズされる ADS がそのエリア用の唯一の ADS である場合、そのエリアは停止され、DBRC の中に「リカバリーが必要」というフラグが立てられます。

多重エリア・データ・セット入出力タイミング (MADSIOT) により、RAMAC ディスク・アレイが内部リカバリー処理を行う間に発生する可能性のある長時間待ち (長時間使用中 とも言います) を回避できます。

制約事項**:** MADSIOT は多重エリア・データ・セット (MADS) にのみ適用されます。単一エリア・データ・セット (ADS) の場合は、IMS は長時間使用中状態を、高速機能入出力許容機能によって処理される永続入出力エラーとして扱います。MADSIOT 機能は、長時間使用中状態をサポートするシステムでのみ作動します。

MADSIOT を呼び出すには、PROCLIB メンバー DFSVSMxx で MADSIOT キーワードを定義する必要があります。 /STA MADSIOT および /DIS AREA MADSIOT コマンドは、MADSIOT 機能を開始し、モニターする役を果たします。

さらに、MADSIOT は、シスプレックス環境でカップリング・ファシリティー (CFLEVEL=1 以上) リスト構造が使用されていることを必要とします。 MADSIOT は、このカップリング・ファシリティーを使用して、DB リカバリーに必要な情報を保管します。 CFRM ポリシーを使用して、リスト構造の名前、サイズ、属性、および位置を定義してください。

以下の表は、CFLEVEL=12 のとき、変更された CI の数が 1 000、5 000、20 000、および 30 000 の場合に必要な CFRM リスト構造ストレージ・サイズを示しています。 CFLEVEL が異なれば、ストレージ・サイズは異なります。

表 *56. CFLEVEL=12* の場合に必要な *CFRM* リスト構造ストレージ・サイズ

変更された **CI** の数 **(entrynum)**	必要なストレージ・サイズ **(listheadernum=50)**
1 000	1 792 KB
5 000	3 584 KB
20 000	11 008 KB

表 56. CFLEVEL=12 の場合に必要な CFRM リスト構造ストレージ・サイズ (続き)

変更された **CI** の数 **(entrynum)**	必要なストレージ・サイズ **(listheadernum=50)**
30 000	15 616 KB

ご使用のシステムに合わせた CFRM リスト構造ストレージ・サイズは、オンライン・ツールである IBM System z CFSizer (Coupling Facility Structure Sizer Tool) で見積もることができます。CFSizer は、www.ibm.com/servers/eserver/zseries/cfsizer/ の Web サイトで入手できます。あるいは、IBM Web サイト (www.ibm.com) で「CFSizer」を検索してください。

関連タスク:

- CFRM ポリシーの定義 (システム管理)

関連資料:

- /START MADSIOT コマンド (コマンド)
- /DISPLAY AREA コマンド (コマンド)
- IMS PROCLIB データ・セットの DFSVSMxx メンバー (システム定義)

DBRC への高速機能データベースの登録

高速機能データベースを DBRC に登録することは、エラー処理を機能させる上で必須ではありませんが、登録することを強くお勧めします。場合によっては、登録が必須になります。

登録されていないデータベースでエラーが発生してシステムが停止した場合、システムを再始動して、データベースにアクセスする前に /DBR コマンドを発行しないと、データベースに損傷を与える場合があります。再始動によりエラー・バッファーの内容がシステム停止時の状態に復元されるからです。

以下のシチュエーションでは、高速機能データベースを DBRC に登録する必要があります。

- 仮想記憶オプション (VSO) (Virtual Storage Option (VSO))
- DEDB エリアの事前オープン
- VSO DEDB エリア制御インターバル (CI) のプリロード
- 高速順次処理 (HSSP) イメージ・コピー (IC)
- 多重エリア・データ・セット (MADS)
- フル・セグメント・ロギング
- データベースの静止
- データ共用
- z/OS カップリング・ファシリティー構成上に存在する共用仮想記憶オプション (SVSO) エリア

第 15 章 論理関係の作成

アプリケーション・プログラムがデータベース内のセグメントを見る必要があるときに、その見方が互いに矛盾することがあります。この矛盾を解消するのが論理関係です。

論理関係を使用することにより、アプリケーション・プログラムは以下のことができます。

- 階層によって定義された順序とは異なる順序で、複数のセグメント・タイプにアクセスできます。
- 複数の物理データベースからのセグメントが入っている 1 つのデータ構造にアクセスできます。

アプリケーションの互いに相異なる要件を解決するのに論理関係以外の手段に頼るとすれば、別個の複数のデータベースを作成するか、単一のデータベースの中に重複データをもつ必要があります。ところが、このいずれの場合にもこれによって重複データが生じます。次の理由により、重複データは避けるようにしてください。

- 重複データがあると、両方のデータ・セットを最新の状態に維持しなければならないため、維持管理に余分な手間がかかります。また、データの一貫性を確保するため、更新を同時に行わなければなりません。
- 重複データを保持するために、DASD 上に余分なスペースが必要です。

2 つのセグメント・タイプ間のパスを設けることによって、論理関係は重複データを保管する必要を取り除きます。

この論理関係を設定するには、常に 3 つのセグメント・タイプが定義されます。

物理親

論理親

論理子

以下のデータベース・タイプが論理関係をサポートします。

- HISAM
- HDAM
- PHDAM
- HIDAM
- PHIDAM

2 つのデータベースがあります。1 つは客先からの注文を記録するためのものであり、もう 1 つは受注可能品目を記録するためのものです。最初のデータベースは ORDER と呼ばれ、2 番目のデータベースは ITEM と呼ばれます。 ORDER データベースには、客先、注文、および配送に関する情報が入っています。 ITEM データベースには、在庫に関する情報が入っています。

1 つのアプリケーション・プログラムがこの両方のデータベースの中のデータを必要とする場合には、この 2 つのデータベースの間の論理関係を定義することによっ

てそれが可能になります。以下の図に示すように、ITEM データベースの中を指す、論理子セグメントと呼ばれるセグメント・タイプを 1 つ使用することにより、ORDER データベースと ITEM データベースの間に 1 つのパスを設定することができます。以下の図は論理関係の簡単な使用例です。この場合、ORDER は ORDITEM の物理親です。 ORDITEM は ORDER の物理子で、ITEM の論理子でもあります。

論理関係においては、論理親セグメント・タイプが存在します。これは、論理子によって指されているセグメント・タイプです。この例では、ITEM は ORDITEM の論理親です。 ORDITEM によって、2 つのセグメント・タイプの間のパスすなわち、接続ができます。今、あるアプリケーション・プログラムが ORDER データベースに入ったとすると、このプログラムは、ORDER データベースから ITEM データベースへと、この論理子セグメントの中のポインターに従って進むことにより、ITEM データベースの中のデータにアクセスすることができます。

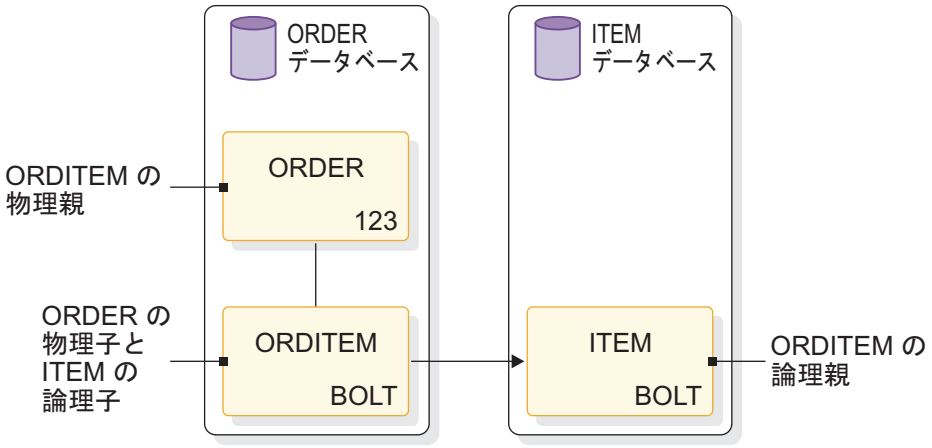


図 *69.* 簡単な論理関係

物理親と論理親は、間にパスが設定される 2 つのセグメント・タイプです。論理子は、このパスを設定するセグメント・タイプです。論理子によって設定されるパスは、ポインターを用いて作成されます。

関連概念:

809 ページの『論理関係の追加』

副次索引対論理関係

副次索引と論理関係は、どちらも論理データ構造を備えており、それらの論理データ構造は、物理 DBD によって表されるデータ構造とは異なった階層データ構造です。どちらのタイプを使用するかは、主にアプリケーション・プログラムに必要なデータ処理の方法によって決まります。高速機能は、副次索引だけをサポートしています。

副次索引を使用すべき場合

アプリケーションの要件を分析してみて、1 つのセグメントのシーケンス・フィールドとして候補が複数の場合は、副次索引を使用してください。物理 DBD 内に定義されるシーケンス・フィールドをまず 1 つ選択します。次に、同じセグメントを

別の順序で処理できるようにするため副次索引を設定します。以下の図に示す例の場合、客先番号 (CUSTNO) と客先名 (CUSTNAME) の順になっている客先 (CUSTOMER) セグメントをアクセスするとします。これを行うために、物理 DBD 内のシーケンス・フィールドとして CUSTNO を定義し、次いで CUSTNAME 順で CUSTOMER セグメントを処理する副次索引を定義することができます。

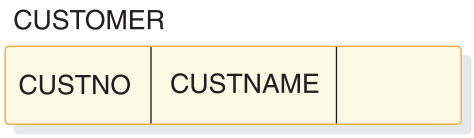


図 *70. CUSTOMER* セグメントのフィールド

論理関係を使用すべき場合

再帰構造を使用している部品表のようなアプリケーションがある場合には、論理関係を使用してください。 再帰構造が存在するのは、同じ物理階層内にある 2 つのセグメント相互の間に多対多の関連がある場合です。例えば、以下の図に示すセグメントでは、「自動車」という組み立て品は多数の部品で構成されており、そのうちの 1 つにエンジンがあります。しかし、エンジンそれ自体が多数の部品で構成されている組み立て品です。

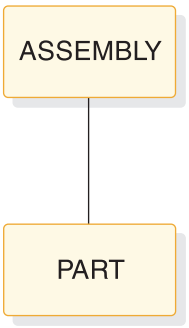


図 *71.* セグメントの論理関係を説明する例としての *ASSEMBLY* と *PARTS*

最後に、アプリケーションの要件により結果として 1 つのセグメントが 2 つの親を持つように見えることがあります。以下の図の例では、客先データベースが注文 (CUSTORDN) を把握しています。各注文は、1 つ以上の製品系列項目 (ORDLINE) を持つことができ、各製品系列項目で 1 つの製品 (PROD) と型 (MODEL) が指定されます。製品データベースにおいては、製品系列項目の多数の未処理の要求が、ある 1 つの型に対して存在します。この種の関係は、多対多の関係と呼ばれ、IMS では論理関係によって処理されます。

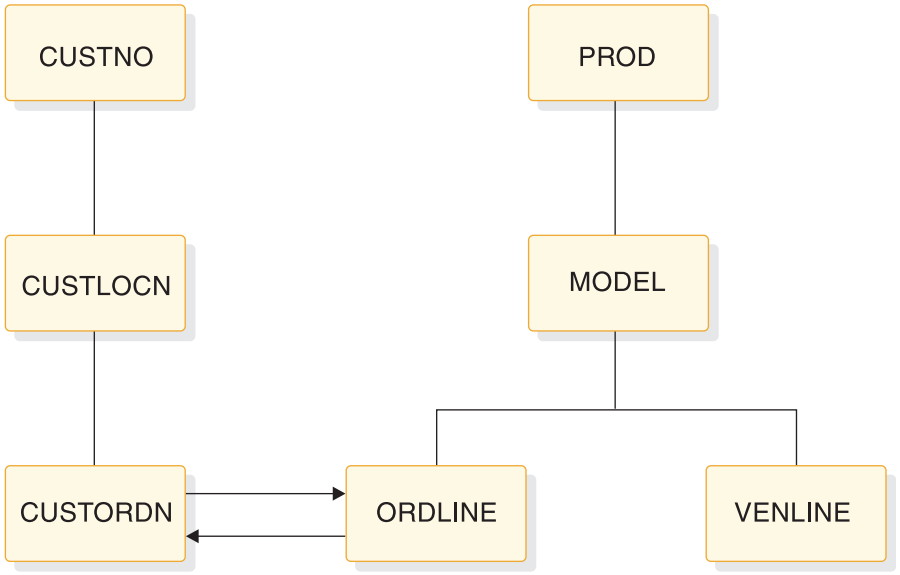


図 72. 2 つの親を持つように見えるセグメントの例

関連概念:

290 ページの『再帰構造: 同一データベース論理関係』

論理関係の種類

このトピックでは、次の 3 種類の論理関係について説明します。

論理関係には次の 3 つのタイプがあります。

1 方向論理関係

物理対の両方向論理関係

仮想対の両方向論理関係

1 方向論理関係

単一方向論理関係とは、2 つのセグメント・タイプ、すなわち、論理子とその論理親を 1 つの方向にリンクするものです。単一方向パスは、論理子の中の 1 つのポインターを用いて設定されます。以下の図は、ORDER および ITEM 両データベース間に設定された単一方向論理関係を示したものです。同一のデータベースまたは異なるデータベースの中の 2 つのセグメント・タイプの間に単一方向論理関係を設定することができます。しかし一般的には、異なるデータベースに属する 2 つのセグメント・タイプの間に、単一方向論理関係が作成されます。この図ではこの論理関係を用いて、ORDER データベースから ITEM データベースに渡ることができます。この論理関係を用いて、ITEM データベースから ORDER データベースに渡ることはできません。ITEM セグメントは ORDER データベースを指していないからです。

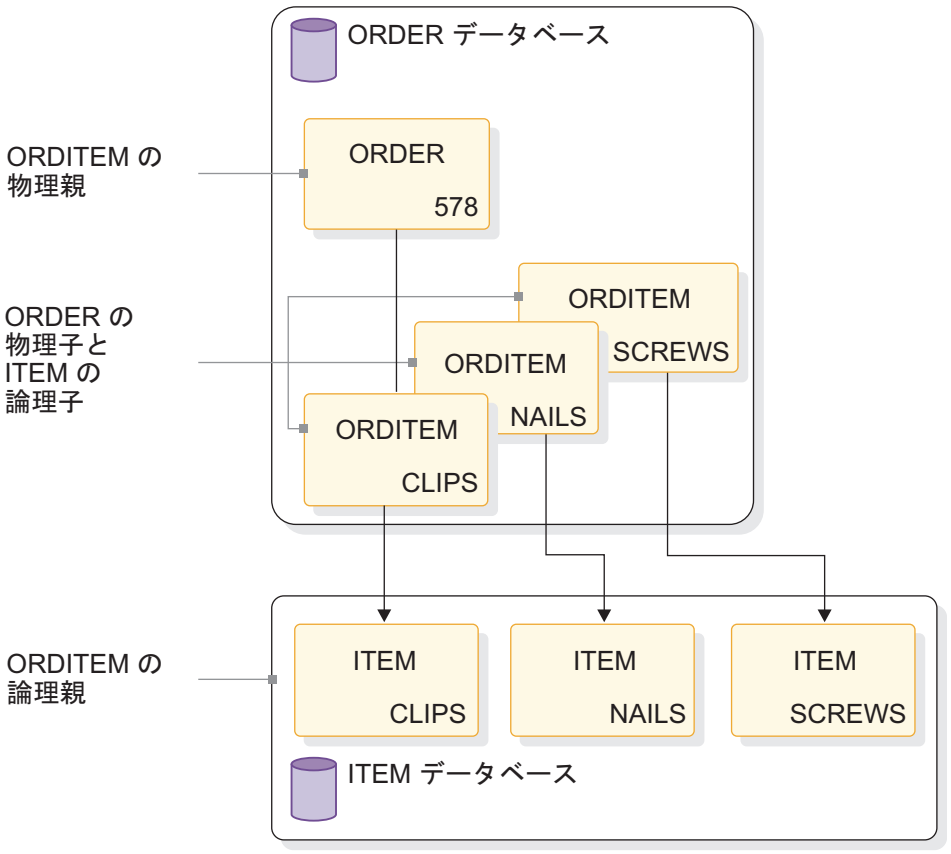


図 73. 単一方向論理関係

以下の図に示すように、2 つの単一方向関係を設定することは可能です。こうすれば、いずれの物理データベースに入っても、それぞれ論理子を用いて、他方の物理データベースに渡ることができます。しかし、IMS は、このそれぞれの単一方向関係を単一方向パスとして扱います。それによって双方のパスでデータが維持されるわけではありません。したがって、一方のデータベースの中のデータの挿入、削除、または置き換えが行われても、もう一方のデータベースの中のこれに対応するデータは更新されません。例えば、DL/I が ORDER 578 の下にある ORDITEM SCREWS を置き換えても、ITEM SCREWS の下にある ITEMORD 578 は置き換えられません。物理対の両方向論理関係と仮想対の両方向論理関係においては、このような維持管理の問題は存在しません。両方の種類の関係について、次に説明します。IMS により、どちらかの物理データベースに入ってデータベースを更新することができ、他方のデータベース内の対応するデータが自動的に更新されます。

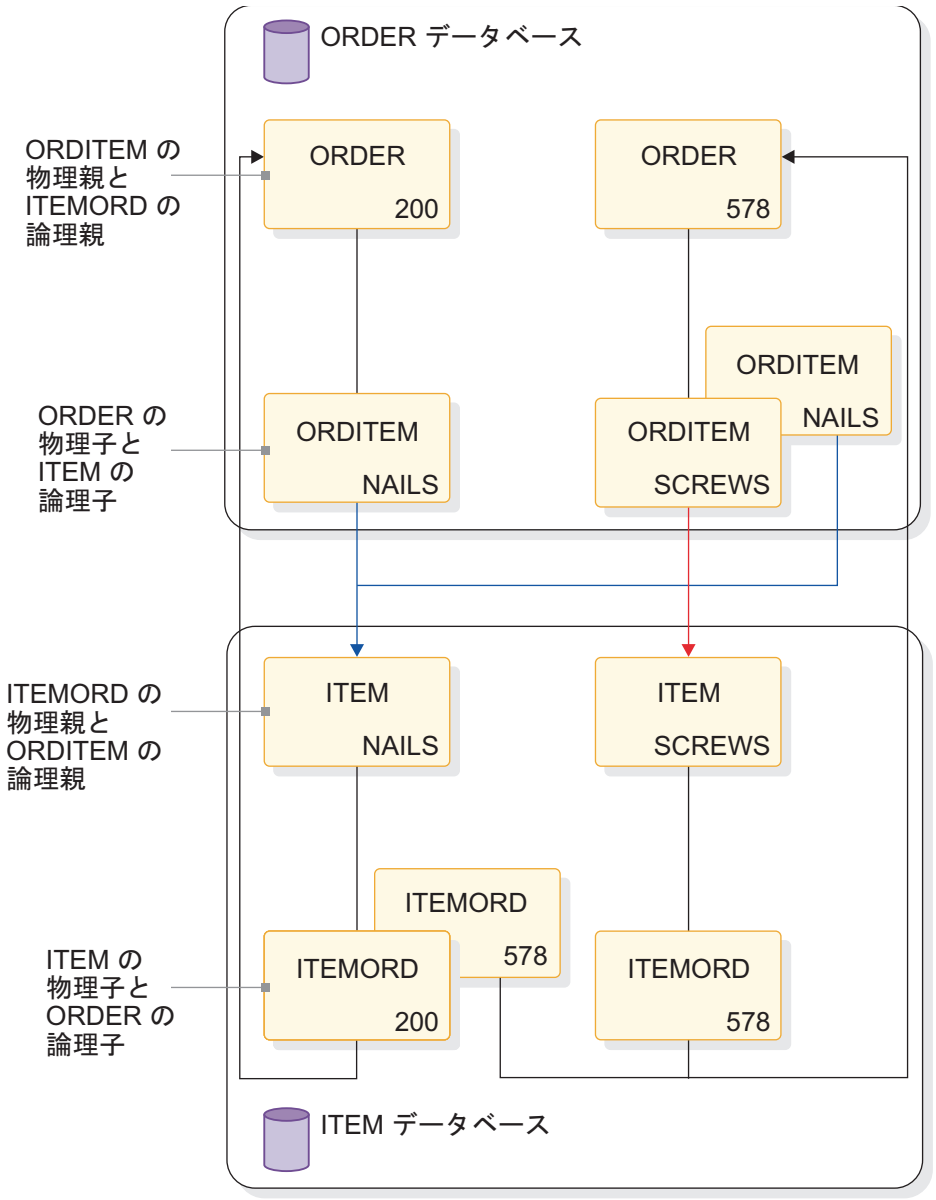


図 74. 2 つの単一方向論理関係

物理対の両方向論理関係

物理対の両方向論理関係とは、2 つのセグメント・タイプ、すなわち、論理子とその論理親を 2 つの方向にリンクするものです。両方向パスは、論理子セグメントの中のポインターを用いて設定されます。以下の図は、ORDER および ITEM 両データベース間に設定された物理対の両方向論理関係を示したものです。

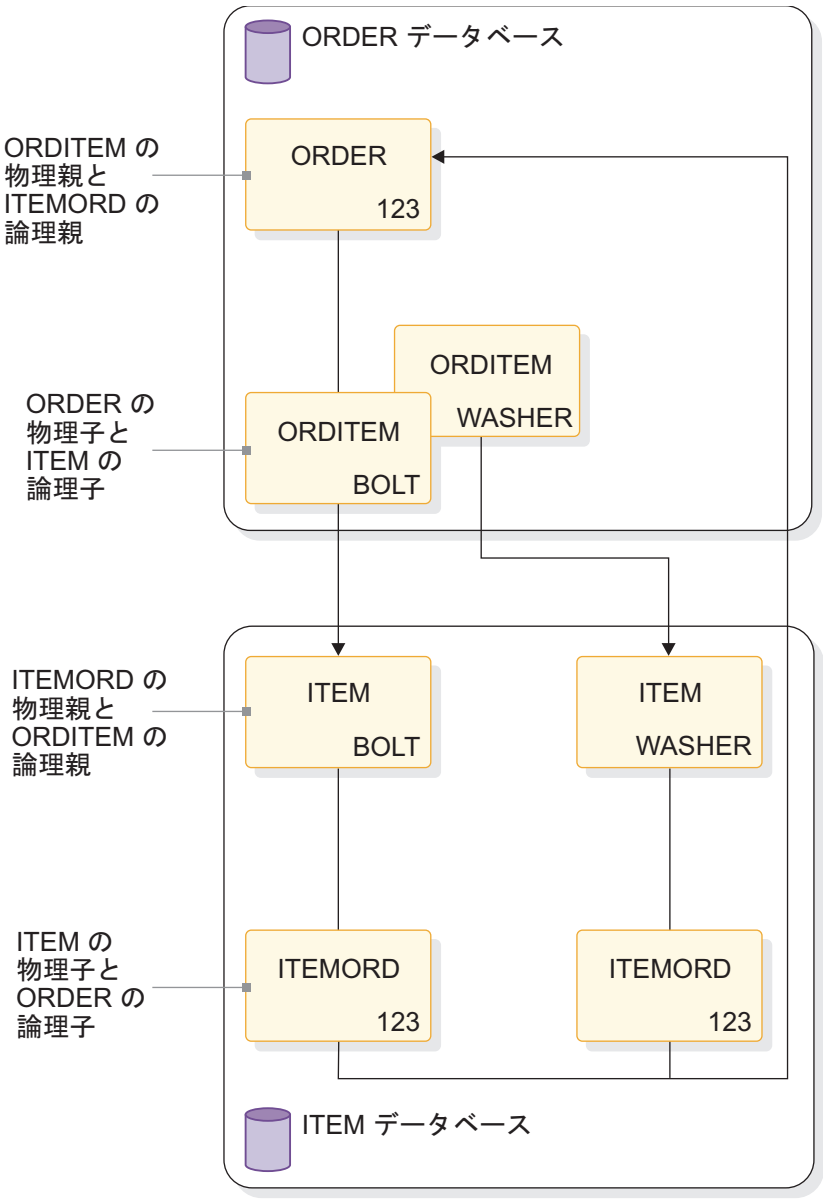


図 75. 物理対の両方向論理関係

他の種類の論理関係と同様、物理対の関係も、同一のデータベースまたは異なるデータベースに属する 2 つのセグメント・タイプの間に設定することができます。上の図の関係を用いると、ORDER データベースあるいは ITEM データベースのいずれにでも入ることができます。いずれのデータベースに入った場合にも、論理子を用いたパスが存在し、これによって一方のデータベースから他方のデータベースに渡ることができます。

物理対の関係においては、1 つの論理子が両方のデータベースの中に保管されます。しかし、その論理子が従属セグメントを持つ場合、この従属セグメントは一方のデータベースにしか保管されません。例えば、IMS は、物理対の関係における両方のパス・データを維持します。上の図において、ORDER 123 を ORDER データベースから削除したとすれば、IMS はこの ORDER 123 セグメントを指すすべての ITEMORD セグメントを ITEM データベースから削除します。ある論理子セグ

メントにおいてデータが変更されると、IMS は、このセグメントと対をなす論理子セグメントの中のこのデータを変更します。あるいは、論理子セグメントが一方のデータベースの中に挿入されると、IMS はこれと対をなす論理子セグメントを他方のデータベースの中に挿入します。

物理対を使用すると、論理子は重複データとなるので、ストレージの必要量が多少増えます。しかも IMS が 2 つのパス・データを維持するので、余分な維持管理が必要です。次に見る種類の論理関係においては、この余分なスペースと維持管理がありません。それにもかかわらず、IMS は、ユーザーがいずれのデータベースにも入ることを可能にします。また、IMS はユーザーに代わって維持管理も行います。

仮想対の両方向論理関係

仮想対の両方向関係は、以下の点で、物理対の両方向関係と似ています。

- 2 つのセグメント・タイプ、すなわち、論理子とその論理親を 2 つの方向にリンクし、これによって 1 つの両方向パスを設定します。
- 同一のデータベースまたは異なるデータベースに属する 2 つのセグメント・タイプの間に設定することができます。

以下の図は、ORDER データベースと ITEM データベースとの間に設定された仮想対の両方向論理関係を示したものです。 1 つの両方向パスがありますが、論理子セグメントは ORDER データベースにしか存在しないという点に注意してください。ORDER データベースから ITEM データベースに行くには、IMS は論理子セグメントの中のポインターを使用します。 ITEM データベースから ORDER データベースに行くには、IMS は論理子セグメントの中のポインターだけでなく、論理親の中のポインターも使用します。

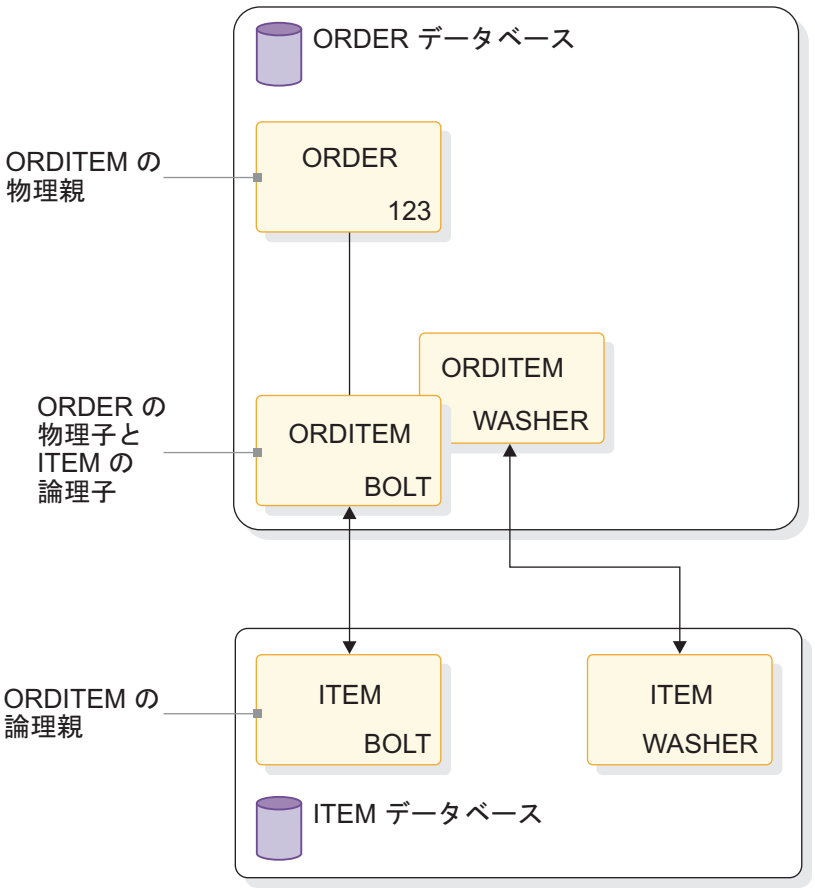


図 *76.* 仮想対の両方向論理関係

仮想対の関係を定義するには、この論理関係に関与する物理データベースに 2 つの論理子セグメント・タイプを定義します。 1 つの論理子だけが実際にストレージの中に置かれます。定義され、しかもストレージの中に置かれる論理子は実論理子 と呼ばれます。定義されるがストレージの中に置かれない論理子は仮想論理子 と呼ばれます。

IMS は仮想対の関係における両方のパスのデータを維持します。しかし、論理子セグメントが 1 つしかないので、物理対の論理関係の場合より、簡単に維持管理できます。例えば、新しい 1 つの ORDER セグメントが挿入された場合、 1 つの論理子セグメントを挿入するだけですみます。置き換えでは、1 つのセグメントの中のデータだけを変更すればよいのです。削除では、この論理子セグメントが両方のパスから削除されます。

物理対と仮想対の間には、トレードオフの関係があることに注意してください。仮想対では、重複論理子は存在せず、対を成す論理子の維持管理もありません。しかし、仮想対では、論理兄弟ポインターと呼ばれる余分なポインターを使用しこれを維持管理する必要があります。

関連資料:

87 ページの『LCHILD セグメント・タイプのフォーマット』

論理関係ポインターの種類

どの論理関係においても、論理子によって 2 つのセグメント・タイプの間にパスが設定されます。このパスは、ポインターを用いることによって設定されます。

HALDB データベースの場合、以下に挙げることを考慮してください。

- HALDB データベースと非 HALDB データベースとの間の論理関係は認められません。
- 直接ポインターと間接ポインターが使用されます。
- HALDB データベースでは、単一方向関係および物理対の両方向関係がサポートされています。
- 物理親ポインターは、PHDAM および PHIDAM セグメントに常に存在します。

論理親ポインター

論理子からその論理親を指すポインターは、論理親 (LP) ポインターと呼ばれます。このポインターが HISAM データベースの中を指している場合には、このポインターはシンボリック・ポインターでなければなりません。 このポインターが HDAM または HIDAM データベースの中を指している場合には、このポインターは直接ポインターあるいは記号ポインターのいずれでもかまいません。 PHDAM または PHIDAM データベースは、直接ポインターが必要です。

まず直接ポインターについて見ましょう。直接ポインターはこのポインターが指すセグメントの直接アドレスで構成されており、この種のポインターを用いて指すことができるデータベースは、いったん保管されたセグメントが動かされることのないようなデータベースです。これは、論理親セグメントが HD (HDAM、PHDAM、HIDAM、および PHIDAM) データベースの中になければならないことを意味しています。このセグメントが、論理子によって指される論理親であるからです。この種のポインターが入っている論理子セグメントは、HALDB の場合を除いて、HISAM データベースまたは HD データベースの中に置かれます。 HALDB の場合、論理子セグメントは HD (PHDAM または PHIDAM) データベースの中になければなりません。直接 LP ポインターは、論理子の接頭部の中に他のポインターと共に保管され、その長さは 4 バイトです。以下の図は直接 LP ポインターの用法を示したものです。 HISAM データベースでは、セグメントが、階層順に相互に物理的隣接して保管されるため、セグメント相互間のポインターは必要ありません。したがって、直接ポインターが HISAM データベースの中に存在する唯一の場合は、HD データベースの中を指す直接ポインターを使用している論理関係が存在する場合です。

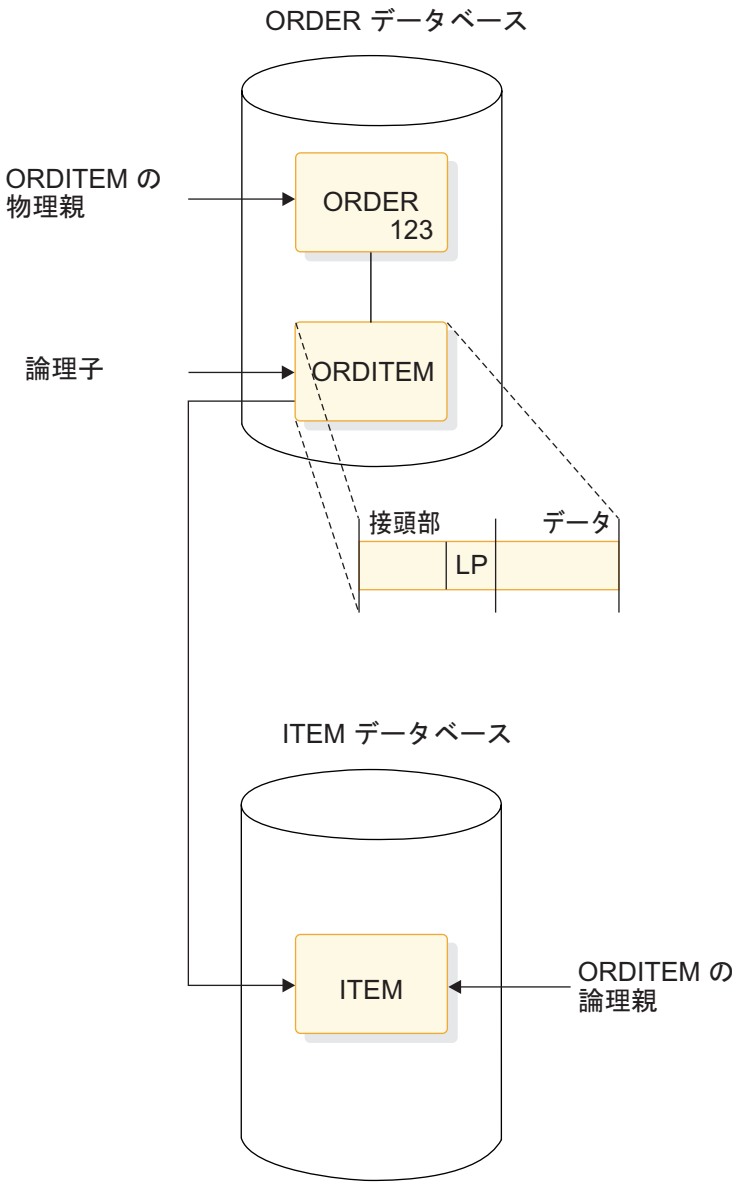


図 77. 直接論理親 (LP) ポインター

前の図では、直接 LP ポインターが、論理子 ORDITEM から論理親 ITEM を指しています。直接ポインターであるので、この LP ポインターは HD データベースしか指せません。しかし、この LP ポインターは HISAM データベースや HD データベースの中に『存在』することができます。この LP ポインターは、論理子の接頭部の中にあり、論理親の 4 バイトの直接アドレスで構成されています。

論理親の連結キー (LPCK) で構成される記号 LP ポインターを用いると、HISAM データベースまたは HD データベースの中を指すことができます。 以下の図は、記号 LP ポインターの用法を示したものです。論理子 ORDITEM は、BOLT に対する ITEM セグメントを指します。したがって、BOLT が ORDITEM の LPCK の中に入れられます。記号 LP ポインターは、論理子セグメントのデータ部の先頭に保管されます。

注**:** 論理子セグメントの LPCK 部は交換不可能と見なされ、入出力域が変更されたかどうかの検査は行われません。仮想的な LPCK の場合、入出力域内に変更があったかどうかを検査すると、パフォーマンスの問題が生じます。入出力域内の LPCK を変更しても、REPL 呼び出しが失敗する原因とはなりません。しかし、論理子セグメントの中では、LPCK は変更されません。

シンボリック・ポインターを使用する場合、論理親が属するデータベースが HISAM または HIDAM であるならば、IMS は、シンボリック・ポインターを用いて索引にアクセスし、該当する論理親セグメントを見つけます。 論理親が入っているデータベースが HDAM であれば、論理親セグメントを見つけるためには、ランダム化モジュールによって、シンボリック・ポインターをブロックと RAP アドレスに変えなければなりません。直接ポインターが使用されている場合のほうが IMS はより速く論理親にアクセスします。

図では LP ポインターを単一方向論理関係として示してありますが、このポインターは 3 種類すべての論理関係において同じように働きます。

以下の図は、記号論理親ポインターの一例を示しています。

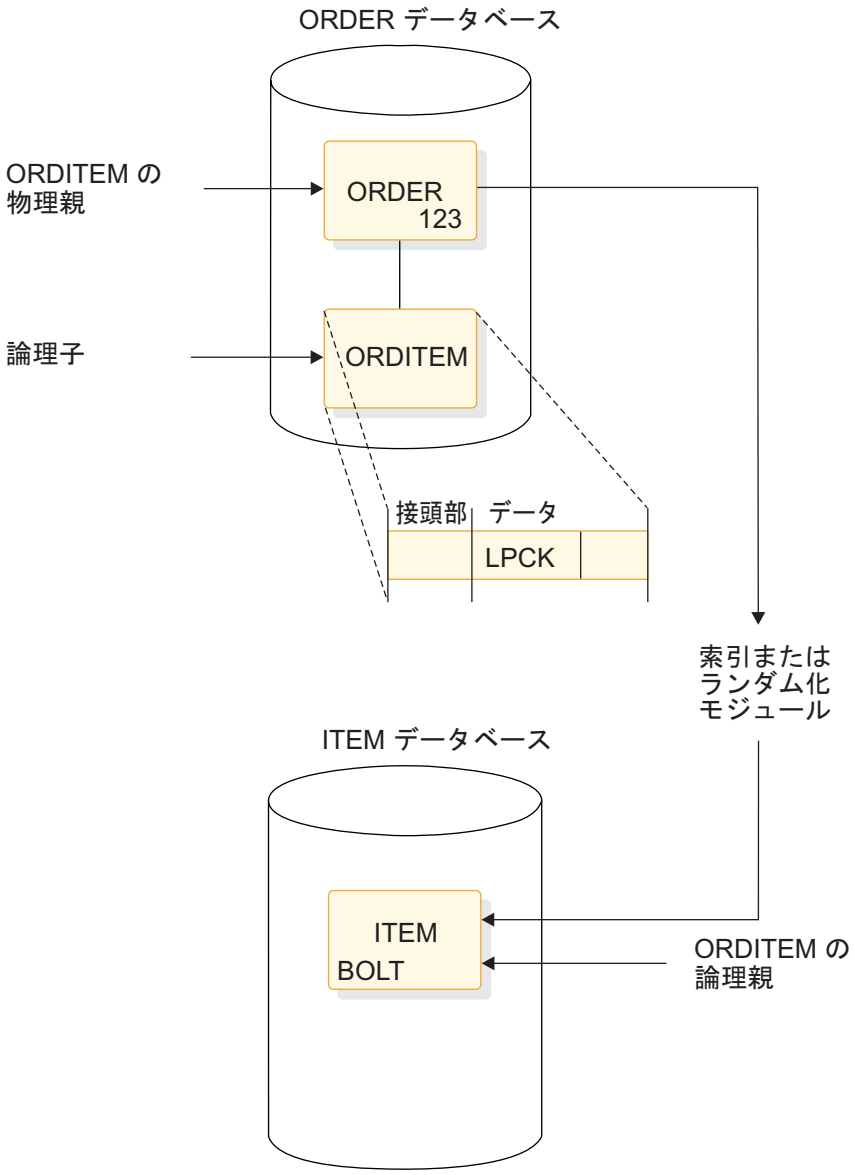


図 *78.* 記号論理親 *(LP)* ポインター

前の図では、シンボリック LP ポインターが、論理子 ORDITEM から論理親 ITEM を指しています。シンボリック・ポインターを使用して指す場合、ORDER と ITEM データベースは、HISAM または HD であることが可能です。論理子のデータ部の先頭にある LPCK は、論理子から論理親を指すポインターとして機能し、論理子の中で使用されます。

論理子ポインター

論理子ポインターが用いられるのは、仮想対の論理関係においてのみです。仮想対が用いられている場合、DASD 上には、実論理子と呼ばれる論理子が 1 つしかありません。この論理子は LP ポインターを備えています。この LP ポインターは、シンボリック・ポインターでも直接ポインターでもかまいません。これまで見てきた ORDER データベースと ITEM データベースにおいては、LP ポインターを使用することによって、論理子が入っているデータベースから、論理親が入っているデ

ータベースに行くことができました。しかし、いずれか一方のデータベースに入って、仮想対を用いて他方のデータベースに渡るためには、論理親の中の論理子ポインターを用います。以下の 2 種類の論理子ポインターを使用することができます。

- 論理第 1 子 (LCF) ポインター、または
- 論理第 1 子 (LCF) ポインターと、論理最終子 (LCL) ポインターの組み合わせ

LCF ポインターは、ある 1 つの論理親から、その各論理子セグメント・タイプの最初のオカレンスを指します。 LCL ポインターは、このポインターの対象として指定されている論理子セグメント・タイプの最後のオカレンスを指します。 LCL ポインターは、LCF ポインターと一緒でなければ指定できません。以下の図は、LCF ポインターの用法を示したものです。このポインターを使用すると、ITEM データベースから ORDER データベースの論理子 ORDITEM に渡ることができます。しかし、論理子ポインターを使用してデータベースからデータベースに渡ることはできますが、ITEM から論理子 ORDITEM にたどり着いたにすぎません。ORDER セグメントに到達するには、物理親ポインターを使用します。

LCF ポインターと LCL ポインターは直接ポインターです。 これらには、このポインターが指すセグメントの 4 バイトの直接アドレスが入っています。これは、このポインターが指すセグメントである論理子セグメントが、 HD データベースの中になければならないことを意味しています。論理親は、HISAM データベースまたは HD データベースいずれの中にあってもかまいません。論理親が HISAM データベースの中にある場合には、論理子セグメントは、シンボリック・ポインターを用いてこの論理親を指さなければなりません。 LCF ポインターと LCL ポインターは、論理親の接頭部の中に他のポインターと共に保管されます。以下の図は LCF ポインターを示しています。

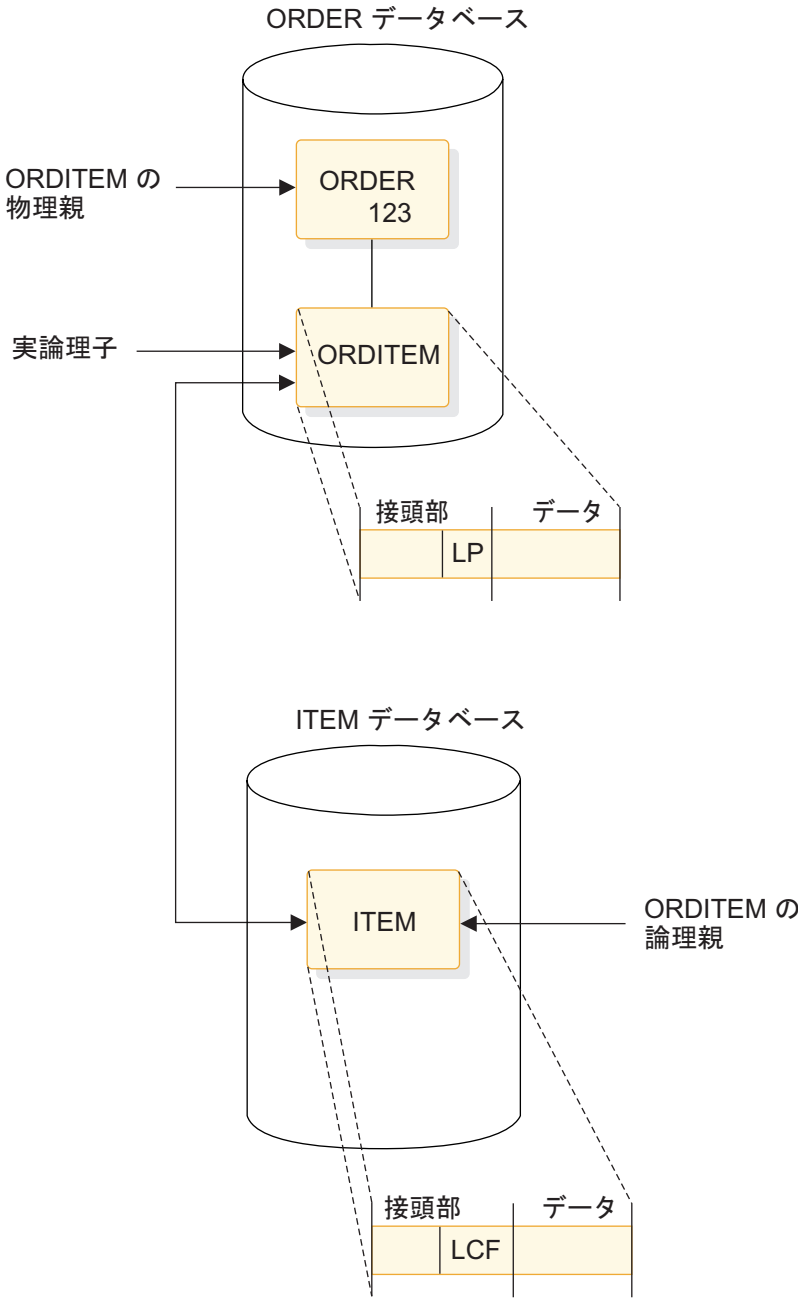


図 79. 論理第 1 子 (LCF) ポインター (仮想対の論理関係においてのみ使用)

前の図では、LCF ポインターが、論理親 ITEM から論理子 ORDITEM を指します。これは直接ポインターであるので、このポインターは、HD データベースしか指せません。しかし、このポインターは HISAM データベースまたは HD データベースいずれの中にあってもかまいません。 LCF ポインターは、論理親の接頭部の中にあり、論理子の 4 バイトの RBA で構成されています。

物理親ポインター

物理親 (PP) ポインターは、あるセグメントからその物理親を指します。このポインターは、論理関係に関与しているすべての HD データベースのために、IMS が自動的に生成します。 PP ポインターは、すべての論理子と論理親セグメントの接

頭部の中に置かれます。また、このポインターは論理子セグメントまたは論理親セグメントがその物理的なデータベースの中で従属しているすべてのセグメントの接頭部の中にも置かれます。これによって、論理子またはその論理親から、逆方向にこれが従属しているルート・セグメントに至るパスができます。ある 1 つの論理子または論理親が従属するすべてのセグメントが、 PP ポインターによって 1 つのルート・セグメントにチェーニングされているので、通常の順序とは逆の順序でこれらのセグメントにアクセスすることが可能です。

前の図では、仮想対が使用されている場合に、ITEM データベースから ORDER データベースに横断できることを確認しました。これには、論理子ポインターが使用されていました。ところが、論理子ポインターに従って行けるのは、ITEM から論理子 ORDITEM までにすぎません。以下の図は ORDER まで行くにはどうしたらよいかを示したものです。 ORDITEM の中の PP ポインターは、その物理親 ORDER を指します。 ORDER と ITEM が HD データベースの中に入っているがルート・セグメントでない場合は、これらのセグメント (およびこのルートのパスにある他のすべてのセグメント) にも、各物理親を指す PP ポインターが入っています。

PP ポインターは直接ポインターです。これらには、このポインターが指すセグメントの 4 バイトの直接アドレスが入っています。 PP ポインターは、論理子または論理親の接頭部の中に他のポインターと一緒に保管されます。

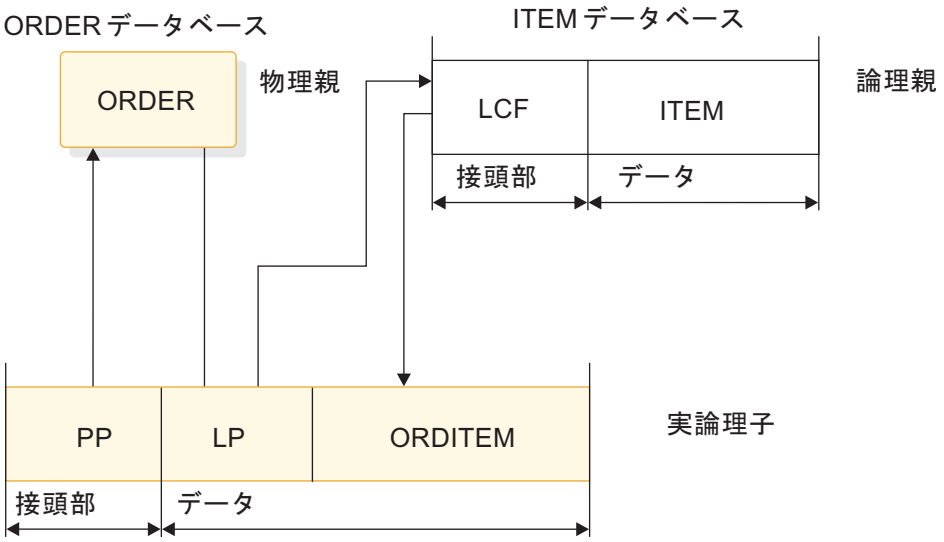


図 *80.* 物理親 *(PP)* ポインター

前の図では、PP ポインターが論理子 ORDITEM からその物理親 ORDER を指しています。このポインターは、HD データベースの中のすべての論理子セグメントと論理親セグメントのために、IMS が自動的に生成します。また、このポインターは、このポインターが入っているセグメントの接頭部の中にあり、その物理親の 4 バイトの直接アドレスで構成されています。 PP ポインターは、論理子または論理親から、逆方向にルート・セグメントに至るまでのすべてのセグメントに生成されます。

論理兄弟ポインター

論理兄弟ポインターが用いられるのは、仮想対の論理関係においてのみです。論理兄弟とは、ある 1 つの論理親の同一のオカレンスを指す複数の論理子セグメントのことです。以下の 2 種類の論理兄弟ポインターを使用することができます。

- 論理兄弟順方向 (LTF) ポインター、または
- 論理兄弟順方向 (LTF) ポインターと論理兄弟逆方向 (LTB) ポインターの組み合わせ

LTF ポインターは、ある特定の論理兄弟から、その後ろに保管されている論理兄弟を指します。 LTB ポインターは、LTF ポインターと一緒でなければ指定することができません。 LTB ポインターが指定されている場合、このポインターは、ある論理兄弟から、その前に保管されている論理兄弟を指します。論理兄弟ポインターは、HD データベースの中で使用される物理兄弟ポインターと同様の機能を持ちます。物理兄弟逆方向ポインターと同様、LTB ポインターを用いると、削除操作のパフォーマンスが向上します。これは、DASD スペースを解放させる削除が物理的なアクセス・パスから行われる場合です。同様に、DASD スペースを解放させる削除が論理的なアクセス・パスから行われる場合には、PTB ポインターを用いるとパフォーマンスが向上します。

以下の図は、LTF ポインターの用法を示したものです。この例の場合、ORDER 123 には 2 つの品目があります。 BOLT (ボルト) と WASHER (ワッシャー (座金)) です。この 2 つの ITEM セグメントの下にある ITEMORD セグメントで、 LTF ポインターが使用されます。 ORDER データベースに入った場合、このデータベースを経由して、ITEM データベースの中の BOLT の ITEMORD セグメントに渡ることができます。次いで、ORDER 123 のすべての品目を取り出したい場合には、ITEMORD セグメントの中の LTF ポインターに従って進めばよいわけです。以下の図では、他の ITEMORD セグメントは 1 つしかありません。これは、WASHER (ワッシャー (座金)) の ITEMORD セグメントです。このセグメント内の LTF ポインターは、このチェーンにおける最後の論理兄弟なので、このポインターにはゼロが入っています。

従属セグメントの LTB ポインターを用いてパフォーマンスが向上するのは、仮想対の論理関係である実論理子の削除の場合です。この削除が物理的なパスに沿うときに、このパフォーマンスが向上が生じます。

LTF ポインターと LTB ポインターは、直接ポインターです。 これらには、このポインターが指すセグメントの 4 バイトの直接アドレスが入っています。これは、LTF ポインターと LTB ポインターが HD データベースの中にしか存在できないことを意味しています。以下の図は LTF ポインターを示しています。

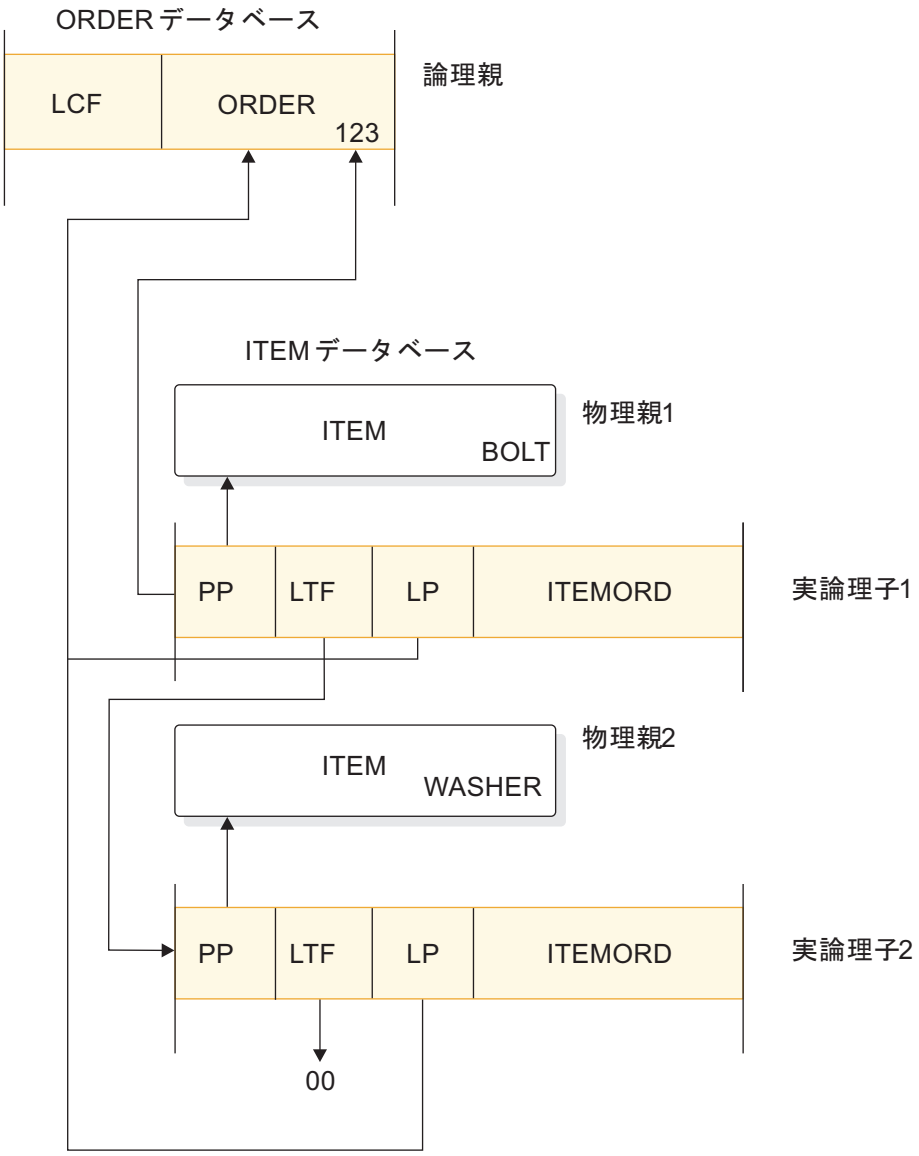


図 *81.* 論理兄弟順方向 *(LTF)* ポインター *(*仮想対においてのみ*)*

前の図では、LTF ポインターが、特定の論理兄弟から、その後ろに保管されている論理兄弟を指しています。この例の場合、このポインターは、BOLT の ITEMORD セグメントから WASHER の ITEMORD セグメントを指します。直接ポインターであるので、LTF ポインターは HD データベースしか指せません。 LTF ポインターは、論理子セグメントの接頭部の中にあり、その後ろに保管されている論理兄弟を指す 4 バイトの RBA で構成されています。

間接ポインター

HALDB データベース (PHDAM、PHIDAM、および PSINDEX データベース) は、あるデータベース・レコードから他のデータベース・レコードを指すために、直接ポインターと間接ポインターを使用します。以下の図は、間接ポインターの使用法を示します。

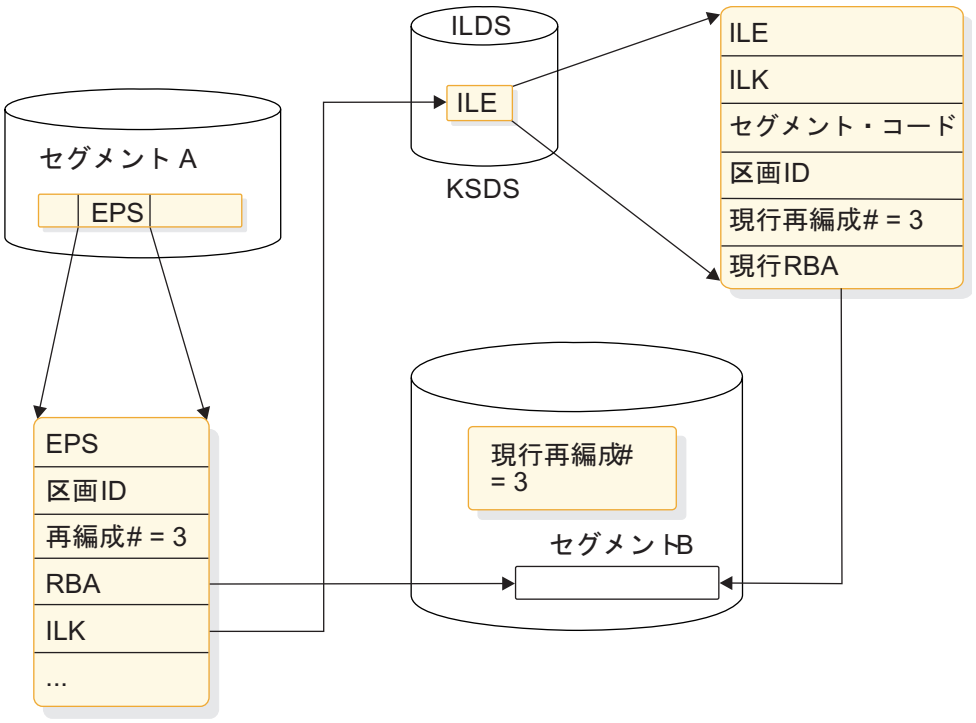


図 *82.* 自己回復ポインター

間接ポインターを使用することによって、データベースの再編成のときに発生することがある、宛先誤りのポインターの問題を回避できます。

間接ポインターのリポジトリーが、間接リスト・データ・セット (ILDS) です。再編成後の宛先誤りのポインターは、間接ポインターを使用して自己回復します。

関連概念:

764 ページの『HALDB 自己回復ポインター処理』

論理関係におけるパス

物理データベースの物理親と論理子の関係、およびそれぞれの論理子の中の LP ポインターによって、物理親から論理親へのパスができます。

このパスの使用を定義するためには、以下の図に示すように、この論理子と論理親を、物理親の物理子である連結セグメント・タイプとして定義します。このパスと連結セグメント・タイプの定義は、いわゆる論理データベースで行われます。

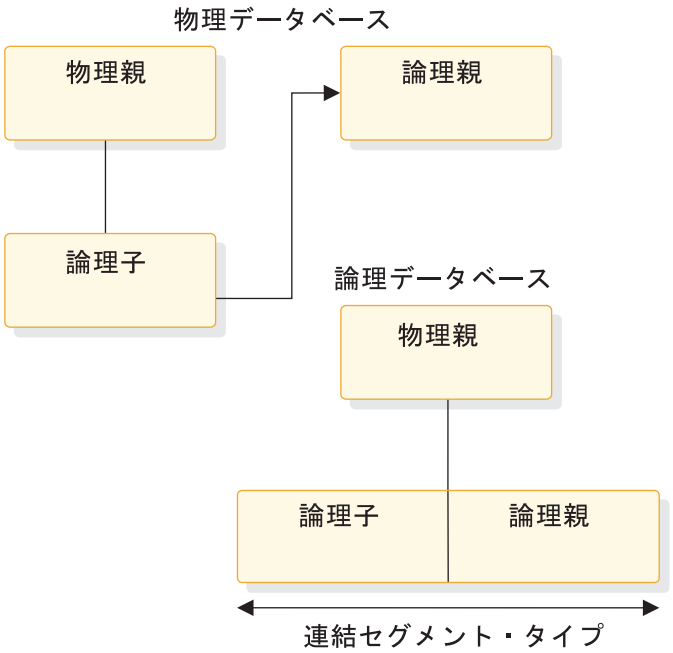


図 *83.* 論理データベースにおける物理親から論理親へのパスの定義

さらに、論理親によって LC ポインターが使用され、論理子によって論理兄弟ポインターと PP ポインターが使用されている場合は、論理親から物理親へのパスができます。このパスの使用を定義するために、論理子と物理親が、以下の図で示すように、この論理親の物理子である 1 つの連結セグメント・タイプとして定義されます。ここでも、このパスの定義は論理データベースで行われます。

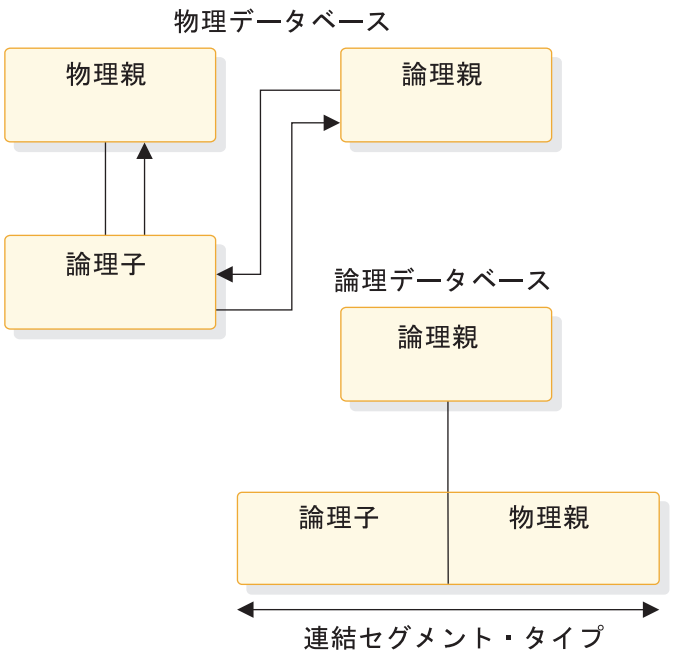


図 *84.* 論理データベースにおける論理親から物理親へのパスの定義

物理親から論理親へのパスの使用が定義されている場合は、この物理親は連結セグメント・タイプの親です。アプリケーション・プログラムが、この連結セグメン

ト・タイプのあるオカレンスを物理親の方から取り出すと、この論理子とその論理親が連結され、これが 1 つのセグメントとしてこのアプリケーション・プログラムに与えられます。論理親から物理親へのパスの使用が定義されている場合は、この論理親は連結セグメント・タイプの親です。アプリケーション・プログラムが、この連結セグメント・タイプのあるオカレンスを論理親の方から取り出すと、この論理子の 1 つのオカレンスとその物理親が連結され、これが 1 つのセグメントとしてこのアプリケーション・プログラムに与えられます。

いずれの場合にも、連結セグメントの中に組み込まれている物理親セグメントまたは論理親セグメントは、目標親 と呼ばれます。物理親から論理親へのパスでは、論理親が連結セグメントの中の目標親です。論理親から物理親へのパスでは、物理親が連結セグメントの中の目標親です。

関連タスク:

300 ページの『論理 DBD での論理関係の指定』

論理子セグメント

論理子をその物理データベースで定義する場合、論理子に対して指定する長さは、論理親の連結キーを保管するのに十分な長さでなければなりません。

交差データ は、論理親の連結キーに必要な長さより大きい任意の長さで保管できます。交差データは、特定の論理関係に固有なデータのタイプです。

論理子によってどの論理親が指されているかを示すには、論理親の連結キーがなければなりません。論理子がデータベースにロードするために最初に与えられる場合、各論理子セグメントはアプリケーション・プログラムの入出力域になくてはなりません。しかし、論理親が HD データベースの中にある場合には、論理子がロードされるとき、論理親の連結キーはストレージへ書き込まれないことがあります。論理親が HISAM データベースの中にある場合には、ストレージの中の論理子は、その論理親の連結キーをもっていなければなりません。

論理子セグメントに対しては、SEGM ステートメントの PARENT= パラメーターで特殊なオペランドを定義することができます。 このオペランドによって、論理親を指すシンボリック・ポインターが、論理子セグメントの一部分として、ストレージ・デバイスに保管されるかどうかが決まります。 PHYSICAL と指定すると、論理親の連結キーが各論理子セグメントと一緒に保管されます。 VIRTUAL と指定すると、各論理子セグメントの交差データ部分のみが保管されます。

連結セグメントが論理データベースを通して取り出された場合には、この連結セグメントには論理子セグメントが入っており、この論理子セグメントは、目標親の連結キーの後ろに交差データが付いているものです。また、この後ろには目標親の中のデータが続いています。以下の図は、入出力域で取り出された連結セグメントのフォーマットを示したものです。目標親の連結キーがそれぞれの連結セグメントと一緒に戻され、どの目標親が取り出されたかを示しています。 IMS は、連結セグメントの中の論理子から連結キーを取得するか、あるいは連結キーを作ります。目標親が論理親であり、その連結キーが論理子と一緒に保管されていない場合は、IMS が連結キーを構成してアプリケーション・プログラムに提供します。目標親が物理親の場合は、IMS は常にその連結キーを作成しなければなりません。

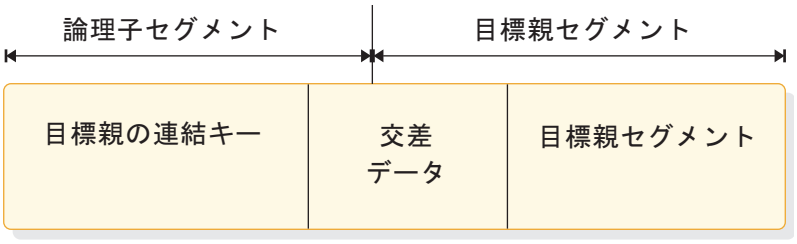


図 *85.* ユーザー入出力域に戻される連結セグメントのフォーマット

関連概念:

287 ページの『交差データ』

関連資料:

87 ページの『LCHILD セグメント・タイプのフォーマット』

論理関係についてのセグメント接頭部情報

論理関係に関与するセグメントの接頭部に関して、2 つの点を認識しておく必要があります。

第 1 に、IMS は特定のシーケンスで接頭部の中にポインターを設けます。第 2 に、IMS は論理子ポインターをもっていない論理親の接頭部の中にカウンターを設けます。

セグメントの接頭部におけるポインターの順序

セグメントに 2 種類以上のポインターが入っており、このセグメントが論理関係に関与している場合には、セグメント接頭部の中のポインターの順序は、次のとおりです。

1. HF
2. HB
3. PP
4. LTF
5. LTB
6. LP

または

1. TF
2. TB
3. PP
4. LTF
5. LTB
6. LP
7. PCF
8. PCL

または

1. TF
2. TB
3. PP
4. PCF
5. PCL
6. EPS

1 つのセグメント・タイプの中で、PCF ポインターと PCL ポインターは複数あってもかまいません。しかし、この他の種類のポインターが複数になることはありません。

論理関係で使用されるカウンター

IMS は、論理子ポインターをもっていないすべての論理親の中に 4 バイトのカウンターを 1 つ設けます。このカウンターは、論理親の接頭部の中に保管され、この論理関係を指している論理子の数の合計を含んでいます。このカウンターは IMS によって維持され、削除操作を正しく扱うために使用されます。このカウントの数値が 0 より大きい場合は、この論理親をデータベースから削除することはできません。この論理親を指している論理子がまだ存在するからです。

交差データ

2 つのセグメントの間に論理関係がある場合、その論理関係についてのみ固有のデータが存在することがあります。

例えば、次の図では、ORDER 123 で注文される品目の 1 つが 5000 個の BOLT (ボルト) です。5000 という数量はこの注文 (ORDER 123) とこの品目 (BOLT) に固有なものです。この数量 5000 は、この注文またはこの品目のいずれか片方に属しているわけではありません。同様に、ORDER 123 において 6000 個の WASHER が注文されます。ここでもまた、このデータは、この特定の注文と品目の組み合わせのみに関係しています。

この種のデータは交差データ (intersection data) と呼ばれます。この種のデータが特定の論理関係に対してのみ意味を持つからです。品目の数量を ORDER 123 のセグメントに保管することはできません。 ORDER 123 のそれぞれの品目ごとに異なる数量がそれぞれ注文されるからです。品目の数量を ITEM セグメントに保管することもできません。それぞれの品目ごとにいくつかの注文があり、それぞれの注文ごとに要求数量が異なるからです。論理子セグメントが ORDER セグメントと ITEM セグメントを結び付けているため、この 2 つのセグメントの関係に固有のデータをこの論理子の中に保管することができます。

交差データには、固定交差データと可変交差データがあります。

固定交差データ

論理子に保管されているデータは固定交差データと呼ばれます。シンボリック・ポインターが使用されている場合、このデータは論理子のデータ部の中で LPCK の後ろに保管されます。直接ポインターが使用されている場合には、これが論理子セグメントの中の唯一のデータです。以下の図では、シンボリック・ポインターが使用

されているため、ボルトとワッシャーが LPCK で、5000 と 6000 が固定交差データです。固定交差データは複数のフィールドで構成することができますが、そのフィールドすべてがこの論理子セグメントの中に保管されます。

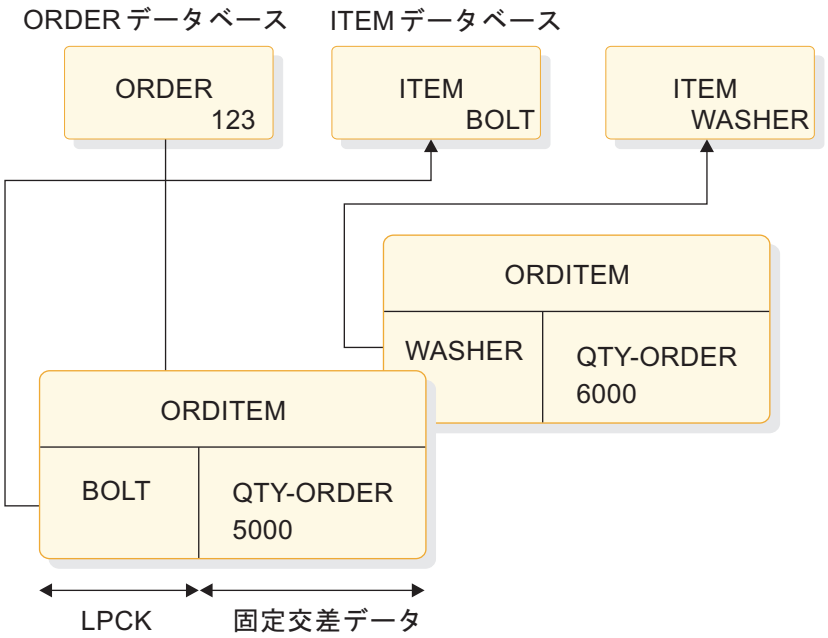


図 *86.* 固定交差データ

可変交差データ

可変交差データは、ある関係に固有のデータがあり、そのオカレンスがいくつかある場合に使用されます。例えば、ある注文に必要とされる品目の数量全体を 1 回の発送で送ることができないものとします。特定の日付に発送された数量を示す発送データを保管しておく必要が生じることがあります。発送日は注文または品目のいずれか一方のみに従属しているわけではありません。発送日は特定の注文と品目の組み合わせに従属しています。したがって、発送日は論理子セグメントの従属セグメントとして保管されます。論理子のこの従属セグメントの中のデータが、可変交差データと呼ばれるものです。各論理子オカレンスごとに、交差データが入っている従属セグメントのオカレンスを必要な数だけ設けることができます。

以下の図は、可変交差データを示したものです。 ORDER 123 の品目ボルトに関しては、3 月 2 日に 3000 個発送され、4 月 2 日に 1000 個発送されました。そのため、DELIVERY セグメントは 2 オカレンス存在します。ある 1 つの論理子セグメントのための交差データを複数のセグメント・タイプに収容することができます。この図では、DELIVERY セグメントのほかに、SCHEDULE セグメント・タイプに注目してください。このセグメント・タイプは、発送予定日と品目の発送予定数量を示しています。この図のように、可変交差データが入っているセグメント・タイプすべてが階層内の同じレベルにあってもかまいませんし、これらが相互に従属関係にあってもかまいません。

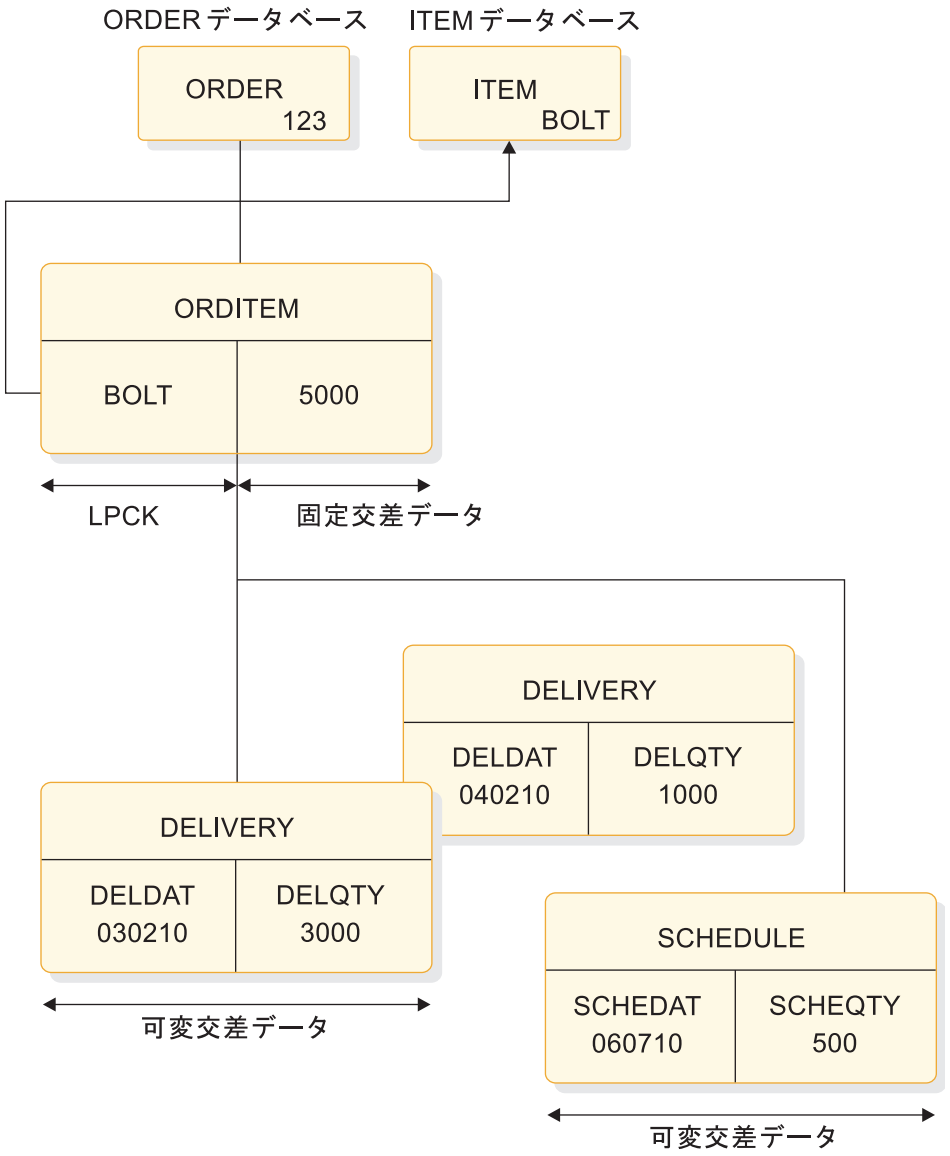


図 *87.* 可変交差データ

固定交差データ、可変交差データ、および物理対

これまで見てきた図では、交差データは単一方向論理関係で保管されていました。交差データは、2 種類の両方向論理関係においても同じ働きをします。ただし、物理対を使用している場合には、論理関係の一方の側にしか可変交差データを保管することはできません。いずれの側にこれを保管してもかまいません。アプリケーション・プログラムは、ORDER データベースまたは ITEM データベースを用いてこれにアクセスできます。固定交差データに関しては物理対を使用する場合、論理関係の両側に FID を保管しなければなりません。この一方の側の固定交差データが変更されると、IMS は自動的にこの論理関係の両側の固定交差データを更新します。ただし、物理対の論理関係において固定交差データが使用されている場合には、更新・維持管理のために余分の時間が必要であり、DASD 上に余分なスペースが必要です。

関連概念:

285 ページの『論理子セグメント』

再帰構造: 同一データベース論理関係

複数の物理データベースの中のセグメントとセグメント間に、論理関係を設定することができます。先にも触れましたが、同一のデータベースに属するセグメント間にも論理関係を設定することができます。これによってできる論理データ構造は再帰構造 と呼ばれます。

多くの場合、再帰構造は製造業において部品表タイプのアプリケーションのために定義されます。例えば、ある会社が自転車を製造しているものとします。この製造メーカーで作る最初の型は 1 型であり、これは少年用自転車です。以下の表は、この自転車を製造するのに必要な部品と、 1 台の 1 型自転車を製造するのに必要な各部品の数をリストしたものです。

表 *57. 1* 型自転車の部品のリスト

部品	必要な数
21 インチ少年用フレーム	1
ハンドル・バー	1
シート	1
チェーン	1
前輪泥よけ	1
後輪泥よけ	1
ペダル	2
クランク	1
前部スプロケット	1
26 インチ・チューブ/タイヤ	2
26 インチ・リム	2
26 インチ・スポーク	72
前輪ハブ	1
ハウジング	1
ブレーキ	1
後部スプロケット	1

製造業では、最終製品を製造するために行わなければならない工程を知っておく必要があります。各工程において、必要部品が用意されていなければならず、ある 1 つの工程において用いられる中間組立品をこれ以前の工程において組み立てておきます。以下の図は 1 型自転車の製造に必要なステップを示したものです。第 2 工程で後部ハブ組み立て品を作るには、ハウジング (支持枠)、ブレーキ、および後部スプロケットが必要です。この後輪ハブ組み立て品ができて初めて、第 3 工程の後輪組み立て品の部分の作業を行うことができます。第 3 工程のこの部分では、26 インチのタイヤとリム各 1 つと 36 本のスポークを用意しておく必要があります。

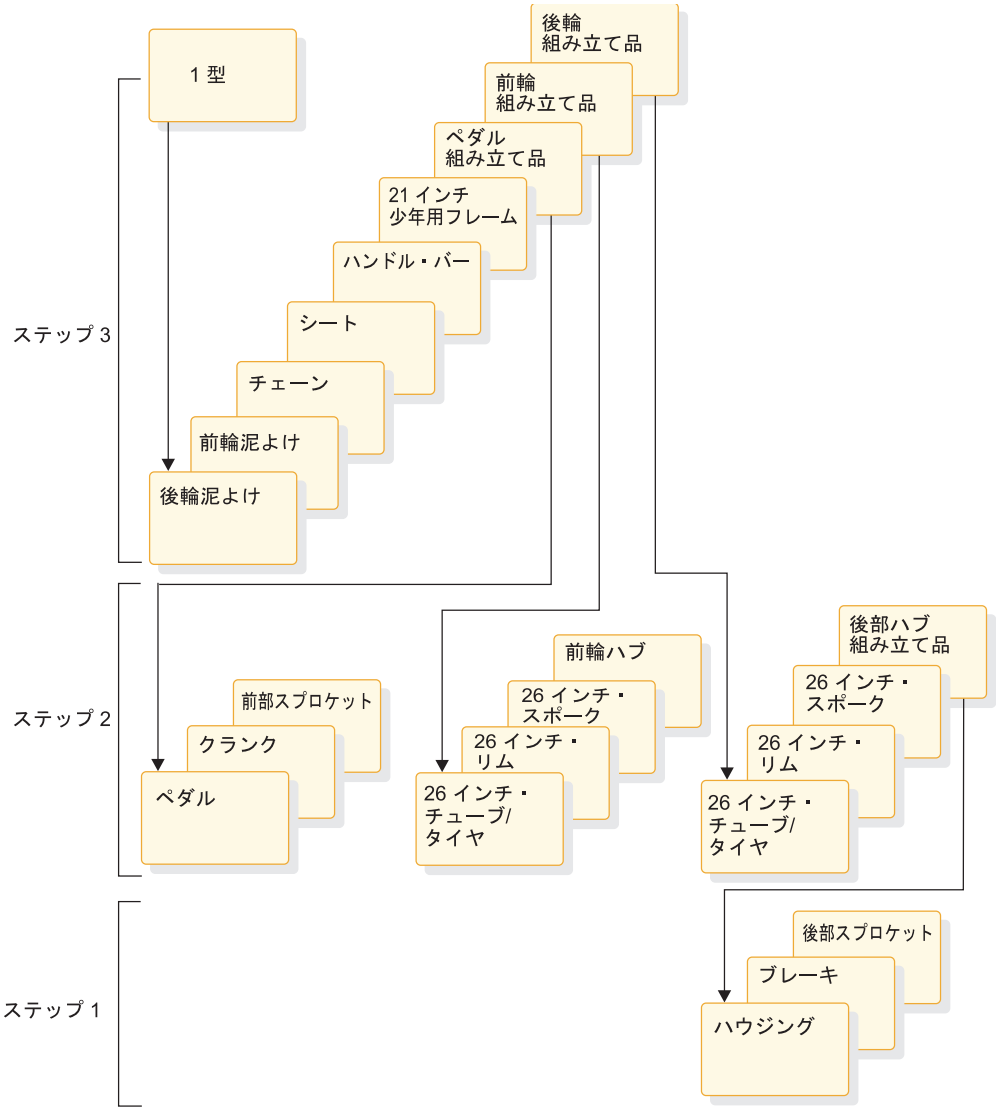


図 *88. 1* 型の部品と部品組み立て品

同じ会社で少女用の 2 型自転車を製造しています。この自転車のフレームが少女用のフレームであるという点を除けば、この自転車の部品と組み立て工程は、1 型自転車と同じです。

この製造メーカーで、この双方の型式のすべての部品と中間組立品を別個のセグメントとしてデータベースに保管すれば、大量の重複データが存在することになります。前の図は、1 型自転車のみのために保管しなければならないセグメントを示したものです。 2 型自転車にも、フレームが少女用のフレームであるという点を除き、同様に 1 組のセグメントを保管しなければなりません。これによって重複データが生じ、そのための維持管理の問題が発生します。再帰構造を作成することによってこの問題は解消されます。以下の図は、1 型自転車用データを用いて、これを行う方法を示したものです。

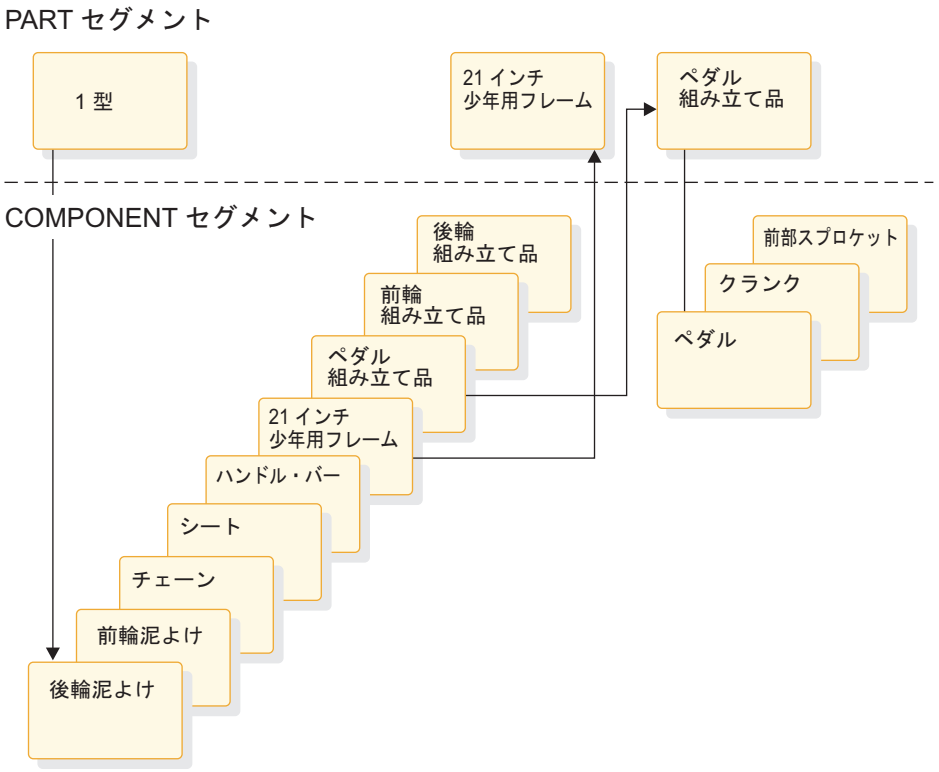


図 *89. 1* 型自転車のためのデータベース・レコード

上の図には、2 つのセグメント・タイプがあります。PART セグメントと COMPONENT セグメントです。この 2 つのセグメント間に単一方向論理関係が設定されています。 1 型の PART セグメントがルート・セグメントです。その下に COMPONENT セグメントの 9 つのオカレンスがあります。各オカレンスは、別の PART ルート・セグメントを指している論理子です。 (図を簡単にするために、実際に示されているポインターは 2 つだけです。) ただし、他の PART ルート・セグメントは、このコンポーネントを作成するのに必要な部品を示しています。

例えば、ペダル組み立て品のコンポーネントは、ペダル組み立てのための PART ルート・セグメントを指しています。このセグメントの下に保管されているのは、組み立てなければならない部品、すなわち、前部スプロケット 1 つ、クランク 1 つ、およびペダル 2 つです。このような構造にすると、2 型自転車のために保管しなければならない大部分の重複データを排除することができます。

以下の図は、前の図のセグメントに加えて、2 型自転車のためのデータベース・レコードに保管しなければならないセグメントを示したものです。この図の論理子は、固有のコンポーネント、つまり 21 インチの少女用フレームに使用する論理子を除いて、前の図で示している同じ PART セグメントを指すことができます。例えば、ペダル組み立て品のための別の PART セグメントを設ける必要はありません。1 型のデータベース・レコードと 2 型のデータベース・レコードには同じペダル組み立て品が入っているので、この両方のデータベース・レコードにおいて、論理子を用いて同じペダル組み立て品の PART セグメントを指すことができます。

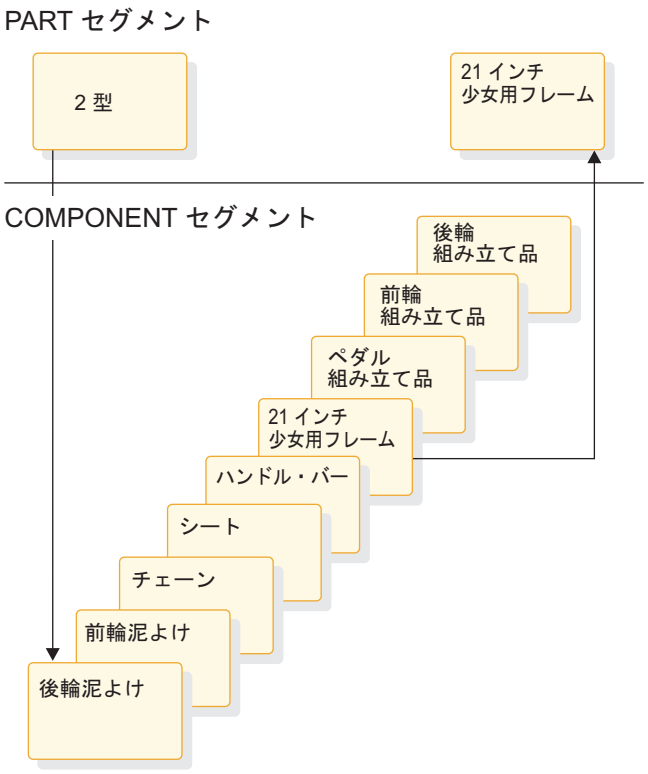


図 *90. 2* 型自転車のために必要な余分のデータベース・レコード

再帰構造について注意すべきことが 1 つあります。論理子の物理親と論理親は、共に同じセグメント・タイプであるということです。例えば、 292 ページの図 89 では *1* 型 の PART セグメントは、ペダル組み立て品の COMPONENT セグメントの物理親です。ペダル組み立て品の PART セグメントは、ペダル組み立て品の COMPONENT セグメントの論理親です。

関連概念:

266 ページの『副次索引対論理関係』

論理関係のためのシーケンス・フィールドの定義

論理関係を使用する場合は、シーケンス・フィールドを定義する際に一定の規則と推奨事項に従う必要があります。

論理親のシーケンス・フィールド

論理関係を用いたデータベース処理で、問題が起きるのを回避するためには、固有のシーケンス・フィールドをすべての論理親セグメントに定義しなければなりません。論理親のすべてのセグメントは、物理データベースに従属しています。固有のシーケンス・フィールドが、ある論理親まで (この論理親も含む) のパスにあるすべてのセグメント で定義されていない場合には、ある 1 つのデータベースの中の複数の論理親が同じ連結キーを持つことがあり得ます。このようなことが起きると、初期データベース・ロードが行われる間と、その後に論理子セグメントの中の記号論理親ポインターを用いてある論理親セグメントに位置が設定されるときに問題が起きることがあります。

初期データベース・ロード時に、データベースの中に非固有の連結キーを持つ複数の論理親があると、解決ユーティリティーは、同じ連結キーを持つすべての論理子を、このデータベースの中でこの連結キーを持つ最初の論理親に結び付けます。

論理親のために 1 つの連結セグメントと位置を挿入あるいは削除するときには、この連結セグメントの部分は、その論理親の連結キーによって判別されます。この論理親の位置付けは、ルート・セグメントから始まり、各レベルでキーが等しいという条件を満たす論理親のデータベースの各レベルで最初のセグメントが終わります。挿入されている論理親へのパスでセグメントがない場合には、この方法を用いて論理親のデータベースの中に位置を設定するときに、 GE 状況コードはアプリケーション・プログラムに戻されます。

実論理子のシーケンス・フィールド

実論理子のシーケンス・フィールドが論理親の連結キーのいずれかの部分で構成されている場合には、この論理子を対象とする SEGM ステートメントの中の PARENT= パラメーターで PHYSICAL と指定しなければなりません。これによって、論理親の連結キーが論理子セグメントと一緒に保管されることになります。

仮想論理子のシーケンス・フィールド

一般的な規則として、1 つのセグメントは 1 つのシーケンス・フィールドだけ持つことができます。しかし、仮想対の場合には、複数の FIELD ステートメントを用いて、仮想論理子の論理シーケンス・フィールドを定義することができます。

仮想論理子のシーケンス・フィールドを指定しなければならないことがありますが、仮想論理子にその論理親からアクセスするとき、実論理子セグメントを取り出す順序を、アプリケーション・プログラムの入出力域で見ることができる仮想論理子の 1 つのフィールドの中のデータで決める必要がある場合です。このシーケンス・フィールドには、論理親から見たときのこのセグメントの任意の部分 (すなわち、実論理子の物理親の連結キーとそれに続く交差データ) を入れることができます。論理親からアクセスされる論理子のシーケンス・フィールドを、隣接していないいくつかの部分に分けて記述しなければならないこともあるため、SEQ パラメーターを持つ複数の FIELD ステートメントを使うことができます。この各ステートメントには、他と重複していない fldname1 パラメーターを 1 つずつ入れなければなりません。

関連概念:

826 ページの『IMS 論理関係の変更』

関連資料:

- データベース接頭部解決ユーティリティー (DFSURG10) (データベース・ユーティリティー)
- データベース接頭部更新ユーティリティー (DFSURGP0) (データベース・ユーティリティー)
- データベース事前再編成ユーティリティー (DFSURPR0) (データベース・ユーティリティー)

論理関係における PSB、PCB、および DBD

論理関係を使用する場合は、物理 DBD を使用して、その関係に関与する物理データベースを IMS に対して定義する必要があります。また、多くの場合、データベースの論理構造、つまりアプリケーション・プログラムが理解する構造を、IMS に対して定義する必要があります。これを行うには、論理 DBD を使用します。

アプリケーション・プログラムの PCB は DBD を参照しますが、物理 DBD はアプリケーション・プログラムがアクセスする必要のある論理データ構造を反映していないので、論理 DBD が必要になるわけです。最後に、アプリケーション・プログラムには、1 つ以上の PCB で構成される PSB が必要になります。論理関係による処理が行われているとき用いられる PCB は、論理 DBD が定義されている場合には、論理 DBD を指します。この PCB は、アプリケーション・プログラムが処理できる論理データベースのセグメントを示しています。また、これは、このアプリケーション・プログラムが各セグメントに対してどのような種類の処理を行うことができるかも示しています。

PSB と PCB、論理 DBD、および物理 DBD は、IMS 内部では制御ブロックとして表されます。以下の図は、この 3 つの制御ブロックの関係を示しています。この図では、2 つの物理データベース間に論理関係が設定されていると仮定しています。

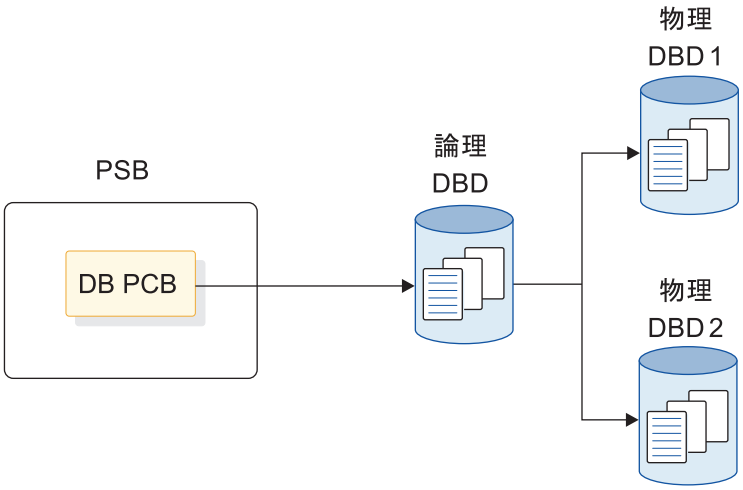


図 *91.* 論理関係を使用する場合の各種制御ブロック間の関係

関連タスク:

『物理 DBD での論理関係の指定』

300 ページの『論理 DBD での論理関係の指定』

物理 DBD での論理関係の指定

論理関係に関与している各データベースごとに、物理 DBD をコーディングしなければなりません。

物理 DBD 内のすべてのステートメントは、SEGM ステートメントと LCHILD ステートメントを除き、論理関係が定義されていない場合と同様にコーディングされます。 SEGM ステートメントは、セグメント、その長さ、およびデータベース階

層での位置について記述するものですが、新しい種類のポインターを取り入れるために拡張されています。また、2 つのセグメント・タイプの間の論理関係を定義するために、LCHILD ステートメントが追加されています。

以下の各図に関連する例の SEGM ステートメントでは、論理関係で必要なポインターだけが示されています。HD データベースで使用する必要のあるポインターは示されていません。 DBD を実際にコーディングするときには、PTR= パラメーターでこのようなポインターを要求しなければなりません。そうしなければ、いったん他の種類のポインターが指定されてしまうと、IMS はこのようなポインターを生成しません。

以下の図は、セグメントのレイアウトを示しています。

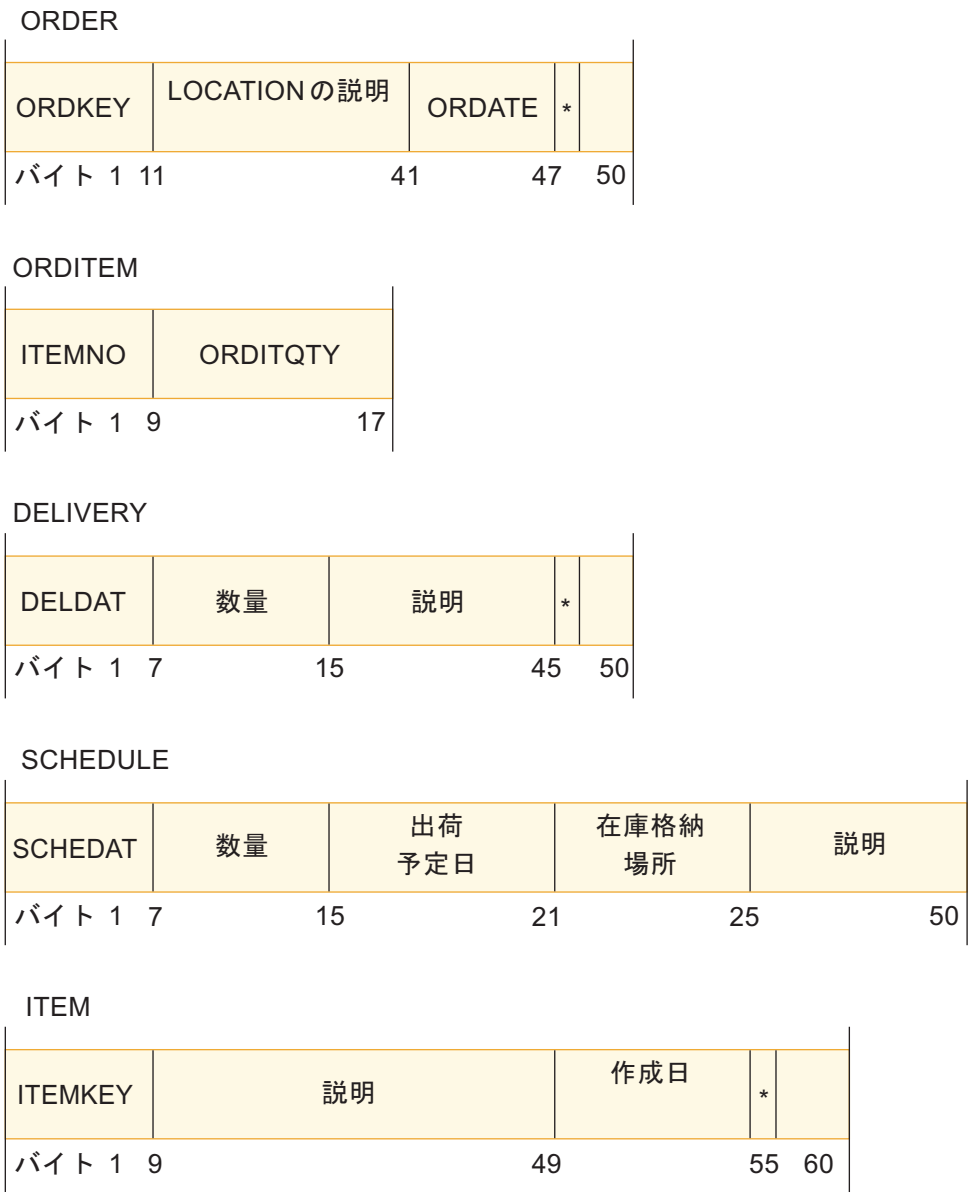


図 92. 例の中で使用されているセグメントのレイアウト

次に示すのが、論理関係に関与する 2 つのデータベースの階層構造です。この例では、シンボリック・ポインターを使用する単一方向論理関係を定義します。ORDITEM は LPCK と固定交差データをもっており、DELIVERY と SCHEDULE は可変交差データです。

以下の図は、単一方向関係のための物理 DBD を示しています。

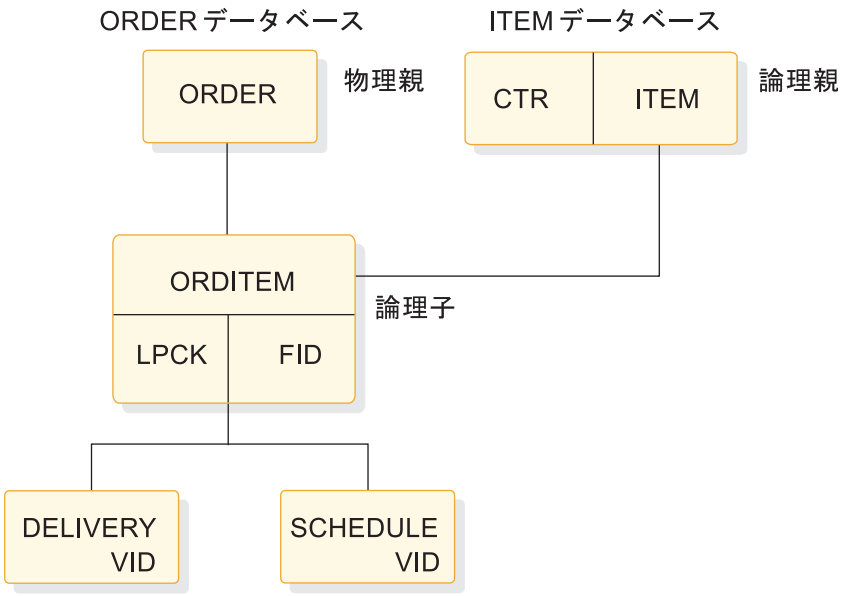


図 *93.* シンボリック・ポインターを使用する単一方向関係のための物理 *DBD*

以下は、ORDER データベースの DBD です。

```
DBD    NAME=ORDDB
SEGM   NAME=ORDER,BYTES=50,FREQ=28000,PARENT=0
FIELD  NAME=(ORDKEY,SEQ),BYTES=10,START=1,TYPE=C
FIELD  NAME=ORDATE,BYTES=6,START=41,TYPE=C
SEGM   NAME=ORDITEM,BYTES=17,PARENT=((ORDER),(ITEM,P,ITEMDB))
FIELD  NAME=(ITEMNO,SEQ),BYTES=8,START=1,TYPE=C
FIELD  NAME=ORDITQTY,BYTES=9,START=9,TYPE=C
SEGM   NAME=DELIVERY,BYTES=50,PARENT=ORDITEM
FIELD  NAME=(DELDAT,SEQ),BYTES=6,START=1,TYPE=C
SEGM   NAME=SCHEDULE,BYTES=50,PARENT=ORDITEM
FIELD  NAME=(SCHEDAT,SEQ),BYTES=6,START=1,TYPE=C
DBDGEN
FINISH
END
```

以下は、ITEM データベースの DBD です。

```
DBD    NAME=ITEMDB
SEGM   NAME=ITEM,BYTES=60,FREQ=50000,PARENT=0
FIELD  NAME=(ITEMKEY,SEQ),BYTES=8,START=1,TYPE=C
LCHILD NAME=(ORDITEM,ORDDB)
DBDGEN
FINISH
END
```

ORDER データベースの中の通常の DBD コーディングと異なる DBD コーディングは、論理子 ORDITEM のためのものです。

ORDITEM のための SEGM ステートメントでは、

1. BYTES= パラメーターは 17 です。 ここで指定する長さは、LPCK の長さに固定交差データの長さを加えたものです。LPCK は ITEM セグメントのキーであり、その長さは 8 バイトです。固定交差データの長さは 9 バイトです。
2. PARENT= パラメーターは、2 つの親を指定しています。 ORDITEM は論理子であり、したがって、物理親と論理親の両方をもっています。物理親は ORDER です。論理親は ITEM であり、ORDER の後ろに指定されます。 ITEM は ORDITEM とは異なる物理データベースの中にあるので、その物理データベースの名前 - ITEMDB - を指定しなければなりません。セグメント名 ITEM とデータベース名 ITEMDB との間に、P という文字があります。文字 P は「物理」を意味します。これは、LPCK をこの論理子セグメントの一部として、DASD 上に保管するよう指定しています。

ORDITEM のための FIELD ステートメントでは、

1. ITEMNO は ORDITEM セグメントのシーケンス・フィールドであり、長さは 8 バイトです。 ITEMNO は LPCK です。論理親は ITEM であり、ITEM データベースの中の ITEM 用の FIELD ステートメントを見れば、ITEM のシーケンス・フィールドが ITEMKEY であり、このフィールドの長さが 8 バイトであることが分かります。 ITEM はルート・セグメントであるので、LPCK の長さは 8 バイトです。
2. ORDITQTY は固定交差データであり、通常どおりコーディングされます。

ITEM データベースでは、DBD コーディングが通常の DBD コーディングと異なるのは、LCHILD ステートメントが追加されているという点です。このステートメントは、論理子 ORDITEM の名前を指定しています。 ORDITEM セグメントは ITEM とは異なる物理データベースの中にあるので、その物理データベースの名前 - ORDDB - を指定しなければなりません。

関連概念:

295 ページの『論理関係における PSB、PCB、および DBD』

両方向論理関係の指定

物理対を使用する両方向関係を定義するときは、両方の論理親に LCHILD ステートメントを追加し、さらに、他のポインターに加えて、両方の論理子の SEGM ステートメントの POINTER= パラメーターに PAIRED オペランドを追加する必要があります。

仮想対の両方向論理関係を定義する場合には、実論理子に対してのみ LCHILD ステートメントをコーディングする必要があります。この LCHILD ステートメントでは、論理子ポインターを取得するために、POINTER=SNGL または DBLE をコーディングします。また、PAIR= オペランドをコーディングして、この実論理子と対をなす仮想論理子を示します。実論理子のための SEGM ステートメントを定義するときには、PARENT= パラメーターは、物理親および論理親の双方を識別します。論理兄弟ポインターを (他のポインターの他に) POINTER= パラメーターで指定しなければなりません。また、仮想論理子は実在しませんが、これを対象とする SEGM ステートメントも定義しなければなりません。この SEGM ステートメントでは、POINTER= パラメーターで PAIRED と指定します。さらに、SOURCE= パラメーターを指定します。 SOURCE= パラメーターでは、実論理子の SEGM 名と

DBD 名を指定します。仮想論理子の SEGM ステートメントで SOURCE= を定義するときには、常に DATA を指定しなければなりません。

関連資料:

DBD ステートメント (システム・ユーティリティー)

物理データベースでの論理関係の定義に関する規則のチェックリスト

物理データベースで論理関係を定義する際には、一定の規則に従う必要があります。

以下のサブトピックでは、いずれの場合もオカレンスではなく、セグメント・タイプについて述べています。

論理子の規則

物理データベース内の論理子セグメント・タイプの定義は、いくつかの規則によって制御されます。

- 論理子は、物理親と論理親を持っていなければなりません。
- 論理子は、物理親を 1 つと論理親 1 つだけ持つことができます。
- 論理子は、その物理親の物理データベースにおいては物理子として定義されます。
- 論理子はその物理データベースにおいては常に従属セグメントであり、したがって、データベースの最初のレベルを除く任意のレベルにおいてこれを定義することができます。
- 論理子はその物理データベースにおいて、データベースのすぐ下のレベルで定義されたものでありしかも論理子でもある物理子を持つことはできません。
- 論理子は物理子を持つことができます。しかし論理子が別の論理子と物理対をなしている場合には、この一対のセグメントのいずれか一方だけが物理子を持つことができます。

論理親の規則

物理データベース内の論理親セグメント・タイプの定義は、いくつかの規則によって制御されます。

- 論理親は、物理データベースの中の、ルート・レベルを含む任意のレベルを定義することができます。
- 論理親は、1 つ以上の論理子を持つことができます。同じ論理親に関連する各論理子は、それぞれ 1 つの論理関係を定義します。
- 物理データベースの中にある 1 つのセグメントは、論理親と論理子の両方として定義されることはできません。
- 論理親は、その論理子と同じ物理データベースで定義することもできるし、異なる物理データベースで定義することもできます。

物理親の規則

物理データベース内に物理親セグメント・タイプを定義するための唯一の規則は、論理子の物理親を同時に論理子にすることはできないというものです。

論理 DBD での論理関係の指定

ある論理データ構造においてどのセグメント・タイプが使用されるかを示すためには、論理 DBD をコーディングしなければなりません。

以下の図は、論理 DBD のコーディング方法の例を示しています。この例は、 295 ページの『物理 DBD での論理関係の指定』で定義した同じ物理データベースのための論理 DBD です。

論理データベースでセグメントを定義する場合、SEGM ステートメントの SOURCE= パラメーターで、KEY または DATA オペランドを使用してプログラムの入出力域に戻すことにするかどうかをこのセグメントで指定することができます。 DATA はこのセグメントのキー部分とデータ部分の両方を入出力域に戻します。 KEY は、このセグメントのデータ部分ではなく、キー部分のみを入出力域に戻します。

連結セグメントの SEGM ステートメントに SOURCE= パラメーターを使用すると、KEY パラメーターと DATA パラメーターは 2 つのセグメントのどちらを (またはその両方を) 検索呼び出しで入出力域に置くかを制御します。言い換えれば、ある 1 つの連結セグメント・タイプに対して SOURCE= パラメーターを 2 回定義します。1 回は論理子部分を対象とし、 1 回は目標親部分を対象とします。

以下の図に示すのがアプリケーション・プログラムで作成するために必要な論理データ構造です。この論理構造は、シンボリック・ポインターを用いる単一方向論理関係でインプリメントされます。ルート・セグメントは、ORDER データベースの中の ORDER です。 ORDER には、論理親の ITEM と連結されている論理子である ORDITEM が従属しています。 ORDIT を対象とする単独の呼び出しが出されると、アプリケーション・プログラムはその入出力域にこの両方のセグメントを受け取ります。 DELIVERY と SCHEDULE は可変交差データです。

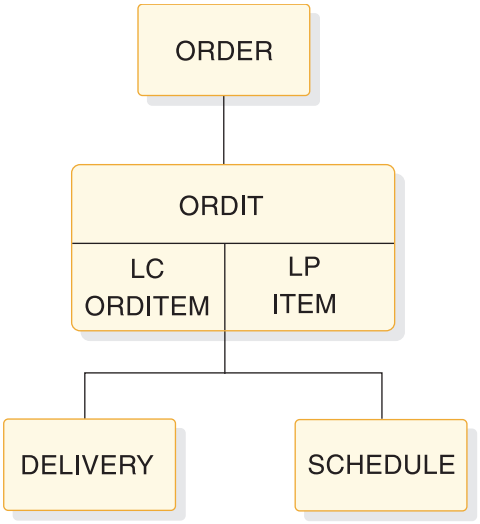


図 *94.* シンボリック・ポインターを使用する単一方向関係のための論理データ構造

次に示すのは、上の図に示す論理データ構造のための論理 DBD です。

```
DBD  NAME=ORDLOG,ACCESS=LOGICAL
DATASET LOGICAL
SEGM NAME=ORDER,SOURCE=((ORDER,DATA,ORDDB))
SEGM NAME=ORDIT,PARENT=ORDER,                                          X
          SOURCE=((ORDITEM,DATA,ORDDB),(ITEM,DATA,ITEMDB))
SEGM NAME=DELIVERY,PARENT=ORDIT,SOURCE=((DELIVERY,DATA,ORDDB))
SEGM NAME=SCHEDULE,PARENT=ORDIT,SOURCE=((SCHEDULE,DATA,ORDDB))
DBDGEN
FINISH
END
```

前の図に関する注**:**

1. DBD ステートメントでは、論理 DBD の名前が指定されています。この例では名前は ORDLOG です。物理 DBD の場合と同様、この名前は固有なものでなければならず、しかも、DBDGEN プロシージャーの MBR オペランドで指定されているものと同じ名前でなければなりません。 ACCESS=LOGICAL はこれが論理 DBD であると指定しているだけです。
2. DATASET ステートメントでは、常に LOGICAL と指定しますが、これは論理 DBD であることを意味しています。このステートメントに他のパラメーターを指定することはできませんが、データ・セットの DD 名はすべて、物理 DBD の DATASET ステートメントで指定されます。
3. SEGM ステートメントは、どのセグメントをこの論理データベースに組み込むか示しています。論理 DBD の SEGM ステートメントで使えるオペランドは、NAME=、 PARENT=、および SOURCE= のみです。セグメントに関する他の情報はすべて、物理 DBD で定義されます。
   - 最初の SEGM ステートメントは、ルート・セグメントの ORDER を定義しています。

     NAME= オペランドは、このセグメントを参照するために PCB で使用される名前を指定します。この名前は、SSA をコーディングするとき、アプリケーション・プログラマーが使用します。この例の場合、セグメント名は物理 DBD で使用している名前と同じ名前 (すなわち、ORDER) です。ただし、このセグメントにその物理 DBD で指定されているのとは異なる名前を付けてもかまいません。

     SOURCE= オペランドは、このセグメントのデータがどこから来るかを IMS に伝えます。最初に、セグメント・タイプの名前、ORDER が、その物理データベースに現れます。 DATA は、このセグメントの中のデータをアプリケーション・プログラムの入出力域に入れる必要があることを示しています。 ORDDB は、ORDER セグメントが入っている物理データベースの名前です。

     論理 DBD では FIELD ステートメントがコーディングされていません。IMS は物理 DBD からこのステートメントを選び出すので、この論理データ構造の中の ORDER セグメントにアクセスするとき、アプリケーション・プログラムは ORDKEY または ORDATE を指す SSA を使用することができます。これらのフィールドは、 297 ページの図 93 に示されるように、物理 DBD で ORDER セグメント用に定義されたものです。
   - 2 番目の SEGM ステートメントは、ORDIT セグメントを対象とするものです。 ORDIT セグメントは、論理親 ITEM と連結されている論理子

ORDITEM で構成されています。図からも分かるように、SOURCE= オペランドは異なる物理データベースの中の ORDITEM セグメントおよび ITEM セグメントの両方を示しています。

- 3 番目と 4 番目の SEGM ステートメントは、可変交差データの DELIVERY と SCHEDULE のためのものです。これらの SEGM ステートメントは、物理 DBD 内でこれらのステートメントが現れるのと同じ 相対順序で、論理 DBD に置かれなければなりません。物理 DBD では、DELIVERY は SCHEDULE の左側にあります。

関連概念:

295 ページの『論理関係における PSB、PCB、および DBD』

283 ページの『論理関係におけるパス』

論理データベースの定義に関する規則のチェックリスト

論理関係は、非常に複雑になる可能性があります。論理関係を使用するデータベースを適切に定義するためには、論理関係を制御する規則を理解し、これに従う必要があります。

論理データベースの定義に関する規則について説明する前に、以下の定義を知る必要があります。

- 論理関係の交差
- 交差した最初の論理関係と追加の論理関係

また、論理 DBD は、アプリケーション・プログラムが連結セグメントへアクセスする必要がある場合、またはアプリケーション・プログラムが論理関係を交差する必要がある場合にのみ必要になります。

論理関係の交差の定義

論理関係が、交差していると見なされるのは、この論理関係が論理データベースで次のようなセグメントをアクセスするのに使用される場合です。

- 目標親のデータベースの中の目標親の物理親
- 目標親の物理データベースの中の目標親の物理従属セグメント

論理関係が論理データベースで目標親のみをアクセスするために用いられる場合には、この論理関係は、交差しているとは見なされません。

以下の図の DBD1 と DBD2 は、間に論理関係が定義されている 2 つの物理データベースです。 DBD3 から DBD6 までは、DBD1 と DBD2 の間の論理関係から定義することができる 4 つの論理データベースです。 DBD3 では、論理関係を交差していませんが、それは DBD3 には目標親の物理親も物理従属セグメントも組み込まれていないからです。 DBD4 から DBD6 までは、いずれの場合にも論理関係が交差していますが、それぞれの目標親の物理親または物理従属セグメントが含まれているからです。

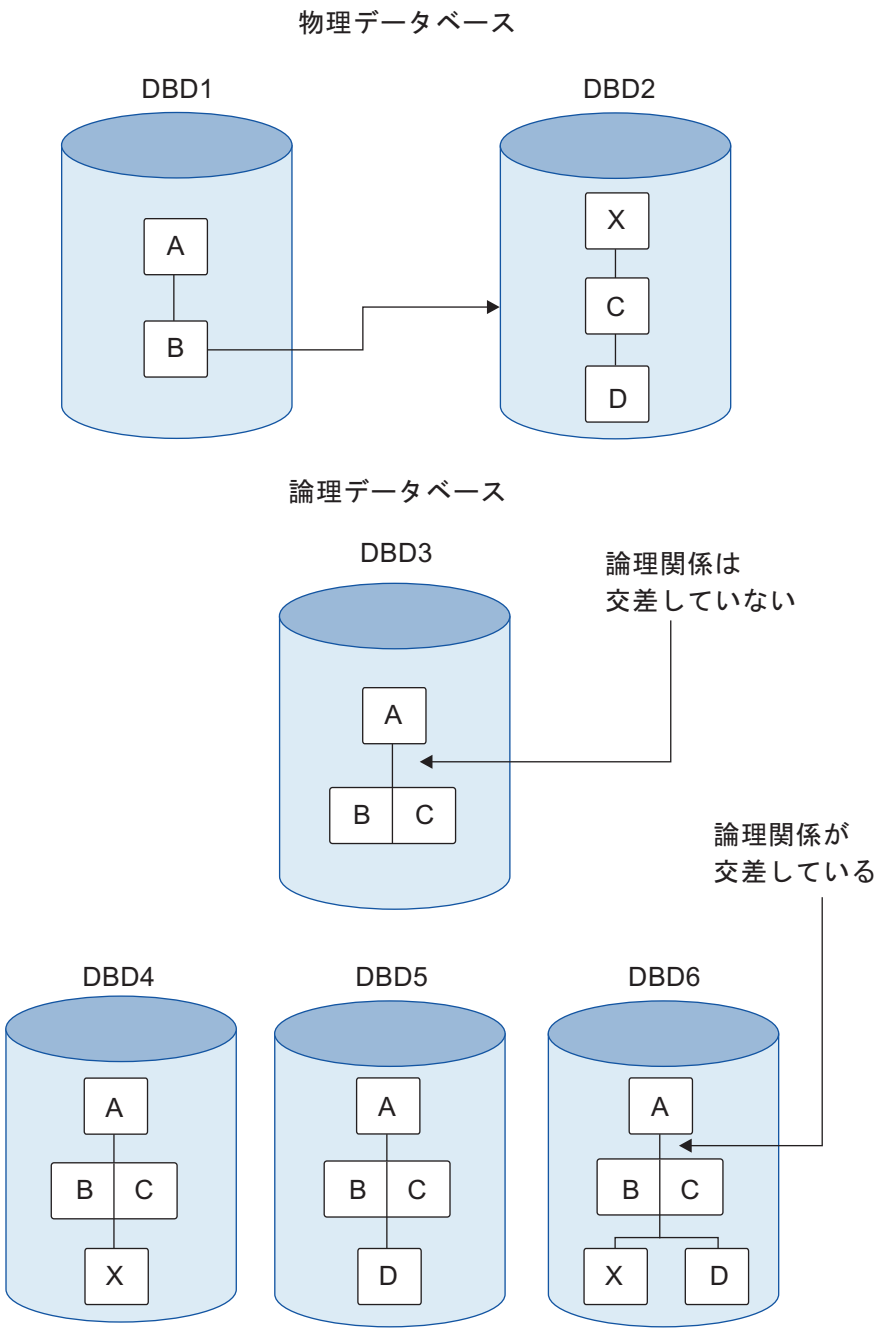


図 *95.* 論理関係の交差の定義

交差した最初の論理関係と追加の論理関係の定義

1 つの論理データベースの中の 1 つの階層パスで、複数の論理関係が交差していることが可能です。以下の図は、論理関係が定義されている 3 つの物理データベース (DBD1、DBD2、および DBD3) を示しています。この図はまた、これらの物理データベースの論理関係から定義することができる (多数の中の) 2 つの論理データベース (DBD4 と DBD5) を示しています。 DBD4 は、2 つの連結セグメント BF と DI を通じて、これらの目標親の階層パスにあるすべてのセグメントにアクセスすることができます。どちらか一方の論理関係または両方の論理関係が交差している場合、それぞれの論理関係が、交差している最初の論理関係であると見なされます。

これは、DBD1 の中のセグメント・タイプの物理階層に従って進めば、各連結セグメント・タイプに達するからです。

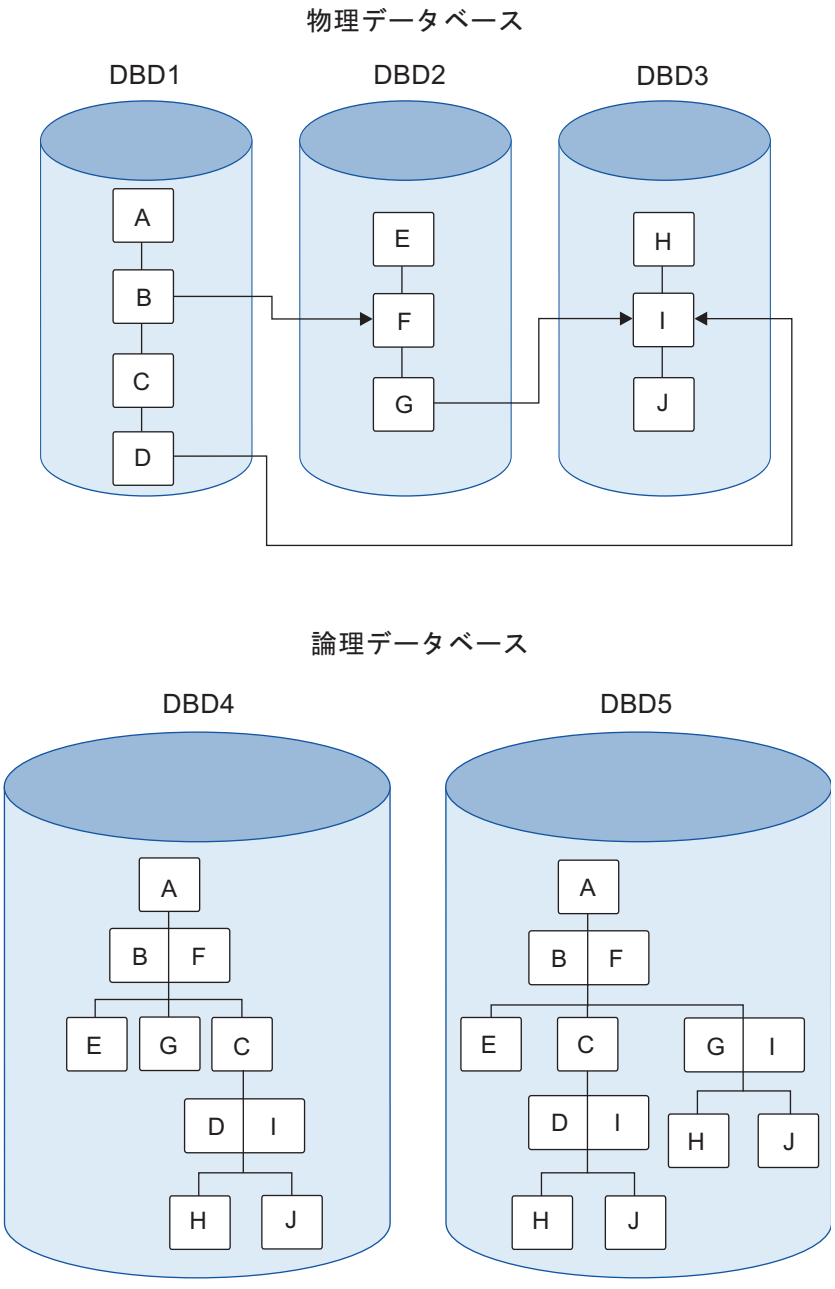


図 *96.* 論理データベースの階層パスにおいて交差している最初の論理関係

前の図の DBD5 では、DBD4 には含まれていなかったもう 1 つの連結セグメント・タイプ GI が定義されています。 GI を使用すればその目標親の階層パスにあるセグメント (交差されていれば) にアクセスすることができます。連結セグメント GI によって可能となった論理関係が交差されていれば、これが交差された追加の論理関係になります。これは、連結セグメント GI にアクセスすることができるようにするために、論理データベースのルート・セグメントから、連結セグメント・タイプ BF によって可能となった論理関係が交差されている必要があるからです。

論理データベースの階層パスにおいて最初の論理関係が交差されていると、次のようにしてその目標親の階層パスにあるすべてのセグメントへのアクセスが可能となります。

- 目標親の親セグメントは、以下の図に示すように、目標親の従属セグメントとして逆の順序でこの論理データベースの中に組み込まれます。
- 目標親の従属セグメントは、以下の図に示すように、目標親の従属セグメントとして順序を変更されることなく、この論理データベースに組み込まれます。

論理データベースで追加の論理関係が交差されていると、最初の交差の場合と同じようにして、その目標親の階層パスにあるすべてのセグメントへのアクセスが可能になります。

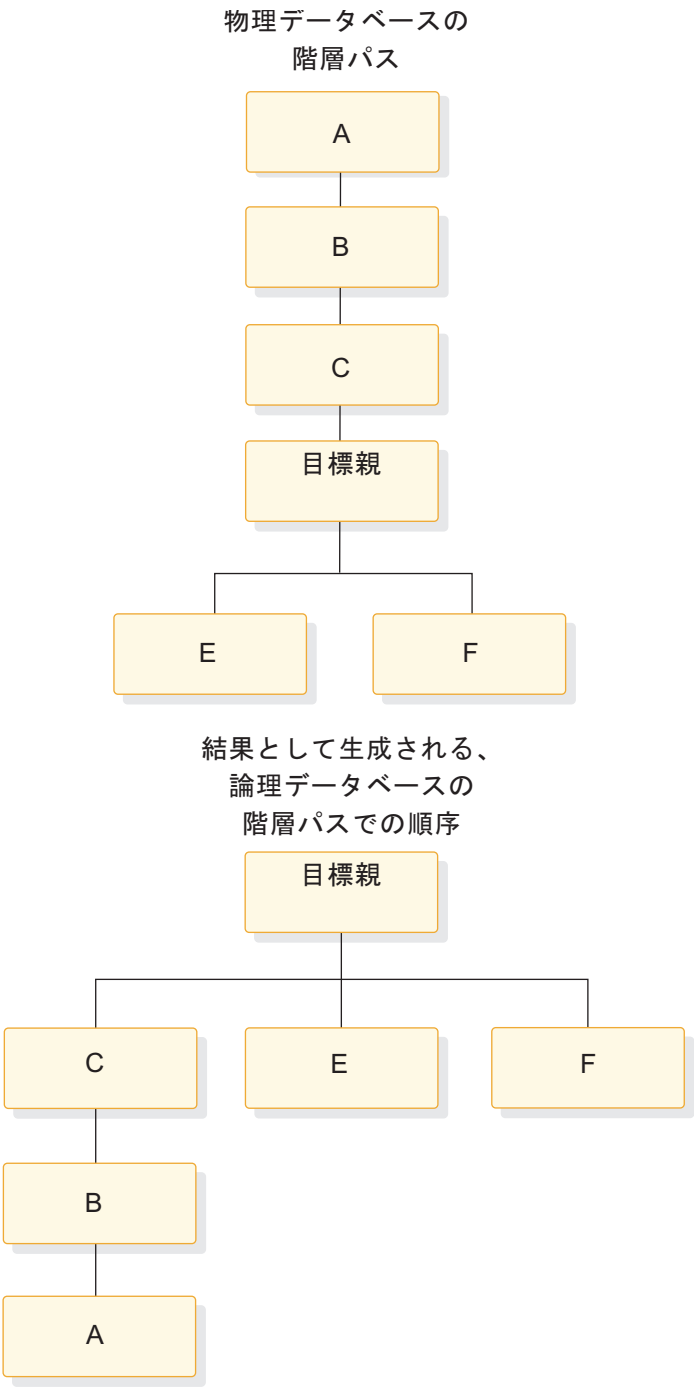


図 97. 最初の論理関係の交差によって可能となった論理データベース階層

論理データベースの定義に関する規則

- 論理データベースの中のルート・セグメントは、物理データベースの中のルート・セグメントでなければなりません。
- 論理データベースは、その論理 DBD で参照される物理 DBD の中で定義されているセグメント、および物理関係パスと論理関係パスだけを使用しなければなりません。

- 論理データベースの中の親と子を結合するのに用いられるパスは、その論理 DBD で参照される物理 DBD の中の物理関係パスないし論理関係パスとして定義されているものでなければなりません。
- 論理データベースの中の 1 つの階層セグメント・パスにおいて、物理関係パスと論理関係パスを混ぜて使用することができます。
- 論理データベースの中の 1 つの階層パスにおいて、ある 1 つの論理関係が交差された後、追加の物理関係パス、論理関係パス、あるいはその両方のパスを組み込むことができます。このパスは目標親から上方、下方、あるいはその両方に行くことによって組み込まれます。目標親から物理関係パスに沿って下方に進む場合には、論理関係が交差されない限り方向は変更できません。目標親から物理関係パスに沿って上方に進む場合には、方向を変えることができます。
- 論理データベースの中での従属セグメントの順序は、物理データベースの中のそれらの親のもとでの相対順序と同じでなければなりません。論理データベースの中の 1 つのセグメントが連結セグメントである場合には、論理子の複数の物理子と目標親の複数の子の間の順序は任意です。論理子の複数の子の相対順序および目標親の複数の子の相対順序は、変更されてはなりません。
- KEY センシティビティーおよび DATA センシティビティーのいろいろな組み合わせを用いて、同一の連結セグメント・タイプを複数回定義することができます。それぞれに連結セグメントのビューに対する別個の名前がなければなりません。そのうちの 1 つのビューだけが従属セグメントを持つことができます。以下の図は、1 つの論理データベースで定義できる同一の連結セグメントの 4 つのビューを示したものです。論理データベースの PCB は、連結セグメント・タイプの 1 つのビューに対してのみセンシティブになることができます。

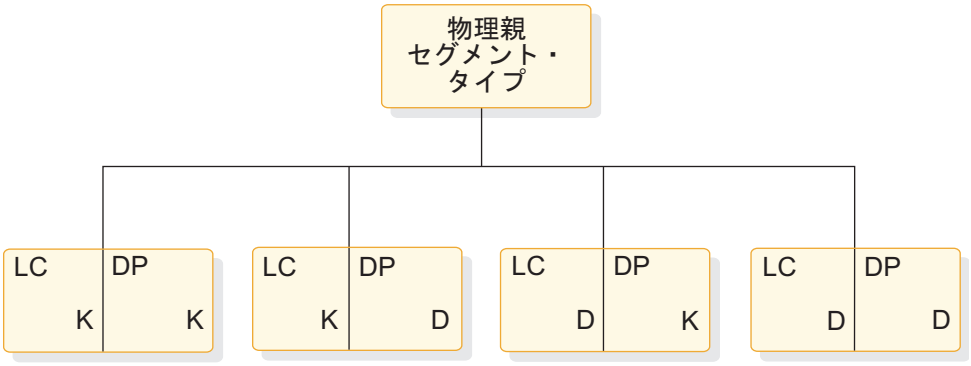


図 *98. KEY* センシティビティーと *DATA* センシティビティーの 各種組み合わせを用いて複数回定義される *1* つの連結セグメント・タイプ

LC 論理子セグメント・タイプ

DP 目標親セグメント・タイプ

K KEY センシティビティーがセグメント・タイプに対して指定されています。

D DATA センシティビティーがセグメント・タイプに対して指定されています。

論理関係の置き換え規則、挿入規則、および削除規則の選択

セグメントが論理関係に関与している場合には、挿入規則、削除規則、および置き換え規則を設ける必要があります。そのようなセグメントは、物理パスと論理パスの 2 つのパスから更新することができるからです。

以下の図と 309 ページの図 100 は、挿入規則、削除規則、および置き換え規則の例を示しています。以下の質問について検討してください。

1. 以下の図の CUSTOMER セグメントは、その物理パスと論理パスの両方から挿入可能にすべきか。
2. BORROW セグメントは、物理パスを用いてのみ置き換え可能にすべきか、あるいは物理パスと論理パスの両方を用いて置き換え可能にすべきか。
3. LOANS セグメントをその物理パスを用いて削除した場合、このセグメントをこのデータベースから消去すべきか。あるいは、このセグメントに物理的に削除という印を付けるだけで、論理パスを用いてこのセグメントにアクセス可能としておくべきか。
4. 論理子セグメント BORROW または連結セグメント BORROW/LOANS を物理パスから削除した場合、論理パス CUST/CUSTOMER も自動的に削除すべきか。あるいはこの論理パスは残すべきか。

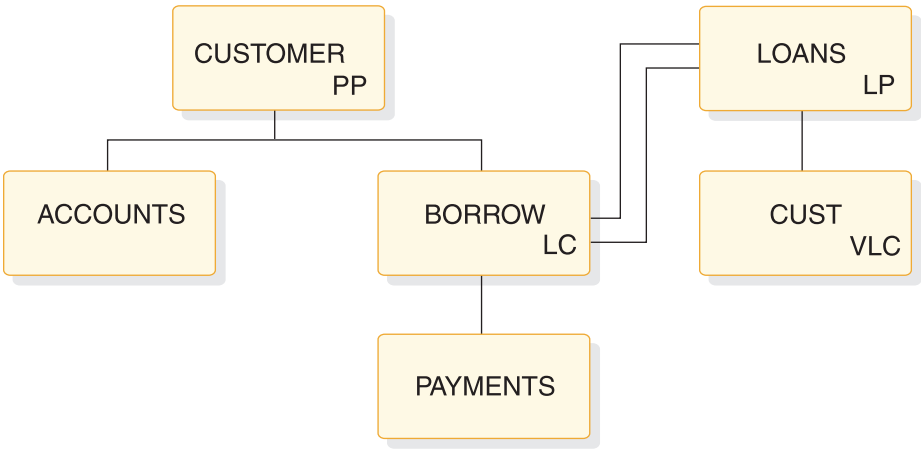


図 *99.* 置き換え規則、挿入規則、および削除規則の例

省略語 説明

PP 物理親セグメント・タイプ

LC 論理子セグメント・タイプ

LP 論理親セグメント・タイプ

VLC 仮想論理子セグメント・タイプ

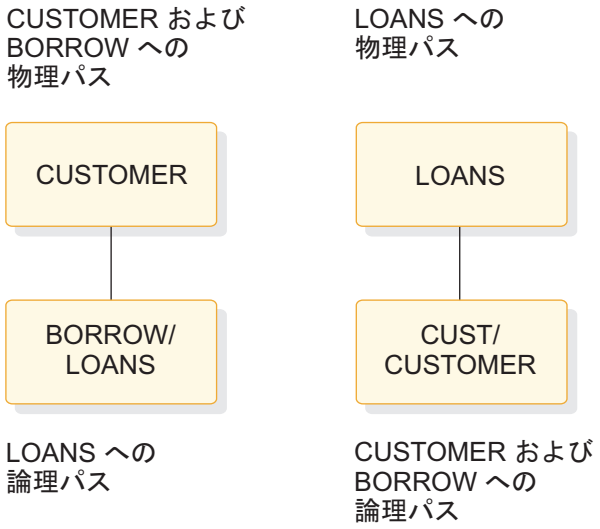


図 *100.* 置き換え規則、挿入規則、および削除規則の例: 前と後

これらの問に対する答はアプリケーション・プログラムによって異なります。しかし、答えの適用方法は、論理子セグメント、論理親セグメント、および物理親セグメントに対する適切な挿入規則、削除規則、および置き換え規則の選択方法によって決まります。最初にアプリケーションの処理要件を決定し、次にこの要件を満たす規則を決定しなければなりません。

例えば、問 1 に対する答は、このアプリケーションにおいて、ローンを受け入れる前にこのデータベースの中に CUSTOMER セグメントを挿入しなければならないかどうかによって決まります。 CUSTOMER セグメントの挿入規則が物理 (P) であれば、物理パス以外による CUSTOMER セグメントの挿入が禁止されています。挿入規則が仮想 (V) であれば、物理パスまたは論理パスによって CUSTOMER セグメントを挿入することができます。ローンを承認して BORROW セグメントを挿入する前に、客先について (過去の貸付限度、現在の仕事の継続年数など) を調査して CUSTOMER セグメントを挿入することは当然といえます。したがって、CUSTOMER セグメントの挿入規則を P として、このセグメントが論理的に挿入されることを防ぐべきでしょう。 (この例で挿入規則を使用すると、このアプリケーションを制御しやすくなります。)

あるいは、問 3 を検討してみます。このローン会社が、ある種のローンを廃止することとし (例えば、10% の自動車ローンを廃止し、12% の自動車ローンを設けることとし)、10% のローンを受けた者の中にまだ完済していない者が残っているうちにこの廃止を実施することがあれば、LOANS セグメントを物理的に削除しても、なお論理パスからこのセグメントにアクセス可能にすることができます。これを行うには、LOANS の削除規則として論理 (L) または (V) を指定し、P を指定しないようにすればよいことになります。

削除規則として P を指定すると、すべての論理子が物理的に削除される前に、その論理親セグメントを物理的に削除することが禁止されます。これはこの論理親への論理パスが最初に削除されることを意味します。

アプリケーションの要件をすべて調べる必要があり、さらに、論理関係に関与するセグメントの挿入、削除、および置き換えを誰が行うか決め、またどのようにして

(物理パスのみで、または物理パスと論理パスで) これらの更新を行うか決める必要があります。物理 DBD における挿入規則、削除規則、および置き換え規則と PCB 内の PROCOPT= パラメーターが制御の手段となります。

関連概念:

『論理関係の挿入規則、削除規則、および置き換え規則』

論理関係の挿入規則、削除規則、および置き換え規則

アプリケーションのすべての要件を調べて、論理関係に関与しているセグメントの挿入、削除、および置き換えを行うことのできる人を決め、さらに、この変更方法 (物理パスのみ、または物理パスと論理パス) を決める必要があります。

物理 DBD の中の挿入規則、削除規則、および置き換え規則によって、論理関係にわたってどのような更新が適用されるかが決まります。

このトピックには汎用プログラミング・インターフェース情報が含まれています。

関連概念:

308 ページの『論理関係の置き換え規則、挿入規則、および削除規則の選択』

796 ページの『使用可能な実記憶の使用状況』

関連資料:

354 ページの『接頭部記述子バイトの中のビット』

物理 DBD における規則の指定

挿入、削除、および置き換え規則は、論理関係の DBD で SEGM ステートメントの RULES= キーワードを使用して指定されます。

以下の図は、RULES= キーワードとそのパラメーターの図です。

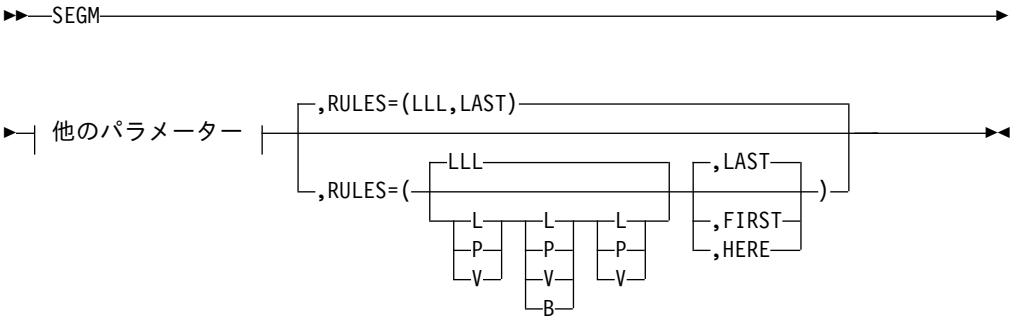


図 *101. DBD* における挿入規則、削除規則、および置き換え規則

挿入規則、置き換え規則、および削除規則の有効なパラメーター値は、以下のとおりです。

B 両方向仮想削除規則を指定します。これは、RULES= キーワードの最初または最後の定位置パラメーターには有効な値ではありません。

L 論理的な挿入、削除、または置き換え規則を指定します。

P 物理的な挿入、削除、または置き換え規則を指定します。

V　仮想の挿入、削除、または置き換え規則を指定します。

RULES= キーワードが受け入れる最初の 3 つの値は、定位置パラメーターです。

- 最初の定位置パラメーターは挿入規則を設定します。
- 2 番目の定位置パラメーターは削除規則を設定します。
- 3 番目の定位置パラメーターは置き換え規則を設定します。

例えば、RULES=(PLV) は、挿入規則が物理であり、削除規則が論理であり、置き換え規則が仮想であることを示しています。B 規則は削除のみに使えます。一般に、P 規則が最も制限が厳しく、V 規則が最も制限が穏やかであり、L 規則がほぼその中間です。

RULES= パラメーターは、論理パスにかかわるセグメント、すなわち、論理子セグメント、論理親セグメント、および物理親セグメントにのみ適用されます。仮想論理子には、RULES= パラメーターはコーディングされません。

挿入規則

挿入規則は、目標親セグメントに適用されますが、論理子セグメントには適用されません。

目標親は論理親または物理親です。挿入規則を論理子セグメントに指定しても、RULES= マクロのコーディング体系を満足させたという以外に何の意味もありません。したがって、論理子に対しては任意の挿入規則 (P、L、V) をコーディングできます。次の条件を満たす限り、論理子を挿入することができます。

- 目標親の挿入規則に違反していない。
- 挿入される論理子がすでに存在しているものでない (重複は許されない)

次に目標親への挿入規則の働きを説明します。

- RULES=P を指定すると、物理パスを用いてのみ、目標親を挿入することができます。これは、論理パスを挿入する前に目標親が存在しなければならないことを意味しています。連結セグメントは必要とされず、また論理子はそれ自体で挿入されます。
- RULES=L を指定すると、物理パスを用いて目標親を挿入することもできるし、論理子と連結された形で論理パスを用いて、目標親を挿入することもできます。論理子/目標親の連結セグメントを挿入する場合、目標親がまだ存在せず、入出力域でのキー検査に失敗しなければ、目標親が挿入されます。目標親が存在する場合には、これは変更されないまま、論理子がこれに結合されます。
- RULES=V を指定すると、物理パスを用いて目標親を挿入することもできるし、論理子と連結された形で論理パスを用いて、目標親を挿入することもできます。論理子/目標親の連結セグメントを挿入する場合、目標親がすでに存在していれば、目標親が置き換えられます。目標親がまだ存在していなければ、目標親が挿入されます。

関連概念:

312 ページの『ISRT 呼び出しの後に発行される可能性がある状況コード』

論理子の挿入呼び出し

論理子セグメントを挿入するには、アプリケーション・プログラムの入出力域に、目標親の挿入規則に従って、論理子または論理子/目標親連結セグメントのいずれかが入っていなければなりません。

すべての DL/I 呼び出しについては、エラーが検出されてエラー状況コードが戻されるか (この場合には、データは変更されない)、あるいはこの呼び出しの影響を受けるすべてのセグメントに必要な変更が加えられます。したがって、必要な機能が連結セグメントの両方の部分に対して実行できない場合には、エラー状況コードが戻されて、論理子と目標親のどちらにも変更が行われません。

挿入操作は、論理 DBD または PCB において指定されている KEY センシティビティーまたは DATA センシティビティーの影響を受けません。プログラムが、論理子と連結セグメントの目標親の両方に対して DATA センシティブであるもの以外のものであり、また、挿入規則が L または V である場合、そのプログラムは、論理パスを用いて挿入を行うときに、依然として、それら両方を入出力域に与えなければなりません。このため、連結セグメントを挿入する保守プログラムは、連結の両方のセグメントに対して DATA センシティブでなければなりません。

ISRT 呼び出しの後に発行される可能性がある状況コード

ISRT 呼び出しの後にアプリケーション・プログラムに戻されることのある非ブランク状況コードは次のとおりです。

- AM - 挿入が試みられたが、PROCOPT の値が「I」ではない。
- GE - 目標親または論理子の親が見つからなかった。
- II - 重複セグメントを挿入しようとする試みがなされた。
- IX - 指定された規則は P であったが、目標親がみつからなかった。

  IX 状況コードを受け取る理由の 1 つは、入出力域でのキー検査の失敗です。連結セグメントには、目標親のキーが 2 回入っていなければなりません。 1 回は論理子の LPCK の一部分として、もう 1 回は親の中の 1 つのフィールドとしてです。この 2 つのキーは、同じものでなければなりません。

以下の 2 つの図は、物理挿入規則の例を示したものです。

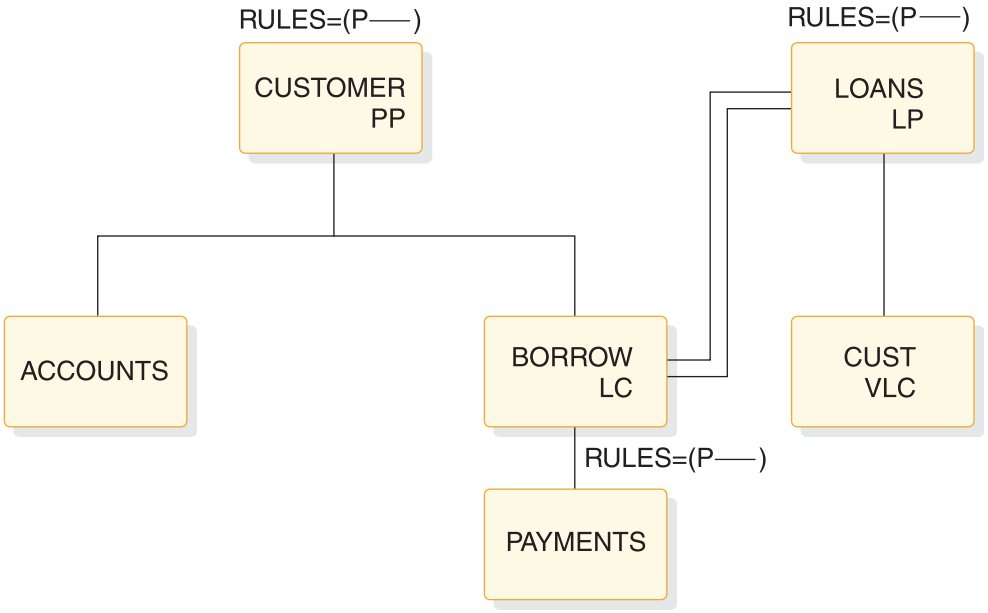


図 *102.* 物理挿入規則の例

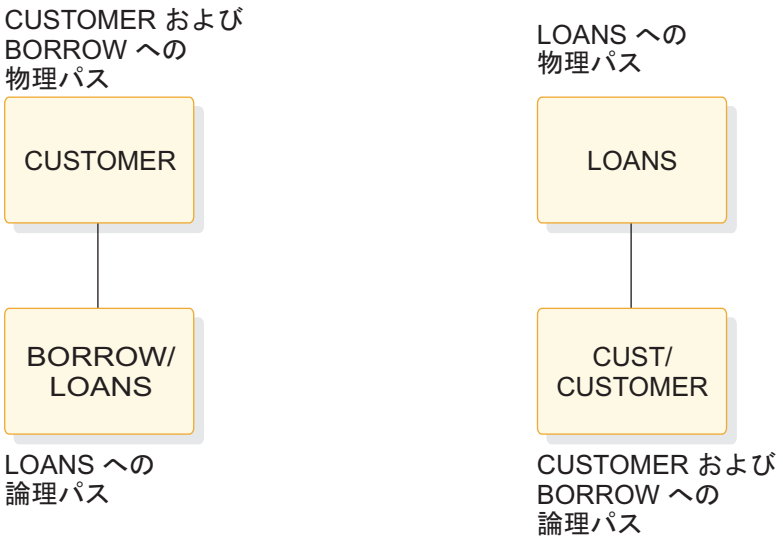


図 *103.* 物理挿入規則の例のパス

```
    ISRT 'CUSTOMER'  STATUS CODE='  '
ISRT 'BORROW'  STATUS CODE='  ' (LOANS が存在しない場合は、'IX')
```

図 *104.* 物理挿入規則の例の *ISRT* および状況コード

以下の 2 つの図は、論理挿入規則の例を示したものです。

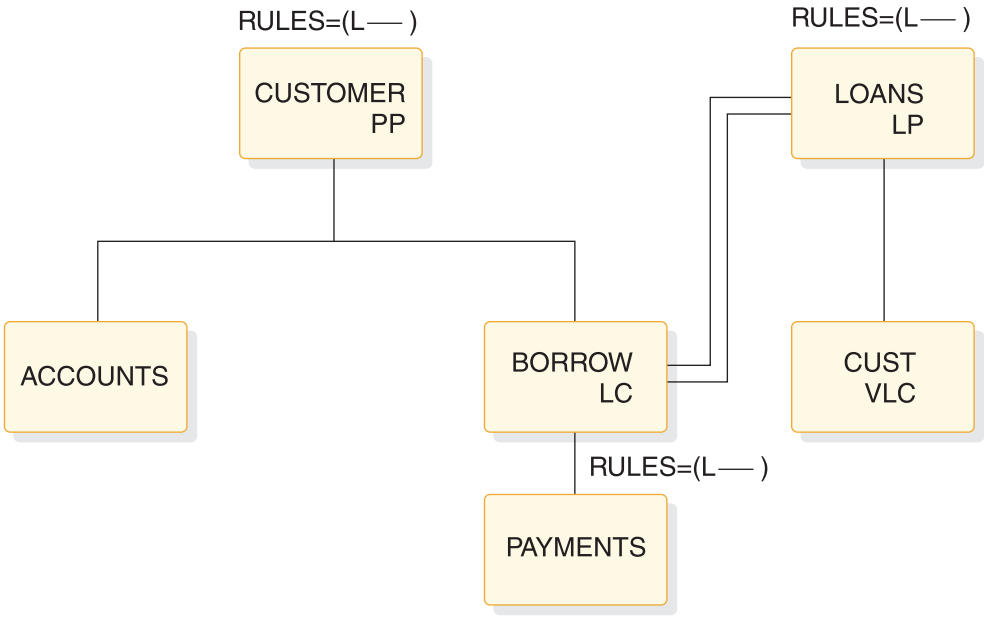


図 *105.* 論理挿入規則の例

```
ISRT 'LOANS'  STATUS CODE='  '
ISRT 'CUST'  STATUS CODE='IX'
```

図 *106.* 論理挿入規則の例の *ISRT* および状況コード

前の図に示されている IX 状況コードは、2 番目の呼び出しで連結セグメント CUST/CUSTOMER を省略した結果です。 IMS は、(入出力域の中で) CUSTOMER セグメントのキーを調べましたが、このキーが見つかりませんでした。挿入規則が L であれば、論理パスを作成するために連結セグメントを挿入しなければなりません。

以下の 2 つの図は、仮想挿入規則の例を示したものです。

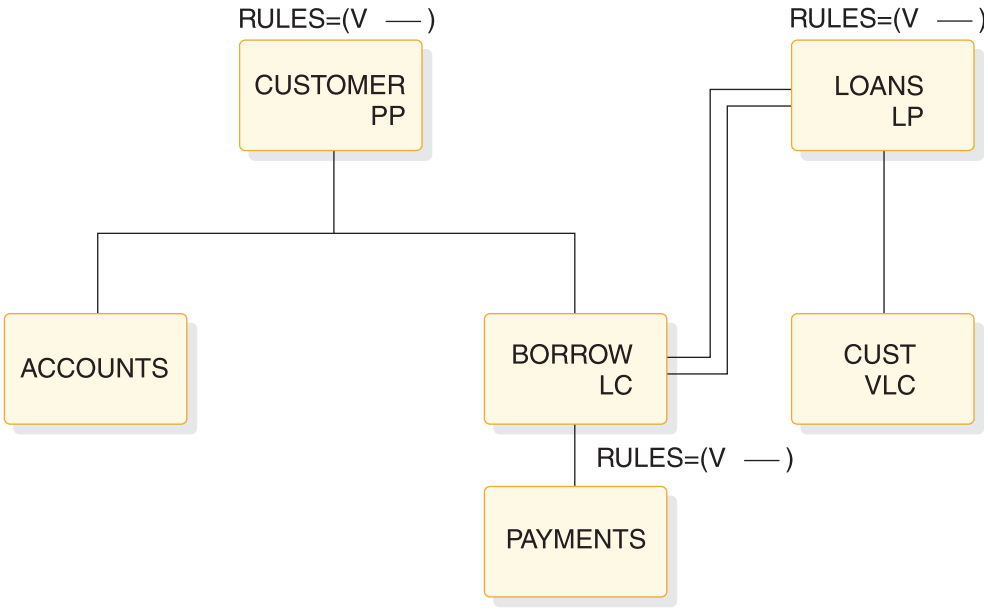


図 *107.* 仮想挿入規則の例

```
ISRT 'CUSTOMER'  STATUS CODE='  '
ISRT 'BORROW/LOANS'  STATUS CODE='  '
```

図 *108.* 仮想挿入規則の例の *ISRT* および状況コード

上記のコードでは、LOANS セグメントが存在すればそのセグメントが置き換えられ、存在しなければ LOANS セグメントが挿入されます。 V 挿入規則はきわめて強力なオプションです。

挿入規則の要約

挿入規則 P、L、および V について、以下に要約します。

挿入規則を P と指定すると、連結セグメントの一部としての目標親の挿入は行われなくなります。これは、目標親は、物理パスを使用してのみ挿入することができることを意味しています。挿入によって論理パスが作成された場合には、論理子のみを挿入する必要があります。

論理親と物理親に対して L という挿入規則を指定すると、物理パスを用いて、または連結セグメントの一部分として論理パスを用いて挿入を行うことができます。連結セグメントを挿入する場合、目標親がすでに存在していれば、この目標親には変更は加えられずに論理子がこれに結合されます。目標親が存在しない場合には、目標親が挿入されます。いずれの場合にも、論理子に関してはこの論理子が他と重複しているものでなく、目標親の挿入規則に違反していなければ、この論理子が挿入されます。

V 挿入規則が 3 つの規則のうちでは一番強力です。 V 挿入規則が最も強力な理由は、目標親が先に存在しなかった場合は、目標親が挿入され (論理パスを用いて連結セグメントとして挿入され)、目標親が存在した場合は、既存の目標親が挿入された目標親で置き換えられるからです。

置き換え規則

置き換え規則は、論理パスの物理親セグメント、論理親セグメント、および論理子セグメントに適用されます。

次に、置き換え規則の働きについて説明します。

- RULES=P と指定すると、物理パスを用いてセグメントを検索するときにのみ、このセグメントを置き換えることができます。この規則に違反すると、データは置き換えられず、RX という状況コードが戻されます。
- RULE=L と指定すると、物理パスを用いてセグメントを検索するときにのみ、このセグメントを置き換えることができます。この規則に違反すると、データは置き換えられません。しかし、RX 状況コードは戻されず、ブランクの状況コードが戻されます。
- RULES=V と指定すると、物理パス、論理パスのどちらで検索するときも、このセグメントを置き換えることができます。

関連概念:

『置き換え規則状況コード』

置き換え呼び出し

連結セグメントの中のアプリケーション・プログラムがデータ・センシティブである部分に対してのみ、置き換え操作を実施することができます。

セグメントの中のデータが変更されなければ、どのデータも置き換えられません。したがって、置き換え規則の違反はありません。連結セグメントの一部ではあっても検索されないセグメントに関しては、置き換え規則は調べられません。

すべての DL/I 呼び出しでは、エラーが検出されてエラー状況コードが戻されるか (この場合にはデータは変更されない)、あるいはこの呼び出しの影響を受けるセグメントのすべてに、必要な変更が加えられます。したがって、必要な機能が連結セグメントの両方の部分に対して実行できない場合には、エラー状況コードが戻されて、論理子と目標親のどちらにも変更が行われません。

置き換え規則状況コード

アプリケーション・プログラムに戻された状況コードは、検出された最初の置き換え規則違反を示しています。

この状況コードは次のとおりです。

- AM - 置き換えが試みられたが、PROCOPT の値が「R」ではない。
- DA - セグメントのキー・フィールドまたは交換不可能なフィールドが変更された。
- RX - 置き換え規則に対する違反があった。

置き換え規則の要約

以下の各表では、置き換え規則を要約します。

ある 1 つの論理関係の任意のセグメントに対して、P という置き換え規則を指定すると、そのセグメントの物理パスを用いてそのセグメントを検索するときを除き、そのセグメントの置き換えが防止されます。論理親の置き換え規則が L と指定され

ている場合には、この論理親が論理子と連結されてアクセスされると、IMS は、データを置き換えることなくブランクの状況コードを戻します。この論理子はその物理パスによってアクセスされているので、その置き換え規則は 3 つのうちいずれでもかまいません。このように、置き換え規則を使用することにより、連結セグメントの論理子部分の置き換えを選択的に行い、ブランクの状況コードを戻すことができます。ある 1 つの論理関係のセグメントに対して、V の置き換え規則を指定すると、物理パスまたは論理パスのどちらか一方のセグメントの置き換えを行います。

以下の各図に続く表では、指定可能な置き換え規則の組み合わせをすべて示しています。これらの図は、論理データベースの中にある連結セグメントを置き換えるための呼び出しが出されたとき、どのようなアクションがとられるのかそれぞれの組み合わせごとに示しています。これらの表は、以下の各図に示されているデータベースと論理ビューに基づいています。

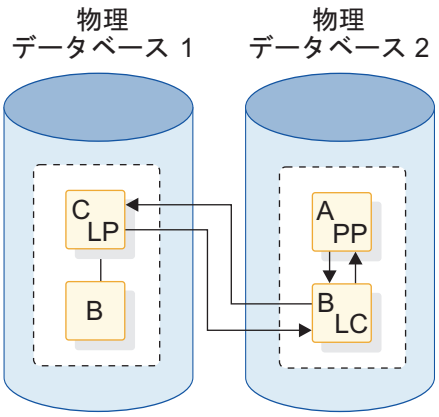


図 *109.* 置き換え規則の表における物理データベース

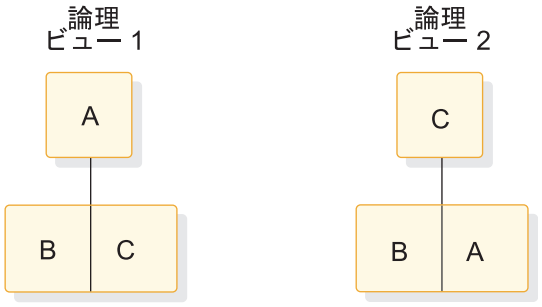


図 *110.* 置き換え規則の表における論理ビュー

置き換え規則 (論理ビュー 1 の場合)

表 *58.* 置き換え規則 (論理ビュー *1* の場合) :

指定した置き換え規則		置き換えようとしているセグメント		状況コード	データが置き換えられたか ?	
B	**C**	**B**	**C**		**B**	**C**
P	P	X			Y	
P	P		X	RX		N
P	P	X	X	RX	N	N

表 *58.* 置き換え規則 *(*論理ビュー *1* の場合*) (*続き*)*:

指定した置き換え規則		置き換えようとしているセグメント		状況コード	データが置き換えられたか **?**	
B	**C**	**B**	**C**		**B**	**C**
P	L	X			Y	
P	L		X			N
P	L	X	X		Y	N
P	V	X			Y	
P	V		X			Y
P	V	X	X		Y	Y
L	P	X			Y	
L	P		X	RX		N
L	P	X	X	RX	N	N
L	L	X			Y	
L	L		X			N
L	L	X	X		Y	N
L	V	X			Y	
L	V		X			Y
L	V	X	X		Y	Y
V	P	X			Y	
V	P		X	RX		N
V	P	X	X	RX	N	N
V	L	X			Y	
V	L		X			N
V	L	X	X		Y	N
V	V	X			Y	
V	V		X			Y
V	V	X	X		Y	Y

置き換え規則 **(**論理ビュー **2** の場合**)**

表 *59.* 置き換え規則 *(*論理ビュー *2* の場合*)*:

指定した置き換え規則		置き換えようとしているセグメント		状況コード	データが置き換えられたか **?**	
B	**A**	**B**	**A**		**B**	**A**
P	P	X		RX	N	
P	P		X	RX		N
P	P	X	X	RX	N	N
P	L	X		RX	N	
P	L		X			N
P	L	X	X	RX	N	N
P	V	X		RX	N	
P	V		X			Y

表 *59.* 置き換え規則 (論理ビュー *2* の場合) (続き) :

指定した置き換え規則		置き換えようとしているセグメント		状況コード	データが置き換えられたか **?**	
B	**A**	**B**	**A**		**B**	**A**
P	V	X	X	RX	N	N
L	P	X			N	
L	P		X	RX		N
L	P	X	X	RX	N	N
L	L	X			N	
L	L		X			N
L	L	X	X		N	N
L	V	X			N	
L	V		X			Y
L	V	X	X		N	Y
V	P	X			Y	
V	P		X	RX		N
V	P	X	X	RX	N	N
V	L	X			Y	
V	L		X			N
V	L	X	X		Y	N
V	V	X			Y	
V	V		X			Y
V	V	X	X		Y	Y

物理置き換え規則の例

以下の図とその後のコードは、物理置き換え規則の例を示しています。

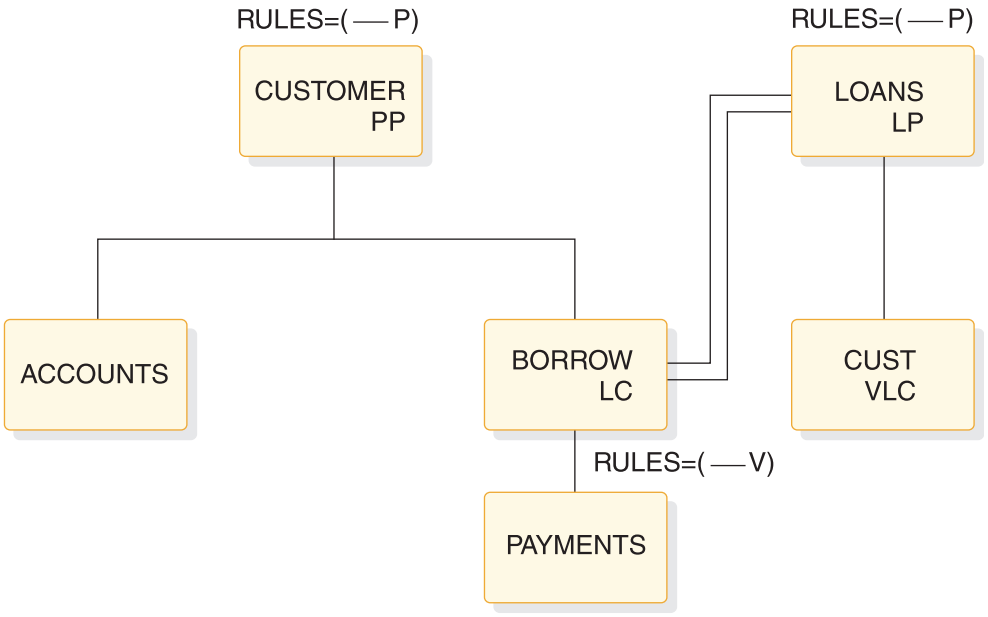


図 *111.* 物理置き換え規則の例

```
GHU 'CUSTOMER'  STATUS CODE='  '
REPL                STATUS CODE='  '
GHN  'BORROW/LOANS'  STATUS CODE='  '
REPL                STATUS CODE='RX'
```

図 *112.* 物理置き換え規則の例の呼び出しおよび状況コード

P 置き換え規則のために、連結セグメントの一部として LOANS セグメントを置き換えることはできません。置き換えは、セグメントの物理パスを用いて行わなければなりません。

関連概念:

321 ページの『論理置き換え規則の例』

322 ページの『仮想置き換え規則の例』

論理置き換え規則の例

以下の図とその後のコードは、論理置き換え規則の例を示しています。

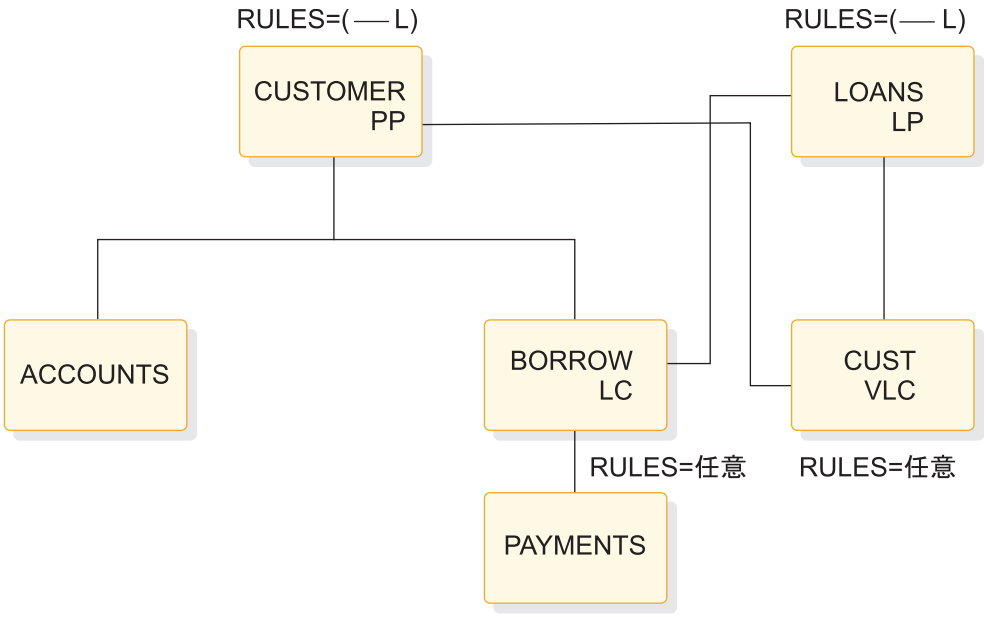


図 *113.* 論理置き換え規則の例

```
GHU 'CUSTOMER'
    'BORROW/LOANS'  STATUS CODE='  '
REPL                STATUS CODE='  '
```

図 *114.* 論理置き換え規則の例の呼び出しおよび状況コード

前の図に示したように、L 置き換え規則のために、連結セグメントの一部として LOANS セグメントを置き換えることはできません。置き換えは、セグメントの物理パスを用いて行わなければなりません。しかし、戻される状況コードはブランクです。物理パスからアクセスされた BORROW セグメントは、置き換えられます。論理子にはその物理パスからアクセスされるので、どの置き換え規則が選定されているかは問題となりません。

L 置き換え規則では、連結セグメントの論理子部分のみを置き換えることができ、ブランクの状況コードが戻されます。

関連概念:

320 ページの『物理置き換え規則の例』

322 ページの『仮想置き換え規則の例』

仮想置き換え規則の例

以下の図とその後のコードは、仮想置き換え規則の例を示しています。

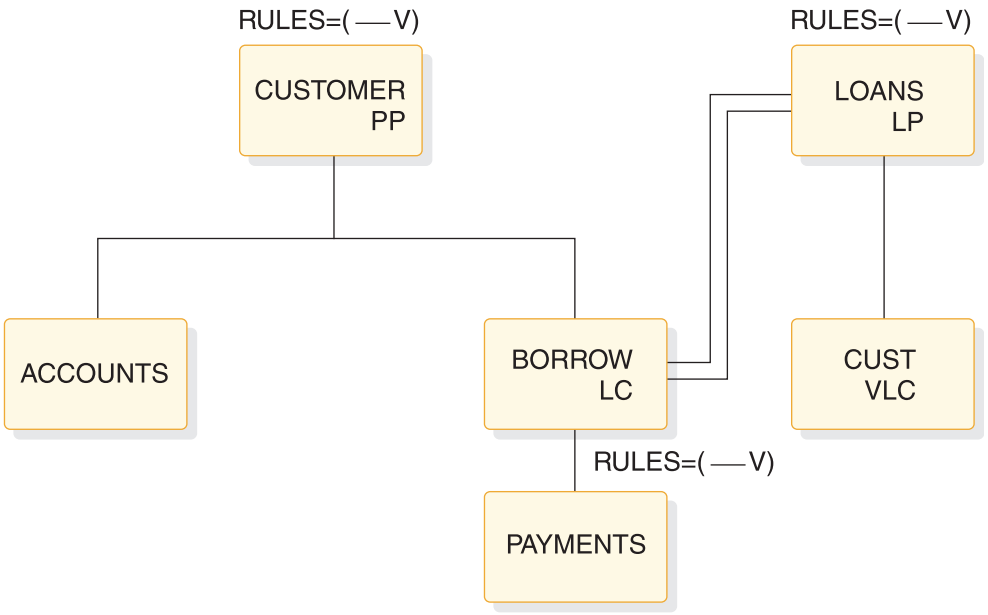


図 *115.* 仮想置き換え規則の例

```
GHU 'LOANS'
    'CUST/CUSTOMER'  STATUS CODE='  '
REPL                 STATUS CODE='  '
```

図 *116.* 仮想置き換え規則の例の呼び出しおよび状況コード

上のコードに示されているように、V 置き換え規則のもとでは、CUSTOMER セグメントをその論理パスを用いて連結セグメントの一部分として置き換えることができます。

関連概念:

321 ページの『論理置き換え規則の例』

320 ページの『物理置き換え規則の例』

削除規則

以下のトピックでは、削除値が論理親、物理親、および論理子に対してどのように働くかを説明します。

論理親の削除規則

以下のリストでは、RULES=P、RULES=L、または RULES=V が指定されているときに論理親を削除すると何が行われるかを説明します。

- RULES=P を指定すると、論理親を論理的に削除した後でないと、DLET 呼び出しはこのセグメントまたはその任意の物理親に対して有効にはなりません。論理親を論理的に削除していないと、この呼び出しは DX 状況コードを戻し、どのセ

グメントも削除されません。しかし、ある論理関係にまたがった波及の結果として、削除要求があるセグメントに出された場合には、P 規則は、次に説明する L 規則と同じ働きをします。

- RULES=L を指定する場合は、最初の削除は物理的な削除でも論理的な削除でもかまいません。論理親が DLET 呼び出しによって処理される場合、論理子はすべて論理的に削除されますが、この論理親は依然として論理子からアクセス可能です。
- RULES=V を指定すると、論理親が DLET 呼び出しによって削除されるとき、この論理親はその物理パスに沿って明示的に削除されます。その論理子はすべて論理的に削除されますが、この論理親は依然としてその論理子からアクセス可能です。

  論理親がもはや論理関係に関与しなくなるとき、この論理親は、その物理パスに沿って暗黙のうちに削除されます。次のようなとき、論理親はもはや論理関係に関与しなくなります。

  – 論理親がそれを指す論理子を持たず (論理子カウンターがあれば、これがゼロである)
  – 物理親がどの論理子も指しておらず (論理子ポインターがあれば、すべての論理子ポインターがゼロである)
  – 物理親が、実論理子でもある物理子をもっていない

物理親 (仮想対のみ) の削除規則

以下のリストでは、仮想対のみを使用する物理親の削除規則について説明します。

- PHYSICAL/LOGICAL/VIRTUAL は、意味がありません。
- BIDIRECTIONAL VIRTUAL (両方向仮想) は、物理親がもはや論理関係に関与しなくなると、この物理親はその物理パスに沿って自動的に削除されることを意味します。次のようなとき、物理親はもはや論理関係に関与しなくなります。
  – 物理親がその親を指す論理子を待たず (論理子カウンターがあれば、これがゼロである)
  – 物理親がどの論理子も指しておらず (論理子ポインターがあれば、すべての論理子ポインターがゼロである)
  – 物理親が、実論理子でもある物理子をもっていない

論理子の削除規則

以下のリストでは、RULES=P、RULES=L、または RULES=V が指定されているときに論理子を削除すると何が行われるかを説明します。

- RULES=P を指定する場合は、最初にこの論理子セグメントを論理的に削除し、次にこれを物理的に削除する必要があります。物理的な削除を最初に試みると、このセグメントまたはその任意の物理親を対象として出された DLET 呼び出しは、DX 状況コードを戻し、どのセグメントも削除されません。論理関係にまたがった波及の結果として、削除要求がこのセグメントに対して出された場合、またはこのセグメントが物理対の 1 つである場合には、この規則は次に述べる L 規則と同じ働きをします。

- RULES=L を指定する場合は、論理子の削除は、この削除が要求されたパスに対して有効です。論理子の物理的な削除と論理的な削除は、任意の順序で実行することができます。論理子および任意の物理従属セグメントは、削除されていないパスから依然としてアクセス可能です。
- RULES=V を指定すると、論理子がその論理パスまたは物理パスを通じて削除されたとき、この論理子が論理的かつ物理的に削除されます (PD ビットまたは LD ビットのいずれかを設定すると、両方のビットが設定されます)。物理対の一方の論理子セグメントに対してのみこの規則が指定されると、この規則は L 規則と同じ働きをします。

注: 単一方向論理関係に関与している論理子に関しては、この 3 つの規則の意味は同じであり、したがって 3 つの規則のいずれを指定してもかまいません。

削除規則の使用例

以下の一連の図は、削除規則を指定できるそれぞれのセグメント・タイプ (論理親、物理親、およびそれらの論理子) に対する削除規則の使い方を示したものです。

それぞれの図は、例に当てはまる規則のみを示しています。例に添えてある説明は、この特定の例にのみ当てはまります。

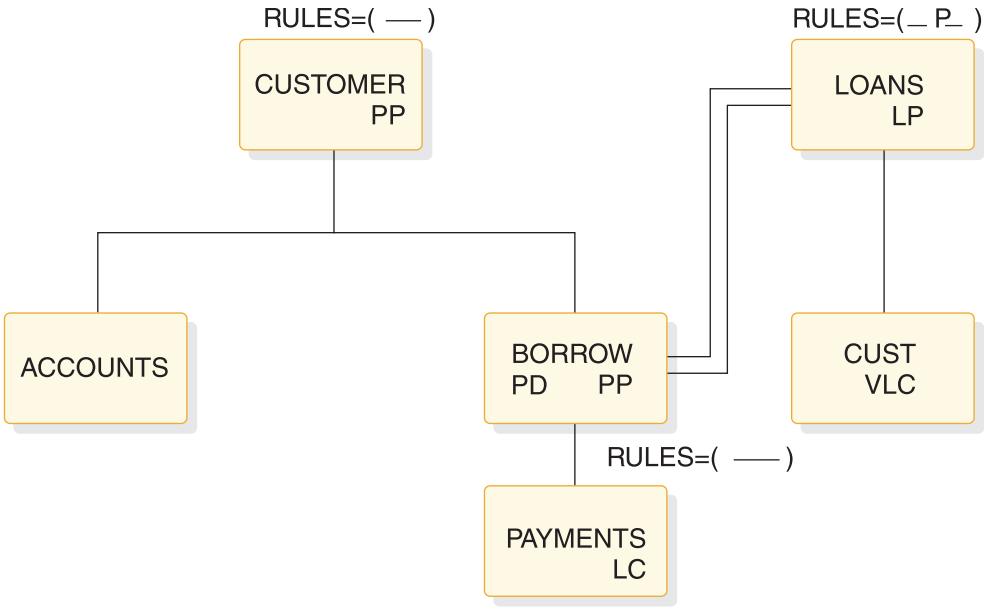


図 *117.* 論理親、仮想対 - 物理削除規則の例

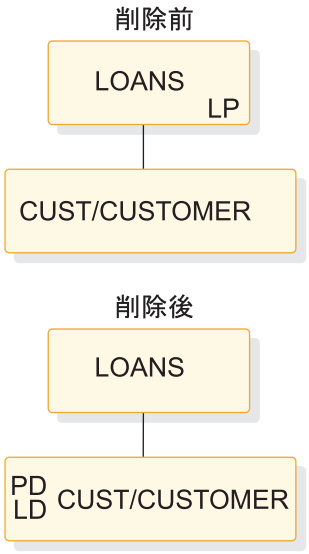


図 *118.* 論理親、物理対 - 物理削除規則の例: 削除前および削除後

```
GHU 'LOANS'  STATUS='  '
DLET         STATUS='  '
```

図 *119.* 論理親、仮想対 - 物理削除規則の例: データベース呼び出し

物理削除規則では、すべての論理子をあらかじめ物理的に削除しておかなければなりません。論理親の物理従属セグメントは物理的に削除されます。

すべての論理子をあらかじめ物理的に削除しておかないと、DLET 状況コードが「DX」 になります。すべての論理子が論理的に削除されます。物理論理子 BORROW で LD ビットがオンに設定されます。

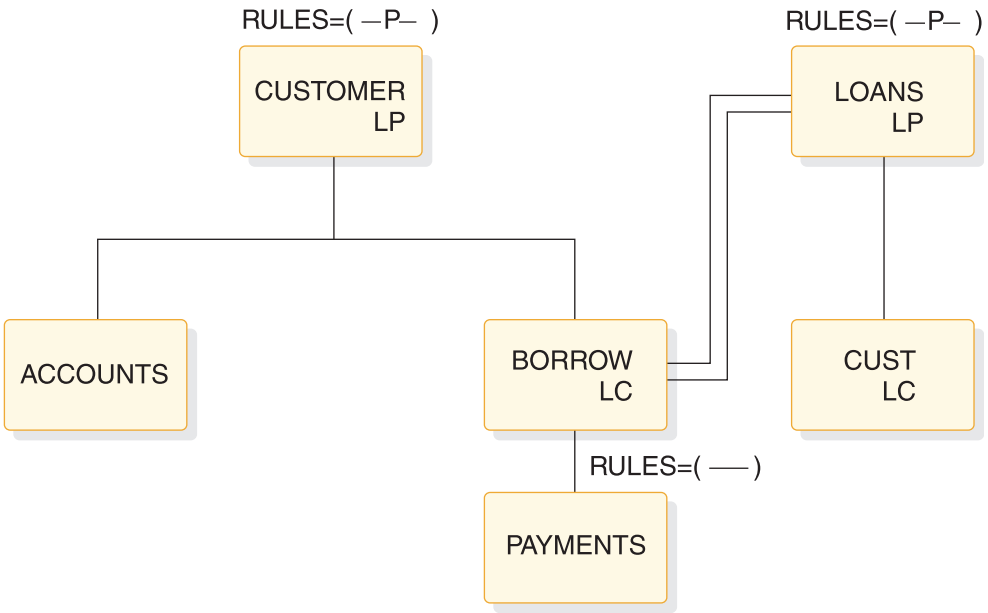


図 *120.* 論理親、物理対 - 物理削除規則の例

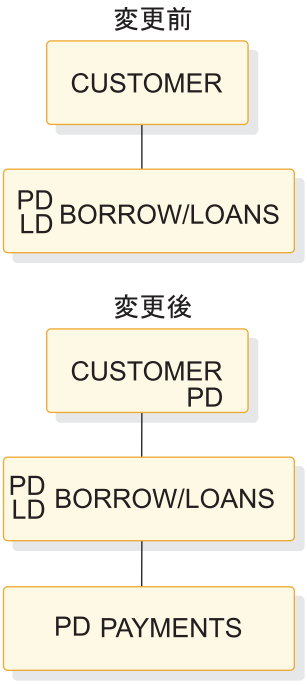


図 *121.* 論理親、物理対 - 物理削除規則の例: 削除前および削除後

```
GHU 'CUSTOMER'  STATUS='  '
DLET               STATUS='  '
```

図 *122.* 論理親、物理対 - 物理削除規則の例: 呼び出しおよび状況コード

物理削除規則では、次のことが必要です。

- すべての論理子をあらかじめ物理的に削除する。
- この論理子と対をなす物理子をあらかじめ削除する。

論理親 CUSTOMER は物理的に削除されています。論理子とその対があらかじめ物理的に削除されています。 (削除前の図では BORROW/LOANS で PD ビットと LD ビットがオンに設定されています。)

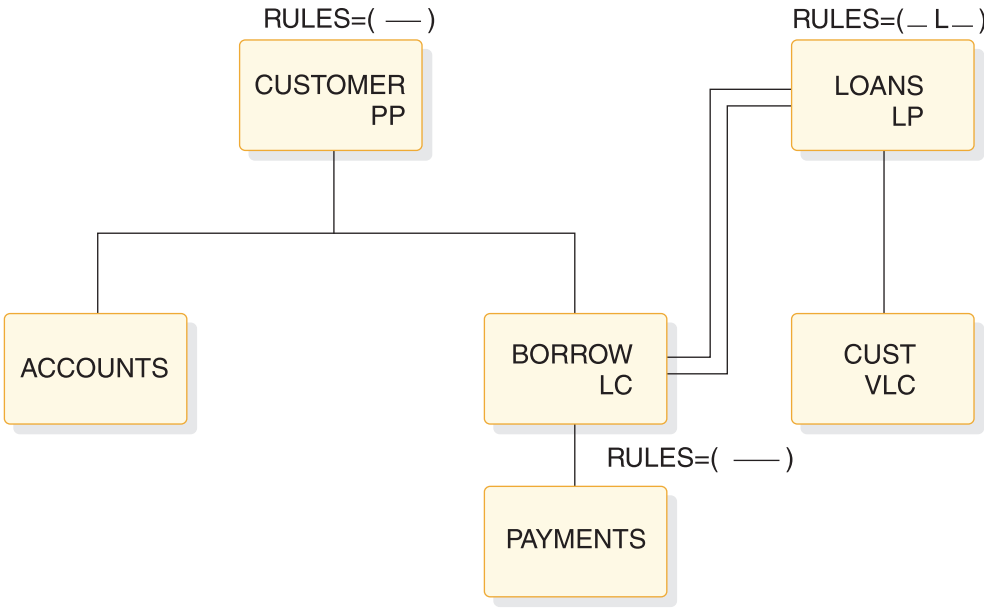


図 *123.* 論理親、仮想対 - 論理削除規則の例

図 *124.* 論理親、仮想対 - 論理削除規則の例: 削除前および削除後

```
GHU 'LOANS'  STATUS='  '
DLET         STATUS='  '
```

図 *125.* 論理親、仮想対 - 論理削除規則の例: 呼び出しおよび状況コード

論理削除規則では、物理的な削除と論理的な削除のどちらを最初に行ってもかまいません。一方の削除によって他方の削除が行われることはありません。論理親の物理従属セグメントは物理的に削除されます。

論理親 LOANS は、論理子からアクセス可能なままです。すべての論理子が論理的に削除されます。物理子 BORROW で LD ビットがオンに設定されます。

論理親 LOANS の削除規則が、論理削除規則でなく仮想削除規則であったとしても、 327 ページの図 123 の処理と結果は同じです。次に示す例は、論理削除規則を説明する追加の例です。

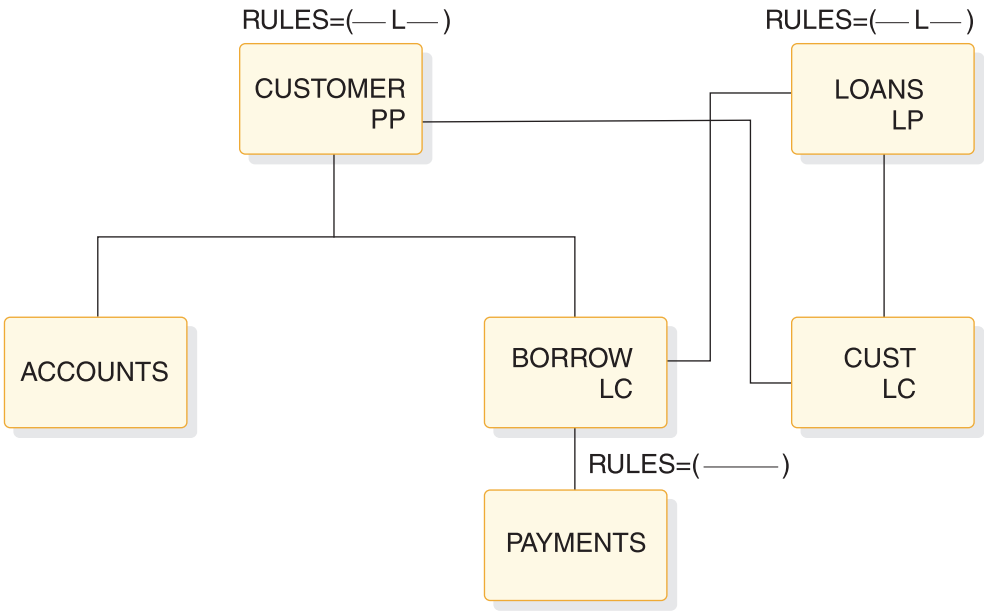


図 *126.* 論理親、物理対 - 論理削除規則の例

図 *127.* 論理親、物理対 - 論理削除規則の例: 削除前および削除後

```
GHU 'LOANS'  STATUS='  '
DLET         STATUS='  '
```

図 *128.* 論理親、物理対 - 論理削除規則の例: 呼び出しおよび状況コード

論理削除規則では、物理的な削除と論理的な削除のどちらを最初に行ってもかまいません。一方の削除によって他方の削除が行われることはありません。論理親の物理従属セグメントは物理的に削除されます。

論理親 LOANS は、論理子からアクセス可能なままです。すべての物理子が物理的に削除されます。対をなす論理子は論理的に削除されます。

論理親 LOANS の削除規則が、論理削除規則でなく仮想削除規則であったとしても、 328 ページの図 126 の処理と結果は同じです。仮想削除規則について説明しているもう 1 つの例を以下の図に示します。

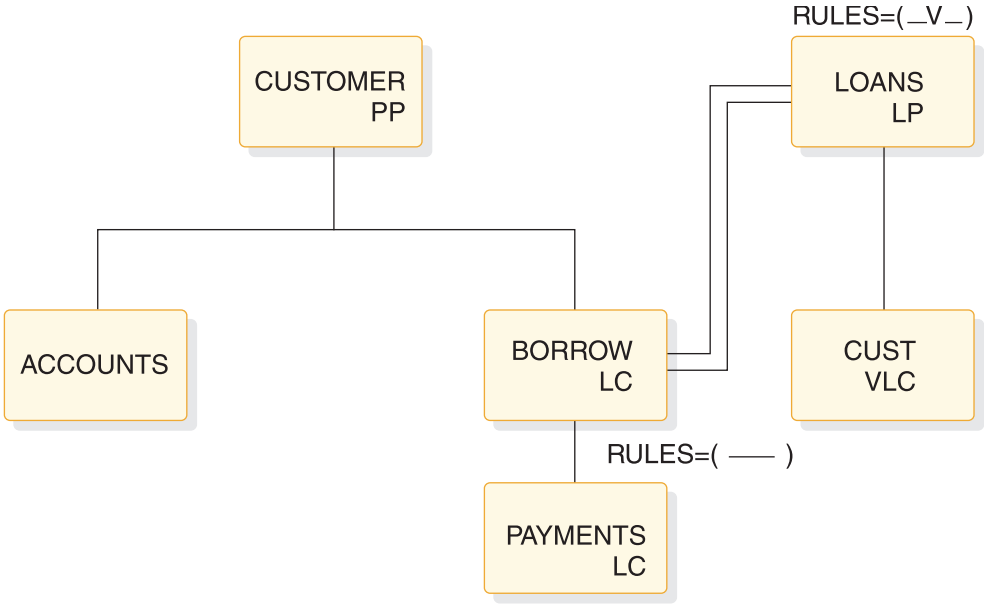


図 *129.* 論理親、仮想対 - 仮想削除規則の例

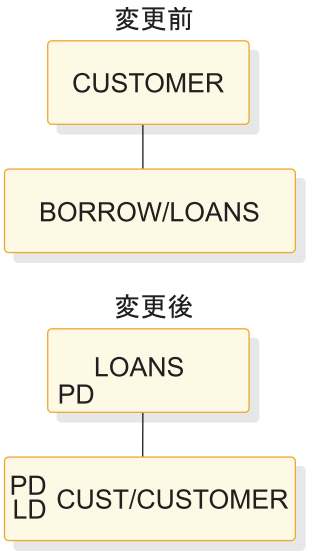


図 *130.* 論理親、仮想対 - 仮想削除規則の例: 削除前および削除後

```
GHU 'CUSTOMER'
    'BORROW/LOANS'  STATUS='  '
DLET                STATUS='  '
```

図 *131.* 論理親、仮想対 - 仮想削除規則の例: 呼び出しおよび状況コード

仮想削除規則では、明示的な削除と暗黙の削除が可能です。明示的な削除は、論理規則を使用する場合と同じです。暗黙の削除では、最後の論理子が物理的に削除されると、論理親が物理的に削除されます。

論理子の物理従属セグメントが物理的に削除されます。論理親が物理的に削除されます。論理親の物理従属セグメントが物理的に削除されます。物理論理子 BORROW で LD ビットがオンに設定されます。

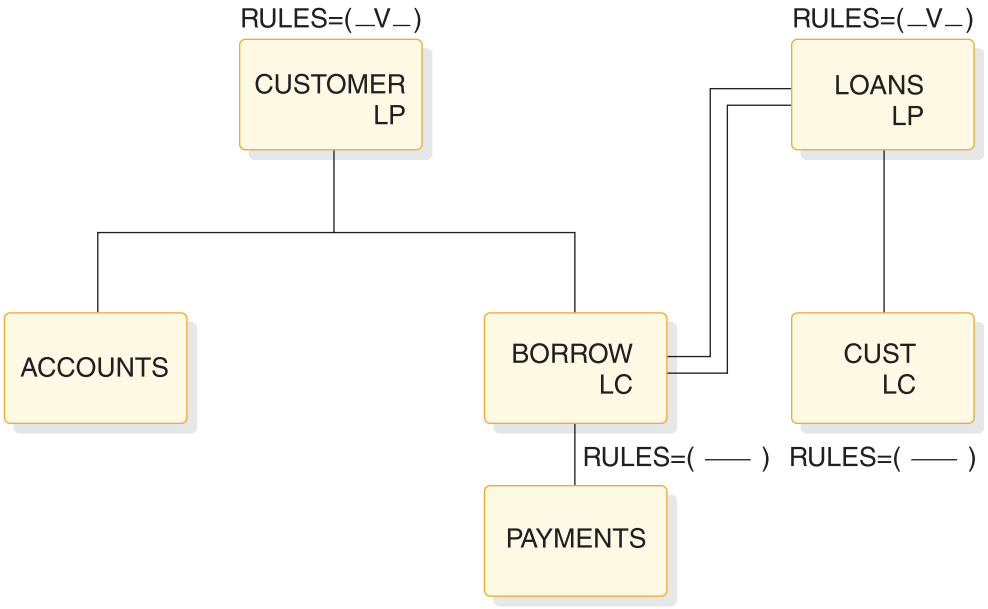


図 *132.* 論理親、物理対 - 仮想削除規則の例

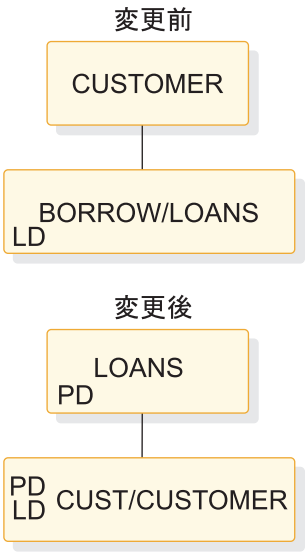


図 *133.* 論理親、物理対 - 仮想削除規則の例: 削除前および削除後

```
GHU 'CUSTOMER'
    'BORROW/LOANS'  STATUS='  '
DLET                STATUS='  '
```

図 *134.* 論理親、物理対 - 仮想削除規則の例: 呼び出しおよび状況

仮想削除規則では、明示的な削除と暗黙の削除が可能です。明示的な削除は、論理規則を使用する場合と同じです。暗黙の削除では、最後の論理子が物理的および論理的に削除されると、論理親が物理的に削除されます。

論理親が物理的に削除されます。論理親のすべての物理従属セグメントが物理的に削除されます。

注: CUST セグメントは DLET 呼び出しが出される前に物理的に削除しなければなりません。 BORROW セグメントで LD ビットがオンに設定されます。

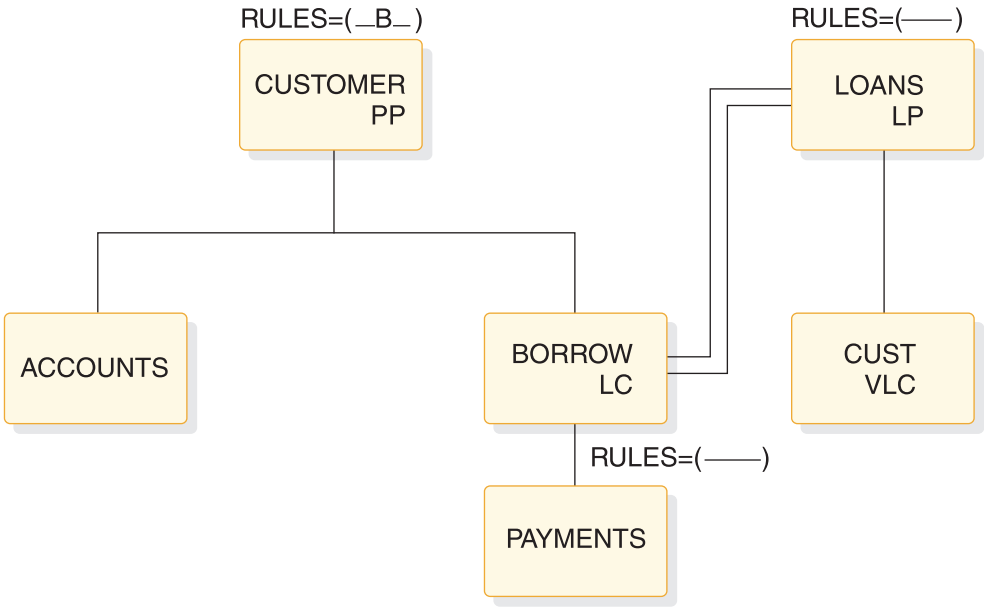


図 *135.* 物理親、仮想対 - 両方向仮想削除規則の例

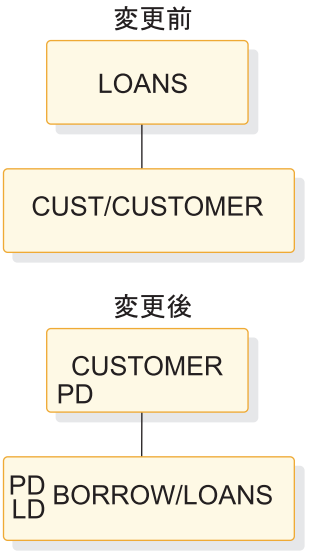


図 *136.* 物理親、仮想対 - 両方向仮想削除規則の例: 削除前および削除後

```
GHU 'LOANS'
    'CUSTOMER'  STATUS='  '
DLET            STATUS='  '
```

図 *137.* 最後の論理子を削除すると物理親が削除される

物理親を対象とする両方向仮想削除規則は、論理親を対象とする仮想削除規則と同じ効果があります。

最後の論理子が論理的に削除されると、物理親が物理的に削除されます。論理子が (物理親の従属セグメントとして) 物理的に削除されます。物理親のすべての物理従属セグメントが物理的に削除されます。すなわち、ACCOUNTS (図には示されてい

ません)、BORROW、および PAYMENT が物理的に削除されます。

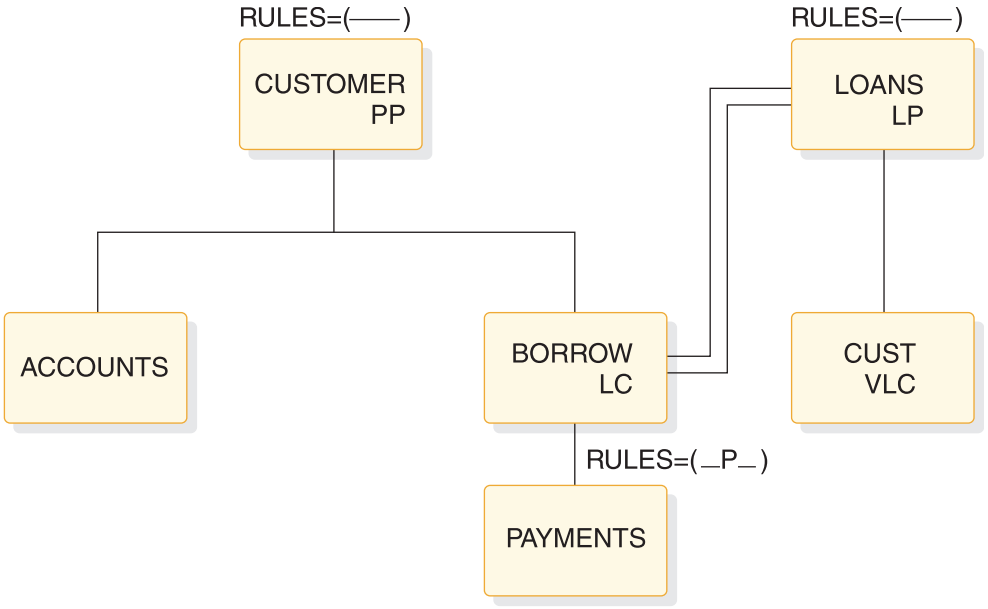


図 *138.* 論理子、仮想対 - 物理削除規則の例

```
GHU 'LOANS'          STATUS='  '
    'CUST/CUSTOMER'
DLET                 STATUS='  '

GHU 'CUSTOMER'       STATUS='  '
    'BORROW/LOANS'
DLET                 STATUS='  '
```

図 *139.* 論理子、仮想対 - 物理削除規則の例*:* 論理子の削除

物理削除規則では、最初に論理子を論理的に削除しなければなりません。この時点で BORROW セグメントで LD ビットが設定されます。

論理子は論理的に削除した後で初めて物理的に削除することができます。 2 回目の削除の後、LD ビットと PD ビットの両方が設定されます。論理子を物理的に削除すると、この論理子の物理従属セグメントも物理的に削除されます。 PD ビットが設定されます。

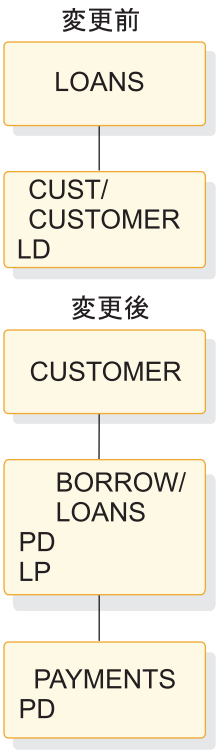


図 *140.* 論理子、仮想対 - 物理削除規則の例: 削除前および削除後

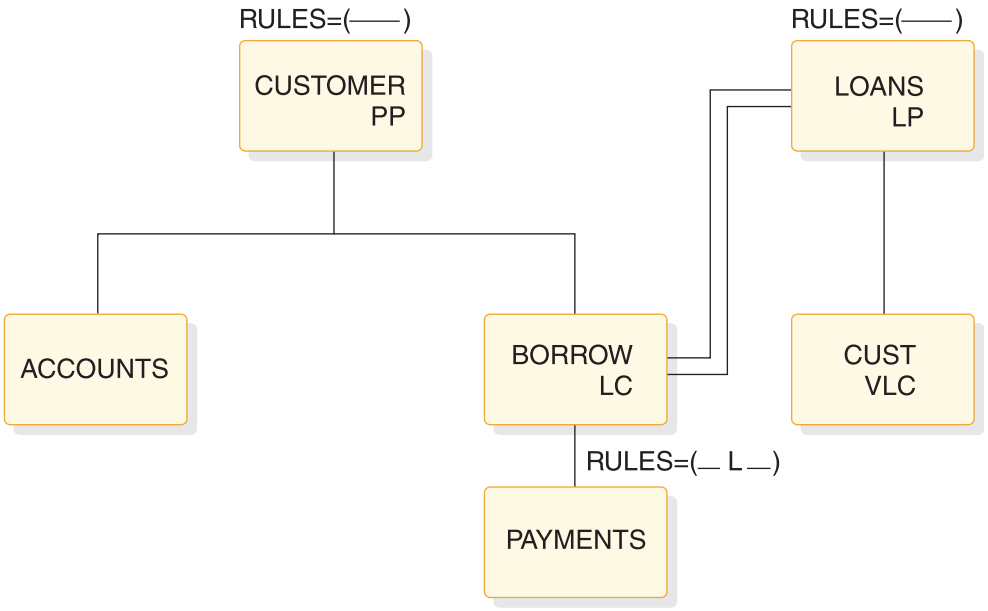


図 *141.* 論理子、仮想対 - 論理削除規則の例

```
GHU 'CUSTOMER   STATUS='  '
    'BORROW/LOANS'
DLET            STATUS='  '

GHU 'LOANS'     STATUS='  '
    'CUST/CUSTOMER'
DLET            STATUS='  '
```

図 *142.* 論理子、仮想対 - 論理削除規則の例: 呼び出しおよび状況

論理削除規則では、論理子を最初に物理的に削除しても論理的に削除してもかまいません。論理子の物理従属セグメントは物理的に削除されますが、論理的に削除されていない論理パスからは物理削除後もアクセスすることができます。

仮想論理子を削除すると、物理論理子 BORROW で LD ビットがオンに設定されます (BORROW は論理的に削除されます)。

図 *143.* 論理子、仮想対 - 論理削除規則の例: 削除前および削除後

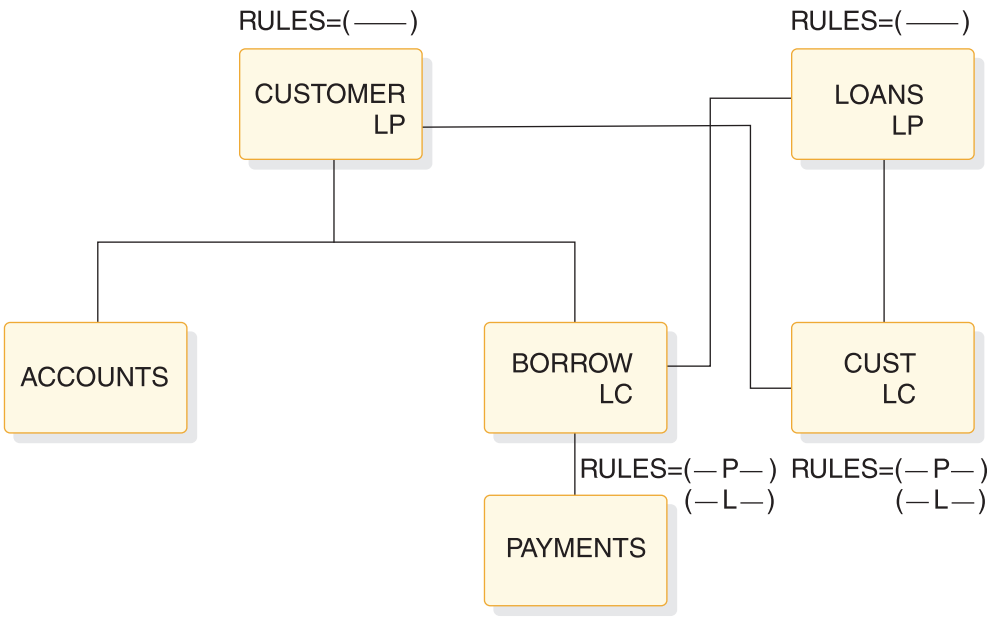


図 *144.* 論理子、物理対 - 物理削除規則または論理削除規則の例

```
GHU 'CUSTOMER   STATUS='  '
    'BORROW/LOANS'
DLET            STATUS='  '

GHU 'LOANS'     STATUS='  '
    'CUST/CUSTOMER'
DLET            STATUS='  '
```

図 *145.* 論理子、物理対 - 物理削除規則または論理削除規則の例: 呼び出しおよび状況

物理削除規則または論理削除規則では、論理子をそれぞれ、物理パスから削除しなければなりません。論理子の物理従属セグメントは物理的に削除されますが、対をなす論理子が削除されていない場合は、物理削除後もその対の論理子からアクセスすることができます。

BORROW を物理的に削除すると、CUST で LD ビットがオンに設定されます。CUST を物理的に削除すると、BORROW セグメントで LC ビットがオンに設定されます。

図 *146.* 論理子、物理対 - 物理削除規則または論理削除規則の例: 削除前および削除後

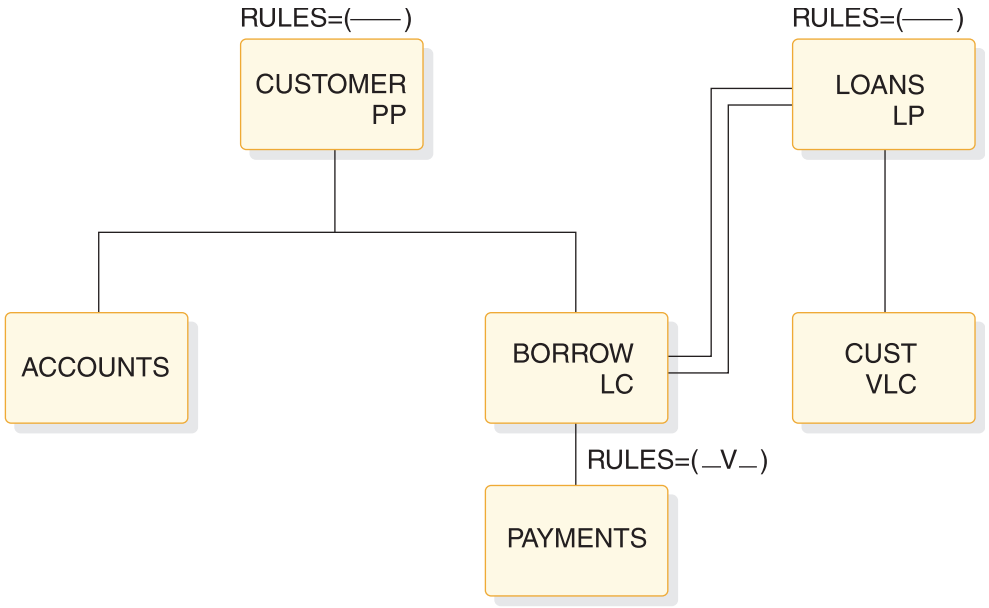


図 *147.* 論理子、仮想対 - 仮想削除規則の例

```
GHU 'CUSTOMER   STATUS='  '
    'BORROW/LOANS'
DLET            STATUS='  '

GHU 'LOANS'     STATUS='GE'
    'CUST/CUSTOMER'
```

図 *148.* 論理子、仮想対 - 仮想削除規則の例: 呼び出しおよび状況

仮想削除規則では、論理子を物理的にも論理的にも削除できます。どちらかのパスを削除すると、両方の部分が削除されます。論理子の物理従属セグメントが物理的に削除されます。

削除規則が仮想規則であるため、前の削除によって両方のパスが削除されました。どちらかのパスを削除すると、両方が削除されます。

図 *149.* 論理子、仮想対 - 仮想削除規則の例: 削除前および削除後

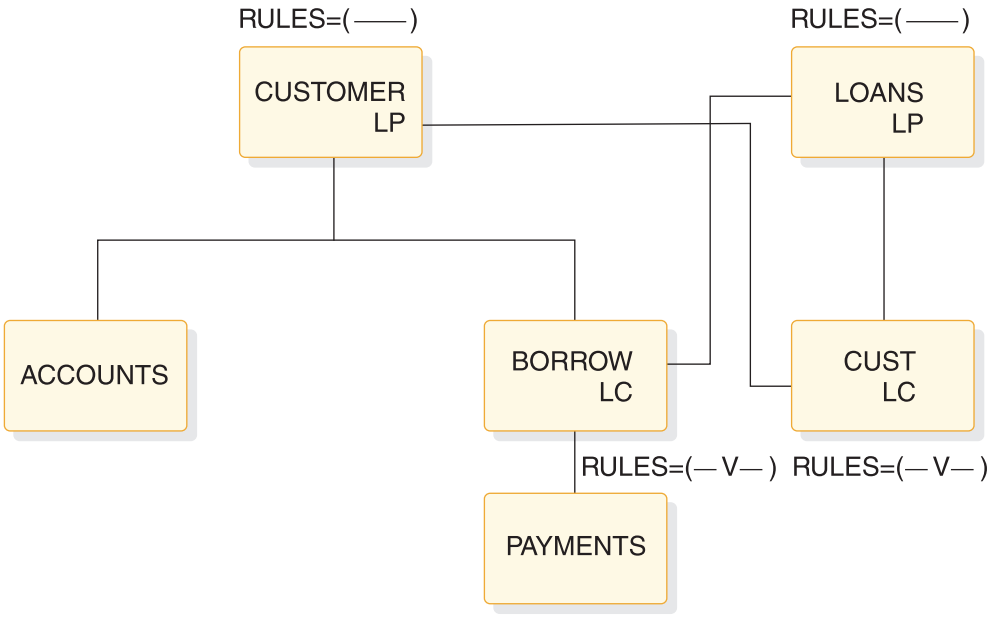


図 *150.* 論理子、物理対 - 仮想削除規則の例

```
GHU 'CUSTOMER   STATUS='  '
DLET            STATUS='  '

GHU 'LOANS'     STATUS='GE'
    'CUST/CUSTOMER'
```

図 *151.* 論理子、物理対 - 仮想削除規則の例: 呼び出しおよび状況

仮想削除規則では、対をなす論理子の一方を削除すると、対の両方が削除されます。 (両方の論理子で PD ビットと LD ビットがオンに設定されます。) 論理子の物理従属セグメントが物理的に削除されます。

論理子の物理従属セグメントが物理的に削除されます。

図 *152.* 論理子、物理対 - 仮想削除規則の例: 削除前および削除後

削除されたセグメントへのアクセシビリティー

物理的または論理的に削除されたセグメントは、特定の環境下で依然としてアクセス可能です。

次に挙げる状況のもとでは、物理的に削除されたセグメントは、依然としてアクセス可能です。

- 削除されたセグメントの物理従属セグメントが、その論理子からアクセス可能な論理親である。
- 削除されたセグメントの物理従属セグメントが、その論理親からアクセス可能な論理子である。
- 削除されたセグメントの物理親が、その論理親からアクセス可能な論理子である。 この場合、削除されたセグメントは、両方向論理関係での可変交差データです。

論理的に削除された論理子は、その論理親からはアクセスできません。

物理的な削除にしても論理的な削除にしても、いずれもあるセグメントをその物理子または論理子からアクセスすることを妨げることはありません。論理関係によって物理構造の否定が行われるので、セグメントを物理的または論理的に削除したり、あるいはあるセグメントを物理的かつ論理的に削除することができ、しかもアクティブな状態にある論理関係により従属セグメントからそのセグメントにアクセスすることが可能です。物理的に削除されたもルート・セグメントは、論理 DBD

において従属セグメントとして定義されている場合には、アクセスすることができます。論理 DBD は、物理 DBD に相対するものを定義します。以下の図は、削除されたセグメントへのアクセシビリティーを示しています。

削除されたセグメントの物理従属セグメントが論理親であり、しかもその論理親が物理的に削除されていない論理子を持つ場合、その論理親とその物理親は、その論理子からアクセス可能です。

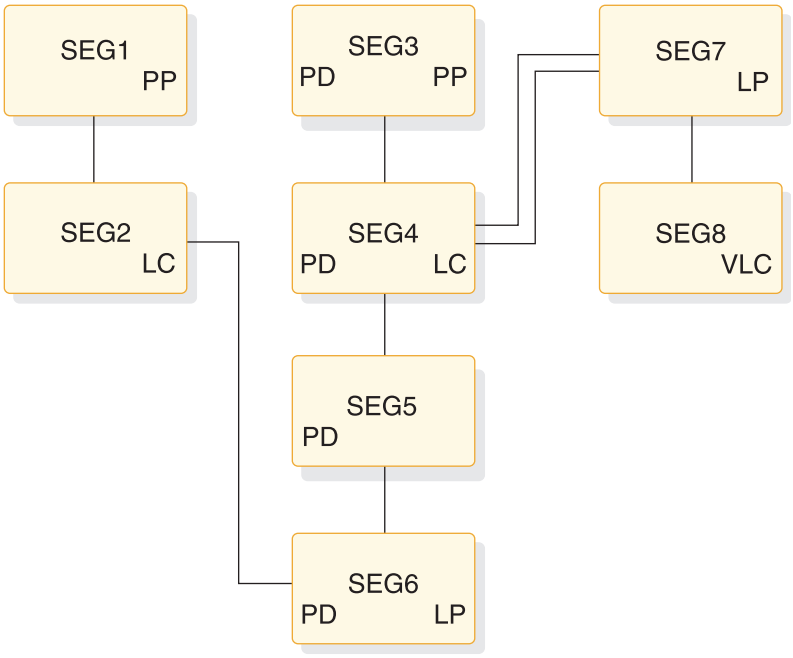


図 *153. (1/5).* 削除されたセグメントのアクセシビリティーの例

前の図の物理構造は、SEG3、SEG4、SEG5、および SEG6 が物理的に削除されていることを示します。おそらくこれは SEG3 を対象とした DLET 呼び出しを出したためです。その結果、SEG3 の従属セグメントがすべて物理的に削除されました。(SEG6 の削除規則は P ではありません。P であれば 'DX' という状況コードが出されます。)

SEG3、SEG4、SEG5、および SEG6 は、依然として SEG6 の論理子である SEG2 からアクセス可能です。 SEG2 は物理的に削除されていないからです。しかし、SEG6 の物理従属セグメントにはアクセスは可能でなく、アクティブな論理関係によってその解放が禁止されていない限り、その DASD スペースは解放されます。

削除されたセグメントの物理従属セグメントが論理子であり、しかもその論理子の論理親が物理的に削除されていない場合には、その論理子、その物理親、およびその物理従属セグメントは、その論理親からアクセス可能です。

論理子セグメント SEG4 は、依然としてその論理親 SEG7 からアクセス可能です (SEG7 は物理的に削除されてはいません)。可変交差データ SEG5 と SEG6 もアクセス可能です。論理子の物理親 (SEG3) も論理子 (SEG4) からアクセス可能です。

物理的かつ論理的に削除された論理子は、その物理従属セグメントからアクセスすることができます。以下の図を参照してください。

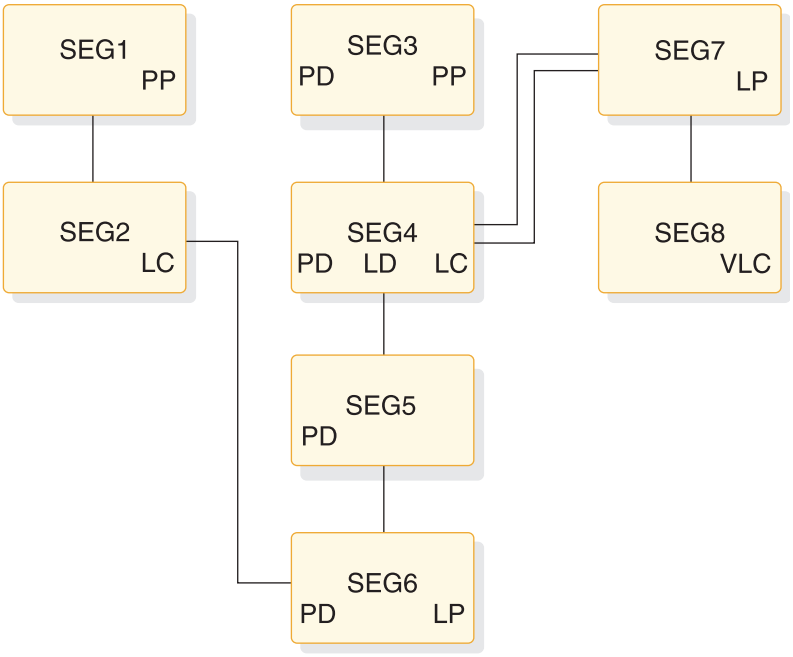


図 *154. (2/5).* 削除されたセグメントのアクセシビリティーの例

前の図の物理構造は、論理子 SEG4 が物理的にも論理的にも削除されていることを示しています。

先の例 (4 の 1) から見て、SEG6 (論理親) は、SEG2 (その論理子) のセグメントが物理的に削除されていなければ、SEG2 からアクセスできることが分かります。また、いったん SEG6 にアクセスすると、その物理親 (SEG5、SEG4、SEG3) にアクセスできることも知っています。この論理子が論理的に削除されていても問題はありません (これがこの例と (4 の 1) との唯一の相違点です)。

パスのための削除ビットは存在しないので、3 番目のパスは阻止できません。したがって、論理子 SEG4 が物理的かつ論理的に削除されていても、SEG4 にはその従属セグメントからアクセスできます。

3 番目のパスからアクセスされるセグメントが削除されるときに、そのセグメントはその物理データベースで物理的に削除されますが、そのセグメントは依然としてその 3 番目のパスからアクセスすることができます。以下の図とコードを参照してください。

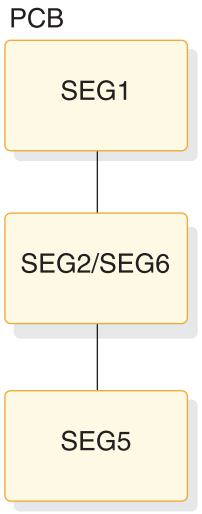


図 *155. (3/5).* 削除されたセグメントのアクセシビリティーの例

```
GHU 'SEG5'  STATUS='  '
DLET        STATUS='  '
```

図 *156. (4/5).* 削除されたセグメントのアクセシビリティーの例: データベース呼び出し

SEG5 が DLET 呼び出しにより物理的に削除され、SEG 6 が波及により物理的に削除されます。 SEG2/SEG6 は単一方向ポインターを持っているため、SEG2 は DLET 呼び出しが出される前に論理的に削除されたと見なされます。 LD ビットがオンに設定されたものと見なされるだけです。以下の図を参照してください。

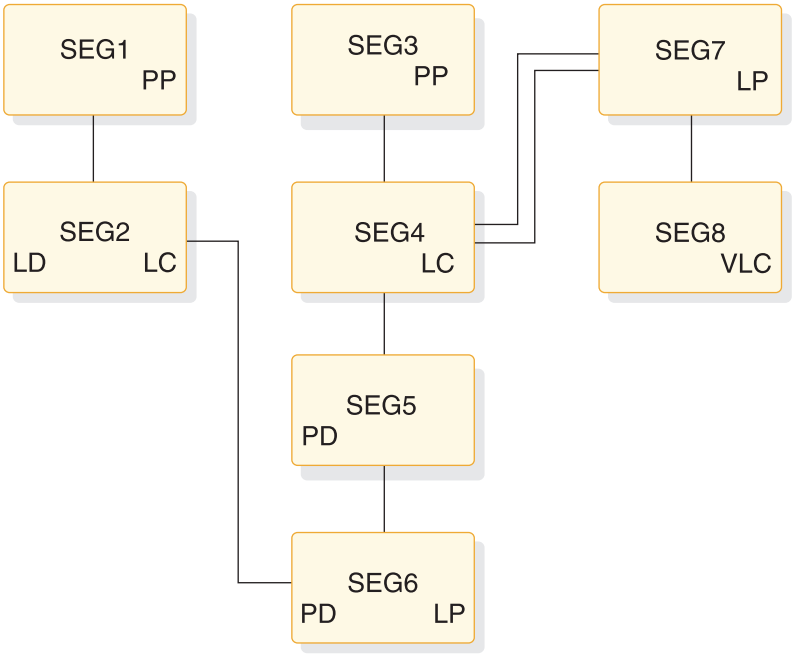


図 *157. (5/5).* 削除されたセグメントのアクセシビリティーの例

結果は興味深いものです。 SEG4 がその論理親 SEG7 からアクセスされない限り、SEG5 をその物理親のパス (SEG4) からアクセスできません (SEG5 と SEG6 は可

変交差データとしてアクセスできます)。 SEG6 はまだその論理子からアクセスできるので、 SEG5 はまだその 3 番目のパス (SEG6) からアクセスすることができます。したがって、あるアプリケーション・プログラムがあるセグメントを物理的に削除しても、なおそのセグメントは、そのセグメントを削除するのに用いたのと同じ PCB を用いて、そのアプリケーション・プログラムからアクセスすることができます。

異常終了の可能性

論理親が物理的かつ論理的に削除されると、その DASD スペースが解放されます。これが行われるためには、その論理子がすべて物理的かつ論理的に削除されていなければなりません。しかし、それらの論理子が物理従属セグメントを持ち、しかもその物理従属セグメントがアクティブな論理関係を持つ場合には、それらの論理子の DASD スペースは解放できません。

次の条件に該当する場合には、この種の論理子 (論理子と論理親の両方が物理的かつ論理的に削除されている) にその物理従属セグメントからアクセスすると、850 から 859 までの異常終了が起きることがあります。

- LPCK がこの論理子に保管されていない。
- 連結定義が、論理親に対してデータ・センシティブである。

以下の図は、異常終了に関する例を示しています。

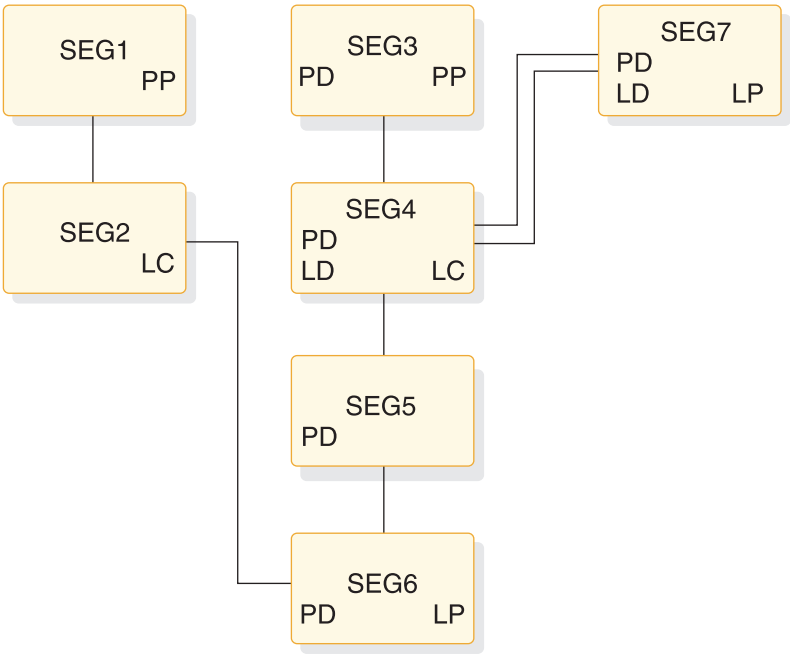


図 *158.* 異常終了の例

論理親 SEG7 は、物理的かつ論理的に削除されています (LD ビットは実際には設定されず、設定されたと見なされるだけです。図に示すためだけにこれが示されています。) この論理親の論理子もすべて、物理的かつ論理的に削除されています。しかし、論理親のセグメント・スペースは解放されていますが、論理子 (SEG4) が

まだ存在します。論理子が、そのスペースを解放することを妨げるアクティブな論理関係をもっている物理従属セグメントをもっているので、その論理子はまだ存在しています。

アプリケーション・プログラムが SEG4 にその従属セグメントからアクセスする場合 (SEG1 から SEG2/SEG6 から SEG5)、該当のキーが論理子に保管されていない場合、IMS は論理親の連結キーを作成しなければなりません。 IMS が論理親 SEG7 にアクセスしようとすると、異常終了が起きます。ポインターをたどっても期待されるセグメントに移動しない場合は、850 から 859 までの異常終了コードが出されます。

関連概念:

351 ページの『3 番目のアクセス・パス』

異常終了の回避

(3 番目のパスを用いて) 物理構造の下方からアクセスできる、物理的に削除された論理子を作成することは避けなければなりません。ある論理子のいずれかの物理従属セグメントが論理パスを通じてアクセス可能であるならば、この論理子は下からアクセスすることができます。

このような状況を避けるために、次の 2 つの方法があります。

- 方法 1 最初の方法は、この論理子を物理的に削除する前に、従属セグメントへの論理パスを絶つことです。物理的な削除がこのデータベースの中に波及しない限り、従属セグメントに対して P 規則を使用することにより、方法 1 による論理パスの中断が行われます。したがって、この論理子あるいはそれより上では、論理子に対して V 規則を使用することはできません。なぜなら、V 規則のもとでは、論理的な削除の波及によって、P 規則違反が指摘されることなく、物理的な削除が行われるからです。 PD ビットがすでにオンに設定されているならば、L 規則によっても波及が引き起こされますが、従属セグメントに対して P 規則が指定されていれば、このケースは防止されます。同様にして、この論理子より上のどの論理親に対しても V 規則を使用することはできません。なぜなら、論理的な削除の状態が起きると、物理的な削除が引き起こされるからです。
- 方法 2 2 番目の方法は、この論理子を物理的に削除するたびに、論理パスを絶つことです。この方法での論理パスの中断は、V 削除規則を用いている従属論理子のセグメントに行われます。従属論理親セグメントは、V 規則が指定されている両方向論理子を持つ必要があり (この論理子に達することができなければならない) あるいは、V 規則が指定されている物理的に対になった論理子を持つ必要があります。単一方向論理子によって指されている従属論理親に対しては、この方法は役に立ちません。

関連概念:

『物理削除規則違反の検出』

物理削除規則違反の検出

DLET 呼び出しが出されると、削除ルーチンは、削除すべきセグメントが入っている物理構造をスキャンします。

削除ルーチンの物理構造のスキャンの目的は、この中に物理削除規則を使用しているセグメントがあるかどうかを決定し、もしあれば、この規則に違反しているかどうかを判別することです。以下の図とコード・サンプルは、物理削除規則の違反の

例を示しています。

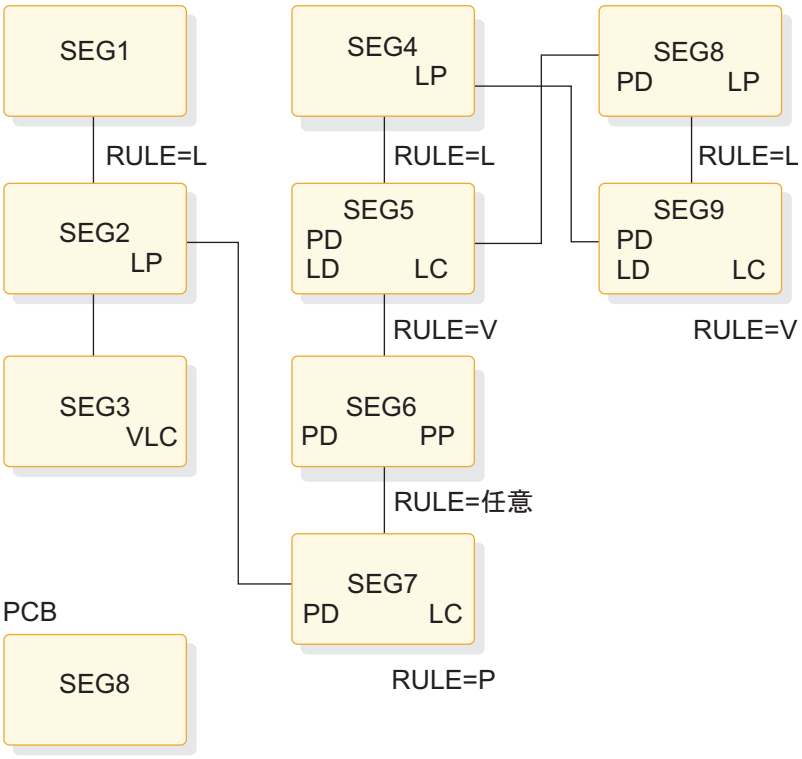


図 *159.* 物理削除規則の違反の例

```
GHU 'SEG4'  STATUS='  '
DLET        STATUS='DX'
```

図 *160.* 物理削除規則の違反の例*:* データベース呼び出し

SEG7 (SEG2 の論理子) は物理削除規則を使用しており、まだ SEG7 は論理的に削除されていません (LD ビットがオンに設定されていません)。したがって、物理削除規則に違反しています。 'DX' という状況コードがアプリケーション・プログラムに戻され、どのセグメントも削除されません。

関連タスク:

345 ページの『異常終了の回避』

論理削除規則としての物理削除規則の取り扱い

削除ルーチンが、DLET 呼び出しが指定されているセグメントも、この物理構造におけるこのセグメントのどの物理従属セグメントも、物理削除規則を使用していないと判断すると、これ以後出会った物理削除規則はすべて論理削除規則として取り扱われます (論理的な削除が、他のデータベースの中の論理子または論理親に波及し、これによって、物理的な削除が引き起こされます。V 削除規則)。

以下の図とコードは、物理削除規則を論理削除規則として取り扱う例を示しています。

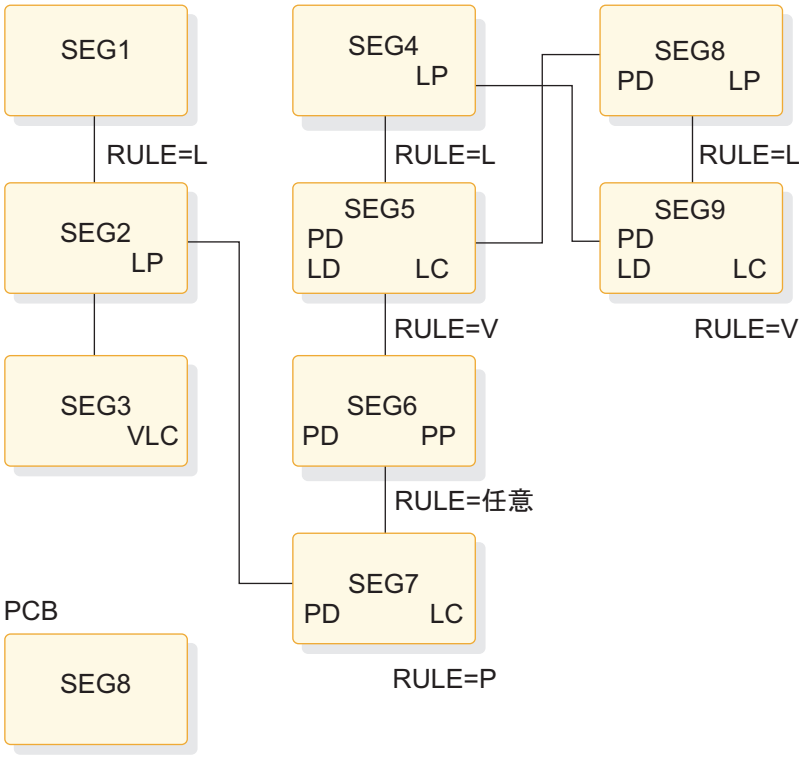


図 *161.* 論理削除規則としての物理削除規則の取り扱いの例

```
GHU 'SEG8'  STATUS='  '
DLET        STATUS='  '
```

図 *162.* 論理削除規則としての物理削除規則の取り扱いの例*:* データベース呼び出し

SEG8 と SEG9 は共に物理的に削除されており、SEG9 は論理的に削除されています (V 規則)。 SEG5 は物理的かつ論理的に削除されます。これは SEG9 の物理対であるからです (物理対においては、一方の LD ビットが設定されると、他方の PID ビットが設定され、この逆も成り立ちます)。 SEG5 を物理的に削除すると、物理的な削除が SEG5 の物理従属セグメントに波及していきます。したがって、SEG6 と SEG7 が物理的に削除されます。

物理的な削除が、SEG4 に対して DLET 呼び出しを出すことにより開始したときは、SEG7 の物理的な削除は防止されるという点に注意してください。しかし、この図のような場合には、SEG7 の物理規則は論理規則として取り扱われます。

物理的および論理的に削除されたセグメントの挿入

セグメントが挿入されたときに、一定の条件が満たされていれば、置き換え操作が行われ (つまりスペースが再使用され)、挿入されたセグメントの既存の従属セグメントが残ります。

次の条件が満たされていれば、セグメントが挿入されたときに置き換え操作が行われ (つまりスペースが再使用され)、セグメントの既存の従属セグメントが残ります。

- 挿入されるセグメントがすでに存在する (物理的順序でも論理的順序でも、セグメント・タイプおよびキー・フィールド値が同じである)

- 挿入のパスに沿って、このセグメントの削除ビットがオンに設定されている

HDAM データベースと HIDAM データベースに関しては、必要に応じて論理兄弟チェーンが設定され、挿入されたセグメントの既存の従属セグメントが残ります。

HISAM データベースに関しては、挿入が行われる前に、ルート・セグメントが物理的かつ論理的に削除されていれば、1 次データ・セット・グループと 2 次データ・セット・グループでのこのルート・セグメントの最初の論理レコードが再使用されます。 OSAM チェーンの残りの論理レコードは除去されます。

削除規則の要約

以下のリストは、削除規則についてまとめたものです。

DLET 呼び出し

連結セグメント (SOURCE=DATA/DATA、DATA/KEY、KEY/DATA) に対して出された DLET 呼び出しは、論理子のみに対する DLET 呼び出しです。

論理親からアクセスした論理子を対象とする DLET 呼び出しは、この論理子を論理的に削除するようにという要求です。

これら以外のすべての場合、あるセグメントを対象として出された DLET 呼び出しは、このセグメントを物理的に削除するための要求です。

物理的な削除

物理的に削除されたセグメントは、その物理パスからはアクセスすることができませんが、1 つだけ例外があります。すなわち、物理的に削除されたセグメントの物理親の 1 つが、その論理親からアクセスすることのできる論理子の場合には、物理的に削除されたセグメントは、この論理子からアクセス可能です。論理子の物理従属セグメントが可変交差データであるからです。

論理的な削除

定義上、論理的に削除された論理子に、その論理親からアクセスすることはできません。単一方向論理子セグメントは、論理的に削除されたものと見なされます。

定義上、ある論理親のすべての論理子が物理的に削除され、しかも、物理対のセットに属するそのすべての物理子が物理的に削除されると、この論理親は論理的に削除されたものと見なされます。

アクセス・パス

セグメントを物理的に削除しても論理的に削除しても、このセグメントにその物理子または論理子からアクセスすることは妨げられず、このセグメントからその物理親または論理親にアクセスすることも妨げられません。物理的に削除されたルート・セグメントには、その物理子または論理子からのみアクセスすることができます。

削除の波及

両方向物理対においては、1 対の論理子の一方を物理的に削除すれば、これと対をなすセグメントが論理的に削除されます。同様に、一方を論理的に削除すれば、他方が物理的に削除されます。

セグメントの物理的な削除によって、論理的な削除の要求がその両方向論理子に波及していきます。また、物理的な削除の要求がその物理子、およびこのセグメントをソース・セグメントとする任意の索引ポインター・セグメントにも波及していきます。

削除規則

さらに、削除操作は次の削除規則に左右されます。

論理親

RULES=P を指定している場合、このセグメントがまだ論理的に削除されていないとき、このセグメントまたはその物理親を対象とする DLET 呼び出しを出すと、DX 状況コードとなります。どのセグメントも削除されません。論理関係にまたがる波及の結果として、このセグメントにある要求が出された場合には、P 規則は L 規則と同様の働きをします。

RULES=L を指定している場合には、物理的、論理的のいずれの削除を最初に実施することもでき、一方の削除が他方の削除を引き起こすことはありません。

RULES=V を指定している場合には、物理的、論理的のいずれの削除を最初に実施することもできます。このセグメントが、DLET 呼び出しの結果として論理的に削除されると、このセグメントは物理的にも削除されます。

仮想対の論理子の物理親

RULES=P、L、または V は、無意味です。

RULES=B が指定されており、仮想対の論理子であるすべての物理子が論理的に削除されると、その物理親セグメントは物理的に削除されます。

論理子

RULES=P が指定されている場合には、セグメントがまだ論理的に削除されていないときには、そのセグメントまたはその任意の物理親の物理削除を要求する DLET 呼び出しを出すと、DX 状況コードが出されます。どのセグメントも削除されません。論理関係にまたがる波及の結果として、このセグメントに対して削除要求が出された場合、あるいはこのセグメントが物理対の 1 つである場合には、この規則は L 規則と同様の働きをします。

RULES=L を指定している場合には、物理的、論理的のいずれの削除を最初に実施することもでき、一方の削除が他方の削除を引き起こすことはありません。

RULES=V を指定している場合には、物理的、論理的のいずれの削除を最初に実施することもでき、一方の削除を行えば他方の削除も行われます。ある物理対の一方のセグメントに対してのみこの規則が使用されている場合には、この規則は L 規則と同様の働きをします。

スペースの解放

DASD スペースが解放されたときに、このスペースを再使用できるか、または再使用できないかは、データベース編成によって異なります。次の条件が満たされると、セグメントの DASD スペースが解放されます。

- セグメントのすべての物理従属セグメント用のスペースが解放されている。
- このセグメントが物理的に削除されている。

- このセグメントが論理子または論理親である場合には、このセグメントが物理的かつ論理的に削除されている。
- このセグメントが論理子の従属セグメントであり (可変交差データ)、この論理子の物理親に DLET 呼び出しが出された場合には、この論理子が物理的かつ論理的に削除される。
- このセグメントが 1 次索引ポインター・セグメントである場合には、そのターゲット・セグメントのためのスペースが解放される。

DLET 呼び出しの使用

DLET 呼び出しは、セグメントの経路を削除するようにという要求であり、セグメントによって使用されている DASD スペースを解放する要求ではありません。

セグメントが論理関係に関与している場合には、そのセグメントは物理パスと論理パスの 2 つのパスに属しているので、削除規則が必要です。論理子とその論理親および物理親 (あるいは物理対が使用されている場合には 2 つの物理親) の削除規則の選択方法は、2 つのアクセス・パスを削除するのに DLET 呼び出しが 1 つ必要か、または 2 つ必要なのかを決定します。

物理削除と論理削除

セグメントを物理的に削除すると、それ以降、このセグメントの物理親を用いてこのセグメントにアクセスすることができなくなります。

セグメントを物理的に削除すると、その物理従属セグメントも削除されますが、これには 1 つの例外があります。すなわち、物理的に削除されたセグメントの物理親の 1 つが、その論理親からアクセスされた論理子である場合には、この論理子から、物理的に削除されたセグメントにアクセスすることができます。削除されたセグメントが論理子からアクセスできる理由は、論理子の物理従属セグメントが可変交差データだからです。

論理子を論理的に削除すると、それ以降、この論理子の論理親を用いてこの論理子にアクセスすることはできません。単一方向論理子セグメントは、論理的に削除されたものと見なされます。ある論理親のすべての論理子が物理的に削除されると、この論理親は論理的に削除されたものと見なされます。物理対の論理関係に関しては、論理子と対をなす物理子も物理的に削除しないと、その論理親が論理的に削除されたとは見なされません。

連結セグメントの削除

次のアプリケーション・プログラムは連結セグメント SOURCE=(DATA/DATA)、(DATA/KEY)、(KEY/DATA) またはその論理子に対してセンシティブになることができます。それは、あらゆる場合に、(アクセスされるパスに応じて) 物理的にあるいは論理的に削除されるのは論理子であるからです。

連結セグメントの関係を以下の図に示します。

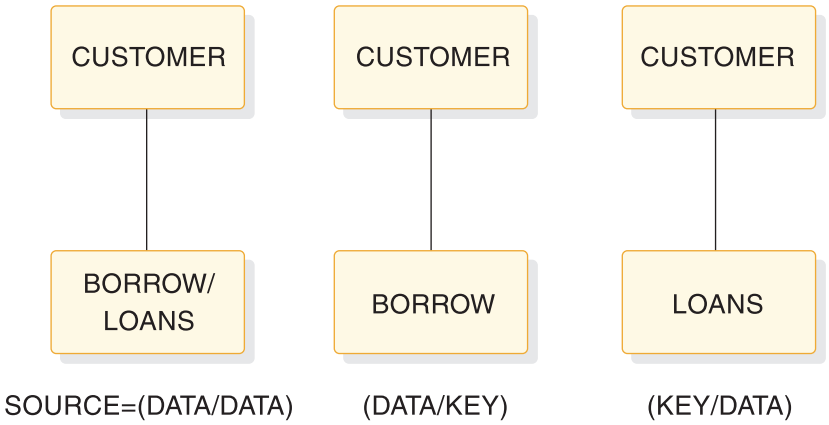


図 *163.* 連結セグメントの関係

関連資料:

IMS PROCLIB データ・セットの DFSVSMxx メンバー (システム定義)

3 番目のアクセス・パス

以下の図では、論理子セグメント SEG4 に関しては、次の 3 つのパスがあります。

論理子セグメント SEG4 への 3 つのパスは次のとおりです。

- 物理親 SEG3 からの物理パス
- 論理親 SEG7 からの論理パス
- SEG4 の物理従属セグメント (SEG5 および SEG6) からの 3 番目のパス (セグメント SEG6 は論理子 SEG2 からアクセス可能な論理親であるからです)

これらのパスは「全二重」パスと呼ばれますが、その意味は、これらのパスを通じてセグメントに 2 方向から (上および下から) アクセスできるということです。これらのパスに沿ったアクセスを制御するための削除ビットが 2 つありますが、これらは「半二重」であり、したがって、これらはそれぞれのパスの半分しか阻止しません。 3 番目のパスを阻止するためのビットはありません。 SEG4 が物理的および論理的に削除されたとしても (この場合、SEG4 の PD ビットと LD ビットが設定されます)、SEG4 にはまだ 3 番目のパスからアクセスでき、その両方の親も同様です。

物理的に削除しても論理的に削除しても、その物理子または論理子からセグメントにアクセスすることは阻止されません。 SEG4 を論理的に削除すると、その論理親 SEG7 から SEG4 へのアクセスが阻止されます。しかし、SEG4 から SEG7 へのアクセスは阻止されません。同様に、SEG4 を物理的に削除すると、SEG4 に、その物理親 SEG3 からアクセスすることは阻止されますが、SEG4 から SEG3 へのアクセスは阻止されません。

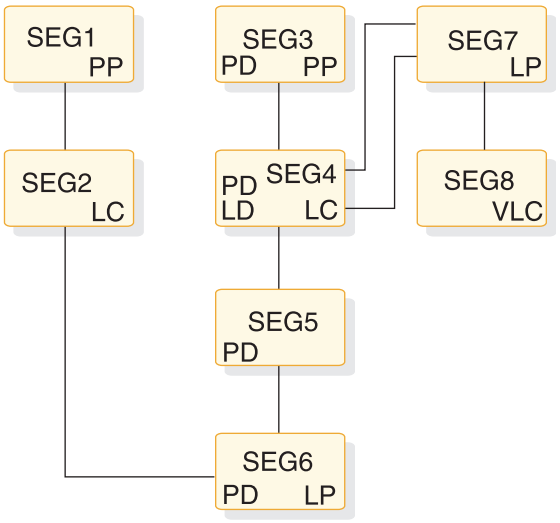


図 *164. 3* 番目のアクセス・パスの例

関連概念:

344 ページの『異常終了の可能性』

削除呼び出しの発行

DLET 呼び出しは、物理 DBD または論理 DBD において定義されているセグメントに対して出すことができます。この呼び出しは、物理セグメントまたは連結セグメントに対して出すことができます。

連結セグメントに対して出された DLET 呼び出しは、アクセスされたパスにある論理子の削除を要求します。連結セグメントまたは論理子がその論理親からアクセスされた場合には、DLET 呼び出しは論理的な削除を要求します。それ以外の場合には、すべての削除呼び出しは物理的な削除を要求します。

あるセグメントを物理的に削除すると、そのセグメントのすべての論理子の論理的な削除が要求され、さらに、このセグメントのすべての物理子の物理的な削除が要求されます。セグメントを物理的に削除すると、物理的に削除されたセグメントをソース・セグメントとする索引ポインター・セグメントも削除するようにという要求が出されます。

削除呼び出しが出されることのあるセグメントの PCB には、削除センシティビティーが指定されていなければなりません。その呼び出しには、これらのセグメントの物理従属セグメントに対する削除センシティビティーを指定しておく必要はありません。削除操作は、PCB または論理 DBD で指定されている KEY センシティビティーまたは DATA センシティビティーの影響を受けません。

状況コード

DLET 呼び出しの後にアプリケーション・プログラムに戻されることのある非ブランク状況コードは、次のとおりです。

- DX - 削除規則に違反した
- DA - 入出力域でキーが変更された

- AM - 呼び出し機能が処理オプションまたはセグメント・センシティビティーに対応していなかった

DASD スペースの解放

DLET 呼び出しは DASD スペースを解放する要求ではありません。 DASD スペースが解放されたときに、このスペースを再使用できるか、または再使用できないかは、データベース編成によって異なります。

次の条件が満たされると、セグメントの DASD スペースが解放されます。

- セグメントのすべての物理従属セグメント用のスペースが解放されている。
- セグメントが物理的に削除されている (PD ビットが設定されている、または PD ビットがオンに設定されつつある)
- セグメントが論理子または論理親である場合は、そのセグメントを物理的および論理的に削除しなければならない (PD ビットが設定されているかあるいはオンに設定されつつあり、しかも LD ビットが設定されているかあるいは設定されていると見なされている)
- このセグメントが論理子の従属セグメント (可変交差データ) であり、しかも論理子の物理親に対して DLET 呼び出しが出されている場合は、その論理子を物理的にも論理的にも削除しなければならない
- セグメントが副次索引ポインター・セグメントである場合、そのターゲット・セグメントのスペースが解放されている

セグメント削除バイト

削除バイトは、データベース内のセグメントの削除状況を維持するために、IMS によって使用されます。

削除バイト内のビットは、論理子セグメントとその論理親にとってのみ意味があります。論理関係に関与しているセグメントの場合には、PD ビットと LD ビットは、以下のように設定され、あるいは設定されたと見なされます。

- セグメントが物理的に削除されると (これによって、これ以降、このセグメントへのその物理親からのアクセスが阻止されます)、削除処理は、削除されたセグメントから下方にその従属セグメントをスキャンしていき、上方に向きを変え、それぞれのセグメントの DASD スペースを解放するか、または各セグメントの PD ビットを設定します。 HISAM はこの処理での 1 つの例外です。 HISAM は、DLET 呼び出しによって指定されたセグメントに削除ビットが設定され、処理は終了します。
- PD ビットが論理親で設定されると、この論理親からアクセスできるすべての論理子に LD ビットが設定されます。
- 物理対が使用される場合、ある 1 対の論理子の片方に PD ビットが設定されると、これと対をなすセグメントの LD ビットが設定されます。
- 仮想対の論理子が論理的に削除された場合 (これによって、これ以降、この論理子へのその論理親からのアクセスが阻止されます)、この論理子において LD ビットが設定されます。
- LD ビットは、単一方向論理関係にあるすべての論理子において設定されたものと見なされます。

- 物理対が使用される場合、ある親の物理子になっている対をなすセグメントのすべてにおいて PD ビットがオンに設定されている場合には、その親の LD ビットは設定されたものと見なされます。

削除バイトの中のビット

削除バイトの意味は、バイト内のどのビットがオンであるかによって決まります。

このトピックには診断、変更、およびチューニングに関する情報が含まれています。

削除バイトの中のビットがオンのときの意味は、次のとおりです。

ビット 削除バイトがオンのときの意味

0 セグメントは削除されるとマークされている。このビットは、HISAM データベースないし副次索引データベースの中のセグメント、あるいは 1 次索引の中のセグメントに使用されます。

1 データベース・レコードが削除されるとマークされている。このビットは、HISAM データベースないし副次索引データベースの中のセグメント、あるいは 1 次索引の中のセグメントに使用されます。

2 セグメントは削除ルーチンによって処理されている。

3 このビットは予約済みである。

4 セグメントの接頭部とデータ部がストレージにおいて分離されている。 (セグメントの分離データ部分のすぐ前の削除バイトのビットはすべてオンです。)

5 セグメントは物理パスから削除されるとマークされている。このビットは PD (物理削除) ビットと呼ばれます。

6 セグメントは論理パスから削除されるとマークされている。このビットは LD (論理削除) ビットと呼ばれます。

7 セグメントはその論理兄弟チェーンから除去されるとマークされている。(ビット 5 と 6 もオンである場合にのみ、このビットがオンに設定されるはずです。)

関連概念:

18 ページの『削除バイト』

接頭部記述子バイトの中のビット

削除バイトは DEDB のルート・セグメントにも使用されますが、この場合のみ接頭部記述子バイトと呼ばれます。

このトピックには診断、変更、およびチューニングに関する情報が含まれています。

各ビットがオンのときの意味は、次のとおりです。

ビット ルート・セグメントの接頭部記述子がオンのときの意味

0 順次従属セグメントが定義されている。

1-3 これらのビットは予約済みである。

4 から 7

定義されたセグメントの数が 8 またはそれ以下の場合に、ビット 4 から 7 には、定義された最大のセグメント・コードが入っている。そうでない場合には、ビットは 000 に設定されます。

関連概念:

310 ページの『論理関係の挿入規則、削除規則、および置き換え規則』

挿入規則、削除規則、および置き換え規則の要約

以下の図では、目的の結果を述べ、その結果を得るために使用できる規則を示すことにより、挿入規則、削除規則、および置き換え規則を要約します。

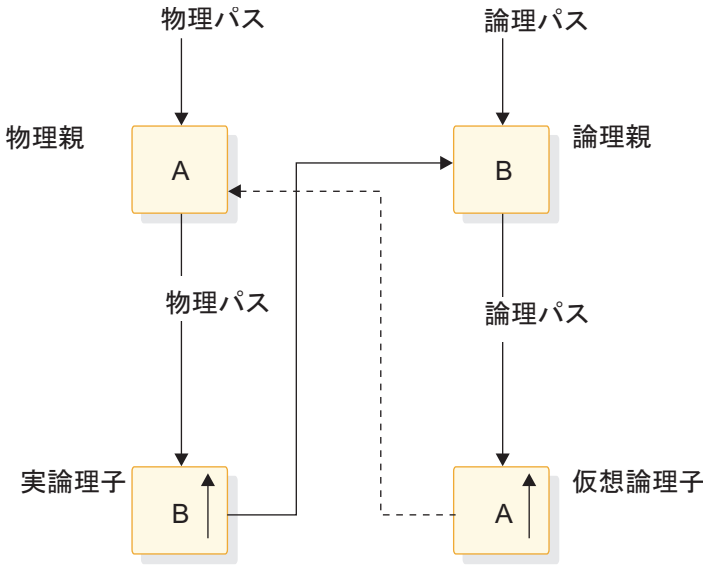


図 *165.* 挿入規則、削除規則、および置き換え規則の要約

以下の表に、挿入規則、削除規則、および置き換え規則を列挙し、その指定方法を示します。

表 *60.* 挿入規則、削除規則、および置き換え規則の指定

規則	**RULES=** 指定
物理挿入規則	RULES= (P,_,_)
論理挿入規則	RULES= (L,_,_)
仮想挿入規則	RULES= (V,_,_)
物理削除規則	RULES= (_,P,_)
論理削除規則	RULES= (_,L,_)
両方向仮想削除規則	RULES= (_,B,_)
仮想削除規則	RULES= (_,V,_)
物理置き換え規則	RULES= (_,_,P)
論理置き換え規則	RULES= (_,_,L)
仮想置き換え規則	RULES= (_,_,V)

物理親セグメント A の挿入規則

物理親 (PP) セグメント A の挿入規則は、PP A への論理パスを用いて PP A の挿入を制御します。

その規則は次のとおりです。

- PP A の論理パス上で PP A を挿入禁止にするために、物理挿入規則を使用します。
- PP A の論理パス上で (仮想論理子セグメント A と連結させて) PP A を挿入できるようにするために、論理規則または仮想規則を使用します。

  PP A がすでに存在する場合には、既存の A セグメントに対して論理接続が設定されます。既存の PP A セグメントは、置き換えることも未変更のまま残すこともできます。
  - 挿入呼び出しによって PP A を変更したくない場合には、論理挿入規則を使用します。
  - 挿入呼び出しによって PP A を置き換えたい場合には、仮想挿入規則を使用します。

物理親セグメント A の削除規則

PP セグメント A の削除規則は、PP A への論理パスを用いて PP A の削除を制御します。

その規則は次のとおりです。

- (実論理子セグメント B から PP セグメント A への) 最後の論理接続が切断された場合に、PP セグメント A が自動的に削除されるようにするには、両方向仮想削除規則を使用します。
- これ以外の削除規則は、PP A に対しては無意味です。

物理親セグメント A の置き換え規則

PP セグメント A の置き換え規則は、PP A への論理パスを用いて PP A の置き換えを制御します。

その規則は次のとおりです。

- PP A の論理パス上で PP A の置き換えを禁止するために、およびこの規則に違反して A を置き換えようとした場合に 'RX' 状況コードを受け取るために、物理置き換え規則を使用します。
- PP A の論理パス上で PP A の置き換えを無視するには、論理置き換え規則を使用します。
- PP A の論理パス上で PP A の置き換えをできるようにするには、仮想置き換え規則を使用します。

論理親セグメント B の挿入規則

論理親 (LP) セグメント B の挿入規則は、LP B への論理パスを用いて LP B の挿入を制御します。

注: 次の規則は、PP セグメント A の挿入規則と同じです。

その規則は次のとおりです。

- LP B の論理パス上で LP B を挿入禁止にするために、物理挿入規則を使用します。
- LP B の論理パス上で (仮想セグメント RLC B と連結させて) LP B を挿入できるようにするために、論理規則または仮想規則を使用します。

  LP B がすでに存在する場合には、既存の LP B セグメントに対して論理接続が設定されます。既存の LP B は、置き換えることも未変更のまま残すこともできます。
  - 挿入呼び出しによって LP B を変更したくない場合には、論理挿入規則を使用します。
  - 挿入呼び出しによって LP B を置き換えたい場合には、仮想挿入規則を使用します。

論理親セグメント B の削除規則

セグメント LP B の削除規則は、その物理パス上で LP B の削除を制御します。連結セグメントに対する削除呼び出しは、論理子のみの削除と解釈されます。

その規則は次のとおりです。

- LP B のオカレンスへの最後の論理関係パスが削除されるまでは B を確実にアクセス可能にしておきたい場合には、物理削除規則を選択します。 LP B を指す実論理子 (RLC) B のオカレンスがある間に、LP B を削除しようとした場合は、'DX' 状況コードが戻され、セグメントは削除されません。
- セグメント LP B をその物理パス上で削除できるようにしたい場合には、論理削除規則を選択します。 LP B が削除された場合、その物理パス上ではもうアクセスすることはできなくなります。 RLC B が存在する限り、RLC B を介して PP A から LP B にアクセスすることは依然として可能です。
- LP B が削除呼び出しによって明示的に削除されるか、あるいはそれを指す RLC B がすべて物理的に削除されたために暗黙のうちに削除された場合には、仮想削除規則を使用して LP B を物理的に削除します。

論理親セグメント B の置き換え規則

LP セグメント B の置き換え規則は、LP B への論理パスを用いて LP B の置き換えを制御します。

注: 次の規則は、PP セグメント A の置き換え規則と同じです。

その規則は次のとおりです。

- LP B の論理パス上で LP B の置き換えを禁止するために、およびこの規則に違反して LP B を置き換えようとした場合に 'RX' 状況コードを受け取るために、物理置き換え規則を使用します。
- LP B の論理パス上で LP B の置き換えを無視するには、論理置き換え規則を使用します。
- LP B の論理パス上で LP B の置き換えができるようにするには、仮想置き換え規則を使用します。

実論理子セグメント B の挿入規則

挿入規則は論理子には適用されません。

実論理子セグメント B の削除規則

RLC セグメント B の削除規則は、その論理パスまたは物理パスを用いた削除呼び出しに適用されます。

その規則は次のとおりです。

- RLC B がその論理パスと物理パス上で削除される順序を制御するには、物理削除規則を使用します。物理削除規則では、物理的に削除する前に論理的に削除することが必要です。違反すると、'DX' 状況コードが戻されます。
- 物理的な削除と論理的な削除のどちらでも先にできるようにするには、論理削除規則を使用します。
- 論理パスまたは物理パスのいずれかから 1 つの削除呼び出しを使用して、RLC B を論理的かつ物理的に削除するには、仮想削除規則を使用します。

実論理子セグメント B の置き換え規則

LP B の置き換え規則は、RLC B への論理パスを用いて RLC B の置き換えを制御します。

注: 次の規則は、PP セグメント A の置き換え規則と同じです。

その規則は次のとおりです。

- RLC B の論理パス上で RLC B の置き換えを禁止するために、およびこの規則に違反して RLC B を置き換えようとした場合に 'RX' 状況コードを受け取るために、物理置き換え規則を使用します。
- RLC B の論理パス上で RLC B の置き換えを無視するには、論理置き換え規則を使用します。
- RLC B の論理パス上で RLC B の置き換えができるようにするには、仮想置き換え規則を使用します。

論理関係と HALDB データベース

HALDB データベースは、非 HALDB DL/I データベースと同じように論理関係をサポートしますが、例外が 1 つあります。すなわち、HALDB データベース内の両方向論理関係は、物理対を使用してインプリメントしなければなりません。

論理関係を持つ新規パーティション・データベースをロードする場合、ロード・ステップの一部として論理子セグメントをロードすることはできません。 IMS は、データベースがロードされた後、通常の更新処理によって論理子を追加します。

HALDB データベースは、論理的に関連付けられたデータベースを再編成するとき、間接リスト・データ・セット (ILDS) を使用して論理関係ポインターを維持します。

関連概念:

764 ページの『HALDB 自己回復ポインター処理』

論理関係に関するパフォーマンスの考慮事項

論理関係をインプリメントしている際に、論理的に関連したセグメントを処理するのに必要なリソースに影響を及ぼすいくつかの選択があります。

論理親ポインター

DASD 上の論理子セグメントには、その論理親を指すポインターがあります。データベース管理者は、このポインターをどのように物理的に外部ストレージに保管するかを選択します。以下の選択肢があります。

- 直接ポインターの使用 (論理子に対して SEGM ステートメントの POINTER=LPARNT をコーディングすることによって指定)
- シンボリック・ポインターの使用 (論理子に対して SEGM ステートメントの PARENT= キーワードで PHYSICAL オペランドをコーディングすることによって指定)
- 直接ポインターとシンボリック・ポインターの両方の使用

直接ポインターの利点は次のとおりです。

- 直接ポインターは長さがわずか 4 バイトなので、通常はシンボリック・ポインターより短くなります。したがって、直接ポインターを保管するほうが、一般に DASD スペースが少なくて済みます。
- HDAM または PHDAM 論理親セグメントを除いて、直接ポインターの方が通常は、論理親セグメント (これらはルート・セグメント) に対してより高速のアクセスができます。シンボリック・ポインターでは、HIDAM データベースの索引を検索するために追加のリソースが必要です。また、論理親がルート・セグメントでない場合、シンボリック・ポインターでは DL/I がルート・セグメントから論理親まで導かなければなりません。

シンボリック・ポインターの利点は次のとおりです。

- シンボリック・ポインターは、DASD 上に論理子セグメントの一部として保管されています。 DASD 上に記号キーを保管しておくと、ユーザーの入出力域において論理子セグメントをフォーマット設定するのに必要なリソースを節約できます。覚えておく必要があるのは、記号キーは常時論理子の一部として入出力域に現れることです。論理子を検索するときに、記号キーが DASD 上に保管されていない場合は、IMS が記号キーを作成しなければなりません。
- 論理子データベースを再編成するまでもなく、論理親データベースを再編成することができます。これは、単一方向の物理対関係と両方向の物理対関係に適用されます (シンボリック・ポインターが使用されている場合)。

次のような場合には、シンボリック・ポインターを使用する必要があります。

- HISAM 論理親データベースを指している場合
- 記号キーの任意の部分に基づいて論理子セグメント (仮想論理的な複数の子以外) を順序付けたい場合

KEY/DATA の考慮事項

1 つの論理 DBD の一部として連結セグメントを組み込む場合には、ユーザーの入出力域に連結セグメントがどのように現れるかを制御します。これを行うには、連結セグメントに対する SEGM ステートメントの SOURCE= キーワード上に KEY または DATA を指定します。 連結セグメントは論理子で構成され、その後に論理 (または目標) 親ができます。両方の部分に対して、KEY または DATA を指定します。例えば、連結セグメントにアクセスし、ユーザーの入出力域の中で次のセグメントを見ることを要求することができます。

- 論理子の部分のみ
- 論理 (または目標) 親の部分のみ
- 両方の部分

KEY または DATA を注意深く選択することによって、より少ない処理と入出力リソースで連結セグメントを取り出すことができます。例えば、以下のとおりです。

- 以下の図の単一方向論理関係があると仮定します。

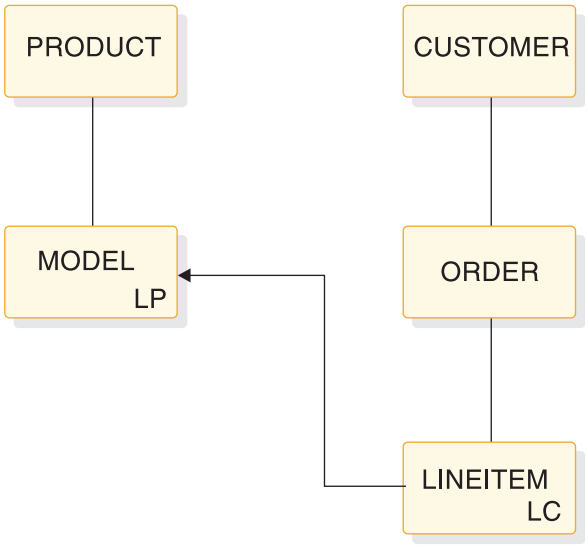


図 *166.* 単一方向論理関係の例

- また、以下の図の論理構造があると仮定します。

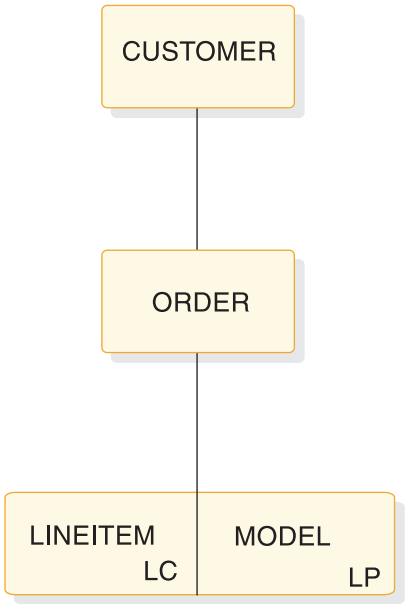


図 *167.* 論理構造の例

- 最後に、連結セグメントの LINEITEM の部分のデータのみを見る必要があると仮定します。

次のようにすると、連結セグメントの MODEL の部分にアクセスするための余分な処理と入出力が必要なくなります。

- 連結セグメントの SEGM ステートメントの SOURCE キーワードを次のようにコーディングします。

  ```
  SOURCE=(lcsegname,DATA,lcdbname),(lpsegname,KEY,lpdbname)
  ```

- LINEITEM 内に記号論理親ポインターを保管します。シンボリック・ポインターを保管しないと、DL/I はシンボリック・ポインターを作成するために MODEL と PRODUCT をアクセスしなければなりません。

要約すると、連結セグメントの論理子の部分と論理親の部分に対して、自動的に DATA センシティビティーを選択してはなりません。論理親の部分を見る必要がない場合には、論理親には KEY センシティビティーをコーディングして、DASD 上に記号論理親ポインターを保管します。

論理兄弟チェーンの順序付け

仮想対を用いて、物理兄弟チェーン上に実論理子を、また論理兄弟チェーン上に仮想論理子を順序付けることができます。できれば、論理兄弟の順序付けが必要になる操作は避けてください。論理兄弟チェーンに従って進む場合には、DL/I では通常、複数のデータベース・レコードにアクセスしなければなりません。複数のデータベース・レコードをアクセスすることは、呼び出しの処理に必要なリソースを増加させます。

呼び出しの処理に必要なリソースが増加する問題は、記号論理親ポインターのすべてまたは一部分によって論理兄弟のチェーンを順序付ける際に特に深刻です。仮想論理子が保管されていないので、仮想論理子が順序付け操作を満たしているかどうかを判別するために、記号論理親ポインターを作成する必要があります。 DL/I は、物理親ポインターに沿って、シンボリック・ポインターを作成しなければなりません。この処理は、順序付け操作で正しい位置が見つかるまでに、論理兄弟のチェーンの中の各仮想論理子に対して行われます。

仮想対関係における実論理子の位置

仮想対関係において実論理子を配置するための考慮事項があります。

- ユーザーが論理的に関連するデータベースの 1 つだけに論理子を順序付ける必要がある場合には、実論理子を該当のデータベースに入れてください。
- ユーザーが論理的に関連する両方のデータベースの中に論理子を順序付ける必要がある場合には、実論理子が最も頻繁に取り出されるデータベースにそれを入れてください。
- 実論理子は、論理兄弟のチェーンがなるべく短くなるように入れてください。このように配置することによって、論理兄弟のチェーンに従って進むために調べなければならないデータベースのレコード数が減ります。

注**:** HISAM データベースに実論理子を保管することはできませんが、これは HISAM データベースが、論理子ポインター (これは直接ポインター) を持つことができないからです。

第 16 章 副次索引の作成

副次索引とは、あるセグメント・タイプをそのセグメントのキーで定義されている順序とは異なる順序で処理する索引のことです。また、副次索引は、あるセグメント・タイプを従属セグメントの中の制限に基づいて処理することもできます。

以下のデータベース・タイプが副次索引をサポートします。

- HISAM
- HDAM
- PHDAM
- HIDAM
- PHIDAM
- DEDB

関連概念:

115 ページの『第 11 章 IMS カタログの副次索引』

関連タスク:

835 ページの『IMS 索引の変更』

838 ページの『索引のドロップ』

関連資料:

98 ページの『XDFLD セグメント・タイプのフォーマット』

副次索引の目的

副次索引を使用すると、各種のアプリケーションのさまざまな処理要件を満たすことができます。副次索引を使用すると、ルート・セグメントのキー・フィールドだけでなく、データベース内の任意のフィールドに基づいた索引を設けることができます。

データベース・レコードを設計するときには、多くのアプリケーションの処理要件を満たすような設計を行います。ある 1 つのデータベース・レコードの中にどのようなセグメントを入れるか決め、また 1 つのセグメントの中にはどのようなフィールドを設けるか決めます。 1 つのデータベース・レコードの中のセグメント相互の順序を決め、 1 つのセグメントの中のフィールド相互の順序を決めます。さらに、ルート・セグメントの中のどのフィールドをキー・フィールドにするか決め、キー・フィールドを固有なものにするかどうか決めます。これらの決定はすべて、アプリケーションのすべての処理要件にとって、何が最も望ましいかによって決められます。しかし、このようにして選択しても、その選択が他のアプリケーションに比べてある種のアプリケーションの処理要件によりよく合致することがあります。

例: 教育用データベースのデータベース・レコードを以下の図に示します。

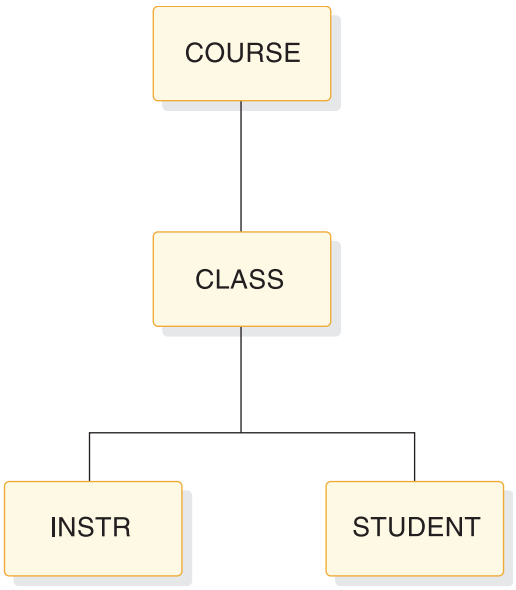


図 *168.* 教育用データベースのデータベース・レコード

以下の図は、ルート・セグメント COURSE およびそれに含まれているフィールドを示しています。講習番号フィールドは固有キー・フィールドです。

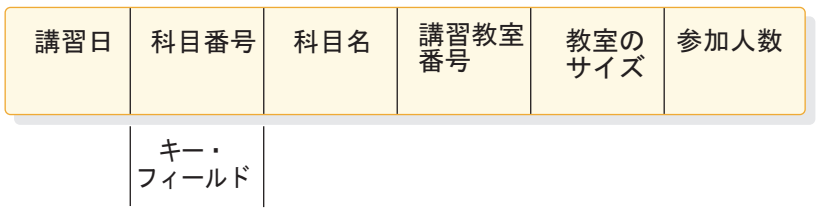


図 *169.* データベース・レコードの固有キー・フィールドの例

COURSE をルート・セグメントとし、科目番号を固有キー・フィールドとした理由の 1 つは、大部分のアプリケーションが科目番号に基づいて情報を要求するからです。このようなアプリケーションにとって、このデータベース・レコードから要求される情報へのアクセスは迅速です。しかしいくつかのアプリケーションにとっては、このデータベース・レコードの編成では、このような高速アクセスは期待できません。例えば、あるアプリケーションでは受講者名によってこのデータベースにアクセスし、受講者がとっている科目の一覧表を入手しようとします。データベース・レコードの現在の編成順序では、ある 1 人の受講者がとっている科目にアクセスするためには、このデータベース全体の順次スキャンを行わなければなりません。各データベース・レコードで、STUDENT セグメントのオカレンスを調べなければなりません。特定の受講者のデータベース・レコードが見つかったら、次に COURSE セグメントを参照して、この受講者がとっている科目の名前を入手しなければなりません。この種のアクセスは比較的低速です。このような場合には、各受講者のすべての COURSE セグメントを指すポインター・セグメントを各受講者ごとに 1 組ずつ持つ副次索引を使用することができます。

あるいは、科目名によって COURSE セグメントへアクセスしようとする別のアプリケーションがあるかもしれません。この場合には、(キー・フィールドである科目

番号の順ではなく) 科目名の順にこのデータベースにアクセスできるようにするための副次索引を使用することができます。

副次索引の特性

副次索引は、HISAM、HDAM、PHDAM、HIDAM、DEDB、および PHIDAM データベースで使用できます。

副次索引は、その索引独自の別個のデータベースに保管され、アクセス方式として VSAM を使用する必要があります。副次索引は、その索引独自のデータベースに保管されるので、別個のデータベースとして処理できます。

全機能データベースの副次索引は、アプリケーション・プログラムからは見えません。アプリケーション・プログラムが副次索引を使用して全機能データベースにアクセスする必要がある場合は、PCB の中に PROCSEQ= パラメーターをコーディングして、そのことを IMS に伝えます。アプリケーション・プログラムが通常の処理シーケンスを使用して処理を実行する必要がある場合は、PROCSEQ= をコーディングしません。アプリケーション・プログラムが通常の処理シーケンスと副次索引の両方を使用して処理を行う必要がある場合は、アプリケーション・プログラムの PSB に 2 つの PCB (1 つは PROCSEQ= がコーディングされているもの、他の 1 つはそれがコーディングされていないもの) が含まれている必要があります。

高速機能データベースの副次索引も、アプリケーション・プログラムからは見えません。DEDB データベースに高速機能副次索引を使用してアクセスする必要があるときは、PCB の中で PROCSEQD= パラメーターを使用して、1 次 DEDB データベースにアクセスするために使用する高速機能副次索引データベースの名前を指定します。PROCSEQD= パラメーターは、全機能 PROCSEQ= パラメーターと同じ機能を備えています。PROCSEQD= パラメーターは、DEDB データベース用の PROCSEQ を表しています。

2 つの PCB を使用する場合には、ある 1 つのアプリケーション・プログラムが使用することのできるデータベースの中へのパスが 2 つあり、シーケンス・フィールドも 2 つあります。 1 つのパスとシーケンス・フィールドは通常の処理シーケンスによって提供されるもので、もう 1 つは副次索引によって提供されるものです。副次索引によって、ある 1 つのアプリケーション・プログラムに、データベースに入るための代替の方法が提供され、データベース・レコードを順次に処理するための別の方法も提供されます。

別個のデータベースとして高速機能副次索引データベースにアクセスするための高速機能副次索引 PCB だけが PSB に含まれている場合は、関連する DEDB PCB を PSB に含める必要があります。DEDB PCB の最小要件として、関連する DEDB データベースのルート・セグメントに対する SENSEG ステートメントが必要です。

全機能副次索引の最後の特性は、1 つのセグメント・タイプに 32 個の副次索引を使用でき、1 つの全機能データベースに合計 1000 個の副次索引を使用できることです。

高速機能副次索引データベースは、HISAM データベースか SHISAM データベースとすることができます。

高速機能副次索引は、全機能副次索引にはない以下の機能を備えています。

ユーザー・データ区分化

単一の高速機能副次索引を複数の物理データベースにまたがって配置し、それぞれのデータベースを区画と見なすことができます。それぞれの区画には、1 つのキー範囲が含まれます。索引キーは、ユーザー区画選択出口ルーチンによって区画に割り当てられます。索引データベースには、個別にアクセスするか、1 つの別個の論理データベースとしてアクセスすることができます。

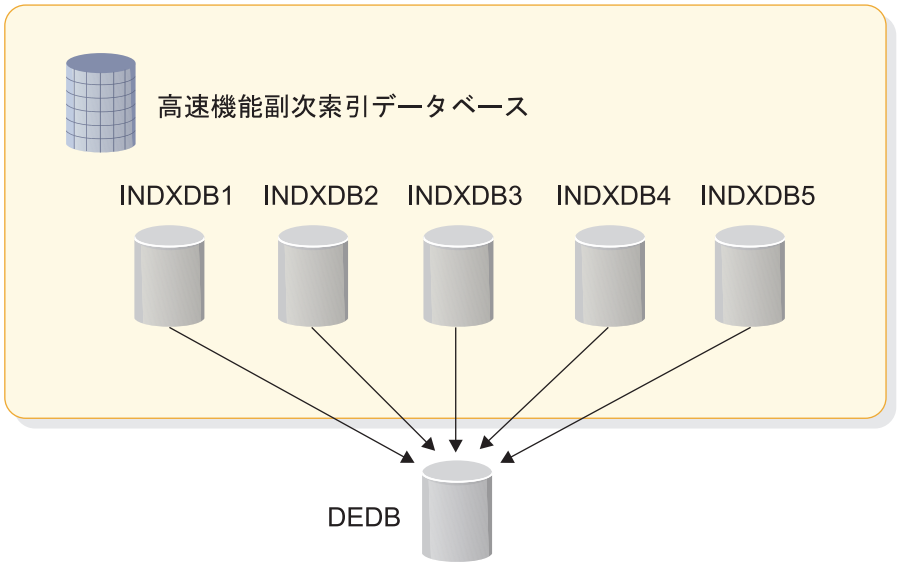
高速機能副次索引の区画は、DBD 生成ユーティリティーを使用して作成されます。高速機能 HISAM 副次索引データベースまたは高速機能 SHISAM 副次索引データベースのユーザー区分化を要求する場合は、以下のようになります。

- PROCSEQD= パラメーターはユーザー区画グループ内の最初の区画データベースの名前を指定します。
- PSELRTN= パラメーターは、実際の区画を決定するために使用される DEDB 区画選択出口ルーチンを指定します。IMS に添付されるサンプルは、高速処理データベース区画選択出口ルーチン (DBFPSE00) です。

DEDB 区画選択出口ルーチンが呼び出されない場合は、PSELOPT=MULT|SNGL パラメーターを使用して、アプリケーションにデータベースの終わりを示す GB 状況コードを返す前に処理される区画データベースの数を示すことができます。PSELOPT= は、XDFLD ステートメントで指定することも、PROCSEQD= ステートメントを伴う PCB で指定することもできます。両方で指定した場合は、PCB ステートメントでの指定が優先されます。

それぞれの高速機能副次索引データベースは、最大 101 個のユーザー区画データベースを持つことができます。1 次 DEDB データベースで、LCHILD ステートメントの NAME= パラメーターの副次索引セグメント名の後ろに、複数の区画データベースを (コンマで区切って括弧で囲んで) 指定できます。

ユーザー・データ区分化は、複数の副次索引セグメントと一緒に使用できます。 次の図は、高速機能副次索引の区画化の概念を示しています。



複数の副次索引セグメント
: 同じソース・セグメントにあるさまざまなフィールドから、複数の索引項目を作成できます。

  これを行うには、ターゲット・セグメントの SEGM ステートメントの下に複数の LCHILD/XDFLD ステートメント・ペアを定義し、LCHILD ステートメントに MULTISEG=YES を指定します。

高速機能副次索引の最後の特性は、1 つのセグメントに最大 32 個の副次索引を使用でき、1 つの DEDB に最大 255 個の副次索引を使用できることです。複数の LCHILD/XDFLD セグメント・ペアのそれぞれが、1 セグメント当たり 32 個という副次索引の制限に対してカウントされます。1 つの高速機能副次索引が複数の区画データベースから構成されている場合、高速機能副次索引データベース自体 (区画でなく) が 1 つの DEDB 当たり 255 個という副次索引の制限に対してカウントされます。

関連資料:

データベース記述 (DBD) 生成ユーティリティー (システム・ユーティリティー)

副次索引のために使用されるセグメント

副次索引をセットアップするには、IMS に対して、ポインター、ターゲット、およびソース・セグメントの 3 種類のセグメントを定義する必要があります。

以下の図は、副次索引に使用されるセグメントを示しています。

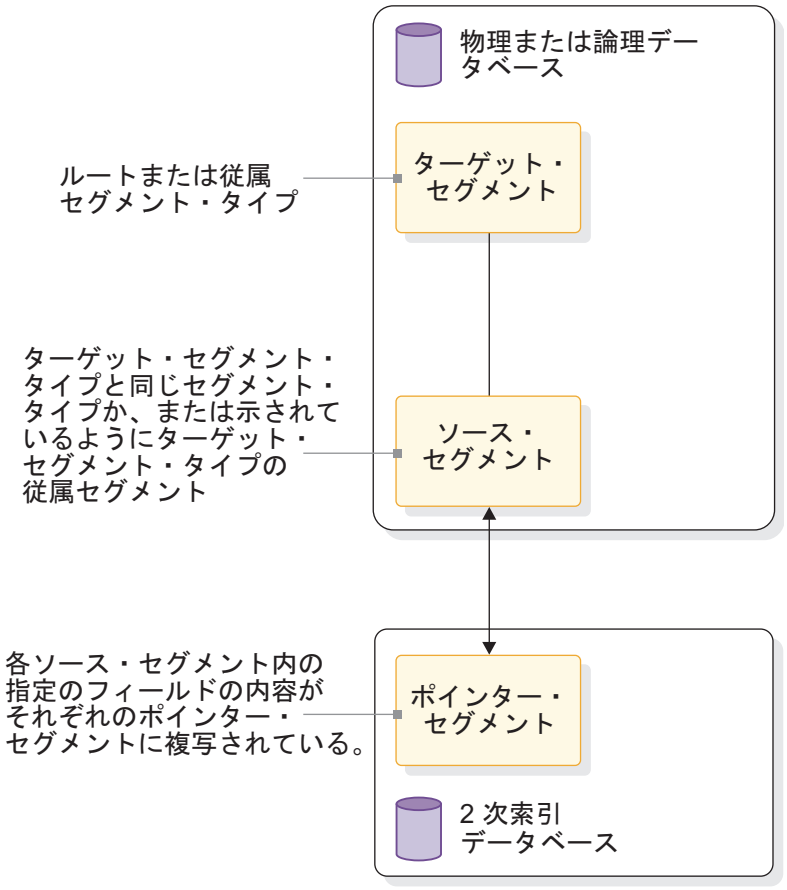


図 *170.* 副次索引のために使用されるセグメント

ポインター・セグメント

このポインター・セグメントは副次索引データベースの中に入っており、これが副次索引データベースの中の唯一のセグメント・タイプです。このセグメントのフォーマットを、以下の図に示します。

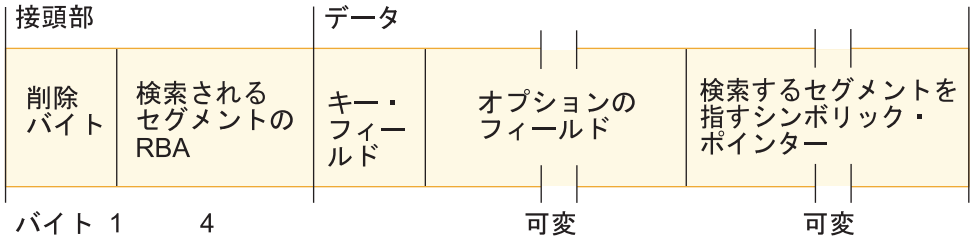


図 *171.* 副次索引データベースに入っているポインター・セグメントのフォーマット

接頭部の中の最初のフィールドは削除バイトです。 2 番目のフィールドは、アプリケーション・プログラムが通常のデータベースから取り出そうとしているセグメントのアドレスです。副次索引でシンボリック・ポインターが使用される場合には、このフィールドはありません。シンボリック・ポインターは、セグメントの連結キーを使用してセグメントを指します。 HIDAM と HDAM は、シンボリック・ポインターを用いることができま

すが、HISAM は、シンボリック・ポインターを用いなければなりません。シンボリック・ポインターは、PHDAM および PHIDAM データベースではサポートされません。

HALDB PSINDEX データベースの場合、ポインター・セグメントのセグメント接頭部はやや異なっています。「検索されるセグメントの RBA」フィールドは、拡張ポインター・セット (EPS) の一部であり、4 バイトを超えます。接頭部の中で、EPS の次にターゲット・ルート・セグメントのキーが続きます。

DEDB データベースの場合、ポインター・セグメントはシンボリックであることが必要です。

ターゲット・セグメント
: ターゲット・セグメントは通常のデータベースの中にあり、これがアプリケーション・プログラムが取り出す必要のあるセグメントです。ターゲット・セグメントは、ポインター・セグメントが指すセグメントです。ターゲット・セグメントは、データベースの中の 15 のレベルのうちどのレベルにあってもよく、ポインター・セグメントに保管されている RBA またはシンボリック・ポインターを用いて直接アクセスされます。 380 ページの『シンボリック・ポインター・フィールド』で説明する特別なケースを除いて、ターゲット・セグメントを検索するために、そのターゲット・セグメントの物理親が調べられることはありません。

ソース・セグメント
: ソース・セグメントも通常のデータベースの中にあります。ソース・セグメントには、ポインター・セグメントがそのキー・フィールドとして持っている (1 つ以上の) フィールドが入っています。 ソース・セグメントからデータがコピーされてこれがポインター・セグメントのキー・フィールドの中に置かれます。ソース・セグメントとターゲット・セグメントは同じセグメントであってもかまいませんし、ソース・セグメントはターゲット・セグメントの従属セグメントであってもかまいません。オプション・フィールドもソース・セグメントからコピーされますが、これには 1 つの例外があり、それについてはこのトピックで後から説明します。

制約事項**:** 順次従属 (SDEP) セグメントを持つ DEDB データベースは、副次索引データベースを持つことができます。ただし、SDEP セグメントを高速機能副次索引のターゲット・セグメントやソース・セグメントとして使用することはできません。

次の図に示す全機能教育用データベースでは、3 つのセグメントが連動します。教育用データベースは、シンボリック・ポインターではなしに RBA を使用する HIDAM データベースです。あるアプリケーション・プログラムが、受講者名によってこの教育用データベースにアクセスし、その受講者がとっているすべての科目の名前の一覧表を入手する必要があるものとします。

- このアプリケーションが取り出そうとしているセグメントは COURSE セグメントです。それが科目名 (COURSENM フィールド) をもっているセグメントだからです。そこで、COURSE がターゲット・セグメントであり、取り出す必要があります。
- この例の場合、このアプリケーション・プログラムは COURSE セグメントを取り出すための DL/I 呼び出しで受講者名を使用しようとします。この DL/I 呼

び出しは修飾された呼び出しであり、受講者名をその修飾子として使用します。ソース・セグメントには、副次索引の中のポインター・セグメントの順序で用いられるフィールドが入っています。この例では、ポインター・セグメントは受講者名に従って順序付ける必要があります。 STUDENT セグメントがソース・セグメントになります。このセグメントの中のフィールドが、キー・フィールドとしてポインター・セグメントのデータ部分にコピーされます。

- このアプリケーション・プログラムから呼び出しが出されると、指定されている受講者名と一致するキー・フィールドを持つポインター・セグメントを求めて、検索が開始されます。該当するポインター・セグメントがこの索引の中で見つかると、そのセグメントの中に、このアプリケーション・プログラムが取り出そうとしている COURSE セグメントのアドレスが入っています。

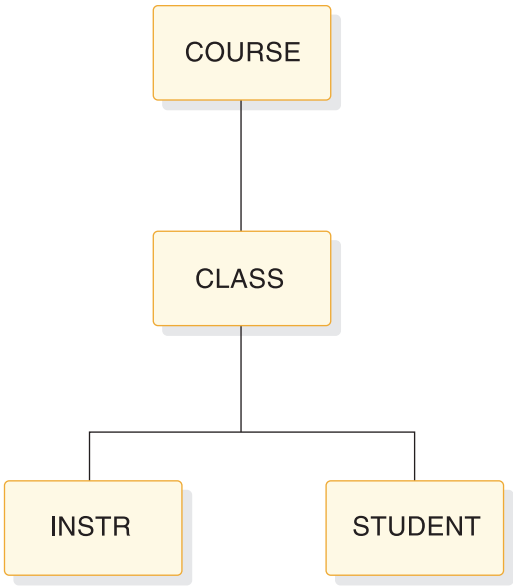


図 *172.* 教育データベース・レコード

以下の図は、ポインター・セグメント、ターゲット・セグメント、およびソース・セグメントが一緒に働く仕組みを示したものです。

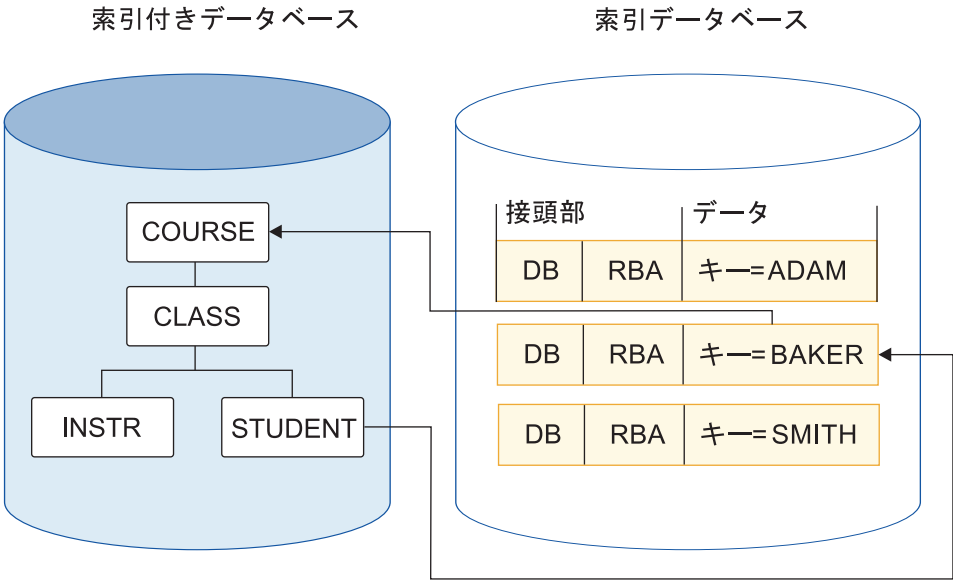


図 *173.* 副次索引を用いたセグメントへのアクセス方法

副次索引が使用されるときにアプリケーション・プログラムが出す呼び出しは、`GU COURSE (XNAME = BAKER ...)` です。

XNAME は、XDFLD ステートメントの NAME パラメーターで指定されているものです。

COURSE はアプリケーション・プログラムが検索するターゲット・セグメントです。

STUDENT は、アプリケーション・プログラムが呼び出しの中で修飾子として使用する 1 つ以上のフィールドが入っているソース・セグメントで、ポインター・セグメントのデータ部にキーとして含まれています。

副次索引の中の BAKER セグメントはポインター・セグメントであり、その接頭部に検索するセグメントのアドレスが入っており、そのデータ・フィールドにアプリケーション・プログラムが呼び出しの中で修飾子として使用するキーが入っています。

副次索引によるデータベースの階層の再構築

アプリケーション・プログラムで副次索引を介してデータベースにアクセスする場合、データベース・レコードは代替の順序で処理されます。

以下のトピックでは、副次索引が全機能データベースおよび DEDB データベースの処理に及ぼす影響について、詳しく説明します。

関連タスク:

835 ページの『IMS 索引の変更』

副次索引による全機能データベースの階層の再構築

PCB に PROCSEQ= パラメーターをコーディングして、アプリケーション・プログラムが副次索引を使用して処理を行う必要があることを指定すると、このアプリケーション・プログラムによるデータベース・レコードの認識方法が変わります。

ターゲット・セグメントがデータベース・レコードの中のルート・セグメントである場合、アプリケーション・プログラムが認識する構造は、アプリケーション・プログラムが通常の処理シーケンスを使用してアクセスできる構造と変わりません。しかし、ターゲット・セグメントがルート・セグメントでない場合には、データベース・レコードの中の階層が、概念上再構築されます。以下の図はこの概念を示したものです。

ターゲット・セグメントは (図 174 で示しているように) セグメント G です。ターゲット・セグメント G は、再構築された階層においては (373 ページの図 175 で示しているように) ルート・セグメントになります。ターゲット・セグメントのすべての従属セグメント (セグメント H、J、I) は、ターゲット・セグメントの従属セグメントのままです。しかし、ターゲットが従属しているすべてのセグメント (セグメント D と A)、およびそれらの従属セグメントは、ターゲットの従属セグメントになり、再構築された階層の左端の位置に置かれます。再構築された階層におけるこれらの位置は、直接的な従属の順序です。 D は G の直接従属セグメントとなり、A は D の直接従属セグメントとなります。

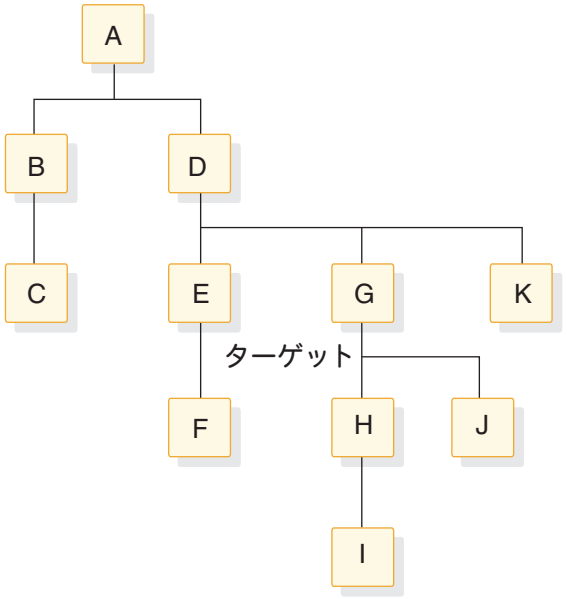


図 *174.* ターゲット・セグメント *G* を持つ物理データベース構造

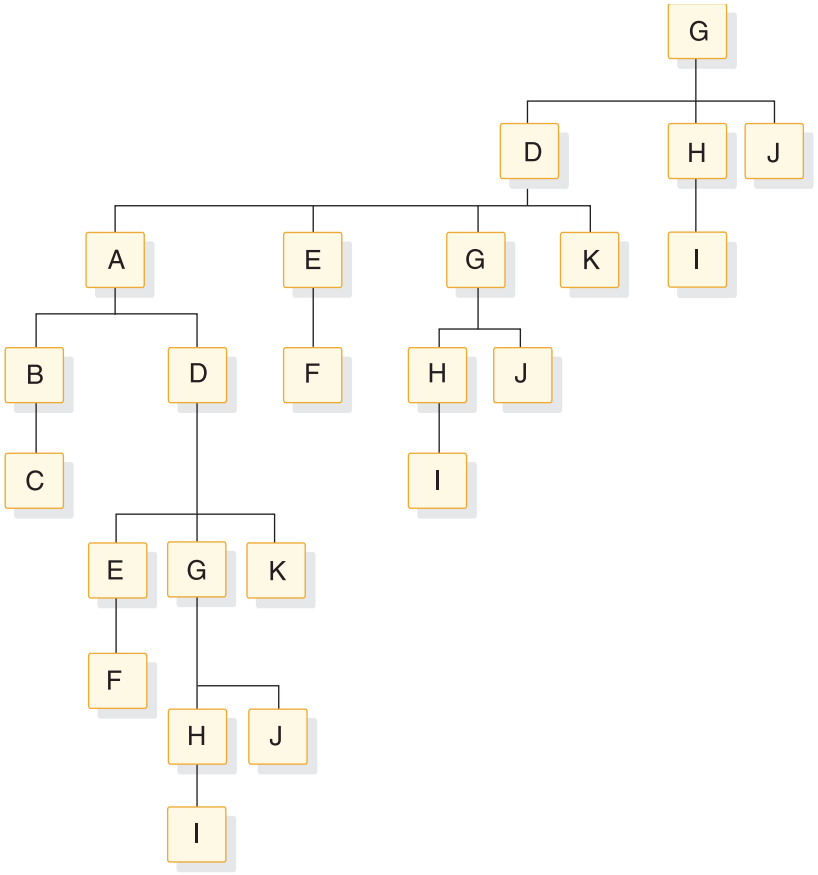


図 *175.* セグメント *G* の副次索引で索引に登録されたときの副次索引構造

この新しい構造は、2 次データ構造 と呼ばれます。 2 次データ構造を使用する場合は、処理上の制約があるため、ターゲット・セグメント、およびターゲット・セグメントが従属していたセグメント (ターゲット・セグメントの物理親、セグメント D および A) を挿入または削除することはできません。

2 次処理シーケンス

データベース・レコード内の階層の再構築によって、アプリケーション・プログラムがセグメントにアクセスする仕方が変わります。セグメントにアクセスする新しい順序は、2 次処理シーケンス と呼ばれます。 図 175 はアプリケーション・プログラムがデータベース・レコードを認識する方法を示したものです。

図 175 のように、同じセグメントが 1 回より多く参照される場合は、DBDGEN ユーティリティーを使用して、追加のセグメントの参照に代替名を割り当てるための論理 DBD を生成しなければなりません。論理 DBD を生成しないと、PSBGEN ユーティリティーから重複する SENSEG 名に対するメッセージ SEG150 が出ます。

副次索引による DEDB データベースの階層の再構築

1 次 DEDB データベースが、PROCSEQD= パラメーターを伴う PCB を使用して副次索引を介してアクセスされる場合、1 次 DEDB データベースは代替シーケンスで処理されます。

PCB に PROCSEQD= パラメーターをコーディングすると、アプリケーション・プログラムによるデータベース・レコードの認識方法が変更されます。ターゲット・セグメントが 1 次 DEDB データベース内のルート・セグメントである場合、副次索引を使用している 1 次 DEDB データベースの構造を逆転したものが、1 次 DEDB データベースの物理構造と同じです。後続の、ターゲット・セグメントの GU 後の非修飾 DL/I GNP 呼び出しまたは GN 呼び出しは、1 次 DEDB データベースのセグメントを物理構造順に返します。

ターゲット・セグメントが 1 次 DEDB データベースのルート・セグメントでない場合、データベース・レコードの中の階層は、逆転構造として概念上再構築されます。DEDB 逆転構造アクセスは、セグメントのサブセットだけに制限されます。DEDB 逆転構造アクセスの場合、ターゲット・セグメント、そのターゲット・セグメントからルート・セグメントまでの直接の親セグメント、およびそのターゲット・セグメントからのすべての子セグメントがアクセス可能です。

ルート・セグメントでないターゲット・セグメントの場合、そのターゲット・セグメントからルート・セグメントまでの直接の親セグメントは、それぞれの直接の親セグメントとターゲット・セグメントが定義された固有キーの FIELD ステートメントを持っている必要があります。FIELD ステートメントに直接の親セグメントの固有キーが存在しない場合、DBDGEN ユーティリティーは MNOTE 8 とメッセージ DGEN332 で終了します。

後続の、ターゲット・セグメントの GU 後の非修飾 DL/I GNP 呼び出しまたは GN 呼び出しは、以下のようにして、1 次 DEDB データベースのセグメントを概念上再構成された DEDB 逆転構造順に返します。

1. ターゲット・セグメントからその直接の親セグメントを通ってルート・セグメントまで、垂直方向に上へ縦断します。
2. ターゲットの直接の親セグメントからルート・セグメントまでが返されます。
3. ターゲット・セグメントからそのすべての子セグメントまで、垂直方向に下へ縦断します。
4. 1 つ以上のターゲットの子セグメントが返されます (要求した場合)。ターゲットの子セグメントに SENSEG を指定します。

次の 2 つの図は、PCB の中に PROCSEQD= パラメーターをコーディングして、1 次 DEDB データベースの副次索引データベースを使用した代替シーケンス処理を指示したときに、逆転構造が概念上再構築される様子を示しています。

次の図は、ルート A とターゲット・セグメント G を持つデータベース・レコードの物理構造の例です。

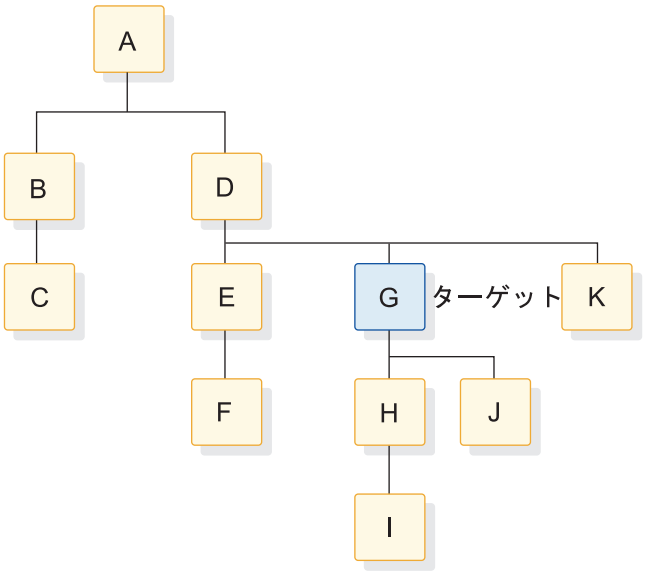


図 *176.* ターゲット・セグメント *G* を持つ物理データベース構造

次の図は、ターゲット・セグメント G を持つ DEDB 逆転データベース構造を示しています。

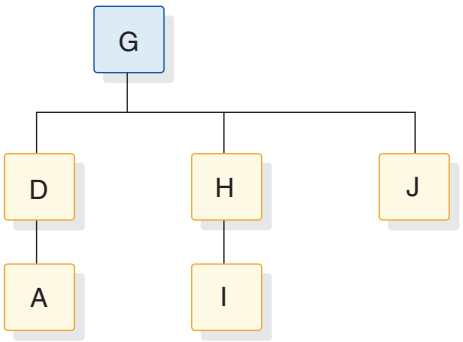


図 *177.* ターゲット・セグメント *G* からの *DEDB* 逆転データベース構造

ターゲット・セグメントは (この図で示しているように) セグメント G です。ターゲット・セグメント G は、再構築された階層においてはルート・セグメントになります。ターゲット・セグメントのすべての従属セグメント (セグメント H、I、J) は、ターゲット・セグメントの従属セグメントのままです。しかし、ターゲットが直接従属しているすべてのセグメント (セグメント D と A) は、再構築された階層の左端の位置に置かれます。再構築された階層におけるこれらの位置は、直接的な従属の順序です。 D は G の直接従属セグメントとなり、A は D の直接従属セグメントとなります。

新しい逆転構造は、*DEDB 2* 次データ構造 と呼ばれます。2 次データ構造を使用する場合は、処理上の制約があるため、ターゲット・セグメント、およびターゲット・セグメントが従属していたセグメント (ターゲット・セグメントの物理親、セグメント D および A) を挿入または削除することはできません。

以下のステップは、PROCSEQD= パラメーターを伴う PCB を使用した DL/I 呼び出しシーケンスを説明したものです。

1. 修飾 GU 呼び出しがターゲット・セグメント G を返します。
2. 後続の非修飾 GNP 呼び出しまたは GN 呼び出しがセグメント D、A、H、I、J を返します。
3. 非修飾 GNP 呼び出しの場合、セグメント J の後ろに GE 状況コードが返されます。
4. 非修飾 GN 呼び出しの場合、GN 呼び出しは副次索引データベース内でセグメント G の後の第 n 番目にあるセグメントを返します。
5. 非修飾 GN 呼び出しの場合、ステップ 2 と 4 を繰り返します。副次索引データベース内にセグメントがなくなった時点で、GB 状況コードが返されます。

関連タスク:

837 ページの『DEDB への副次索引の追加』

副次索引の保管方法

副次索引データベースには、ルート・セグメントしか入っていません。

ポインター・セグメントの中のキーが固有なものであれば、このルート・セグメントが単独の VSAM KSDS の中に保管されます。 キーが固有なものでない場合には、重複キーを持つセグメントを保管するために、もう 1 つのデータ・セット (すなわち ESDS) を使用しなければなりません。 (KSDS データ・セットでは、重複キーの使用はできません。) 重複キーが存在するのは、例えば、受講者名に基づいて科目を取り出すために副次索引を使用するような場合です。以下の図に示すように、生徒 1 人当たりいくつかのソース・セグメントが存在することがあります。

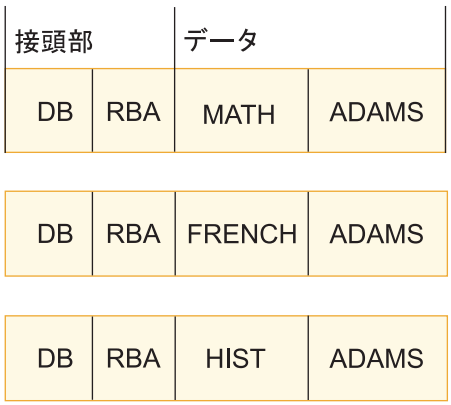


図 *178.* 受講者ごとのソース・セグメントの例

副次索引の中の各ポインター・セグメントが、1 つの論理レコードの中に保管されます。ポインター・セグメントが入っている論理レコードを以下の図に示します。

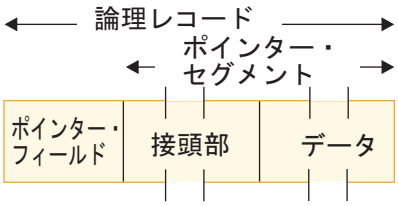


図 *179.* ポインター・セグメントが入っている論理レコードの例

HALDB の副次索引レコードが、以下の図に示されています。

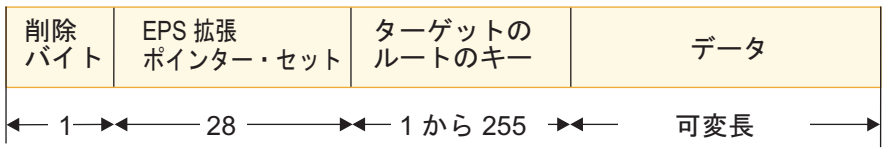


図 *180. HALDB* における副次索引項目

論理レコードのフォーマットは、KSDS データ・セットと ESDS データ・セットの中でも同じです。論理レコードの先頭にあるポインター・フィールドは、このセグメントのデータ部分の中のキーが固有なものでない場合にのみ存在します。キーが固有なものでない場合、いくつかのポインター・セグメントに重複キーが入ります。これらのポインター・セグメントを一緒にチェーニングしなければなりませんが、これを行うのに論理レコードの先頭のポインター・フィールドが使用されます。

重複キーが入っているポインター・セグメントは、LIFO 順 (後入れ先出し順) で ESDS の中に保管されます。最初の重複キー・セグメントが挿入されると、それは ESDS に書き込まれ、このセグメントを持つ KSDS 論理レコードが ESDS 内のセグメントを指します。 2 番目の重複キーが挿入されると、これは ESDS 内の使用可能な次の場所に収容されます。 KSDS 論理レコードは 2 番目の重複キー・セグメントを指すよう更新されます。重複ポインター・セグメントが LIFO 順に ESDS の中に挿入される効果は、元のポインター・セグメント (KSDS の中のポインター・セグメント) が最後に取り出されるということです。重複キーには、定義により、特別の順序はないので、この検索順序は問題ではありません。

ポインター・セグメントの中のフィールドのフォーマットと用法

すべてのセグメントと同じように、ポインター・セグメントには接頭部とデータ部があります。

このトピックには診断、変更、およびチューニングに関する情報が含まれています。

接頭部には削除バイトがあり、さらにシンボリック・ポインターではなく直接ポインターが使用される場合には、ターゲット・セグメントのアドレス (4 バイト) があります。データ部には一連のフィールドがあり、その一部のものはオプション・フィールドです。ポインター・セグメントのデータ部のすべてのフィールドには、ソース・セグメントから取られたデータが入っています (ただし、ユーザー・データを除きます)。これらのフィールドは、定数フィールド (オプション)、検索フィール

ド、サブシーケンス・フィールド (オプション)、重複データ・フィールド (オプション)、連結キー・フィールド (HISAM を除きオプション)、およびデータ (オプション) です。

以下の図は、ポインター・セグメントの中の諸フィールドを示したものです。

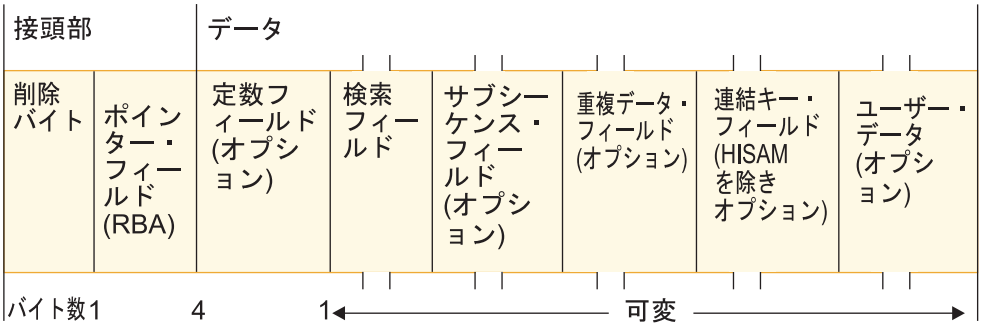


図 *181.* 副次索引の中のポインター・セグメントのフィールド

削除バイト

削除バイトは、セグメントがデータベースから削除されているかどうかを判別するために、IMS によって使用されます。

ポインター・フィールド

このフィールドには、存在する場合には、ターゲット・セグメントの RBA が入っています。 HD データベースを指す索引用に直接ポインターを指定している場合には、このポインター・フィールドが存在します。直接ポインターは、セグメントを指すのに単純にその実際のアドレスを使用します。

指定できるもう 1 つの種類のポインターは、シンボリック・ポインターです。シンボリック・ポインターについては、『シンボリック・ポインター・フィールド』で説明します。HD データベースを指すためにシンボリック・ポインターを使用することもでき、HISAM データベースを指すためにはシンボリック・ポインターを使用する必要があります。シンボリック・ポインターを使用する場合には、このフィールドは存在しません。

定数フィールド

このフィールドには、存在する場合、1 バイトの定数が入っています。ある 1 つの索引データベースに複数の索引が収容される場合、この定数が使用されます。この定数の共用索引データベースの中の特定の 1 つの索引に関するすべてのポインター・セグメントを示します。この定数フィールドの中の値はキーの一部分となります。

高速機能副次索引ポインター・セグメントには定数フィールドが存在しません。

検索フィールド

検索フィールドの中のデータは、ポインター・セグメントのキーです。検索フィールドの中のデータはすべて、ソース・セグメントからきたデータです。検索フィールドには、ソース・セグメントからのフィールドを 5 つまで入れることができま

す。これらのフィールドは、ソース・セグメントの中で隣接していなくてもかまいません。これらのフィールドをポインター・セグメントに保管するときは、任意の順序で保管することができます。保管時にこれらのフィールドは連結されます。検索フィールドの中のデータ (すなわち、キー) は、固有である場合も非固有である場合もあります。

ソース・セグメントに変更が加えられるたびに、IMS は自動的にポインター・セグメントの検索フィールドを更新します。

サブシーケンス・フィールド

サブシーケンス・フィールドには、検索フィールドと同様、ソース・セグメントからとられた 1 個から 5 個までのデータ・フィールドが入っています。サブシーケンス・フィールドはオプション・フィールドであり、非固有のキーがある場合に、このフィールドを使用することができます。サブシーケンス・フィールドによって、非固有キーを固有キーにすることができます。非固有キーには多くの欠点があるので、非固有キーを固有キーにすることは望ましいことです。例えば、非固有キーでは、重複キーを持つすべての索引セグメントを保管するために、余分なデータ・セット (ESDS) を 1 つ使用する必要が生じます。 ESDS には余分なスペースが必要です。さらに重要なのは、重複するキー・セグメントの中の特定のオカレンスを探すために余分な入出力操作が必要であり、これによってパフォーマンスが低下する恐れがあるという点です。

サブシーケンス・フィールドを使用すると、このフィールドの中のデータが検索フィールドの中のデータと連結されます。これらの連結フィールドが、ポインター・セグメントのキーになります。適切な選択を行えば、この連結フィールドによって固有キーができます。サブシーケンス・フィールドの中のソース・データを用いて固有キーを作ることは、常に可能とは限りません。したがって、固有キーを形成するためには、システム関連フィールドを使用できます。

サブシーケンス・フィールドの使用について注意すべき重要な点が 1 つあります。このフィールドを使用する場合にも、SSA のコーディング方法を変える必要はないという点です。 SSA は、依然として検索フィールドの中にあるものを指定することができ、検索フィールドとサブシーケンス・フィールドの中にあるものを加えたものを指定することはできません。アプリケーション・プログラムが副次索引を別個のデータベースとして処理するのでない限り、アプリケーション・プログラムはサブシーケンス・フィールドを見ることはありません。

サブシーケンス・フィールドには、ソース・セグメントからとられた最高 5 つのフィールドを入れることができます。これらのフィールドは、ソース・セグメントの中で隣接していなくてもかまいません。これらのフィールドをポインター・セグメントに保管するときは、任意の順序で保管することができます。保管時にこれらのフィールドは連結されます。

ソース・セグメントに変更が加えられるたびに、IMS は自動的にポインター・セグメントの中のサブシーケンス・フィールドを保守します。

重複データ・フィールド

重複データ・フィールドには、検索フィールドと同様、ソース・セグメントからとられた 1 個から 5 個までのデータ・フィールドが入っています。重複データ・フ

ィールドはオプション・フィールドです。アプリケーションで副次索引を別個のデータベースとして処理する場合には、重複データ・フィールドを使用してください。サブシーケンス・フィールドと同様に、アプリケーション・プログラムが副次索引を別個のデータベースとして処理するのでない限り、アプリケーション・プログラムは重複データ・フィールドを見ることはありません。

重複データ・フィールドには、ソース・セグメントからとられた最高 5 つのフィールドを入れることができます。これらのフィールドは、ソース・セグメントの中で隣接していなくてもかまいません。これらのフィールドをポインター・セグメントに保管するときは、任意の順序で保管することができます。保管時にこれらのフィールドは連結されます。

ソース・セグメントに変更が加えられるたびに、IMS は自動的にポインター・セグメントの中の重複データ・フィールドを保守します。

シンボリック・ポインター・フィールド

このフィールドが存在する場合には、このフィールドにはターゲット・セグメントの連結キーが入っています。ポインター・セグメントが、直接ではなくシンボリック・ポインターによってターゲット・セグメントを指す場合には、このフィールドが存在します。直接ポインターは、セグメントを指すのに単純にその実際のアドレスを使用します。シンボリック・ポインターは、セグメントを指すのに、その実際のアドレス以外の手段を使用します。副次索引においては、ターゲット・セグメントの連結キーがシンボリック・ポインターとして使用されます。

副次索引を用いて HDAM または HIDAM データベースの中のセグメントにアクセスする場合、シンボリック・ポインターを使用してこのセグメントにアクセスすることができます。HISAM データベースの中のセグメントにアクセスするには、シンボリック・ポインターを使用しなければなりません。その理由は、HISAM データベースの中のセグメントは「動き回る」ことができ、したがって、直接アドレス・ポインターの保守が大仕事となることがあるということです。シンボリック・ポインターを使用する場合に付随する問題点の 1 つは、ターゲット・セグメントに到達するにはターゲット・セグメントの物理親にアクセスしなければならないことです。この余分なアクセスのため、シンボリック・ポインターを使用してのターゲット・セグメントの検索は、直接ポインターを使用した検索ほど速くはありません。また、一般に、シンボリック・ポインターの方がポインター・セグメントより多くのスペースを中に必要とします。シンボリック・ポインターを使用する場合、接頭部の中にポインター・フィールド (4 バイト長) はありませんが、通常、ターゲット・セグメントの完全連結キーの長さが 4 バイトより大きくなります。

シンボリック・ポインターが指定されている場合は、IMS が自動的に連結キー・フィールドを生成します。

シンボリック・ポインターが指定されている場合でも、IMS が自動的に連結キー・フィールドを生成しない場合があります。これは、連結キーが完全に収容されるように、システム関連フィールド /CK を、サブシーケンス・フィールドまたは重複データ・フィールドとして指定している場合です。このような場合、サブシーケンス・フィールドか重複データ・フィールドのいずれかのシンボリック・ポインター部分が使用されます。

ポインター・セグメントの中のユーザー・データ

データを収容するのに十分なセグメントの長さを指定すれば、任意のユーザー・データをポインター・セグメントのデータ部分に収容することができます。アプリケーションで副次索引を別個のデータベースとして処理する場合には、ユーザー・データが必要です。サブシーケンス・フィールドと重複データ・フィールドの中のデータと同様、アプリケーションが副次索引を別個のデータベースとして処理するのでない限り、アプリケーション・プログラムはユーザー・データを見ることはありません。

ユーザー・データは最初、データベース管理者がロードしなければなりません。データベース管理者はまた、その維持管理を行わなければなりません。副次索引を使用するデータベースの再編成の間に、この副次索引データベースは IMS によって作り直されます。この処理の間に、ポインター・セグメントの中のすべてのユーザー・データは失われます。

関連概念:

392 ページの『副次索引データベースの共用』

関連タスク:

390 ページの『別個のデータベースとしての副次索引の処理』

HISAM 副次索引ポインターのフィールド

HISAM 副次索引データベースのフィールドは、ユニーク・キーと非ユニーク・キーの両方をサポートしています。

このトピックには診断、変更、およびチューニングに関する情報が含まれています。

HISAM 副次索引データベースは、ユニーク・キーと非ユニーク・キーの両方をサポートしています。非固有キーは、後入れ先出し法 (LIFO) の順で保管および検索されます。副次索引データベースが非固有キーをサポートしている場合は、KSDS と ESDS の両方のデータ・セットが必要です。最初に挿入される非固有キーは KSDS データ・セットに保管され、残りの非固有キーは ESDS データ・セットに LIFO 順で保管されます。

HISAM 副次索引データベースは、サブシーケンス・フィールド、重複データ・フィールド、ユーザー・データ・フィールド、および /CK オペランドをサポートします。サブシーケンス・フィールドおよび /CK オペランドは、副次索引キーを固有にするために使用できます。

HISAM 副次索引データベースは固定長セグメントを含み、区画選択出口ルーチンを使用したデータ区分化ができ、セグメント編集/圧縮出口ルーチン (DFSCMPX0) をサポートしています。

固有キーを持つ HISAM 副次索引ポインターのフィールド

次の図は、固有キーを持つ HISAM 副次索引ポインターのフィールドを示しています。

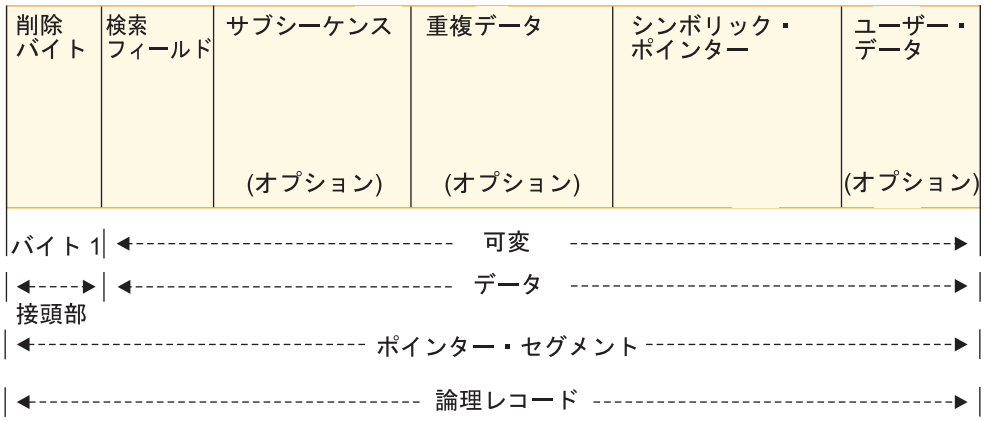


図 *182.* 固有キーを持つ *HISAM* 副次索引ポインターの例

論理レコード
: 副次索引ポインター・セグメントは、論理レコードに保管されます。

ポインター・セグメント
: 副次索引ポインター・セグメントには、接頭部フィールドとデータ・フィールドが入っています。

接頭部
: 削除バイト: 1 バイト

データ・フィールド

検索フィールド
: ソースからの最大 5 フィールドによって形成される可変長のバイト。

サブシーケンス・フィールド
: ソース値または IMS 生成値からの最大 5 フィールドによって形成される可変長のバイト (オプション)。これは、副次索引キーを固有にするために使用されます。これを使用して、副次索引データベース内のセグメントを順序付けることができます。検索フィールドとサブシーケンス・フィールドを一緒にしたものが、副次索引のキーを形成します。

重複データ・フィールド
: ソースからの最大 5 フィールドによって形成される可変長のバイト (オプション)。これは、副次索引をデータベースとして処理する場合にだけ使用されます。

シンボリック・ポインター・フィールド
: 可変長のバイト。これは、ターゲットへの連結キーです。

ユーザー・データ
: いくつかのユーザー・データ・フィールドによって形成される可変長のバイト (オプション)。これは、副次索引をデータベースとして処理する場合にだけ使用されます。

非固有キーを持つ HISAM 副次索引ポインターのフィールド

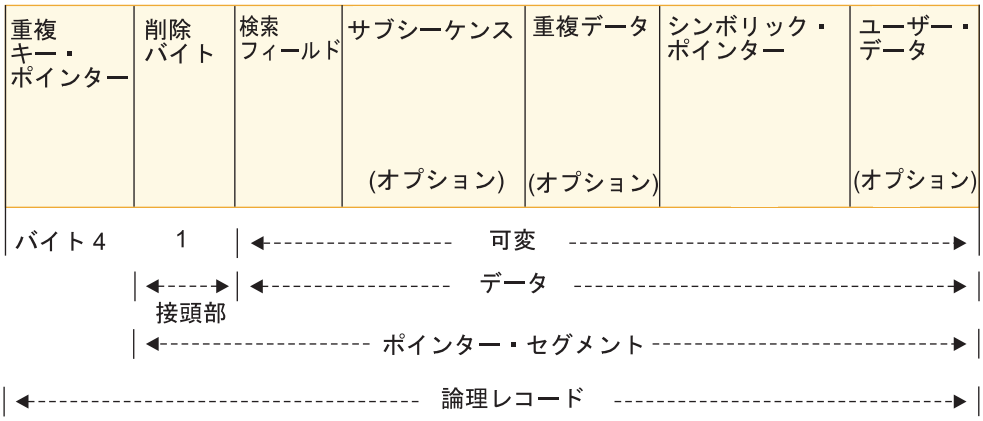


図 *183.* 非固有キーを持つ *HISAM* 副次索引ポインターの例

論理レコード
: 副次索引ポインター・セグメントは、論理レコードに保管されます。

  重複キー・ポインター
  : 副次索引キーが非固有である場合の HISAM 副次索引の 4 バイトからなるポインター。キーが固有なものでない場合、いくつかのポインター・セグメントに重複キーが入ります。これらのポインター・セグメントはまとめてチェーニングする必要があり、これを行うのに論理レコードの先頭の重複キー・ポインター・フィールドが使用されます。重複キー・ポインターは、DATASET ステートメントで OVFLW= オペランドが定義されている場合に存在します。

ポインター・セグメント
: 副次索引ポインター・セグメントには、接頭部フィールドとデータ・フィールドが入っています。

接頭部
: 削除バイト: 1 バイト

データ・フィールド

  検索フィールド
  : ソースからの最大 5 フィールドによって形成される可変長のバイト。

  サブシーケンス・フィールド
  : ソース値または IMS 生成値からの最大 5 フィールドによって形成される可変長のバイト (オプション)。これは、副次索引キーを固有にするために使用されます。これを使用して、副次索引データベース内のセグメントを順序付けることができます。検索フィールドとサブシーケンス・フィールドを一緒にしたものが、副次索引のキーを形成します。

  重複データ・フィールド
  : ソースからの最大 5 フィールドによって形成される可変長のバイト (オプション)。これは、副次索引をデータベースとして処理する場合にだけ使用されます。

シンボリック・ポインター・フィールド
: 可変長のバイト。これは、ターゲットへの連結キーです。

ユーザー・データ・フィールド
: いくつかのユーザー・データ・フィールドによって形成される可変長のバイト (オプション)。これは、副次索引をデータベースとして処理する場合にだけ使用されます。

関連タスク:

390 ページの『別個のデータベースとしての副次索引の処理』

SHISAM 副次索引ポインターのフィールド

SHISAM 副次索引データベースを DBRC に登録する必要はありません。SHISAM データベースは KSDS のみをサポートしており、ESDS をサポートしていないため、SHISAM 副次索引データベースはユニーク・キーのみをサポートします。

SHISAM 副次索引データベースは、サブシーケンス・フィールド、重複データ・フィールド、ユーザー・データ・フィールド、および /CK オペランドをサポートします。サブシーケンス・フィールドおよび /CK オペランドは、副次索引キーを固有にするために使用できます。

SHISAM 副次索引データベースは固定長セグメントを含み、DEDB 区画選択出口ルーチンを使用したデータ区分化ができ、セグメント編集/圧縮出口ルーチンをサポートしています。

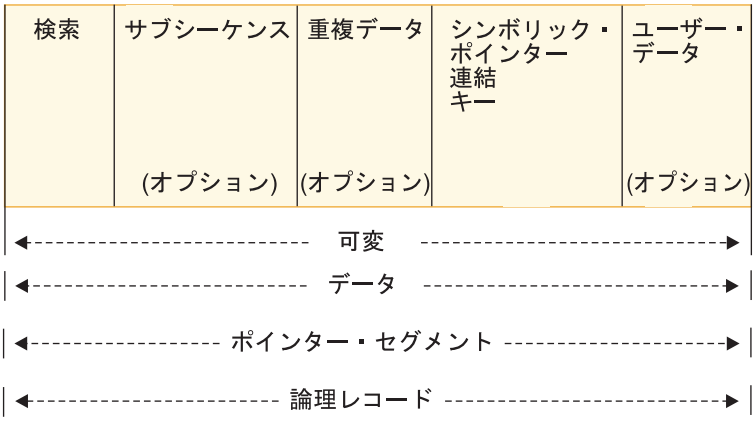


図 *184. SHISAM* 副次索引ポインターの例

論理レコード
: 副次索引ポインター・セグメントは、論理レコードに保管されます。

ポインター・セグメント
: 副次索引ポインター・セグメントには、接頭部フィールドとデータ・フィールドが入っています。SHISAM 副次索引セグメントは接頭部フィールドを備えていないため、副次索引ポインター・セグメントにはデータ・フィールドだけが含まれています。

データ・フィールド

検索フィールド
: ソースからの最大 5 フィールドを含んでいる可変長のバイト。検索フィールドは、副次索引のキーです。

サブシーケンス・フィールド **(**オプション**)**
: ソース値または IMS 生成値からの最大 5 フィールドを含んでいる可変長のバイト。サブシーケンス・フィールドは副次索引キーを固有のものにします。また、副次索引データベース内のセグメントを順序付けるために使用できます。サブシーケンス長は、連結キーの長さを決定するために使用されます。

重複データ・フィールド **(**オプション**)**
: ソースからの最大 5 フィールドを含んでいる可変長のバイト。このフィールドは、副次索引をデータベースとして処理する場合にだけ使用されます。

シンボリック・ポインター連結キー・フィールド
: 可変長のバイト。このフィールドは、ターゲットへの連結キーです。

ユーザー・データ・フィールド **(**オプション**)**
: いくつかのユーザー・データ・フィールドを含んでいる可変長のバイト。このフィールドは、副次索引をデータベースとして処理する場合にだけ使用されます。

関連タスク:

390 ページの『別個のデータベースとしての副次索引の処理』

システム関連フィールドの使用によるキーの固有化

ポインター・セグメントのサブシーケンス・フィールドにソース・セグメントから追加情報を保持する方法で固有キーを作成しても効果がない場合は、強制的にキーを固有にする方法がほかに 2 つあります。この方法はどちらも、DBD でソース・セグメントの FIELD ステートメントのオペランドを使用します。

FIELD ステートメントは、1 つのセグメント・タイプの中の各フィールドを定義します。

/SX オペランドの使用

HD データベースに対しては、/SX で始まる NAME フィールドを持つ FIELD ステートメントをコーディングすることができます。 この /SX の後ろには、必要な任意の文字 (5 文字以内) を続けることができます。このオペランドを使用すると、セグメントの挿入の間に、ソース・セグメントの RBA、あるいは、PHDAM または PHIDAM 用の 8 バイトの ILK がシステムにより生成されます。また、このシステムではポインター・セグメントのサブシーケンス・フィールドにその RBA または ILK が設定されるので、キーが固有なものであることが保証されます。 /SX がコーディングされる FIELD ステートメントは、ソース・セグメントの中のフィールドを定義するためのステートメントです。しかし、/SX の値がソース・セグメントの中に置かれるわけではありません。この値はポインター・セグメントの中に置かれます。

/SX オペランドを使用する場合には、この DBD の中の XDFLD ステートメントでも、/SX (と /SX オペランドに付け加えた文字すべて) を指定しなければなりません。 XDFLD ステートメントの働きの 1 つは、ソース・セグメントからどのフィールドを取り出してこれをポインター・セグメントの中に置くか指定することができます。 /SX オペランドは、XDFLD ステートメントの SUBSEQ= オペランドで指定されます。

/CK オペランドの使用

強制的にキーを解消する他の方法は、/CK で始まる NAME パラメーターを持つ FIELD ステートメントをコーディングすることです。 /CK は、サブシーケンス・フィールドとして使用されると、ポインター・セグメントのキーが固有なものになることを保証します。このオペランドは、HISAM、HDAM、PHDAM、HIDAM、または PHIDAM データベースに使用することができます。 /CK の後ろには、任意の文字 (5 文字以内) を続けることができます。 /CK オペランドは /SX オペランドと同様の働きをしますが、ソース・セグメントの RBA ではなく連結キーが使用されるという点で異なっています。もう 1 つの違いは、連結キーが、ポインター・セグメントのサブシーケンス・フィールドまたは重複データ・フィールドに置かれるという点です。連結キーをどこに置くかは、どこで /CK を指定するかに応じて決まります。

/CK を使用する場合、ソース・セグメントの連結キーの一部分を使用することもでき (一部分を使用するだけでも、キーを固有なものにすることができるとき)、あるいは連結キーの全体を使用することもできます。 FIELD ステートメントの BYTES= オペランドと START= オペランドを用いて、何が必要かを指定します。

例えば、以下の図のデータベース・レコードを使用しているとします。

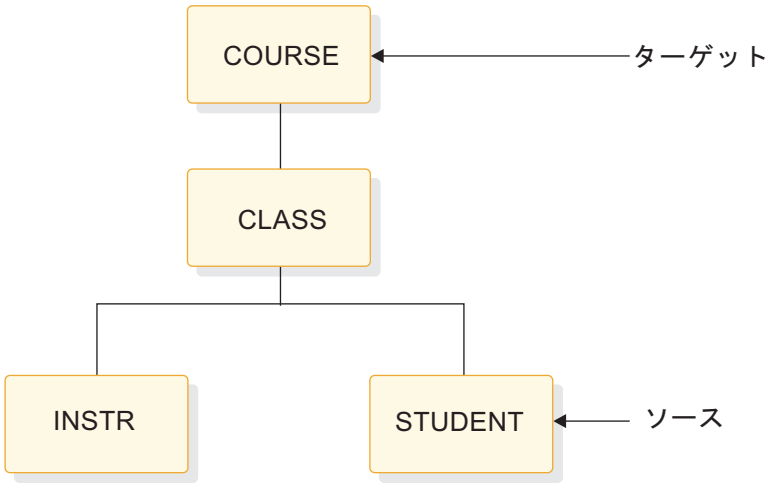


図 *185.* 副次索引用のソースおよびターゲットを示すデータベース・レコード

STUDENT セグメントの連結キーを以下の図に示します。

COURSECD	CLASSNO	SEQ
バイト 15	3	3

図 *186. STUDENT* セグメントの連結キー

/CK で始まる名前を持つ FIELD ステートメントで、BYTES=21、START=1、と指定すると、ソース・セグメントの連結キー全体がポインター・セグメントの中に置かれます。 BYTES=6、START=16、と指定すると、連結キーの最後の 6 バイト (CLASSNO と SEQ) だけがポインター・セグメントの中に置かれます。 BYTES= オペランドは、PCB キー・フィードバック域の中にあるソース・セグメントの連結キーから何バイト取るべきであるかシステムに伝えます。 START= オペランドは、取るべき情報の (連結キーの始まりを基準にした) 開始位置をシステムに伝えます。 /SX オペランドの場合と同様に、DBD の中の XDFLD ステートメントにも /CK を指定する必要があります。

要約すると、キー・フィールドを固有なものにするために、XDFLD ステートメントの SUBSEQ= パラメーターに /SX フィールドと /CK フィールドを組み込むことができます。キー・フィールドを固有なものにすれば、重複キーを保持するための ESDS の使用に伴うオーバーヘッドを回避することができます。 XDFLD ステートメントの DDATA= パラメーターでも /CK フィールドを指定することができますが、このフィールドはキー・フィールドの一部とはなりません。

キーの固有化を行うときに、シンボリック・ポインターが使用されるのであれば、固有のシーケンス・フィールドをターゲット・セグメント・タイプで定義しなければなりません。また、(再構築された階層ではなく、データベースの物理階層の中において) ターゲット・セグメント・タイプが従属しているすべてのセグメント・タイプで、固有のシーケンス・フィールドが定義されなければなりません。

疎索引による索引項目の抑止方法

データベースにおいてソース・セグメントのロード、挿入、および置き換えを行うときに、DL/I は自動的に索引の中にポインター・セグメントを作成するか更新します。一部のポインター・セグメントが作成される必要はないと指定しない限り、これは自動的に行われます。

例えば、教育用データベースのための副次索引があるものとします。STUDENT がソース・セグメントであり、COURSE がターゲット・セグメントです。受講者がある客先番号と結び付いている場合にのみ、受講者のポインター・セグメントを設ける必要があるものとします。その場合には、疎索引を用いるとこれを行うことができます。これは副次索引のパフォーマンスの向上策の 1 つです。

HALDB 区分副次索引 (PSINDEX) は疎索引をサポートしますが、PSINDEX でセグメントに疎索引を作成しても、索引付きデータベースと PSINDEX の間でポインターを管理する間接リスト・データ・セット (ILDS) 内の間接リスト項目 (ILE) の数は減りません。 ILDS 内の ILE の数は、PSINDEX 内のセグメント数ではなく、索引付きデータベース内のターゲット・セグメントの数と一致します。

疎索引の利点

疎索引を用いると、どのような条件のもとでポインター・セグメントを抑止すべきであるか、すなわち、ポインター・セグメントを生成せず、したがって、これを索引データベースに収容すべきでないことを指定することができます。疎索引には 2 つの利点があります。主な利点は、索引のサイズを縮小し、これによってスペースを節約すると共に索引の維持管理を少なくすることです。索引のサイズの縮小により、パフォーマンスが向上します。もう 1 つの利点は、不要な索引項目を生成しないで済むということです。

BMP 領域に対する索引保守の抑止

DEDB データベースに副次索引が定義されている場合、IMS はソース・ステートメントの挿入、更新、または削除が行われたときに自動的に索引保守を実行します。索引抑止オプションを使用すると、1 つ以上の副次索引が定義されている DEDB データベースを、索引保守を使用せずに更新することができます。アプリケーションで 1 次 DEDB データベースに対して多数の更新を行う場合、本来なら関連する副次索引データベースに対して大量の索引保守が発生しますが、そのアプリケーションについて、索引保守を抑止することができます。その場合、後で社内アプリケーションまたはベンダー・ツール製品を使用して、1 次 DEDB データベースとその複数の副次索引データベースを同期させます。

索引保守を抑止するには、IMS BMP 領域の JCL の中に //DFSCTL DD ステートメントを指定します。

```
 //DFSCTL  DD  *
SETI PSB=psbname
```

SETI PSB=psbname パラメーターは、psbname という PSB の BMP アプリケーション用に副次索引が定義されているすべての DEDB データベースについて、索引保守を抑止します。

SETI ステートメント内の PSB=*psbname* パラメーターの *psbname* が BMP アプリケーションの PSB 名に一致しない場合、または SETI ステートメントの中で PSB= パラメーターが指定されていない場合は、メッセージ DFS0510E が発行され、アプリケーションは ABENDU1060 で終了します。ユーザーは SETI ステートメントを修正して、BMP アプリケーションを再実行する必要があります。

疎索引の指定

以下の 2 つの方法で疎索引を指定することができます。

第 1 に、ポインター・セグメントを索引に収容する必要のない条件を DBD の中の XDFLD ステートメントの NULLVAL= オペランドの値をコーディングすることができます。 BLANK、ZERO、または 1 バイトの任意の値 (例えば、X'10'、C'Z'、5、B'00101101' など) を NULLVAL= オペランドに入れることができます。

- BLANK は、C ' ' または X'40' と同じです。
- ZERO は X'00' と同じですが、C'0' と同じではありません。

NULLVAL= オペランドを使用した場合、ソース・フィールドのどのバイトも、このオペランドで使用されている値をもっていれば、ポインター・セグメントの生成が抑止されます。

第 2 に、NULLVAL= オペランドにコーディングできる値では目的を達成できない場合には、ポインター・セグメントを索引の中に置かない条件を決定するための索引保守出口ルーチンを作成することができます。ユーザーが独自の索引保守出口ルーチンを作成した場合には、その名前を DBD の中の XDFLD ステートメントの EXTRTN= オペランドで指定します。 各副次索引には、索引保守出口ルーチンを 1 つずつだけ用意できます。しかし、この出口ルーチンは、複数の副次索引によって使用される汎用出口ルーチンでもかまいません。

出口ルーチンは、ある特定のポインター・セグメントを索引の中に置くべきであるかどうかを首尾一貫して決定できるものでなければなりません。出口ルーチンは、同じポインター・セグメントを異なる 2 つの時点で調べることはできませんが、1 回のみこのセグメントの抑止を決定することができます。また、この出口ルーチンにおいて、あるポインター・セグメントを索引の中に置くかどうか決定するにあたって、ユーザー・データを使用することはできません。というのは、ポインター・セグメントを索引に挿入する必要がある場合に、この出口ルーチンは、挿入の直前の実際のポインター・セグメントしか見ないからです。ポインター・セグメントの置き換えないし削除を行おうとしているときには、この出口ルーチンはポインター・セグメントの原型しか見ません。この原型には、定数フィールド、検索フィールド、サブシーケンス・フィールド、および重複データ・フィールドの内容が入っており、さらにシンボリック・ポインターがあればシンボリック・ポインターも入っています。

副次索引保守出口ルーチンのコーディングに必要な情報は、「*IMS V13* 出口ルーチン」に記載されています。

副次索引の維持管理方法

データベースでソース・セグメントの挿入、削除、または置き換えが行われると、この更新を行うアプリケーション・プログラムが副次索引を使用しているかどうかに関係なく、IMS は索引を最新の状態に保ちます。

行われる操作に応じて、IMS が索引を維持管理する方法は異なります。行われる操作に関係なく、IMS は常に挿入、削除、置き換えの対象となるソース・セグメントの中の情報を使用して、1 つのポインター・セグメントを作成することにより、索引の維持管理を開始します。 (このポインター・セグメントは作成されますが、まだ副次索引データベースの中には保管されません。)

ソース・セグメントの挿入

ソース・セグメントが挿入された場合には、DL/I はポインター・セグメントを抑止する必要があるかどうかを判断します。これを抑止する必要があれば、このポインター・セグメントを副次索引の中には置きません。抑止する必要がなければ、このポインター・セグメントは副次索引の中に入れられます。

ソース・セグメントの削除

ソース・セグメントが削除されるときに、IMS はポインター・セグメントが抑止されていたかどうかを判別します。抑止されていた場合、IMS は索引の保守は行いません。セグメントが抑止されていた場合、索引の中にはこれに対応する、削除すべきポインター・セグメントありません。ポインター・セグメントが抑止されていな

い場合、IMS は索引の中で対応するポインター・セグメントを探し、それを削除します。 ESDS データ・セットを指すポインターがセグメントに含まれている場合 (これは固有でない副次索引で起こります) を除いて、KSDS データ・セット内の削除されたポインター・セグメントが入っている論理レコードは消去されます。

ソース・セグメントの置き換え

ソース・セグメントが置き換えられる場合には、索引の中のポインター・セグメントに影響が及ぶこともあり、そうでないこともあります。索引の中のポインター・セグメントは、置換または削除が必要な場合もあれば、変更が不要な場合もあります。置き換えあるいは削除がおこなわれた後に、新しいポインター・セグメントが挿入されます。 IMS は、作成したポインター・セグメント (新しいポインター・セグメント) と副次索引内の対応するポインター・セグメント (古いポインター・セグメント) とを比較して、どのような変更を行う必要があるかを判別します。

- 新旧のポインター・セグメントを両方とも抑止する必要がある場合は、IMS は処理を行いません (索引の中にポインター・セグメントは存在しません)。
- 新しいポインター・セグメントを抑止する必要があるが、古いポインター・セグメントを抑止する必要がない場合には、古いポインター・セグメントが索引から削除されます。
- 新しいポインター・セグメントを抑止する必要はないが、古いポインター・セグメントが抑止されている場合には、新しいポインター・セグメントが副次索引に挿入されます。
- 新旧のポインター・セグメントを両方とも抑止する必要がない場合で、しかも、
  - 古いポインター・セグメントに対する変更がない場合は、IMS は処理を行いません。
  - 新しいポインター・セグメントの中のキー・データ部以外の部分が古いポインター・セグメントと異なっている場合には、古いポインター・セグメントが置き換えられます。索引ポインター・セグメントのユーザー・データは、ポインター・セグメントが置き換えられるときに保持されます。
  - 新しいポインター・セグメントの中のキー部分が古いポインター・セグメントと異なる場合には、古いポインター・セグメントが削除され、新しいポインター・セグメントが挿入されます。ユーザー・データは、索引ポインター・セグメントが削除されて新しいポインター・セグメントが挿入されるときには保持されません。

ユーザーが副次索引の再編成を行った場合、この副次索引に非固有キーが含まれていると、その結果としてのポインター・セグメントの順序は、予測できません。

別個のデータベースとしての副次索引の処理

副次索引は実際のデータベースであるので、独立して処理することができます。

アプリケーション・プログラムが副次索引を独立のデータベースとして処理しようとする理由がいくつかあります。例えば、あるアプリケーション・プログラムは、データベースから小さなデータを取り出すために、副次索引を使用することができます。このデータをポインター・セグメントの中に置いておけば、このアプリケーション・プログラムは、通常のデータベースに対する入出力操作を行わずに、このデータを取り出すことができます。このデータがソース・セグメントの中にあると

すれば、このデータをポインター・セグメントの重複データ・フィールドの中に置くことができます。そうしない場合には、このデータをポインター・セグメントの中のユーザー・データとして保持しなければならなくなります。 (データをユーザー・データとして保持する場合には、 1 次データベースを再編成して副次索引を作り直すとこのデータは失われます。)

副次索引を別個のデータベースとして処理しようとするもう 1 つの理由は、副次索引の維持管理のためです。例えば、サブシーケンス・フィールドまたは重複データ・フィールドをスキャンして、複数の索引の間で論理比較ないしデータ整理を行うことができます。あるいは、ポインター・セグメントのユーザー・データ部分への追加ないし変更を行うこともできます。副次索引を別個のデータベースとして処理しない限り、アプリケーション・プログラムはユーザー・データまたは重複データ・フィールドの内容を見ることはできません。

副次索引を別個のデータベースとして処理する場合には、主として副次索引データベースの保護を目的とするいくつかの処理上の制約事項があります。その制約事項を次に示します。

- セグメントを挿入することはできません。
- セグメントを削除することができます。ただし、セグメントを削除すると、この副次索引は索引としての使用に適さなくなる可能性があります。
- ポインター・セグメントのキー・フィールド (検索フィールド、および存在するならば定数フィールドならびにサブシーケンス・フィールドで構成されています) を置き換えることはできません。

副次索引データベースを保護するためのシステム上の制約事項に加えて、 DBD ステートメントの PROT オペランドを用いて、副次索引データベースの保護を強化することができます。 PROT が指定されている場合には、アプリケーション・プログラムはポインター・セグメントの中のユーザー・データを置き換えることしかできません。ただし PROT が指定されていても、ポインター・セグメントを削除することはできます。ポインター・セグメントを削除するときは、このポインター・セグメントを作成する原因になったソース・セグメントは削除されません。したがって、IMS がすでに削除されたポインター・セグメントに関して維持管理を行おうとする可能性があることに注意してください。既存のソース・セグメントのためのポインター・セグメントが見当たらないと、NE 状況コードを戻します。 NOPROT が指定されている場合には、アプリケーション・プログラムは、ポインター・セグメントの中の定数フィールド、検索フィールド、およびサブシーケンス・フィールドを除くすべてのフィールドを置き換えることができます。 PROT がこのパラメーターのデフォルトです。

アプリケーション・プログラムが副次索引を別個のデータベースとして処理するためには、このアプリケーション・プログラムに PCB を 1 つコーディングすれば済みます。この PCB は、副次索引の DBD を参照しなければなりません。アプリケーション・プログラムが、副次索引データベースを処理するために、修飾された SSA を使用する場合には、この SSA では、修飾子としてポインター・セグメントの完全なキーを使用しなければなりません。完全なキーは、検索フィールド、サブシーケンス・フィールド、および定数フィールドで構成されています (ただし、後の 2 つのフィールドが存在する場合です)。アプリケーション・プログラムの PCB キー・フィードバック域にキー・フィールド全体が収容されています。

共用副次索引を使用する場合、ある 1 つのアプリケーション・プログラムから出された複数の呼び出し (例えば、一連の GN 呼び出し) によって、これらの呼び出しが対象とする副次索引の境界違反が起こることはありません。共用データベースの中の各副次索引は、固有の DBD 名とルート・セグメント名を持っています。

関連概念:

377 ページの『ポインター・セグメントの中のフィールドのフォーマットと用法』

副次索引データベースの共用

索引データベースは、最大 16 個の副次索引を格納できます。データベースに複数の副次索引が含まれている場合、このデータベースは共用索引データベース と呼ばれます。HALDB と DEDB は、共用副次索引をサポートしていません。

共用索引データベースを使用することによって主記憶域をいくらか節約することはできますが、一般に、共用索引データベースの使用によって生じる不利益の方が、その使用によって少量のスペースが節約されることよりも大きいといえます。

共用索引データベースのもともとの利点は、バッファーおよびいくつかの制御ブロックで使用される大量の主ストレージを節約することでした。しかし、VSAM が共用リソースにより拡張されると、ストレージの節約の重要性は低くなりました。

例えば、複数のアプリケーション・プログラムが同時に共用索引データベースを使用すれば、パフォーマンスの低下をきたす恐れがあります。 (アームが複数の副次索引の間を行ったり来たりしなければならないので、検索時間が増えます。) さらに、共用索引データベースの維持管理、リカバリー、および再編成によって、パフォーマンスが低下することがあります。1 つの副次索引に変動があれば、すべての副次索引がある程度その影響を受けるからです。例えば、副次索引を用いてアクセスされるデータベースが再編成されると、IMS は自動的に新しい副次索引を 1 つ作成します。つまり、この共用索引データベースの中の他のすべての索引を新しい共用索引にコピーしなければならないことを意味しています。

共用索引データベースを使用する場合には、このデータベースについて以下の点を知っておく必要があります。

- 共用索引データベースは、1 つの副次索引を持つ索引データベースと同じように作成され、アクセスされ、さらに、維持されます。
- 共用索引データベースの中のさまざまな副次索引は、同じデータベースを索引する必要はありません。
- ご使用のシステムのすべての副次索引を 1 つの共用索引データベースに収容できます (副次索引の数が 16 を超えない場合)。

共用索引データベースにおいては、

- すべての索引セグメントが同じ長さでなければなりません。
- すべてのキーが同じ長さでなければなりません。
- セグメントの先頭から検索フィールドに至るまでのオフセットが同じでなければなりません。これは、すべてのキーが固有であるか、逆にすべて非固有でなければならないことを意味しています。非固有キーの場合は、ポインター・フィールドがターゲット・セグメントの中に存在します。固有キーの場合には、このフィ

ールドは存在しません。したがって、固有キーと非固有キーとが混じっていると、キー・フィールドまでのオフセットに 4 バイトの差が出てくることになります。

さまざまな副次索引の中の検索フィールドの長さが同じでない場合は、サブシーケンス・フィールドを用いて、強制的にキー・フィールドの長さを等しくすることができます。各キー・フィールドを等しい長さにするのに必要な数だけのバイトを、サブシーケンス・フィールドの中に置くことができます。

- 副次索引データベースの中の各副次索引を固有なものとして他の副次索引から区別するために、各共用副次索引に対して定数を 1 つずつ指定しておかなければなりません。 IMS は、副次索引データベースの中の各ポインター・セグメントの定数フィールドにこの ID を入れます。共用索引に関しては、キーは定数フィールド、検索フィールド、および (もし使用されていれば) サブシーケンス・フィールドです。

共用副次索引データベース・コマンド

コマンドは、最初の副次索引のために出されたのか、後続の副次索引のために出されたのかにより、異なる働きをする場合があります。最初の副次索引は、共用副次索引 DBDGEN の DBDUMP ステートメントで指定された最初のデータベース名です。この最初のデータベースは、実データベースです。他の副次索引データベースは、物理的には実データベースの一部ですが、論理的には別個のものです。

以下の表の最初の列は出すコマンド、2 番目の列はコマンドが出される場所、3 番目の列は出されたコマンドの影響、そして 4 番目の列は補足のコメントを示しています。

表 *61.* 共用副次索引データベース・コマンドの効果

発行するコマンド	発行場所	影響	コメント
/STOP /LOCK UPDATE DB STOP(SCHD) UPDATE DB SET(LOCK(ON))	最初の副次索引	指定されたデータベースのみ	DBRC による実データベースの許可を生じさせる共用副次索引についてスケジュールされているアプリケーションがまったくない場合は、これらのコマンドの効果は、最初の副次索引に対する /DBRECOVERY または UPD DB STOP(ACCESS) コマンドの場合と同じです。 共用副次索引データベースに対して /DISPLAY DB または QUERY DB コマンドが出された場合は、後続の副次索引が停止またはロック状態として表示されるのは、/STOP、UPD DB STOP(SCHD)、/LOCK、UPD DB SET(LOCK(ON))、UPD DB STOP(ACCESS)、または /DBRECOVERY コマンドが出された場合のみです。 /STOP、UPD DB STOP(SCHD)、UPD DB SET(LOCK(ON))、または /LOCK コマンドを取り消すには、最初の副次索引に対して、/START、UPD DB START(ACCESS)、UPD DB SET lOCK(OFF))、または /UNLOCK コマンドを出します。
/STOP UPD DB STOP(SCHD) /LOCK UPD DB SET(LOCK(ON))	後続の副次索引	指定されたデータベースのみ	/STOP、UPD DB STOP(SCHD)、UPD DB SET(LOCK(ON))、または /LOCK コマンドを取り消すには、指定されているデータベースに対して、/START、UPD DB START(ACCESS)、UPD DB SET(LOCK(OFF))、または /UNLOCK コマンドを出します。

表 61. 共用副次索引データベース・コマンドの効果 (続き)

発行するコマンド	発行場所	影響	コメント
/DBDUMP UPD DB STOP(UPDATES)	最初の副次索引	副次索引データ・セットを共用する全データベース	/DBDUMP または UPD DB STOP(UPDATES) コマンドは、共用データベース内のすべての索引についてのアクティビティーを静止します。この後、データベースはクローズされ、入力専用に再オープンされます。 /DBDUMP または UPD DB STOP(UPDATES) コマンドを取り消すには、最初の副次索引に対して /START または UPD DB START(ACCESS) コマンドを出します。
/DBDUMP UPD DB STOP(UPDATES)	後続の副次索引	指定されたデータベースのみ	副次索引は読み取り専用で使用可能です。 /DBDUMP または UPD DB STOP(UPDATES) コマンドを取り消すには、指定されているデータベースに対して /START または UPD DB START(ACCESS) コマンドを出します。
/DBRECOVERY UPD DB STOP(ACCESS)	最初の副次索引	副次索引データ・セットを共用する全データベース	/DBRECOVERY および UPD DB STOP(ACCESS) コマンドは、共用データベース内のすべての索引についてのアクティビティーを静止します。その後、データベースはクローズされ停止されます。 共用副次索引データベースに対して /DISPLAY コマンドが出された場合は、後続の副次索引が停止またはロック状態として表示されるのは、/STOP、UPD DB STOP(SCHD)、/LOCK、UPD DB SET(LOCK(ON))、/DBRECOVERY、または UPD DB STOP(ACCESS) コマンドが出された場合のみです。 /DBRECOVERY または UPD DB STOP(ACCESS) コマンドを取り消すには、最初の副次索引に対して /START または UPD DB START(ACCESS) コマンドを出します。

表 61. 共用副次索引データベース・コマンドの効果 (続き)

発行するコマンド	発行場所	影響	コメント
/DBRECOVERY UPD DB STOP(ACCESS)	後続の副次索引	指定されたデータベースのみ	このコマンドは、指定されたデータベースに対する /STOP および UPD DB STOP(SCHD) コマンドと同じです。ただし、/DBRECOVERY および UPD DB STOP(ACCESS) コマンドは即時に効力が発生するのに対し、/STOP および UPD DB STOP(SCHD) コマンドは現在の作業が静止するのを許します。 /DBRECOVERY または UPD DB STOP(ACCESS) コマンドを取り消すには、指定されたデータベースに対して /START または UPD DB START(ACCESS) コマンドを出します。

関連概念:

377 ページの『ポインター・セグメントの中のフィールドのフォーマットと用法』

INDICES= パラメーター

PCB の SENSEG ステートメントで INDICES= パラメーターを指定して、索引先セグメント・タイプ用の SSA を限定するための検索フィールドを含んでいる副次索引を指定できます。

INDICES= パラメーターを使用しても、PROCSEQ= パラメーターの有無によって PCB のために選択された処理シーケンスは変わりません。

INDICES= パラメーターは、1 次データベースが DEDB である場合はサポートされません。INDICES= パラメーターは、高速機能副次索引ではサポートされません。

以下の図とコード例は、INDICES= パラメーターの用法を示しています。

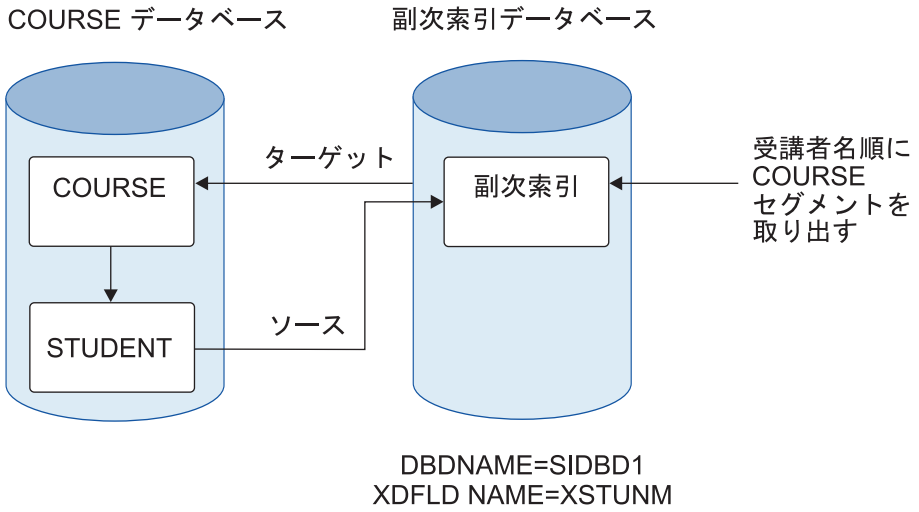


図 *187. INDICES=* パラメーターの最初の例でのデータベース

```
PCB
SENSEG NAME=COURSE, INDICES=SIDBD1
SENSEG NAME=STUDENT
```

図 *188. INDICES=* パラメーターの最初の例での *PCB*

```
GU COURSE COURSENM=12345&.XSTUNM=JONES
```

図 *189. INDICES=* パラメーターの最初の例で発行されるアプリケーション・プログラム呼び出し

前の GU 呼び出しが使用されると、IMS は番号が 12345 である COURSE セグメントを取得します。次に、IMS は XSTUNM が JONES である副次索引項目を取得します。 IMS は、副次索引の中のポインターが科目番号 12345 という COURSE セグメントを指しているかどうかを調べます。このセグメントを指していれば、IMS はアプリケーション・プログラムの入出力域に COURSE セグメントを返します。副次索引のポインターが科目番号 12345 という COURSE セグメントを指していない場合には、IMS は XSTUNM が JONES に等しい他の副次索引項目を調べて同じ比較を繰り返します。

XSTUNM が JONES に等しいすべての副次索引項目においてこの比較が成立しないと、このアプリケーション・プログラムにセグメントは返されません。このため、IMS が STUDENT セグメントを検索して、NAME が JONES である受講者を見つける必要はありません。ソース・セグメントとターゲット・セグメントが異なっている場合には、INDICES= パラメーターの使用が必要になるこの手法が役に立ちます。

以下の図は、INDICES パラメーターの 2 番目の例の場合のデータベースを示しています。

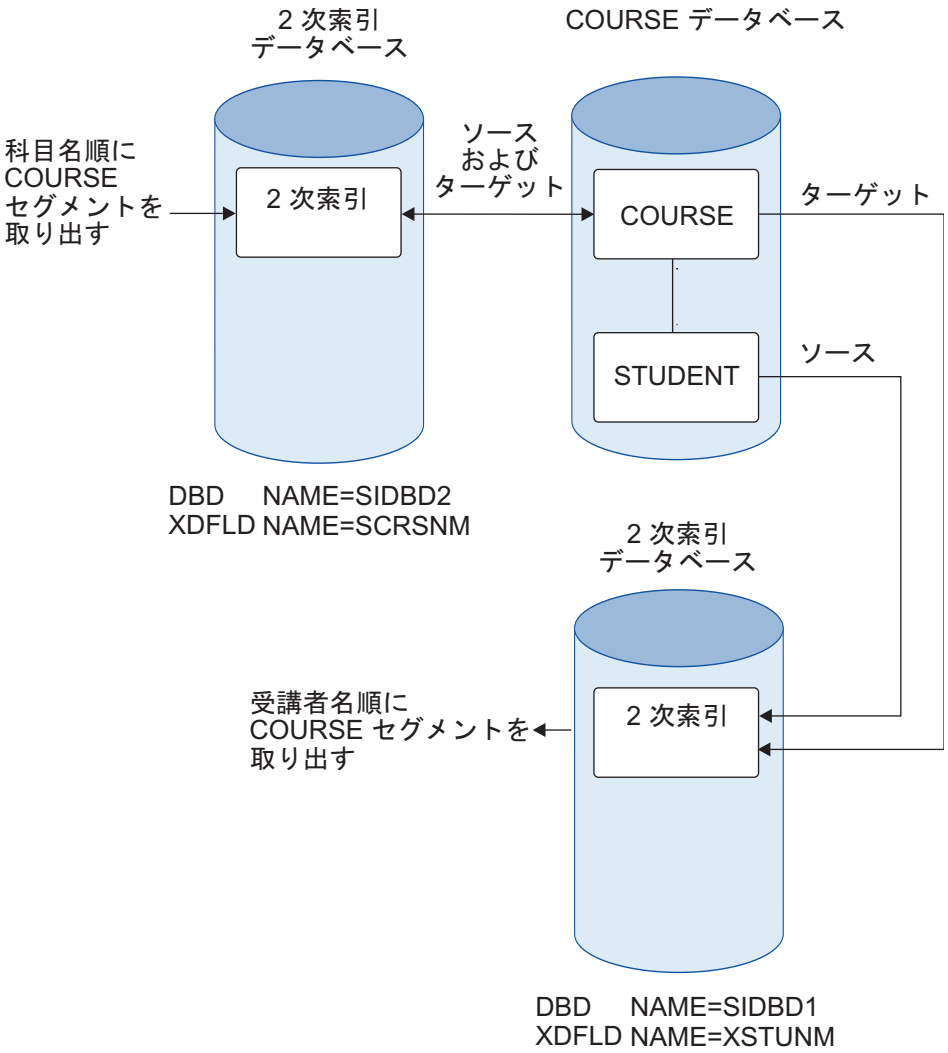


図 *190. INDICES=* パラメーターの *2* 番目の例でのデータベース

以下のコードは、PCB の例を示しています。

```
PCB       PROCSEQ=SIDBD2
SENSEG NAME=COURSE, INDICES=SIDBD1
SENSEG NAME=STUDENT
```

図 *191. INDICES=* パラメーターの *2* 番目の例での *PCB*

以下のコードは、アプリケーション・プログラム呼び出しの例を示しています。

```
GU COURSE SCRSNM=MATH&XSTUNM=JONES
```

図 *192. INDICES=* パラメーターの *2* 番目の例で発行される アプリケーション・プログラム呼び出し

比較のプロセスとパフォーマンス

前の GU 呼び出しから COURSENM=12345 を取り除くと、パフォーマンスに影響が出ます。IMS は、COURSE データベースから最初の COURSE セグメントを取り出し、XSTUNM が JONES である副次索引項目を取り出します。 IMS は、副次

索引の中のポインターが取り出した COURSE セグメントを指しているかどうかを調べます。このセグメントを指していれば、IMS はアプリケーション・プログラムの入出力域に COURSE セグメントを返します。副次索引のポインターがこの COURSE セグメントを指していない場合は、IMS は XSTUNM が JONES である他の副次索引項目があるかどうかを調べてこの比較を繰り返します。 XSTUNM が JONES である副次索引項目すべての比較結果が無効である場合、IMS は前回と同様に次の COURSE セグメントを取り出してから、副次索引項目を取り出し、この比較を繰り返します。すべての COURSE セグメントにおいてこの比較が成立しないと、アプリケーション・プログラムにセグメントは返されません。

ソース・セグメント呼び出しにおいて複数の副次索引を参照しようとしている場合にも、INDICES= パラメーターを使用できます。 398 ページの図 190の GU 呼び出しは、INDICES= パラメーターの別の例を示しています。

398 ページの図 192の図では、IMS が SIDBD2 の副次索引を使用して MATH の COURSE セグメントを取得しています。次に IMS は、SIDBD1 を使用して COURSE セグメントを取得します。この後、IMS は比較を行いこの 2 つの COURSE セグメントが同じものであるかを調べることができます。この 2 つのセグメントが同じであれば IMS はこの COURSE セグメントをアプリケーション・プログラムの入出力域に返します。この比較の結果、等しくない場合、IMS は COURSE セグメントを指す他の SIDBD1 ポインターを探し、比較を繰り返します。それでも等しい比較結果が出ない場合、IMS は COURSE セグメントを指す他の SIDBD2 ポインターについて調べて、SIDBD1 ポインターと比較して等しいものを探します。可能な比較をすべて試みても等しい結果が得られない場合には、このアプリケーション・プログラムにはセグメントは返されません。

注**:** この比較の処理によって、パフォーマンスがかなり低下する可能性があります。

関連概念:

『論理関係を伴う副次索引の使用』

論理関係を伴う副次索引の使用

高速機能副次索引を除き、副次索引を論理関係と一緒に使用できます。

論理関係を持つデータベースに副次索引を作成したりあるいは使用する場合には、次の制約があります。

- 論理子セグメントまたは論理子の従属セグメントをターゲット・セグメントとすることはできません。
- 論理子をソース・セグメントとして使用することはできません。ただし、論理子の従属セグメントをソース・セグメントとして使用することはできます。
- 論理データベースの中の連結セグメントまたはその従属セグメントをターゲット・セグメントとすることはできません。
- 論理関係を使用する場合、連結セグメントのための SSA で、索引付きフィールドによる修飾は使えません。ただし、連結セグメントのどの従属セグメントのための SSA でも、索引付きフィールドによって修飾することができます。

関連概念:

396 ページの『INDICES= パラメーター』

可変長セグメントによる副次索引の使用

可変長セグメントがソース・セグメントの場合には、そのオカレンスが挿入されたとき、ポインター・セグメントの検索フィールド、サブシーケンス・フィールド、または重複データ・フィールドで使用されるフィールドがこのオカレンスで指定されないならば、次のイベントが発生する場合があります。

- 実在しないソース・セグメント・データがポインター・セグメントの検索フィールドにおいて使用されている場合には、ポインター・セグメントは索引の中には置かれません。
- 実在しないソース・セグメント・データが、ポインター・セグメントのサブシーケンス・フィールドまたは重複データ・フィールドにおいて使用されている場合には、このポインター・セグメントが索引の中に置かれます。しかし、このサブシーケンス・フィールドまたは重複データ・フィールドには、以下の 3 つのゼロ表現のうちいずれか 1 つが入ります。

  P = X'0F'

  X = X'00'

  C = C'0'

  このいずれが使用されるかは、このソース・セグメント・フィールドを定義している DBD 内の FIELD ステートメントで何が指定されているかによって決まります。

副次索引を使用する場合の考慮事項

データベースで副次索引を使用する際には、いくつかの考慮事項があります。

副次索引に関して認識しておく必要がある考慮事項は次のとおりです

- データベースでソース・セグメントの挿入または削除が行われると、副次索引では索引ポインター・セグメントの挿入または削除が行われます。この保守は、この更新を行うアプリケーション・プログラムが副次索引を使用しているかどうかには関係なく、常時行われます。
- 索引ポインター・セグメントが REPL 呼び出しまたは DLET 呼び出しによって削除されたときには、削除された索引ポインター・セグメントを使用して PCB 位置が設定されていたすべての呼び出しに関係する位置はデータベース・レコードから失われます。
- ソース・セグメントの中のデータを置き換える場合、このデータが副次索引の検索フィールド、サブシーケンス・フィールドまたは重複データ・フィールドで使用されていると、この変更を反映するよう副次索引が以下に示すようにして更新されます。
  - ポインター・セグメントの重複データ・フィールドで使用されているデータがソース・セグメントにおいて置き換えられると、このポインター・セグメントが新しいデータによって更新されます。
  - ポインター・セグメントの検索フィールドまたはサブシーケンス・フィールドで使用されているデータがソース・セグメントにおいて置き換えられた場合には、このポインター・セグメントが新しいデータによって更新されます。さらに、ポインター・セグメントの検索フィールドまたはサブシーケンス・フィールドに変更が加えられると、ポインター・セグメントのキーが変

化するので、索引の中のこのポインター・セグメントの位置が変更されます。索引の更新のために、まず古いキーに従って決まっていた位置からこのポインター・セグメントが削除されます。次いでこのポインター・セグメントが、新しいキーに従って決められた位置に挿入されます。

- 副次索引を使用すると、処理オプションによってソース・セグメントの更新が可能になる場合、特定の PCB 内のすべての呼び出しに必要なストレージ要件が増えます。使用されている各副次索引データベースごとに必要となる追加のストレージは、6K から 10K のバイトの範囲です。この追加のストレージの一部分は、VSAM によって実記憶の中に固定されます。
- 副次索引の使用と、同じ結果を達成するための他の方法とを常に比較してください。例えば、ある HDAM または PHDAM データベースからルート・キー・シーケンスによる報告書を作成しようとしている場合、副次索引を用いてこれを行うことができます。しかし多くの場合、各ルート・セグメントへ順次にアクセスをするのはランダム操作です。各ルート・セグメントへのアクセスがランダム操作である場合、大きなデータベースを完全にスキャンするにはきわめて時間がかかります。このデータベースを物理的順序でスキャンし (副次索引を使用せず、GN 呼び出しを使用する)、次いでその結果をルート・セグメント・キーによってソートしてルートのキー・シーケンスによる最終報告書を作成した方が効率がよくなります。
- ターゲット・セグメントに対する呼び出しが、副次索引の検索フィールドによって修飾されているときに、副次索引を用いてこの索引付きデータベースを処理しない場合には、余分の入出力操作が必要になります。余分の入出力操作が必要になるのは、このターゲット・セグメントのオカレンスを調べるたびに、副次索引にアクセスしなければならないからです。副次索引の検索フィールドの中のデータは、ソース・セグメントの中のデータの複写であるので、ソース・セグメントを検査した方が同じ結果がより早く手に入ることになるのではないかということを判断しなければなりません。
- 2 次データ構造を使用する場合には、ターゲット・セグメントとこれが従属しているセグメント (ターゲット・セグメントの物理親) の挿入または削除はできません。

副次索引の定義の例

この例の副次索引は、受講者名に基づいて COURSE セグメントを取り出すために用いられます。

この例は、全機能データベース用で、シンボリック・ポインターではなく直接ポインターを使用します。

高速機能 DEDB データベースの副次索引の定義などの他の例については、副次索引のある例 (システム・ユーティリティー)を参照してください。

403 ページの図 193 は、EDUC データベースとその副次索引を示しています。以下のコード例は、これらのデータベースに必要な 2 つの DBD を示しています。

副次索引のポインター・セグメントには、検索フィールドの中に受講者名、サブシーケンス・フィールドの中にシステム関連フィールドが入っています。これらのフ

ィールドは共に STUDENT セグメントで定義されています。STUDENT セグメントがソース・セグメントです。 COURSE セグメントがターゲット・セグメントです。

コード例にある DBD は、副次索引が使用される場合にコーディングされるステートメントおよびパラメーターを強調したものです。(ステートメントあるいはパラメーターが省略されている場合には、副次索引が使用されているかどうかに関係なく、DBD の中のこのパラメーターが同じようにコーディングされることを意味しています。)

EDUC データベースの DBD

LCHILD ステートメントと XDFLD ステートメントは副次索引を定義するために用いられています。 これらのステートメントは、ターゲット・セグメントのための SEGM ステートメントの後でコーディングされます。

- LCHILD ステートメント。 LCHILD ステートメントは、副次索引の SEGM ステートメントの名前と副次索引データベースの名前を NAME= パラメーターで指定します。副次索引を使用する場合、PTR= パラメーターは常に PTR=INDX です。
- XDFLD ステートメント。 XDFLD ステートメントは、ポインター・セグメントの内容と副次索引において使用されるオプションを定義します。このステートメントは、DBD 入力デックにおいて、ポインター・セグメントを指している LCHILD ステートメントの後ろになければなりません。

403 ページの図 193 の例では、システム関連フィールド (/SX1) が SUBSEQ パラメーターで使用されています。このソース・セグメントのための SEGM の後の FIELD ステートメントでもシステム関連フィールドをコーディングしなければなりません。

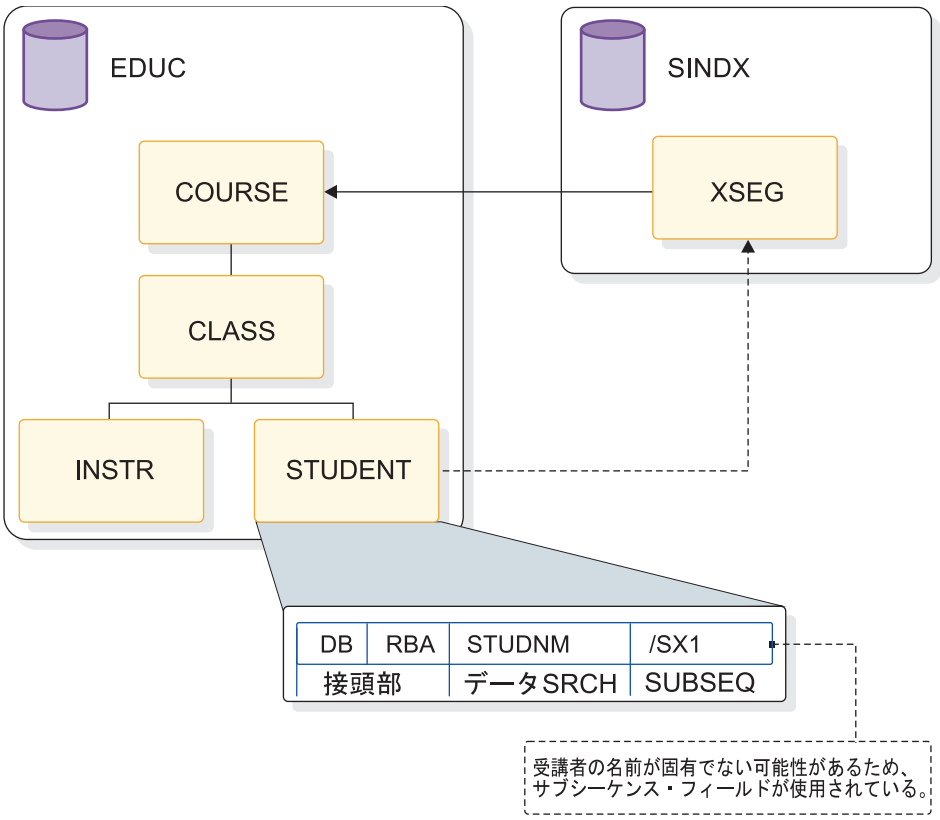


図 *193.* 副次索引の例でのデータベース

以下のコードは、図 193 の例の EDUC DBD を示しています。

```
DBD    NAME=EDUC,ACCESS=HDAM,...
SEGM   NAME=COURSE,...
FIELD  NAME=(COURSECD,...
LCHILD NAME=(XSEG,SINDX),PTR=INDX
XDFLD  NAME=XSTUDENT,SEGMENT=STUDENT,SRCH=STUDNM,SUBSEQ=/SX1
SEGM   NAME=CLASS,...
FIELD  NAME=(EDCTR,...
SEGM   NAME=INSTR,...
FILED  NAME=(INSTNO,...
SEGM   NAME=STUDENT,...
FIELD  NAME=SEQ,...
FIELD  NAME=STUDNM,BYTES=20,START=1
FIELD  NAME=/SX1

DBDGEN
FINISH
END
```

以下のコードは、図 193 の例の SINDX DBD を示しています。

```
DBD    NAME=SINDX,ACCESS=INDEX
SEGM   NAME=XSEG,...
FIELD  NAME=(XSEG,SEQ,U),BYTES=24,START=1
LCHILD NAME=(COURSE,EDUC),INDEX=XSTUDENT,PTR=SNGL

DBDGEN
FINISH
END
```

SINDX データベースの DBD

DBD ステートメント

DBD ステートメントは、副次索引データベースの名前を NAME= パラメーターで指定します。副次索引の DBD では、ACCESS= パラメーターは常に ACCESS=INDEX です。

SEGM ステートメント

NAME= パラメーターで何を使用するかをユーザーが選択します。この副次索引を別個のデータベースとして処理する際に、この値が使用されます。

FIELD ステートメント

NAME= パラメーターは副次索引のシーケンス・フィールドを指定します。この場合、シーケンス・フィールドは検索フィールドとサブシーケンス・フィールドのデータの両方、すなわち受講者名とシステム関連フィールド /SX1 で構成されます。 NAME= パラメーターで、何を選定するかをデータベース管理者が指定します。

LCHILD ステートメント

LCHILD ステートメントは NAME= パラメーターで、ターゲット・セグメント SEGM の名前とターゲット・データベースの名前を指定します。INDEX= パラメーターは、ターゲット・データベースの XDFLD ステートメント上にその名前をもっています。ポインター・セグメントにターゲット・セグメントを指す直接アドレス・ポインターが入っている場合には、PTR= パラメーターは PTR=SNGL です。このポインター・セグメントに、ターゲット・セグメントを指すシンボリック・ポインターが入っている場合には、PTR= パラメーターは PTR=SYMB です。

関連タスク:

835 ページの『全機能データベースへの副次索引の追加』

385 ページの『システム関連フィールドの使用によるキーの固有化』

関連資料:

副次索引のある例 (システム・ユーティリティー)

DBDGEN ステートメント (システム・ユーティリティー)

DEDB 区分副次索引

DEDB 区分副次索引を使用すると、DEDB 副次索引を複数の物理データベースに分散することができます。 1 つの高速機能副次索引データベースにつき、最大 101 個のユーザー区画データベースがサポートされます。

非常に大規模な副次索引は、複数の物理 HISAM データベースまたは SHISAM データベースに分割できます。ユーザー区画選択ルーチンは、どの区画に索引キーを割り当てるかを決定します。最初の区画名は、PCB 定義の中でユーザー区画グループ全体を表すために使用されます。

次の図は 5 つの索引データベースを表しており、それぞれの索引データベースは、すべてが 1 つの DEDB を指すユーザー指定のポインター範囲を備えています。

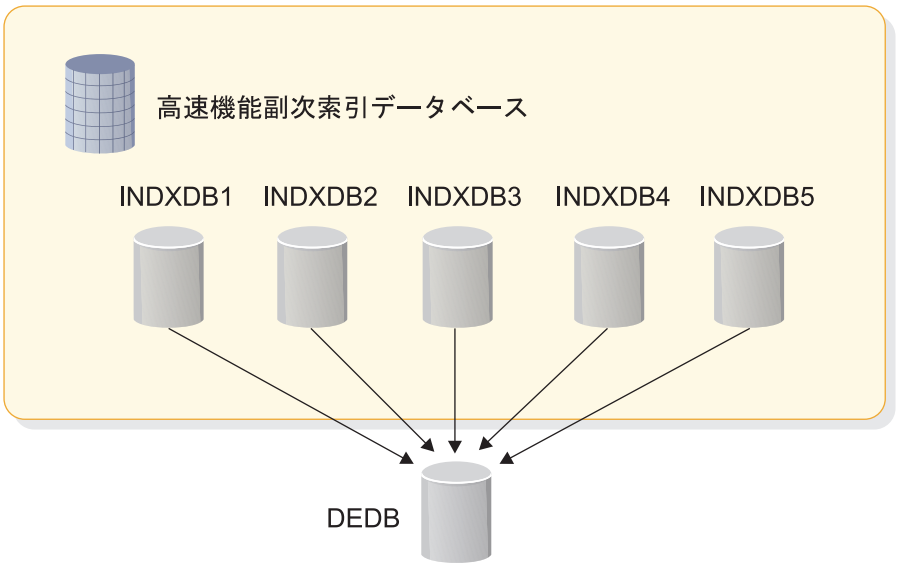


図 *194.* 複数の物理データベースにまたがった副次索引

それぞれの索引データベースには、1 つのキー範囲が含まれています。索引キーは、ユーザー区画選択出口ルーチンによって索引データベースに割り当てられます。

HISAM と SHISAM のどちらも使用できますが、ユーザー区画グループ内のすべての区画データベースは、すべてが HISAM として定義されているか、すべてが SHISAM として定義されている必要があります。索引内の各データベースは、同じ構造を属性を備えている必要があります。各データベースは、さまざまなキー範囲に存在する可能性がある多数の項目を収容するために、さまざまなサイズにすることができます。

HISAM 副次索引または SHISAM 副次索引のユーザー区分化を要求する場合は、それぞれ、1 次 DEDB データベース DBD の LCHILD ステートメントの NAME= パラメーターに複数の HISAM または HISAM 副次索引データベースを指定できます。

HISAM 副次索引は固定長副次索引セグメントを使用した固有キーと非固有キーをサポートし、SHISAM 副次索引は固定長副次索引セグメントを使用した固有キーだけをサポートしているため、HISAM 副次索引のユーザー区分化では、同じユーザー区画グループに SHISAM 副次索引データベースを含めることはできません。SHISAM 副次索引のユーザー区分化では、同じユーザー区画グループに HISAM 副次索引データベースを含めることはできません。 ユーザー区画グループに含まれるすべての区画データベースには、以下のように、区画データベースの副次索引 DBD 定義内で同じアクセス・タイプが定義されている必要があります。

- HISAM 副次索引データベースの場合は、ACCESS=(INDEX,VSAM)
- SHISAM 副次索引データベースの場合は、ACCESS=(INDEX,SHISAM)

LCHILD ステートメントの NAME= パラメーターで同じユーザー区画グループに HISAM 副次索引データベースと SHISAM 副次索引データベースが含まれている

場合、ACBGEN ユーティリティーは不整合を検出し、メッセージ DFS2293E を発行します。1 次 DEDB データベースおよびその副次索引データベースは、ACBLIB 内で削除されます。

ユーザー区画グループ内のすべての HISAM 副次索引区画データベースは、固有キーだけをサポートするために KSDS データ・セットとして定義されているか、非固有キーをサポートするために KSDS データ・セットと ESDS データ・セットの両方として定義されている必要があります。

HISAM 副次索引ユーザー区画グループに固有キーと非固有キーの両方の HISAM 2次索引データベースが含まれている場合、ACBGEN ユーティリティーは不整合を検出し、メッセージ DFS2294E を発行します。1 次 DEDB データベースおよびその副次索引データベースは、ACBLIB 内で削除されます。

HISAM 索引または SHISAM 索引の単一または複数の区画データベース

ユーザー区分化を要求するために XDFLD ステートメントで PSELRTN= パラメーターを指定する場合、PCB の PROCSEQD= パラメーターを使用して、IMS がデータベースの終わりを示す GB 状況コードを返す前に処理する区画データベースの数を制御することもできます。

PSELRTN= パラメーターのほかに、DEDB 区画選択出口ルーチンが呼び出されない場合、アプリケーションにデータベースの終わりを示す GB 状況コードを返す前に処理する論理 HISAM 副次索引データベースまたは SHISAM 副次索引データベースの区画データベースの数を示すために、PSELOPT=MULT|SNGL パラメーターを使用することができます。

「等しい」または「より大か等しい」の関係演算子を持つ SSA を使用した修飾 GU 呼び出しの場合、SSA 内の検索キーまたはその他のユーザー定義の区画選択基準に基づいて、選択する区画データベースを決定するために、DEDB 区画選択出口ルーチンが呼び出されます。

「等しい」または「より大か等しい」の関係演算子を持つ SSA を使用した修飾 GU/GN 呼び出しの場合、DEDB 区画選択出口ルーチンは呼び出されません。代わりに、ユーザー区画グループ内の最初のユーザー区画データベースの始めから、セグメントが検索されます。

非修飾 GN 呼び出し、または SSA を使用しない修飾 GN 呼び出しの場合、DEDB 区画選択出口ルーチンは呼び出されません。

「等しい」または「より大か等しい」の関係演算子を持つ SSA を使用した修飾 GN 呼び出しの場合、位置がまだ設定されていなければ、DEDB 区画選択出口ルーチンが呼び出されます。位置が既に設定されている場合、DEDB 区画選択出口ルーチンは呼び出されません。

次の表は、DEDB 区画選択出口ルーチンが呼び出されるかどうかについての DL/I GU 呼び出しと GN 呼び出しの条件を要約したものです。

表 62. *DL/I GU* 呼び出しおよび *GN* 呼び出しの条件

呼び出しタイプ	処置
「等しい」関係演算子を使用した修飾 GU GU COURSE(NAMEINDX =CHEMISTRY)	DEDB 区画選択出口ルーチンを呼び出してユーザー区画データベースを選択します。
「より大か等しい」関係演算子を使用した修飾 GU GU COURSE (NAMEINDX>=CHEMISTRY)	DEDB 区画選択出口ルーチンを呼び出してユーザー区画データベースを選択します。
「より小か等しい」関係演算子を使用した修飾 GU GU COURSE (NAMEINDX<=CHEMISTRY)	DEDB 区画選択出口ルーチンを呼び出しません。最初のユーザー区画データベースの始めからセグメントを検索します。
非修飾 GN GN	DEDB 区画選択出口ルーチンを呼び出しません。
SSA を使用しない修飾 GN GN COURSE	DEDB 区画選択出口ルーチンを呼び出しません。
「等しい」関係演算子を使用した修飾 GN GN COURSE(NAMEINDX=CHEMISTRY)	現在位置がまだ設定されていなければ、DEDB 区画選択出口ルーチンを呼び出してユーザー区画データベースを選択します。現在位置が既に設定されていれば、その位置より後のセグメントを検索します。
「より大か等しい」関係演算子を使用した修飾 GN GN COURSE(NAMEINDX >=CHEMISTRY)	現在位置がまだ設定されていなければ、DEDB 区画選択出口ルーチンを呼び出してユーザー区画データベースを選択します。現在位置が既に設定されていれば、その位置より後のセグメントを検索します。
「より小か等しい」関係演算子を使用した修飾 GN GN COURSE (NAMEINDX<=CHEMISTRY	DEDB 区画選択出口ルーチンを呼び出しません。最初のユーザー区画データベースの始めからセグメントを検索します。

ユーザー区分化が要求された場合、XDFLD ステートメントではデフォルトで PSELOPT=MULT|SNGL になります。しかし、PROCSEQD= パラメーターを使用した PCB ステートメントでは、PSELOPT= MULT|SNGL を明示的に指定する必要があります。PROCSEQD= パラメーターを使用した PCB ステートメントの PSELOPT= パラメーターにデフォルトはありません。PCB ステートメントの値が XDFLD ステートメントで指定された値をオーバーライドするからです。PROCSEQD= パラメーターを使用した PCB ステートメントの PSELOPT= パラメーターは、デフォルトで PSELOPT=MULT (XDFLD ステートメントのデフォルト値) となることはできません。その PSELOPT= パラメーターの値が、1 次 DEDB DBD内の XDFLD ステートメントで明示的に指定されるか暗黙にデフォルトで設定された値をオーバーライドするからです。

PSELOPT=MULT|SNGL パラメーターは、次の 2 つの場所で指定できます。

- PSB 内の PROCSEQD= パラメーターを使用した PCB ステートメントの PSELOPT=MULT|SNGL
- HISAM または SHISAM 副次索引が定義されている 1 次 DEDB DBD の XDFLD ステートメント

両方の場所で PSELOPT= パラメーターが指定された場合は、PCB ステートメントで指定されたパラメーターが、XDFLD ステートメントで指定されたパラメーターをオーバーライドします。

PROCSEQD= パラメーターを使用した PCB ステートメント、または XDFLD ステートメントで PSELOPT=MULT が定義された場合、これはユーザー区画グループ内の複数のユーザー区画データベースが処理されることを意味し、この場合、処理は DEDB 区画選択出口ルーチンによって選択されたユーザー区画データベースから始まり、1 次 DEDB データベース DBD の LCHILD ステートメントの NAME= パラメーターで定義されている最後のユーザー区画データベースまで順次に実行されます。 1 次 DEDB DBD の LCHILD ステートメントの NAME= パラメーターで定義されている最後のユーザー区画でデータベースの終わりに到達すると、GB 状況コードが返されます。PSELOPT=MULT が指定された場合は、1 つ以上のユーザー区画データベースが処理されます。

PROCSEQD= パラメーターを使用した PCB ステートメント、または XDFLD ステートメントで PSELOPT=SNGL が定義された場合、これはユーザー区画グループ内の、DEDB 区画選択出口ルーチンによって選択されたユーザー区画データベースが 1 つだけ処理されることを意味します。選択されたユーザー区画データベースでデータの終わりに到達すると、GB 状況コードが返されます。PSELOPT=SNGL が指定された場合は、1 つのユーザー区画データベースだけが処理されます。

制約事項*:*

DEDB 区分副次索引には、以下の制約事項があります。

- XLFLD ステートメントで PSELOPT= パラメーターが指定され、対応する LCHILD ステートメントで副次索引データベースが 1 つだけ指定された場合、DBDGEN ユーティリティーは MNOTE 8 とメッセージ XDFLD233 で終了します。
- XDFLD ステートメントで指定された PSELOPT= パラメーターが PSELOPT= MULT と PSELOPT= SNGL のどちらでもない場合、DBDGEN ユーティリティーは MNOTE 8 とメッセージ XDFLD234 で終了します。

PSELOPT=MULT を使用したユーザー区画の例

次の例は、4 つの副次索引データベースと PSELOPT=MULT を使用したユーザー区画グループを示しています。

```
LCHILD NAME=(NAMEXSEG,(NAMSXDB1,NAMSXDB2,NAMSXDB3,NAMSXDB4)),PTR=SYMB
XDFLD NAME=NAMEINDX,SRCH=COURNAME,PSELRTN=DBFPSE00,PSELOPT=MULT
```

この例では、ユーザー区画グループ内に 4 つの副次索引データベースが定義されています。DEDB 区画選択出口ルーチンは DBFPSE00 です。ユーザー区画選択オプションは PSELOPT=MULT として定義されています。

DEDB 区画選択出口ルーチンは、修飾 GU 呼び出しでユーザー区画データベース NAMSXDB2 を選択します。SSA がない後続の修飾 GN 呼び出しは、データの終わりに到達するまで、ユーザー区画データベース NAMSXDB2、NAMSXDB3、および NAMSXDB4 を処理します。

PSELOPT=MULT および開始ユーザー区画データベースの NAMSXDB2 を使用して、IMS は NAMSXDB2、NAMSXDB3、および NAMSXDB4 を 1 つの論理データベースとして処理します。IMS は、SSA 処理を使用しない修飾 GN 呼び出しで、論理データベース内の最後のユーザー区画 NAMSXDB4 でデータの終わりに到達したときに、GB 状況コードを返します。

PSELOPT=SNGL を使用したユーザー区画の例

次の例は、4 つの副次索引データベースと PSELOPT=SNGL を使用したユーザー区画グループを示しています。

```
LCHILD NAME=(NAMEXSEG,(NAMSXDB1,NAMSXDB2,NAMSXDB3,NAMSXDB4)),PTR=SYMB
XDFLD NAME=NAMEINDX,SRCH=COURNAME,PSELRTN=DBFPSE00,PSELOPT=SNGL
```

この例では、ユーザー区画グループ内に 4 つの副次索引データベースが定義されています。DEDB 区画選択出口ルーチンは DBFPSE00 です。ユーザー区画選択オプションは PSELOPT=SNGL として定義されています。

DEDB 区画選択出口ルーチンがユーザー区画データベース NAMSXDB2 を返し、PSELOPT=SNGL が定義されていると、IMS は 2 番目のユーザー区画データベースだけを処理します。IMS は、SSA を使用しない修飾 GN 呼び出しで、ユーザー区画データベース NAMSXDB2 のデータの終わりに到達したときに、アプリケーションに GB 状況コードを返します。

関連資料:

- 高速処理データベース区画選択出口ルーチン (DBFPSE00) (出口ルーチン)

高速機能副次索引の複数の索引項目

同じソース・セグメントにあるさまざまなフィールドから、複数の高速機能副次索引項目を作成できます。

同じソース・セグメントにある複数のフィールドから、1 つの副次索引に複数の検索フィールドを定義できます。副次索引の 1 つの項目を作成するために、1 つの検索フィールド (または 1 つの検索フィールド・セット) が使用されます。これらの検索フィールド (または複数の検索フィールド・セット) は、それぞれが同じサイズであることが必要です。

単一のソース・セグメントから複数の索引項目を作成するには、ターゲット・セグメントの SEGM ステートメントの下に複数の LCHILD/XDFLD ステートメント・ペアを定義し、LCHILD ステートメントに MULTISEG=YES を指定します。

次の DEDB では、OWNER セグメントに電話番号を表す 3 つのフィールドがあります。

```
DBD NAME=ACCTDB,ACCESS=DEDB,RMNAME=RMD4
AREA DD1=ACCT1,SIZE=1024,UOW=(100,10),
ROOT=(236,36)
SEGM NAME=ACCT,PARENT=0,BYTES=100
FIELD NAME=(ACCTNO,SEQ,U),BYTES=12,START=1
LCHILD NAME=(PHONKEY,PHONINDX),PTR=SYMB,MULTISEG=YES
XDFLD NAME=XPHON,SEGMENT=OWNER,SRCH=HOMEPHN
LCHILD NAME=(PHONKEY,PHONINDX),PTR=SYMB,MULTISEG=YES
XDFLD NAME=XPHON,SEGMENT=OWNER,SRCH=WORKPHN
LCHILD NAME=(PHONKEY,PHONINDX),PTR=SYMB,MULTISEG=YES
```

```
XDFLD NAME=XPHON,SEGMENT=OWNER,SRCH=CELLPHN
SEGM NAME=OWNER,BYTES=300,PARENT=ACCT
FIELD NAME=OWNNAME,BYTES=40,START=1
FIELD NAME=HOMEPHN,BYTES=10,START=41
FIELD NAME=WORKPHN,BYTES=10,START=51
FIELD NAME=CELLPHN,BYTES=10,START=61
DBDGEN
```

この例の副次索引 DBD は、以下のとおりです。

```
DBDSX DBD NAME=PHONINDX,ACCESS=(INDEX,VSAM),FPINDEX=YES
DATASET DD1=PHONKSDS,OVFLW=PHONOVFL
SEGM NAME=PHONSEG,PARENT=0,BYTES=22
FIELD NAME=(PHONEKEY,SEQ,U),BYTES=10,START=1
LCHILD NAME=(OWNER,ACCTDB),INDEX=XPHON,PTR=SYMB
DBDGEN
```

HALDB 区分副次索引に関する考慮事項

HALDB データベースの副次索引は、HALDB 区分副次索引 (PSINDEX) でなければなりません。

PSINDEX は、1 つ以上の区画を持つことができます。副次索引の区画用のキー範囲は、そのターゲット・データベースのキー範囲と一致するとは思われません。副次索引用の区画の選択は、副次索引のキーに基づきます。ターゲット・データベース用の区画の選択は、ターゲット・データベースのキーに基づきます。通常、これらのキーは関連付けられていません。場合によっては、区画選択出口ルーチンを使って、メイン・データベースの同じ境界に沿って副次索引を区分化します。区画の選択は常にルート・キーに依存するので、適切なキーが必要です。区画選択出口ルーチンは、正しい区画を選択するためにユーザーが指定するルート・キーの一部を使用します。

PSINDEX 内の区画を初期設定するには、HALDB 区画データ・セット初期設定ユーティリティー (DFSUPNT0) を使用してください。 DFSUPNT0 は、PSINDEX 用のリカバリー・ポイントを自動的に生成します。ユーザーが自分の PSINDEX 区画を削除し、再定義し、その後でその区画の PINIT フラグをオフにすると、リカバリー・ポイントは作成されません。

以上に加えて、HALDB 区分副次索引について他の制限および要件があります。

- シンボリック・ポインターはサポートされていません。
- 共用副次索引はサポートされていません。
- 副次索引は固有キーを持たなければなりません。
- /SX フィールドおよび /CK フィールドを使用して一意性を与えることができます。
- ターゲット・データベースを再編成しても、その副次索引を再作成する必要はありません。 HALDB データベースは、間接リスト・データ・セット (ILDS) を使用して、PSINDEX とターゲット・セグメント間のポインターを維持します。

関連概念:

764 ページの『HALDB 自己回復ポインター処理』

第 17 章 データベースのバージョン管理方式

データベースのバージョン管理方式および IMS カタログが IMS システムで使用可能であるとき、複数の異なるバージョンのデータベースを維持できます。古いバージョンのデータベースを変更しなくても、アプリケーション・プログラムはアクセスできます。一方、新しいバージョンのデータベースを変更して、新しいアプリケーション・プログラムの要件をサポートできます。

データベースのバージョン管理の概要

データベースの新しいバージョンを作成するには、DFSDFxxx PROCLIB メンバーで DBVERSION=Y と指定することによって、データベースのバージョン管理を使用可能にする必要があります。

データベースのバージョン管理を使用可能にした後、新しいバージョンに適用するバージョン番号と構造的な変更によってデータベースのデータベース定義 (DBD) を更新することにより、新しいバージョンのデータベースを作成します。バージョン番号は、DBVER キーワードで指定し、新しいバージョンごとに 1 ずつ増やす必要があります。

新しいバージョンの DBD メンバーおよび ACB メンバーが生成され、生成された ACB メンバーが ACB ライブラリーでアクティブになった後、データベースの古いバージョンは IMS カタログ内のデータベースの DBD レコード内にのみ存在します。アプリケーション・プログラムが、ACB ライブラリー内のアクティブ・バージョンではないデータベースのバージョンのデータを要求すると、IMS は、要求されたバージョンを IMS カタログから取得します。

重要: IMS カタログ内の DBD レコードに特定の DBD バージョンの複数のインスタンスが含まれている場合、IMS は、ACB 生成タイム・スタンプに基づいて、最新のインスタンスを使用してデータを返します。

アプリケーション・プログラムが要求するデータベース・バージョンは、PCB の DBVER= パラメーターで指定するか、または実行時に VERSION 副次機能を指定して INIT DL/I 呼び出しを発行することによって指定できます。

アプリケーション・プログラムでバージョン番号が指定されない場合、IMS はデフォルトで、ACB ライブラリー内で現在アクティブな DBD に基づいてデータを返します。通常、ACB ライブラリーでアクティブなバージョンは、DASD 上のデータベースの現行の物理的構造を示しています。

DBD のバージョン 0 がまだ IMS カタログ内にある場合は、IMS が DBD のバージョン 0 を使用してデータを返すようにデフォルトを変更することができます。DBD のバージョン 0 は、DBVER パラメーターが含まれていない、IMS カタログ内の DBD の最新のインスタンスです。DBD のバージョン 0 が IMS カタログ内に存在しない場合は、DBD のアクティブ・バージョンのみを DBD のデフォルト・バージョンとして使用できます。

デフォルトをシステム・レベルおよびプログラム仕様ブロック (PSB) レベルで設定することができます。PSB レベルのデフォルトは、システム・レベルのデフォルトに優先します。

システム・レベルのデフォルトは、DFSDFxxx メンバーの DBLEVEL= キーワードによって設定されます。 PSB レベルのデフォルトは、PSB 定義の PSBGEN ステートメントの DBLEVEL= キーワードによって設定されます。

アプリケーション・プログラムが IMS カタログ内に存在しないデータベース・バージョンを要求した場合、IMS はアプリケーション・プログラムに状況コードを返します。

データベースのバージョン管理が使用可能でない場合、IMS は PSB マクロや DBD マクロで指定されたバージョン番号を無視して、単に物理データベース構造によって定義されている形式でデータを返します。データベースのバージョン管理が使用可能でない場合に、アプリケーション・プログラムが PCB の DBVER= パラメーターでバージョン番号を指定すると、IMS は、アプリケーション・プログラムの設計方法に応じて、3303 アベンドでアプリケーション・プログラムを強制終了するか、またはアプリケーション・プログラムに BA DL/I 状況コードを返します。

推奨事項: 新しいバージョンのデータベースを作成したらすぐに、アプリケーション・プログラムがまだ以前のバージョンのデータベースにアクセスできることを確認してください。データベースのバージョン管理は、データベースの構造に対する特定の変更のみをサポートします。新しい変更がデータベースのバージョン管理によってサポートされることを確認する唯一の方法は、アプリケーション・プログラムをスケジュールして以前のバージョンのデータベースにアクセスする方法です。最新のバージョンのデータベースでは、データをアプリケーション・プログラムに返す前に再フォーマットする必要がないため、最新バージョンにアクセスすることで、サポートされない変更を検出することはできません。

データベースのバージョン管理では、データベース定義に対する以下の変更がサポートされます。

- セグメント長の増加。
- セグメント末尾にある既存の未定義スペースへの新規フィールドの追加。

重要: IMS に対して定義されていない既存スペース内のセグメントに新規フィールドを追加する前に、既存のアプリケーション・プログラムが、フリー・スペースの初期化、変更、またはその他の方法による使用を行わないことを確認する必要があります。

データベースのバージョン管理は、以下のデータベース・タイプでサポートされます。

- DEDB
- HDAM
- HIDAM
- PHDAM
- PHIDAM

IMS データベースのバージョン管理では、論理関係を持つデータベースと副次索引を持つデータベースの両方がサポートされます。ただし、2 つのデータベース間の関係に関与するセグメント、または同じデータベース内の別のセグメントとペアになっているセグメントは、DBD の新規バージョンで変更できません。関係に関与しない、データベース内のセグメントのみを変更することができます。

関連概念:

47 ページの『第 5 章 IMS カタログの概要』

関連資料:

- IMS PROCLIB データ・セットの DFSDFxxx メンバー (システム定義)
- データベース記述 (DBD) 生成ユーティリティー (システム・ユーティリティー)
- プログラム仕様ブロック (PSB) 生成ユーティリティー (システム・ユーティリティー)

データベース・バージョン管理に対する IMS カタログ・サポート

データベースのバージョン管理には IMS カタログが必要です。

IMS カタログでアクティブな DBD として指定されている DBD バージョンを除き、データベースの以前の DBD バージョンはすべて、データベースの DBD レコードの IMS カタログ内にのみ存在します。

アプリケーション・プログラムが以前の DBD バージョンを必要とするデータベース呼び出しを行うと、IMS では、以前のバージョンの DBD とアクティブな DBD の両方を使用して、データベースから取得したデータを変換します。アプリケーション・プログラムが以前のバージョンの DBD を必要としないデータベース呼び出しを行うと、IMS ではアクティブな DBD のみを使用してデータを返します。

重要: 以前のバージョンの DBD がカタログから削除されると、データベースのその定義は失われ、そのバージョンの DBD を使用するアプリケーション・プログラムはデータベースにアクセスできなくなります。さらに、IMS カタログ全体が使用できなくなると、IMS ではデータベースの以前のどのバージョンにもアクセスできなくなります。IMS カタログでアクティブな DBD バージョンを使用するアプリケーション・プログラムのみが、データベースにアクセスできます。

IMS カタログのバックアップとリカバリーは、他のデータベースに使用されるものと同じバックアップ手順とリカバリー手順を使用して行います。ただし、データベースのバージョン管理を使用する場合、複数の DBD バージョン、PSB、および ACB メンバーが処理対象となるため、IMS カタログのリカバリーはより複雑になり、所要時間も長引く可能性があります。

関連概念:

47 ページの『第 5 章 IMS カタログの概要』

データベースのバージョン管理方式によってサポートされるデータベースの変更

複数のバージョンのデータベースを維持しているとき、新しいバージョンのデータベースに対して特定の構造上の変更のみを行うことができます。

データベースのバージョン管理では、データベース定義に対する以下の変更がサポートされます。

- セグメント長の増加。
- セグメント末尾にある既存の未定義スペースへの新規フィールドの追加。

重要: IMS に対して定義されていない既存スペース内のセグメントに新規フィールドを追加する前に、既存のアプリケーション・プログラムが、フリー・スペースの初期化、変更、またはその他の方法による使用を行わないことを確認する必要があります。

データベースのバージョン管理では、以下の変更はサポートされません。

- 可変長セグメントの長さの増加
- 既存のフィールドの開始位置の変更
- 既存のフィールドの長さの変更

以下の一連のセグメント定義は、データベースのバージョン管理によってサポートされる変更のタイプの例を示しています。

以下の例の前提として、バージョン管理を使用可能にする前に、既存のデータベースで次のセグメントが定義されているものとします。これは、バージョン 0 のデータベースです。

```
----------------------------
| FLD1  |  FLD2  |  space |
----------------------------
```

データベースのバージョン管理が使用可能になった後、セグメント長を長くするよう、データベースの定義が変更されたため、バージョン 1 のデータベースでは、セグメントの最後に未定義のスペースが追加されています。

```
-------------------------------------
| FLD1  |  FLD2  |  space |  space |
-------------------------------------
```

バージョン 2 のデータベースでは、セグメントの最後の追加スペースに新しいフィールド FLD3 が定義され、以下のようなセグメントになっています。

重要: 次の例に示すような変更を行う前に、既存のアプリケーション・プログラムのいずれもそのフリー・スペースに対する処理 (例えば、REPL 呼び出し中にその初期化を行ったり、いずれかの方法でそれを使用したりするなど) を何も行っていないことを、ご使用のアプリケーション・グループで確認する必要があります。そのような処理を行っている場合は、セグメントの長さを拡張して、既存のスペースの後ろに新規フィールドを (次の例の後に示すバージョン 3 の例におけるフィールド FLD4 と FLD5 のように) 追加してください。

```
-----------------------------------
| FLD1  |  FLD2  |  space |  FLD3  |
-----------------------------------
```

バージョン 3 のデータベースでは、セグメント長がさらに長くなり、FLD3 の後ろに、追加のフィールド FLD4 と FLD5、および未定義のスペースが追加されています。

```
-------------------------------------------------------------
| FLD1  |  FLD2  |  space |  FLD3  |  FLD4  |  FLD5  |  space |
-------------------------------------------------------------
```

関連資料:

データベース記述 (DBD) 生成ユーティリティー (システム・ユーティリティー)

データベースのバージョン管理、既存のフリー・スペース、および新規フィールド

IMS におけるセグメントの定義では、セグメントの長さに含まれるバイトのうち、そのセグメントで定義されているフィールドによって使用されていないバイトはすべてフリー・スペースとみなされます。

フリー・スペースは IMS からは未使用のように見えますが、IMS に認識されていなくても、アプリケーション・プログラムはそのフリー・スペースを使用、あるいは変更することができます。実際に、セグメント内のフリー・スペースは、COBOL コピーブックなどの、アプリケーション・プログラム・コードによってマップされるフィールドやデータ用に使用される場合があります。

データベースのバージョン管理では、セグメント末尾にある既存のフリー・スペースへの新規フィールドの追加がサポートされますが、そのフリー・スペースに新規フィールドを定義する前に、既存のアプリケーション・プログラムによってそのフリー・スペースが使用されていないことを確認する必要があります。

以下の例では、既存のアプリケーション・プログラムと、既存のフリー・スペースに追加された新規フィールドとの間で発生する可能性のある競合を示しています。

いずれかのアプリケーション・プログラムによってそのフリー・スペースが使用されている場合は、セグメントの長さを拡張して、既存のフリー・スペースの後ろに新規フィールドを追加するか、またはそのフリー・スペースを使用しないように既存のアプリケーション・プログラムを変更することができます。

以下の例では、潜在的な問題とその回避方法を示します。

既存のアプリケーション・プログラムが既存のフリー・スペース内の新規フィールドと競合する可能性のある状況

以下の例では、データベースの基本バージョンに 40 バイトのセグメントが含まれており、そのセグメントの末尾には 10 バイトのフリー・スペースがあります。これは、既存のアプリケーション・プログラム PGMA から確認できます。

```
10 bytes  20 bytes           10-byte undefined space at the end
---------------------------------------
|  123    |  Maple Ave.      |        |
---------------------------------------
```

同じデータベースのバージョン 1 ではセグメント・サイズが 50 バイトに拡張され、セグメント末尾には新規の 20 バイト・フィールドが定義されています。このような変更はデータベースのバージョン管理でサポートされます。ただし、セグメント末尾の 10 バイトのスペースを初期化するコードが既存のアプリケーション・プログラムに含まれている場合、このデータベースのバージョン 1 の新規フィールドに挿入されたデータが、既存のアプリケーション・プログラムによって破壊されるおそれがあります。

次の例は、セグメントが 10 バイト拡張され、新規の 20 バイト・フィールドが定義され、アプリケーション・プログラム PGMB がデータを追加してセグメントを更新した後に、データベースのバージョン 1 でこのセグメントがどのような状態になるかを示しています。

```
10 bytes  20 bytes           20 bytes
-------------------------------------------------
|  123    |  Maple Ave.      | San Francisco     |
-------------------------------------------------
```

この後に、PGMA は REPL 呼び出しを行います。さらに、PGMA にはセグメント末尾の 10 バイトのフリー・スペースを初期化するコードが含まれているため、次の例に示すように、PGMA は PGMB によって追加されたデータを上書きします。

```
10 bytes  20 bytes           20 bytes
-------------------------------------------------
|  456    |  Orchard Ave.    |         cisco     |
-------------------------------------------------
```

PGMA がどのようにコーディングされているかを考慮すると、PGMA が変更されない限り、既存のスペースに新規フィールドをコーディングしてはなりません。PGMA を変更する代わりに、セグメントの長さを増加して、DBD の新規バージョン内の既存のフリー・スペースの後ろに新規フィールドを追加することができます。

既存のフリー・スペースの後の新規スペースに新規フィールドを定義する場合の例

以下は、PGMA から見たアドレス・セグメントです。

```
10 bytes  20 bytes           10-byte undefined space at the end
---------------------------------------
|  123    |  Maple Ave.      |        |
---------------------------------------
```

このセグメントは 20 バイト拡張され、新規の 20 バイト・フィールドが定義されています。 PGMB はデータを追加してこのセグメントを更新しますが、元の 10 バイトのフリー・スペースに対しては何も行われません。

```
10 bytes  20 bytes           10 bytes  20 bytes
-------------------------------------------------------------
|  123    |  Maple Ave.      |         | San Francisco     |
-------------------------------------------------------------
```

PGMA は REPL 呼び出しを行って、10 バイトのフリー・スペースを初期化します。ただし、新規の 20 バイト・フィールドに対しては何も行われません。

```
10 bytes  20 bytes            10 bytes  20 bytes
------------------------------------------------------------
|  789    |  Walnut Ave.      |         | San Francisco     |
------------------------------------------------------------
```

データベースのバージョン管理のシステム・デフォルト

アプリケーション・プログラムが、バージョン管理されているデータベースの特定バージョンを指定しない場合、デフォルトで、IMS は IMS カタログで現在アクティブなデータベース定義 (DBD) のバージョンを使用して、そのアプリケーション・プログラムにデータを戻します。このデフォルトを変更して、IMS が代わりに DBD のバージョン 0 を使用してデータを返すようにすることができます。

IMS が使用する DBD バージョンの IMS システム・デフォルトは、IMS PROCLIB データ・セットの DFSDF*xxx* メンバーのデータベース・セクション内の DBLEVEL= パラメーターによって制御されます。

DFSDF*xxx* メンバーで指定される IMS システム・デフォルトは、プログラム仕様ブロック (PSB) の定義で DBLEVEL 値を指定することによって、またはプログラム連絡ブロック (PCB) の定義か、アプリケーション・プログラムによって発行される DL/I INIT VERSION 呼び出しのいずれかで特定のバージョン番号を指定することによってオーバーライドすることができます。

既存のアプリケーション・プログラムが変更されないようにするための DBLEVEL=BASE の指定

新規バージョンのデータベースが導入されても、多数のアプリケーション・プログラムが引き続きバージョン 0 を使用する場合は、システム・デフォルトを変更して DFSDFxxx メンバーに DBLEVEL=BASE を指定することで、DBD のバージョン 0 が返されるようにします。システム・デフォルトを DBLEVEL=BASE に変更しない場合、現在アクティブなバージョン以外のバージョンを使用するアプリケーション・プログラムでは、以下を変更する必要があります。

- DBLEVEL=BASE を指定してシステム・デフォルトをオーバーライドするように PSB を変更する
- 必要なバージョンを DBVER パラメーターに指定するように PCB を変更する
- 実行時に INIT VERSION 呼び出しを発行するようにコードを変更する

DBLEVEL=BASE の意味

システム・レベルで DBLEVEL=BASE が指定されている場合、バージョン管理されている各データベースの DBD のバージョン 0 インスタンスが、IMS カタログ内の DBD レコードに存在していなければなりません。DBLEVEL=BASE が指定されている場合に、バージョンを指定していないアプリケーション・プログラムが既にバージョン 0 がなくなっているデータベースにアクセスしようとすると、そのアプリケーション・プログラムは異常終了します。

DBLEVEL=BASE が指定されているときに、アプリケーション・プログラム、ツール、または製品で物理 IMS データベースの現在の構造が必要になった場合は、そ

のアプリケーション・プログラム、ツール、または製品が IMS データベースへのアクセスに使用する PSB に DBLEVEL=CURR を指定して、システム・デフォルトをオーバーライドします。

ご使用のほとんどのアプリケーション・プログラムが現在アクティブなデータベースを使用する場合は、**DBLEVEL=CURR** を指定します。

ほとんどのアプリケーション・プログラムが現行データベース構造を使用するように変更されている一方で、少数のアプリケーション・プログラムが前のデータベース・バージョンの構造を引き続き必要としている場合は、DBLEVEL=CURR を使用します。データベースの旧バージョンを必要とするアプリケーション・プログラムに対しては、必要なデータベース・バージョン番号を PCB の DBVER= パラメーターに指定できます。

以前のバージョンのデータ変換処理

現在アクティブな DBD バージョン以外の DBD バージョンによって定義されたフォーマットのデータがアプリケーション・プログラムで必要となった場合、IMS では、必ず現行フォーマットのデータをアプリケーション・プログラムで必要なフォーマットに変換する必要があります。この結果、DBLEVEL=BASE を指定して実行する場合、または現在アクティブな DBD バージョン以外のデータベース・バージョンを使用するアプリケーション・プログラムが多数ある場合は、CPU 使用率が増加する可能性があります。

アプリケーション・プログラムが現在アクティブな DBD バージョンを使用する場合、IMS ではデータ変換は行いません。データは DASD に保管されているフォーマットでアプリケーション・プログラムに返されるためです。

関連タスク:

『データベースのバージョン管理の実装』

関連資料:

IMS PROCLIB データ・セットの DFSDFxxx メンバー (システム定義)

プログラム仕様ブロック (PSB) 生成ユーティリティー (システム・ユーティリティー)

データベースのバージョン管理の実装

複数のバージョンのデータベースを維持およびアクセスするには、データベースのバージョン管理を使用可能にし、新規バージョンのデータベースを定義し、必要な DBD、PSB、および ACB の各メンバーを生成する必要があります。

前提条件**:** データベースのバージョン管理をサポートするには、ご使用の IMS システムで IMS カタログを使用可能にする必要があります。IMS カタログの定義と調整 (システム定義)を参照してください。

データベースのバージョン管理を使用可能にして実装するには、以下の手順を実行します。

1. IMS.PROCLIB データ・セットの DFSDFxxx メンバーの DATABASE セクションで DBVERSION=Y と指定して、データベースのバージョン管理を使用可能にします。
2. オプション: DFSDFxxx メンバーの DBLEVEL= パラメーターで、アプリケーション・プログラムによってバージョンが指定されていないときに IMS が使用するデータベース・バージョンのシステム・デフォルトを設定します。 有効な値は CURR および BASE です。CURR では、IMS カタログで現在アクティブなバージョンを使用してデータを返します。これはデフォルトです。
   BASE は、バージョン 0 のデータベースを使用してデータを返します。バージョン 0 は、データベースのバージョン管理を実装する前に存在していたデータベースのバージョンです。
3. DBD 生成ユーティリティーへの入力マクロにデータベースに対する変更をコーディングします。
   a. DBD ステートメントの DBVER キーワードに新しいバージョン番号を指定します。 バージョン番号は整数で、新しいバージョンを定義するたびに値を 1 ずつ増やす必要があります。
   b. セグメントの長さを変更する場合は、SEGM ステートメントの BYTES キーワードに新しい長さをコーディングします。
   c. 新しいフィールドを定義する場合は、FIELD ステートメントをコーディングします。 既存のフィールドの開始オフセットおよび長さを変更することはできません。
4. アプリケーション・プログラムが必要とするバージョンを PSB 生成ユーティリティーの入力マクロにコーディングします。

   注**:** 1 つの PSB 内の複数のデータベース PCB が同じデータベースを参照する場合、すべての PCB で同じバージョンの DBD を指定する必要があります。
   a. オプションで、1 つ以上の PSB ステートメントの DBLEVEL キーワードに、アプリケーション・プログラムに返すデフォルトのバージョン・レベルを指定します。
   b. 必要に応じて、PCB ステートメントの DBVER キーワードに、アプリケーション・プログラムに返す特定のバージョンを指定します。
   c. PSB 生成ユーティリティーを実行して PCB を生成します。
5. オンライン・データベースにデータベースの変更を適用するオンライン変更機能を使用しない場合は、既存のデータベース構造の DBD メンバーを使用してデータベースをオフラインでアンロードします。
6. DBD 生成ユーティリティーを実行して、新しいデータベース構造の DBD メンバーを生成します。
7. オンライン・データベースにデータベースの変更を適用するオンライン変更機能を使用しない場合は、新しいデータベース構造の DBD メンバーを使用してデータベースを再ロードします。
8. ACB Generation and Catalog Populate ユーティリティー (DFS3UACB) を実行することにより、新しいデータベース・バージョンのACB メンバーを生成して IMS カタログを更新します。

   変更するデータベースに副次索引がある場合、または別のデータベースと論理的な関連付けがある場合は、ACB 保守ユーティリティー制御ステートメント

で、変更するデータベースの BUILD DBD ステートメントのほかに、各副次索引データベースおよび論理的に関連付けられている各データベースの BUILD DBD ステートメントも指定する必要があります。

9. オンライン・データベースにデータベースの変更を適用するオンライン変更機能を使用する場合は、変更機能を開始します。
10. 新しいデータベース構造の ACB メンバー、およびそのデータベースを使用するアプリケーション・プログラムをアクティブにします。オンライン・システムで、オンライン変更 (OLC) 機能を使用します。
11. データベースを開始します。
12. 重要: データベースの以前のバージョンにアクセスするアプリケーション・プログラムを実行して、データベースに対して行った変更がデータベース管理でサポートされていることを確認します。 IMS は、古いバージョンを必要とする最初のアプリケーション・プログラムが実行のためにスケジュールされるまで、サポートされないデータベース変更を検出しません。

関連概念:

417 ページの『データベースのバージョン管理のシステム・デフォルト』

関連資料:

- IMS PROCLIB データ・セットの DFSDFxxx メンバー (システム定義)
- データベース記述 (DBD) 生成ユーティリティー (システム・ユーティリティー)
- プログラム仕様ブロック (PSB) 生成ユーティリティー (システム・ユーティリティー)
- アプリケーション制御ブロック保守ユーティリティー (システム・ユーティリティー)

論理関係、副次索引、およびデータベースのバージョン管理

IMS データベースのバージョン管理では、論理関係を持つデータベースと副次索引を持つデータベースの両方がサポートされます。ただし、2 つのデータベース間の関係に関与するセグメント、または同じデータベース内の別のセグメントとペアになっているセグメントが DBD の新規バージョンで変更されない場合に限ります。

関係に関与しない、データベース内のセグメントのみを変更することができます。

副次索引を持つデータベースはデータベースのバージョン管理でサポートされます。ただし、索引付きデータベースの DBD の基本バージョンで副次索引関係が定義されていて、それらの関係がそのデータベースの以降のバージョンで変更されない場合に限ります。副次索引のセグメントとフィールドの関係は、索引付きデータベースの DBD 内で XDFLD ステートメントによって定義されます。XDFLD ステートメント、および XDFLD ステートメントによって参照される索引付きデータベース内のフィールドはいずれも、その索引付きデータベースの以降のバージョンで変更せずに維持される必要があります。

別のデータベースと直接論理関係を持つデータベースの新しいバージョンを作成する場合は、同時に両方のデータベースの新しい ACB メンバーを生成する必要があります。

両方のデータベースのデータベース定義 (DBD) を ACB 保守ユーティリティーの BUILD ステートメントで明示的に指定する必要があります。

変更しているデータベースに間接的にしか関与していないデータベースの DBD は、BUILD ステートメントで指定する必要はありません。

例えば、データベース A の新しいバージョンを作成する場合に、データベース A がデータベース B に関連しており、データベース B がデータベース C に関連している場合は、データベース A と B の DBD のみを BUILD ステートメントで指定する必要があります。

別の例で説明すると、前のシナリオでデータベース B の新しいバージョンを作成する場合は、A、B、C の 3 つのデータベースの DBD をすべて BUILD ステートメントで指定する必要があります。

関連概念:

47 ページの『第 5 章 IMS カタログの概要』

関連資料:


アプリケーション制御ブロック保守ユーティリティー (システム・ユーティリティー)

第 18 章 オプション・データベース機能

IMS データベースでは、別のトピックで説明している論理関係と副次索引に加えて、さまざまなオプションの機能をサポートしています。これらの機能は、使用しているデータベースのタイプおよびインストール済み環境のニーズに応じて、選択的にインプリメントできます。

このトピックで説明するオプションの機能は、GSAM、MSDB、HSAM、および SHSAM の各データベースには適用されません。 DEDB に適用されるのは、可変長セグメント機能、セグメント編集/圧縮出口ルーチン、およびデータ・キャプチャー出口ルーチンのみです。

関連概念:

35 ページの『第 3 回の設計レビュー』

21 ページの『オプション・データベース機能の概説』

507 ページの『SB によるデータのバッファリング方法』

可変長セグメント

可変長セグメントとは、いくつかのセグメント・タイプのオカレンスの中で単に長さが変動するセグメントのことです。

1 つのデータベースに、可変長セグメント・タイプと固定長セグメント・タイプの両方を収容することができます。

可変長セグメントをサポートするデータベース・タイプは次のとおりです。

- HISAM
- HDAM
- PHIDAM
- HIDAM
- PHDAM
- DEDB

関連概念:

146 ページの『セグメントの置き換え』

関連タスク:

806 ページの『可変長セグメントの追加または可変長セグメントへの変換』

可変長セグメントの指定方法

長さが変動するのは可変長セグメントのデータ部分です。このデータ部分が、最小のバイト数と最大のバイト数の間で変動します。

以下の図に示すように、DBD の中の SEGM ステートメントの BYTES= キーワードで、最小サイズと最大サイズを指定します。 IMS は可変長セグメントのデータ部分の長さを知る必要があるので、各セグメントをロードするとき、ユーザーが各

セグメントに 2 バイトのサイズ・フィールドを組み込みます。 このサイズ・フィールドは、セグメントのデータ部分の中にあります。指定するデータ部分の長さは、サイズ・フィールドのために使用される 2 バイトを含めた長さでなければなりません。セグメント・タイプがシーケンス・フィールドを持つ場合には、サイズ・フィールドで指定する最小の長さは、少なくともサイズ・フィールドおよびシーケンス・フィールドの終わりまでのすべてのデータに等しい長さでなければなりません。

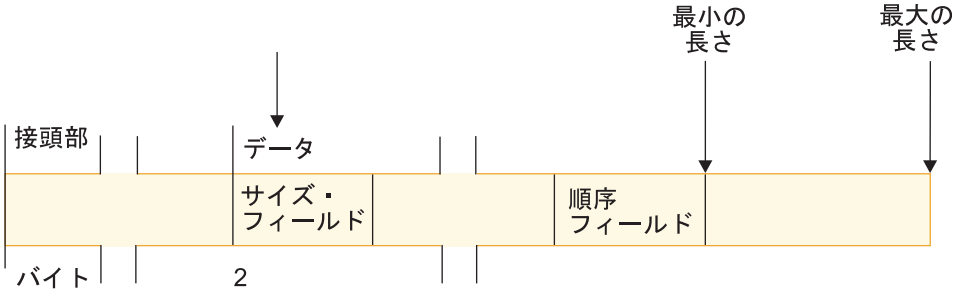


図 *195.* 可変長セグメントの指定方法

可変長セグメントの保管と処理方法

可変長セグメントが最初にロードされるとき、そのデータ部分を保管するのに使用されるスペースは、MINBYTES オペランドで指定されている長さ、あるいはサイズ・フィールドで指定されている長さのいずれか大きい方です。

MINBYTES オペランドのスペースの方が大きい場合には、必要以上のスペースがこのセグメントに割り振られます。このセグメントの中の既存のデータがこれよりも長いデータと置き換えられるとき、この余分のスペースを使用することができます。

更新が行われるとき、HDAM、PHDAM、HIDAM、および PHIDAM の可変長セグメントの接頭部とデータ部分とがストレージにおいて分離されることが可能です。このような分離が行われた場合、接頭部の次の最初の 4 バイトが、このセグメントの分離されたデータ部分を指します。

以下の図に示すのは HISAM 可変長セグメントのフォーマットです。また、セグメントの接頭部とデータ部とがストレージにおいて分離されていない場合の、HDAM、PHDAM、HIDAM、または PHIDAM の可変長セグメントのフォーマットです。

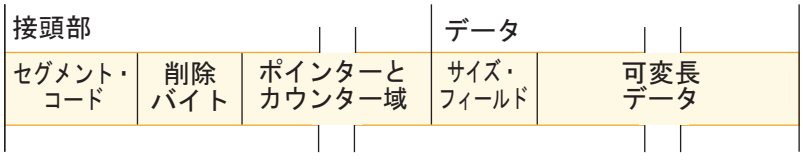


図 *196. HISAM* 可変長セグメントのフォーマット

以下の図に示すのは、セグメントの接頭部とデータ部分とがストレージで分離されている場合の、 HDAM、 PHDAM、 HIDAM、または PHIDAM の可変長セグメントのフォーマットです。

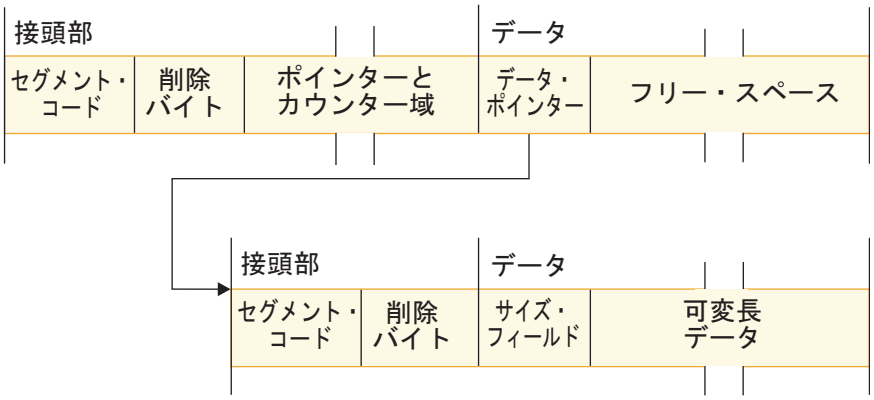


図 *197. HDAM*、*PHDAM*、*HIDAM* または *PHIDAM* 可変長セグメントのフォーマット

可変長セグメントがロードされた後、置き換え操作によって、このセグメントの中のデータのサイズが大きくなったり小さくなったりすることがあります。既存の HISAM セグメントの中のデータの長さが増えた場合には、スペースを追加して取得するために、このセグメントが入っている論理レコードが書き直されます。この書き直しによって配置換えされるセグメントは、オーバーフロー・ストレージに置かれます。オーバーフロー・ストレージにセグメントがそのように配置換えされると、パフォーマンスに影響が及ぶことがあります。既存の HISAM セグメントの中のデータの長さが短くなった場合には、この論理レコードの中のすべてが物理的隣接セグメントになるように、このレコードが書き直されます。

置き換え操作によって既存の HDAM、PHDAM、HIDAM または PHIDAM セグメントの中のデータの長さが増える場合には、次の 2 つのうちいずれかが起きます。

- この既存のセグメントに割り振られているスペースが新しいデータにとって十分な長さである場合は、単にこの新しいデータがこのセグメントの中に置かれます。このセグメントの接頭部とデータ部分とが以前にこのデータ・セットにおいて分離されていたかどうかには関係ありません。
- この既存のセグメントに割り振られているスペースが新しいデータにとって十分な長さでない場合は、このセグメントの接頭部とデータ部分とがストレージにおいて分離されます。 IMS は、このセグメントのデータ部分をできるだけ接頭部に近いところに置きます。いったんセグメントが分離されると、接頭部の次の最初の 4 バイトの中に、このセグメントのデータ部分を指すためのポインターが置かれます。この分離によって、このセグメントに必要なスペース量が増えますが、これは接頭部と一緒に保持されるポインターに加えて、1 バイトのセグメント・コードと 1 バイトの削除コードがこのセグメントのデータ部分に追加されるからです (424 ページの図 196 を参照)。さらに、セグメントの分離によってその 2 つの部分が異なるブロックに保管されることになると、このセグメントにアクセスするのに 2 回の読み取り操作が必要になります。

置き換え操作によって既存の HDAM、PHDAM、HIDAM または PHIDAM セグメントの中のデータの長さが減る場合には、次の 3 つのうちいずれかが起きます。

- 接頭部とデータとが分離されていない場合には、既存のセグメントの中のデータが、より短い新しいデータとこれに続くフリー・スペースとによって置き換えられます。
- 接頭部とデータとが分離されているがセグメントを再結合するために元の接頭部のすぐ後に十分なスペースが使用可能でない場合には、このセグメントの分離されたデータ部分の中のデータが、より短い新しいデータとこれに続くフリー・スペースとによって置き換えられます。
- overlay 接頭部とデータとが分離されていてセグメントを再結合するために元の接頭部のすぐ後ろに十分なスペースが使用可能な場合には、この新しいデータが元のこのスペースに置かれ、データ・ポインターをオーバーレイします。セグメントの分離された元のデータ部分は、HD データベースにおけるフリー・スペースとして使用可能になります。

可変長セグメントを使用する場合

セグメントの中のデータの長さが、例えば記述データのように変動する場合には、可変長セグメントを使用してください。

可変長セグメントを使用することにより、セグメント・タイプのデータ部分を手持ちの一番長い記述データと同じ長さにしなくても済みます。これによって、ストレージ・スペースが節約されます。しかし、注意すべきことは、
HDAM、PHDAM、HIDAM または PHIDAM データベースを使用していて、しかもセグメント・データの特質上、時間の経過と共にそのサイズが大きくなる場合には、セグメントが分割されるという点です。セグメントの分割によって、ある 1 つのセグメントの 2 つの部分が異なるブロックに収容されることになれば、このデータベースが再編成されるまでは、このセグメントにアクセスするのに 2 回の読み取り操作が必要です。したがって、セグメント・サイズは変動するが安定している場合、可変長セグメントはうまく働きます (例えば、住所のセグメントのように)。セグメント・サイズが増大傾向にあれば (例えば、売上手数料の累計表を含むセグメント・タイプのように) 可変長セグメントは適切に働かないおそれがあります。

可変長セグメントに関してアプリケーション・プログラマーが知っている必要のある事項

データベースにおいて可変長セグメントを使用している場合には、このデータベースを使用するアプリケーション・プログラマーに以下の事柄を知らせる必要があります。

アプリケーション・プログラマーは、アクセスすることのできるセグメント・タイプのうち、どれが可変長セグメントであるか知る必要があり、また各可変長セグメント・タイプの最大のサイズを知る必要があります。入出力域のサイズを計算するにあたって、アプリケーション・プログラマーは、可変長セグメントの最大のサイズを使用しなければなりません。さらに、アプリケーション・プログラマーは、可変長セグメントのデータ部分の最初の 2 バイトに、このサイズ・フィールドを含むデータ部分の長さが入っていることを知っておく必要があります。

データベース管理者は、アプリケーション・プログラマーと協力して、可変長セグメントの中のデータにアクセスする方式を作成すべきです。

セグメント編集/圧縮出口ルーチン

セグメント編集/圧縮出口ルーチンを使用すると、セグメントのデータ部のエンコード、編集、圧縮が可能になります。

この機能は、全機能データベースおよび高速機能 DEDB のセグメント・データで使用することができます。セグメントの中のデータを実際に操作するためのルーチンを書くのは、ユーザーです (このルーチンを編集ルーチンと呼ぶことにします)。IMS のコードは、セグメントの位置に関する情報をユーザーの編集ルーチンに与え、さらに、バッファー・プールとアプリケーション・プログラムの入出力域との相互間でセグメントを移動するのを助けます。

以下のデータベース・タイプがセグメント編集/圧縮出口ルーチンをサポートします。

- HISAM
- HDAM
- PHDAM
- HIDAM
- PHIDAM
- DEDB

セグメント編集/圧縮出口ルーチンの働きとその使用法の詳細については、「*IMS V13* 出口ルーチン」で説明しています。

このセグメント編集/圧縮出口ルーチンを用いて、以下のことを行うことができます。

- セキュリティーの目的のためのデータのエンコード。 データのエンコードは、セグメント・データが装置上にあるとき、セグメント・データを「かき混ぜる」ことであり、これによって編集ルーチンにアクセスすることのできるプログラムのみが、デコードされた形でセグメントを見ることができるようになります。
- データ編集。 データを編集することによって、アプリケーション・プログラムは、保管されているときのフォーマットとは異なるフォーマットでデータを受け取ることができます。例えば、アプリケーション・プログラムが、複数セグメント・フィールドを、これが収容されている順序とは異なる順序で受け取る場合もあり、また、アプリケーション・プログラムは、記述データからブランクのスペースをすべて取り除く必要がある場合もあります。
- データ圧縮。 これによって、 DASD ストレージの使用効率をあげることができます。装置に書き込むときにはセグメントを圧縮し、アプリケーション・プログラムに戻すときにはセグメントを拡張することができるからです。例えば、すべてのブランクとゼロを除去することにより、セグメント・データを圧縮します。
- データ拡張。 DBD で SDEP セグメント圧縮を指定し、さらに DEDB スキャン・ユーティリティーでキーワードの EXPANDSEG を指定すると、DEDB 順次従属スキャン・ユーティリティーはセグメント編集/圧縮出口ルーチン (DFSCMPX0) を呼び出して、圧縮された SDEP セグメントを解凍します。

  関連資料 : EXPANDSEG および DEDB スキャン・ユーティリティーについては、「*IMS V13* データベース・ユーティリティー」で説明されています。セグメント圧縮出口については、「*IMS V13* 出口ルーチン」で説明されています。

セグメント編集/圧縮出口ルーチンを用いると、以下の 2 種類のセグメント操作が可能です。

- データ圧縮。 キー・フィールドの内容や位置を変更しない形でのセグメント内のデータの移動または圧縮。一般的には、この方法にはキー・フィールドの終わりからセグメントの終わりまでのデータの圧縮が関係します。固定長セグメントを圧縮する場合には、ユーザー・データ圧縮ルーチンで 2 バイトのフィールドをセグメントのデータ部分の初めに付け加える必要があります。この 2 バイトのフィールドを使用して IMS は 2 次ストレージの必要量を決めますが、キー・フィールドの位置はこの時のみ変更できます。可変長セグメントのセグメント・サイズ・フィールドを圧縮することはできませんが、圧縮されたセグメントの長さを反映するために更新しなければなりません。
- キー圧縮。 キー・フィールドの相対位置、値、または長さおよびサイズ・フィールド以外の他のフィールドを変更できる仕方でのセグメント内の任意のデータの移動または圧縮。可変長セグメントのセグメント・サイズ・フィールドは、圧縮されたセグメントの長さを反映するように、圧縮ルーチンで更新しなければなりません。

セグメント編集/圧縮出口ルーチンの使用は、セグメント・タイプを定義するときに指定します。セグメントが以下の条件を満たしている限り、(データ圧縮またはキー圧縮を用いて) どのセグメント・タイプでも編集または圧縮することができます。

- 論理子ではない
- セグメントが HSAM、SHISAM、あるいは索引データベースの中にない

セグメント編集/圧縮出口ルーチンの使用は、物理データベース DBD 内で定義されます。この出口ルーチンの使用は、論理データベース DBD 内では定義できません。

次のセグメントの場合には、データ圧縮は可能ですが、キー圧縮はできません。

- HISAM データベースの中のルート・セグメント
- DEDB データベースの中のセグメント

関連タスク:

846 ページの『セグメント編集/圧縮出口ルーチンへの変換』
807 ページの『方法 2. セグメントまたはデータベースの変換』
42 ページの『データベースの暗号化』
920 ページの『出口ルーチンの変更と HALDB データベース』

セグメント編集/圧縮出口ルーチンの使用上の考慮事項

セグメント編集/圧縮出口ルーチンを使用する前に、いくつかの点に注意する必要があります。

注意が必要な点は次のとおりです。

- 編集ルーチンは DL/I 呼び出しの一部として実行されるのでこのルーチンが異常終了すると IMS 領域全体が異常終了します。
- ユーザー・ルーチンではオペレーティング・システム・マクロの LOAD、GETMAIN、SPIE、または STAE を使用することはできません。

- セグメント編集/圧縮出口ルーチンの名前は、DBDNAME と同じであってはなりません。
- アプリケーション・プログラムとの間での往復の途中で各セグメントの編集と圧縮が行われるたびに余分のプロセッサー時間が必要になります。

選択したオプションによっては、特定のセグメントの場所を見つけるための検索時間が増えることがあります。キー圧縮を用いてセグメントを完全に圧縮している場合には、完全に修飾されたキー・フィールド要求またはデータ・フィールド要求を満たす候補であるすべてのセグメント・タイプを拡張または分割する必要があります。これを行ってから、IMS は該当するフィールドを調べます。キー・フィールド修飾の場合には、検索の間に、セグメントの先頭からシーケンス・フィールドまでのフィールドのみが拡張されます。データ・フィールド修飾の場合には、セグメント全体が拡張されます。データ圧縮とキー・フィールド要求の場合には、セグメントの場所を見つけるのに、圧縮されていないセグメント場合よりも少し多くの処理が必要になります。セグメント・オカレンスが、修飾を満たすかどうかを決めるには、セグメントのシーケンス・フィールドのみが使用されます。

他の考慮事項が、特にオンライン環境で、システム全体のパフォーマンスに影響を及ぼすことがあります。例えば、アルゴリズム・テーブルをストレージにロードすることができれば圧縮ルーチンに大幅な柔軟性が与えられます。しかし、こうすると要求されたメンバーがストレージに入るまで IMS 制御領域全体が待ち状態に置かれる可能性があります。このような影響を軽減するためにすべての代案について検討するようお勧めします。

関連資料:

セグメント編集/圧縮出口ルーチン (出口ルーチン)

分割セグメントのパフォーマンスへの影響の防止

分割セグメントは、セグメントの両方の部分を検索するために余分な読み取りが必要となって、パフォーマンスに悪影響を及ぼすことがあります。

セグメントが分割されると、その接頭部は既存の場所に残りますが、データ部分は新しい場所に保管され、それが別のブロックや CI になる場合もあります。全機能データベース内のセグメントがその現在の場所のサイズより大きくなると、置き換え呼び出しによってセグメントが分割されることがあります。

IMS が圧縮セグメントを分割するのを防止するために、余分な埋め込みスペースを含むセグメントの最小サイズを指定することができます。これによって圧縮セグメントの余地が広がり、IMS がセグメントを分割する可能性が減少します。

固定長全機能セグメントの最小サイズの指定は、可変長全機能セグメントの最小サイズの指定とは異なります。

- 固定長セグメントの場合、SEGM ステートメントの COMPRTN= パラメーターで 4 番目と 5 番目のサブパラメーターを両方使用して最小サイズを指定します。 4 番目のサブパラメーター *size* は、5 番目のサブパラメーター PAD も指定する場合のみ、最小サイズを定義します。
- 可変長セグメントの場合、SEGM ステートメントの BYTES= パラメーターの 2 番目のサブパラメーター *min_bytes* を使用して、最小サイズを指定します。

DEDB セグメントは、置き換え呼び出しによって分割されることはありません。DEDB セグメントが現在場所のサイズより大きく成長すると、接頭部も含むセグメントの全体が新しい場所に移動します。このため、圧縮 DEDB セグメントに埋め込みを行う必要はありません。

関連資料:

SEGM ステートメント (システム・ユーティリティー)

セグメント編集/圧縮出口ルーチンの指定

セグメントに対してセグメント編集/圧縮出口ルーチンの使用を指定するには、DBD の中の SEGM ステートメントの COMPRTN= キーワードを使用します。

関連資料:

SEGM ステートメント (システム・ユーティリティー)

データ・キャプチャー出口ルーチン

データ・キャプチャー出口ルーチンは DL/I データベースからセグメント・レベルのデータをキャプチャーし、これを複数の Db2 for z/OS データベースに波及させます。 IMS と Db2 for z/OS データベースを稼働しているご使用のシステムでは、データ・キャプチャー出口ルーチンを使用して、2 つのタイプのデータベース間でデータを交換することができます。

データ・キャプチャー出口ルーチンは、インストール先作成出口ルーチンです。データ・キャプチャー出口ルーチンは、データベースの共存を促進し強化します。

以下のデータベース・タイプがデータ・キャプチャー出口ルーチンをサポートします。

- HISAM
- SHISAM
- HDAM
- PHDAM
- HIDAM
- PHIDAM
- DEDB

データ・キャプチャー出口ルーチンは、アセンブラー言語、C、COBOL、または PL/I で作成することができます。

データ・キャプチャー出口ルーチンは、IMS のトランザクション・マネージャーとデータベース・マネージャーでサポートされます。 DBCTL は、BMP に対してのみサポートされます。

データ・キャプチャー出口ルーチンは、副次索引の中のセグメントをサポートしません。

データ・キャプチャー出口ルーチンは、DBD 内のセグメント・レベルの指定に基づいて呼び出されます。データ・キャプチャー出口ルーチンがデータベース・セグメ

ント上で指定されると、このルーチンは、どの PSB がアクティブであるかに関係なく、あらゆるアプリケーション・プログラムのそのセグメントに関するアクティビティーによって呼び出されます。したがって、データ・キャプチャー出口ルーチンはグローバル・ルーチンです。データ・キャプチャー出口ルーチンを使用すると、データベース・システム全体のパフォーマンスに影響が及ぶことがあります。

関連タスク:

847 ページの『データ・キャプチャー出口ルーチンおよび非同期データ・キャプチャーのためのデータベースの変換』

920 ページの『出口ルーチンの変更と HALDB データベース』

関連資料:

データ・キャプチャー出口ルーチン (出口ルーチン)

データ・キャプチャー出口ルーチン用の DBD パラメーター

データ・キャプチャー出口ルーチンを使用するには、1 つまたは 2 つの DBD パラメーターとその後の DBDGEN を指定する必要があります。

このトピックにはプロダクト・センシティブ・プログラミング・インターフェース情報が含まれています。

EXIT= パラメーターは、データベースの中のセグメントに対してどのデータ・キャプチャー出口ルーチンが実行されるかを示します。 VERSION= パラメーターは、データ・キャプチャー出口ルーチンが使用するための DBD に関する重要な情報を記録します。

EXIT= パラメーター

データ・キャプチャー出口ルーチンを使用するには、DBD ステートメントまたは SEGM ステートメントでオプションの EXIT= パラメーターを使用しなければなりません。 EXIT= は、物理データベース定義の DBD ステートメントまたは SEGM ステートメント上で指定します。

DBD ステートメント上で EXIT= を指定すると、1 つのデータベース構造の中にあるすべてのセグメントにデータ・キャプチャー出口ルーチンが適用されます。SEGM ステートメント上で EXIT= を指定すると、そのセグメント・タイプのみにデータ・キャプチャー出口ルーチンが適用されます。

SEGM ステートメント上に EXIT= を指定することにより、DBD ステートメント上に指定されていたデータ・キャプチャー出口ルーチンをオーバーライドすることができます。 SEGM ステートメントで EXIT=NONE を指定すると、そのセグメント・タイプに対して DBD ステートメントで指定されていたすべてのデータ・キャプチャー出口ルーチンが取り消されます。物理子は、その物理親の SEGM ステートメントに指定されている EXIT= パラメーターを継承しません。

単一の DBD または SEGM ステートメントに複数のデータ・キャプチャー出口ルーチンを指定することができます。例えば、DBD ステートメントを次のようにコーディングすることができます。

```
DBD   EXIT=((EXIT1A),(EXIT1B))
```

EXIT= パラメーターに必要なオペランドは、使用しようとしているデータ・キャプチャー出口ルーチンの名前だけです。出口名は、最大 8 文字の英数字で構成することができます。例えば、あるデータベースの SEGM ステートメント上で EXITA という名前でデータ・キャプチャー出口ルーチンを指定する場合、EXIT= パラメーターは次のようにコーディングされます。

```
SEGM   EXIT=(EXITA,KEY,DATA,NOPATH,DLET,BEFORE,(CASCADE,KEY,DATA,NOPATH,DLET,BEFORE))
```

KEY、DATA、NOPATH、DLET、BEFORE、CASCADE、KEY、DATA、NOPATH、DLET、および BEFORE はデフォルト・オペランドです。これらのデフォルトは、あるアプリケーション・プログラムによってセグメントが更新された場合、その出口ルーチンによってどのデータがキャプチャーされるか定義します。

VERSION= パラメーター

VERSION= は、データ・キャプチャー出口ルーチンをサポートするオプション・パラメーターです。 VERSION= は、DBD ステートメントで次のように指定されます。

```
VERSION='character string'
```

文字ストリングの最大長は 255 バイトです。 VERSION= を使用して、データ・キャプチャー出口ルーチンの本来の機能に影響を及ぼすデータベースの特性を示す命名規則を作成することができます。論理関係が入っている DBD にフラグを立てたり、あるいはどのデータ・キャプチャー出口ルーチンを DBD または SEGM ステートメント上に定義するか示すのに、VERSION= を使うことができます。 VERSION= は、次のようにコーディングされます。

```
DBD   VERSION='DAL-&SYSDATE-&SYSTIME'
```

このステートメントの中の DAL は、DBD ステートメント (D) 上でデータ・キャプチャー出口ルーチン A が指定され、データベースに論理関係 (L) が入っていることを示します。 &SYSDATE と &SYSTIME は、DBD が生成された日付と時刻を示します。

VERSION= パラメーターを指定しないと、DBDGEN がデフォルトの 13 文字の日付/タイム・スタンプを生成します。デフォルトは、次に示すフォーマットの、8 バイトの日付スタンプと 5 バイトのタイム・スタンプで構成されます。

```
MM/DD/YYHH.MM
```

VERSION= 上のデフォルトの日付/タイム・スタンプは、DBDGEN 日付/タイム・スタンプと同じです。

VERSION= パラメーターは、2 バイトの長さの VERSION= と共に可変長文字ストリングとして渡されますが、これには LL の長さは含まれていません。

関連資料:

- DBD ステートメント (システム・ユーティリティー)
- SEGM ステートメント (システム・ユーティリティー)

68 ページの『CAPXSEGM セグメント・タイプのフォーマット』
66 ページの『CAPXDBD セグメント・タイプのフォーマット』

データ・キャプチャー出口ルーチンの呼び出しシーケンス

データ・キャプチャー出口ルーチンが指定されている各セグメントに対するセグメント更新のたびに、データ・キャプチャー出口ルーチンが 1 回呼び出されます。特定の条件のもとでは、データ・キャプチャー出口ルーチンは、単一の呼び出しに対して複数回呼び出されます。

このトピックにはプロダクト・センシティブ・プログラミング・インターフェース情報が含まれています。

単一の呼び出しに対してデータ・キャプチャー出口ルーチンが複数回呼び出される条件は、次のとおりです。

- パスの更新
- 複数のセグメント・タイプまたは複数のセグメント・オカレンスが削除される場合のカスケード削除
- 論理子での更新
- 論理親での更新
- 複数のデータ・キャプチャー出口ルーチンが単一セグメントに対して指定されている場合の、そのセグメントでの更新。各出口は DBD または SEGM ステートメントにリストされている順番に 1 回ずつ呼び出されます。

1 回のアプリケーション・プログラムの呼び出しで複数のセグメントが更新されると、データ・キャプチャー出口ルーチンは、IMS がそのセグメントを物理的に更新するのと同じ順序で呼び出されます。

1. パスの挿入は DL/I では「トップダウン」で実行されます。したがって、親セグメントに対するデータ・キャプチャー出口ルーチンは、その親の従属セグメントに対するデータ・キャプチャー出口ルーチンより前に呼び出されます。
2. カスケード削除は「ボトムアップ」で実行されます。カスケード削除では、従属セグメントの出口はすべて、各親の出口より前に呼び出されます。 IMS は、階層に沿って最低レベルのセグメントまで進むことによって削除規則を検証し終わって初めて、カスケード削除で従属セグメントを物理的に削除します。削除規則の妥当性を検証した後、IMS は従属セグメント・チェーンの最低レベルのセグメントから始めて、チェーンを上方へ進み、最後に階層内の最高レベルの親セグメントを削除します。カスケード削除においてセグメントに指定されているデータ・キャプチャー出口ルーチンは、階層構造の逆順に呼び出されます。
3. パスの置き換えは、IMS では「トップダウン」で行われます。パスの置き換えにおいて、セグメントに対して定義されているデータ・キャプチャー出口ルーチンでは、親セグメントが最初に置き換えられます。次いでその従属セグメントがすべて階層構造の降順に置き換えられます。

アプリケーション・プログラムが論理的に関連しているセグメントにカスケード削除を行う場合には、常に、論理親に定義されているデータ・キャプチャー出口ルーチンより前に論理子に定義されているデータ・キャプチャー出口ルーチンが呼び出されます。セグメントの削除を行うと結果として削除されたセグメントの親が削除されることになる再帰構造の場合を除き、論理子の方が物理的階層の上位にあっても、データ・キャプチャー出口ルーチンが呼び出されます。

データ・キャプチャー出口ルーチンに受け渡されるデータおよび捕そくされるデータ

データは、アプリケーション・プログラムが DL/I の挿入、削除、または置き換え呼び出しで IMS を更新するときに、データ・キャプチャー出口ルーチンに渡されます。SSPCMD キーワードが指定されると、このデータにはサブセット・ポインター・コマンド・コード (M、S、W、Z) の更新、および R コマンド・コードも含まれます (DEDB のみ)。

このトピックにはプロダクト・センシティブ・プログラミング・インターフェース情報が含まれています。

SSPCMD キーワードが指定されると、サブセット・ポインターを更新する GET タイプ呼び出しでもデータがキャプチャーされます。 R コマンド・コード (指定されている場合) もキャプチャーされます。同様に、サブセット・ポインターを更新する挿入呼び出しまたは置き換え呼び出しで、指定されたパスの更新されていないセグメントについてもデータがキャプチャーされます。

INPOS キーワードが指定されていて、HERE の ISRT ルールによって非固有セグメントが挿入される場合、後に続くツイン (存在する場合) のセグメント・データがキャプチャーされます。

データ・キャプチャー出口ルーチンに渡されるセグメント・データは、常に物理データです。その更新が論理子に関与している場合には、渡されるデータは物理データとその論理親セグメントの連結キーです。セグメント編集/圧縮出口ルーチン (DFSCMPX0) を使用するセグメントの場合、渡されるデータは拡張されたデータです。

アプリケーションがセグメントを置き換えるときに、既存および置き換えの両方の物理データがキャプチャーされます。一般に、アプリケーションの呼び出しがデータを変更しない場合でも、セグメント・データはキャプチャーされます。しかし、全機能データベースに関しては、IMS は前と後のデータの比較を行います。データが変化していないと、IMS はそのデータベースを更新したり、置き換えデータを記録したりしません。データが置き換えられていないので、そのセグメントに指定されているデータ・キャプチャー出口ルーチンは呼び出されず、そのデータはキャプチャーされません。

次の場合には、セグメント・データが変化しなくても、置き換え中にデータがキャプチャーされることがあります。

1. アプリケーションが論理子と論理親の連結を挿入し、IMS が論理親を置き換え、親データは変化しない。
2. アプリケーションが DEDB データベース内のセグメントに対して置き換えを発行する。
3. SSPCMD キーワードが指定されている場合に、指定されたパスの更新されていないセグメントにサブセット・ポインターの更新が含まれている (DEDB のみ)。

いずれの場合も、IMS は前と後のデータを比較せずにデータベースを更新するので、データは、変化していなくてもキャプチャーされます。

推奨事項**:** アプリケーションがある 1 つのセグメントを置き換えるときに、前と後のセグメント全体がデータ・キャプチャー出口ルーチンに渡されます。出口ルーチンの処理対象となるフィールドがほんのわずかである場合は、該当のフィールドの置換前と置換後のデータを比較して、フィールドが変更されたかどうかを確認するまでは、SQL 更新要求を発行しないでください。

データ・キャプチャー呼び出し機能

データ・キャプチャー出口ルーチンは、アプリケーション・プログラムが挿入、置き換え、または削除呼び出しによってセグメント・データを更新するときに呼び出されます。サブセット・ポインターの更新をキャプチャー (SSPCMD) する場合は、GET 呼び出しもキャプチャーされます。

このトピックにはプロダクト・センシティブ・プログラミング・インターフェース情報が含まれています。

アプリケーション・プログラムが親セグメントを削除したことが原因で DL/I がその従属セグメントを削除する (カスケード削除と呼ばれる処理) ときに、オプションとして、データ・キャプチャー出口ルーチンが呼び出されるように選択することもできます。データ・キャプチャー出口ルーチンには、以下のアクションを識別するために、2 つの機能が渡されます。

1. アプリケーション・プログラムにより実行されるアクション
2. IMS により実行されるアクション

データ・キャプチャー出口ルーチンに渡される 2 つの機能は次のものです。

- 呼び出し機能。セグメントに対してアプリケーション・プログラムが出す DL/I 呼び出し、ISRT、REPL、Gx (GET タイプ)、または DLET。
- 物理機能。呼び出しの結果として IMS が実行する物理的なアクション ISRT、REPL、または DLET。物理機能 SSPU は、更新されていないものの、サブセット・ポインターの更新を含んでいるセグメントに対して設定されます。この物理機能は、データの伝搬を行うときに出す SQL 要求のタイプを決めるために用います。

出口ルーチンに渡される呼び出し機能と物理機能は、置き換え呼び出しに対しては常に同じです。ただし、削除呼び出しと挿入呼び出しでは渡される機能が異なる場合があります。

- カスケード削除をもたらす削除呼び出しの場合、渡される呼び出し機能は CASC (カスケード削除を示すため) であり、渡される物理機能は DLET です。
- 論理子の挿入と論理親の置き換え (論理親が既に存在するため) をもたらす挿入呼び出しの場合、渡される呼び出し機能は ISRT であり、渡される物理機能は REPL です。 IMS は、親データが変化しない場合でも、論理親をアプリケーション・プログラムによって挿入されたデータで物理的に置き換えます。次に、呼び出し機能と物理機能はともに、データ伝搬の要件に基づいて、データ・キャプチャー出口ルーチンで出す SQL 要求を決定するために使用されます。

論理関係間を交差するカスケード削除

EXIT= オプションで NOCASCADE を指定すると、カスケード削除に対してデータはキャプチャーされません。しかし、カスケード削除が、ある 1 つの論理関係か

ら別の物理データベースに交差して従属セグメントを削除するときは、データ・キャプチャー出口ルーチンを呼び出す必要があります。

このトピックにはプロダクト・センシティブ・プログラミング・インターフェース情報が含まれています。

Db2 for z/OS 内の物理構造の親に対して SQL 削除を出すには、データ・キャプチャー出口ルーチンを呼び出す必要があります。 IMS は定義された出口ルーチンで物理データベース・レコード内の親セグメントを削除する場合、そのセグメントに CASCADE オプションが指定されているか、NOCASCADE オプションが指定されているかには関係なく、EXIT= CASCADE オプションを要求するのではなしに、セグメントに対する出口ルーチンを常に呼び出します。 IMS は論理関係を別の物理データベースに交差する場合にのみ、NOCASCADE オプションを迂回します。すべてのカスケード削除の場合と同様に、渡される呼び出し機能は CASC であり、渡される物理機能は DLET です。

データ・キャプチャー出口ルーチンおよび論理的に関連するデータベース

データ・キャプチャー出口ルーチンに渡されるセグメント・データは、常に物理データです。したがって、データ・キャプチャー出口ルーチンをサポートする論理的に関連するデータベースの削除規則に、制約を設ける必要があります。

このトピックにはプロダクト・センシティブ・プログラミング・インターフェース情報が含まれています。

以下の表は、セグメントでデータ・キャプチャー出口ルーチンが指定されている論理リレーショナル・データベースで使用できる削除規則と使用できない削除規則をまとめたものです。

表 *63.* データ・キャプチャー出口ルーチンを使用する論理リレーショナル・データベースに対する削除規則の制約事項

セグメント・タイプ	仮想削除規則	論理削除規則	物理削除規則
論理子	あり	なし	なし
論理親	なし	あり	あり

論理的に関連するデータベースで論理子の削除規則違反があった場合、結果は次のようになります。

- その論理子はデータ・キャプチャー出口ルーチンを指定できない。
- その論理子の祖先には、データ・キャプチャー出口ルーチンをもつことができるものはない。

論理的に関連しているデータベースが、論理親で削除規則を違反すると、その論理親にはデータ・キャプチャー出口ルーチンを指定できません。 ACBGEN は、論理削除規則の制約事項を検証し、その制約事項に違反したデータベースを参照する PSB が続行することを許さないようにします。

フィールド・レベル・センシティビティー

フィールド・レベル・センシティビティーは、アプリケーション・プログラムのデータ独立性、データ・セキュリティーの強化、およびセグメント・タイプのフォーマット設定における柔軟性の向上に関連する多数の利点をもたらします。

フィールド・レベル・センシティビティーは、アプリケーション・プログラムを次のことから分離することによって、データ独立性のレベルを向上させます。

- セグメント内の諸フィールドの配置の変更
- セグメント内のデータの追加または削除

また、フィールド・レベル・センシティビティーは、1 つのアプリケーション・プログラムをセグメントの中のフィールドの 1 つのサブセットのみに限定したり、置き換え操作をフィールド・レベルで制御することによって、データのセキュリティーが強化します。

フィールド・レベル・センシティビティーを使用すると、セグメント・タイプのフォーマットを変更することができます。この再フォーマット設定は、諸フィールドが除去されていたり、あるいはフィールドの長さやデータ・タイプが変更されたりしていない限り、アプリケーション・プログラムからのセグメント・データに対するビューを変更することなく実行することができます。アプリケーション・プログラムには感知されないような方法で、セグメントにフィールドを追加したりセグメントの中のフィールドの位置を変更したりすることができます。フィールド・レベル・センシティビティーは、アプリケーションのセグメント編成を、常に SENFLD ステートメントの指定に合った編成にします。

以下のデータベース・タイプがフィールド・レベル・センシティビティーをサポートします。

- HSAM
- HISAM
- SHISAM
- HDAM
- PHDAM
- HIDAM
- PHIDAM

マッピング・インターフェースとしてのフィールド・レベル・センシティビティーの使用

フィールド・レベル・センシティビティーは、PSBGEN フィールドの位置を、DBDGEN フィールドの位置とは異なるものにすることによって、マッピング・インターフェースとしての役割を果たします。マッピングは、入力ではセグメント編集ルーチンの後、出力ではセグメント編集ルーチンの前に呼び出されます。データベース情報から順次データ・セットを作成する場合 (あるいは順次データ・セットからデータベース情報を作成する場合)、フィールド・レベル・センシティビティーを使用すると、アプリケーション・プログラムが実行しなければならないフォーマット設定の量を削減したり、なくしたりすることができます。

可変長セグメントでのフィールド・レベル・センシティビティーの使用

可変長セグメントでフィールド・レベル・センシティビティーを使用すると、データベースを再編成することなく、セグメントに新しいフィールドを追加することができます。 DBDGEN の中の FIELD 定義を使用することにより、セグメントの以前のユーザーに影響を及ぼすことなく、そのセグメント・タイプを拡大することができます。 DBDGEN FIELD ステートメントでは、物理セグメントの中にまだ実在しないが、このセグメントが取り出されるとき動的に作成されるフィールドを指定させるようにします。

フィールド・レベル・センシティビティーは、アプリケーション・プログラムが非データベース環境からデータベース環境に移る際に役立ちます。データベースが変換を意図して設計されたものであれば、以前 z/OS ファイルにアクセスしていたアプリケーション・プログラムが、同じ情報を同じフォーマットで受け取ることができる場合もあります。

DEDB と MSDB には、フィールド・レベル・センシティビティーはサポートされません。

関連タスク:

808 ページの『セグメント内のデータの位置の変更』

DBD と PSB でのフィールド・レベル・センシティビティーの使用法の指定方法

アプリケーション・プログラムからのデータに対するビューは、PSBGEN ユーティリティーを通じて、SENSEG ステートメントに続く複数の SENFLD ステートメントを用いて定義されます。

SENFLD ステートメントの中で、NAME= パラメーターは DBDGEN ユーティリティーによってこのセグメント内に定義されたフィールドを識別します。SENFLD ステートメントは、高速機能副次索引ではサポートされません。

START= パラメーターは、アプリケーション・プログラムの入出力域でのこのフィールドの開始位置を定義します。入出力域では、フィールドが特定の順序で並んでいる必要はなく、フィールドとフィールドが隣接している必要もありません。入出力域におけるセグメントの終わりは、右端のフィールドの終わりによって定義されます。フィールド・レベル・センシティビティーを使用するセグメントは、すべてこの入出力域では固定長のように見えます。この長さは、1 つの SENSEG ステートメントに関連した複数の SENFLD ステートメント上のフィールドの長さの和によって決まります。

以下の図に、フィールド・レベル・センシティビティーの例を示します。図の後に、フィールド・レベル・センシティビティーのコーディングについての情報を記載します。

次の例では、アプリケーション・プログラムの入出力域で、物理セグメントの 3 つのフィールドの位置変更をするために、フィールド・レベル・センシティビティーを使用しています。

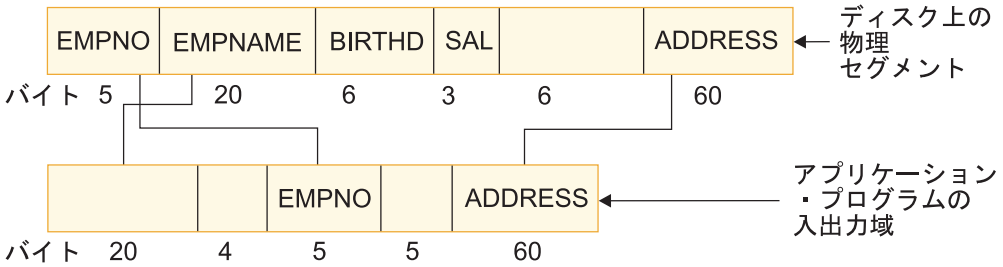


図 *198.* フィールド・レベル・センシティビティーのための *DBD* および *PSB* コーディング

以下のコードは、上の図に示したフィールド・レベル・センシティビティーの DBD の例を示しています。

```
SEGM  NAME=EMPREC,BYTES=100
 FIELD NAME=(EMPNO,SEQ),BYTES=5,START=1,TYPE=C
 FIELD NAME=EMPNAME,BYTES=20,START=6,TYPE=C
 FIELD NAME=BIRTHD,BYTES=6,START=26,TYPE=C
 FIELD NAME=SAL,BYTES=3,START=32,TYPE=P
 FIELD NAME=ADDRESS,BYTES=60,START=41,TYPE=C
```

以下のコードは、前の図に対する PSB を示しています。

```
SENSEG NAME=EMPREC,PROCOPT=A
 SENFLD NAME=EMPNAME,START=1,REPL=N
 SENFLD NAME=EMPNO,START=25
 SENFLD NAME=ADDRESS,START=35,REPL=Y
```

SENFLD ステートメントは、入出力域に現れることのある各フィールドごとに 1 つずつコーディングします。各 SENSEG ステートメントごとに、最高 255 の SENFLD ステートメントをコーディングすることができます。単一の PSB に対する SENFLD ステートメント数の上限は 10000 です。

SENFLD ステートメントのオプション・パラメーターである REPL= パラメーターは、このフィールドにおいて置き換え操作が許されるかどうかを指定するのに用いられます。この図では、置き換えは EMPNAME に対しては許されていませんが、EMPNO と ADDRESS に対しては許されています。 REPL= を SENFLD ステートメントにコーディングしないと、デフォルトは REPL=Y です。

DBD の中の FIELD ステートメントの TYPE= パラメーターは、挿入操作における充てん値を決めるのに用いられます。

フィールド・レベル・センシティビティーを使用するセグメントの検索

フィールド・レベル・センシティビティーを使用してセグメントを検索する場合は、以下のことに注意してください。

- 検索呼び出しにあたって、入出力域の中のフィールドとフィールドの間のギャップはブランクに設定されます。
- アプリケーション・プログラムが SSA の中のある 1 つのフィールドを使用する場合、そのフィールドは SENFLD ステートメント上でコーディングしなければなりません。この規則は、検索操作の SSA の中で使用するシーケンス・フィールドには適用されません。

以下の図は、 438 ページの『DBD と PSB でのフィールド・レベル・センシティビティーの使用法の指定方法』 の DBD と PSB に基づいた検索呼び出しの一例を示したものです。

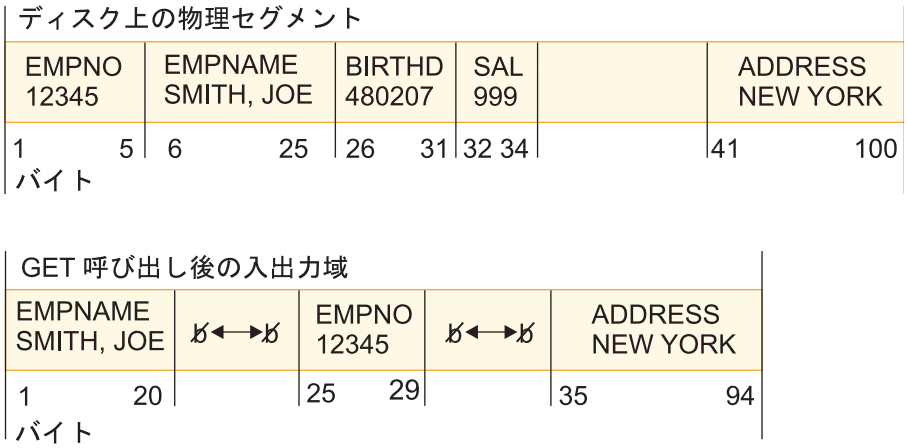


図 *199.* 検索呼び出しの例

フィールド・レベル・センシティビティーを使用するセグメントの置き換え

アプリケーション・プログラムがセグメントの中のデータを置き換えようとしている場合には、置き換え操作 (REPL=Y) を SENFLD ステートメントで許可しておかなければなりません。

438 ページの『DBD と PSB でのフィールド・レベル・センシティビティーの使用法の指定方法』の例では、EMPNAME 用の SENFLD ステートメントで REPL=N と指定しています。アプリケーション・プログラムが EMPNAME フィールドを置き換えようとすると、「DA」という状況コードが返されます。以下の図は、 438 ページの『DBD と PSB でのフィールド・レベル・センシティビティーの使用法の指定方法』 の DBD と PSB に基づいた REPL 呼び出しの例を示しています。

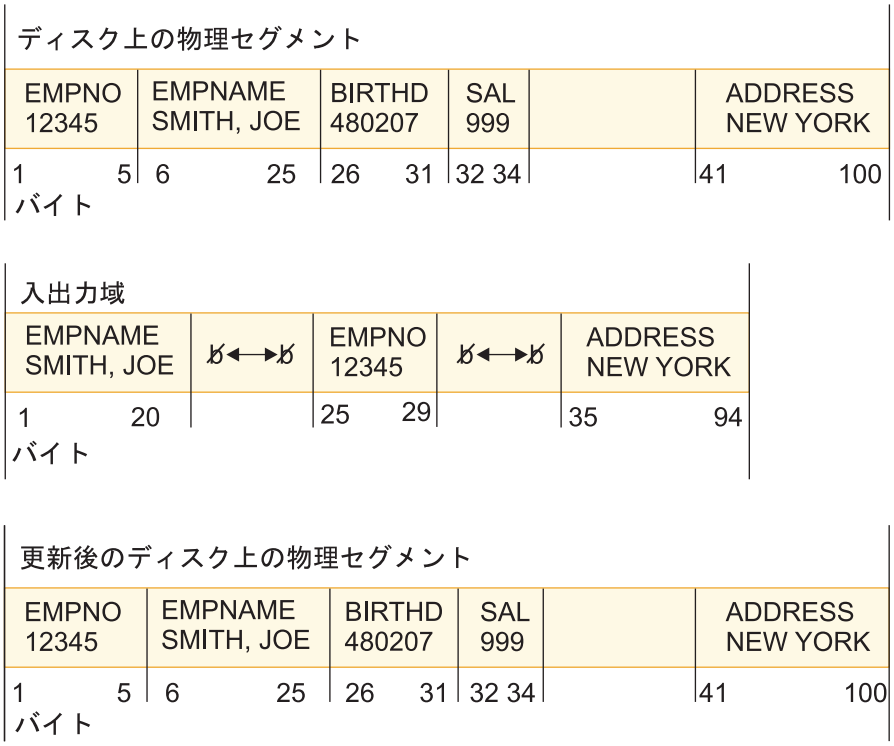


図 *200. REPL* の呼び出しの例

フィールド・レベル・センシティビティーを使用するセグメントの挿入

DBD の SEGM ステートメント上の TYPE= パラメーターは、アプリケーション・プログラムが挿入呼び出しで、あるフィールドに対してセンシティブではない場合の物理セグメントの中の充てん値を決定します。

TYPE= 充てん値

X 2 進数ゼロ

P パック 10 進数ゼロ

C ブランク

次のような場合には、物理セグメントの中の充てん値は 2 進数ゼロです。

- セグメントの中のスペースが DBD の中の FIELD マクロによって定義されていない、または
- 定義されている DBD フィールドが挿入操作において参照されていない。

以下の図は、 439 ページの図 198 の DBD と PCB に基づいた挿入操作の例を示しています。

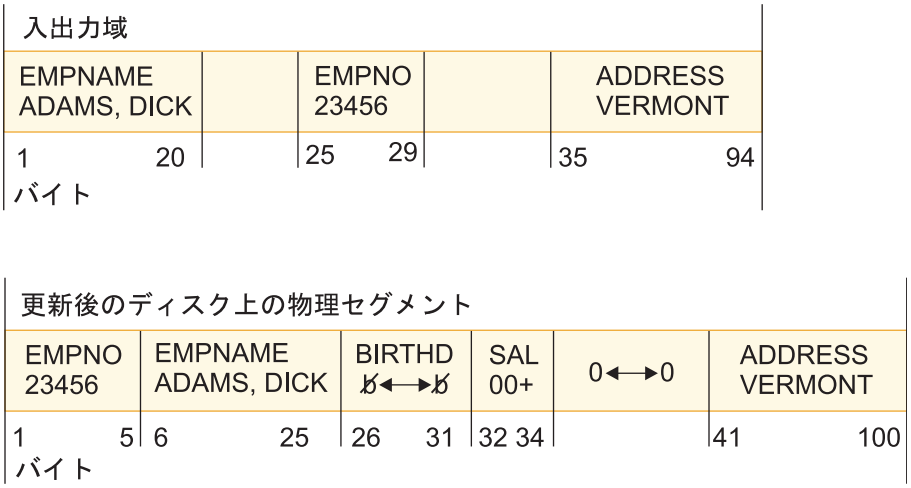


図 *201. ISRT* 呼び出しの例

BIRTHD フィールドには、DBD の FIELD ステートメントで TYPE=C が指定されているため、ブランクが挿入されます。SAL フィールドには、DBD の FIELD ステートメントで TYPE=P が指定されているため、パック 10 進数のゼロが挿入されます。位置 35 から 40 には、DBD でこのスペースのための FIELD ステートメントがコーディングされていないため、2 進数ゼロが挿入されます。

フィールドがオーバーラップしている場合のフィールド・レベル・センシティビティーの使用

SENFLD ステートメントでは、入出力域でのフィールドの開始位置をコーディングします。

入出力域においてフィールドとフィールドがオーバーラップする場合、次の規則に従わなければなりません。

- 入力に際して 2 つの異なるバイト数のデータを入出力域の同じ位置に移動することはできません。
- 検索操作にあたっては、同じデータを入出力域の中の異なる位置に移動することができます。
- 出力に際して、入出力域の中の異なる位置からの 2 バイトを同じ DBD フィールドに移動することはできません。

パス呼び出しを出す場合のフィールド・レベル・センシティビティーの使用

アプリケーション・プログラムがフィールド・レベル・センシティビティーを使用している間にパス呼び出しを出す場合には、以下のいくつかの規則に従わなければなりません。

従う必要がある規則は次のとおりです。

- 異なる物理セグメントから取られた 2 つのフィールドが入出力域の中の同じセグメントの中に来るような形で、SENFLD ステートメントをコーディングしてはなりません。

- PCB ステートメントには PROCOPT=P が必要です。

論理関係でのフィールド・レベル・センシティビティーの使用

論理関係に関与しているセグメントに対してフィールド・レベル・センシティビティーを使用する場合は、いくつかの規則に従う必要があります。

従う必要がある規則は次のとおりです。

- アプリケーション・プログラムは、挿入呼び出しに当たって論理子にセンシティブになることはできません。
- ある 1 つの SENSEG に属する複数の SENFLD ステートメントにおいて、同じフィールドを参照することができます。このような重複フィールド名が連結セグメントの一部分であり、しかも、同じフィールド名が連結の両方の部分の中に現れる場合には、最初の部分が論理子を参照します。 2 番目の部分とこれ以降のすべての部分が論理親を参照します。この参照順序によって、フィールドを入出力域に移動する順序が決まります。
- 仮想論理子でフィールド・レベル・センシティビティーを使用する場合、対をなすセグメントのフィールド・リストが検索されるのは、仮想セグメントのフィールド・リストでの検索の後で、論理親のフィールド・リストでの検索の前です。

可変長セグメントでのフィールド・レベル・センシティビティーの使用

可変長セグメントでフィールド・レベル・センシティビティーを使用する場合、このセグメントのアプリケーション・プログラムのビューは固定長で、 2 バイトの長さフィールドを含んでいません。

このトピックとサブトピックでは、可変長セグメントにフィールド・レベル・センシティビティーを使用する場合、検討すべき特殊な状態について説明します。その前にまず、可変長セグメントでのフィールド・レベル・センシティビティーの使用に関する一般情報をいくつか示します。

- 可変長セグメントを挿入する場合、使用される長さは、すべてのセンシティブ・フィールドを保持するのに必要な最小の長さです。
- 可変長セグメントを置き換える場合、アプリケーション・プログラムが変更したデータを収容するために長さを長くする必要があるのであれば、使用される長さは、変更されたデータを保持するのに必要な最小の長さです。
- 可変長セグメントの中のフィールドがオーバーラップしている場合、これらのフィールドのいずれかのデータ・タイプが文字でなければ、アプリケーション・プログラムは取得センシティビティーまたは更新センシティビティーをもっているこれらの重複フィールドにセンシティブになることはできません。
- 長さフィールドを用いてフィールドの存在または不在を判断するための可変長セグメントを処理する既存のプログラムは、フィールド・レベル・センシティビティーの使用によるセグメントの挿入または更新の場合には、変更される必要があることがあります。

可変長セグメントでフィールド・レベル・センシティビティーを使用する場合、2 つの状況について知っておく必要があります。 1 番目はフィールドがないときです。 2 番目はフィールドが部分的に存在する場合です。このトピックでは、次に挙げる情報を検討することにします。

- 不在フィールドの検索
- 不在フィールドの置き換え
- 不在フィールドの挿入
- 部分的に存在するフィールドの検索
- 部分的に存在するフィールドの置き換え

不在フィールドの検索

検索時に、あるフィールドが可変長物理セグメントの中に存在しない場合は、アプリケーション・プログラムの入出力域の中のこれに対応するフィールドには、DBD で指定されているデータ・タイプに基づいた値が充てんされます。

以下の図は、図の後ろに示している DBD と PSB の例に基づいた検索呼び出しでの不在フィールドの一例です。

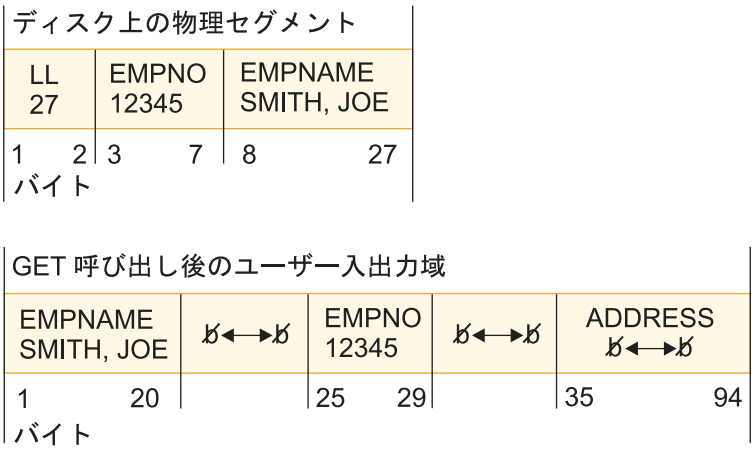


図 *202.* 検索呼び出しにおける不在フィールドの例

以下のコードは、可変長セグメントを使用したフィールド・レベル・センシティビティーの DBD の例です。

```
DBD

SEGM  NAME=EMPREC,BYTES=(102,7)
FIELD NAME=(EMPNO,SEQ),BYTES=5,START=3,TYPE=C
FIELD NAME=EMPNAME,BYTES=20,START=8,TYPE=C
FIELD NAME=BIRTHD,BYTES=6,START=28,TYPE=C
FIELD NAME=ADDRESS,BYTES=60,START=43,TYPE=C
```

以下のコードは、可変長セグメントを使用したフィールド・レベル・センシティビティーの PSB の例です。

```
PSB

SENSEG  NAME=EMPREC,PROCOPT=A
SENFLD  NAME=EMPNAME,START=1,REPL=N
SENFLD  NAME=EMPNO,START=25
SENFLD  NAME=ADDRESS,START=35,REPLY=Y
```

入出力域の中には長さフィールドはありません。また、DBD の FIELD ステートメントで TYPE=C が指定されているので、住所フィールドはブランクで充てんされています。

関連概念:

『不在フィールドの置き換え』

448 ページの『部分的に存在するフィールドの置き換え』

不在フィールドの置き換え

置き換えられていない不在フィールドは、可変長物理セグメントに影響を及ぼしません。

以下の図は、 444 ページの『不在フィールドの検索』 の DBD と PSB に基づいたある置き換え呼び出しでの不在フィールドの一例です。

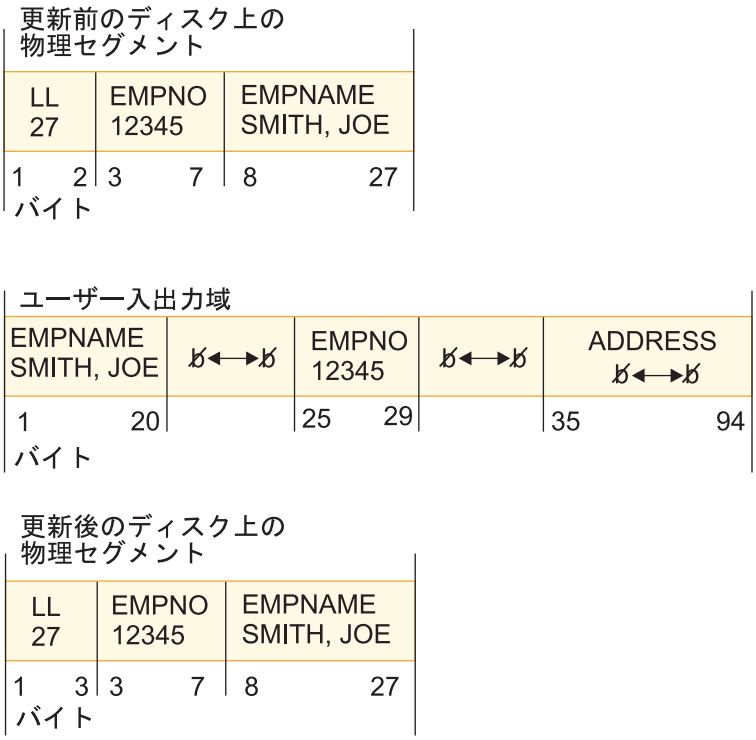


図 *203.* 置き換え呼び出しにおける不在フィールドの最初の例

IMS によって維持される長さフィールドには、このフィールドが存在せず置き換えられなかったので、住所フィールドのためのスペースが組み込まれていません。

置き換え呼び出しにおいて、充てん値のあるアプリケーション・プログラムに返されたフィールドが、充てん値以外の値に変更されると、セグメント長は、変更されたフィールドを保持するのに必要な最小のサイズまで増やされます。

- 「LL」フィールドは、追加フィールドと追加フィールドまでのすべてのフィールドの全長を含むよう更新されます。
- DBD の中の TYPE= パラメーターが、追加されたフィールドまでの、非センシティブ DBD フィールドの充てん値を決定します。

- DBD の中の FIELD ステートメントによって定義されていない追加フィールドまでのスペースの充てん値は 2 進数ゼロです。

以下の図は、 444 ページの『不在フィールドの検索』 の DBD と PSB に基づいたある置き換え呼び出しでの不在フィールドの一例です。

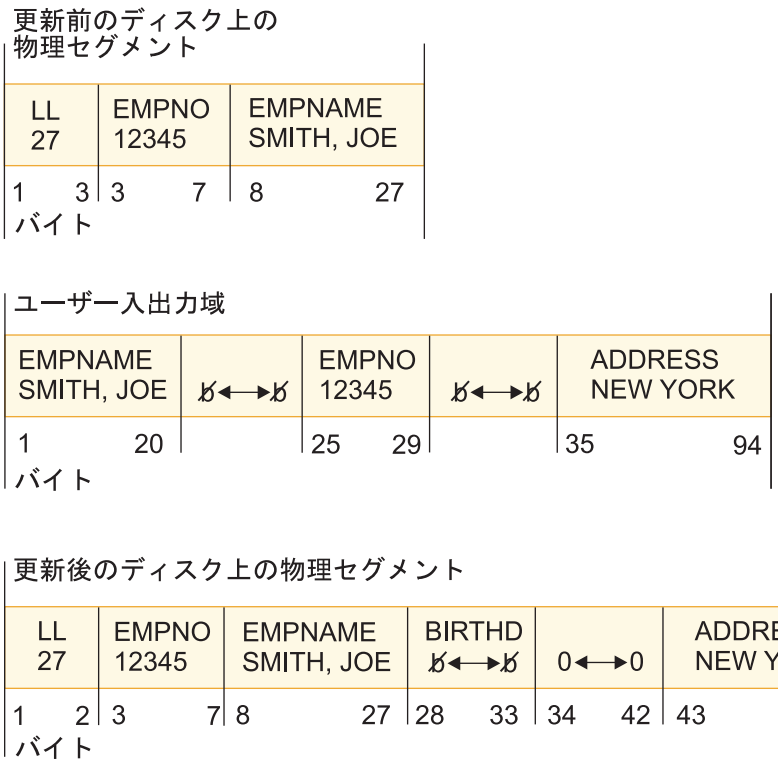


図 *204.* 置き換え呼び出しにおける不在フィールドの *2* 番目の例

「LL」フィールドは IMS によって、ADDRESS フィールドと ADDRESS フィールドに至るすべてのフィールドの全長が入るように保守されます。 BIRTHD は、DBD の中の FIELD ステートメントで TYPE=C が指定されているので、ブランクで充てんされます。スペースが DBD の中の FIELD ステートメントで定義されていないので、位置 34 から 42 までは 2 進数ゼロに設定されます。

関連概念:

444 ページの『不在フィールドの検索』

不在フィールドの挿入

可変長セグメントがデータベースの中に挿入される場合、その長さフィールドは、すべてのセンシティブ・フィールドを保持するのに必要とされる最小サイズの値に設定されます。

- 「LL」フィールドは、すべてのセンシティブ・フィールドを組み込めるように更新されます。
- DBD での TYPE= パラメーターが (可変長セグメントを持つフィールド・レベル・センシティビティーの DBD の例 を参照) 非センシティブ DBD フィールドの充てん値を決定します。

- DBD 中の FIELD ステートメントによって定義されていないスペースの充てん値は 2 進数ゼロです。

以下の図は、可変長セグメントを持つフィールド・レベル・センシティビティーの DBD の例に示した DBD および PSB を使用した挿入呼び出しでの不在フィールドの例です。

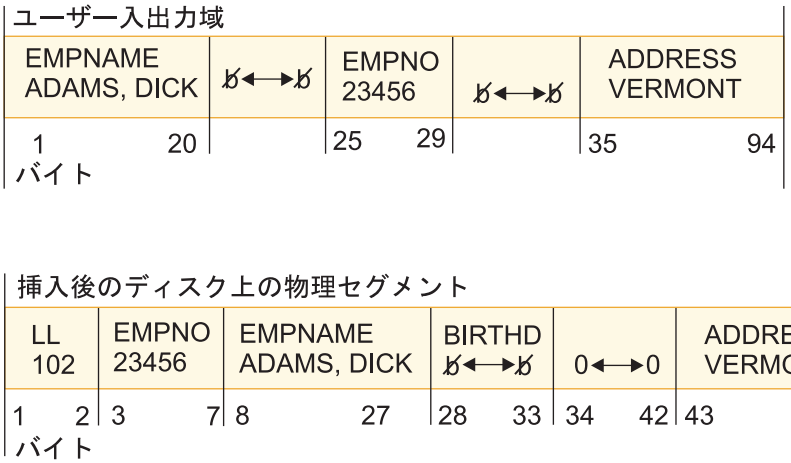


図 *205.* 挿入呼び出しにおける不在フィールドの例

「LL」フィールドは IMS によって、ADDRESS フィールド自身を含めて ADDRESS フィールドに至るすべてのセンシティブ・フィールドの全長が入るように保守されます。 BIRTHD は、DBD の中の FIELD ステートメント上で TYPE=C が指定されているので、ブランクによって充てんされています。位置 34 から 42 までは、DBD の中の FIELD ステートメントでスペースが定義されていないので、2 進数ゼロに設定されます。

部分的に存在するフィールドの検索

検索のとき、可変長物理セグメントの中の最後のフィールドが部分的にしか存在せず、しかもそのデータ・タイプが文字 (TYPE=C) である場合には、データの右側にブランクが埋め込まれてこれがアプリケーション・プログラムに戻されます。そうでない場合には、データ・タイプに基づいた充てん値を持つフィールドがアプリケーション・プログラムに戻されます。

以下の図は、 444 ページの『不在フィールドの検索』 の DBD と PSB に基づいたある検索呼び出しでの部分的に存在するフィールドの一例です。

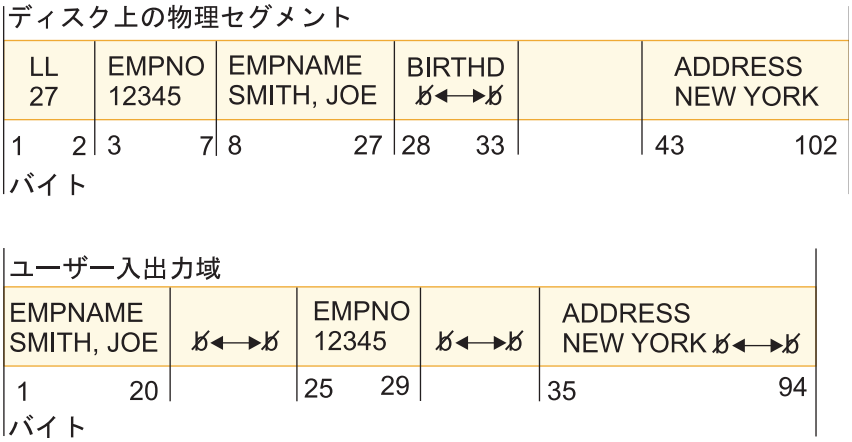


図 *206.* 検索呼び出しにおける部分的に存在するフィールドの例

入出力域では、ADDRESS フィールドにブランクが埋め込まれ、その長さが DBD の中の SEGM ステートメントで定義されている長さと等しくなります。

部分的に存在するフィールドの置き換え

部分的に存在するフィールドを置き換える際には、以下の点に注意してください。

- REPL 呼び出しにおいてセグメント長が増加した場合は、アプリケーション・プログラムに戻されるフィールドが変更されていなければ、そのフィールドがデータベースに書き込まれます。
- フィールドのデータ・タイプが文字であり、しかもこのフィールドが REPL 呼び出しにおいて変更される場合には、必要に応じてセグメント長が増やされて、変更されたデータ内のすべての非ブランク文字が組み込まれるようになります。
- データ・タイプが文字ではなく、このフィールドが REPL 呼び出しで変更される場合には、セグメント長がこのフィールド全体が入るように増やされます。

以下の図は、 444 ページの『不在フィールドの検索』 の DBD と PSB に基づいたある REPL 呼び出しにおける部分的に存在するフィールドの一例です。

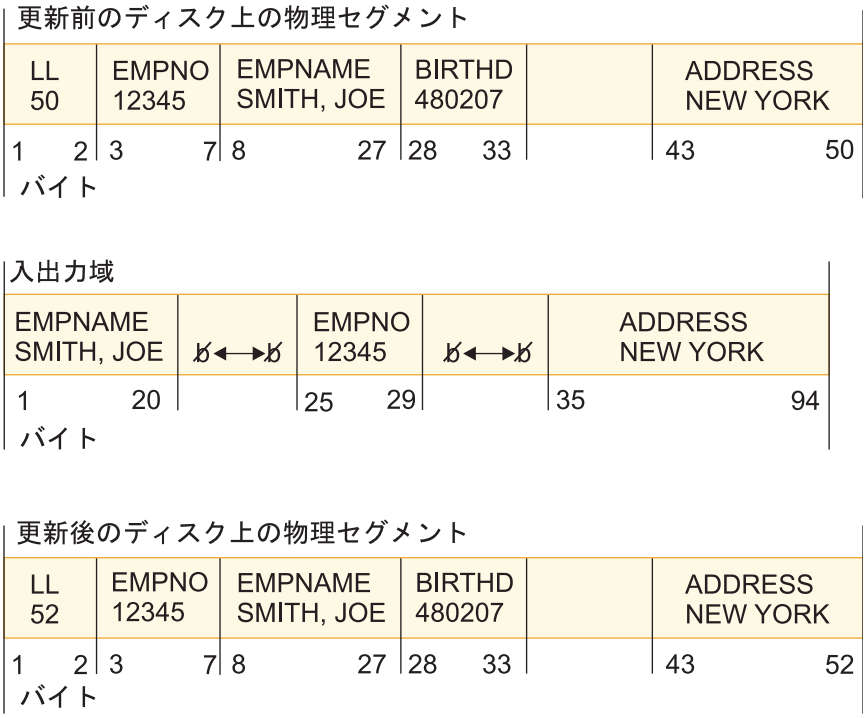


図 *207. REPL* 呼び出しにおける部分的に存在するフィールドの例

ADDRESS のフィールド長の変化に対処するために、「LL」フィールドが DL/I によって 50 から 52 に変更されます。

関連概念:

444 ページの『不在フィールドの検索』

フィールド・レベル・センシティビティーの使用上の一般的な考慮事項

フィールド・レベル・センシティビティーを使用する場合は、以下の一般的な考慮事項に注意してください

- GSAM、MSDB、DEDB の各データベースでは、フィールド・レベル・センシティビティーはサポートされていません。
- PSBGEN の中で SENFLD ステートメントで参照されるフィールドは、DBDGEN の中で FIELD ステートメントで定義されているものでなければなりません。
- 同じ DBD フィールドを複数の SENFLD ステートメントで参照することもできます。
- フィールド・レベル・センシティビティーの使用時には、アプリケーション・プログラムは任意の PCB に関して、セグメントが固定長であっても可変長であっても常にそのセグメントを固定長セグメントと見なします。
- アプリケーション・プログラムは、シーケンス・フィールドを除き、 SSA の中で参照するどのフィールドに対してもセンシティブでなければなりません。
- 挿入またはロードにおいては、シーケンス・フィールドがあれば、アプリケーション・プログラムはシーケンス・フィールドに対してセンシティブでなければなりません。

- 同じ PCB の中でフィールド・レベル・センシティビティーとセグメント・レベル・センシティビティーを併用することができます。
- 挿入操作ないし置き換え操作の間は、参照も定義もされていないフィールドは、必要時に充てん文字としての 2 進数ゼロに設定されます。
- 比較オプションを持つ呼び出しまたはトレースを使用すると、PSB 作業プールで必要なストレージの量が増えます。

複数データ・セット・グループ

HD データベースは複数のデータ・セットに保管することができます。つまり、データベースの保管に必要な 1 つまたは 2 つのデータ・セットよりも多くのデータ・セットに保管することができます。

以下のデータベース・タイプが複数データ・セット・グループをサポートします。

- HDAM
- PHDAM
- HIDAM
- PHIDAM

1 つのデータベースを複数のデータ・セットに保管する場合には、1 次データ・セット・グループと 2 次データ・セット・グループという用語を用いて、このデータベースに対して指定しなければならない 1 つ以上のデータ・セット (1 次データ・セット・グループと呼ばれる) と、このデータベースに対して指定することが許される 1 つ以上のデータ・セット (2 次データ・セット・グループと呼ばれる) とを区別します。

HD データベースでは、これを保管するのに 1 対のデータ・セットではなく、単一のデータ・セットが使用されます。したがって、1 次データ・セット・グループは、このデータベースを保管するために指定しなければならない ESDS (VSAM 使用の場合) または OSAM データ・セット (OSAM 使用の場合) で構成されています。 2 次データ・セット・グループは、このデータベースを保管することが許される追加の 1 つの ESDS または OSAM データ・セットです。

HD データベースでは 10 個までのデータ・セット・グループを使用できます。すなわち、1 つの 1 次データ・セット・グループと最大 9 個の 2 次データ・セット・グループです。

関連タスク:

838 ページの『データ・セット・グループの数の変更』

複数データ・セット・グループを使用する場合

データベース・レコードを設計するときには、多くのアプリケーションの処理要件を満たすような設計を行います。データベース・レコードの中にどのようなセグメントを置くかを決め、データベース・レコード内でのセグメントの階層順を決めます。

これらの決定は、アプリケーション・プログラムの要件のすべてに対して、どうするのが最善であるかということに基づいています。しかし、データベース・レコー

ドの中でのセグメントの配置は、間違いなく、あるアプリケーションの処理要件に対しての方が他のアプリケーションの処理要件に対してよりも適しています。例えば、以下の図の 2 つのデータベース・レコードを見てください。両方ともおなじセグメントが入っていますが、セグメントの階層順は異なっています。

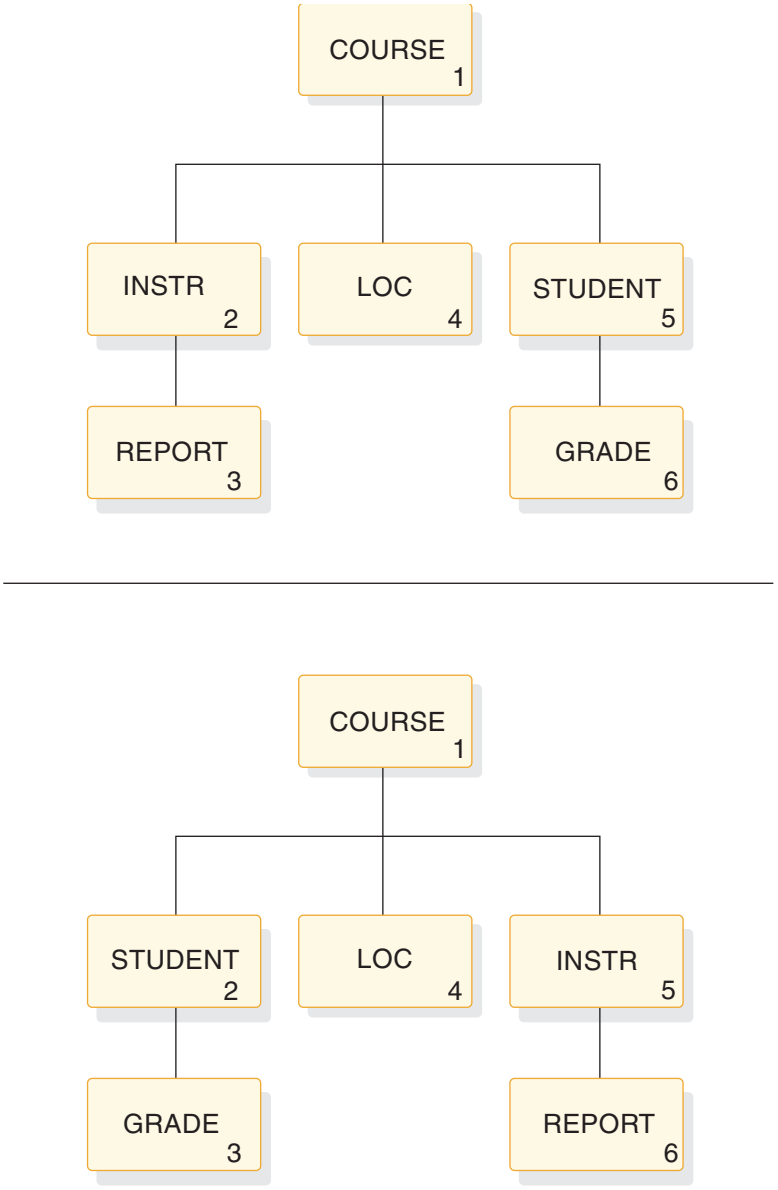


図 *208. INSTR* および *LOC* セグメントをアクセスする必要のあるアプリケーションのための階層

上部に示した階層は、INSTR セグメントと LOC セグメントにアクセスする必要があるアプリケーションに適しています。下部に示した階層は、STUDENT セグメントと GRADE セグメントにアクセスする必要があるアプリケーションに適しています。(適しているというのは、このコンテキストでは、セグメントへのアクセスが速いという意味です。) INSTR セグメントと LOC セグメントにアクセスするアプリケーションの方が、STUDENT セグメントと GRADE セグメントにアクセスするアプリケーションより重要なものであれば、左側のデータベース・レコードを使用します。しかし、両方のアプリケーションが同じように重要なものであれば、この

データベース・レコードを異なるデータ・セット・グループに分割することができます。そうすれば、両方のアプリケーションが、それぞれの必要とするセグメントへ好ましいアクセスを行うことができます。

データベース・レコードを分割するためには、2 つのデータ・セット・グループを使用することになります。以下の図に示すように、最初のデータ・セット・グループには COURSE、INSTR、REPORT、および LOC セグメントが入っています。 2 番目のデータ・セット・グループには STUDENT および GRADE セグメントが入っています。

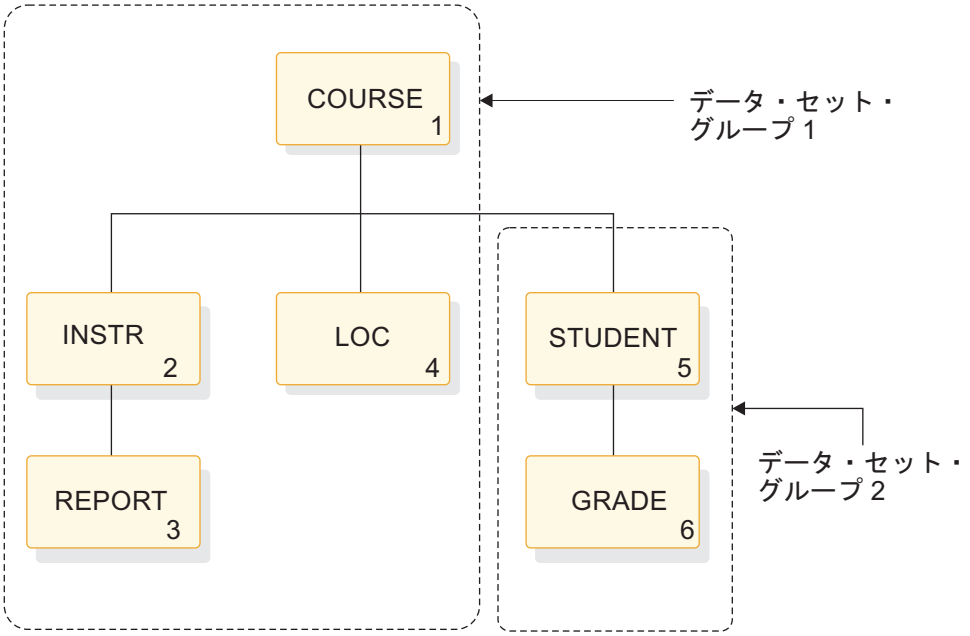


図 *209. 2* つのデータ・セット・グループに分割されたデータベース・レコード

複数データ・セット・グループの他の用途としては、次のようなものがあります。

- あまり使用されないセグメントを頻繁に使用されるセグメントから分離する。
- 頻繁に情報が追加されるセグメントをそうでないセグメントから分離する。前者のセグメントのためには、条件が追加に最適となるよう追加のフリー・スペースを指定することもできます。
- 頻繁に追加または削除されるセグメントをそうでないセグメントから分離する。そうすれば、主なデータベースでスペースがフラグメント化されるのを防止することができます。
- セグメント・サイズが平均のセグメント・サイズから大幅に違っているセグメントを分離する。こうすれば、データベースにおけるスペースの使用効率を向上させることができます。 HD データベースの中のビットマップは、データ・セット・グループの中で定義されている最長 のセグメント・タイプのためのスペースが使用可能かどうかを示していることを覚えていてください。ビットマップは、少量のスペースの存在を追跡しません。大きなセグメント・タイプが 1 つ以上ある場合、より小さいセグメントにとって使用可能なスペースは、ビットマップに追跡されないので、使用されないことになります。

複数データ・セット・グループを使用する HD データベース

複数データ・セット・グループを使用する場合は、データ・セット・グループを 10 個まで定義できます。

データベース・レコードの中のルート・セグメントは、1 次データ・セット・グループの中になければなりません。

以下の図に示すデータベース・レコードでは、セグメント COURSE (1)、INSTR (2)、LOC (4)、および STUDENT (5) を 1 つのデータ・セット・グループに入れ、セグメント REPORT (3) と GRADE (6) をもう 1 つのデータ・セット・グループに入れることができます。

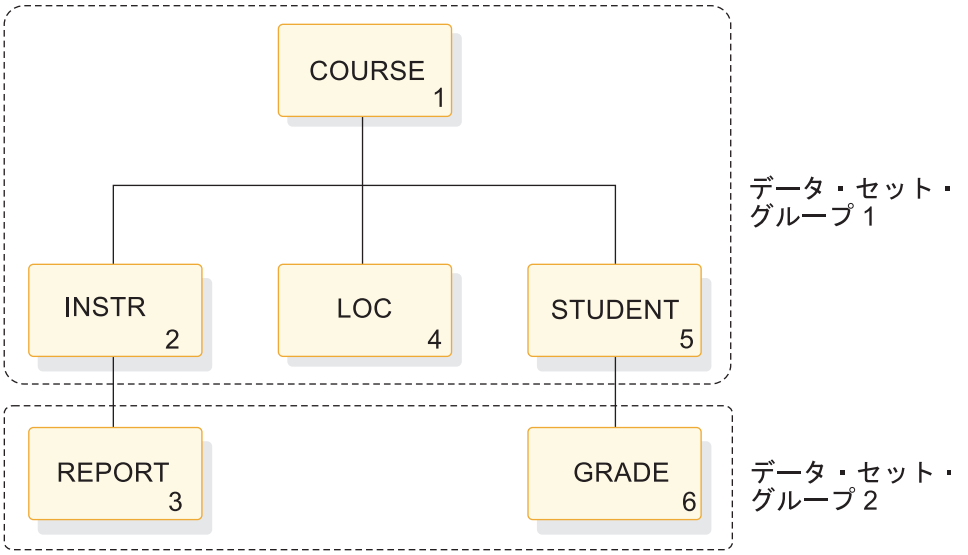


図 *210. HD* データベース・レコードの分割方法の例

上の図に示す HD データベース・レコードを 3 つのグループに分割する例を、以下の表に示します。

表 *64.* 複数データ・セット・グループの例

データ・セット・グループ **1**	データ・セット・グループ **2**	データ・セット・グループ **3**
セグメント 1	セグメント 2、5、6	セグメント 3、4
セグメント 1、3、6	セグメント 2、5	セグメント 3
セグメント 1、3、6	セグメント 2、5	セグメント 4

異なるデータ・セット・グループに分けられたセグメントは、物理第 1 子ポインターによって接続する必要があります。例えば、以下の図では、1 次データ・セット・グループの中の INSTR セグメントは、2 次データ・セット・グループの中でその物理子 REPORT の最初のオカレンスを指さなければならず、また STUDENT は GRADE を指さなければなりません。

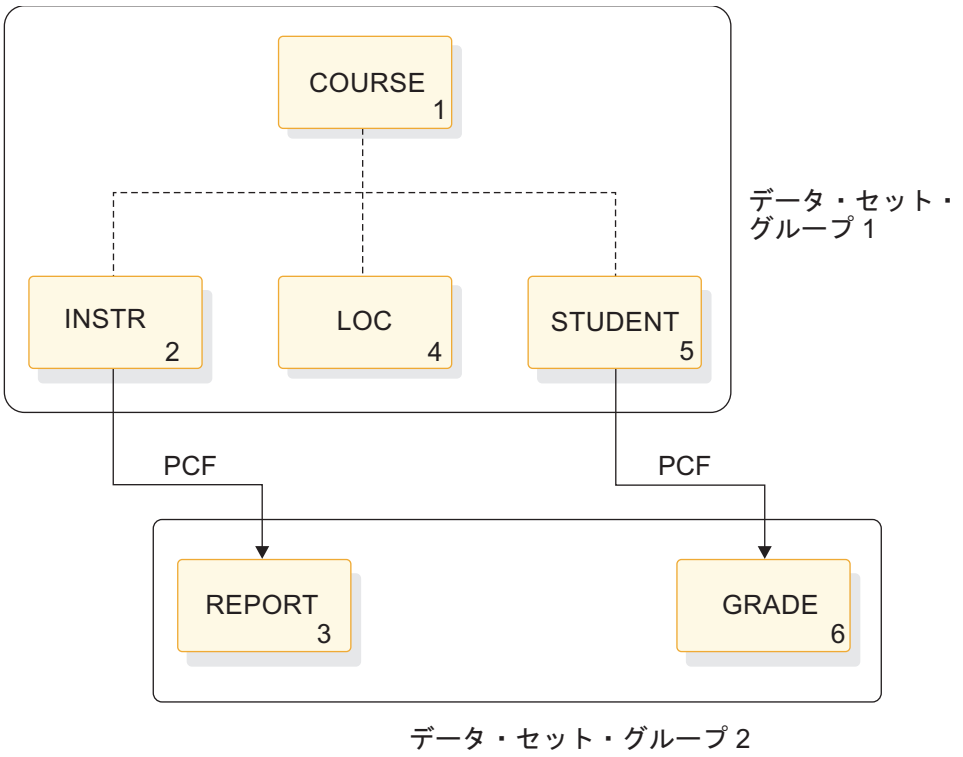


図 *211.* 物理第 *1* 子ポインターを用いた複数データ・セット・グループ内のセグメントの接続

複数データ・セット・グループの中への HD レコードの保管方法

DBD ステートメントで、データ・セット・グループにどのセグメント・タイプを保管するかを指定します。すると、IMS がセグメントを適切なデータ・セット・グループにロードします。

以下の図は、1 つのデータベース・レコードを示したものです。

- 2 つのデータ・セット・グループを用いて HDAM または PHDAM データベースに保管されている。
- 2 つのデータ・セット・グループを用いて HIDAM または PHIDAM データベースに保管されている。

この例では、ユーザーがこのデータベース・レコードの中の 4 つのセグメント・タイプ (COURSE、INSTR、 LOC、STUDENT) を 1 次データ・セット・グループに入れ、2 つのセグメント・タイプ (REPORT、GRADE) を 2 次データ・セット・グループに入れるよう指定しています。

HDAM または PHDAM データベースは、1 次データ・セット・グループのみがルート・アドレス可能域を持つという点に注意してください。 2 次データ・セット・グループは追加のオーバーフロー・ストレージです。

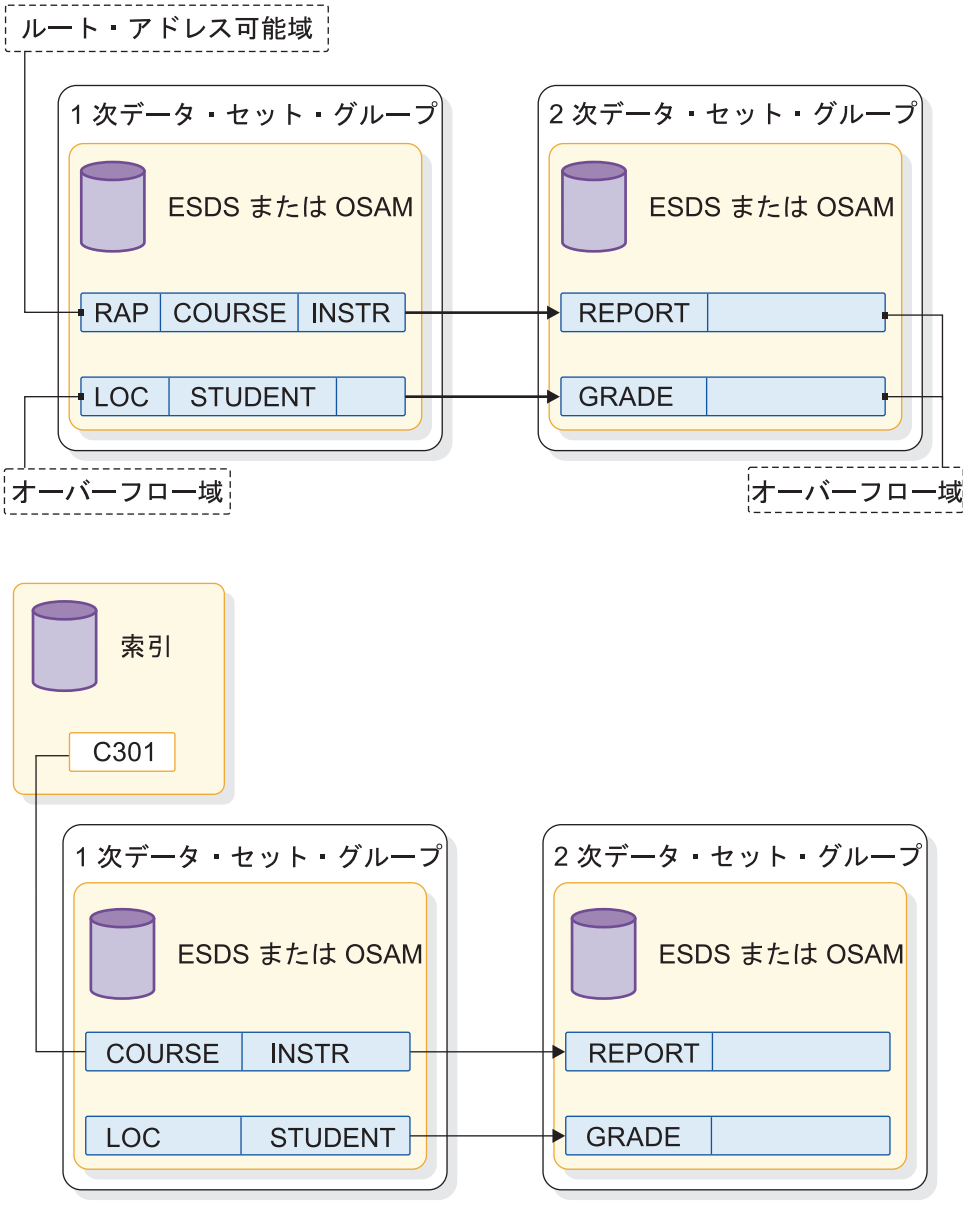


図 *212.* 複数データ・セット・グループを用いたときのストレージ内の *HD* データベース・レコード

HD および PHD データベースにおける複数データ・セット・グループの使用の指定

DBD において、複数データ・セット・グループを IMS に対して指定することができます。 HDAM データベースの場合、DATASET ステートメントを使用します。 PHDAM データベースの場合、SEGM ステートメントの DSGROUP パラメーターを使用します。

任意の方法でセグメントをグループ化することができますが、DBD においてセグメントを階層順にリストする必要があります。

以下の例では、 450 ページの『複数データ・セット・グループを使用する場合』と 453 ページの『複数データ・セット・グループを使用する HD データベース』で使用したデータベース・レコードを使用します。最初の例 (以下の図) は、次の 2 つのデータ・セット・グループを示しています。データ・セット・グループ A には COURSE と INSTR が含まれており、データ・セット・グループ B には他のすべてのセグメントが含まれています。 2 番目の例は別のグループ化を示しています。グループがセグメントの階層順になっていないときの DBD での違いに注意してください。

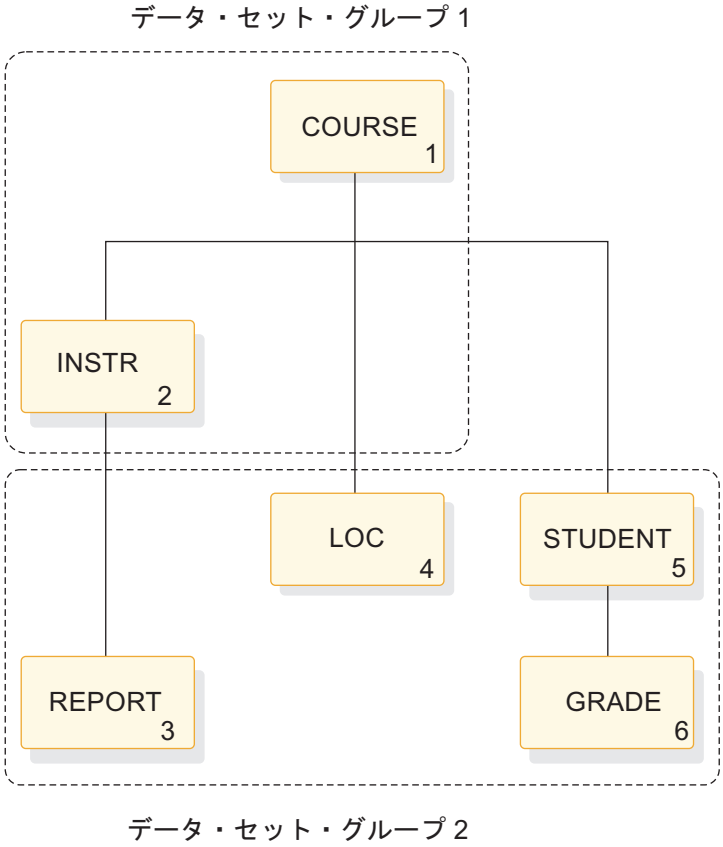


図 *213.* データ・セット・グループの最初の例

以下のコードは、最初の例の HDAM DBD です。セグメントが DATASET ステートメントとその後の SEGM ステートメントによってグループ化されていること、およびセグメントが階層順にリストされていることに注意してください。各 DATASET ステートメントでは、DD1= パラメーターが、使用される VSAM ESDS または OSAM データ・セットの名前を指定しています。また、各データ・セット・グループはそれぞれ独自の特性 (例えば、装置タイプなど) を持つことができます。

```
DBD    NAME=HDMDSG,ACCESS=HDAM,RMNAME=(DFSHDC40,8,500)
DSA    DATASET  DD1=DS1DD
SEGM   NAME=COURSE,BYTES=50,PTR=T
FIELD  NAME=(CODCOURSE,SEQ),BYTES=10,START=1
SEGM   NAME=INSTR,BYTES=50,PTR=T,PARENT=((COURSE,SNGL))
DSB    DATASET  DD1=DS2DD,DEVICE=2314
SEGM   NAME=REPORT,BYTES=50,PTR=T,PARENT=((INSTR,SNGL))
```

```
 SEGM   NAME=LOC,BYTES=50,PTR=T,PARENT=((COURSE,SNGL))
 SEGM   NAME=STUDENT,BYTES=50,PTR=T,PARENT=((COURSE,SNGL))
 SEGM   NAME=GRADE,BYTES=50,PTR=T,PARENT=((STUDENT,SNGL))
 DBDGEN
```

以下のコードは PHDAM データベース用の DBD を示しています。 DATASET ステートメントを使用せずに、SEGM ステートメントの DSGROUP パラメーターを使用します。最初の 2 つのセグメントは最初のグループに含まれるものと想定されるため、これらのセグメントには DSGROUP パラメーターはありません。

```
DBD    NAME=HDMDSG,ACCESS=PHDAM,RMNAME=(DFSHDC40,8,500)
SEGM   NAME=COURSE,BYTES=50,PTR=T
FIELD  NAME=(CODCOURSE,SEQ),BYTES=10,START=1
SEGM   NAME=INSTR,BYTES=50,PTR=T,PARENT=((COURSE,SNGL))
SEGM   NAME=REPORT,BYTES=50,PTR=T,PARENT=((INSTR,SNGL)),DSGROUP=B
SEGM   NAME=LOC,BYTES=50,PTR=T,PARENT=((COURSE,SNGL)),DSGROUP=B
SEGM   NAME=STUDENT,BYTES=50,PTR=T,PARENT=((COURSE,SNGL)),DSGROUP=B
SEGM   NAME=GRADE,BYTES=50,PTR=T,PARENT=((STUDENT,SNGL)),DSGROUP=B
DBDGEN
```

2 番目の例 (以下の図) は、グループが階層順になっていないという点で最初の例と異なります。 DBD ではセグメントを階層順にリストする必要があるため、追加の DATASET ステートメントまたは DSGROUP パラメーターが必要です。

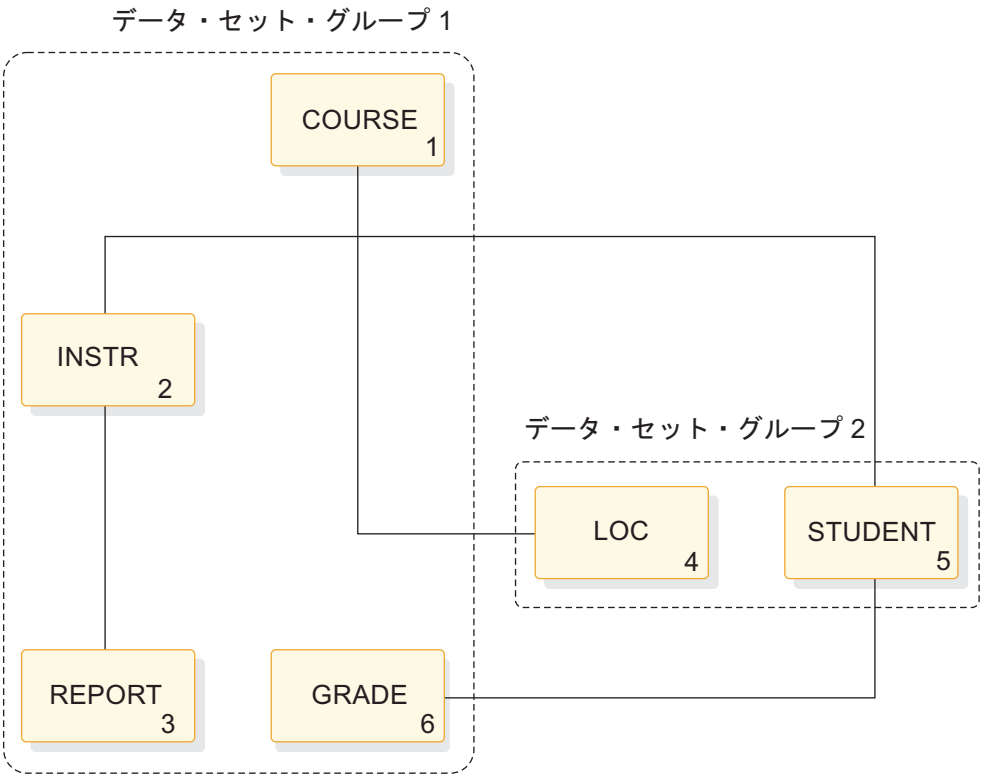


図 *214.* データ・セット・グループの *2* 番目の例

以下のコードは 2 番目の例の HDAM データベース用の DBD です。これは最初の例に類似していますが、6 番目のセグメントが最初のグループに含まれているため、DSA ラベルを持つ別の DATASET ステートメントが 6 番目のセグメントの前に必要である点が異なります。追加の DATASET ラベルは、6 番目のセグメントを最初の 3 つのセグメントと一緒にグループ化します。

```
DBD    NAME=HDMDSG,ACCESS=HDAM,RMNAME=(DFSHDC40,8,500)
DSA    DATASET  DD1=DS1DD
SEGM   NAME=COURSE,BYTES=50,PTR=T
FIELD  NAME=(CODCOURSE,SEQ),BYTES=10,START=1
SEGM   NAME=INSTR,BYTES=50,PTR=T,PARENT=((COURSE,SNGL))
SEGM   NAME=REPORT,BYTES=50,PTR=T,PARENT=((INSTR,SNGL))
DSB    DATASET  DD1=DS2DD,DEVICE=2314
SEGM   NAME=LOC,BYTES=50,PTR=T,PARENT=((COURSE,SNGL))
SEGM   NAME=STUDENT,BYTES=50,PTR=T,PARENT=((COURSE,SNGL))
DSA    DATASET  DD1=DS1DD
SEGM   NAME=GRADE,BYTES=50,PTR=T,PARENT=((STUDENT,SNGL))
DBDGEN
```

以下のコードは 2 番目の例の PHDAM データベース用の DBD です。これは最初の例に類似していますが、6 番目のセグメントが最初のグループに含まれているため、DSGROUP パラメーターを使用して明示的に 6 番目のセグメントを最初の 3 つのセグメントと一緒にグループ化する必要があります。

```
DBD    NAME=HDMDSG,ACCESS=PHDAM,RMNAME=(DFSHDC40,8,500)
SEGM   NAME=COURSE,BYTES=50,PTR=T
FIELD  NAME=(CODCOURSE,SEQ),BYTES=10,START=1
SEGM   NAME=INSTR,BYTES=50,PTR=T,PARENT=((COURSE,SNGL))
SEGM   NAME=REPORT,BYTES=50,PTR=T,PARENT=((INSTR,SNGL)),
SEGM   NAME=LOC,BYTES=50,PTR=T,PARENT=((COURSE,SNGL)),DSGROUP=B
SEGM   NAME=STUDENT,BYTES=50,PTR=T,PARENT=((COURSE,SNGL)),DSGROUP=B
SEGM   NAME=GRADE,BYTES=50,PTR=T,PARENT=((STUDENT,SNGL)),DSGROUP=A
DBDGEN
```

関連タスク:

584 ページの『HALDB 区画定義ユーティリティーによる HALDB データベースの作成』

関連資料:

DATASET ステートメント (システム・ユーティリティー)

全機能データベースのための VSAM KSDS CI レクラメーション処理

データ共用または XRF 環境では、IMS は、CI レクラメーション処理と呼ばれるプロセスを使用することによって、全機能データベース内で空の VSAM KSDS 制御インターバル (CI) に使用されたストレージを再利用することができます。

CI レクラメーション処理は、後続の DL/I 呼び出し時に VSAM が読む空の CI の数を減らすことにより、データベースの GN 呼び出し、GU 呼び出し、および ISRT 呼び出しのパフォーマンスを向上させます。

IMS は CI の最後のレコードの削除がコミットされると、IMS はその CI にレクラメーション処理のフラグを立てます。次にその CI が、更新アクセス権を持つ DL/I 呼び出し用に読み取られるとき、IMS はその CI のレクラメーション処理を行います。単一 DL/I 呼び出しでは、IMS は、その DL/I 呼び出しが完了するまでに IMS が読む空の CI のうち、レクラメーション処理のフラグが立てられているすべての CI にレクラメーション処理を行います。

CI レクラメーション処理は、定期的な KSDS データ・セットの再編成に代わるものではありません。

CI レクラメーション処理は、以下の状況のもとでのみ有効です。

- VSAM サブプールを使用する場合、DFSVSMxx PROCLIB メンバーの DBD ステートメントに ERASE=YES を指定したか、またはこれをデフォルトとして受け入れた。
- ご使用の KSDS が固有キーを使用している。
- GN 呼び出し、GU 呼び出し、または ISRT 呼び出しを発行するアプリケーション・プログラムが、対象のデータベースに対して更新アクセス権を持つ。処理オプション PROCOPT に「I」、「R」、「D」、または「A」を使用した場合、そのデータ・セットは更新用にオープンされることが保証されます。

制約事項**:** SHISAM データベースは CI レクラメーション処理をサポートしていません。SHISAM データベースで大量のレコードが削除されると、特に連続した大量のレコードの場合、重大なパフォーマンスの低下が発生する可能性があります。VSAM REPRO を使用して、空の CI を除去し、問題を解決してください。

関連資料**:** VSAM REPRO コマンドについては、「*z/OS DFSMS* カタログのためのアクセス方式サービス・プログラム」を参照してください。

関連概念:

707 ページの『データベースの再編成』

IMS 環境でのデータ共用 (システム管理)

拡張回復機能の概要 (システム管理)

IMS データベースへの XML データの保管

IMS バージョン 12 は、IMS XML DB 機能をサポートする最後のリリースです。この機能により、Java アプリケーション・プログラムを使用して IMS データベースに XML 文書を保管したり、データベースから取得したりすることができます。IMS バージョン 12 がサービス休止になると、IMS バージョン 13 での IMS XML DB のサポートは終了します。

XML 文書の保管と検索を行う場合、その XML 文書は IMS Enterprise SuiteDLIModel ユーティリティー・プラグイン で生成された XML スキーマに対して有効なものでなければなりません。XML スキーマは、IMS データベースの階層構造に一致している必要があります。

XML 文書は、次の 2 つの保管方式をその XML 文書の構造に最も適した任意の組み合わせで使用して、IMS データベースに保管することができます。

XML の分解保管

XML タグが XML 文書から除去され、データのみが抽出されます。抽出されたデータは、従来の IMS フィールド・タイプに変換され、データベースに挿入されます。この方法は、次のシナリオで使用してください。

- XML アプリケーションおよび非 XML アプリケーションが、同じデータベースにアクセスしなければならない。
- 必要なデータベース検索の量が非常に多い。
- 厳密な XML スキーマが使用可能。

XML の完全保管

XML 文書は、完全な XML 文書の保管専用に設計された IMS データベー

スに、XML 構造とタグが完全な形のままで保管されます。この場合、Java アプリケーション・プログラムのみがそのデータベース内のデータにアクセスできます。データベースからデータを検索する場合に XML 文書の再生成が必要ないため、XML タグが除外された保管の場合より、通常は XML データの検索が高速になります。この方法は、次のシナリオで使用してください。

- XML 文書の高速の保管および検索が必要。
- 必要なデータベース検索の量がそれほど多くない。
- XML スキーマに柔軟性が必要。

関連概念:

461 ページの『第 19 章 IMS データベース内の XML 保管』

第 19 章 IMS データベース内の XML 保管

XML および IMS データベースはともに階層データベースであるため、IMS は、XML 文書を管理するのに効果的なデータベース管理システムです。

IMS では、着信する XML 文書の受信と保管、および IMS データベースに保管されている既存の情報からの XML 文書の構成を簡単に行うことができます。

例えば、次のことが行えます。

- 例えば、企業間オンデマンド・トランザクションおよびデータの組織内共用をサポートするために、既存の IMS データベースのすべてのタイプから XML 文書を構成する。
- 着信 XML 文書を受信し、既存または新規の IMS データベース内に保管する。

IMS では、XML 文書を原形保管モード か分解保管モード、またはその 2 つの組み合わせで保管できます。

分解保管モードでは、IMS は XML 文書を構文解析し、エレメントのデータと属性を通常の IMS データとしてセグメント・フィールドに保管します。分解保管モードは、データ中心の文書に適しています。

原形保管の場合、着信文書は、その XML タグも含めてデータベースに直接に保管され、IMS はその構造を認識しません。原形保管は、文書中心の XML 文書に適しています。

XML を IMS データベースに保管したり、IMS から検索したりするには、まず 2 つの成果物、つまり XML スキーマと IMS 用の Java メタデータ・クラスを生成する必要があります。 Java メタデータ・クラスは、IMS Enterprise Suite Explorer for Development を使用して生成できます。XML スキーマは、手作業で生成するか、IMS Enterprise Suite DLIModel ユーティリティー・プラグインを使用して生成できます。メタデータおよびスキーマは、XML の保管および検索の際に使用されます。

アプリケーションは、IMS Universal タイプ 4 JDBC ドライバーを使用して XML を IMS データベースに保管し、IMS データから XML を作成し、IMS データベースから XML 文書を検索します。あるいは、IMS のクラシック JDBC ドライバーを使用している場合は、ユーザー定義の IMS 機能 `storeXML` および `retrieveXML` を使用することもできます。

以下の図に、IMS での XML の保管および検索についてのプロセスを示します。

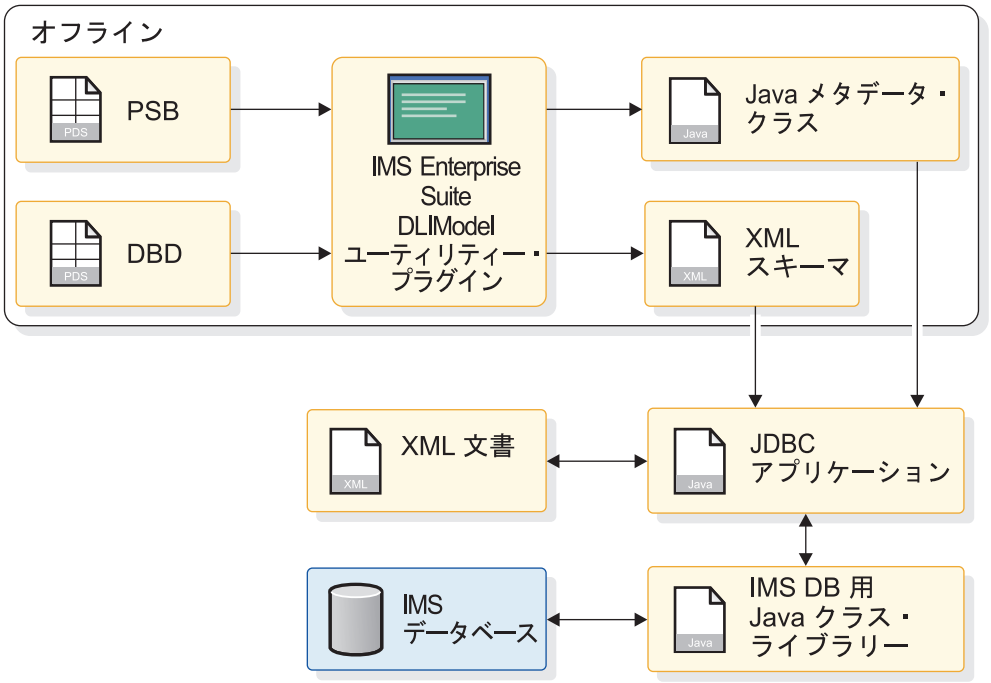


図 *215. IMS* への *XML* の保管の概要

関連概念:

459 ページの『IMS データベースへの XML データの保管』

XML の分解保管モード

分解保管モード では、すべてのエレメントおよび属性が、オプションで DL/I セグメントを繰り返す通常のフィールドとして保管されます。

構文解析の際、すべてのタグおよびその他の XML 構文情報は、妥当性を検査されてから廃棄されます。構文解析されたデータは、標準の IMS データとして物理的にデータベースに保管されます。つまりセグメント内の各定義済みフィールドは、IMS の標準タイプのフィールドです。XML データはすべて、妥当性検査 XML スキーマ内にタイプ情報が存在するストリング・タイプ (一般的にはユニコード) から構成されているため、構文解析された各データ・エレメントおよび属性は、対応する IMS 標準フィールド値に変換して、ターゲット・データベースに保管できます。

逆に、XML の検索の際は、DL/I セグメントの検索、フィールドの宛先 XML エンコード方式への変換、タグおよび XML 構文情報 (XML スキーマ内に保管) の追加、ならびに XML 文書の構成が行われます。

以下の図に、XML エレメントの分解方法および IMS セグメントへの保管方法を示します。

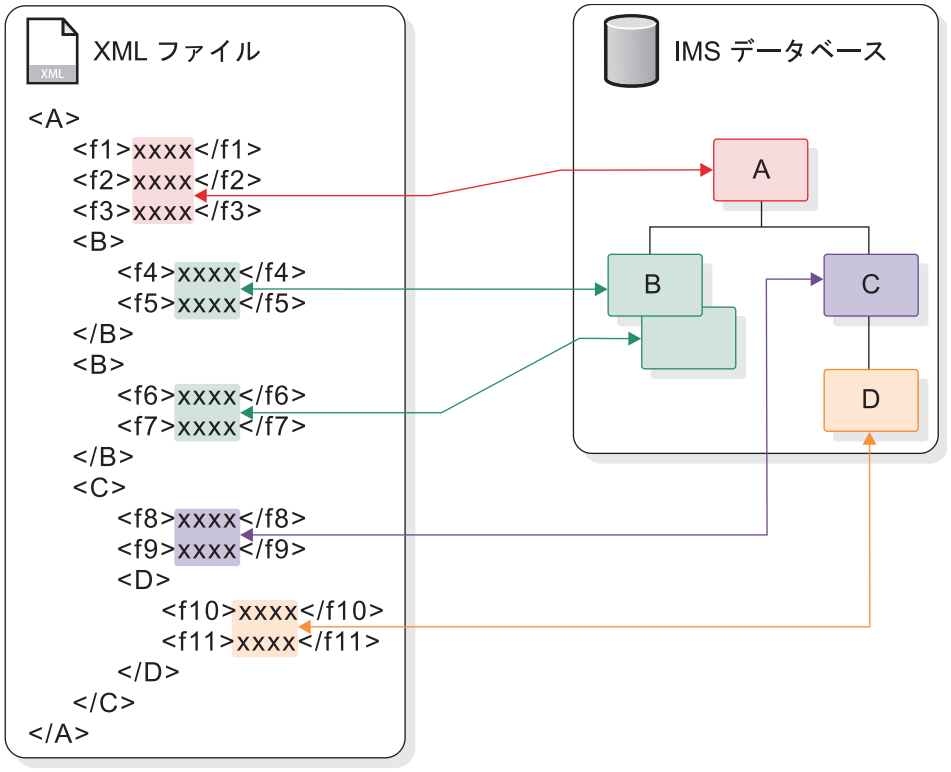


図 *216. XML* 分解方法および *IMS* セグメントへの保管方法

分解保管モード、はデータ中心の XML 文書に適しています。データ中心の XML 文書では、文書のエレメントおよび属性は既知の長さの文字項目か数値項目であることが一般的であり、比較的に簡単にセグメント内のフィールドにマップできます。長さは、短いか中位の長さで、一般的には (必ずというわけではありませんが) 固定されています。

XML 文書データは、IMS セグメント階層内の任意のセグメントから開始できます。開始セグメントは、XML 文書内のルート・エレメントです。開始セグメントの下にあるサブツリー内のセグメントも、XML 文書に含まれています。 XML 文書のエレメントおよび属性は、ルート・エレメント・セグメントの従属セグメントに保管されます。そのルート・エレメント・セグメントの従属セグメントではない、階層内のその他のセグメントは、いずれも XML 文書の一部ではなく、したがって、関連する XML スキーマには含まれません。

XML 文書がデータベースに保管される際、すべてのセグメント・フィールドの値が、XML 文書から直接取り出されます。したがって、すべての XML セグメント内のすべての固有キー・フィールドは、属性または単純なエレメントとして、XML 文書内に存在する必要があります。

XML 階層は、物理データベースまたは論理データベースのいずれかに基づく PCB 階層によって定義されます。 論理関係は、XML 文書の検索および構成の場合はサポートされますが、文書を挿入する場合はサポートされません。

非 XML データベースの場合は、データベース階層全体も階層のすべてのサブツリーも、分解されるデータ中心の XML 文書と見なすことができます。分解される

XML データを構成するセグメントおよびフィールドは、これらのセグメントとフィールドおよび文書間のマッピング (XML スキーマ) の定義によってのみ決められます。

データベース PCB ごとに 1 つの XML スキーマが生成されます。したがって、異なる XML スキーマによって、複数の文書を物理データベース階層から派生させることができます。これらの複数の文書のオーバーラップ、ならびに共通セグメントまたはフィールドの共用方法には制限がありません。

新規データベースは、特に、特定タイプのデータ中心の XML 文書を分解形式で保管するように設計できます。

XML の原形保管モード

原形保管モード では、XML 文書のすべてまたは一部が、フィールド内にそのまま保管されます。XML タグは除去されず、IMS は、文書の構文解析を行いません。

XML 文書は、基本原形フィールド (XML ルート・エレメントを含む) およびオーバーフロー・セグメント内のフィールドをまたがる大きさであってもかまいません。原形の XML 文書を含むセグメントは、標準の IMS セグメントであり、他のすべての IMS セグメントと同様に処理できます。フィールドは、構文解析されない XML データを含んでいるため、標準の IMS フィールドとして処理することはできません。しかし、文書の原形の保管には、次のような分解保管モードに勝る利点があります。

- IMS は、保管および検索の際に、XML の構成も分解も必要ありません。したがって、原形 XML 文書は分解 XML 文書より速く処理できます。
- XML 文書の内容を、IMS フィールドのデータ・タイプまたは長さと一致させる必要はありません。したがって、同じ IMS データベース内で、XML 文書を異なる構造、内容、および長さで保管できます。

原形の XML を保管するには、 XML 文書を、特に原形の XML の保管用に調整したセグメントおよびフィールドに保管する必要があるため、新しい IMS データベース、または既存のデータベースの拡張が必要です。

XML 文書のすべてまたは一部をそのまま IMS データベース内に保管する場合、データベースは、原形の XML サブツリーのルート・エレメントが入る基本セグメントを定義する必要があります。原形の XML サブツリーの残りは、基本セグメントの子セグメントであるオーバーフロー・セグメントに保管されます。

基本セグメントには、原形の XML サブツリーのルート・エレメントと、任意の分解されたフィールドまたは非 XML フィールドが含まれます。以下の表に、基本原形フィールドのフォーマットを示します。このフォーマットは DBD で定義されています。

表 *65.* 基本原形フィールド・フォーマット

バイト	内容
1	0x01
2	予約済み

表 65. 基本原形フィールド・フォーマット (続き)

バイト	内容
3–4	ビット **1** オーバーフロー・セグメントの有無を示します。 ビット **2** から **16** このフィールドの XML データの長さを示します。
残りのフィールド	XML データ

オーバーフロー・セグメントに含まれるのは、オーバーフロー XML データ・フィールドのみです。以下の表に、オーバーフロー XML データ・フィールドのフォーマットを示します。このフォーマットは DBD で定義されています。

表 66. オーバーフロー *XML* データ・フィールド・フォーマット

バイト	内容
1–2	キー・フィールド・シーケンス番号
2–4	ビット **1** このセグメントの直後にさらにオーバーフロー・セグメントがあるかどうかを示します。 ビット **2** から **16** このフィールドの XML データの長さを示します。
残りのフィールド	XML データの継続

原形 XML 保管用の DBD

このトピックの例では、原形 XML 文書を保管するために使用される DBD ステートメントを示します。最初の例では、IMS Universal JDBC ドライバーを使用しています。 2 番目の例では、IMS クラシック JDBC ドライバーを使用します。

IMS Universal JDBC ドライバー用に原形 XML を保管するための DBD 構造

次の DBD ステートメントの例では、原形 XML を保管するデータベースを定義しています。このデータベースは、タイプ 4 接続機能を備えた IMS Universal JDBC ドライバーを使用するアプリケーション・プログラムによってアクセスされます。

```
         DBD      NAME=DH41SK01,ACCESS=(HIDAM,OSAM)
DSG01    DATASET  DD1=HIDAMD1,DEVICE=3390,BLOCK=1024
*
    SEGM    NAME=HOSPITAL,                                          C
               PARENT=0,                                            C
               BYTES=(900),                                         C
               RULES=(LLL,HERE)
      FIELD NAME=(HOSPCODE,SEQ,U),                                  C
               START=3,                                             C
               BYTES=12,                                            C
               TYPE=C
      FIELD NAME=(HOSPNAME),                                        C
               START=15,                                            C
               BYTES=17,                                            C
               TYPE=C
```

```
      FIELD NAME=(HOSPLL),                                          C
               START=1,                                             C
               BYTES=2,                                             C
               TYPE=X
      LCHILD NAME=(INDEX,DX41SK01),PTR=INDX
*
*
*********************************************************************
*        SEGMENT NUMBER 2
*********************************************************************
    SEGM    NAME=PAYMENTS,                                          C
               PARENT=HOSPITAL,                                     C
               BYTES=(900),                                         C
               TYPE=DIR,                                            C
               RULES=(LLL,LAST)
      FIELD NAME=(PATMLL),                                          C
               START=1,                                             C
               BYTES=2,                                             C
               TYPE=X
      FIELD NAME=(PATNUM),                                          C
               START=3,                                             C
               BYTES=4,                                             C
               TYPE=C
      FIELD NAME=(AMOUNT),                                          C
               START=7,                                             C
               BYTES=8,                                             C
               TYPE=C
*********************************************************************
*        SEGMENT NUMBER 3
*********************************************************************
DSG02    DATASET  DD1=HIDAMD2,DEVICE=3380,BLOCK=1024
*
    SEGM    NAME=WARD,                                              C
               PARENT=HOSPITAL,                                     C
               BYTES=(900),                                         C
               TYPE=DIR,                                            C
               RULES=(LLL,LAST)
      FIELD NAME=(WARDNO,SEQ,U),                                    C
               START=3,                                             C
               BYTES=4,                                             C
               TYPE=C
      FIELD NAME=(WARDINFO),                                        C
               START=7,                                             C
               BYTES=100,                                           C
               TYPE=C
      FIELD NAME=(WARDLL),                                          C
               START=1,                                             C
               BYTES=2,                                             C
               TYPE=X
*********************************************************************
*        SEGMENT NUMBER 4
*********************************************************************
    SEGM    NAME=OFSEG,                                             C
               PARENT=WARD,                                         C
               BYTES=(900),                                         C
               TYPE=DIR,                                            C
               RULES=(LLL,HERE)
      FIELD NAME=(SEQNO,SEQ,U),                                     C
               START=1,                                             C
               BYTES=2,                                             C
               TYPE=C
    DBDGEN
    FINISH
    END
```

IMS クラシック JDBC ドライバー用に原形 XML を保管するための DBD 構造

次の DBD ステートメントの例では、基本セグメントとオーバーフロー・セグメントを定義しています。基本セグメントの XML 原形フィールドには、4 バイトのヘッダーが含まれるので、フィールドを 4 バイトより大きく定義する必要があります。オーバーフロー・セグメントの XML 原形フィールドには、長さが 2 バイトのヘッダーが含まれるので、フィールドを 2 バイトより大きく定義する必要があります。このデータベースは、IMS クラシック JDBC ドライバーを使用するアプリケーション・プログラムによってアクセスされます。

図 *217.* 原形 *XML* 保管および副次索引なしの *DBD*

```
DBD     NAME=dbdname,ACCESS=(PHDAM,VSAM),RMNAME=(DFSHDC40,1,5,bytes)
*Base segment
SEGM    NAME=segname1,PARENT=0,BYTES=seglen1
* XML intact field, which contains a 4-byte header
FIELD   NAME=INTDATA,BYTES=length,START=startpos,TYPE=C
* Additional non-intact fields can be specified in segment
*
* Overflow Segment
SEGM    NAME=segname2,PARENT=segname1,BYTES=seglen2
FIELD   NAME=(SEQNO,SEQ,U),BYTES=2,START=1,TYPE=C
* XML intact field, which contains a 2-byte header for length
FIELD   NAME=INTDATA,BYTES=1,START=3,TYPE=C
DBDGEN
FINISH
END
```

次の DBD ステートメントの例では、基本セグメントとオーバーフロー・セグメントを定義し、2 つの副次索引によって使用されるサイド・セグメントを定義しています。

図 *218.* 原形 *XML* 保管と *2* つの副次索引のための *DBD* ステートメント

```
DBD     NAME=dbdname,ACCESS=(PHDAM,VSAM),RMNAME=(DFSHDC40,1,5,200)
* Base segment
SEGM    NAME=segname1,PARENT=0,BYTES=seglen1
* XML intact field, which contains a 4-byte header
FIELD   NAME=INTDATA,BYTES=length,START=startpos,TYPE=C
*
LCHILD  NAME=(issegname1,isdbd1),POINTER=INDX
XDFLD   NAME=issrch1,SRCH=iskey1,SEGMENT=ssegname1
LCHILD  NAME=(issegname2,isdbd2),POINTER=INDX
XDFLD   NAME=issrch2,SRCH=iskey2,SEGMENT=ssegname2
* Overflow segment
SEGM  NAME=segname2,PARENT=segname1,BYTES=seglen2
FIELD   NAME=(SEQNO,SEQ,U),BYTES=2,START=1,TYPE=C
* XML intact field, which contains a 2-byte header for length
FIELD   NAME=INTDATA,BYTES=1,START=3,TYPE=C
*
* Index side segment 1
SEGM    NAME=ssegname1,PARENT=segname1,BYTES=iseglen1
FIELD   NAME=(iskey1,SEQ,U),BYTES=islen1,START=1,TYPE=C
*
* Index side segment 2
SEGM    NAME=ssegname2,PARENT=segname1,BYTES=iseglen2
FIELD   NAME=(iskey2,SEQ,U),BYTES=islen2,START=1,TYPE=C
```

```
*
DBDGEN
FINISH
END
```

次の DBD ステートメントの例では、 467 ページの図 218 で定義されたデータベースの最初の副次索引を定義しています。

図 *219.* 原形 *XML* 保管用の最初の副次索引 *DBD* ステートメント

```
DBD    NAME=isdbd1,ACCESS=(PSINDEX,VSAM)

SEGM   NAME=issegname1,PARENT=0,BYTES=iseglen
FIELD  NAME=(isfld1,SEQ,U),BYTES=islen1,START=1,TYPE=C
LCHILD NAME=(ssegname1,dbdname),INDEX=issrch1
DBDGEN
FINISH
END
```

次の DBD ステートメントの例では、 467 ページの図 218 で定義されたデータベースの 2 番目の副次索引を定義しています。

図 *220.* 原形 *XML* 保管用の *2* 番目の副次索引 *DBD* ステートメント

```
DBD    NAME=isdbd2,ACCESS=(PSINDEX,VSAM)

SEGM   NAME=issegname2,PARENT=0,BYTES=iseglen
FIELD  NAME=(isfld2,SEQ,U),BYTES=islen2,START=1,TYPE=C
LCHILD NAME=(ssegname2,dbdname),INDEX=issrch2
DBDGEN
FINISH
END
```

副次索引用のサイド・セグメント

IMS は、文書内の特定エレメントの原形の XML 文書は検索できません。 ただし、IMS クラシック JDBC ドライバーを使用している場合は、特定の XML エレメント・データが入ったサイド・セグメントを作成できます。

制約事項: IMS Universal JDBC ドライバーは、サイド・セグメントをサポートしていません。

IMS で XML を原形のまま保管する場合は、XML 文書の特定の部分を標準の IMS セグメントに分解することができます。その場合、副次索引を使用してこのセグメントを検索できます。

以下の図に、副次索引用の基本セグメント、オーバーフロー・セグメント、およびサイド・セグメントを示します。

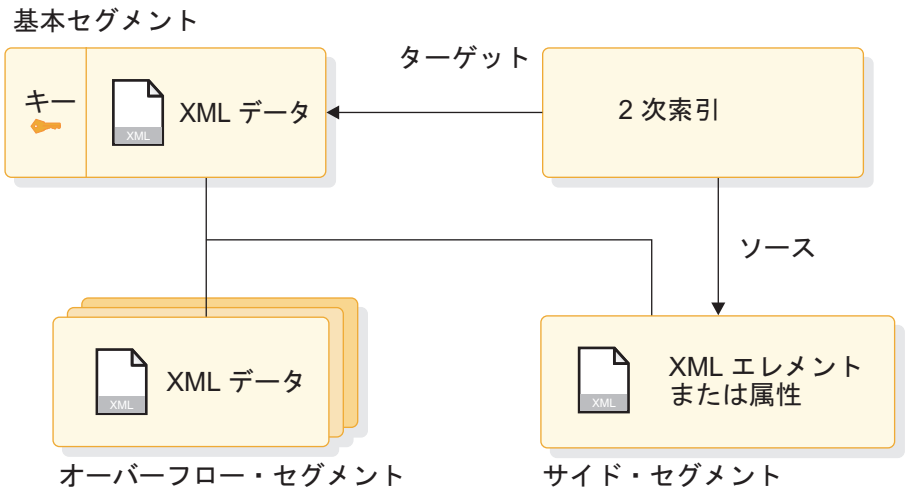


図 221. 副次索引による XML の原形保管

関連概念:

465 ページの『原形 XML 保管用の DBD』

XML スキーマの生成

IMS バージョン 12 は、IMS XML DB 機能をサポートする最後のリリースです。この機能により、Java アプリケーション・プログラムを使用して IMS データベースに XML 文書を保管したり、データベースから取得したりすることができます。IMS バージョン 12 がサービス休止になると、IMS バージョン 13 での IMS XML DB のサポートは終了します。IMS で XML を検索または保管するには、XML スキーマが必要です。生成済み XML スキーマは、PCB に基づいて IMS データベースを記述する XML 文書です。

IMS は XML スキーマを使用して、IMS に保管されるかあるいは IMS から検索される、XML 文書の妥当性を検査します。データベース内の XML の構造レイアウトを決めるのは、アプリケーション・プログラムではなく XML スキーマです。データをデータベース内に物理的に保管する方法は、`DLIDatabaseView` サブクラスが決定します。

実行時に、生成された XML スキーマはデータベースから検索されるデータの XML 構造を提供し、IMS に保管される着信 XML 文書の XML 構造を提供します。

XML スキーマを生成するには、次のようにします。

1. IMS Enterprise Suite DLIModel ユーティリティー・プラグインを使用して、PCB に基づいたスキーマを生成します。
2. 生成された XML スキーマが実行時に確実に使用できるようにします。 デフォルトでは、スキーマは PSB 名と PCB 名に基づいて HFS ルート・ディレクトリーからロードされます。デフォルトの場所 (ルート・ファイル・システム) は、環境変数 `http://www.ibm.com/ims/schema-resolver/file/path` を XML スキーマの場所の値で定義することにより、オーバーライドすることができます。例えば、XML スキーマがディレクトリー `/u/schemas` にある場合は、以下のように環境変数を SDK に定義します。

```
-Dhttp://www.ibm.com/ims/schema-resolver/file/path=/u/schema/
```

システム・プロパティーを設定することによって、XML スキーマをアプリケーション・プログラムで指定することもできます。以下に例を示します。

```
System.setProperty("http://www.ibm.com/ims/schema-resolver/file/path", "/u/schema");
```

XML から JDBC データ・タイプへのマッピング

IMS には、固有のタイプ情報はなく、そのセグメントをすべて単純なバイトの配列で保管します。したがって、IMS セグメントにアクセスするすべてのアプリケーション・プログラムは、そのセグメントに保管されているデータに同じデータ・タイプ・マッピングを使用する必要があります。

具体的には、アプリケーション・プログラムが以下の 3 つの情報について合意している必要があります。

- 各セグメント内で表されるフィールドのリスト
- 各フィールドが保管するデータ・タイプ
- フィールドの再定義を含む、各データ・タイプをバイトとして表す方法

IMS でデータベースから XML 文書を正しく生成し、XML 文書を正しく分解してデータベースに保管するには、以下の条件を満たすことも必要です。

フィールドのタイプに加えて、各 XML スキーマ文書は、すべてのフィールドを使用可能な 42 の XML タイプのいずれかとしてリストします。この情報は、有効な XML 文書のすべてのユーザーに文書内の情報の解釈方法を指示し、IMS に発信 XML 文書の生成方法、または着信 XML 文書の分解方法を通知します。XML 文書は、生成された XML スキーマに従って妥当性が検査され、IMS 用の Java メタデータを使用して、フィールドおよびセグメントに設定するデータのエレメント値と属性値の抽出方法が決定されます。

次の表は、IMS JDBC コネクターでサポートされている XML スキーマ・データ・タイプを説明したものです。

表 *67. IMS JDBC* コネクターでサポートされている *XML* スキーマ・データ・タイプ

JDBC データ・タイプ	**XML** スキーマ・データ・タイプ
BIGINT	xsd:long
BINARY	xsd:hexBinary
BIT	xsd:boolean
CHAR	xsd:string
日付	xsd:gYear (yyyy-MM の場合) xsd:date (yyyy の場合) xsd:gYearMonth (yyyy-MM-dd の場合)
DOUBLE	xsd:double
FLOAT	xsd:float
INTEGER	xsd:int
PACKEDDECIMAL	xsd:decimal
SMALLINT	xsd:short
TIME	xsd:time

表 67. *IMS JDBC* コネクターでサポートされている *XML* スキーマ・データ・タイプ (続き)

JDBC データ・タイプ	**XML** スキーマ・データ・タイプ
TIMESTAMP	xsd:dateTime
TINYINT	xsd:byte
VARCHAR	xsd:string
ZONEDECIMAL	xsd:decimal

XML の保管および検索のための JDBC インターフェース

Java アプリケーション・プログラムでは、タイプ 4 接続機能を備えた IMS Universal JDBC ドライバーを使用して、XML を IMS に保管し、IMS からXML を検索することができます。

Java アプリケーション・プログラムは、以下のどの環境でも実行できます。

- IMS 従属領域 (JMP または JBP)
- WebSphere Application Server for z/OS
- 非 z/OS プラットフォーム上の WebSphere Application Server
- Db2 for z/OS ストアード・プロシージャー
- CICS JCICS 領域

IMS クラシック JDBC ドライバーも、XML の保管と検索をサポートしています。

IMS が提供する JDBC ドライバーについて詳しくは、「*IMS V13* アプリケーション・プログラミング *API*」を参照してください。

第 4 部 データベースの設計と実装

このセクションでは、IMS データベースの設計と実装について説明します。説明する内容は、データ要件の分析、および各データベース・タイプの計画、設計、実装などです。

第 20 章 データ要件の分析

データベース設計の初期の段階の 1 つは、エンド・ユーザーの処理要件を満たすような概念的なデータ構造を作成することです。

したがって、概念的なデータ構造を作成する前に、エンド・ユーザーの処理要件とデータ要件に精通する必要があります。

データ構造の作成とは、実行する必要がある各作業のデータ要件をまとめて、それぞれの要件を満たすような 1 つ以上のデータ構造にする過程です。この章では、各業務処理ごとに作成されているローカル・ビューを用いて、 1 つのデータ構造を作成する方法を述べます。

アプリケーションにおいて、業務処理とは、エンド・ユーザーが必要とする作業の 1 つです。例えば、教育アプリケーションにおいて、クラスの名簿の印刷は業務処理の 1 つです。

ローカル・ビューとは、ある 1 つの業務処理のための概念的な 1 つのデータ構造と、個々のデータ相互間の関係を記述したものです。

このトピックで説明する方法を理解するには、「*IMS V13* アプリケーション・プログラミング」のアプリケーション設計に関する概要で説明している用語と例について熟知する必要があります。この概要では、アプリケーションにおける業務処理のためのローカル・ビューの作成方法を説明しています。

関連概念:

35 ページの『第 3 回の設計レビュー』

36 ページの『第 4 回の設計レビュー』

業務処理のローカル・ビュー

ある 1 つのアプリケーションに属するさまざまな業務処理のデータ要件を満たす構造を設計するには、これらの業務処理のそれぞれの要件を理解する必要があります。

業務処理のローカル・ビューとはこの要件について記述したものであるといえます。それは、ローカル・ビューは次の情報を提供するからです。

- 業務処理が必要とするすべてのデータ・エレメントの一覧表、および、これらのデータ・エレメントを制御するキー。
- 各処理ごとに開発されている概念的なデータ構造。これは、どのようにしてデータ・エレメントがデータ集合にグループ化されているかを示しています。
- 各処理のデータ集合間のマッピング。

このトピックでは、カスタマーに技術教育を提供しているある会社を例として使用します。この教育会社には、HQ と呼ばれる本社が 1 つあり、何個所かに Ed センターと呼ぶ地方教育センターがあります。 HQ は、各 Ed センターで提供され

る講習を開発します。各 Ed センターは、それぞれが提供する講習のスケジューリングと、これらの講習の受講者登録を担当しています。

講習とは、ある 1 個所の Ed センターで、ある特定の日付に、ある 1 つの科目について講習を 1 回開催することです。異なる何個所かの Ed センターで、同じ科目の講習が行われるかもしれませんが、これらはそれぞれ別々の講習です。

このトピックで使用するローカル・ビューは、ある 1 つの教育アプリケーションにおける以下の業務処理のためのものです。

現行名簿

クラス・スケジュール

インストラクター・スキル報告書

インストラクター・スケジュール

ローカル・ビューについての注意:

- 各ローカル・ビューごとのデータ構造の中にあるアスタリスク (*) は、データ集合を識別するデータ・エレメントを示しています。これは、データ集合のキーです。データ集合を固有に識別するために、複数のデータ・エレメントを必要とする場合もあります。
- 各処理におけるさまざまなデータ集合の間のマッピングは、マッピング表記で示されています。 1 対多のマッピングとは、各 *A* 集合ごとに 1 つ以上の *B* 集合があることを意味しています。 ◄——►►

  多対多の関係とは、各 *A* 集合ごとに多数の *B* 集合があり、しかも各 *B* 集合ごとに多数の *A* 集合があることを意味しています。 ◄◄——►►

ローカル・ビュー 1. 現行名簿

このトピックでは、現行名簿業務処理のデータ要件を満たすために使用されるエレメント、データ構造、データ集合、およびデータ集合間の関係のマッピングについて説明します。

現行名簿データ・エレメントの一覧表

以下に示すのは、技術教育の提供会社の例に関するデータ・エレメントとその説明の一覧表です。

データ・エレメント
　　説明

CRSNAME
　　科目名

CRSCODE
　　講習のコード

LENGTH
　　講習期間

EDCNTR
　　講習を行う Ed センター

日付　講習日

CUST 受講者を送り込む客先

LOCTN
客先の住所

STUSEQ#
生徒番号

STUNAME
生徒の名前

STATUS
受講者の登録状況

ABSENCE
受講者の欠席

GRADE
受講者の講習成績

INSTRS
講習の講師

以下の図は、現行名簿の概念的なデータ構造を示したものです。

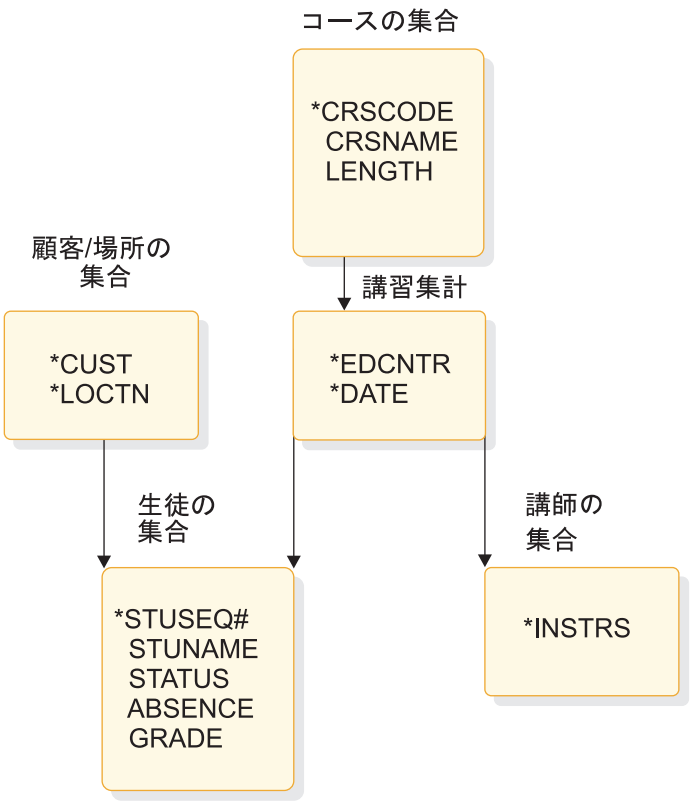


図 222. 現行名簿の概念的なデータ構造

現行名簿のマッピング

現行名簿のマッピングは次のとおりです。

Curse ◄——►► Class

Class ◄——►► Student

Class ◄——►► Instructor

Customer/location◄——►► Student

ローカル・ビュー 2. クラス・スケジュール

このトピックでは、クラス・スケジュール業務処理のデータ要件を満たすために使用されるエレメント、データ構造、データ集合、およびデータ集合間の関係のマッピングについて説明します。

クラス・スケジュール・データ・エレメントの一覧表

以下に示すのは、例に関するクラス・スケジュール・データ要素とその説明の一覧表です。

データ・エレメント
　　説明

CRSCODE
　　講習のコード

CRSNAME
　　科目名

LENGTH
　　講習日数

PRICE
　　科目受講料

EDCNTR
　　講習を行う Ed センター

日付　　ある特定の Ed センターで講習を行う日付

以下の図は、クラス・スケジュールの概念的なデータ構造を示したものです。

図 223. クラス・スケジュールの概念的なデータ構造

クラス・スケジュールのマッピング

このローカル・ビューのマッピングは次に示すもののみです。

Curse ◄——►► Class

ローカル・ビュー 3. インストラクター・スキル報告

このトピックでは、インストラクター・スキル報告業務処理のデータ要件を満たすために使用されるエレメント、データ構造、データ集合、およびデータ集合間の関係のマッピングについて説明します。

インストラクター・スキル報告データ・エレメントの一覧表

以下に示すのは、技術教育の提供会社の例に関するインストラクター・スキル報告データ・エレメントとその説明の一覧表です。

データ・エレメント
: 説明

INSTR
: 講師

CRSCODE
: 講習のコード

CRSNAME
: 科目名

以下の図は、インストラクター・スキル報告の概念的なデータ構造を示したものです。

インストラクターの集合

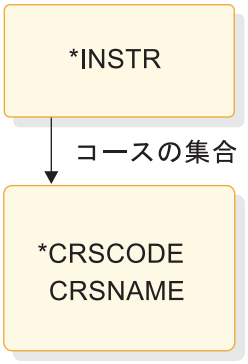


図 224. インストラクター・スキル報告の概念的なデータ構造

インストラクター・スキル報告のマッピング

このローカル・ビューのマッピングは次に示すもののみです。

Instructor ◄——►► Course

ローカル・ビュー 4. インストラクター・スケジュール

このトピックでは、インストラクター・スケジュール業務処理のデータ要件を満たすために使用されるエレメント、データ構造、データ集合、およびデータ集合間の関係のマッピングについて説明します。

インストラクター・スケジュール・データ・エレメントの一覧表

以下に示すのは、例に関するインストラクター・スケジュール・データ・エレメントとその説明の一覧表です。

データ・エレメント
: 説明

INSTR
: 講師

CRSNAME
: 科目名

CRSCODE
: 講習のコード

EDCNTR
: Ed センター

日付
: 講習を行う日付

以下の図は、インストラクター・スケジュールの概念的なデータ構造を示したものです。

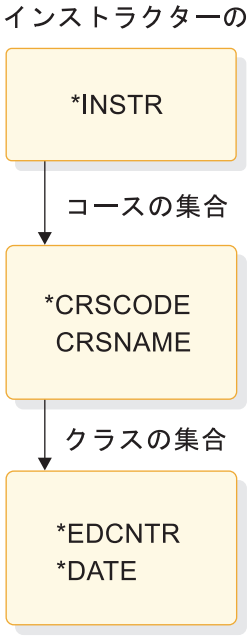


図 225. インストラクター・スケジュールの概念的なデータ構造

インストラクター・スケジュールのマッピング

このローカル・ビューのマッピングは次のとおりです。

Instructor ◄——►► Course

Curse ◄——►► Class

関連概念:

アプリケーションの設計: データ・ビューとローカル・ビュー (アプリケーション・プログラミング)

概念的なデータ構造の設計

すべてのローカル・ビューのマッピングを分析してみることは、概念的なデータ構造を設計する最初のステップの 1 つです。

セグメントに影響を与えるマッピングは、2 種類あります。それは、1 対多と多対多です。

1 対多のマッピングとは、セグメント *A* ごとに 1 つ以上のセグメント *B* があることを意味しています。これは、A ◄——►► B のように示されます。例えば、475 ページの『業務処理のローカル・ビュー』に示す現行名簿では、科目と講習の間に 1 対多の関係があります。各科目ごとに何回かの講習をスケジュール可能ですが、1 回の講習は 1 科目とつながりを持つだけです。 1 対多のマッピングは、1 つの従属関係として表すことができます。科目対講習の例では、講習はある特定の科目に従属しています。

多対多のマッピングとは、各セグメント *A* ごとに多数のセグメント *B* があり、しかも各セグメント *B* ごとに多数のセグメント *A* があることを意味しています。これは、A ◄◄——►► B のように示されます。多対多の関係は従属関係ではあり

ません。通常、このような関係は 2 つの別個のデータ構造に属するデータ集合の間において起こるからで、これは 2 つの業務処理でこのデータを処理する必要がある方法に矛盾が起こることを示しています。

あるデータ構造を DL/I によってインプリメントする場合には、データの矛盾を解消するために適用できる 3 つの方法があります。

論理関係を定義する

副次索引を設定する

データを両方の場所に保管する (言い換えれば、重複データを持つ)

概念的なデータ構造を設計する最初のステップは、すべてのローカル・ビューのマッピングをひとまとめにすることです。これを行うには、各のローカル・ビューごとにマッピングをくまなく見ていき、マッピングの総合一覧表を作成します (以下の表を参照)。マッピングを調べるときには、次の注意事項を守ってください。

- 重複するマッピングを記録してはなりません。この段階では、各種のマッピングをすべてカバーする必要がありますが、各オカレンスは必要ありません。
- 2 つのデータ集合が別々のローカル・ビューにおいて逆のマッピングをもつ場合には、より複雑なマッピングを使用します。こうすれば、これらのマッピングをひとまとめにしたとき、これは両方のマッピングが入っていることになります。例えば、ローカル・ビュー #1 にマッピング A ◄——►► B があり、ローカル・ビュー #2 にマッピング A ◄◄——► B があるものとすれば、この両方のマッピングが入っているマッピングを使用します。このケースでは、これは ◄◄——►► B です。

表 *68.* ローカル・ビューに対する結合マッピング

マッピング	ローカル・ビュー
Curse ◄——►► Class	1、2、4
Class ◄——►► Student	1
Class ◄——►► Instructor	1
Customer/location ◄——►► 生徒	1
Instructor ◄——►► 講習	3、4

結合マッピングを用いて、以下の図に示されるデータ構造を構成することができます。

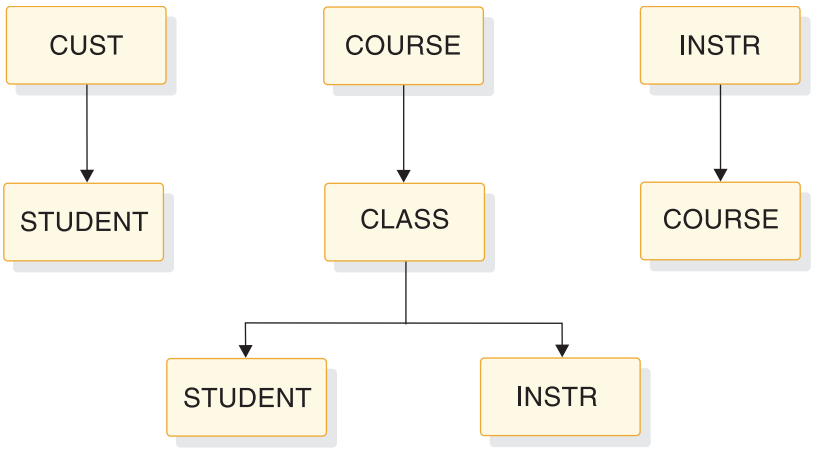


図 226. 教育用データ構造

このデータ構造には、2 つの矛盾があります。まず、STUDENT は、CUST にも CLASS にも従属しています。次に、COURSE と INSTR の間、および INSTR と COURSE の間には、逆のマッピングがあります。 DL/I によってこれらの構造をインプリメントする場合には、論理関係を用いてこれらの矛盾を解消することもできます。

関連タスク:

485 ページの『論理関係に関する要件の分析』

484 ページの『データの矛盾の解消』

DL/I によるデータ構造のインプリメント

データ構造を DL/I を用いてインプリメントする場合には、これを 1 つの階層としてインプリメントします。階層はいくつかのセグメントによって構成されています。

階層において、1 対多の関係は親子関係と呼ばれます。階層において各セグメントは 1 つ以上の子を持つことができますが、それぞれのセグメントは 1 つの親しか持つことができません。

DL/I を使用する場合には、作成した構造の中のそれぞれのデータ・エレメントをどのようにしてセグメントにグループ化したらよいか考察する必要があります。また、構造の中に存在するデータの矛盾を DL/I によってどのように解消することができるか考察する必要があります。

セグメントへのデータ・エレメントの割り当て

階層の中でデータ・エレメントをどのように関連付けるかを決定したら、各データ・エレメントを 1 つのセグメントと関連付けます。

それを行うために、すべてのキーとそれらの関連データ・エレメントの一覧表を作成します。 1 つのキーとその関連データ・エレメントが複数のローカル・ビューにある場合は、この関連を 1 回のみ記録します。

以下の表に示すように、データ・エレメントをそのキーの隣にリストします。キーとその関連データ・エレメントがセグメントの内容になります。

表 *69.* キーと関連データ・エレメント

データ集合	キー	データ・エレメント
COURSE	CRSCODE	CRSNAME, LENGTH, PRICE
CUSTOMER/LOCATION	CUST、LOCTN	
CLASS	EDCNTR, DATE	
STUDENT	STUSEQ#	STUNAME, ABSENCE, STATUS, GRADE
INSTRUCTOR	INSTR	

1 つのデータ・エレメントがいくつかの異なるローカル・ビューにおいていくつかの異なるキーと結び付いている場合には、どのセグメントにこのデータ・エレメントを保管したらよいかを決定する必要があります。それを行わなければ、重複データを保管するしかありません。このようなことになるのを避けるために、階層の中で一番上にあるキーと一緒にこのデータ・エレメントを保管します。例えば、キー ALPHA と BETA が共にデータ・エレメント XYZ と結び付いている場合 (一方はローカル・ビュー 1 において、他方はローカル・ビュー 2 において)、この階層の中で ALPHA の方が上にあるものとすれば、XYZ を ALPHA と一緒に保管して、これを 2 回保管するのを回避します。

データの矛盾の解消

設計されるデータ構造は、アプリケーションのすべての処理要件を必ずしも満たしているとはかぎりません。

例えば、ある 1 つの業務処理において、特定のセグメントを取り出すのに、キー・フィールドとして選定しているフィールド以外のフィールドを用いていなければならないことがあります。別の業務処理では、異なる複数のデータ構造に属するセグメントを組み合わせる必要がある場合もあります。データ構造においてこのような種類の矛盾を突き止めた場合、DL/I を使用するものとすれば、その矛盾を解消するための 2 つの DL/I オプション、すなわち副次索引と論理関係にあたってみることができます。

関連タスク:

481 ページの『概念的なデータ構造の設計』

副次索引に関する要件の分析

副次索引を使用すると、特定のセグメントを、そのセグメントのキー・フィールド以外のフィールドで識別できるようになります。

例えば、技術教育を提供する会社において、ある特定の受講者がある講習に登録されているかどうかを (ある端末から) 突き止める必要があるものとします。この受講者がこの講習に登録されているかどうかわからないとすれば、おそらく受講者番号もわからないはずです。しかし、STUDENT セグメントのキーは STUSEQ# です。例えば、STUDENT セグメントを対象とする要求を出し、受講者の名前 (STUNAME) を用いて必要なセグメントを識別したとします。 IMS は、受講者番号 (STUSEQ#) の代わりに、すべての STUDENT セグメントの中からこのセグメン

トを検索します。 STUDENT セグメントが受講者番号順に保管されている場合は、STUNAME を与えるだけでは IMS が STUDENT セグメントの位置を知ることはできません。

この例において副次索引を使用すると、この業務処理に関しては STUDENT セグメントのキー・フィールドを STUNAME にしたようにすることができます。他の業務処理では、依然として STUSEQ# をキーとしてこのセグメントを処理します。

これを行うためには、STUDENT セグメントを STUNAME について副次索引に登録します。セグメント内の任意のフィールドを副次索引に登録することができます。フィールドを索引に登録して、そのセグメントに副次索引を使用することを IMS に示すと、IMS は索引に登録されたフィールドがキーであるものとして、そのセグメントを処理します。

論理関係に関する要件の分析

業務処理において異なるいくつかの階層に属するセグメントを組み合わせる必要がある場合には、論理関係を使用すればこれを行うことができます。

論理関係を定義すれば、ストレージの中には実在しない階層構造を作成し、ストレージの中にあたかも実在するかのようにそれを処理することができます。別個の階層に属するセグメントとセグメントを関係付けることができます。このような論理関係から作成されたデータ構造は、論理構造と呼ばれます。別個の階層のセグメントを関係付けるには、より頻繁にアクセスされるパスにこのセグメントを保管しておきます。このセグメントが、それほど頻繁にはアクセスされないパスに、このセグメントを指すポインターを保管します。

481 ページの『概念的なデータ構造の設計』の図に示す階層では、STUDENT セグメントに 2 つの親が存在する可能性があります。 CUST セグメントは既存のデータベースの一部分であるとすると、CUST セグメントと STUDENT セグメントの間に論理関係を定義することができます。そうすると、以下の図に示すような階層を持つことができます。CUST/STUDENT 階層は、論理構造です。

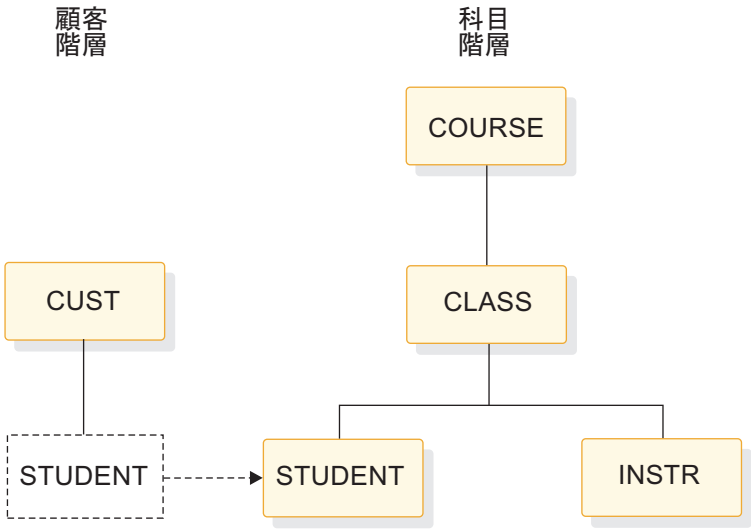


図 227. 教育用階層

この種の論理関係は、関係が「片道」なので、単一方向論理関係と呼ばれます。

481 ページの『概念的なデータ構造の設計』の図には、COURSE と INSTR の間にもう 1 点の矛盾が見られます。ある 1 科目の講習が何回か開かれ、1 回の講習には数人の講師が関係しますが (COURSE ◄——►► CLASS ◄——►► INSTR)、各講師は数科目を教えることができます (INSTR ◄——►► COURSE)。両方向論理関係を用いると、この矛盾を解消することができます。 INSTR セグメントを別の階層に保管し、このセグメントを指すポインターを講習階層の INSTR セグメントの中に保管することができます。さらに、COURSE セグメントを講習階層の中に保管し、このセグメントを指すポインターを講師階層の COURSE セグメントの中に保管することができます。この両方向論理関係によって、以下の図に示すような 2 つの階層ができあがるので、重複データを持つ必要がなくなります。

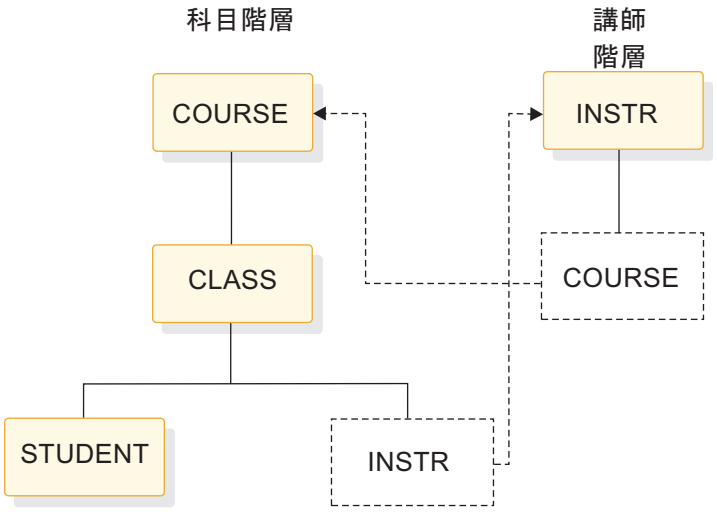


図 *228.* 両方向論理関係

関連タスク:

481 ページの『概念的なデータ構造の設計』

第 21 章 全機能データベースの設計

アプリケーションの処理要件に最も適したデータベース・タイプとオプションの機能を決定した後で、データベース設計とオプションの使用に関して一連の決定をする必要があります。

これらの決定によって、主として、データベースのパフォーマンスと使用可能スペースの使用効率が決まります。以下のことに基づいて一連の決定を行います。

- すでに選択されているデータベース・タイプとオプションの機能
- アプリケーションのパフォーマンス要件
- オンラインで使用可能なストレージの量

関連概念:

35 ページの『第 3 回の設計レビュー』

関連タスク:

774 ページの『HDAM または PHDAM オプションの調整』

フリー・スペースの指定 (HDAM、PHDAM、HIDAM、および PHIDAM のみ)

HD データベースがロードされた後に挿入される従属セグメントは、このセグメントが関連しているセグメントにできるだけ近いところに置かれます。

しかし、データベースが次第に大きくなり、使用可能なスペースが減ってくると、従属セグメントは関連するセグメントからますます遠いところに置かれるようになります。こうなると、パフォーマンスが低下しますので、この問題を解消するためには、データベースの再編成を行うしかありません。 (セグメントが、このセグメントを指すセグメントに近いところにあれば、従属セグメントを検索するのに必要な入出力時間が短くて済みます。シーク・タイムと回転待ち時間が短くて済むからです。)

データベースをロードした後の挿入操作の影響を最小化するために、データベースを最初にロードするときフリー・スペースをこのデータベースの中に割り振ってください。データベース内にフリー・スペースを割り振っておけば、挿入操作によるパフォーマンス面への悪影響が軽減され、したがって、HD データベースを再編成する頻度が少なくなります。

OSAM データ・セットと VSAM ESDS に対しては、DBD の中の DATASET ステートメントの FRSPC= キーワードでフリー・スペースを指定します。 このキーワードでは、以下のオペランドの一方または両方を指定できます。

- 空きブロックの頻度係数 (fbff)。 fbff は、データベースがロードされるときに、データ・セット・グループの中の n 番目のブロックまたは CI ごとに、このブロックまたは CI をフリー・スペースとして残しておくように指定するものです (ここで、fbff=n)。 fbff の範囲は、0 から 100 までの間の 1 を除くすべての整数値です。HDAM または PHDAM データベースには fbff を指定しないでくだ

さい。HDAM または PHDAM データベースに fbff を指定した場合、ロード時に相対ブロック、ブロックの CI 番号、またフリー・スペースとマークされている CI 番号をランダム化モジュールが生成すると、ランダマイザーはこのルート・セグメントを別のブロックに保管しなければなりません。

fbff を指定すると、n 番目のブロックまたは CI が HD スペース検索アルゴリズムによって、2 番目に望ましいブロックまたは CI と見なされます。これが当てはまるのは、DBDGEN ユーティリティーの DATASET ステートメントで SEARCHA=1 を指定していない場合です。 SEARCHA=1 を指定すると、2 番目に望ましい CI またはブロックの中でフリー・スペースを検索しないように IMS に指示することになります。

- フリー・スペース・パーセント係数 (fspf)。 fspf は、データベースがロードされるときに、データ・セット・グループの中の各ブロックまたは CI において、最低何 % をフリー・スペースとして残しておくべきかを指定するものです。 fspf の範囲は 0 から 99 です。

注**:** このフリー・スペースは、VSAM ESDS および OSAM データ・セットに適用されます。これは HIDAM 索引データベースまたは PHIDAM 索引データベース、あるいは DEDB には適用されません。

VSAM KSDS の場合は、`DEFINE CLUSTER` コマンドの FREESPACE パラメーターでフリー・スペースを指定します。この VSAM パラメーターは、HIDAM、PHIDAM、HDAM、または PHDAM で使用される VSAM ESDS データ・セットでは無視されます。このコマンドについては、*z/OS DFSMS* カタログのためのアクセス方式サービス・プログラム に詳細な説明があります。

関連概念:

925 ページの『第 31 章 データベース・タイプの変換』

188 ページの『HD スペース検索アルゴリズムの動作』

関連タスク:

774 ページの『編成の良いデータベースの確実化』

611 ページの『ステップ 4. フリー・スペースに必要なブロックまたは CI の数の決定』

関連資料:

DATASET ステートメント (システム・ユーティリティー)

ルート・アドレス可能域のサイズの見積もり (HDAM または PHDAM のみ)

ルート・アドレス可能域のサイズを見積もるには、単純な数式を使用します。

次の数式を使用して、ルート・アドレス可能域のサイズを見積もることができます。

```
(A x B) / C = D
```

ここで、

A = ルート・アドレス可能域に保管される 1 つのデータベース・レコードのバイト数

B = データベース・レコードの予測数

C = 各 CI またはブロックにおいてデータ収容のために使用可能なバイト数 (言い換えると、CI またはブロックのサイズからオーバーヘッドを引いた差)

D = ルート・アドレス可能域のブロック単位または CI 単位の所要サイズ

データベースに対してフリー・スペースを指定した場合には、ルート・アドレス可能域のサイズを決定するための計算にそのスペースを含める必要があります。次の公式を使用して、このステップを完了します。

```
(D x E x G) / F = H
```

ここで、

D = 最初の数式で計算したサイズ (ブロック単位または CI 単位で表されたルート・アドレス可能域の所要サイズ)

E = データベースにおいてフリー・スペースとして空けておくべきブロックまたは CI の頻度 (DBD の中の fbff オペランドで指定した値)

F = E-1 (fbff-1)

G = *100* または 100 - fspf。 fspf は、各ブロックまたは CI においてフリー・スペースとして残しておくべき最小のパーセント (DBD の中の fspf オペランドで指定した値)

H = ブロック単位または CI 単位の所要サイズ

ルート・アドレス可能域で必要なブロックまたは CI の数を、DBD の中の DBD ステートメントの RMNAME=rbn キーワードで指定します。

使用するランダム化モジュールの決定 (HDAM および PHDAM のみ)

HDAM または PHDAM データベース・レコードを保管したり、アクセスしたりするには、ランダム化モジュールが必要です。

ランダム化モジュールは、ルート・セグメントのキーを相対ブロック番号と RAP 番号に変換します。次いで、この番号を用いて HDAM または PHDAM ルート・セグメントを保管したり、アクセスしたりします。1 つの HDAM データベースまたは PHDAM 区画はランダム化モジュールを 1 つしか使用しませんが、いくつかのデータベースおよび区画が同じモジュールを共用することができます。 4 つのランダム化モジュールが IMS で提供されます。

通常は、システムで提供される 4 つのランダム化モジュールのうちの 1 つが、ユーザーのデータベースの役に立つでしょう。

PHDAM データベースでは、ランダム化モジュールを呼び出す前に区画の選択が完了しています。ランダム化モジュールは、1 つの区画内でのみ位置を選択します。

ユーザー独自のランダム化モジュールの作成

ユーザーでのルート・キーの分布が分かっていて、これらのランダム化モジュールがいずれも望ましい働きをしない場合には、ユーザーが独自のランダム化モジュールを作成してください。独自のランダム化モジュールを作成する場合、その目標の 1 つは、このモジュールにルート・セグメントを適当に分布させ、その結果、後にルート・セグメントにアクセスする際に 1 回の読み取り操作と 1 回のシーク操作だけで済むようにすることです。あるルート・キーがランダム化モジュールに与えられたとき、ランダマイザーが生成する相対ブロック番号が、このルート・セグメントを実際にもっているブロックの番号であれば、1 回の読み取り操作と 1 回のシーク操作だけですみます (アクセスは高速です)。ユーザーが作成するランダム化モジュールは、システムのチューニングにブロックと RAP を使用できるようにするために自分が指定するブロックと RAP の数を変更できるような形にしておくべきです。また、このランダム化モジュールは、ルート・セグメントをランダムに分布させるべきであり、ビットマップの位置にランダム化すべきでなく、しかもパッキング密度を高く保つようにすべきです。

ランダム化モジュールの有効性の評価

ユーザーのデータベースに対する特定のランダム化モジュールの有効性を決める 1 つの方法は、IMS ハイパフォーマンス・ポインター・チェッカー (HD チューニング・エイド) を実行することです。このツールは、データベースでルート・セグメントがどのように保管されているかを示すマップの形の報告書を作成します。ルート・セグメントの保管は、ユーザーがルート・アドレス可能域に対して指定したブロックまたは CI の数、および各ブロックまたは CI に対して指定した RAP の数に基づいていることを示します。さまざまなランダム化モジュールに対して HD チューニング・エイドを実行することにより、どのモジュールがユーザーのデータベースにおいてルート・キーの最良の分布をもたらすかを判断することができます。さらに、指定する RAP の数とブロックまたは CI の数を変更してみれば、(特定のランダム化モジュールのもとで) RAP の数とブロックまたは CI の数のどの組み合わせが最良のルート・セグメントの分布をもたらすか判断することができます。

関連概念:

925 ページの『第 31 章 データベース・タイプの変換』

『HDAM または PHDAM オプションの選択』

関連タスク:

774 ページの『編成の良いデータベースの確実化』

774 ページの『HDAM または PHDAM オプションの調整』

関連資料:

HDAM および PHDAM ランダム化ルーチン (DFSHDC40) (出口ルーチン)

HDAM または PHDAM オプションの選択

HDAM データベースまたは PHDAM データベースでは、ユーザーが選択したオプションがパフォーマンスに大きな影響を及ぼすことがあります。

ここで言うオプションとは、DBD ステートメントの RMNAME キーワードで指定するか、または HALDB 区画定義ユーティリティーの使用時に指定するオプション

です。以下の図は、RMNAME パラメーターを指定するときのフォーマットを示しています。この図の次にある定義リストに、*mod*、*anch*、*rbn*、および *bytes* の意味を説明します。

```
RMNAME=(mod,anch,rbn,bytes)
```

mod 選定したランダム化モジュールの名前。

anch ブロックまたは CI の中の RAP の数。

rbn ルート・アドレス可能域の中のブロックまたは CI の数。

bytes データベース・レコード内の各セグメントが (処理操作の介在なしに) 連続して挿入されるときに、ルート・アドレス可能域に保管されるデータベース・レコードの最大バイト数。

入出力操作の最小化

これらの HDAM または PHDAM オプションを選定する際の主な目標は、ある 1 つのデータベース・レコードまたはセグメントにアクセスするのに必要な入出力操作の数を最小限に抑えることです。入出力操作の数が少なければ少ないほど、アクセス時間が短くなります。以下の場合にパフォーマンスが最高になります。

- 1 つのブロックまたは CI の中の RAP の数が、このブロックまたは CI の中のルート・セグメントの数に等しい (使用されない RAP によって、ブロックまたは CI のスペースが浪費されることがありません)。
- ほとんどのルート・セグメントに対して固有のブロック番号と RAP 番号が生成される (したがって、長いシノニム・チェーンが避けられます)。
- ルート・セグメントがキー・シーケンスで保管されている。
- 頻繁に使用される従属セグメントがすべてルート・アドレス可能域の中にあり (ルート・アドレス可能域へのアクセスの方がオーバーフロー域へのアクセスより速くなります)、しかもルート・セグメントと同じブロックまたは CI の中にある。

ランダム化モジュールの選定により、ルート・セグメントごとの固有のアドレスの数、およびルート・セグメントがキー・シーケンスで保管されるかどうかが決まります。一般に、ルート・セグメントがルート・アドレス可能域において一様に分布しているならば、このランダム化モジュールは効率がよいものと見なされます。さまざまなランダム化モジュールを用いて実験してみることができます。このような実験をする場合には、anch、rbn、および bytes オペランドのさまざまな組み合わせを試してみて、これがルート・セグメントの分布にどのような影響を及ぼすか調べてください。

パッキング密度の最大化

HDAM または PHDAM オプションを選定する際の第 2 の目標は、パフォーマンスに悪影響を及ぼすことなく、パッキング密度をできるだけ大きく保つことです。パッキング密度とは、ルート・アドレス可能域においてルート・セグメントとその従属セグメントのために使用されているスペースのパーセントのことです。パッキング密度は、次のようにして求められます。

```
パッキング密度 =
(ルート・セグメントの数  x  ルート・セグメントのバイト数) /
(ルート・アドレス可能域内の CI の数  x  CI で使用可能なスペース)
```

ルート・セグメントのバイト数
: ルート・アドレス可能域の中の各ルート・セグメントの平均バイト数

CI で使用可能なスペース
: CI またはブロックのサイズから FSEAP、RAP、VSAM CIDF、VSAM RDF のためのスペースとフリー・スペース (該当するもの) を引いた差

パッキング密度は大きくすべきですが、パッキング密度の割合が増えるにつれて、オーバーフロー・ストレージに収容される従属セグメントの数が増えることになります。しかも、従属セグメントがオーバーフロー・ストレージに保管されている場合には、これらの従属セグメントを処理するパフォーマンスが低下します。RMNAME= キーワードで指定できるオペランドはすべて、パッキング密度に影響します。したがって、最適なパッキング密度を得るためにさまざまなランダム化モジュール、および anch、rbn、bytes オペランドのさまざまな組み合わせを試してみてください。

関連概念:

925 ページの『第 31 章 データベース・タイプの変換』

489 ページの『使用するランダム化モジュールの決定 (HDAM および PHDAM のみ)』

関連タスク:

774 ページの『HDAM または PHDAM オプションの調整』

HISAM データベースの論理レコード長の選択

HISAM データベースでは、この選定がデータベースの中のスペースの使用とアクセス時間の両方に影響するので、論理レコード長を選定することが重要になります。

個々の状況に応じて個々の要因の相対的重要度は異なります。しかし、最良のパフォーマンスと、スペースの使用とアクセス時間との最適なバランスを得るためにいくつか試みに論理レコード長を決めて、最終的な選定を行う前にこれらをテストしてみてください。

論理レコード長の考慮事項

以下のことを考慮してください。

- 論理レコードの中には、完全なセグメントのみ保管することができます。したがって、論理レコードの中に収まる最終のセグメントと、その論理レコードの終わりまでのスペースは使用されません。
- 各データベース・レコードは、論理レコードの先頭から始まります。したがって、データベース・レコードの終わりからこのデータベース・レコードが入っている最後の論理レコードの終わりまでのスペースは使用されません。使用されないこのスペースは、データベース・レコードの平均サイズに関連します。
- 極めて短い論理レコードあるいは極めて長い論理レコードは、無駄なスペースを多くする傾向があります。論理レコードが短ければ、使用されないスペース域の数が増えます。論理レコードが長いと、使用されないスペースのサイズが増えます。以下の図に、論理レコードが短い場合、または長い場合に無駄なスペースが増える理由を示します。

論理レコードの末尾の使用されないスペースの量を最小限にするような論理レコード長を選定してください。

以下の図に示すデータベース・レコードは、図 230 では 3 つの短い論理レコードに保管され、図 231 では 2 つのより長い論理レコードに保管されます。

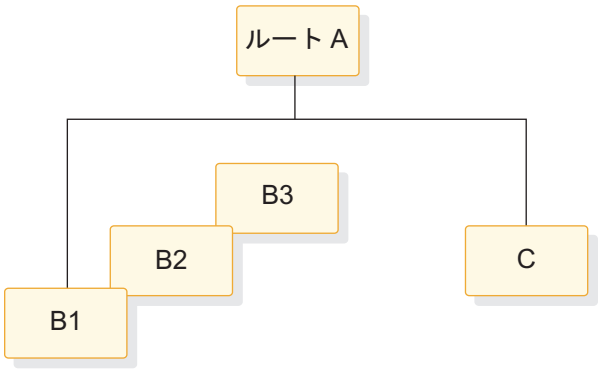


図 *229.* 論理レコードの例のためのデータベース・レコード

以下の図では、使用されない 3 つのスペース域に注意してください。図 231 では、使用されないスペース域は 3 つではなく 2 つだけですが、これらのスペース域の合計サイズは大きくなっています。

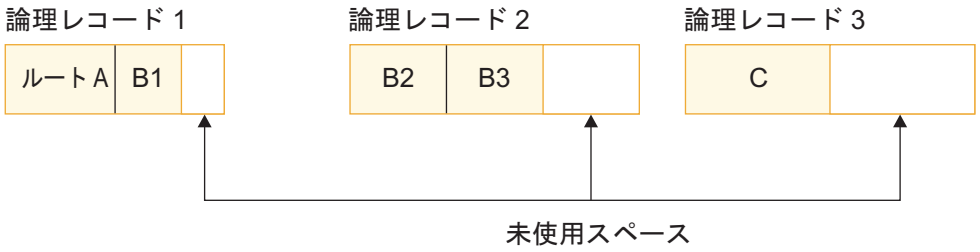


図 *230.* 短い論理レコード

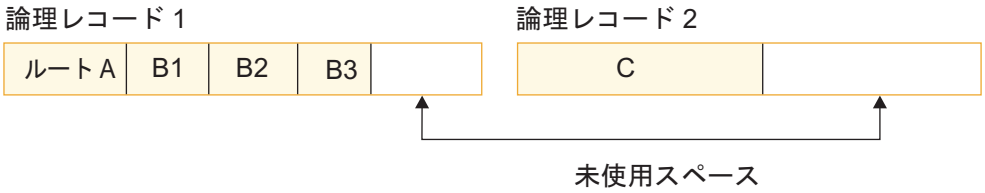


図 *231.* 長い論理レコード

ある 1 つのデータベース・レコードに属するセグメントのうち、1 次データ・セットの中の論理レコードに収まらないものは、オーバーフロー・データ・セットの中の 1 つ以上の論理レコードに入れられます。オーバーフロー・データ・セットの中の論理レコードにアクセスするには、1 次データ・セットの中の論理レコードにアクセスするよりも多くの読み取り操作とシーク操作が必要で、したがって、より長いアクセス時間が必要です。これは、データベースのサイズが次第に大きくなり、オーバーフロー・レコードのチェーンが作成されるにつれて、特に顕著になります。このような理由からデータベース・レコードの中の一番よく使用されるセグメ

ントは、1 次データ・セットに入れるように努めるべきです。 1 次データ・セットの中の論理レコード長の選定は、できる限り平均のデータベース・レコード長に近づけておくべきです。こうすれば、オーバーフロー論理レコードが最小限になり、したがって、パフォーマンスの問題が最も小さくなります。レコード長の平均値を求める際には、異常に長いレコードか異常に短いレコードによって、偏った結果が生じる場合があることを忘れないでください。

1 回の読み取り操作で 1 つの CI がバッファー・プールに読み込まれます。 CI には、1 つのデータベース・レコードの 1 つ以上の論理レコードが入っています。このため、ある 1 つのデータベース・レコード全体にアクセスするには、このデータベース・レコードを入れるのに必要な CI の数と同じ数だけの読み取り操作とシーク操作が必要です。 495 ページの図 233 では、CI ごとに 2 つの論理レコードがあり、以下の図に示すようにデータベース・レコードを保管するのに 2つの CI が必要です。したがって、これらの 4 つの論理レコードをバッファーの中に取り込むには、2 回の読み取り操作が必要になります。

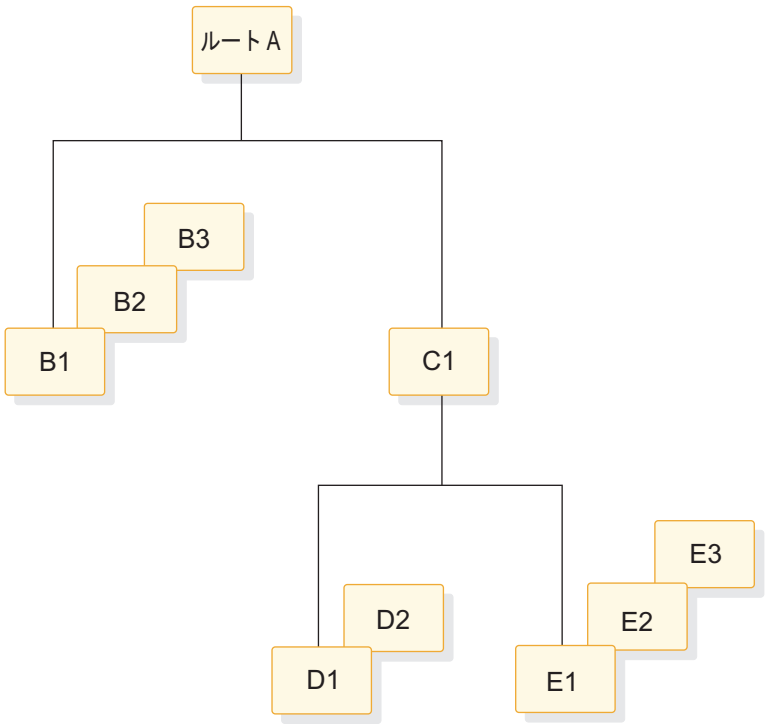


図 232. 論理レコードの例のためのデータベース・レコード

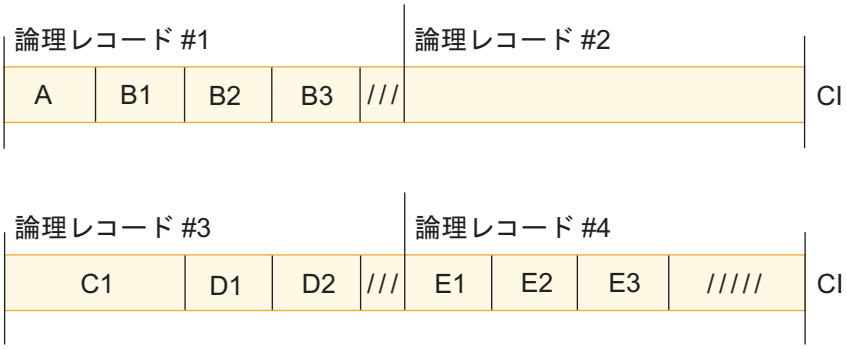


図 233. 2 つの読み取り操作が必要になる論理レコードの例

論理レコードのサイズが小さくなるにつれて、データベース・レコードにアクセスするのに必要な読み取り操作とシーク操作の数が増えます。データベース・レコード全体に頻繁にアクセスする必要があるかどうかを考えてみてください。その必要があれば、データベース・レコード全体を保管するような論理レコード・サイズを選定するように努めなければなりません。他方、通常はデータベース・レコードの中の 1 つまたは少数のセグメントだけにアクセスする場合には、平均的なデータベース・レコードを入れるのに十分な大きさの論理レコード・サイズを選定することは、それほど重要ではないことになります。

データベース・レコードを次々に読み取る必要のある次のようなセットアップの例では、どのようなことが起きるか考えてみましょう。

- CI またはブロックのサイズは 2048 バイトです。
- 論理レコードのサイズは 512 バイトです。
- データベース・レコードの平均サイズは 500 バイトです。
- データベース・レコードのサイズは 300 から 700 バイトの範囲です。

論理レコードのサイズとデータベース・レコードの平均サイズがほぼ同じ (512 と 500) なので、データベース・レコード 2 つごとにほぼ 1 つが、1 回の読み取り操作でバッファー・プールに読み込まれます。 (この仮定はデータベース・レコードの平均サイズに基づいたものです。) これに反して、論理レコードのサイズが 650 であると、ほとんどのデータベース・レコードは 1 回の読み取り操作でアクセスされるでしょう。これには明らかなトレードオフがあり、HISAM データ・セットの論理レコード長を選別する際には、常にこの点を考慮しなければなりません。論理レコードのサイズが 650 であったとすれば、平均的なデータベース・レコードの終わりから、これを保管している最後の論理レコードの終わりまでの間に、多量の未使用スペースが存在することになります。

守るべき規則

HISAM データ・セットの論理レコード長を選定するときには、以下の規則を守らなければなりません。

- 1 次データ・セットの中の論理レコード・サイズは、少なくともルート・セグメントのサイズにその接頭部およびオーバーヘッドを加えた和に等しくなければなりません。可変長セグメントが使用されている場合には、論理レコード・サイズ

は、少なくとも最長のルート・セグメントのサイズにその接頭部とオーバーヘッドを加えた和に等しくなければなりません。 VSAM に対しては 5 バイトのオーバーヘッドが必要です。

- オーバーフロー・データ・セットの中の論理レコード・サイズは、少なくともそのオーバーフロー・データ・セットの中の最長セグメントのサイズにその接頭部およびオーバーヘッドを加えた和に等しくなければなりません。 VSAM に対しては 5 バイトのオーバーヘッドが必要です。
- オーバーフロー・データ・セットの中の論理レコード長は、1 次データ・セットの中の論理レコード長に等しいかそれより大きくなければなりません。
- 最大の論理レコード・サイズは 30720 バイトです。
- SHISAM データベースを除いて、論理レコード長は偶数でなければなりません。

1 つのデータベース・レコードを保持するために必要な論理レコード数の計算

さまざまの論理レコード・サイズを仮定する前に、まずデータベース・レコードの平均的サイズを計算してみます。この平均サイズが求まれば、特定の論理レコード・サイズのもとで、 (平均的なサイズの) 1 つのデータベース・レコードを保持するのに論理レコードがいくつ必要であるか分かります。

論理レコード長の指定

論理レコードの長さは、DBD 生成ユーティリティーの DATASET ステートメントの RECORD= パラメーターで指定します。

関連タスク:

603 ページの『データベースの最小サイズの見積もり』

関連資料:

DATASET ステートメント (システム・ユーティリティー)

HD データベースの論理レコード長の選択

HD データベースでは、重要な選定は論理レコード長ではなく、CI またはブロックのサイズです。

VSAM が使用されている場合には、論理レコード長はブロック・サイズと同じです。論理レコード・サイズは、CI サイズから 7 バイトのオーバーヘッド (CIDF に対して 4 バイト、RDF に対して 3 バイト) を引いた値に等しくなります。

HISAM データベースの場合と同様、論理レコードの長さの指定は DBD の中の DATASET ステートメントの RECORD= オペランドで行います。

関連概念:

497 ページの『CI とブロックのサイズの決定』

925 ページの『第 31 章 データベース・タイプの変換』

CI とブロックのサイズの決定

データベースの CI サイズを指定することができます。IDCAMS を使用してデータ・セットを作成する際に、DEFINE CLUSTER コマンドで、DBDGEN ユーティリティーを用いて計算した CI サイズを使用する必要があります。

CI サイズに基づいて、VSAM は DASD トラック上の物理ブロックのサイズを決定します。 VSAM は常に、可能な最大の物理ブロック・サイズを使用しますが、それは最大のブロック・サイズにすれば、トラック上のスペースの使用効率が一番良くなるからです。したがって、データベース管理者の行う CI サイズの選定が重要になります。その選別の際には、装置のオーバーヘッドでなく、ユーザーのデータのためにトラック上の高いスペース使用率を確保することを目標として考えます。

トラック・サイズは装置ごとに異なっており、多くの異なる CI サイズを指定することができます。さまざまな CI のサイズを指定することができるので、VSAM が選定する物理ブロック・サイズはさまざまになり、装置のオーバーヘッドの要因に基づいて決まります。VSAM データ・セットの使用法については、 *z/OS DFSMS* カタログのためのアクセス方式サービス・プログラム を参照してください。

関連概念:

925 ページの『第 31 章 データベース・タイプの変換』

関連タスク:

772 ページの『直接アクセス・ストレージ・デバイスの変更』

496 ページの『HD データベースの論理レコード長の選択』

ブロック、CI、およびレコードのサイズ指定に関する推奨事項

以下の推奨事項によって、適切なサイズのブロック、CI、およびレコードをデータベースが確実に使用するようにできます。

- 一般に、大きな CI サイズは順次処理には好適ですが、ランダム処理には向いていません。順次処理が重要な索引および HISAM データベースの場合、一般的には 8 K 以上の CI サイズが最適です。
- 副次索引や HIDAM 1 次索引などの INDEX データベースの場合は、DATASET ステートメントで RECORD パラメーターを指定しないでください。IMS は適切な CI サイズを自動的に計算します。
- PSINDEX、INDEX、PHIDAM、HIDAM、および HISAM データベースによって使用される KSDS データ・セットの場合、IDCAMS DEFINE ステートメントの RECORDSIZE パラメーターの値を決定するには、DBDGEN プロセスからの出力リストを使用します。
- KSDS データ・セットのデータ・コンポーネントの場合は、IDCAMS DEFINE ステートメントで CI サイズを指定します。データ・コンポーネントの CI サイズは、CI に格納される論理レコードの数を決定します。KSDS データ・セットの索引コンポーネントには、CI サイズを指定しないでください。デフォルトでは、DFSMS が KSDS データ・セットの索引コンポーネントに最適な CI サイズを自動的に選択します。
- VSAM ESDS および KSDS データ・セットの CI サイズについては、IDCAMS DEFINE ステートメントの RECORDSIZE パラメーターで指定した値を使用し

て、DATASET ステートメントで SIZE パラメーターの値を指定します。VSAM ESDS および KSDS データ・セットの CI サイズは、DBDGEN SIZE パラメーターではなく、IDCAMS RECORDSIZE パラメーターによって決定されます。それぞれのパラメーターに同じ値を使用すれば、有効な値を取り違えずに済みます。

- HDAM および HIDAM OSAM データ・セットの場合は、DATASET ステートメントの BLOCK パラメーターでなく、SIZE パラメーターを指定します。
- HISAM データベースの場合は、レコード・サイズが可変である場合を除き、最大レコード・サイズを格納するのに十分な大きさの値を、DATASET ステートメントの RECORD パラメーターに指定します。使用する HISAM データベースのレコード・サイズが可変である場合は、スペースの浪費を避けるために、大多数のデータベース・レコードを保持するのに十分な大きさの RECORD サイズを指定し、オーバーフロー・データ・セットに、最大レコード・サイズを保持するのに十分な大きさのレコード・サイズを指定します。
- OSAM ブロック・サイズおよび VSAM ESDS CI サイズの場合、これらのブロック・サイズと CI サイズは、データベース・レコード全体を保持するのに十分な大きさにしてください。データベース・レコードのサイズが非常に大きいか、大きく変化する場合は、ブロックまたは CI のサイズを、最も頻繁にアクセスするデータベース・レコード内のセグメントを保持するのに十分な大きさにします。
- SEGM ステートメントでは、FREQ パラメーターを指定しないでください。

オープンしている全機能データベース・データ・セットの数

IMS では、複数のデータベースによって同時にオープンできる全機能データベース・データ・セットの数に制限はありません。ただし、使用する IMS 構成と、ご使用のシステムで稼動している可能性があるその他の z/OS サブシステム (例えば、Db2 for z/OS など) の両方によって、オープンできるデータ・セットの数が潜在的に制限される場合があります。

全機能データベースの場合、オープンするデータ・セットの数が多いと制限を受ける可能性があるリソースの 1 つは、16 MB 境界の上と下の両方のストレージを必要とする DL/I 分離アドレス・スペース (DLISAS) の専用ストレージです。

バッファリング・オプション

データベース・バッファーとは、仮想記憶域内に定義されるスペース域です。アプリケーション・プログラムがデータベースの中のあるセグメントを処理しようとするときには、このセグメントが入っているブロックまたは CI 全体が、このデータベースから 1 つのバッファーの中に読み込まれます。アプリケーション・プログラムは、このセグメントがバッファーの中にある間にこのセグメントを処理します。

処理の一部として、バッファーの中のセグメントが変更される場合、バッファーの内容をデータベースに書き戻して、データベースを最新の状態に保たなければなりません。

バッファーのサイズと数とは、最高のパフォーマンスが得られるものを選ぶ必要があります。データベースで OSAM を使用する場合には、OSAM 順次バッファリングを使用するよう決めることもできます。このトピックのサブトピックが、これらの決定に役立ちます。

関連タスク:

511 ページの『SB 制御ステートメントによる SB の要求』

仮想記憶域内における複数のバッファー

ユーザーが、仮想記憶域内に必要なバッファーの数とそのサイズを指定することができます。サイズが異なる複数のバッファーを指定することができます。

完全な 1 つのブロックまたは CI が 1 つのバッファーの中に読み込まれるので、バッファー・サイズは、少なくともこのバッファーに読み込まれるブロックまたは CI と同じ大きさでなければなりません。最高のパフォーマンスを得るためには、仮想記憶域に複数のバッファーを設けてください。そのようにする理由を理解するためには、バッファーの概念と仮想記憶域におけるバッファーの用法についていくつかの事項を知る必要があります。

アプリケーション・プログラムが必要とするデータがすでにバッファーの中にある場合には、このデータをすぐに使うことができます。アプリケーション・プログラムは、このデータがデータベースからバッファーに読み込まれるのを待つ必要はありません。アプリケーション・プログラムは待たないので、パフォーマンスが良くなります。仮想記憶域の中に複数のバッファーを設けており、1 つのバッファーは 1 つの CI またはブロックの中のすべてのセグメントを収容するのに十分なだけ大きくすることにより、アプリケーション・プログラムが必要とするデータがすでに仮想記憶域にある可能性が高くなります。したがって、仮想記憶域に複数のバッファーを設けておく理由は、アプリケーション・プログラムの待ち時間の一部を除去することにあります。

仮想記憶域においては、バッファーはすべてバッファー・プールに置かれます。VSAM と OSAM に対しては別個のバッファー・プールがあります。バッファー・プールはいくつかのサブプールに分割されています。各サブプールは、サブプール定義ステートメントで定義されています。各サブプールは指定された数の同一サイズのバッファーで構成されます。 OSAM と VSAM を使用すると、同じサイズのバッファーを持つ複数のサブプールを指定することができます。

サブプール・バッファー使用チェーン

サブプールにおいては、バッファーは使用された順序でチェーニングされています。この編成は使用チェーン と呼ばれます。

最後に使用されたバッファーは使用チェーンの一番上に、最も使用頻度の少ないバッファーは使用チェーンの一番下にあります。

バッファー・ハンドラー

バッファーが必要になると、バッファー・ハンドラーという内部コンポーネントが、使用チェーンの一番下にあるバッファーを選びます。というのは、最も使用頻度の少ないバッファーに、アプリケーション・プログラムに再使用のために必要なデータが入っていることは比較的少ないからです。

選定されたバッファーにアプリケーション・プログラムが変更したデータが入っていれば、バッファーの内容をデータベースに書き戻してからそのバッファーを使用します。これによって、前述のアプリケーション・プログラムの待ち時間が発生します。

バックグラウンド書き込みオプション

VSAM を使用している場合には、バックグラウンド書き込みというオプションを用いて待ち時間を短縮または除去することができます。

それを使用していない場合には、バッファーの数とサイズを慎重に選定することにより待ち時間を制御してこれを短縮します。

関連タスク:

514 ページの『VSAM オプション』

共用リソース・プール

ユーザーは、複数の VSAM ローカル共用リソース・プールを定義することができます。複数のローカル共用リソース・プールがあれば、同一サイズの VSAM サブプールを複数指定することができます。

まず複数の共用リソース・プールを生成してから、各プールに、他のローカル共用リソース・プールにある他の VSAM サブプールとサイズが等しい VSAM サブプールを保管します。そうすると、ある特定のデータベースのデータ・セットを共用リソース・プールに割り当てることによりこのデータ・セットを特定のサブプールに割り当てることができます。このデータ・セットは、その制御インターバル・サイズに基づいて、割り当てられた共用リソース・プールの中の特定のサブプールに送られます。

分離サブプールの使用

制御インターバル・サイズにほとんどあるいはまったく同じ VSAM データ・セットが数多くある場合は、大きい 1 つのサブプールをバッファーのサイズが等しい別のサブプールで置き換えることにより、パフォーマンス上の利益が得られます。VSAM データ・セット用にサイズが等しい別々のサブプールを生成すると、OSAM 複数サブプール・サポートに似た利点が得られます。

1 つの VSAM ローカル共用リソース・プール内に、VSAM KSDS 索引とデータ・コンポーネントのための別々のサブプールを生成することができます。別々のサブプールを生成する利点は、索引とデータのコンポーネントがバッファーを共用したり、同じサブプールの中のバッファーについて競合したりする必要がないことです。

ハイパースペース・バッファリング

複数の VSAM ローカル共用リソース・プールを使用すると、ハイパースペース・バッファリングを使用する利点が広がります。

ハイパースペース・バッファリングを用いると、4K および 4K の倍数のバッファーのバッファリングを拡大して、仮想記憶域に割り振られたバッファーばかりでなく、拡張された記憶域に割り振られたバッファーも組み込むことができます。複数

ローカル共用リソース・プールおよびハイパースペース・バッファリングを使用すると、特定の参照パターンがあるデータ・セット (例えば、1 次索引データ・セット) を、ハイパースペースで支持されたサブプールに分離することができます。これによって、データベースの処理に必要な VSAM 読み取り入出力アクティビティーが減ります。

ハイパースペース・バッファリングは、IMS の初期設定時に活動化されます。バッチ・システムでは、必要な制御ステートメントを DFSVSAMP データ・セットに入れます。オンライン・システムでは、制御ステートメントを、メンバー名 DFSVSMnn の IMS.PROCLIB データ・セットに入れます。ハイパースペース・バッファリングは、VSRBF サブプール定義ステートメントに適用される 1 つまたは 2 つのオプション・パラメーターを通して、VSAM バッファーに対して指定されます。

Hiperspace™・バッファー・プールに割り振ることができる合計スペースは、2 GB までに制限されています。バッファー数にバッファー・サイズを乗算した値が 2 GB を超えた場合、IMS はプール・サイズを 2 GB に設定し、警告メッセージを発行します。

関連概念:

779 ページの『ハイパースペース・バッファリングのパラメーター』

バッファー・サイズ

バッファー・サイズとしては、バッファーの中に読み込まれる CI とブロックのサイズに等しいかそれより大きいものを選定してください。

有効なバッファー・サイズは数多くあります。 CI サイズまたはブロック・サイズより大きいバッファーを選択すると、仮想記憶域が無駄になります。

例えば、CI サイズが 1536 バイトであるとします。この CI を保持するための有効な最小のバッファー・サイズは、2048 バイトです。この場合には 512 バイト (2048 - 1536) が無駄になるので、この CI サイズとバッファー・サイズは良い選択ではありません。

バッファーの数

各サイズのバッファー数を選定する際には、バッファーが必要なときには通常使用可能になっていること、アプリケーション・プログラムの処理中には仮想記憶域に最適量のデータが保持されていること、およびアプリケーション・プログラムの待ち時間が最小限になるようなバッファー数を選定してください。

バッファーの数を選別する際のトレードオフは、各バッファーが仮想記憶域を使い尽くすかどうかです。

最初にバッファー・サイズとバッファーの数を選定するときには、データベースの設計とアプリケーションの処理要件について知っている事柄に基づいて、科学的な推定を行います。バッファーのサイズと数を選定し、インプリメントした後、用意されているさまざまなモニター・ツールの助けを借りて先の科学的な推定がどのような効果をあげているか判別することができます。

バッファー・サイズとバッファーの数は、システムの初期設定時に指定します。この両方は、最適なパフォーマンス得るために必要に応じていつでも変更 (チューニング) することができます。

関連概念:

699 ページの『第 28 章 データベースのモニター』

707 ページの『第 29 章 データベースのチューニング』

925 ページの『第 31 章 データベース・タイプの変換』

VSAM バッファー・サイズ

VSAM をアクセス方式として使用している場合、選定できるバッファー・サイズは以下のとおりです (単位はバイト数)。

512

1024

2048

4096

8192

12288

16384

20480

24576

28672

32768

バッファー・スペースを無駄にしないように、バッファー・サイズに有効な CI サイズと同じサイズを選択してください。 VSAM データ・クラスターに対する有効な CI サイズは以下のとおりです。

- 8192 バイト (すなわち 8K バイト) までのデータ・コンポーネントに対しては、CI サイズは 512 の倍数でなければなりません。
- 8192 バイト (すなわち 8K バイト) を超えるデータ・コンポーネントに合うように、CI サイズは 2048 の倍数でなければなりません (最大 32768 バイトまで)。

VSAM カタログを使用する場合の VSAM 索引クラスターに対する有効な CI サイズ (単位はバイト数) は、以下のとおりです。

512

1024

2048

4096

統合カタログ機能のカタログを使用する場合の VSAM 索引クラスターに対する有効な CI サイズは、以下のとおりです。

- 8192 バイト (すなわち 8K バイト) までの索引コンポーネントの場合は、CI サイズは 512 の倍数でなければなりません。
- 8192 バイト (すなわち 8K バイト) を超える索引コンポーネントに対しては、CI サイズは 2048 の倍数でなければなりません (最大 32768 バイトまで)。

OSAM バッファー・サイズ

OSAM データ・セットの場合には、バッファー・スペースが無駄にならないように、できれば、有効なブロック・サイズと同じバッファー・サイズを選定するようにしてください。 OSAM データ・セットに応じた有効なブロック・サイズは、 18 から 32768 バイトまでの任意のサイズです。

OSAM をアクセス方式として使用している場合、選定できるバッファー・サイズは以下のとおりです (単位はバイト数)。

512

1024

2048

2048 の倍数で最大 32768 まで

制約事項: OSAM データ・キャッシングのために順次バッファリングとカップリング・ファシリティーを使用する場合は、OSAM データベース・ブロック・サイズを 256 バイト (10 進数) の倍数で定義する必要があります。ブロック・サイズをそのように定義しなければ、カップリング・ファシリティーから ABENDS0DB が戻される可能性があります。この条件は、IMS システムが読み取り専用モードでデータベースにアクセスしている場合でも存在します。

バッファーの指定

システムの初期設定時に、必要なバッファーの数とそのサイズを指定します。

ユーザーの指定は制御ステートメントの形式でシステムに渡され、以下のところに保管されます。

- バッチ、ユーティリティーでは、DFSVSAMP データ・セット
- IMS DCCTL および DBCTL 環境では、メンバー名 DFSVSMnn の IMS.PROCLIB データ・セット

次の例は、必要な制御ステートメントの指定方法を示します。

- OSAM の場合は 2048 バイトのバッファー 4 つ
- VSAM の場合は 2048 バイトのバッファー 4 つ、および 1024 バイトのバッファー 15 個

```
//DFSVSAMP DD *
⋮
VSRBF=2048,4
VSRBF=1024,15
IOBF=(2048,4)
/*
```

z/OS DFSMS は、VSAM データ・セットの索引コンポーネントの CI について、最適サイズを計算します。DFSMS は、CI サイズを増やす必要があると判断した場合、IDCAMS DEFINE ステートメントで指定された CI サイズをオーバーライドします。DFSMS が、関連するバッファー・プールに指定されたサイズを超えて CI サイズを増やした場合、データベースを開くことができず、IMS はメッセージ

DFS0730I と判別コード O,DC を発行します。CI サイズがバッファー・プール・サイズを超えて増やされた場合は、CI サイズに合わせてバッファー・プール・サイズを増やす必要があります。

IOBF= パラメーターを用いて、OSAM バッファーをストレージの中に固定することができます。 VSAM では、バッファーは OPTIONS ステートメントの中の VSAMFIX= パラメーターを用いて固定します。一般に、バッファーをストレージの中に固定すると、ページ不在は起こらなくなり、パフォーマンスは向上します。ページ不在が起きるのは、命令があるページ (ある特定のストレージ) を必要とするとき、そのページがストレージの中にない場合です。

OSAM については、ストレージにバッファーとそのバッファーの接頭部またはバッファーの接頭部とサブプール・ヘッダーを固定することができます。さらに、バッファー・サブプールを選択して固定することが可能です。すなわち、あるバッファー・サブプールを固定し、他のバッファー・サブプールを固定しないでおくことができます。バッファー・サブプールは、IOBF= パラメーターを用いて固定します。

IOBF= パラメーターを使用すると、以下を指定できます。

- サブプール内のバッファーのサイズ。
- サブプール内のバッファーの数。3 以下の数を指定すると、IMS は 3 を、また 4 以上を指定すると、IMS は指定した数を提供します。十分な数のバッファーを指定しないと、アプリケーション・プログラムの呼び出しにより、バッファー・スペースの待ち時間が浪費されることになります。
- このサブプール内のバッファーとバッファー接頭部を固定する必要があるかどうか。
- このサブプール内のバッファー接頭部、およびサブプール・ヘッダーを固定するか必要があるかどうか。
- サブプールに割り当てる ID。この ID は、DBD ステートメントと一緒に、ある 1 つのデータ・セットに特定のサブプールを割り当てるために使用されます。この DBD ステートメントは、DBD 生成中に使用される DBD ステートメントではなく、実行時に指定されるものです。この ID を使用すると、同一サイズのバッファーを持つ複数のサブプールを持つことができます。以下のようなことを行うのにこのパラメーターを使用することができます。
  - サブプール相互間のアクティビティー分布をよりよくする
  - 新しいデータベースのアプリケーションを「専用」サブプールに差し向ける
  - サブプールに対する BMP と MPP 間の競合を制御する

関連概念:

IMS バッファー・プール (システム定義)

OSAM サブプール定義 (システム定義)

関連タスク:

514 ページの『VSAM オプション』

関連資料:

IMS PROCLIB データ・セットの DFSDFxxx メンバー (システム定義)

OSAM サブプールの定義 (システム定義)

IMS プロシージャーの DD ステートメント (システム定義)

OSAM 順次バッファリング

順次バッファリング (SB) は、OSAM データベースのデータ・セット用に使用される通常のバッファリング技法の拡張機能です。

SB がアクティブな場合には、複数の連続ブロックを 1 回の入出力操作でデータベースから読み取ることができます。 (SB には OSAM 書き込み操作の拡張機能はありません。) この技法を用いると、データベースの順次処理を行う多くのプログラムとユーティリティーで経過時間を節減することができます。

関連タスク:

937 ページの『既存のデータベースのアンロード』

順次バッファリングの概要

OSAM 順次バッファリングでは、1 回の入出力操作で 10 個の連続ブロックの順次読み取り が実行されます。一方、通常の OSAM バッファリング方式では、1 回の入出力操作で 1 つのブロックのみのランダム読み取り が実行されます。

SB を用いないと、ユーザー・プログラムで OSAM バッファー・プールに入っていないブロックを処理しようとするとそのたびに、IMS はランダム読み取りを出さなければなりません。データベースの順次処理を行うプログラムの場合、ランダム読み取りは、それぞれの読み取りの間に DASD が 1 回またはそれ以上回転しなければならないので、時間がかかります。

SB は次の 3 つの方法によって入出力読み取り操作に必要な時間を短縮します。

- 順次読み取りでは、10 個の連続ブロックが 1 回の入出力操作で読み取られるため、データベース・データ・セットを順次処理するために必要な入出力操作の回数が減ります。

  順次読み取りを出すと、ユーザー・プログラムで要求したセグメントが入っているブロックとこれに隣接する 9 つのブロックが、データベースから仮想記憶域の中の SB バッファー・プールに読み込まれます。 ユーザー・プログラムで残りの 9 つのブロックのいずれかにあるセグメントを処理しようとする場合には、これらのブロックはすでに SB バッファー・プールに入っているので、入出力操作は不要です。

  例: ユーザー・プログラムで 100,000 個の連続ブロックが入っている OSAM データ・セットを順次処理しようとする場合には、通常の OSAM バッファリング方式を使用すると、100,000 回の入出力操作が必要になります。 SB によって同じデータ・セットを処理すると、10,000 回の入出力操作で済みます。
- データベースの入出力参照パターンをモニターして、特定の入出力要求を満たすのに順次読み取りとランダム読み取りのどちらを用いるとより効果的かを決めます。この決定は、SB で処理する各入出力要求ごとに行われます。
- 順次読み取り入出力操作を、同じアプリケーションの CPC 処理と他の入出力操作とオーバーラップさせます。 オーバーラップした順次読み取りが使用される場合には、SB は将来のブロックに対する要求を予期して、そのブロックがアプ

リケーションで実際に必要になる前に、これらを SB バッファーに読み込んでおきます。 (オーバーラップした入出力がサポートされるのは、バッチ領域と BMP 領域に対してのみです。)

順次バッファリングの利点

OSAM 順次バッファリング (SB) を使用することで、OSAM データ・セットを順次処理するアプリケーション・プログラムまたはユーティリティーを、より速く実行できます。

しかし、ジョブの経過時間は多くの要因の影響を受けるので、節約できる時間の予測は困難です。実際に節約できる時間を判断するには、SB を用いて実験してみる必要があります。

SB から利益を受けるプログラム

SB を使用することによって利益を受けるプログラム、ユーティリティー、機能には、以下のものがあります。

- データベースの順次処理を行う IMS バッチ・プログラム
- データベースの順次処理を行う BMP
- データベースを順次に処理している実行時間の長い MPP、高速機能、および CICS プログラム。

  注**:** 実行時間の短い MPP、IFP、および CICS のプログラムについても SB は使用可能ですが、お勧めできません。このようなオンライン・プログラムでは、実行のたびに SB のための初期設定オーバーヘッドが大きくかかるため、SB はお勧めできません。
- IMS ユーティリティー。次のものが含まれます。
  - オンライン・データベース・イメージ・コピー
  - HD 再編成アンロード
  - データベース部分再編成
  - 調査プログラム
  - データベース・スキャン
  - データベース接頭部更新
  - バッチ・バックアウト
- HALDB オンライン再編成機能

SB の生産性における代表的な利点

データベースの順次処理を行うプログラムとユーティリティーに対して SB を使用することによって、以下のことを行うことができます。

- 既存の順次アプリケーション・プログラムを、減少していく「バッチ・ウィンドウ時間」内に実行します。例えば、バッチ・アプリケーション・プログラムを実行するためにとっておいた時間が 1 時間短縮された場合でも、通常実行するプログラムはすべてこの短縮された時間内に実行することができます。
- 追加の順次アプリケーション・プログラムを同じ時間枠内に実行する
- 順次アプリケーション・プログラムのあるものをより頻繁に実行する

- オンライン・イメージ・コピーをより速くとる
- データベースの再編成に要する時間を短縮する

SB 使用方法の柔軟性

IMS には、SB を要求する方法がいくつかあります。

特定のプログラムやユーティリティーについて SB を使用するように要求するには、PSBGEN の間かまたは SB 制御ステートメントを使用して行います。 SB 初期設定出口ルーチンを用いて、バッチ・プログラムと BMP プログラムのすべてまたは一部で SB を使用するよう要求することができます。

IMS ではまた、システム・プログラマーあるいはマスター端末オペレーター (MTO) が、SB 使用を禁止することによって、この使用要求をオーバーライドすることができます。これを行うには、`SB MTO` コマンドを出すか、または SB 初期設定出口ルーチンを使用します。 1 日の一定の時間帯に SB の使用を禁止することにより、仮想記憶または実記憶の不足の問題を回避することができます。

関連タスク:

510 ページの『SB 使用の要求方法』

SB によるデータのバッファリング方法

以下のセクションでは、順次バッファリング (SB) で行われる内容 (SB によって何がバッファーに入るか、 SB がいつどのように活動化されるか、SB によってバッファーに入れられたデータに何が起きるかなど) について説明します。

SB がバッファーに入れるもの

HD データベースは、複数データ・セット・グループで構成することができます。したがって、1 つのデータベース PCB が複数のデータ・セット・グループを参照することができます。 PCB が参照しているデータベースが論理関係に関与している場合に、1 つのデータベース PCB が複数のデータ・セット・グループを参照することができます。ある特定のデータベース、つまり、ある特定のデータ・セット・グループは、複数のデータベース PCB によって参照されることができます。特定のデータベース PCB が参照する特定のデータ・セット・グループを、以下のセクションでは、*DB-PCB/DSG* の対 と呼びます。

SB が活動化されると、SB はある特定の DB-PCB/DSG の対に対応した OSAM データ・セットのデータをバッファーに入れます。 SB は複数の DB-PCB/DSG の対の場合に同時にアクティブであることが可能ですが、それぞれの対は別々に活動化する必要があります。

HALDB データベースで OSAM SB を使用する場合は、各データベースが複数の区画を持つ可能性があるため、HALDB DB-PCB/DSG の対が区画 ID でさらに修飾されます。HALDB 区画に対して作成された SB ブロックは、アプリケーション・プログラム PST の間で共用できません。

SB の定期的評価と条件付き活動化

IMS は、SB が要求されても、直ちにそれを活動化するわけではありません。その代わり、SB があるプログラムに対して要求されると、IMS は、そのプログラムで

使用している DB-PCB/DSG の対ごとに、その対についての入出力参照パターンとアクティビティー率のモニターを始めます。その後しばらくして、IMS はバッファリング処理に対する一連の定期的評価 の内の第 1 回目を行います。 IMS は、DB-PCB/DSG の対ごとに定期的評価を実行して、その DB-PCB/DSG の対で SB を使用することが有益であるかどうかが決まります。SB の使用が有益であるならば、IMS はその DB-PCB/DSG の対に対して SB を活動化します。これを条件付き活動化 と呼んでいます。

SB が活動化された後も、IMS は引き続き入出力参照パターンとアクティビティー率を定期的に評価していきます。これらの評価に基づいて、IMS は以下のいずれかのアクションを実行できます。

- 一時的に SB を非活動化して、入出力参照パターンとアクティビティー率のモニターを続ける。 一時的な非活動化は、SB バッファーを解除してページ解放するために行います。
- 一時的に、入出力参照パターンとアクティビティー率のモニターを非活動化する。このような一時的な非活動化が行われるのは、SB が非活動化されていて、さらに IMS が引き続き行った評価によっても、SB の使用が依然として有益にはならないと結論された場合のみです。

SB は一時的に非活動化されても、その後の評価に基づいて再度活動化することができます。

DB-PCB/DSG の対に応じて、それぞれ個別に定期的評価を実行します。したがって、IMS は、他の DB-PCB/DSG のいくつかの対に対しては SB がアクティブな状態にある間に、ある 1 つの DB-PCB/DSG の対に関して SB を非活動化することができます。

SB バッファー・ハンドラーの役割

SB が 1 つの DB-PCB/DSG の対の場合に活動化されると、SB バッファーの 1 つのプールがその対に割り振られます。 それぞれの SB バッファー・プールは *n* 個のバッファー・セット (デフォルトの値は 4) で構成され、それぞれのバッファー・セットには 10 個のバッファーが入っています。これらのバッファーは、SB バッファー・ハンドラーと呼ばれる内部コンポーネントにより使用され、順次読み取りによって読み取った 10 個の連続ブロックを保持します。

SB がアクティブである間に、OSAM バッファー・プールの中に見つからないデータベース・ブロックに対する要求はすべて、SB バッファー・ハンドラーに送られます。 SB バッファー・ハンドラーは、これらの要求に以下の方法で応答します。

- 要求されたブロックがすでに SB バッファーの中にある場合には、そのブロックのコピーを OSAM バッファーの中に入れます。
- 要求されたブロックが SB バッファーの中にない場合には、SB バッファー・ハンドラーはこれ以前の入出力要求の記録を分析して、順次読み取りとランダム読み取りのどちらを出すか決めます。ランダム読み取りを出すように決めた場合には、要求されたブロックは OSAM バッファーの中に直接読み込まれます。順次読み取りを出すことに決めた場合には、要求されたブロックとこれに隣接した 9 つのブロックが SB バッファー・セットの中に読み込まれます。順次読み取りが完了すると、要求されたブロックのコピーが OSAM バッファーに入れられます。

- SB バッファー・ハンドラーは、オーバーラップする順次読み取りをいつ開始するかについても決定します。

注: SB バッファー・ハンドラーがオンライン・プログラムからの要求を処理する場合、SB バッファー・ハンドラーが検索するのは、このオンライン・プログラムに割り振られた SB バッファー・プールのみです。

関連概念:

423 ページの『第 18 章 オプション・データベース機能』

『SB の仮想記憶域に関する考慮事項』

シスプレックス・データ共用の概念と用語 (システム管理)

SB の仮想記憶域に関する考慮事項

SB によってバッファーに入れられる DB-PCB/DSG の対には、それぞれ独自の SB バッファー・プールがあります。

デフォルトの値として、各 SB バッファー・プールには、4 つのバッファー・セットが入っています (ただし、IMS はこの値をユーザーに変更させます)。各バッファー・セットには、10 個のバッファーがあります。バッファーの大きさはそれぞれ 1 つの OSAM データ・セット・ブロックを保持するのに十分なものです。

したがって、各 SB バッファー・プールの合計サイズは次のとおりです。

```
4 * 10 * block size
```

SB バッファーは、ページ不在を除去し、入出力操作のパス長を短縮し、パフォーマンスを向上させるために、ストレージの中でページ固定されます。定期的評価により一時的に SB が非活動化された場合、SB バッファーのページ固定が外されてページが解放されます。

バッチ、オンライン、または DBCTL 領域に、SB バッファー・プールを保管するのに十分な仮想記憶域を確保する必要があります。 このストレージ要件は、ブロック・サイズと SB を使用しているプログラムの数に応じて考える必要があります。

バッチや DB/TM のように実記憶が制約される環境では、SB の使用はお勧めできません。

しかし、システムの中には、オンラインのピーク時などのように、一定の時間帯のみストレージが制約されるものがあります。この場合には、SB 初期設定出口ルーチンを使用して、1 日の特定の基準 (時刻) に従って、SB の使用を制御することができます。

関連概念:

507 ページの『SB によるデータのバッファリング方法』

関連タスク:

511 ページの『SB 制御ステートメントによる SB の要求』

関連資料:

順次バッファリング初期設定出口ルーチン (DFSSBUX0) (出口ルーチン)

SB 使用の要求方法

IMS には、どのユーザー・プログラムとデータベースで OSAM 順次バッファリング (SB) を使用するかを指定する方法が 2 つあります。

- PSB の生成中に、または SB 制御ステートメントを使用して、どのプログラムとユーティリティーで SB を使用すべきかを明示的に指定することができます。
- デフォルトの指定として、ユーザーのバッチ・プログラムと BMP プログラムおよびユーティリティーのすべてまたは一部で SB を使用すべきであることを指定することができます。これを行うには、 SB ユーザー出口ルーチンをコーディングするか、あるいは IMS で提供される SB ユーザー出口ルーチンのサンプルを使用します。

使用する方法を決定してください。どの BMP プログラムとバッチ・プログラムで順次処理を使用するか知る必要はないので、2 番目の方法の方が簡単です。しかし、デフォルトの指定として SB を使用すると、実記憶域と仮想記憶域の使用が増加して制御が不可能になり、システムのパフォーマンスに影響を及ぼすことがあります。一般に、ストレージが制約された z/OS 環境において IMS を実行している場合には、最初の方法を使用します。ストレージに制約がない z/OS 環境で IMS を実行している場合には、2 番目の方法を使用します。

関連概念:

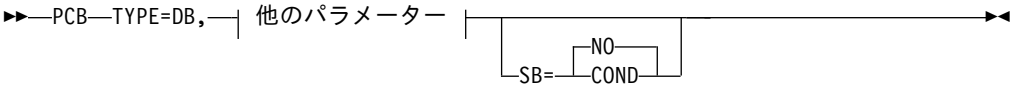
507 ページの『SB 使用方法の柔軟性』

PSB 生成中の SB の要求

PSB の生成中に SB を要求するには、アプリケーションの PSB の PCB マクロ命令に SB=COND を指定します。

このキーワードは、SB を用いてバッファーに入れたいデータベースの PCB ごとにコーディングします。 (これは、実行時に PSB を使用しない IMS ユーティリティーでは、行えません。)

次の図は、PCB ステートメントの中の SB キーワードの構文を示したものです。



COND
SB をこの PCB に応じて条件付きで活動化するという指定です。

NO SB をこの PCB 用として使用しないという指定です。

SB キーワードをユーザーの PCB に組み込まないと、SB 出口ルーチンで他の指定がされていない限り、IMS のデフォルトによって NO になります。

SB キーワードの値は、SB 制御ステートメントによってオーバーライドすることができます。

以下の例は、SB の条件付き活動化を要求するための PCB ステートメントのコーディングを示しています。

```
SKILLA   PCB   TYPE=DB,DBDNAME=SKILLDB,KEYLEN=100,
               PROCOPT=GR,SB=COND
```

関連タスク:

『SB 制御ステートメントによる SB の要求』

関連資料:

プログラム仕様ブロック (PSB) 生成ユーティリティー (システム・ユーティリティー)

SB 制御ステートメントによる SB の要求

SBPARM 制御ステートメントは、オプションの //DFSCTL ファイルに入れることができます。このファイルは、バッチ領域、従属領域、またはオンライン領域の JCL の中の //DFSCTL DD ステートメントで定義します。

SBPARM 制御ステートメントは、次の目的で使用できます。

- どのデータベースの PCB (およびこのデータベースの PCB によって参照されるどのデータ・セット) で SB を使用するかの指定。
- バッファー・セットの数のデフォルトのオーバーライド

この制御ステートメントによって、ユーザーは PSB を生成し直すことなく PSB の指定を変更することができます。

SB を使用するように要求するキーワードは、DBD 名、DD 名、PSB 名、および PCB ラベルのすべてのあるいは特定のものに対して指定することができます。また、これらのキーワードを組み合わせて SB をいつ使用するかをさらに制限することができます。

SBPARM 制御ステートメントの BUFSETS キーワードを使用することにより、SB バッファー・プールに割り振られるバッファー・セットの数を変更することができます。バッファー・セットのデフォルトの数は 4 です。データベースの編成がよくないと、順次処理の効率を上げるために 6 個以上のバッファー・セットが必要になることがあります。データベースの編成が適切であれば、バッファー・セットが 2 個で済むことがあります。ユーザーのデータベースがどの程度よく編成されているかの指標は、オプションの //DFSSTAT 報告書で調べることができます。

次の例は、すべての DBD 名、DD 名、PSB 名、および PCB 名用に SB の条件付き活動化を要求するのに必要な SBPARM 制御ステートメントを示しています。

```
SBPARM ACTIV=COND
```

次の例は、以下のことに必要なパラメーターを示しています。

- PSB 生成の間に「DBDNAME=SKILLDB」を指定してコーディングしたすべての PCB に対して SB の条件付き活動化を要求する。
- バッファー・セットの数を 6 にセットする。

```
SBPARM ACTIV=COND,DB=SKILLDB,BUFSETS=6
```

関連概念:

509 ページの『SB の仮想記憶域に関する考慮事項』

775 ページの『バッファーの調整』

498 ページの『バッファリング・オプション』

//DFSSTAT 報告書 (システム管理)

関連タスク:

510 ページの『PSB 生成中の SB の要求』

関連資料:

順次バッファリング制御ステートメント (システム定義)

SB 初期設定出口ルーチンによる SB の要求

SB 出口ルーチンによって、アプリケーションのスケジューリング時に SB の使用を動的に制御することができます。

SB 初期設定出口ルーチンを使用すると、以下のことができます。

- バッチ・プログラムと BMP プログラムのすべてまたはその一部について、SB の条件付き活動化を要求する。
- SB の使用を許可したり禁止したりする。
- バッファー・セットのデフォルトの数を変更する。

これを行うには、ユーザー独自の SB 出口ルーチンを作成するか、あるいはサンプルの SB 出口ルーチンを選択して、それを DFSSBUX0 という名前で IMS.SDFSRESL の中にコピーします。

IMS では、SB 出口ルーチンの 5 つのサンプルが IMS.SDFSSRC と IMS.SDFSRESL の中に提供されています。サンプル・ルーチンの中の 3 つは、アプリケーション・プログラムとユーティリティーの各種サブセットに対して SB を要求します。サンプル・ルーチンの 1 つは一日の一定の時間帯に SB を要求し、もう 1 つのルーチンは、SB の使用を禁止します。これらのサンプル・ルーチンは、そのまま使用することもできるし、ユーザーの必要に合わせて変更することもできます。

SB 初期設定出口ルーチンに関する詳細な指示については、*IMS V13* 出口ルーチンに記載されています。

複数のソース・ステートメントによって提供される SB オプションまたはパラメーター

同一の SB オプションまたはパラメーターを 2 個所以上に指定する場合は、IMS が以下の一連の優先順位に従って、どのオプションを実行するかを決定します。

同一の SB オプションまたはパラメーターを 2 か所以上に指定する場合は、以下の優先順位が適用されます (項目 1 が一番高い優先順位)。

1. SB 制御ステートメントの指定 (*n* 番目の制御ステートメントが *m* 番目の制御ステートメントに優先します (*n* > *m*))
2. PSB の指定
3. SB 初期設定出口ルーチンで変更したデフォルト
4. IMS のデフォルト

オンライン・システムでの SB の使用

オンラインの IMS 環境または DBCTL 環境で SB を使用可能にするには、IMS システム・プログラマーが SB モジュールを IMS にロードするように明示的に要求する必要があります。そのためには、DFSVSMxx メンバーに SBONLINE 制御ステートメントを入れます。

デフォルトでは、IMS はオンライン環境で SB モジュールをロードしません。これは、仮想記憶域の要件が著しく増加するのを防ぐのに役立ちます。

SBONLINE 制御ステートメントの 2 つの形は次のとおりです。

```
SBONLINE
```

または

```
SBONLINE,MAXSB=nnnnn
```

ここで、*nnnnn* は SB バッファーに使用できる最大のストレージ (K バイト単位) です。

この MAXSB の限界に到達すると、終了するオンライン・プログラムが SB バッファー・スペースを解放するまで、IMS は SB バッファーをオンライン・アプリケーションに割り振るのを中止します。 MAXSB= キーワードを指定しないと、デフォルトにより、SB バッファーの最大ストレージの限界はなくなります。

SBONLINE 制御ステートメントの詳細なコーディング方法の指示については、「*IMS V13* システム定義」に記載されています。

SB 使用の禁止

IMS システム・プログラマーまたはマスター端末オペレーター (MTO) は、SB の使用を禁止することができます。

SB の使用が禁止された場合には、SB の条件付き活動化の要求は無視されます。

SB の使用を禁止する方法は 3 つあります。次に挙げる 3 つの方法です。

1. SB 初期設定出口ルーチンを作成して (あるいは出口ルーチンのサンプルを調整して)、SB の使用を動的に禁止したり許可したりすることができます。この方法は、IMS バッチ、オンライン、または DBCTL 環境で SB を用いる場合に使用できます。
2. MTO コマンド /STOP SB および /START SB を発行すると、IMS オンライン・サブシステム内で、SB の使用を動的に禁止したり、可能にしたりすることができます。
3. SBONLINE 制御ステートメントは、DFSVSMxx メンバーから省略することができます。これにより、IMS が SB モジュールをオンライン・サブシステムにロードできなくなります。この場合には、オンライン・サブシステム内のプログラムから SB を使用できなくなります。

関連資料:

- /STOP SB コマンド (コマンド)
- /START SB コマンド (コマンド)

VSAM オプション

VSAM を使用するデータベースに対して数種類のオプションを選択することができます。

ESDS データ・セットのフリー・スペース、論理レコード・サイズ、および CI サイズなどのオプションの指定については、この章の前のトピックで説明されています。このトピックでは、以下のオプションの機能について説明します。

- IMS を初期設定するとき OPTIONS 制御ステートメントで指定する機能。
- IMS を初期設定するとき POOLID、VSRBF、および DBD の各制御ステートメントで指定する機能。
- データ・セットを定義するときアクセス方式サービスの DEFINE CLUSTER コマンドで指定する機能。

OPTIONS 制御ステートメントで指定するオプションの機能

IMS システム初期設定の間に、VSAM を使用するデータベースのために選択することのできるいくつかのオプションがあります。これらのオプションは、OPTIONS 制御ステートメントで指定します。バッチ・システムでは、指定したオプションは DFSVSAMP という DDNAME を持つデータ・セットの中に置かれます。オンライン・システムでは、指定したオプションは DFSVSMnn というメンバー名を持つ IMS.PROCLIB データ・セットの中に置かれます。どの VSAM オプションを選定するかで、パフォーマンス、データベース内のスペースの使用効率、およびリカバリーに影響を及ぼす可能性があります。このトピックでは、これらの各オプションについて説明し、その使用にかかわる事項についても述べています。

バックグラウンド書き込みの使用 **(BGWRT** パラメーター**)**

データをデータベースから読み取る必要が生じて、アプリケーション・プログラムから呼び出しが出されると、そのデータがバッファーに読み込まれます。このデータが読み込まれるバッファーの中に、変更されたデータが入っている場合には、変更されたこのデータをデータベースに書き戻した後でなければ、このバッファーを使用することはできません。データがデータベースに書き戻されないと、新しいデータがこのバッファーの中に読み込まれたとき、先のデータが失われます (先のデータの上に新しいデータが重ねられます)。こうなると、データベースを更新する方法はありません。

そこで、アプリケーション・プログラムがデータをあるバッファーの中に読み込む必要があるのに、このバッファーの中に変更されたデータが入っている場合には、このバッファーがデータベースに書き込まれる間、このアプリケーション・プログラムは待っています。この待ち時間でパフォーマンスは低下します。アプリケーション・プログラムはすぐにでも処理に取りかかれますが、バッファーが使用できる状態ではありません。バックグラウンド書き込みとは、このような理由で失われる待ち時間を減らすために OPTIONS ステートメントで選択できる機能です。

バックグラウンド書き込みの働きを理解するためには、サブプールでバッファーがどのように使用されるかについて多少の知識が必要です。ユーザーは、必要なバッファー数とそのサイズを指定します。同じサイズのバッファーはすべて同じサブプールに属しています。 1 つのサブプール内のバッファーは使用チェーン上にあります。すなわち、これらのバッファーは最高使用頻度または最低使用頻度の順序でチ

ェーニングされています。最近使用されたバッファーは使用チェーンの上部にあり、最低使用頻度のバッファーは下部にあります。

バッファーが必要になると、VSAM のバッファー・マネージャーが使用チェーンの一番下のところにあるバッファーを選定します。この理由は、使用チェーンの下部のバッファーに、再び使用されそうなデータが入っていることは比較的少ないからです。 VSAM バッファー・ハンドラーが選定したバッファーの中に、変更されたデータが入っている場合には、このバッファーが使用される前にこのデータはデータベースに書き込まれます。これが行われている間、アプリケーション・プログラムは待っていることになります。

バックグラウンド書き込みは次の問題を解決します。つまり、VSAM バッファー・マネージャーが任意のサブプールの中のバッファーを取得するとき、それは (バックグラウンド書き込みが使用されていると) 使用チェーンにおける次のバッファーを見ます。使用チェーンにおける次のバッファーが、次に使用されます。このバッファーに変更されたデータが入っていると、IMS に通知され、ここでバックグラウンド書き込み機能が呼び出されます。バックグラウンド書き込み機能は、VSAM に、使用チェーンの最下部にある何 % かのバッファーからデータベースにデータを書き込ませます。 VSAM はすべてのサブプールでこれを行います。データベースに書き込まれたデータは、依然としてバッファーの中にあるので、アプリケーション・プログラムはまだバッファーの中の任意データを使用することができます。

順次処理では、バックグラウンド書き込みはきわめて有益な機能です。オンライン・システムでは、バッチ・システムほど、バックグラウンド書き込みは重要ではありません。それは、オンライン環境では、IMS が同期点で自動的にバッファーをデータベースに書き込むからです。

バックグラウンド書き込みを使用するには、OPTIONS ステートメントで BGWRT=YES,n と指定します。ここで n は、データベースに書き込まれるそれぞれのサブプール内のバッファーのパーセントです。 BGWRT= パラメーターをコーディングしない場合、デフォルトは BGWRT=YES で、デフォルトのパーセントは 34% です。アプリケーション・プログラムがバッファーを継続して使用するが、その中のデータを再検討することがないのであれば、n を 99% にすることもできます。こうすれば、必要なときには、バッファーは通常、使用可能になります。

CICS は、バックグラウンド書き込み機能をサポートしません。

挿入方針の選択 **(INSERT** パラメーター**)**

KSDS の中の CI のフリー・スペースを取得するには、DEFINE CLUSTER コマンドでそのことを指定します。 DBD の FRSPC= キーワードを用いて KSDS のフリー・スペースを指定することはできません。

DEFINE CLUSTER コマンドでフリー・スペースを指定する場合は、次のことを決定する必要があります。

- 複数のルート・セグメントが同時に KSDS に挿入される場合、指定したフリー・スペースを残しておこうとしているのか、あるいはこのフリー・スペースを使用してもよいのか。

- CI の分割を引き起こすようなルート・セグメントが挿入される場合、このルート・セグメントが挿入される点でこの CI を分割したいのかあるいはこの CI の中央で分割したいのか。

これらの選択は OPTIONS ステートメントの INSERT= パラメーターで指定します。 INSERT=SEQ は、フリー・スペースを残しておきたい、またルート・セグメントが挿入される点で CI を分割したいという指定です。 INSERT=SKP は、フリー・スペースを残しておきたくない、さらに CI の分割はその CI の中央で行いたいという指定です。大抵の場合、INSERT=SEQ を指定して、将来、ルート・セグメントを挿入するときフリー・スペースが使用可能であるようにします。いずれを選択したら最良のパフォーマンスが得られるかは、アプリケーションに応じて決まります。

INSERT= パラメーターを指定しない場合、デフォルト INSERT=SKP が使用されます。

IMS トレース・パラメーターの使用

IMS トレース・パラメーターは、特定のトレース域において問題の解決にトレース情報を提供します。すべてのトレースはシーケンス番号を共用しており、すべてのトレースを調べるとその IMS 環境の概略像が得られるようになっています。

IMS の DL/I、LOCK、および検索のトレースは、バッチ領域の場合を除いて、デフォルトでオンであり、バッチ領域では、デフォルトでオフです。他のトレース・タイプはデフォルトでオフです。

IMS 初期設定時にトレースをオンにすることができます。また、IMS の実行中に `/TRACE` コマンドでトレースを開始したり、停止したりすることもできます。実行時間の長いトレースの出力は、要求すればシステム・ログに保管することができます。

使用するダンプ・オプションの決定 **(DUMP** パラメーター**)**

ダンプ・オプションは、パフォーマンスには一切影響を及ぼさない保守支援機能です。このオプションは、単にバッファー・ハンドラー (内部コンポーネント) で異常終了が起きたとき、どのような異常終了が発生したか記述するにすぎません。DUMP=YES と指定すると、バッファー・ハンドラーで異常終了が起きたときに、制御領域が異常終了します。

VSAM データベース・バッファーと **IOB** をストレージに固定するかどうかの決定 **(VSAMFIX** パラメーター**)**

各 VSAM サブプールには、いくつかのバッファーと入出力制御ブロック (IOB) が入っています。これらのバッファーと IOB をストレージの中で固定すれば、一般にパフォーマンスが向上します。こうすればページ不在が起きることはありません。ページ不在は、命令が、実記憶の中にないページ (特定のストレージ) を参照すると起きます。

OPTION ステートメントの中の VSAMFIX= パラメーターで、バッファー または IOB、あるいはその両方をストレージにおいて固定するかどうかを指定することができます。バッファーまたは IOB を固定すると、これらはすべてのサブプールで固定

されます。 VSAMFIX= パラメーターをコーディングしないと、デフォルトによりバッファーおよび IOB は固定されません。

IMS でバッファーが指定されていると、このパラメーターを CICS 環境の中で使用することができます。

ローカル共用リソースの使用 **(VSAMPLS** パラメーター**)**

OPTIONS ステートメントの中で VSAMPLS=LOCL を指定するのは、ローカル共用リソース (LSR) を対象としたものです。 VSAMPLS=LOCL を指定すると、VSAM 制御ブロックおよびサブプールは、IMS 制御領域の中に置かれます。VSAMPLS=LOCL は、唯一の有効なオペランドであり、デフォルトです。

関連概念:

500 ページの『バックグラウンド書き込みオプション』

786 ページの『VSAM オプションの調整』

関連タスク:

503 ページの『バッファーの指定』

518 ページの『KSDS のフリー・スペースの指定 (FREESPACE パラメーター)』

関連資料:

IMS PROCLIB データ・セットの DFSVSMxx メンバー (システム定義)

POOLID、DBD、および VSRBF 制御ステートメントで指定するオプションの機能

IMS の初期設定中に選択されるオプションにより、VSAM ローカル共用リソース・プールのサイズと構造が決まります。

バッチ環境では、これらのオプションを DFSVSAMP という DDNAME を持つデータ・セットで指定します。オンライン・システムでは、DFSVSMnn というメンバー名を持つ IMS.PROCLIB データ・セットでオプションを指定します。

これらのオプションを使用すると、次の定義によって IMS のパフォーマンスを向上させることができます。

- 複数のローカル共用リソース・プールを定義する
- サブプールを特定のデータ・セット専用にする
- VSAM データ・セットの索引コンポーネントとデータ・コンポーネントに対して、別個のサブプールを定義する

関連概念:

VSAM および OSAM サブプールの指定 (システム定義)

関連資料:

IMS PROCLIB データ・セットの DFSVSMxx メンバー (システム定義)

アクセス方式サービス DEFINE CLUSTER コマンドで指定するオプションの機能

VSAM データ・セットを定義する際に、パフォーマンスに影響するいくつかのオプションの機能を選択できます。

これらの機能は、アクセス方式サービス・コマンド DEFINE CLUSTER で指定されます。 HALDB データベースの場合、DEFINE CLUSTER コマンドに REUSE パラメーターを指定する必要があります。IMS オンライン・リカバリー・サービスは、指定されていれば、その REUSE パラメーターを利用します。

関連資料 このコマンドとそのすべてのパラメーターの詳細については、「*z/OS DFSMS カタログのためのアクセス方式サービス・プログラム*」で説明されています。

「ファジー」イメージ・コピーをデータベース・イメージ・コピー 2 ユーティリティー (DFSUDMT0) で取得できるようにする指定

データベース・イメージ・コピー 2 ユーティリティー (DFSUDMT0) で KSDS の「ファジー」イメージ・コピーを取得できるように設定するためには、KSDS が SMS の管理下にある AMS DEFINE または ALTER コマンドで BWO(TYPEIMS) パラメーターを指定してください。

KSDS のフリー・スペースの指定 (FREESPACE パラメーター)

KSDS 内の CI にフリー・スペースを挿入するには、DEFINE CLUSTER コマンドで FREESPACE パラメーターを使用します。

DBD の FRSPC= キーワードを用いて KSDS のフリー・スペースを指定することはできません。

FREESPACE パラメーターでフリー・スペースをパーセントとして指定します。パラメーターのフォーマットは、FREESPACE (x,y) です。ここで、

x データベースがロードされるとき、または最初のロードの後で CI 分割が起きるとき、1 つの CI の中でフリー・スペースとして残しておくべきスペースの割合です。

y データベースがロードされるとき、または最初のロードの後で CA (制御域) 分割が起きるとき、1 つの CA の中にフリー・スペースとして残しておくべきスペースの割合です。

OPTIONS 制御ステートメントに INSERT=SEQ とコーディングすれば、CI または CA が分割されるとき、フリー・スペースが保存されます。

FREESPACE パラメーターを指定しないと、デフォルトによりデータベースのロード時にこの KSDS データ・セットの中にフリー・スペースは残されません。

関連タスク:

514 ページの『VSAM オプション』

最初のロード時にデータ・セット・スペースを事前フォーマット設定するかどうかの指定

最初の VSAM データ・セットのロード時に、SPEED|RECOVERY パラメーターで、このデータ・セットの事前フォーマット設定を行うかどうかを指定することができます。

SPEED が指定されていると、このデータ・セットの事前フォーマット設定を行うべきでないと指示していることになります。データ・セットの事前フォーマット設定の利点は、最初のロードが失敗した場合、この失敗からリカバリーして、正しく書き込まれた最後のレコードの後から、データベース・レコードのロードを継続できるというところにあります。ただし、IMS は RECOVERY オプションをサポートしません (ユーティリティー制御機能の使用時を除く)。したがって、このオプションは指定できますが、リカバリー処理を実行することはできません。 RECOVERY パラメーターを指定してもリカバリーを利用することはでないので、最初のロードの間のパフォーマンスをよくするために SPEED を指定する必要があります。

ロードの間にデータ・セットのリカバリーを行えるようにしたい場合には、ユーティリティー制御機能の制御のもとでこのデータ・セットをロードすべきです。このユーティリティーは、「*IMS V13* データベース・ユーティリティー」で説明しています。

RECOVERY がこのパラメーターのデフォルトです。

OSAM オプション

OSAM を使用するデータベースに対して使用できるオプションには、次の 2 種類があります。

2 種類のオプションは次のとおりです。

1. DBD で指定するオプション (フリー・スペース、論理レコード・サイズ、CI サイズ)。

   これらのオプションについては、この章の前のセクションで説明しています。
2. IMS を初期設定するとき OPTIONS 制御ステートメントで指定するオプション。

   バッチ・システムでは、このオプションは DFSVSAMP という DDNAME を持つデータ・セットの中に置かれます。オンライン・システムでは、指定したオプションは DFSVSMnn というメンバー名を持つ IMS.PROCLIB データ・セットの中に置かれます。これらの OSAM オプションの選択方法は、パフォーマンス、データベース内でのスペース使用効率、およびリカバリーに影響を及ぼします。

   OPTIONS ステートメントの詳細については、「*IMS V13* システム定義」で説明しています。このステートメントとそのすべてのパラメーターはオプションです。

ダンプ・オプション (DUMP パラメーター)

ダンプ・オプションは、パフォーマンスには一切影響を及ぼさない保守支援機能です。

このオプションは、バッファー・ハンドラー (内部コンポーネント) の異常終了時に、発生した異常終了のタイプを示します。

維持管理の計画

データベースを設計する際、維持管理の計画を立てることを覚えておいてください。例えば、アプリケーションの要求により、データベースが 1 日 16 時間使用可能でなければならないとすれば、アンロードして再ロードするのに 10 時間かかるデータベースは設計しないはずです。

維持管理の計画を立てるためのガイドラインをここで示すことができないのは、このような計画はすべてアプリケーションに依存しているからです。しかし、この計画を立てることを忘れないでください。

例えば、前述した問題の解決策の 1 つは、別個のデータベースを 3 つ設け、これらを別々のボリュームに保管するというものです。別個のデータベースによってキー範囲が異なる場合、どのデータベースを処理したらよいか判断するための論理をアプリケーション・プログラムの中に加えておくことができます。この解決策を採用すれば、この 3 つのデータベースを異なる時点で再編成することができ、1 回に 10 時間の再編成の必要はなくなります。データベースが HDAM または HIDAM を使用している場合には、この問題に対する別の解決策として、データベース部分再編成ユーティリティーを用いて部分再編成を行うという方法もあります。

オンライン環境では、イメージ・コピー・ユーティリティーを用いると、データベースをオフラインに切り替えずに一部の維持管理を行うことができます。これらのユーティリティーを使用することにより、データベースまたは区画が、オンライン IMS システムに割り振られ、使用されている間でも、イメージ・コピーを取得することができます。

HALDB は、ラージ・データベースの可用性を大幅に向上させます。ラージ・データベースを区分化することにより、1 つの区画でオフライン保守を実行し、残りの区画は使用可能なままにすることが可能です。

また HALDB データベースをオンラインで再編成すると、そのデータへのアクセスを中断することなく、HALDB のパフォーマンスを向上させることができます。HALDB のオンラインでの再編成を計画する場合は、再編成処理を許容する十分な DASD スペースがあることを確認してください。

関連概念:

726 ページの『データベース部分再編成ユーティリティー (DFSPRCT1)』

738 ページの『HALDB オンライン再編成』

第 22 章 高速機能データベースの設計

アプリケーションの処理要件に最適のデータベース・タイプとオプションの機能をいったん決定したら、データベース設計とオプションの使用に関して一連の決定を行う必要があります。

これらの決定によって、主として、データベースのパフォーマンスと使用可能スペースの使用効率が決まります。以下の事項に基づいて一連の決定を行います。

すでに選択されているデータベース・タイプとオプションの機能

アプリケーションのパフォーマンス要件

オンラインで使用可能なストレージの量

DEDB の設計の指針

高速処理データベース (DEDB) を定義するには、CI と UOW のサイズを決定し、設計ガイドラインに従う必要があります。

関連概念:

IMS 環境でのデータ共用 (システム管理)

DEDB 設計の指針

以下のリストは、DEDB を設計する際の指針を示したものです。

- 親とその子の関係を除き、(PCB で定義される) 論理構造は、DBD によって定義されるセグメント・タイプの階層の順序に従う必要はありません。

  例えば、DDEP セグメントを対象とする SENSEG ステートメントが、SDEP セグメントを対象とする SENSEG ステートメントの前にきてもかまいません。こうすれば、修飾されていない GN 処理において、最初の DDEP セグメントにアクセスする前に、すべての SDEP セグメントが取り出されることがないようにすることができます。
- ほとんどの場合、SDEP セグメントは DEDB 順次従属スキャン・ユーティリティーを使用して、一度にすべて検索されます。あとで SDEP セグメントをそのルート・セグメントと関連付ける必要がある場合には、SDEP セグメント・データの一部としてルート・セグメント識別を加えるよう計画する必要があります。
- ジャーナルは、ある 1 つの DEDB を使用する複数のトランザクションにまたがってデータを収集することによりインプリメントすることができます。競合を最小限に抑えるために、複数のルート・セグメントを持つエリアを設けるよう計画する必要があります。例えば、1 つのルート・セグメントを 1 つのトランザクションまたは領域の専用とする、あるいは、ルート・セグメントを 1 つずつ各端末の専用にすることができます。リソース競合をさらに抑えるために、これらのルート・セグメントに異なる CI を割り当てます。CI は DEDB 割り振りの基本単位であるからです。
- 次に、ユーザーが直面することのある条件とこれを解決する 1 つの方法を示します。ある 1 つの DEDB レコードを対象とするトランザクションのジャーナル

が SDEP セグメントを使用して記録され、さらに、その最後の 20 余りのトランザクションを照会する要件があるとします。

SDEP セグメントには高速挿入能力がありますが、検索されるセグメントごとに、平均 1 回の入出力操作が必要です。ジャーナル・データを SDEP セグメントと DDEP セグメントの両方として挿入すると共に、DDEP セグメントの兄弟チェーンを 20 のオカレンスに限定することにより、これ以外の入出力操作を避けることができるでしょう。 DDEP セグメントに対する置き換え呼び出しまたは挿入呼び出しは、それらのセグメントがルート・セグメント CI の中に収まる場合があるので、必ずしも余分の入出力操作を必要とするわけではありません。データベースに対する唯一の呼び出しが SDEP セグメントの挿入呼び出しである場合にも、ルート・セグメント CI は常にアクセスされます。このためジャーナル項目のオンライン検索要求に応えるのに、SDEP セグメントではなく DDEP セグメントを用いることができます。

- 物理 DDEP 兄弟チェーンが作成されるにつれて、入出力アクティビティーが増えます。アプリケーションで使用が許されるのであれば、SDEP セグメント・タイプは多少の助けとなります。

  このような設計を行うには、1 つのタイプの DDEP セグメントを集めておき、その数が一定の限度に達するたびに、これらを単一セグメントとして挿入する必要があります。この種のセグメントと通常のジャーナル・セグメントを区別するのに ID を利用することができます。この設計では、いったんデータが SDEP セグメントに変換されてしまうと更新はできません。

DEDB エリアの設計の指針

DEDB を設計するときは、DEDB エリアに関する考慮事項と指針を認識しておいてください。

DEDB エリアの設計に関する以下の考慮事項は、DEDB がエリアに分割されている理由を理解するうえでも役立ちます。

- アプリケーション・プログラムにとって意味をなすような方法で DEDB をエリアに分割する必要があります。

  例: ある情報サービス会社では、1 組のアプリケーションをその客先が有効に利用できるものとします。この設計では、共通の 1 つのデータベースがこの 1 組のアプリケーションのすべてのユーザーによって使用されることが要求されます。エリアの概念がこのような設計に適合するのは、データ要求がこのユーザーのエリアのみに向けられるように、ランダム化ルーチンとレコード・キーを設定することができるからです。さらに、操作面では、ユーザーに特定の時間帯を与えることができます。こうすれば、現在同じ DEDB を使用している他のサービス業務を中断することなく、このユーザーのエリアを動的に割り振ったり動的にその割り振り解除をしたりすることができます。

  複数の時間帯に及ぶ営業地区を持つ国内の会社または国際企業の場合にも、パーティション・データベースの概念を利用することができます。すべての地区が常時オンラインの状態になければならないというわけではないので、時間帯別にエリアを設けてこれらのエリアにデータを分散させることができます。

特定のレコード (特定のアカウント、特定の客先など) に対して、新しいデータベースを使用せずに優先処理を実施することができます。それには、例えば、一部のレコードに関してより多くの順次従属セグメントをオンラインに保持するだけで済みます。これらのレコードを 1 つのエリアにまとめておくことにより、より大きな順次従属セグメント部分を定義して、保存期間を適宜管理することができます。

- DEDB を使用することにより、永続入出力エラーと重大な障害の影響を軽減することができます。 DL/I では、HALDB データベースを除くすべてのデータベース・データ・セットが常時使用可能である必要があります。DEDB を使用している場合、使用可能でないデータは、障害の影響を受けたエリアのみに限定されます。 DEDB ユーティリティーはエリアのレベルで実行するので、障害のあるエリアのリカバリーを行いつつ、このデータベースの他の部分をオンライン処理にアクセス可能にしておくことができます。現在割り振られているログ・ボリュームを /DBR AREA コマンドを実行して解放し、リカバリー操作で使用する必要があります。トラック・リカバリーもサポートされます。リカバリーされたエリアは、次に動的に再び作動環境に割り振ることができます。

  アプリケーション・プログラムに対するデータの可用性をより高めるために、DEDB エリア・データ・セットの複数コピーを作成してください。
- エリアごとにスペース管理パラメーターを変えることができます。これに含まれるのは、CI サイズ、UOW サイズ、ルート・アドレス可能部、オーバーフロー部、および順次従属部です。また、エリアごとに装置タイプが異なっていてもかまいません。
- 1 つのエリアを複数のボリュームの上に定義し、1 つのボリュームを順次従属部の専用とすることが可能です。こうすれば、順次従属セグメントが順次従属部の末尾に連続して追加されていくので、シーク・タイムが節約できます。節約時間は、順次従属部の現在のサイズと、順次従属セグメントのために使用されているブロック化因数とによって決まります。 1 つのエリアが複数のボリュームにまたがっている場合には、これらのボリュームは同じタイプのものでなければなりません。
- DEDB の独立オーバーフロー部だけが拡張可能です。 DEDB を設計するときには、すべての部分に十分なスペースを与えなければなりません。

  /DISPLAY コマンドと POS 呼び出しは、補助記憶スペースの使用状況をモニターするのに役立ちます。ルート・アドレス可能部と独立オーバーフロー部の中の未使用スペースは、再編成を行いレクラメーション処理することができます。注意することは、オーバーフロー域では ISRT 呼び出しによってスペースが自動的に再利用されないことです。呼び出し時に再利用するためには、スペースの量が 1 つの完全な CI に達していなければならず、その場合にはこの CI が ISRT スペース管理アルゴリズムで使われることになります。データベースの中に多少使用可能なスペースがあっても、部分的にスペース不足の状態が起きることがあります。
- DEDB の追加または削除には、DBDGEN と ACBGEN が必要です。既存エリアの真中でエリアの追加または削除を行う場合には、データベースの再ロードが必要です。エリアを末尾以外に追加すると、エリアに割り当てられたエリア・シーケンス番号が変わります。この後に書き出されるログ・レコードにはこの番号が反映され、これがリカバリーのために使用されることになります。既存エリアの間にエリアを追加した場合には、それ以前のログ・レコードは無効になります。

したがって、アンロード/再ロードの後でイメージ・コピーをとらなければなりません。 DBD の中の AREA ステートメント順序によって、ランダム化ルーチンに入る際に渡される MRMB 項目の順序が決まるという点に注意してください。あるエリアが処理に必要でなければ、このエリアを取り付ける必要はないので、実際には、エリアを処理から論理的に除外するのに DBDGEN/ACBGEN は必要ありません。

- それぞれのログの保存期間を注意深くモニターすれば、一度に 1 つのエリアのイメージ・コピーを作成することができます。また、高速 DEDB 直接再編成ユーティリティー・ログは変化するので、再編成の実行後にイメージ・コピーを作成する必要はありません。
- エリアの概念のもとでは、DEDB 全体にわたるランダム化ではなく、エリア・レベルのランダム化が可能です。これは、ランダム化ルーチンを特定のエリアに向けるために、キーに何らかの情報を加えておかなければならないことを意味しています。

関連タスク:

528 ページの『エリア・データ・セットの複数コピー』

887 ページの『DEDB 独立オーバーフロー・オンラインの拡張』

CI のサイズ決定

CI サイズの選定はいくつかの要因に左右されます。

要因には、次のものがあります。

- CI サイズは、512、1 KB、2 KB、4 KB、さらに 4 KB の増分で 28 KB までサポートされます。
- RAP は CI 当たり 1 つしかありません。 平均レコード長を考慮しなければなりません。ルート・アドレス可能部の基本セクションにおいては、1 つの CI を共用することができるのは、その RAP にランダム化された複数のルート・セグメントとこれらの DDEP セグメントだけです。
- 装置タイプに応じたトラック使用率。
- SDEP セグメント書き込み。 CI が大きければ、同じ量の SDEP セグメントを書き込むのに入出力の回数が少なくて済みます。
- 最大セグメント・サイズは 28K バイトの CI サイズを使用している場合、28,552 バイトです。

UOW のサイズ決定

UOW は、ルート・アドレス可能部と独立オーバーフロー部のサイズを指定する際のスペース割り振りの単位です。

以下の 3 つの要因が UOW のサイズに影響を及ぼします。

1. 高速 DEDB 直接再編成ユーティリティー (DBFUHDR0) は UOW 単位で実行します。したがって、ある UOW が再編成されている間、その CI とこの中のデータは一切他の処理には使えません。

   UOW が大きければリソース競合が引き起こされることがあり、そのため、この再編成ユーティリティーをオンライン時間帯に実行すると応答時間が長くなり

ます。大きな UOW の副次作用の 1 つは「再編成 UOW」のために DASD 上に予約されるスペースですが、これは、今後は使用されることはなく、互換性の目的で保持されているだけです。

UOW が小さすぎると、再編成中に再編成ユーティリティーが有用な作業を毎回ごくわずかだけ行い、ある UOW から次の UOW へと切り替わるため、オーバーヘッドは多少増加することがあります。しかし、再編成時間が重大な問題でない場合には、これはそれほど重要ではありません。

2. 処理オプション P の使用。この要因は、BMP 領域を使用する順次処理に関係します。アプリケーション・プログラムが「GC」状況コードを利用するようコーディングされている場合には、計画した同期インターバルに十分収まるように頻繁にこの状況コードを戻さなければなりません。

   ここで、ルート・セグメント CI をことごとく変更しようとしているものとし、リソース制御の理由から、それぞれの同期インターバルで、データの順次処理が行われる CI の数は 20 以下であるものとします。 UOW のサイズを 20 個の CI を超える大きさに設定すべきではありません。それを超える大きさにすると、アプリケーション・プログラムが同期点を始動させ、リソースを解放し、しかもデータベース内の位置を失わない時点に間に合うようには、予測される「GC」状況コードは戻されません。

   UOW が小さすぎる場合、例えば最低 2 つの CI の場合には、UOW が交差するたびに、あまりに多くの「データベース呼び出し不成功」状態が引き起こされる可能性があります。「GC」状況コードのときは、セグメントは何も返されず、オプションの SYNC 呼び出しまたは CHKP 呼び出しの後でデータベース呼び出しを再び出さなければなりません。

3. UOW が大きい方が、小さい場合よりも従属オーバーフロー・セグメント (DASD スペース) の使用効率がよくなります。

関連概念:

526 ページの『処理オプション P (PROCOPT=P)』

『SDEP CI の事前割り振りおよび報告書作成』

SDEP CI の事前割り振りおよび報告書作成

データ共用のため、SDEP CI を一度に 1 つずつ割り振ることはできません。また、それぞれのデータ共用システムが独自の現行 CI を必要とします。したがって、割り振り呼び出し時に 1 組の SDEP CI が各 IMS に事前割り振りされます。

IMS が取得する CI の数はシステムの挿入率の機能です。エリアのオープン処理ではなく、挿入プロセスが現行 CI を取得します。

挿入プロセスが現行 CI を取得するため、スペース使用と報告書作成は複雑です。事前割り振りの試行が要求された数の CI を取得できない場合、ISRT 呼び出しまたは同期点呼び出しは、その呼び出し用のスペースが十分にある場合でも、状況 FS を受け取ります。 FS 処理はこのエリアにフルのマークを付けます。後続のより小さい挿入も失敗します。

使用可能な SDEP CI がエリア内に少数しかない場合、SDEP 挿入用に実際に使用できる数はシステムの挿入率に応じて異なります。また、コマンド /DIS AREA

は、事前割り振りに使用可能なフリーの SDEP CI の数およびこのコマンドを出した IMS に事前割り振りされている未使用の CI (ある場合) の数を計算します。エリア・クローズ処理は IMS に事前割り振りされている CI を廃棄し、未使用の CI は SDEP 削除ユーティリティーが実行されるまで失われます。したがって、エリア・クローズ処理後に /DIS AREA コマンドによって報告される未使用の CI の数は、事前割り振りされた CI がもはや使用可能でないため、より小さくなります。

さらに、DEDB 順次従属の削除ユーティリティー (DBFUMDL0) または DEDB 順次従属のスキャン・ユーティリティー (DBFUMSC0) の実行中に、事前割り振りされた CI の廃棄が要求されると、それらの CI は使用不可になります。DBFUMDL0 および DBFUMSC0 ユーティリティーで事前割り振りされた CI の廃棄が要求されるのは、DFSVSMxx PROCLIB メンバーの SDEPQCI パラメーターまたはユーティリティー SYSIN データ・セットの QUITCI 制御ステートメントで、QUITCI オプションが指定された場合です。SDEPQCI パラメーターは、DBFUMDL0 ユーティリティー、DBFUMSC0 ユーティリティー、またはその両方での QUITCI のデフォルトの動作を設定します。ユーティリティーに SDEPQCI が指定されている場合は、そのユーティリティーを実行する際に QUITCI 制御ステートメントを組み込む必要はありません。

関連概念:

524 ページの『UOW のサイズ決定』

処理オプション P (PROCOPT=P)

PROCOPT=P オプションは、PCB 生成時に PCB ステートメント、またはルート・セグメントのための SENSEG ステートメントに指定されます。

このオプションは、領域タイプが BMP の場合に限り効力を持ちます。このオプションを指定すると、以下のような利点が得られます。

ある DEDB セグメントを検索したりあるいは挿入したりする試みがなされ、このため UOW 境界を交差する場合に、PCB に「GC」状況コードが設定されますが、セグメントが返されたり挿入されたりすることはありません。このようなことが起きる呼び出しは、G(H)U、G(H)N、POS、および ISRT だけです。

大抵のアプリケーションにとっては、UOW 境界を交差することに特別に重要な意味はありませんが、戻される「GC」状況コードは、これが同期点処理を呼び出す好機となることも示しています。というのは、UOW 境界は CI 境界でもあるからです。後で順次処理に関して説明するように、CI 境界は同期点を要求するのに好都合な場所です。

同期点は、SYNC 呼び出しまたは CHKP 呼び出しによって呼び出されるが、通常これによって、現在アクセスされるすべてのデータベースの上の位置が失われます。したがって、アプリケーション・プログラムは、まず最初に位置を再設定することによって処理を再開しなければなりません。特に、修飾されていない G(H)N 処理の場合には、この状況を解決するのは必ずしも容易ではありません。

この処理オプションのもう 1 つの利点は、「GC」状況コードの後で SYNC 呼び出しまたは CHKP 呼び出しが出されても、データベースの位置が保持されることです。データベース位置は、「GC」状況コードの後で出された修飾されていない G(H)N 呼び出しによって、次の UOW の最初のルート・セグメントが戻されるよ

うにするものです。「GC」状況コードが戻されるときには、データは提示も、挿入もされません。したがって、アプリケーション・プログラムは、必要に応じて、同期点を要求し、「GC」状況コードが戻される原因となったデータベース呼び出しを再び出し、先へ進みます。アプリケーション・プログラムは「GC」状況コードを無視することができます。次のデータベース呼び出しは通常どおりに機能します。

データベースのリカバリーと累積処理は、同期点間で書かれるすべてのログ・レコードをバッファーに入れなければなりません。同期点は、使用可能ストレージが使い果たされるのを回避するために、頻繁にとる必要があります。そうでないと、データベースがリカバリーできないことがあります。

関連概念:

524 ページの『UOW のサイズ決定』

555 ページの『DBCTL 環境における NBA/FPB 限度と同期点』

548 ページの『NBA 限度と同期点』

DEDB ランダム化ルーチンの設計

DEDB にルート・セグメントを置くため、および DEDB からルート・セグメントを検索するためには、DEDB ランダム化モジュールが必要です。

IMS システムでは、1 つ以上のこの種のモジュールを使用することができます。DEDB にはそれぞれランダム化モジュールを 1 つだけ関連付けることができます。

ランダム化モジュールの目的は、HDAM 処理の場合と同じです。ルート・セグメント検索引数キー・フィールド値がアプリケーション・プログラムによって与えられ、これが相対ルート・アンカー・ポイント番号に変換されます。入り口と出口のインターフェースが異なるので、DEDB と HDAM のランダム化ルーチンは、オブジェクト・コードの段階では、互換性はありません。 DEDB 全体にわたってランダム化するのであれば、HDAM のランダム化論理の主要部分を変更する必要はありません。

DEDB と HDAM のランダム化ルーチンの他の相違点をいくつか以下に挙げておきます。

- ISRT アルゴリズムは、データベース・レコード全体をルート・セグメントの近くに置こうと試みます (ただし、SDEP セグメントを除きます)。ルート・アドレス可能部に挿入されるレコード部分のサイズを限定するための BYTES パラメーターはありません。
- DEDB では、各ルート・アドレス可能 CI で RAP を 1 つずつしか定義することができません。
- ランダム化によってセグメントを割り当てられなかった CI は、空 (から) のままです。

| 標準ランダマイザーは、データベース・レコードをデータベース全体にランダムに
| 分散します。2 ステージ・ランダマイザーは、RMNAME パラメーターで定義され
| ます。2 ステージ・ランダマイザーは、そのデータベース・レコードのターゲット
| として、特定のエリアおよびそのエリア内の RAP を選択します。オンライン変更
| を使用して特定のエリアの UOW パラメーターまたは RAP パラメーターを変更
| し、DEDB 内の他のエリアに影響を及ぼさないためには、2 ステージ・ランダマイ

ザーを使用してください。DEDB 変更ユーティリティーを使用して DEDB データベースまたはエリアを変更するには、2 ステージのランダム化ルーチンが必要です。

推奨事項: 2 ステージのランダム化ルーチンを使用します。

エリアの概念により、アプリケーションによっては、HDAM 処理におけるような DEDB 全体にわたるランダム化ではなく、特定の 1 つのエリアでランダム化を行うことがあります。したがってこの種のランダム化モジュールの予期される出力は、エリア内の相対ルート・アンカー・ポイント番号、および選定されたエリアを表す制御ブロック (DMAC) のアドレスで構成されています。

同じ RAP にランダム化されるキーは、キーの昇順にチェーニングされます。

DEDB 論理は並列的に実行されるため、DEDB ランダム化ルーチンは再入可能でなければなりません。これらのランダム化ルーチンは共通ストレージ (CSA) から作動します。これらのルーチンは、LOAD、DELETE、GETMAIN、FREEMAIN などのオペレーティング・システム・サービスを使用する場合、「*IMS V13* 出口ルーチン」に記述されている規則と同じ規則に従わなければなりません。

DEDB ランダム化ルーチンは ECSA ストレージに保管され、通常は、タイプ 1 コマンド `/DBRECOVERY DB` またはタイプ 2 コマンド `UPDATE DB STOP(ACCESS)` の実行によって DEDB へのアクセスを停止するときにストレージからアンロードされます。タイプ 2 コマンド `UPDATE DB STOP(ACCESS)` の NORAND パラメーターを使用すると、DEDB ランダマイザーをアンロードすることなく DEDB へのアクセスを停止できます。

関連資料:

高速処理データベース・ランダム化ルーチン (DBFHDC40 / DBFHDC20 DBFHDC44 / DBFHDC24 DBFHDC2S) のサンプル (出口ルーチン)

エリア・データ・セットの複数コピー

エリア内のデータは、エリア・データ・セット (ADS) と呼ばれる VSAM データ・セットの中にあります。ご使用のシステムでは、各 ADS のコピーを 7 個まで作成できるので、アプリケーション・プログラムに対するデータの可用性はさらに高くなります。

エリア・データ・セットの複数コピーを使用している場合、その ADS は 多重エリア・データ・セット (MADS) と呼ばれます。

MADS の各コピーには、完全に同一のユーザー・データが入っています。アプリケーション処理の間、高速機能は、複数のコピーの中のデータを同一に保持することによって、データの保全性を維持します。アプリケーション・プログラムがこのエリア内でデータを更新する場合には、高速機能では、MADS の各コピーの該当データを更新します。

アプリケーション・プログラムがデータをある 1 つのエリアから読み取った場合には、IMS は常に、RECON リストに示されている最初の ADS から読み取ろうとします。最初の ADS が利用不能になっているか、または長時間使用中の状態である

場合、IMS は、使用可能な ADS が見つかるまで、リスト内の後続の各 ADS から読み取ろうとします。すべての ADS が長時間使用中の場合、IMS は、リストの中の最初の ADS を使用します。

MADS のすべてのコピーは、同一の定義を持っていなければなりませんが、別の装置と別の装置タイプに入っていてかまいません。

MADS コピーが別々の装置にある場合、最高の読み取りパフォーマンスを得るためには、RECON データ・セットに登録されている最初の ADS を、ご使用の DASD のうちの最高速の DASD に入れてください。ADS のそれ以降のコピーは、遅い方の DASD に入れても、読み取りパフォーマンス全体には影響しません。

MADS を使用すると、DASD のマイグレーション時 (例えば、3380 装置から 3390 装置に移る場合) にも役立ちます。

エリア・データ・セットのコピーを作成するには、DBRC コマンド INIT.ADS を実行します。 AREA(*name*) パラメーターには、RECON データ・セットに記録されているとおりに、元の ADS の名前を指定してください。追加コピーごとに INIT.ADS コマンドを実行してください。 MADS の作成について詳しくは、「*IMS V13* コマンド第 *3* 巻*: IMS* コンポーネントおよび *z/OS* コマンド」の中の INIT.ADS コマンドを参照してください。

DEDB の標準オープン処理の間に ADS がオープンに失敗すると、ADS のどのコピーの割り振りもできず、このエリアが停止します。しかし、緊急時再始動している間にオープンの障害が発生した場合、失敗した ADS だけが割り振り解除されて停止します。 ADS の他のコピーは、使用可能な状態になっています。

関連概念:

522 ページの『DEDB エリアの設計の指針』

レコードの非活動化

アプリケーション・プログラムが DEDB を更新している間にエラーが起きた場合には、データベースあるいはエリアを停止する必要はありません。

IMS は継続してアプリケーション・プログラムが該当エリアにアクセスすることができるようにします。IMS は、アプリケーション・プログラムがエラーの制御インターバルにアクセスできないようにするだけです。 ADS の複数コピーがある場合には、データの 1 つのコピーは、常時使用可能になっています。 (ADS の 7 つのコピーで、同じ制御インターバルがエラーになることはありません。) エラーの件数が 10 に達すると、IMS は自動的にレコードを非活動化します。

レコードの非活動化によって、データベースの障害とデータのエラーの影響が以下の方法で最小限に抑えられます。

- エリア・データ・セットの複数コピーを使用している場合には、アプリケーション・プログラムが該当エリアを更新している間にエラーが起きても、エラーをすぐに訂正する必要はありません。このとき他のアプリケーション・プログラムは、該当エリアの他の使用可能なコピーを使用してそのエリア内のデータに引き続きアクセスすることができます。

- エリアのコピーにエラーがある場合には、DEDB データ・セット作成ユーティリティーを使用して ADS の既存コピーで新しいコピーを作成することができます。この後で、エラーのあるコピーを破棄することができます。

物理最終子ポインター

PCL ポインターは、セグメント・タイプの最終物理子に物理親から直接アクセスすることができるようにします。 INSERT 規則の LAST を使用すると、長くなる可能性のある物理子ポインター・チェーンに従って進まなくてもすみます。

サブセット・ポインター

長いセグメント・チェーンの最後の部分にアクセスする必要がある場合には、サブセット・ポインターの助けを借りて無駄な GET 呼び出しを回避することができます。

これらのポインターは、同じ親のもとで発生したセグメント・オカレンスのチェーンを複数のグループ、すなわちサブセットに分けます。任意のセグメント・タイプに対して最高 8 つのサブセット・ポインターを定義することができ、そのチェーンを最高 9 つのサブセットに分けることができます。 各サブセット・ポインターは、新しいサブセットの開始位置を指します。

制約事項: サブセット・ポインターが入っている DEDB をアンロードし、再ロードするときには、IMS は自動的にはサブセット・ポインターの位置の保存を行いません。DEDB をアンロードするときには、情報を永続的な場所に保管し、サブセット・ポインターの位置を記録しておかなければなりません。 (例えば、セグメントごとにフィールドを付けて、どのサブセット・ポインターが、該当セグメントを指しているかを示すことができます。) あるいは、兄弟チェーンの中のセグメントが固有に判別できる場合には、サブセット・ポインターが指しているセグメントを判別して、そのセグメントに再ロード用の一時的な印を付け加えます。 DEDB を再ロードするときは、セグメントに対するサブセット・ポインターを以前の設定に戻して、サブセット・ポインターを再定義する必要があります。

関連概念:


サブセット・ポインター・コマンド・コードを使用した高速機能 DEDB の処理 (アプリケーション・プログラミング)

主記憶データベース (MSDB) の設計

この節では、MSDB を設計するにあたって行う必要のあるいくつかの選択について述べ、これらの選択に役立つ指針を示します。

以下のリストは、MSDB データベースを設計する際の考慮すべき問題点を列挙したものです。

- データベースの仮想記憶域の必要量をどのように計算するか。
- 高速機能バッファー・プール用の仮想記憶域の必要量をどのように計算するか。
- 入出力域のためのストレージの必要量はどのくらいか。
- MSDB パフォーマンスと DEDB のパフォーマンスの向上のために、FLD 呼び出しを使用する必要があるか、または他の DL/I 呼び出しを使用する必要があるか。

- MSDB と DL/I データベースとの間のリソース割り振りの相違をどうしたらパフォーマンス向上の手掛かりとすることができるか。
- 混合モード環境においてリソース競合を最小限にするための設計の要件は何か。
- 仮想記憶域にロードされる MSDB セグメントの数をどのようにして制御するか。
- MSDB のための補助ストレージの必要量はどのくらいか。
- どのようにして MSDB のチェックポイントをとるか。

MSDB のための仮想記憶域必要量の計算

MSDB のためのストレージの必要量は、次の数式を使用して計算できます。

MSDB のためのストレージの必要量を計算する数式は次のとおりです。

```
(L + 4)S + C + 14F + X
```

ここで、

S = IMS.PROCLIB のメンバー DBFMSDBx によって指定されている MSDB 内のセグメント数
L = DBD メンバーで指定されたとおりのセグメント長

C = 80・・・ 端末関連キーなしの非関連 MSDB の場合、または
94・・・ 他のタイプの MSDB の場合

F = DBD メンバーで定義されているフィールド数

X = 2・・・ C + 14F が 4 の倍数でない場合、または
0・・・ C + 14F が 4 の倍数である場合

MSDB は、z/OS 拡張共通ストレージ域 (ECSA) に存在します。

MSDB バッファーに関する考慮事項

MSDB データベースのバッファー必要量を計算する際には、いくつかの考慮事項があります。

実行中には以下の考慮事項が適用されます。

- 高速機能バッファーの必要量は、MSDB への呼び出しの種類に応じて変わります。
- GHx/REPL の呼び出しシーケンスでは、同期点に達するまで 1 つのセグメント全体が高速機能バッファーの中に保持されます。一連のセグメントのサイズの合計が通常のバッファー割り振り (NBA) を超える場合には、通常どおりにオーバーフロー・バッファー (OBA) を使用するのではなく、NBA パラメーターを調整する必要があります。同期点と同期点との間で使用されるセグメントの総数を収容できるようにする必要があります。
- FLD 呼び出しを使用する場合には、VERIFY と CHANGE の論理が高速機能バッファーに存在します。

関連概念:

541 ページの『DEDB または MSDB バッファー・プールの設計』

アプリケーション入出力域のためのストレージの計算

GHx/REPL 呼び出しでは、処理されるセグメントのうち最大のものを収容できるだけの入出力域が必要です。 FLD 呼び出しでは、フィールド検索引数 (FSA) の必要量全体を収容するためのストレージが必要です。

パフォーマンスの鍵となるリソース割り振りについての理解

MSDB のリソース割り振りの仕組みは、DL/I のリソース割り振りの仕組みとは異なります。

パフォーマンス上の理由から、MSDB のリソース割り振りを理解することは重要です。

MSDB レコードは、複数のユーザーで共用することも、1 人のユーザーが排他的に所有することもできます。同一のレコードが同時に両方 (共用と排他) の状況をとることができます。

同期点処理の間に、MSDB の更新が適用されます。この場合、同期点処理の継続中、常にリソースが排他モードで所有されます。

MSDB レコードの異なるエンキュー・レベル (レコードがエンキューされるときの) および期間を以下の表に要約します。

表 *70. MSDB* レコードのエンキューのレベル

エンキュー・レベル	場合	期間
READ	更新意図のない GH	呼び出し時から同期点まで (フェーズ 1)[1]
	VERIFY/get 呼び出し	呼び出し処理
HOLD	更新意図がある GH	呼び出し時から同期点まで (フェーズ 1)[1]
	同期点で、再び VERIFY を適用する	同期点処理のフェーズ 1、その後解放
UPDATE[2]	同期点で、CHANGE、REPL、DLET、または ISRT 呼び出しの結果を適用する	同期点処理、そのあと解放

注:

1. このリソースを対象とする FLD/VERIFY 呼び出しがなかった場合、またはこのリソースがこれから更新されるのでない場合、このリソースは解放されます。これ以外の場合、FLD/VERIFY 論理だけを再び適用するのであれば、この MSDB レコードは HOLD レベルでエンキューされます。同じレコードが更新操作に関与している場合には、上の表で示すように、このレコードは UPDATE レベルでエンキューされます。
2. DLET/REPL 呼び出しのときには、エンキュー・レベルを設定したのは以前の GH 呼び出しですからエンキュー・アクティビティーは何も行われません。

以下の表は、MSDB レコードの状況が、関与する各プログラムのエンキュー・レベルに依存していることを示しています。したがって、ある 1 つの MSDB レコード

が、同時に共用と排他という状況でエンキューされることはあり得ます。例えば、この種のレコードをプログラム A (更新のための GH 呼び出し) とプログラム B (GU 呼び出し) の間で共用することはできますが、レコードの更新のために同期点に入っている第 3 のプログラム、プログラム C によってこのレコードを同時に共用することはできません。

表 *71. MSDB* レコードの状況の例: 共用 *(S)* または排他的所有 *(E)* :

プログラム **B** における	プログラム **A** におけるエンキュー・レベル		
エンキュー・レベル	**READ**	**HOLD**	**UPDATE**
READ	共用	共用	排他的
HOLD	共用	排他的	排他的
UPDATE	排他的	排他的	排他的

FLD/CHANGE 呼び出しはどの割り振りにも参加しません。したがって、同期点処理の間に、あるデータベース・レコードが更新されていても、このレコードに対する FLD/CHANGE 呼び出しを実行することができます。

同一の FLD 呼び出しで FLD/CHANGE 呼び出しと FLD/VERIFY 呼び出しが併存している場合、最初の FLD/VERIFY 呼び出しが出てきたとき、この FLD 呼び出しの残りに対してはエンキュー・レベルが READ に設定されます。

リソース競合の最小化のための設計

MSDB を使用する 1 つの理由は、そのデータへの高速アクセスと処理に対する高可用性です。

高可用性を維持するためには、トランザクション率が高い環境において起こりがちなリソース競合を避けるよう設計しなければなりません。

パフォーマンスに関する考慮事項を以下に列挙します。一部の考慮事項は MSDB だけに当てはまるものではありませんが、操作環境の理解を深めるためにこれらも列挙してあります。

- 高速機能トランザクションが DL/I データベースにアクセスして代替 PCB を使用することは、最小限に抑えるべきです。代替 PCB の使用を最小限に抑える理由は、FP トランザクションが IMS トランザクションとリソースを競合しなければならないためです (一部の IMS トランザクションは実行時間が長いためです)。また、共通同期点処理が呼び出され、それが IMS 制御領域で完全に逐次化されます。
- 高速機能トランザクションと DL/I トランザクションとの間で MSDB を共用する場合には、リソース競合を避けるために頻繁にコミット処理を行うようにして、長時間のスキャンを避けるようにします。
- 読み取りまたは更新のための GH によって、同じ MSDB リソースを更新しようとしている同期点処理が遅くなります。したがって、参照されるセグメントが当該トランザクションの完了まで変更されることがないと想定している場合にだけ GH 論理を使用すべきです。このリソースが更新されている場合の解放は、同期点が完了したときに行われます。そうでなければ、解放は同期点に入ったときです。

- 以下の考慮事項は、デッドロックの防止を取り扱ったものです。トランザクションが複数の MSDB リソースを取得しようとすると (GH 呼び出し)、デッドロックが起きることがあります。

  すでに割り振られている MSDB リソースに対する要求があり、関係するレベルが HOLD または UPDATE であれば、そのたびに潜在的なデッドロック状況を検出するために制御が IMS に渡されます。その結果、パス長が増加して応答時間が増えます。デッドロックが起きれば、後者は重大な問題となることもあり、したがって、トランザクションの疑似異常終了が必要になります。

  リソース競合によるデッドロックの可能性を減らすために、同期点処理は MSDB リソースを、定義された順に (UPDATE レベルで) エンキューします。この順序は、セグメント・アドレスの昇順です。 MSDB セグメントは、MSDB 名の昇順、最初にページ固定 MSDB、次いでページ可能 MSDB の範囲内でのキーの昇順に取得されます。

  アプリケーション・プログラマーは、同じ順序で MSDB リソースを取得する (GH 呼び出し) ことによって、呼び出し時に潜在的なデッドロック状況を排除することができます。
- すでに説明したリソース割り振りの仕組みから考えて、できる限り、GH/REPL 論理の代わりに FLD 論理を使用すべきであることが理解できます。
  - FLD/VERIFY の呼び出しは結果的に READ レベルでエンキューすることになり、他のレベルが関係していなければ、制御は IMS には渡されません。この結果、パス長が短くなります。
  - FLD/CHANGE の呼び出しは、VERIFY 論理との関連で出されない場合には、結果として高速機能あるいは IMS のいずれの中でもエンキューされることはありません。
  - FLD 論理の方が、プログラム要求ハンドラーを介するパス長が短くなっていますが、これは GH/REPL 論理では処理すべき呼び出しが 2 つ必要なのに、FLD 論理では 1 つしかないからです。
  - FLD/CHANGE 呼び出しは、同じこのリソースが同期点処理で更新されていても、リソースを待つことは決して ありません。
  - FLD/VERIFY 呼び出しは、同じリソースが更新されている同期点処理の間だけ待っています。
  - FLD 論理は、同期点処理の間だけリソースが排他モードに保持されます。

要約すると、FLD 論理を用いたプログラミングは、トランザクション率の向上と応答時間の短縮に貢献することができます。

以下の例は、MSDB レコードがどのように排他モードに保持されるかを示したものです。

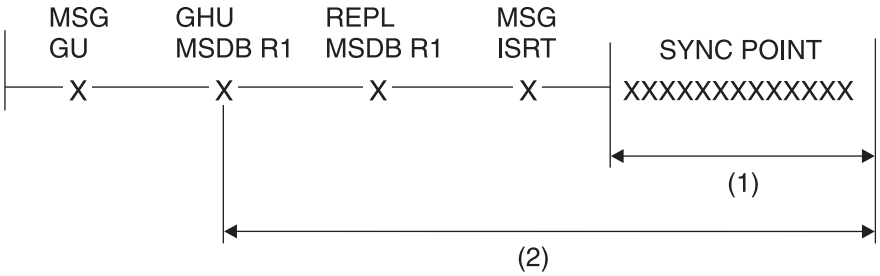


図 *234.* 排他モードに保持される *MSDB* レコードの最初の例

以下は、前の図に関する注釈です。

1. MSDB レコード R1 は、以下のものに対して排他モードに保持されます。
   - CHANGE 呼び出しを除くいずれかの MSDB 呼び出し
   - 同じレコードを更新しようと意図するその他の同期点処理
2. MSDB レコード R1 は、以下のものに対して排他モードに保持されます。
   - 更新のためのその他の GH
   - 同じレコードを更新しようと意図するその他の同期点処理

図 *235.* 排他モードに保持される *MSDB* レコードの *2* 番目の例

以下は、前の図に関する注釈です。

1. MSDB レコード R1 は、以下のものに対して排他モードに保持されます。
   - CHANGE 呼び出しを除くいずれかの MSDB 呼び出し
   - 同じレコードを更新しようと意図するその他の同期点処理
2. MSDB レコードは、FLD 呼び出しの間、同じリソースを更新しようとするその他の同期点処理に対して排他モードに保持されます。

ロードとページ固定を行う MSDB の選択

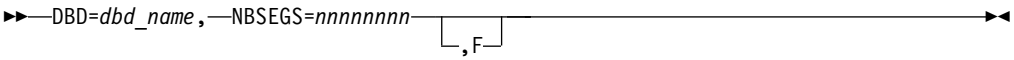
どの MSDB をロードしページ固定するかを決定することは、必要なアプリケーション・パフォーマンスと使用可能な主記憶域の量とのトレードオフにかかわる問題です。これを決定する際には、高速機能アプリケーションの要件全体を配慮する必要があります。

IMS システムの初期設定の際に追加情報を加えないと、MSDB ロードとページ固定は行われません。この情報は、IMS.PROCLIB のメンバー DBFMSDBx の中で指定します。このメンバーは、制御領域始動プロシージャー IMS を実行することにより呼び出されます。接尾部の 'x' は、プロシージャー IMS の中の EXEC ステートメントの MSDB キーワードで与えられるパラメーターと一致します。

MSDB をロードしこれをページ固定するための制御情報は、メンバー DBFMSDBx で 80 文字のレコード・フォーマットをしています。ユーザーがこの情報を与えるか、または MSDB 保守ユーティリティーの出力によってこの情報を与えることができます。 /NRE コマンドが MSDBLOAD を要求するとき、ロードすべきデータベースの定義は、DBFMSDBx プロシージャーの中にあります。

DBFMSDBx の中のこの定義は、DD ステートメント MSDBINIT によって識別される順次データ・セット上に現存する複数の MSDB の 1 つのサブセットを表すことができます。 IMS にロードさせたい各 MSDB を明示的に指定してください。各 MSDB を明示的に指定しないと、IMS は異常終了します。

DBFMSDBx の形式は次のとおりです。



dbd_name
DBDGEN 時に指定された DBD 名。

nnnnnnnn
この MSDB のデータベース・セグメント数の予測値として指定する数。この数は、再始動時にロードされる MSDB セグメントの数に等しいかそれより大きくなければなりません。

NBRSEGS パラメーターは、またデータを最初にロードする必要のない端末関連の動的 MSDB のスペースを予約するために使用されます。

F この MSDB のオプション・ページ固定標識。

MSDB がユーザーの高速機能アプリケーションにとって、これがないと IMS が実行できないほど重要である場合には、DBFMSDBx メンバーの先頭に、最初のカード・イメージを入れてください。各カード・イメージには、空白なしで「MSDBABND=*n*」という文字をタイプし、文字はすべてカード・イメージの 1 桁目から 72 桁目までの間になければなりません。カード・イメージには 4 種類があり、それぞれには以下の文字セットのいずれかが入ります。

MSDBABND=Y
このカード・イメージによって、システム初期設定時の MSDB ロード中のエラーが発生した場合に、IMS 制御領域は異常終了します。エラーには次のようなものがあります。

- MSDBINIT データ・セットでのオープン障害
- MSDB 定義の中のエラー
- MSDBINIT データ・セットでの入出力エラー

MSDBABND=C
このカード・イメージによって IMS の始動後の最初のチェックポイントで MSDBCP1 または MSDBCP2 データ・セットへの MSDB の書き込み中にエラーが発生した場合、IMS 制御領域は異常終了します。

MSDBABND=I
このカード・イメージによって、MSDBINIT データ・セットから MSDB の初期ロード中にエラーが発生した場合、IMS 制御領域は異常終了して、1 つ以上

の MSDB が使用不可になります。このエラーには、MSDBINIT データ・セットでのデータ・エラー (例えば、「定義済みの MSDB 用 MSDBINIT データ・セットの中にセグメントがない」など)、および 『MSDBABND=Y』 で説明されているエラーが含まれています。

MSDBABND=A
このカード・イメージによって、『MSDBABND=C』で述べた MSDBCPn データ・セットへの MSDB の書き込み中、または『MSDBABND=I』で述べた MSDBINIT データ・セットから MSDB の初期ロード中にエラーが発生した場合、IMS 制御領域は異常終了します。

MSDBABND=B
このカード・イメージによって、『MSDBABND=C』で述べた MSDBCPn データ・セットへの MSDB の書き込み中、または『MSDBABND=Y』で述べたシステムの初期設定時の MSDB のロード中にエラーが発生した場合、IMS 制御領域は異常終了します。

MSDB のための補助ストレージの必要量

システム・チェックポイントとシャットダウン・チェックポイントで MSDB をダンプするとき、MSDB のイメージ・コピーを保持するために DASD スペースが必要です。これに関係するデータ・セットは、MSDBCP1 データ・セットと MSDBCP2 データ・セットです。

MSDBDUMP データ・セットにも同じ計算を適用できます。このデータ・セットには、/DBDUMP DATABASE MSDB の後ろに MSDB のコピーが入っています。

上記のデータ・セットは、2K バイト・ブロック単位で書かれます。最初のエクステントしか使用されないため、スペースの割り振りはシリンダーの境界になければならず、しかも隣接していなければなりません。

スペース割り振りの計算は次のようになります。

```
SPACE=(2048,(R),,CONTIG,ROUND)
```

割り振るレコード (R) の数を計算するには、次の数式から導き出します。

```
(E + P + 2047)/2048
```

ここで、

E = CNT (ECNT) の高速機能拡張に必要な
主記憶域 (バイト数)

P = PROCLIB メンバー DBFMSDBx で定義された、
すべての MSDB に必要な主記憶域

E は、次の式から得られます。

```
E = (20 + 4D)T
```

ここで、

D = 論理端末名をキーとして使用する MSDB の数

T = システムで定義される論理端末名の総数

高速順次処理 (HSSP)

高速機能順次処理 (HSSP) は、DEDB の順次処理を行う高速機能の機能の 1 つです。

HSSP 機能の利点

高速順次処理 (HSSP) 機能には、多数の利点があります。

HSSP 機能には次のような利点があります。

- HSSP は一般的に、応答時間が通常のバッチ処理より速い。
- HSSP は DEDB の順次処理を最適化する。
- HSSP はプログラム実行時間を短縮する。
- HSSP は一般的に、出力が通常のバッチ処理より少ない。
- HSSP は DEDB の更新とイメージ・コピー操作の回数が減少する。
- HSSP イメージ・コピーは、データベースのリカバリーを援助できる。
- HSSP は UOW レベルでロックして、交差した IRLM 通信間の「ボトルネック化」を解消できる。
- HSSP は専用バッファー・プールを使用して NBA/OBA バッファーに対する影響を削減できる。
- HSSP は他のプログラムと並行する、混合モード環境での実行と、IRLM を使用するグローバル共用環境での実行ができる。
- HSSP は更新したデータベースに対するイメージ・コピーのオプションの使用を可能にすることによって、データベースの維持管理を最適化する。

関連概念:

『HSSP を使用する場合の制限と制約事項』

540 ページの『HSSP 処理オプション H (PROCOPT=H)』

関連タスク:

539 ページの『HSSP の使用』

HSSP を使用する場合の制限と制約事項

HSSP は、他のプログラムと並行する混合モードで実行することができるばかりでなく、IRLM によるグローバル共用を使用している環境でも実行することができますが、HSSP を使用するプログラムは、非メッセージ・ドリブン BMP としてしか実行できません。

HSSP のその他の制約事項と制限として次のものがあります。

- 任意の時点で、1 つのエリアではただ 1 つの HSSP 処理のみを活動化できます。`/DIS AREA` コマンドは、1 つのエリアを処理している任意の HSSP ジョブ処理の IMSID を識別します。

- HSSP 処理とオンライン・ユーティリティーが同じエリアで並行して処理することはできません。
- HSSP を使用している場合、非順次方向参照は許可されません。
- HSSP を使用しているプログラムは、その後でコミット処理を行うことによって、「GC」状況コードを正しく処理する必要があります。

イメージ・コピーに関する制約事項と制限には、次のものがあります。

- イメージ・コピーのオプションは、HSSP 処理に対してのみ使用可能です。
- PROCOPT = H の場合のみ、HSSP イメージ・コピーが許可されます。
- イメージ・コピー処理を実行することができるのは、データベースが DBRC に登録されている場合だけです。さらに、イメージ・コピーのデータ・セットは、DBRC の中で初期設定しなければなりません。

PROCOPT=H には以下の制約事項と制限があります。

- PROCOPT=H は DEDB に対してのみ許可されます。
- PROCOPT=H はセグメント・レベルでは許可されず、PCB レベルでのみ許可されます。
- HSSP を使用している場合、逆方向参照は許可されません。 HSSP PCB を使用して、DEDB 内の以前の UOW を参照することはできません。
- 1 つの PSB につき、データベース当たり 1 つの PROCOPT=H PCB しか許されません。
- H を含め最大 4 つの PROCOPT を指定することができます。
- PROCOPT=H は、GH や IH のように、他の高速機能処理オプションと共に使用する必要があります。
- GC 状況コードが戻されると、その PCB に対して他の呼び出しが行われる前に、プログラムがコミット処理を行わせなければなりません。
- PROCOPT=H を活動化するには、ACBGEN を行わなければなりません。
- H は、PROCOPT=O を除く他のすべての PROCOPT と共に指定できます。

関連概念:

538 ページの『HSSP 機能の利点』

540 ページの『HSSP 処理オプション H (PROCOPT=H)』

HSSP の使用

HSSP を使用するには、PSBGEN の実行中に PROCOPT=H を指定する必要があります。

さらに、HSSP を使用しているプログラムが、後からコミット処理を行って「GC」状況コードを正しく処理することを確認する必要があります。

HSSP 機能は、ユーティリティーと同様に、ユーティリティーと同じ要件の対象になり、また、共通バッファーのほかにユーティリティー専用バッファーを使用します。

HSSP には、イメージ・コピーのオプションおよびエリアの範囲を設定する能力が含まれています。これらの機能を利用するには、次に挙げるものが 1 つ以上必要になります。

- SETR ステートメント
- SETO ステートメント
- 従属領域のための DFSCTL データ・セット
- DBRC
- PROCOPT=H

関連概念:

538 ページの『HSSP 機能の利点』

関連資料:

高速順次処理制御ステートメント (システム定義)

HSSP 処理オプション H (PROCOPT=H)

PROCOPT=H は PSBGEN OPTION の 1 つです。これによって、PCB でどの処理を HSSP 処理として扱うか定義できます。

PROCOPT=H を使用すると、この PSB を使用しているアプリケーション・プログラムに HSSP の機能が提供されます。次に、PROCOPT=H を使用した PSBGEN のマクロとキーワードの例を示します。

```
Label      PCB TYPE = DB
          ,DBNAME = name
          ,PROCOPT = AH
```

Label は、PCB マクロのオプション・パラメーターです。このパラメーターは、長さが最高 8 文字まで可能で、関連する SETO または SETR ステートメントのラベルと同じです。 H は、PROCOPT=O を除く他のすべての高速機能 PROCOPT と互換性があり、共に指定できます。また、PROCOPT=H は 1 つ以上の PCB の中で使用することができます。

PROCOPT=H が指定されている PCB は、ECSA ストレージ内の EPCB プールで追加のスペースを使用します。EPCB プールのサイズ変更に役立つように、ACB 保守ユーティリティーの出力には、関連する PSB が EPCB プール内で必要とするストレージの最小量と最大量をリストするメッセージ DFS0943I が表示されます。ただし、PSB が必要とする追加ストレージの正確な大きさは、PSB がスケジュールされるまで予測できません。

関連概念:

538 ページの『HSSP 機能の利点』

538 ページの『HSSP を使用する場合の制限と制約事項』

関連資料:

高速順次処理制御ステートメント (システム定義)

イメージ・コピーのオプション

HSSP でイメージ・コピーのオプションを選択すると、DEDB の更新とその後のイメージ・コピー操作との総経過時間が短縮されます。

ユーザーは、データベース管理者として、HSSP を使用してデータベースのイメージ・コピーを作成するかどうかを決定します。イメージ・コピーを指定する場合には、HSSP が並行イメージ・コピーと同様な非同期コピーを作成します。

イメージ・コピー処理を実行することができるのは、データベースが DBRC に登録されている場合だけです。さらに、イメージ・コピーのデータ・セットは、DBRC の中で初期設定しなければなりません。

HSSP イメージ・コピーは、データベースのリカバリーのためにも使用することができます。しかし、データベース・リカバリー・ユーティリティーは、HSSP イメージ・コピーが提供されていることを知っていなければなりません。

関連概念:

639 ページの『第 26 章 データベースのバックアップおよびリカバリー』

HSSP イメージ・コピー (システム管理)

IMS 障害リカバリー (オペレーションおよびオートメーション)

UOW のロッキング

グローバルに共用される環境では、データは、IMS サブシステム間ばかりでなく、CPC (中央演算処理装置複合システム) 間でも共用されます。

このような環境での 2 つの IRLM 間の通信は、「ボトルネック」になる可能性があり、妨げられることがあります。この問題を緩和するために、HSSP は更新モードの UOW レベルでロッキングして、ロッキング・オーバーヘッドを減らします。非 HSSP または DEDB オンライン処理は共用モードの UOW レベルでロックします。これ以外の場合には、CI レベルで DEDB オンライン処理はロッキングします。 UOW のロッキングについては、「*IMS V13* システム管理」を参照してください。

専用バッファー・プール

HSSP は、1 つの UOW の中のエリアごとに、最大で CI の数の 3 倍までの個数のバッファーを動的に割り振ります。各バッファーのサイズは CI 1 個分です。

HSSP は、専用バッファーを RAP CI の読み取りに、また共通バッファーを IOVF CI の読み取りに使用します。非 HSSP ジョブの場合と同様に、HSSP ジョブを実行中に NBA が使い切られてしまうと、FW 状況コードが戻されます。

専用バッファー・プールは 31 ビット拡張共通ストレージ (ECSA) に配置されます。 共通バッファーは ECSA に配置されるか、または 高速機能 64 ビット・バッファー・マネージャーが使用可能であれば 64 ビット・ストレージに配置されます。

DEDB または MSDB バッファー・プールの設計

データベース呼び出しの結果として生じる要求を満たすために必要なバッファーは、高速機能バッファー・プールと呼ばれるグローバル・プールから取得されます。

高速機能 64 ビット・バッファー・マネージャーを使用している場合は、IMS によって高速機能バッファー・プールが作成および管理され、DEDB バッファーが 64 ビットのストレージに配置されます。高速機能 64 ビット・バッファー・マネージャーが使用可能である場合は、DEDB または MSDB バッファー・プールを設計する必要がなく、高速機能バッファー・プールを定義する DBBF、DBFX、および BSIZ の各パラメーターを指定する必要もありません。

高速機能 64 ビット・バッファー・マネージャーを使用可能にするには、DFSDFxxx PROCLIB メンバーで FPBP64=Y および FPBP64M を指定します。高速機能 64 ビット・バッファー・マネージャーを使用可能にすると、DBBF、DBFX、および BSIZ パラメーターを指定した場合に、IMS でそれらが無視されます。

高速機能 64 ビット・バッファー・マネージャーを使用していない場合は、IMS システム定義時および IMS 始動時に、プールの特性を各自で指定する必要があります。

特性を各自で指定する場合は、以下の 3 つのパラメーターで高速機能バッファー・プールの特性を設定します。

DBBF
バッファーの総数

バッファー・プールは IMS 始動時に、ECSA の中に、または DFSFDRxx に FPBUFF=LOCAL が指定されている場合は FDBR 専用領域の中に、割り振られます。緊急時再始動処理の間には、バッファー・プール全体が短時間でページ固定されることがあります。 DBBF 値を設定するときには、使用可能な実記憶の量を考慮してください。 IMS はバッファーの総数を X'5937' ログに書き込みます。

IMS リソースのページ固定について詳しくは、「*IMS V13* システム管理」を参照してください。

DBFX
システム・バッファー割り振り

これは、IMS 初期化時にページ固定されたバッファーのセットです。この値は、任意の一時点において処理を行う出力スレッドでアクティブであることが期待されるバッファーの最大数に近似させる必要があります。この値が小さすぎると、従属領域でバッファーを待機しなければならない場合があります。

BSIZ
バッファー・サイズ

このサイズは、処理されるすべての DEDB の中の最大 CI のサイズに等しいかそれよりも大きくなければなりません。バッファー・サイズは 28 KB までにすることができます。

関連概念:

531 ページの『MSDB バッファーに関する考慮事項』

高速機能バッファーの用途

高速機能バッファーは、さまざまなタイプの高速機能データを保持するために使用します。

高速機能バッファーは、以下のものを保持するために使用します。

- 更新情報。例えば、
  - MSDB FLD/VERIFY 呼び出し論理
  - MSDB FLD/CHANGE 呼び出し論理
  - MSDB 更新 (REPL、ISRT、および DLET 呼び出しの結果)
  - 挿入された SDEP セグメント
- ルート・アドレス可能部と順次従属部から参照された DEDB CI
- ルート・アドレス可能部からの更新済み DEDB CI
- 同期点を通過した SDEP セグメント。これらの SDEP セグメントは現在の SDEP セグメント・バッファーに集められます。 SDEP セグメント・タイプによって定義された各エリアごとに割り振られたこの種のバッファーが 1 つずつ存在します。このバッファーの割り振りは、エリアのオープン時に行われます。

高速機能 64 ビット・バッファー・マネージャー

高速機能 64 ビット・バッファー・マネージャーは、高速機能バッファー・プール (高速処理データベース (DEDB)、主記憶データベース (MSDB)、および高速機能システム・サービス用の各バッファー・プールを含む) の数とサイズを自律的に制御します。高速機能 64 ビット・バッファー・マネージャーでは、システム・プログラマーがシステム定義時にバッファー・プールの仕様を手動で設定する必要がありません。

また、高速機能 64 ビット・バッファー・マネージャーは、DEDB バッファー・プールを 64 ビットの制御領域専用ストレージの 2 GB 境界より上に配置するため、ECSA ストレージの使用量が削減されます。

高速機能 64 ビット・バッファー・マネージャーを使用する場合も、以下の項目は引き続き ECSA ストレージで管理されます。

- MSDB データベース用の高速機能バッファー
- 順次従属 (SDEP) セグメントを挿入するためのバッファー
- システム・サービス用のバッファー
- バッファー・ヘッダー
- FLD 呼び出しおよび MSDB 用の内部 IMS 作業域

高速機能 64 ビット・バッファー・マネージャーを使用しない場合は、システム定義時に DBBF、DBFX、および BSIZ の各実行パラメーターを使用して高速機能バッファー・プールの数とサイズを設定する必要があり、高速機能バッファー・プールはすべて 32 ビットの ECSA ストレージに配置され、オーバーフロー・バッファーへの従属領域のアクセスが逐次化されます。また、IMS 始動後に高速機能バッファー・プールの数とサイズを変更するには、IMS を停止して再始動しなければなりません。

高速機能 64 ビット・バッファー・マネージャーは、DFSDFxxx PROCLIB メンバーの高速機能セクション (SECTION=FASTPATH) で FPBP64 という 1 つのパラメーターを指定することで使用可能になります。使用可能になった高速機能 64 ビット・バッファー・マネージャーは、追加のバッファー・サブプールを必要に応じて割り振ります。また、データベースがオンライン IMS システムに追加され、そのデー

タベースの CI サイズを収容できるアクティブなバッファー・サブプールがない場合は、高速機能 64 ビット・バッファー・マネージャーが適切なサイズの新しいバッファー・サブプールを割り振ります。

要件**:** 高速機能 64 ビット・バッファー・マネージャーの要件は、次のとおりです。

- 最小で 2.1 ギガバイトの 64 ビット・ストレージが必要です。
- 高速データベース・リカバリー (FDBR) アドレス・スペースによってトラッキングされているシステムで高速機能 64 ビット・バッファー・マネージャーが使用される場合は、FDR プロシージャーで DFSDF= キーワードを指定する必要があります。

高速機能 64 ビット・バッファー・マネージャーは、DEDB バッファー・プールを 64 ビットのストレージに配置するため、DEDB に対して呼び出しを出す各従属領域により多くのバッファー・プールを割り当てることができます。バッファー・プールが追加されるたびに、ECSA ストレージの使用量はバッファー・ヘッダーに必要な量だけ増えます。従属領域が使用できるバッファーの数が多いほど、アプリケーション・プログラムがチェックポイント間で実行できる作業量が増えます。

また、高速機能 64 ビット・バッファー・マネージャーは複数の従属領域が同時にオーバーフロー・バッファーにアクセスすることを可能にするため、オーバーフロー・バッファーに対する従属領域間の競合が解消されます。

1 つの従属領域に割り振ることができる最大バッファー数 (通常バッファーとオーバーフロー・バッファーの合計) は、従属領域定義の NBA パラメーターと OBA パラメーターによって決まり、高速機能 64 ビット・バッファー・マネージャーでは制限されません。

高速機能 64 ビット・バッファー・マネージャーを使用可能にすると、次のことが可能になります。

- データベースの CI サイズを変更する場合に、それに合わせてバッファー・サイズを調整する必要がありません。新規または変更された CI サイズを収容できるアクティブなバッファー・サブプールがない場合は、高速機能 64 ビット・バッファー・マネージャーが適切な CI サイズのバッファー・サブプールを自動的に割り振ります。
- IMS タイプ 2 コマンド QUERY POOL TYPE(FPBP64) を実行することで、高速機能バッファーの統計を表示できます。
- 高速機能 64 ビット・バッファーの使用統計は、コマンド UPDATE IMS SET(LCLPARM(FPBP64STAT(Y))) を発行してキャプチャーすることができます。IMS は従属領域内の各作業単位の使用統計をキャプチャーし、その統計をオンライン・ログ・データ・セットに X'5945' ログ・レコードとして書き込みます。これらのログ・レコードは、マクロ DBFL5945 および DBFBPND6 によってマップされます。
- IMS で高速機能 64 ビット・バッファー使用統計のロギングが使用可能であるかどうかは、QUERY IMS SHOW (ALL | LOCAL) コマンドを使用して確認できます。

関連概念:

549 ページの『DBCTL 環境における DEDB バッファー・プールの設計』

通常のバッファー割り振り (NBA)

高速機能領域、および高速機能リソースにアクセスする IMS 領域では、通常バッファーとして割り振るバッファーの数を、領域始動プロシージャーで NBA 始動パラメーターを使用して指定する必要があります。

通常バッファーとして割り振られるバッファーは最初に使用されるため、割り振られる通常バッファーの数がほとんどのトランザクション要件に対応できることが必要です。

高速機能リソースにアクセスする可能性がある場合、トランザクションまたは同期点インターバルで使用できるバッファーの数を、領域ごとに指定する必要があります。

IMS が領域に割り振ることのできるバッファーの最大数は、NBA パラメーターと OBA パラメーターを組み合わせた値によって規定されます。

高速機能 64 ビット・バッファー・マネージャーを使用する場合は、NBA パラメーターで指定される通常バッファーの数が、バッファーが割り振られるときに高速機能バッファー・プールでページ固定されます。高速機能 64 ビット・バッファー・マネージャーを使用しない場合は、NBA パラメーターで指定される通常バッファーの数が、領域の始動時に高速機能バッファー・プールでページ固定されます。

オーバーフロー・バッファー割り振り (OBA)

オーバーフロー・バッファー割り振り (OBA) は、オプションであり、通常のバッファー割り振り (NBA) が使い尽くされたときの例外的なバッファー要件のために使用されます。

高速機能 64 ビット・バッファー・マネージャーを使用する場合は、オーバーフロー・バッファーへのアクセスがマルチスレッド化され、複数の領域が同時にオーバーフロー・バッファーを使用できます。

高速機能 64 ビット・バッファー・マネージャーを使用しない場合は、領域によるオーバーフロー・バッファーへのアクセスが、現在オーバーフロー・バッファー状態にあるすべての領域を逐次化するラッチの取得に依存します。このラッチが使用可能でなければ、使用可能になるまで領域は待たなければなりません。このラッチが取得されたあと、NBA 値が OBA 値だけ増やされて、通常の処理が再開されます。このオーバーフロー・バッファー・ラッチは、同期点処理の間は解放されます。どの時点でも、すべてのアクティブ領域の中での最大の OBA 要求だけが高速機能バッファー・プールにページ固定されます。

高速機能 64 ビット・バッファー・マネージャーを使用する場合は、従属領域に指定されている NBA パラメーターと OBA パラメーターを組み合わせた値によって、高速機能 64 ビット・バッファー・マネージャーがその領域に割り振ることができるバッファーの最大数が規定されます。

高速機能バッファー割り振りアルゴリズム

高速機能バッファーは、要求に応じて、各従属領域の NBA パラメーターで指定された限度まで割り振られます。このように指定されたバッファーは、1 つの同期点インターバルによって使用される通常のバッファー割り振り (NBA) と呼ばれます。

NBA 割り振りからの要求を満たす前に、すでに割り振られている SDEP CI が入っているバッファーを再使用しようとする試みがなされます。この処理は、NBA 限度に達するまで続けられます。その限度以降は、高速機能データベースの呼び出しに対して戻される FW 状況コードの形で、BMP 領域に警告が送られます。 MD 領域と MPP 領域はこの警告を受けません。

追加バッファーに対して次の要求が出されると、バッファー・スチール機能が呼び出され、次に、すでに割り振られている各バッファーと CI がアルゴリズムにより調べられます。その結果、解放しようとしている CI が入っているバッファーがローカル・キュー (SDEP バッファー・チェーン) に送られ、この同期インターバルで再使用されます。

バッファー・スチール機能を呼び出した後も、使用可能なバッファーが見つからない場合には、オーバーフロー・バッファー・ラッチに対する要求が出されます。オーバーフロー・バッファー・ラッチは、オーバーフロー・バッファー割り振り (OBA) と呼ばれる追加バッファー割り振りの使用を制御するためのものです。OBA 割り振りも、領域の開始時にパラメーターとして指定されます。この時点以降は、要求をローカルに満たすことができなくなるつど、OBA 限度に達するまでバッファーが OBA 割り振りから取得されます。その限度に達すると、MD 領域と BMP 領域では、内部 ROLB 呼び出しが実行された後、各「FW」状況コードが「FR」状況コードで置き換えられます。 MD 領域と MPP 領域では、トランザクションが異常終了し、停止します。

関連概念:

228 ページの『セグメント CI のエンキュー・レベル』

DBFX パラメーター使用時の高速機能バッファー割り振り

IMS は、DBBF パラメーターで指定された高速機能バッファーの総数から、DEDB 書き込みに使用するバッファーの数を確保します。IMS が確保するバッファーの数は、DBFX パラメーターで指定します。

1 つのトランザクションまたは同期インターバルの結果が、1 つの出力スレッドによって書き戻されます。これらの出力スレッドは、制御領域から SRB モードで実行されます。したがって、出力スレッドに割り振られたバッファーは、従属領域に入っている CI が書き戻されるまで従属領域では使用可能になりません。

高速機能バッファー・プールがすべての NBA の和として正確に定義されている場合、従属領域は、バッファーがグローバル・プールに戻ってくるのを待つ必要があります。高速機能領域は、同期点が完了すればすぐに、次のトランザクションを処理することができます。同期点処理は、出力スレッドが完了するのを待ちません。バッファーの割り振りは、NBA 要求を指定している最初の領域の開始時にページ固定されます。

高速機能 64 ビット・バッファー・マネージャーを使用する場合は、IMS によって高速機能バッファーが管理されるので、DBFX、DBBF、または BSIZ パラメーターを指定する必要がなくなります。バッファーの割り振りは、バッファーが割り振られるときにページ固定されます。

高速機能バッファー・プールのサイズ決定

高速機能 64 ビット・バッファー・マネージャーを使用していない場合は、高速機能バッファー・プールに入っている必要があるバッファーの数を、`DBBF ≥ A + N + OBA + DBFX` という数式で計算できます。

上記の数式の各項について、以下で説明します。

DBBF 指定する高速機能バッファー・プールのサイズ

A SDEP セグメントを持つアクティブ・エリアの数

NBA 各アクティブ領域の通常のバッファー割り振り

N すべての NBA の合計

OBA 最大オーバーフロー・バッファー割り振り

DBFX システム・バッファー割り振り

高速機能バッファーのパフォーマンスに関する考慮事項

高速機能バッファーのパフォーマンスに関する考慮事項は、高速機能 64 ビット・バッファー・マネージャーを使用しているかどうかによって異なります。

高速機能 64 ビット・バッファー・マネージャーを使用して高速機能バッファー・プールの作成と管理を行っている場合は、バッファー・プールのパフォーマンスに関する多くの側面が IMS によって自動的に最適化されます。

高速機能 64 ビット・バッファー・マネージャーを使用していない場合は、バッファー・プールの最適なパフォーマンスを維持するために、バッファー・プールの仕様を手動で変更しなければならないことがあります。

以下の考慮事項は、高速機能 64 ビット・バッファー・マネージャーによって管理されるバッファーと、高速機能 64 ビット・バッファー・マネージャーによって管理されない高速機能バッファーの両方に適用されます。

- NBA 値が大きすぎると、他のトランザクションによる競合 (したがって、遅延) の可能性が大きくなることがあります。すべての CI が排他レベルで取得され、バッファー・スチール機能が呼び出されるまで排他レベルで保持されることがあります。バッファー・スチール機能が呼び出されるのは、NBA 限度に達した後です。したがって、NBA が大きすぎると、リソース競合が増えることになります。
- (NBA + OBA) 値が小さすぎると、処理の失敗がより頻繁に起きる可能性があります。処理の失敗とは、BMP 領域に対する「FR」状況コード状態あるいは MD 領域と MPP 領域でのトランザクションの異常終了を意味します。
- 照会専用プログラムは、オーバーフロー・バッファー指定 (OBA) を使用しません。 NBA 限度に達すると、すでに割り振られているバッファーが再使用されるからです。

- IMS はバッファーとその使用に関する情報を X'5937' ログに記録します。この情報は、高速機能バッファーがどれだけ効率的に使用されているかを判別する際に役立つことがあります。

以下の考慮事項は、高速機能バッファーが高速機能 64 ビット・バッファー・マネージャーで管理されていない場合にのみ適用されます。

- DBBF の指定が不適切な (小さすぎる) 場合、エリアのオープンや領域の初期設定がリジェクトされる場合があります。システムはバッファー・プールのサイズを計算し、実際の DBBF 値がそれより小さければ、オープンも初期設定もリジェクトします。
- DBFX 値が小さすぎると、領域の待ちが引き起こされたり、応答時間が延びたりしがちです。
- NBA 値が小さすぎると、オーバーフロー・バッファー・ラッチのために領域処理が逐次化されるかもしれず、こうなればまた遅延が起きることになります。

関連タスク:

547 ページの『高速機能バッファー・プールのサイズ決定』

NBA 限度と同期点

BMP 領域では、NBA 限度に達すると「FW」状況コードが戻されます。 この状況コードは、これ以降のすべての高速機能データベース呼び出しに対して OBA 限度状態に達するまで提供されます。

最初に発生した「FW」状況コードは、これ以上 NBA バッファーが存在しないことを示しています。このことは、同期点を要求するのに都合の良い点です。高速機能リソース (および他のリソース) が解放され、次の同期点インターバルは新しい 1 組の NBA バッファーの使用を許可されることになります。オーバーフロー・バッファー・ラッチは、オーバーフロー・バッファー状態にあるすべての領域を逐次化するので、これらの領域での処理に遅延が生じます。

処理が主として順次処理の場合には、UOW 境界を交差するときに同期点を呼び出します。

関連概念:

526 ページの『処理オプション P (PROCOPT=P)』

DBFX 値と低アクティビティー環境

システムでの IMS アクティビティーまたは高速機能アクティビティーが比較的不活発であると、ログ・バッファーの書き込みがそれほど頻繁でなく、したがって、出力スレッドのスケジュールまたはディスパッチもそれほど頻繁ではありません。このような状況では、書き込み待ちバッファーの数が多くなる可能性があるため、バッファー待ち状態が起こることがあります。

バッファー待ちの状態を緩和または回避するには、高速機能バッファーをユーザーに代わって動的に管理する高速機能 64 ビット・バッファー・マネージャーを使用可能にします。高速機能 64 ビット・バッファー・マネージャーを使用可能にすると、IMS がバッファーの使用状況をトラッキングし、必要に応じてバッファーを追加または削除します。DBBF、DBFX、および BSIZ の各パラメーターが指定されている場合は、それらが無視されます。

高速機能 64 ビット・バッファー・マネージャーを使用しない場合は、DBFX により大きな値を指定してください。

高速機能 64 ビット・バッファー・マネージャーを使用しない場合は、BMP 領域で DEDB のロードまたは処理が行われ、それがシステムでの唯一のアクティビティーであるという特殊なケースを考慮する必要があります。例えば、NBA が 20 のバッファーが存在するとします。バッファー待ち状態を回避するためには、DBFX 値を NBA 値の 1 倍と 2 倍の間に指定しなければなりません。こうすれば、DBBF が指定した NBA の数の 3 倍となることもあり、これは、60 バッファーが高速機能バッファー・プールに与えられることを意味します。

次の場合を除き、複数のトランザクションまたは同期点インターバルにわたるバッファーのルックアサイド機能 (グローバル・バッファー・ルックアサイド) はありません。

ある領域が、現在書き込み中であるか、または他の領域に所有されており、最終的には書き込まれる (出力スレッド処理) DEDB CI リソースを要求するとします。この場合、この書き込みが正常に完了した後 (読み取りは不要)、この CI とバッファーは要求した領域に渡されます。他の領域はすべてそれをディスクから読み取らなければなりません。

DBCTL 環境における DEDB バッファー・プールの設計

DBCTL 環境では、データベース呼び出しからの要求を満たすために必要なバッファーは、高速機能バッファー・プールと呼ばれるグローバル・プールから取得されます。

高速機能 64 ビット・バッファー・マネージャーを使用している場合は、IMS によって高速機能バッファー・プールが作成および管理され、DEDB バッファーが 64 ビットのストレージに配置されます。高速機能 64 ビット・バッファー・マネージャーが使用可能である場合は、DEDB または MSDB バッファー・プールを設計する必要がなく、高速機能バッファー・プールを定義する DBBF、DBFX、および BSIZ の各パラメーターを指定する必要もありません。

高速機能 64 ビット・バッファー・マネージャーを使用可能にするには、DFSDFxxx PROCLIB メンバーで FPBP64=Y を指定します。高速機能 64 ビット・バッファー・マネージャーを使用可能にすると、DBBF、DBFX、および BSIZ パラメーターを指定した場合に、IMS でそれらが無視されます。

高速機能 64 ビット・バッファー・マネージャーを使用していない場合は、IMS システム定義時および IMS 始動時に、プールの特性を各自で指定する必要があります。

特性を各自で指定する場合は、以下の 3 つのパラメーターで高速機能バッファー・プールの特性を設定します。

DBBF
バッファーの総数

バッファー・プールは IMS 始動時に、ECSA の中に、または DFSFDRxx に FPBUFF=LOCAL が指定されている場合は FDBR 専用領域の中に、割り振られます。 IMS はバッファーの総数を X'5937' ログに書き込みます。

DBFX
システム・バッファー割り振り

これは、高速機能リソースにアクセスする最初の領域の始動の際にページ固定される高速機能バッファー・プールの中の 1 組のバッファーです。

BSIZ
バッファー・サイズ

このサイズは、処理されるすべての DEDB の中の最大 CI のサイズに等しいかそれよりも大きくなければなりません。バッファー・サイズは 28 KB までにすることができます。

関連概念:

543 ページの『高速機能 64 ビット・バッファー・マネージャー』

DBCTL 環境での高速機能バッファーの用途

高速機能バッファーは、さまざまなタイプの高速機能データを保持するために使用します。

高速機能バッファーは、以下のものを保持するために使用します。

- 挿入された SDEP セグメントのような更新情報
- ルート・アドレス可能部と順次従属部から参照された DEDB CI
- ルート・アドレス可能部からの更新済み DEDB CI
- 同期点を通過した SDEP セグメント。これらのセグメントは、現在の SDEP セグメント・バッファーに集められます。 SDEP セグメント・タイプによって定義されたエリアごとに割り振られたバッファーが 1 つずつ存在します。このバッファーの割り振りは、エリアのオープン時に行われます。

DBCTL 環境での BMP に対する通常のバッファー割り振り

高速機能リソースにアクセスする BMP 領域では、通常バッファーとして割り振るバッファーの数を、領域始動プロシージャーで NBA 始動パラメーターを使用して指定する必要があります。

通常バッファーとして割り振られるバッファーは最初に使用されるため、割り振られる通常バッファーの数がほとんどのトランザクション要件に対応できることが必要です。

高速機能リソースにアクセスする可能性がある場合、トランザクションまたは同期点インターバルで使用できるバッファーの数を、領域ごとに指定する必要があります。

IMS が領域に割り振ることのできるバッファーの最大数は、NBA パラメーターと OBA パラメーターを組み合わせた値によって規定されます。

高速機能 64 ビット・バッファー・マネージャーを使用する場合は、NBA パラメーターで指定される通常バッファーの数が、バッファーが割り振られるときに高速機

能バッファー・プールでページ固定されます。高速機能 64 ビット・バッファー・マネージャーを使用しない場合は、NBA パラメーターで指定される通常バッファーの数が、領域の始動時に高速機能バッファー・プールでページ固定されます。

CCTL 領域と CCTL スレッドに対する通常のバッファー割り振り

CCTL (コーディネーター・コントローラー) 領域で高速機能リソースが必要である場合は、CCTL 領域と CCTL スレッドの両方に、通常のバッファー割り振りを指定する必要があります。

通常のバッファー割り振りを指定するには、データベース・リソース・アダプター (DRA) 始動テーブルで指定される以下のパラメーターを使用します。

CNBA
各アクティブ CCTL 領域の通常のバッファー割り振りを指定します。

FPB
CCTL スレッドに対する通常のバッファー割り振りを指定します。

CCTL が DBCTL に接続するとき、CNBA バッファーの数が高速機能バッファー・プールにページ固定されます。しかし、CNBA バッファーが使用可能でない場合、接続は失敗します。

DEDB バッファーが必要な各 CCTL スレッドは、CNBA バッファーの総数からそれぞれ高速機能バッファー (FPB) を割り当てられます。

CCTLNBA パラメーターの詳細については、「*IMS V13* システム管理」を参照してください。

BMP に対するオーバーフロー・バッファー割り振り

BMP に対するオーバーフロー・バッファー割り振りはオプションであり、通常のバッファー割り振りが使い果たされたときの例外的なバッファー要件に使用されます。

高速機能 64 ビット・バッファー・マネージャーを使用する場合は、オーバーフロー・バッファーへのアクセスがマルチスレッド化され、複数の BMP 領域と CCTL スレッドが同時にオーバーフロー・バッファーを使用できます。

高速機能 64 ビット・バッファー・マネージャーを使用しない場合は、BMP 領域または CCTL スレッドによるオーバーフロー・バッファーへのアクセスが、現在オーバーフロー・バッファー状態にあるすべての BMP と CCTL スレッドを逐次化するラッチの取得に依存します。このラッチが使用可能でなければ、使用可能になるまで領域は待たなければなりません。このラッチが取得されたあと、NBA 値が OBA 値だけ増やされて、通常の処理が再開されます。このオーバーフロー・バッファー・ラッチは、同期点処理の間は解放されます。どの時点でも、すべてのアクティブ BMP と CCTL スレッドの中での最大の OBA 要求だけが高速機能バッファー・プールにページ固定されます。

高速機能が領域に割り振ることのできるバッファーの最大数は、従属領域に対して指定されている NBA パラメーターと OBA パラメーターを組み合わせた値によって規定されます。

CCTL スレッドに対するオーバーフロー・バッファー割り振り

CCTL スレッドのための OBA は、BMP の場合と似ています。各スレッドに使用される OBA 値は、始動テーブルの FPOB パラメーターで設定します。

このバッファー割り振りは、オプションであり、FPB が使い尽くされたときの例外的なバッファー要件のために使用されます。その使用は、現在オーバーフロー・バッファー状態にあるすべての BMP および CCTL スレッドを逐次化するためのラッチ (留め金) の取得にかかっています。このラッチが取得されないと、FPB 値が FPOB 値だけ増やされて、通常の処理が再開されます。このオーバーフロー・バッファー・ラッチは、同期点処理の間は解放されます。どの時点においても、すべてのアクティブ BMP と CCTL スレッドの中での最大の OBA/FPOB 要求だけが高速機能バッファー・プールにページ固定されます。

BMP に対する高速機能バッファー割り振りアルゴリズム

FPB は、要求に応じて、領域の始動時に指定された限度まで割り振られます。NBA として指定されたバッファーは、1 つの同期点インターバルによって使用されます。

NBA 割り振りからの要求を満たす前に、すでに割り振られている SDEP CI が入っているバッファーを再使用しようとする試みがなされます。この処理は、NBA 限度に達するまで続けられます。その限度以降は、高速機能データベースの呼び出しに対して戻される 'FW' 状況コードの形で、BMP 領域に警告が送られます。

追加バッファーに対して次の要求が出されると、バッファー・スチール機能が呼び出され、次に、すでに割り振られている各バッファーと CI がアルゴリズムにより調べられます。その結果、解放しようとしている CI が入っているバッファーがローカル・キュー (SDEP バッファー・チェーン) に送られ、この同期インターバルで再使用されます。

バッファー・スチール機能を呼び出した後も、使用可能なバッファーが見つからない場合には、オーバーフロー・バッファー・ラッチに対する要求が出されます。オーバーフロー・バッファー・ラッチは、OBA と呼ばれる追加バッファー割り振りの使用を制御するためのものです。この割り振りも、領域の開始時にパラメーターとして指定されます。この時点以降は、要求をローカルに満たすことができなくなるつど、OBA 限度に達するまでバッファーが OBA 割り振りから取得されます。その限度に達すると、BMP 領域では、内部 ROLB 呼び出しが実行された後、「FW」状況コードが「FR」状況コードで置き換えられます。

CCTL スレッドに対する高速機能バッファー割り振りアルゴリズム

CCTL スレッドが FPB を使用してスケジュール要求を出すときに、CNBA 合計からバッファーが割り振られます。

CNBA=0 が指定されていない場合、あるいは高速機能 64 ビット・バッファー・マネージャーを使用していない場合に、FPB を CNBA から割り振ることができないと、スケジュール要求は失敗します。CNBA=0 が指定されているか、あるいは高速機能 64 ビット・バッファー・マネージャーを使用している場合に、FPB を CNBA から割り振ることができないと、IMS は、スレッドをスケジュールできるように追加のバッファーを取得します。

FPB 割り振りからの要求を満たす前に、すでに割り振られている SDEP CI が入っているバッファーを再使用しようとする試みがなされます。この処理は、FPB の限度に達するまで続けられます。その時点以降は、高速機能データベース呼び出しに戻される「FW」状況コードの形で警告が、CCTL スレッドに送られます。

追加バッファーに対する次の要求が出されると、バッファー・スチール機能が呼び出され、次いで、すでに割り振られているそれぞれのバッファーと CI がアルゴリズムによって調べられます。その結果、解放しようとしている CI が入っているバッファーがローカル・キュー (SDEP バッファー・チェーン) に送られ、この同期インターバルで再使用されます。

バッファー・スチール機能を呼び出した後も、使用可能なバッファーが見つからない場合には、オーバーフロー・バッファー・ラッチに対する要求が出されます。オーバーフロー・バッファー・ラッチは、オーバーフロー・バッファー割り振り OBA (FPOB) と呼ばれる追加バッファー割り振りの使用を制御するためのものです。この時点以降は、要求をローカルに満たすことができなくなるつど、FPOB 限度に達するまでバッファーが FPOB 割り振りから取得されます。その限度に達すると、CCTL スレッドでは、内部 ROLB 呼び出しが実行された後、「FW」状況コードが「FR」状況コードで置き換えられます。

DBCTL 環境での高速機能バッファー割り振り

IMS が高速機能バッファーを割り振るのは、DEDB 書き込みが同期点処理の後まで延期されるためです。

1 つの同期インターバルの結果が、1 つの出力スレッドによって書き戻されます。これらの出力スレッドは、制御領域から SRB モードで実行されます。したがって、出力スレッドに割り振られたバッファーは、これに入っている CI が書き戻されるまで BMP と CCTL スレッドに対して使用可能になりません。

高速機能 64 ビット・バッファー・マネージャーを使用する場合は、IMS によって高速機能バッファーの割り振りが管理されます。バッファーの割り振りは、バッファーが割り振られるときにページ固定されます。

高速機能 64 ビット・バッファー・マネージャーを使用しない場合は、DBFX パラメーターを使用して IMS が割り振るバッファーの数を指定します。高速機能バッファー・プールがすべての NBA の和として正確に定義されている場合、BMP と CCTL スレッドは、バッファーがグローバル・プールに戻ってくるのを待たなければなりません。 BMP と CCTL スレッドは、同期点が完了するとすぐに、次のトランザクションを処理することができます。同期点処理は、出力スレッドが完了するのを待ちません。バッファーの割り振りは、NBA 要求または FPB 要求を指定している最初の領域の始動時にページ固定されます。

DBCTL に対する高速機能バッファー・プールのサイズの決定

高速機能 64 ビット・バッファー・マネージャーを使用していない場合は、DBCTL 環境で高速機能バッファー・プールに入っている必要があるバッファーの数を、`DBBF ≥ A + N + LO + DBFX + CN` という数式で計算できます。

上記の数式の各項について、以下で説明します。

DBBF 指定する高速機能バッファー・プールのサイズ

A SDEP セグメントを持つアクティブ・エリアの数

N すべての NBA の合計

LO アクティブ BMP と CCTL スレッドの中の最大オーバーフロー・バッファー割り振り

DBFX システム・バッファー割り振り

CN すべての CNBA の合計

DBCTL における高速機能バッファーのパフォーマンスに関する考慮事項

高速機能バッファーのパフォーマンスに関する考慮事項は、高速機能 64 ビット・バッファー・マネージャーを使用しているかどうかによって異なります。

高速機能 64 ビット・バッファー・マネージャーを使用して高速機能バッファー・プールの作成と管理を行っている場合は、バッファー・プールのパフォーマンスに関する多くの側面が IMS によって自動的に最適化されます。

高速機能 64 ビット・バッファー・マネージャーを使用していない場合は、バッファー・プールの最適なパフォーマンスを維持するために、バッファー・プールの仕様を手動で変更しなければならないことがあります。

以下の考慮事項は、高速機能 64 ビット・バッファー・マネージャーによって管理されるバッファーと、高速機能 64 ビット・バッファー・マネージャーによって管理されない高速機能バッファーの両方に適用されます。

NBA/FPB 値が大きすぎると、他の BMP および CCTL スレッドによる競合 (したがって遅延) の可能性が大きくなることがあります。すべての CI が排他レベルで取得され、バッファー・スチール機能が呼び出されるまで排他レベルで保持されることがあります。バッファー・スチール機能が呼び出されるのは、NBA 限度に達した後です。したがって、NBA/FPB が大きすぎると、リソース競合が増えます。また、FPB 値が大きすぎるのは、高速機能 PSB を同時にスケジュールすることのできる CCTL スレッドの数が少なくなることを示しています。

(NBA + OBA) 値が小さすぎると、処理の失敗がより頻繁に起きる可能性があります。これは、BMP 領域と CCTL スレッドで「FR」状況コード状態になっていることを意味しています。

照会専用の BMP または CCTL プログラムは、オーバーフロー・バッファー指定論理を使用しません。NBA/FPB 限度に達すると、すでに割り振られているバッファーが再使用されるからです。

IMS はバッファーとその使用に関する情報を X'5937' ログに記録します。この情報は、高速機能バッファーがどれだけ効率的に使用されているかを判別する際に役立つことがあります。

以下の考慮事項は、高速機能バッファーが高速機能 64 ビット・バッファー・マネージャーで管理されていない場合にのみ適用されます。

DBBF の指定が不適切な (小さすぎる) 場合、エリアのオープンや領域の初期設定がリジェクトされる場合があります。システムはバッファー・プールのサイズを計算し、実際の DBBF 値がそれより小さければ、オープンも初期設定もリジェクトします。

DBFX 値が小さすぎると、領域の待ちが引き起こされたり、応答時間が延びたりしがちです。

NBA/FPB 値が小さすぎると、オーバーフロー・バッファー・ラッチのために領域処理が逐次化される場合があり、こうなればまた遅延が起きることになります。

関連タスク:

553 ページの『DBCTL に対する高速機能バッファー・プールのサイズの決定』

DBCTL 環境における NBA/FPB 限度と同期点

BMP 領域と CCTL スレッドでは、NBA/FPB 限度に達すると、「FW」状況コードが戻されます。 この状況コードは、OBA/FPOB 限度状態に達するまで、これ以降のすべての高速機能データベース呼び出しに対して提供されます。

最初に発生した「FW」状況コードは、これ以上 NBA/FPB バッファーが存在しないことを示しています。このことは、同期点を要求するのに都合の良い点です。高速機能リソース (および他のリソース) が解放され、次の同期点インターバルは新しい 1 組の NBA/FPB バッファーの使用を許可されることになります。オーバーフロー・バッファー・ラッチは、オーバーフロー・バッファー状態にあるすべての領域を逐次化するので、これらの領域での処理に遅延が生じます。

関連概念:

526 ページの『処理オプション P (PROCOPT=P)』

DBCTL 環境における低アクティビティーと DBFX 値

システムでの IMS アクティビティーまたは高速機能アクティビティーが比較的不活発であると、ログ・バッファーの書き込みがそれほど頻繁でなく、したがって出力スレッドのスケジュールまたはディスパッチもそれほど頻繁ではありません。このような状況では、書き込み待ちバッファーの数が多くなる可能性があるため、バッファー待ち状態が起こることがあります。

バッファー待ちの状態を緩和または回避するには、高速機能バッファーをユーザーに代わって動的に管理する高速機能 64 ビット・バッファー・マネージャーを使用可能にします。高速機能 64 ビット・バッファー・マネージャーを使用可能にすると、IMS がバッファーの使用状況をトラッキングし、必要に応じてバッファーを追加または削除します。DBBF、DBFX、および BSIZ の各パラメーターが指定されている場合は、それらが無視されます。

高速機能 64 ビット・バッファー・マネージャーを使用しない場合は、DBFX により大きな値を指定してください。

高速機能 64 ビット・バッファー・マネージャーを使用しない場合は、BMP 領域で DEDB のロードまたは処理が行われ、それがシステムでの唯一のアクティビティーであるという特殊なケースを考慮する必要があります。例えば、NBA が 20 のバッファーが存在するとします。バッファー待ち状態を回避するためには、DBFX 値を

NBA 値の 1 倍と 2 倍の間に指定します。こうすれば、DBBF が指定した NBA の数の 3 倍となることもあり、これは、60 バッファーが高速機能バッファー・プールに与えられることを意味します。

次の場合を除き、複数の BMP 領域と CCTL スレッドまたは同期点インターバルにわたるバッファーのルックアサイド機能 (グローバル・バッファー・ルックアサイド) はありません。

ある領域が、現在書き込み中であるか、または他の領域に所有されており、最終的には書き込まれる (出力スレッド処理) DEDB CI リソースを要求するとします。この場合、この書き込み (読み取りは不要) が正常に完了した後、この CI とバッファーはリクエスターに渡されます。他の BMP 領域と CCTL スレッドはそれをディスクから読み取らなければなりません。

IMS 領域での高速機能バッファー割り振り

高速機能リソースにアクセスする IMS 領域は、その始動プロシージャーで NBA (通常のバッファー割り振り) パラメーターと OBA パラメーター (オーバーフロー・バッファー割り振り) を指定しなければなりません。

MODE=MULT では、これらの割り振りは、同期点と同期点の間で処理されるトランザクションに必要なすべてのバッファーを保管するのに十分な大きさでなければなりません。

MODE=SNGL では、トランザクション・クラスを設けて、同様のバッファー要件を持つトランザクションは同じ領域で実行すべきです。

第 23 章 データベース設計のインプリメント

データベースとアプリケーション・プログラムを設計した後、それらを使用する前に、それらの特性を IMS に対して記述する必要があります。

各データベースのデータベース記述子 (DBD) をコーディングおよび生成することにより、データベースの物理的および論理的な特性を IMS に対して記述する必要があります。

アプリケーション・プログラムでデータベースを使用する前に、IMS にアプリケーション・プログラムの特性とデータと端末の使用について伝えなければなりません。プログラム仕様ブロック (PSB) をコーディングし、生成することによって、アプリケーション・プログラムの特性を IMS に伝えます。

最後に、アプリケーション・プログラムの実行をスケジュールする前に、アプリケーション・プログラム用の PSB と DBD の情報が、アプリケーション制御ブロック (ACB) と呼ばれる特別内部フォーマットで、IMS から使用可能になっている必要があります。以下の図は、PSB および DBD から情報を収集する方法を示しています。

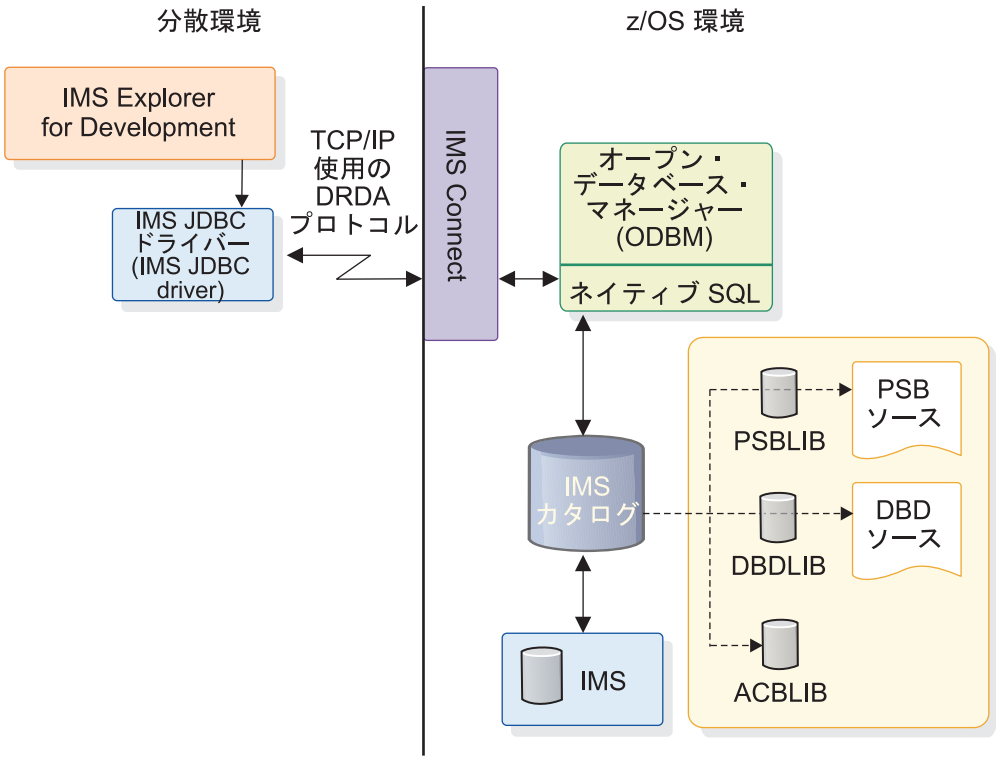


図 236. 分散環境および z/OS 環境

関連概念:

37 ページの『第 1 回コード検査の参加者』

487 ページの『第 21 章 全機能データベースの設計』

521 ページの『第 22 章 高速機能データベースの設計』

DBDGEN ユーティリティーの入力としてのデータベース記述のコーディング

データベース記述子 (DBD) は、データベースの編成やアクセス方式、データベース・レコードの中のセグメントやフィールド、セグメント・タイプ間の関係などについて記述する一連のマクロ命令によって定義されます。

DBD マクロ命令は、コーディングした後、DBDGEN ユーティリティーへの入力として使用されます。このユーティリティーは、DBD 制御ブロックを生成し、それを IMS.DBDLIB ライブラリーに保管してその後のデータベースの処理時に使用できるようにするマクロ・アセンブラーです。

IMS カタログが使用可能の場合、DBD マクロ命令の中にコーディングした情報は、IMS カタログに保管されているデータベースおよびアプリケーション・プログラムのメタデータの多くも提供します。このメタデータには、フィールドのデータ・タイプ、アプリケーション・プログラムのデータ構造、日時のフォーマットなども含まれています。このメタデータは、主に ACB メンバーが生成された後に IMS.ACBLIB データ・セットからカタログに読み込まれますが、場合によっては IMS.DBDLIB データ・セットから直接読み取られることもあります。

以下の図は DBD 生成処理を示しています。

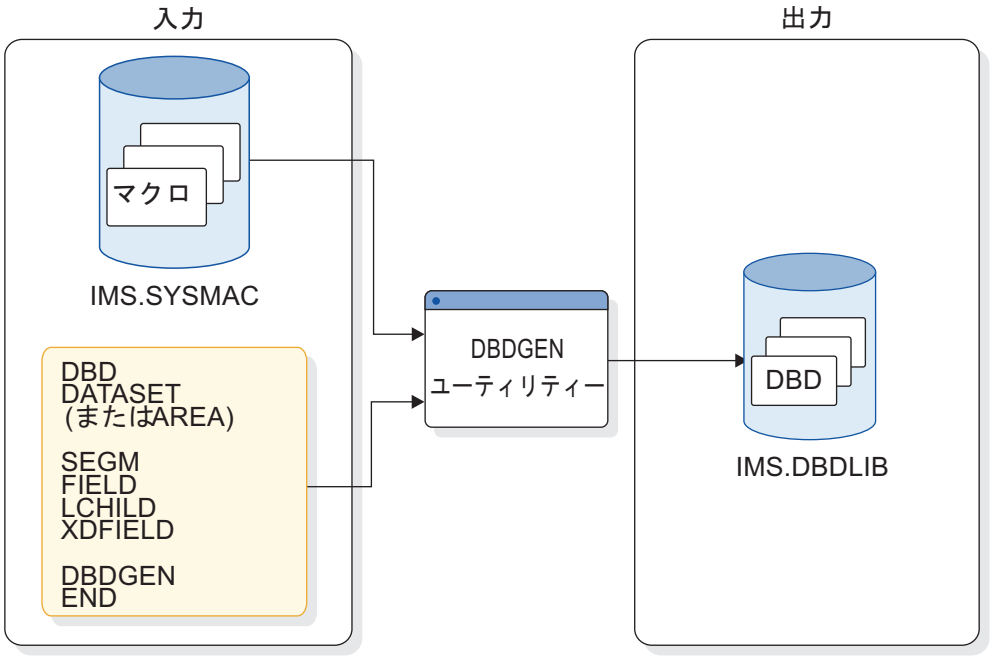


図 *237. DBD* 生成処理

以下の JCL は、DBDGEN ユーティリティーへの入力の例を示しています。定義するデータベースごとに別々の入力が必要です。

```
//DBDGEN    JOB  MSGLEVEL=1
//                EXEC        DBDGEN,MBR=APPLPGM1
//C.SYSIN       DD     *
```

```
    DBD                 DBD 生成ごとに必要
    data set(or AREA)    データ･セット･グループごとに必要
                         (または高速機能 DEDB の AREA)
    SEGM                セグメント･タイプごとに必要
    FIELD               DBD 生成ごとに必要
    LCHILD              副次索引ごとまたは
                         論理関係ごとに必要
    XDFLD               副次索引関係ごとに必要
        .
        .
        .
    DBDGEN              DBD 生成ごとに必要
    END                 DBD 生成ごとに必要
/*
```

上記の例に示したステートメントのほかに、DBDGEN ユーティリティーへの入力に以下のマクロ・ステートメントを含めることもできます。

- DFSMARSH (これは、個々のフィールドのデータ・マーシャル・プロパティーを定義します。)
- DFSMAP および DFSCASE (これらはセグメントの代替フィールド・マップを定義します。)

関連タスク:

584 ページの『HALDB 区画定義ユーティリティーによる HALDB データベースの作成』

関連資料:

データベース記述 (DBD) 生成ユーティリティー (システム・ユーティリティー)

DBD ステートメントの概要

入力では、DBD ステートメントは、記述しようとしているデータベースの名前を指定し、その編成および出口ルーチン (ある場合) を含む、各種の属性を指定します。

入力デックには DBD ステートメントを 1 つだけコーディングします。

関連資料:

DBD ステートメント (システム・ユーティリティー)

DATASET ステートメントの概説

DATASET ステートメントは、該当のデータベースのために使用されるデータ・セットの物理的特性を定義します。

少なくとも 1 つの DATASET ステートメントが、データベースの中のそれぞれのデータ・セット・グループごとに必要です。データベース・タイプに応じて、最高 10 のデータ・セット・グループを定義することができます。それぞれの DATASET ステートメントの後ろには、各データ・セット・グループの中に置かれるすべてのセグメントのための SEGM セグメントが続きます。

DATASET ステートメントは HALDB データベースには使用できません。HALDB 区画定義ユーティリティーまたは DBRC コマンド INIT.DB と INIT.PART を使用して HALDB 区画を定義してください。

データベースが DEDB の場合には、DATASET ステートメントの代わりに、AREA ステートメントが使用されます。

関連資料:

- DATASET ステートメント (システム・ユーティリティー)
- AREA ステートメント (システム・ユーティリティー)

AREA ステートメントの概説

AREA ステートメントは、高速機能の高速処理データベース (DEDB) のエリアを定義します。

少なくとも 1 つの AREA ステートメントが必要であり、複数のエリアを定義するのに 最大 2,048 個の AREA ステートメントを使用することができます。

AREA ステートメントはすべて DBD ステートメントと最初の SEGM ステートメントの間になければなりません。

AREA ステートメントは、高速機能 DEDB データベース・タイプにのみ使用されます。全機能データベースは、代わりに DATASET ステートメントを指定します。

関連資料:

64 ページの『AREA セグメント・タイプのフォーマット』

- AREA ステートメント (システム・ユーティリティー)
- DATASET ステートメント (システム・ユーティリティー)

SEGM ステートメントの概説

SEGM ステートメントは、データベースでのセグメント・タイプ、階層におけるセグメントの位置、セグメントの物理的特性、およびそのセグメントと他のセグメントとの関係を定義します。

SEGM ステートメントは、階層順に入力デックの中に置かれ、最大 15 の階層レベルを定義することができます。許されるデータベース・ステートメントの数は、データベース・タイプによって異なります。 SEGM ステートメントは、関連するデータ・セット・ステートメントまたは AREA ステートメントの直後に指定する必要があります。

関連資料:

- SEGM ステートメント (システム・ユーティリティー)

94 ページの『SEGM セグメント・タイプのフォーマット』

FIELD ステートメントの概要

FIELD ステートメントは、セグメント・タイプの中の 1 つのフィールドを定義します。

FIELD ステートメントは、以下のタイプのフィールドに必須です。

- セグメント内のシーケンス・フィールド

- アプリケーション・プログラムが DL/I 呼び出しのセグメント検索引数 (SSA) の中で名前で参照するフィールド
- PSB 内の SENFLD ステートメントで参照されるフィールド
- XDFLD ステートメントで参照されるフィールド

上記の各フィールド・タイプの FIELD ステートメントでは、NAME パラメーターを指定する必要があります。NAME パラメーターが指定された場合、フィールド名はデータベース・データ管理ブロック (DMB) に保管されます。DMB のストレージを節約するために、NAME パラメーターが必須である場合以外は、EXTERNALNAME パラメーターだけを指定してください。

IMS に対し、それ以外ではアプリケーション・ソース・コードおよびコピーブックの中で定義するしか方法がないデータ・フィールドと構造を定義するために、FIELD ステートメントをコーディングすることもできます。それらのオプションの FIELD ステートメントをコーディングする利点は、IMS カタログが使用可能な場合、アプリケーション開発者およびその他のユーザーは、それらのデータ・フィールドと構造に関して、IMS カタログにメタデータを照会できることです。これにより、アプリケーション・ソース・コードを見つけたり、IMS Enterprise Suite Explorer for Developmentを使用してメタデータを生成したりする必要がなくなります。この種のオプションの FIELD ステートメントでは、EXTERNALNAME パラメーターをコーディングすれば NAME パラメーターは必要ありません。

FIELD ステートメントは任意の順序で入力デックに置くことができますが、以下の場合は除きます。

- FIELD ステートメントは、関連する SEGM ステートメントの直後になければなりません。
- シーケンス・フィールドが定義されている場合には、常にシーケンス・フィールドを最初に置く必要があります。
- FIELD ステートメントで CASENAME パラメーターを指定した場合は、CASENAME パラメーターで指定した DFSCASE ステートメントの直後に FIELD ステートメントを置く必要があります。
- FIELD ステートメントで DEPENDSON パラメーターを指定した場合は、DEPENDSON パラメーターで指定した FIELD ステートメントの直後に FIELD ステートメントを置く必要があります。
- FIELD ステートメントで REDEFINES パラメーターを指定した場合は、REDEFINES パラメーターで指定した FIELD ステートメントの直後に FIELD ステートメントを置く必要があります。
- FIELD ステートメントで STARTAFTER パラメーターを指定した場合は、STARTAFTER パラメーターで指定した FIELD ステートメントの直後に FIELD ステートメントを置く必要があります。
- FIELD ステートメントで PARENT パラメーターを指定した場合は、PARENT パラメーターで指定した FIELD ステートメントの直後に FIELD ステートメントを置く必要があります。

1 つのデータベースに最大 1,000 個のフィールドを定義できます。

それぞれのセグメント・タイプの中に、NAME パラメーターを指定した FIELD セグメントを最大 255 個まで組み込むことができます。NAME パラメーターを使用

すると、SSA 検索にフィールドが使用可能になります。NAME パラメーターを省略した場合、1 つのセグメントに対して定義できる FIELD ステートメントの数に制限はありませんが、定義されたフィールドの数が 1000 フィールドというデータベースの最大値を超えてはなりません。ただし、NAME パラメーターを使用せずに定義したフィールドは、SSA で指定することができず、また、IMS によって検索することもできません。

セグメントの中のフィールドの定義はオーバーラップしていてもかまいません。例えば、以下の図に示されているように、セグメントの中の日付「フィールド」を、3 つの 2 バイト・フィールドとして定義し、これを 1 つの 6 バイト・フィールドとして定義することもできます。

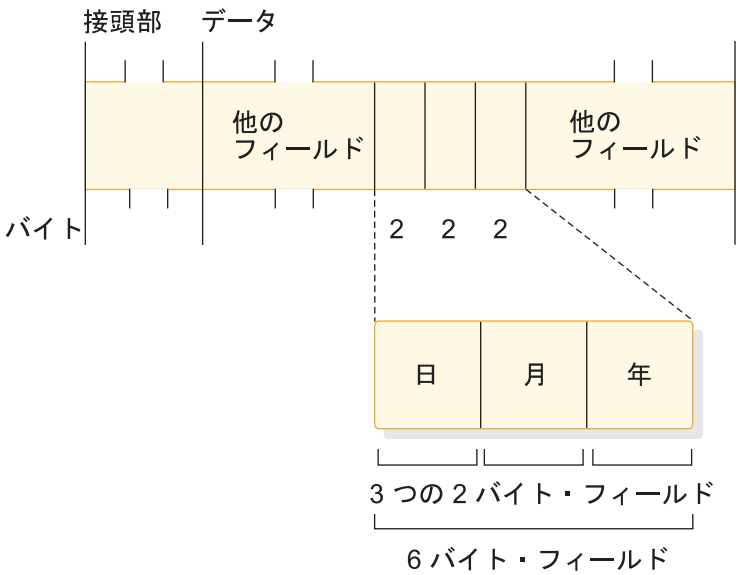


図 *238.* セグメント内の日付フィールドが、*3* つの *2* バイト・フィールド、かつ *1* つの *6* バイト・フィールドとして定義されている例

重複フィールド定義の手法を使用すると、アプリケーション・プログラムで同じデータに多様な方法でアクセスできます。同じデータに多様な方法でアクセスするには、各フィールドに別個の FIELD ステートメントをコーディングします。この例は、4 つの FIELD ステートメントをコーディングすることになります。示されている例では、6 バイトの日付全体を対象として 1 つコーディングし、日付の 2 バイト・フィールドに 1 つずつ (計 3 つ) コーディングしていることを示しています。

関連資料:

FIELD ステートメント (システム・ユーティリティー)

DFSMARSH ステートメントの概要

DFSMARSH ステートメントは、フィールド・データのマーシャル属性を定義します。

DFSMARSH ステートメントを使用して、フィールドに含まれているデータについて、以下のデータ・マーシャル属性を指定できます。

- ENCODING パラメーターで、フィールド内の文字データを定義するコード・ページまたは文字エンコードを指定できます。デフォルトは、EBCDIC コード・ページ Cp1047 です。
- フィールド・データを IMS がデータを物理的に保管するために使用するデータ・タイプからアプリケーション・プログラムが予期するデータ・タイプに変換する場合に、IMS が使用するデータ変換ルーチンを指定できます。IMS 提供のルーチンを INTERNALTYPECONVERTER パラメーターで指定するか、ユーザー提供のルーチンを USERTYPECONVERTER パラメーターで指定できます。
- ISSIGNED パラメーターで 10 進データ・タイプが符号付きかそうでないかを指定できます。
- PATTERN パラメーターで日付と時刻に使用するパターンを指定できます。
- PROPERTIES パラメーターで、ユーザー提供のデータ変換ルーチンが使用するプロパティーを指定できます。

DBD 生成ユーティリティーへの入力の中で、DFSMARSH ステートメントは、その適用対象である FIELD ステートメントの直後に置く必要があります。

関連資料:

DFSMAP ステートメント (システム・ユーティリティー)

LCHILD ステートメントの概説

LCHILD ステートメントは、副次索引、または 2 つのセグメント・タイプ間の論理関係を定義し、あるいは、HIDAM (または PHIDAM) 索引データベースと HIDAM (または PHIDAM) データベースの中のルート・セグメント・タイプとの間の関係を定義します。

LCHILD ステートメントは、この関係に関与するセグメントの SEGM、FIELD、または XDFLD ステートメントの直後に置きます。データベース当たり最高 255 の LCHILD ステートメントを定義できます。

制約事項: LCHILD ステートメントは PHIDAM データベースの 1 次索引用に指定することはできません。1 次索引は自動的に生成されるからです。

関連資料:

LCHILD ステートメント (システム・ユーティリティー)

XDFLD ステートメントの概説

XDFLD ステートメントは副次索引が存在する場合のみ使用されます。

このステートメントは、ターゲット・セグメントと関連しており、以下の事項を指定します。

- 索引付きフィールドの名前。
- ソース・セグメントの名前。ソース・セグメントは、ターゲット・セグメントと同じセグメントであってもターゲット・セグメントの従属セグメントであってもかまいません。
- ソース・セグメント内の、副次索引で使用されるデータを含んでいるフィールド。

セグメント当たり最高 32 の XDFLD ステートメントを定義できます。しかし、XDFLD ステートメントと FIELD ステートメントの合計数は、セグメント当たり 255 以内、またデータベース当たり 1000 を超えることはできません。

制約事項**:** CONST パラメーターは、HALDB データベースには使用できません。共用副次索引はサポートされていません。

関連資料:

XDFLD ステートメント (システム・ユーティリティー)

DFSMAP ステートメントの概要

DFSMAP ステートメントは、マップ・ケース・セットの制御フィールドを参照するマップ を定義します。

セット内の各マップ・ケースは、DFSMAP ステートメント内の NAME パラメーターで定義された名前によってマップを参照します。

マップは、DFSMAP ステートメントの DEPENDINGON パラメーターで指定されたフィールドの外部名によって制御フィールドを参照します。

関連タスク:

581 ページの『セグメントの代替フィールド・マップの定義』

関連資料:

DFSMAP ステートメント (システム・ユーティリティー)

DFSCASE ステートメントの概要

DFSCASE ステートメントは、セグメント内のフィールド・セットの条件付きマッピングである「マップ・ケース」を定義します。

マップ・ケースには名前と ID があります。

マップ・ケースを形成する各フィールドは、FIELD ステートメントの CASENAME パラメーターで所属先のマップ・ケースの名前を指定します。

マップ・ケースの ID は、CASEID パラメーターで指定されます。セグメント・インスタンス内では、ID は制御フィールドに挿入されて、どのマップ・ケースが使用中であるかを示します。

マップ・ケース ID は、Cp1047 文字データでも 16 進データでもかまいません。ID のデータ・タイプは、CASEIDTYPE パラメーターで指定されます。制御フィールドの長さは、マップ・ケース ID に選択したデータ・タイプとの互換性があることが必要です。

それぞれのマップ・ケースは 1 つのマップ に属します。マップは、DFSMAP ステートメントによって定義されます。マップは、セグメント・インスタンスにどのマップ・ケースを使用するかを決める制御フィールドを参照します。マップ・ケースは、所属先のマップを MAPNAME パラメーターで指定します。

関連資料:

DFSCASE ステートメント (システム・ユーティリティー)

DBDGEN ステートメントおよび END ステートメントの概説

各 DBD 生成入力デックの最後には、DBDGEN ステートメントと END ステートメントが 1 つずつ置かれます。

これらのステートメントは以下のことを指定します。

- DBD を定義するのに用いられるステートメント (DBDGEN) の終わり
- アセンブラーへの入力ステートメントの終わり (END)

関連資料:

DBDGEN ステートメント (システム・ユーティリティー)

PSBGEN ユーティリティーへの入力としてのプログラム仕様ブロックのコーディング

PSB とは、ある 1 つのアプリケーション・プログラムの特性、このプログラムによるデータベース内のセグメントとフィールドの使い方、このプログラムによる論理端末の使い方などを記述している一連のマクロ命令です。

PSB は、1 つ以上の PCB (プログラム連絡ブロック) で構成されています。 PCB は 2 つのタイプで構成されています。そのうち 1 つは、代替メッセージ宛先のために使用されるものであり、もう一方は、アプリケーション・アクセスと操作の定義のために使用されるものです。

PSB マクロ命令をコーディングした後、このマクロ命令は PSBGEN ユーティリティーへの入力として使用されます。このユーティリティーは、PSB 制御ブロックを生成し、それを IMS.PSBLIB ライブラリーに保管して、後でデータベースの処理に使用できるようにするマクロ・アセンブラーです。

以下の図は PSB 生成処理を示しています。

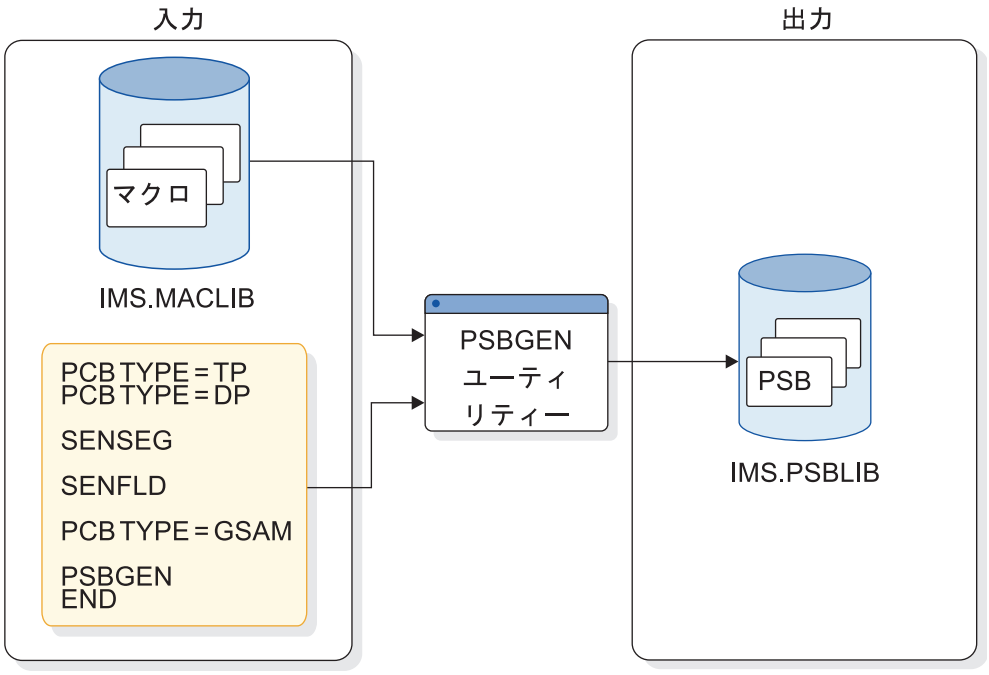


図 *239. PSB* 生成処理

以下の JCL は PSBGEN ユーティリティーへの入力として使用されるデックの構造を示しています。

```
//PSBGEN    JOB  MSGLEVEL=1
//                  EXEC        PSBGEN,MBR=APPLPGM1
//C.SYSIN        DD     *

    PCB TYPE=TP       出力メッセージ宛先用に必要
    PCB TYPE=DB       アプリケーション・プログラムがアクセスできる
                      データベースごとに必要

    SENSEG            アプリケーション・プログラムがアクセスできる
                       データベース内のセグメントごとに必要
    SENFLD            アプリケーション・プログラムがアクセスできる
                       セグメント内のフィールドごとに必要 (フィールド・レベル・
                       センシティビティーを指定する場合)
    PCB TYPE=GSAM

⋮
    PSBGEN            PSB 生成ごとに必要
    END               PSB 生成ごとに必要
/*
```

関連資料:

107 ページの『PSB セグメント・タイプのフォーマット』

103 ページの『PCB セグメント・タイプのフォーマット』

代替 PCB ステートメント

次の 2 種類の PCB ステートメントを入力デックの中に置くことができます。すなわち、代替 PCB ステートメントとデータベース PCB ステートメントです。

代替 PCB ステートメントは、メッセージの宛先とメッセージが入れられた場所とが異なる場合に、どこにメッセージを送ることができるか記述します。代替 PCB

ステートメントは入力デックの先頭になければなりません。代替 PCB の詳細については、「*IMS V13* システム管理」で説明しています。

データベース PCB ステートメント

データベース PCB ステートメントは、アプリケーション・プログラムがアクセスするデータベースの DBD を定義しています。

また、データベース PCB ステートメントは、アプリケーション・プログラムがデータベースのセグメントに対して行うことのできる操作のタイプ (get、insert、replace など) を定義します。データベースは物理データベースでも論理データベースでもかまいません。アプリケーション・プログラムがアクセスする各データベースごとに、別個のデータベース PCB ステートメントが 1 つずつ必要です。各 PSB 生成では、データベース PCB の最大数の 2500 から、入力デックで定義されている代替 PCB の数を引いた差だけ、データベース PCB を定義することができます。 PSB に適用される他の形のステートメントは、SENSEG、SENFLD、PSBGEN、および END です。

注意**:** PROCOPT 値で BMP アプリケーションによるデータベース内のセグメントの挿入、置換、または削除を許可する場合は、その BMP アプリケーションが、合計で 300 を超えるデータベースと HALDB 区画の更新を、変更をコミットすることなく行わないようにしてください。

論理的に関連付けられたデータベースや副次索引を含むすべての全機能データベース・タイプには、コミットされていない更新は、300 までという制限があります。HALDB データベースを設計する場合は、特に注意が必要です。コミットされていない更新についての 300 というデータベース制限に近づく最も一般的な理由は、HALDB データベース内の区画の数だからです。

SENSEG ステートメント

SENSEG ステートメントは、アプリケーション・プログラムがセンシティブになるデータベース内の 1 つのセグメント・タイプを定義します。

セグメント・タイプごとに、別個の SENSEG ステートメントが 1 つずつ必要です。これらのセグメントは、1 つのデータベースの中で物理的に存在することもできるし、複数の物理データベースから取り込むこともできます。あるアプリケーション・プログラムが、ルート・セグメントの下のあるセグメントに対してセンシティブである場合には、そのプログラムは、このルート・セグメントからセンシティブ・セグメントまでのパスにあるすべてのセグメントに対してもセンシティブである必要があります。例えば、以下の図で、あるアプリケーション・プログラムに対して D がセンシティブ・セグメントとして定義されていると、B と A もセンシティブ・セグメントとして定義されていなければなりません。

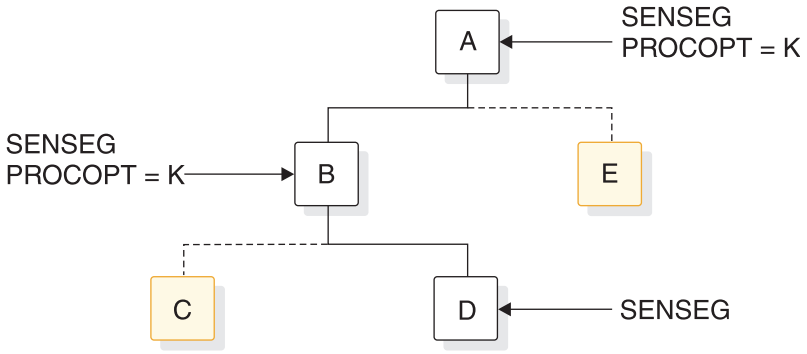


図 240. SENSEG の関係の例

アプリケーション・プログラムは、ユーザーが実際にセンシティブにしたいセグメントに至るパスにあるすべてのセグメントに対してセンシティブである必要があります。ただし、アプリケーション・プログラムをこれら他のセグメント内のセグメント・キーに対してのみセンシティブにすることができます。このオプションを使用すると、アプリケーション・プログラムは、センシティブ・セグメントに到達するために必要とするキー以外のセグメントにはアクセスできません。アプリケーション・プログラムをセグメントのセグメント・キーに対してのみセンシティブにするには、SENSEG ステートメントで PROCOPT=K を指定します。アプリケーション・プログラムは、セグメント内のセグメント・キー以外のフィールドにはアクセスできません。上の例では、アプリケーション・プログラムは、セグメント A とセグメント B のキーに対してセンシティブになりますが、A のデータと B のデータに対してはセンシティブになりません。

SENSEG ステートメントは、関連する PCB ステートメントの直後になければなりません。各 PSB 生成ごとに、最高 30000 の SENSEG ステートメントを定義できます。

関連資料:

111 ページの『SS セグメント・タイプのフォーマット』

SENFLD ステートメント

SENFLD ステートメントは、必ずフィールド・レベル・センシティビティーと並行して使用され、アプリケーション・プログラムがセンシティブになるセグメント・タイプのフィールドを定義します。

SENFLD ステートメントは、SENSEG ステートメントと共に使用すると、データの保護に役立ちます。各 SENFLD ステートメントは、関連のある SENSEG ステートメントの後ろに指定する必要があります。 1 つのセグメント・タイプ当たり最高 255 のセンシティブ・フィールドが定義でき、それぞれの PSB 生成ごとに最大 10000 のセンシティブ・フィールドを定義することができます。

PSBGEN ステートメント

このステートメントは、この PSB に名前を与えると共に、このアプリケーション・プログラムのさまざまな特性、例えば、このプログラムが書かれている言語、このプログラムが使用することのできる最大の入出力域のサイズなどを指定します。入力デックには、PSBGEN ステートメントを 1 つだけ置くことができます。

END ステートメント

各 PSB 生成入力デックの最後には END ステートメントが 1 つずつ置かれます。

END ステートメントは、アセンブラーへの入力ステートメントの終わりを指定します。

PSB ステートメントのコーディング方法の詳細と PSB の例は、「*IMS V13* システム・ユーティリティー」に記載されています。

アプリケーション制御ブロック (ACBGEN) の作成

システム内で使用する DBD ごと、および PSB ごとに、アプリケーション制御ブロック (ACB) を作成できます。このためには、ACB 保守ユーティリティーまたは ACB Generation and Catalog Populate ユーティリティー (DFS3UACB) のいずれかを実行します。

ユーティリティーは、入力として読み取る IMS.DBDLIB データ・セットおよび IMS.PSBLIB データ・セットのデータベース定義 (DBD) およびプログラム仕様ブロック (PSB) から ACB を生成します。生成された各 ACB は、*ACB* ライブラリー・メンバー として IMS.ACBLIB データ・セットに保管されます。

ACB を生成するためにどのユーティリティーを使用するか、および IMS カタログが使用可能であるかどうかに応じて、ACB 生成プロセスの中で、または後で IMS カタログにデータを設定することができます。

DFS3UACB ユーティリティーは、同じジョブ・ステップ内で ACB の生成と IMS カタログへのデータの設定を行います。これにより、IMS カタログは常に最新の ACB ライブラリーを使用した状態であることが保証されます。 ACB メンバーの生成だけを行う ACB 保守ユーティリティーを使用する場合は、後で IMS Catalog Populate ユーティリティー (DFS3PU00) を使用して IMS カタログにデータを設定できます。ただし、後から IMS カタログにデータを設定した場合、ACB ライブラリーが最新でない IMS カタログを使用して IMS を実行するリスクがあります。

推奨事項: システム内で IMS カタログが使用可能の場合は、DFS3UACB ユーティリティーを使用して、単一のジョブ・ステップで ACB メンバーの作成と IMS カタログへのデータの設定を行ってください。

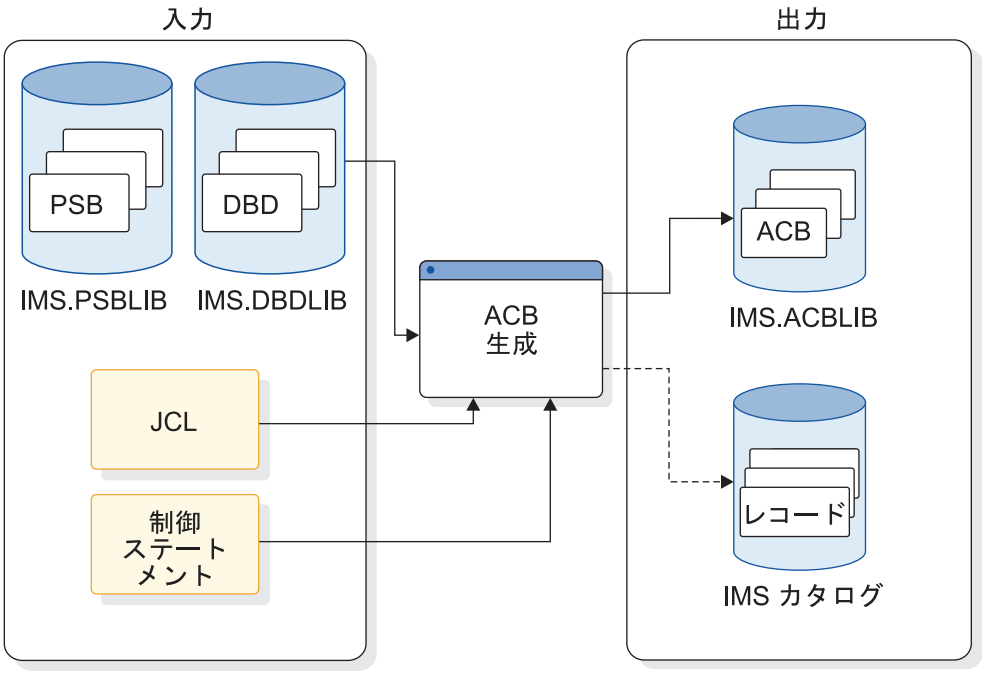


図 *241. ACB* 生成処理

ACB は、常駐であるか非常駐であるかに応じて、IMS 制御領域が初期設定されるとき、または ACB を必要とするアプリケーション・プログラムがスケジュールされるときに、IMS.ACBLIB データ・セットからオンライン用ストレージにロードされます。

オンライン・ストレージでは、PSB の ACB と DBD の ACB が PSB プールと DMB プールにそれぞれ分けて保管されます。これらのプールのストレージは、各タイプの ACB メンバーの数とサイズに基づくサイズを使用して割り振る必要があります。

ACB の作成元であるプログラムまたはデータベースがシステム定義時に常駐と定義されている場合、IMS は IMS 制御領域の初期設定時にその ACB を 31 ビットのストレージにロードします。常駐 ACB メンバーは、IMS がシャットダウンされるまでストレージに残ります。

ACB の作成元であるプログラムまたはデータベースがシステム定義時に常駐と定義されていない場合、IMS は、非常駐 ACB をその ACB を必要とするアプリケーション・プログラムがスケジュールされたときにのみ、31 ビットのストレージにロードします。非常駐 ACB メンバーは、非常駐 ACB 用に割り振られたストレージが使い果たされるまでストレージに残ります。ストレージが使い果たされると、IMS は、最も長い期間使用されていない ACB を除去することによってストレージを解放します。

非常駐 ACB メンバーを 64 ビットのストレージ・プールにキャッシュして、プログラム・スケジューリングのパフォーマンスを高め、31 ビットのストレージの使用量を減らすことも可能です。64 ビットのキャッシュを有効にすると、非常駐 ACB が 31 ビットのストレージ・プールに最初にロードされるときに、64 ビットのストレージ・プールにもキャッシュされます。その後、オンライン・ストレージから除

去された非常駐 ACB が、スケジュールに入れられたアプリケーション・プログラムで必要となった場合は、それらの ACB が DASD 上の IMS.ACBLIB データ・セットではなく 64 ビットのストレージから取得されます。

ACB を 64 ビットのストレージにキャッシュできるようにするには、DFSDFxxx PROCLIB メンバーで ACBIN64 パラメーターを指定します。64 ビットのストレージにキャッシュされている ACB の情報を表示するには、タイプ 2 コマンド QUERY POOL TYPE(ACBIN64) を使用します。

PSB プールと DMB プールのサイズは、ACB ライブラリー・メンバーの数とサイズによって決まります。PSB と DMB は 64 ビットのストレージ・プールに一緒に保管されるため、64 ビットのストレージ・プールのサイズは、非常駐の PSB と DMB をまとめてすべて保管できるだけの大きさにする必要があります。ACB メンバーのサイズは、ACB メンバーが作成されるときにアプリケーション制御ブロック (ACB) 保守ユーティリティーから報告されます。

バッチ環境で実行する場合、IMS は、ACB を動的に作成することも、または事前に作成された ACB を使用することもできます。ACB を動的に作成するには、バッチ始動プロシージャーの PARM パラメーターで DLI を指定する必要があります。事前に作成された ACB を使用するには、バッチ始動プロシージャーの PARM パラメーターで DBB を指定する必要があります。事前に作成された ACB は、始動プロシージャーで IMSACB*x* DD ステートメントによって指定される IMS.ACBLIB データ・セットに保管されています。IMS には、動的に作成された ACB を使用するバッチ・ジョブを実行する DLIBATCH プロシージャー、および事前に作成された ACB を使用するバッチ・ジョブを実行する DBBATCH プロシージャーが用意されています。これらのプロシージャーについては、「*IMS V13* システム定義」を参照してください。

バッチ・アプリケーション・プログラムの場合は、IMS で ACB の 64 ビットのキャッシュがサポートされません。

オンライン・アプリケーション・プログラムの場合は、ACB を事前に作成する必要があります。

GSAM DBD の場合は、ACB を事前に作成できません。ただし、GSAM データベースを参照する PSB の場合は、ACB を事前作成できます。

IMS.PSBLIB 内のすべての PSB について ACB を事前作成するか、特定の 1 つの PSB について ACB を事前作成するか、特定の DBD を参照するすべての PSB について ACB を事前作成することができます。事前に作成された ACB は、IMS.ACBLIB ライブラリーの中に保管されます。ACB が事前に作成されなければ、IMS.ACBLIB ライブラリーは使用されません。ACB が事前に作成されていて、アプリケーション・プログラムのスケジュールが決められる場合、このアプリケーション・プログラムの ACB が IMS.ACBLIB から直接、ストレージの中に読み込まれます。これは、アプリケーション・プログラムのスケジュールを決めるのに必要とされる時間が少なくて済むことを意味しています。また、事前に作成された ACB が使用される場合には、ストレージが少なくて済みます。

ACB がユーティリティーによって事前作成されると、ユーティリティーは PSB およびその PSB に関連付けられている DBD で使用されている名前にエラーがない

かどうかを検査します。エラーのある相互参照が見つかった場合、ユーティリティーは該当するエラー・メッセージを出力します。

IMS.ACBLIB は排他的に使用しなければなりません。このため、ACB を生成するユーティリティーは、アクティブな IMS システムに現在割り振られていない IMS.ACBLIB を使用する場合にのみ実行できます。また、IMS.ACBLIB は変更されるので、ACB を生成するユーティリティーの実行中、IMS.ACBLIB を他の目的に使用できません。

ACBLIB の「非アクティブ」コピーの中で ACB を変更または追加してから、オンライン変更機能を使用して、その変更済みまたは新規のメンバーをアクティブ IMS オンライン・システムで使用可能にすることができます。

DMB プール、PSB プール、および 64 ビットの非常駐 ACB ストレージ・プールの作成およびサイズ変更の追加情報については、以下を参照してください。

- *IMS V13* システム管理
- *IMS V13* システム定義

関連概念:

ACBLIB データ・セットの割り振り (システム定義)

関連タスク:

849 ページの『オンライン HALDB データベースを変更するためのステップ』

872 ページの『オンライン変更機能を使用したデータベース変更の活動化』

関連資料:

アプリケーション制御ブロック保守ユーティリティー (システム・ユーティリティー)

データベース記述 (DBD) 生成ユーティリティー (システム・ユーティリティー)

プログラム仕様ブロック (PSB) 生成ユーティリティー (システム・ユーティリティー)

ACB Generation and Catalog Populate ユーティリティー (DFS3UACB) (システム・ユーティリティー)

生成ユーティリティーに対する DBD および PSB メタデータの定義

IMS カタログを使用する場合は、データ・タイプ、データ構造、アクセス方式など、さまざまな項目に関するメタデータがカタログに保管されます。このメタデータは、データベースとアプリケーション・プログラム・ビューを IMS に対して定義する際に作成する、データベース記述子 (DBD) とプログラム仕様ブロック (PSB) から派生したものです。

一部のメタデータは、物理データベースを記述したデータベース・メタデータ です。それ以外のメタデータはアプリケーション・プログラム・メタデータ であり、アプリケーション・プログラムによるデータベース内のデータの表示方法または使

用方法を記述しています。DBD にコーディングするアプリケーション・プログラム・メタデータの多くは、アプリケーション・プログラムや、COBOL コピーブックのような成果物の要件に由来します。

以下のエレメントは、アプリケーション・プログラム・メタデータによって記述される、DBD 内にコーディングできるものの例です。

- セグメントおよびフィールドの外部名。これは、セグメントまたはフィールドを定義する際に EXTERNALNAME パラメーターで指定します。
- 配列や構造などの複雑なデータ・エレメント
- フィールド・データ・タイプおよびデータ・マーシャル特性
- 文字エンコード
- セグメントの代替フィールド・マップ

アプリケーション・プログラム・メタデータによって記述されたエレメントを DBD にコーディングすると、アプリケーション・プログラムのデータ要件が単一の信頼できる場所、つまり IMS カタログに統合されます。 IMS カタログが使用可能の場合、新しいアプリケーション・プログラムを開発するとき、または既存のデータベースやアプリケーション・プログラムに対する変更の影響を評価するときに、IMS カタログ内のメタデータを照会し、分析することができます。

IMS に対してメタデータを定義するには、DBD および PSB 生成マクロ・ステートメントの中にパラメーターをコーディングするか、IMS Enterprise Suite Explorer for Development などのツールを使用します。

関連資料:

データベース記述 (DBD) 生成ユーティリティー (システム・ユーティリティー)

プログラム仕様ブロック (PSB) 生成ユーティリティー (システム・ユーティリティー)

アプリケーション・プログラムのデータ・タイプの指定

DBD 生成ユーティリティーに対してフィールドを定義する FIELD ステートメントで DATATYPE パラメーターを指定することにより、所定のフィールドにアプリケーション・プログラムが予期するデータ・タイプを指定できます。

DATATYPE キーワードが指定されると、データ・タイプ・コンバーターは IMS が返したフィールド・データを、IMS が物理データの保管に使用するバイナリー・データ・タイプから DATATYPE パラメーターで指定されたデータ・タイプに変換します。

それぞれの DATATYPE の指定に対し、IMS は自動的に互換性のある IMS 提供のデータ・タイプ・コンバーターを選択します。IMS によって選択されたデータ・タイプ・コンバーターがユーザーのニーズに正確に適合しない場合、ユーザーは自身でデータ・タイプ・コンバーターを選択するか、独自のカスタムのデータ・タイプ・コンバーターを提供することができます。 IMS 提供のデータ・タイプ・コンバーターは INTERNALTYPECONVERTER パラメーターで指定し、カスタムのデータ・タイプ・コンバーターは USERTYPECONVERTER パラメーターで指定します。どちらのパラメーターも、DFSMARSH ステートメントで指定します。

データ・タイプを指定するときは、フィールドのバイト長がそのデータ・タイプをサポートしていることを確認する必要があります。一部のデータ・タイプは特定のフィールド長を必要とします。例えば、DATATYPE=LONG を指定する場合は、BYTES=8 を指定することにより、フィールドの長さを 8 バイトとして定義する必要があります。

データ・タイプを指定しなかった場合、アプリケーション・プログラムに返されるデフォルトのデータ・タイプは、TYPE パラメーターの値によって決まります。TYPE=C が指定されるかデフォルトで選択された場合、デフォルトのアプリケーション・データ・タイプは CHAR です。それ以外の TYPE パラメーターを指定した場合、デフォルトのアプリケーション・データ・タイプは、すべて BINARY です。

DFSMARSH ステートメントで追加パラメーターを指定すると、文字エンコードや日時フォーマット、10 進タイプなど、データ・タイプ選択の追加特性を定義できます。

DFSMARSH ステートメントの例

以下の一連の例は、さまざまな DATATYPE およびタイプ・コンバーター指定に可能な DFSMARSH ステートメントの使用方法を示したものです。

DATATYPE=DATE:

```
FIELD    EXTERNALNAME=XDATE,
         BYTES=8,
         START=84,
         DATATYPE=DATE
DFSMARSH ENCODING=Cp1047,
         INTERNALTYPECONVERTER=CHAR,
         PATTERN='MMddyyyy'
```

DATATYPE=TIME:

```
FIELD    EXTERNALNAME=XTIME,
         BYTES=6,
         START=92,
         DATATYPE=TIME
DFSMARSH ENCODING=Cp1047,
         INTERNALTYPECONVERTER=CHAR,
         PATTERN='HHmmss'
```

DATATYPE=TIMESTAMP:

```
FIELD    EXTERNALNAME=XTIMESTAMP,
         BYTES=16,
         START=84,
         DATATYPE=TIMESTAMP
DFSMARSH ENCODING=Cp1047,
         INTERNALTYPECONVERTER=CHAR,
         PATTERN='MMddyyyyHHmmssff'
```

DATATYPE=ZONEDDECIMAL:

```
FIELD NAME=ORDPRICE,
         BYTES=10,
         START=21,
         DATATYPE=DECIMAL(10,2)
DFSMARSH INTERNALTYPECONVERTER=ZONEDDECIMAL,
         ISSIGNED=Y
```

DATATYPE=PACKEDDECIMAL:

```
FIELD    EXTERNALNAME=XPACKEDDEC1,
         BYTES=4,
         START=60,
         DATATYPE=DECIMAL(7,2)
DFSMARSH INTERNALTYPECONVERTER=PACKEDDECIMAL,
         ISSIGNED=Y
```

USERTYPECONVERTER=:

```
FIELD    EXTERNALNAME=PACKEDDATEFIELD,
         BYTES=5,
         START=40,
         DATATYPE=DATE
DFSMARSH USERTYPECONVERTER=class://com.ibm.ims.dli.types.PackedDateConverter,
         PROPERTIES=(ZONE=PACIFIC,DAYLIGHTSAVINGS=TRUE)
```

関連資料:

- FIELD ステートメント (システム・ユーティリティー)
- DFSMAP ステートメント (システム・ユーティリティー)

DBD ソース・ステートメントでの配列の定義

アプリケーション・プログラムで使用する配列を IMS に対して定義するには、それらの配列を DBD ソース・ステートメントの中にコーディングします。

配列 とは、繰り返されるエレメント を格納するデータ構造のことです。配列とそのエレメントを定義するには、DBD 生成ユーティリティーの FIELD ステートメントを使用します。

配列は、IMS に対して ARRAY というデータ・タイプのフィールドとして定義されます。配列のエレメントは、IMS に対して、繰り返される一連のフィールドとして定義されます。1 つの配列エレメント内のすべてのフィールドは、PARENT パラメーターで配列フィールドの外部名を指定します。

IMS に対して定義できる配列は、静的配列 または動的配列 です。静的配列には、常に、あるセグメント・タイプのすべてのインスタンス内に同じ数のエレメントが入っています。動的配列に入っているエレメントの数は、セグメント・タイプのインスタンスごとに異なる可能性があります。使用する配列型は、アプリケーション・プログラムの要件によって決まります。

配列エレメントが配列内で繰り返される回数は、MINOCCURS パラメーターと MAXOCCURS パラメーター、および動的配列に限ってはセグメント内の別個の制御フィールドで指定された数値によって決まります。

静的配列の場合、配列エレメントの数は MINOCCURS パラメーターと MAXOCCURS パラメーターの両方で指定されます。それぞれのパラメーターで指定される数値は、同じであることが必要です。

動的配列の場合、MINOCCURS パラメーターで指定される値は、所定のセグメント・インスタンス内で発生する可能性があるエレメントの最小数を定義します。MAXOCCURS パラメーターで指定される値は、所定のすべてのセグメント・インスタンス内で発生する可能性があるエレメントの最大数を定義します。あるセグメ

ント・インスタンスにおける実際のエレメントの発生回数は、配列の FIELD ステートメントの DEPENDSON パラメーターで名前が指定されている制御フィールドで指定された値によって決まります。

配列として定義されたフィールドは、外部名だけをサポートします。したがって、SSA の中でそれらのフィールドを指定することはできません。

関連資料:

FIELD ステートメント (システム・ユーティリティー)

IMS に対する静的配列の定義

静的配列は、あるセグメント・タイプのすべてのインスタンス内に固定数の配列エレメントを持っています。

静的配列は、DBD 生成ユーティリティーの入力制御ステートメントの中に FIELD ステートメントをコーディングすることによって定義されます。以下の手順では、対応する SEGM、DBD、およびその他のユーティリティー制御ステートメントが正しくコーディングされていることを想定しています。

1. 静的配列の FIELD ステートメントをコーディングします。
   a. EXTERNALNAME パラメーターで配列の外部名を指定します。配列は NAME パラメーターをサポートしていません。
   b. DATATYPE=ARRAY を指定します。
   c. MINOCCURS パラメーターおよび MAXOCCURS パラメーターで配列のエレメントの数を指定します。 両方のパラメーターで指定される値は、同じであることが必要です。
   d. BYTES パラメーターで配列バイト・サイズを指定します。 指定するバイト・サイズは、配列と配列エレメントを形成するすべてのフィールドのバイト・サイズの合計以上であることが必要です。
2. 繰り返される配列エレメントを形成している各フィールドの FIELD ステートメントをコーディングします。 配列エレメントに含まれる各フィールドの FIELD ステートメントで、以下を指定します。
   a. EXTERNALNAME パラメーターでフィールドの外部名を指定します。配列は NAME パラメーターをサポートしていません。
   b. RELSTART パラメーターで、配列エレメント内のフィールドの開始バイト・オフセットを指定します。
   c. PARENT パラメーターで配列の外部名を指定します。
   d. BYTES パラメーターでフィールドのバイト・サイズを指定します。

以下の FIELD ステートメントは、静的配列定義の例を示しています。この例では、最初の FIELD ステートメントで配列を定義しています。この配列の名前は COURSES であり、MINOCCURS パラメーターと MAXOCCURS パラメーターで指定されているように、繰り返されるエレメントの 8 つのインスタンスを含んでいます。繰り返される配列エレメントは、最後の 3 つの FIELD ステートメントで定義されています。それぞれの配列エレメントは合計 64 バイトの 3 つのフィールドから形成され、この配列は 8 つの配列エレメントから形成されているため、この配列には最小でも 512 バイト必要です。

配列はセグメント内のバイト 5 から始まり、最初の配列エレメント内の最初のフィールドは、配列のバイト 1 から始まります。後続の配列エレメントは、バイト 65、129、193 などから始まります。

```
FIELD EXTERNALNAME=COURSES,DATATYPE=ARRAY,START=5,                  X
          BYTES=512,MINOCCURS=8,MAXOCCURS=8
 FIELD EXTERNALNAME=COURSE_ID,RELSTART=1,BYTES=8,PARENT=COURSES
 FIELD EXTERNALNAME=COURSE_TITLE,RELSTART=9,BYTES=48,              X
          PARENT=COURSES
 FIELD EXTERNALNAME=INSTRUCTOR_ID,RELSTART=57,BYTES=8,             X
          PARENT=COURSES
```

上記の配列定義は、次のような COBOL コピーブックの例からの抜粋に基づいています。

```
******* COPYBOOK for STUDENT/COURSES (STATIC)01   STUDENT.
        20   COURSES OCCURS 8 TIMES.
          25   COURSE_ID   PIC 9(8).
          25   COURSE_TITLE  PIC X(48).
          25   INSTRUCTOR_ID PIC 9(8).
```

関連資料:

FIELD ステートメント (システム・ユーティリティー)

IMS に対する動的配列の定義

動的配列とは、繰り返される配列エレメントの数がセグメント・タイプのインスタンスごとに変わる可能性がある配列のことです。

動的配列は、DBD 生成ユーティリティーの入力制御ステートメントの中に FIELD ステートメントをコーディングすることによって定義されます。以下の手順では、対応する DBD、DATASET、SEGM、およびその他のユーティリティー制御ステートメントが正しくコーディングされていることを想定しています。

1. セグメント・インスタンス内の配列の配列エレメント数が入る制御フィールドを定義します。
2. 次のように FIELD ステートメントをコーディングして、配列を定義します。
   a. EXTERNALNAME パラメーターで配列の外部名を指定します。配列は NAME パラメーターをサポートしていません。
   b. DATATYPE=ARRAY を指定します。
   c. MINOCCURS パラメーターで、配列が 1 つのセグメント・インスタンス内に持つことができる最小エレメント数を指定します。 MINOCCURS パラメーターで指定する値は MAXOCCURS パラメーターで指定する値より小さいことが必要です。
   d. MAXOCCURS パラメーターで、配列が 1 つのセグメント・インスタンス内に持つことができる最大エレメント数を指定します。 MAXOCCURS パラメーターで指定する値は、MINOCCURS パラメーターで指定する値より大きいことが必要です。
   e. MAXBYTES パラメーターで配列に可能な最大バイト・サイズを指定します。 指定するバイト・サイズは、最大配列エレメント数を持つ配列が形成するすべてのフィールドのバイト数を合計した値以上であることが必要です。MAXBYTES パラメーターは、静的配列のサイズを定義するために使用される BYTES パラメーターと同時に使用することはできません。
   f. DEPENDSON パラメーターで制御フィールドの名前を指定します。

3. 繰り返される配列エレメントを定義するために、エレメントに含まれるフィールドごとに FIELD ステートメントをコーディングします。 配列エレメントに含まれる各フィールドの FIELD ステートメントで、以下を指定します。
   a. EXTERNALNAME パラメーターでフィールドの外部名を指定します。配列エレメントは NAME パラメーターをサポートしていません。
   b. RELSTART パラメーターで、配列エレメント内のフィールドの開始バイト・オフセットを指定します。
   c. PARENT パラメーターで配列の外部名を指定します。
   d. BYTES パラメーターでフィールドのバイト・サイズを指定します。 フィールドに動的配列が含まれる場合は、代わりに MAXBYTES パラメーターを使用してください。

次の例は、動的配列の FIELD ステートメント定義を示しています。

この例で、最初の FIELD ステートメントは、配列のインスタンス内の配列エレメント数を指定する制御フィールドを定義しています。

2 番目の FIELD ステートメントは、配列を定義しています。この配列の名前は BOOKS であり、MINOCCURS パラメーターと MAXOCCURS パラメーターで指定されているように、1 つから 5 つまでの配列エレメントを含むことができます。フィールド NUMOF_BKS は、配列の 1 インスタンスに含まれる配列エレメントの数を決定します。

繰り返される配列エレメントは、最後の 3 つの FIELD ステートメントで定義されています。配列エレメントは、合計 40 バイトになる 3 つのフィールドから形成され、最大 5 回繰り返すことができるため、少なくとも 200 バイトの MAXBYTES 値を指定して定義する必要があります。

配列はセグメント内のバイト 5 から始まり、最初の配列エレメント内の最初のフィールドは、配列のバイト 1 から始まります。後続の配列エレメントは、配列のバイト 41、81 などから始まります。

```
FIELD EXTERNALNAME=NUMOF_BKS,DATATYPE=INT,START=1,BYTES=4
FIELD EXTERNALNAME=BOOKS,DATATYPE=ARRAY,START=5,MAXBYTES=200        X
            MINOCCURS=1,MAXOCCURS=5,DEPENDSON=NUMOF_BKS
  FIELD EXTERNALNAME=ISBN,RELSTART=1,BYTES=10,PARENT=BOOKS
  FIELD EXTERNALNAME=BOOK_TITLE,RELSTART=11,BYTES=22,PARENT=BOOKS
  FIELD EXTERNALNAME=RETURN_DATE,RELSTART=33,BYTES=8,PARENT=BOOKS
```

上記の配列定義は、次のような COBOL コピーブックの例からの抜粋に基づいています。

```
******* COPYBOOK for STUDENT/COURSES (DYNAMIC)01 STUDENT.
       20   NUMOF-BKS      PIC 9(4) COMP.
       20   BOOKS OCCURS 1 TO 5 TIMES DEPENDING ON NUMOF-BKS.
       30   ISBN         PIC X(10).
       30   BOOK-TITLE    PIC X(22).
       30   RETURN-DATE  PIC 9(8).
```

関連資料:

FIELD ステートメント (システム・ユーティリティー)

DBD ソース・ステートメントでのデータ構造の定義

アプリケーション・プログラムによって使用されるデータ構造を DBD ソース・ステートメントで定義し、アプリケーションがデータを処理するのと同じ方法で IMS がデータを保管できるようにしてください。

構造とは、関連するデータ・エレメントの固定のセットです。配列と異なり、構造のエレメントは繰り返されません。

構造は、STRUCT というデータ・タイプのフィールドとして IMS に対して定義されます。構造のエレメントは、その構造を親として指定したフィールドです。

構造には、名前、外部名、またはその両方を定義できます。名前を定義すると、その構造フィールドの名前を SSA の中で指定できます。ただし、構造のエレメントが動的配列である場合は、外部名だけを指定でき、IMS 名を指定できません。動的配列、および動的配列を含んでいるすべての構造は IMS で検索できず、したがって SSA に含めることはできません。

1. 構造は、以下の属性を指定する FIELD ステートメントをコーディングすることによって定義します。
   a. フィールド名 (NAME パラメーター、または構造に動的配列が含まれている場合は EXTERNALNAME パラメーターで指定)。
   b. DATATYPE=STRUCT
   c. 構造に動的配列が含まれている場合は、以下を指定します。
      - 構造の最大バイト・サイズ (MAXBYTES パラメーターで指定)。構造の最大バイト・サイズには、動的配列の MAXBYTES 値と、構造に含まれるその他の全エレメントのバイト・サイズが含まれている必要があります。
2. 構造の各エレメントを定義するために、PARENT パラメーターで構造の名前または外部名を指定する FIELD ステートメントをコーディングします。

以下の FIELD ステートメントは、構造定義の例を示しています。この例では、最初の FIELD ステートメントで構造を定義しています。構造の名前は ADDRESS_INFO です。構造のエレメントは、直後の 3 つの FIELD ステートメントによって定義されており、これらのステートメントはすべて、PARENT パラメーターで ADDRESS_INFO を指定しています。この構造は合計 45 バイトになる 3 つのフィールドを含んでいるため、最小でも 45 バイトを必要とします。

この構造とその最初のエレメントは、セグメント内のバイト 5 から始まります。構造内の後続のエレメントの開始位置も、セグメントの先頭からの相対位置として計算されます。

```
FIELD NAME=ADDRESS_INFO,DATATYPE=STRUCT,START=5,BYTES=45
FIELD NAME=CITY,DATATYPE=CHAR,START=5,BYTES=15,PARENT=ADDRESS_INFO
FIELD NAME=STREET,DATATYPE=CHAR,START=21,BYTES=25,PARENT=ADDRESS_INFO
FIELD NAME=ZIP,DATATYPE=CHAR,START=46,BYTES=5,PARENT=ADDRESS_INFO
```

上記の配列定義は、次のような COBOL コピーブックの例からの抜粋に基づいています。

```
02 ADDRESS-INFO.
04 CITY PIC X(15).
04 STREET PIC X(25).
04 ZIP PIC X(5).
```

関連資料:

FIELD ステートメント (システム・ユーティリティー)

フィールドの再定義

COBOL など、一部のアプリケーション・プログラミング言語では、アプリケーション・プログラムでフィールドをさまざまな目的のために再定義できます。IMS では、それらの再定義されたフィールドに対応する定義がデータベース記述 (DBD) の中に組み込まれていないと、フィールドの再定義が認識されません。

1 つのフィールドを 1 つ以上のフィールドとして再定義できますが、再定義でのフィールドの長さの合計は、再定義されるフィールドの長さ以下でなければなりません。

1 つのフィールドを複数のフィールドとして再定義する場合は、再定義の複数のフィールドを入れるために、構造フィールドを使用します。

DATATYPE=ARRAY が指定されているフィールドを再定義することはできません。

次の例は、DBD ソース・ステートメントでの 1 対 1 のフィールド再定義を示しています。FLD2 は FLD1 を再定義しています。FLD0 と FLD3 は再定義の例の一部ではなく、この例をコンテキスト内で示すために含まれているにすぎません。

```
FIELD NAME=FLD0,START=1,BYTES=4
FIELD NAME=FLD1,START=5,BYTES=15
FIELD NAME=FLD2,REDEFINES=FLD1,START=5,BYTES=15
FIELD NAME=FLD3,START=20,BYTES=6
```

次に示す例では、COBOL プログラミング言語で 1 つのフィールドが複数のフィールドとして再定義されています。FLD2 は、FLD1 を 2 つのフィールド FLD3 および FLD4 として再定義しています。FLD0 と FLD5 は再定義の例の一部ではなく、この例をコンテキスト内で示すために含まれいるにすぎませんが、再定義でのフィールドの相対的な開始位置を例示するのに役立っています。

```
*     01  FLD0              PIC X(4).
*     01  FLD1              PIC X(15).
*     01  FLD2 REDEFINES FLD1.
*       01 FLD3   PIC X(7).
*       01 FLD4   PIC X(8).
*     01  FLD5              PIC X(1).
```

次の例の FIELD ステートメントは、前述のコピーブックに対応する、DBD ソース内の FIELD ステートメント定義を示しています。この例で、FLD2 は FLD1 を再定義しており、FLD1 と同じバイト長です。FLD2 は DATATYPE=STRUCT を指定しています。FLD3 および FLD4 は、どちらも PARENT=FLD2 を指定しています。FLD3 と FLD4 の BYTES パラメーターの合計は、FLD1 と FLD2 のどちらの BYTES 値にも等しくなっています。

```
FIELD NAME=FLD0,START=1,BYTES=4
FIELD NAME=FLD1,START=5,BYTES=15
FIELD NAME=FLD2,REDEFINES=FLD1,START=5,BYTES=15,
```

```
        DATATYPE=STRUCT
    FIELD NAME=FLD3,START=5,BYTES=7,PARENT=FLD2
    FIELD NAME=FLD4,START=12,BYTES=8,PARENT=FLD2
FIELD NAME=FLD5,START=20,BYTES=1
```

関連資料:

FIELD ステートメント (システム・ユーティリティー)

セグメントの代替フィールド・マップの定義

セグメント定義の中で、単一のバイト・シーケンスに複数の代替フィールド・マップを定義できます。データベース内にセグメントのインスタンスが作成されると、アプリケーション・プログラムはフィールド・マップの 1 つを選択してデータをマップし、制御フィールドにフィールド・マップの ID を挿入することにより、有効なフィールド・マップを指示します。

アプリケーション・プログラムは、フィールド・マッピングを使用しているセグメント・インスタンス内のバイト・シーケンスにアクセスするとき、制御フィールドを評価して、どのフィールド・マップが有効であるかを判別する必要があります。

セグメント定義内の各フィールド・マップ、つまりマップ・ケースは、DFSCASE ステートメントをコーディングすることにより、IMS に対して定義されます。フィールド・マップを形成する各フィールドは、CASENAME パラメーターでマップ・ケース名を指定する FIELD ステートメントによって定義されます。

1 つのセグメント内で同じバイト・シーケンスをマップする一連のマップ・ケースは、同じ制御フィールドを共用します。セグメント定義の中で、DFSMAP ステートメントは関連する一連のマップ・ケースを制御フィールドにリンクします。マップ・ケース定義では、DFSCASE ステートメント内の MAPNAME パラメーターで DFSMAP ステートメントの名前を指定する必要があります。

マップに含まれるフィールドの FIELD ステートメントは、所属先のマップを定義している DFSCASE ステートメントの後ろに指定する必要があります。マップ内の各フィールドは、CASENAME パラメーターでマップ名を指定する必要があります。

セグメント定義の中でバイト・シーケンスの代替フィールド・マップを定義するには、以下のようにします。

1. セグメント・インスタンス内で、どのマップ・ケースが有効であるかを決定するケース ID を入れる制御フィールドを定義します。 ケース ID は、文字値でも 16 進値でもかまいません。制御フィールド定義の中で指定するデータ・タイプとバイト長は、マップ・ケースの定義で指定したケース ID のデータ・タイプと長さに合わせる必要があります。
2. マップ・ケースの集合をグループ化する DFSMAP ステートメントを定義します。 制御フィールドの外部名は、DEPENDINGON パラメーターで指定する必要があります。
3. 各マップ・ケースの DFSCASE ステートメントを定義します。 制御フィールドの長さは、CASEID パラメーターで指定する値の長さとデータ・タイプをサポートしている必要があります。

4. 各マップ・ケースに属するフィールドの FIELD ステートメントを定義します。各フィールドが属するマップ・ケースの名前を CASENAME パラメーターで指定します。

マッピング例: DFSMAP および DFSCASE

以下の例は、自動車保険契約、住宅保険契約、船舶保険契約の 3 つの異なるタイプの保険契約に関するデータの保管に使用される単一のセグメントを定義するために、DBD ソース内でマッピングを使用する方法を示しています。それぞれの保険契約タイプごとに、そのタイプに固有の情報を保持するための異なるフィールドが必要です。

DBD ソースでは、各保険契約タイプ用のフィールドが、異なる DFSCASE ステートメントによってマップされています。この例の 3 つのマップ・ケースには、それぞれ AUTOMAP、HOMEMAP、BOATMAP の名前が付けられています。任意の DFSCASE ステートメントによって定義されたマップを構成するフィールドは、FIELD ステートメント内の CASENAME パラメーターでそれぞれが属している DFSCASE ステートメントの名前を指定しています。DFSCASE ステートメントは、セグメント CUSTOMERPOLICY 内の DFSMAP ステートメント POLICYMAPS によってグループ分けされます。

各マップ・ケースの CASEID パラメーターで指定されている値は、そのマップ・ケースを一意的に識別し、制御フィールド値の役割を果たします。セグメント・インスタンスが最初にデータベースに挿入される時点で、そのセグメント・インスタンスが使用するマップ・ケースの ID が制御フィールドに挿入されます。この例では、制御フィールドの名前は POLICYTYPE です。実行時には、アプリケーション・プログラムがデータベースからセグメントを取り出す際に制御フィールド値を評価して、フィールドの正しいマッピングを判別しなければなりません。

```
  DBD       NAME=POLICYDB,                                          C
                ENCODING=CP1047,                                    C
                ACCESS=(DEDB),                                      C
                RMNAME=(RMOD3),                                     C
                PASSWD=NO
       AREA     DD1=PLCYAR01,                                       C
                DEVICE=3330,                                        C
                SIZE=(2048),                                        C
                UOW=(15,10),                                        C
                ROOT=(10,5),                                        C
                REMARKS='AREA NUMBER 1 FOR POLICYDB DATABASE'
    SEGM        NAME=CUSTOMER,                                      C
                PARENT=0,                                           C
                BYTES=(390,20)
      FIELD     NAME=(CUSTKEY,SEQ,U),                               C
                BYTES=12,                                           C
                START=1,                                            C
                TYPE=C
    SEGM    NAME=POLICY,                                            C
                EXTERNALNAME=CUSTOMERPOLICY,                        C
                ENCODING=CP1047,                                    C
                PARENT=CUSTOMER,                                    C
                BYTES=(900),                                        C
                TYPE=DIR,                                           C
                RULES=(LLL,HERE)
*************************************************************************
*        CONTROL FIELD:
*************************************************************************
       FIELD    EXTERNALNAME=POLICYTYPE,                            C
```

```
               BYTES=4,                                                C
               START=1,                                                C
               DATATYPE=CHAR
*********************************************************************
*        DFSMAP STATEMENT:
*********************************************************************
      DFSMAP   NAME=POLICYMAPS,                                        C
               DEPENDINGON=POLICYTYPE
*********************************************************************
*        DFSCASE STATEMENT 1:
*********************************************************************
      DFSCASE  NAME=AUTOMAP,                                           C
               CASEID=AUTO,                                            C
               CASEIDTYPE=C,                                           C
               MAPNAME=POLICYMAPS,                                     C
               REMARKS='DEFINES THE FIELDS OF AN AUTO INSURANCE POLICY'
      FIELD    EXTERNALNAME=AUTOMAKE,                                  C
               CASENAME=AUTOMAP,                                       C
               BYTES=15,                                               C
               START=5,                                                C
               DATATYPE=CHAR
      FIELD    EXTERNALNAME=MODEL,                                     C
               CASENAME=AUTOMAP,                                       C
               BYTES=15,                                               C
               START=20,                                               C
               DATATYPE=CHAR
      FIELD    EXTERNALNAME=YEAR,                                      C
               CASENAME=AUTOMAP,                                       C
               BYTES=4,                                                C
               START=35,                                               C
               DATATYPE=CHAR
*********************************************************************
*        DFSCASE STATEMENT 2:
*********************************************************************
      DFSCASE  NAME=HOMEMAP,                                           C
               CASEID=HOME,                                            C
               CASEIDTYPE=C,                                           C
               MAPNAME=POLICYMAPS,                                     C
               REMARKS='DEFINES THE FIELDS OF A HOME INSURANCE POLICY'
      FIELD    EXTERNALNAME=DWELLING_TYPE,                             C
               CASENAME=HOMEMAP,                                       C
               BYTES=20,                                               C
               START=5,                                                C
               DATATYPE=CHAR
      FIELD    EXTERNALNAME=ROOMS,                                     C
               CASENAME=HOMEMAP,                                       C
               BYTES=5,                                                C
               START=25,                                               C
               DATATYPE=CHAR
      FIELD    EXTERNALNAME=SQ_FOOT,                                   C
               CASENAME=HOMEMAP,                                       C
               BYTES=6,                                                C
               START=30,                                               C
               DATATYPE=CHAR
*********************************************************************
*        DFSCASE STATEMENT 3:
*********************************************************************
      DFSCASE  NAME=BOATMAP,                                           C
               CASEID=BOAT,                                            C
               CASEIDTYPE=C,                                           C
               MAPNAME=POLICYMAPS,                                     C
               REMARKS='DEFINES THE FIELDS OF A BOAT INSURANCE POLICY'
      FIELD    EXTERNALNAME=CLASS,                                     C
               CASENAME=BOATMAP,                                       C
               BYTES=10,                                               C
               START=5,                                                C
               DATATYPE=CHAR
```

```
      FIELD    EXTERNALNAME=LENGTH,                                  C
               CASENAME=BOATMAP,                                     C
               BYTES=6,                                              C
               START=15,                                             C
               DATATYPE=CHAR
      FIELD    EXTERNALNAME=BOATMAKE,                                C
               CASENAME=BOATMAP,                                     C
               BYTES=10,                                             C
               START=21,                                             C
               DATATYPE=CHAR
   DBDGEN
   FINISH
   END
```

関連概念:

564 ページの『DFSMAP ステートメントの概要』

関連資料:

FIELD ステートメント (システム・ユーティリティー)

DFSMAP ステートメント (システム・ユーティリティー)

DFSCASE ステートメント (システム・ユーティリティー)

HALDB 設計のインプリメント

HALDB データベースを作成するには、2 段階の処理を実行します。まず DBDGEN ユーティリティーを実行し、次にデータベースとその区画を DBRC に対して定義します。

HALDB データベースとその区画を DBRC に対して定義するには、DBRC バッチ・コマンドまたは区画定義ユーティリティーを使用します。このトピックでは、区画定義ユーティリティーを使用した HALDB データベースの定義、および間接リスト・データ・セット (ILDS) の割り振りについて説明します。

関連概念:

731 ページの『HALDB データベースの再編成』

関連タスク:

889 ページの『HALDB データベースの変更』

関連資料:

HALDB 区画定義ユーティリティー (%DFSHALDB) (データベース・ユーティリティー)

HALDB 区画定義ユーティリティーによる HALDB データベースの作成

HALDB 区画定義ユーティリティー (%DFSHALDB) は、IMS HALDB 区画を管理できるようにする ISPF のアプリケーションです。

前提条件**:** HALDB 区画定義ユーティリティーを使用して HALDB データベースの区画を定義するには、事前にデータベース記述生成 (DBDGEN) ユーティリティーを使用して HALDB マスター・データベースを定義しておく必要があります。

区画定義ユーティリティーによる **HALDB** 区画の作成

最初の HALDB 区画を定義するときは、HALDB マスター・データベースを DBRC RECON データ・セットに登録することも必要です。HALDB 区画定義ユーティリティーまたは DBRC の INIT.DB および INIT.PART コマンドを使用して、これを実行することができます。

HALDB 区画定義ユーティリティーは、オンライン IMS サブシステム間の RECON データ・セットの競合には影響を及ぼしません。 RECON データ・セットは、DBRC 要求の処理に要する期間のみ予約されます。これはユーティリティー実行の間を通じて保持されるわけではありません。

区画定義ユーティリティーを使用して HALDB 区画を定義する場合には、区画名、データ・セット接頭部名、およびハイ・キー値のような重要な情報を提供する必要があります。区画定義ユーティリティーは可能な限り、必須フィールドにデフォルト値を提供します。

自動および手動 **HALDB** 区画定義

Partition Default Information (区画デフォルト情報) パネルの Processing Options (処理オプション) セクションの中の Automatic Definition (自動定義) フィールドに `Yes` または `No` を指定することにより、自動区画定義または手動区画定義を選択することができます。

Automatic Definition (自動定義) フィールドに `Yes` を入力することは、区画定義ユーティリティーが HALDB 区画を自動的に定義することを示します。前もってデータ・セットを作成しておき、そこに HALDB 区画選択ストリングを入れておく必要があります。 Input data set (入力データ・セット) フィールドに、そのデータ・セットの名前を指定してください。

Automatic Definition (自動定義) フィールドに `No` を入力することは、HALDB 区画を手動で定義することを示します。

HALDB 区画を手動で定義する場合でも、input data set (入力データ・セット) を使用することができます。

新規の HALDB を定義するステップは、以下のとおりです。

1. ダイアログ・データ・セットを TSO ユーザーが使用できるようにします。 それらのデータ・セットを LOGON プロシージャーに追加するか、TSO コマンドを使ってそれらのデータ・セットを割り当てることができます。 TSOLIB コマンドを使用して、データ・セットを STEPLIB に追加することができます。以下の表は、割り振りが必要なファイル名とデータ・セットを示しています。必ずユーザー自身の高位修飾子を使用してください。

表 72. ファイル名とデータ・セットの割り振り

ファイル名	サンプル・データ・セット名	後処理
STEPLIB	IMS.SDFSRESL	N/A
SYSPROC	IMS.SDFSEXEC	SHR
ISPMLIB	IMS.SDFSMLIB	SHR
ISPPLIB	IMS.SDFSPLIB	SHR
ISPTLIB	IMS.SDFSTLIB	SHR

表 72. ファイル名とデータ・セットの割り振り (続き)

ファイル名	サンプル・データ・セット名	後処理
IMS	IMS.DBDLIB	SHR

ログオン・プロシージャーを使用する場合、再度ログオンして新規のプロシージャーにログオンを指定する必要があります。割り振りコマンドを使用する場合、ISPF の外でコマンドを発行する必要があります。データ・セットの割り振りが終了した後で、ISPF を再始動し、Install/IVP ダイアログを再始動し、このタスクの記述に戻り、そして残りのステップを継続して行ってください。

2. ISPF コマンド行から以下のコマンドを発行して、HALDB 区画定義ユーティリティーを開始します。TSO %DFSHALDB
3. Database Name (データベースの名前) を指定します。 最初の区画名を、以下の例に示すように入力してください。以下の例に示されている高位修飾子の代わりに、ユーザーのデータ・セット用のデータ・セット名を使用して、 Data set name prefix (データ・セット名接頭部) を入力してください。ただし、最下位修飾子を IVPDB1A として指定して、以前に割り振ったクラスター名と一致させる必要があります。

   推奨事項**:** 区画に命名する場合は、後で順序を乱すことなく新しい名前を追加できるような命名順序を使用してください。例えば、区画を xxxx010、xxxx020、xxxx030 のように命名しておけば、後で大きくなりすぎたために区画 xxxx020 を分割する場合に、新規区画には命名順序の配列を乱すことなく xxxx025 という名前を付けることができます。

```
Help
-------------------------------------------------------------
                 Partition Default Information

Type the field values.  Then press Enter to continue.

Database Name . . . . . . . IVPDB1

                      Processing Options
Automatic Definition . . . .No
Input data set . . . . . . .
Use defaults for DS groups .No

                     Defaults for Partitions
Partition Name . . . . . . .IVPDB11
Data set name prefix . . . .IXUEXEHQ.IVPDB1A

Free Space
  Free block freq. factor . 0
  Free space percentage . . 0

                     Defaults for data set groups
Block Size . . . . . . . . .8192

DBRC options
   Max. image copies . . . .2
   Recovery period . . . . .0
   Recovery utility JCL . . RECOVJCL
   Default JCL . . . . . . ._________
   Image Copy JCL . . . . . ICJCL
   Online image copy JCL . .OICJCL
   Receive JCL . . . . . . .RECVJCL
   Reusable? . . . . . . . .No

To exit the application, press F3

Command = = = >
```

図 *242. Partition Default Information* (区画デフォルト情報)

4. Change Partition (区画変更) パネルでそれぞれの区画を定義します。 区画名とデータ・セット名接頭部が正しいことを確認してから、各区画用のハイ・キーを定義します。

   ハイ・キーは、区画に入れることができるレコードの最高位ルート・キーを示し、 Change Partition (区画変更) パネルの Partition High Key (区画ハイ・キー) フィールドに直接入力する 16 進値で表されます。 16 進値を受け入れ、Partition High Key (区画ハイ・キー) フィールドの右側セクションにそれに相当する英数字を表示するには、F5 を押してください。

   Partition High Key (区画ハイ・キー) フィールドの 16 進数セクションに変更を加える前に、F5 を押すことにより、英数字を使用して区画ハイ・キー値を入力することができます。これにより、ISPF 編集パネルが表示されます。この編集パネルに入力する英数字入力データは、F3 を押して保管し、ISPF エディターを終了すると、 Change Partition (区画変更) パネルに 16 進数形式と英数字形式の両方で表示されます。

   HALDB データベースに対して定義する最後の区画は、ハイ・キー値 X'FF' を含んでいる必要があります。これにより、HALDB データベースに入れられるすべてのレコードのキーは HALDB データベースの中の最高位ハイ・キーより低位のキーとなります。区画定義ユーティリティーは、 Partition High Key (区画ハイ・キー) フィールドの中の残りのすべてのバイトを 16 進数 X'FF' で

埋めます。データベース内で最大のハイ・キーが指定された区画に X'FF' 以外のキー値が与えられた場合に、指定されたハイ・キーより大きいキーを持つデータベース・レコードにアクセス、またはそれを挿入しようとすると、その呼び出しに対して FM 状況コードが生成されます。非 HALDB データベース用に作成されたアプリケーション・プログラムがこの状況コードを予期しているとは思われません。

区画ハイ・キーの定義が完了したら、改行キーを押して区画を作成します。他の区画を作成できるように、Change Partition (区画変更) パネルはアクティブのままになります。他の区画を作成するためには、partition name (区画名) と partition high key (区画ハイ・キー) を変更する必要があります。

以下の図はChange Partition (区画変更) パネルを示しています。Partition High Key (区画ハイ・キー) フィールドには、サンプル入力が含まれています。

```
Help
-------------------------------------------------------------
                     Change Partition
Type the field values. Then press Enter.

Database name.........IVPDB1
Partition name.........IVPDB11
Partition ID...........1
Data set name prefix...IXUEXEHQ.IVPDB1A
Partition Status......._______


Partition High Key
  +00 57801850 00F7F4F2 40C5A585 99879985 | ...&.742 Evergre |
  +10 859540E3 85999981                   | en Terra         |

Free Space
   Free block freq. factor...0
   Free space percentage.....0

Attributes for data set group A
   Block Size...............8192

DBRC options
   Max. image copies.........2
   Recovery period...........0
   Recovery utility JCL......________
   Image copy JCL............ICJCL
   Online image copy JCL.....OICJCL
   Receive JCL...............RECVJCL
   Reusable?.................No


Command  = = = >
```

図 *243. Change Partition* (区画変更) パネル

5. 区画の定義を完了したら、取り消しキー (F12) を押して Change Partition (区画変更) パネルを終了します。 現行セッションで定義された区画のリストが表示されます。

   HALDB 区画定義ユーティリティーを完全に終了するためには、F12 を再度押します。

関連概念:

733 ページの『HALDB データベースのオフライン再編成のオプション』

558 ページの『DBDGEN ユーティリティーの入力としてのデータベース記述のコーディング』

DBRC 管理 (システム管理)

関連タスク:

455 ページの『HD および PHD データベースにおける複数データ・セット・グループの使用の指定』

関連資料:

データベース記述 (DBD) 生成ユーティリティー (システム・ユーティリティー)

ILDS の割り振り

ILDS は、論理関係または副次索引を含んでいる HALDB 区画に対するポインターを管理します。

データベースを区分化することは、データベース・レコード間のポインターの使用を複雑にする可能性があります。その理由は、ある区画が再編成された後、以下のポインターが無効になる場合があるからです。

- この区画内の他のデータベース・レコードからのポインター
- この区画を指す他の区画からのポインター
- 副次索引からのポインター

間接ポインターを使用すると、1 つの区画を再編成したときに、他のデータベース・レコード全体にわたってポインターを更新する必要性がなくなります。間接リスト・データ・セット (ILDS) は、間接ポインターのリポジトリーの役割を果たします。 PHDAM および PHIDAM のデータベースの区画ごとに 1 つの ILDS があります。

ILDS には間接リスト項目 (ILE) が含まれています。 ILDS 内の各 ILE には、ターゲット・セグメントの間接リスト・キー (ILK) にターゲット・セグメントのセグメント・コードが付加された、9 バイトのキーがあります。 ILK は、セグメントの作成時にセグメントに割り当てられる固有のトークンです。

レコード間ポインティングに関係するセグメントの再編成再ロードまたはマイグレーション再ロードの後、ILE は ILE のターゲット・セグメントの位置の変化を反映して更新されます。レコード間ポインティングに関係するセグメントは、以下のいずれかのタイプです。

- 物理的に対になった論理子
- 単一方向論理子の論理親
- 副次索引のターゲット

以下のサンプル・コマンドは、ILDS を定義しています。論理レコード上でキー・サイズは 9 バイト、オフセットは 0 (ゼロ) であることに注意してください。また、レコード・サイズは、現在の ILE の長さである 50 バイトが指定されていることにも注意してください。

```
DEFINE CLUSTER ( -
         NAME (FFDBPRT1.XABCD01O.L00001) -
         TRK(2,1) -
         VOL(IMSQAV) -
         FREESPACE(80,10) -
         REUSE -
         SHAREOPTIONS(3,3) -
         SPEED ) -
       DATA ( -
         NAME(FFDBPRT1.XABCD01O.INDEXD) -
         CISZ(8192) -
         KEYS(9,0) -
         RECSZ(50,50) ) -
       INDEX ( -
         NAME(FFDBPRT1.XABCD01O.INDEXS) -
         CISZ(2048) )
```

ILDS のサイズを計算するためには、ILE のサイズに、物理的に対になった論理子の数、単一方向関係の論理親の数、および副次索引のターゲットの数の合計数を乗算します。

ILDS にフリー・スペースを組み込むと、CI および CA 分割の回数が減少して ILDS 処理のパフォーマンスが向上する可能性があります。HD 再編成再ロード・ユーティリティー (DFSURGL0) および HALDB 索引/ILDS 再作成ユーティリティー (DFSPREC0) には、VSAM ロード・モードを使用して ILDS の更新や再作成を行うフリー・スペース・オプションが用意されています。VSAM ロード・モードでは、DEFINE CLUSTER コマンドの FREESPACE パラメーターで要求されたフリー・スペースが追加されます。

関連概念:

764 ページの『HALDB 自己回復ポインター処理』

HALDB 間接リスト・データ・セットの定義 (システム定義)

関連資料:

HALDB 索引/ILDS 再作成ユーティリティー (DFSPREC0) (データベース・ユーティリティー)

HD 再編成再ロード・ユーティリティー (DFSURGL0) (データベース・ユーティリティー)

SQL アプリケーション用の生成済みプログラム仕様ブロックの定義

生成済み PSB (GPSB) は、PSBGEN または ACBGEN を必要としないタイプの PSB です。

GPSB には、入出力 PCB および単一の変更可能代替 PCB が入っています。GPSB は PSBGEN を通じては定義されません。代わりに、APPLCTN マクロを通じてシステム定義処理で定義されます。 GPSB パラメーターは生成した PSB の使用およびそれに関連した名前を指定します。 LANG パラメーターは、GPSB の言語フォーマットを指定します。 GPSB の定義について詳しくは、「*IMS V13* システム定義」の APPLCTN マクロのトピックを参照してください。

入出力 PCB は、アプリケーション・プログラムが入力メッセージを取得し入力を行っている端末に出力を送るために使用できます。代替 PCB は、アプリケーション・プログラムが他の端末またはプログラムに出力を送るために使用できます。

構造化照会言語 (SQL) の呼び出しのみを行うアプリケーションは、入出力 PCB 以外、どの PCB も必要としません。しかしそのアプリケーションが、アプリケーション・プログラムの名前と IMS 言語タイプを IMS に定義することは必要です。GPSB は、この目的で使用できます。

オンライン・システムへのデータベースの導入

オンライン IMS システムでデータベースが認識されるようにするには、事前にオンライン・システムで適切なデータベース制御ブロックを作成する必要があります。

オンライン IMS システムでは、アプリケーション制御ブロック (ACB) と、特定のランタイム属性を保管する制御ブロックの 2 つのタイプの制御ブロックがデータベースに必要です。データベースの ACB は、オンライン変更プロセスによってオンライン・システムに追加されます。オンライン・システムへの ACB の追加について詳しくは、オンライン変更の実行 (システム管理)を参照してください。

オンライン・システムでデータベース・ランタイム属性の制御ブロックを作成するには、ステージ 1 システム定義に各オンライン・データベース用の DATABASE マクロを組み込むか、または動的リソース定義が使用可能になっているシステムでタイプ 2 コマンド CREATE DB を使用します。

動的リソース定義が使用可能になっているオンライン・システムでは、リソース定義データ・セット (RDDS) または IMSRSC リポジトリーからデータベース定義をインポートするか、またはタイプ 2 コマンド IMPORT DEFN を使用する方法でデータベースを導入することもできます。

IMS が IMSRSC リポジトリーを使用している場合は、IMS ウォーム・リスタートまたは緊急時再始動の終わりに、リポジトリー内の IMS 変更リストからデータベースを作成または更新することができます。

DATABASE システム定義マクロ、CREATE DB コマンド、および IMPORT DEFN コマンドでは、いずれも IMS がオンライン環境でデータベースを管理するために必要なデータベース・ディレクトリー (DDIR) が作成されます。

CREATE DB コマンドまたは IMPORT DEFN コマンドでオンライン・システムに追加したデータベースの定義をコールド・スタート後も維持するには、データベース定義を RDDS またはリポジトリーにエクスポートするか、またはシステム定義によって IMS MODBLKS データ・セットに追加し、コールド・スタート時にインポートする必要があります。ウォーム・スタートおよび緊急時再始動後は、IMS によって動的リソース定義の変更がログからリカバリーされます。

データベース定義を RDDS またはリポジトリーにエクスポートするには、EXPORT DEFN コマンドを使用します。また、定義がシステム・チェックポイントで自動的に RDDS にエクスポートされるように IMS を構成することもできます。自動エクスポートを使用可能にするには、AUTOEXPORT=AUTO または

AUTOEXPORT=RDDS を IMS PROCLIB データ・セット内の DFSDFxxx メンバーの DYNAMIC_RESOURCES セクションに指定します。

関連概念:

IMSRSC リポジトリーのリソース・リスト (システム定義)

IMSRSC リポジトリーの概要 (システム定義)

関連タスク:

869 ページの『オンライン・システムでのデータベースの動的な変更』

848 ページの『オンライン・データベースの変更』

オンライン IMS システムへのデータベースの動的な追加

動的リソース定義が使用可能になっている IMS システムでは、CREATE DB コマンドを実行することによってオンライン・システムにデータベースを追加できます。

CREATE DB コマンドの機能は、DATABASE システム定義マクロの機能と一致します。どちらも IMS 制御領域でデータベース・ディレクトリー (DDIR) 制御ブロックを作成することによって、オンライン IMS システムに対してデータベースを宣言します。

CREATE DB コマンドは、DATABASE システム定義マクロで定義できるすべてのデータベース属性を定義します。属性には、次のものが含まれます。

- オンライン IMS システムによるデータベースの認識
- アクセス・タイプ
- データベースの常駐状況

CREATE DB コマンドは、データベースに DBD、PSB、および ACB を定義する前または後に実行できますが、アプリケーション制御ブロック生成 (ACBGEN) 処理の後に CREATE DB コマンドを実行すると、定義されているデータベースのタイプに基づいて IMS が追加のアクションを実行することがあります。

ACBGEN 処理が完了し、ACB ライブラリー内にデータベース用の DMB がある場合、IMS は CREATE DB コマンドが実行されると、以下に示す追加のアクションを実行します。

- 全機能データベースの場合は、最初のスケジュールで IMS がデータベース用の DMB を DMB プールにロードし、常駐オプションが選択されていれば、 IMS の次回再始動時に IMS によってデータベース用の DMB が常駐化されます。
- DEDB データベースの場合は、IMS が DMB を DEDB DMCB チェーンに連結します。

  また IMS は、DMCB 内のエリアの CI サイズが、高速機能グローバル・バッファー・プールに定義されているサイズを超えていないかどうかを検査します。CI サイズが大きすぎる場合は、CREATE DB コマンドが条件コード E3 で失敗します。

  また、DEDB エリア名が高速機能エリア・リスト (FPAL) にすでに存在する場合は、CREATE DB コマンドが条件コード E4 で失敗します。

CREATE DB コマンドが実行されたときにデータベース用の DMB が ACB ライブラリーに存在しない場合でも、IMS はデータベースをオンライン・システムに追加しますが、データベースの状況は NOTINIT になります。データベースを使用するには、事前にデータベース用の DMB を ACB ライブラリーに追加して開始しておく必要があります。MSDB データベースを追加する場合を除いて、ACB ライブラリー・メンバーのオンライン変更機能を使用して DMB を追加することができます。

CREATE DB コマンドの使用のほか、以下のいずれの方法でもオンライン・システムにデータベースを追加できます。

- 自動インポート機能または IMPORT コマンドを使用して、リソース定義データ・セット (RDDS) から保管されたリソース定義をインポートする。
- 自動インポート機能または IMPORT DEFN コマンドを使用して、IMSRSC リポジトリーから保管されたリソース定義をインポートする。

オンライン IMS システムへの MSDB データベースの動的な追加

動的リソース定義を使用して MSDB データベースをオンライン IMS システムに追加する手順は、他のデータベース・タイプを追加する手順とは異なります。

1. MSDB データベースに対して DBDGEN 処理と ACBGEN 処理を実行します。
2. ACB ステージング・ライブラリーを非アクティブな ACB ライブラリーにコピーします。
3. 完全な ACBLIB オンライン変更を実行します。ACB ライブラリーに対するメンバー・レベルでのオンライン変更機能は、MSDB をサポートしていません。
4. MSDB 保守ユーティリティー (DBFDBMA0) を使用して、セグメントを MSDBINIT データ・セットに挿入します。
5. MSDB データベースで CREATE DB コマンドを発行します。

MSDB データベースは、オンライン・ストレージにロードされるまで使用できないので注意してください。

MSDB データベースをロードするには、次のようにします。

1. IMS をシャットダウンする。
2. MSDBLOAD キーワードを指定して IMS をウォーム・スタートする。例えば、コマンド /NRE MSDBLOAD を実行します。

z/OSMF を使用した高速機能 DEDB データベースのプロビジョン

IMS で GitHub を介して提供される IBM z/OS Management Facility ワークフローを使用して、IMS 高速機能 DEDB データベースをプロビジョンできます。

プロビジョニング・ワークフローを使用した場合、IMS は、DEDB データベースをプロビジョン解除するためのワークフローも提供します。

ワークフローは、複数の XML ファイルとプロパティー・ファイルから構成されます。

1. GitHub の「IMS-zOSMF-Workflows」リポジトリーから、必要なワークフローが含まれる圧縮フォルダー (.zip ファイル) をダウンロードし、コンテンツをローカル・ディレクトリーに解凍します。

2. .zip ファイルに含まれている README を参照し、ワークフローに関する詳細を確認します。
3. 解凍したファイルを、z/OSMF インスタンスから使用可能な z/OS 上のデータ・セットまたはディレクトリーに移動します。
4. z/OSMF のウェルカム・ページで、左側のナビゲーション・ペインから「**Workflows**」を選択します。
5. 「**Create a workflow**」を選択し、ワークフロー・ファイルの場所およびその他の初期情報 (ユーザー名など) を指定します。
6. z/OSMF インターフェースにワークフローのステップが表示されたら、各ステップで必要な情報を提供することで、各ステップの実行を開始することができます。 一部のステップでは、ステップを展開してサブステップから開始する必要があります。

関連概念:

z/OS: IBM z/OS Management Facility

関連資料:

IMS-zOSMF-Workflows

第 24 章 テスト・データベースの構築

ユーザーのデータベースをアクセスすることになるアプリケーション・プログラムは、本稼働段階に移る前に、テストしなければなりません。実動データベースに損傷を与えないようにするには、テスト・データベースが必要です。

IBM は、テスト・データベース (DL/I Test Program (DFSDDLT0) を含む) の構築に役立つ各種プログラムを提供しています。使用可能な IMS ツールの詳細については、www.ibm.com/ims にアクセスし、IBM® DB2 および IMS ツールの Web サイトにリンクしてください。

関連概念:

37 ページの『第 1 回コード検査の参加者』

36 ページの『第 4 回の設計レビュー』

IMS アプリケーション・プログラムのテスト (アプリケーション・プログラミング)

システムのテスト (システム管理)

テスト要件

ご使用のシステム構成、ユーザーの要件、およびデータベースとデータ通信システムの設計特性に応じて、DL/I 呼び出しシーケンス、アプリケーション決定パス、およびパフォーマンスをテストする必要があります。

以下のテストを行います。

- DL/I 呼び出しシーケンスが実行され、その結果が正しいものである。
  - この種のテストには、普通、2、3 のレコードしか必要でありません。この場合には、DL/I テスト・プログラムすなわち、DFSDDLT0 を用いてレコードを作成することができます。
  - これが単体のテストの一部分である場合には、既存のデータベースからレコードを抽出することができます。必要なレコードを抽出するには、IMS DataRefresher™ などのプログラムを使用することができます。
- 可能なすべてのアプリケーション決定パスを通じて呼び出しが実行される。
  - この場合には、ユーザーの本稼働データベースに近似させる必要があるかもしれません。そのためには、IMS DataRefresher やその他の IMS ツールなどのプログラムを使用することができます。
- 例えば、システム・テストあるいは回帰テスト用のモデルのパフォーマンスとの比較
  - この種のテストの場合には、本稼働データベースの 1 つのサブセットのコピーが必要になる場合があります。 IMS ツールを使うと役に立つことがあります。

これらの機能についてテストするには、本稼働データベースにできる限り近いテスト・データベースが必要です。この種のテスト・データベースを設計するためには、データベース、サンプル・データ、およびアプリケーション・プログラムの要件を理解しなければなりません。

本稼働データベースを保護するために、アプリケーション・プログラムをテストする人には、テスト用の JCL プロシージャーを用意することを考えてください。テスト用の JCL を提供すれば、正しいライブラリーが使用されていることを保証するのに役立ちます。

どのようなデータベースか

テスト・データベースとしては、しばしば本稼働データベースの 1 つのサブセットのコピー、または何らかの方法による本稼働データベースの複製を使います。実動データベースを設計したことがある場合には、この要件についての直接の知識をお持ちのはずです。本稼働データベースの DBD によってその詳細が得られます。本稼働データベースがデータ・ディクショナリーに定義してある場合には、この定義を見れば、必要な情報の大部分が分かります。この章のトピックでは、テスト・データベースの設計および生成に役立てるために入手可能なエイドについて説明します。

どのようなサンプル・データか

サンプル・データは、実データに近似させることが重要です。システムは、フィールド値の範囲などの同じ特性を持つデータを処理できるかどうかについてテストする必要があるからです。必要なサンプル・データの種類は、呼び出しをテストするのかプログラム論理をテストするのかに応じて異なります。

- 呼び出しをテストするためには、シーケンス・フィールドになるフィールドまたは SSA で参照されるようなフィールドの中にだけ値が必要です。
- プログラム論理をテストするためには、加算、比較などのプログラム論理でアクセスされるすべてのフィールドの中にデータが必要です。

この場合にも、実データベースのサブセットのコピーを使用してかまいません。ただし、まずどのフィールドに機密データが入っているかを判別し、その結果、テスト・データベースの中で架空データを使用する必要があります。

どのようなアプリケーション・プログラムか

開発中の操作可能アプリケーション・プログラムを効果的にテストするテスト・データベースを設計するために、このプログラムについて知っておく必要のあるいくつかの事項があります。必要な情報の大部分は、アプリケーション・プログラム設計文書、PSB などの記述子、ユーザーが考えているテスト計画、およびデータ・ディクショナリーの中にあります。

テスト環境での RECON データ・セットに対する DBRC セキュリティーの無効化

RECON データ・セットを実稼働環境からテスト環境にコピーする場合に、RECON データ・セットのテスト用コピーに対して DBRC セキュリティーを無効にすることができます。

無許可で使用される DBRC コマンドや API 要求から RECON データ・セットを保護する DBRC セキュリティーを無効にすると、実動 RECON データ・セットのコピーをテスト環境でより簡単に操作できるようになり、また、問題の判別にも役立ちます。

DBRC セキュリティーを無効にしても、他のセキュリティー製品 (RACF など) で実行されるセキュリティー検査が無効になったり、それ以外の影響を受けることはありません。

RECON データ・セットのコピーに対して DBRC セキュリティー検査を無効にするには、次の手順を実行します。

1. INIT.RECON または CHANGE.RECON DBRC コマンドの CMDAUTH キーワードの *rcnqual* パラメーターで、保護されている RECON データ・セットのデータ・セット名から 1 から 44 文字のサブストリングを指定します。 この時点ではまだセキュリティー検査がアクティブであるため、INIT.RECON または CHANGE.RECON コマンドを実行するための適切なセキュリティー許可が必要です。
2. *rcnqual* パラメーターで指定した文字ストリングを含まない名前で、新しいデータ・セットを定義します。
3. 保護された RECON データ・セットを新しいデータ・セットにコピーします。 *rcnqual* パラメーター内の文字ストリングが RECON データ・セット名のどの部分とも一致しなければ、その RECON データ・セットのコピーに対するセキュリティー検査は無効になります。

例

rcnqual パラメーターを指定するコマンドの例を以下に示します。

- `CHANGE.RECON CMDAUTH(SAF,SAFHLQ1,IMSTESTS.DSHR)`
- `INIT.RECON CMDAUTH(SAF,SAFHLQ1,IMSTESTS.DSHR)`

以下の例は、セキュリティーが無効になっている RECON データ・セットの状況リストを示しています。RCNQUAL フィールドに示されている `IMSTESTS.DSHR` というストリングは、上記のコマンド例のいずれかで設定されたもので、COPY1 RECON データ・セットの名前 `IMSTESTS.COPYDSHR.RECON1` のどの部分とも完全には一致しません。

```
RECON
 RECOVERY CONTROL DATA SET, IMS V13R1
 DMB#=7                              INIT TOKEN=13122F2233528F
 NOFORCER  LOG DSN CHECK=CHECK17    STARTNEW=NO
 TAPE UNIT=3480      DASD UNIT=SYSDA     TRACEOFF   SSID=**NULL**
 LIST DLOG=NO                 CA/IC/LOG DATA SETS CATALOGED=NO
 MINIMUM VERSION = 11.1       CROSS DBRC SERVICE LEVEL ID= 00001
 REORG NUMBER VERIFICATION=NO
 LOG RETENTION PERIOD=00.001 00:00:00.0
 COMMAND AUTH=SAF  HLQ=SAFHLQ1
 RCNQUAL = IMSTESTS.DSHR
 ACCESS=SERIAL      LIST=STATIC
 SIZALERT DSNUM=15      VOLNUM=16     PERCENT= 95
 LOGALERT DSNUM=3       VOLNUM=16

 TIME STAMP INFORMATION:

   TIMEZIN = %SYS
```

```
   OUTPUT FORMAT:  DEFAULT = LOCORG NONE   PUNC YY
                   CURRENT = LOCORG NONE   PUNC YY


 IMSPLEX = ** NONE **    GROUP ID = ** NONE **

-DDNAME-      -STATUS-       -DATA SET NAME-
 RECON1        COPY1          IMSTESTS.COPYDSHR.RECON1
 RECON2        COPY2          IMSTESTS.COPYDSHR.RECON2
 RECON3        SPARE          IMSTESTS.COPYDSHR.RECON3

 NUMBER OF REGISTERED DATABASES =        7
```

以下の例は、セキュリティーがアクティブになっている RECON データ・セットの状況リストを示しています。この例では、RCNQUAL フィールドに示されている IMSTESTS.DSHR というストリングが、COPY1 RECON データ・セットの名前 IMSTESTS.DSHR.RECON1 の一部と一致します。

```
RECON
 RECOVERY CONTROL DATA SET, IMS V13R1
 DMB#=7                              INIT TOKEN=13122F2233528F
 NOFORCER  LOG DSN CHECK=CHECK17    STARTNEW=NO
 TAPE UNIT=3480      DASD UNIT=SYSDA     TRACEOFF   SSID=**NULL**
 LIST DLOG=NO                CA/IC/LOG DATA SETS CATALOGED=NO
 MINIMUM VERSION = 11.1      CROSS DBRC SERVICE LEVEL ID= 00001
 REORG NUMBER VERIFICATION=NO
 LOG RETENTION PERIOD=00.001 00:00:00.0
 COMMAND AUTH=SAF  HLQ=SAFHLQ1
 RCNQUAL = IMSTESTS.DSHR
 ACCESS=SERIAL      LIST=STATIC
 SIZALERT DSNUM=15      VOLNUM=16     PERCENT= 95
 LOGALERT DSNUM=3       VOLNUM=16

 TIME STAMP INFORMATION:

   TIMEZIN = %SYS

   OUTPUT FORMAT:  DEFAULT = LOCORG NONE   PUNC YY
                   CURRENT = LOCORG NONE   PUNC YY


 IMSPLEX = ** NONE **    GROUP ID = ** NONE **

-DDNAME-      -STATUS-       -DATA SET NAME-
 RECON1        COPY1          IMSTESTS.DSHR.RECON1
 RECON2        COPY2          IMSTESTS.DSHR.RECON2
 RECON3        SPARE          IMSTESTS.DSHR.RECON3

 NUMBER OF REGISTERED DATABASES =        7
```

テスト・データベースの設計、作成、およびロード

テスト・データベースは、本稼働データベースを構築するのと同じように構築することができます。

例えば、テスト・データベースでの特別な要件に留意しながら、データベース管理情報で説明されている作業を実行します。ご使用のシステムでテストの標準と手順がある場合には、それに従ってテスト・データベースを構築しなければなりません。

テスト基準の使用

テストの標準と手順は、テスト・データベースの構築に関して、 IMS 開発の基準が実動データベースに関して役立つのと同じ種類の問題の回避に役立ちます。

テスト・システム基準に組み込まれる可能性のある事項の中で、テスト・データベースの設計に影響を及ぼすものをいくつか以下に列挙します。

- テスト・システムの目標
  - どの開発段階で、何をテストするか
  - テストの種類 - オフライン、オンライン、統合 DB/DC、または分離 DB/DC
- テスト編成の記述および各グループの責任の決定
- テスト・モード操作と実動モード操作の関係
- テスト・システム開発過程での以下の項目に対する処理方法
  - DB/TM 構造
  - 開発ツール
  - DB/TM 機能
  - バックアップおよびリカバリー

関連概念:

 システムのテスト (システム管理)

 CICS アプリケーション・プログラムのテスト (アプリケーション・プログラミング)

 IMS アプリケーション・プログラムのテスト (アプリケーション・プログラミング)

IBM プログラムを用いたテスト・データベースの構築

本稼働データベースの構築に使用するのと同じ開発エイドを用いてテスト・データベースを構築すれば、使い慣れたツールを使用するという利点があります。

また、テスト・データベースと本稼働データベースの間の相違から起きる問題を避けることができます。

IMS アプリケーション開発機能 II

IMS アプリケーション開発機能 II (IMSADF II) は、IMS システム用のアプリケーションを開発するプログラマーの生産性を上げるように設計されています。

IMSADF II は、IMS と Db2 for z/OS を使用して作業するための ISPF ベースのダイアログを提供し、プログラマーは、それによって、IMS データベースおよびデータ通信アプリケーションを開発、保守、および拡張するために必要な時間と労力が軽減されます。

ご使用のシステムで IMSADF II を用いてアプリケーション・プログラムを開発している場合には、これを用いて単純なテスト・データベースを作成することができます。 IMSADF II は対話式なので、データベースにセグメントを動的に追加する

ことができます。 SEGM ステートメントと FIELD ステートメントによりテスト・データベースを定義し、必要に応じてそれを更新することができます。

関連資料: IMS アプリケーション開発機能 II の使用方法については、「*IMS Application Development Facility (ADF) II User's Guide*」を参照してください。

File Manager for z/OS (IMS データ用)

ファイル・マネージャーの IMS コンポーネント (FM/IMS) は、IMS データベースに保管されているデータを操作できる ISPF アプリケーションです。

FM/IMS では、IMS データベースに接続するための柔軟な方法が多数提供されています。例えば、BMP モードでは、オンラインのマルチユーザー・データベースに接続して、そのデータを操作できます。 DLI モードでは、単一ユーザーとしてデータをオフラインで処理することもできるし、データを他と共用することもできます。

さらに、FM/IMS は、バッチ・ジョブで使用できる機能を 2 つ提供しています。FM/IMS Edit Batch (IEB) は、セグメントの挿入、更新、検索、削除、または印刷とビューの作成が可能な REXX プロシージャーを実行します。FM/IMS Batch Print (IPR) は、ビューに基づいて、いくつかの使用可能な表示フォーマットの 1 つでデータベース全体、またはデータベースの選択されたサブセットを印刷できます。

DL/I テスト・プログラム (DFSDDLT0) の使用

DL/I テスト・プログラム (DFSDDLT0) を使用して、DL/I 呼び出しシーケンスのテストまたはセグメントの挿入を実行できる場合があります。

例えば、データベース・レコードの数が比較的少ないテスト・データベースが必要である場合は、DFSDDLT0 を使用して DL/I 呼び出しシーケンスをテストできます。特に、まず初めに機械可読のデータベースがない場合には、PCB を定義してから、DFSDDLT0 を用いてセグメントを挿入することができます。このステップによって、ロード・プログラムにテスト・データベースを生成させる必要がなくなります。

DL/I テスト・プログラムは CICS では使用できませんが、独立型バッチ・プログラムで使用することができます。独立型バッチ・プログラムで使用する場合、データベースのパフォーマンスをオンラインまたは共用データベース・プログラムにインプリメントされるものとして解釈すると役に立ちます。

関連概念:

IMS プログラムのテストに先立つ DL/I 呼び出しシーケンスのテスト (DFSDDLT0) (アプリケーション・プログラミング)

第 5 部 データベース管理作業

このセクションでは、IMS データベースの管理に関連する基本作業について説明します。説明する内容は、データベースのロード、バックアップとリカバリー、モニター、チューニング、および変更などです。

第 25 章 データベースのロード

データベースの設計をインプリメントした後、データベースの作成とロードの準備ができたことになります。しかし、ロード・プログラムを作成する前に、データ・セットの割り振りとデータベースの最小サイズの見積もりを行う必要があります。

関連概念:

37 ページの『第 1 回コード検査の参加者』

関連タスク:

786 ページの『アクセス方式サービス DEFINE CLUSTER コマンドで指定されている VSAM オプションの調整』

データベースの最小サイズの見積もり

データベースのサイズを見積もるにあたって、最初にデータをロードするのにどのくらいのスペースが必要であるかを見積もります。

データベースがロードされた後はデータベースにセグメントを挿入しない計画でない限り、最初のロードのために実際に見積もるよりも多いスペースを割り振ってください。

このトピックでは、最小データベース・スペースを見積もるための手順をステップごとに説明します。データベースに必要とされる最小サイズを見積もるためには、データベースの物理的なインプリメンテーションに関して設計上の決定をすでにいくつか下していなければなりません。この決定は各 DL/I アクセス方式ごとに異なっているので、この決定については、このプロシージャーのステップ 3 の中の該当するアクセス方式のところで説明します。

HALDB 区画のオンライン再編成を計画する場合は、再編成に必要な追加のスペースを考慮してください。オンライン再編成では、HALDB 区画を最初にロードする時点では追加スペースは必要ありませんが、この処理は再編成の時点で追加スペースを必要とします。

関連概念:

925 ページの『第 31 章 データベース・タイプの変換』
492 ページの『HISAM データベースの論理レコード長の選択』
738 ページの『HALDB オンライン再編成』
809 ページの『論理関係の追加』

関連タスク:

804 ページの『セグメント・タイプの削除』
788 ページの『割り振られるスペース量の変更』
805 ページの『セグメント・サイズの変更』
838 ページの『データ・セット・グループの数の変更』
774 ページの『HDAM または PHDAM オプションの調整』

786 ページの『アクセス方式サービス DEFINE CLUSTER コマンドで指定されている VSAM オプションの調整』

802 ページの『再編成ユーティリティーによるアンロードおよび再ロード』

ステップ 1. 平均的なデータベース・レコードのサイズの計算

データベース・レコード内の各セグメント・タイプのサイズを決定し、次に各セグメント・タイプの平均出現回数を決定します。この 2 つの数を掛け合わせることにより、平均的なデータベース・レコードのサイズが得られます。

セグメント・サイズの決定

ここでのセグメント・サイズは物理セグメント・サイズです。すなわち、このサイズには、セグメントの接頭部とデータ部の両方が含まれています。

データ部のサイズは、データベース管理者が定義します。これには、将来の使用のための未使用スペースを組み込むことができます。セグメントのデータ部のサイズは、DBD の中の SEGM ステートメントの BYTES = オペランドで指定した数です。

セグメントの接頭部は、セグメント・タイプ、および使用されているオプションに応じて長さが変わります。以下の表は、セグメント・タイプ別に接頭部のサイズを決める助けとなります。この図を使用するには、所要の接頭部情報に必要なバイト数、および選定したオプションのために接頭部の中に生成される余分のフィールドならびにポインターに必要なバイト数を加えて、この図を上から下まで見てください。セグメントの接頭部は 4 バイトのポインターを複数もつことができます。この種の余分のポインターをすべて計算に入れる必要があります。

表 73. セグメントの接頭部に必要なフィールドとポインター

セグメントの種類	セグメントの接頭部で使用されるフィールドとポインター	フィールドまたはポインターのサイズ (バイト単位)
すべての種類	セグメント・コード (SHSAM、SHISAM、GSAM、または副次索引ポインター・セグメントにはありません)	1
	削除バイト (SHSAM、SHISAM、または GSAM セグメントにはありません)	1
HDAM、PHDAM、HIDAM、および PHIDAM	PCF ポインター	4
	PCL ポインター	4
	PP ポインター	4
	PTF ポインター	4
	PTB ポインター	4
HDAM および HIDAM のみ	HF ポインター	4
	HB ポインター	4
DEDB	PCF ポインター	4
	PCL ポインター	4
	サブセット・ポインター	4

表 73. セグメントの接頭部に必要なフィールドとポインター (続き)

セグメントの種類	セグメントの接頭部で使用されるフィールドとポインター	フィールドまたはポインターのサイズ (バイト単位)
論理親 (HDAM および HIDAM の場合)	LCF ポインター	4
	LCL ポインター	4
	論理子カウンター	4
論理親 (PHDAM および PHIDAM の場合)	論理子カウンター (単一方向論理親の場合のみ存在)	4
論理子	LTF ポインター	4
	LTB ポインター	4
	LP ポインター	4
論理子 (PHDAM および PHIDAM)	EPS	28
副次索引	ターゲット・セグメントを指すシンボリック・ポインターまたは直接アドレス・ポインター	4
PSINDEX	EPS にターゲット・セグメント・ルート・キーを加えたもの	28 + ターゲット・セグメント・ルート・キーの長さ
PHDAM および PHIDAM の全セグメント	ILK	8

関連概念:

166 ページの『各種ポインターの併用』

セグメント頻度の決定

1 つのセグメント・タイプ全体のサイズを決めた後で、セグメントの頻度を決定する必要があります。

セグメント頻度とは、データベース・レコードの中の特定の 1 つのセグメント・タイプのオカレンス数の平均値です。セグメントの頻度を決めるには、まずある 1 つのセグメントがその直接の物理親の下に平均何回現れるか決定します。

例えば、以下の図のデータベース・レコードでは、ITEM セグメントは、各 DEPOSITS セグメントのもとに平均 10 回現れます。 DEPOSITS セグメントは、各 CUSTOMER ルート・セグメントのもとに平均 4 回現れます。ルート・セグメントの頻度は常に 1 であるという点に注意してください。

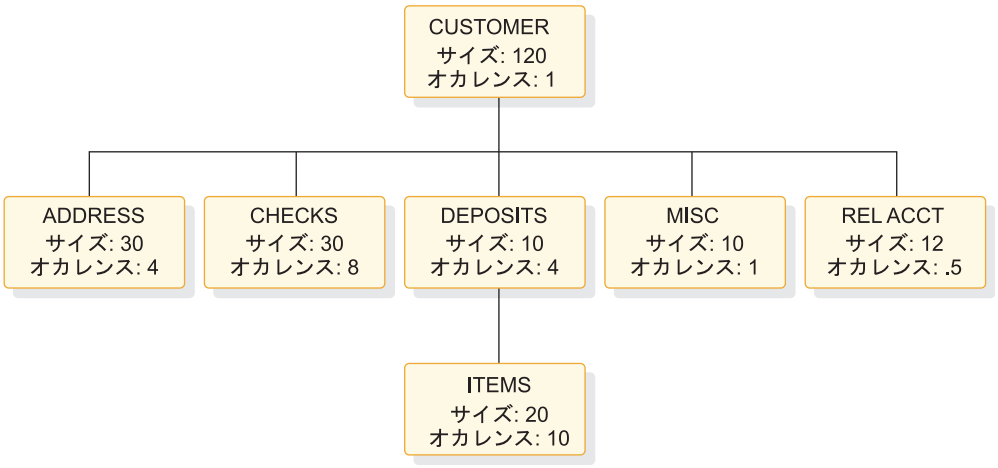


図 *244.* セグメント・サイズおよび平均のセグメント・オカレンス

データベース・レコードの中の特定のセグメント・タイプのオカレンス数の平均値を決めるには、このセグメントから逆方向にルート・セグメントに至るパスにある各セグメントのセグメント頻度を掛け合わせます。 ITEMS セグメント・タイプの場合は、このパスには ITEMS セグメントと DEPOSITS セグメントが組み込まれています。 ITEMS のセグメント頻度は 10 であり、DEPOSITS のセグメント頻度は 4 です。したがって、このデータベース・レコードの中の ITEMS セグメントのオカレンス数の平均値は 40 (すなわち、10 × 4) です。別の言い方をすれば、各カスタマーが平均 4 つの DEPOSITS を持ち、それぞれの DEPOSIT は平均 10 ITEMS で構成されています。したがって、客先ごとにこのデータベース・レコードの中に平均 40 (= 10 × 4) の ITEMS が存在します。

データベース・レコードの平均サイズの決定

セグメント・サイズとセグメント頻度を決定したら、データベース・レコードの平均サイズを決定することができます。

HISAM データベースの場合、データベース・レコードの平均サイズを決めるには、このデータベース・レコードの中のセグメント・タイプごとに、セグメント・サイズとセグメント頻度を掛け合わせ、次に、この結果を加算します。例えば、605 ページの『セグメント頻度の決定』 に示すデータベース・レコードの場合、データベース・レコード平均サイズは、以下の表のように計算されます。

表 *74.* データベース・レコード平均サイズの計算

セグメント・タイプ	セグメント・サイズ	平均オカレンス数	合計サイズ
CUSTOMER	120	1	120
ADDRESS	30	4	120
CHECKS	30	8	240
DEPOSITS	10	4	40
ITEMS	20	40 (10x4)	800
MISC	10	1	10
REL ACCT	12	.5	6
レコード合計			**1336**

ステップ 2. CI のリソースに必要なオーバーヘッドの決定

VSAM を使用している場合には、CI に必要なオーバーヘッドの量を決めないと、残りのスペース計算を行うことができません。

VSAM を使用していない場合には、このステップを省略することができます。

オーバーヘッドとは、1 つの CI で 2 つの制御フィールドのために使用されるスペースです。 VSAM は、CI の中のスペースを管理するためにこれらの制御フィールドを使用します。これらの制御フィールドとそのサイズを以下の表に示します。

表 *75. VSAM* 制御フィールド

フィールド	サイズ (バイト数)
CIDF (制御インターバル定義フィールド)	4
RDF (レコード定義フィールド)	3

CI 当たり 1 つの論理レコードがある場合、CI のオーバーヘッドは、CIDF 1 つと RDF 1 つで構成されます (したがって、合計 7 バイトです)。 HDAM と HIDAM データベース、および PHDAM と PHIDAM 区画は、CI 当たり 1 つの論理レコードを使用します。

各 CI に複数の論理レコードがある場合には、CI オーバーヘッドは、CIDF 1 つと RDF 2 つで構成されます (したがって、合計 10 バイトです)。 HISAM (KSDS と ESDS) データベース、HIDAM と PHIDAM 索引データベース、および副次索引データベースでは、いずれもそれぞれの CI ごとに複数の論理レコードを使用できます。

ステップ 3 で、いつ CI オーバーヘッドをスペース計算に入れたらよいかを述べます。

ステップ 3. 必要な CI またはブロックの数の決定

このステップでの計算は、データベース・タイプ別に行います。

データベース・レコードを収容するのに必要な CI またはブロックの数を決めるには、このステップの中の、使用しているデータベース・タイプに該当するトピックに進んでください。 VSAM を使用している場合には、データベースの最初の CI が VSAM のために予約されています。

HISAM: 必要な **CI** またはブロックの数の決定

HISAM の中の 1 つの CI には 1 つ以上の論理レコードを入れることができます。 1 次データ・セットでは、1 つの論理レコードに 1 つのデータベース・レコード (または 1 つのデータベース・レコードの一部) しか入れることができません。オーバーフロー・データ・セットでは、1 つの論理レコードに、同じデータベース・レコードのセグメントしか入れることができませんが、単一のデータベース・レコードのオーバーフロー・セグメントのために複数の論理レコードを使用することができます。

HISAM においては、論理レコードの働きを覚えておくことが大切です。というのは、論理レコードのオーバーヘッドを計算に入れないと、データベース・レコード

を収容するのに必要な CI (制御インターバル) の数を決めることができないからです。論理レコードのオーバーヘッドは、1 つの論理レコードに常に必要なオーバーヘッドと、データベース・レコードが論理レコードに保管される方法によって生まれるオーバーヘッド (すなわち、セグメントの保管によって、ほとんど常に、残りのスペースつまり未使用スペースが生じます) との組み合わせです。

一部のオーバーヘッドは各論理レコードと関連しているので、論理レコード・オーバーヘッドを計算に入れたあと使用可能なスペースの量を計算する必要があります。いったん、1 つの論理レコードの中でデータに使用可能なスペースの量が分かったら、データベース・レコードを収容するのに必要な論理レコードの数を決めることができます。論理レコードがいくつ必要であるかが分かると、必要になる CI またはブロックの数を決定することができます。

例えば、VSAM を使用して 500 のデータベース・レコードをロードしようとしているとします。また、KSDS および ESDS 両方で、 2048 バイトの CI サイズを使用しようとしているとします。また、各 KSDS CI に 4 つの論理レコードを保管し、各 ESDS CI に 2 つの論理レコードを保管する必要があるものとします。

1. まず、CI サイズからオーバーヘッドを引いて、CI オーバーヘッドを計算に入れます。KSDS および ESDS の両方では、これは 2048 - 10 = 2038 バイトになります。 10 バイトのオーバーヘッドは、4 バイトの CIDF 1 つと 3 バイトの RDF 2 つ分です。
2. 次に、論理レコード・サイズを計算します。それには、使用可能な CI スペースを CI 当たりの論理レコード数で割ります。すなわち、KSDS では 2038/4 = 509 バイト、ESDS では 2038/2 = 1019 バイトです。 HISAM の論理レコード・サイズは偶数値でなければならないので、KSDS では 508 バイトを使用し、ESDS では 1018 バイトを使用します。
3. 最後に、論理レコード・サイズからオーバーヘッドを引いて、論理レコード・オーバーヘッドを計算に入れます。すなわち、KSDS では、 508 - 5 = 503 バイト、ESDS では 1018 - 5 バイトです。 HISAM 論理レコード・オーバーヘッドは、VSAM では 5 バイトです (論理レコードのチェーンのための 4 バイトの RBA ポインター 1 つと 1 バイトのデータ終了標識の 1 つです)。

   したがって、KSDS では 508 バイトという論理レコード・サイズを指定すると、この中の 503 バイトをデータの保管に使用することができます。 ESDS で 1018 バイトという論理レコード・サイズを指定すると、この中の 1013 バイトがデータの保管に使用することができます。

先の例をさらに続けます。 データベース・レコードの平均サイズは、 1336 バイトなので、KSDS でデータに使用できるスペースはこれを入れるのに十分な大きさではありません。平均的なデータベース・レコードを収容するには、1 つの KSDS 論理レコードおよび 1 つの ESDS 論理レコードの中の使用可能なスペースが必要です (使用可能なスペースは 503 + 1013 = 1516 バイトです)。このレコード・サイズは、平均的なデータベース・レコードの 1336 バイトより大きくなっています。500 のデータベース・レコードをロードする必要があるので、KSDS および ESDS の両方に 500 の論理レコードが必要です。

- KSDS で CI 当たり 4 つの論理レコードを保管するためには、KSDS には最小限、各 2048 バイトの CI が 500/4 = 125 だけ必要です。

- ESDS で CI 当たり 2 つの論理レコードを保管するためには、ESDS には最小限、各 2048 バイトの CI が 500/2 = 250 だけ必要です。

HIDAM または **PHIDAM:** 必要な **CI** またはブロックの数の決定

HIDAM または PHIDAM では、CI またはブロック 1 つにつき 1 つの VSAM 論理レコードがあります。このコンテキストでは、論理レコードは、レコードを get または put するアクセス方式 (VSAM など) の呼び出し時の転送単位です。論理レコードのオーバーヘッドは、FSEAP (4 バイト) で構成されます。 RAP (HIDAM のみ) を使用する場合は、論理レコードのオーバーヘッドは RAP (4 バイト) 1 つで構成されます。例えば、VSAM を用いて 500 のデータベース・レコードをロードする必要があるものとします。 CI サイズを 2048 バイトとし、RAP を使用しないものとします (HIDAM の場合、RAP を抑止するために PTR=TB をルート上で指定します)。

1. まず、CI サイズから CI オーバーヘッドを引いて、論理レコードのサイズを決めます。すなわち、ESDS 論理レコード・サイズでは 2048 - 7 = 2041 バイトです。 7 バイトのオーバーヘッドは、4 バイトの CIDF と 3 バイトの RDF の分です。
2. 次に論理レコード・オーバーヘッドを計算に入れて、データに使用できる論理レコード・スペースの量を決めます。この例では、論理レコード・オーバーヘッドは FSEAP 1 つで構成されており、したがって、2041 - 4 = 2037 バイトです。これは、各論理レコードにおいて、データを保管するために 2037 バイトだけ使えることを意味します。

HIDAM 索引または **PHIDAM** 索引**:** 必要なスペースの計算

HIDAM 索引または PHIDAM 索引のためのスペース計算は、HISAM KSDS のためのスペースの計算とほぼ同じです。相違点は、論理レコード・オーバーヘッドがないという点です。 1 つの索引レコードは 1 つの論理レコードに保管され、複数の論理レコードは 1 つの CI またはブロックに保管できます。

HDAM または **PHDAM:** 必要なスペース量の決定

HDAM または PHDAM では変数の数が多いために、データベース・スペースの必要量を見積もるための明確な数式はありません。したがって、スペース計算エイドの助けを借りて、HDAM または PHDAM データベースに必要となるスペース量を決めるようにすべきです。

VSAM を用いており、データベースに割り振るスペースの量を、エイドを用いずに見積もることにした場合に、データベースの最初の CI は VSAM に予約されます。このために、ビットマップは、2 番目の CI の中にあります。

HDAM または PHDAM では、論理レコード・オーバーヘッドは、データベース管理者が選定したデータベースの設計オプションに左右されます。ルート・アドレス可能域の中の CI またはブロックの数、および各 CI またはブロック当たりの RAP の数はユーザーが選択しなければなりません。データベースに関する知識に基づいてこれらの選択を行います。

ランダマイザーが完全であれば、データベース・レコードと同じ数だけの RAP が必要です。しかし、完ぺきなランダマイザーなどというものは存在しないので、デ

ータベース・レコードの数の約 20% 増しの数だけ RAP を計画してください。余分の RAP を用意すれば、シノニム・チェーンの可能性を減らすことになります。例えば、500 のデータベース・レコードを保管する必要があるものとします。ルート・アドレス可能域に関して、

- CI またはブロック当たり 1 つの RAP を使用すると、600 の CI またはブロックが必要です。
- CI またはブロック当たり 2 つの RAP を使用すると、300 の CI またはブロックが必要です。
- CI またはブロック当たり 3 つの RAP を使用すると、200 の CI またはブロックが必要です。

ランダマイザーの動作から見て、それぞれ最もよく機能する 2 つの RAP を持つ 300 の CI またはブロックを使用することを決定します。 VSAM を使用して 500 のデータベース・レコードを保管する必要があり、ルート・アドレス可能域で 300 の CI を使用し、CI 当たり 2 つの RAP を使用することを選択したと想定します。この決定が CI サイズの選択に影響を及ぼします。 CI 当たり 2 つの RAP を使用しているので、各 CI にデータベース・レコードが 2 つずつ保管されることが予期されます。 2048 バイトの CI では、2 つのデータベース・レコード (2 x 1336 = 2672 バイト) を収容するのに十分な大きさではありません。また、3072 バイトの CI では、平均的なサイズの 2 つのデータベース・レコードには大きすぎます。したがって、おそらく 2048 バイトの CI を使用し、バイト限界カウントを用いて、CI に平均 2 つのデータベース・レコードを保管することになります。

バイト限界カウントを決めるには、次のようにします。

1. まず、CI サイズから CI オーバーヘッドを引いて、論理レコードのサイズを決めます。すなわち、ESDS 論理レコード・サイズでは 2048 - 7 = 2041 バイトです。
2. 次に論理レコード・オーバーヘッドを計算に入れて、データに使用できる論理レコード・スペースの量を決めます。 (HDAM または PHDAM では CI 当たり 1 つしか論理レコードがないことを思い出してください。) この例では、論理レコード・オーバーヘッドは 4 バイトの FSEAP 1 つと 4 バイトの RAP 2 つで構成されています。すなわち、2041 - 4 - (2 x 4) = 2029 バイトとなります。これは、ルート・アドレス可能域の中の各論理レコードの中に、データの保管に使用できるスペースが 2029 バイトずつあることを意味しています。
3. 最後に、使用可能な論理レコード・スペースを CI 当たりの RAP 数で割ることにより、RAP 当たりの使用可能なスペースを決めます。すなわち、2029/2 = 1014 バイトとなります。したがって、おそらく約 1000 バイトのバイト限界カウントを使用することになります。

この例を続けますが、ルート・アドレス可能域には、各 2048 バイトの CI が 300 必要なことが分かっています。ここで、オーバーフロー域に必要な CI の数を計算する必要があります。これを行うには、次のようにします。

- ルート・アドレス可能域に収まらないバイト数の平均値を決めます。バイト限界カウントが 1000 バイトであるものとします。平均のデータベース・レコード・サイズからバイト限界カウントを引きます。すなわち、1336 - 1000 = 336 バイ

トとなります。平均のオーバーフロー・バイト数にデータベース・レコードの数を掛けます。すなわち、500 x 336 = 168000 バイトが非ルート・アドレス可能域に必要になります。

- 非ルート・アドレス可能域で必要となる CI の数を決めます。それには、オーバーフロー・バイトの数を 1 つの CI でのデータに使えるバイト数で割ります。1 つの CI でデータのために使えるバイト数を決めるには、CI サイズから CI オーバーヘッドおよび論理レコード・オーバーヘッドを引きます。すなわち、2048 - 7 - 4 = 2037 (7 バイトは CI オーバーヘッド、4 バイトは FSEAP です) となります。オーバーフロー・バイト数を CI データ・バイト数で割ると、オーバーフロー域の CI が求まります。すなわち、168000/2037 = 83 CI となります。

こうして、ルート・アドレス可能域には最小 300 CI 必要であり、非ルート・アドレス可能域には最小 83 CI 必要であることが見積もられました。

副次索引: 必要なスペース量の決定

副次索引のスペース計算は、HISAM KSDS のためのスペースの計算とほぼ同じです。違っているのは、別にすべき論理レコード・オーバーヘッドがないという点です。

1 つの索引レコードは 1 つの論理レコードに保管され、複数の論理レコードは 1 つの CI またはブロックに保管できます。

ステップ 4. フリー・スペースに必要なブロックまたは CI の数の決定

HDAM、HIDAM、PHDAM、および PHIDAM データベースにおいては、データベースを初期ロードするときにフリー・スペースを割り振ることができます。

フリー・スペースは、HD の VSAM ESDS または OSAM データ・セットで割り振ることができます。ここで説明するフリー・スペースと `DEFINE CLUSTER` コマンドを使用して VSAM KSDS に割り振ることができるフリー・スペースを混同しないでください。

データベースの中で割り振る必要のある CI またはブロックの総数を計算するには、次の数式を使用します。

```
A = B x (fbff / (fbff - 1)) x (100 / (100 - fspf))
```

ここで、それぞれの値は次のとおりです。

A フリー・スペースを含めて、必要とされる CI またはブロックの総数。

B データベースの中のブロックまたは CI の数。

fbff データベース内のブロックまたは CI をフリー・スペースとして空けておくべき頻度。

HDAM データベースまたは HIDAM データベースの場合、これは、DBD の FRSPC= パラメーターで指定された *fbff* 値です。

fspf 各ブロックまたは CI においてフリー・スペースとして残しておくべき最小のパーセント。

HDAM データベースまたは HIDAM データベースの場合、これは、DBD の FRSPC= パラメーターで指定された *fbff* 値です。

関連タスク:

487 ページの『フリー・スペースの指定 (HDAM、PHDAM、HIDAM、および PHIDAM のみ)』

ステップ 5. ビットマップに必要なスペース量の決定

HDAM、HIDAM、HIDAM、PHIDAM データベースにおいては、ビットマップに必要なスペース量を計算に入れる必要があります。

データベースの中のビットマップに必要なバイト数を計算するには、次の数式が使用できます。

```
A = D / ((B - C) x 8)
```

ここで、それぞれの値は次のとおりです。

A データベースに必要なビットマップ・ブロックまたはビットマップ CI の数。

B 指定した CI サイズまたはブロック・サイズ (バイト単位) から 4 を引いた差。

CI またはブロック・サイズから 4 を引くのは、各 CI または

C 1 つの CI またはブロックに対して指定した RAP の数を 4 倍した値。

RAP の数を 4 倍するのは、各 RAP の長さが 4 バイトだからです。この数式では (B - C) に 8 が掛けられていますが、これによって CI またはブロックにおいてビットマップに使用できるビット総数が求まります。

D データベースの中の CI またはブロックの数。

ビットマップに必要な CI またはブロックの数をスペース計算に加算します。

関連概念:

168 ページの『HD データベースの一般フォーマットと特殊フィールドの用途』

データベース・データ・セットの割り振り

データベースに必要なスペースの量が決まれば、データ・セットを割り振り、次いでデータベースをロードすることができます。

VSAM データ・セットは DEFINE CLUSTER コマンドを使用して割り振ることができます。 HALDB データ・セットを割り振るとき、REUSE パラメーターを指定する必要があります。このコマンドの使用については、z/OS DFSMS カタログのためのアクセス方式サービス・プログラムに説明があります。

重要: データベースのイメージ・コピーを取るためにデータベース・イメージ・コピー 2 ユーティリティーを使用する場合は、DFSMS 並行コピー機能または DFSMS 高速複製機能をサポートするハードウェアにデータ・セットを割り振る必要があります。

論理関係または副次索引、またはその両方が入っているデータベース (HALDB データベースを除く) をロードする場合には、DL/I は制御レコードを作業ファイル (DFSURWF1) に書き込みます。この作業ファイルもまた JCL で割り振られていなければなりません。

他のデータ・セットはすべて、通常の z/OS JCL を用いて割り振られます。データベースが MSDB 以外の場合には、z/OS プログラム IEFBR14 を用いてデータ・セットを事前に割り振ることができます。MSDB の場合には、z/OS プログラム IEHPROGM を使用する必要があります。

データ・セット割り振りの標準の DFSMS 方式および一般的なデータ・セットについて詳しくは、以下を参照してください。

- *z/OS DFSMS: Using Data Sets*
- *z/OS DFSMS* カタログのためのアクセス方式サービス・プログラム

関連タスク:

838 ページの『データ・セット・グループの数の変更』

アクセス方式としての OSAM の使用

OSAM は、IMS と共に提供される特別なアクセス方式です。

このトピックにはプロダクト・センシティブ・プログラミング・インターフェース情報が含まれています。

データベースにおいて OSAM アクセス方式を使用している場合は、OSAM に関する以下の情報を知っている必要がかあります。

- IMS は、OPEN マクロ、CLOSE マクロ、READ マクロ、および WRITE マクロを用いて OSAM と連絡をとります。
- OSAM は、入出力ドライバー・インターフェースを用いて入出力監視プログラムと連絡をとります。
- OSAM データ・セットは、BSAM アクセス方式または QSAM アクセス方式を用いて読み取ることができます。
- DD ステートメントで DSNTYPE=LARGE を指定することにより、BSAM アクセス方式を使用する順次 OSAM データ・セットを、z/OS のラージ・フォーマット・データ・セットと定義することができます。ラージ・フォーマット・データ・セットは、65,535 トラックを超えることができます。
- OSAM データ・セットにおけるエクステントの最大数は 60 です。ただし、OSAM データ・セットに関連付けられている z/OS データ・エクステント・ブロック (DEB) の長さが増加した場合、この最大数が少なくなる可能性があります。さらに、エクステントの最大数は、回転位置感知機構 (RPS) 装置用に作成されるセクター番号テーブルの長さによっても制限されます。

  最小サイズのセクター・テーブルでは、DEB は最大 60 の DASD エクステントを反映することができます。最大サイズのセクター・テーブルでは、DEB は最大 52 の DASD エクステントを反映することができます。

  さらに、それぞれのエクステント域 (2 つのダブルワード) について、OSAM では装置割り付け状況データを入れる同様のエリアが必要です。各エクステントに

は合計 4 つのダブルワードが必要です。 OSAM DEB のフォーマットと長さ (ダブルワードによる表示) は、以下の表に示されています。

表 *76. OSAM DEB* の長さとフォーマット

フォーマット	ダブルワードでの長さ
付加セクター・テーブル	5
基本 DEB	4
アクセス方式従属セクション	2
サブルーチン名セクション	1
標準 DEB エクステント	120 (60 エクステント)
OSAM エクステント・データ	120
最小セクター・テーブル	2

- OSAM データ・セットは、その位置での更新、および末尾での拡張用として 1 つのデータ制御ブロック (DCB) によってオープンすることができます。「末尾の拡張」という句は、レコードをデータ・セットの末尾に追加でき、しかも新しい直接アクセス・エクステントを取得できることを意味しています。
- 使用前に OSAM データ・セットのフォーマットを設定する必要はありません。
- OSAM データ・セットでは、固定長のブロック化レコードまたは非ブロック化レコードを使用することができます。
- OSAM データ・セットの最大サイズは、データ・セットのブロック・サイズ、およびそのデータ・セットが HALDB OSAM データ・セットであるかどうかに応じて決まります。OSAM データ・セットのサイズ限度は、次のとおりです。
  - 非 HALDB データベースの場合は 4 GB (データ・セットのブロック・サイズが奇数の場合) または 8 GB (データ・セットのブロック・サイズが偶数の場合)
  - HALDB データベース区画の場合は 4 GB または 8 GB (HALDB が 8-GB OSAM データ・セットをサポートするとして DBRC に登録されている場合)
- データ・セットの現在の終わりを示すために、ファイル・マーク定義が常に使用されます。新しいブロックがデータ・セットの末尾に追加されるときは、新しいブロックが、論理シリンダー単位におかれていた (OSAM による) 事前フォーマット設定のダミー・ブロックと置き換わります。論理シリンダーのフォーマット設定の操作中にファイル・マークがあると、それは次のシリンダーの最初に書き込まれます。この手法は、OSAM データ・セットがオープンされている間の信頼性エイドとして用いられます。
- OSAM ボリューム終わり (EOV) とクローズ処理の間に、OSAM EXCP カウントが蓄積されます。
- ADRDSSU および z/OS DFSMS: DFSMSdss の DFSMSdss コンポーネントを利用した OSAM データ・セットのマイグレーションは、データ・セットのトラックを、マイグレーションされているボリューム用に DSCB で指定した最終ブロック書き出し値 (DS1LSTAR) までマイグレーションします。OSAM データ・セットが事前割り振りされていない複数のボリュームにまたがる場合には、各 DSCB の DS1LSTAR フィールドは有効であり、 DFSMSdss はデータを正しくマイグレーションすることができます。

OSAM データ・セットが事前割り振りされている複数のボリュームにまたがる場合は、各ボリューム (最後のボリュームは除く) の DSCB の中の DS1LSTAR フィールドはゼロになることがあります。この状態は、事前割り振り済みのマルチボリューム・データ・セットのロード操作中に起こります。事前割り振り済みボリュームを使用すると、あるボリュームから別のボリュームに移る場合に EOV 処理が行われないので、これらのボリュームの DSCB は更新されません。ロードされた最後のボリュームの DSCB は、データ・セットのクローズ処理の間に更新されます。

マイグレーションする OSAM データ・セットが事前割り振り済みの複数のボリュームにまたがる場合には、ALLEXCP パラメーターあるいは ALLDATA パラメーターを指定した DFSMSdss 物理 DUMP または RESTORE コマンドを使用しなければなりません。これらのパラメーターにより、DFSMSdss は OSAM データ・セットを正しくマイグレーションすることができます。

関連資料: DFSMS の z/OS DFSMSdss コンポーネント、および DUMP コマンドと RESTORE コマンドの ALLEXCP パラメーターと ALLDATA パラメーターについて詳しくは、「*z/OS DFSMS Storage Administration Reference (for DFSMSdfp, DFSMSdss, and DFSMShsm)*」を参照してください。

- OSAM データ・セットを、z/OS V1.12 以降で使用可能な拡張アドレス・ボリューム (EAV) の拡張アドレス方式スペースに配置できるようにすることができます。OSAM データ・セットを EAV の拡張アドレス方式スペースに配置できるようにするには、OSAM データ・セットを割り振るときに VOL=SER= パラメーターで EAV ボリュームを指定します。さらに、属性 EATTR=OPT を指定して、データ・セットが拡張アドレス方式エリアを使用するために必要な拡張属性をサポートしていることを示します。

  制約事項: EATTR=OPT が指定されているデータ・セットを、IMS V10 システムや IMS V11 システムと共用することはできません。これらのバージョンの IMS は拡張属性をサポートしていないからです。

他の z/OS アクセス方式 (VSAM および SAM) は、データの物理的な保管のために、OSAM に加えて使用されます。

関連概念:

153 ページの『HD データベースの最大サイズ』

OSAM サブプール定義 (システム定義)

関連資料:

高速順次処理制御ステートメント (システム定義)

OSAM データ・セットの割り振り

HALDB データベース以外のデータベースの場合、データ・セットのロードの時点で、JCL を使用して OSAM データ・セットを割り振る必要があります。SPACE パラメーターを使用すれば、このモードの割り振りは単一ボリュームにもマルチボリュームにも使えます。

OSAM は、マルチボリューム・データ・セットの代わりとして使用できるラージ・フォーマット順次データ・セットもサポートしています。

関連概念:

663 ページの『リカバリーおよびデータ・セット』

単一ボリューム OSAM データ・セットの割り振り

単一ボリューム OSAM データ・セットは、JCL または z/OS DFSMS アクセス方式サービス (AMS) IDCAMS を使用して割り振ることができます。

- JCL を使用して単一ボリューム OSAM データ・セットを割り振る場合は、以下の JCL 例をモデルとして各自で JCL を作成できます。

```
//ALLOC1  EXEC  PGM=IEFBR14
//DD1      DD   DSN=HIGHNODE.MIDNODE.LOWNODE,
//              DISP=(NEW,CATLG),
//              SPACE=(CYLS,(200,100)),
//*  add UNIT and VOLSER if needed
//*  add SMS parameters if needed
//*  do not code DCB parameters
/*
```

- AMS IDCAMS を使用して単一ボリューム OSAM データ・セットを割り振る場合は、以下の JCL 例をモデルとして各自で JCL を作成できます。

```
//ALLOC1  EXEC  PGM=IDCAMS,REGION=4096K
//SYSPRINT DD   SYSOUT=*
//SYSIN    DD   *
  /*                                          */
  /* DELETE OLD DATA SET IF IT EXISTS         */
  /*                                          */
  DELETE 'HIGHNODE.MIDNODE.ENDNODE'

  IF LASTCC <= 8 THEN SET MAXCC = 0

  /*                                          */
  /*  ALLOCATE - WITH SMS PARAMETERS          */
  /*                                          */

  ALLOCATE DSN('HIGHNODE.MIDNODE.ENDNODE') -
           NEW CATALOG -
           DATACLAS(DCLAS) -
           MGMTCLAS(MCLAS) -
           STORCLAS(SCLAS) -
           SPACE(200 100) CYLINDERS
/*
```

- 拡張アドレス方式スペースに配置できる OSAM データ・セットを非 SMS 管理対象 DASD 上の拡張アドレス・ボリュームに割り振る場合は、以下の JCL 例をモデルとして JCL を作成できます。

```
//OSAMALLO   JOB  A,OSAMEXAMPLE
//S1        EXEC  PGM=IEFBR14
//EAVD            DD  VOL=SER=EAV001,SPACE=(CYL,(20,5)),UNIT=3390,
//              DSN=OSAM.SPACE,DISP=(,KEEP),EATTR=OPT
```

マルチボリューム OSAM データ・セットの割り振り

マルチボリューム HALDB データベース・データ・セットの場合、OSAM データ・セットを事前割り振りする必要があります。単一ボリューム HALDB OSAM データ・セットの場合、割り振りはロード・ステップの前またはその途中で行うことができますが、ロード・ステップ内で行うことを推奨します。

ご使用のシステムにおける直接アクセス記憶装置のスペースおよびボリュームの管理で、OSAM データ・セットを事前に割り振っておくことが必要である場合、また

はメッセージ・キュー・データ・セットに複数のボリュームが必要である場合には、OSAM データ・セットは事前割り振りされなければなりません。

何らかの受け入れられている方法を用いて事前割り振りを行う場合には、次に挙げる制限を守ってください。

- DCB パラメーターを指定してはなりません。
- 1 次割り振りを超えてデータ・セットが拡張される場合は、すべてのボリュームに 2 次割り振りを指定する必要があります。
- 事前割り振りされているが初期ロードまたは再ロードの処理によって書き込まれてはいないボリュームに書き込むために、すべてのボリュームに対して 2 次割り振りを指定する必要があります。
- キュー・データ・セットは、最初のスペース数量または事前割り振りのスペース数量以上に拡張されないので、キュー・データ・セットでは、2 次割り振りは認められません。しかし、キュー・データ・セットはマルチボリュームの割り振りをもつことはできます。
- 最初のボリュームで定義されている 2 次割り振りサイズは、他のボリュームで指定されている 2 次割り振りサイズには関係なく、すべてのボリュームのすべての 2 次割り振りに使用されます。すべてのボリュームは、混乱を避けるために同じ 2 次割り振りサイズを定義すべきです。
- OSAM データ・セットがカタログされる場合には、すべてのボリュームがカタログ項目に組み込まれるようにするために IEHPROGM またはアクセス方式サービスを使用してください。

マルチボリューム・データ・セットを事前割り振りする場合は、使用されるすべてのボリューム上のエクステントを割り振らなければなりません。割り振りの方式としては、スペースが必要な各ボリュームに IEFBR14 ユーティリティーのステップを 1 つ実行するようお勧めします。

制約事項**:**

- マルチボリューム・データ・セットを DD カードに指定して IEFBR14 を実行しないでください。これを行うと、最初のボリューム上だけにエクステントが割り振られます。
- データベース・イメージ・コピー 2 ユーティリティー (DFSUDMT0) を使用する予定の場合、上記で説明するように、各ボリューム上でエクステントを割り振るために個別の IEFBR14 ユーティリティー・ステップを使用しないでください。各ボリュームに個別の IEFBR14 ステップを使用すると、同じ ID (ボリューム 1) を持つエクステントが作成されます。データベース・イメージ・コピー 2 ユーティリティーを使用する場合、標準の DFSMS 方式でマルチボリューム・データ・セットを割り振る必要があります。

SMS の管理下にない **DASD** 上でマルチボリューム **OSAM** データ・セットを割り振る例**:**

以下の例の JCL では、SMS の管理下にない直接アクセス・ストレージ・デバイス (DASD) 上でマルチボリューム OSAM データ・セットを割り振ります。

この JCL で割り振られたデータ・セットは、データベース・イメージ・コピー 2 ユーティリティーと互換性はありません。

SMS の管理下にない **DASD** 上でマルチボリューム **OSAM** データ・セットを割り振る **JCL**

```
//OSAMALLO   JOB  A,OSAMEXAMPLE
//S1        EXEC  PGM=IEFBR14
//EXTENT1     DD  VOL=SER=AAAAAA,SPACE=(CYL,(20,5)),UNIT=3390,
//              DSN=OSAM.SPACE,DISP=(,KEEP)
//S2        EXEC  PGM=IEFBR14
//EXTENT2     DD  VOL=SER=BBBBBB,SPACE=(CYL,(30,5)),UNIT=3390,
//              DSN=OSAM.SPACE,DISP=(,KEEP)
      .
      .
      .
//LAST      EXEC  PGM=IEFBR14
//EXTENTL     DD  VOL=SER=LLLLLL,SPACE=(CYL,(30,5)),UNIT=3390,
//              DSN=OSAM.SPACE,DISP=(,KEEP)
//          EXEC  PGM=IEHPROGM
//SYSPRINT    DD  SYSOUT=*
//DD1         DD  UNIT=3390,VOL=SER=AAAAAA,DISP=SHR
//DD2         DD  UNIT=3390,VOL=SER=BBBBBB,DISP=SHR
      .
      .
      .
//DDL         DD  UNIT=3390,VOL=SER=LLLLLL,DISP=SHR
//SYSIN       DD  *
  CATLG DSNAME=OSAM.SPACE,VOL=3390=(AAAAAA,BBBBBB,LLLLLL)
/*
```

関連概念:

619 ページの『マルチボリューム OSAM データ・セットの割り振り時の注意』

SMS の管理下にある **DASD** 上でマルチボリューム **OSAM** データ・セットを割り振る例**:**

以下の各例では、SMS の管理下にある DASD 上でマルチボリューム OSAM データ・セットを割り振るための代替方法を示します。

以下の JCL を使用してデータ・セットを割り振る場合は、保証スペースのある (GUARANTEED_SPACE=YES) インストール済み環境で定義されている SMS ストレージ・クラスを指定し、さらにボリューム通し番号を指定する必要があります。保証スペース属性およびボリューム通し番号が指定されていないと、最初のボリュームだけが 1 次エクステントを取得し、他のボリュームは 2 次エクステントを取得します。

以下の JCL は、マルチボリューム OSAM データ・セットを割り振るために使用できる JCL の一例です。これらのデータ・セットは、データベース・イメージ・コピー 2 ユーティリティーと互換性があります。この例の *gtdstcls* は、保証スペース属性付きで定義されているストレージ・クラスです。

```
//OSAMALLO   JOB  A,OSAMEXAMPLE
//          EXEC  PGM=IEFBR14
//DD123       DD DSN=HIGHNODE.MIDNODE.ENDNODE,
//              DISP=(NEW,CATLG),
//              SPACE=(CYL,(200,100)),
//              UNIT=(3390)
//              VOL=SER=(VOL111,VOL222,VOL333),
//              STORCLAS=gtdstcls
/*
```

以下の JCL は、DFSMS アクセス方式サービスの ALLOCATE コマンドを使用してマルチボリューム OSAM データ・セットを割り振る例です。これらのデータ・

セットは、データベース・イメージ・コピー 2 ユーティリティーと互換性があります。この例の *gtdstcls* は、保証スペース属性付きで定義されているストレージ・クラスです。

```
//OSAMALLO   JOB  A,OSAMEXAMPLE
//          EXEC  PGM=IDCAMS,REGION=4096K
//SYSPRINT    DD  SYSOUT=A
//SYSIN       DD *
          DELETE 'HIGHNODE.MIDNODE.ENDNODE'

            IF LASTCC<=8 THEN SET MAXCC=0

            ALLOCATE DSN('HIGHNODE.MIDNODE.ENDNODE') -
            NEW CATALOG -
            SPACE(200,100) CYLINDERS -
            UNIT(3390) -
            VOL(VOL111,VOL222,VOL333) -
            STORCLAS(gtdstcls)
/*
```

SMS の管理下にある DASD でのデータ・セットへのスペースの割り振りについて詳しくは、「*z/OS DFSMS Storage Administration Reference (for DFSMSdfp, DFSMSdss, and DFSMShsm)*」を参照してください。

関連概念:

『マルチボリューム OSAM データ・セットの割り振り時の注意』

マルチボリューム **OSAM** データ・セットの割り振り時の注意**:**

マルチボリューム OSAM データ・セットを割り振る際には、以下の点に注意してください。

1. 初期ロードまたは再ロード処理中に使用されたよりも多くのボリュームを OSAM データ・セット・エクステントに事前割り振りすると、次のような状況のもとで、初期ロードまたは再ロード時に書き込まれた最後のボリュームを越えてデータ・セットを拡張しようとした場合に異常終了が発生します: 初期ロードまたは再ロードのステップで事前に割り振られたデータ・セットの最後のボリュームにデータが書き込まれず、データ・セットの事前割り振り時に 2 次割り振りが指定されていない場合。
2. データベースをロードすると、データ・セットのボリュームは、CATLG ステートメントでそのボリューム通し番号がリストされている順にロードされます。
3. マルチボリュームの OSAM データ・セットを再使用する場合は、最初にデータ・セットをスクラッチし、その後にスペースを再割り振りするようにしてください。これは、HALDB オンライン再編成機能での出力データ・セットの場合も同様です。

   データ・セットのスクラッチと再割り振りを行わないと、データ・セットが次の状態のときに、データ・セットの最後のボリュームの DSCB 内に、無効な EOF マークが残るおそれがあります。

   a. 最初に (データベースの再編成に使用されるアンロード/再ロード・ユーティリティーのような) IMS ユーティリティーによって再使用される。
   b. 次に、通常の処理のために OSAM によってオープンされる。

   例えば、1 つのデータ・セットが 3 つのボリュームに割り振られ、その 3 番目のボリューム上に EOF マークがある場合があります。しかし、再編成ユーティ

リティーの実行後、このデータ・セットが最初の 2 つのボリュームしか必要としないことがあります。そこで 2 番目のボリュームに新しい EOF マークが置かれます。再編成後、通常の処理のために OSAM によってこのデータ・セットがオープンされると、OSAM は最後のボリュームの DSCB に EOF マークがないか調べます。 OSAM は、3 番目のボリュームに EOF を見つけると、再編成ユーティリティーによって 2 番目のボリュームに生成された EOF マークの後ろにデータを挿入する代わりに、3 番目のボリュームにある古い EOF マークの後ろに新しいデータを挿入します。

その後イメージ・コピー・ユーティリティーのような別のユーティリティーが処理を行う場合には、再編成ユーティリティーが 2 番目のボリューム上にセットした EOF マークを使用し、OSAM が 3 番目のボリュームに挿入した新しいデータを無視します。

4. マルチボリューム・データベースをバックアップおよび復元するためにイメージ・コピー 2 ユーティリティー (DFSUDMT0) を使用する予定の場合、標準の DFSMS 手法によってデータ・セットを割り振る必要があります。

   データ・セット割り振りの DFSMS 方式について詳しくは、「*z/OS DFSMS: Using Data Sets*」を参照してください。

関連概念:

617 ページの『SMS の管理下にない DASD 上でマルチボリューム OSAM データ・セットを割り振る例』

618 ページの『SMS の管理下にある DASD 上でマルチボリューム OSAM データ・セットを割り振る例』

OSAM の大規模な形式での順次データ・セットの割り振り

OSAM は、ラージ・フォーマット順次データ・セットをサポートします。ラージ・フォーマット順次データ・セットは、単一ボリューム上で、65 535 トラックまたは 3.4 GB を超えることができます。

以下の JCL は、OSAM のラージ・フォーマット順次データ・セットを割り振る例です。この JCL で割り振られたデータ・セットは、データベース・イメージ・コピー 2 ユーティリティーと互換性があります。

```
//OSAMALBG   JOB  A,OSAMEXAMPLE
//S1        EXEC  PGM=IEFBR14
//AJOSAMDB DD  DSN=IMSTESTL.AJOSAMDB,UNIT=SYSDA,
//             DISP=(NEW,CATLG),DSNTYPE=LARGE,
//             SPACE=(CYL,(4500,100)),VOL=SER=LRGVS1
```

以下の JCL は、OSAM のラージ・フォーマット順次データ・セットを拡張アドレス・ボリューム (EAV) に割り振る例です。この JCL で割り振られたデータ・セットは、データベース・イメージ・コピー 2 ユーティリティーと互換性があります。

```
//OSAMALBG   JOB  A,OSAMEXAMPLE
//S1        EXEC  PGM=IEFBR14
//AJOSAMDB DD  DSN=IMSTESTL.AJOSAMDB,UNIT=SYSDA,
//             DISP=(NEW,CATLG),DSNTYPE=LARGE,
//             SPACE=(CYL,(4500,100)),VOL=SER=EAV001,EATTR=OPT
```

ロード・プログラムの作成

データベースにどれだけのスペースが必要かを決め、それに対してデータ・セットを割り振った後で、データベースをロードすることができます。

ロード・プロセス

データベース・ロードを行うには、初期ロード・プログラムを使用します。オンライン・アプリケーション・プログラムでは、データベースをロードすることはできないので、初期ロード・プログラムは、バッチ・プログラムでなければなりません。このプログラムを作成することは、ユーザーの責任です。

基本的には、初期ロード・プログラムは、データベース・レコードが入っている既存のファイルを読み取ります。 DBD はデータベースの物理的特性を定義しているものですが、この DBD とロード PSB を使用して、ロード・プログラムは、データベース・レコードのためのセグメントを構築し、これらのセグメントを階層順にデータベースの中に挿入します。

以下の図はロード処理を示しています。

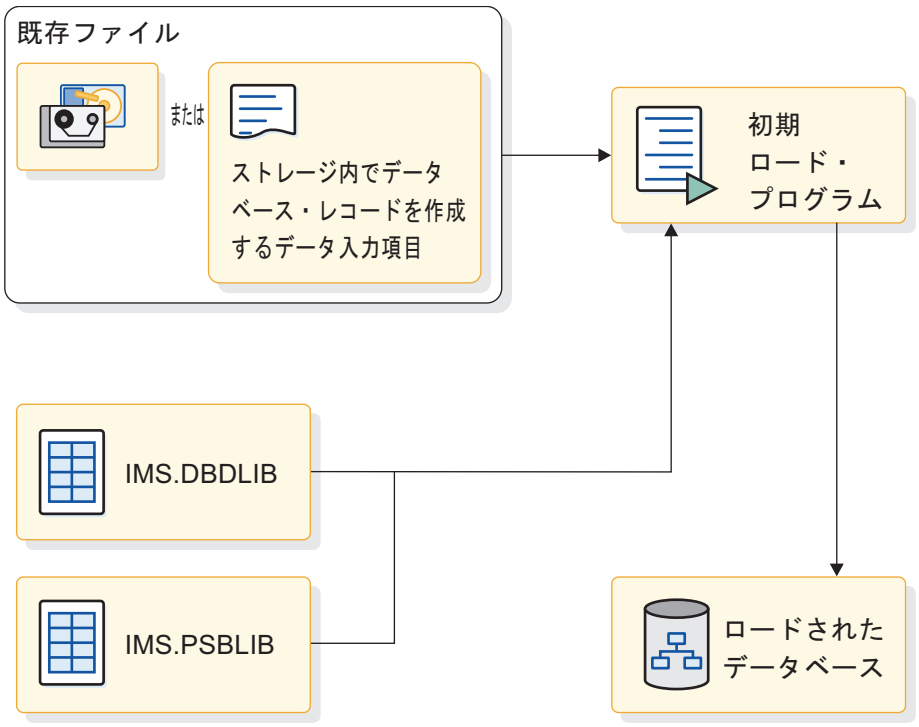


図 245. ロード・プロセス

データベースにロードすべきデータが既に 1 つ以上のファイルの中に存在する場合には、正しい順序でこれを初期ロード・プログラムに与えるために、必要であればデータのマージとソートを行ってください。また、重複データが含まれている既存の複数のファイルを 1 つのデータベースにマージする場合には、重複データを削除し、必要であれば間違っているデータを訂正してください。

以下の図はデータベースのロードを図で示しています。

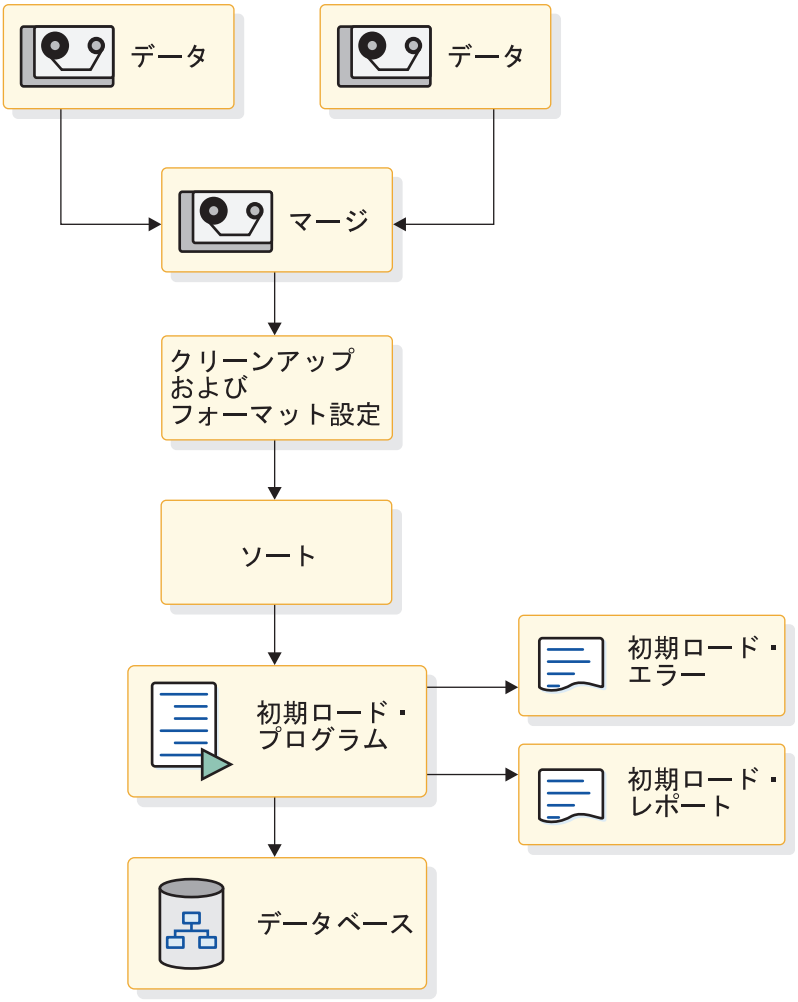


図 *246.* 既存ファイルを用いたデータベースのロード

データベースを定義し終わった後で、ISRT 呼び出しを使用するアプリケーション・プログラムを作成してこのデータベースをロードします。初期ロード・プログラムは、プログラムの入出力域で各セグメントを構築し、次に、ISRT 呼び出しを出すことによってそれをデータベースにロードします。 PCB に PROCOPT=L を指定している場合には、DL/I 要求で許可されているのは ISRT 呼び出しだけです。「L」オプションを使用するのは、データベースの初期ロードの場合だけです。このオプションはバッチ・プログラムでのみ有効です。

制約事項: PROCOPT=L が指定された PCB ステートメントを含む PSB には、PROCOPT 値 A、D、G、I、または R を指定する他の PCB ステートメントを追加できません。

ロード中のデータベースが DBRC に登録されている場合は、このデータベースと論理的に関連するすべてのデータベースにも DBRC 許可が必要となります。データベースのロード時に DBRC がアクティブの場合は、DBRC は、RECON データ・セット内の DBDS レコードの中でこのデータベースのイメージ・コピー状況を IC NEEDED に設定します。

HDAM データベースをロードしているのでない限り、データベースのロード時に FIRST、LAST、および HERE 挿入規則は適用されません。 HDAM データベースをロードする場合には、これらの規則によってどのように非固有シーケンス・フィールドを持つルート・セグメントが順序付けされるのかが決まります。 HSAM を使用してデータベースをロードしている場合、同じ規則が適用されます。

推奨事項: ロード・プログラムはチェックポイントを出す必要はありません。

大抵包括的なデータベースは、セグメント・タイプ別またはセグメント・タイプのグループ別に、いくつかの段階になってロードされます。通常は、1 つのアプリケーション・プログラムだけを使用してロードするにはセグメントの数が多すぎるために、ロードを行うのにいくつかのプログラムが必要になります。最初のロード・プログラムの後の各ロード・プログラムは、技術的には「追加」プログラムであり、ロード・プログラムではありません。追加プログラムに PCB の中の処理オプションとして「L」を指定しないでください。プログラムのパフォーマンスが受け入れられるものであるかどうかを確認するために、データベースをロードするために作成されたロード・プログラムのすべての追加タイプを調べる必要があります。というのは、通常セグメントのグループを追加する方がこれをロードするよりも時間がかかるからです。

HSAM、HISAM、HIDAM、および PHIDAM では、アプリケーション・プログラムが挿入するルート・セグメントは、ルート・セグメントのキー・フィールドで事前にソートしなければなりません。各ルート・セグメントの従属セグメントは、このルート・セグメントに階層順で続いている必要があり、セグメント・タイプ内のキーの値に従っている必要があります。言い換えれば、セグメントが階層順に検索される場合には、セグメントの挿入もプログラムがこれを検索するのと同じ順序で行われます (親の後ろに子で、データベース・レコードはキー・フィールド順)。

HDAM または PHDAM データベースをロードする場合は、キー・フィールドでルート・セグメントを事前ソートする必要はありません。

データベースをロードするときには、

- ロードされたセグメントにキーがある場合には、そのキーの値が入出力域の正しい位置になければなりません。
- 論理子セグメントをロードする場合には、入出力域に論理親の連結キーがあり、その後ろに挿入される論理子セグメントが続いていなければなりません。
- ISRT 呼び出しを出したあと、現在位置は正しくロードされた最後のセグメントの後で、次に使用可能なスペースの直前です。次にロードするセグメントは、このスペースに置かれます。

推奨事項: データベースのロード、再ロード、または再編成の直後に、常にイメージ・コピーを作成する必要があります。

関連概念:

809 ページの『論理関係の追加』

ロード・プログラムの状況コード

ISRT 呼び出しが成功すると、DL/I はプログラムのために空白の状況コードを戻します。成功しなかった場合、DL/I はいくつかある状況コードのいずれかを戻します。

ISRT 呼び出しが成功しなかった場合、DL/I は以下のいずれかの状況コードを戻します。

LB ロードしようとしているセグメントが既にデータベースにあります。 DL/I は、キー・フィールドのあるセグメントにだけ、この状況コードを戻します。

呼び出しレベル・プログラムでは、エラー・ルーチンに制御権移動すべきです。

LC ロードしようとしているセグメントが、キー・シーケンスから外れています。

LD このセグメントには親セグメントがありません。この状況コードは通常、ロードしようとしているセグメント・タイプの順序が正しくないことを意味します。

LE 複数の SSA を持つ ISRT 呼び出しの中で、SSA で指定されている複数のセグメントがそれぞれの正しい階層順にありません。

LF PHDAM または PHIDAM の初期ロードが、論理子セグメントの ISRT を試みました。

V1 長さが無効の可変長セグメントを与えました。

ロード・プログラムでの SSA の使用

DL/I がセグメントを次々に挿入するので、データベースにセグメントをロードするときには、位置を気にする必要はありません。データベースのロードで最も重要なことは、セグメントを構築し挿入する順序です。

与えなければならない SSA は、挿入するセグメント・タイプの名前を指定する非修飾 SSA だけです。

位置を気にする必要はないので、挿入しようとするセグメントの親セグメントに対して SSA を使用する必要もありません。これを使用する場合には、等号 (EQ、=b、または b=) 関係演算子しか含まれていないことを確認してください。比較値として、セグメントのキー・フィールドも使用しなければなりません。

HISAM、HIDAM、および PHIDAM の場合、キー X'FFFF' は IMS のために予約されています。このキーをもっているセグメントを挿入しようとすると、IMS は LB という状況コードを戻します。

D コマンド・コードを使用した一連のセグメントのロード

入出力域の中のセグメントを連結し、非修飾 SSA のリストを DL/I に与えることによって、1 つの呼び出しで一連のセグメントをロードすることができます。

最初の SSA に D コマンド・コードを含めなければなりません。 SSA が定義する一連のセグメントは、階層を下方へ移動し、入出力域の各セグメントが前のセグメントの子セグメントであるという状態になります。

2 種類の初期ロード・プログラム

初期ロード・プログラムには 2 つの種類があります。基本 初期ロード・プログラムと再始動可能 初期ロード・プログラムです。

基本プログラムは、実行中に問題が生じると、最初から再始動しなければなりません。再始動可能プログラムは、問題が発生する前にとられた最後のチェックポイントから再始動することができます。再始動可能初期ロード・プログラムは、ユーティリティー制御機能 (UCF) の制御の下で実行しなければならず、しかもアクセス方式として VSAM が必要です。以下のトピックでは、この 2 種類のロード・プログラムについて説明します。

基本初期ロード・プログラム

ロードを必要としているデータ量がそれほど多くなく、したがって、プログラムの実行中に問題が生じても、あまり後戻りしなくてもよい場合には、基本初期ロード・プログラム (すなわち、再始動可能でないもの) を書くべきです。

問題が生じた場合は、基本初期ロード・プログラムは最初から再実行しなければなりません。

高速機能の高速処理データベース (DEDB) は、DL/I データベースとは異なりバッチ・ジョブではロードすることができません。 DEDB はまず初めに DEDB 初期設定ユーティリティーによって初期設定されてから、普通 BMP 領域で実行されるユーザー作成の高速機能アプリケーション・プログラムによってロードされます。

高速機能の主記憶データベース (MSDB) は、IMS 制御領域が初期設定されるまでは、ロードされません。これらのデータベースは以下の要件が満たされた場合に、IMS 始動プロシージャーによってロードされます。

- メンバー名 IMS の EXEC ステートメント上の MSDB= パラメーターで、IMS.PROCLIB の中の DBFMSDB の 1 文字の接尾部を指定する。
- メンバーに、ロードすべき各 MSDB のレコードが入っている。

  レコードには、各 MSDB のレコード、ロードされるセグメントの数、およびオプションの「F」(MSDB をストレージの中に固定することを示す) が入っています。
- 順次データ・セット、すなわち、ds 名 IMS.MSDBINIT(0) を持つ世代別データ・グループ (GDG) の一部が生成されています。

  このデータ・セットの作成には、ユーザー作成プログラムまたは MSDB 保守ユーティリティーの INSERT 機能を使用することができます。データ・セットの中のレコードの順序は、MSDB の名前順、そして MSDB の中ではキーの順です。

以下の図は、基本初期ロード・プログラム開発のための論理を示したものです。

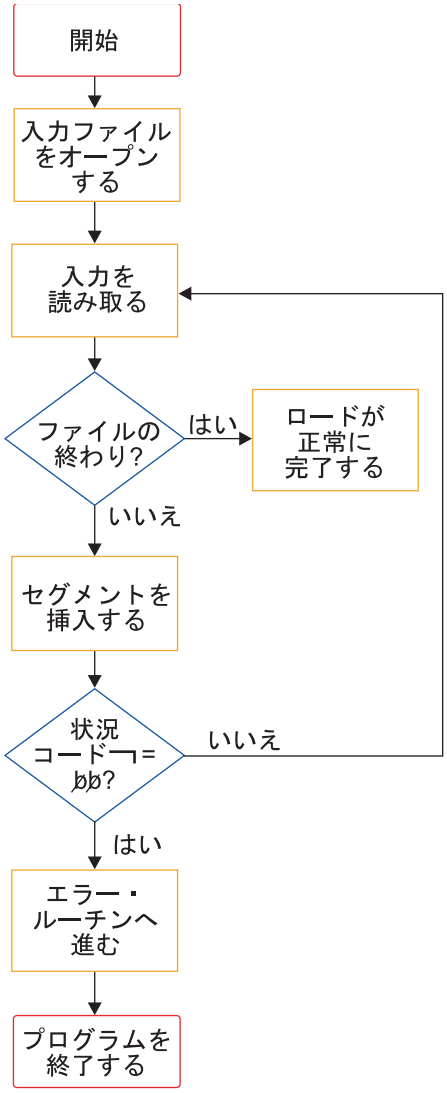


図 247. 基本初期ロード・プログラムの論理

以下のコードは、基本的な IMS データベース・ロード要件を満たしているサンプル・ロード・プログラムです。ユーティリティー制御機能を用いて、これをどのように行うかを示しているサンプル・プログラムも提供されています。

```
DLITCBL  START
         PRINT   NOGEN
         SAVE    (14,12),,LOAD1.PROGRAM   SAVE REGISTERS
         USING   DLITCBL,10               DEFINE BASE REGISTER
         LR      10,15                    LOAD BASE REGISTER
         LA      11,SAVEAREA              PERFORM
         ST      13,4(11)                  SAVE
         ST      11,8(13)                   AREA
         LR      13,11                       MAINT
         L       4,0(1)                   LOAD PCB BASE REGISTER
         STCM    4,7,PCBADDR+1            STORE PCB ADDRESS IN CALL LIST
         USING   DLIPCB,4                 DEFINE PCB BASE REGISTER
         OPEN    (LOAD,(INPUT))           OPEN LOAD DATA SOURCE FILE
LOOP     GET     LOAD,CARDAREA            GET SEGMENT TO BE INSERTED
INSERT   CALL    CBLTDLI,MF=(E,DLILINK)   INSERT THE SEGMENT
         AP      SEGCOUNT,=P'1'           INCREMENT SEGMENT COUNT
         CLC     DLISTAT,=CL2' '          WAS COMPLETION NORMAL?
         BE      LOOP                     YES - KEEP GOING
```

```
ABEND    ABEND   8,DUMP                    INVALID STATUS
EOF      WTO     'DATABASE 1 LOAD COMPLETED NORMALLY'
         UNPK    COUNTMSG,SEGCOUNT         UNPACK SEGMENT COUNT FOR WTO
         OI      COUNTMSG+4,X'F0'          MAKE SIGN PRINTABLE
         WTO     MF=(E,WTOLIST)            WRITE SEGMENT COUNT
         CLOSE   (LOAD)                    CLOSE INPUT FILE
         L       13,4(13)                  UNCHAIN SAVE AREA
         RETURN  (14,12),RC=0              RETURN NORMALLY
         LTORG
SEGCOUNT DC      PL3'0'
         DS      0F
WTOLIST  DC      AL2(LSTLENGT)
         DC      AL2(0)
COUNTMSG DS      CL5
         DC      C' SEGMENTS PROCESSED'
LSTLENGT EQU     (*-WTOLIST)
DLIFUNC  DC      CL4'ISRT'                 FUNCTION CODE
DLILINK  DC      A(DLIFUNC)                DL/I CALL LIST
PCBADDR  DC      A(0)
         DC      A(DATAAREA)
         DC      X'80',AL3(SEGNAME)
CARDAREA DS      0CL80                     I/O AREA
SEGNAME  DS      CL9
SEGKEY   DS      0CL4
DATAAREA DS      CL71
SAVEAREA DC      18F'0'
LOAD     DCB     DDNAME=LOAD1,DSORG=PS,EODAD=EOF,MACRF=(GM),RECFM=FB
DLIPCB   DSECT   ,                         DATABASE PCB
DLIDBNAM DS      CL8
DLISGLEV DS      CL2
DLISTAT  DS      CL2
DLIPROC  DS      CL4
DLIRESV  DS      F
DLISEGFB DS      CL8
DLIKEYLN DS      CL4
DLINUMSG DS      CL4
DLIKEYFB DS      CL12
         END
```

関連概念:

環境に合わせた IMS システムの調整 (システム定義)

関連資料:

定義ユーティリティーおよび初期設定ユーティリティー (データベース・ユーティリティー)

MSDB 保守ユーティリティー (DBFDBMA0) (データベース・ユーティリティー)

再始動可能初期ロード・プログラム

ロードしようとしているデータの量が多く、したがって、プログラムの実行中に問題が発生すれば大幅に後戻りしなければならない場合には、再始動可能初期ロード・プログラム (すなわち、最後のチェックポイントから再始動することができるもの) を作成する必要があります。

問題が発生して、プログラムが再始動可能プログラムでない場合には、ロード・プログラム全体を最初から再実行しなければなりません。

再始動可能ロード・プログラムでは、論理の点で、基本ロード・プログラムと異なっています。 すでに基本初期ロード・プログラムができあがっている場合には、こ

れを再始動可能プログラムにするには通常小さな変更を加えるだけで済みます。基本プログラムは、再始動がいつ行われるのか、処理を停止する要求を WTOR がいつ出したのか、およびチェックポイントがいつ取られたのかを認識できるように変更しなければなりません。

初期データベース・ロード・プログラムを UCF のもとで再始動可能にするには、プログラムを計画し作成する時点で以下の点を考慮してください。

- プログラムが再始動されている場合、最初に DL/I 呼び出しが出される前に、PCB 状況コードには UR が入っています。キー・フィードバック域には、最後の UCF チェックポイントが取られる前に挿入された最後のセグメントの完全な連結キーが入っています。 (障害が発生する前にチェックポイントが取られていない場合には、このエリアに 2 進数ゼロが入っています。)
- UCF は、ユーザー・ファイルのチェックポイントまたは再配置を行いません。再始動時は、このようなファイルすべての再配置は、ユーザーの責任です。
- 再始動時に出す最初の DL/I 呼び出しは、ルート・セグメントの挿入でなければなりません。

  HISAM および HIDAM 索引データベースの場合には、GN と VSAM ERASE という順序で再始動が始まり、より高位のキーを再挿入します。続いて、再開操作が行われます。 KSDS の中のスペースは再使用 (リカバリー) されますが、ESDS のスペースは再使用されません。

  HDAM の場合には、ルート・セグメントのシーケンス・フィールドが固有であるかどうかデータが比較され、チェックポイント後に挿入されたセグメントのためにデータベースの中にすでに存在しているセグメントのためのルート・セグメントが挿入されます。セグメント・データが同じ場合には、古いセグメントが重ね書きされ、従属セグメントは後続のユーザー/再ロード挿入により再度挿入されるので、この従属セグメントは除去されます。これは、重複していないルート・セグメントが見つかるまで続きます。新しいキーまたは異なるデータを持ったセグメントがいったん出てくると、その後の重複では LB 状況コードが戻されます。したがって、ルート・セグメントのスペースは再使用されますが、従属セグメントのスペースは失われます。

  HDAM でキーが非固有の場合には、再始動が行われたチェックポイント後に挿入されたルート・セグメントはデータベースの中に残ります。これは、その従属セグメントについても言えます。
- 停止要求を受け取った場合には、UCF は次のルート・セグメントを挿入する直前にチェックポイントを取ります。アプリケーション・プログラムが正しく停止できなかった場合には、その後でルート・セグメントを挿入するたびに、同じ状況コードがプログラムに与えられ、このプログラムが正常終了するまで続きます。
- HISAM データベースの場合には、RECOVERY オプションを指定しなければなりません。 HD 編成については、RECOVERY または SPEED のいずれかをアクセス方式サービスで定義することができます。
- チェックポイント・カウント (CKPNT=) が期限切れになっているときにルート・セグメントの挿入が要求されると、UCF チェックポイントが取られます。このカウントは挿入されたルート・セグメントの数を示しており、チェックポイントはルート・セグメントの挿入の直前に取られます。

以下の図は再始動可能初期ロード・プログラムの開発のための論理を示したものです。

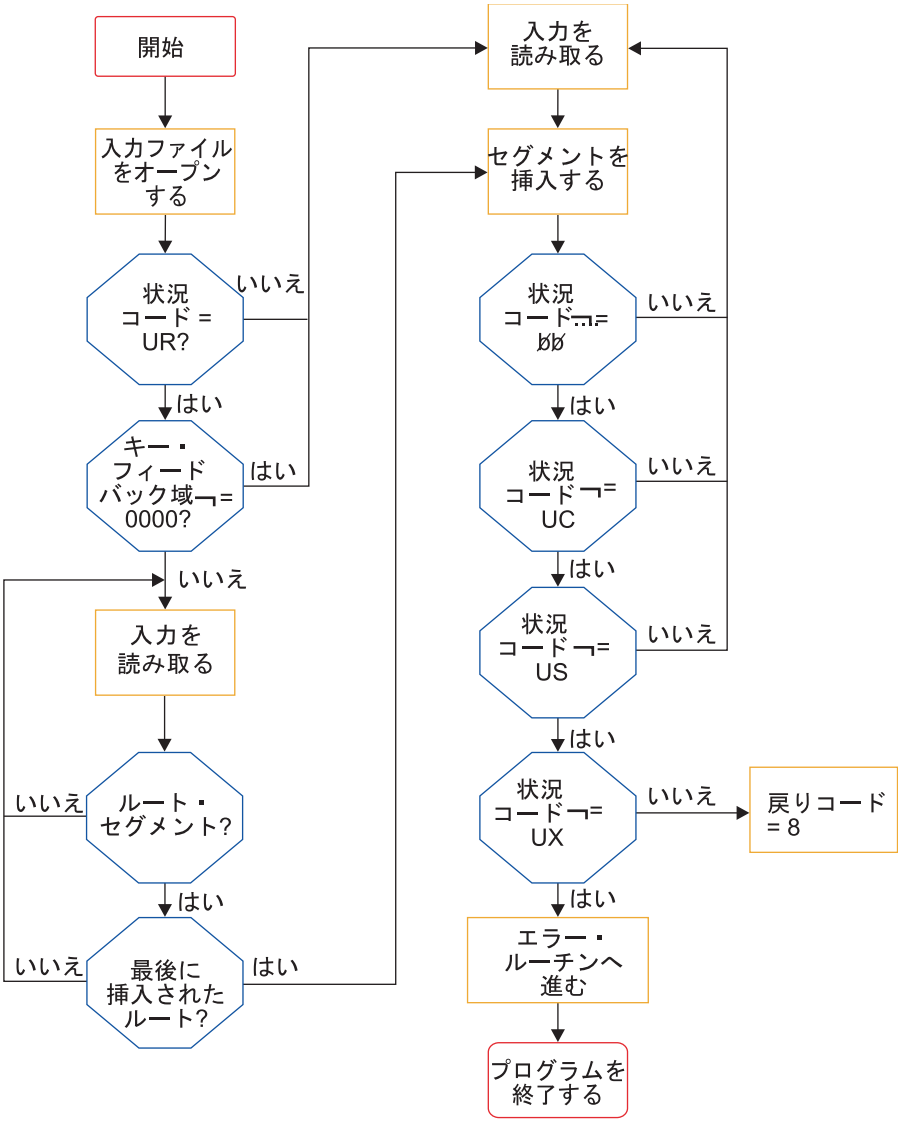


図 *248.* 再始動可能初期ロード・プログラムの論理

前の図に示した状況コードについて以下に説明します。

UR UCF の制御のもとでロード・プログラムが再始動されている。

UC チェックポイント・レコードが UCF ジャーナル・データ・セットに書き込まれた。

US 初期ロード・プログラムが処理の停止を準備している。

UX チェックポイント・レコードが書き込まれ、処理が停止した。

サンプルの再始動可能初期ロード・プログラムのコードを以下に示します。

```
DLITCBL  START
         PRINT   NOGEN
         SAVE    (14,12),,LOAD1.PROGRAM    SAVE REGISTERS
         USING   DLITCBL,10                DEFINE BASE REGISTER
```

```
         LR      10,15                     LOAD BASE REGISTER
         LA      11,SAVEAREA               PERFORM
         ST      13,4(11)                   SAVE
         ST      11,8(13)                    AREA
         LR      13,11                        MAINT
         L       4,0(1)                    LOAD PCB BASE REGISTER
         STCM    4,7,PCBADDR+1             STORE PCB ADDRESS IN CALL LIST
         USING   DLIPCB,4                  DEFINE PCB BASE REGISTER
         OPEN    (LOAD,(INPUT))            OPEN LOAD DATA SOURCE FILE
         CLC     DLISTAT,=C'UR'            IS THIS A RESTART?
         BNE     NORMAL                    NO - BRANCH
         CLC     DLIKEYFB(4),=X'00000000'  IS KEY FEEDBACK AREA ZERO?
         BE      NORMAL                    YES - BRANCH
RESTART  WTO     'RESTART LOAD PROCESSING FOR DATABASE 1 IS IN PROCESS'
RLOOP    GET     LOAD,CARDAREA             GET A LOAD RECORD
         CLC     SEGNAME(8),=CL8'SEGMA'    IS THIS A ROOT SEGMENT RECORD?
         BNE     RLOOP                     NO - KEEP LOOKING
         CLC     DLIKEYFB(4),SEGKEY        IS THIS THE LAST ROOT INSERTED?
         BNE     RLOOP                     NO - KEEP LOOKING
         B       INSERT                    GO DO IT
NORMAL   WTO     'INITIAL LOAD PROCESSING FOR DATABASE 1 IS IN PROCESS'
LOOP     GET     LOAD,CARDAREA             GET SEGMENT TO BE INSERTED
INSERT   CALL    CBLTDLI,MF=(E,DLILINK)    INSERT THE SEGMENT
         AP      SEGCOUNT,=P'1'            INCREMENT SEGMENT COUNT
         CLC     DLISTAT,=CL2' '           WAS COMPLETION NORMAL?
         BE      LOOP                      YES - KEEP GOING
         CLC     DLISTAT,=CL2'UC'          HAS CHECKPOINT BEEN TAKEN?
         BNE     POINT1                    NO - KEEP CHECKING
POINT0   WTO     'UCF CHECKPOINT TAKEN FOR LOAD 1 PROGRAM'
         UNPK    COUNTMSG,SEGCOUNT         UNPACK SEGMENT COUNT FOR WTO
         OI      COUNTMSG+4,X'F0'          MAKE SIGN PRINTABLE
         WTO     MF=(E,WTOLIST)            WRITE SEGMENT COUNT
         B       LOOP                      NO - KEEP GOING
POINT1   CLC     DLISTAT,=CL2'US'          HAS OPERATOR REQUESTED STOP?
         BNE     POINT2                    NO - KEEP CHECKING
         B       LOOP                      KEEP GOING
POINT2   CLC     DLISTAT,=CL2'UX'          COMBINED CHECKPOINT AND STOP?
         BNE     ABEND                     NO - GIVE UP
         WTO     'LOAD1 PROGRAM STOPPING PER OPERATOR REQUEST'
         B       RETURN8
ABEND    ABEND   8,DUMP                    INVALID STATUS
EOF      WTO     'DATABASE 1 LOAD COMPLETED NORMALLY'
         UNPK    COUNTMSG,SEGCOUNT         UNPACK SEGMENT COUNT FOR WTO
         OI      COUNTMSG+4,X'F0'          BLAST SIGN
         WTO     MF=(E,WTOLIST)            WRITE SEGMENT COUNT
         CLOSE   (LOAD)                    CLOSE INPUT FILE
         L       13,4(13)                  UNCHAIN SAVE AREA
         RETURN  (14,12),RC=0              RETURN NORMALLY
RETURN8  WTO     'DATABASE 1 LOAD STOPPING FOR RESTART'
         UNPK    COUNTMSG,SEGCOUNT         UNPACK SEGMENT COUNT FOR WTO
         OI      COUNTMSG+4,X'F0'          BLAST SIGN
         WTO     MF=(E,WTOLIST)            WRITE SEGMENT COUNT
         CLOSE   (LOAD)                    CLOSE INPUT FILE
         L       13,4(13)                  UNCHAIN SAVE AREA
         RETURN  (14,12),RC=8              RETURN AS RESTARTABLE
         LTORG
SEGCOUNT DC      PL3'0'
         DS      0F
WTOLIST  DC      AL2(LSTLENGT)
         DC      AL2(0)
COUNTMSG DS      CL5
         DC      C' SEGMENTS PROCESSED'
LSTLENGT EQU     (*-WTOLIST)
DLIFUNC  DC      CL4'ISRT'                 FUNCTION CODE
DLILINK  DC      A(DLIFUNC)                DL/I CALL LIST
PCBADDR  DC      A(0)
         DC      A(DATAAREA)
```

```
         DC      X'80',A13(SEGNAME)
CARDAREA DS      0CL80                      I/O AREA
SEGNAME  DS      CL9
SEGKEY   DS      0CL4
DATAAREA DS      CL71
SAVEAREA DC      18F'0'
STOPNDG  DC      X'00'
LOAD     DCB     DDNAME=LOAD1,DSORG=PS,EODAD=EOF,MACRF=(GM),RECFM=FB
DLIPCB   DSECT                              DATABASE PCB
DLIDBNAM DS      CL8
DLISGLEV DS      CL2
DLISTAT  DS      CL2
DLIPROC  DS      CL4
DLIRESV  DS      F
DLISEGFB DS      CL8
DLIKEYLN DS      CL4
DLINUMSG DS      CL4
DLIKEYFB DS      CL12
         END
```

関連資料:

ユーティリティー制御機能 (DFSUCF00) (データベース・ユーティリティー)

初期ロード・プログラムのための JCL

この JCL は、データベースを最初にロードするときに使用する JCL の例です。

//DFSURWF1 DD ステートメントは、論理関係または副次索引がある場合にのみ存在します。

データベースの初期ロードに使用される JCL

```
//          EXEC  PGM=DFSRRC00,PARM='DLI,your initial load program name,
//                your PSB name'
//DFSRESLB    DD  references an authorized library that contains IMS
                  SVC modules
//STEPLIB     DD  references library that contains your load program
//            DD  DSN=IMS.SDFSRESL
//IMS         DD  DSN=IMS.PSBLIB,DISP=SHR
//            DD  DSN=IMS.DBDLIB,DISP=SHR
//DFSURWF1    DD  DCB=(RECFM=VB,LRECL=300,
//                BLKSIZE=(you must specify),
//                DSN=WF1,DISP=(MOD,PASS)
//DBNAME      DD  references the database data set to be
                  initially loaded or referenced by
                  the initial load program
//INPUT       DD  input to your initial load program
//DFSVSAMP    DD  input for VSAM and OSAM buffers and options
.
.
.
//*
```

HISAM データベースのロード

HISAM データベースのセグメントは、ロード・プログラムに与えられた順序で保管されます。

ルート・セグメントのすべてのオカレンスをキーの昇順に従って与え、各ルート・セグメントのすべての従属セグメントを階層順に与えなければなりません。PCB において PROCOPT=L (ロードを表します) と指定しなければなりません。

SHISAM データベースのロード

SHISAM データベースのセグメントは、ロード・プログラムに与えられた順序で保管されます。

ルート・セグメントのすべてのオカレンスをキーの昇順で与えなければなりません。 PCB において PROCOPT=L (ロードを表します) と指定しなければなりません。

GSAM データベースのロード

GSAM データベースは、論理レコードを使用し、セグメントやデータベース・レコードは使用しません。 GSAM 論理レコードは、ロード・プログラムに与えられた順序で保管されます。

HDAM または PHDAM データベースのロード

HDAM または PHDAM データベースでは、ユーザーのランダム化モジュールがデータベース・レコードをどこに保管するかを決めます。したがって、ルート・セグメントをロード・プログラムに与える順序は問題になりません。

ある 1 つのルート・セグメントのすべての従属セグメントは、このルート・セグメントの後ろに階層順に続いている必要があります。PCB では、PROCOPT=L (ロードを表す) または PROCOPT=LS (昇順のロード・セグメントを表す) と指定しなければなりません。

HIDAM または PHIDAM データベースのロード

HIDAM または PHIDAM データベースをロードする場合には、ルート・セグメントをキーの昇順で与えなければならず、ある 1 つのルート・セグメントのすべての従属セグメントをこのルート・セグメントの後ろに階層の順で続けます。

PCB において PROCOPT=LS (昇順のロード・セグメントを表す) と指定しなければなりません。

制約事項: PHIDAM データベース用のロード・プログラムは、DLI 領域タイプで実行する必要があります。HIDAM データベース用のロード・プログラムには、この制約事項はありません。

論理関係または副次索引を持つデータベースのロード

論理関係または副次索引を持つデータベースをロードしようとする場合は、ユーザーのロード・プログラム以外に、いくつかの再編成ユーティリティーを組み合わせたものを実行する必要があります。

再編成ユーティリティーを実行して、各セグメントの接頭部に適切なポインター情報を入れる必要があります。

関連概念:

709 ページの『再編成ユーティリティー』

高速機能データベースのロード

このトピックでは、MSDB、DEDB および順次従属セグメントのロードの方法について説明します。

MSDB のロード

MSDB は主記憶に常駐するため、システム初期設定時にデータ・セットから読み取られてロードされます。

MSDB は、他の IMS データベースのようにユーザーが提供するロード・プログラムを使用してロードしません。最初にこのデータ・セットを構築するには、ユーザーが提供するプログラムを使用するか、あるいは MSDB 保守ユーティリティー (DBFDBMA0) を実行します。

関連概念:

235 ページの『MSDB の保管』

関連資料:

MSDB 保守ユーティリティー (DBFDBMA0) (データベース・ユーティリティー)

DEDB のロード

DEDB データベースにデータをロードするには、他の IMS データベースのロードに用いられているのと同様のロード・プログラムが用いられます。他のロード・プログラムとは異なり、このロード・プログラムは、バッチ・メッセージ・プログラムとして実行されます。

DEDB をロードするには、以下の 5 つのステップが必要です。

1. スペース必要量の計算

   次の例では、ルート・セグメント・タイプと順次従属セグメント・タイプが確実に 1 つのエリアにロードされているものとします。

   ルート・セグメントは、すべて長さが 200 バイトであり (198 バイトのデータと 2 バイトの長さフィールド)、このエリアの中に 850 のルート・セグメントがあるものとします。レコード当たり、平均 30 の SDEP セグメントがあります。このセグメントは、各 150 バイトの長さです (148 バイトのデータと 2 バイトの長さフィールド)。 CI サイズは、1024 バイトです。

   A. ルート・セグメントを収容するのに必要な最小のスペースを計算します。

```
1024               CI length minus
- 21               CI control fields
____               equals amount of space for root segments
1003               and their prefixes.

1003 / 214 = 4.6   Amount of root and root prefix space
                   divided by length of one root with its
                   prefix equals the number of segments
                   that will fit in one CI.
                   DEDB segments do not span CIs.
                   Therefore, only four
                   roots will fit in a CI.
```

```
 850 / 4 = 212.5    The minimum amount of space to hold
                    the defined number of roots to be
                    inserted in this area (850)
                    requires 213 CIs.
```

UOW サイズを選定すると、前述の計算結果を基礎として、ルート・アドレス可能部および独立オーバーフロー部に対する DBD の指定を決めることができます。

B. 順次従属セグメントを収容するのに必要な最小のスペースを計算します。

```
1024                CI length minus
- 17                CI control fields
____                equals amount of space for sequential
1007                dependents and their prefixes.

1007 / 160 = 6.2    Amount of sequential dependent and
                    prefix space divided by length of one
                    sequential dependent plus its prefix
                    equals the number of segments that
                    will fit in one CI.
                    Six SDEP segments will fit in a
                    CI.

30 / 6 = 5 CIs      Minimum amount of space required to
                    hold 30 sequential dependent
                    segments from one root.  For 850
                    roots, the minimum amount of space
                    required is 850 * 5 = 4250 CIs.
```

C. 以下のもののための追加スペースを計算に入れます。

- 「再編成 UOW」。これは通常の UOW と同じサイズです。
- ルート・アドレス可能部の先頭に割り振られる 2 つの制御データ CI。
- 独立オーバーフロー部の中の 120 の各 CI に 1 つずつの制御データ CI。

UOW のサイズを 20 CI とすれば、割り振る最小のスペースの量は、 213 + 4250 + 20 + 2 + 1 = 4486 CI となります。

2. この結果に従って DBD の仕様を決め、DBD 生成を実行します。
3. VSAM アクセス方式サービスを用いて、VSAM クラスターを割り振ります。

次の例は、後で DBDGEN で AREA1 と呼ばれるエリアの割り振り方を示したものです。

```
DEFINE -
   CLUSTER -
     (NAME (AREA1) -
     VOLUMES (SER123) -
     NONINDEXED -
     CYLINDERS (22) -
     CONTROLINTERVALSIZE (1024) -
     RECORDSIZE (1017) -
     SPEED) -
   DATA -
     (NAME(DATA1)) -
     CATALOG(USERCATLG)
```

エリアを定義するときには、以下のキーワードは特別に重要な意味を持ちます。

NAME
クラスターのために与える名前は、後でエリア名と呼ばれる名前です。データ部分の名前はオプションです。

NONINDEXED
DEDB エリアは索引なしクラスターです。

CONTROLINTERVALSIZE
VSAM ICIP の要件から与える値は、512、 1024、2048、または 4096 でなければなりません。

RECORDSIZE
レコード・サイズは CI サイズから 7 を引いた数です。この 7 バイトは、各 CI の末尾の VSAM 制御情報のために使用されます。

SPEED
パフォーマンス上の理由から、このキーワードの使用をお勧めします。

CATALOG
このオプション・パラメーターは、ユーザー・カタログを指定するために使用することができます。

4. DEDB 初期設定ユーティリティー (DBFUMIN0) を実行します。

   DBD の指定に従って各エリアのフォーマットを設定するために、このオフライン・ユーティリティーを実行しなければなりません。これによって、ルート・アドレス可能部と独立オーバーフロー部が割り振られます。 VSAM クラスターの中に残されたスペースは、順次従属部のために予約されます。このユーティリティーの 1 回の実行の間に、最高 2048 のエリアを指定することができます。ただし、エリアの初期設定は順次に行われます。実行の後、統計情報報告書をスペース計算結果に照らして検査してみてください。

5. ユーザーの DEDB ロード・プログラムを実行します。

   DEDB をロードするために BMP プログラムが使用されます。 DEDB のロード中に使用されるランダム化ルーチンが、特定の範囲のデータを DEDB の特定のエリアに差し向けるように調整されていることもあります。

   ロード操作が失敗に終わった場合には、このエリアをスクラッチし、再び割り振って、これを初期設定しなければなりません。

順次従属セグメントのロード

順次従属セグメントの順序が重要であれば、順次従属セグメントを DEDB にロードする方法について考える必要があります。

次に 2 つの代替案を挙げておきます。

- ルート・セグメントとその順次従属セグメントを追加する。

  ある 1 つのルート・セグメントのすべての順次従属セグメントは物理的に同じ所に書き込まれますが、これらの順次従属セグメントの物理的順序は、当初のデータ入力の順序を反映してはいません。この反映は、従属セグメントが主として

トランザクションのジャーナルとして使用されているのであれば、必ずしもアプリケーションが従属セグメントを見るべき見方ではありません。

- すべてのルート・セグメントを追加し、次いで、順次従属セグメントを追加する。

  この手法により、SDEP セグメントの順序は当初の入力順に復元されます。しかし、各 SDEP セグメントを追加するたびにそれぞれのルート・セグメントにアクセスする必要があるので、この方法では処理時間が長くなります。

副次索引を持つ HALDB のロード

デフォルトで、副次索引を持つ HALDB データベースの初回ロードの実行時に、ロード処理の一環として副次索引が作成されます。

ソース・セグメントが HALDB データベースにロードされる際に、ソース・セグメントの副次索引項目が副次索引区画に挿入されます。ロード・プログラムによって使用されるバッファー・プールには、副次索引用のバッファーが含まれていなければなりません。副次索引区画は、データベースをロードする前に初期化する必要があります。

すべての副次索引項目は、その拡張ポインター・セット (EPS) 内に正確な RBA ポインターを持っているため、初期ロードでは、間接リスト・データ・セット (ILDS) 内に項目は作成されません。

HALDB データベースの初期ロード時に副次索引が作成されないようにするには、IMS.PROCLIB データ・セットの DFSVSM*xx* メンバーか、または DFSVSAMP データ・セットで、OPTIONS 制御ステートメントに BLDSNDX=NO を指定します。例えば、次のようになります。

```
DFSVSAMP DD *
OPTIONS,BLDSNDX=NO
VSRBF=4096,500
IOBF=(8192,200)
/*
```

BLDSNDX=NO を指定する場合は、他の方法によって副次索引を作成する必要があります。これには、IBM IMS Index Builder for z/OS などのツールを使用することができます。

BLDSNDX=NO を使用すると、副次索引区画は許可されず、そのデータ・セットが割り振られません。

BLDSNDX=NO を指定して索引ビルダー・ツールを使用すると、特に、副次索引項目が多い場合は、ロードおよび索引作成処理に要する時間を短縮できます。ロード処理中に HALDB 副次索引が作成される場合、副次索引への挿入はランダムであるため、ロード処理時間がかなり長くなる可能性があります。IMS Index Builder のような索引ビルダー・ツールは、ロード済みのデータベースを読み取り、索引項目を副次索引に書き込まずに作成し、副次索引キー・シーケンスで項目をソートして、順次処理で副次索引に書き込みます。

関連タスク:

786 ページの『OPTIONS 制御ステートメントで指定されている VSAM オプションの調整』

関連資料:

- VSAM パフォーマンス・オプションの定義 (システム定義)

第 26 章 データベースのバックアップおよびリカバリー

障害発生後にデータベースのリカバリーが成功するかどうかは、エラーが発生する前に行う計画と準備によって決まります。データベース・バックアップ・コピーの作成は、この準備の重要な一部分です。

関連概念:

194 ページの『HIDAM 1 次索引と PHIDAM 1 次索引のバックアップおよびリカバリー』

153 ページの『HD データベースを対象として発行可能な DL/I 呼び出し』

540 ページの『イメージ・コピーのオプション』

200 ページの『HALDB 区画のデータ・セットおよびリカバリー』

データベース障害

IMS が DL/I 入出力エラーを検出するたびに、 IMS は、エラーのあるブロックまたは VSAM 制御インターバルを識別する拡張エラー・キュー・エレメント (EEQE) を作成します。

IMS データベースにエラーがあると、IMS は、メッセージ DFS0451I または DFS0451A を出します。

IMS は、データベースをクローズする際、DL/I データベース上の読み取りエラーおよび書き込みエラーを自動的に再試行します。正常であれば、データベースの順方向リカバリーは必要ありません。正常でない場合は、結局は順方向リカバリーが必要です。リカバリーを、もっと都合の良いときに遅らせることもできる場合があります。リカバリーを遅らせても、アクセスのスケジューリングや更新が禁止になることはありません。

DEDB 複数エリア・データ・セットを使用すると、アプリケーション・プログラムは、入出力エラーが存在しても続行することができます。 DEDB 入出力エラーの場合、IMS は、メッセージ DFS2571、DFS2572、DFS3712、および DFS3713 を出します。DEDB エリアが使用できない場合、アプリケーションは FH 状況コードを受け取ります。

IMS は、再始動を通して入出力情報とバッファー・イメージを維持します。IMS は、EEQE を DBRC に記録し、IRLM を使用するすべてのシステムに通知し、初期設定およびチェックポイントの間に EEQE と仮想バッファーを OLDS にロギングすることによって、これを行います。

関連概念:

682 ページの『DL/I 入出力エラーおよびリカバリー』

DFS メッセージ (メッセージおよびコード)

関連タスク:

DB - データベース保守援助プログラム (診断)

データベース書き込みエラー

IMS アプリケーション・プログラムは、データベースの書き込みエラーを認識しません。IMS は、書き込みオペレーションが完了しても、完了できなくても、アプリケーション・プログラムに戻りコードを渡しません。

代わりに、書き込みエラーが発生すると、IMS は拡張エラー・キュー・エレメント (EEQE) を作成し、エラーのブロックまたは制御インターバル用のバッファーを割り振り、バッファーに関する情報をログ・レコードに書き込みます。

IMS は、データベースの更新や読み取り要求を含む、ブロックまたは CI への次の入出力のすべてにこのバッファーを使用します。IMS は、データベースをクローズする際に、失敗した元の書き込みオペレーションを再試行します。それが正常であれば、IMS は、バッファーを解放し、対応する EEQE を DBRC から除去し、そしてすべてのサブシステムにその対応する EEQE を無視するように通知します。

IMS が作成できる EEQE の数には制限がありません。高速機能領域については、領域は 100 EEQE の後で停止します。全機能データベースについては、制限は 32,767 EEQE です。この制限に触れる前に、複数の EEQE を書き込むために必要となるバッファーに対する仮想記憶域の制限に触れることが考えられます。

リカバリーは数週間も遅らせることができますが、リカバリーの遅延が長すぎると、IMS がリカバリーの実行に要する時間が長くなることがあります。

IMS は、DBRC に各 EEQE について通知します。DBRC は各 EEQE を、RECON データ・セットの DBRC レコードに記録します。

データベース読み取りエラー

読み取りエラーが発生すると、IMS は、アプリケーション・プログラムに AO 状況コードを戻します。IMS は、エラーのデータにも EEQE を作成しますが、そのためのバッファーは作成しません。

データベースの静止

データベース、DEDB エリア、HALDB 区画、またはデータベース・グループを静止することで、複数のデータベースとログにわたってリカバリー・ポイントを作成できるほか、データベースをオフラインにすることなくクリーンなデータベース・イメージ・コピーを作成できます。また、データベース変更累積ユーティリティー (DFSUCUM0) のパフォーマンスを向上させることもできます。

データベースが静止しているときは、データベースに対する進行中の更新がなく、それ以前のすべての更新がコミットされて DASD に書き込まれています。また、データベースに対して新たな更新を行うアプリケーション・プログラムは、データベースの静止状態が解除されるまで待ち状態になります。進行中の最後の更新がデータベースでコミットされると、データベースは完全に静止します。

静止状態に達すると、DBRC によって、RECON データ・セット内のデータベース、エリア、または区画用の割り振り (ALLOC) レコードに割り振り解除 (DEALLOC) タイム・スタンプが記録されます。DEALLOC タイム・スタンプは、

リカバリーおよびリカバリー・ユーティリティー (データベース・リカバリー・ユーティリティー (DFSURDB0) や DFSUCUM0 ユーティリティーなど) でリカバリー・ポイントとして使用されます。

データベースを静止することで、データベース・データ・セットが変更されることはありません。静止機能は、データベース・データ・セットを、静止機能が開始されたときと同じ状態のまま残します。

リカバリー・ポイントを作成する他の方法 (/DBRECOVERY コマンドの実行など) とは異なり、データベースの静止では 1 つのコマンドを実行するだけで済みます。IMS は、進行中の更新がコミットされると自動的にアプリケーション・プログラムを待ち状態にし、静止が解除されると自動的にデータベースへのアクセスを復元します。/DBRECOVERY コマンドを使用してリカバリー・ポイントを作成する場合は、データベース・アクティビティーをすべて停止し、データベースを不許可にしてオフラインにし、後からすべてを再始動する必要があります。

データベース静止機能は相対的に操作が容易で処理速度が速く、データベースの可用性に与える影響が最小限であるため、より少ないコストで頻繁にリカバリー・ポイントを作成できます。リカバリー・ポイントを頻繁に作成すると、データベース変更累積ユーティリティー (DFSUCUM0) でマージしなければならないログ・レコードの数が減るので、リカバリーに要する時間が短くなり、このユーティリティーのパフォーマンスが向上します。

静止機能をサポートするデータベース・タイプ

静止できるのは、高速機能 DEDB データベース、全機能データベース、データベース・グループ、DEDB エリア、および HALDB 区画です。

具体的には、次のタイプのデータベースが静止機能をサポートします。

- DEDB
- HDAM
- HIDAM
- HISAM
- HSAM
- PHDAM
- PHIDAM
- PSINDEX
- SHISAM
- SHSAM

GSAM および MSDB データベースは、静止機能をサポートしません。

DEDB データベースの場合は、データベース全体 (つまり DEDB 内の全エリア) を静止するか、または 1 つ以上のエリアのサブセットを直接静止できます。同様に、HALDB PHDAM、PHIDAM、および PSINDEX の各データベースの場合は、HALDB マスター・データベース (つまりデータベース内の全区画) を静止するか、または 1 つ以上の区画のサブセットを直接静止できます。DEDB データベースまた

は HALDB データベースをデータベース・レベルで静止すると、データベース・レコードではなく、各エリアまたは区画のレコードで静止状況が維持されます。

データベースへのアクセスの管理、およびリカバリーに使用されるタイム・スタンプの記録には DBRC が必要であるため、登録されていないデータベースを静止することは、技術的には可能ですが、お勧めできません。

静止のオプション

データベースを静止する場合は、次のオプションを選択できます。

- リカバリー・ポイントを作成するために必要な時間だけデータベースを静止できます。
- イメージ・コピー・ユーティリティーまたはデータベース変更累積ユーティリティー (DFSUCUM0) を実行するために、データベースの静止状態を無期限に保持できます。データベースの静止状態を解除するには、STOP(QUIESCE) キーワードを指定して適切な UPDATE コマンドを実行します。

  STOP(QUIESCE) キーワードを指定できる UPDATE コマンドは、次のとおりです。

  - UPDATE AREA
  - UPDATE DATAGRP
  - UPDATE DB
- IMS が静止処理を取り消すまでアプリケーション・プログラムによる更新のコミットを待つタイムアウト間隔を指定できます。

リカバリー・ポイントを作成するためにデータベースを静止した場合は、データベースが整合点に到達した時点で直ちに IMS がデータベースの静止状態を解除し、アプリケーション・プログラムがデータベースの更新を再開できるようになります。

リカバリー・ポイントを作成する目的でデータベース、エリア、区画、またはデータベース・グループを一時的に静止するには、適切な UPDATE コマンドで START(QUIESCE) キーワードを指定します。例えば、`UPDATE DB NAME(DBXYZ) START(QUIESCE)` と指定します。

データベースを静止状態に保持すると、STOP(QUIESCE) キーワードを指定した適切な UPDATE コマンドを実行して静止状態を解除するまで、データベースは静止状態のままとなります。データベースが静止状態に保持されている間は、データベース・イメージ・コピー 2 ユーティリティー (DFSUDMT0) などのオンライン・イメージ・コピー・ユーティリティー、またはデータベース・イメージ・コピー・ユーティリティー (DFSUDMP0) などのバッチ・イメージ・コピー・ユーティリティーを実行して、データベースをオフラインにすることなくクリーン・イメージ・コピーを作成できます。

データベースを静止して静止状態に保持するには、適切な UPDATE コマンドで START(QUIESCE) と OPTION(HOLD) の両方のキーワードを指定します。例えば、`UPDATE DB NAME(DBXYZ) START(QUIESCE) OPTION(HOLD)` と指定します。

静止機能のタイムアウト間隔では、静止機能が開始されたときに進行中であったアプリケーション・プログラムによるデータベースへの更新がコミットされるまで静止機能が待つ時間を規定します。この時間間隔が終了した時点で更新がまだ進行中である場合は、データベースの静止が取り消されるため、データベースを静止するには再び START(QUIESCE) オプションを指定して適切な UPDATE コマンドを実行しなければなりません。

静止機能のデフォルトのタイムアウト値は 30 秒です。静止機能に別のタイムアウト値を規定する場合は、DFSCGxxx PROCLIB メンバーで DBQUIESCETO パラメーターを使用します。タイムアウト値は、静止機能開始時に適切な UPDATE コマンドで SET(TIMEOUT(nnn) パラメーターを指定することでオーバーライドできます。

データベースの静止がアプリケーション・プログラムに与える影響

データベース、エリア、または区画の静止が進行中または保持されている間は、アプリケーション・プログラムをスケジュールすることは可能ですが、アプリケーション・プログラムからデータベース、エリア、または区画へのアクセスは読み取りのみが許可されます。更新のためにデータベースにアクセスする必要があるアプリケーション・プログラムは、静止が解除されるまで待ち状態になります。

アプリケーション・プログラムがデータベースを更新しているときに静止が開始されると、静止機能はアプリケーション・プログラムが進行中の更新をコミットして更新が DASD に書き込まれるまで待機します。進行中の更新をコミットした後にさらにデータベースを更新するアプリケーション・プログラムは、静止が解除されるまで待ち状態になります。静止機能に指定されているタイムアウト間隔が終了するまでにアプリケーション・プログラムが更新をコミットしなかった場合は、データベースの静止が取り消されます。

データベースを静止状態に保持すると、従属領域で待機するアプリケーション・プログラムの数が増え、従属領域の数がインストール済み環境で許可されている最大数に達する可能性があります。

多数のアプリケーション・プログラムが待ち状態になることで発生する問題を回避するために、データベースを静止状態に保持する期間は、低アクティビティーで、かつ必要な作業 (イメージ・コピーの作成など) を実行するために必要な期間に限定してください。

DBRC、RECON データ・セット、および静止機能

IMSplex 内で静止機能を確実に調整するには、静止機能に関与する IMS システムに属する DBRC インスタンスが、固有の DBRC グループ ID で定義されている同一の DBRC グループに含まれていること、および Common Service Layer (CSL) の Structured Call Interface (SCI) に登録されていることが必要です。

DBRC は、データベース、DEDB エリア、または HALDB 区画の静止状況を RECON データ・セットに記録します。静止機能が実行されると、DBRC は、データベース、エリア、または区画の静止が進行中であることを示すために RECON データ・セットを更新します。データベース、エリア、または区画が静止して、静止

状態に保持されると、DBRC は、データベース、エリア、または区画の静止が進行中であり、かつ静止状態に保持されていることを示すために RECON データ・セットを更新します。

データベース、エリア、または区画が静止している間は、RECON データ・セット内でそのレコードのリストに QUIESCE IN PROGRESS と示されます。データベース、エリア、または区画が静止状態に保持されている間は、RECON データ・セット内でレコードのリストに QUIESCE IN PROGRESS と QUIESCE HELD の両方が示されます。

データベース、エリア、または区画の静止が進行中でありながら、静止状態に保持されていない期間は、DBRC がデータベース、エリア、または区画に対する許可とアクセスを以下のように管理します。

- 許可されないもの
  - 更新または排他のアクセス・インテントを必要とするユーティリティー
  - イメージ・コピー・ユーティリティー
  - 更新アクセスを行うバッチ・アプリケーション・プログラム
- 許可されるが、データベース、エリア、または区画にはアクセスできないもの
  - 静止に関与していない IMS システム
  - 更新特権を持つアプリケーション・プログラムがあるオンライン IMS システム
- 許可され、かつデータベース、エリア、または区画にアクセスできるもの
  - 読み取り専用インテントが設定されているバッチ・アプリケーション・プログラム
  - 読み取り専用特権を持つアプリケーション・プログラムがあるオンライン IMS システム
  - 読み取り専用インテントを必要とするユーティリティー (イメージ・コピーを除く)

データベース、エリア、または区画が静止状態に保持されているときは、DBRC が次のようにデータベース、エリア、および区画の許可とアクセスを管理します。

- 許可されないもの
  - 更新または排他のアクセス・インテントを必要とするユーティリティー
  - 更新権限を持つバッチ・アプリケーション・プログラム
- 許可されるが、データベース、エリア、または区画にはアクセスできないもの
  - 静止に関与していない IMS システム
  - 更新権限を持つアプリケーション・プログラムがあるオンライン IMS システム
- 許可され、かつデータベース、エリア、または区画にアクセスできるもの
  - 読み取り専用アクセス権限を必要とするユーティリティー
  - イメージ・コピー・ユーティリティー
  - 読み取り専用権限を持つバッチ・アプリケーション・プログラム (アプリケーション・プログラムに対する PCB ステートメントの PROCOPT パラメーターで G、GO、GOx のいずれかの値が指定されている)

– 読み取り専用権限を持つアプリケーション・プログラムがあるオンライン IMS システム

RECON データ・セットにオープン ALLOC レコードがあるデータベースが静止すると、IMSplex 全体で静止が達成された時点と一致する DEALLOC タイム・スタンプで ALLOC レコードがクローズされます。ALLOC レコードには、データベースが静止されたために割り振り解除されたことが示されます。クローズされた ALLOC レコード内の DEALLOC タイム・スタンプは、その後、データベース・リカバリー・ユーティリティー (DFSURDB0) およびタイム・スタンプ・リカバリーでリカバリー・ポイントとして使用できます。

データベース、エリア、または区画に対して新しい ALLOC レコードが作成されると、新しい更新セット ID (USID) およびデータ・セット・シーケンス番号 (DSSN) が割り当てられます。

静止が解除されると、DL/I 呼び出しで再びデータベースにアクセスできるようになります。DEDB エリアについては、RECON データ・セットに新しい ALLOC レコードが作成されます。全機能データベースについては、最初の更新でデータベースがアクセスされるまで新しい ALLOC レコードは作成されません。

静止機能の要件と制約事項

データベース静止機能を使用するには、以下の CSL マネージャーが使用可能になっている IMSplex 環境が必要です。

- Operations Manager (OM)
- Resource Manager (RM) (IMSplex 環境に IMS システムが 1 つしかなく、かつ DFSCGxxx PROCLIBメンバーで、または DFSDFxxx PROCLIB メンバーの CSL セクションで RMENV=N が指定されている場合を除く)。
- 構造化呼び出しインターフェース (SCI) (Structured Call Interface (SCI))

DBRC コマンド CHANGE.RECON の MINVERS キーワードを使用して、RECON データ・セットに最小バージョンとして 11.1 以上を指定する必要があります。

リモート・サイト・リカバリー (RSR) 構成では、リモート IMS システムによって静止コマンドがトラッキングされません。

以下の環境では、静止を開始しても失敗します。

- 静止機能を開始したときに、更新権限を持つバッチ・アプリケーション・プログラムがデータベースにアクセスしていた場合。ただし、読み取り専用権限を持つバッチ・アプリケーション・プログラムであれば、静止機能は失敗しません。
- データベースのリカバリーまたはバックアウトが必要である場合。
- RECON データ・セット内でデータベース、エリア、または区画のレコードに再編成インテント・フラグが設定されている場合。
- HALDB オンライン再編成機能が開始されていて、かつターゲットの HALDB 区画がカーソル・アクティブの状態にあるか、または静止しようとする区画に対して IMS システムがオンライン再編成機能を所有している場合。
- データベースの静止がすでに進行中である場合。

- 静止しようとする HALDB 区画が初期設定を必要としている場合。
- データベースに対して次のいずれかのコマンドが実行され、処理が完了していない場合。
  - /DBD DB
  - /DBR AREA
  - /DBR DB
  - /INIT OLREORG
  - INIT OLREORG
  - /STA AREA
  - /STA DB
  - /STO AREA
  - /STO DB
  - UPDATE AREA
  - UPDATE DATAGRP
  - UPDATE DB

関連概念:

650 ページの『非並行イメージ・コピー』

関連タスク:

665 ページの『リカバリーのためのデータベース変更累積入力の使用』

関連資料:

UPDATE コマンド (コマンド)

データベース・バックアップ・コピーの作成

このトピックでは、データベースのバックアップ・コピーの作成方法を説明します。ここでは、ユーティリティーとそれらがどのように IMS オペレーションに影響を及ぼすのかについて述べます。

関連概念:

メッセージ・キューのバックアップ・コピー (システム管理)

システム・データ・セットのバックアップ・コピー (システム管理)

イメージ・コピーおよび IMS イメージ・コピー・ユーティリティー

データベースのバックアップ・コピーは、イメージ・コピーと呼ばれます。イメージ・コピーは、IMS 付属のイメージ・コピー・ユーティリティーのいずれかを用いて作成できます。使用するユーティリティーに応じて、データベースがオンライン状態、オフライン状態、または静止状態のときにイメージ・コピーを作成できます。

IMS には、イメージ・コピーを作成するのに使用できる以下の 3 つのユーティリティーが用意されています。

- データベース・イメージ・コピー・ユーティリティー (DFSUDMP0)

- オンライン・データベース・イメージ・コピー・ユーティリティー (DFSUICP0)
- データベース・イメージ・コピー 2 ユーティリティー (DFSUDMT0)

制約事項: HALDB オンライン再編成 (OLR) によって再編成中の HALDB 区画に対しては、これらのユーティリティーを実行することはできません。

これらのユーティリティーは、リカバリー可能およびリカバリー不能データベースのイメージ・コピーを作成します。イメージ・コピー・ユーティリティーは、データ・セット上で操作します。

イメージ・コピーを作成する頻度は、リカバリー要件によって決まります。最低限必要なことは、データベースを再編成、再ロード、または初期ロードした直後にイメージ・コピーを作成することです。データベース・リカバリーは物理的な置き換えによって行われるため、再ロードされたデータ・セットは、アンロード前の状態と物理的に同じではありません。

データベース・イメージ・コピー・ユーティリティー (DFSUDMP0) およびオンライン・データベース・イメージ・コピー・ユーティリティー (DFSUICP0) では、データベースが複数のデータ・セットまたは領域を含んでいる場合は、ユーティリティーに対して複数の指定を提供する必要があります。ただし、データベース・イメージ・コピー 2 ユーティリティーについては、ユーティリティーの 1 回の実行に対して複数のデータベース・データ・セットをコピーすることが可能です。グループ名を指定して、1 回の実行でコピーされるデータベース・データ・セットの集合を表示することができます。

DBRC コマンド INIT.DB および CHANGE.DB を使用して、DBRC にリカバリー可能データベースを識別させてください。DBRC は、すべてのイメージ・コピー・ユーティリティーと一緒にそれらと同じように動作します。

データベース・イメージ・コピー・ユーティリティー (DFSUDMP0) およびオンライン・データベース・イメージ・コピー・ユーティリティー (DFSUICP0) からの出力データ・セットは、同じフォーマットです。イメージ・コピー・データ・セットの事前定義と再利用に関する規則は、この両方のユーティリティーで使用されるデータ・セットに適用されます。

データベース・イメージ・コピー 2 ユーティリティーを使用する場合は、使用するイメージ・コピー・オプションによって、出力データ・セットのフォーマットが異なります。データベース・イメージ・コピー 2 ユーティリティーの並行コピー機能を使用して生成されるイメージ・コピーは、DFSMSdss ダンプ・フォーマットです。データベース・イメージ・コピー 2 ユーティリティーの高速複製機能を使用して生成されるイメージ・コピーは、元のデータ・セットの正確なコピーです。

データベース・イメージ・コピー 2 ユーティリティーを使用する場合は、イメージ・コピー・データ・セットの事前定義と再利用に関する DBRC の規則が、並行コピー機能に使用されるデータ・セットだけに適用されます。データベース・イメージ・コピー 2 ユーティリティーの高速複製機能は、イメージ・コピー・データ・セットの事前定義と再利用をサポートしません。

これらのすべてのユーティリティーは、DBRC を呼び出して入力を検査し (DBRC がこれらの実行を許可するのは入力が有効な場合だけです)、また DBRC を呼び出

して、これらが作成したイメージ・コピー・データ・セットに関する情報を RECON データ・セットに記録します。RECON データ・セット内のイメージ・コピー・レコードのフォーマットは、そのレコードに対応するイメージ・コピー・データ・セットをどのユーティリティーが作成したかとはかかわりなく、すべて同じです。ただし、オンライン・データベース・イメージ・コピー・ユーティリティーは基幹 JCL については独自の PDS メンバーをもっているため、オンライン・データベース・ イメージ・コピー・ユーティリティー用のジョブを生成する際には別個のDBRC コマンド GENJCL.OIC を使用してください。

推奨事項: 1 つのデータベースのデータ・セットまたはエリアは、すべて同時にコピーするようにしてください。あるデータベースを以前の状態にリカバリーする際には、データ保全性に関する問題を回避するために、そのデータベースに属するすべてのデータ・セットならびに論理的に関連するすべてのデータベース (アプリケーション処理で関連付けられているものも含む) を同じポイントでリカバリーする必要があります。

イメージ・コピー・ユーティリティーを使用する場合は、DFSUDMT0 の高速複製機能を使用する場合を除いて、1 つ以上の出力イメージ・コピーを作成することができます。複数のコピーを作成する利点は次のとおりです。

- 1 つのコピーに入出力エラーが発生した場合に、ユーティリティーがそれ以外のコピーを使用して完了するまで処理を継続できる。
- 1 つのコピーが読み取り不能であっても、別のコピーを使用してリカバリーを実行できる。

複数のコピーを作成するかどうかを決定する際に検討すべきトレードオフは、他のコピーを書き込むために必要な時間のためにイメージ・コピー・ユーティリティーのパフォーマンスが低下することです。

データベース・イメージ・コピー 2 ユーティリティーの高速複製オプションを使用する場合は、ユーティリティーの 1 回の実行で各ソース・データベース・データ・セットの出力イメージ・コピーを 1 つしか作成できません。

各ユーティリティーは、出力レコードの長さが最小限でも 64 バイトを下回らないように強制します。したがって、論理レコード長が非常に短いデータベースのイメージ・コピーがオリジナルのデータベースよりも大きなスペースを必要とする可能性があります。

推奨事項: IMS ロギングしないでデータベースを更新するバッチ・ジョブの実行後は、ただちにイメージ・コピーを作成してください。プール・サイズが同一であっても、システムが VSAM バックグラウンド書き込みを使用している場合は、バッチ・ジョブによる更新をビット単位で反復することはできません。リカバリーが必要な場合に、イメージ・コピーを作成すれば、データベースの保全性を維持できます。

ログ・テープを使用してバッチ・ジョブ開始時点までリカバリーしてからバッチ・ジョブを再処理した場合、結果のデータベースは、前のバッチ実行後のデータベースと論理的には同一でも、ビット単位では同一でない場合があります。データベースがビット単位で同一でないと、前のバッチ・ジョブの実行後に作られたログ・テープは、再処理後は有効でなくなります。したがって、リカバリーを試みる場合

に、イメージ・コピーで開始し、ログ・テープを適用し、ログに記録されていないバッチ実行を再処理し、その後で別のログ・テープを適用する方法は採らないでください。

いずれかの IMS イメージ・コピー・ユーティリティーによって作成されていないイメージ・コピーは、標準外 イメージ・コピーまたはユーザー・イメージ・コピーと呼ばれます。

関連タスク:

658 ページの『標準外イメージ・コピー・データ・セット』

並行イメージ・コピー

IMS では、DBRC に登録されているデータベースのイメージ・コピーを、データベースの更新中に作成することができます。

データベースの更新中に作成されたイメージ・コピーは、並行イメージ・コピー と呼ばれます。また、このコピーはデータベースのある瞬間の状態ではなく、ある期間の状態を表すことから、ファジー・イメージ・コピー とも呼ばれます。並行イメージ・コピーを作成すると、コピー処理中にデータベースに対して行われた更新の一部または全部が最終的なコピーに反映される場合もあれば、更新がまったく反映されないこともあります。

IMS データベース・イメージ・コピー 2 ユーティリティー (DFSUDMT0) が提供する並行コピー・オプションを、並行イメージ・コピーと混同しないようにしてください。DFSUDMT0 ユーティリティーの並行コピー・オプションでは、並行イメージ・コピーまたはクリーン・イメージ・コピーのいずれかを作成できます。DFSUDMT0 ユーティリティーの並行コピー・オプションという名前は、z/OS DFSMSdss DUMP コマンドの CONCURRENT キーワードに由来しており、このオプションによって作成されるイメージ・コピーのタイプを意味するわけではありません。DFSMS および 3990 ハードウェアの並行コピー機能によって、データ・セットがオフライン、オンラインのいずれの状態でも、データ・セットのコピーを作成できます。

DFSUDMT0 ユーティリティーの高速複製オプションでも、並行イメージ・コピーまたはクリーン・イメージ・コピーを作成することができます。

制約事項:

- IMS イメージ・コピー・ユーティリティーでは、DBRC に登録されているデータベースの並行イメージ・コピーのみを作成することができる。
- リカバリー不能データベースのコピーを作成することはできるが、イメージ・コピーを作成するためにユーティリティーを実行する前にそれらを停止する必要がある。IMS はリカバリー不能データベースへの変更はログに記録しないため、リカバリー不能データベースがオンラインの間はそれらの並行イメージ・コピーを作成することはできません。リカバリー不能データベースのファジー・イメージ・コピーがそのデータベースをリカバリーするのに使用された場合は、そのデータベース自体がファジーとなります。
- データベース・イメージ・コピー・ユーティリティー (DFSUDMP0) を使用している場合は、並行イメージ・コピーは OSAM および VSAM 入力順データ・セット (ESDS) DBDS のものしか作成できず、VSAM キー順データ・セット

(KSDS) の並行イメージ・コピーはサポートされない。これ以外の 2 つのイメージ・コピー・ユーティリティーのいずれかを使用する場合は、ESDS または KSDS のイメージ・コピーを作成できます。

関連タスク:

681 ページの『並行イメージ・コピーのリカバリー』

非並行イメージ・コピー

非並行イメージ・コピーとは、データベースへの更新が行われていないときに取られるイメージ・コピーです。

非並行イメージ・コピーは、データベースがオフラインのときに作成される場合が多いことからバッチ・イメージ・コピーと呼ばれたり、ファジー・データを一切含んでいないことからクリーン・イメージ・コピーと呼ばれたりします。イメージ・コピーに反映されるデータはすべてコミットされているため、イメージ・コピーを単独でリカバリーの開始点として使用することができます。データ・セットのリカバリーに必要となるのは、イメージ・コピー作成後に行われたデータ・セットに対する更新のログ・レコードだけです。

非並行イメージ・コピーは以下の 2 つの方法で作成できます。

- 静止して保持するオプションを指定してデータベース、エリア、または区画を静止し、イメージ・コピー・ユーティリティーのいずれかを実行する。START(QUIESCE) OPTION(HOLD) キーワードを指定して適切な UPDATE コマンドを実行することで、データベース、エリア、および区画は静止状態に保持されます。バッチ・イメージ・コピー・ユーティリティーを使用する場合は、データベースの DD ステートメントで DISP=SHR を指定する必要があります。イメージ・コピーが完了したら、STOP(QUIESCE) キーワードを指定した別の UPDATE コマンドで静止を解除する必要があります。
- データベースをオフラインにして、バッチ・イメージ・コピー・ユーティリティーを実行する。この方法では、STOP(ACCESS) キーワードを指定した適切な UPDATE コマンドを実行することにより、データベースへのアクセスを停止し、データベースの割り振りを解除し、さらに DBRC でデータベースを不許可にする必要があります。

非並行イメージ・コピーを作成するこの 2 つの方法では、静止機能を使用する方が簡単かつ迅速ですが、アプリケーション・プログラムがそれぞれの従属領域で待ち状態となるため、低データベース・アクティビティーの期間に限定してデータベースを静止状態に保持する必要があります。

関連概念:

640 ページの『データベースの静止』

高速複製イメージ・コピー

データベース・イメージ・コピー 2 ユーティリティー (DFSUDMT0) の高速複製オプションを使用して、高速複製イメージ・コピーを作成することができます。

DFSUDMT0 ユーティリティーでは、`FASTREPLICATION` キーワードを指定した z/OS DFSMSdss `COPY` コマンドが使用されます。

DFSUDMT0 ユーティリティーの高速複製オプションを使用すると、ほかのイメージ・コピー方法よりも迅速にイメージ・コピーを作成できます。クリーン高速複製イメージ・コピーまたは並行高速複製イメージ・コピーを作成し、データベース・リカバリー・ユーティリティー (DFSURDB0) のリカバリー手順で使用できるように DBRC に登録することができます。

高速複製イメージ・コピーは入力データ・セットの正確なコピーであり、イメージ・コピー 2 ユーティリティーで CONCURRENT キーワードを指定した DFSMSdss DUMP コマンドによって作成したイメージ・コピーと同じようにはフォーマット設定されません。

DFSMSdss での高速複製には、ハードウェア・サポートとして IBM Enterprise Storage Server® (ESS) の FlashCopy® 機能または IBM RAMAC Virtual Array (RVA) ストレージ・システムの Snapshot 機能が必要です。

イメージ・コピー 2 ユーティリティーは、高速複製イメージ・コピーをサポートする唯一の IMS イメージ・コピー・ユーティリティーです。

関連資料:

データベース・イメージ・コピー 2 ユーティリティー (DFSUDMT0) (データベース・ユーティリティー)

イメージ・コピー後のリカバリー

データベース・イメージ・コピー・ユーティリティー (DFSUDMP0) およびデータベース・イメージ・コピー 2 ユーティリティー (DFSUDMT0) は、HISAM、HIDAM、HDAM、PHDAM、および PHIDAM データベース用のデータ・セット、ならびに DEDB 用のエリアをコピーします。

これに続いてリカバリーを実行する場合は、入力データとしてリカバリーに何が必要かはコピーが並行的か否かによって決まります。

- 並行コピーでない場合。必要なのは、イメージ・コピーと、データベースがオンライン・システムに復元された後に作成されたログだけです。
- 並行コピーの場合。必要なのは、イメージ・コピーと、データベースがオンライン・システムに復元される前と復元された後に作成されたログです。 DBRC は、どのログが必要かについての判断に役立ちます。

推奨事項: データベースおよびエリアに対してデータベース・イメージ・コピー・ユーティリティー (DFSUDMP0) を (このユーティリティー用の EXEC ステートメントに CIC を指定せずに) 実行するときには、それらのデータベースを他のサブシステムが更新しないようにする必要があります。/DBDUMP DB コマンドまたは UPDATE DB STOP(ACCESS) コマンドを出すと、全機能データベースへの更新を阻止できます。/STOP AREA コマンドまたは UPDATE AREA STOP(SCHD) コマンドのどちらかを出すと、DEDB エリアへの更新を阻止できます。

オンライン・データベース・イメージ・コピー・ユーティリティー (DFSUICP0) は BMP プログラムとして稼働します。これを使用できるのは、HISAM、HIDAM、および HDAM データベースの場合のみです。このユーティリティーの実行中に IMS がこれらのデータベースを更新する場合は、IMS は、それ以降に行うリカバリーでは常に、このユーティリティーの開始時に使用中であったログも含めたすべてのロ

グを必要とします。IMS がログを必要とするのは、イメージ・コピーはある瞬間におけるデータベースのイメージではないからです。

HSSP イメージ・コピー

HSSP のイメージ・コピー・オプションを使用すると、IMS は DEDB エリアのイメージ・コピーを作成します。

このイメージ・コピーには、HSSP PCB の変更後イメージが含まれています。このコピーはファジー・コピーです。IMS は、他のすべての PCB データベース変更は通常どおりログに記録します。データベース・リカバリー時には、イメージ・コピー処理の開始以降のあらゆる並行更新を適用する必要があります。イメージ・コピー処理の間に非 HSSP 領域が同じ DEDB エリアを更新した場合は、ファジー・イメージ・コピーが作成される可能性があります。

定義: ファジー は、イメージ・コピー処理の間に並行更新が行われた可能性があることを意味します。

ただし、イメージ・コピー処理が正常に完了しなかった場合は、このデータ・セットは単に使用可能 HSSP イメージ・コピー・データ・セットのセットに戻されます。

IMS は、QSAM アクセス方式を使用して HSSP イメージ・コピーを作成します。イメージ・コピー・データ・セットの 1 次割り振りは、DEDB エリア・データ・セットのサイズより大きいか、または等しくなければなりません。

IMS は、HSSP イメージ・コピーを並行イメージ・コピーと同じように扱います。そのため、データベース・リカバリー・ユーティリティー (DFSURDB0) を使用する際にそのイメージ・コピーが HSSP イメージ・コピーであることを指示しなくても済みます。

制約事項: HSSP イメージ・コピーを作成できるのは、データベースが DBRC に登録されている場合のみです。さらに、イメージ・コピー・データ・セットを INIT.IC コマンドを使用して初期設定する必要があります。

関連タスク:

682 ページの『HSSP イメージ・コピーのリカバリー』

将来の使用のためのイメージ・コピー・データ・セットの作成

イメージ・コピー・データ・セットは、それらが必要になる前に割り振っておくことができます。

DBRC コマンド GENJCL.IC または GENJCL.OIC のいずれかを使用してデータベース・イメージ・コピー・ユーティリティー用またはオンライン・データベース・イメージ・コピー・ユーティリティー用のジョブを生成すれば、IMS はこれらのデータ・セットを自動的に選択して出力データ・セットとして使用できるようにします。

制約事項: データベース・イメージ・コピー 2 ユーティリティー (DFSUDMT0) の高速複製機能は、INIT.DBDS コマンドの REUSE パラメーターをサポートしませ

ん。このため、データベース・イメージ・コピー 2 ユーティリティーの高速複製機能で使用するデータ・セットを、必要となる前に作成しておくことはできません。

DBRC コマンド INIT.DBDS を使用して RECON データ・セット内の DBDS またはエリアを識別した場合は、その DBDS またはエリアに対して以下のいずれのキーワードでも指定できます。

GENMAX
このキーワードは、それらについての情報を DBRC に維持させたい DBDS 用またはエリア用のイメージ・コピー・データ・セットの個数を指定するのに使用します。

物理的リカバリー用として、特定の数のイメージ・コピー・データ・セットを維持できます。 DBRC は、指定された数の最新イメージ・コピー・データ・セットのレコードを保持します。この数は、この DBDS について GENMAX キーワードに指定する値です。重複イメージ・コピー・データ・セットの数はこの数には含めません。

RECOVPD
このキーワードは、データを特定の期間維持するために使用します。

REUSE
このキーワードは、イメージ・コピー・データ・セットを定義してそれらを将来の使用のために RECON データ・セットに記録したいことを DBRC に通知するのに使用します。 DBRC は、これらの使用可能イメージ・コピー・データ・セットをそれらの名前、装置タイプ、およびボリューム通し番号情報と共に RECON データ・セットに記録し、GENJCL.IC コマンドの処理時にそれらを選択して、この情報をデータベース・イメージ・コピー・ユーティリティー用のジョブの該当する DD 名の中で使用します。

NOREUSE
このキーワードは、DBRC に対し、既存のデータ・セットは使用したくないこと、およびデータベース・イメージ・コピー・ユーティリティーに使用させる出力イメージ・コピー・データ・セットのデータ・セット名を自分で付けたいことを通知するのに使用します。そのデータ・セット名は、DBRC が GENJCL.IC コマンドを処理するために使用する JCL 区分データ・セット (PDS) メンバーの中またはユーザー作成のジョブの中のいずれかで指定してください。NOREUSE を指定すると、DBRC は出力イメージ・コピー・データ・セットの装置タイプを当該の装置 (INIT.RECON コマンドおよび CHANGE.RECON コマンドで指定されているもの) のデフォルト装置タイプに動的に設定します。

データベース・イメージ・コピー・ユーティリティーが使用可能イメージ・コピー・データ・セットを使用するとき、DBRC は RECON データ・セット内のこのユーティリティーのレコードに現在のタイム・スタンプを追加します。

関連タスク:

『イメージ・コピー・データ・セットのリカバリー期間』

イメージ・コピー・データ・セットのリカバリー期間

リカバリー期間 とは、現在日付以前に DBRC が RECON データ・セット内でリカバリー情報を維持した期間のことです。

例えば、ある DBDS またはエリアのリカバリー期間が 14 日であるとすると、DBRC は十分なリカバリー世代情報を少なくとも 14 日間は維持していることになります。

リカバリー期間は、INIT.DBDS および CHANGE.DBDS コマンドの GENMAX および RECOVPD キーワードを使用して指定してください。これらのトピックの例では、各種の DBRC キーワードがリカバリー期間に及ぼす影響について説明します。

関連タスク:

652 ページの『将来の使用のためのイメージ・コピー・データ・セットの作成』

例 1: イメージ・コピー・データ・セットのリカバリー期間

例 1 では、RECOVPD パラメーターで指定されたリカバリー期間が終了していない段階で、イメージ・コピー・データ・セットの数が GENMAX 値に達し、かつ REUSE=Y であるとどうなるかを説明します。

以下の表では、DBRC キーワードおよびこの例についてのそれぞれの値を表示しています。

REUSE	GENMAX	RECOVPD
Y	達した	未超過

使用可能なイメージ・コピーがある場合は、次のようになります。

- IMS は使用可能なイメージ・コピーを使用する。
- IMS はメッセージ `DSP0065I` を出して、事前定義イメージ・コピーが使用されたことを指示する。

事前定義データ・セットの数が世代の最大数に等しい場合は、次のようになります。

- IMS は最も古いイメージ・コピーを再使用できない。
- IMS はメッセージ `DSP0063I` を出して、リカバリー期間内のイメージ・コピーを再使用できないことを指示する。
- IMS は処理を停止する。

例 2: イメージ・コピー・データ・セットのリカバリー期間

例 2 では、RECOVPD パラメーターで指定されたリカバリー期間が終了している段階で、イメージ・コピー・データ・セットの数が GENMAX 値に達しておらず、かつ REUSE=Y であるとどうなるかを説明します。

以下の表では、DBRC キーワードおよびこの例についてのそれぞれの値を表示しています。

REUSE	GENMAX	RECOVPD
Y	達していない	超過

IMS が使用可能イメージ・コピー・データ・セットを使用する場合を除き、 656 ページの『イメージ・コピー・データ・セットの再使用』の説明のように処理が継続されます。

例 3: イメージ・コピー・データ・セットのリカバリー期間

例 3 では、RECOVPD パラメーターで指定されたリカバリー期間が終了している段階で、イメージ・コピー・データ・セットの数が GENMAX 値に達し、かつ REUSE=N であるとどうなるかを説明します。

以下の表では、DBRC キーワードおよびこの例についてのそれぞれの値を表示しています。

REUSE	GENMAX	RECOVPD
N	達した	超過

IMS は、RECOVPD の値を超えた最も古いイメージ・コピー・データ・セット・レコードを削除する。

例 4: イメージ・コピー・データ・セットのリカバリー期間

例 4 では、RECOVPD パラメーターで指定されたリカバリー期間が終了していない段階で、イメージ・コピー・データ・セットの数が GENMAX 値に達し、かつ REUSE=N であるとどうなるかを説明します。

以下の表では、DBRC キーワードおよびこの例についてのそれぞれの値を表示しています。

REUSE	GENMAX	RECOVPD
N	達した	未超過

使用済みのイメージ・コピー・データ・セットの数が GENMAX の値に達した場合は、IMS はリカバリー期間内の最も古いイメージ・コピー・データ・セットを削除できない。この場合は次のようになります。

- IMS はメッセージ DSP0064I を出して、リカバリー期間内のイメージ・コピー・データ・セットを削除できないことを指示する。
- 処理が継続され、DBRC は新しいイメージ・コピー・データ・セットを RECON データ・セットに記録する。

例 5: イメージ・コピー・データ・セットのリカバリー期間

例 5 では、RECOVPD パラメーターで指定されたリカバリー期間が終了している段階で、イメージ・コピー・データ・セットの数が GENMAX 値に達しておらず、かつ REUSE=N であるとどうなるかを説明します。

以下の表では、DBRC キーワードおよびこの例についてのそれぞれの値を表示しています。

REUSE	GENMAX	RECOVPD
N	達していない	超過

DBRC は新しいイメージ・コピー・データ・セットを RECON データ・セットに記録します。最も古いイメージ・コピーがリカバリー期間を超過していたとしても、GENMAX にはまだ達していないため、そのイメージ・コピーは削除されません。

リカバリー期間に関するその他の考慮事項

CHANGE.DBDS コマンドを出して GENMAX に新しい値として既存の値より小さい値を指定した場合は、最も古いイメージ・コピーがリカバリー期間 (RECOVPD) 内であるために削除できないかそうでないかとは無関係に、IMS はその新しい値を記録します。

DELETE.IC コマンドを出した場合は、RECOVPD の値が超過済みかどうかとは無関係に、IMS は指定されたイメージ・コピー・データ・セットをすべて削除します。

イメージ・コピー・データ・セットの再使用

DBRC を使用すると、古いイメージ・コピー・データ・セットを再使用できます。

INIT.DBDS コマンドの REUSE キーワードを使用すれば、将来の使用のためにユーザーがイメージ・コピー・データ・セットを定義できるのに加え、DBRC がイメージ・コピー・データ・セットを再使用できるようになります。イメージ・コピー・データ・セットの再使用とは、DBRC が古いイメージ・コピー・データ・セットと同じ名前、ボリューム、物理スペース、および RECON データ・セット内のレコードを新しいイメージ・コピー・データ・セットのものとして使用することを意味します。

制約事項: データベース・イメージ・コピー 2 ユーティリティーの高速複製機能を使用してコピーするデータ・セットには、REUSE キーワードを使用できません。高速複製機能ではデータ・セットの再利用がサポートされません。

イメージ・コピー・ユーティリティーのいずれかを実行した場合は、以下の条件が両方とも満たされれば、IMS は、可能であれば DBDS 用またはエリア用の最も古いイメージ・コピー・データ・セットを自動的に再使用します。

- RECON データ・セット内に GENMAX の現行値と等しい数のイメージ・コピー・データ・セットのためのレコードが存在する。GENMAX の現行値を表示するには、LIST.DBDS コマンドを使用してください。
- 最も古いイメージ・コピーがリカバリー期間を超過している。

GENJCL.IC コマンドを使用してデータベース・イメージ・コピー・ユーティリティー用またはデータベース・イメージ・コピー 2 ユーティリティー用のジョブを生成する場合は、IMS は再使用するイメージ・コピー・データ・セットを自動的に選択します。イメージ・コピー・データ・セットの数が GENMAX の値より少なく、かつすべてのイメージ・コピー・データ・セットが使用済みである場合は、データベース・イメージ・コピー・ユーティリティーまたはデータベース・イメージ・コピー 2 ユーティリティーを実行する前に、DBDS 用またはエリア用のイメージ・コピー・データ・セットをさらに定義する必要があります。リカバリー期間を使用したい場合は、イメージ・コピー・データ・セットの数を GENMAX の値より多くしなければなりません。

IMS がイメージ・コピー・データ・セットを再使用するのを許可していないが、GENMAX の値にすでに達していて、さらに RECOVPD をすでに超えている場合は、データベース・イメージ・コピー・ユーティリティーまたはデータベース・イメージ・コピー 2 ユーティリティーを実行すると、DBRC は新しいイメージ・コピー・データ・セットを選択し、RECON データ・セット内の最も古いタイム・スタ

ンプをもつレコードを削除します。IMS は、自動的にイメージ・コピー・データ・セットをスクラッチすることはしません。また、DBRC はすでにそのデータ・セットの存在を認識しなくなるため、ユーザーがそのデータ・セットのスクラッチまたはトラッキングを自ら行う必要があります。

HISAM コピー (DFSURUL0 および DFSURRL0)

HISAM 再編成アンロード・ユーティリティー (DFSURUL0) では、HISAM データベース全体を再編成しながらそのバックアップ・コピーを作成する処理を、1 回のパスで実行できます。

このアンロード・ユーティリティー (DFSURUL0) はデータベースを再編成するため、通常オンライン・オペレーションを再開する前に、以下の図に示されているように HISAM 再編成再ロード・ユーティリティー (DFSURRL0) を使用してデータ・セットを再ロードする必要があります。データを再ロードしないと、アンロード後ではあるが再ロード前に実行されたロギングは、データ・セットの古い編成を反映してしまいます。したがって、将来そのデータ・セットをリカバリーするためにこのログを使用する必要が生じた場合には、編成が一致しないために、そのデータ・セットの保全性が損なわれることになります。

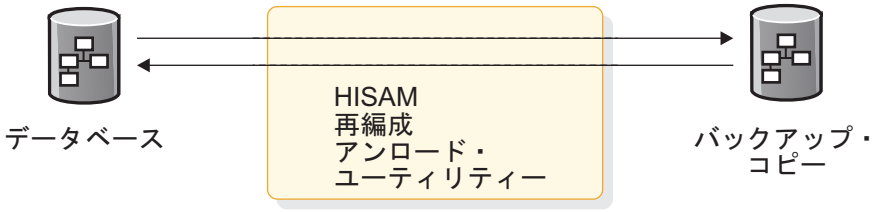


図 *249. HISAM* アンロードでのバックアップ・コピーの作成

HISAM ユーティリティーを使用してバックアップ・コピーを作成した場合はただちに再ロードする必要があります。そうしないと、実際のデータベースがバックアップ・データベースと一致しなくなります。再ロード・ユーティリティー (DFSURRL0) は、その再編成ステップの出力を、それがイメージ・コピー・データ・セットであるかのように扱って、再ロード・ステップへの入力として使用します。データベースの再編成後は、そのデータベースのイメージ・コピーを作成してからでなければ DBDS を許可できません。

共用データベースを再編成する場合は、その前に次のようにして、再編成処理中に他のサブシステムが許可されるのを防止する必要があります。

- データベースが再編成されるようにグローバル /DBRECOVERY DB または UPDATE DB STOP(ACCESS) コマンドを発行する。/DBRECOVERY DB GLOBAL コマンドは、再編成ユーティリティーおよびリカバリー・ユーティリティーを除き、それら以外についての許可が行われないようにします。
- NOAUTH キーワードを指定した CHANGE.DB コマンドを出して RECON データ・セットを手動で更新する。このコマンドは、それ以降に再編成ユーティリティーおよびリカバリー・ユーティリティー以外のものが許可されるのを防止します。再編成が完了したら、その再編成が済んだばかりのデータベースに対する AUTH キーワードを指定した CHANGE.DB コマンドを出して、RECON データ・セットを手動で更新してください。

推奨事項: 再編成中にリカバリー・ユーティリティーが実行されることが決してないようにしてください。

標準外イメージ・コピー・データ・セット

いずれかの IMS イメージ・コピー・ユーティリティーを使用して作成されていないイメージ・コピー・データ・セットは、標準外イメージ・コピー・データ・セット またはユーザー・イメージ・コピー・データ・セット と呼ばれます。

例えば、IBM 装置サポート機能 (ICKDSF) 製品を使用すると、DBDS が常駐しているボリュームのコピーを作成できます。

IMS は標準外イメージ・コピーに関するこの情報をトラッキングしないため、IMS ユーティリティーのいずれかを使用せずにイメージ・コピーを作成する場合は、イメージ・コピーのデータ・セット名およびコピーの作成に使用するツールをトラッキングする必要があります。

データ・セットを正しくコピーできないと、データベース・リカバリー処理は失敗する可能性があります。KSDS の場合、 KSDS データ内容を保存しておくユーティリティーであればどのユーティリティーでも、そのコピーの作成に使用できます (IMS は、キーで KSDS にアクセスするため、KSDS 内での RBA 順序には依存しません)。これ以外のすべてのデータ・セット・タイプについては、データ・セットの内容の物理的順序を保存しておく必要があります。データ・セットを正しくコピーするユーティリティーだけを使用するか、あるいは IMS 提供のイメージ・コピー・ユーティリティーを使用して標準イメージ・コピーを作成してください。

DBRC が標準外イメージ・コピー・データ・セットの存在を RECON データ・セットに自動的に記録することはないため、NOTIFY.UIC コマンドを使用する必要があります。この情報を RECON データ・セットに記録しないと、DBRC が DBDS への変更に関するそれ以降の情報を正しく解釈できない可能性があります。NOTIFY.UIC コマンドは、バッチおよび並行の標準外イメージ・コピーをサポートします。

制約事項: NOTIFY.UIC コマンドを使用する場合は、INIT.DBDS コマンドの REUSE キーワードは指定できません。

DBDS または DEDB エリアを、標準外クリーン・イメージ・コピー・データ・セットを使用してリカバリーする場合は、事前にその DBDS または DEDB エリアを標準外イメージ・コピー・データ・セットから復元する必要があります。ただし、リカバリーを完了するプロセスは、標準外イメージ・コピーがバッチ・イメージ・コピーと並行イメージ・コピーのどちらであるかによって異なります。

関連概念:

646 ページの『イメージ・コピーおよび IMS イメージ・コピー・ユーティリティー』

バッチ標準外イメージ・コピーからのリカバリー

バッチ標準外ユーザー・イメージ・コピーを使用する場合は、DBDS または DEDB エリアをイメージ・コピーから復元して、データベース・リカバリー・ユーティリティーを実行するまでの間に、NOTIFY.RECOV コマンドを実行してデータ・セットまたはエリアを復元したことを DBRC に通知する必要があります。

その標準外イメージ・コピーのタイム・スタンプが既存のタイム・スタンプ・リカバリーの範囲内に入っていると、NOTIFY.RECOV コマンドは失敗します。NOTIFY.RECOV コマンドが成功したら、データベース・リカバリー・ユーティリティーを実行して、標準外イメージ・コピー・データ・セットのタイム・スタンプ以降に行われた変更を追加してください。DBRC が提供および検査する JCL は、復元済みの DBDS または DEDB エリアに適用される変更レコードのソース用の JCL のみです。

バッチ標準外イメージ・コピーを使用している場合に、GENJCL.RECOV コマンドを使用してリカバリー JCL を生成するときは、USEDBDS パラメーターを追加して、生成された JCL にイメージ・コピー・データ・セットが組み込まれないように指定します

並行標準外イメージ・コピーからのリカバリー

DBDS または DEDB エリアを並行標準外イメージ・コピー・データ・セットでリカバリーする場合は、事前に RECOV パラメーターを指定した CHANGE.DBDS コマンドを実行して「要リカバリー」状況を設定することで、その DBDS または DEDB エリアが使用不可であることを指定します。

DBDS または DEDB エリアがイメージ・コピーから復元されたら、データベース・リカバリー・ユーティリティーを実行して、標準外イメージ・コピー・データ・セットのタイム・スタンプ以降に行われた変更を追加してください。DBRC が提供および検査する JCL は、復元済みの DBDS または DEDB エリアに適用される変更レコードのソース用の JCL のみです。

並行標準外イメージ・コピーを使用している場合に、GENJCL.RECOV コマンドを使用してリカバリー JCL を生成するときは、USEDBDS パラメーターを指定しないでください。この場合は、DBRC RECON データ・セットに、生成された JCL にイメージ・コピー・データ・セットが組み込まれないことを示すフラグが設定されます。

データベース・データ・セットをリカバリーするときに、並行標準外イメージ・コピーが作成された実行時間を指定する必要があります。そのためには、GENJCL.RECOV コマンドの USERIC(*time_stamp*) キーワードまたは LASTUIC キーワードを指定します。これにより、並行標準外イメージ・コピー・データ・セットが作成された後の変更を追加する JCL ストリームが、データベース・リカバリー・ユーティリティー用に生成されます。DBRC が生成および検査する JCL は、復元済みの DBDS または DEDB エリアに適用される変更レコードのソース用の JCL のみです。

データベース・リカバリー・ユーティリティーは変更レコードの処理にイメージ・コピーを使用しないため、DBRC はデータベース・リカバリー・ユーティリティーがリカバリー範囲外の変更を含んでいるログを処理することを許可しません。リカバリー範囲は、タイム・スタンプ・リカバリー・レコードの RECOV TO 値 (イメージ・コピーの時刻) および RUNTIME 値によって定義されます。

推奨事項: イメージ・コピー・ユーティリティーを実行する前に、/DBRECOVERY コマンド (NOFEOV キーワードを指定せずに) または UPDATE DB STOP(ACCESS) OPTION(FEOV) コマンドを使用して、データベースをクローズしてください。

バックアップ・コピーの頻度および保存

データベースのバックアップ方針を策定する際には、新しいコピーを作成する頻度、および古いバックレベルのコピーを保存する期間を検討する必要があります。

これらの問題に対する的確な答えはありません。一般的には、コピーを作成する頻度を高くすればするほど、リカバリーに要する時間が短縮されます。また、古いコピーを作成時刻がより前のものへとたどっていけば、それだけ前の時点からのリカバリーが可能になります。プログラム・ロジック・エラーは何週間も発見されない場合があることを思い出してください。しかし、新しいコピーを作成する度に作業が必要になりますし、保管しておく古いコピーが 1 つ増えるごとにその分使用されるリソースが増えます。

確かなガイドラインは次のものだけです。

- データベースの初期ロードを実行したら、ただちにそのデータベースのバックアップ・コピーを作成する。
- データベースが複数データ・セットから構成されている場合は、必ずすべてのデータ・セットを一時にコピーする。
- データベースを再編成したら、ただちにそのデータベースの新しいバックアップ・コピーを作成する。

  例外: DEDB オンライン再編成の場合は、その後にバックアップ・コピーを作成する必要はありません。

データベースがオフラインのときではなく静止しているときにイメージ・コピーを作成することで、非並行イメージ・コピーの作成に要する作業量を削減できます。データベースを静止した場合、データベースはオンライン状態を保ち、割り振られて DBRC に許可されたままとなります。アプリケーション・プログラムは待ち状態になります。データベースの静止には手間がかからないため、イメージ・コピーを作成する頻度を上げることが可能になります。

非並行イメージ・コピーを作成するためにデータベースをオフラインにすると、データベースはオフラインとなり、割り振り解除され、さらに DBRC の許可も解除されます。アプリケーション・プログラムでは、データベースが使用不能となります。

関連タスク:

664 ページの『データベース・リカバリー戦略のプラン』

RSR 環境におけるイメージ・コピー

リモート・サイト・リカバリー (RSR) 環境には、アクティブ・サイトとトラッキング・サイト間でのイメージ・コピーの作成と処理に関する特別な要件があります。

データベース・イメージ・コピーをトラッキング・サイトに送る必要があるのは、次の 3 とおりの場合です。すなわち、データベースのトラッキングが始まる前、アクティブ・サイトにおけるトラッキングされたデータベースのためのタイム・スタンプ (部分) リカバリーの後、およびアクティブ・サイトにおけるデータベース再編成の後です。

アクティブ・サイトにおいてデータベースの再編成を実行した後、再編成された各データベースのイメージ・コピーをトラッキング・サイトに送る必要があります。これらのイメージ・コピーは、データベースの再編成後できる限り速やかに送ることが重要です。これらのイメージ・コピーのいずれかがトラッキング・サイトに到着しないうちに予期しないテークオーバーが発生した場合は、そのデータベースおよび論理的に関係のあるすべてのデータベースを元どおり実動に戻すとき遅延が発生します。

論理的に関係のあるデータベースのセットのいずれかのイメージ・コピーがリモート・テークオーバーの後で使用不能になると判断した場合は、論理的に関係のあるデータベースをすべて、データベースの再編成前の状態に戻す必要があります。これには、既にイメージ・コピーが適用されているデータベースのタイム・スタンプ・リカバリーが必要です。したがって、再編成後に元のアクティブ・サイトで作成されたデータベースに対して行われた更新はすべて破棄されます。

データベースによっては、明示的な論理関係で他のデータベースに接続されないが、特定のアプリケーションによって行われる処理により暗黙的に関係付けられる場合があります。これらのデータベースは、リモート・テークオーバー後に、手動で管理する必要があります。 RSR ではこのような暗黙的な相互関係については認識されないため、それらのデータベースのイメージ・コピーをトラッキング・サイトで適用する場合に特にこの手動管理が必要です。

データベース・リカバリーができる限り効率よく行われるために、他のイメージ・コピー (データベース再編成ユーティリティーの 1 つによって、またはタイム・スタンプ・リカバリーによって発生したコピーでないもの) も、できるだけ速やかにトラッキング・サイトに送る必要があります。しかし、これらのイメージ・コピーは、データベース再編成およびタイム・スタンプ・リカバリーにより作成されたものほど RSR にとって重要ではありません。

関連タスク:

689 ページの『RSR 環境における標準外イメージ・コピーを使用したデータベースのリカバリー』

データベースのリカバリー

データベースが、中のレコードがアクセス不能になるなどして、物理的に失われたり、あるいは損傷を受けたりした場合は、そのデータベースを、保持している情報 (イメージ・コピー、ログ、など) から再構成することができます。このタイプのリカバリーは、順方向リカバリーとして知られています。

定義: 順方向リカバリー には、情報の再構成と、それをデータベースのバックアップ・コピーに再適用することが含まれます。これは、ある時点でのデータの内容、およびそれ以降にデータに加えられた変更がわかっていれば、データを処理してデータが失われる直前の状態にデータベースを戻すことができるという考えに基づいています。

以下の図は、この概念の説明です。データベースのバックアップ・コピーを作成しているので、データベースがある時点でどのようであったかがわかります。バックアップ・コピーが作成された以降にデータベースにどんな変更 (追加、削除、変更) が行われたかは、IMS がこれらの変更をログに記録しているので、わかります。し

たがって、この 2 つを結合するだけで、新規データ・セットを作成し、脱落したデータベースと置き換えます。

図 *250.* データベースの再構成 *(*順方向リカバリー*)*

IMS には、順方向リカバリーを実行するデータベース・リカバリー・ユーティリティー (DFSURDB0) があります。以下の図は、このユーティリティーの働き方の説明です。

さらに、IBM IMS Database Recovery Facility for z/OS 製品がインストールされている場合は、そのツールを IMS 内部から呼び出す /RECOVER コマンドを IMS で使用できます。

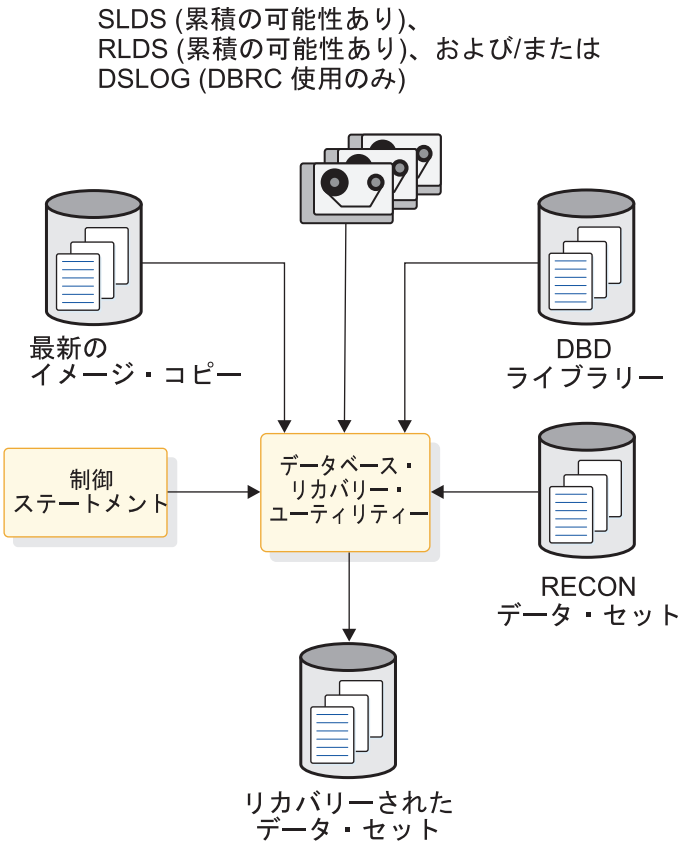


図 *251. IMS* データベース・リカバリー・ユーティリティー *(DFSURDB0)* を使用したデータベースのリカバリー

制約事項: DBRC は、DBRC に登録されていないデータベースをリカバリーするための JCL を生成することはできません。現行の OLDS、RLDS、および SLDS のリストを作成する場合は、LIST.LOG コマンド (または DSPAPI FUNC=QUERY API 要求) を使用できます。このコマンドを使用すると、変更累積またはデータベース・リカバリーに使用している JCL に、すべてのアクティブ・ログ・データ・セットに関する DD ステートメントが入っていることを、手動でチェックすることができます。 DBRC バッチ・ログ制御を使用するときは、LIST.LOG コマンド (または DSPAPI FUNC=QUERY API 要求) を使用することもできます。

推奨事項: リカバリー不能な DBDS にデータベース・リカバリー・ユーティリティーを実行しているときは、ログまたは変更累積入力を指定しないでください。

データベース・リカバリー・ユーティリティーおよびデータベース・リカバリー・サービスは、データベースのリカバリーをデータ・セット・レベルで行います。通常は、データベースの単一のデータ・セットのみがリカバリーを必要とします。複数のデータ・セットが失われているか損傷を受けている場合で、データベース・リカバリー・ユーティリティーを使用します。データベース・リカバリー・サービスを使用する場合は、失われたデータ・セットまたは損傷したデータ・セットのすべてを 1 回のリカバリー・オペレーションでリカバリーできます。

DEDB の場合は、複数エリア・データ・セットを使用すると、リカバリーの必要が削減されます。 1 つのコピーに入出力エラーがある場合は、オンラインで別のコピーを作成して、エラー・コピーを除去することができます。

関連タスク:

909 ページの『区画を使用可能にする際に HALDB データベースをリカバリーする』

データベースのリカバリー (システム管理)

関連資料:

データベース・リカバリー・ユーティリティー (DFSURDB0) (データベース・ユーティリティー)

リカバリーおよびデータ・セット

データベース・リカバリー・ユーティリティー (DFSURDB0) は、データベースのリカバリーをデータ・セット・レベルで行います。複数のデータ・セットが失われているか損傷を受けている場合で、データベース・リカバリー・ユーティリティーを使用するときは、それぞれのデータ・セットを別々にリカバリーする必要があります。

IMS Database Recovery Facility for z/OS を使用する場合は、失われたデータ・セットまたは損傷したデータ・セットのすべてを 1 回のリカバリー・オペレーションでリカバリーできます。

DEDB の場合は、複数エリア・データ・セットを使用すると、リカバリーの必要が削減されます。エリア・データ・セットの 1 つのコピーに入出力エラーがある場合は、オンラインで別のコピーを作成して、エラー・コピーを除去することができます。

データベース・データ・セットがマルチボリューム OSAM データ・セットである場合は、適切な割り振り指定を使用して、すべてのボリュームが 1 次エクステントを取得していることを確認してください。

一部のケースでは、データ・セットをリカバリーする前に削除と再定義を行う必要はありません。実際、データ・セットの削除と再定義がオプションであるケースでは、これを必要以上に行うと、リカバリー処理が遅くなります。

以下の場合は、リカバリー処理中に、データ・セットの削除と定義が必要です。

- データベース・リカバリー・ユーティリティーを使用して VSAM データベース・データ・セットをリカバリーする場合。
- 統合 HALDB オンライン再編成機能の出力データ・セットをリカバリーしている場合で、データ・セットがカーソルのアクティブ状態にある場合。
- HALDB 索引/ILDS 再作成ユーティリティーを使用して、HALDB 間接リスト・データ・セット (ILDS) を再作成する場合。

以下の場合は、リカバリー処理中に、データ・セットの削除と定義が不要です。

- データベース・リカバリー・ユーティリティーを使用して OSAM データベース・データ・セットをリカバリーし、ユーティリティーへの入力としてイメージ・コピーを組み込む場合。
- HALDB 索引/ILDS 再作成ユーティリティーを使用して、HALDB PHIDAM の 1 次索引を再作成する場合。

データ・セットを削除して再定義する必要がある場合、リカバリー処理中に、削除/定義ステップをいつ実行する必要があるかは、リカバリーの事情によって異なります。例えば、次のようになります。

- データベース・リカバリー・ユーティリティーへの入力としてイメージ・コピーを組み込む場合は、ユーティリティーを実行する前に、データ・セットの削除と定義を行います。
- 前のステップでデータベースにイメージ・コピーを適用するときのように、データベース・リカバリー・ユーティリティーへの入力としてイメージ・コピーを組み込まない場合は、イメージ・コピーを適用する前にデータ・セットの削除と定義を行います。
- RSR トラッキング・サイト内のデータ・セットをリカバリーする場合は、RSR アクティブ・サイトからイメージ・コピーを適用する前に、削除/定義ステップを実行します。
- GENJCL.RECOV コマンドが実行されたときにリカバリー JCL を自動的に実行する場合は、GENJCL.RECOV コマンドが実行される前に、データ・セットの削除と定義を行います。

関連タスク:

615 ページの『OSAM データ・セットの割り振り』

データベース・リカバリー戦略のプラン

データベースのリカバリーは、さまざまな方法で行うことができます。

おそらく最も時間がかかりますが、最も単純な方法は、イメージ・コピーに加えて、イメージ・コピー以降に作成されたすべてのログ・データ・セットを使用することです。以下のようにすることもできます。

- データベース・リカバリー・サービスを使用する
- 変更累積ユーティリティーを使用して、データベース・リカバリー・ユーティリティーへのログ入力の量を減らす
- ログ保存ユーティリティーで RLDS を作成し、SLDS の代わりに RLDS を使用する

データベース・リカバリー・ユーティリティーは、常に、変更累積データ・セットを処理した後にログ・テープを処理します。したがって、変更累積データ・セットの前に作成されたログ・テープを組み込む必要がある場合は、 DFSURDB0 を 2 回 (イメージ・コピーとログ・テープに 1 回、そして変更累積データ・セットにもう 1 回) 実行する必要があります。

データベースを DBRC に登録すると、DBRC は、必ず正しい入力データがデータベース・リカバリー・ユーティリティーに提供されるようにして、必要な JCL をオプションで生成します。 DBRC の詳細については、「*IMS V13* システム管理」を参照してください。

異なったリカバリー技法の利点および欠点のいくつかをこのトピックで説明します。

関連概念:

660 ページの『バックアップ・コピーの頻度および保存』

リカバリーのためのデータベース変更累積入力の使用

データベース変更累積ユーティリティー (DFSUCUM0) を定期的に使用して、古いログ・ボリュームを統合する場合は、これらの累積ボリュームをデータベース・リカバリー・サービスやデータベース・リカバリー・ユーティリティーへの入力データとして使用することができます。

最後にデータベース変更累積ユーティリティーを実行してから以降に記録されたすべてのログ・ボリュームを、入力データとしてリカバリーに組み込む必要があります (その現行のフォーマットで組み込むか、データベース変更累積ユーティリティーに渡された後に組み込む)。

データベース変更累積ユーティリティーを実行するとき、変更レコードを累積したい DBDS を指定することができます。 DBDS ごとに、変更レコードは、無視しても、オプションの出力ログ・データ・セットに書き込んでも構いません。 DBDS のサブセット (変更累積グループと呼ばれる) を指定すると、変更累積データ・セットのサイズを削減し、データベース・リカバリー・ユーティリティーのパフォーマンスを向上させることができます。しかし、変更累積グループごとに、データベース変更累積ユーティリティーを実行し (ログ・データ・セットを渡す) ことも必要です。

データベース変更累積ユーティリティーを実行するたびに、DBDS を 1 つしか選択しない場合は、DBDS ごとに 1 回、ユーティリティーを実行して (ログ・データ・セットを渡す) 必要があります。この技法では、各出力変更累積ファイルには、単一 DBDS のリカバリーに必要なレコードしか入りません。

データベース変更累積ユーティリティーのパフォーマンスは、データベースの静止を頻繁に実行することで向上する場合があります。データベースを静止すると、複数のデータベースとログにわたって整合点が作成されて、累積する必要があるログ・レコードの数が減ります。整合点が作成される際に、新しい更新は保留され、進行中の更新はすべてコミットされて DASD に書き込まれます。進行中の最後の更新が DASD に書き込まれると、RECON データ・セット内のデータベース割り振り (ALLOC) レコードに割り振り解除 (DEALLOC) タイム・スタンプが追加されて新しいリカバリー・ポイントが作成され、データベースに対する更新が再開されます。

以下に、一種の共通リカバリー技法を示します。

- すべての DBDS に、単一の変更累積グループを定義する。

  この場合、データベース変更累積ユーティリティーはログ・データ・セットを 1 回だけ読み取りますが、データベース・リカバリー・ユーティリティーは、場合によっては不要なデータベースのために出力変更累積データ・セットのレコードを読み取る必要があります。この技法では、データベース変更累積ユーティリティーのパフォーマンスは向上しますが、データベース・リカバリーのパフォーマンスは低下します。
- いくつかの変更累積グループを定義する。

  この場合は、データベースをいくつかのデータベースのグループ (ボリューム別かアプリケーション別の場合が多い) に分割します。イメージ・コピーを取る頻度が高い場合は、データベース・リカバリー・ユーティリティーがスキャンするレコード数を最小に保つことができます。
- 変更累積グループを定義しない。データベース変更累積ユーティリティーを実行しますが、正規のジョブとしては実行しません。

  データベースをリカバリーする必要があるときは、ユーティリティーが、影響を受けた DBDS のみの変更を累積することを指定します。リカバリーするデータ・セットの最新のイメージ・コピー以降に作成された、すべてのログ・データ・セットを指定します。いずれかのイメージ・コピー・ユーティリティーを使用する場合は、そのイメージ・コピーがデータベース変更累積ユーティリティーへの入力データとして作成されたときに作成された、ログを組み込む必要があります。

  この方法を考える場合は、若干のテスト・ランを行って、データベース変更累積ユーティリティーとデータベース・リカバリー・ユーティリティーの実行時間の合計が、累積されなかったログ・データ・セットにデータベース・リカバリー・ユーティリティーを実行するのに必要な時間未満かどうかを調べる必要があります。

関連概念:

640 ページの『データベースの静止』

リカバリーのためのログ・データ・セットの使用

データベース変更累積ユーティリティーを使用しないことにした場合は、ログ保存ユーティリティー (DFSUARC0) が作成したリカバリー・ログ・データ・セット (RLDS) の使用を考慮してください。

RLDS には対応する SLDS からリカバリーに関連したログ・レコードのサブセットのみが収められることから、RLDS は SLDS よりもサイズが小さくなります。ログ保存ユーティリティーは、リカバリーに必要なレコードだけを RLDS に書き込みます。この場合、イメージ・コピーの頻度がリカバリー時間に与える主要なインパクトで、データベース更新の量によって決まります。

ログ・ボリュームは、古いほうから新しいほうへと、発生順にリカバリー・ユーティリティーに実行依頼する必要があります。

データベースを、そのイメージ・コピーの作成後データベース・バッチ・バックアウト・ユーティリティーを使用してバックアウトする場合は、バッチ・バックアウト・ユーティリティーが作成したログと元のログの両方を、データベース・リカバリー・ユーティリティーへの入力データとして使用する必要があります。なぜなら、元のログに記録された変更のいくらかを、バッチ・バックアウト・ユーティリティーがバックアウトするからです。

DBRC を用いたリカバリーの監視

データベース・リカバリーの監視には、DBRC を使用してください。DBRC を使用すると、リカバリーの作業が著しく単純化されます。

日常オペレーションの間、DBRC はデータベースのアクティビティー (バックアップ・コピーを取ったり、関係のあるログ・ボリュームを記録したり、変更累積を作成したり、など) をトラッキングします。したがって、DBRC には、データベースをリカバリーするために、リカバリー・ユーティリティーに何を提供すべきかがわかります (イメージ・コピーとログ・ボリューム)。 DBRC は、ユーティリティーを実行するために必要な JCL を生成し、正しい順序の正確な入力を保証します。

DBRC は、DBDS およびエリア・データ・セット (ADS) 用に、次の 2 タイプのデータベース・リカバリーをサポートします。

- 完全リカバリー は、DBDS または ADS を、RECON データ・セットに記録されたままの現状に復元します。
- タイム・スタンプ・リカバリー は、DBDS または ADS を、指定した時点の内容に復元します。

DBDS または ADS の全リカバリーは、一般には 2 ステップの処理です。 DBDS または ADS のコピーを復元してから、そのコピー以降に加えられた変更を適用する必要があります。これらの変更は、変更累積データ・セットやログ・データ・セットに含めることができます。データベースのバックアップ・コピーを作成する場合は、データベース・リカバリー・ユーティリティーを実行すれば、両方のステップを行うことができます。

DBDS または ADS のタイム・スタンプ・リカバリーの場合は、DBDS または ADS は、過去の一定の時点 (通常は、DBDS または ADS が更新されていない時点) にリカバリーすることになります。一般に、タイム・スタンプ・リカバリーは、不良入力データなどの論理エラー、あるいはバッチ・ジョブの重複実行などの操作エラーをリカバリーするために行います。タイム・スタンプ・リカバリーには、1 つまたは複数の最新セットの更新をバックアウトする効果があります。

推奨事項**:** タイム・スタンプ・リカバリーは細心の注意を持って行ってください。1 つの DBDS または ADS をリカバリーするときは、関連するすべての DBDS または ADS に同様のリカバリーを行う必要があります。この種の関連データ・セットの例としては、論理関係によって接続されたものだけでなく、複数のデータ・セット・グループを含む索引やデータベースもあります。 DBRC には、DBDS または ADS がどう関連しているかがわからないので、必ず、関連するすべてのタイム・スタンプ・リカバリーを行う必要があります。

DEDB をリカバリーする一般的な方法は、変更累積ユーティリティーを使用してから、順方向リカバリーを行うことです。有効な入力データを入手しやすくするために、ログ・リカバリー・ユーティリティーを使用して、不完全な DEDB 更新を除外する OLDS のコピーを作成することができます。 DEDB リカバリー・レコードのみのデータ・セットを作成することもできます。損傷したエリア・データ・セットの修理を支援するときは、DEDB エリア・データ・セット比較ユーティリティーを使用します。

推奨事項**:** 共用 VSO オプションを使用する領域の場合、この領域が使用した CF 構造への XES 接続に障害がないことを確認するまでは、その領域をオンラインに戻したり、あるいは、接続を所有していた、障害のある IMS システムを再始動したりしないでください。z/OS `SETXCF` コマンドを使用して、接続を削除してください。

関連概念:

669 ページの『データベースのリカバリーの概説』

DBRC の概要 (システム管理)

関連タスク:

データベースのリカバリー (システム管理)

関連資料:

リカバリー・ユーティリティー (データベース・ユーティリティー)

リカバリー・ユーティリティー許可のリジェクト

DBRC が、IMS リカバリー・ユーティリティーが稼働する許可をリジェクトした場合、メッセージ `DFS3710A` を受け取る場合があります。これは、通常、直前のリカバリー・ユーティリティーが処理の間に異常終了し、そのデータベース許可を解放しなかったことを示します。

この状態でリカバリーするには、以下のコマンド・シーケンスまたは API 要求シーケンスを使用して、RECONデータ・セットのサブシステム・レコードを削除してください。

- コマンド・シーケンス:
  1. `CHANGE.SUBSYS SSID(name) STARTRCV`
  2. `CHANGE.SUBSYS SSID(name) ENDRECOV`
  3. `DELETE.SUBSYS SSID(name)`

  DBRC コマンドについて詳しくは、「*IMS V13* コマンド第 *3* 巻*: IMS* コンポーネントおよび *z/OS* コマンド」を参照してください。
- API 要求シーケンス:

1. DSPAPI FUNC=COMMAND COMMAND=CHANGE.SUBSYS SSID(name) STARTRCV
2. DSPAPI FUNC=COMMAND COMMAND=CHANGE.SUBSYS SSID(name) ENDRECOV
3. DSPAPI FUNC=COMMAND COMMAND=DELETE.SUBSYS SSID(name)

DBRC API について詳しくは、「*IMS V13* システム・プログラミング *API*」を参照してください。

データベースのリカバリーの概説

データベースの順方向リカバリーは、データベースのバックアップ・コピーを復元し、さらに、バックアップ・コピーの作成後に起きた、ログ記録されたデータベース変更のすべてを再適用することによって行います。

これらの 2 つの基本ステップは、データベースの順方向リカバリーのすべてに共通ですが、順方向リカバリーの具体的なステップは、リカバリーするデータベースのタイプ、および、データ共用環境または非データ共用環境のどちらで操作を行っているかによって異なります。

IMS データベースのバックアップ・コピーは、通常、データベースのイメージ・コピーです。イメージ・コピーは、IMS イメージ・コピー・ユーティリティーのいずれか、または、別のイメージ・コピー・ツールによって作成されます。 IMS イメージ・コピー・ユーティリティーは、作成するイメージ・コピーをデータベース・リカバリー管理機能 (DBRC) に自動的に登録します。 その他の方法で作成されたイメージ・コピー、あるいはイメージ・コピー・ユーティリティーの EXEC パラメーター・リストで DBRC=N を指定して作成されたイメージ・コピーは、DBRC およびデータベース・リカバリー・ユーティリティー (DFSURDB0) でリカバリーに使用する前に、DBRC に登録する必要がある場合があります。

データベース変更が入っているログは、リカバリーの前に、変更累積ユーティリティー (DFSUCUM0) を使用して処理し、データベース変更レコードのすべてをリカバリー用に統合し、最適化できます。データ共用環境では、変更累積ユーティリティーを使用することが必須です。非データ共用環境では、変更累積ユーティリティーの使用はオプションですが、これを使用することは、パフォーマンス上の理由で、利点があります。

DBRC は、順方向リカバリー処理を管理し、バックアップとリカバリーに関連したすべてのユーティリティー (リカバリーされるデータベースが DBRC に登録されていないケースを除く) に対して、必要な JCL を生成できます。データベース・リカバリー・ユーティリティー用に DBRC コマンド GENJCL.RECOV によって生成される JCL は、正しいイメージ・コピー、ログ、DBD 名、DD 名、およびタイム・スタンプを自動的に指定して組み込みます。

データベース・データ・セット、すなわち、データベース・レコードが入っているデータ・セットをリカバリーするプロセスは、全機能、HALDB 区画全機能、および高速機能 DEDB データベース・タイプを含むほとんどのデータベース・タイプで同じです。 HIDAM データベースの 1 次索引には、データベース・データ・セットのリカバリーと同じ処理、すなわち、イメージ・コピーとログを使用する方法が使用されます。

ただし、HALDB データベースの場合、PHIDAM 1 次索引および間接リスト・データ・セット (ILDS) は、バックアップがとられず、またリカバリーもされませ

ん。その代わりに、これらがサポートしているデータベース・データ・セットがリカバリーされると、HALDB 1 次索引および ILDS が、HALDB 索引/ILDS 再作成ユーティリティー (DFSPREC0) を使用して、再作成されます。

HALDB データベースのリカバリーについてのもう 1 つの違いは、DBRC コマンド GENJCL.RECOV が、HALDB データベースのすべての区画を一度にリカバリーするか、1 つの区画だけをリカバリーする JCL を生成できることです。また、GENJCL.RECOV コマンドには、高速機能 DEDB データベースの全体をリカバリーするか、個々の DEDB エリアをリカバリーする、同様のオプションがあります。

HALDB 1 次索引および ILDS を再作成するとき、DBRC コマンド GENJCL.USER を使用して、必要なデータ・セットを割り振る基幹 JCL メンバーを作成することを考慮します。 1 次索引データ・セットおよび ILDS には、個々の基幹 JCL メンバーが必要です。 ILDS を再作成するときには、HALDB 索引/ILDS 再作成ユーティリティーのフリー・スペース・オプションを使用するために、ユーティリティー制御ステートメントで ILEF または BOTHF を指定します。フリー・スペース・オプションは、VSAM DEFINE CLUSTER コマンドで要求されたフリ ー・スペースを組み込むには VSAM ロード・モードを使用します。

データ共用環境での順方向リカバリーでは、追加ステップとして、リカバリーするデータベースを共用しているすべての IMS システムのログを統合することが必要です。

順方向リカバリーでは、完全リカバリー または タイム・スタンプ・リカバリー のどちらかを使用できます。 完全リカバリーは、DBDS またはエリア・データ・セットを、RECON データ・セットに記録されている現状に復元します。タイム・スタンプ・リカバリーは、DBDS またはエリア・データ・セットを、ユーザーが指定する時点の内容に復元します。

以下のトピックでは、HIDAM および PHIDAM の両データベースの全順方向リカバリーを実行するステップの例について説明します。データベース・データ・セットのリカバリー処理はデータベース・タイプを通じて同じですので、HIDAM DBDS および PHIDAM DBDS のリカバリー用に説明されているステップは、HDAM、 PHDAM、および DEDB の各データベース・データ・セットにも適用されます。同様に、PHIDAM 内の ILDS 用のリカバリー・ステップは、PHDAM 内の ILDS にも適用されます。

関連概念:

667 ページの『DBRC を用いたリカバリーの監視』

関連タスク:

671 ページの『例: 非データ共用環境における HIDAM データベースのリカバリー』

673 ページの『非データ共用環境における PHIDAM データベースのリカバリー』

678 ページの『例: 非データ共用環境における単一 HALDB 区画のリカバリー』

679 ページの『例: データ共用環境における HIDAM データベースのリカバリー』

例: 非データ共用環境における HIDAM データベースのリカバリー

以下のステップは、非データ共用環境における、副次索引を持った HIDAM データベースの全順方向リカバリーの例です。

このデータベースは、そのデータベース・データ・セットとして、OSAM を使用しています。しかし、データベース・データ・セットは OSAM または VSAM のどちらでもかまいません。 1 次索引および副次索引は、常に VSAM です。このデータベースは DBRC に登録され、オンライン環境で作動し、データ共用に参加しません。データベースがデータ共用に参加しないので、ログに記録されるデータベース変更の累積は必要ありません。

HIDAM データベースの場合、データベース・データ・セット (DBDS) とは別に、1 次索引データ・セットのバックアップとリカバリーが必要です。ただし、データベースと 1 次索引の両方の DBD 名を含む DBRC グループを定義することによって、バックアップとリカバリーの処理を単純化できます。次に、DBRC グループに対して GENJCL コマンドを実行し、両方の DBD で必要な JCL を同時に生成します。

HIDAM データベースをリカバリーするには、次のステップを実行します。

1. リカバリーに必要なデータベース変更レコードが入っている OLDS がある場合は、DBRC コマンド GENJCL.ARCHIVE を実行して、必要な JCL を生成し、ログ保存ユーティリティー (DFSUARC0) を実行します。
2. ログ保存ユーティリティーを実行します。ログ保存ユーティリティーは、OLDS 内のレコードを SLDS に保存します。
3. OSAM データベース・データ・セットおよび VSAM KSDS 1 次索引データ・セットの両方を削除し、定義します。
4. GENJCL.RECOV コマンドを実行して、HIDAM データベースのリカバリーに必要な JCL をすべて生成します。JCL では、使用する正しいイメージ・コピー、該当するすべてのログ、正しいすべての DD 名、および、正しいタイム・スタンプを指定します。
5. GENJCL.RECOV コマンドによって生成された JCL を実行することによって、データベース・リカバリー・ユーティリティー (DFSURDB0) をデータベースに対して実行します。データベース・リカバリー・ユーティリティーは、イメージ・コピーおよびログに記録されているデータベース変更からデータベース・データ・セットをリカバリーします。
6. GENJCL.RECOV コマンドを実行して、HIDAM データベースの 1 次索引のリカバリーに必要な JCL をすべて生成します。JCL では、使用する正しいイメージ・コピー、該当するすべてのログ、正しいすべての DD 名、および、正しいタイム・スタンプを指定します。
7. GENJCL.RECOV コマンドによって生成された JCL を実行することによって、データベース・リカバリー・ユーティリティーを 1 次索引に対して実行します。データベース・リカバリー・ユーティリティーは、イメージ・コピーから 1 次索引データ・セットをリカバリーし、さらに、ログに記録されている 1 次索引の変更を使用してデータ・セットを更新します。
8. 副次索引 VSAM KSDS データ・セットを削除し、定義します。
9. 以下の方法のいずれかで、副次索引をリカバリーします。

- HIDAM DBDS をリカバリーする際に使用するものと同じ順方向リカバリー・ステップを使用する。
- 別売の索引ビルダー・ツールを使用して、副次索引を再作成する。

次の JCL に、副次索引がある HIDAM データベースをリカバリーするのに必要な JCL を生成する GENJCL.RECOV コマンドの例を示します。

```
//GENRECVX JOB CLASS=A,REGION=6M,
//          MSGCLASS=U,NOTIFY=&SYSUID
//MYLIB JCLLIB ORDER=(IMS.PROCLIB,IMSVS.CMXXX.TEAM01.JCL)
//*
//DELDEF1 EXEC PGM=IDCAMS
//*************************************************************
//*    D E F I N E     V S A M   D A T A    D B S
//*************************************************************
//SYSPRINT DD  SYSOUT=*
//SYSIN     DD DSN=IMSVS.CMXXX.TEAM01.UTIL(XSTAP),DISP=SHR
//          DD DSN=IMSVS.CMXXX.TEAM01.UTIL(XSTAX),DISP=SHR
//          DD DSN=IMSVS.CMXXX.TEAM01.UTIL(XSTAY),DISP=SHR
//*
//* RUN DB RECOVERY
//A   EXEC   DBRC,
//SYSIN DD *
 GENJCL.RECOV DBD(X1STAP) LIST DEFAULTS(T01DFLT) ONEJOB JOB(T01RJOB) -
   MEMBER(RECV1JCL)
 GENJCL.RECOV DBD(X1STAX) LIST DEFAULTS(T01DFLT) ONEJOB JOB(T01RJOB) –
 GENJCL.RECOV DBD(X1STAY) LIST DEFAULTS(T01DFLT) ONEJOB JOB(T01RJOB) –
```

次の例に、副次索引がある HIDAM データベースをリカバリーするために GENJCL.RECOV コマンドによって生成される JCL を示します。

```
//T01RECOV JOB TIME=1,MSGCLASS=H,REGION=4096K,
//  CLASS=A
//RCV1 EXEC PGM=DFSRRC00,REGION=1300K,
//             PARM='UDR,DFSURDB0,X1STAP,,,,,,,,,,,,,,,,,'
//*
//*     THIS JCL ORIGINATES FROM THE USER'S 'JCLPDS' LIBRARY.
//*     KEYWORDS ARE REPLACED BY THE GENJCL FUNCTION OF
//*     THE IMS/VS DATA BASE RECOVERY CONTROL FEATURE.
//*
//*              JCL FOR RECOVERY.
//*
//STEPLIB   DD DSN=IMS.SDFSRESL,DISP=SHR
//SYSPRINT  DD SYSOUT=A
//IMS       DD DSN=IMS.DBDLIB,DISP=SHR
//X1STAP    DD DSN=IMSVS.CMXXX.TEAM01.X1STAP,
//             DISP=OLD
//DFSUDUMP  DD DSN=IMSVS.IMS1.TEAM01.X1STAP.D2006157.T155726,
//             UNIT=3390,
//             VOL=(PRIVATE,,,,SER=(SM4104)),
//             LABEL=(1,SL),
//             DISP=(OLD,KEEP),DCB=BUFNO=10
//DFSVDUMP  DD DUMMY
//DFSUCUM   DD DUMMY
//DFSULOG   DD DUMMY
//DFSVSAMP  DD *
//SYSIN     DD *
//*...
//T01RECOV JOB TIME=1,MSGCLASS=H,REGION=4096K,
//  CLASS=A
//RCV1 EXEC PGM=DFSRRC00,REGION=1300K,
//             PARM='UDR,DFSURDB0,X1STAX,,,,,,,,,,,,,,,,,'
//STEPLIB   DD DSN=IMS.SDFSRESL,DISP=SHR
//SYSPRINT  DD SYSOUT=A
//IMS       DD DSN=IMS.DBDLIB,DISP=SHR
```

```
//X1STAX    DD DSN=IMSVS.CMXXX.TEAM01.X1STAX,
//             DISP=OLD
//DFSUDUMP  DD DSN=IMSVS.IMS1.TEAM01.X1STAX.D2006157.T155727,
//             UNIT=3390,
//             VOL=(PRIVATE,,,,SER=(SM4106)),
//             LABEL=(1,SL),
//             DISP=(OLD,KEEP),DCB=BUFNO=10
//DFSVDUMP  DD DUMMY
//DFSUCUM   DD DUMMY
//DFSULOG   DD DUMMY
//DFSVSAMP  DD *
//SYSIN     DD *
//*...
//T01RECOV JOB TIME=1,MSGCLASS=H,REGION=4096K,
//  CLASS=A
//RCV1 EXEC PGM=DFSRRC00,REGION=1300K,
//             PARM='UDR,DFSURDB0,X1STAY,,,,,,,,,,,,,,,,,'
//STEPLIB   DD DSN=IMS.SDFSRESL,DISP=SHR
//SYSPRINT  DD SYSOUT=A
//IMS       DD DSN=IMS.DBDLIB,DISP=SHR
//X1STAX    DD DSN=IMSVS.CMXXX.TEAM01.X1STAY,
//             DISP=OLD
//DFSUDUMP  DD DSN=IMSVS.IMS1.TEAM01.X1STAY.D2006157.T155727,
//             UNIT=3390,
//             VOL=(PRIVATE,,,,SER=(SM4106)),
//             LABEL=(1,SL),
//             DISP=(OLD,KEEP),DCB=BUFNO=10
//DFSVDUMP  DD DUMMY
//DFSUCUM   DD DUMMY
//DFSULOG   DD DUMMY
//DFSVSAMP  DD *
//SYSIN     DD *
```

関連概念:

669 ページの『データベースのリカバリーの概説』

関連タスク:

679 ページの『例: データ共用環境における HIDAM データベースのリカバリー』

非データ共用環境における PHIDAM データベースのリカバリー

次のステップは、2 つの区画と区分副次索引 (PSINDEX) がある PHIDAM データベースの全順方向リカバリーの例です。

このデータベースは、そのデータベース・データ・セットとして、OSAM を使用しています。しかし、データベース・データ・セットは OSAM または VSAM のどちらでもかまいません。 1 次索引および副次索引は、常に VSAM です。このデータベースは DBRC に登録され、オンライン環境で作動します。データベースは、データ共用に参加しません。データベースがデータ共用に参加しないので、ログに記録されるデータベース変更の累積は必要ありません。

PHIDAM データベースをリカバリーするには、次のステップを実行します。

1. リカバリーに必要なデータベース変更レコードが入っている OLDS がある場合は、DBRC コマンド GENJCL.ARCHIVE を実行して、必要な JCL を生成し、ログ保存ユーティリティー (DFSUARC0) を実行します。

   GENJCL 出力の DBRC JCLOUT DD ステートメントが内部読み取りプログラムに送信されると、保存ジョブが自動的に開始されます。

2. 保存ジョブが自動的に開始されない場合は、OLDS 内に未保存のレコードがある各共用 IMS システムで、ログ保存ユーティリティーを実行します。 ログ保存ユーティリティーは、OLDS 内のレコードを SLDS に保存します。
3. OSAM データベース・データ・セットを削除し、定義します。
4. GENJCL.RECOV コマンドを実行して、PHIDAM データベースのリカバリーに必要な JCL をすべて生成します。JCL では、使用する正しいイメージ・コピー、正しいログ、正しい DD 名、および正しいタイム・スタンプを指定します。
5. GENJCL.RECOV コマンドによって生成された JCL を実行することによって、データベース・リカバリー・ユーティリティー (DFSURDB0) をデータベースに対して実行します。データベース・リカバリー・ユーティリティーは、イメージ・コピーおよびログに記録されているデータベース変更からデータベース・データ・セットをリカバリーします。
6. 1 次索引データ・セットおよび ILDS の両方を削除し、定義します。
7. HALDB 索引/ILDS 再作成ユーティリティー (DFSPREC0) を実行して、各区画に 1 次索引と ILDS の両方を再作成します。

   HALDB 索引/ILDS 再作成ユーティリティーのフリー・スペース・オプションを選択するには BOTHF を指定します。フリー・スペース・オプションは、DEFINE CLUSTER コマンドの FREESPACE パラメーターで要求されたフリー・スペースを組み込むには VSAM ロード・モードを使用します。

   HALDB 索引/ILDS 再作成ユーティリティーは、一時に 1 つの区画の 1 次索引データ・セットと ILDS を再作成します。しかし、このユーティリティーの複数インスタンスを複数の区画で並列に実行できます。
8. 副次索引 VSAM KSDS データ・セットを削除し、定義します。
9. 以下の方法のいずれかで、PSINDEX をリカバリーします。
   - PHIDAM DBDS をリカバリーする際に使用するものと同じ順方向リカバリー・ステップを使用する。
   - 別売の索引ビルダー・ツールを使用して、副次索引を再作成する。

次のコードに、PHIDAM データベースをリカバリーする JCL を生成する GENJCL.RECOV コマンドの例を示します。

```
//DBRC     EXEC PGM=DSPURX00
//STEPLIB  DD DISP=SHR,DSN=IMS.SDFSRESL
//         DD DISP=SHR,DSN=IMS.MDALIB
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//IMS      DD DISP=SHR,DSN=IMS.DBDLIB
//JCLPDS   DD DISP=SHR,DSN=IMS.PROCLIB
//JCLOUT   DD DISP=SHR,DSN=JOUKO4.HALDB.CNTL(RECOVOUT)
//JCLOUTS  DD SYSOUT=*
//SYSIN    DD *
GENJCL.RECOV NOJOB DBD(NORDDB) MEMBER(RECOVJCL)
/*
```

次のコードに、2 つの区画がある PHIDAM データベースをリカバリーするために必要な JCL を示します。

```
//RCV1 EXEC PGM=DFSRRC00,
//            PARM='UDR,DFSURDB0,NORDDB,,,,,,,,,,Y,,,,,,,,'
//STEPLIB   DD DISP=SHR,DSN=IMS.SDFSRESL
```

```
//SYSPRINT  DD SYSOUT=*
//SYSUDUMP  DD SYSOUT=*
//IMS       DD DISP=SHR,DSN=IMS.DBDLIB
//NORDDB1A    DD DSN=IMSPSA.IM0A.NORDDB.A00001,
//             DISP=OLD,
//             DCB=BUFNO=10
//DFSUDUMP  DD DSN=JOUKO3.IMCOPY2.NORDDB.C01,
//             DISP=OLD,DCB=BUFNO=10
//DFSVDUMP  DD DUMMY
//DFSUCUM   DD DUMMY
//DFSULOG   DD DUMMY
//DFSVSAMP  DD DISP=SHR,
//             DSN=IMS.PROCLIB(DFSVSMDB)
//SYSIN     DD *
S NORDDB NORDDB1A
/*
//RCV2 EXEC PGM=DFSRRC00,
//             PARM='UDR,DFSURDB0,NORDDB,,,,,,,,,,Y,,,,,,,,'
//STEPLIB   DD DISP=SHR,DSN=IMS.SDFSRESL
//SYSPRINT  DD SYSOUT=*
//SYSUDUMP  DD SYSOUT=*
//IMS       DD DISP=SHR,DSN=IMS.DBDLIB
//NORDDB2A    DD DSN=IMSPSA.IM0A.NORDDB.A00002,
//             DISP=OLD,
//             DCB=BUFNO=10
//DFSUDUMP  DD DSN=IMSPSA.IMCOPY.NORDDB2,
//             DISP=OLD,DCB=BUFNO=10
//DFSVDUMP  DD DUMMY
//DFSUCUM   DD DUMMY
//DFSULOG   DD DUMMY
//DFSVSAMP  DD DISP=SHR,
//             DSN=IMS.PROCLIB(DFSVSMDB)
//SYSIN     DD *
S NORDDB NORDDB2A
/*
```

次のコードに、PSINDEX をリカバリーする JCL を生成する GENJCL.RECOV コマンドの例を示します。

```
//DBRC     EXEC PGM=DSPURX00
//STEPLIB  DD DISP=SHR,DSN=IMS.SDFSRESL
//         DD DISP=SHR,DSN=IMS.MDALIB
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//IMS      DD DISP=SHR,DSN=IMS.DBDLIB
//JCLPDS   DD DISP=SHR,DSN=IMS.PROCLIB
//JCLOUT   DD DISP=SHR,DSN=JOUKO4.HALDB.CNTL(RECOVOU3)
//JCLOUTS  DD SYSOUT=*
//SYSIN    DD *
GENJCL.RECOV DBD(CUSTSI) ONEJOB LIST MEMBER(RECOVJCL)
```

次のコードに、2 つの区画を持つ PSINDEX をリカバリーするための JCL を示します。

```
//IVPGNJCL JOB (999,POK),
// 'JJ',
// CLASS=A,MSGCLASS=X,MSGLEVEL=(1,1),
// REGION=64M
/* JOBPARM SYSAFF=SC42
//RCV1 EXEC PGM=DFSRRC00,
//             PARM='UDR,DFSURDB0,CUSTSI,,,,,,,,,,Y,,,,,,,,'
//STEPLIB   DD DISP=SHR,DSN=IMS.SDFSRESL
//SYSPRINT  DD SYSOUT=*
//SYSUDUMP  DD SYSOUT=*
//IMS       DD DISP=SHR,DSN=IMS.DBDLIB
//CUSTSI1A    DD DSN=IMSPSA.IM0A.CUSTSI.A00001,
```

```
//             DISP=OLD
//DFSUDUMP  DD DSN=IMSPSA.IC1.CUSTSI1.CUSTSI1A.D03073.T143654,
//             DISP=OLD,DCB=BUFNO=10
//DFSVDUMP  DD DUMMY
//DFSUCUM   DD DUMMY
//DFSULOG   DD DUMMY
//DFSVSAMP  DD DISP=SHR,
//             DSN=IMS.PROCLIB(DFSVSMDB)
//SYSIN     DD *
S CUSTSI CUSTSI1A
/*
//RCV2 EXEC PGM=DFSRRC00,
//             PARM='UDR,DFSURDB0,CUSTSI,,,,,,,,,,Y,,,,,,,,'
//STEPLIB   DD DISP=SHR,DSN=IMS.SDFSRESL
//SYSPRINT  DD SYSOUT=*
//SYSUDUMP  DD SYSOUT=*
//IMS       DD DISP=SHR,DSN=IMS.DBDLIB
//CUSTSI2A    DD DSN=IMSPSA.IM0A.CUSTSI.A00002,
//             DISP=OLD
//DFSUDUMP  DD DSN=IMSPSA.IC1.CUSTSI2.CUSTSI2A.D03073.T143654,
//             DISP=OLD,DCB=BUFNO=10
//DFSVDUMP  DD DUMMY
//DFSUCUM   DD DUMMY
//DFSULOG   DD DUMMY
//DFSVSAMP  DD DISP=SHR,
//             DSN=IMS.PROCLIB(DFSVSMDB)
//SYSIN     DD *
S CUSTSI CUSTSI2A
/*
```

関連概念:

669 ページの『データベースのリカバリーの概説』

関連資料:

JCL とユーザー定義出力の生成 (コマンド)

データ共用環境における PHIDAM データベースのリカバリー

次のステップは、2 つの区画と区分 2 次索引 (PSINDEX) がある PHIDAM データベースの全順方向リカバリーの例です。 PHIDAM データベースは、複数の IMS システムによって共用されます。

データベース・データ・セットのイメージ・コピーが使用可能でなければなりません。

PHIDAM データベース・データ・セットを DBRC 変更累積グループのメンバーとして登録する必要があります。

このデータベースは、そのデータベース・データ・セットとして、OSAM を使用しています。しかし、データベース・データ・セットは OSAM または VSAM のどちらでもかまいません。 1 次索引および副次索引は、常に VSAM です。このデータベースは DBRC に登録され、オンライン環境で作動します。データベースは、複数の IMS システムによって共用されます。データベースが共用されるため、ログに記録されるデータベース変更の累積が必要になります。

全リカバリーでは、副次索引のリカバリーは必須ではないため、以下の手順の例では、副次索引のリカバリーに関するステップはオプションです。

IMS は、HALDB 区画の索引データ・セットまたは間接リスト・データ・セット (ILDS) を再作成するための基幹 JCL メンバー DSPUPJCL を提供します。

PHIDAM データベースをリカバリーするには、次のステップを実行します。

1. リカバリーに必要なデータベース変更レコードが入っている OLDS がある場合は、DBRC コマンド GENJCL.ARCHIVE を実行して、必要な JCL を生成し、ログ保存ユーティリティー (DFSUARC0) を実行します。

   GENJCL 出力の DBRC JCLOUT DD ステートメントが内部読み取りプログラムに送信されると、保存ジョブが自動的に開始されます。
2. 保存ジョブが自動的に開始されない場合は、OLDS 内に未保存のレコードがある各共用 IMS システムで、ログ保存ユーティリティーを実行します。 ログ保存ユーティリティーは、OLDS 内のレコードを SLDS に保存します。
3. GENJCL.CA コマンドを実行して、IMS システムのログを最新の CA にマージします。
4. OSAM データベース・データ・セットを削除し、定義します。
5. GENJCL.RECOV コマンドを実行して、PHIDAM データベースのリカバリーに必要な JCL をすべて生成します。JCL では、使用する正しいイメージ・コピー、正しいログ、正しい変更累積データ・セット、正しい DD 名、および正しいタイム・スタンプを指定します。
6. GENJCL.RECOV コマンドによって生成された JCL を実行することによって、データベース・リカバリー・ユーティリティー (DFSURDB0) をデータベースに対して実行します。データベース・リカバリー・ユーティリティーは、イメージ・コピーおよびログに記録されているデータベース変更からデータベース・データ・セットをリカバリーします。
7. 1 次索引データ・セットおよび ILDS の両方を削除し、定義します。
8. HALDB 索引/ILDS 再作成ユーティリティー (DFSPREC0) を実行して、各区画に 1 次索引と ILDS の両方を再作成します。

   HALDB 索引/ILDS 再作成ユーティリティーのフリー・スペース・オプションを選択するには BOTHF を指定します。フリー・スペース・オプションは、DEFINE CLUSTER コマンドの FREESPACE パラメーターで要求されたフリー・スペースを組み込むには VSAM ロード・モードを使用します。

   HALDB 索引/ILDS 再作成ユーティリティーは、一時に 1 つの区画の 1 次索引データ・セットと ILDS を再作成します。しかし、このユーティリティーの複数インスタンスを複数の区画で並列に実行できます。
9. オプションで、副次索引 VSAM KSDS データ・セットを削除し、定義します。
10. オプションで、以下の方法のいずれかで、PSINDEX をリカバリーします。
    - PHIDAM DBDS をリカバリーする際に使用するものと同じ順方向リカバリー・ステップを使用する。
    - 別売の索引ビルダー・ツールを使用して、副次索引を再作成する。

関連概念:

669 ページの『データベースのリカバリーの概説』

例: 非データ共用環境における単一 HALDB 区画のリカバリー

単一 HALDB 区画の全順方向リカバリーを実行するには、DBRC コマンド GENJCL.RECOV の DBD パラメーターに HALDB マスター・データベース名ではなく区画名を指定します。

GENJCL.RECOV コマンドは、指定した区画だけをリカバリーする JCL を生成します。 DBDS がリカバリーされたら、1 次索引と ILDS を再作成してください。

PHIDAM データベース内の単一区画をリカバリーするには、次のステップを実行します。

1. リカバリーに必要なデータベース変更レコードが入っている OLDS がある場合は、DBRC コマンド GENJCL.ARCHIVE を実行して、必要な JCL を生成し、ログ保存ユーティリティー (DFSUARC0) を実行します。
2. ログ保存ユーティリティーを実行します。ログ保存ユーティリティーは、OLDS 内のレコードを SLDS に保存します。
3. OSAM パーティション・データベース・データ・セットを削除し、定義します。
4. DBD パラメーターに区画名を指定して GENJCL.RECOV コマンドを実行し、区画のデータベース・データ・セットのリカバリーに必要な JCL を生成します。 JCL では、使用する正しいイメージ・コピー、正しいログ、正しい DD 名、および正しいタイム・スタンプを指定します。
5. GENJCL.RECOV コマンドによって生成された JCL を実行することによって、データベース・リカバリー・ユーティリティー (DFSURDB0) を区画に対して実行します。指定した区画について、データベース・リカバリー・ユーティリティーは、イメージ・コピーおよびログに記録されているデータベース変更からデータベース・データ・セットをリカバリーします。
6. 1 次索引データ・セットおよび ILDS の両方を削除し、定義します。
7. HALDB 索引/ILDS 再作成ユーティリティー (DFSPREC0) を実行して、区画に 1 次索引と ILDS の両方を再作成します。HALDB 索引/ILDS 再作成ユーティリティーのフリー・スペース・オプションを選択するには BOTHF を指定します。

   フリー・スペース・オプションは、DEFINE CLUSTER コマンドの FREESPACE パラメーターで要求されたフリ ー・スペースを組み込むには VSAM ロード・モードを使用します。

   HALDB 索引/ILDS 再作成ユーティリティーは、一時に 1 つの区画の 1 次索引データ・セットと ILDS を再作成します。しかし、このユーティリティーの複数インスタンスを複数の区画で並列に実行できます。

次のコードに、単一 HALDB 区画をリカバリーする JCL を生成する GENJCL.RECOV コマンドの例を示します。

```
//DBRC     EXEC PGM=DSPURX00
//STEPLIB  DD DISP=SHR,DSN=IMS.SDFSRESL
//         DD DISP=SHR,DSN=IMS.MDALIB
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//IMS      DD DISP=SHR,DSN=IMS.DBDLIB
//JCLPDS   DD DISP=SHR,DSN=IMS.PROCLIB
```

```
//JCLOUT   DD DISP=SHR,DSN=JOUKO4.HALDB.CNTL(RECOVOUT)
//JCLOUTS  DD SYSOUT=*
//SYSIN    DD *
  GENJCL.RECOV NOJOB DBD(NORDDB1) MEMBER(RECOVJCL)
/*
```

次のコードに、単一 HALDB 区画をリカバリーするために、GENJCL.RECOV コマンドによって生成される JCL を示します。

```
//RCV1 EXEC PGM=DFSRRC00,
//            PARM='UDR,DFSURDB0,NORDDB,,,,,,,,,,Y,,,,,,,'
//*
//STEPLIB  DD DISP=SHR,DSN=IMS.SDFSRESL
//         DD DISP=SHR,DSN=IMS.MDALIB
//IMS      DD DISP=SHR,DSN=IMS.DBDLIB
//SYSPRINT  DD SYSOUT=*
//SYSUDUMP  DD SYSOUT=*
//NORDDB1A    DD DSN=IMSPSA.IM0A.NORDDB.A00001,
//             DISP=OLD,
//             DCB=BUFNO=10
//DFSUDUMP  DD DSN=IMSPSA.NORDDB1.IMCOPY,
//             DISP=OLD,DCB=BUFNO=10
//DFSVDUMP  DD DUMMY
//DFSUCUM   DD DUMMY
//DFSULOG   DD DUMMY
//DFSVSAMP DD DSN=IMS.PROCLIB(DFSVSM0S),DISP=SHR
//SYSIN    DD *
S NORDDB NORDDB1A
/*
```

関連概念:

669 ページの『データベースのリカバリーの概説』

例: データ共用環境における HIDAM データベースのリカバリー

以下のステップでは、データ共用環境で、HIDAM データベースの全順方向リカバリーを実行します。

このデータベースは、そのデータベース・データ・セットとして、OSAM を使用しています。しかし、データベース・データ・セットは OSAM または VSAM のどちらでもかまいません。 1 次索引および副次索引は、常に VSAM です。このデータベースは DBRC に登録され、オンライン環境で作動します。データベースはデータ共用に参加するので、データベースのリカバリーの前に、データベース変更累積ユーティリティー (DFSUCUM0) を実行する必要があります。データベース変更累積ユーティリティーは、データベースを共用するそれぞれの IMS システムのログに記録されているデータベース変更を統合してソートします。

データ共用環境におけるリカバリー・ステップは、データベース変更をログ内に累積する必要がある点を除き、非データ共用環境におけるリカバリー・ステップと同じです。

HIDAM データベースの場合、データベース・データ・セット (DBDS) とは別に、1 次索引データ・セットのバックアップとリカバリーが必要です。ただし、それぞれのデータベース定義に定義されているデータベースと 1 次索引の両方のデータベース名が入っている DBRC グループを定義することによって、バックアップとリカバリーの処理を単純化できます。次に、DBRC グループに対して GENJCL コマンドを実行し、両方のデータベースで必要な JCL を同時に生成します。

データ共用環境で HIDAM データベースをリカバリーするには、次のステップを実行します。

1. データベースを共用しているそれぞれの IMS システム内の OLDS が、DBRC コマンド GENJCL.ARCHIVE の発行によって保存されていることを確認します。GENJCL.ARCHIVE コマンドは、ログ保存ユーティリティー (DFSUARC0) を実行するのに必要な JCL を生成します。
2. 共用 IMS システムのそれぞれで、ログ保存ユーティリティーを実行します。ログ保存ユーティリティーは、OLDS 内のレコードを SLDS に保存します。
3. DBRC コマンド GENJCL.CA を実行して、変更累積 JCL を作成します。
4. データベース変更累積ユーティリティーを実行し、データベース変更ログ・レコードを統合し、ソートし、さらに変更累算データ・セットに保管します。変更累積ユーティリティーは、変更累積データ・セットを DBRC に登録します。
5. OSAM データベース・データ・セットおよび VSAM KSDS 1 次索引データ・セットの両方を削除し、定義します。
6. GENJCL.RECOV コマンドを実行して、HIDAM データベースのリカバリーに必要な JCL をすべて生成します。JCL では、使用する正しいイメージ・コピー、該当するログ、正しい DD 名、および正しいタイム・スタンプを指定します。
7. GENJCL.RECOV コマンドによって生成された JCL を実行することによって、データベース・リカバリー・ユーティリティー (DFSURDB0) をデータベースに対して実行します。データベース・リカバリー・ユーティリティーは、イメージ・コピーおよびログに記録されているデータベース変更からデータベース・データ・セットをリカバリーします。
8. GENJCL.RECOV コマンドを実行して、HIDAM データベースの 1 次索引のリカバリーに必要な JCL をすべて生成します。JCL では、使用する正しいイメージ・コピー、該当するすべてのログ、正しいすべての DD 名、および、正しいタイム・スタンプを指定します。
9. GENJCL.RECOV コマンドによって生成された JCL を実行することによって、データベース・リカバリー・ユーティリティーを 1 次索引に対して実行します。データベース・リカバリー・ユーティリティーは、イメージ・コピーおよびログに記録されている 1 次索引変更から 1 次索引をリカバリーします。

次の JCL に、データ共用環境における HIDAM データベース用にデータベース変更累積ユーティリティーを実行するのに必要な JCL を生成する GENJCL.CA コマンドの例を示します。

```
//STEP1   EXEC PGM=DSPURX00,REGION=4096K
//STEPLIB   DD  DISP=SHR,DSN=IMS.SDFSRESL
//IMS       DD  DSN=IMS.DBDLIB,DISP=SHR
//JCLPDS    DD  DSN=IMS.JCLLIB,DISP=SHR
//JCLOUT    DD  SYSOU&JCLOUT
//SYSPRINT  DD  SYSOUT=*
//SYSIN     DD  *
GENJCL.CA GRPNAME(NORDDBCA) JOB MEMBER(CAJCLF) LIST JOB(CAJOB)
//*
```

次の JCL に、 GENJCL.CA コマンドによって生成される JCL の例を示します。

```
//CA1  EXEC PGM=DFSUCUM0,PARM='CORE=100000,DBRC=Y'
//*            JCL FOR CHANGE ACCUMULATION.
//STEPLIB   DD DISP=SHR,DSN=IMS.SDFSRESL
//SYSPRINT  DD SYSOUT=*
//SYSUDUMP  DD SYSOUT=*
```

```
//IMS       DD DISP=SHR,DSN=IMS.DBDLIB
//SYSOUT    DD SYSOUT=*
//SORTLIB   DD DSN=SYS1.SORTLIB,DISP=SHR
//SORTWK01  DD UNIT=SYSALLDA,SPACE=(CYL,(5),,CONTIG)
//SORTWK02  DD UNIT=SYSALLDA,SPACE=(CYL,(5),,CONTIG)
//SORTWK03  DD UNIT=SYSALLDA,SPACE=(CYL,(5),,CONTIG)
//SORTWK04  DD UNIT=SYSALLDA,SPACE=(CYL,(5),,CONTIG)
//SORTWK05  DD UNIT=SYSALLDA,SPACE=(CYL,(5),,CONTIG)
//SORTWK06  DD UNIT=SYSALLDA,SPACE=(CYL,(5),,CONTIG)
//DFSUCUMO  DD DUMMY,DCB=BLKSIZE=100
//DFSUCUMN  DD DSN=IMS.CA.NORDDBCA.CA172538,
//             DISP=(NEW,CATLG),
//             UNIT=SYSALLDA,
//             VOL=SER=DJX112,
//             SPACE=(CYL,(1,1))
//DFSULOG   DD DSN=IMS.SLDS.G2170V00,
//             DCB=RECFM=VB,
//             DISP=OLD
//          DD DSN=IMS.SLDS.G2171V00,
//             DCB=RECFM=VB,
//             DISP=OLD
//          DD DSN=IMS.SLDS.G2172V00,
//             DCB=RECFM=VB,
//             DISP=OLD
//          DD DSN=IMS.SLDS.G2173V00,
//             DCB=RECFM=VB,
//             DISP=OLD
//          DD DSN=IMS.SLDS.G2174V00,
//             DCB=RECFM=VB,
//             DISP=OLD
//          DD DSN=IMS.SLDS.G2175V00,
//             DCB=RECFM=VB,
//             DISP=OLD
//DFSUDD1   DD DUMMY
//SYSIN     DD *
DB0NORDDB  061392202286+0000         NORDDB1A
DB0NORDDB  061392202304+0000         NORDDB2A
DB0NORDDB  061392202321+0000         NORDDB3A
```

関連概念:

669 ページの『データベースのリカバリーの概説』

関連タスク:

671 ページの『例: 非データ共用環境における HIDAM データベースのリカバリー』

並行イメージ・コピーのリカバリー

並行イメージ・コピーは「ファジー」です。これは、IMS がデータベースのイメージ・コピーを作成する間にデータベースの更新を同時に行う場合があるためです。

例えば、IMS は以下のタイミングでデータベースを更新する可能性があります。

- 並行イメージ・コピーの開始前。IMS はこれらの更新を DBDS にまだ書き込んでいない場合があります。
- 並行イメージ・コピーが取られている間

並行イメージ・コピーからのデータ・セットの復元後、IMS は、イメージ・コピーがログから取られていた間に発生した変更を適用します。したがって、ファジー・イメージ・コピーを持つ DBDS をリカバリーするには、タイム・スタンプが並行イ

メージ・コピーの開始より前のログを、リカバリー・ジョブに提供することが必要になる場合があります。DBRC GENJCL.RECOV コマンドは、正しいログをリストした JCL を生成します。

データベース・リカバリー・サービスを使用してファジー・イメージ・コピーを持つ DBDS をリカバリーする場合は、データベース・リカバリー・サービスがリカバリーに必要なすべてのログを自動的に処理します。

関連概念:

649 ページの『並行イメージ・コピー』

HSSP イメージ・コピーのリカバリー

HSSP イメージ・コピーは、HSSP イメージ・コピーが正常に完了するたびに、ファジー・イメージ・コピーとして使用可能となります。

データベースのリカバリーに HSSP イメージ・コピーを使用するときは、HSSP イメージ・コピーを提供することを、データベース・リカバリー・ユーティリティーに知らせる必要があります。

イメージ・コピー処理が、イメージ・コピーを作成する間に失敗すると、IMS は、そのイメージ・コピー・データ・セットを使用可能なイメージ・コピー・データ・セットのプールに戻します。

関連概念:

652 ページの『HSSP イメージ・コピー』

DL/I 入出力エラーおよびリカバリー

IMS が DL/I 入出力エラーを処理する方法を使用すると、データベース・リカバリーを都合の良いときまで遅らせることができます。

エラーが再発生しないか、自動的に訂正できる場合は、リカバリーはまったく不要です。 IMS は、後のウォーム・リスタートまたは緊急時再始動で、入出力エラーに関する情報を再構成することによってデータ保全性を維持します。

入出力エラーが発生すると、IMS はメッセージ DFS0451I を出して、拡張エラー・キュー・エレメント (EEQE) を作成します。IMS が EEQE によって DBDS を正常に順方向にリカバリーすると、DBRC は RECON データ・セットの適切な項目を更新し、IMS は EEQE を除去します。

/DISPLAY DB コマンドを BKERR キーワードとともに使用し、データベースに関連する未解決の EEQE に関する情報を表示することができます。

関連概念:

639 ページの『データベース障害』

DEDB 満杯状態

DEDB エリアのデータベース満杯状態とは、DEDB のルート・アドレス可能エリアの 1 つまたは複数の部分が ISRT 呼び出しを受け入れられなくなったことを示します。

IMS は、同期点処理の際に、DEDB 順次従属セグメントのスペース不足状態を検出します。アプリケーション・プログラムは、警告の状況コードを受け取る場合がありますが、高速機能メッセージ・ドリブン・プログラムまたは MPP 領域から実行される ISRT 呼び出しによっては、領域が異常終了コード U0844 で異常終了する場合もあります。

推奨事項: 以下のいずれかの方法を使用して、DEDB エリア内のスペース使用量をトラッキングします。

- アプリケーション・プログラムは、位置 (POS) 呼び出しから得られる情報を使用することができます。
- マスター端末オペレーターは、適切な /DISPLAY DB コマンドまたは QUERY DB コマンドを使用してそれをモニターし、そのエリアのルート・アドレス可能部分と順次従属部分の両方における、CI の割り振られた合計、および使用された合計を表示することができます。

DEDB エリアがデータベース満杯状態に達した場合は、高速機能は、オフライン・アプリケーション・プログラム使用の必要をなくす 2 つのオンライン・ユーティリティーを使用して、データベースのアンロードおよび再ロード・オペレーションを行い、問題を解決します。その 2 つのユーティリティーを以下に示します。

- 順次従属の削除ユーティリティー。エリアの順次従属部分の、全部または一部の順次従属セグメントを削除します。これにより、スペースは、今後の順次従属の挿入に使用できるようになります。
- 直接再編成ユーティリティー。エリア内のスペース・フラグメントを除去します。このユーティリティーは、一度に 1 つの DEDB エリアに対して稼働します。

  推奨事項: 再編成を、挿入を許さない 1 つの作業単位に限定することができますが、多くのエリアまたはすべてのエリアを再編成してください。1 つの作業単位から解放される独立オーバーフロー・スペースは、そのエリア内のほかのすべての作業単位が使用できるようになります。

データベース許可の継続

IMS は、DBDS で書き込み入出力エラーが発生した場合にいくつかのアクションを実行します。

IMS が DBDS で書き込み入出力エラーを検出し、DBDS が DBRC に登録されていると、IMS は、以下のことを行います。

1. DBDS に書き込めなかったデータの EEQE を作成する
2. DBRC を呼び出す

データベースを DBRC に登録すると、DBRC は、以下のことを行って、DBDS の RECON データ・セットを更新します。

- DBDS レコードにフラグを立てて、リカバリーが必要であることを示す
- EEQE のアドレスを DBDS レコードに記録する
- DB レコードを増分して、EEQE をカウントする

リカバリー必要フラグによって、バッチ・イメージ・コピー・ユーティリティーのような IMS ユーティリティーに対する、新しい許可が禁止されます。また、リカ

バリー必要フラグが設定され、EEQE がない場合、DBRC は、リカバリー・ユーティリティー以外のすべての許可をリジェクトします。

データベースを DBRC に登録しない場合、IMS は、ウォーム・リスタートまたは緊急時再始動の際にログ・レコードから作成した EEQE が正しいと想定します。IMS は、/DBRECOVERY DB または UPDATE DB STOP(ACCESS) コマンドを使用してデータベースを停止する際、非登録データベースの EEQE 情報を廃棄します。

入出力エラーの再試行

IMS がデータベースを更新して閉じる際に、 IMS は、自動的に EEQE を再試行して読み取りエラーおよび書き込みエラーの有無を調べます。

IMS は、タスク生成されたサブシステムが、書き込みエラーがあるデータベースを使用することを許可しますが、 IMS は、正常に書き込まれなかったブロックまたは CI にアクセスすることは許可しません。これらのサブシステムのいずれかのアプリケーション・プログラムが、これらのブロックまたは CI のいずれかにアクセスする必要があるデータを要求すると、IMS は、そのプログラムに読み取りエラーを示す AO 状況コードを与えます。

入出力エラーの再試行が正常に行われたときは、IMS は EEQE を削除し、MTO および z/OS コンソールにメッセージを送ります。IMS は、再試行が正常に行われたエラーごとにメッセージ DFS0614I を送り、すべての未解決エラーを解決したときにメッセージ DFS0615I を送ります。

不良ポインターの訂正

通常、ユーザーのデータベースには不良ポインターは起こりません。起こった場合に考えられる原因はごくわずかです。

不良ポインターの一般的な原因は次のとおりです。

- データベースのバックアウトを実行するのに失敗
- 緊急時再始動を実行するのに失敗
- バックアウト時またはリカバリー時にログを省略

不良ポインターを訂正する通常の方法は、リカバリーを実行することです。しかし、不良ポインターは再編成を通して訂正できる場合がいくつかあります。これが可能な状況か、または不可能な状況かについて、以下に説明します。

- PC/PT ポインター。 HD アンロード・ユーティリティーは、修飾されていない GN 呼び出しを出してデータベースを読み取ります。不良ポインターが PC ポインターまたは PT ポインターである場合には、DL/I は不良ポインターに従うので、GN 呼び出しが失敗します。したがって、再編成を使用して PC ポインターまたは PT ポインターを訂正することはできません。
- LP/LT ポインター。 LP ポインターおよび LT ポインターは再編成時に再び設定されます。しかし、DL/I は、アンロード時に LP ポインターに従うことができます。論理子セグメントが直接 LP ポインターを収容しており、しかも論理親の連結キーが論理子セグメントに物理的に保管されていない場合には、DL/I は論理親の連結キーを構成するのに不良 LP ポインターに従います。これは、ABEND (異常終了) を起こす原因となります。

- LP ポインター。 DBR= を事前再編成ユーティリティーに指定し、データベースが直接 LP ポインターを持っている場合には、HD アンロード・ユーティリティーは古い LP ポインターを保管します。不良 LP ポインターは、論理親がない論理子が存在することを伝えるエラー・メッセージ (DFS879) を生成します。
- LP ポインター。論理子または論理親のデータベースについて、DBIL= を事前再編成ユーティリティーに指定した場合は、LP ポインターを解決するユーティリティーは、連結キーを使って論理親セグメントと論理子セグメントを対応付けます。新しい LP ポインターが作成されます。

RSR 環境におけるリカバリー

データベース・リカバリー・ユーティリティーは、RSR トラッキング・サイトで使用できます。

GENJCL.RECOV コマンドを使用して JCL を生成します。これにより、必要なイメージ・コピーおよび変更累積データ・セットが組み込まれます。

RSR トラッキング・サイトでデータベースのグループに対するすべてのリカバリー・ジョブが完了したら、/START DATABASE|AREA|DATAGRP コマンドを実行して、対象のデータベースで通常のトラッキングを行えるようにオンライン順方向リカバリー (OFR) を開始します。

OFR は、RSR によって古い状態のシャドー・データベースおよびエリアを現行のトラッキング状態に戻す処理であり、RSR データベース・レベル・トラッキング (DLT) サブシステムで、データベース作動可能レベルでトラッキングされるデータベースに対してのみ使用できます。

アクティブ・サイトでタイム・スタンプまでデータベースをリカバリーする場合は、そのデータベースが再び使用される前にデータベースのイメージ・コピーを作成し、トラッキング・サイトで DBRC コマンド NOTIFY.IC DBD(*name*) DDN(*name*) RUNTIME(*time_stamp*) を使用してデータベースを DBRC に登録します。その後に、データベースの完全リカバリーを実行してください。

リモート・サイト・リカバリーついての詳細な説明は、「*IMS V13* システム管理」を参照してください。

RSR 環境における DFSURDB0 ユーティリティーを使用したデータベースのリカバリー

RSR 環境でデータベースをリカバリーするステップでは、まずアクティブ・サイトでデータベースをリカバリーし、リカバリーされたアクティブ・サイト・データベースのイメージ・コピーを使用して、トラッキング・サイトでデータベースをリカバリーする必要があります。

RSR 環境でデータベース・リカバリー・ユーティリティーを使用してデータベースをリカバリーするステップは以下のとおりです。

- アクティブ・サイトでは次のことを行います。
  1. タイム・スタンプ・リカバリーを実行します。
  2. 当該データベースのイメージ・コピーを作成します。
  3. そのイメージ・コピーをトラッキング・サイトへ送信します。

- トラッキング・サイトでは次のことを行います。
  1. イメージ・コピーを受信します。
  2. NOTIFY.IC コマンドを使用して、DBRC へイメージ・コピーを登録します。
  3. リカバリー・ユーティリティーを実行して、イメージ・コピー (変更累積データがある場合はこれを含む) を適用します。
  4. /START DATABASE|AREA|DATAGRP コマンドを発行して、データベースでトラッキングが行えるようにします。/START DATABASE|AREA|DATAGRP コマンドによって、OFR がログからの変更を適用し、通常のトラッキングが再開されます。

RSR 環境におけるデータベース・ユーティリティー

リモート・サイト・リカバリー (RSR) 環境では、アクティブ・サイトにおけるユーティリティーの実行に関係のある特定の情報がトラッキング・サイトで使用できるようになっていなければなりません。

この情報の大部分は、RSR により自動的に処理されます。ただし、以下の点については、ユーザーが自分で行う必要があります。

- トラッキングされたデータベースがかかわってくるときは常に、次のユーティリティーが実行される場合には、DBRC がアクティブになっていなければなりません。そのユーティリティーとは、バッチ・バックアウト・ユーティリティー (DFSBBO00)、データベース変更累積ユーティリティー (DFSUCUM0)、データベース・リカバリー・ユーティリティー (DFSURDB0)、およびデータベース再編成ユーティリティーです。

  トラッキングされたデータベースがかかわってくるときは常に、データベース部分再編成ユーティリティー (DFSPRCT2) が実行される場合、 DBRC と IMS ロギングがアクティブでなければなりません。
- アクティブ・サイトからトラッキング・サイトへイメージ・コピーを送り、それらをトラッキング・サイト RECON データ・セットに記録するのは、ユーザーの責任です。

トラッキング・サイトで実行できるのは、次に挙げるデータベース・ユーティリティーだけです。

- データベース変更累積ユーティリティー (DFSUCUM0)
- データベース・イメージ・コピー・ユーティリティー (DFSUDMP0)
- データベース・リカバリー・ユーティリティー (DFSURDB0)
- データベース・イメージ・コピー 2 ユーティリティー (DFSUDMT0)

RSR テークオーバーの後にトラッキング・サイトであったサイトがアクティブ・サイトになると、HALDB データベースで ILDS と索引データ・セットをリカバリーするための索引/ILDS 再作成ユーティリティー (DFSPREC0) が新しいアクティブ・サイトで必要となります。

アクティブ・サイトでのデータベース・ユーティリティー検証

RSR サポートには、アクティブ・サイトで IMS データベース・ユーティリティーについてさらに検証することが必要です。トラッキングされたデータベースまたは区域については、以下の制約があります。

- トラッキングされたデータベースの 1 つの DBDS のタイム・スタンプ・リカバリーには、そのデータベースの他のあらゆる DBDS の対応するタイム・スタンプ・リカバリーが必要です。(データベース・リカバリー・ユーティリティーを除き) データベースに対する後続の許可要求は、すべての DBDS が同時点にリカバリーされるまで、リジェクトされます。
- リカバリー・ユーティリティーは、ログ・ボリューム境界ではなく、 USID 境界へのリカバリーをサポートします。そのため、DBRC のタイム・スタンプ・リカバリーに対する要件を減らすことができます。特に、ログ・ボリューム境界要件は適用されません。したがって、タイム・スタンプ・リカバリーの準備を行う際に `/DBRECOVERY` コマンドで `NOFEOV` キーワードを使用できます。

トラッキング・サイトでのデータベース・ユーティリティー検証

RSR サポートには、トラッキング・サイトで IMS データベース・ユーティリティーについてさらに検証することが必要です。対象となるデータベースまたは区域については、以下の制約があります。

- データベース再編成ユーティリティー

  トラッキング・サイトではデータベース再編成は使用できません。データベース再編成ユーティリティーが、トラッキング・サービス・グループ (SG) にサインオンする一方でトラッキングされたデータベースに対する許可を要求した場合、その許可はリジェクトされます。
- データベース・イメージ・コピー・ユーティリティー

  トラッキング DBDS または区域のイメージ・コピーは、このユーティリティーの実行時にトラッキング・サブシステムに対してそのデータベースまたは区域が許可されない限り、作成できます。イメージ・コピー・パラメーターに `NOCIC` を指定するとバッチ・イメージ・コピーを作成できますが、トラッキング・サイトで並行イメージ・コピー (`CIC`) を作成することはできません。

  トラッキング・サイトで作成されるイメージ・コピー・データ・セットは、シャドー・データベースがコピーされている間、HALDB マスターが普通はまだ更新中であるという点で、並行イメージ・コピーとよく似ています。 しかし、バッチ・イメージ・コピーの場合と同様、イメージ・コピーは、事実上、DBDS または区域の即時コピーです。これは、ユーティリティーが実行している間、トラッキングが延期されているためです。

  トラッキング・サイトで作成されるイメージ・コピー・データ・セットのタイム・スタンプ (`RUNTIME`) は、それが作成された時刻ではなく、アクティブ・サイトの RECON 内のデータベースまたは区域について記録された時刻です。そのため、トラッキング・サイト・イメージ・コピーには、アクティブ・サイト・データベースまたは区域と関連する「有効時刻」があります。この時刻は、データベースまたは区域に最後に適用された更新より前である場合もあります。 した

がって、トラッキング・サイト・イメージ・コピーを使用したリカバリーには、なんらかのデータの再処理が必要となります。

- データベース・イメージ・コピー 2 ユーティリティー

  トラッキング DBDS または区域のイメージ・コピーは、このユーティリティーの実行時にトラッキング・サブシステムに対してそのデータベースまたは区域が許可されない限り、作成できます。イメージ・コピー・パラメーターに NOCIC を指定するとバッチ・イメージ・コピーを作成できますが、トラッキング・サイトで並行イメージ・コピー (CIC) を作成することはできません。

  トラッキング・サイトで作成されるイメージ・コピー・データ・セットは、シャドー・データベースがコピーされている間、HALDB マスターが普通はまだ更新中であるという点で、並行イメージ・コピーとよく似ています。 しかし、バッチ・イメージ・コピーの場合と同様、イメージ・コピーは、事実上、DBDS または区域の即時コピーです。これは、ユーティリティーが実行している間、トラッキングが延期されているためです。

  トラッキング・サイトで作成されるイメージ・コピー・データ・セットのタイム・スタンプ (RUNTIME) は、それが作成された時刻ではなく、アクティブ・サイトの RECON 内のデータベースまたは区域について記録された時刻です。そのため、トラッキング・サイト・イメージ・コピーには、アクティブ・サイト・データベースまたは区域と関連する「有効時刻」があります。この時刻は、データベースまたは区域に最後に適用された更新より前である場合もあります。 したがって、トラッキング・サイト・イメージ・コピーを使用したリカバリーには、なんらかのデータの再処理が必要となります。

- データベース・リカバリー・ユーティリティー

  トラッキング DBDS または区域のデータベース・リカバリーは、ユーティリティーの実行時にそのデータベースまたは区域がトラッキング・サブシステムに対して許可されていない限り、実行できます。

  リカバリー作動可能レベル (RLT) でトラッキングされるブロック・レベル・データ共用サブシステムの場合、タイム・スタンプ・リカバリーはアクティブ・サイトで実行されるため、そのデータベースの各データ・セットのイメージ・コピーは、タイム・スタンプ・リカバリーに続くトラッキング・サイトに送る必要があります。

  データベース作動可能レベル (DLT) でトラッキングされるブロック・レベル・データ共用サブシステムの場合、タイム・スタンプ・リカバリーは、オンライン順方向リカバリーによりトラッキング・サイトで実行することができます。

- バッチ・バックアウト・ユーティリティー

  バッチ・バックアウトは、トラッキング・サイトでは使用できません。バッチ・バックアウトで、トラッキング SG へサインオンすることはできません。

RSR 環境における標準外イメージ・コピーを使用したデータベースのリカバリー

RSR 環境では、IMS システムがアクティブ・サブシステムとトラッキング・サブシステムのどちらであるかによって、標準外イメージ・コピーでデータベースをリカバリーする手順が若干異なります。

ただし、どちらのサブシステムでも、データベース・リカバリー・ユーティリティー (DFSURDB0) は標準外イメージ・コピーを入力として受け付けません。このため、標準外イメージ・コピーをリカバリーに使用する場合は、DFSURDB0 ユーティリティーを実行する前に、ほかの方法でイメージ・コピーからデータベース・データ・セットを復元する必要があります。

アクティブ・サブシステムでデータベースをリカバリーするには、次のステップを実行します。

1. 標準外イメージ・コピーから DBDS を復元します。
2. RECON データ・セットでの復元を記録するために、イメージ・コピーの実行時間を RCVTIME キーワードに指定した DBRC コマンド `NOTIFY.RECOV` を実行します。
3. リカバリーを完了させるために、USEDBDS キーワードを指定した DBRC コマンド `GENJCL.RECOV` を実行して、イメージ・コピー作成後に行われた変更を適用します。

トラッキング・サブシステムでデータベースをリカバリーするには、次のステップを実行します。

1. 最後に記録された標準外イメージ・コピーを使用して DBDS を復元します。
2. この復元を DBRC に記録するために、イメージ・コピー実行時間を RUNTIME キーワードに指定し、イメージ・コピーの USID を RUNUSID キーワードに指定した `NOTIFY.RECOV` コマンドを入力します。
3. データベースで `/START` コマンドを発行します。`/START DATABASE|AREA|DATAGRP` コマンドによって、オンライン順方向リカバリー (OFR) がログからの変更を適用し、通常のトラッキングが再開されます。

関連概念:

リモート・サイトの RSR のための IMS エラー処理 (システム管理)

リモート・サイト・リカバリー (RSR) の使用による IMS のリカバリー (オペレーションおよびオートメーション)

関連タスク:

660 ページの『RSR 環境におけるイメージ・コピー』

第 27 章 データベース・バックアウト

逆方向リカバリー、すなわちバックアウト は、リカバリーの 2 つの主要なタイプのうちの 1 つです。バックアウトを使用すると、誤っているかまたは加えたくない変更を既存の情報または作業から除去できます。

動的バックアウト

IMS は、さまざまな状況で、データベースへの変更を自動的にバックアウトします。

以下のイベントのいずれかが起こった場合は、常に IMS はデータベースへの変更を自動的にバックアウトします。

- アプリケーション・プログラムが異常終了した。
- アプリケーション・プログラムがロールバック呼び出し (ROLL または ROLB) を出したか、あるいはトークンを指定せずに ROLS 呼び出しを出した。
- アプリケーション・プログラムが使用不能なデータベースへのアクセスを試行したが、その前に INIT 呼び出しを出していなかった。
- デッドロックが発生した。

バッチでは、バッチ・アプリケーション・プログラムが異常終了したか、ROLB 呼び出しを出したか、あるいはトークンを指定せずに ROLS 呼び出しを出した場合は IMS が自動的かつ動的に変更をバックアウトするように (JCL の中で) 指定できます。このケースでは、ログ・データ・セットは DASD 上になければなりません。

ただし、GSAM DB の場合はバックアウトされませんが、XRST 呼び出しによる BMP の再始動プロセス中に位置変更されます。 XRST 呼び出しは、データ・セット・ポインターの位置を、呼び出しで指定されたチェックポイント ID に変更します。アプリケーションは、始動時にそのポイントから再開して先に進みます。XRST 呼び出しで指定されたチェックポイント ID は、動的バックアウトまたはバッチ・バックアウトによる非 GSAM DB のバックアウト先のチェックポイント ID と同じである必要があります。

動的バックアウトおよびコミット・ポイント

動的バックアウト時には、IMS はアプリケーション・プログラムがその最後のコミット・ポイント以降に行ったすべてのデータベース変更をバックアウトします。

コミット・ポイント (または同期点) は、バッチ・プログラムまたは BMP プログラムが CHKP 呼び出しを出したときにそのプログラム内で発生します。メッセージ・ドリブン・アプリケーション・プログラムの場合のコミット・ポイントは、トランザクション・モード (TRANSACT マクロの MODE パラメーターによって指定されている) に依存します。

コミット・ポイントが頻繁に発生するとパフォーマンスが低下しますが、これらには次のような利点もあります。

- IMS が出力メッセージ (応答) をよりすみやかに送信できるようになる。
- 緊急時再始動およびデータベース・リカバリーに要する時間が削減される。
- ロックのためのストレージ不足の回避に役立つ。
- バックアウトのための SLDS の読み取りの回避に役立つ。この読み取りは、パフォーマンスを低下させます。

アプリケーション・プログラムは、バッチ指向のもの (非メッセージ・ドリブン BMP) でも構わず、行ったデータベース変更をコミットする時点を CHKP 呼び出しを使用して選択できます。

以下の種類に属するプログラムは、ロールバック (ROLB) 呼び出しを出すことができます。

- 入力メッセージを獲得しようと試みるたびにコミット・ポイントを設定するメッセージ・ドリブン・アプリケーション・プログラム
- 非メッセージ・ドリブン・バッチ指向アプリケーション・プログラム

アプリケーション・プログラムが高速機能のリソースまたは外部サブシステムを使用している場合は、IMS は内部ロールバック呼び出しを出してデータを最後のコミット・ポイントまでバックアウトする可能性があります。この呼び出しの結果として、アプリケーション・プログラムは状況コード FD を受け取ります。

IMS は、ROLB 呼び出しに応答して以下の処理を実行します。

- 当該のプログラムがその最新の同期点以降に行ったすべてのデータベース変更に対するロックをバックアウトしてリリースする。
- そのプログラムの最新同期点以降のすべての出力メッセージを取り消す。
- 最新コミット・ポイント以降の最初の入力メッセージの最初のセグメントをアプリケーション・プログラムに戻す。これは、入力メッセージがこのポイントでまだ処理の途中にあり、さらにこの呼び出しによって入出力域が提供される場合に実行されます。

IMS が ROLB 処理を完了すると、アプリケーション・プログラムはその最後のコミット・ポイントから処理を再開します。

アプリケーション・プログラムが異常終了したために IMS が動的バックアウトを開始するときには、多くのケースで IMS はトランザクションとアプリケーション・プログラムの両方を停止させます。 MTO は、そのトランザクションまたはアプリケーション・プログラムを再始動できます。その異常終了の確かな原因を判別する前にそのアプリケーション・プログラムを再始動すると、そのプログラムが再び異常終了する可能性があります。ただし、トランザクションを再始動した場合は、トランザクションのキューイングは継続されます。

MTO がトランザクションを再始動する必要があるかどうかを判断する際には、そのトランザクションのモードを考慮に入れてください。

- 再始動するのが応答モードのトランザクションであり、かつその再始動に続いて IMS が新しいメッセージをエンキューする場合は、応答がまったく受信されないために、そのトランザクションを入力中の端末はロックされる。

- 応答モード・トランザクションを再始動しない場合は、端末オペレーターは IMS がトランザクションを停止させたというメッセージを受け取るが、発信元端末はロックされない。
- 当該のトランザクションが応答モード・トランザクションでない場合は、これを再始動すると、端末オペレーターが入力を継続できる。ただし、このケースでは、しばらくの間は応答が受信されない場合があることを端末オペレーターに警告するための手順を確立してください。

アプリケーション・プログラムが異常終了した場合は、IMS はマスター端末にメッセージ DFS554A を出します。長引くことはないと予測されるリソース不足 (短期ロックなど) が発生した場合は、IMS はこのメッセージを MPP 領域には出しません。このケースでは、IMS はアプリケーション・プログラムをバックアウトし、再スケジューリングするために入力メッセージをキューに戻します。 BMP が異常終了した場合は、z/OS オペレーターが BMP を再始動する必要があるため、IMS は常にメッセージ DFS554A を出します。

以下に、メッセージ DFS554A によって識別されるものを示します。

- アプリケーション・プログラム (PSB) 名
- トランザクション名
- システム完了コードまたはユーザー完了コード
- 入力論理端末名
- プログラムまたはトランザクションが停止したかどうか

入力メッセージの処理中にアプリケーション・プログラムが異常終了した場合は、IMS は入力端末またはマスター端末にメッセージ DFS555I も送信します。このエラー・メッセージは、そのアプリケーションが処理中であった最後の入力メッセージを IMS が廃棄したことを意味します。

この DFS555I メッセージには次のものが含まれています。

- システム完了コードまたはユーザー完了コード
- 当該の入力メッセージの最初の数文字 (最大 78 文字まで)
- 時刻と日付

関連情報:

DFS554A (メッセージおよびコード)

DFS555I (メッセージおよびコード)

バッチにおける動的バックアウト

バッチ環境では、SLDS が DASD 上にある場合は、IMS が疑似異常終了したとき、またはアプリケーション・プログラムが JCL 内で BKO=Y を指定して ROLB 呼び出しを出したときに、IMS が動的バックアウトを実行するよう要求できます。

このケースでは、IMS は最後のプログラム同期点までのバックアウトを実行します。 BKO=Y を指定すると、バッチに異常終了 U0828 は発生しません。

データベース・バッチ・バックアウト

バッチ・バックアウト・ユーティリティー (DFSBBO00) を使用すると、IMS バッチ・ジョブおよびオンライン・プログラムが行ったデータベース変更を除去できます。

(オンライン IMS サブシステムでは) 同じ PSB を使用する従属領域を複数もつことが可能なため、オンライン・ログに対してバッチ・バックアウトを実行すると、このユーティリティーが変更を複数の従属領域からバックアウトする場合があります。

バッチ・バックアウト・ユーティリティーを使用して、バッチ・ジョブが正常に完了しなかった最後のチェックポイントまで変更をバックアウトできます。当該のバッチ領域がデータ共用を使用しておらず、かつ BMP でもない場合は、バッチ・バックアウト・ユーティリティー用の JCL で CHKPT 制御ステートメントを使用して、そのバッチ領域に対する変更を (そのバッチ領域が正常に完了したかどうかは問わずに) 任意の有効チェックポイントまでバックアウトできます。BMP への変更をバックアウトする場合は CHKPT 制御ステートメントを指定しないでください。ユーティリティーはそれをリジェクトまたは無視します。

バッチ・バックアウト・ユーティリティーは、ログを順方向に読み取ります。バックアウトを行うのが最後のチェックポイントまででであっても指定されたチェックポイントまででであっても、このユーティリティーは、そのチェックポイントからログの終わりまでの間に発生した、すべてのデータベースへのすべての変更をバックアウトしようと試行します。 BMP をバックアウトする場合には、このユーティリティーはそれらを常にログ上の最後のチェックポイントまでバックアウトします。

IMS DBCTL 環境では、バッチ・バックアウト・ユーティリティーはすべての未完了更新をバックアウトしますが、未確定更新をバックアウトするのは COLDSTART オプションが指定されている場合のみです。

動的バックアウトまたは緊急時再始動時のバックアウトが失敗した場合は、IMS はバックアウトが完了していないデータベースを停止し、ユーザーがそれらのデータベースを再始動するときにバックアウトを再試行します。

バッチ・バックアウト・ユーティリティーを使用する場合

以下の状況では、バッチ・バックアウト・ユーティリティーを使用してください。

- バッチ・ジョブに障害が起こり、かつユーザーが自動的に動的バックアウトを要求していなかった場合。
- バッチ・ジョブに障害が起こり、かつユーザーは動的バックアウトを要求してあったが、その障害が疑似異常終了ではなかった場合。
- NOBMP オプションを指定して IMS の緊急時再始動を行い、かつ障害発生時に BMP がアクティブであった場合。この場合は、このユーティリティーをそれぞれの BMP ごとに 1 回ずつ実行する必要があります。このユーティリティーを再始動が完了する前に実行するときは、ACTIVE 制御ステートメントを使用する必要があります。
- COLDBASE オプションを指定して IMS の緊急時再始動を行い、かつ障害発生時に PSB がアクティブであった場合。この PSB は、全機能 DL/I データベー

スを更新したアプリケーション・プログラムによって使用された PSB (MPP、BMP、および混合モード IFP も含む) のことです。この場合は、バッチ・バックアウトをそれぞれの PSB ごとに 1 回ずつ実行する必要があります。

また、それに対する未確定リカバリー単位 (UOR) が存在する PSB の場合も、それぞれについて 1 回ずつバッチ・バックアウトを実行する必要があります。これを行うと、全機能データベースが、コミット・ポイントまでは更新されない高速機能データベースと同じ状態になります。未解決の未確定 UOR はバックアウトしないことを選択した場合は、その UOR を RECON バックアウト・レコードから除去しなければなりません。

上記の各バックアウトを再始動が完了する前に開始したい場合は、バッチ・バックアウト・ユーティリティーに対して COLDSTART 制御ステートメントを指定する必要があります。再始動後に実行するバックアウトについては COLDSTART ステートメントを指定する必要はありません。

DBRC は、未完了および未確定の UOR の影響を受けたすべての登録済みデータベースを、それらのバックアウトが完了するまで、他のアプリケーション・プログラムからアクセスされないように保護します。これは、バックアウトがコールド・スタート後まで完了しない場合であっても変わりありません。ユーザーは、未登録のあらゆるデータベースについて、それらのバックアウトが完了するまではプログラムが決してそれらにアクセスしないようにする必要があります。

/ERESTART COLDSYS コマンドを出す前に、COLDSTART または ACTIVE 制御ステートメントを使用して、バッチ・バックアウトを少なくとも 1 回は実行する必要があります。これらの制御ステートメントは、バックアウトを必要とする登録済みデータベースをバックアウトが完了するまで誤ったアクセスから保護するのに必要な情報を DBRC に提供します。

DB/DC または DBCTL サブシステムでのオンライン・バックアウトのいずれかにおいて、IMS がバックアウトを据え置く原因となる障害が起こった場合は、次のようにしてください。

1. 元の問題を解決する。
2. /START DB コマンドまたは UPDATE DB START(ACCESS) コマンドを出す。
3. 部分的 (再始動可能) バックアウトを実行して当該のバックアウトを完了させる。

場合によっては、必要な情報が保管されていなかったために再始動可能バックアウトも失敗する可能性があります。このようなケースでは、バッチ・バックアウト・ユーティリティーを使用する必要があります。また、IMS がバックアウト障害のいずれかをコールド・スタート後まで据え置いた場合も、このユーティリティーを使用する必要があります。

バックアウト中のシステム障害

バックアウト中にシステム障害が起こった場合は、バッチ・バックアウト・ユーティリティーをもう一度実行してください。入力ログおよび正常に実行されたバックアウトからの出力ログを、データベース変更累積ユーティリティーまたはデータベース・リカバリー・ユーティリティーへの入力として保管してください。

バックアウト中の DL/I 入出力エラー

IMS は、DL/I データベースの入出力エラーを、動的バックアウト、バッチ・バックアウト、および緊急時再始動バックアウトの場合と同じ方法で処理します。

動的バックアウト中のエラー

バックアウト中にデータベース入出力エラーが起こった場合は、IMS はメッセージを出し、EEQE を作成してそのイベントを記録し、さらに DBRC を呼び出して、RECON データ・セット内の該当するデータベース・エントリーにバックアウトが必要であることを示すフラグを立てます。

データベース・バックアウト・ユーティリティーを実行し、該当するデータベース・エントリーをバックアウトして、DBRC に該当する「要バックアウト」フラグおよびカウンターをリセットするように通知する必要があります。DBRC は、データベース・バックアウト・ユーティリティーが実行されるまで、フラグを立てられたデータベースの使用を許可しません。

書き込みエラーの場合は、IMS は、エラーが発生しているブロックまたは制御インターバルのバッファー内容を EEQE によってポイントされている仮想バッファーにコピーします。これに加え、IMS は、データベースがクローズされるときに未解決エラーがある各バッファーの書き込みを試行します。IMS は、IMS が書き込みまたは読み取りを正常に行えるそれぞれのエラー・バッファーごとにメッセージ DFS0614I を出し (EEQE が読み取りエラー用の場合)、すべての未解決エラーが解決されたときにメッセージ DFS0615I を出します。

動的バックアウト中にデータベース読み取りエラーが起こった場合は、IMS は、メッセージ DFS983I を出し、エラーがあるデータベースだけを停止させて、PSB 内で他のデータベースの動的バックアウトを継続します。IMS は、正常にバックアウトした各データベースについては X'4C01' ログ・レコードを書き込み、読み取りエラーのために停止させた各データベースについては X'4C80' ログ・レコードを書き込みます。その後、IMS は、停止状態の任意のデータベースをリカバリーしてそれをオフラインでバックアウトすることを許可します。

動的バックアウトが失敗した場合は、バッチ・バックアウトを実行してデータベースを再構成する必要があります。

関連概念:

- IMS 障害リカバリー (オペレーションおよびオートメーション)

動的バックアウト中のエラーからのリカバリー

障害の種類が異なれば、必要なリカバリー手順も異なります。

具体的には、

- あるデータベースが動的バックアウト時または緊急時再始動時には使用不能であったが、その後使用可能になった場合 (例 : そのデータベースをオンラインに戻したか、または物理装置を修理した場合など) は、/START DB または UPDATE DB START(ACCESS) コマンドを使用して、(現在は使用可能となっている) データベースをバックアウトするために動的バックアウトのスケジュールを変更する。

- そのデータベースが (例 : リカバリーする必要のあるエラーがあるなどして) 使用不能なままである場合は、そのエラーのあるデータベースに対するさらなる処理を許可する前にデータベース・リカバリー・ユーティリティーを実行してから、バッチ・バックアウト・ユーティリティーを実行する必要がある。バッチ・バックアウトは X'4C01' ログ・レコードを認識して、正常にバックアウトされたデータベースに対する変更のバックアウトは試行しません。
- オンライン環境では、動的バックアウトは、ログ・レコードが使用不能である場合でも失敗する可能性がある。使用可能な唯一のログ・レコードが SLDS からのものである場合、動的バックアウトのパフォーマンスは低くなります。低速動的バックアウトは、以下の場合に発生する可能性があります。
  - 該当のログ・レコードを含んでいる OLDS がアーカイブ済みで、すでに再使用されている場合。

    推奨事項: 必ず、十分な OLDS スペースが定義されていることを確認し、さらにバックアウトに必要なレコードを確保できるだけの高い頻度でプログラムが (GU 呼び出しまたは CHKP 呼び出しを使用して) 同期点に達するようにしてください。
  - 単一ロギングまたは重複ロギングが有効であるときにログでリカバリー不能読み取りエラーが起こった場合、IMS は動的バックアウト中にデータベースを停止させる。

    このケースでは、有効なログを入手するためにログ・リカバリー・ユーティリティーを実行する必要があります。この出力ログは正常に終了しなければなりません。このログをデータベース変更累積ユーティリティーまたはデータベース・リカバリー・ユーティリティーへの入力として使用してください。入力エラーの訂正を済ませたら、バッチ・バックアウト・ユーティリティーをもう一度実行してデータベースを訂正してください。

バッチ・バックアウト中のエラー

バッチ・バックアウト中に入出力エラーが起こった場合は、バッチ・バックアウト・ユーティリティーは、読み取りエラーの影響を受けたデータベースを除くすべてのデータベースのバックアウトを完了します。

IMS はこれらの入出力エラーを、動的バックアウト中に起こった入出力エラーを処理するのと同じ方法で処理します。

バッチ・バックアウト・ユーティリティーは、更新を行う前に必ず、あらゆる未解決入出力エラーのためのデータベース・バッファーを作成します。このユーティリティーは、更新をこれらのバッファーに適用し、データベースがクローズされるまでは更新を書き込もうとはしません。

バックアウト済みのデータベースに未解決のままの入出力エラーが 1 つでもある場合は、最終的には順方向リカバリーを実行する必要がありますが、バッチ・バックアウト・ユーティリティーが正常に完了したのであれば (IMS がメッセージ DFS395I を出します)、これを再実行する必要はありません。

バッチ・バックアウト中のログでのエラー

入出力エラーが入力ログで起こった場合は、そのエラーを訂正するためにログ・リカバリー・ユーティリティーを実行する必要があります。

この出力ログは正常に終了しなければなりません。このログをデータベース変更累積ユーティリティーまたはデータベース・リカバリー・ユーティリティーへの入力として保持してください。この入力エラーの訂正を済ませたら、バッチ・バックアウト・ユーティリティーをもう一度実行してください。

入出力エラーが出力ログで起こった場合は、その出力ログを正しく終了させてください。次に、再度バッチ・バックアウト・ユーティリティーを実行します。両方の出力ログをデータベース変更累積ユーティリティーまたはデータベース・リカバリー・ユーティリティーへの入力として保持してください。

緊急時再始動バックアウト中のエラー

IMS は、データベースを開くために緊急時再始動時に読み取りエラーと障害の両方を動的バックアウト時と同じ方法で処理します。

ただし、読み取りエラーの場合は、データベースが停止中であったとしても、同じデータベースに対して他の再始動バックアウトが行われる可能性があります。

バックアウト中に OLDS に (重複ロギングが有効となっている場合は両方の OLDS に) リカバリー不能読み取りエラーが起こったときは、ログ・リカバリー・ユーティリティーを実行し、再度 IMS の再始動を試みてください。

再始動処理中に、ただしバックアウトを開始する前に、IMS は、WADS を使用して OLDS をクローズする必要があるかどうかを判別します。リカバリー不能読み取りエラーが WADS に発生した場合は、コミット済みログ・レコードが損失する可能性があります。このケースでは、OLDS をクローズしてアーカイブしてから順方向リカバリーとバッチ・バックアウトを行って、影響を受けたデータベースを再構成する必要があります。

第 28 章 データベースのモニター

多くの IMS ツールを使用して、データベースのパフォーマンスをモニターすることができます。

このトピックでは説明していませんが、以下のツールもモニターの目的で使用することができます。

- IMS Performance Analyzer
- IMS DB 制御スイート (オンデマンド・スペース・モニター)
- IMS DB ツール・スペース・モニター・ユーティリティー
- DB 保全性制御機能

関連資料**:** これらの、およびその他の IMS ツールについては、www.ibm.com/ims にアクセスし、IBM DB2 および IMS ツールの Web サイトにリンクしてください。

関連概念:

777 ページの『VSAM バッファーのモニター』

37 ページの『第 2 回コード検査』

501 ページの『バッファーの数』

708 ページの『いつデータベースの再編成を行うべきか』

IMS モニター (システム管理)

IMS 環境でのデータ共用 (システム管理)

関連タスク:

788 ページの『割り振られるスペース量の変更』

IMS モニター

IMS モニターは、バッチ環境における DL/I データベースのパフォーマンスに関するデータを記録するためのツールです。

記録されたデータは、多様な報告書として作成されます。このモニターは 2 つの面で役に立ちます。まず、定期的にこのモニターを実行する場合には、ある期間にわたってパフォーマンスに関するデータが得られます。このデータを比較することにより、パフォーマンスの傾向が受け入れられるものかどうかを判断することができます。このような使い方は、データベースのチューニングに関する決定の助けとなり、また、データベースの再編成を行う必要のある時期を決める助けとなります。

このモニターのもう 1 つの用途は、変更内容がどのようにパフォーマンスに影響を与えるかを評価することです。正常なデータベース処理について記述している報告書がいくつか集まったら、これ以後、変更の影響を比較するためのプロファイルとしてこれらの報告書を使用できます。変更するのは (その後パフォーマンスをテストするのは)、次のような場合です。

- データベースの構造の変更
- ある DL/I アクセス方式から別のアクセス方式への変更
- データベース・バッファー・プールの数やサイズの変更
- アプリケーション・プログラムの論理の変更

これらのどの場合にも、変更を行う主な目標は、おそらく操作に必要となる入出力の数を最小限に抑えようとすることです。このモニターは、目的が達成できたかどか判別するのに役立ちます。

次の例は、IMS モニターをどのように使用するかを示しています。新規または改訂アプリケーションの最終テストを行っているものとします。このモニターからの報告書では、このプログラムの中の一部の DL/I 呼び出しは、単一の入出力検索を必要とすると予測されていたのに、実際には、多くの入出力が必要となる大掛かりなデータベース・スキャンを必要としていたことが示されます。アプリケーション・プログラムの論理に変更を加えることにより、この問題を解消できる場合があります。

以下の図に示してあるように、このモニター自体は実際には 2 つのプログラムです。

- IMS モニター (DFSMNTR0)
- IMS モニター報告書印刷ユーティリティー (DFSUTR20)

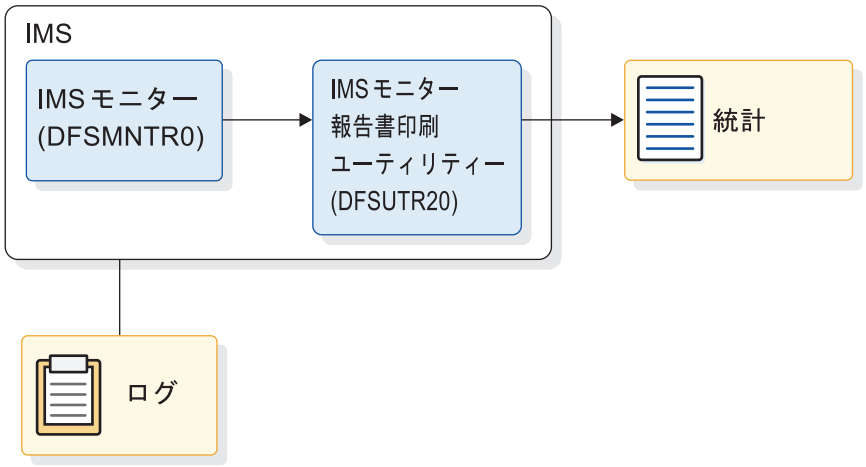


図 *252. IMS* モニターの動作

IMS モニターは、(DL/I の稼働中に) IMS 制御ブロックからデータを収集し、このデータを独立データ・セットまたは IMS ログに記録します。このモニター・プログラムは、少なくともシステムには干渉しないようにしながら、データを収集します。モニターは IMS ジョブと同じアドレス・スペースで実行され、実行 JCL 内の MON= パラメーターでオンまたはオフに切り替えることができます。

IMS モニター報告書印刷ユーティリティーは、IMS モニターが収集した情報を要約した各種報告書を作成するためのオフライン・プログラムです。このプログラムは、次の報告書を作成します。

- VSAM バッファー・プール報告書

- VSAM 統計報告書
- データベース・バッファー・プール報告書
- プログラム入出力報告書
- DL/I 呼び出し要約報告書
- 配布付録報告書
- モニター・オーバーヘッド報告書

これらの報告書の多くは、IMS モニターでも提供されます。

IMS モニターがオンになっている場合、バッチ実行が終了するまでオンのままになり、その間はオーバーヘッドが必要です。このモニターは、システム・コンソールからオン/オフに切り替えることはできません。モニターによる影響を最小限にするには、マルチスレッド・アプリケーション環境よりも、単一スレッド・テスト環境で IMS モニターを使用してください。

こうすると、IMS モニターが収集したデータを特定プログラムと関連付けることができます。

関連概念:

DB モニター報告書 (システム管理)

IMS モニター報告書 (システム管理)

高速機能システムのモニター

メッセージ・ドリブンの高速機能アプリケーションが組み込まれている IMS オンライン・システムをモニターする際に強調すべきことは、迅速な応答と高いトランザクション率とのバランスです。

高速機能では、パフォーマンス・データがシステム・ログ情報の一部になります。急送メッセージ処理によって大量のオンライン・トラフィックが処理され、かつこれらはメッセージ・キューに存在していないものと予測されているので、高速機能ログ分析ユーティリティー (DBFULTA0) は、高速機能アプリケーションのモニターにとって基本ツールです。 IMS モニターは、高速機能システムをモニターするのにも使用できます。

高速機能ログ分析ユーティリティー (DBFULTA0) を使用して、IMS システム・ログに記録されているデータを基にして高速機能の統計報告書を作成してください。このユーティリティーはオフライン・ユーティリティーであり、システムのインストール、チューニング、障害追及に役立つ次の 5 つの報告書を作成します。

- 例外トランザクションの詳細リスト
- MPP (メッセージ処理プログラム) 領域に対するトランザクション・コード別の例外の詳細の要約
- MPP 領域に対するトランザクション・コード別要約
- PSB 名またはトランザクション・コード別の IFP、BMP、および CCTL トランザクションの要約
- ログ分析の要約

このユーティリティーを、IMS モニターまたは IMS ログ・トランザクション分析ユーティリティーと混同しないでください。

ユーザーは、高速機能環境の中の管理担当者として、モニター・ストラテジー、パフォーマンス・プロファイル、および分析プロシージャーの設定といった作業を行う必要があります。このトピックでは、分析ユーティリティーを使用して作業を行う方法に焦点を当て、チューニング・アクティビティーが重要になる場合を示します。

関連概念:

- IMS モニター (システム管理)
- IMS モニター報告書 (システム管理)

関連資料:

- 高速機能ログ分析ユーティリティー (DBFULTA0) (システム・ユーティリティー)
- CCTL 出口ルーチン (出口ルーチン)

高速機能ログ分析ユーティリティー

高速機能ログ分析ユーティリティーは、高速機能従属領域に渡される、高速機能の排他的トランザクションと発生する可能性のあるトランザクションの統計情報を収集します。

このユーティリティーは、他の PSB (高速機能 PCB と同期点処理に入るプログラムを含む) についての情報を報告し、次の 3 種類の出力を生成します。

- 定様式の要約と詳細報告書
- ユーティリティー実行への入力を形成するシステム・ログから抽出した高速機能トランザクションのトラフィック合計についての固定フォーマット・レコードのデータ・セット
- 例外条件に基づいて選択された、同一フォーマットの、レコードのデータ・セット (特定の固定応答時間を超えるトランザクションといったもの)

後者のデータ・セットは、ご使用のシステムにより作成されたプログラムによって、さらに詳細な分析ができます。またこれらのデータ・セットは重要なトランザクションまたはイベントのグループにソートすることができます。レコード・フォーマットとフィールドの意味については、「*IMS V13* システム・ユーティリティー」で詳しく説明しています。

高速機能ログの縮小

ログのボリュームを縮小するのに、LGNR パラメーターを使用することができ、これは DBC、FDR、および IMS の始動プロシージャーで指定します。

LGNR は、CI 全体がログに記録される前に保持される DEDB バッファー変更の最大数を示します。

ログのボリュームを縮小するもう 1 つの方法は、DEDB をリカバリー不能として指定することです。データベースへの変更がログに記録されず、データベース更新の記録が DBRC RECON データ・セットに保存されません。

関連概念:

219 ページの『非リカバリー・オプション』

DBCTL での高速機能 EXEC パラメーター (システム定義)

DCCTL または DB/DC での高速機能 EXEC パラメーター (システム定義)

関連資料:

IMS プロシージャーのパラメーターの説明 (システム定義)

高速機能トランザクションのタイミング

各高速機能トランザクションごとに、別個に計算されるシステム内の時間に 4 つの時間間隔があります。

以下の図は、境界イベントと間隔を示しています。

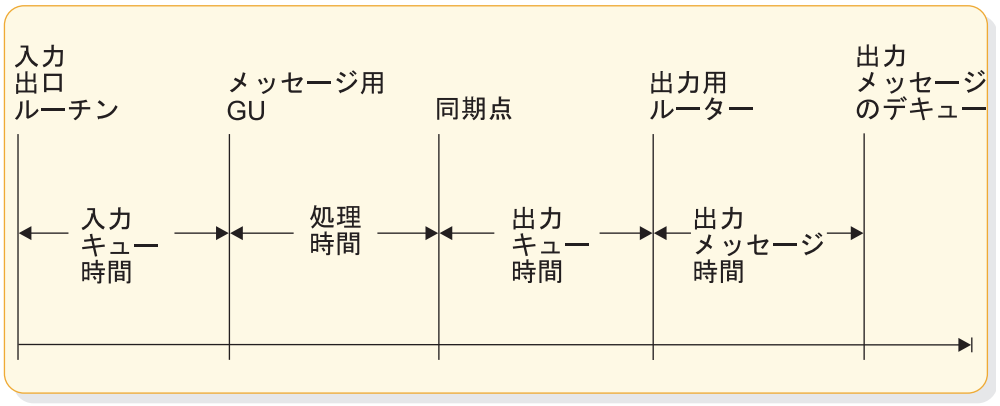


図 253. 高速機能トランザクション・イベントのタイミング

上の図の 4 つの間隔について以下に説明します。

1. 入力キュー時間: 作業を分散させるために、平衡グループ内でキューに入っているトランザクション入力キューイングを反映しています。
2. 処理時間: 個々のトランザクションに対して実際に経過した処理時間を記録します。
3. 出力キュー時間: ロギング後まで出力メッセージの解放を遅らせる際の同期点の効果を示します。
4. 出力メッセージ時間: 出力メッセージを受けるため、回線と装置の可用性を示します。トランザクションがプログラマブル・コントローラーから発生した場合には、次に入力があるまで出力が確認されないことによって生じるデキューの遅れが、出力時間に反映されることになります。

通過時間 は、最初の 3 つの間隔の合計を表す用語です。この時間は、応答時間とはわずかに異なります。メッセージの回線アクティビティー、メッセージのフォーマット設定、およびメッセージ・セグメントが出口ルーチンを出る時間までの入力編集処理を含まないからです。

高速機能に関するイベントのモニター

制御プログラムは、システム作動中は自動的に高速機能イベント・データを収集します。

以下の表は、各高速機能トランザクションごとのシステム・ログ・レコードの一部になる情報一覧表です。

表 77. 高速機能トランザクションのモニター・データ

モニター対象のデータ	メッセージ・ドリブン領域	他の領域
通過および出力メッセージ時間	x	
LTERM 名	x	
宛先コード	x	
平衡グループ・キュー・カウント	x	
DEDB 呼び出しの数	x	x
入出力から DEDB までの数	x	x
MSDB 呼び出しの数	x	x
CI 競合の数	x	x
割り振られたバッファーの数	x	x
バッファー待ちの数	x	x
同期点障害理由コード	x	x

トランザクションの選択

分析ユーティリティーを用いると、詳細に報告されるトランザクションを選択することができます。

トランザクション・コードと各トランザクションが超えることになる通過時間を最高 65.5 秒まで与えます。ユーティリティーを 1 回実行するごとに、いくつかのコードを選択できます。一定の通過時間を超える全トランザクションを要求するという方法もあります。その場合、個々の例外指定によって、一般指定が変更されます。

このような例外が発生しても、それをすべて印刷する必要はない場合には、印刷される最大数の明細レコードを与えることができます。デフォルトは個別レコード 1000 個ですが、最大数として 9999999 まで指定できます。印刷されるレコードの数を省略すると、例外レコードのデータ・セットには、選択基準に合った全トランザクションが入ります。

トランザクション報告間隔について、開始時間と終了時間も指定できます。 開始時間は、ユーティリティーの入力制御ステートメントによって指定されたクロック時間 (HH:MM:SS のフォーマット) を満たす最も早いトランザクションに対応します。終了時間は、入力制御ステートメントで指定された終了のクロック時間より前に同期点処理に入る最後のトランザクションによって設定されます。

使用できるもう 1 つの選択方法は、報告用に非メッセージ・ドリブン・トランザクションだけを選択する方法です。これは、IMS プログラムまたは BMP からの呼び出しによって MSDB または DEDB に対して発生したアクティビティーを調べる場合に使用します。

高速機能分析報告書の解釈

この分析報告書の性質は、各トランザクション・コードの起点、データベース・アクティビティー、および処理イベントを示すことにあります。しかし、最も多く報告される事項は、平均値および最大値を示します。

以下のような報告書が生成されます。

- トランザクションによる全体的な要約

  トランザクション・コードで要約されて、通過時間と入出力メッセージ長が与えられます。データベース呼び出しとバッファーの使用も組み込まれます。

- 例外の詳細

  選択されたトランザクションのために、そのトランザクションの個々のオカレンスに対して端末の起点と宛先コードが与えられます。詳細には、全体的な要約に現れるデータも含まれます。

- トランザクション・コードによる例外の詳細の要約

  これは例外報告書のトランザクションに基づく報告書です。報告される事項は、全体的な要約についての事項と同じです。

- PSB によるトランザクションの要約

  非メッセージ・ドリブン領域、MPP 領域、および同期点処理に入る BMP 領域にあるプログラムがすべて報告されます。報告される事項は、例外の詳細の要約についての事項と同じです。

- 分析の概括

  これは、トランザクション入力とオンライン・ユーティリティーについて入出力全合計を分析に加えるドキュメンテーションの手助けです。

ユーティリティーに対するシステム・ログの入力によって扱われる間隔と、入力制御ステートメントでオペレーターが定義する例外基準との組み合わせで、以上の報告書の内容が決まります。

報告書のフォーマットと報告される項目の定義の例は、「*IMS V13* システム・ユーティリティー」の高速機能ログ分析ユーティリティーの説明の中に記載されています。

報告されたイベントを解釈するにあたって以下のことを提案します。

- 要約報告書を検査し、同期点障害の理由を調査します。
- 要約報告書を検査し、バッファーの使用が一貫して NBA 値より下だったか調べます。オーバーフロー・バッファーの必要性を示すすべてのマイナスの差をチェックし、それが異常な発生だったかどうかを調べます。
- データベース呼び出し回数を予期されたプロファイルの回数と比較します。実行について異常パターンを示すトランザクションを選択し、詳細例外報告書を生成します。
- 平衡グループ・キュー・カウントを検査し、それがスケジューリング・アルゴリズムの期待値と一致するか調べます。

第 29 章 データベースのチューニング

データベースのチューニングは、パフォーマンスの向上のため、またはデータベース・スペースを有効に使用するために行います。

再編成ユーティリティーを使用して、データベースをさまざまな形でチューニングすることができます。

データベースのチューニングを行うときには、単純な変更を上回る複雑なことを行う場合が多いということに留意してください。例えば、データベースを再編成する必要があり、同時にオペレーティング・システム・アクセス方式を変更する必要があることがあります。このトピックには、それぞれの種類の変更を加えるための手順を記載しています。同時に複数の変更を行う場合は、 838 ページの『データ・セット・グループの数の変更』 のフローチャートを見てください。このフローチャートを、このトピックに示す個々の手順と併用すれば、ある種の複数の変更をデータベースに加えるときの手順がわかります。

また、一部のチューニング変更がアプリケーション・プログラムの論理に影響を及ぼすことがあるかもしれません。その影響を分析するために、変更する前にディクショナリーを使用することができます。さらに、ある種の変更を加える場合には、新しい DBD および PSB をコーディングすることが必要になります。ディクショナリーにおいて変更を初期設定しておけば、このディクショナリーの助けを借りて、新しい DBD および PSB を作成することができます。

関連概念:

501 ページの『バッファーの数』

IMS 環境でのデータ共用 (システム管理)

データベースの再編成

データベースの再編成は、パフォーマンスを向上させるためにデータベース内のデータの編成方法を変更することを意味します。

場合によっては、データベースの再編成は、データベースの構造またはデータベース内のレコードとセグメントの構造を変更することを指すこともあります。このトピックではデータの編成方法の変更に焦点をあてて説明しますが、ここで説明する再編成ユーティリティーの多くは構造の変更にも使用することができます。

DEDB と HALDB の 2 つのデータベース・タイプは、ここで説明するオフライン方式の再編成に加えて、オンライン再編成もサポートしています。

全機能データベースの場合、CI レクラメーション処理を行うと、VSAM キー順データ・セットへの DL/I 呼び出しのパフォーマンスも向上します。 CI レクラメーション処理を行うと、DL/I 呼び出し時に読み取られる空の CI の数が減少します。

関連概念:

738 ページの『HALDB オンライン再編成』

458 ページの『全機能データベースのための VSAM KSDS CI レクラメーション処理』

801 ページの『第 30 章 データベースの変更』

関連タスク:

774 ページの『編成の良いデータベースの確実化』

関連資料:

高速 DEDB 直接再編成ユーティリティー (DBFUHDR0) (データベース・ユーティリティー)

いつデータベースの再編成を行うべきか

以下の環境では、データベースを再編成すべきです。

- データベースのパフォーマンスが低下した。複数の CI またはブロックにわたって保管されるデータベース・レコードのセグメントが多くなりすぎたり、データベース内のフリー・スペースがなくなったりすると、このような必要が生じます。
- DASD への物理的 I/O が多すぎる。
- データベースの構造が変更された。例えば、HALDB 区画の境界やハイ・キーの変更後には、HALDB 区画を再編成すべきです。
- HDAM または PHDAM ランダム化ルーチンが変更された。
- HALDB 区画選択出口ルーチンが変更された。

DB モニターは、データベースをモニターするのを支援して、データベースを再編成する時期を決定する助けとなります。

関連概念:

699 ページの『第 28 章 データベースのモニター』

オフラインでのデータベースの再編成

データベースに構造的な変更を加えていない場合に、オフラインでデータベースを再編成するには、3 つの基本的なステップを実行します。

データベースをオフラインで再編成するためのステップは次のとおりです。

1. 既存のデータベースをアンロードします。
2. 古いデータベース・スペースを削除して新しいデータベース・スペースを定義します。 (こうするのが常によい方法ではありますが、これが必要であるのは、複数のエクステントまたはボリュームがある場合、または VSAM を使用している場合だけです。) VSAM の場合には、データベース・スペースとは、データベース・データ・セットのために VSAM に対して定義されたクラスターを指しています。
3. データベースを再ロードします。

関連概念:

801 ページの『第 30 章 データベースの変更』

オフライン再編成中のデータベースの保護

データベースをオフラインで再編成するときには、そのデータベースを削除します。したがって、システム障害の場合、または、再編成時の障害に備えてこのデータベースを保護する必要があります。

既存のデータベースを保護するには、このデータベースが占めているスペースを名前変更して、新しいデータベース・スペースを定義すればよいわけです。 データベースの再ロードを行ったら、そのデータベースに対してアプリケーション・プログラムを実行する前に、ただちにこのデータベースのイメージ・コピーをとるようにします。イメージ・コピーをとっておくとデータベースのバックアップ・コピーができ、システム障害発生時の DBRC によるリカバリーのポイントが確立されます。データベースのイメージ・コピーを作成するには、データベース・イメージ・コピー・ユーティリティーまたはデータベース・イメージ・コピー 2 ユーティリティーを使用することができます。これらのユーティリティーの詳細は、「*IMS V13* データベース・ユーティリティー」に記載されています。

再編成ユーティリティー

データベースは、IMS ユーティリティーを使用して再編成することができます。このトピックでは、これらのユーティリティーを紹介し、これらを併用した場合どのような働きをするかを説明します。

HALDB では、以下の再編成ユーティリティーを使用することができます。

- HD 再編成アンロード・ユーティリティー (DFSURGU0)
- HD 再編成再ロード・ユーティリティー (DFSURGL0)
- データベース事前再編成ユーティリティー (DFSURPR0)
- HALDB 区画データ・セット初期設定ユーティリティー (DFSUPNT0)

HALDB では、データベース事前再編成ユーティリティーおよび HALDB 区画データ・セット初期設定ユーティリティー (DFSUPNT0) はどちらも区画を初期設定します。

HDAM または HIDAM データベースから HALDB へのマイグレーションを行う場合、DFSURCDS データ・セットでスキャン・ユーティリティー、接頭部解決ユーティリティー、接頭部更新ユーティリティーなどの全機能データベース・ユーティリティーを使用不可化すれば、事前再編成ユーティリティーで既存の JCL を一部再利用することができます。データベースのマイグレーションの後、無条件で特定区画を初期設定するなどの追加機能をもつ HALDB 区画データ・セット初期設定ユーティリティーを使用することができます。

これらのユーティリティーは、HSAM、SHSAM、または GSAM のデータベースを再編成するためには使用できません。これらのデータベースを再編成するには、古いデータベースを読み取り、次いで新しいデータベースを作成するためのプログラムを書かなければなりません。

データベースを再編成するためにはこれらの再編成ユーティリティーを必ず使用しなければならないというわけではありません。ユーザーがデータをアンロードしたり再ロードしたりするための独自のプログラムを書くことができます。ユーザーが

データベースに対して、上記のユーティリティーを用いたのでは実現できないような構造上の変更を行おうとする場合にのみ、ユーザーは自分自身のプログラムを作成する必要があります。

再編成ユーティリティーのうちのいくつかは、データベースの初期ロード時に使用できます。これらのユーティリティーは、データベースをロードするために使用されるのではなく、セグメントの接頭部に必要とされるポインター情報を収集したりソートしたりするために使用されます。このため、ユーティリティーに関する記述をよく読むと、「初期ロードまたは再編成のために使用されます」と説明してあることがあります。

予定している再編成の種類に応じて、これらの再編成ユーティリティーは次の 3 つのグループに分類することができます。

- 部分再編成
- UCF による再編成
- UCF によらない再編成

関連概念:

632 ページの『論理関係または副次索引を持つデータベースのロード』

801 ページの『第 30 章 データベースの変更』

関連資料:

再編成および変換ユーティリティー (データベース・ユーティリティー)

部分的なオフライン再編成

HD データベースを再編成しようとしている場合には、データベース全体を再編成するのではなく、その一部分を再編成することができます。

ある 1 つのデータベースの全体ではなく、その一部分を再編成する必要が生じる理由は次の 2 つです。

- その一部分だけを再編成する必要がある。
- その一部分だけを再編成することにより、全体の再編成に必要とされる時間をより短い複数の時間に分割できる。

部分再編成を行うのに使用するユーティリティーは、次の 2 つです。

- データベース調査ユーティリティー。これは、データベースのどの部分を再編成する必要があるか決める助けとなります。
- データベース部分再編成ユーティリティー。実際の再編成を行います。

HALDB 区画は、部分的なオフライン再編成をサポートしていません。

UCF を使用するオフライン再編成

再編成は、ユーティリティー制御機能 (UCF) と呼ばれる単一のプログラムを用いても、あるいはさまざまな組み合わせのユーティリティーを用いても実行することができます。

UCF を使用する場合には、これが制御機能の働きをして、さまざまな再編成ユーティリティーのうちのいずれを実行する必要があるか決定し、次いで、このユーティリティーの実行が行われるようにします。 UCF を使用するのは、これを使用する

ことにより作成しなければならない JCL ステートメントの数が少なくなり、しかもさまざまなユーティリティーを実行のために順序付ける必要がなくなるからです。また、オペレーターが行わなければならない決定の数も少なくなります。

再編成ユーティリティーを使用するオフライン再編成

ユーティリティー制御機能 (UCF) を使用しない場合は、いくつかのユーティリティーを組み合わせてデータベースの再編成を行います。どのユーティリティーを使用する必要があるか、また、どれだけ多くのユーティリティーを使用する必要があるかは、データベース・タイプ、およびデータベースが論理関係や副次索引を使用しているかどうかに応じて決まります。

データベースが論理関係または副次索引を使用していない場合には、適切なアンロード・ユーティリティーと再ロード・ユーティリティーを実行するだけですみます。これらのユーティリティーは以下のとおりです。

- HISAM データベースの場合、HISAM 再編成アンロード・ユーティリティーと HISAM 再編成再ロード・ユーティリティー
- HIDAM 索引データベースの場合 (HIDAM データベースとは別にこれを再編成する場合)、HISAM 再編成アンロード・ユーティリティーと HISAM 再編成再ロード・ユーティリティー
- SHISAM、HDAM、および HIDAM データベースの場合、HD 再編成アンロード・ユーティリティーと HD 再編成再ロード・ユーティリティー

データベースが論理関係または副次索引を使用している場合には、(これが HISAM データベースであっても) HD 再編成アンロード・ユーティリティーおよび HD 再編成再ロード・ユーティリティーを実行する必要があります。さらに、他のさまざまなユーティリティーを実行して、セグメントの接頭部のポインター情報の収集、ソート、または復元を行う必要があります。データベースを再編成すると、セグメントの位置が変わるということを忘れないでください。論理関係または副次索引を使用している場合には、新しいセグメント位置を反映するように接頭部を更新しなければなりません。セグメントの接頭部の更新に関係するユーティリティーは以下のとおりです。

- データベース事前再編成ユーティリティー
- データベース・スキャン・ユーティリティー
- データベース接頭部解決ユーティリティー
- データベース接頭部更新ユーティリティー

これらのユーティリティーは、データベースの初期ロード時に接頭部情報を解決するために使用できます。

このセクションでのユーティリティーに関する説明では、まず、4 つのアンロード・ユーティリティーおよび再ロード・ユーティリティーについて検討します。そのあと、接頭部情報を解決するために用いられる 4 つのユーティリティーについて説明します。ユーティリティーに関する記述を最初に読み進んでいくときには、次のことを理解しておく必要があります。すなわち、論理関係または副次索引が存在する場合 (したがって、後の 4 つのユーティリティーを使用する必要がある場合)、操作が行われる順序は以下のとおりです。

1. アンロード

2. 他の接頭部情報の収集
3. 再ロード
4. 他の接頭部情報の収集
5. 接頭部の更新

この図から、次のことに気付くでしょう。すなわち、副次索引または論理関係が存在する場合には、例えば、HD 再編成再ロード・ユーティリティーはデータベースに再ロードするだけではないということです。このユーティリティーは、すでに収集されている一部の接頭部情報が入っているデータ・セットを入力の 1 つとして使用して、データベースへ再ロードします。次に、このユーティリティーは、再ロードからの出力として、より多くの接頭部情報を含んでいるデータ・セットを作成します。さまざまなユーティリティーが処理を行うごとに発生することは、これらのユーティリティーが、先に実行されたユーティリティーで作成されたデータ・セットを用いて、後に実行されるユーティリティーによって使用されることになるデータ・セットを作成することです。ユーティリティーに関する記述を読み進んでいくとき、入力データ・セット名と出力データ・セット名に注目すれば、何が行われているか理解する助けとなります。

以下の図は、論理関係または副次索引が存在する場合のユーティリティーの実行順序を示したものです。

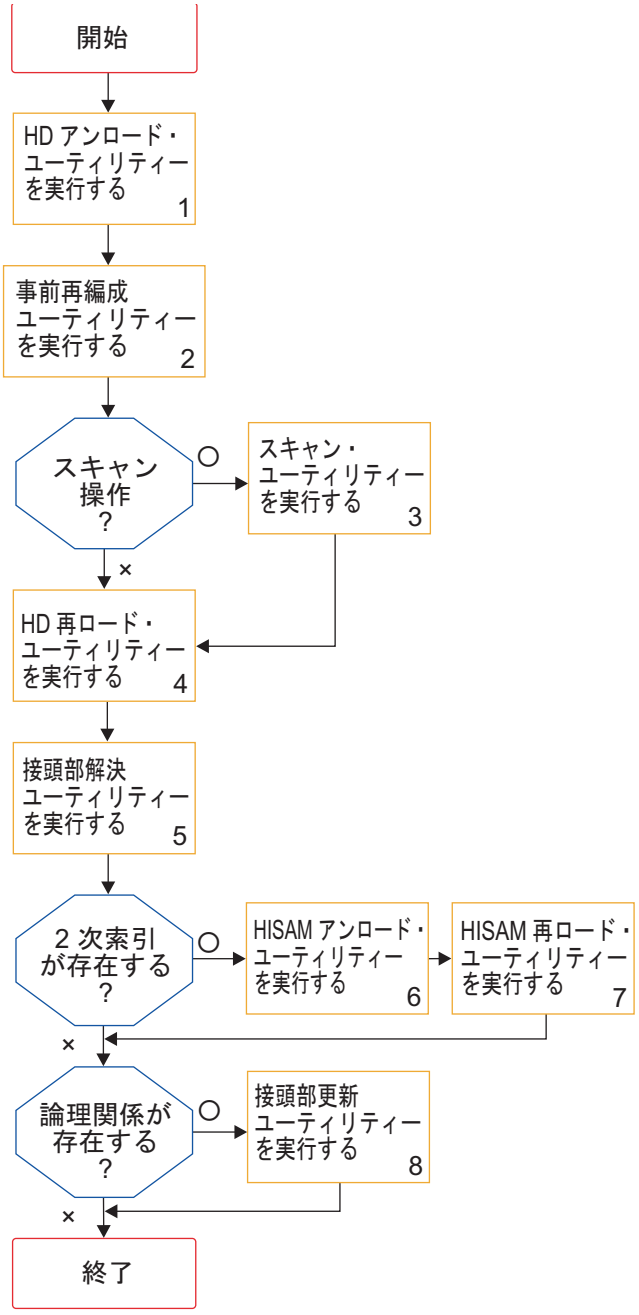


図 *254.* 論理関係または副次索引が存在する場合の再編成のステップ

以下の図は、HALDB 区画を使用した場合の、これらのユーティリティーの順序を示しています。

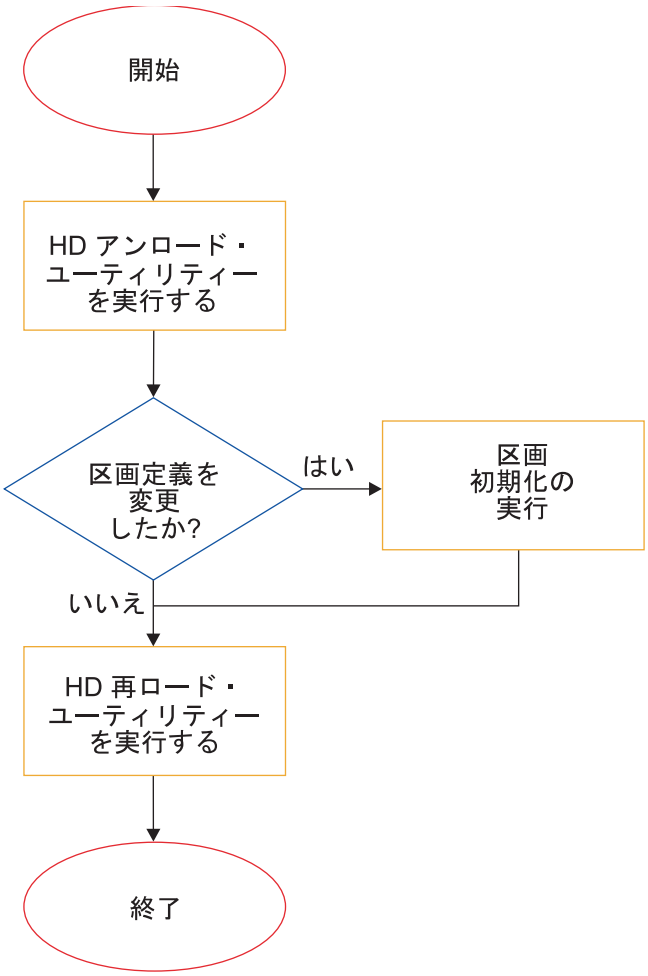


図 *255.* 論理関係または副次索引が存在する場合の *HALDB* 区画の再編成ステップ

代わりの方法として、上の図で区画初期設定ユーティリティーが必要な場合、事前再編成ユーティリティーを実行することができます。

関連タスク:

732 ページの『HALDB オフライン再編成の概説』

802 ページの『再編成ユーティリティーによるアンロードおよび再ロード』

HISAM 再編成アンロード・ユーティリティー (DFSURUL0)

HISAM データベースまたは HIDAM 索引データベースをアンロードするために、HISAM アンロード・ユーティリティーを使用します。

SHISAM データベースは、HD 再編成アンロード・ユーティリティーを用いてアンロードされます。

データベースが副次索引を使用している場合にも、(再編成処理の後の方で) 副次索引に関係する他のさまざまな操作を行うために、HISAM アンロード・ユーティリティーを使用します。

以下の図は、HISAM 再編成アンロード・ユーティリティーへの入力およびこのユーティリティーからの出力を示したものです。

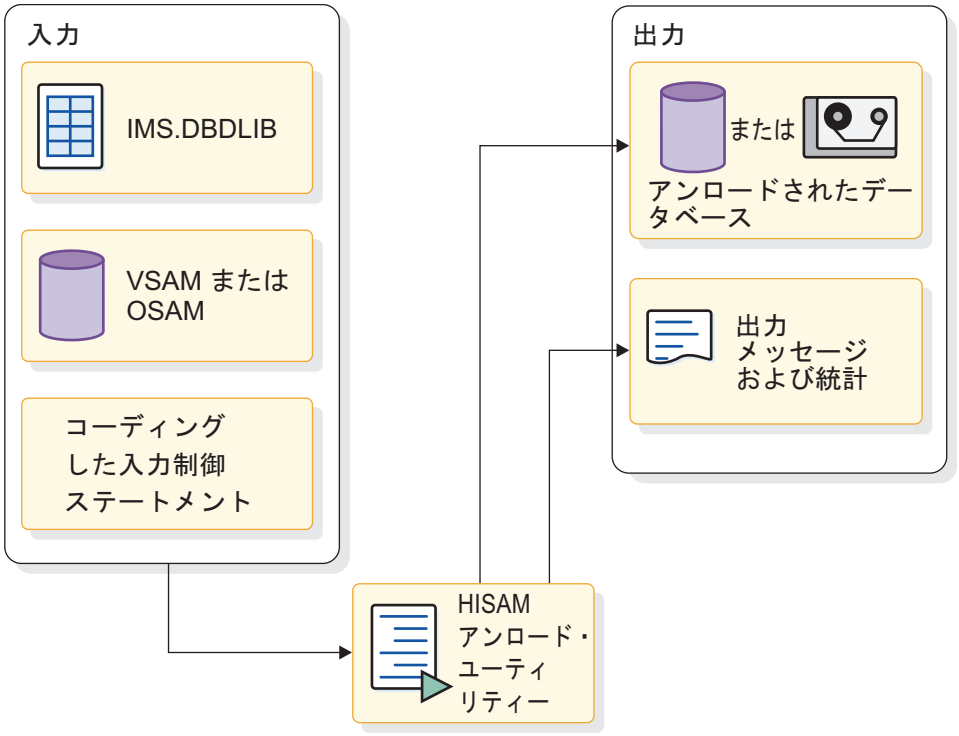


図 256. *HISAM* 再編成アンロード・ユーティリティー *(DFSURUL0)*

HISAM 再編成再ロード・ユーティリティー (DFSURRL0)

HISAM データベースを再ロードするために、HISAM 再ロード・ユーティリティーを使用します。 HIDAM データベースの 1 次索引を再ロードするためにも、HISAM 再ロード・ユーティリティーを使用します。

SHISAM データベースは、HD 再編成再ロード・ユーティリティーを用いて再ロードします。

データベースが副次索引を使用している場合には、(再編成処理の後の方で) 副次索引に関係する他のさまざまな操作を行うために、HISAM 再ロード・ユーティリティーを使用します。

以下の図は、HISAM 再編成再ロード・ユーティリティーへの入力およびこのユーティリティーからの出力を示したものです。

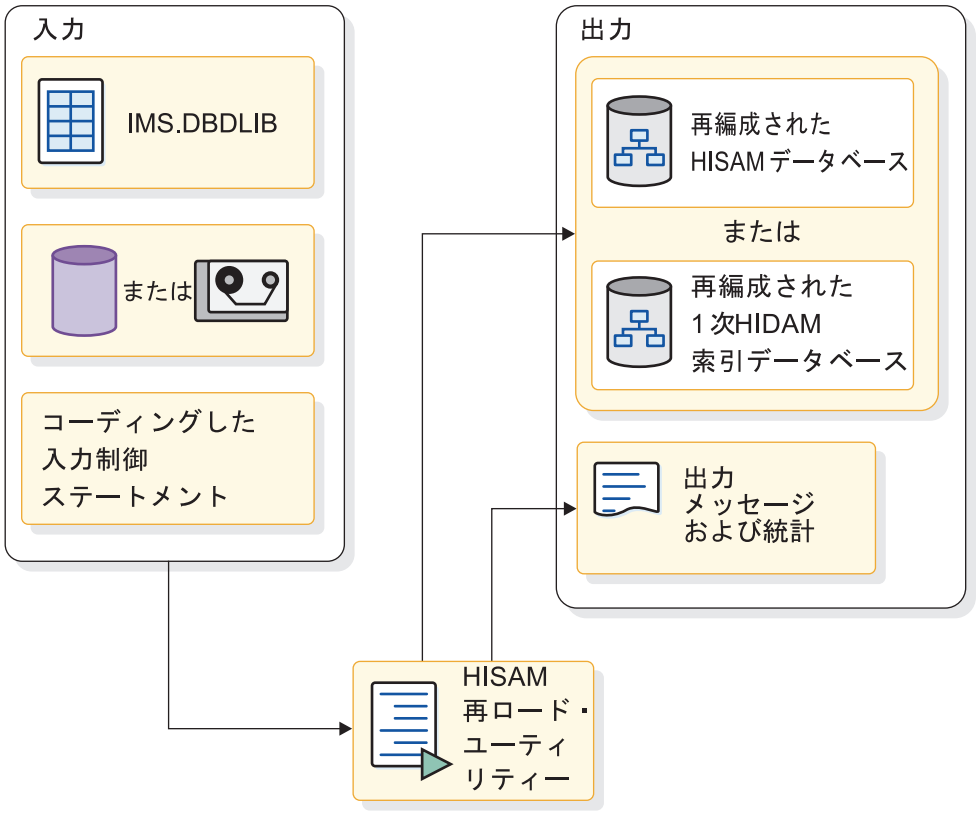


図 257. *HISAM* 再編成再ロード・ユーティリティー *(DFSURRL0)*

HD 再編成アンロード・ユーティリティー (DFSURGU0)

HD 再編成アンロード・ユーティリティーは、階層直接 (HD) データベースをアンロードする場合に使用します。

HD 再編成アンロード・ユーティリティーは、次のタイプのデータベースをアンロードする場合に使用できます。

- HDAM、HIDAM、または SHISAM の各データベース
- 副次索引を使用している HISAM データベース
- 論理関係においてシンボリック・ポインターを使用している HISAM データベース
- セグメント/編集・圧縮付きの HISAM データベースに変換中のセグメント/編集・圧縮付きでない HISAM データベース
- PHDAM データベースまたは区画
- PHIDAM データベースまたは区画
- PSINDEX データベースまたは区画

以下の図は、HD 再編成アンロード・ユーティリティーのへ入力およびこのユーティリティーからの出力を示したものです。

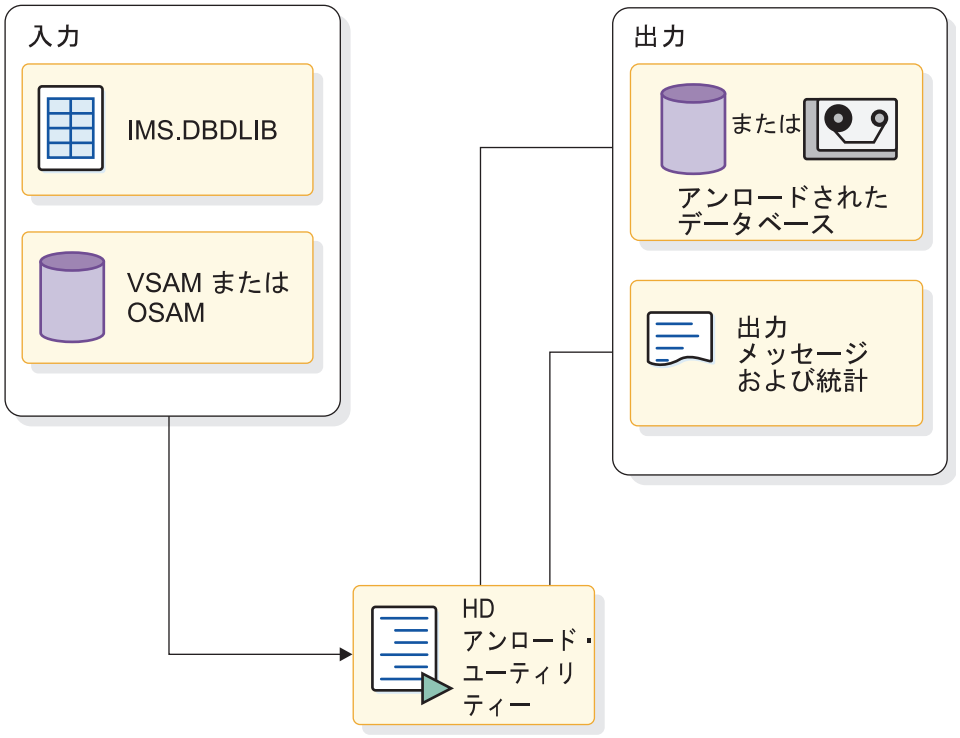


図 258. HD 再編成アンロード・ユーティリティー (DFSURGU0)

HD 再編成アンロード・ユーティリティーを使用して HALDB データベース (PHDAM、PHIDAM、または PSINDEX データベース) をアンロードする場合、データベース・データ・セットのための DD ステートメントを組み込む必要はありません。HD 再編成アンロード・ユーティリティーは、HALDB データ・セットの動的割り振りを使用します。

HD 再編成再ロード・ユーティリティー (DFSURGL0)

HD 再編成再ロード・ユーティリティーは、階層直接 (HD) データベースを再ロードする場合に使用します。

HD 再編成再ロード・ユーティリティーは、次のタイプのデータベースを再ロードする場合に使用できます。

- HDAM、HIDAM、PHDAM、PHIDAM、PSINDEX、または SHISAM の各データベース
- 論理関係または副次索引を使用している HISAM データベース
- セグメント/編集・圧縮付きの HISAM データベースに変換中のセグメント/編集・圧縮付きでない HISAM データベース

以下の図は、HD 再編成再ロード・ユーティリティーへの入力およびこのユーティリティーからの出力を示したものです。

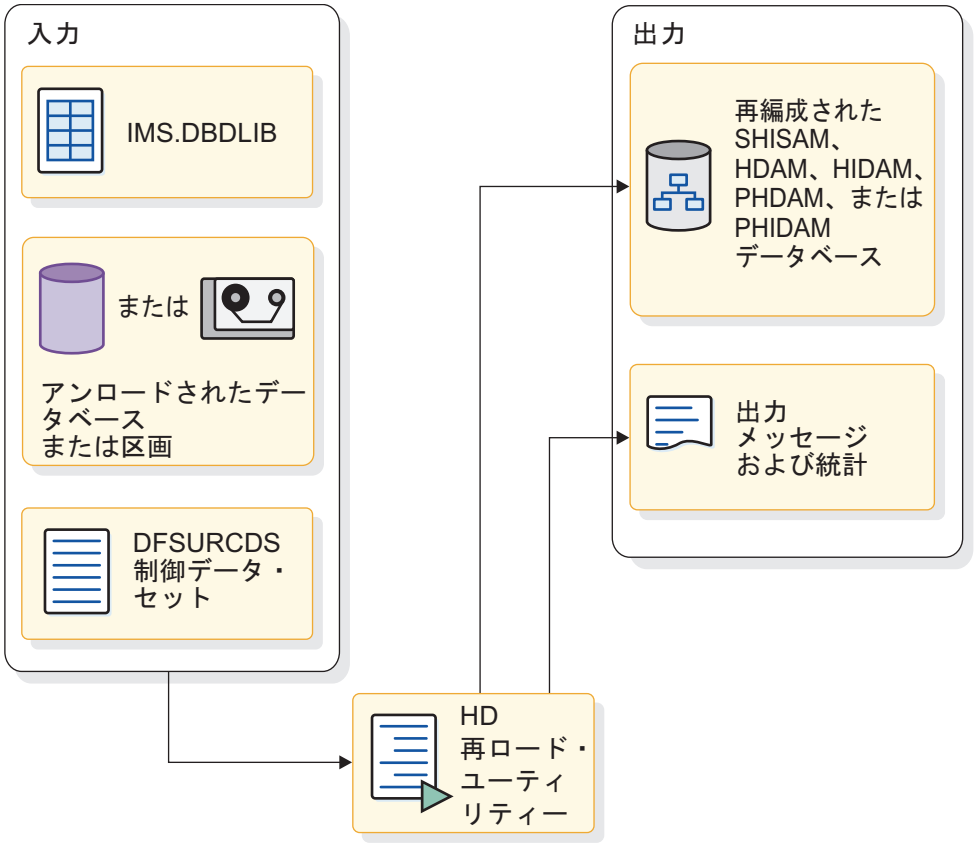


図 *259. HD* 再編成再ロード・ユーティリティー *(DFSURGL0)*

再ロードされるデータベースに論理関係または副次索引が存在する場合には、事前再編成ユーティリティーによって作成された DFSURCDS 制御データ・セットが HD 再編成再ロード・ユーティリティーへの入力の 1 つとして使用されます。DFSURCDS 制御データ・セットには、副次索引または論理関係のポインターを解決するのに必要とされる情報が入っています。

論理関係または副次索引が存在する場合、HD 再編成再ロード・ユーティリティーは、出力として DFSURWF1 作業データ・セットを作成します。 DFSURCDS は、DFSURWF1 に収集される情報を識別します。

DFSURWF1 作業データ・セットは、データベース接頭部解決ユーティリティーへの入力となります。前の図では、再ロードされるデータベースに 1 次索引がある場合に、メイン・データベースが再ロードされると自動的に索引が再ロードされる点に注意してください。なお、HIDAM 索引データベースは、HISAM アンロード・ユーティリティーおよび HISAM 再ロード・ユーティリティーを用いる別の操作として再編成することもできます。

例外: DFSURWF1 は HALDB データベースには使用されません。

データベース事前再編成ユーティリティー (DFSURPR0)

データベース事前再編成ユーティリティー (DFSURPR0) は、副次索引または論理関係を持つデータベースをロードまたは再編成する前に実行します。

以下のような場合に、データベース事前再編成ユーティリティーを使用します。

- 初期ロードを行うデータベースまたは再編成するデータベースが、副次索引または論理関係を持っている場合
- 初期ロードあるいは再編成されていないデータベースに、ロードまたは再編成されているデータベースと論理関係にかかわりのあるセグメントが含まれている場合

以下の図は、データベース事前再編成ユーティリティーへの入力およびこのユーティリティーからの出力を示したものです。

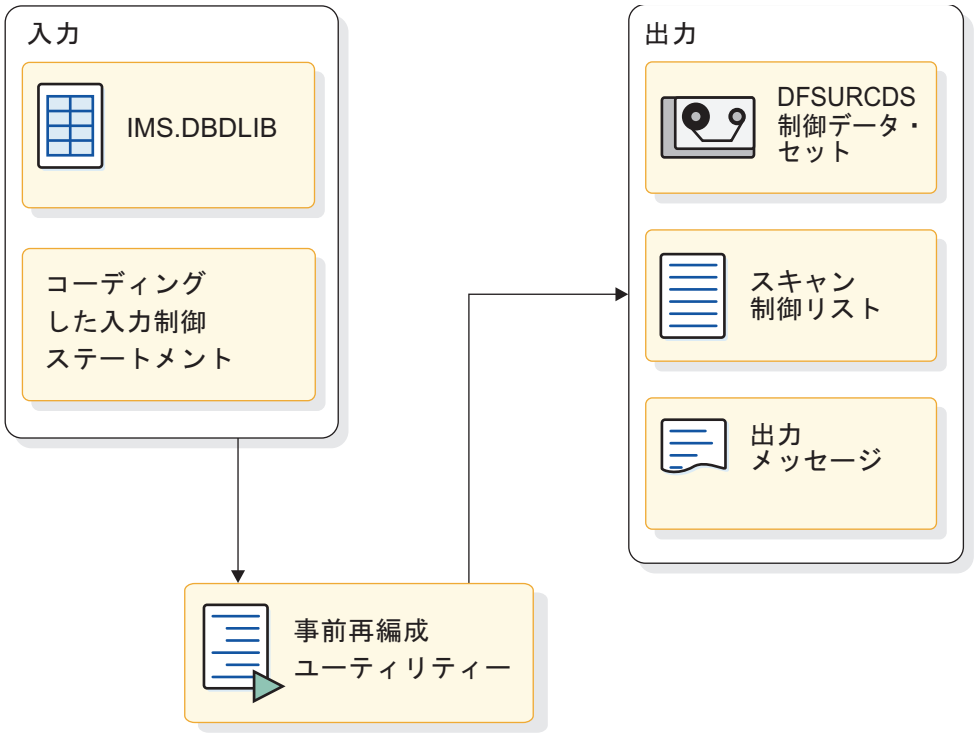


図 *260.* データベース事前再編成ユーティリティー *(DFSURPR0)*

DFSURPR0 ユーティリティーは、副次索引または論理関係が存在する場合に後で解決する必要のあるポインターについての情報が含まれている DFSURCDS 制御データ・セットを作成します。

事前再編成ユーティリティーによって作成される DFSURCDS 制御データ・セットは、以下のユーティリティーへの入力として使用されます。

- データベース・スキャン・ユーティリティー (このユーティリティーを実行する必要がある場合)
- HD 再編成再ロード・ユーティリティー (副次索引または論理関係が存在する場合)
- データベース接頭部解決ユーティリティー (データベースのロードの後または再ロードの後)

事前再編成ユーティリティーは、初期ロードまたは再編成がされていないどのデータベースに、初期ロードまたは再編成が行われているデータベースとの論理関係に関与しているセグメントが含まれているかを示すリストも作成します。

このユーティリティーが実行されるのは、常に (初期ロードの場合は) データベースがロードされる前に、あるいは (再編成の場合は) データベースの再ロードがされる前になります。

データベース・スキャン・ユーティリティー (DFSURGS0)

初期ロードまたは再編成されてはいないが、初期ロードまたは再編成されているデータベースとの論理関係に関与しているセグメントが含まれているデータベースをスキャンするために、データベース・スキャン・ユーティリティーを使用します。

以下の図は、データベース・スキャン・ユーティリティーのへ入力およびこのユーティリティーからの出力を示したものです。

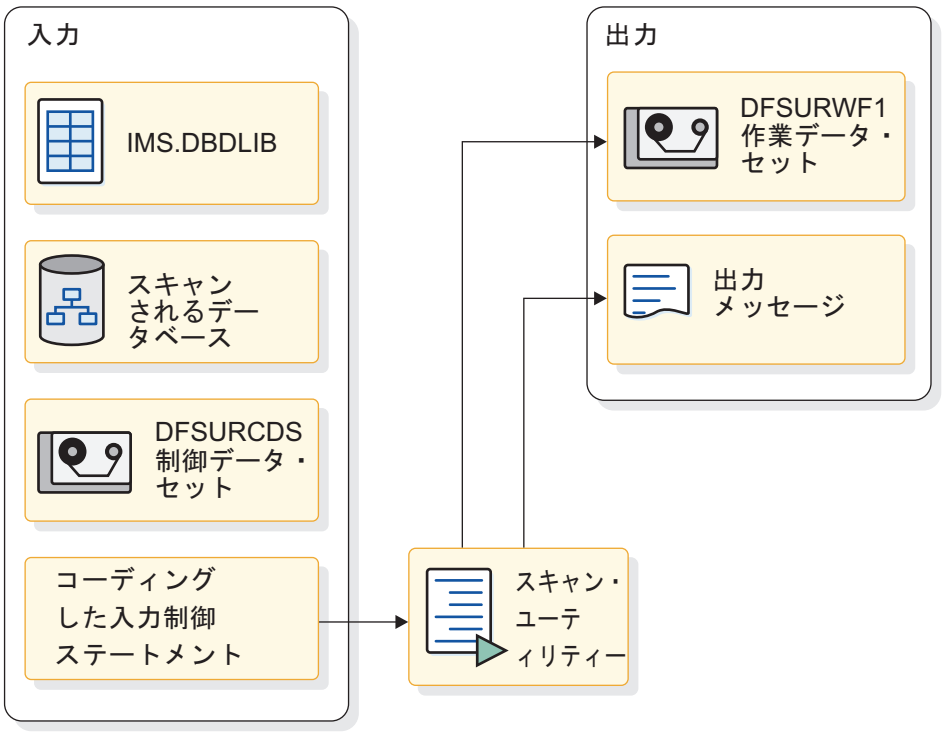


図 *261. データベース・スキャン・ユーティリティー (DFSURGS0)*

入力として、このユーティリティーは、事前再編成ユーティリティーによって作成された DFSURCDS 制御データ・セットを使用します。出力として、このユーティリティーは、論理関係を解決するのに必要な接頭部情報が入っている DFSURWF1 作業データ・セットを作成します。 DFSURWF1 作業データ・セットは、データベース接頭部解決ユーティリティーへの入力として使用されます。

このユーティリティーが実行されるのは、常に (初期ロードの場合は) データベースがロードされる前に、あるいは (再編成の場合は) データベースの再ロードがされる前になります。

データベース接頭部解決ユーティリティー (DFSURG10)

接頭部解決ユーティリティーを使用して、ロードまたは再ロードの処理で、この時点までに DFSURWF1 作業データ・セットに入れられた情報を累積し、ソートします。

以下の図は、データベース接頭部解決ユーティリティーへの入力およびこのユーティリティーからの出力を示したものです。

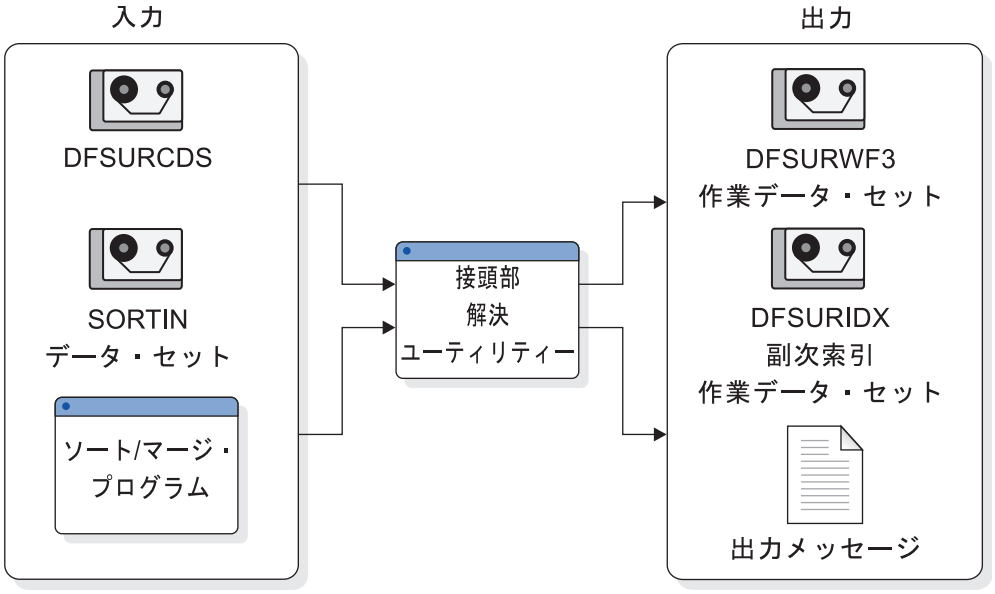


図 *262.* データベース接頭部解決ユーティリティー *(DFSURG10)*

DFSURG10 ユーティリティーへの入力となり得るさまざまな作業データ・セットには、次のようなものがあります。

- 事前再編成ユーティリティーによって作成された DFSURCDS 制御データ・セット
- スキャン・ユーティリティーによって作成された DFSURWF1 作業データ・セット
- HD 再編成再ロード・ユーティリティーによって作成された DFSURWF1 作業データ・セット

複数の DFSURWF1 作業データ・セットは、連結して、接頭部解決ユーティリティーのための 1 つの入力データ・セットを形成しなければなりません。この入力データ・セットの名前は SORTIN です。

接頭部解決ユーティリティーは、z/OS ソート・マージ・プログラムを使用して、累積された情報をソートします。出力として、このユーティリティーは DFSURWF3 作業データ・セットを作成します。この作業データ・セットには、論理関係を解決するのに必要とされるソートされた接頭部情報が入っています。この DFSURWF3 データ・セットは、データベース接頭部更新ユーティリティーへの入力となります。

副次索引が存在する場合には、このユーティリティーは、DFSURIDX 作業データ・セットを作成します。この作業データ・セットには、新しい副次索引を作成したり、共用副次索引データベースを更新したりするのに必要な情報が入っています。DFSURIDX 作業データ・セットは、HISAM アンロード・ユーティリティーへの入力として使用されます。 HISAM アンロード・ユーティリティーは、HISAM 再ロード・ユーティリティーが副次索引を作成したり共用副次索引データベースを更新したりする前に、副次索引情報をフォーマットします。

このユーティリティーは、常に (初期ロードの場合は) データベースがロードされた後に、あるいは (再編成の場合は) データベースの再ロードが行われた後に実行されます。

データベース接頭部更新ユーティリティー (DFSURGP0)

接頭部更新ユーティリティーは、データベースの初期ロードまたは再編成により影響を受けた接頭部を持つ各セグメントの接頭部を更新するために使用します。

以下の図は、データベース接頭部更新ユーティリティーへの入力およびこのユーティリティーからの出力を示したものです。

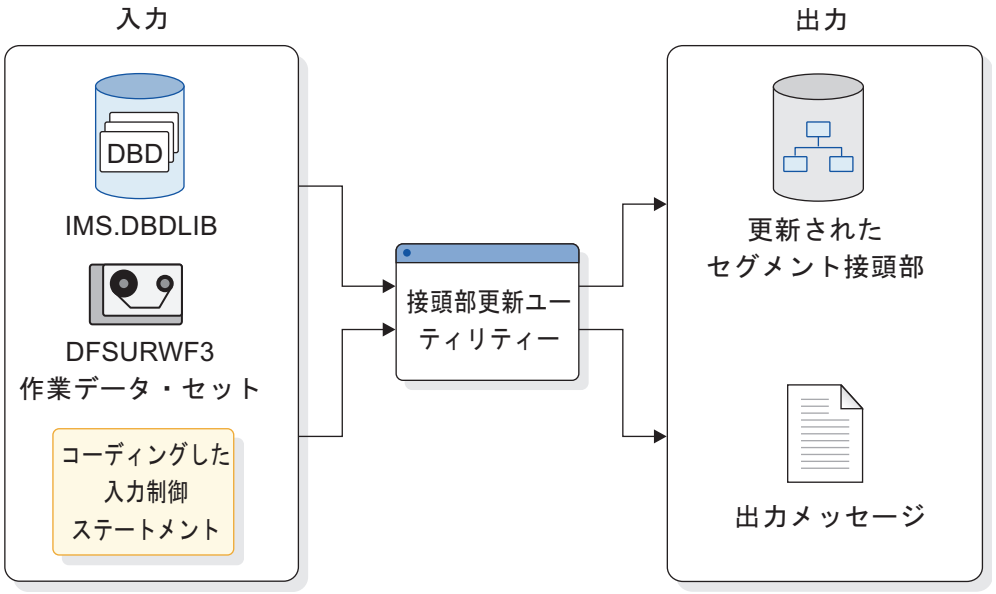


図 263. データベース接頭部更新ユーティリティー (DFSURGP0)

更新される接頭部フィールドには、論理親ポインター・フィールド、論理兄弟ポインター・フィールド、論理子ポインター・フィールド、論理親に対するカウンター・フィールドが組み込まれています。接頭部更新ユーティリティーは、入力として、接頭部解決ユーティリティーによって作成された DFSURWF3 データ・セットを使用します。

このユーティリティーは、常に (初期ロードの場合は) データベースがロードされた後に、あるいは (再編成の場合は) データベースの再ロードが行われた後に、さらに接頭部解決ユーティリティーの実行が済んだ後に実行されます。

副次索引操作のための HISAM アンロードおよび再ロード・ユーティリティーの使用

HISAM アンロードおよび再ロード・ユーティリティーでは、データベースのアンロードと再ロードに加えて、副次索引を作成し、共用副次索引に含まれている副次索引を操作することができます。

具体的には、HISAM アンロードおよび再ロード・ユーティリティーを使用して、以下の処理を実行できます。

- 副次索引データベースを作成する

- ある 1 つの副次索引を共用副次索引データベースの中にマージする
- 共用副次索引データベースの中の副次索引を置き換える
- 共用副次索引データベースの中から副次索引を抽出する

これらの操作は、それぞれ別個に行われます。すなわち、これらのうちのどの操作も、通常のデータベースのアンロードまたは再ロードを行うための HISAM アンロードと再ロード・ユーティリティーの実行と一緒に行うことはできません。

以下の図は、最初の 3 つの操作を実施する場合の HISAM アンロードおよび再ロード・ユーティリティーへの入力およびこのユーティリティーからの出力を示したものです。 HISAM アンロード・ユーティリティーへの入力として使用される DFSURIDX 作業データ・セットは、接頭部解決ユーティリティーによって作成されたものです。この作業データ・セットには、共用副次索引データベースを作成したり更新したりするのに必要な情報が入っています。 HISAM アンロード・ユーティリティーは HISAM 再ロード・ユーティリティーによる使用に備えて、副次索引情報をフォーマットします。 HISAM アンロード・ユーティリティーが通常のデータベースのアンロードのためではなく、副次索引操作のために使用される場合には、このユーティリティーへの入力制御ステートメントの 1 桁目に X を指定することに注意してください。 3 桁目には、以下のいずれかの文字を入れます。

- M - 操作が新しい副次索引データベースの作成、または共用副次索引データベースの中への副次索引のマージであることを意味しています。
- R - 操作が共用副次索引データベースの中の副次索引の置き換えであることを意味しています。

HISAM 再ロード・ユーティリティーは、HISAM アンロード・ユーティリティーからの出力を用いて、共用副次索引データベースの中に新しい副次索引を作成したり、あるいは、共用副次索引データベースの中で副次索引をマージしたりまたは置き換えたりします。

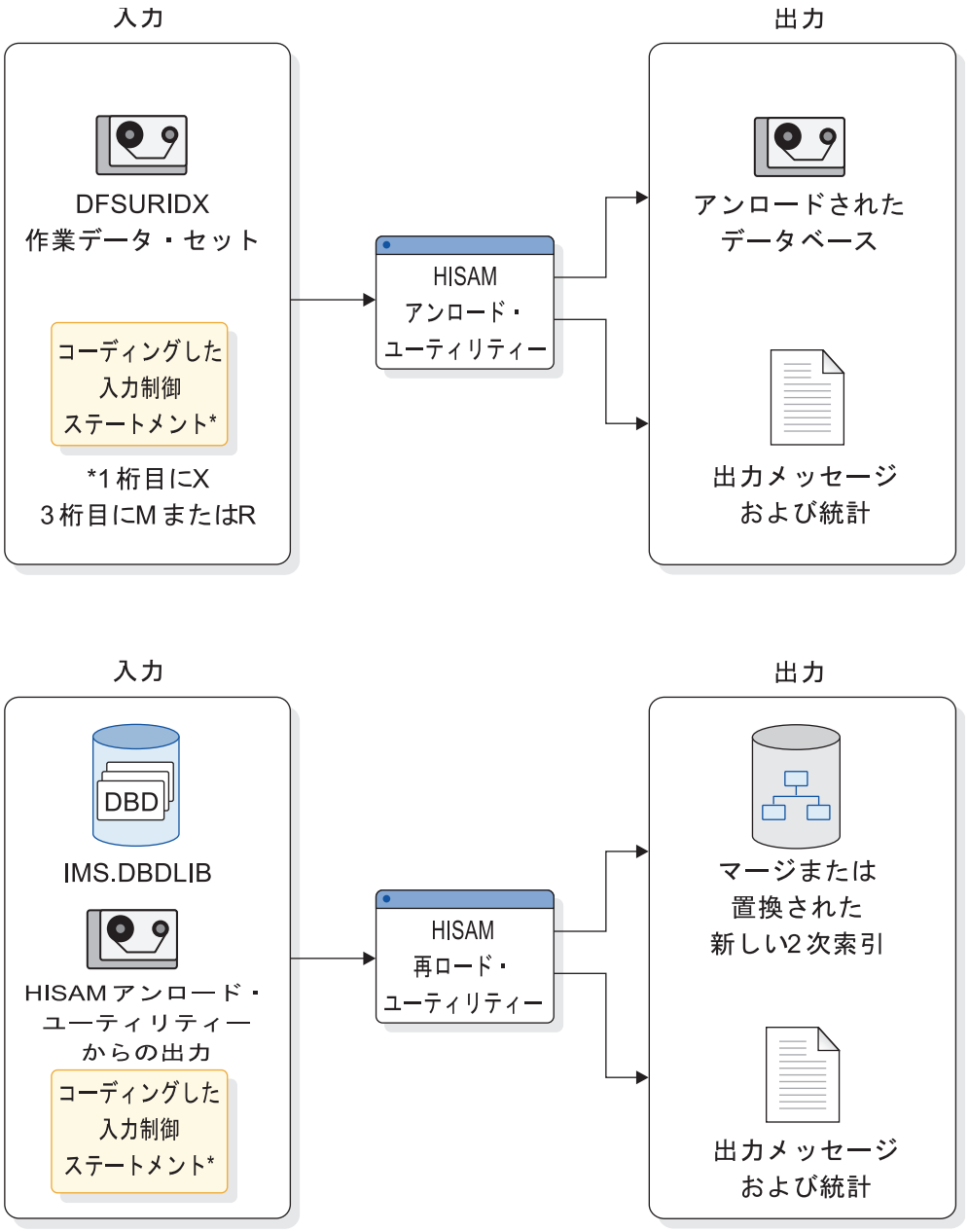


図 *264.* 副次索引の作成、マージまたは置き換え操作に使用される *HISAM* 再編成アンロードおよび再ロード・ユーティリティー

以下の図は、一連の共用索引の中から 1 つの索引を抽出する場合の HISAM アンロード・ユーティリティーの入力と出力を示したものです。 次のいずれかが入力となることができる点に注意してください。

- 接頭部解決ユーティリティーによって作成された DFSURIDX 作業データ・セット
- 共用副次索引データベース

ここでも、入力制御ステートメントの 1 桁目には X が入ります。3 桁目には E が入り、操作が副次索引の抽出であることを意味しています。

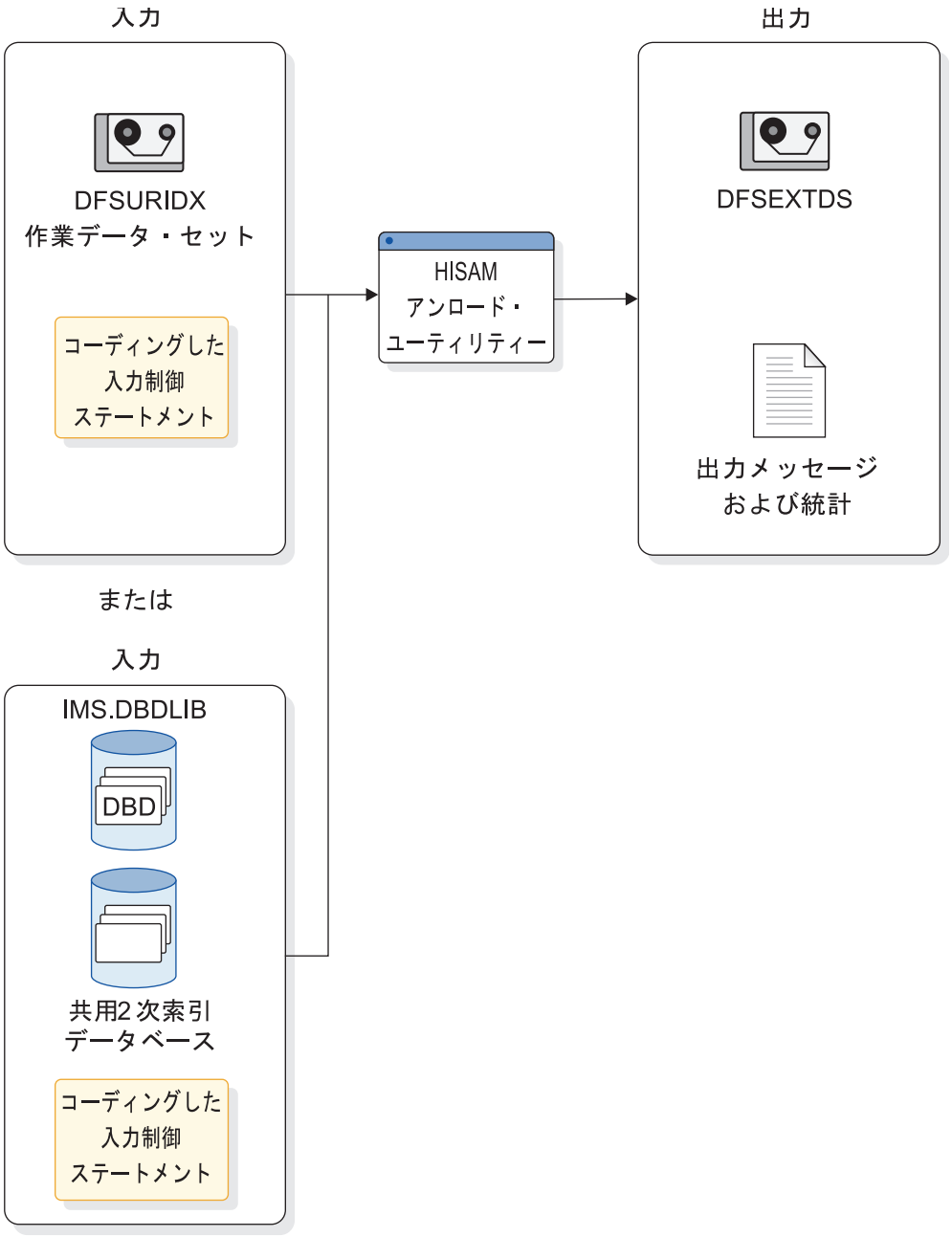


図 *265.* 副次索引の抽出操作に使用される *HISAM* 再編成アンロード・ユーティリティー

ユーティリティー制御機能 (DFSUCF00)

ユーティリティー制御機能は、再編成ユーティリティーおよびリカバリー・ユーティリティーの実行を制御するプログラムです。

ここでの制御とは、ユーザーが作成しなければならない JCL ステートメントの中の多くものを生成し、また、実行のためにさまざまなユーティリティーを順序付ける必要を除くという意味です。 UCF の制御の下で実行できない再編成ユーティリティーは、データベース調査ユーティリティーとデータベース部分再編成ユーティリ

ティーだけです。 UCF は、他のユーティリティーの実行を制御するほか、ジョブを停止した後にこのジョブを再始動することができるようにします。

データベース調査ユーティリティー (DFSPRSUR)

調査ユーティリティーは、再編成が必要であるかどうかを判別する目的で、HDAM または HIDAM データベースの全体またはその一部をスキャンするために使用します。

以下の図は、データベース調査ユーティリティーへの入力およびこのユーティリティーからの出力を示したものです。

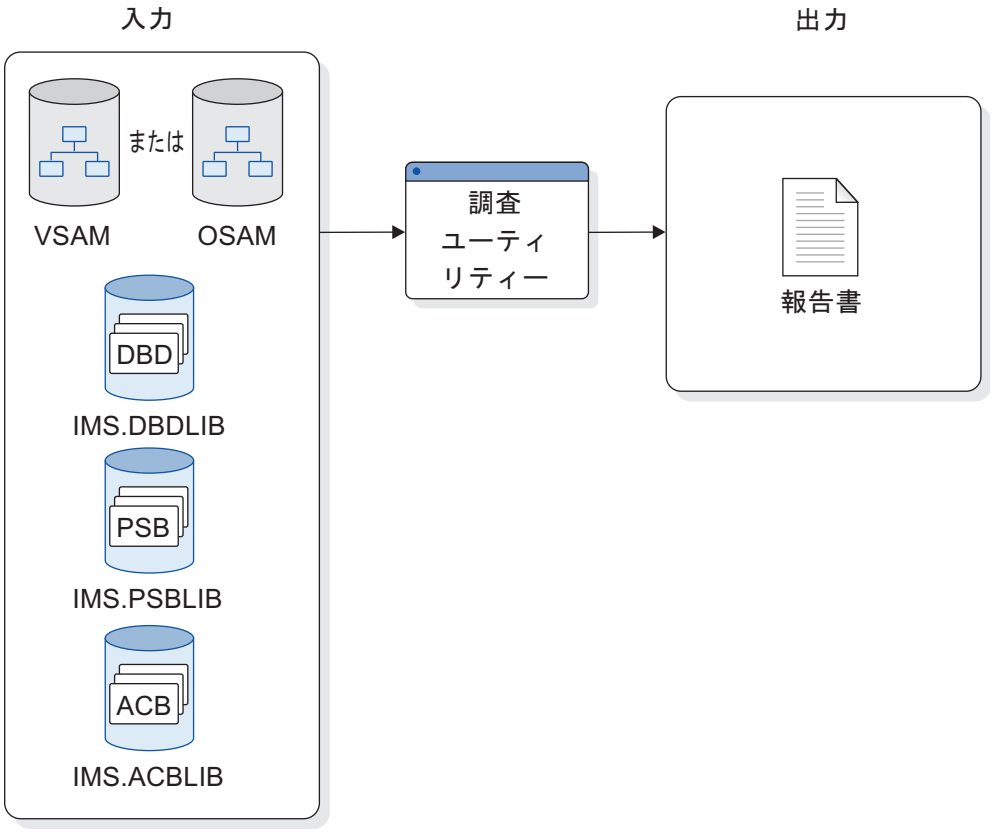


図 *266. データベース調査ユーティリティー (DFSPRSUR)*

調査ユーティリティーは、データベースの物理編成について記述する報告書を作成します。この報告書には、フリー・スペースのそれぞれの区域のサイズと位置が入っています。ユーザーが部分再編成を行う場合、ユーザーは再編成されたデータベース・レコードを入れることのできるフリー・スペースがどこにあるかを知ります。

関連概念:

『データベース部分再編成ユーティリティー (DFSPRCT1)』

データベース部分再編成ユーティリティー (DFSPRCT1)

データベース部分再編成ユーティリティーを使用して、HD データベースの各部分を再編成することができます。このユーティリティーは、HD データベースが副次索引または論理関係を使用しているときに使用できます。

ユーザーは、再編成しようとしているレコードの範囲をこのユーティリティーに伝えます。

- HDAM データベースでは、範囲は連続する相対ブロック番号を持つ一群のデータベース・レコードです。
- HIDAM データベースでは、範囲は連続するキー値を持つ一群のデータベース・レコードです。

一般に、データベース部分再編成ユーティリティーを使用する前に、データベース調査ユーティリティーを実行します。調査ユーティリティーは、再編成が必要かどうか決めて、フリー・スペースの各域の位置とサイズを知る助けとなります。再編成されたデータベース・レコードをどこに入れたらよいかを知るために、フリー・スペースの各域の位置とサイズを知る必要があります。

データベース部分再編成ユーティリティーは次の 2 つのステップに分けてデータベースを再編成します。

1. ステップ 1 では、このユーティリティーは、実際の再編成が行われるステップ 2 で使用される制御テーブルを作成します。オプションとして、このユーティリティーは、ステップ 2 で使用される PSB を作成するための PSB ソース・ステートメントを作成できます。また、このユーティリティーは、論理的に関係がある複数のデータベースの中の論理的な関係を持つ複数のセグメントのうちどれをステップ 2 でスキャンしなければならないか、またどれをステップ 2 のオプションでスキャンするとよいかを示す報告書を生成します。(GSAM データベースには、PSB が必要になるステップ 2 に関与するものがあります。)
2. ステップ 2 では、このユーティリティーは実際の再編成を行います。ユーザーが指定したデータベース・レコードが、あるデータ・セットにアンロードされます。これらのデータベース・レコードがデータベースにおいて占めていたスペースは、解放されます。次いで、これらのデータベース・レコードがデータベースの中のユーザーによって指定された範囲のフリー・スペースの中に再ロードされます。最後に、新しい場所に移されたデータベース・レコードを指すポインターがすべて、新しい場所を指すよう変更されます。ステップ 2 の終わりに報告書が作成され、これによって何が行われたか報告されます。

以下の図は、データベース部分再編成ユーティリティーへの入力およびこのユーティリティーからの出力を示したものです。

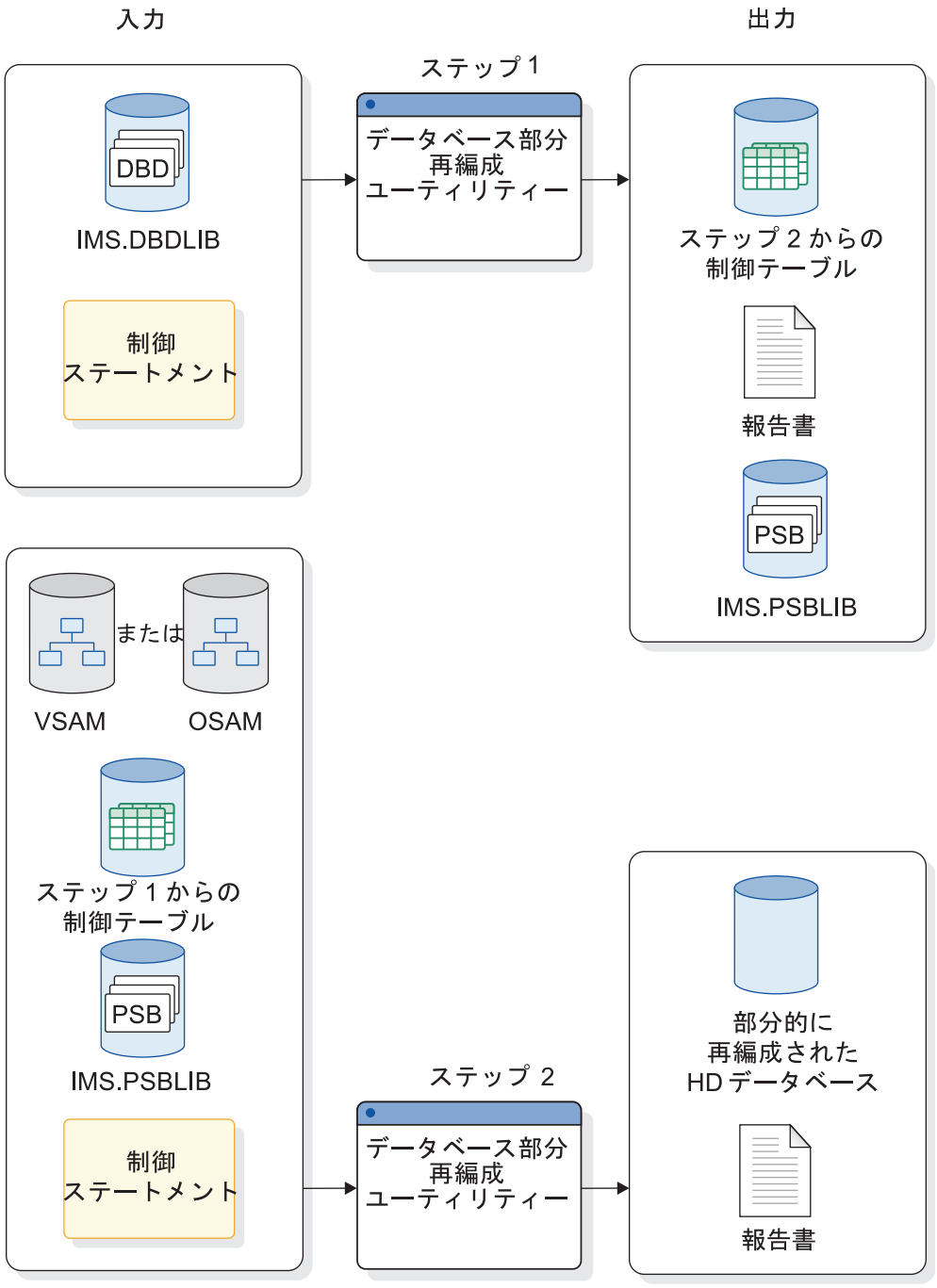


図 *267. データベース部分再編成ユーティリティー (DFSPRCT1)*

関連概念:

726 ページの『データベース調査ユーティリティー (DFSPRSUR)』

520 ページの『維持管理の計画』

RSR およびデータベース再編成ユーティリティー

リモート・サイト・リカバリー (RSR) 環境において、トラッキングされたデータベースが再編成にかかわってくることによりアクティブ・サイトで新しいイメージ・

コピーが必要とされる場合、それらのイメージ・コピーを、トラッキング・サイトで使用できるようにする必要があります。

新しいイメージ・コピーが必要であることをトラッキング・サイトに知らせるために、データベースを次に割り振る際に情報レコードがログに書き込まれますが、トラッキング・サイトへイメージ・コピーを送るのはユーザーの責任です。

トラッキング・サイトではデータベース再編成は使用できません。データベース再編成ユーティリティーが、トラッキング・サービス・グループ (SG) にサインオンする一方でトラッキングされたデータベースに対する許可を要求した場合、その許可はリジェクトされます。

次の場合には、アクティブ・サイトからトラッキング・サイトへイメージ・コピーを送らなければなりません。

- HISAM 再ロード・ユーティリティー (DFSURRL0) でデータベースが再ロードされる場合。再ロードされたアンロード・データ・セットを、イメージ・コピーとして使用することができます。
- 接頭部更新ユーティリティー (DFSURGP0) によりデータベースが処理され、更新がアクティブ・サイトで記録されており、これらの更新を通常のトラッキング時にトラッキング・サイトで適用する必要がある場合。接頭部更新ユーティリティーがログ・データを作成する場合は、再ロード・ステップ後の任意の時点でとられたイメージ・コピーがトラッキング・サイトで必要になります。 接頭部更新ユーティリティーがログ・データを作成しない場合は、接頭部更新ステップ後にとられたイメージ・コピーがトラッキング・サイトで必要になります。
- HD 再ロード・ユーティリティー (DFSURGL0) でデータベースが再ロードされ、接頭部の更新が不要な場合。

アクティブ・サイトでデータベースを再編成した後、次の手順で新しいイメージ・コピーをトラッキング・サイトへ送り、それをインストールしてください。

- アクティブ・サイトでは次のことを行います。
  1. データベースを再編成します。
  2. データベースのイメージ・コピーを作成します。
  3. そのイメージ・コピーをトラッキング・サイトへ送信します。
  4. RSR が、データベース再編成に関する情報ログ・レコードをトラッキング・サイトへ送ります。
- トラッキング・サイトでは次のことを行います。

  再編成されたデータベースをリカバリー作動可能レベルでトラッキングする場合、ユーザーが行う必要があるのは、そのイメージ・コピーを、 NOTIFY.IC コマンドで RECON に記録することだけです。データベース作動可能レベルでトラッキングされるデータベースには、以下のことが当てはまります。

  1. RSR は、再編成情報ログ・レコードを使用して RECON データ・セットを更新し、そのリカバリーが必要であることを示すメッセージを出します。

     このステップは、次のような処理時にはいつでも発生する可能性があります。イメージ・コピーが RECON データ・セットに記録される前にデータベース再編成情報レコードが処理された場合、 RSR は、そのデータベースを

自動的に停止します。イメージ・コピーが適用された後でレコードが処理された場合は、メッセージは出されません。

2. イメージ・コピーが到着した時点でそのデータベースが使用されていないことを確認します。
   - データベースが現在停止されていない場合は、そのデータベースで /DBRECOVERY DATABASE|AREA コマンドを発行します。
   - データベース・リカバリー・ジョブがそのデータベースで現在実行されている場合は、そのジョブを取り消します。
3. NOTIFY.IC コマンドを使用して、RECON データ・セットにイメージ・コピーを記録します。
4. データベース・リカバリー・ユーティリティーを実行して、そのイメージ・コピーを適用します。 GENJCL.RECOV コマンドを使用して、必要な JCL を生成することができます。
5. /START DATABASE|AREA|DATAGRP コマンドを発行して、データベースのトラッキングを再開します。

オフラインでの HISAM、HD、および索引データベースの再編成

このトピックでは、HISAM、HDAM、HIDAM、1 次索引、および副次索引の各データベース・タイプをオフラインで再編成する方法について説明します。

HISAM データベースの再編成 (副次索引がない場合)

論理関係または副次索引を使用していない HISAM データベースを再編成するには、以下のステップを実行します。

1. HISAM 再編成アンロード・ユーティリティーを用いて、データベースをアンロードします。
2. データ・セットをアンロードしたときはいつでも、再ロードする前にデータ・セットを削除し、再割り振りを行ってください。
3. HISAM 再編成再ロード・ユーティリティーを用いて、データベースを再ロードします。
4. データベースを再ロードしたら、このデータベースのイメージ・コピーを作成します。

HDAM または HIDAM データベースの再編成 (論理関係または副次索引がない場合)

論理関係または副次索引を使用していない HDAM または HIDAM データベースを再編成するには、以下のステップを実行します。

1. HD 再編成アンロード・ユーティリティーを用いて、データベースをアンロードします。
2. データ・セットをアンロードしたときはいつでも、再ロードする前にデータ・セットを削除し、再割り振りを行ってください。
3. HD 再編成再ロード・ユーティリティーを用いて、データベースを再ロードします。
4. 再ロードされたデータベースのイメージ・コピーを作成します。

1 次索引または副次索引の再編成

HIDAM の 1 次索引、および HISAM、HDAM、または HIDAM データベースの副次索引は、同じ方法で再編成します。

HIDAM の 1 次索引、または HISAM、HDAM、HIDAM の副次索引を再編成するには、以下のステップを実行します。

1. HISAM 再編成アンロード・ユーティリティーを用いて、索引データベースをアンロードします。
2. データ・セットをアンロードしたときはいつでも、再ロードする前にデータ・セットを削除し、再割り振りを行ってください。
3. HISAM 再編成再ロード・ユーティリティーを用いて、索引データベースを再ロードします。データベースを再ロードしたらすぐに、このデータベースのイメージ・コピーを作成します。

HALDB データベースの再編成

HALDB の主な利点の 1 つは、単純化されて短くなった再編成処理と、組み込まれた HALDB オンライン再編成機能を使用して HALDB データベースをオンラインで再編成できることです。

PHDAM および PHIDAM HALDB データベースはどちらも、オンラインまたはオフラインで再編成することができます。PSINDEX HALDB については、オフラインの再編成のみが可能です。 HALDB をオンライン、オフラインのどちらで再編成する場合も、再編成処理は他の全機能データベースで使用される再編成処理と異なっています。

論理関係および副次索引をもつ HALDB データベースの再編成には、これらのポインターを更新するユーティリティーの実行は必要ありません。その代わりに、HALDB はこれらのポインターが使用された場合に自己回復ポインター処理を使用して訂正します。

関連概念:

584 ページの『HALDB 設計のインプリメント』

HALDB のオフライン再編成

HALDB データベースと他の全機能 IMS データベースのオフライン再編成処理は類似しており、どちらもデータベースのアンロードと再ロードで構成されています。

HALDB オフライン再編成処理には、他の全機能データベースの再編成処理にくらべて次のような利点があります。

- 1 回に 1 つの HALDB 区画を再編成することも、複数の区画を並列的に再編成することもできます。
- HALDB データベースの自己回復ポインター処理により、HALDB データベースの再編成後に論理関係および副次索引を手動で更新する必要がありません。
- HALDB データベースを再編成する場合に HALDB データ・セットの DD ステートメントを組み込む必要がありません。HALDB データ・セットは動的に割り振りされます。

関連タスク:

763 ページの『HALDB のオンライン再編成後のオフライン再編成』

HALDB オフライン再編成の概説

HALDB データベースのオフライン再編成は、1 つ以上の並列処理で実行することができます。これらの処理は、1 つ以上の区画をアンロードし、再ロードします。

データベースに副次索引または論理関係がある場合、追加のステップは必要ありません。 HALDB の自己修復処理により、再編成中のポインターの更新が不要になります。再編成の所要時間は、区画のサイズによって異なります。区画が小さければ時間が短くなります。より多くの区画を作成し、それらを並列的に再編成することによって、再編成の所要時間を短縮することができます。

HALDB データベースのオフライン再編成に必要な基本ステップは、次のとおりです。

1. HD 再編成アンロード・ユーティリティー (DFSURGU0) を実行して、データベース全体、区画の範囲、または単一の区画をアンロードします。
2. オプションで、次のいずれかのユーティリティーを実行して区画を初期設定します。
   - HALDB 区画データ・セット初期設定ユーティリティー (DFSUPNT0)
   - データベース事前再編成ユーティリティー (DFSURPR0)
3. HD 再編成再ロード・ユーティリティーを実行してデータベースまたは区画を再ロードします。
4. 再ロードされたすべての区分データ・セットのイメージ・コピーを作成します。

以下の図は、論理関係および副次索引をもつ HALDB データベースの再編成に使用されるオフライン処理を示したものです。この場合、複数の区画が並列処理によって再編成されています。各区画のアンロードと再ロードは、データベース全体をアンロードして再ロードするより短い時間で実行することができます。これは、非 HALDB 全機能データベースの処理に比べて非常に高速です。その上、論理関連データベースのポインター更新や副次索引の再作成のための時間が必要ありません。これによって処理はさらに短縮されます。

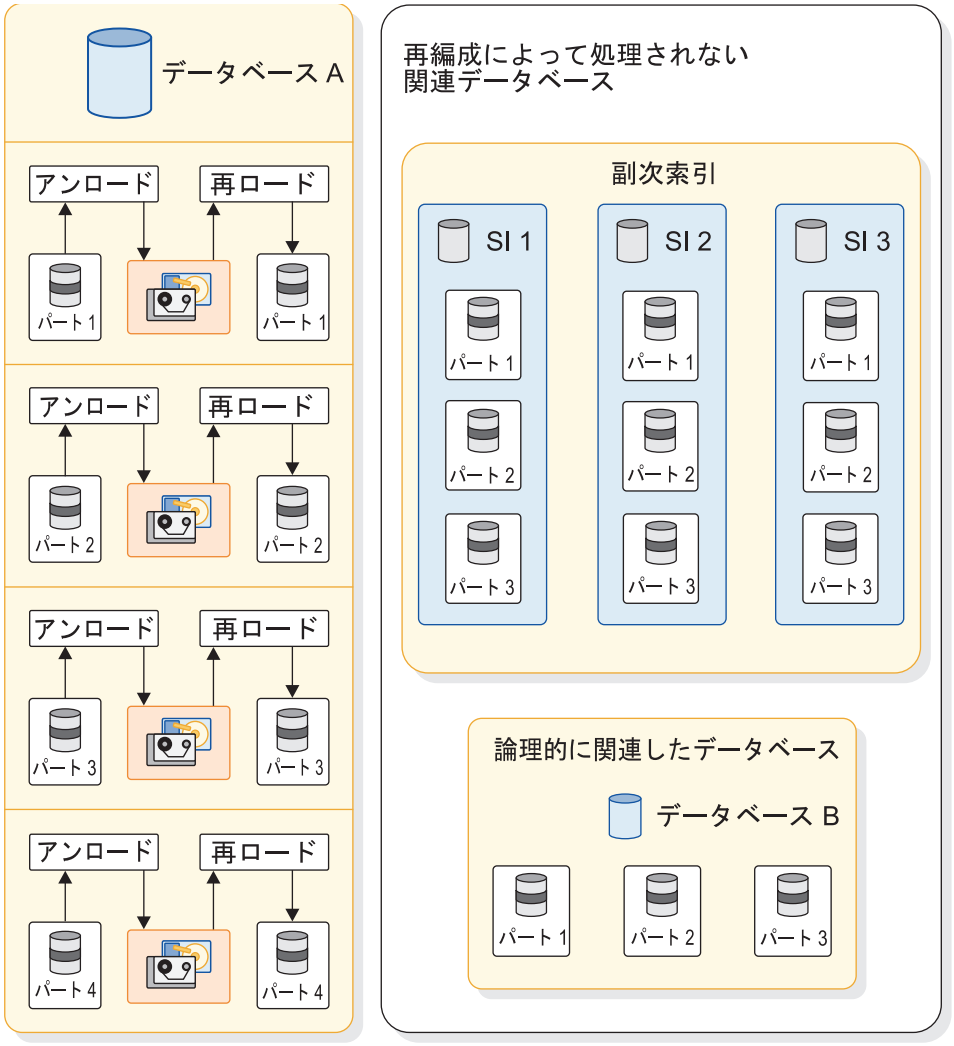


図 *268. HALDB* データベースのオフライン再編成

関連概念:

738 ページの『HALDB 区画に分割された副次索引データベースの再編成』

関連タスク:

711 ページの『再編成ユーティリティーを使用するオフライン再編成』

HALDB データベースのオフライン再編成のオプション

HALDB データベースを再編成するには、以下に示すいくつかのオプションがあります。

HALDB データベースを再編成するためのオプションは次のとおりです。

- 任意の数の区画を再編成することができます。 1 つの区画だけを再編成する必要がある場合は、他の区画を処理せずに、その区画のみをアンロードおよび再ロードすることができます。
- 複数の区画を並列的に再編成するか、データベースの再編成を 1 回の処理で行うことができます。並列処理の程度は、実行している再編成ジョブの数によって決まります。それぞれのジョブで、1 つの区画または複数の区画を処理すること

ができます。並列処理の程度を高めるには、再編成ジョブの数を増やし、それぞれのジョブが処理する区画の数を減らします。

- 既存のデータベース・データ・セットを再利用するか、既存のデータベース・データ・セットをアンロード後に削除し、再ロード用に新規データ・セットを割り振ることができます。
- 区画の追加、区画の削除、または区画境界の変更を行うことができます。

関連タスク:

889 ページの『HALDB データベースの変更』

584 ページの『HALDB 区画定義ユーティリティーによる HALDB データベースの作成』

オフライン再編成のための HALDB 区画およびデータベースのアンロード

HD 再編成アンロード・ユーティリティー (DFSURGU0) を使用して、 HALDB 区画またはデータベースをアンロードすることができます。

HALDB データベース全体をアンロードするには、SYSIN DD ステートメントを組み込まないでください。その他の任意の数の区画をアンロードする場合には、SYSIN データ・セットに制御ステートメントを組み込む必要があります。制御ステートメントはアンロードする最初の区画の名前を識別し、複数の区画をアンロードしている場合には、アンロードする区画の数も識別します。

複数の区画は、連続してアンロードされます。キー範囲の区分化では、連続した区画はハイ・キーによって決まります。区画選択出口ルーチンを使用している場合は、連続した区画は出口ルーチンで割り当てられた順序によって決まります。

HALDB データベース・データ・セットには、DD ステートメントを組み込まないでください。 HD 再編成アンロード・ユーティリティーは、HALDB データ・セットの動的割り振りを使用します。これは、非 HALDB データベースには当てはまりません。

要件**:** アンロードされる区画内のすべてのデータ・セットについて、バッファー・プールを提供しなければなりません。これには ILDS も含まれます。

推奨事項**:** OSAM を使用するデータベースには、OSAM 順次バッファリングを使用可能にしてください。

以下のコードは、1 つの区画をアンロードする HD 再編成アンロード・ユーティリティーの制御ステートメントの例です。

```
SYSIN DD *
PARTITION=PEO02
```

以下のコードは、3 つの区画をアンロードする HD 再編成アンロード・ユーティリティーの制御ステートメントを示しています。

```
SYSIN DD *
PARTITION=PEO04,NUMBER=3
```

以下の JCL は、HALDB 区画をアンロードするサンプル・ジョブを示しています。

```
//JOUKO3C JOB  (999,POK),JOUKO3,CLASS=A,NOTIFY=&SYSUID,
//        MSGLEVEL=(1,1),MSGCLASS=X,REGION=0M
//JOBLIB    DD DSN=IMSPSA.IMS0.SDFSRESL,DISP=SHR
//          DD DSN=IMSPSA.IM0A.MDALIB,DISP=SHR
//*******************************************************************
//*   HD UNLOAD FOR THE PARTITION PEO02 OF PEOPLE DATABASE
//*******************************************************************
//UNLOAD   EXEC PGM=DFSRRC00,REGION=1024K,
//         PARM='ULU,DFSURGU0,PEOPLE,,,,,,,,,,Y,N'
//DFSRESLB  DD DSN=IMSPSA.IMS0.SDFSRESL,DISP=SHR
//IMS      DD DISP=SHR,DSN=JOUKO3.HALDB.DBDLIB
//DFSURGU1 DD DSN=JOUKO3.UNLOAD.PEO02,UNIT=3390,VOL=SER=TOTIMN,
//         SPACE=(CYL,(10,5),RLSE),DISP=(NEW,CATLG)
//DFSVSAMP DD *
IOBF=(4096,50)
VSRBF=8192,50
/*
//SYSPRINT DD SYSOUT=*
//DFSCTL    DD *
SBPARM ACTIV=COND
/*
//SYSIN   DD *
PARTITION=PEO02
/*
```

IMS ハイパフォーマンス・アンロード・ツールは、HD 再編成アンロード・ユーティリティーの代わりに使用することができます。ハイパフォーマンス・アンロード・ツールを使用して、任意の数の区画またはデータベース全体をアンロードすることができます。

関連資料: ハイパフォーマンス・アンロード・ツールについての詳細な情報は、「*IMS High Performance Unload for z/OS* ユーザーズ・ガイド」を参照してください。

オフライン再編成のための HALDB データベース・データ・セットの再割り振り

HALDB データベース・データ・セットを再ロードする前に、それらを削除および再定義する必要はありません。これは OSAM および VSAM データ・セットの両方に当てはまります。

VSAM データ・セットは、ILDS を除いて、REUSE オプションを使用して指定しなければなりません。 HALDB はこのオプションをサポートします。

区分データ・セットを削除および再定義して、そこにデータを再ロードしない場合は、区分データ・セットを初期設定しなければなりません。区分データ・セットを削除および再定義した後でそこにデータを再ロードする場合は、区分データ・セットを初期設定する必要はありません。

VSAM データ・セットを削除および再定義すると、区画の再ロード時に z/OS IEC161I システム・メッセージを受け取ります。これはエラー・メッセージではありません。これは、VSAM データ・セットがオープン時に空だったことを示しています。以下の例は ILDS の場合のメッセージを示したものです。

```
IEC161I 152-061,JOUKO3D,RELOAD,PEO01L,,,
IEC161I JOUKO3.HALDB.DB.PEOPLE.L00001,
IEC161I JOUKO3.HALDB.DB.PEOPLE.L00001.DATA,CATALOG.TOTICF2.VTOTCAT
```

関連資料: IEC システム・メッセージの詳細については、「*z/OS MVS* システム・メッセージ 第 *7* 巻 *(IEB - IEE)*」を参照してください。

オフライン再編成のための HALDB 区画およびデータベースの再ロード

HALDB 区画およびデータベースは、HD 再編成再ロード・ユーティリティー (DFSURGL0) で再ロードすることができます。

HD 再編成再ロード・ユーティリティーは、HD 再編成アンロード・ユーティリティーから出力ファイルを読み取ります。再ロードする区画は指定しません。それらは、HD 再編成再ロード・ユーティリティーへの入力ファイル内のレコードによって決定されます。

HALDB データベース・データ・セットには、DD ステートメントを組み込まないでください。 HD 再編成再ロード・ユーティリティーは、HALDB データ・セットの動的割り振りを使用します。これは、非 HALDB データベースには当てはまりません。

HALDB オンライン再編成機能を前に使用した結果としてオフライン再編成の前のアクティブ・データ・セットが M-V データ・セットであった場合でも、HD 再編成再ロード・ユーティリティーに割り振られるデータ・セットは、常に、A-J データ・セットです。

再ロードされる区画内のすべてのデータ・セットについて、バッファー・プールを提供しなければなりません。これには ILDS も含まれます。

HD 再編成再ロード・ユーティリティーは、ロードする区画内のデータ・セットにイメージ・コピーで必要なフラグをセットします。それらが再ロードされた後、データベース・データ・セットの場合と同じように、イメージ・コピーしておくようにします。

以下のコードは、HALDB 区画を再ロードするサンプル・ジョブを示しています。再ロードする区画は、入力ファイル内のレコードによって決まります。

```
//JOUKO3D JOB  (999,POK),JOUKO3,CLASS=A,NOTIFY=&SYSUID,
//        MSGLEVEL=(1,1),MSGCLASS=X,REGION=0M
//JOBLIB    DD DSN=IMSPSA.IMS0.SDFSRESL,DISP=SHR
//          DD DSN=IMSPSA.IM0A.MDALIB,DISP=SHR
//*****************************************************************
//*   HD RELOAD FOR THE PEOPLE DATABASE
//*****************************************************************
//RELOAD   EXEC PGM=DFSRRC00,REGION=1024K,
//         PARM='ULU,DFSURGL0,PEOPLE,,,,,,,,,,Y,N'
//DFSRESLB  DD DSN=IMSPSA.IMS0.SDFSRESL,DISP=SHR
//IMS      DD DISP=SHR,DSN=JOUKO3.HALDB.DBDLIB
//DFSUINPT DD DSN=JOUKO3.UNLOAD.PEOPLE,DISP=OLD
//DFSVSAMP DD *
VSRBF=8192,50
IOBF=(4096,50)
/*
//SYSPRINT DD SYSOUT=*
//DFSSTAT  DD SYSOUT=*
```

ILDS 再編成更新:

HD 再編成再ロード・ユーティリティーは、論理関係または副次索引のターゲットを含む区画の ILDS を更新します。

再編成の後、このユーティリティーには、ILDS 更新の 3 つのオプションがあります。

- 制御ステートメントなし
- ILDSINGLE 制御ステートメント
- NOILDS 制御ステートメント

HD 再編成再ロード・ユーティリティーの SYSIN データに制御ステートメントを指定しないと、副次索引または論理関係のターゲットが区画に挿入された時点で、ILDS 項目が更新または作成されます。直前の再編成がターゲット・セグメントを区画にロードした場合は、項目が存在します。 ILDS の更新は VSAM 更新モードで実行されます。 CI または CA が埋まった場合は、VSAM によって分割の必要があります。 ILDS 内のフリー・スペースが、このような分割を避けるのに役立つことがあります。更新は、ランダムまたは順次の場合があります。これは、これらのセグメントが挿入された順序およびそれらの ILK によって異なります。 ILDS キーは、作成された時点のターゲット・セグメントの位置に基づいた ILK に基づきます。

ILDSINGLE 制御ステートメントを指定することにより、ILDS の中にフリー・スペースを作成できます。 ILDSINGLE オプションは、CI および CA 分割の除去、未使用 ILE の除去、およびそれ以降の再編成とリカバリーに関するパフォーマンスの改善など、多くのパフォーマンス上の利点を提供します。 ILDS は、ILDSINGLE オプションを指定してユーティリティーを実行する前に、削除されて再定義されていなければなりません。

SYSIN データに NOILDS 制御ステートメントを指定すると、HD 再編成再ロード・ユーティリティーは ILDS 内の項目を更新または作成しません。HALDB 索引/ILDS 再作成ユーティリティー (DFSPREC0) などのほかの手段を使って ILDS を再作成する必要があります。

NOILDS 制御ステートメントを使用すると最速で再ロードを実行でき、HALDB 索引/ILDS 再作成ユーティリティーによって各区画の ILDS を並行して更新できます。ただし、HALDB 索引/ILDS 再作成ユーティリティーは、区画を 1 つずつ読み取ってその ILDS を更新します。オプションとして、HALDB 索引/ILDS 再作成ユーティリティーでは ILDS を VSAM ロード・モードで再作成することができます。これにより、パフォーマンスが向上し、ILDS にフリー・スペースが組み込まれます。

HD 再編成再ロード・ユーティリティーには ILDSMULTI 制御ステートメントもありますが、ILDSMULTI はマイグレーション再ロードにのみ適用されます。ILDSMULTI について詳しくは、「*IMS V13* データベース・ユーティリティー」の『HD 再編成再ロード・ユーティリティー』のセクションを参照してください。

HALDB 区画に分割された副次索引データベースの再編成

区画に分割された副次索引 (PSINDEX) データベースを再編成しなければならないことがあります。 HALDB データベースの再編成には副次索引の再作成は必要ないため、PSINDEX データベースは、時間をかけて項目が追加されるにつれて編成が混乱することがあります。

PSINDEX データベースを再編成するには、HD 再編成アンロード・ユーティリティーと HD 再編成再ロード・ユーティリティーを使用することができます。他の HALDB データベースを再編成する場合の制約事項および推奨事項が PSINDEX データベースにも当てはまりますが、1 つだけ例外があります。HALDB 副次索引には ILDS がありません。副次索引には、HD 再編成再ロード・ユーティリティーの制御ステートメントを使用してはなりません。

PSINDEX データベースを再編成するステップは、他のタイプの HALDB データベースをオフラインで再編成するステップと同じです。

関連タスク:

732 ページの『HALDB オフライン再編成の概説』

HALDB オンライン再編成

IMS の HALDB オンライン再編成機能が組み込まれ、データベース再編成中にも HALDB 区画をオンラインのままにし、IMS アプリケーション・プログラムで使用可能にすることができます。

HALDB オンライン再編成は、パフォーマンスを向上させる、再クラスター化およびスペース配分による利点を提供します。

PHDAM または PHIDAM HALDB 区画オンライン再編成の実行には破壊性がないため、データ共用 IMS システムによる更新を含めて、同時 IMS 更新が可能です。オンライン再編成に破壊性がないのは、IMS が少量のデータを区画のオリジナル・データ・セット (入力データ・セット) から別の出力データ・セットにコピーするためです。IMS はどのデータがコピーされたかを追跡して、IMS アプリケーションが正しいデータ・セットの集まりから自動的にデータを検索または更新できるようにします。

HALDB オンライン再編成は、確立されているデータ定義およびデータ・セットの命名規則を HALDB データベース用に拡張しています。HALDB データベース内のデータ・セット・グループは、サポートされている 10 のデータ・セット・グループの DDNAME およびデータ・セット名に文字 A から J までを使用します。PHIDAM データベースの 1 次索引は、その名前に文字 X を使用します。このデータ定義およびデータ・セットの命名規則が拡張されて、IMS はデータ・セットの代替セットに文字 M から V (および Y) を使用します。

HALDB 区画の初期ロードまたはオフライン再編成再ロードでは、必ず A から J まで (および X) データ・セットを使用します。 HALDB 区画を最初にオンラインで再編成するまでは、A から J まで (および X) データ・セットが使用されます。

HALDB 区画のオンライン再編成には 3 つの段階があります。

1. 初期設定段階では、IMS が出力データ・セットを準備し、RECON データ・セットを更新します。

2. コピー段階では、IMS が入力データ・セットから出力データ・セットにデータをコピーして、実際の再編成を実行します。
3. 終了段階では、IMS が入力データ・セットをクローズし、RECON データ・セットを更新します。

HALDB オンライン再編成機能は、オンライン HALDB データベースの構造に特定の変更を行う場合にも使用できます。HALDB オンライン再編成機能の変更オプションを使用する際には、追加の制約事項および要件が適用されます。変更オプションについて詳しくは、848 ページの『オンライン HALDB データベースの定義の変更』を参照してください。

関連概念:

520 ページの『維持管理の計画』

707 ページの『データベースの再編成』

153 ページの『HD データベースを対象として発行可能な DL/I 呼び出し』

747 ページの『HALDB オンライン再編成のデータ・セット命名規則』

855 ページの『ALTER オプションが指定されている場合のオンライン再編成処理』

関連タスク:

603 ページの『データベースの最小サイズの見積もり』

848 ページの『オンライン HALDB データベースの定義の変更』

HALDB オンライン再編成の初期設定段階

HALDB 区画のオンライン再編成は、INITIATE OLREORG コマンドを使用して開始します。

初期設定段階では、IMS が RECON データ・セットを更新して、オンライン再編成を実行している IMS システムによるオンライン再編成の所有権を確立します。この所有権は、現行オンライン再編成が完了するか、所有権が別の IMS システムに転送されるまで、他の IMS システムは HALDB 区画の再編成を実行できないことを意味します。IMS は、M-V (および Y) DBDS レコードがまだ存在しない場合、RECON データ・セットにそれらの DBDS を追加します。 IMS はまた、対応する A-J (および X) DBDS を含む既存の変更累積グループおよび DBDS グループに、M-V (および Y) DBDS を追加します。

HALDB 区画のオンライン再編成が開始される前には、HALDB 区画にはアクティブ・データ・セットが 1 セットあります。これらのアクティブ・データ・セットは、コピー段階のための入力データ・セットです。また、前のオンライン再編成で残された非アクティブ・データ・セットの、IMS アプリケーション・プログラムで使用されていないセットが存在することもあります。

初期設定段階では、IMS は非アクティブ・データ・セットのそれぞれを評価して、それが出力データ・セットの要件に合うかどうかを確認します。出力データ・セットのいずれかが存在しないと、IMS はこの段階で自動作成します。

初期設定段階の最後に、IMS はオリジナル・アクティブ・セットのデータ・セットを入力セットとして扱い、非アクティブ・データ・セットを出力セットとして扱います。このようなデータ・セットの入力セットと出力セットの使用は、区画の カー

ソル・アクティブ 状況によって表され、RECON データ・セットのオンライン再編成レコードに記録されます。 RECON データ・セット内の区画のデータベース・レコードのリストは、OLREORG CURSOR ACTIVE=YES を示します。区画のリストは、DBDS の両方のセットがアクティブであることも示します。リストされている DBDS の最初のセットは入力データ・セット、DBDS の 2 番目のセットが出力データ・セットで、例えば DBDS ACTIVE=A-J and M-V のようになります。区画がカーソル・アクティブ状況にある間、データ・セットの両方のセットが、どのアプリケーションによって処理される区画に対しても使用可能になっていなければなりません。

次の例は、カーソル・アクティブ状況を持つ HALDB 区画の、RECON データ・セットのリストの一部を示しています。

```
--------------------------------------------------------------------------------
04.174 12:30:54.1                  LISTING OF RECON                   PAGE 0003
--------------------------------------------------------------------------------
 DB
  DBD=POHIDKA   MASTER DB=DBOHIDK5  IRLMID=*NULL  CHANGE#=6     TYPE=PART
  USID=0000000004  AUTHORIZED USID=0000000004  HARD USID=0000000004
  RECEIVE USID=0000000004  RECEIVE NEEDED USID=0000000000
  DBRCVGRP=**NULL**
  DSN PREFIX=IMSTESTS.DBOHIDK5                          PARTITION ID=00001
  PREVIOUS PARTITION=**NULL**    NEXT PARTITION=POHIDKB
  OLRIMSID=IMS1       ACTIVE DBDS=A-J and M-V
  REORG#=00004

  FREE SPACE:
    FREE BLOCK FREQ FACTOR=2    FREE SPACE PERCENTAGE=20

  PARTITION HIGH KEY/STRING (CHAR):               (LENGTH=5  )
    K2000
  PARTITION HIGH KEY/STRING (HEX):
    D2F2F0F0F0404040404040404040404040404040404040404040404040404040

  OSAM BLOCK SIZE:
    A = 4096
    B = 4096

  FLAGS:                              COUNTERS:
    BACKOUT NEEDED        =OFF          RECOVERY NEEDED COUNT   =0
    READ ONLY             =OFF          IMAGE COPY NEEDED COUNT =0
    PROHIBIT AUTHORIZATION=OFF          AUTHORIZED SUBSYSTEMS   =1
                                        HELD AUTHORIZATION STATE=6
                                        EEQE COUNT              =0
    TRACKING SUSPENDED    =NO           RECEIVE REQUIRED COUNT  =0
    OFR REQUIRED          =NO           OLR ACTIVE HARD COUNT   =0
    PARTITION INIT NEEDED =NO           OLR INACTIVE HARD COUNT =0
    OLREORG CURSOR ACTIVE =YES
    PARTITION DISABLED    =NO
    ONLINE REORG CAPABLE  =YES
    REORG INTENT          =NO
    QUIESCE IN PROGRESS   =NO
    QUIESCE HELD          =NO
    ALTER IN PROGRESS     =NO
    ALTER COMPLETE        =NO

  ASSOCIATED SUBSYSTEM INFORMATION:
                              ENCODED  B/O NEEDED
     -SSID-   -ACCESS INTENT-  -STATE-   -COUNT-  -SS ROLE-
    IMS1         UPDATE           6         0      ACTIVE
--------------------------------------------------------------------------------
```

初期設定段階の間に、事前に存在しているデータ・セットが受け入れ不能、あるいは自動作成されるデータ・セットのディスク・スペースが不足、などのさまざまなエラー条件によって、初期設定に障害が発生することがあります。ただし、エラーの発生がデータ・セットの作成と検査プロセスの後で、IMS がカーソル・アクティブ状況を RECON データ・セットに記録する前であれば、自動作成された出力データ・セットは事前に存在しているデータ・セットとともに保存されます。

オンライン再編成の初期設定段階中にはまた、IMS が再編成用のプログラム仕様ブロック (PSB) を動的に作成します。 PSB 名は、7 文字の HALDB 区画名の接頭部に文字ゼロ ('0') が 1 個ついたものです。例えば、名前が SSN5603 の HALDB 区画の場合、再編成作業用の動的 PSB の名前は 0SSN5603 です。この PSB は PSBLIB または ACBLIB の中には存在しませんが、名前は RECON のリストまたはユーティリティーからの出力に現れることがあります。

関連概念:

749 ページの『既存の出力データ・セットの要件』

関連資料:

INITIATE OLREORG コマンド (コマンド)

HALDB オンライン再編成のコピー段階

HALDB オンライン再編成のコピー段階では、HALDB 区画が A-J (および X) データ・セットと M-V (および Y) データ・セットで構成されます。IMS アプリケーションが区画にアクセスするためには、両セットのデータ・セットが使用可能であることが必要です。

IMS が HALDB 区画をオンラインで再編成する間、IMS アプリケーションは区画に対してデータベース更新を実行することができます。データベース更新は、どのデータがアプリケーションによって更新されるかに応じて、入力データ・セットに対して行われるものと出力データ・セットに対して行われるものがあります。どのデータ・セットが更新されるかは、アプリケーション・プログラムにとっては透過的です。以下の図は、オンライン再編成中のある時点での入力データ・セットと出力データ・セットとの関係を示しています。

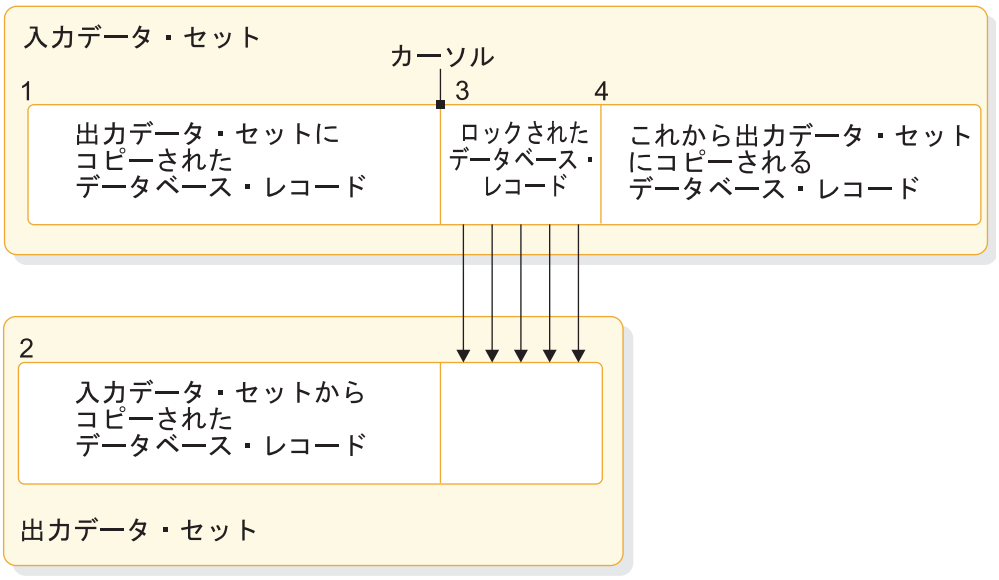


図 *269. HALDB* 区画のオンライン再編成中の入力データ・セットと出力データ・セットの関係

前の図は、HALDB 区画のデータベース・データ・セットの 2 つのセットを示しています。再編成されていない入力データ・セットと、(少なくとも一部は) 再編成された出力データ・セットです。この図は、左から右へ、上部の入力データ・セットから下部の出力データ・セットへと進行する再編成を示したものです。図のデータ・セットは 4 つの区域に分割されています。

1. 出力データ・セットにコピーされた、入力データ・セット内のデータ。この区域は古い (再編成前の) データ編成を反映しており、そのデータ・セットが後のオンライン再編成で出力データ・セットとして再利用されるまでは、IMS アプリケーションによって再度使用されることはありません。
2. 入力データ・セットからコピーされた、出力データ・セット内のデータ。この区域のデータは再編成され、IMS アプリケーションが再編成中に使用することができます。
3. ロックされて、入力データ・セットから出力データ・セットにコピーされ再編成される処理中の、入力データ・セットおよび出力データ・セット内のデータ。このロック・レコードの区域は、再編成単位 と呼ばれます。リカバリーの観点からすると、この再編成単位はリカバリー単位と同等です。

   IMS が現行の再編成単位を処理している間、ロックされたデータ・レコードにアクセスする IMS アプリケーションは、IMS がそれらのレコードの再編成を完了するまで待機しなければなりません。再編成単位のコピーと再編成が完了した後、IMS は変更をコミットしてレコードをアンロックし、再び IMS アプリケーションによって使用可能にします。
4. まだ出力データ・セットにコピーされていない、入力データ・セット内のデータ。このデータはまだ再編成されておらず、IMS アプリケーションが再編成中に使用することができます。

オンライン再編成の進行につれ、IMS はカーソル と呼ばれる一種のポインターを使用し、すでに入力データ・セットから出力データ・セットにコピーされたデータ

ベース・レコードの終点をマークします。再編成とコピーの進行に伴って、このカーソルは区画の中を (前の図の左から右に) 移動します。

IMS アプリケーション・プログラムがオンライン再編成中の HALDB 区画のデータにアクセスすると、IMS は次の場所からデータ・レコードを検索します。

- そのデータベース・レコードがカーソル位置の『上または前』にある場合は、出力データ・セットから。
- そのデータベース・レコードがカーソル位置の『後』にある場合は、入力データ・セットから。

そのデータ・レコードがたまたま再編成単位の中にある場合は、IMS はレコードがアンロックされた後にデータ・アクセスを再試行します。アプリケーション・プログラムは、再編成単位内のデータについては、エラー状況コードを受け取りません。

入力データ・セットと出力データ・セットのリカバリーを可能にするために、オンライン再編成中にはすべてのデータベース変更がログに記録され、それには入力データ・セットから出力データ・セットにコピーされたデータベース・レコードも含まれます。

HALDB オンライン再編成の終了段階

HALDB 区画のオンライン再編成は、コピー段階の終了後、または IMS が再編成中にエラー条件を検出した時点で、終了します。

HALDB 区画のオンライン再編成は、`TERMINATE OLREORG` コマンドを使用して終了することもできます。複数の区画を再編成している場合は、NAME キーワードでアスタリスクを指定して、すべての区画の再編成を終了することができます。

HALDB 区画のコピー段階が完了した後、出力データ・セットはアクティブ・データ・セットになり、入力データ・セットは非アクティブ・データ・セットになります。アクティブ・データ・セットは、IMS アプリケーション・プログラムによるすべてのデータ・アクセスに使用されます。非アクティブ・データ・セットはアプリケーション・プログラムによっては使用されませんが、後続のオンライン再編成で再利用することができます。区画の初期ロードまたはバッチ再編成再ロードを実行しない限り、ある区画に対する連続したオンライン再編成は、データ・セットのこれらの 2 つのセットの間を交互に行き来します。

IMS は RECON データ・セット内にある区画のデータベース・レコードを更新して、区画のカーソル・アクティブ状況をリセットし、現在は 1 セットのデータ・セットのみがあることを示します。 RECON データ・セットのこのレコードのリストには `OLREORG CURSOR ACTIVE=NO` が示され、ACTIVE DBDS フィールドにはアクティブ (新規に再編成された) データ・セットが示されます。 IMS はまた、RECON データ・セット内のオンライン再編成レコードも更新し、再編成が完了した時点のタイム・スタンプを付けます。

`DEL` キーワードを `INITIATE OLREORG` コマンド (または `UPDATE OLREORG` コマンド) に指定した場合は、IMS は区画のカーソル・アクティブ状況をリセットした後で、非アクティブ・データ・セットを削除します。非アクティブ・データ・セットを削除する前に、IMS はバッチ・ジョブを含むすべての共用 IMS システムに対し、オンライン再編成が完了して RECON データ・セットに記録されたことを通知しま

す。オンライン再編成を実行している IMS システムは、これら共用 IMS システムのそれぞれから、現在の非アクティブ・データ・セットをクローズして割り振り解除したことを伝える肯定応答を受け取るまで待ってから、それらのデータ・セットを削除します。ただし、肯定応答を 4.5 分以内に受け取らないと、所有 IMS システムは非アクティブ・データ・セットの削除を試みます。肯定応答がない場合、削除の試行は失敗に終ることが多くなります。

最後に終了段階の終わりで、IMS は RECON データ・セットを更新してオンライン再編成の所有権をリセットし、どの IMS システムも所有権を持たないようにします。この所有権のリセットは、どの IMS システムでも HALDB データベースの後続の再編成を実行できることを意味します。

HALDB 区画のオンライン再編成が、エラーの発生または `TERMINATE OLREORG` コマンドの実行によって完了前に終了した場合には、その区画に対してオンライン再編成を再始動するかオフライン再編成を実行しなければなりません。

以下の図は、正常に行われた HALDB 区画オンライン再編成の、通常の処理ステップを示しています。各列は、オンライン再編成の各段階を通したユーザーから IMS への制御のフロー、および処理イベント発生時のデータ・セットの状況を示しています。

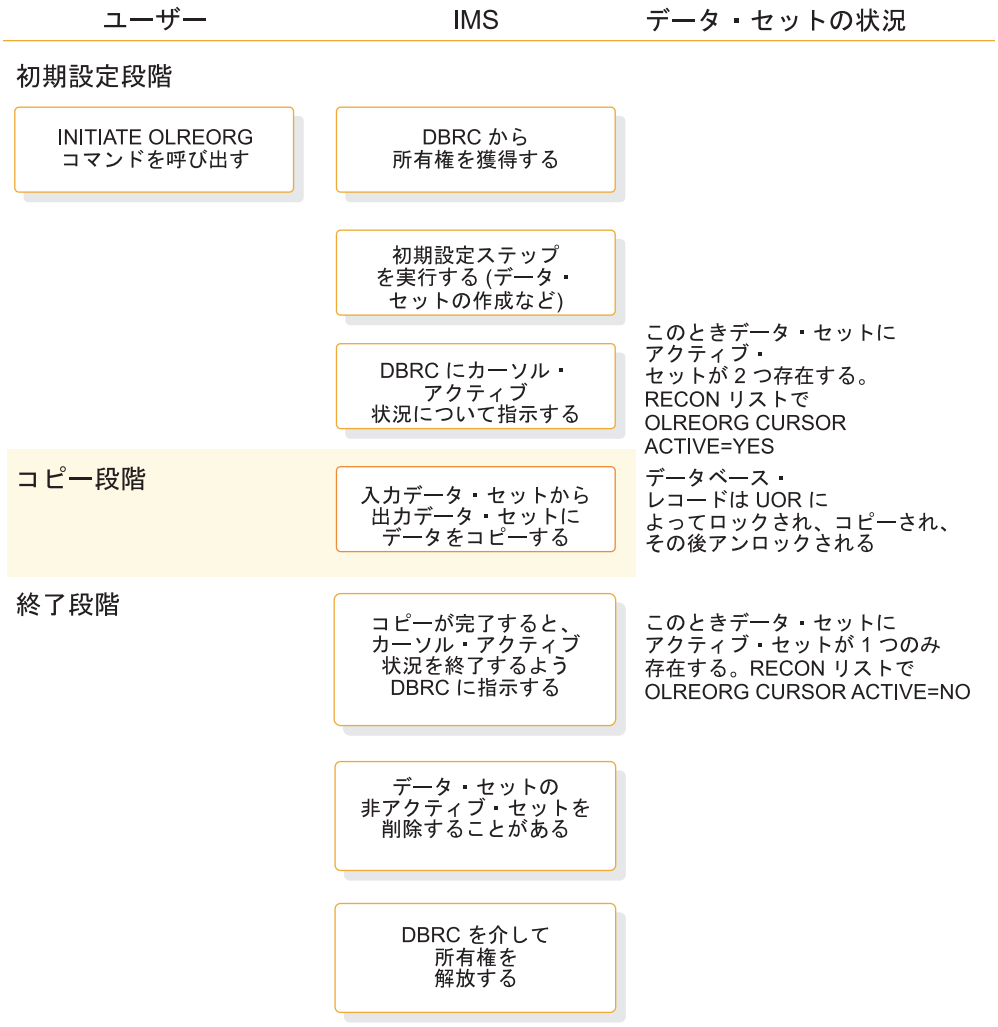


図 *270. HALDB* オンライン再編成の通常の処理ステップ

関連タスク:

860 ページの『変更処理の完了前での停止』

関連資料:

- INITIATE OLREORG コマンド (コマンド)
- TERMINATE コマンド (コマンド)
- UPDATE コマンド (コマンド)

HALDB オンライン再編成の制約事項

HALDB オンライン再編成には、以下の制約事項が適用されます。

制約事項は次のとおりです。

- オンライン再編成を実行できるのは、RECON データ・セット内でオンライン再編成可能として定義されている HALDB のみです。INIT.DB コマンドまたは CHANGE.DB コマンドの OLRCAP パラメーターによって、HALDB は OLRCAP であると定義されます。

- 8-GB OSAM データ・セットを使用する HALDB ではオンライン再編成を実行できません。オンライン再編成を行うには、HALDB が 4-GB OSAM データ・セットを使用している必要があります。INIT.DB コマンドまたは CHANGE.DB コマンドの OSAM8G パラメーターと NOOSAM8G パラメーターは、データ・セットの最大容量を定義します。 OLRCAP と OSAM8G は RECON データ・セット内で同時に指定できないパラメーターとフラグです。
- ある区画についてのオンライン再編成をすでに別の IMS システムが所有している場合は、その区画のオンライン再編成を開始することはできません。これがデフォルトの動作です。このデフォルト動作を変更するには、IMS.PROCLIB データ・セットの DFSDFxxx メンバーの DATABASE セクションで RELOLROWNER パラメーターを指定して、オンライン再編成中に終了した IMS システムに対し、再編成の所有権を解除するよう指示します。その場合、別の IMS システムで再編成を再開できます。また、INITIATE OLREORG コマンドの OPTION(REL) パラメーターを使用して、単一の特定のオンライン再編成について、デフォルトの動作をオーバーライドすることもできます。
- 状況が ALTER IN PROGRESS である区画のオンライン再編成を開始することはできません。
- 区画がカーソル・アクティブ状況にあると、その区画のイメージ・コピーをとることはできません。この制約事項は、カーソル・アクティブ状況がリセットされる前にオンライン再編成が終了していて、その区画のオンライン再編成がどの IMS によっても所有されていない場合にも適用されます。
- 区画の状況が ALTER IN PROGRESS である場合は、区画のイメージ・コピーをとることはできません。
- オンライン再編成から経過中の作業をバックアウトするには、DL/I 領域タイプを使用してバッチ・バックアウトを実行しなければなりません。
- タイプ 2 のコマンドを使用して HALDB 区画のオンライン再編成を開始するには、Operations Manager および構造化呼び出しインターフェースを含む IMS 共通サービス・レイヤーを持っていなければなりません。
- HALDB オンライン再編成は、ローカル・ストレージ・オプションが従属 (LSO=S) に設定された環境 (DL/I が分離したアドレス・スペースで実行される) でのみ実行されます。ローカル・ストレージ・オプションが YES (LSO=Y) に設定された環境で HALDB 区画のオンライン再編成を開始しようとすると、IMS はそれをリジェクトします。
- XRF 複合システムの代替 IMS システムから、HALDB 区画のオンライン再編成を実行することはできません。ただし、XRF のテークオーバー後は、テークオーバー処理開始時点でアクティブだった再編成を新規アクティブ IMS システムが継続します。
- RSR 複合システムのトラッキング IMS システムから、HALDB 区画のオンライン再編成を実行することはできません。ただし、トラッキング IMS システムで DBTRACK として登録されている HALDB については、IMS はすべてのデータベースの更新をトラッキングするのと同じ方法で、オンライン再編成の影響をトラッキングします。
- HALDB 区画のオンライン再編成中には、その区画に対して以下のコマンドを実行することはできません。
  - /START DATABASE または UPDATE DATABASE NAME(*name*) START(ACCESS)

- /DBRECOVERY DATABASE または UPDATE DATABASE NAME(*name*) STOP(ACCESS)
- /DBDUMP DATABASE または UPDATE DATABASE NAME(*name*) STOP(UPDATES)
- /STOP DATABASE または UPDATE DATABASE NAME(*name*) STOP(SCHD)

オンラインでアクティブに再編成中の HALDB 区画に対してこれらのコマンドを発行すると、IMS はエラー・メッセージ DFS0488I を戻りコード 58 と一緒に表示し、指定の区画に対するコマンドを処理しません。

- HALDB のいずれかの区画のオンライン再編成中には、HALDB マスターに対して以下のコマンドを実行することはできません。
  - /START DATABASE ACCESS UP または UPDATE DATABASE NAME(*name*) START(ACCESS)
  - /DBRECOVERY DATABASE または UPDATE DATABASE NAME(*name*) STOP(ACCESS)
  - /DBDUMP DATABASE または UPDATE DATABASE NAME(*name*) STOP(UPDATES)

関連概念:

757 ページの『HALDB オンライン再編成のための IMS のリモート・サイト・リカバリー処理』

タイプ 2 コマンド環境 (システム管理)

DL/I 分離アドレス・スペースの使用 (システム定義)

関連資料:

DBRC コマンド (コマンド)

HALDB オンライン再編成のデータ・セット命名規則

HALDB 区画のデータ・セットは指定された命名規則を使用します。 HALDB オンライン再編成はこの命名規則を、データ・セットの第 2 のセットを含むよう拡張しています。

HALDB オンライン再編成に加わる区画のデータ・セットは、次の命名規則を使用します: *bbbbbbb.dppppp*

bbbbbbb
HALDB 区画定義ユーティリティーまたは DBRC バッチ・コマンド (INIT.DB、INIT.PART、CHANGE.DB、または CHANGE.PART) を使用して定義した、37 文字以内のデータ・セット名接頭部を表します。1 つの HALDB 区画内のすべてのデータ・セットに、同一のデータ・セット・ベース名が使用されます。

d
HALDB 区画の特定のデータ・セットを一意的に識別する、IMS が割り当てたデータ・セット名タイプ文字を表します。可能な単一文字値は、次のとおりです。

A-J
『A』 はその DBD に定義されている最初の、あるいは唯一の、データ・セット・グループに対応し、『B』 は 2 番目のデータ・セット・グループに対応し、以下同様です。文字 A-J は、HALDB 区画がオンライン再編成可能かどうかにかかわらず、どの HALDB 区画にも一般的に使用されます。

M-V
『M』 はその DBD に定義されている最初の、あるいは唯一の、

データ・セット・グループに対応し、『N』 は 2 番目のデータ・セット・グループに対応し、以下同様です。文字 M-V は、オンライン再編成が可能な HALDB 区画のみに使用されます。

L 間接リスト・データ・セット (ILDS)。オンライン再編成処理は、このデータ・セットのコピーを作成しません。

X PHIDAM データベースの 1 次索引。このデータ・セットは A-J データ・セットの索引で、オンライン再編成処理が A-J および X データ・セットから M-V および Y データ・セットにデータベース・レコードをコピーするとき、Y データ・セットに置き換えられます。文字 X は、HALDB 区画がオンライン再編成可能かどうかにかかわらず、どの HALDB 区画にも一般的に使用されます。

Y PHIDAM データベースの 1 次索引。このデータ・セットは M-V データ・セットの索引で、オンライン再編成処理が M-V および Y データ・セットから A-J および X データ・セットにデータベース・レコードをコピーするとき、X データ・セットに置き換えられます。文字 Y は、オンライン再編成が可能な HALDB 区画のみに使用されます。

ppppp
IMS によって割り当てられた 5 桁の区画 ID を指定します。

出力データ・セットのデータ・セット名は、IMS が割り当てるデータ・セット名タイプ文字 (A-J、M-V、X、または Y) を除き、対応する入力データ・セットの名前と同じです。以下の表はデータ・セット名の例を示しています。

表 *78. HALDB* オンライン再編成のデータ・セット名の例

オンライン再編成前のアクティブ・データ・セット	データ・セット・グループまたは索引	区画 **ID**	**Input data set name**	**Output data set name**
A-J (および X)	1	00003	DH41.A00003	DH41.M00003
A-J (および X)	索引	00065	ACCT.X00065	ACCT.Y00065
M-V (および Y)	2	00005	PAY.MST.N00005	PAY.MST.B00005
M-V (および Y)	8	00001	PAY.EMP.T00001	PAY.EMP.H00001

関連概念:

29 ページの『HALDB 区画、DD 名、およびデータ・セットのための命名規則』

738 ページの『HALDB オンライン再編成』

HALDB オンライン再編成の出力データ・セット要件

HALDB 区画のオンライン再編成の初期設定段階中に、IMS はまだ存在していない出力データ・セットを作成します。

例えば、入力データ・セットが A-J (および X) セットのとき、M および P 出力データ・セットがすでに存在していて N および O 出力データ・セットがまだない場合、IMS は N および O データ・セットを作成し、既存の M および P データ・セットを使用します。

既存の出力データ・セットは、既存の出力データ・セットに関する HALDB の要件に従っている必要があります。既存の出力データ・セットに入っているデータは、オンライン再編成のコピー段階で上書きされます。

既存の出力データ・セットの要件:

既存の出力データ・セットがこのセクションで説明されている要件に合わない場合、IMS はエラー・メッセージを表示し、HALDB 区画のオンライン再編成は開始されません。

OSAM 出力データ・セットには以下の要件があります。

- カタログされていなければならない。
- DASD データ・セットでなければならない。
- PHIDAM データベースの 1 次索引データ・セットを除き、VSAM データ・セットであってはならない。
- PDS、PDSE、あるいは PDS または PDSE のメンバーであってはならない。

VSAM 出力データ・セットには以下の要件があります。

- PHIDAM データベースの 1 次索引データ・セットを除き、VSAM 入力順データ・セット (ESDS) でなければならない。
- REUSE 属性をもたなければならない。
- 対応する入力データ・セットと等しい固定長レコード長をもたなければならない。
- 対応する入力データ・セットと等しい制御インターバル・サイズをもたなければならない。
- データベースが DBRC に対して SHARELVL 属性値 2 または 3 で定義されている場合、対応する入力データ・セットの SHAREOPTIONS 属性値と同じかそれ以上の SHAREOPTIONS 属性値をもたなければならない。

OSAM データ・セットの場合も VSAM データ・セットの場合も、INIT OLREORG OPTION(ALTER) コマンドを使用してデータベースの構造を変更する際にブロック・サイズまたは CI サイズを増やす場合は、以前のオンライン再編成からの既存のデータ・セットを使用することはできません。ALTER オプションが指定されているときに出力データ・セットを手動で割り振る場合は、新規データ・セットが、変更操作のために RECON データ・セットに指定されているブロック・サイズまたは CI サイズの増加に対応できるようにする必要があります。

1 次索引データ・セットには以下の要件があります。

- VSAM キー順データ・セット (KSDS) でなければならない。
- 対応する入力 KSDS と等しいキー・オフセットおよび長さをもたなければならない。
- VSAM 出力データ・セットについてリストされているその他の必須特性をもたなければならない。

関連概念:

739 ページの『HALDB オンライン再編成の初期設定段階』

757 ページの『HALDB オンライン再編成のための IMS のリモート・サイト・リカバリー処理』

自動作成される出力データ・セットの属性:

オンライン再編成の初期設定段階の開始時にまだ存在していない出力データ・セットについては、IMS はデータ・セットを作成します。

IMS は、次の属性でデータ・セットを作成します。

ボリュームの数
: その入力データ・セットが SMS の管理下にある場合、IMS が等しいボリューム数で対応する出力データ・セットを作成します。

  その入力データ・セットが SMS の管理下にない場合、入力データ・セットが単一ボリューム上にある場合のみ、IMS が自動的に対応する出力データ・セットを作成します。 SMS で管理されておらず、複数ボリューム上にある入力データ・セットについては、ユーザーがオンライン再編成を開始する前に対応する出力データ・セットを作成しなければなりません。

SMS の管理下にある出力データ・セットの場所
: その入力データ・セットが SMS の管理下にある場合、対応する出力データ・セットも SMS で管理され、入力データ・セットと同じストレージ・クラスを使用します。

  サイトのストレージ管理者は、このストレージ・クラスが出力データ・セットを保持できる十分なスペースをもったストレージ・グループを参照していること、または自動クラス選択 (ACS) ルーチンがそのデータ・セットに適したストレージ・クラスを選択することを、確認しなければなりません。

SMS の管理下にない非 **VSAM** 出力データ・セットの場所
: 入力データ・セットがある DASD のタイプとは無関係に、IMS は DD ステートメントの `UNIT=SYSALLDA` パラメーターと同等のものを使用して、対応する非 VSAM 出力データ・セットを作成します。

  IMS は出力データ・セットを作成する際に特定のボリューム通し番号を要求しないため、ストレージ・ボリューム上に、ストレージ・ボリュームが使用可能でない場合は公用ボリューム上に、データ・セットを作成することができます。

SMS の管理下にない **VSAM** 出力データ・セットの場所
: IMS は VSAM 出力データ・セットを、対応する入力データ・セットと同じボリューム上に作成します。この制約事項により、SMS の管理下にない VSAM データ・セットの自動作成の便利さが限られることがあります。

出力データ・セットのフォーマット **(DSNTYPE)**
: IMS は、入力データ・セットのフォーマット・タイプを出力データ・セットにも使用します。入力データ・セットが OSAM のラージ・フォーマット順次データ・セットと定義されている場合 (DSNTYPE=LARGE)、IMS は自動的に出力データ・セットをラージ・フォーマット・データ・セットと定義します。

単一ボリューム上の出力データ・セットのサイズ
: 入力データ・セットのエクステントが 1 つの DASD ボリュームのみにある場合、IMS は DD ステートメントの `VOLUME=(,,,1)` パラメーターと同等のものを使用して、単一ボリューム上に出力データ・セットを作成します。

出力データ・セットの 1 次スペースの量は、入力データ・セットのスペース割り振りから導き出されます。

- 非 VSAM データ・セットの場合、1 次スペースはボリューム上の最初の 5 つのエクステントにあるスペースの合計量です。
- VSAM データ・セットの場合、1 次スペースは入力データ・セット作成時に使用された 1 次スペース割り振りになります。

入力データ・セットに 2 次スペースの量を指定した場合には、IMS はこれと同じ 2 次スペースの量を出力データ・セットにも使用します。

入力データ・セットに予約されていたスペースの量とおよそ同じ量のスペースを出力データ・セットにも予約するために、使用されている DASD タイプとは関係なく、IMS は出力データ・セットのスペースを OSAM ブロック数または VSAM レコード数で要求します。 OSAM ブロック数または VSAM レコード数を指定しなかった入力データ・セットについては、IMS はシリンダーまたはトラックの割り振りを等価のブロック数またはレコード数に変換します。

自動作成された出力データ・セットの場合、使用可能な DASD スペースの量が入力データ・セットで使用されていた量とはかなり異なることがあります。例えば、2 次割り振りを使用していた入力データ・セットの場合、自動作成処理は出力データ・セットに 1 次スペースを予約しますが、オンライン再編成の間またはそれより後のデータベース処理の間にはボリューム上に 2 次割り振り用の十分なスペースがないこともあります。

複数ボリューム上の出力データ・セットのサイズ **(SMS** の管理下にある場合のみ**)**

IMS は、入力データ・セットが (したがって出力データ・セットも) SMS の管理下にある場合のみ、複数ボリュームの出力データ・セットを自動作成します。入力データ・セットまたはサイトの ACS ルーチンを調べることによって、ストレージ・クラスを判別することができます。

SMS の管理下にある複数ボリューム出力データ・セットの場合、厳密な要件ではありませんが、ストレージ・クラスで保証スペース属性が指定されていることを確認します。保証スペース属性を指定することにより、VSAM が出力データ・セットを作成する時点で、それぞれのボリュームごとに 1 次スペース割り振りを使用できるようにします。 2 次スペースは必要に応じて使用されます。ただし、保証スペース属性があっても、出力データ・セットには入力データ・セットと同じ量のスペースがないことがあります。特に、入力データ・セットに 2 次スペース割り振りが使用された場合はそうです。

要求される 1 次および 2 次スペースは、最初の DASD ボリューム上での入力データ・セットのスペース割り振りに基づいています。

出力データ・セットのブロック・サイズまたは制御インターバル **(CI)** サイズ

データベースの構造を変更しない限り、IMS がオンライン再編成のために作成する各出力データ・セットのブロック・サイズまたは制御インターバル・サイズは、対応する各入力データ・セットのそれぞれのサイズと同じです。

RECON データ・セットの 1 つ以上のデータ・セット・グループに対して ALTERSZE 値を指定しており、INIT OLREORG コマンドの ALTER オプ

ションを使用してデータベースの構造を変更すると、IMS は、ALTERSZE 値が指定されたデータ・セットについて、その ALTERSZE に一致するブロック・サイズまたは CI サイズの出力データ・セットを作成します。

ALTER オプションを使用すると、オンライン再編成が HALDB データベース全体に適用されます。INIT OLREORG コマンドを実行する前に、変更する必要があるブロック・サイズまたは CI サイズが含まれている各区画について、ALTERSZE 値を個々に設定する必要があります。

データベースの構造を変更する再編成を行う前に ALTERSZE 値を設定するには、DBRC コマンド CHANGE.PART を使用します。

VSAM データ・セットの場合、ALTERSZE サイズを指定した場合に、その ALTERSZE に一致しない CI サイズの出力データ・セットが既に存在していると、その区画に対する再編成は失敗します。この場合は、ALTERSZE 値と CI サイズの相違を訂正し、データベース内の他のすべての区画に対して変更処理が完了した後、OPTION(ALTER) を指定して INIT OLREORG コマンドを再発行する必要があります。ALTERSZE 値と CI サイズの相違を訂正する方法の 1 つとしては、変更機能が自動的に新しい CI サイズでデータ・セットを作成できるように、既存の VSAM データ・セットを削除する方法があります。

OSAM データ・セットの場合、出力データ・セットが既に存在している場合でも、変更機能はブロック・サイズを変更することができます。

HALDB オンライン再編成の開始

HALDB オンライン再編成を開始するには、INITIATE OLREORG コマンドのいずれかのバージョンを使用します。

HALDB 区画をオンラインで再編成するには、事前に INIT.DB OLRCAP または CHANGE.DB OLRCAP のいずれかの DBRC コマンドを発行して、HALDB マスター・データベースをオンライン再編成用に使用可能にしておく必要があります。

以下の表は、HALDB 区画のオンライン再編成を開始または再開するタスクとコマンドを説明しています。

表 *79. HALDB* オンライン再編成の始動タスクのコマンドへのマッピング

タスク	タイプ **1** コマンド・フォーマット	タイプ **2** コマンド・フォーマット
1 つ以上の区画について HALDB オンライン再編成を開始します。	/INITIATE OLREORG	INITIATE OLREORG
すべての区画にわたって HALDB オンライン再編成を開始して HALDB データベースの構造を変更します。	サポートされていません	INITIATE OLREORG OPTION(ALTER)
1 つ以上の区画について HALDB オンライン再編成を再開します。	/INITIATE OLREORG	INITIATE OLREORG

表 79. *HALDB* オンライン再編成の始動タスクのコマンドへのマッピング (続き)

タスク	タイプ **1** コマンド・フォーマット	タイプ **2** コマンド・フォーマット
HALDB オンライン再編成に RATE を設定します。	/INITIATE OLREORG SET(RATE(*rate*)) または /UPDATE OLREORG SET(RATE(*rate*))	INITIATE OLREORG SET(RATE(*rate*)) または UPDATE OLREORG SET(RATE(*rate*))

関連資料:

INITIATE OLREORG コマンド (コマンド)

HALDB オンライン再編成のモニター

いくつかのコマンドを使用して、HALDB オンライン再編成をモニターすることができます。

以下の表 は、HALDB 区画のオンライン再編成をモニターするタスクとコマンドを説明しています。

表 80. *HALDB* オンライン再編成のモニター・タスクのコマンドへのマッピング

タスク	コマンド
進行中の HALDB オンライン再編成について状況および速度情報を表示します。	QUERY OLREORG タイプ 2 コマンド
ALTER オプションが指定されている場合に、HALDB オンライン再編成についての状況および速度情報を表示します。	QUERY OLREORG STATUS(ALTER) タイプ 2 コマンド
指定されたデータベースまたは区画 (進行中の HALDB オンライン再編成を含む) の状況をモニターし、表示します。	/DISPLAY DB OLR タイプ 1 コマンド
HALDB オンライン再編成の状況を表示します。	QUERY DB タイプ 2 コマンド
HALDB オンライン再編成が進行中のデータベースをすべてリストします。	QUERY DB STATUS(OLR) タイプ 2 コマンド

HALDB オンライン再編成の変更およびチューニング

HALDB オンライン再編成の変更とチューニングでは、主に、再編成が実行される速度を変更します。

以下の表は、HALDB 区画のオンライン再編成を変更およびチューニングするタスクとコマンドを説明しています。

表 *81. HALDB* オンライン再編成の変更およびチューニング・タスクのコマンドへのマッピング

タスク	タイプ **1** コマンド	タイプ **2** コマンド
1 つ以上の区画について、HALDB オンライン再編成のシステム全体のパフォーマンスへの影響を変更します。	/UPDATE OLREORG SET(RATE(*rate*))	UPDATE OLREORG SET(RATE(*rate*))
コピー段階の完了後に非アクティブ・データ・セットを削除するかどうか指定します。	/UPDATE OLREORG OPTION(DEL | NODEL)	UPDATE OLREORG OPTION(DEL | NODEL)

データベースが共用されている場合、そのデータベースを共用するすべての IMS システムに送信できるのはタイプ 2 コマンドのみです。タイプ 1 コマンドは、そのコマンドが発行されたシステムにのみ適用されます。

OPTION(ALTER) を指定した INITIATE OLREORG コマンドによって再編成が開始されると、データベース内のすべての区画が再編成されます。この場合に、データベース内のすべての区画のオンライン再編成を変更または調整する必要がある場合は、例えば UPDATE OLREORG NAME(*) SET(RATE(*rate*)) のように、NAME キーワードにアスタリスクを指定します。

関連資料:

UPDATE コマンド (コマンド)

HALDB オンライン再編成の停止

1 つ以上の区画に対して TERMINATE OLREORG コマンドを発行することにより、HALDB オンライン再編成を停止することができます。

IMS タイプ 1 コマンド /TERMINATE OLREORG、またはタイプ 2 コマンド TERMINATE OLREORG のいずれかのコマンドを発行できます。

データベースが共用されている場合、そのデータベースを共用するすべての IMS システムに送信できるのはタイプ 2 コマンドのみです。タイプ 1 コマンドは、そのコマンドが発行されたシステムにのみ適用されます。

OPTION(ALTER) を指定した INITIATE OLREORG コマンドによって再編成が開始されると、データベース内のすべての区画が再編成されます。この場合に、データベース全体にわたって再編成を停止する必要がある場合は、NAME キーワードにアスタリスクを指定します。例えば、 TERMINATE OLREORG NAME(*) のようになります。

例: 以下の図は、HALDB 区画のオンライン再編成の処理ステップと、それが再編成を一時的に停止させる TERMINATE OLREORG コマンドによってどのような影響を受けるかを示しています。

- TERMINATE OLREORG コマンドを実行すると、IMS は終了段階に入って再編成を終了します。
- その後 INITIATE OLREORG コマンドを実行すると、IMS は初期設定段階から再編成を再始動し、コピー段階に進みます。図では、その後再編成は終了段階まで正常に完了します。

2 回目の初期設定段階では、再編成が完了していないため、2 セットのデータ・セットがあることに注意してください。

図では、各列はオンライン再編成の各段階を通したユーザーから IMS への制御のフロー、および処理イベント発生時のデータ・セットの状況を示しています。

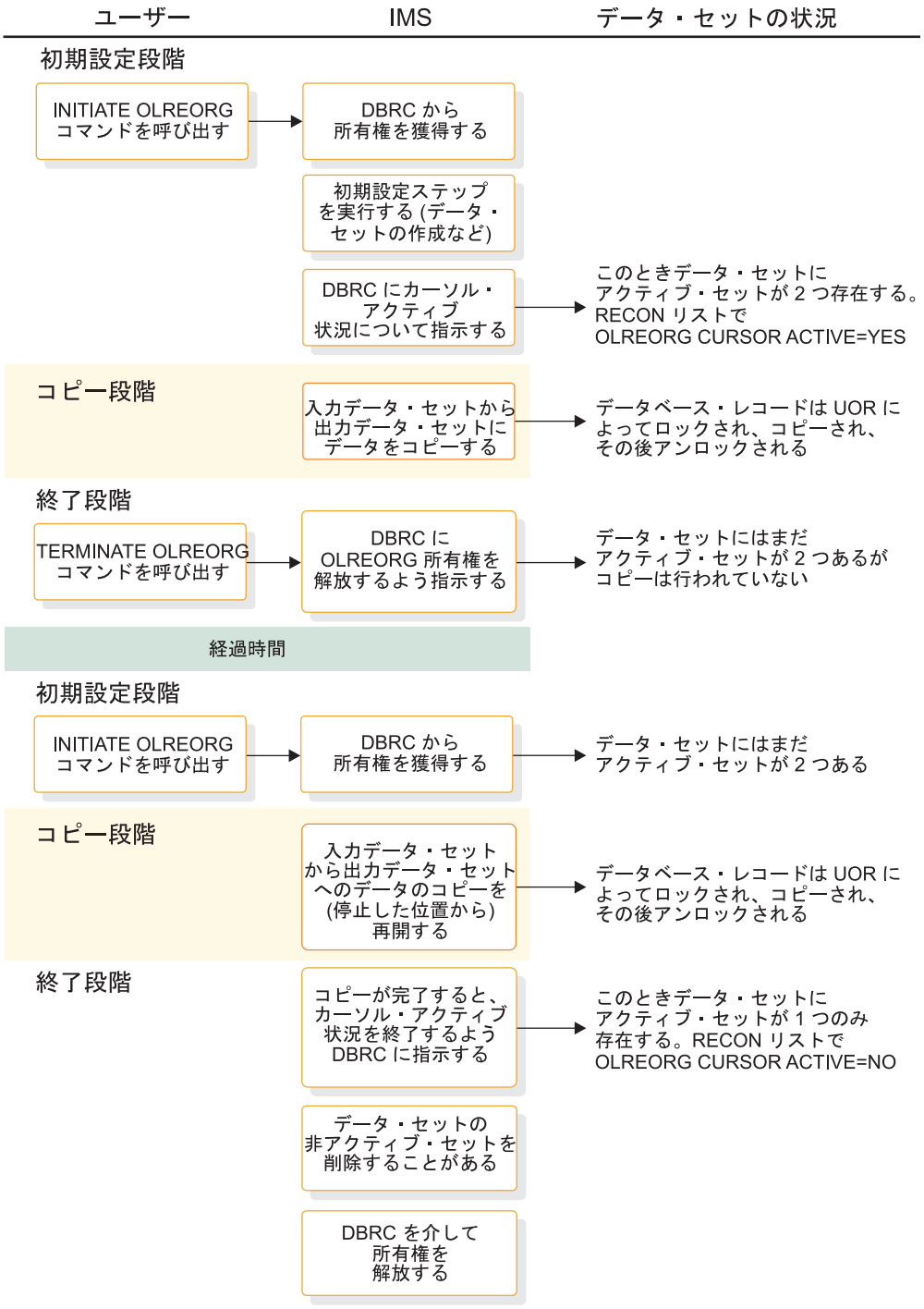


図 *271.* 中断された *HALDB* 区画オンライン再編成の処理ステップ

関連タスク:

860 ページの『変更処理の完了前での停止』

関連資料:

TERMINATE コマンド (コマンド)

HALDB オンライン再編成の IMS ロギングへの影響

HALDB 区画のオンライン再編成は、その区画内のすべてのデータについて、X'50' のデータベース変更ログ・レコードを生成します。

IMS はまた、その他の HALDB オンライン再編成情報を少数の X'29' のログ・レコードに記録します。そのため、HALDB オンライン再編成によって生成されるログ・データの合計量は、区画内のデータの量よりかなり大きくなります。

オンラインで区画を再編成すると、特に同じ IMS システム上で並列的に複数の区画を再編成すると、通常のトランザクション処理に影響を与えるほど多くのログ・データが生成されることがあります。生成される大量のログ・レコードは、OLDS 切り替えおよびログ保存の速度にも影響することがあります。

HALDB オンライン再編成のシステム全体に対する影響の制御

HALDB 区画のオンライン再編成は、IMS システムの全体的なシステム・パフォーマンスに影響を与えることがあり、同様に、他の IMS 作業がオンライン再編成のパフォーマンスに影響を与えることがあります。

オンライン再編成が IMS システムに与える影響を制御するためのオプションには、以下のものがあります。

- IMS.PROCLIB データ・セットの DFSVSMxx メンバーで DBD ステートメントを使用して、再編成されるデータ・セットに独立したバッファー・サブプールを指定する。
- INITIATE OLREORG および UPDATE OLREORG コマンドの RATE パラメーターを使用して、オンライン再編成が実行される速度を調整する。

再編成されるデータ・セットに独立したバッファー・サブプールを指定すると、オンライン再編成機能と、同じくバッファー・リソースを必要とする他の処理との間で生じるバッファーの競合を軽減できます。

また、RATE パラメーターの値を調整して、オンライン再編成処理に周期的な遅延を意図的に取り入れることで、他の IMS 処理の進行が促され、オンライン再編成が IMS システムに与える影響を最小限に抑えられます。

RATE 値はパーセントで指定し、意図的に導入する遅延の追加を表す 100 より小さい値を使用します。

RATE パラメーターのデフォルト値は 100 で、システム・リソース、システム競合、およびログ競合に応じ、意図的な遅延を導入せずにできるだけ速くオンライン再編成を実行させます。しかし、例えば RATE 値を 25 に設定すると、IMS は再編成処理に遅延を追加して、再編成単位の合計処理時間のうち 25% をデータのコピーに費やし、残る 75% を意図的に導入する遅延に費やします。したがって、RATE(25) を指定した場合、オンライン再編成の所要時間は RATE(100) で実行する場合のおよそ 4 倍になります。

RATE 値は、UPDATE OLREORG コマンドの実行によっていつでも変更することができます。

HALDB オンライン再編成のための IMS の再始動と XRF 処理

HALDB 区画のオンライン再編成を実行中に IMS をシャットダウンすると、IMS はシャットダウン・チェックポイントの完了前に再編成を中断します。

IMS の再始動後、IMS は自動的にオンライン再編成を再開します。 IMS は、DFSVSMxx メンバーに NOPDBO オプションを指定してある場合でもオンライン再編成を再開します。

オンライン再編成の実行中に IMS が異常終了すると、IMS はこれらの再編成のコミットされていない変更をすべて、最新の同期点まで動的にバックアウトします。IMS の再始動後、IMS は自動的にオンライン再編成を再開します。

XRF のテークオーバーが発生すると、IMS は新規のアクティブ IMS システム上でオンライン再編成を自動的に再開します。

INITIATE OLREORG または UPDATE OLREORG のいずれかのコマンドの OPTION(REL) パラメーターを使用するか、または IMS.PROCLIB データ・セットの DFSDFxxx メンバーで RELOLROWNER=Y をコーディングして、IMS システムに対し、再編成が完了する前にシステムが (正常または異常を問わず) 終了した場合に所有権を解除するよう指示することもできます。これらのオプションのいずれかを使用すると、中断状態の再編成を別の IMS システムで再開することができます。

関連資料:

INITIATE OLREORG コマンド (コマンド)

HALDB オンライン再編成のための IMS の再始動と高速データベース・リカバリー処理

FDBR テークオーバーが行われると、IMS はこれらの再編成のコミットされていない変更をすべて最新の同期点までバックアウトし、区画をクローズし、区画を無許可にします。

FDBR が終了した後、障害を起こした IMS システムは、もはやオンライン再編成の所有権を持ちません。オンライン再編成を再開するには、別の IMS システムで、または障害を起こした IMS システムを再始動した後に、INITIATE OLREORG コマンドを発行します。

HALDB オンライン再編成のための IMS のリモート・サイト・リカバリー処理

リモート・サイト・リカバリー (RSR) のトラッキング中、IMS はトラッキング・サイトの RECON データ・セットを HALDB オンライン再編成に関する情報で更新します。

データベース・レベル・トラッキング (DLT) IMS トラッキング・システムでデータベース・レベル・トラッキング (DBTRACK) として登録されている HALDB 区画の場合、IMS はオンライン再編成のトラッキング中に以下のステップを実行します。

- 必要に応じて、シャドー区画用の出力データ・セットを作成します。
- シャドー区画の入力データ・セットおよび出力データ・セットの両方を更新します。
- オンライン再編成のトラッキング完了時に、オリジナルの入力データ・セットに非アクティブのマークを付け、出力データ・セットにアクティブ・データ・セットのマークを付けます。
- 削除オプションが有効であれば、非アクティブ・データ・セットを削除します。このオプションは、アクティブ・サイトで INITIATE OLREORG コマンドの OPTION キーワードを使用して指定します (あるいはデフォルトを受け入れます)。

出力データ・セットの検証または作成中にエラーが発生すると、IMS はシャドー区画を停止します。アクティブ・サイトでトラッキングされている区画は、トラッキング・サイトのエラーによって影響を受けることはありません。エラーの原因となった問題を訂正した後、トラッキング IMS システム上でシャドー区画を再始動して区画のオンライン順方向リカバリー開始し、トラッキングを続けます。

トラッキングの開始前にオンライン再編成用の出力データ・セットがすでにトラッキング・サイトにある場合は、それらのデータ・セットの特性 (ブロック・サイズ、レコード・サイズ、制御インターバル・サイズなど) がアクティブ・サイトの特性と同じであることを確認してください。アクティブ・サイトで出力データ・セットの特性を手動で変更する場合、トラッキング・サイトでも同じ変更を行わなければなりません。

RSR テークオーバーの後、IMS は、オンライン再編成を処理中の区画も含めたすべての HALDB 区画を停止します。新規アクティブ・サイトで HALDB 索引/ILDS 再作成ユーティリティー (DFSPREC0) を使用して 1 次索引および間接リスト・データ・セットを再作成した後、必要があれば INITIATE OLREORG コマンドを実行してオンライン再編成を再開します。テークオーバーの後、オンライン再編成が自動的に再始動することはありません。

関連概念:

749 ページの『既存の出力データ・セットの要件』

745 ページの『HALDB オンライン再編成の制約事項』

HALDB オンライン再編成のロッキングの影響

HALDB 区画のオンライン再編成中の区画の保全性とリカバリー可能性を維持するために、HALDB オンライン再編成は、UOR ごとにアクティブ・ロック・マネージャー (プログラム分離ロック・マネージャーまたは内部リソース・ロック・マネージャー (IRLM) コンポーネント) に非常に多くのロックを要求します。

IRLM がロック・マネージャーである場合、特に RATE(100) を指定して再編成を実行している場合には、オンライン再編成での 1 秒当たりの IRLM ロック構造へのアクセスがかなりの回数にのぼる可能性があります。また、IRLM が使用されている場合にデータベースが複数の IMS システムによって共用されていると、オンライン再編成機能は、アプリケーション・プログラムのように、グローバル・ロックを要求します。

推奨事項:

- カップリング・ファシリティー構成が、オンライン再編成によって起こる余分な負荷を処理できることを確認するために、IBM System z CFSizer (Coupling Facility Structure Sizer Tool) を使用して追加のカップリング・ファシリティー・アクティビティーをモデル化してください。
  - PC=NO が指定された IRLM 2.1 では、並行して保持されるロックが 1000 回追加になるごとに、256 KB の ECSA ストレージが必要です。
  - IRLM 2.2 では、並行して保持されるロックが 1000 回追加になるごとに、IRLM 専用ストレージから 540 KB を獲得する必要があります。 ECSA ストレージの増加は不要です。
- LOGL ラッチ競合率、OLDS ロギング速度、IRLM ロック構造アクセス、および DBBP (OSAM 用) ラッチ競合を検討してください。

CFSizer は、www.ibm.com/servers/eserver/zseries/cfsizer/ の Web サイトで入手できます。あるいは、IBM Web サイト (www.ibm.com) で「CFSizer」を検索してください。

HALDB オンライン再編成での IMS ユーティリティーの使用

IMS ユーティリティーのバッチ・バックアウト (DFSBBO00)、データベース変更累積 (DFSUCUM0)、データベース・イメージ・コピー (DFSUDMP0)、データベース・リカバリー (DFSURDB0)、および 1 次索引/ILDS 再作成 (DFSPREC0) には、HALDB オンライン再編成とともに使用する場合の特別な考慮事項があります。

バッチ・バックアウト **(DFSBBO00)**

再編成単位の動的バックアウトが失敗に終ると、IMS は RECON データ・セット内に動的 PSB 名を含んだバックアウト・レコードを作成します。リストされた PSB 名について、バッチ・バックアウト・ユーティリティー (DFSBBO00) を実行してください。

動的バックアウトが試みられなかった場合は、PSB オプションを指定したログ・リカバリー・ユーティリティー (DFSULTR0) を使用して、バックアウトを必要とする PSB をリストしてください。バックアウトを必要とする HALDB 区画の経過中オンライン再編成作業がある場合は、ログ・リカバリー・ユーティリティーは動的 PSB 名をリストします。リストされた PSB 名のそれぞれについて、バッチ・バックアウト・ユーティリティー (DFSBBO00) を実行してください。

データベース変更累積 **(DFSUCUM0)**

データベース変更累積ユーティリティー (DFSUCUM0) を使用して、HALDB 区画 A-J データ・セットおよび M-V データ・セットの変更を累積することができます。 DB0 制御ステートメントで、データベース変更累積ユーティリティーが、オンライン再編成開始前の変更を累積するか消去するかを指定する必要があります。

データベース変更累積ユーティリティーは、データベース変更累積の開始時点でオンライン再編成チェックポイントが完了していない場合、対応する A-J および M-V データ・セットにデータベース変更累積ヘッダー・レコード (タイプ X'25' レコード) を作成します。

データベース・イメージ・コピー **(DFSUDMP0)**

データベース・イメージ・コピー・ユーティリティー (DFSUDMP0) を使用

して、RECON データ・セットに記録されている現在のアクティブ・データ・セットをコピーすることができます。データベース・イメージ・コピー・ユーティリティーはまた、M-V データ・セットまたは A-J データ・セットをコピーすべきかどうかも決定します。ただし、区画がカーソル・アクティブの状況にあると、その区画に対してデータベース・イメージ・コピー・ユーティリティーを実行することはできません。

HALDB 区画データ・セットをコピーする場合、それらは動的に割り振られるため、JCL で DD ステートメントをコーディングする必要はありません。

データベース・リカバリー **(DFSURDB0)**

データベース・リカバリー・ユーティリティー (DFSURDB0) は出力データ・セットが存在することを予期し、それらを作成しようと試みることはありません。

1 次索引および **ILDS** 再作成 **(DFSPREC0)**

HALDB 索引/ILDS 再作成ユーティリティー (DFSPREC0) を使用すると、入力データ・セットおよび出力データ・セットの両方について、1 次索引データ・セットをリカバリーすることができます。また HALDB 索引/ILDS 再作成ユーティリティーを使用して、X および Y 1 次索引データ・セットと ILDS の両方を 1 回の実行でリカバリーすることもできます。特定の入力データ・セットまたは出力データ・セットを指定することはできませんが、区画がカーソル・アクティブ状況にある場合は、ユーティリティーが必要なデータ・セットをすべて割り振って再作成します。

関連タスク:

『HALDB オンライン再編成のリカバリー』

関連資料:

データベース・イメージ・コピー 2 ユーティリティー (DFSUDMT0) (データベース・ユーティリティー)

リカバリー・ユーティリティー (データベース・ユーティリティー)

HALDB オンライン再編成のリカバリー

DBRC が RECON データ・セット内の区画にカーソル・アクティブ状況を設定した後、コピー段階が完了して DBRC がカーソル・アクティブ状況をリセットするまでの間、データベース・リカバリー・ユーティリティー (DFSURDB0) を使用してどの入力または出力データ・セットでもリカバリーすることができます。

出力データ・セットを復元するために、データベース・リカバリー・ユーティリティーはデータベース変更レコード (タイプ X'50' ログ・レコード) を使用し、それらを空の出力データ・セットに適用します。

推奨: オンライン再編成の完了後、できるだけ早く出力データ・セットのイメージ・コピーを作成してください。このイメージ・コピーからのリカバリーは、オンライン再編成中にログに記録されるデータベース変更レコードからのリカバリーより高速です。ただし、区画がカーソル・アクティブ状況にある間は、イメージ・コピーを作成することはできません。

オンライン再編成が完了する前に出力データ・セットをリカバリーするには、次のタスクを実行します。

1. TERMINATE OLREORG コマンドを使用してオンライン再編成を停止します。オンライン再編成が異常終了を検出した場合は、自動的に停止します。
2. HALDB 区画に対して /DBR または UPDATE DB コマンドを実行します。
3. 必要に応じて、データベース変更累積を実行します。 GENJCL.CA コマンドを実行して JCL を作成するか、独自の JCL からデータベース変更累積ユーティリティー (DFSUCUM0) を実行することができます。データ・セットの初期空白状態から復元することを表すために、変更累積の消去時刻は、オンライン再編成の開始時刻と等しくなければなりません。
4. JCL DD ステートメントを使用するかアクセス方式サービス・プログラムを使用して、リカバリーする出力データ・セットを作成します。
5. データベース変更をリカバリーします。 GENJCL.RECOV コマンドを実行して JCL を作成することができます。あるいはそれに代えて、復元の元になるイメージ・コピーがないことを示すために DUMMY に指定した DFSUDUMP の DD ステートメントで、独自の JCL からデータベース・リカバリー・ユーティリティー (DFSURDB0) を実行することができます。
6. コミットされていないデータをバックアウトする必要があると思われるため、バッチ・バックアウト・ユーティリティー (DFSBBO00) を実行します。
7. HALDB 区画の必要なデータ・セットをすべてリカバリーし、また該当する場合はバックアウトもした後、その HALDB 区画に対して /STA DB または UPDATE DB コマンドを実行します。
8. INITIATE OLREORG コマンドを実行してオンライン再編成を再開します。

また、オンライン再編成の完了後、イメージ・コピーが作成されるより前に、出力データ・セットをリカバリーすることもできます。その場合は、オンライン再編成を停止して再始動するステップを除いて、オンライン再編成完了前の出力データ・セットのリカバリーと同じステップに従ってください。

さらに、オンライン再編成を完了したか強制終了したかにかかわらず、既存のプロシージャーを使用してオンライン再編成の開始時以外のポイント (例えば DASD ボリュームのフル・ダンプなど) から、出力データ・セットをリカバリーすることができます。

データベース変更累積ユーティリティーの消去時刻の指定

出力データ・セットの 1 つに対してデータベース変更累積ユーティリティー (DFSUCUM0) を実行する場合は、オンライン再編成の開始時刻と等しい消去時刻を指定します。

この消去時刻の指定は、出力データ・セットを含む変更累積レコード (または入力ログ) がオンライン再編成の開始時刻にまでわたっている場合に必要となります。消去時刻の指定により、この時刻以前のデータベース変更レコードを除去することができ、それはイメージ・コピーの開始時刻以前のデータベース変更レコードを除去するのと類似しています。

GENJCL.CA および **GENJCL.RECOV** コマンドの開始点の指定

出力データ・セットに対するイメージ・コピーが存在しなくても、RECON データ・セットは開始点としてオンライン再編成の先頭を反映しており、オンライン再編成の完了後でも、そこからこれらのデータ・セットの順方向リカバリーを実行することができます。ユーザーが出力データ・セットのイメージ・コピーを作成するまで、GENJCL.CA コマンドはこの開始点を最新のイメージ・コピーかのように取り扱い、出力データ・セットへの変更をその点から累積します。同様に GENJCL.RECOV コマンドは、物理的なイメージ・コピーが存在しない場合でも、この点から出力データ・セットのリカバリーを作成します。

データベース・イメージ・コピー・ユーティリティーのアクティブ・データ・セットの指定

データベース・イメージ・コピー・ユーティリティーは常に、RECON データ・セットに記録されている現在のアクティブ・データ・セットからコピーを実行します。 A-J または M-V データ・セットのどちらがアクティブかに関係なく、これらのユーティリティーの JCL または制御ステートメントを変更してデータ・セットのどのセットを使用するか指定する必要はありません。

データベース・イメージ・コピー・ユーティリティー (DFSUDMP0) のユーティリティー制御ステートメントで、 DDNAME が現在のアクティブ・データ・セットを参照している必要はありません。 A-J または M-V データ・セットのどちらがアクティブかに関係なく、ユーティリティーは自動的に現在のアクティブ・データ・セットを使用します。

例: DBD で定義されている 2 番目のデータ・セット・グループのデータ・セットをコピーし、区画名は PARTNO3 であるものとします。どちらのセットのデータ・セットがアクティブなのかに関係なく、制御ステートメントでは DDNAME を PARTNO3B または PARTNO3N のいずれかにコーディングすることができます。A-J のデータ・セットがアクティブであれば、PARTNO3B または PARTNO3N のどちらを指定しても、ユーティリティーは PARTNO3B からコピーします。同様に、M-V データ・セットがアクティブであれば、ユーティリティーは PARTNO3N からコピーします。

データベース・イメージ・コピー・ユーティリティーの JCL ステートメントでは、入力データ・セットを参照する DD ステートメントを省略する必要があります。A-J または M-V データ・セットのどちらがアクティブかに基づいて、ユーティリティーは動的に適切なデータ・セットを割り振ります。特定の DD データ・セット名を参照する DD ステートメントは、ジョブ・ステップの開始時に **"データ・セットが見つからない"** という条件の発生によりユーティリティー・ジョブが失敗に終る原因となります。この条件は、JCL に非アクティブ・データ・セット名がコーディングされ、そのデータ・セットが存在しない場合に発生します。

カーソル・アクティブ状況にあるときの **1** 次索引と **ILDS** の再作成

統合 HALDB オンライン再編成機能が完了前に停止して、区画がカーソル・アクティブ状況のままになっていても、HALDB 索引/ILDS 再作成ユーティリティー (DFSPREC0) を使用して 1 次索引データ・セット X および Y と ILDS の両方を再作成できます。

1 次索引を再作成する必要がある場合は、DFSPREC0 ユーティリティーによって X と Y の両方のデータ・セットが再作成されます。これは、区画内のレコードがデータ・セットの入力セットと出力セットに分かれているためです。

関連概念:

759 ページの『HALDB オンライン再編成での IMS ユーティリティーの使用』

HALDB のオンライン再編成後のオフライン再編成

HALDB オンライン再編成を行っても、オフライン再編成が実行できなくなることはありません。

オンライン再編成が進行中である場合は、TERMINATE OLREORG コマンドを実行してオンライン再編成を停止し、さらに /DBRECOVERY コマンドまたは UPDATE DB NAME(*partition_name*) STOP(ACCESS) コマンドを実行して区画へのアクセスを停止した後に、オフライン再編成を実行できます。 HD 再編成アンロード・ユーティリティー (DFSURGU0) は、A-J データ・セットと M-V データ・セット両方のアクティブ部分からレコードを自動的にアンロードします。 HD 再編成再ロード・ユーティリティー (DFSURGL0)は A-J データ・セットに全レコードをロードし、RECON データ・セットの中のオンライン再編成のカーソル・アクティブ状況をオフにします。

オンライン再編成が進行中でない場合は、特別なステップを追加で実行することなく、HALDB データベースのオフライン再編成を実行できます。

関連概念:

856 ページの『HALDB の変更およびオフライン再編成』

関連タスク:

731 ページの『HALDB のオフライン再編成』

HALDB オンライン再編成のパフォーマンスを向上させるための順次バッファリングの活動化

IMS.PROCLIB データ・セット・メンバー DFSVSMxx に SBONLINE ステートメントを組み込むことにより、OSAM データベースのオンライン再編成のパフォーマンスを向上させるために順次バッファリングを使用することができます。

SBONLINE ステートメントを使用すると、IMS は初期設定時に順次バッファリング・モジュールをロードし、OSAM 区画のためにオンライン再編成を開始すると必ず IMS が即時に順次バッファリングを活動化するようになります。 SBONLINE ステートメントを組み込まないと、IMS は DL/I 呼び出しを分析して、順次バッファリングが再編成の処理に適しているかどうかを判別します。

SBONLINE 制御ステートメントの 2 つの形は次のとおりです。

```
SBONLINE
SBONLINE,MAXSB=nnnnn
```

ここで、*nnnnn* は順次バッファーに割り振ることのできる最大のストレージ (K バイト単位) です。

最大のストレージに達すると、IMS は (HALDB オンライン再編成を含む) オンライン・アプリケーションが順次バッファー・スペースを解放するまで、それらのア

プリケーションへの順次バッファーの割り振りを停止します。 MAXSB= キーワードを指定しないと、順次バッファーの最大ストレージに限界はなくなります。

関連概念:

OSAM 順次バッファリングのための指定 (システム定義)

関連資料:

オンライン・システムでの順次バッファリングの指定 (システム定義)

HALDB 自己回復ポインター処理

論理関係および副次索引をもつ HALDB データベースの再編成には、ポインターを更新するユーティリティーの実行は必要ありません。その代わりに、HALDB は自己回復ポインター処理を使用して論理関係および副次索引ポインターを訂正します。

この処理は、副次索引または論理的に関連したデータベースにターゲット・キーと拡張ポインター・セット (EPS) を入れ、PHDAM および PHIDAM データベースの各区画内で間接リスト・データ・セット (ILDS) を使用することによって、インプリメントされます。

関連概念:

196 ページの『HALDB 区画再編成番号』

410 ページの『HALDB 区分副次索引に関する考慮事項』

358 ページの『論理関係と HALDB データベース』

274 ページの『論理関係ポインターの種類』

899 ページの『HALDB 副次索引および論理関係ポインターの自動更新』

関連タスク:

589 ページの『ILDS の割り振り』

自己回復ポインター処理の動作方法

自己回復ポインター処理のエレメントは以下の図で見ることができ、これはデータベース再編成前のエレメント間の相互関係を示しています。

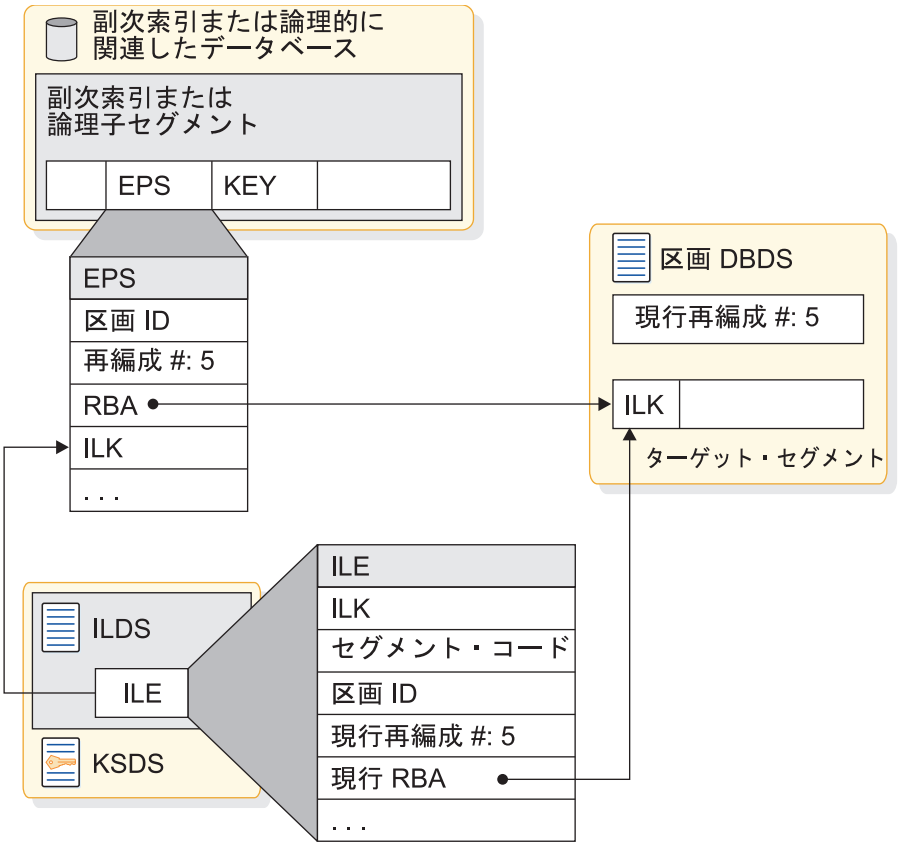


図 272. 再編成前の *HALDB* ポインター

各副次索引項目および各論理子セグメントには、そのターゲット・レコードのキーが入っています。副次索引の場合、ターゲットのルート・セグメントのキーが接頭部に含まれています。論理子セグメントの場合、論理親の連結キーがセグメント・データに含まれています。

PHDAM または PHIDAM データベースの各セグメントには、間接リスト・キー (ILK) があります。 ILK はデータベース全体を通して、セグメント・タイプに固有です。以下の図に示すように、セグメントが最初に作成されたときのセグメントの相対バイト・アドレス (RBA)、区画 ID、および区画再編成番号で構成されています。セグメントの ILK は変化しません。それは再編成の間ずっと維持されています。

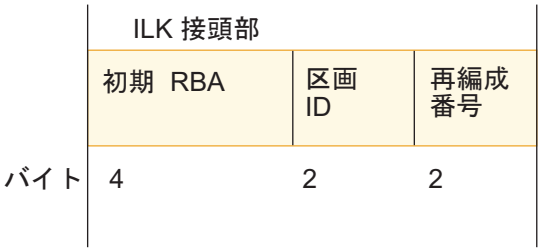


図 273. *ILK* のフォーマット

各副次索引項目または論理子セグメントには、拡張ポインター・セット (EPS) があります。 EPS には、そのターゲット・セグメントの ILK が含まれています。また、RBA、区画 ID、およびターゲット・セグメントの区画再編成番号も含まれています。 EPS のこれらの部分は、正確でない場合があります。すなわち、それらがターゲット・セグメントの現在場所を反映していないか、ターゲット・セグメントの区画の現在の再編成番号を反映していないことがあります。前の図では、それらは正確です。

ターゲット・セグメントには、区画の ILDS 内に間接リスト項目 (ILE) があります。 ILE にはターゲット・セグメントに関する正確な情報が含まれています。ここには、現在の RBA、正しい区画 ID、および区画の現在の再編成番号が入っています。 ILE のキーは、ILK およびターゲット・セグメントのセグメント・コードで構成されています。

区画の再編成番号は、区画の最初のデータベース・データ・セットに物理的に保管されています。この番号は区画の初期設定またはロードによって初期設定され、その区画内のセグメントを再ロードする再編成が 1 回行われるごとに 1 ずつ増加します。

ターゲット・セグメントの検出

IMS が副次索引項目または論理子セグメントのターゲット・セグメントにアクセスする場合、まずそのターゲットがある区画を判別しなければなりません。

IMS は区画を判別する為に、副次索引または論理子の中にあるキーを使用します。次に、ターゲット・パーティション・データベース・データ・セットの中の場所を判別しなければなりません。それには、ターゲット区画の区画 ID および再編成番号を、EPS に保管されている区画 ID および再編成番号と比較します。それらが一致すると、IMS はその EPS 内の RBA を使用して、ターゲット・セグメントの場所を探索します。一致しない場合、その EPS 内の RBA を使用することはできません。

EPS 内の RBA を使用することができない場合、IMS は ILE 内の情報を使用してターゲット・セグメントの場所を探索します。 ILE キーは、EPS およびターゲットのセグメント・コードの ILK を使用して検出されます。 ILE は、ターゲットのキーから判別された区画の ILDS から読み取られます。

以下の図は、EPS 内の RBA を使用できない状態を示しています。この図では、ターゲット区画は EPS が正確だった時点から 3 回再編成されています。ターゲット・セグメントが移動され、区分データ・セット内の再編成番号が更新されています。 EPS には再編成番号としてまだ 5 が入っていますが、区分データ・セット内の再編成番号は 8 です。ILE 内の情報は、HD 再編成再ロード・ユーティリティーによって更新されました。 IMS は EPS 内の ILK を使用して ILE を検出し、ILE 内の RBA を使用してターゲット・セグメントを検出します。

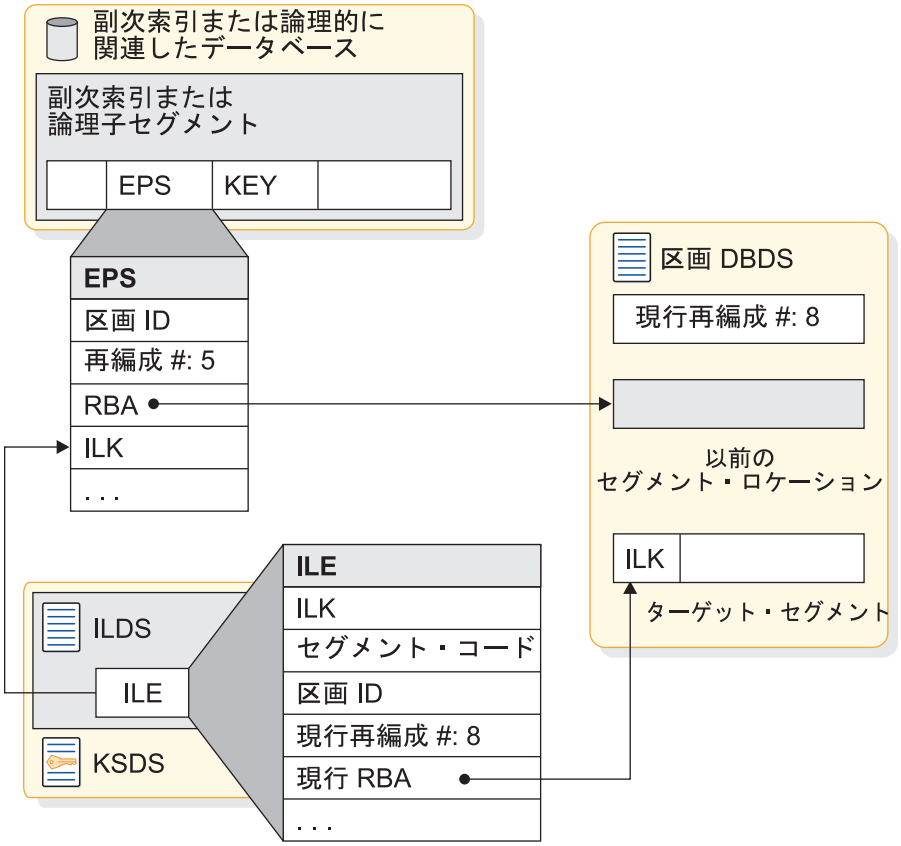


図 *274.* 再編成後の *HALDB* ポインター

検索が間接的でも、多くの場合 ILE を含む CI がすでに IMS のバッファー・プール内にあります。

推奨: 可能な場合、間接処理によってターゲット・セグメントの場所を探索するのは避けてください。その代わりに、ILE を読み取らずに EPS からターゲット・セグメントの場所を求めるようにします。自己修復処理により、IMS に ILE の使用を制限させることができます。

修復ポインター

自己修復処理は、拡張ポインター・セット内の情報を更新または訂正します。

ILE が使用されると、ILE 内のセグメントの現在場所に関する情報は EPS に移されます。これにより、その後の検索に EPS が使用されれば IMS は間接処理を避けることができます。データベース・バッファー・プール内のこのような EPS の修正は、常時実行されます。

ロッキングを考慮して、更新情報が DASD 上のデータベースに書き込まれないことがあります。アプリケーション・プログラムがデータベースの更新を許可されていれば、更新された EPS をもつ項目またはセグメントが入っているバッファーに変更のマークが付けられます。呼び出しは更新を許可する PCB によって行われなければならず、また IMS システムには更新を許可している区画へのアクセス・インテントがなければなりません。更新が許可されなければ、バッファーには変更のマークは付きません。

アプリケーションが同期点に達したとき、バッファーに変更のマークがなければ、バッファーを DASD に書き込みません。更新された EPS が DASD に書き込まれないと、次にそれを DASD から検索してそのターゲットの検出に使用する場合、IMS は間接処理を使用しなければなりません。すなわち、IMS はもう一度 ILE を読み取らなければなりません。

以下の図は修復された後の EPS を示しています。 RBA は現在場所を指しています。区画 ID は正しいものです。区画再編成番号は区画データベース・データ・セットに保管されている番号と一致しています。

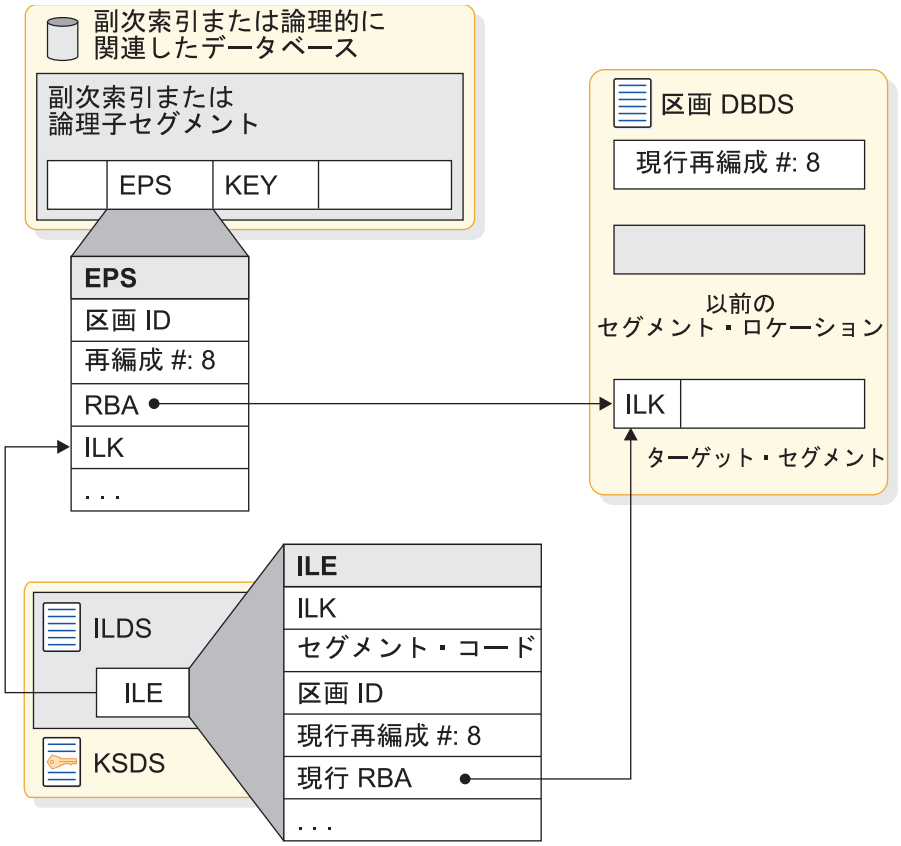


図 *275.* 自己修復処理後の *HALDB* ポインター

自己修復処理のパフォーマンス

自己修復処理のパフォーマンスは、予想よりはるかに効率的にすることができます。

少ない ILDS 読み取り回数で、数多くのポインターを修復することができます。これは IMS データベース・バッファリングを使用しているためです。ILDS はデータベース・データ・セットです。それらは、他のデータベース・データ・セットと同じ方法でデータベース・バッファー・プールを使用します。 CI がすでにそのバッファー・プールにあれば、DASD から読み取る必要はありません。

各 ILE は 50 バイトです。ユーザーが ILDS の CI サイズを指定します。 8 KB の ILDS CI には最大 163 の ILE が入り、16 KB の CI には最大 327 の ILE が

入るため、1 つの CI は多数の ILE を保持することができます。再編成の後、IMS は再編成された区画を指すポインターを数多く修復しなければならないことがあります。

ILDS 内の CI を頻繁に使用すれば、それらはバッファー・プールの中にとどまる傾向があります。数 100 個のポインターを修復するのに、ILDS CI を 1 回読み取るだけで十分なことがあります。ほとんどの IMS データベース・チューニングと同様、頻繁に使用されるデータ・セットには多数のバッファーを用意すると、大きな利点があります。

自己修復処理のもう 1 つの利点は、使用されないポインターの修復でリソースを無駄にすることがない点です。多くの副次索引では、実際に使用される項目は少数です。非 HALDB データベースの場合、索引付きデータベースが再編成されるごとに索引全体が再作成されます。 HALDB の場合には、索引は再作成されず、少数の参照される索引項目のみが更新されます。 HALDB は使用されないポインターの更新にはリソースを使いません。

EPS が更新されると、アプリケーションが更新を行うことを許可されている場合のみ、バッファーに変更のマークを付けます。更新が許可され、ブロック・レベルのデータ共用環境が使用されている場合、変更されたブロックに対してブロック・ロックが要求されます。ブロック・レベルのデータ共用環境は、IRLM が使用され、データベースの共用レベルが 2 または 3 の場合に存在します。ブロック・ロックはアプリケーション・プログラムがその作業単位をコミットするまで保持されるため、パフォーマンスの問題を引き起こすことがあります。

自己修復パフォーマンスの最適化:

通常、更新権限をもつアプリケーション・プログラムは頻繁にコミットを行います。これは適切なプログラミングの習慣です。

しかし場合によって、更新を許可されているアプリケーション・プログラムが実際にはそうしないことがあります。例えば、PROCOPT=A を指定した PCB をもつプログラムが、読み取りを行うだけのことがあります。この場合、頻繁にコミットはしません。読み取りを行うだけなので、多数のロックを保持することはありません。これが、HALDB のインプリメンテーションによって変化します。そのプログラムがブロック・レベルのデータ共用環境で実行され、修復処理を呼び出すと、コミットされるまでブロック・ロックを保持することになります。これは 2 つの問題を引き起こします。第 1 に、長期にわたってロックを保持し、他のプログラムをブロック更新が可能になるまで待機させることがあります。第 2 に、多数のロックを保持することがあります。それが原因で IRLM またはロック構造でストレージが不足することがあります。

自己回復ポインター処理の実行時にプログラムが長期間ロックを保持する、または多数のロックを保持するという問題がある場合には、次の 4 つのオプションがあります。

- アプリケーション・プログラムが更新を行わない場合は、PROCOPT=G を使用します。
- プログラムが頻繁にコミットを行うようにします。

- ポインター修復処理を、PROCOPT=A を使用するアプリケーション・プログラムの実行前に呼び出し、更新を行わないようにします。このタイプのアプリケーション・プログラムより前に、別のプログラムまたはユーティリティーを実行します。 HALDB 変換および保守支援ツールが、ポインター修復ユーティリティーを提供しています。
- IMS Index Builder for z/OS などの索引ビルダーを使用して、副次索引を再作成します。 IMS Index Builder for z/OS は、正しい RBA をもつ EPS を作成します。

このシナリオは一般的なものではありません。大部分のユーザーは、特別な予防措置をとらずにポインター修復処理を進めることができます。

推奨事項: 再編成の後では副次索引を再作成しないでください。 HALDB の自己修復処理でポインターの訂正を行ってください。この方法により、再編成による停止を短縮し、リソースの使用を最小限に抑える傾向があります。

関連資料:

- IMS 高可用性ラージ・データベース変換および保守支援ツールについて詳しくは、*IMS High Availability Large Database Conversion and Maintenance Aid for z/OS User's Guide* を参照してください。
- IMS 索引ビルダーについて詳しくは、「*IMS Index Builder for z/OS* ユーザーズ・ガイド」を参照してください。

データベース・レコードの階層構造の変更

データベース・レコードの構造に対する変更を必要とするチューニングの変更のうちユーザーが行う必要があるものには、2 つの種類があります。

第 1 は、パフォーマンス向上を図るために、データベース・レコードの中のセグメント・タイプの階層順を変更することです。第 2 は、スペースの使用効率を最大化するためのセグメントの結合です。

関連概念:

801 ページの『レコード・セグメントの変更』

セグメント・タイプの順序の変更

一般に、頻繁に使用される従属セグメントがルート・セグメントの近くにあり、あまり使用されない従属セグメントがデータベース・レコードの終わりの方にあるとき、パフォーマンスが最もよくなります。

このような配置によってパフォーマンスが最良になるのは、(HSAM を除き) すべての種類のデータベースがルート・セグメントへの直接アクセス (したがって高速のアクセス) を行うことができるからです。しかし、いったんルート・セグメントが見つかると、次のいずれかの方法で従属セグメントを見つけることができます。

- データベース・レコードを順次に検索する (HSAM と HISAM)、または
- ルート・セグメントからポインターに従い 1 つの従属パスに行き、次いで正しいセグメントに達するまで兄弟チェーン全体を検索する (HDAM、HIDAM、PHDAM、と PHIDAM)。

階層内の従属セグメント・タイプの順序が効率のよいものであるかを判断する 1 つの方法は、DL/I 呼び出し要約報告書の IWAITS/CALL フィールドを調べることです。

関連資料: DL/I 呼び出し要約報告書の詳細については、「*IMS V13* システム管理」を参照してください。

この IWAITS/CALL フィールドには、特定のセグメントに対する DL/I 呼び出しごとに、ある 1 つのセグメントを処理できるようになるまでこのセグメントが入出力操作の終了を待たなければならなかった平均回数が示されています。この数が大きければ (もちろん、どのような数を大きい数と見るかは、アプリケーションによって異なります)、このセグメントを処理するのに複数の入出力操作が必要であったことになります。

このデータベースを再編成する必要がなければ、この数が大きいときは、これが頻繁に使用されるセグメントであり、しかもデータベース・レコードの初めから遠すぎるところに置かれていることを意味していることがあります。これが実状であると判断すれば、このセグメント・タイプの配置を変更することができます。この変更によって他のセグメントの IWAITS/CALL フィールドの値が増えることもあり得ます。

あるセグメント・タイプの配置を変更するためには、データベースからセグメントを新しい階層順にアンロードするためのプログラムを書き込む必要があります。(このような変更を行うために再編成ユーティリティーを使用することはできません。) 次いでこのセグメントを新しいデータベースにロードする必要があります。ここでも、再ロードのためのプログラムを書き込む必要があります。

セグメントの結合

データベース・レコードの構造に場合によって加える必要がある第 2 の種類の変更は、スペース使用効率を最大化するために行うセグメント・タイプの結合です。

例えば、大学の講義を内容とする従属セグメントがあり、これに従属するセグメントとして、講義を担当する講師のセグメントがある場合、1 つの講義を担当する講師が通常 1 人か 2 人だけであれば、2 つのセグメント・タイプを用いるのは、スペース効率が悪い使い方です。講師用に別個のセグメントを設けるのではなく、この 2 つのセグメント・タイプを結合すればスペースが節約できます。

セグメントを結合する場合にも、アンロード・プログラムおよび再ロード・プログラムを書き込む必要があります。 (このような変更を行うために再編成ユーティリティーを使用することはできません。)

HALDB データベースの階層構造の変更

HALDB データベースの階層構造を変更することができます。

階層構造を変更するためには、以下の手順を行う必要があります。

1. 加えようとしている変更がアプリケーション・プログラムの中のコードに影響を及ぼすかどうかを判別します。影響を及ぼすようであれば、そのコードが変更されることを確認しておいてください。

2. ユーザーのロード・プログラムおよび既存の DBD を用いて、データベースをアンロードします。
3. 新しい DBD をコーディングします。
4. 加えようとしている変更がアプリケーション・プログラムの中のコードに影響を与える場合には、このアプリケーション・プログラムの PSB に必要な変更を行います。 DB/DC データ・ディクショナリーをもっている場合には、そのディクショナリーを使用して、DBD の変更の影響を受けるアプリケーション・プログラムと PCB を判別することができます。
5. 非 VSAM データ・セットの場合は、古いデータベース・スペースを削除して、新しいデータベース・スペースを定義します。 VSAM データ・セットでは、古いクラスターに割り振られているスペースを削除して、新しいクラスターのスペースを定義します。
6. ユーザーのロード・プログラムおよび新しい DBD を用いて、データベースを再ロードします。データベースを再ロードしたらすぐに、このデータベースのイメージ・コピーを作成することを忘れないでください。
7. データベースで論理関係または副次索引を使用している場合には、接頭部情報を解決するために、再ロードの前後で再編成ユーティリティーを実行する必要があります。

関連タスク:

711 ページの『再編成ユーティリティーを使用するオフライン再編成』

直接アクセス・ストレージ・デバイスの変更

DASD (直接アクセス・ストレージ・デバイス) を変更して、データベースのチューニングを行うことが適切であるという状況がいくつかあります。

1 つは、アプリケーションの要件が変化して、より高速または低速の装置を使用しなければならない場合です。変更のもう 1 つの理由は、よりパフォーマンスのよい新しい装置を利用したいという場合です。最後に、装置の使用を求めて生じる競合を最小限に抑えるために、装置を変更してデータベース・データ・セットを異なる 2 つの装置に収容しようとする場合もあります。

再編成ユーティリティーを用いてデータベース (またはその一部) をある装置から他の装置に変更することができます。直接アクセス記憶装置を変更するには、以下の手順を行います。

1. 既存の DBD と該当するアンロード・ユーティリティーを使用してデータベースをアンロードします。
2. 新しい装置におけるトラック・スペースの使用効率を最大化するために、CI サイズまたはブロック・サイズを計算し直します。
3. 新しい DBD をコーディングします。
4. 非 VSAM データ・セットの場合は、古いデータベース・スペースを削除して、新しいデータベース・スペースを定義します。 VSAM データ・セットでは、古いクラスターに割り振られているスペースを削除して、新しいクラスターのスペースを定義します。

5. 新しい DBD と該当する再ロード・ユーティリティーを用いて、データベースを再ロードします。データベースを再ロードしたらすぐに、このデータベースのイメージ・コピーを作成することを忘れないでください。
6. データベースで論理関係または副次索引を使用している場合には、接頭部情報を解決するために、データベースの再ロードの前後で再編成ユーティリティーを実行する必要があります。

関連概念:

497 ページの『CI とブロックのサイズの決定』

関連タスク:

711 ページの『再編成ユーティリティーを使用するオフライン再編成』

OSAM 順次バッファリングのチューニング

OSAM 順次バッファリングを使用している場合には、次の 2 つのことに注意すると、データベースを効率よく処理するのに役立ちます。

- データベースを良い編成に保ちます。つまり、論理 (データベース・レコード) 順序を物理 (DASD ブロック) 順序とほぼ同じものであるようにします。
- 適切な数の SB バッファー・セットを選択します。

関連概念:

785 ページの『OSAM 順次バッファリングの使用状況データ』

編成の良いデータベースの例

OSAM 順次バッファリング (SB) を処理するうえで、データベースを編成の良い状態に保つことは、適切な数の SB バッファー・セットを選択することよりも重要です。

SB で処理するデータベースが良い編成であれば、効率が向上して経過時間が少なくてすみます。ユーザー・プログラムで処理されるのは、IMS データベース・セグメントおよびレコードであり、DASD ブロックが直接処理されるわけではないからです。良い編成のデータベースを論理レコード順に処理すると、入出力参照パターンにおいて、ほとんどの DASD ブロックが物理的順序でアクセスされることになります。 SB は多くの順次読み取りを出すことにより、これらの順次入出力パターンを利用することができます。順次読み取りを広範囲に使用すると、ジョブの経過時間がかなり短縮されます。

編成の良くないデータベースの例

編成のよくないデータベースを論理レコード順に処理すると、一般的には、多くの DASD ブロックがランダムな順序でアクセスされる入出力参照パターンになります。

これは、データベースのロードまたは再編成によって、多くのセグメントがランダムに分散するブロックに保管されているからです。データベースが大部分ランダムなパターンでアクセスされる場合には、SB バッファー・ハンドラーから出る入出力操作は、ほとんどがランダム読み取りになります。 SB は順次読み取りを多くは出せないので、ユーザー・ジョブの経過時間を大幅に短縮することはできません。

オプションの //DFSSTAT 報告書を使用して、その中の SB バッファリング統計から、データベースが良い編成かどうかを調べることができます。順次処理中にランダム読み取りを用いて読み取るブロックの割合が高い場合には、データベースの編成が適切でないことがあります。また、この割合をある期間にわたってモニターすることにより、データベースが古くなるにつれてこの割合が増加しているか調べることもできます。

関連概念:


//DFSSTAT 報告書 (システム管理)

編成の良いデータベースの確実化

いくつかの一般的な慣例に従うことで、データベースを確実に編成の良いものにすることができます。

編成の良いデータベースにするには、データベース設計時に以下の措置を講じます。

- データベースのロード時または再編成時に、フリー・スペースを十分に組み込みます。 IMS はこのフリー・スペースを使用して、新しいセグメントをその関連セグメント (同じデータベース・レコードの中のセグメント) のそばに挿入することができます。

  ヒント: ユーザーのデータベースの増加率およびパフォーマンスの特性に基づいて、フリー・スペースの量を選択してください。新規のデータベースの場合、25% という値を使用し、必要に応じてこの値を増加または減少させてください。再使用可能なフリー・スペースが 5% 未満になったら、そのデータベースの再編成をスケジュールするのもよい案です。
- 適切なデータベースの再編成の頻度を選択します。
- 効率のよい HDAM および PHDAM ランダム化モジュールやランダム化パラメーターを使用します。

関連概念:

707 ページの『データベースの再編成』

489 ページの『使用するランダム化モジュールの決定 (HDAM および PHDAM のみ)』

関連タスク:

487 ページの『フリー・スペースの指定 (HDAM、PHDAM、HIDAM、および PHIDAM のみ)』

HDAM または PHDAM オプションの調整

それぞれがパフォーマンスに影響する多くのさまざまな HDAM および PHDAM オプションを選択することができます。

再編成ユーティリティーを用いて、HDAM および PHDAM オプションを調整することができます。

1. 加えようとしている変更がアプリケーション・プログラムの中のコードに影響を及ぼすかどうかを判別します。これを行うのは、順次ランダム化モジュールに対して変更しようとする場合だけです。

2. 既存の DBD および該当するアンロード・ユーティリティーを用いて、データベースをアンロードします。
3. TSO 区画定義ユーティリティーを使用して、新規の DBD (PHDAM 以外の場合) をコーディングします。 CI サイズまたはブロック・サイズを変更した場合には、その新しいサイズに合わせたバッファーを割り振る必要があります。
4. 加えようとしている変更がアプリケーション・プログラムの中のコードに影響を与える場合には、このアプリケーション・プログラムの PSB に必要な変更を行います。 DB/DC データ・ディクショナリーをもっている場合には、そのディクショナリーを使用して、DBD の変更の影響を受けるアプリケーション・プログラムと PCB を判別することができます。
5. データベース・スペースを計算し直す必要があるかどうかを判断します。
6. 非 VSAM データ・セットの場合は、古いデータベース・スペースを削除して、新しいデータベース・スペースを定義します。 VSAM データ・セットでは、古いクラスターに割り振られているスペースを削除して、新しいクラスターのスペースを定義します。
7. 新規の DBD (もし、あれば) と適切な再ロード・ユーティリティーを使用して、データベースまたは区画を再ロードします。データベースを再ロードしたらすぐに、このデータベースのイメージ・コピーを作成します。

関連概念:

490 ページの『HDAM または PHDAM オプションの選択』

487 ページの『第 21 章 全機能データベースの設計』

489 ページの『使用するランダム化モジュールの決定 (HDAM および PHDAM のみ)』

関連タスク:

603 ページの『データベースの最小サイズの見積もり』

バッファーの調整

バッファーを調整すると、パフォーマンスに影響する可能性があります。バッファーのパフォーマンスが低下し始めたことに気付いたとき、あるいはバッファー・アクティビティーの増加を見越して、パフォーマンスを向上させるためにオプションを追加したい場合は、バッファーを調整します。

そのトピックでは、パフォーマンスへの影響があるバッファーの数とバッファーのサイズの選択についても説明しています。

選択できるバッファーのサイズと数については、 499 ページの『仮想記憶域内における複数のバッファー』に説明があります。パフォーマンスを向上させるには、このトピックを読み、現在のバッファー設定を評価してからバッファーを調整してください。

関連タスク:

789 ページの『オペレーティング・システム・アクセス方式の変更』

511 ページの『SB 制御ステートメントによる SB の要求』

動的データベース・バッファー・プールの概要

IMS システムをオフラインにせずに、OSAM サブプールおよび VSAM 共用リソース・プールを動的に再構成できます。

動的データベース・バッファー・プール機能は、OSAM および VSAM のサブプール定義を指定するために、IMS PROCLIB データ・セットの DFSVSMxx メンバーを変更する代わりに使用できます。DFSVSMxx は IMS 初期設定のときにのみ処理され、サブプール定義は IMS のオンライン中には動的に変更できません。

動的データベース・バッファー・プール機能を使用すると、サブプールに対するすべてのアクティビティーを静止させ、UPDATE POOL コマンドを使用してサブプール定義を動的に変更することができます。このコマンドは、IMS リソースがまだアクティブに使用されているときに、IMS PROCLIB データ・セットの DFSDFxxx メンバーで定義されているバッファー・プール再構成を開始します。

単一の DFSDFxxx メンバー内で、さまざまな定義セットを含んでいる複数の固有な OSAMxxx セクションまたは VSAMxxx セクションを指定できます。UPDATE POOL コマンドを呼び出すときは、更新された定義の取得元となる特定の OSAMxxx セクションまたは VSAMxxx セクションを指定します。

動的変更は再始動データ・セット (RDS) に保管されるため、緊急時再始動の後も保存されています。しかし、その後にコールド・スタートまたはウォーム・スタートがあると、変更は失われます。変更を永続的なものにするには、DFSVSMxx メンバーに変更を加える必要があります。

UPDATE POOL コマンドを実行した後、QUERY POOL コマンドを呼び出して現行バッファー・プールの統計を返し、更新された定義が意図したとおりに実装されていることを確認できます。

サポートされているバッファー・プール定義の更新のタイプ

バッファー・プール定義には以下のタイプの変更を加えることができます。

- 既存の OSAM サブプールのバッファー数を変更する
- 新しいサブプール ID を使用して OSAM サブプールを作成する
- 既存の OSAM サブプールに新しいバッファー・セットを割り当てる
- OSAM サブプールを削除する
- 既存の VSAM サブプールのバッファー数を変更する
- 既存の VSAM 共用リソース・プールのサブプールを作成する
- 既存の VSAM 共用リソース・プールからサブプールを削除する
- VSAM サブプールのバッファーのサイズを変更する
- VSAM 共用リソース・プールを作成する

関連概念:

781 ページの『OSAM および VSAM データベース・バッファーの調整』

777 ページの『VSAM バッファーのモニター』

777 ページの『VSAM バッファー』

778 ページの『VSAM バッファーのパフォーマンスを向上させるためのオプション』

244 ページの『VSO DEDB エリアの定義』

関連タスク:

779 ページの『OSAM バッファー』

781 ページの『OSAM データベース・バッファーの動的な調整』

783 ページの『VSAM データベース・バッファーの動的な調整』

関連資料:

- IMS PROCLIB データ・セットの DFSDFxxx メンバー (システム定義)
- UPDATE POOL コマンド (コマンド)
- QUERY POOL コマンド (コマンド)

VSAM バッファー

IMS は、IMS PROCLIB データ・セット (あるいは、バッチ環境またはユーティリティー環境では DFSVSAMP データ・セット) の DFSVSMxx メンバーで指定されたバッファー数とサブプール・サイズに基づいて、VSAM バッファー・プールを作成します。

動的データベース・バッファー・プール機能を使用して、IMS システムをオフラインにすることなく、バッファー数とサブプール・サイズを変更することもできます。

システム定義時の定義

ISRT や GU などの DL/I 呼び出しが VSAM データ・セットにアクセスする必要がある場合は、IMS が必要に応じて VSAM PUT および GET の呼び出しを行います。VSAM バッファー・プールは、IMS でなく DFSMS によって管理および操作されます。

動的な変更

IMS システムをオフラインにしなくても、新しいバッファー・プールを作成したり、既存のプールを変更 (または削除) したりすることができます。 IMS リソースがまだアクティブに使用されているときに、IMS PROCLIB データ・セットの DFSDFxxx メンバーで新しい VSAM 共用リソース・プール定義を指定してから、適切な UPDATE POOL TYPE(DBAS) コマンドを発行してください。

関連概念:

781 ページの『OSAM および VSAM データベース・バッファーの調整』

関連資料:

- IMS PROCLIB データ・セットの DFSVSMxx メンバー (システム定義)

VSAM バッファーのモニター

VSAM を使用している場合は、DB モニター報告書、IMS モニター・ツール、または IMS コマンドを使用してバッファーをモニターすることができます。

DB モニター報告書では、定義したバッファー・サイズごとに VSAM サブプール報告書が作成されます。VSAM バッファー・プール報告書には、(SUBPOOL

BUFFER SIZE フィールドと TOTAL BUFFERS IN SUBPOOL フィールドに) サブプール内のバッファーの数およびそのサイズが示されています。

IMS モニターは、バッチ環境における DL/I データベースのパフォーマンスに関するデータを記録するためのツールです。

VSAM データベース・バッファー・プールのプロセッサー・ストレージ使用統計を表示するには、タイプ 1 の /DISPLAY POOL DBAS コマンドを使用します。

動的データベース・バッファー・プール機能を使用して VSAM 共用リソース・プールを動的に再構成する場合は、タイプ 2 の QUERY POOL TYPE(DBAS) コマンドを発行してバッファー・プールに関する情報を取得し、動的更新が意図したとおりに完了したことを確認できます。

関連概念:

699 ページの『第 28 章 データベースのモニター』

DB モニター報告書 (システム管理)

関連資料:

QUERY POOL コマンド (コマンド)

/DISPLAY POOL コマンド (コマンド)

VSAM バッファーのパフォーマンスを向上させるためのオプション

VSAM バッファーのパフォーマンスを向上させるために、いくつかの調整を行うことができます。

- バックグラウンド書き込み機能がオンに切り換えられる場合、 NUMBER OF VSAM WRITES TO MAKE SPACE IN THE POOL フィールドの数がゼロでなければ、このサブプールに割り振られているバッファーの数が十分でない可能性があります。より多くのバッファーを割り振って、この数が減少するかあるいはゼロになるよう試みてください。
- 特定のアプリケーションのパフォーマンスを向上させる必要がある場合は、以下のことを行うと、特定のデータ・セット用にサブプールを確保することができます。
  - 複数のローカル共用リソース・プールを定義する。
  - サブプールを特定のデータ・セット専用にする。
  - VSAM データ・セットの索引コンポーネントとデータ・コンポーネントに対して、別個のサブプールを定義する。
- VSAM 順次モード処理を使用していない場合には、DFSVSAMP DD ステートメントで指定する VSAM バッファー数が、パフォーマンスに大きく影響します。この問題は、読み取らなければならない VSAM KSDS の索引の数にデータ部分のための 1 を加えた数が、割り振られた VSAM バッファーの数と等しいかそれより大きい場合に起こります。このようなパフォーマンスの問題は、バッファーの数を増加するか、あるいは VSAM 順次モードを使用することにより改善することができます。VSAM 順次モードを使用すれば、順次セットより上の索引を読み取る必要性が軽減されます。しかし、順次モードが取得できるのは、単一 PCB によって参照される DBD を使用しているバッチ環境、および LOAD ま

たは RETRIEVE の処理オプションを指定しているバッチ環境だけです。VSAM 順次モードはデータ共用には使用できません。

- VSAM バッファーは z/OS ハイパースペース・バッファリングを利用することができます。

関連概念:

781 ページの『OSAM および VSAM データベース・バッファーの調整』

ハイパースペース・バッファリングのパラメーター:

ハイパースペース・バッファリングを使用するには、VSRBF サブプール定義ステートメントで、オプション・パラメーターを 1 つまたは 2 つ指定しなければなりません。

指定するパラメーターは次のとおりです。

HSO|HSR
サブプールのために要求されたハイパースペース・バッファリングが、使用不能になった場合に IMS がとるアクションを指定します。

HSO ハイパースペース・バッファリングはオプションです。IMS は実行を続けます。

HSR ハイパースペース・バッファリングが必要です。IMS は終了します。

HSn
1 つのサブプールのために構築されるハイパースペース・バッファーの数を指定します。*n* は 1 から 8 の数字です。

ハイパースペース・パラメーターは、バッファー・サイズが 4K または 4K の倍数にのみ有効です。 4K より小さいバッファーにハイパースペース・パラメーターを指定すると、エラーが発生します。 ハイパースペース・バッファリングを使用するには、データベースをアンロードしてから、それを 4K または 4K の倍数の CI サイズに再ロードして、ハイパースペースの要件に対応しなければなりません。

CI サイズが 4K 未満のデータベースを、そのまま手を付けないでおくと決めた場合には、4K 未満のバッファーは割り振らないでください。 4K 未満のバッファーを割り振ると、4K 未満の CI が 4K 以上のバッファー・プールに入れられます。ただし、CI はすでにある VSAM データ・セットと競合します。この方法は、短期間には適切かもしれません。

関連資料: ハイパースペース・バッファーも含め、VSAM バッファーについて詳しくは、「*z/OS DFSMS: Using Data Sets*」を参照してください。

関連概念:

500 ページの『ハイパースペース・バッファリング』

関連資料:

VSAM サブプールの定義 (システム定義)

OSAM バッファー

OSAM を使用している場合には、個々のサブプール・バッファー報告書があります。ただし、拡張 OSAM バッファー・サブプールの統計機能を使用することで、

使用中のバッファーの数をモニターできます。OSAM サブプールを動的に定義してある場合は、QUERY コマンドを発行してバッファー・プールに関する情報を取得することもできます。

拡張 OSAM バッファー・サブプールの統計機能では、以下の値がサポートされます。

DBESF
完全な OSAM サブプール統計を定様式で示します。

DBESU
完全な OSAM サブプール統計を不定様式で示します。

DBESS
OSAM データベース・バッファー・プール統計の要約を定様式で示します。

DBESO
/DIS POOL コマンドの結果として戻されたオンライン統計に関する、完全な OSAM データベース・バッファー・プール統計を定様式で示します。

パフォーマンスが向上するもう 1 つの方法があります。それは、特定のアプリケーションを対象とするもので、特定のデータ・セット用に使用するサブプールを確保します。例えば、ブロック・サイズが 512 バイトである索引データ・セットがあるとすれば、このデータ・セットのために 512 バイトのバッファーが入っているサブプールを確保します。これを行うには、このデータベース内の他のどのデータ・セットにも 512 バイトというブロック・サイズを定義しなければよいのです。(ブロック・サイズは、DBD の中の DATASET ステートメントの BLOCK= オペランドで、データ・セットごとに 指定されることに注意してください。) こうしておいて、この索引の中のすべてのブロックを保持するのに十分な数だけの 512 バイト・バッファーを割り振れば、このバッファー・プールの中に読み込まれたブロックすべてが、このバッファー・プールの中にとどまります。

パフォーマンスは、IMS.PROCLIB の DFSVSMxxx メンバーまたは DFSVSAMP のいずれかで指定する IOBF 制御ステートメントの **co** (キャッシング・オプション) パラメーターの使用によっても向上することができます。

IMS システムをオフラインにせずに、OSAM サブプールを動的に再構成し、新しいバッファー・プールを作成し、既存のプールを変更することができます。

関連概念:

781 ページの『OSAM および VSAM データベース・バッファーの調整』

OSAM バッファー・プールの互換性定義 (システム定義)

強化/拡張 OSAM バッファー・サブプール統計の形式 (アプリケーション・プログラミング)

データベース・バッファー・プール報告書 (システム管理)

関連資料:

QUERY POOL コマンド (コマンド)

OSAM データ・キャッシュのカップリング・ファシリティーの使用 (システム定義)

OSAM サブプールの定義 (システム定義)

OSAM および VSAM データベース・バッファーの調整

システム定義のとき、IMS PROCLIB データ・セットの DFSVSMxx メンバーでパラメーターを指定することにより、OSAM および VSAM のデータベース・バッファーを調整できます。

また、IMS PROCLIB データ・セットの DFSDFxxx メンバーでパラメーターを指定することによって、OSAM および VSAM のデータベース・バッファーを動的に調整することもできます。

関連概念:

776 ページの『動的データベース・バッファー・プールの概要』

777 ページの『VSAM バッファー』

IMS バッファー・プール (システム定義)

関連タスク:

779 ページの『OSAM バッファー』

関連資料:

IMS PROCLIB データ・セットの DFSVSMxx メンバー (システム定義)

IMS PROCLIB データ・セットの DFSDFxxx メンバー (システム定義)

UPDATE POOL コマンド (コマンド)

QUERY POOL コマンド (コマンド)

DFSVSMxx での OSAM および VSAM のデータベース・バッファーの調整

OSAM および VSAM のデータベース・バッファーを調整するためには、バッファーのサイズと数を指定している制御ステートメントを変更し、それらの制御ステートメントを IMS PROCLIB データ・セットの適切なメンバーに入れます。

新しい制御ステートメントを以下のデータ・セットに入れます。

- バッチ環境およびユーティリティー環境では DFSVSAMP データ・セット
- IMS TM および DBCTL 環境ではメンバー名 DFSVSMxx を持つ IMS.PROCLIB データ・セット

OSAM データベース・バッファーの動的な調整

IMS リソースがまだアクティブに使用されているときに OSAM データベース・バッファーを調整するには、IMS PROCLIB データ・セットの DFSDFxxx メンバーで新しい OSAM サブプール定義を指定した後、適切な UPDATE POOL TYPE(DBAS) コマンドを発行します。

前提条件: 以下の条件が満たされていることを確認してください。

- OSAM バッファー・プールの現在の数またはサイズが、アプリケーション・プログラムのワークロード処理に不十分であるか不要である。
- IMS PROCLIB データ・セットの DFSDFxxx メンバーが存在する。
- IMS が、少なくとも最小の Common Service Layer を使用して (タイプ 2 コマンドの発行をサポートするために) 構成されている。

1. IOBF ステートメントを使用して、IMS PROCLIB データ・セットの DFSDFxxx メンバーを更新します。以下の 1 つ以上のステップを実行します。
   - 既存の OSAM サブプールのバッファー数を変更します。

     既存の OSAM サブプールのバッファー数を変更する場合、*bufnum*、*fix1*、および *fix2* の値を変更できます。
   - 新しいサブプール ID を使用して OSAM サブプールを作成します。

     存在しない *id* パラメーターの値を使用して、IOBF ステートメントを追加します。

     既に存在するバッファー・サイズを使用してサブプールを作成できます。このサブプールは、重複するサブプールと見なされるはずのもので、本来なら固有のサブプール ID を使用して定義されている必要があります。別のサブプールを割り当てるには、データベース・データ・セットをいったんクローズしてから再度オープンする必要があります。

     バッファー・サイズが既存のサブプールと同じで、固有のサブプール ID が割り当てられていない場合、事実上、既存のサブプールの定義を変更していることになります。
   - 既存のサブプールに新しいバッファー・セットを追加します。

     新しいバッファー・セットを追加したいサブプール ID を使用して IOBF ステートメントを追加します。

     IOBF ステートメント内の新しい値は、以前に設定された値を置き換えません。

     例えば、異なる *bufsize* 値を指定しても、サブプール内のバッファーのサイズは変更されません。バッファー・サイズが異なる新しいバッファー・セットが、既存のサブプールに追加されます。既存のサブプール ID を使用して異なる *bufsize* 値を指定した場合、サブプール内のバッファー全体のサイズが増えます。
   - *bufnum* を 0 に設定して、OSAM サブプールを削除します。

   IOBF ステートメントのフォーマット:

   ```
   IOBF=(bufsize,bufnum,fix1,fix2,id,co)
   ```
2. 次のコマンドを発行します。

   ```
   UPD POOL TYPE(DBAS) SECTION(OSAMxxx) MEMBER(xxx)
   ```

プロシージャーが正常に実行された場合、IMS システムは再構成の間、ターゲット・サブプールに対するアクティビティーを休止します。

既存の OSAM サブプールのバッファー数と変わらないバッファー数を指定した場合、OSAM サブプールを更新する要求は (要求がなかったかのように) 無視されます。

以下のようにして、更新されたデータベース・バッファー・プール構成がワークロードに十分であるかどうかを判別します。

1. 次のコマンドを発行します。

   ```
   QUERY POOL TYPE(DBAS) SUBTYPE(OSAM,VSAM) SIZE() POOLID() SHOW(STATISTICS)
   ```

   IMS システムは OSAM および VSAM のバッファー・プールに関する統計を返します。
2. 返された統計に基づいて、OSAM バッファー・プールの構成に対してさらに調整が必要かどうかを評価します。

関連概念:

776 ページの『動的データベース・バッファー・プールの概要』

関連資料:

- IMS PROCLIB データ・セットの DFSDFxxx メンバー (システム定義)
- UPDATE POOL コマンド (コマンド)
- QUERY POOL コマンド (コマンド)

VSAM データベース・バッファーの動的な調整

IMS リソースがまだアクティブに使用されているときに VSAM データベース・バッファーを調整するには、IMS PROCLIB データ・セットの DFSDFxxx メンバーで新しい VSAM 共用リソース・プール定義を指定した後、適切な UPDATE POOL TYPE(DBAS) コマンドを発行します。

以下の条件が満たされていることを確認してください。

- VSAM 共用リソース・プールの現在の数またはサイズが、アプリケーション・プログラムのワークロード処理に不十分であるか不要である。
- IMS PROCLIB データ・セットの DFSDFxxx メンバーが存在する。
- IMS が、少なくとも最小の Common Service Layer を使用して (タイプ 2 コマンドの発行をサポートするために) 構成されている。

1. IMS PROCLIB データ・セットの DFSDFxxx メンバーを更新します。以下の 1 つ以上を実行します。
   - 既存の VSAM サブプールのバッファー数を変更します。

     POOLID を使用して DFSDFxxx メンバーを定義します。この POOLID では、サブプールが属する VSAM 共用リソース・プールを識別し、その後に VSRBF ステートメントを続けます。例えば、次のようになります。

     ```
     POOLID=(id,VSRBF=bufsize,bufnum)
     ```

     bufnum を別の値に設定し、指定されたバッファー・サイズのサブプールのバッファー数を変更することを示します。
   - 既存の VSAM 共用リソース・プールのサブプールを作成します。

POOLID を使用して DFSDFxxx メンバーを更新します。この POOLID では、サブプールが属する VSAM 共用リソース・プールを識別し、その後に VSRBF ステートメントを続けます。例えば、次のようになります。

```
POOLID=(id,VSRBF=bufsize,bufnum,type,HSO,HSR,HSn)
```

- 既存の VSAM 共用リソース・プールからサブプールを削除します。

  DFSDFxxx メンバーの POOLID ステートメントで、*bufnum* を 0 に設定し、指定されたバッファー・サイズのサブプールを削除することを示します。例えば、次のようになります。

  ```
  POOLID=(id,VSRBF=bufsize,0)
  ```

- VSAM サブプールのバッファーのサイズを変更します。

  バッファー・サイズが現行のサブプールと異なるサブプールを作成してから、現行のサブプールを削除します。

- VSAM 共用リソース・プールを作成します。

  新しい VSAM 共用リソース・プールを追加するときは、POOLID ステートメントと、その直後に 1 つ以上の VSRBF サブプール定義ステートメントを指定して、DFSDFxxx メンバーを更新します。 例えば、次のようにします。

  ```
  POOLID=id,Fixdata=,Fixindex,Fixblock,Stringnm=n
  VSRBF=bufsize,bufnum,type,HSO,HSR,HSn
  ```

2. 次のコマンドを発行します。

   ```
   UPD POOL TYPE(DBAS) SECTION(VSAMxxx) MEMBER(xxx)
   ```

プロシージャーが正常に実行された場合、IMS システムは再構成の間、ターゲット・サブプールに対するアクティビティーを休止します。

以下のようにして、更新されたデータベース・バッファー・プール構成がワークロードに十分であるかどうかを判別します。

1. 次のコマンドを発行します。

   ```
   QUERY POOL TYPE(DBAS) SUBTYPE(OSAM,VSAM) SIZE() POOLID() SHOW(STATISTICS)
   ```

   IMS システムは VSAM バッファー・プールに関する統計を返します。

2. 返された統計に基づいて、VSAM バッファー・プールの構成に対してさらに調整が必要かどうかを評価します。

関連概念:

776 ページの『動的データベース・バッファー・プールの概要』

関連資料:

- IMS PROCLIB データ・セットの DFSDFxxx メンバー (システム定義)
- UPDATE POOL コマンド (コマンド)
- QUERY POOL コマンド (コマンド)

OSAM 順次バッファリングの使用状況データ

OSAM 順次バッファリングを使用している場合には、順次バッファリング要約報告書および順次バッファリング詳細報告書を用いて、ユーザー・プログラムの実行中に SB バッファーがどのように使用されたかを調べることができます。

デフォルトにより、各 SB バッファー・プールにはそれぞれ 4 つのバッファー・セットがあります。報告書から、ランダム読み取り入出力操作を使用した割合が高いこと、およびプログラムでデータベースを順次処理していることが分かっている場合には、バッファー・セットの数を 6 以上に増やすと、パフォーマンスを向上させることができます。バッファー・セットの数を増やすことによって、あるブロックが要求された時に、依然として SB バッファーの中に残っていることが多くなり、読み取り入出力操作の必要がなくなります。

ユーザー・プログラムの実行中に、ランダム読み取りが少ししか使用されなかったという場合があります。これは、データベースが非常によく編成されており、大部分の要求が SB バッファー・プールからまたは順次読み取りにより充足されたことを示しています。このような場合には、各 SB バッファー・プールの中のバッファー・セットの数を 2 か 3 に減らすことによって、仮想記憶域スペースを節約することができます。

関連タスク:

773 ページの『OSAM 順次バッファリングのチューニング』

順次バッファーの調整

各 SB バッファー・プールに割り振られたバッファー・セットの数は、2 つの方法で変更することができます。

方法は以下のとおりです。

- BUFSETS キーワードを指定して SBPARM 制御ステートメントをコーディングする。
- SB 初期設定出口ルーチンを使用する。

いったん、バッファー・セットの数を変更したら、SB テスト・ユーティリティーを使用して、ユーザー・プログラムの実行中に出された SB バッファー・ハンドラー呼び出しシーケンスを処理し直すことができます。次いで、この結果出力された //DFSSTAT 報告書を検討して、変更の影響を調べることができます。

関連資料:

- 順次バッファリング要約報告書と順次バッファリング明細報告書、および SB テスト・ユーティリティーの使用法は、「*IMS V13* データベース・ユーティリティー」で説明しています。
- SBPARM 制御ステートメントのコーディング方法は、「*IMS V13* システム定義」で詳しく説明しています。
- SB 初期設定出口ルーチンの詳細については、「*IMS V13* 出口ルーチン」で説明しています。

VSAM オプションの調整

VSAM については、OPTIONS 制御ステートメントで指定されているオプションおよびアクセス方式サービスの DEFINE CLUSTER コマンドで指定されているオプションを調整できます。

特定してモニターすることのできる唯一の VSAM オプションは、バックグラウンド書き込み機能だけです。バックグラウンド書き込み機能を使用しない場合は、VSAM バッファー・プール報告書で調べることができます。この報告書については、「*IMS V13* システム管理」で説明しています。報告書の Number of VSAM Writes To Make Space in the Pool フィールドは、バッファー内のデータをデータベースに書き込んだ後でないと、バッファーを使用することができなかった回数を示します。バックグラウンド書き込み機能を使用すると、この回数を減らすことができ、したがって、バッファー・プールのサイズを縮小することができます。

すでにバックグラウンド書き込み機能を使用している場合には、VSAM バッファー・プール報告書のフィールドの Number OF Times Background Write Function Invoked で、バックグラウンド書き込み機能が何回呼び出されたかを示します。(DB モニターによって提供される別の報告書である) VSAM 統計報告書の BKG WTS フィールドには、バックグラウンド書き込み機能が呼び出されたかどうかが示されます。この報告書の USR WRTS フィールドでは、他の情報の中で、バックグラウンド書き込み機能が呼び出された回数が示されます。

関連タスク:

514 ページの『VSAM オプション』

789 ページの『オペレーティング・システム・アクセス方式の変更』

OPTIONS 制御ステートメントで指定されている VSAM オプションの調整

VSAM オプションを調整するには、OPTIONS 制御ステートメントの中の該当するパラメーターを変更し、新しい制御ステートメントを適切なデータ・セットに入れます。

制御ステートメントを入れるデータ・セットは、使用する IMS システムがバッチとオンラインのどちらであるかによって異なります。

- バッチ・システムでは、DFSVSAMP データ・セットを使用します。
- オンライン・システムでは、DFSVSMnn というメンバー名を持つ IMS.PROCLIB データ・セットを使用します。

この制御ステートメントのコーディング方法については、「*IMS V13* システム定義」で詳しく説明しています。

アクセス方式サービス DEFINE CLUSTER コマンドで指定されている VSAM オプションの調整

この種の VSAM オプションを調整するには、DEFINE CLUSTER コマンドの該当するパラメーターを変更します。これ以外に何をしなければならないかについては、変更する VSAM パラメーターに応じて決まります。

FREESPACE パラメーターの変更

再編成ユーティリティーを用いて、フリー・スペースの用い方を変更したり、先に指定したフリー・スペースの割合を変更したりすることができます。このような変更を行うには、以下の手順を行います。

1. 既存の DBD と該当するアンロード・ユーティリティーを使用してデータベースをアンロードします。
2. データベース・スペースを計算し直します。それを行う必要があるのは、その変更を加えることにより、データベース・スペースの必要量が変わってくるからです。
3. 古いデータベース・クラスターを削除し、FREESPACE パラメーターに変更を加えて新しいデータベース・クラスターを定義します。
4. 既存の DBD (DBD に変更を加えなかった場合) または新しい DBD を用いて、データベースを再ロードします。該当する再ロード・ユーティリティーを使用します。
5. 再編成されるデータベースが直接ポインターを使用する副次索引である場合には、接頭部情報を解決するために、再ロードの前後でいくつかの再編成ユーティリティーを実行する必要があります。

SPEED / RECOVERY パラメーターの変更

SPEED/RECOVERY パラメーターを変更するだけのために、データベースをアンロードし再ロードしないでください。 RECOVERY パラメーターを指定している場合は、データベースが再ロードされ、ロード・プログラムの再始動が使用されないときに、パラメーターを SPEED に変更してパフォーマンスを向上をしてください。IMS は RECOVERY パラメーターをサポートしていません。データベース・ロード・プログラムを UCF の下で実行する場合にのみ、リカバリーを実行することができます。

データベースをアンロードし再ロードすることが必要になるような他のデータベース変更を加える場合にのみ、このパラメーターを変更するものと想定しているので、ここではこのパラメーターの変更手順は示しません。

関連概念:

603 ページの『第 25 章 データベースのロード』

関連タスク:

603 ページの『データベースの最小サイズの見積もり』

711 ページの『再編成ユーティリティーを使用するオフライン再編成』

OSAM オプションの調整

OSAM オプションを調整するには、OPTIONS 制御ステートメントの中の該当するパラメーターを変更します。次に、新しい制御ステートメントを適切なデータ・セットに入れます。

適切なデータ・セットとは、以下のいずれかです。

- バッチ・システムでの DFSVSAMP データ・セット

- オンライン・システムでのメンバー名 DFSVSMxx を持つ IMS.PROCLIB データ・セット

選択できる OSAM オプションについては、 519 ページの『OSAM オプション』で説明しています。その節では、パフォーマンスへの影響がある各 OSAM オプションについても述べています。パフォーマンスの向上を図るためには、そのトピックを読み直して、最初に行った選択を再検討してください。特定してモニターできる OSAM オプションはありません。

OPTIONS 制御ステートメントのコーディング方法については、「*IMS V13* システム定義」で詳しく説明しています。

関連タスク:

789 ページの『オペレーティング・システム・アクセス方式の変更』

割り振られるスペース量の変更

以下の 2 つの状況が生じた場合にはデータベースに割り振るスペース量を変更してください。

第 1 の状況は、1 次スペースが不足しているという状況です。大幅なパフォーマンス低下を招く恐れがあるので、2 次スペース割り振りは使用しないでください。データベースに割り振るスペース量を変更したい 2 つ目の状況は、ある 1 つの DL/I 呼び出しを処理するのに必要な入出力操作の数が多すぎて、パフォーマンスが受け入れられないレベルになっている状況です。データベースの中のデータが DASD スペースに分散している場合、そのスペースが多すぎると、パフォーマンスが受け入れられない状態になることがあります。

日常業務の一環として、スペースの使用状況をモニターする 1 つの方法は、DL/I 呼び出し要約報告書の IWAITS/CALL フィールドを監視することです。この IWAITS/CALL フィールドの中に、比較的大きな数が入っている場合、この大きな数が、スペースの問題によってもたらされたものであることがあります。スペースが問題になっているのではないかと考えられる場合、次の 2 つのより特定的な方法によってその真偽を調べることができます。

- VSAM データ・セットの場合は、LISTCAT コマンドを使用して VSAM カタログから報告書を入手することができます。この報告書では、CI/CA 分割、EXCP、および EXTENTS を調べます。
- 非 VSAM データ・セットでは、LISTVTOC コマンドを使用して VTOC に関する報告書を入手することができます。この報告書では、NOEXT フィールドを調べます。

データベースに割り振られるスペース量を変更することにした場合には、JCL ユーティリティーまたは z/OS ユーティリティーを用いてこれを行います。データベースをその新しいスペースに入れるために、再編成ユーティリティーを実行する必要があります。データベースをその新しいスペースに入れる手順は、以下のとおりです。

1. 既存の DBD および該当するアンロード・ユーティリティーを用いて、データベースをアンロードします。
2. データベース・スペースを計算し直します。

3. 非 VSAM データ・セットの場合は、古いデータベース・スペースを削除し新しいデータベース・スペースを定義します。 VSAM データ・セットでは、古いクラスターに割り振られているスペースを削除して、新しいクラスターのスペースを定義します。
4. HDAM データベースのルート・アドレス可能域の中のスペースを変更している場合には、他の HDAM パラメーターを調整する必要があることがあります。この場合は、再ロードの前に新しい DBD をコーディングする必要があります (PHDAM 区画が変更される場合は新規の DBD は不要です)。 PHDAM 区画のルート・アドレス可能域のスペースを変更するには、HALDB 区画定義ユーティリティーを使用する必要があります。
5. 既存の DBD (DBD に変更を加えなかった場合) または新しい DBD を用いて、データベースを再ロードします。該当する再ロード・ユーティリティーを使用します。
6. HALDB 以外のデータベースで論理関係または副次索引を使用している場合には、接頭部情報を解決するために、データベースの再ロードの前後で再編成ユーティリティーを実行する必要があります。

関連概念:

699 ページの『第 28 章 データベースのモニター』

DL/I 呼び出し要約報告書 (システム管理)

関連タスク:

603 ページの『データベースの最小サイズの見積もり』

711 ページの『再編成ユーティリティーを使用するオフライン再編成』

オペレーティング・システム・アクセス方式の変更

アクセス方式を OSAM から VSAM へ、または VSAM から OSAM へ変更するために、再編成ユーティリティーを使用することができます。

アクセス方式を変更するには、次のようにします。

1. データベースをアンロードします。
2. 新しい DBD をコーディングします (ステップ 1 で述べたようにこれを済ませていない限り)。
3. 非 VSAM から VSAM に変更する場合には、古いデータ・セットを削除して新しいクラスターを定義します。 VSAM から非 VSAM に変更する場合には、古いクラスターを削除して新しいデータベース・データ・セットを定義します。
4. OSAM オプションおよびバッファーから VSAM のオプションおよびバッファーに、あるいはその逆に変更する必要があります。
5. 新しい DBD を用いて、データベースを再ロードします。データベースを再ロードしたらすぐに、このデータベースのイメージ・コピーを作成することを忘れないでください。
6. HALDB 以外のデータベースで論理関係または副次索引を使用している場合には、接頭部情報を解決するために、データベースのロードの前後で再編成ユーティリティーを実行する必要があります。

関連概念:

775 ページの『バッファーの調整』
786 ページの『VSAM オプションの調整』
関連タスク:
787 ページの『OSAM オプションの調整』
711 ページの『再編成ユーティリティーを使用するオフライン再編成』

高速機能システムのチューニング

高速機能アプリケーションが存在するときに IMS オンライン・システムをチューニングする場合には、その目標はメッセージ・ドリブン・プログラムの重要性と許容可能な応答時間に関するこのプログラムの基準によって決まります。

オペレーターは以下のパフォーマンス分析調査を行ってください。

- 十分な実記憶の可用性の調査
- 平衡グループの有効性の検査
- 高速機能従属領域の数と並列処理の可能性の調査
- 細分化された作業単位を減らすのに必要な DEDB の再編成の回数のモニター
- DEDB オーバーフロー・バッファーの使用のモニター
- EXEC ステートメントの DBFX パラメーターによって指定されたオーバーフロー・バッファーを同時に使用する必要があるプログラムの強制シリアライゼーションのモニター
- エリア・キー範囲の検査、およびランダム化アルゴリズムが改良できるかどうかの調査
- 混合モード処理量の削減

高速機能パフォーマンスは、以下のことによって生じる不必要な遅れをなくすことによっても向上させることができます。

- 特定の高速機能アプリケーション・プログラムへのトランザクション量
- DEDB 構造についての考慮事項
- DEDB 制御インターバル (CI) リソースに対する競合
- DEDB DASD のスペース切れ
- 使用可能な実記憶の使用状況
- 同期点処理と物理ロギング
- 出力スレッド (OTHR) に対する競合
- 再処理に伴うオーバーヘッド
- プロセッサーが支配的な場合と入出力が支配的な場合のディスパッチング優先順位
- DEDB 上での入出力によって生じた DASD の競合
- ブロック・レベル共用を指定するリソース・ロッキングの考慮事項
- バッファー・プールの使用と、似通ったバッファー使用特性をもつ高速機能アプリケーションを 1 つ以上のメッセージ・クラスにまとめるという手法を使用しないこと

トランザクション処理と CI に対する競合についての統計は、高速機能ログ分析ユーティリティー (DBFULTA0) の出力から取得でき、このユーティリティーは (システム・ログ入力から) 高速機能リソースの使用に関するデータを取り出します。

関連概念:

232 ページの『IMS ツールによる使用不可スペースの管理』

関連資料:

高速機能ログ分析ユーティリティー (DBFULTA0) (システム・ユーティリティー)

特定の高速機能アプリケーション・プログラムへのトランザクション量

特定の平衡グループへのキューに入っているトランザクションの量が異常に多い場合は、その平衡グループに関連している領域の数を増やすことを考えてください。

高速機能ログ分析報告書が、平衡グループ・キューイングについての情報を提供します。

DEDB 構造についての考慮事項

DEDB の使用に関するいくつかの特性が、アプリケーション・プログラムの応答時間に影響します。

これらの特性とは、次のようなものです。

- データの複製
- サブセット・ポインター
- エリアの数
- 階層構造の複雑性
- DL/I 呼び出しの複雑性
- IMS 全体にわたる共用の使用
- 最終的な子ポインター
- リカバリー可能性

最初は、DEDB 固有の特性であり、後の 5 つは、データベース一般に当てはまるものです。データの複製により、 1 つのエリアごとに最高 7 つのデータ・セットが可能になります。複数データ・セットで表される 1 つのエリアを読み込むときには、CI に障害がない限り、パフォーマンスは影響を受けません。更新の際には、最高 7 つの書き込みが必要になることがあります。物理的書き込みはトランザクション処理と非同期に実行されますが、さまざまな DASD 装置へのアクセス・パスによって遅れが生じることがあります。

最高 8 つのサブセット・ポインターによって、アプリケーション・プログラムは 1 つの親の複数の子を 1 つの DEDB 内の複数のグループに分けることができ、このときサブセット・ポインターは各グループの開始を指します。このようなポインターを使用すると、順次従属セグメントのチェーンの中で位置が大きく変位しているセグメントにアクセスするのに必要な時間を短縮することにより、パフォーマンスを向上させるのに役立ちます。

高速機能バッファー・プールからのバッファーの使用

高速機能バッファー・プールは、独自のバッファー・プールを持つ DEDB オンライン・ユーティリティーおよび高速順次処理 (HSSP) 機能を除く、すべての高速機能プログラムで使用されます。高速機能バッファー・プールは、MSDB と DEDB の処理をサポートするのに使用されます。

IMS システムの高速機能バッファー・プールは、IMS で高速機能 64 ビット・バッファー・マネージャーを使用可能にすることで自動的に定義するか、または IMS および DBC 始動プロシージャーで DBBF、DBFX、BSIZ の各パラメーターを指定することで手動で定義できます。

関連概念:

- IMS 実行パラメーターの指定 (システム定義)
- データベース・バッファー・プールのチューニング (システム管理)

高速機能バッファー・プールの動的な定義と割り振り

高速機能 64 ビット・バッファー・マネージャーを使用可能にすると、高速機能バッファー・プールが動的に定義されて割り振られます。
DBBF、DBFP、DBFX、BSIZ の各パラメーターは、指定されても無視されます。

高速機能 64 ビット・バッファー・マネージャーは、バッファーの使用状況と CI サイズの要件に基づいて高速機能バッファー・プールを動的に定義します。高速機能バッファーのサイズも数も指定する必要はありません。追加のバッファーが必要であるか、または CI サイズが現在割り振られているどのバッファー・プールにも適合しないことを IMS が検出した場合は、高速機能 64 ビット・バッファー・マネージャーによって必要なバッファー・プールが割り振られます。

高速機能 64 ビット・バッファー・マネージャーが従属領域に割り振ることのできるバッファーの最大数は、NBA および OBA 従属領域パラメーターの値の組み合わせによって定義されます。

拡張共通ストレージ域 (ECSA) の使用を軽減するために、高速機能 64 ビット・バッファー・マネージャーは DEDB バッファー・プールを 64 ビットの専用ストレージに配置します。

高速機能 64 ビット・バッファー・マネージャーを使用可能にするには、DFSDFxxx PROCLIB メンバーで FPBP64=Y を指定します。

関連資料:

- IMS PROCLIB データ・セットの DFSDFxxx メンバー (システム定義)

高速機能バッファー・プールの手動定義

高速機能 64 ビット・バッファー・マネージャーを使用しない場合は、IMS または DBC 始動プロシージャーで DBBF、DBFX、および BSIZ の各パラメーターを指定して高速機能バッファー・プールを定義する必要があります。

バッファー・プールを変更するには、DBBF、DBFX、および BSIZ の各パラメーターの値を変更し、IMS を再始動する必要があります。また IMS は、DEDB バッファー・プールを含むすべての高速機能バッファー・プールを、ECSA ストレージに配置します。

高速機能 64 ビット・バッファー・マネージャーを使用しない場合は、システム始動時に BSIZ パラメーターによって定義されるサイズのバッファーから高速機能バッファー・プールが構成されます。選択されたこのバッファー・サイズは、オープンされる DEDB エリアからの最大の CI を保持できなければなりません。ページ固定のバッファーの数は、提供されるパラメーターの値によって決まります。

- 通常のバッファー割り振り (NBA) 値は、従属領域の始動時にバッファー・プールに固定されるバッファーの数が定義されるようにします。 (この数は、NBA パラメーターを使用して従属領域始動プロシージャーに対して指定できます。) この従属領域の中のアプリケーション・プログラムは、次のいずれかが生じる前に、ある同期インターバル内で最高この数までのバッファーを受け取る資格があります。
  - バッファー・マネージャーが、要求しているアプリケーション・プログラムから、未変更バッファーを獲得する。
  - 要求しているアプリケーション・プログラムのために、これ以上バッファーを獲得できない (NBA に等しいバッファーの数が要求され、受け取られ、変更されています)。この場合、バッファー・マネージャーは、このプログラムについてこの値が指定されているならば、オーバーフロー・バッファー割り振り (OBA) にアクセスできなければなりません。 OBA が指定されていない場合には、これまでの同期インターバル処理の間にこのプログラムのために獲得したすべてのリソースが解放されます。
- この OBA 値とは、NBA を超えた場合にプログラムが順次に獲得できるバッファー数です。 (これは、OBA パラメーターを使用して従属領域始動プロシージャーに対して指定できます。) オーバーフロー・インターロック機能は、オーバーフロー・バッファーのアクセスを逐次化します。つまり、1 回に 1 つのアプリケーション・プログラムしか、このオーバーフロー・バッファー割り振りにアクセスできません。したがって、オーバーフロー・バッファーは、デッドロックに巻き込まれることがあります。
- システム始動パラメーターである DBFX 値は、最初の高速機能アプリケーション・プログラムの始動時にページ固定のバッファーの予約を定義します。非同期 OTHREAD 処理が、同期インターバル処理で出された要求をサポートするほど迅速にバッファーを解放しない場合に、このバッファーを使用します。

結果として、以下のようになります。

- BSIZ をオンラインになる最大の DEDB CI と等しく設定します。バッファー・マネージャーは、複数の制御インターバル対処するようなバッファーの分割は行はないので、すべての DEDB 制御インターバルを同じサイズにすると、ストレージを最適に利用できます。大きなブロック・サイズ (最大 28K) も使用できますが、こうすると、これより小さい CI サイズが多い場合に、バッファー・プールの一部しか使用できなくなります。

- NBA 値は、同期インターバルの間に行われるバッファーの更新の通常の数に、ほぼ等しく設定しなければなりません。照会専用のプログラムの NBA 値は、小さくすべきです。変更されることのないバッファーは再使用でき、同期時にすべて解放されるからです。
- OBA は限られた比率の同期インターバルとの関連でしか使用しないでください。 OBA は照会専用のプログラムでは必要ありません。一般に、ユーザーは OBA 値を、意図されたように使用するように注意しなければなりません。すなわち、アプリケーション・プログラムの論理が、変更されたバッファーの全体的ニーズを変更するように要求し、そのために例外的に OBA へのアクセスが必要となる場合に、同期インターバルをサポートするためにこれを使用すべきであるからです。 BMP では、1 より大きい OBA 値は必要ありません。これは、NBA 割り振りが超過されたときに戻される「FW」状況コードを、SYNC 呼び出しを呼び出すのに使用できるためです。 SYNC 呼び出しを呼び出すと、すべてのリソースを解放することができます。このようなアプリケーションの設計で、オーバーフロー・インターロック機能の使用時に生じるデッドロックやシリアライゼーションが減少します。
- DBFX 値は、ロードがピークに達したときに OTHREAD 処理で必要とされるバッファーの総数を計算に入れて設定しなければなりません。この値が低すぎると、バッファー待ちが大量に発生する状態が、IMS 高速機能ログ分析報告書に反映されることになります。

バッファーの使用を最適化するには、類似のバッファー使用特性をもつメッセージ処理アプリケーション・プログラムをグループにまとめ、それらを特定のメッセージ・クラスに割り当てて、それらのアプリケーションが領域のバッファーを共用するようにします。

関連概念:

IMS 実行パラメーターの指定 (システム定義)

関連資料:

IMS プロシージャーのパラメーターの説明 (システム定義)

DEDB 制御インターバル (CI) リソースに対する競合

DEDB データへの逐次化アクセスを維持するために、DEDB CI リソースでキューイングが生じます。 2 つの独立したアプリケーション・プログラムが、1 つの特定の CI へのアクセスを同時に要求した場合、一方の要求者は待機する必要があります。

このように待つことによってデッドロックが生じると、そのアプリケーション・プログラムの一方が選ばれ、そのリソースを解放させられ、処理が前の同期点に戻されます。 (注意することは、オーバーフロー・バッファー・インターロックもデッドロックに巻き込まれることがある点です。) デッドロックのために割り込まれるプログラムを選択する規則は、以下のとおりです。

- 1 つ以上のメッセージ・ドリブン・プログラムに関するデッドロックの場合、プログラムのうちの 1 つは異常終了し、前の同期点まで戻され、その後スケジュール変更されます。

- 1 つの BMP が他の BMP のためデッドロックすると、後で同期点に達した BMP が異常終了し、そのリソースが解放され、前の同期点まで戻されて、戻りコードが与えられます。
- デッドロックに DEDB ユーティリティーが関与している場合、もう一方のプログラムは終了し、スケジュール変更されます。 2 つのユーティリティーが同時に同じ DEDB エリアにアクセスすることはできないので、2 つのユーティリティーが 1 つのデッドロックに関与することはできません。

競合とデッドロック状態の数は、以下のステップを行うことによって減らすことができます。

- CI に必要以上のセグメントがないことを確認する。 (CI のサイズは DBD で指定されています。)
- 高速機能 64 ビット・バッファー・マネージャーを使用可能にする。高速機能 64 ビット・バッファー・マネージャーは、バッファーの使用状況と CI サイズの要件に基づいて、高速機能バッファー・プールの定義、割り振り、および管理を動的に行います。また、マルチスレッド化により、複数の従属領域が同時にオーバーフロー・バッファーにアクセスすることを可能にします。高速機能 64 ビット・バッファー・マネージャーを使用可能にするには、DFSDFxxx PROCLIB メンバーで FPBP64=Y を指定します。
- 高速機能 64 ビット・バッファー・マネージャーを使用していない場合は、NBA 値を増やすことでオーバーフロー・バッファー・インターロックの使用を限定する。インターロックと CI の併用がデッドロックに関与することがあります。
- 大半のケースを扱い、OBA を使用して例外条件に対処するのに必要な値に、NBA 値を限定する。あるプログラムの全バッファー割り振り (NBA または NBA と OBA) が超過した場合、バッファー・マネージャーは、このプログラムから未変更バッファーをスチールできます。 CI に関連したすべてのバッファーがスチールされると、PCB が現在これを使用していなければ、CI は解放できます。バッファーのスチールとそれに関連した CI の解放は、全バッファー割り振りを超えることがきっかけになって引き起こされます。 NBA と OBA を最小化することは、CI を適時解放することに役立ち、それによって CI の競合を減らすことができます。
- DEDB にアクセスする BMP が、SYNC 呼び出しを頻繁な間隔で出していることを確認します。 (BMP は、同期点と同期点の間に多数の呼び出しを出すように設計できるため、かなりの数の CI を排他的に制御することができます。)
- DEDB 内で、物理的な順次処理を実行する BMP は、CI の境界を超えるときに (この時点の計算が可能な場合)、SYNC 呼び出しを出す必要があります。これによって、このアプリケーション・プログラムが CI を 1 つより多くは保持していないことを確認できます。

高速機能ログ分析ユーティリティーで生成された報告書は、CI 競合についての統計を出します。

DEDB DASD のスペース切れ

エリアの順次従属部とルート・アドレス可能部で、スペース切れの状態 (その結果としての DEDB エリアの停止を伴う) が生じることがあります。このような状況はシステム全体の操作に影響を及ぼし、長いリカバリー手順が必要になることもあります。

スペース切れ状態の数は、以下の方法によって減らすことができます。

- ランダム化アルゴリズムの設計または通常の再編成を通じて、独立オーバーフロー CI の使用数を制限してみる
- 定期的に順次従属 CI を削除する
- 表示コマンドまたは DEDB POS 呼び出しを用いて、スペース使用状況を追跡する

887 ページの『DEDB 独立オーバーフロー・オンラインの拡張』で説明してある手順を行えば、IMS を停止せずにスペース切れの状態を緩和することができます。

使用可能な実記憶の使用状況

定義されるページ固定ストレージの量は、ストレージの限られたシステムでは重要な考慮事項になります。

関連概念:

310 ページの『論理関係の挿入規則、削除規則、および置き換え規則』

同期点処理と物理ロギング

DEDB の更新は物理ロギングの終了後まで据え置かれるので、出力の「クラスター化」と更新された CI やバッファーの解放が起きることがあります。

BMP の場合では、特に、プログラムを大量のメッセージ処理と同時に実行した場合、任意の同期インターバルで行われる更新の回数を最小限に抑える効果があります。

パフォーマンスの理由から、物理的なログ・レコードは大きくなることが多いので、ロギング・アクティビティーが低調な間の一定時間は、ログ・レコードが書き込まれないことがあります。しかし、IMS は、物理ロギングの定期的な呼び出し間のインターバルを変化させます。このインターバルは、IMS システムのすべてのロギング・アクティビティーに直接関係します。 (アクティビティーが低調であると、設定されるインターバルが小さくなります。)

WADS/OLDS データ・セットに対する物理的なログ・バッファーまたはチャネルまたは制御装置の競合が小さいために、物理ロギング処理が比較的遅くなることがあります。

高速機能環境では、トランザクション率やロギング・アクティビティーが高くなることがあります。したがって、ロギング処理をサポートする物理構成も、最適のパフォーマンスが得られるように分析し、変更しなければなりません。

出力スレッドに対する競合

定義された各 OTHR は、サービス要求ブロック (SRB) をスケジュールする可能性に対応して、特定の同期インターバルに関連した変更されたバッファーの書き込みを制御できるようにします。

OTHR の値が低いと、出力スレッドを待つ書き込みバッファーのキューイングが生じることがあります。一般に、DEDB の変更をもたらすような従属領域が始動されるごとに 1 つの OTHR を設けるのが最良の方法です。

再処理に伴うオーバーヘッド

メッセージ・ドリブンの環境または、非メッセージ・ドリブンの環境のいずれかで、再処理を実行する必要があると、オーバーヘッドが生じます。

以下の状態は、再処理を必要とします。

- CI と (おそらく) オーバーフロー・インターロックを伴うデッドロック
- 同期点時における検証障害
- 呼び出し時での検証障害のような状態によって生じた、ユーザーによるロールバック

デッドロックの場合、アプリケーション・プログラムは動的バックアウトのために疑似的に異常終了します。プログラム制御サブタスクが消去され、その後、再び生成されます。検証障害またはロールバック呼び出しの場合、再スケジューリングは、保持されているリソースの解放とアプリケーション・プログラムへの戻りだけを伴います。

上記の状態が過剰に起きると、応答時間と合計オーバーヘッドが増加します。異常終了のインターセプトの後にダンプとアプリケーション・プログラムの復元が生じるような状態は、オーバーヘッドを増加させます。

プロセッサーが支配的な場合と入出力が支配的な場合のディスパッチング優先順位

同期インターバル内の MSDB 処理は、プロセッサーが支配的なので、MSDB のみ、または、ほとんど MSDB を処理するアプリケーション・プログラムのディスパッチは DEDB のみ、または、ほとんど DEDB を処理する (入出力が支配的な) プログラムより優先順位が低くなければなりません。

DEDB 上での入出力による DASD 競合

いつものように、DEDB エリアに対する入出力の競合は、パフォーマンスへの制約となります。

この影響を最小限に抑えるには、以下のことを行います。

- 1 つの装置ごとに、頻繁に使用されるエリアの数を限定する。
- ある 1 つの DEDB エリアにアクセスするアプリケーション・プログラムの数を限定する。ここでは、ある 1 つのエリアに対するアクセスを、特定のある 1 つのアプリケーション・プログラムまたは複数のアプリケーション・プログラムに限定するようにトランザクション、入力編集/経路指定出口、およびランダム化アルゴリズムの組み合わせを設計するという方法もあります。
- 適切なアプリケーション・プログラムの設計により、未変更バッファーをスチールする発生率と影響を限定する。バッファーをスチールすると、次の入出力が、スチールされたバッファー/制御インターバルをリカバリーしなければならないことがあります。こうなるのは、最初の検索の後に大量の呼び出しが出された場合、アプリケーション・プログラムの論理によってバッファーの処理が必要になるときです。

多重エリア・データ・セットの読み取りパフォーマンスの維持

エリア・データ・セット (ADS) の複数のコピーを使用する場合、最高の読み取りパフォーマンスを得るためには、RECON データ・セットに登録されている最初の ADS を、ご使用の DASD のうちの最高速の DASD に入れてください。

ADS のそれ以降のコピーは、遅い方の DASD に入れても、読み取りパフォーマンス全体には影響しません。

IMS は、常に、RECON リストに示されている最初の ADS から読み取ろうとします。最初の ADS が利用不能になっているか、または長時間使用中の状態である場合、IMS は、使用可能な ADS が見つかるまで、リスト内の後続の各 ADS から読み取ろうとします。すべての ADS が長時間使用中の場合、IMS は、リストの中の最初の ADS を使用します。

ブロック・レベルのデータ共用を指定するリソース・ロッキングの考慮事項

リソース・ロッキングは、非シスプレックス環境ではローカルに、あるいはシスプレックス環境ではグローバルに発生する可能性があります。

非シスプレックス環境では、ローカル・ロックが以下の 3 つの方法のいずれかで付与されます。

- 下記のいずれかの理由により、即時に付与されます。

  IMS は必要な IRLM ラッチを取得でき、しかもこのリソースに関して他のインタレストがない。

  要求は、他の保有者または待機者のものに対応している。
- 要求が、必要な IRLM ラッチを取得できずに中断されたので、非同期に付与されます。(これは、シスプレックス環境でも発生する可能性があります。) ロックは、ラッチが使用可能になり、しかも下記の 2 つの条件のうちのいずれかが存在する場合に付与されます。

  他に保有者がいない。

  要求は、他の保有者または待機者のものに対応している。
- 要求は、他の保有者または待機者のものに対応せず、それらのインタレストが解放された後で付与されたため、ローカル・ロックは非同期に付与されます。(これは、シスプレックス環境でも発生する可能性があります。)

シスプレックス環境では、グローバル・ロックが以下の 3 つの方法のいずれかで付与されます。

- 次の 2 つのいずれかの理由により、**IRLM** によってローカルに付与されます。

  このリソースに対して他のインタレストがない。

  この IRLM にはインタレストしかなく、この要求はこのシステムで保有者または待機者のものに対応しており、さらに XES はすでにこのリソースについて認識している。
- 下記のいずれかの理由により、**XES** 呼び出しと同期して付与されます。

  このリソースに対して、XES は他のインタレストを示さない。

あるいは、ハッシュ・クラスに対して、XES は SHARE インタレストのみを示す。

- 下記のいずれかの理由により、**XES** 呼び出しと非同期に付与されます。

  IRLM によって XES はハッシュ・クラスに EXCLUSIVE インタレストを示すが、リソース名が一致しない (RMF による FALSE CONTENTION)。

  要求は、他の保有者と共存できず、それらのインタレストが解放された後で CONTENTION 出口により付与される (IRLM REAL CONTENTION)。

リソース名ハッシュ・ルーチン

高速機能リソース名ハッシュ・ルーチンは、IRLM が使用するハッシュ値を生成します。

このようなルーチンの名前は、UHASH= 始動パラメーターを使用して指定できますが、無視されます。UHASH の値は、IMS によって常に DBFLHSH0 に設定されます。

IMS 提供の高速機能リソース名ハッシュ・ルーチン (DBFLHSH0) では、相対 CI 番号で暗黙に示された値の範囲を広げる、という技法も用います。それには、31 ビットの CI 番号の一部を、データベースの DMCB 番号から得た値、およびそのエリア番号と以下のように組み合わせます。つまり、DMCB 番号のビット 11 から 15 が、エリア番号のビット 7、6、5、4、3 と XOR され、組み合わせ 5 ビット位置番号を作ります。(エリア番号のビットを逆の順序で使用すると、DMCB 番号とエリア番号の両方が、この組み合わせた値を変化させるのに役立ちます。)

相対 CI 番号は次のとおりです (ビット 0 から 15 は使用されません)。

- ビット 16 から 20 は、組み合わせた値と XOR されます。
- ビット 21 から 25 は、組み合わせた値と XOR されます。
- ビット 26 から 29 は、変更なしで使用されます。
- ビット 30 と 31 は、使用されません。したがって、GHT 項目として使用されるハッシュ相対 CI 番号は、4 つの CI を表します。

ハッシュ・リソース名は次のとおりです。

- ハッシュ相対 CI のビット 16 から 29 は、IRLM に渡されたハッシュ値の 18 から 31 になります。
- ハッシュ値のビット 18 から 26 は、リソース・ハッシュ・テーブル (RHT) への変位として使用されます。
- ビット 18 から 31 は、GHT への変位として使用されます。

第 30 章 データベースの変更

データベース構造は、再編成ユーティリティーやその他の方法を使ってさまざまな形で変更することができます。

時間の経過と共に、ユーザーの要件が変化し、データベース設計の変更を余儀なくされる場合があります。あるいは、新しいオプションまたは機能あるいは異なるオプションまたは機能を使用しようと選択する場合もあります。あるいはおそらく、単に、データベースの構造を形成するより効率のよい方法が見つかったという場合もあります。

データベースの変更をする場合、普通、1 つの単純な変更をデータベースに加えるというだけにとどまりません。例えば、セグメント・タイプを追加すると共に、副次索引を追加する必要があることがあります。このトピックには、それぞれの種類の変更を加えるための手順を記載しています。

一度に複数の変更を加える場合は、 838 ページの『データ・セット・グループの数の変更』を参照してください。ここに記載されている一連のフローチャートをこのトピックに示す個々の手順と併用すれば、ある種の複数の変更をデータベースに加えるときの手順がわかります。

重要: 既存の MSDB の DBD が変更されると、データベースのセグメントが変更されなくても、ヘッダー情報 (BHDR) が変更されることがあります。この場合、MSDBCPx データ・セットの中のヘッダーは無効か、あるいは長さに誤りがあります。 MSDB ヘッダーを変更すると、メッセージ DFS2593I が出されます。 MSDB PROCLIB メンバーで ABND=Y を指定すると、ABENDU1012 も発行されます。この問題を修正するには、ウォーム・スタートまたはコールド・スタート時に MSDBLOAD オプション を使用して、MSDBINIT データ・セットから MSDB をロードします。

関連概念:

709 ページの『再編成ユーティリティー』

707 ページの『データベースの再編成』

925 ページの『第 31 章 データベース・タイプの変換』

IMS 環境でのデータ共用 (システム管理)

関連タスク:

708 ページの『オフラインでのデータベースの再編成』

レコード・セグメントの変更

レコード・セグメントを様々な方法で変更することができます。セグメント・タイプの追加、削除、または移動、セグメントのサイズの変更、およびセグメントに保管されているデータの変更などができます。

関連概念:

770 ページの『データベース・レコードの階層構造の変更』

関連タスク:

916 ページの『レコード・セグメントの変更』

セグメント・タイプの追加

セグメント・タイプをデータベースに追加する方法はいくつかあります。

セグメント・タイプをデータベースに追加する方法は以下のとおりです。

- 再編成ユーティリティーによるアンロードおよび再ロード
- アンロードまたは再ロードを行わない
- ユーザー固有のアンロード・プログラムと再ロード・プログラムを使用する

再編成ユーティリティーによるアンロードおよび再ロード

再編成ユーティリティーを使用してセグメント・タイプをデータベース・レコードに追加できる場合があります。

次のような場合には、再編成ユーティリティーを用いてセグメント・タイプをデータベース・レコードに追加することができます。

- 追加するセグメント・タイプが階層内のある 1 つのパスの最下位レベルにある。以下の図は、既存のデータベース・レコード (実線) と新しいセグメント・タイプを追加できる場所 (点線) を示したものです。
- データベース・レコード内のセグメントの既存の相対順序が変わらない。言い換えれば、既存の親と子の関係を変えることはできません。
- 既存のセグメント名が変わらない。

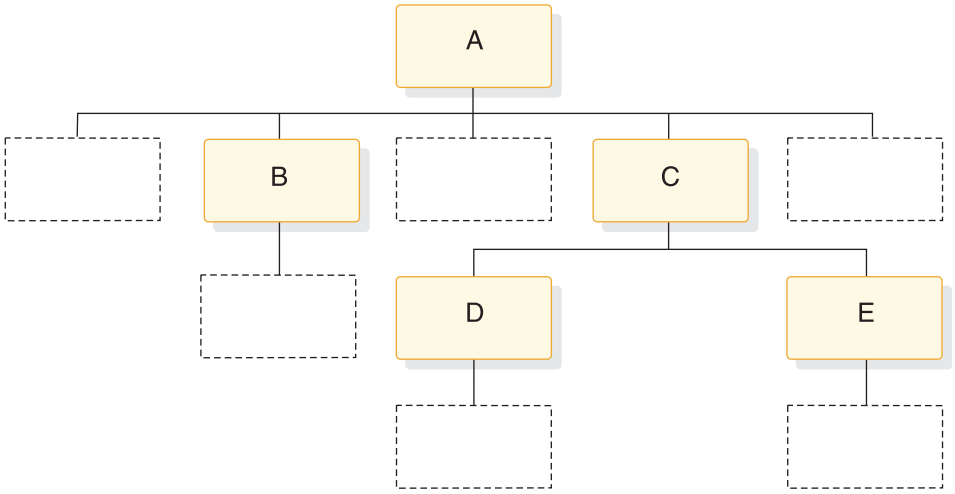


図 276. データベース・レコードの中でのセグメント・タイプを追加できる場所

再編成ユーティリティーを用いてセグメント・タイプをデータベースに追加するには、次の手順を行ってください。

1. 加えようとしている変更がアプリケーション・プログラムの中のコードに影響を及ぼすかどうかを判別します。コードに影響がある場合は、アプリケーション・プログラムに、必要な変更を加えます。
2. 既存の DBD を用いて、データベースをアンロードします。

3. 新しい DBD をコーディングします。新しいセグメント・タイプのための SEGM = ステートメントを DBD に追加する必要があります。アンロードと再ロードの間では、データベースは更新できません。
4. 加えようとしている変更がアプリケーション・プログラムの中のコードに影響を及ぼす場合には、このアプリケーション・プログラムの PSB に必要な変更を行います。
5. ACB を動的に作成するのではなく、事前に作成しておく場合は、ACB を作成し直します。
6. データベース・スペースを計算し直します。 それを行う必要があるのは、その変更を加えることにより、データベース・スペースの必要量が変わってくるからです。
7. 非 VSAM データ・セットの場合は、古いデータベース・スペースを削除し、新しいデータベース・スペースを定義します。 VSAM データ・セットでは、古いクラスターに割り振られているスペースを削除して、新しいクラスターのスペースを定義します。
8. 新しい DBD を用いて、データベースを再ロードします。データベースを再ロードしたらすぐに、このデータベースのイメージ・コピーを作成します。
9. データベースで論理関係または副次索引を使用している場合には、接頭部情報を解決するために、再ロードの前後で再編成ユーティリティーを実行する必要があります。
10. 新しいセグメント・タイプをデータベースに挿入するためのアプリケーション・プログラムをコーディングし、それを実行します。

関連タスク:

603 ページの『データベースの最小サイズの見積もり』

711 ページの『再編成ユーティリティーを使用するオフライン再編成』

アンロードまたは再ロードを行わない

次のような状況がある場合には、データベースをアンロードせずに、データベース・レコードにセグメント・タイプを追加することができます。

- HISAM データベースにおいては、追加するセグメント・タイプは階層の中の最後のセグメントでなければなりません。さらに、追加するセグメント・タイプが既存の論理レコードの中に収まらなければなりません。
- HD データベースにおいても、追加するセグメント・タイプは階層の中の最後のセグメントでなければなりません。新しいセグメント・タイプの親は階層ポインターを使用する必要があります。また、そのセグメント・タイプがデータ・セット・グループの中の最大のセグメント・タイプであってはなりません。

アンロードおよび再ロードを行わずに、セグメント・タイプをデータベースに追加するには、次の手順を行ってください。

1. 加えようとしている変更がアプリケーション・プログラムの中のコードに影響を及ぼすかどうかを判別します。コードに影響がある場合には、必ずそのコードを変更します。
2. 新しい DBD をコーディングします。新しいセグメント・タイプのための SEGM = ステートメントを DBD に追加する必要があります。

3. 加えようとしている変更がアプリケーション・プログラムの中のコードに影響を与える場合には、このアプリケーション・プログラムの PSB に必要な変更を行います。 DB/DC データ・ディクショナリーをもっている場合には、そのディクショナリーを使用して、DBD の変更の影響を受けるアプリケーション・プログラムと PCB を判別することができます。
4. ACB を動的に作成するのではなく、事前に作成しておく場合は、ACB を作成し直します。
5. 新しいセグメント・タイプをデータベースに挿入するためのアプリケーション・プログラムをコーディングし、これを実行します。

ユーザー固有のアンロード・プログラムと再ロード・プログラムを使用する

セグメント・タイプを追加するためにデータベースをアンロードする必要があり、かつ再編成ユーティリティーを使用できない場合は、ユーザーが独自のアンロード・プログラムと再ロード・プログラムを作成しなければなりません。

一般に、新規のセグメント・タイプによって、データベース内の既存のセグメントの階層構造またはそれらのセグメント間の関係が変わる場合は、独自のアンロード・プログラムと再ロード・プログラムを作成する必要があります。

セグメント・タイプの削除

IMS 付属の再編成ユーティリティー、またはユーザー独自のアンロードおよび再ロード・プログラムを使用して、データベースからセグメント・タイプを削除することができます。

次のような場合には、再編成ユーティリティーを用いて、セグメント・タイプをデータベースから削除することができます。

- データベース・レコード内のセグメントの既存の相対順序が変わらない。言い換えれば、既存の親と子の関係を変えることはできません。
- 既存のセグメント名が変わらない。

セグメント・タイプをデータベースから削除するために再編成ユーティリティーを使用するには、次の手順を行ってください。

1. 削除しようとしているセグメント・タイプのすべてのオカレンスを削除するためのアプリケーション・プログラムをコーディングし、実行します。データベースをアンロードする前に、アプリケーション・プログラムをコーディングし、実行しなければなりません。
2. 加えようとしている変更がアプリケーション・プログラムの中のコードに影響を及ぼすかどうかを判別します。コードに影響がある場合には、必ずそのコードを変更します。
3. 既存の DBD を用いて、データベースをアンロードします。
4. 新しい DBD をコーディングします。 SEGM = ステートメントを、次に挙げるものに関する DBD から除去する必要があります。
   - 削除しようとしているセグメント・タイプ
   - 削除されたセグメントの子

5. 加えようとしている変更がアプリケーション・プログラムの中のコードに影響を与える場合には、このアプリケーション・プログラムの PSB に必要な変更を行います。 DB/DC データ・ディクショナリーをもっている場合には、そのディクショナリーを使用して、DBD の変更の影響を受けるアプリケーション・プログラムと PCB を判別することができます。
6. データベース・スペースを計算し直します。 それを行う必要があるのは、その変更を加えることにより、データベース・スペースの必要量が変わってくるからです。
7. ACB を動的に作成するのではなく、事前に作成しておく場合は、ACB を作成し直します。
8. 非 VSAM データ・セットの場合は、古いデータベース・スペースを削除して、新しいデータベース・スペースを定義します。 VSAM データ・セットでは、古いクラスターに割り振られているスペースを削除して、新しいクラスターのスペースを定義します。
9. 新しい DBD を用いて、データベースを再ロードします。データベースを再ロードしたらすぐに、このデータベースのイメージ・コピーを作成することを忘れないでください。
10. データベースで論理関係または副次索引を使用している場合には、接頭部情報を解決するために、再ロードの前後で再編成ユーティリティーを実行する必要があります。

関連タスク:

603 ページの『データベースの最小サイズの見積もり』

711 ページの『再編成ユーティリティーを使用するオフライン再編成』

セグメント・タイプの移動

セグメント・タイプは、再編成ユーティリティーを用いて移動することはできないので、セグメント・タイプを移動するために、ユーザーが独自のアンロード・プログラムおよび再ロード・プログラムを作成しなければなりません。

セグメント・サイズの変更

再編成ユーティリティーを用いて、セグメント・タイプの終わりのところで、セグメント・サイズを大きくしたり小さくしたりすることができます。

セグメント・サイズを大きくする場合には、セグメントの終わりにデータを追加していきます。セグメント・サイズを小さくする場合には、IMS がセグメントの終わりのデータを切り捨てます。

セグメント・サイズを大きくする場合、そのセグメントが再ロードされたとき、その終わりに何があるか予期できません。また、セグメントの終わりに新しいデータを追加するには、データベースを再ロードした後に、ユーザー独自のプログラムを用いて行わなければなりません。

セグメント・サイズを大きくしたり小さくしたりするには、次の手順を行ってください。

1. 加えようとしている変更がアプリケーション・プログラムの中のコードに影響を及ぼすかどうかを判別します。コードに影響がある場合には、必ずそのコードを変更します。
2. 既存の DBD を用いて、データベースをアンロードします。 HISAM データベースを変更している場合には、構造を変更するのに HISAM ユーティリティーは使えないので、HD UNLOAD/RELOAD ユーティリティーを使用しなければなりません。
3. 新しい DBD をコーディングします。新しいセグメント・サイズを反映するように DBD の中の SEGM ステートメントの BYTES = オペランドを変更する必要があります。 DBD の中に FIELD ステートメントでコード化されているデータをセグメントから除去しようとしている場合、その FIELD ステートメントを除去する必要があります。データをセグメントに追加しようとしており、しかも、そのデータがアプリケーション・プログラムの SSA の中で参照される場合には、FIELD ステートメントをコーディングする必要があります。アンロードと再ロードの間では、データベースは更新できません。
4. 加えようとしている変更がアプリケーション・プログラムの中のコードに影響を与える場合には、このアプリケーション・プログラムの PSB に必要な変更を行います。 DB/DC データ・ディクショナリーをもっている場合には、そのディクショナリーを使用して、DBD の変更の影響を受けるアプリケーション・プログラムと PCB を判別することができます。
5. ACB を動的に作成するのではなく、事前に作成しておく場合は、ACB を作成し直します。
6. データベース・スペースを計算し直します。それを行うのは、その変更を加えた結果として、データベース・スペースの必要量が変わってくるからです。
7. 非 VSAM データ・セットの場合は、古いデータベース・スペースを削除して、新しいデータベース・スペースを定義します。 VSAM データ・セットでは、古いクラスターに割り振られているスペースを削除して、新しいクラスターのスペースを定義します。
8. 新しい DBD を用いて、データベースを再ロードします。データベースを再ロードしたらすぐに、このデータベースのイメージ・コピーを作成します。
9. データベースで論理関係または副次索引を使用している場合には、接頭部情報を解決するために、再ロードの前後で再編成ユーティリティーを実行する必要があります。

関連タスク:

603 ページの『データベースの最小サイズの見積もり』

711 ページの『再編成ユーティリティーを使用するオフライン再編成』

可変長セグメントの追加または可変長セグメントへの変換

データベースの中の一部の選択されたセグメントを固定長から可変長に変更する場合、あるいは、データベース全体を可変長セグメントに変換する場合には、2 つの方法があります。

いずれの方法を使用するにしても、変換の目的は、可変長にしようとしているセグメントにサイズ・フィールドを設け、次いで DBD においてこのセグメントを可変長として定義することです。

関連概念:

423 ページの『可変長セグメント』

方法 1. セグメントまたはデータベースの変換

可変長セグメントを使用するようにセグメントまたはデータベースを変換する方法 1 では、新しい DBD を作成し、独自のアプリケーション・プログラム (各セグメントを検索して 2 バイトのサイズ・フィールドを追加し、そのセグメントをデータベースに戻す) を作成します。

選択したいくつかのセグメントまたはデータベース全体をこの方法で変換するためには、次の手順を行ってください。

1. 加えようとしている変更がアプリケーション・プログラムの中のコードに影響を及ぼすかどうかを判別します。コードに影響がある場合には、必ずそのコードを変更します。
2. 可変長となるセグメント・タイプおよびそのサイズを識別する新しい DBD をコーディングし、これを生成します。
3. 加えようとしている変更がアプリケーション・プログラムの中のコードに影響を与える場合には、このアプリケーション・プログラムの PSB に必要な変更を行います。 DB/DC データ・ディクショナリーをもっている場合には、そのディクショナリーを使用して、DBD の変更の影響を受けるアプリケーション・プログラムと PCB を判別することができます。
4. ACB を動的に作成するのではなく、事前に作成しておく場合は、ACB を作成し直します。
5. 可変長にしたいすべてのセグメントをデータベースから順次検索するためのプログラムを作成します。このプログラムは、検索された各セグメントに 2 バイトのサイズ・フィールドを付け加え、次いでこの各セグメントを再びデータベースの中に挿入しなければなりません。

方法 2. セグメントまたはデータベースの変換

可変長セグメントを使用するようにセグメントまたはデータベースを変換する方法 2 では、2 つの DBD (一時的な DBD と最終的な DBD) を作成し、データベースをアンロードし、さらにデータベースの再ロード中にセグメント編集/圧縮出口ルーチンを使用して 2 バイトのサイズ・フィールドを追加します。

選択したいくつかのセグメントまたはデータベース全体をこの方法で変換するためには、次の手順を行ってください。

1. 加えようとしている変更がアプリケーション・プログラムの中のコードに影響を及ぼすかどうかを判別します。コードに影響がある場合には、必ずそのコードを変更します。
2. 既存の DBD を用いて、データベースをアンロードします。
3. 新しい (一時的) DBD をコーディングしこれを生成します。この DBD では、可変長セグメントに変換されているすべてのセグメントを固定長セグメントを指定してください。また、変換しようとしている各セグメントに、セグメント編集/圧縮出口ルーチンの使用も指定します。(ステップ 9 で説明するように、この一時的な DBD は、既存の固定長セグメントにサイズ・フィールドを追加するために使用されます。)

4. 加えようとしている変更がアプリケーション・プログラムの中のコードに影響を与える場合には、このアプリケーション・プログラムの PSB に必要な変更を行います。 DB/DC データ・ディクショナリーをもっている場合には、そのディクショナリーを使用して、DBD の変更の影響を受けるアプリケーション・プログラムと PCB を判別することができます。
5. ACB を動的に作成するのではなく、事前に作成しておく場合は、ACB を作成し直します。
6. 必要に応じてデータベース・スペースを計算し直します。それを行う必要があるのは、この変更を加えた結果として、データベース・スペースの必要量が変わってくる場合です。
7. 非 VSAM データ・セットの場合は、古いデータベース・スペースを削除して、新しいデータベース・スペースを定義します。 VSAM データ・セットでは、古いクラスターに割り振られているスペースを削除して、新しいクラスターのスペースを定義します。
8. セグメント編集/圧縮出口ルーチンの出口となる編集ルーチンを作成します。この編集ルーチンは、このルーチンが受け取った各セグメントにサイズ・フィールドを 1 つずつ付け加えてください。
9. 一時的な DBD を用いて、データベースを再ロードします。変換する必要のあるセグメント・タイプの各オカレンスがロードのために提示されるたびに、ユーザーの編集ルーチンが制御を受け取り、サイズ・フィールドをこのセグメントに付け加えます。この編集ルーチンが制御を返すと、このセグメントがデータベースにロードされます。ロードしたらすぐにユーザーのデータベースのイメージ・コピーを作成することを忘れないでください。
10. データベースで論理関係または副次索引を使用している場合には、接頭部情報を解決するために、再ロードの前後で再編成ユーティリティーを実行する必要があります。
11. データベースをロードした後、データベースの中の可変長セグメントであるセグメントのタイプおよびそのサイズを指定する新しい DBD をコーディングし、これを生成します。

関連概念:

427 ページの『セグメント編集/圧縮出口ルーチン』

関連タスク:

711 ページの『再編成ユーティリティーを使用するオフライン再編成』

セグメント内のデータの変更 (セグメントの終わりのデータ以外)

再編成ユーティリティーを用いて、セグメントの中のデータのサイズを大きくしたり小さくしたりすることはできません。フィールドのサイズを変更するためには、ユーザーが独自のアンロード・プログラムおよび再ロード・プログラムを作成しなければなりません。

セグメント内のデータの位置の変更

再編成ユーティリティーを用いて、セグメントの中のデータの位置を変更することはできません。

この種の変更を行うためには、ユーザーが独自のアンロード・プログラムおよび再ロード・プログラムを作成するか、フィールド・レベル・センシティビティーを使用するか、あるいは DB セグメント再構成ユーティリティーを含む、IMS High Performance Pointer Checker for z/OS のようなツールを使用しなければなりません。

関連資料: IMS High Performance Pointer Checker for z/OS については、「*IMS High Performance Pointer Checker for z/OS* ユーザーズ・ガイド」を参照してください。

関連概念:

437 ページの『フィールド・レベル・センシティビティー』

セグメントの名前の変更

セグメントに関して、セグメントの名前は変更するが、それ以外は変更しない場合、データベースのアンロードや再ロードは必要ありません。それ以外の場合はデータベースを再編成します。

セグメントの名前を変更するには、以下を実行する必要があります。

1. セグメントを定義している DBD ステートメントでセグメント名を変更します。
2. セグメントが論理関係または副次索引で参照されている場合、DBD の中で論理データベースまたは索引データベースのセグメント名を変更する必要があります。
3. DBD 生成ユーティリティーを実行して DBDLIB を更新します。
4. セグメントを参照する PSB ステートメントのセグメント名を変更します。
5. PSB 生成ユーティリティーを実行して PSBLIB を更新します。
6. ACB 保守ユーティリティーとオンライン変更を実行して、ACBLIB を更新します。
7. セグメントを参照するすべてのアプリケーション・コードを更新します。

関連資料:

データベース記述 (DBD) 生成ユーティリティー (システム・ユーティリティー)

プログラム仕様ブロック (PSB) 生成ユーティリティー (システム・ユーティリティー)

アプリケーション制御ブロック保守ユーティリティー (システム・ユーティリティー)

論理関係の追加

既存のデータベースに論理関係を追加することができます。

このトピックでは、論理的な関係を持つデータベースを既存のデータベースに追加する例やそのための手順を説明します。論理関係を追加する必要のあるすべての状況を、このトピックで説明しているわけではありません。この節で説明する例がユ

ーザーの特定の要件に当てはまらない場合でも、これらの例を見れば、次のことを決定するために十分な情報を収集することができます。

- 既存のデータベースへの論理関係の追加が可能であるかどうか
- 論理関係の追加方法

例では、論理親をルート・セグメントとして示してあります (ただしこれは要件ではありません)。論理親が階層内のより下のレベルのものである場合にも、この例は有効です。

既存のデータベースに論理関係の追加を行う場合、常にテスト・データベース上でその変更を行うようにする必要があります。その変更を徹底的にテストした後で初めて、本稼働データベースを用いて論理関係の追加を実施します。

以下に挙げる例では、次のような規則を用いています。

- 既存のデータベースは、実線で表示します。
- 追加されるデータベースは、点線で表示します。
- 追加されるデータベースには、論理親と論理子の関係にラベルを付けてあります。 この関係は、LP と LC でラベル付けしています。
- 用語 DBX、DBY、および DBZ は、第 1 データベース、第 2 データベース、および第 3 データベースをそれぞれ指しています。

関連概念:

265 ページの『第 15 章 論理関係の作成』

関連タスク:

603 ページの『データベースの最小サイズの見積もり』

621 ページの『ロード・プログラムの作成』

論理関係の追加例

以下の各例では、図を使って論理関係の追加を説明します。また、各例で示されている論理関係を追加するために必要なステップも示してあります。

例 1. DBX が存在し、DBY を追加する

例 1 では、A のセグメント接頭部にカウンター・フィールドを追加するために、DBX を再編成しなければなりません。

例 1 を以下の図に示します。

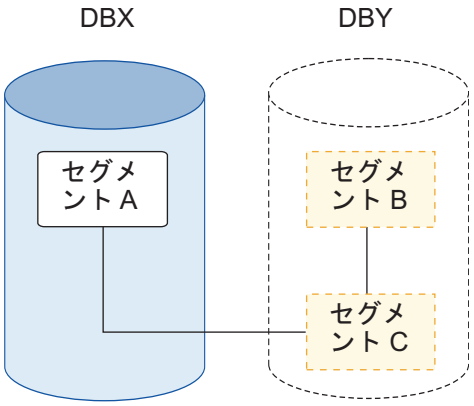


図 *277. DBX* が存在し、*DBY* を追加する

DBX の制御ステートメントに、DBIL を指定する必要があります。以下の手順において、セグメント A のカウンター・フィールドは、セグメント C の番号を示すように更新されます。なぜなら、セグメント C がユーザーのロード・プログラムによってロードされるからです。

例 **1** の手順

1. 加えようとしている変更がアプリケーション・プログラムの中のコードに影響を及ぼすかどうかを判別します。コードに影響がある場合には、必ずそのコードを変更します。
2. 既存の DBD および HD アンロード・ユーティリティーを用いて、 DBX をアンロードします。
3. DBX および DBY のために新しい DBD をコーディングします。
4. 加えようとしている変更が、アプリケーション・プログラムの中のコードに影響を及ぼした場合は、そのアプリケーション・プログラムの PSB に必要な変更を行います。 DB/DC データ・ディクショナリーをもっている場合には、そのディクショナリーを使用して、DBD の変更の影響を受けるアプリケーション・プログラムと PCB を判別することができます。
5. ACB を動的に作成するのではなく、事前に作成しておく場合は、ACB を作成し直します。
6. DBX のデータベース・スペースを計算し直し、DBY のためのスペースを計算します。
7. 非 VSAM データ・セットの場合は、古いデータベース・スペースを削除して、新しいデータベース・スペースを定義します。 VSAM データ・セットでは、古いクラスターに割り振られているスペースを削除して、新しいクラスターのスペースを定義します。
8. DBX および DBY の制御ステートメントに DBIL を指定して、事前再編成ユーティリティーを実行します。
9. 新しい DBD および HD 再ロード・ユーティリティーを用いて、DBX を再ロードします。
10. 初期ロード・プログラムを用いて、DBY をロードします。
11. ステップ 9 およびステップ 10 からの出力である DFSURWF1 作業ファイルを入力として、接頭部解決ユーティリティーを実行します。

12. ステップ 11 からの出力である DFSURWF3 作業ファイルを入力として、接頭部更新ユーティリティーを実行します。
13. ロードしたらすぐに、両方のデータベースのイメージ・コピーを作成することを忘れないでください。

関連タスク:

300 ページの『論理 DBD での論理関係の指定』

例 2. DBX および DBY が存在し、DBZ を追加する

この例では、セグメント C の接頭部にカウンターがあります。セグメント C の接頭部にあるカウンター用の新しい値を計算するために DBX および DBY を再編成しなければなりません。

例 2 を以下の図に示します。

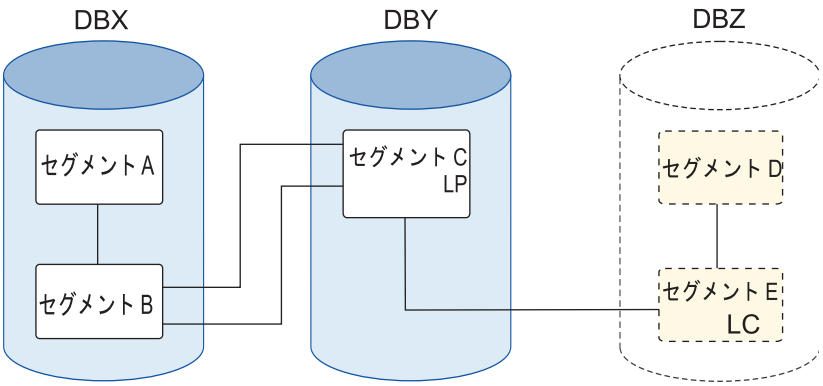


図 *278. DBX および DBY が存在し、DBZ を追加する*

DBX および DBY の制御ステートメントでは、DBIL を指定します。以下の手順において、セグメント A のカウンター・フィールドは、セグメント C の番号を示すように更新されます。なぜなら、セグメント C がユーザーのロード・プログラムによってロードされるからです。

例 2 の手順

1. 加えようとしている変更がアプリケーション・プログラムの中のコードに影響を及ぼすかどうかを判別します。コードに影響がある場合には、必ずそのコードを変更します。
2. 既存の DBD および HD アンロード・ユーティリティーを用いて、DBX および DBY をアンロードします。
3. DBY および DBZ のための新しい DBD をコーディングします。
4. 加えようとしている変更が、アプリケーション・プログラムの中のコードに影響を及ぼした場合は、そのアプリケーション・プログラムの PSB に必要な変更を行います。 DB/DC データ・ディクショナリーをもっている場合には、そのディクショナリーを使用して、DBD の変更の影響を受けるアプリケーション・プログラムと PCB を判別することができます。
5. ACB を動的に作成するのではなく、事前に作成しておく場合は、ACB を作成し直します。

6. DBX および DBY のデータベース・スペースを計算し直し、DBZ のためのスペースを計算します。
7. 非 VSAM データ・セットの場合は、古いデータベース・スペースを削除して、新しいデータベース・スペースを定義します。 VSAM データ・セットでは、古いクラスターに割り振られているスペースを削除して、新しいクラスターのスペースを定義します。
8. DBX、DBY、および DBZ の制御ステートメントに DBIL を指定して、事前再編成ユーティリティーを実行します。
9. 新しい DBD および HD 再ロード・ユーティリティーを用いて、DBX および DBY を再ロードします。
10. 初期ロード・プログラムを用いて、DBZ をロードします。
11. ステップ 9 およびステップ 10 からの出力である DFSURWF1 作業ファイルを入力として、接頭部解決ユーティリティーを実行します。
12. ステップ 11 からの出力である DFSURWF3 作業ファイルを入力として、接頭部更新ユーティリティーを実行します。
13. ロードしたらすぐに 3 つのデータベースすべてのイメージ・コピーを作成することを忘れないでください。

関連タスク:

300 ページの『論理 DBD での論理関係の指定』

例 3. DBX および DBY が存在し、DBZ を追加する

例 3 では、セグメント C の接頭部にカウンター・フィールドを追加するために、DBY を再編成しなければなりません。 DBY の制御ステートメントに、DBIL を指定する必要があります。論理親 (セグメント C) の初期ロード (DBIL) は、初期ロード (論理子の DBIL) を前提としているために DBX を再編成しなければなりません。

例 3 を以下の図に示します。

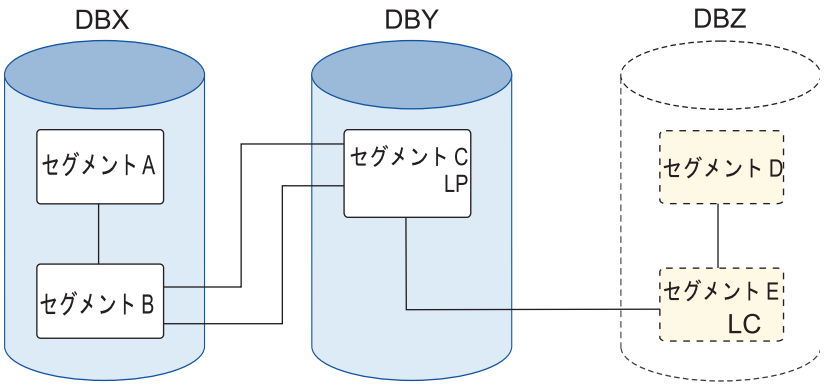


図 279. *DBX* および *DBY* が存在し、*DBZ* を追加する

この例の手順 (および、すべての条件と考慮事項) は、例 2 の場合とまったく同じです。

例 4. DBX および DBY が存在し、DBZ を追加する

例 4 では、手順、およびすべての条件と考慮事項は、例 2 の場合とまったく同じです。

例 4 を以下の図に示します。

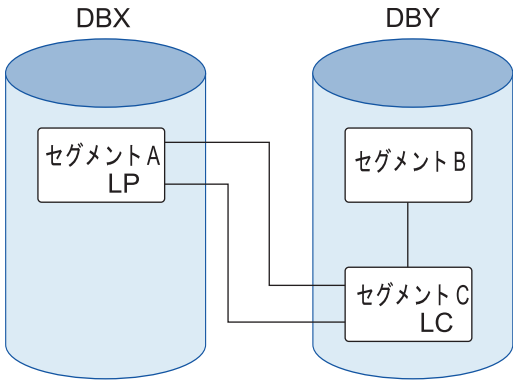


図 *280. DBX* および *DBY* が存在し、*DBZ* を追加する

例 5. DBX が存在し、DBY を追加する

例 5 では、セグメント A の接頭部に論理子ポインターを追加するために、DBX を再編成しなければなりません。

例 5 を以下の図に示します。

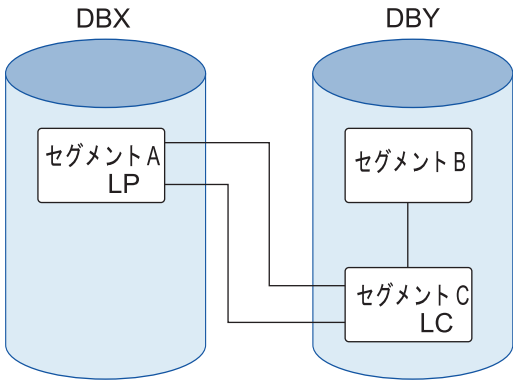


図 *281. DBX* が存在し、*DBY* を追加する

手順

1. 加えようとしている変更がアプリケーション・プログラムの中のコードに影響を及ぼすかどうかを判別します。コードに影響がある場合には、必ずそのコードを変更します。
2. 既存の DBD および HD アンロード・ユーティリティーを用いて、 DBX をアンロードします。
3. DBX および DBY のために新しい DBD をコーディングします。

4. 加えようとしている変更が、アプリケーション・プログラムの中のコードに影響を及ぼした場合は、そのアプリケーション・プログラムの PSB に必要な変更を行います。 DB/DC データ・ディクショナリーをもっている場合には、そのディクショナリーを使用して、DBD の変更の影響を受けるアプリケーション・プログラムと PCB を判別することができます。
5. ACB を動的に作成するのではなく、事前に作成しておく場合は、ACB を作成し直します。
6. DBX のデータベース・スペースを計算し直し、DBY のためのスペースを計算します。
7. 非 VSAM データ・セットの場合は、古いデータベース・スペースを削除して、新しいデータベース・スペースを定義します。 VSAM データ・セットでは、古いクラスターに割り振られているスペースを削除して、新しいクラスターのスペースを定義します。
8. DBX の制御ステートメントに DBR を、DBY の制御ステートメントに DBIL を指定して、事前再編成ユーティリティーを実行します。
9. 新しい DBD および HD 再ロード・ユーティリティーを用いて、DBX を再ロードします。
10. 初期ロード・プログラムを用いて、DBY をロードします。
11. ステップ 9 およびステップ 10 からの出力である DFSURWF1 作業ファイルを入力として、接頭部解決ユーティリティーを実行します。
12. ステップ 11 からの出力である DFSURWF3 作業ファイルを入力として、接頭部更新ユーティリティーを実行します。
13. ロードしたらすぐに、両方のデータベースのイメージ・コピーを作成することを忘れないでください。

関連タスク:

300 ページの『論理 DBD での論理関係の指定』

例 6. DBX および DBY が存在し、DBZ を追加する

例 6 では、セグメント C の接頭部に論理子ポインターを追加するために、DBY を再編成しなければなりません。

例 6 を以下の図に示します。

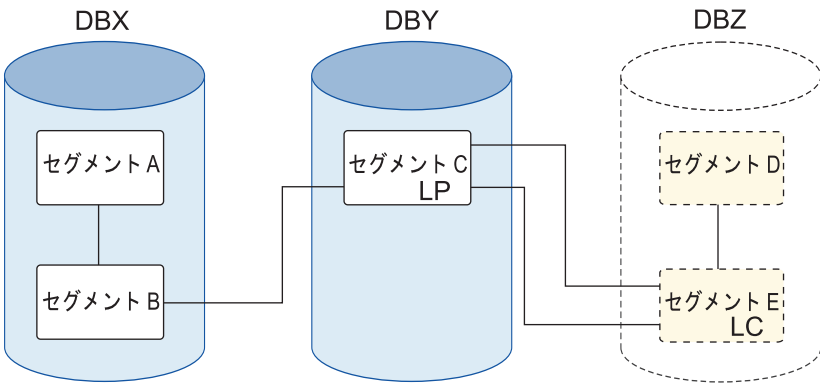


図 *282. DBX* および *DBY* が存在し、*DBZ* を追加する

セグメント C の接頭部に論理子ポインターを追加するために、以下の 3 つの手順のいずれかを使用する必要があります。

DBY の再編成の際の手順 (セグメント B にシンボリック・ポインターが入っている場合)

1. 加えようとしている変更がアプリケーション・プログラムの中のコードに影響を及ぼすかどうかを判別します。コードに影響がある場合には、必ずそのコードを変更します。
2. 既存の DBD および HD アンロード・ユーティリティーを用いて、 DBY をアンロードします。
3. DBY および DBZ のための新しい DBD をコーディングします。
4. 加えようとしている変更が、アプリケーション・プログラムの中のコードに影響を及ぼした場合は、そのアプリケーション・プログラムの PSB に必要な変更を行います。 DB/DC データ・ディクショナリーをもっている場合には、そのディクショナリーを使用して、DBD の変更の影響を受けるアプリケーション・プログラムと PCB を判別することができます。
5. ACB を動的に作成するのではなく、事前に作成しておく場合は、ACB を作成し直します。
6. DBY のデータベース・スペースを計算し直し、DBZ のためのスペースを計算します。
7. 非 VSAM データ・セットの場合は、古いデータベース・スペースを削除して、新しいデータベース・スペースを定義します。 VSAM データ・セットでは、古いクラスターに割り振られているスペースを削除して、新しいクラスターのスペースを定義します。
8. DBY の制御ステートメントに DBR を、DBZ の制御ステートメントに DBIL を指定して、事前再編成ユーティリティーを実行します。(この事前再編成ユーティリティーからの出力は、DBX のスキャンが必要であることを示しています。)
9. 新しい DBD および HD 再ロード・ユーティリティーを用いて、DBY を再ロードします。
10. 初期ロード・プログラムを用いて、DBZ をロードします。
11. ステップ 9 およびステップ 10 からの出力である DFSURWF1 作業ファイルを入力として、接頭部解決ユーティリティーを実行します。
12. ステップ 11 からの出力である DFSURWF3 作業ファイルを入力として、接頭部更新ユーティリティーを実行します。
13. ロードしたらすぐに、両方のデータベースのイメージ・コピーを作成することを忘れないでください。

DBY の再ロード時に、セグメント C の各オカレンスごとにタイプ 00 レコードが 2 つずつ生成されます。その 1 つのタイプ 00 レコードには、DBZ という名前の論理子データベースが含まれており、そのレコードは、セグメント E に対して生成されたタイプ 10 レコードと一致しています。もう 1 つのタイプのレコードには、DBX という名前の論理子データベースが含まれています。 DBX はスキャンも再編成も行われていないので、セグメント B については一致するタイプ 10 レコードは生成されません。この場合、接頭部解決ユーティリティーはメッセージ DFS878 を生成します。印刷された 00 レコードが DBY および DBX を指している限り、そ

のメッセージを無視することができます。 DBY および DBZ を対象とするメッセージが出されたら、そのすべてについて調べてください。

DBY の再編成と DBX のスキャンの際の手順 (セグメント B に直接ポインターが入っている場合)

1. 加えようとしている変更がアプリケーション・プログラムの中のコードに影響を及ぼすかどうかを判別します。コードに影響がある場合には、必ずそのコードを変更します。
2. 既存の DBD および HD アンロード・ユーティリティーを用いて、 DBY をアンロードします。
3. DBY および DBZ のための新しい DBD をコーディングします。
4. 加えようとしている変更が、アプリケーション・プログラムの中のコードに影響を及ぼした場合は、そのアプリケーション・プログラムの PSB に必要な変更を行います。 DB/DC データ・ディクショナリーをもっている場合には、そのディクショナリーを使用して、DBD の変更の影響を受けるアプリケーション・プログラムと PCB を判別することができます。
5. ACB を動的に作成するのではなく、事前に作成しておく場合は、ACB を作成し直します。
6. DBY のデータベース・スペースを計算し直し、DBZ のためのスペースを計算します。
7. 非 VSAM データ・セットの場合は、古いデータベース・スペースを削除して、新しいデータベース・スペースを定義します。 VSAM データ・セットでは、古いクラスターに割り振られているスペースを削除して、新しいクラスターのスペースを定義します。
8. DBY の制御ステートメントに DBR を、DBZ の制御ステートメントに DBIL を指定して、事前再編成ユーティリティーを実行します。(この事前再編成ユーティリティーからの出力は、DBX のスキャンが必要であることを示しています。)
9. DBX を対象としてスキャン・ユーティリティーを実行します。
10. 新しい DBD および HD 再ロード・ユーティリティーを用いて、DBY を再ロードします。
11. 初期ロード・プログラムを用いて、DBZ をロードします。
12. ステップ 9、10 および 11 からの出力である DFSURWF1 作業ファイルを入力として、接頭部解決ユーティリティーを実行します。
13. ステップ 12 からの出力である DFSURWF3 作業ファイルを入力として、接頭部更新ユーティリティーを実行します。
14. ロードしたらすぐに、両方のデータベースのイメージ・コピーを作成することを忘れないでください。

DBX および DBY の再編成の際の手順

1. 加えようとしている変更がアプリケーション・プログラムの中のコードに影響を及ぼすかどうかを判別します。コードに影響がある場合には、必ずそのコードを変更します。
2. 既存の DBD および HD アンロード・ユーティリティーを用いて、DBX および DBY をアンロードします。

3. DBY および DBZ のための新しい DBD をコーディングします。
4. 加えようとしている変更が、アプリケーション・プログラムの中のコードに影響を及ぼした場合は、そのアプリケーション・プログラムの PSB に必要な変更を行います。 DB/DC データ・ディクショナリーをもっている場合には、そのディクショナリーを使用して、DBD の変更の影響を受けるアプリケーション・プログラムと PCB を判別することができます。
5. ACB を動的に作成するのではなく、事前に作成しておく場合は、ACB を作成し直します。
6. DBX および DBY のデータベース・スペースを計算し直し、DBZ のためのスペースを計算します。
7. 非 VSAM データ・セットの場合は、古いデータベース・スペースを削除して、新しいデータベース・スペースを定義します。 VSAM データ・セットでは、古いクラスターに割り振られているスペースを削除して、新しいクラスターのスペースを定義します。
8. DBX および DBY の制御ステートメントに DBR を、DBZ の制御ステートメントに DBIL を指定して、事前再編成ユーティリティーを実行します。(この事前再編成ユーティリティーからの出力は、DBX のスキャンが必要であることを示しています。)
9. 新しい DBD および HD 再ロード・ユーティリティーを用いて、DBX および DBY を再ロードします。
10. 初期ロード・プログラムを用いて、DBZ をロードします。
11. ステップ 9 およびステップ 10 からの出力である DFSURWF1 作業ファイルを入力として、接頭部解決ユーティリティーを実行します。
12. ステップ 11 からの出力である DFSURWF3 作業ファイルを入力として、接頭部更新ユーティリティーを実行します。
13. ロードしたらすぐに 3 つのデータベースすべてのイメージ・コピーを作成することを忘れないでください。

関連タスク:

300 ページの『論理 DBD での論理関係の指定』

例 7. DBX および DBY が存在し、DBZ を追加する

例 7 では、セグメント C の接頭部に論理子ポインターを追加するために、DBY を再編成しなければなりません。セグメント C からセグメント B を指す論理子ポインターは、アンロードされません。したがって、DBX を再編成するか、またはこれをスキャンする必要があります。セグメント A の接頭部に論理子ポインターを追加するために、DBX を再編成しなければなりません。

例 7 を以下の図に示します。

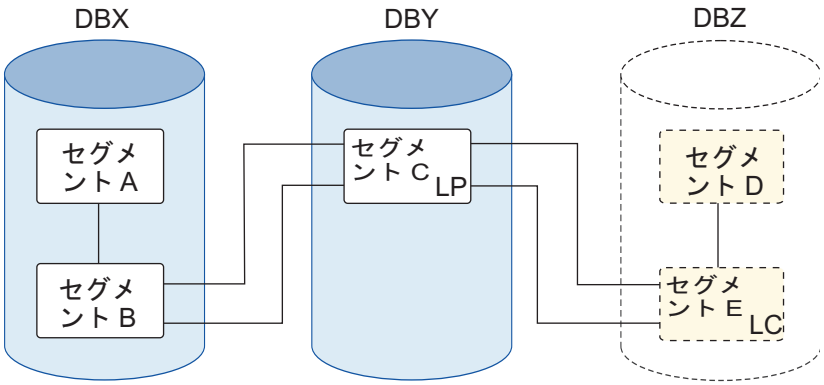


図 *283. DBX* および *DBY* が存在し、*DBZ* を追加する

スキャンを用いる手順

1. 加えようとしている変更がアプリケーション・プログラムの中のコードに影響を及ぼすかどうかを判別します。コードに影響がある場合には、必ずそのコードを変更します。
2. 既存の DBD および HD アンロード・ユーティリティーを用いて、 DBY をアンロードします。
3. DBY および DBZ のための新しい DBD をコーディングします。
4. 加えようとしている変更が、アプリケーション・プログラムの中のコードに影響を及ぼした場合は、そのアプリケーション・プログラムの PSB に必要な変更を行います。 DB/DC データ・ディクショナリーをもっている場合には、そのディクショナリーを使用して、DBD の変更の影響を受けるアプリケーション・プログラムと PCB を判別することができます。
5. ACB を動的に作成するのではなく、事前に作成しておく場合は、ACB を作成し直します。
6. DBY のデータベース・スペースを計算し直し、DBZ のためのスペースを計算します。
7. 非 VSAM データ・セットの場合は、古いデータベース・スペースを削除して、新しいデータベース・スペースを定義します。 VSAM データ・セットでは、古いクラスターに割り振られているスペースを削除して、新しいクラスターのスペースを定義します。
8. DBY の制御ステートメントに DBR を、DBZ の制御ステートメントに DBIL を指定して、事前再編成ユーティリティーを実行します。(この事前再編成ユーティリティーからの出力は、DBX のスキャンが必要であることを示しています。)
9. DBX を対象としてスキャン・ユーティリティーを実行します。
10. 新しい DBD および HD 再ロード・ユーティリティーを用いて、 DBY を再ロードします。
11. 初期ロード・プログラムを用いて、DBZ をロードします。
12. ステップ 9、10 および 11 からの出力である DFSURWF1 作業ファイルを入力として、接頭部解決ユーティリティーを実行します。
13. ステップ 12 からの出力である DFSURWF3 作業ファイルを入力として、接頭部更新ユーティリティーを実行します。

14. ロードしたらすぐに、両方のデータベースのイメージ・コピーを作成することを忘れないでください。

DBX および **DBY** の再編成の際の手順

1. 加えようとしている変更がアプリケーション・プログラムの中のコードに影響を及ぼすかどうかを判別します。コードに影響がある場合には、必ずそのコードを変更します。
2. 既存の DBD および HD アンロード・ユーティリティーを用いて、DBY と DBY をアンロードします。
3. DBY および DBZ のための新しい DBD をコーディングします。
4. 加えようとしている変更が、アプリケーション・プログラムの中のコードに影響を及ぼした場合は、そのアプリケーション・プログラムの PSB に必要な変更を行います。 DB/DC データ・ディクショナリーをもっている場合には、そのディクショナリーを使用して、DBD の変更の影響を受けるアプリケーション・プログラムと PCB を判別することができます。
5. ACB を動的に作成するのではなく、事前に作成しておく場合は、ACB を作成し直します。
6. DBX および DBY のデータベース・スペースを計算し直し、DBZ のためのスペースを計算します。
7. 非 VSAM データ・セットの場合は、古いデータベース・スペースを削除して、新しいデータベース・スペースを定義します。 VSAM データ・セットでは、古いクラスターに割り振られているスペースを削除して、新しいクラスターのスペースを定義します。
8. DBX および DBY の制御ステートメントに DBR を、DBZ の制御ステートメントに DBIL を指定して、事前再編成ユーティリティーを実行します。(この事前再編成ユーティリティーからの出力は、DBX のスキャンが必要であることを示しています。)
9. 新しい DBD および HD 再ロード・ユーティリティーを用いて、DBX および DBY を再ロードします。
10. 初期ロード・プログラムを用いて、DBZ をロードします。
11. ステップ 9 およびステップ 10 からの出力である DFSURWF1 作業ファイルを入力として、接頭部解決ユーティリティーを実行します。
12. ステップ 11 からの出力である DFSURWF3 作業ファイルを入力として、接頭部更新ユーティリティーを実行します。
13. ロードしたらすぐに、両方のデータベースのイメージ・コピーを作成することを忘れないでください。

関連タスク:

300 ページの『論理 DBD での論理関係の指定』

例 8. DBX および DBY が存在し、DBZ を追加する

例 8 では、セグメント C の接頭部に論理子ポインターを追加するために、DBY を再編成しなければなりません。この例の手順、およびすべての条件と考慮事項は、例 6 の場合とまったく同じです。

例 8 を以下の図に示します。

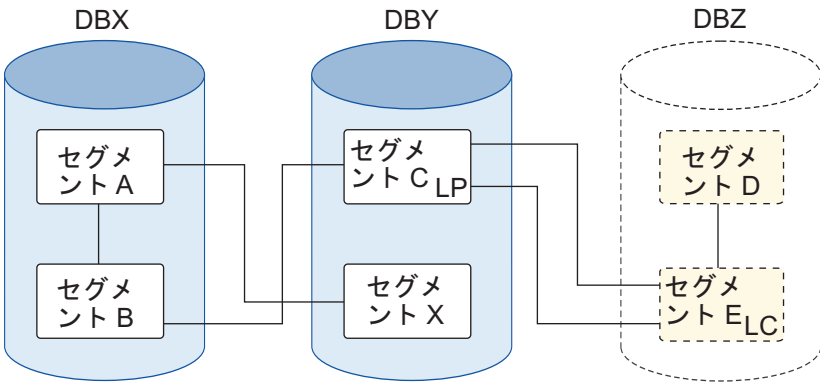


図 *284. DBX* および *DBY* が存在し、*DBZ* を追加する

例 9. DBY が存在し、DBZ を追加する

DBY を再編成しなければなりません。初期ロード・プログラムを用いて、DBZ をロードする必要があります。 DBY の制御ステートメントに、DBIL を指定する必要があります。 DBY の制御ステートメントに DBR を指定してはなりません。

例 9 を以下の図に示します。

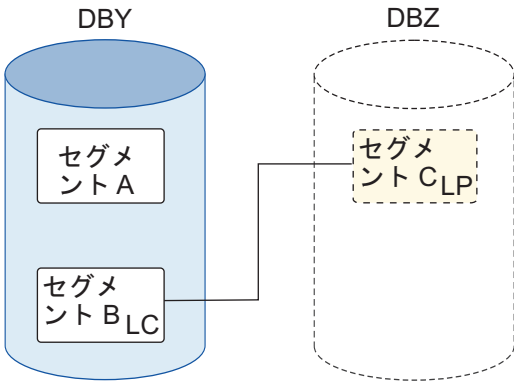


図 *285. DBY* が存在し、*DBZ* を追加する

手順

1. 加えようとしている変更がアプリケーション・プログラムの中のコードに影響を及ぼすかどうかを判別します。コードに影響がある場合には、必ずそのコードを変更します。
2. 既存の DBD および HD アンロード・ユーティリティーを用いて、 DBY をアンロードします。
3. DBY および DBZ のための新しい DBD をコーディングします。
4. 加えようとしている変更が、アプリケーション・プログラムの中のコードに影響を及ぼした場合は、そのアプリケーション・プログラムの PSB に必要な変更を行います。 DB/DC データ・ディクショナリーをもっている場合には、そ

のディクショナリーを使用して、DBD の変更の影響を受けるアプリケーション・プログラムと PCB を判別することができます。

5. ACB を動的に作成するのではなく、事前に作成しておく場合は、ACB を作成し直します。
6. DBY のデータベース・スペースを計算し直し、DBZ のためのスペースを計算します。
7. 非 VSAM データ・セットの場合は、古いデータベース・スペースを削除して、新しいデータベース・スペースを定義します。 VSAM データ・セットでは、古いクラスターに割り振られているスペースを削除して、新しいクラスターのスペースを定義します。
8. DBY および DBZ の制御ステートメントに DBIL を指定して、事前再編成ユーティリティーを実行します。
9. 新しい DBD および HD 再ロード・ユーティリティーを用いて、 DBY を再ロードします。
10. 初期ロード・プログラムを用いて、DBZ をロードします。
11. ステップ 9 およびステップ 10 からの出力である DFSURWF1 作業ファイルを入力として、接頭部解決ユーティリティーを実行します。
12. ステップ 11 からの出力である DFSURWF3 作業ファイルを入力として、接頭部更新ユーティリティーを実行します。
13. ロードしたらすぐに、両方のデータベースのイメージ・コピーを作成することを忘れないでください。

例 10. DBY が存在し、DBZ を追加する

例 10 では、セグメント D のキーがセグメント X の中の所定の位置にある場合、セグメント X は論理子と見なすことができます。論理親 (セグメント D) の初期ロード (DBIL) は、論理子の初期ロード (DBIL) を前提としているため、DBY を再編成しなければなりません。

この例では、セグメント X に対して、シンボリック・ポインターあるいは直接ポインターを使用できます。どのような状況においても、DBY のための制御ステートメントに DBR を指定してはなりません。 こう指定すると、再ロード・ユーティリティーは、セグメント D のための作業レコードを生成しません。セグメント D の中の論理子ポインターが解決されないことになるからです。この例のための手順 (および、すべての条件と考慮事項) は、例 9 の場合とまったく同じです。

例 10 を以下の図に示します。

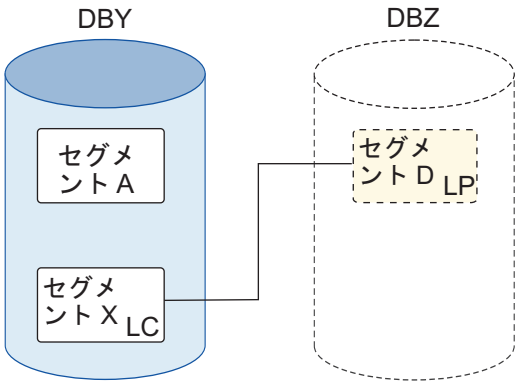


図 *286. DBY* が存在し、*DBZ* を追加する

例 11. DBX および DBY が存在し、DBZ を追加する

例 11 では、DBX および DBY を再編成しなければなりません。初期ロード・プログラムを用いて、DBZ をロードする必要があります。 DBZ (論理親のデータベース) の制御ステートメントに DBIL を指定しなければならないので、DBY (論理子のデータベース) に対しても DBIL を指定しなければなりません。 DBY は論理親データベースでもあります。したがって、DBX (論理子データベース) の制御ステートメントに DBIL を指定しなければなりません。

この例の手順、およびすべての条件と考慮事項は、例 2 の場合とまったく同じです。

例 11 を以下の図に示します。

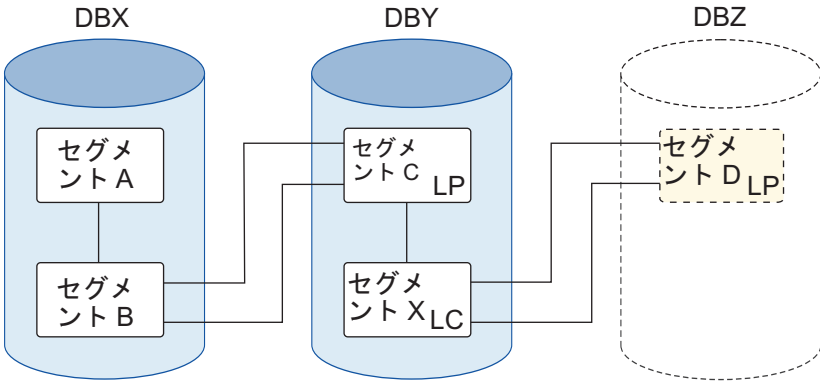


図 *287. DBX* および *DBY* が存在し、*DBZ* を追加する

例 12. DBX および DBY が存在し、DBZ を追加する

例 12 では、セグメント B はシンボリック・ポインターを持っています。この例の手順、およびすべての条件と考慮事項は、例 2 の場合とまったく同じです。

例 12 を以下の図に示します。

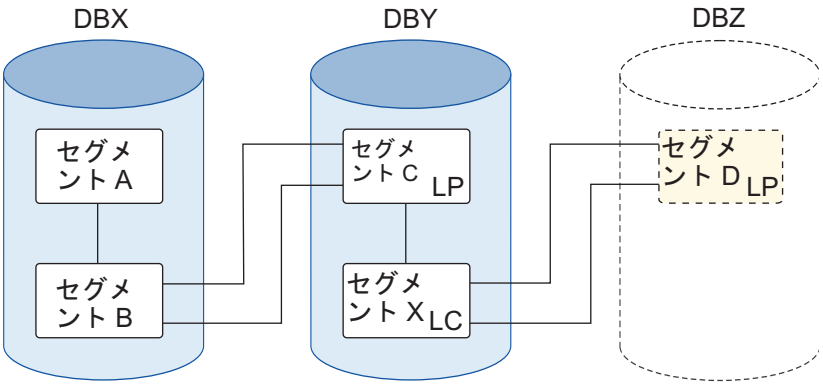


図 *288. DBX* および *DBY* が存在し、*DBZ* を追加する

例 13. DBX および DBY が存在し、セグメント Y および DBZ を追加する

例 13 を以下の図に示します。

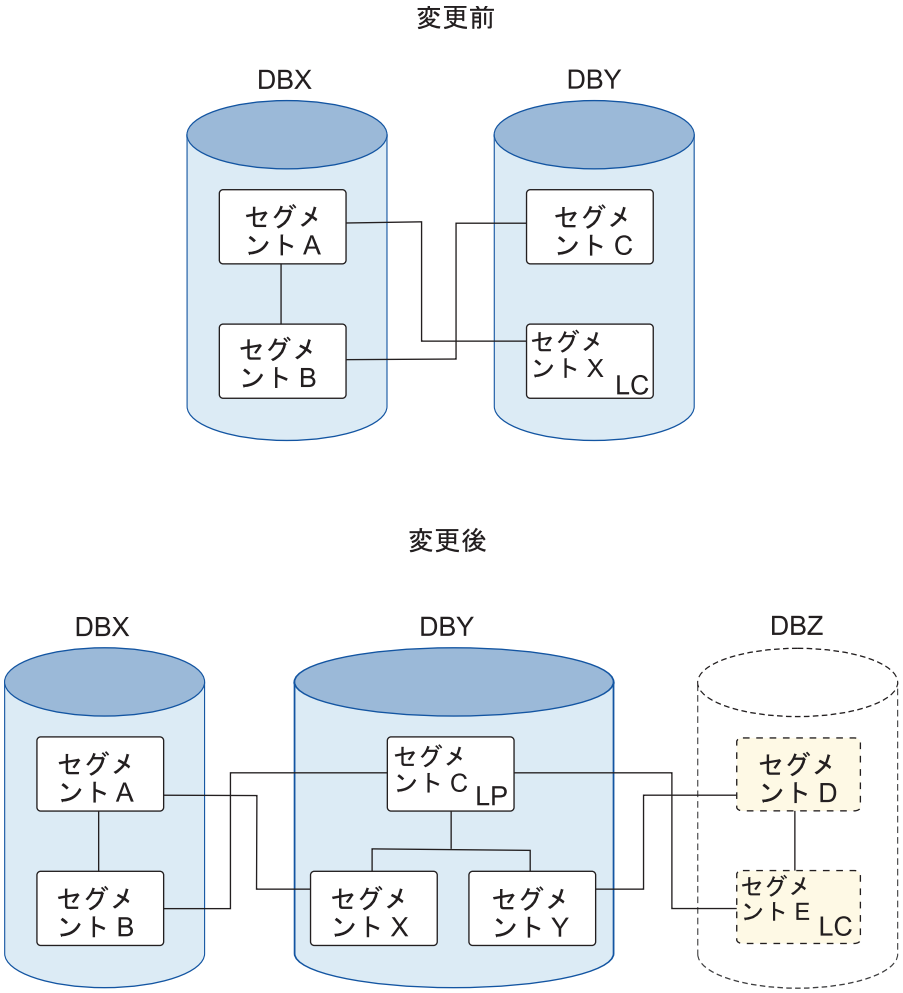


図 *289. DBX* および *DBY* が存在し、セグメント *Y* および *DBZ* を追加する

1. 加えようとしている変更がアプリケーション・プログラムの中のコードに影響を及ぼすかどうかを判別します。コードに影響がある場合には、必ずそのコードを変更します。
2. 既存の DBD および HD アンロード・ユーティリティーを用いて、 DBX をアンロードします。
3. DBY および DBZ のための新しい DBD をコーディングします。
4. 加えようとしている変更が、アプリケーション・プログラムの中のコードに影響を及ぼした場合は、そのアプリケーション・プログラムの PSB に必要な変更を行います。 DB/DC データ・ディクショナリーをもっている場合には、そのディクショナリーを使用して、DBD の変更の影響を受けるアプリケーション・プログラムと PCB を判別することができます。
5. ACB を動的に作成するのではなく、事前に作成しておく場合は、ACB を作成し直します。
6. DBX および DBY のデータベース・スペースを計算し直し、DBZ のためのスペースを計算します。

7. 非 VSAM データ・セットの場合は、古いデータベース・スペースを削除して、新しいデータベース・スペースを定義します。 VSAM データ・セットでは、古いクラスターに割り振られているスペースを削除して、新しいクラスターのスペースを定義します。
8. DBX、DBY、および DBZ の制御ステートメントに DBIL を指定して、事前再編成ユーティリティーを実行します。
9. 新しい DBD および HD 再ロード・ユーティリティーを用いて、DBX を再ロードします。
10. 初期ロード・プログラムを用いて、DBY および DBZ をロードします。
11. ステップ 9 およびステップ 10 からの出力である DFSURWF1 作業ファイルを入力として、接頭部解決ユーティリティーを実行します。
12. ステップ 11 からの出力である DFSURWF3 作業ファイルを入力として、接頭部更新ユーティリティーを実行します。
13. ロードしたらすぐに、両方のデータベースのイメージ・コピーを作成することを忘れないでください。

関連タスク:

300 ページの『論理 DBD での論理関係の指定』

IMS 論理関係の変更

データベースを再編成することによって論理関係を追加または変更するために必要なステップを以下の表で概説します。

表の列 1 には開始時の論理関係およびポインター、列 2 には再編成するデータベース、列 3 から列 6 には追加する論理関係およびポインターがそれぞれ示されており、これらに基づいて、行う必要があることが示されています。列 1 は「変更前」列です。列 3 から列 6 は「変更後」列です。

表では、既存の論理関係およびポインターと、新しい論理関係およびポインターが交差している場所に、以下の情報が記載されています。

- いつ論理的な関係を持つデータベースをスキャンしなければならないか
- いつ論理関係の双方の側を再編成しなければならないか
- いつ接頭部解決および接頭部更新ユーティリティーを実行しなければならないか

接頭部解決ユーティリティーおよび接頭部更新ユーティリティーを実行する必要がない場合は、表のセルに「完了」と記載されています。

この図が適用されるのは再編成に対してだけです。データベースを最初にロードする場合には、作業データ・セットが生成されるたびに、必ず接頭部解決ユーティリティーおよび接頭部更新ユーティリティーを実行しなければなりません。

以下の表は、データベース・ポインターが変更されるかどうかにかかわりなく、すべての再編成の状況を載せています。この図を使用するにあたっては、物理対の両方向論理関係を 2 つの単一方向論理関係として取り扱わなければなりません。特に断りがない限り、事前再編成ユーティリティーが実行されている場合には、再編成されたデータベースに DBR を指定しなければなりません。

次の 2 つの例は、この表の使い方の指針になります。

例 1: 単一方向シンボリック・ポインターを持つデータベースを、ポインターを変更せずに再編成する

単一方向シンボリック・ポインターを持つデータベースがあり、ポインターは変更しないものとします。以下の表の左側の「変更前」列で、単一方向シンボリック・ポインターを見つけて、行を右方向に、単一方向シンボリック・ポインターの「変更後」列と交差する所までたどっていきます。この図は、次のデータベースを再編成するためにしなければならないことを示しています。

- 論理親が入っているデータベース
- 論理子が入っているデータベース
- 必要であれば、両方のデータベース

この 3 つのいずれの場合にも、接頭部解決ユーティリティーあるいは接頭部更新ユーティリティーは実行する必要はありません (これが「完了」という言葉の意味です。)

例 2: 再編成時に両方向シンボリック・ポインターを両方向直接ポインターに変更する

両方向シンボリック・ポインターを持つデータベースがあり、このポインターを両方向直接ポインターに変更する必要があるものとします。「変更前」列で、両方向シンボリック・ポインターの行を見つけます。「変更後」列で、両方向直接ポインターを見つけます。表の行と列が交差した場所に、次のことが示されています。

- 論理親ポインターは論理子データベースの論理子セグメントの中で生成しなければならないので、論理親データベースのみを再編成することはできません。
- 論理子データベースを再編成することはできます。論理子データベースをスキャンするには、論理親データベースをスキャンしなければならなりません。事前再編成ユーティリティーの制御ステートメントは、論理子データベースに対して DBIL を指定する必要があります。接頭部解決ユーティリティーおよび接頭部更新ユーティリティーも実行しなければなりません。
- 両方のデータベースを再編成することもできます。この場合には、事前再編成ユーティリティーの制御ステートメントは、論理子データベースには DBIL を指定し、論理親データベースには DBR を指定しなければなりません。ここでもまた、接頭部解決ユーティリティーおよび接頭部更新ユーティリティーを実行しなければなりません。

表 *82.* データベース再編成中に論理関係およびポインターを変換するために必要なステップ：

変更前: 既存の論理関係およびポインター	再編成するデータベース	変更後: 新しい論理関係およびポインター			
		単一方向シンボリック・ポインター	単一方向直接ポインター	両方向シンボリック・ポインター	両方向直接ポインター
単一方向でシンボリック・ポインターをもつ	論理親データベースのみ	終了[1]	無効。記号 LP ポインターが現在存在しており、直接 LP ポインターが論理子データベースに追加されなければならないため。	1. 論理子データベースをスキャンする。 2. 接頭部の解決および更新を実行する。 注: 論理子セグメントには、再編成しなければ LT ポインターは入らない。	無効。直接 LP および LT ポインターを論理子データベースに置かなければならないため。
	論理子データベースのみ	完了	1. 論理親データベースをスキャンする。 2. 接頭部の解決および更新を実行する。 論理子データベースには DBIL を指定する。	無効。カウンターがいま存在しており、LCF/LCL ポインターを論理親データベースに置かなければならないため。	無効。カウンターがいま存在しており、LCF/LCL ポインターを論理親データベースに置かなければならないため。
	両方のデータベース	終了[2]	接頭部の解決および更新を実行する。 論理子データベースには DBIL、論理親データベースには DBR を指定する。	接頭部の解決および更新を実行する。 両方のデータベースに DBR を指定する。	接頭部の解決および更新を実行する。 論理子データベースには DBIL、論理親データベースには DBR を指定する。

表 *82.* データベース再編成中に論理関係およびポインターを変換するために必要なステップ *(*続き*)*:

変更前: 既存の論理関係およびポインター	再編成するデータベース	変更後: 新しい論理関係およびポインター			
		単一方向シンボリック・ポインター	単一方向直接ポインター	両方向シンボリック・ポインター	両方向直接ポインター
単一方向で直接ポインターをもつ	論理親データベースのみ	無効。直接 LP ポインターが現在存在しており、記号 LP ポインターが論理子データベースに追加されなければならないため。	1. 論理子データベースをスキャンする。 2. 接頭部の解決および更新を実行する。	無効。直接 LP ポインターが現在存在しており、記号 LP ポインターが論理子データベースに追加されなければならないため。 LT ポインターも論理子データベースに追加されなければならない。	1. 論理子データベースをスキャンする。 2. 接頭部の解決および更新を実行する。 注: 論理子セグメントには、データベースを再編成しなければ LT ポインターは入らない。
	論理子データベースのみ	完了	完了	無効。LCF/LCL ポインターが論理親データベースに置かれなければならないため。	無効。LCF/LCL ポインターが論理親データベースに置かれなければならないため。
	両方のデータベース	終了[2]	接頭部の解決および更新を実行する。	接頭部の解決および更新を実行する。	接頭部の解決および更新を実行する。

表 82. データベース再編成中に論理関係およびポインターを変換するために必要なステップ (続き) :

変更前: 既存の論理関係およびポインター	再編成するデータベース	変更後: 新しい論理関係およびポインター			
		単一方向シンボリック・ポインター	単一方向直接ポインター	両方向シンボリック・ポインター	両方向直接ポインター
両方向でシンボリック・ポインターをもつ	論理親データベースのみ	無効。論理親データベースのカウンターが解決されなくて、現在 LT ポインターが論理子データベースに存在しているため。	無効。記号 LP および LT ポインターが現在存在しており、直接 LP ポインターが論理子データベースに追加されなければならないため。	1. 論理子データベースをスキャンする。 2. 接頭部の解決および更新を実行する。 注: LCF/LCL ポインターはアンロードおよび再ロードされません。	無効。記号 LP ポインターが現在存在しており、直接 LP ポインターが論理子データベースに追加されなければならないため。
	論理子データベースのみ	無効。論理親データベースの中に現在 LCF/LCL ポインターが存在しており、カウンターが論理親データベースに追加されなくてはならないため。	無効。論理親データベースの中に現在 LCF/LCL ポインターが存在しており、カウンターが論理親データベースに追加されなくてはならないため。	1. 論理親データベースをスキャンする。 2. 接頭部の解決および更新を実行する。	1. 論理親データベースをスキャンする。 2. 接頭部の解決および更新を実行する。 3. 論理子データベースには DBIL を指定する。
	両方のデータベース	接頭部の解決および更新を実行する。 論理子データベースには DBIL、論理親データベースには DBR を指定する。	接頭部の解決および更新を実行する。 論理子データベースには DBIL、論理親データベースには DBR を指定する。	接頭部の解決および更新を実行する。	接頭部の解決および更新を実行する。 論理子データベースには DBIL、論理親データベースには DBR を指定する。

表 82. データベース再編成中に論理関係およびポインターを変換するために必要なステップ (続き) :

変更前: 既存の論理関係およびポインター	再編成するデータベース	変更後: 新しい論理関係およびポインター			
		単一方向シンボリック・ポインター	単一方向直接ポインター	両方向シンボリック・ポインター	両方向直接ポインター
両方向で直接ポインターをもつ	論理親データベースのみ	無効。論理子データベースの中に現在直接 LP および LT ポインターが存在しており、記号 LP ポインターが追加されなければならないため。	無効。論理親データベースの中のカウンターが解決されず、また LT ポインターが論理子データベースから除去されないため。	無効。直接 LP ポインターが論理子データベースに存在しており、変更は記号 LP ポインターに対するものであるため。	1. 論理子データベースをスキャンする。 2. 接頭部の解決および更新を実行する。 注: LCF/LCL ポインターはアンロードおよび再ロードされません。
	論理子データベースのみ	無効。LCF/LCL ポインターが論理親データベースに存在しており、カウンターが論理親データベースに追加されなければならないため。	無効。論理親データベースの中に現在 LCF/LCL ポインターが存在しており、カウンターが論理親データベースに追加されなくてはならないため。	1. 論理親データベースをスキャンする。 2. 接頭部の解決および更新を実行する。	1. 論理親データベースをスキャンする。 2. 接頭部の解決および更新を実行する。
	両方のデータベース	接頭部の解決および更新を実行する。 論理子データベースには DBIL、論理親データベースには DBR を指定する。	接頭部の解決および更新を実行する。 論理子データベースには DBIL、論理親データベースには DBR を指定する。	接頭部の解決および更新を実行する。	接頭部の解決および更新を実行する。

注:

1. 事前再編成ユーティリティーは、論理子データベースをスキャンするように指示するため、スキャンが実行されると DFSURWF1 レコードが生成されます。
2. DFSURWF1 レコードが生成されます。ただし、接頭部解決ユーティリティーおよび接頭部更新ユーティリティーを実行する必要はありません。

関連概念:

293 ページの『論理関係のためのシーケンス・フィールドの定義』

既存の論理関係の変更についてのいくつかの制約事項

既存の論理関係を変更するために IMS ユーティリティーを使用することができない場合があります。既存の論理関係を変更することができない場合には、ユーザーが独自にプログラムを作成しなければなりません。

以下のトピックで例を示します。

例 1: 両方向仮想対から両方向物理対への変更

以下の例は、仮想対から物理対への変更を示しています。

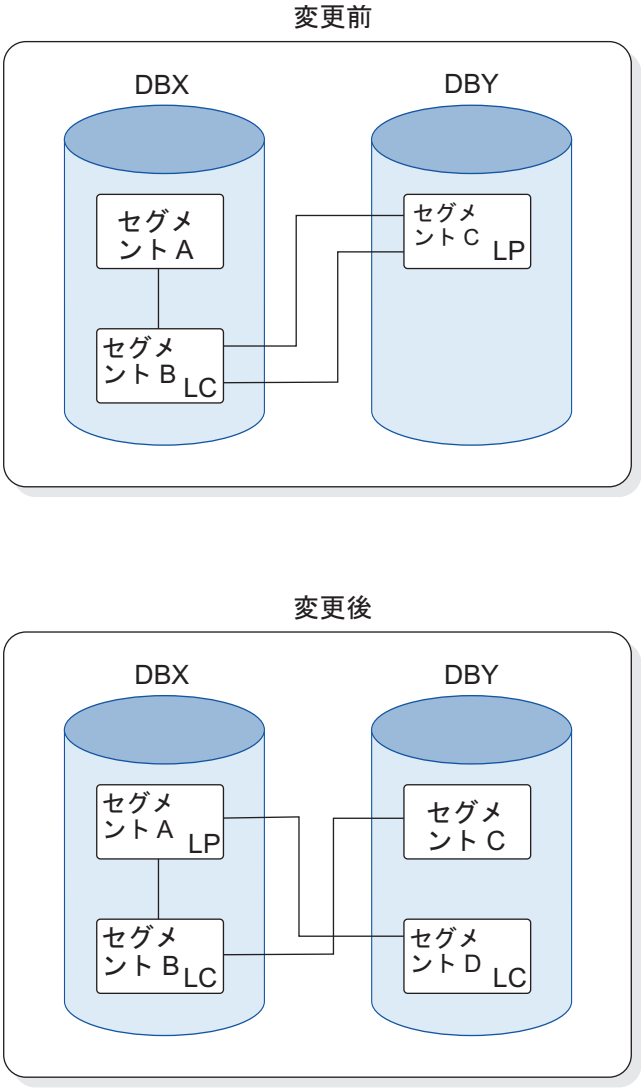


図 *290.* 仮想対から物理対への変更

例 2: 両方向論理関係における実論理子の位置の変更

以下の図は、ある論理的関連データベースから別の論理的関連データベースへの実論理子の位置の変更を示しています。

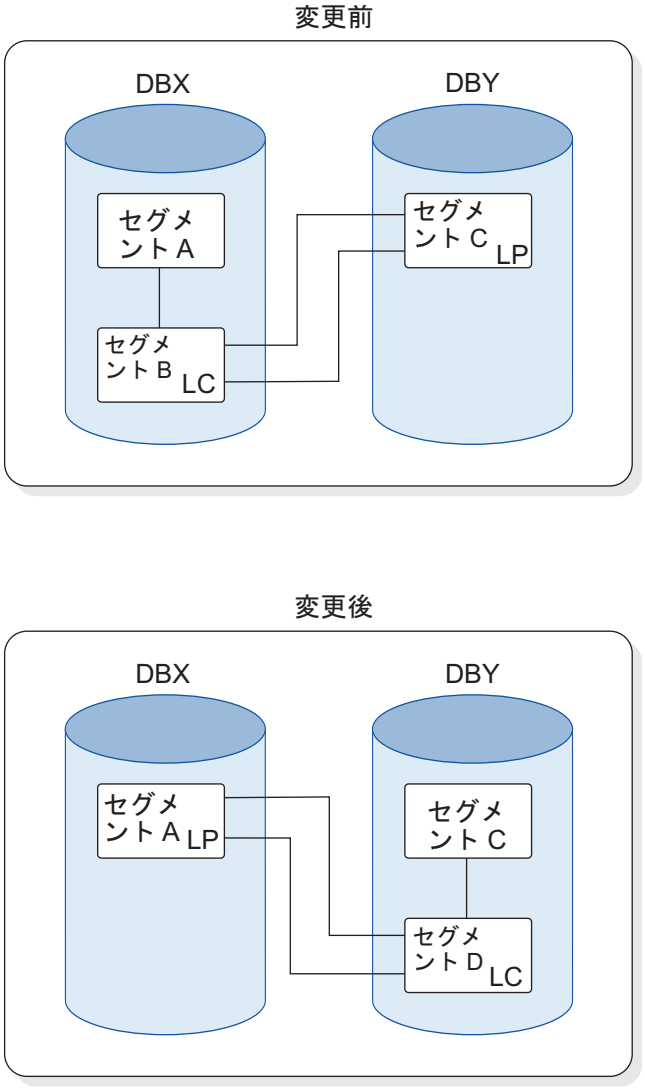


図 *291.* 実論理子について、論理的に関連付けられたあるデータベースから他への位置変更

両方の例の「変更前」には、物理的には削除されていても、論理的には削除されていないセグメント B のオカレンスがあることがあります。言い換えれば、その論理子は論理パスからはアクセスできますが、物理パスからはアクセスできません。DBX のアンロード時には、HD アンロード・ユーティリティーは、論理的でなく物理的に削除されているセグメント B のオカレンスにアクセスすることはできません。したがって、この種の再編成を行う場合には、ユーザーが独自にプログラムを作成しなければなりません。

論理関係を追加する場合のユーティリティーの使用の要約

以下の項目は、IMS ユーティリティーを使用した論理関係の追加についてまとめたものです。

- 初期ロード・プログラムを用いてロードされた論理子を数えることによって、または論理的に関連したデータベースが再編成された場合には、制御ステートメントの DBIL を使用することによって、カウンターの値が増やされます。

- データベースを再編成することにより、カウンターの問題を訂正することができます。カウンターの問題を訂正する場合には、関係するデータベースの制御ステートメントで DBIL を指定しなければなりません。
- LCF ポインターおよび LCL ポインターは、アンロードおよび再ロードされません。 接頭部解決ユーティリティーおよび接頭部更新ユーティリティーによって、これらのポインターを作成し直さなければなりません。
- DBIL が対応するすべての論理子データベースに指定されていない限り、論理親データベースの制御ステートメントには DBIL を指定しないでください。
- シンボリック・ポインターから直接ポインターに変更するためには、論理子データベース用の制御ステートメントで DBIL を指定しなければなりません。

仮想から物理、または物理から仮想への論理親の連結キーの変換

DBDGEN および HD の各再編成ユーティリティーを用いて、論理親連結キーを仮想から物理または物理から仮想へ変換することができます。

論理親の連結キーを仮想から物理へ、または物理から仮想へ変換するには、以下の手順を実行します。

1. 既存の DBD を用いて、データベースをアンロードします。
2. 連結キーの物理/仮想の指定を変更して、新しい DBD をコーディングします。
3. 動的に作成された ACB ではなく事前に作成された ACB を使用する場合には、ACB を作成し直します。
4. データベース・スペースを計算し直します。それを行う必要があるのは、この変更を加えた結果として、データベース・スペースの必要量が変わってくるからです。
5. 非 VSAM データ・セットの場合は、古いデータベース・スペースを削除して、新しいデータベース・スペースを定義します。 VSAM データ・セットでは、古いクラスターに割り振られているスペースを削除して、新しいクラスターのスペースを定義します。
6. データベースで論理関係または副次索引を使用している場合には、接頭部情報を解決するために、再ロードの前後で再編成ユーティリティーを実行する必要があります。
7. 新しい DBD を用いて、データベースを再ロードします。データベースを再ロードしたらすぐに、このデータベースのイメージ・コピーを作成することを忘れないでください。
8. 必要であれば、接頭部情報を解決するために再編成ユーティリティーを実行します。

関連タスク:

711 ページの『再編成ユーティリティーを使用するオフライン再編成』

IMS 索引の変更

全機能データベースまたは DEDB データベースの副次索引を追加または削除できます。

関連概念:

363 ページの『第 16 章 副次索引の作成』

371 ページの『副次索引によるデータベースの階層の再構築』

関連タスク:

838 ページの『索引のドロップ』

全機能データベースへの副次索引の追加

セグメントのキーによって定義されたシーケンス以外のシーケンスでセグメント・タイプを処理するために、全機能データベースに副次索引を追加できます。

副次索引を追加するには、次のようにします。

1. 加えようとしている変更がアプリケーション・プログラムの中のコードに影響を及ぼすかどうかを判別します。コードに影響がある場合には、必ずそのコードを変更してください。
2. 既存の DBD および HD 再編成アンロード・ユーティリティーを使用して、データベースをアンロードします。
3. 2 つの新しい DBD をコーディングします。1 つは既存のデータベース用、もう 1 つは新しい副次索引データベース用です。
4. 加えようとしている変更がアプリケーション・プログラムの中のコードに影響を及ぼす場合には、このアプリケーション・プログラムの PSB に必要な変更を行います。 DB/DC データ・ディクショナリーをもっている場合には、そのディクショナリーを使用して、DBD の変更の影響を受けるアプリケーション・プログラムと PCB を判別することができます。
5. オプション: ACB を動的に作成するのではなく、事前に作成しておく場合は、ACB を作成し直します。
6. 古いデータベース・スペースを削除し、新しいデータベースのスペースを定義する (非 VSAM データ・セット)、あるいは、クラスターに割り振られているスペースを削除して、新しいクラスターのスペースを定義します。さらに、副次索引のスペースを定義します。
7. 新しい DBD および HD 再編成再ロード・ユーティリティーを使用して、既存のデータベースを再ロードします。
8. ステップ 7 からの出力である DFSURWF1 作業ファイルを入力として使用して、データベース接頭部解決ユーティリティーを実行します。
9. ステップ 8 からの出力である DFSURIDX 作業ファイルを入力として使用して、HISAM 再編成アンロード・ユーティリティーを実行します。ユーティリティー制御ステートメントでは、副次索引を作成するために HISAM アンロードを使用することを示してください。
10. ステップ 9 での HISAM アンロードからの出力を入力として使用して、HISAM 再編成再ロード・ユーティリティーを実行します。
11. メイン・データベースの処理に副次索引を使用しない場合でも、JCL を変更して、副次索引データ・セットの DD ステートメントを追加してください。

12. 副次索引を追加する場合は、再編成手順を変更します。
13. 副次索引が指しているデータ・セットを再編成するときは、再編成ユーティリティーを実行して副次索引を再作成する必要があります。

関連概念:

363 ページの『第 16 章 副次索引の作成』

401 ページの『副次索引の定義の例』

関連資料:

副次索引のある例 (システム・ユーティリティー)

新しい基本 DEDB への副次索引の追加

セグメントのキーによって定義されたシーケンス以外のシーケンスでセグメント・タイプを処理するために、新しい 1 次 DEDB データベースに副次索引を追加できます。

副次索引を追加するには、次のようにします。

1. 副次索引が定義されている DEDB 1 次データベースの DBD 定義を作成します。
2. 高速機能副次索引データベースの DBD 定義を作成します。
3. 高速機能副次索引データベースを介してアクセスされる 1 次 DEDB データベースの PCB の中に、高速機能副次索引 DBD 名を指定した PROCSEQD= パラメーターを追加します。
4. 高速機能副次索引が定義されている定義を使用して、DBDGEN ユーティリティーを実行します。
5. 高速機能副次索引が定義されている定義を使用して、PSBGEN ユーティリティーを実行します。
6. 高速機能副次索引が定義されている定義を使用して、ACB 保守ユーティリティーを実行します。
7. 新しいデータ・セットをエリアに割り振ります。
8. オプション: 新しい DEDB データベースの新しい DB/DBDS/ADS を DBRC に登録します。
9. DEDB 初期設定ユーティリティー (DBFUMIN0) を使用して、新しいエリア・データ・セットをフォーマット設定します。
10. 以下のいずれかの方法で、新しい DEDB とその副次索引データベースの DMB ディレクトリー項目 (DDIR) を作成します。
    - MODBLKS システム定義を実行し、ACBLIB オンライン変更プロセスを使用します。
    - 動的リソース定義を使用している場合は、CREATE DB コマンドを発行してから、ACBLIB オンライン変更またはメンバー・オンライン変更を実行します。
11. 新しい 1 次 DEDB データベースのイメージ・コピーを作成します。
12. オプション: データ共用高速機能副次索引データベースの場合、新しい高速機能副次索引データベースを DBRC に登録します。

13. 高速機能副次索引が定義されている 1 次 DEDB データベースをロードするアプリケーション・プログラムを作成します。
14. アプリケーション・プログラムを実行して 1 次 DEDB データベースとその高速機能副次索引データベースの両方を作成し、ロードします。

1 次 DEDB データベースとその高速機能副次索引データベースのどちらも IMS オンライン・アクセスに使用できます。

関連概念:

363 ページの『第 16 章 副次索引の作成』

関連資料:

副次索引のある例 (システム・ユーティリティー)

DEDB への副次索引の追加

セグメントのキーによって定義されたシーケンス以外のシーケンスでセグメント・タイプを処理するために、既存の DEDB データベースに副次索引を追加できます。

1. DEDB 1 次データベースの DBD 定義を変更して、副次索引情報を追加します。
2. 高速機能副次索引データベースの DBD 定義を作成します。
3. 高速機能副次索引データベースを介してアクセスされる 1 次 DEDB データベースの PCB の中に、高速機能副次索引 DBD 名を指定した PROCSEQD= パラメーターを追加します。
4. 高速機能副次索引が定義されている定義を使用して、DBDGEN ユーティリティーを実行します。
5. 高速機能副次索引が定義されている定義を使用して、PSBGEN ユーティリティーを実行します。
6. 高速機能副次索引が定義されている定義を使用して、ACBGEN ユーティリティーを実行します。
7. データ共用高速機能副次索引データベースの場合、新しい高速機能副次索引データベースを DBRC に登録します。
8. 高速機能副次索引データベースを作成する準備として、1 次 DEDB データベースのアクセス権限を「読み取り専用」に変更するか、1 次 DEDB データベースをオフラインにします。
9. 高速機能副次索引データベースを作成し、ロードします。IMS ツールのベンダーが提供するツールを使用してください。
10. オンライン変更を実行する前に、1 次 DEDB データベースをオフラインにします (まだそうしていない場合)。
11. 新しい高速機能副次索引データベースの DMB ディレクトリー項目 (DDIR) を作成します。DDIR は、以下のいずれかの方法で作成できます。
    - MODBLKS システム定義を実行し、ACBLIB オンライン変更プロセスを使用します。
    - 動的リソース定義を使用している場合は、CREATE DB コマンドを発行して新しい高速機能副次索引データベースの DDIR を作成した後、オンライン変更 (ACBLIB オンライン変更またはメンバー・オンライン変更) を実行します。

12. 1 次 DEDB データベースとその高速機能副次索引データベースを開始します。

1 次 DEDB データベースとその高速機能副次索引データベースのどちらも IMS オンライン・アクセスに使用できます。

関連概念:

373 ページの『副次索引による DEDB データベースの階層の再構築』

363 ページの『第 16 章 副次索引の作成』

関連資料:

副次索引のある例 (システム・ユーティリティー)

索引のドロップ

必要でなくなった副次索引は、DEDB データベースから削除できます。

副次索引を削除するには、次のようにします。

1. 1 次 DEDB データベースの DBD の中で、LCHILD ステートメントおよび XDFLD ステートメントを削除します。
2. /CK 名を指定している FIELD ステートメントを削除または変更します。
3. 関連する高速機能副次索引データベースの DBD 定義を削除します。
4. 高速機能副次索引データベース用の PROCSEQD= パラメーターを持つ PSB 定義があれば削除します。
5. 高速機能副次索引が定義されていない新しい定義を使用して、DBDGEN ユーティリティーを実行します。
6. 高速機能副次索引が定義されていない新しい定義を使用して、PSBGEN ユーティリティーを実行します。
7. 高速機能副次索引が定義されていない新しい定義を使用して、ACBGEN ユーティリティーを実行します。
8. 1 次 DEDB データベースとその高速機能副次索引データベースをオフラインにします。
9. 副次索引が定義されていない 1 次 DEDB データベース定義内の変更を元に戻すためにオンライン変更を実行します。
10. 1 次 DEDB データベースでデータベースを開始し、データベースを IMS のオンライン・アクセスに使用可能にします。

関連概念:

363 ページの『第 16 章 副次索引の作成』

データ・セット・グループの数の変更

通常、1 つのデータベースは物理的に 1 つのデータ・セットに保管され、あるいは HISAM におけるように、1 対のデータ・セットに保管されます。しかし、データベースは物理的に複数のデータ・セット、または 1 対のデータ・セットに保管できます。これを行っている場合、各データ・セットあるいは 1 対のデータ・セットは、データ・セット・グループと呼ばれます。

データベースをチューニングするために、複数データ・セット・グループに変更することができます。複数データ・セット・グループを使用した場合に、パフォーマンスが向上したりスペースの使用効率の改善がもたらされるかどうか判断するためのデータベースを特定してモニターする方法はありません。それよりも、ユーザーのアプリケーション要件に関する知識とデータベースの使用状況に関する多種類の統計が、この判断を行うのに役立ちます。

複数データ・セット・グループに変更すると、データベース再編成/ロード処理ユーティリティー (つまり、HISAM アンロード/再ロード、HD アンロード/再ロード、接頭部解決、および接頭部更新の各ユーティリティー) を使用できます。また、これらのユーティリティーは 1 つ以上のデータベースで同時に使用できます。例えば、1 つ以上の既存のデータベースの再編成を、他のデータベースの初期ロード中に行うことができます。操作が行われている複数のデータベースのうちの一部または全部を、論理的に相互に関連付けることができます。データベース操作は、初期データベース・ロード、データベース・アンロード/再ロード (再編成)、またはデータベース・スキャンとして定義されます。

複数データ・セット・グループを導入する以外に HISAM データベースに構造上の変更を加える場合 (例えば、HISAM から HDAM へのアクセス方式の変更、ポインターの変更、セグメントの追加、副次索引の追加など) は、 843 ページの『論理関係または副次索引を持つ HD データベースでのフロー例』に示すような手順を実行する必要があります。

データベースの中のデータ・セット・グループの数を変更するには、以下の手順を行います。

1. 既存の DBD を用いて、データベースをアンロードします。
2. データベースが PHDAM または PHIDAM であれば、HALDB 区画定義ユーティリティーを使用して、DBRC RECON データ・セットからそのデータベースの定義を削除します。
3. 新しい DBD をコーディングします。
4. データベース・スペースを計算し直します。 それを行う必要があるのは、この変更によってデータベース・スペースの必要量が変わってくるからです。
5. 非 VSAM データ・セットに関しては、古いデータベースのスペースを削除し、新しいデータベースのスペースを定義します。 VSAM データ・セットに関しては、古いクラスターに割り振られているスペースを削除して、新しいクラスターのスペースを定義します。
6. 新規のデータベースが PHDAM または PHIDAM であれば、HALDB 区画定義ユーティリティーを実行して、そのデータベース用の区分データ・セットを定義します。
7. 使用しているデータ・セット数とサイズが変わるので、データ・セットを割り振りし直します。
8. 新しい DBD を用いて、データベースを再ロードします。データベースを再ロードしたらすぐに、このデータベースのイメージ・コピーをとります。
9. データベースで論理関係または副次索引を使用している場合には、接頭部情報を解決するために、再ロードの前後で再編成ユーティリティーを実行する必要があります。

関連概念:

450 ページの『複数データ・セット・グループ』

OSAM データ・セットの割り振り (システム定義)

関連タスク:

603 ページの『データベースの最小サイズの見積もり』
612 ページの『データベース・データ・セットの割り振り』
711 ページの『再編成ユーティリティーを使用するオフライン再編成』

単純な HD データベースでのフロー例

再編成ユーティリティーを使用して単純な HD データベースに変更を加える処理のプロセス・フローは、論理関係または副次索引を使用する HD データベースでのプロセス・フローに比べて単純です。

以下の図は、複数データ・セット・グループを使用するためにデータベースを変更する処理のプロセス・フローを示しています。

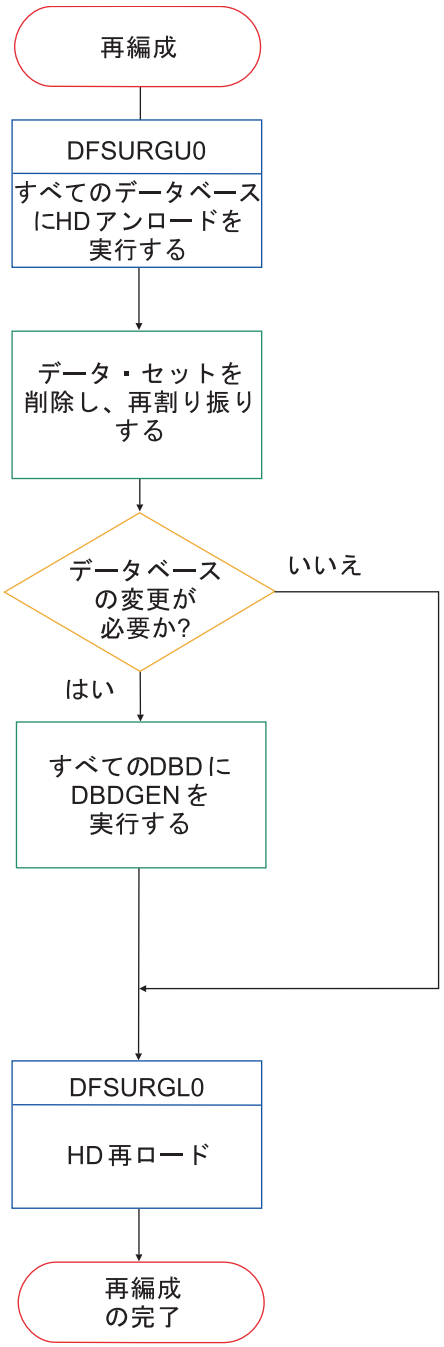


図 292. 複数データ・セット・グループを使用するためにデータベースを変更する処理のフローチャート

HISAM データベースを再編成ユーティリティーで変更する処理のフロー例

以下のフローチャートは、複数データ・セット・グループを使用するために、再編成ユーティリティーを使用してデータベースを変更する処理を示しています。

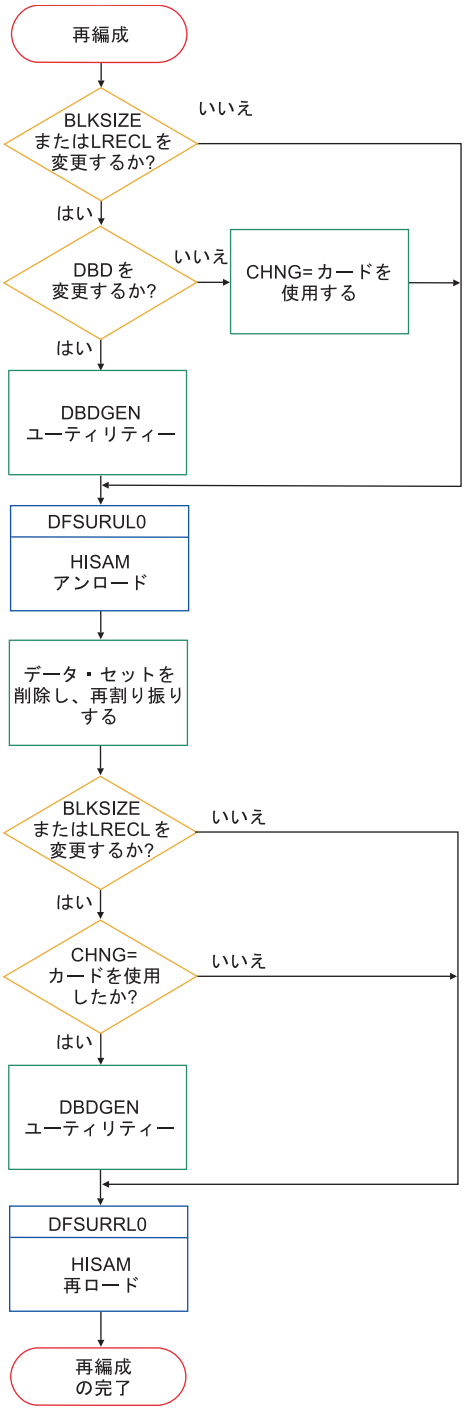


図 *293.* 複数データ・セット・グループを使用するために *HISAM* データベースを変更する処理のフローチャート

論理関係または副次索引を持つ HD データベースでのフロー例

HD データベースに論理関係または副次索引がある場合、あるいは複数データ・セット・グループを追加する際にデータベースに構造上の変更を加える場合は、以下の図に示すようなプロセスをたどることになります。

以下の図は、複数データ・セット・グループを使用するために論理関係または副次索引を使用するデータベースを変更する処理のプロセス・フローを示しています。

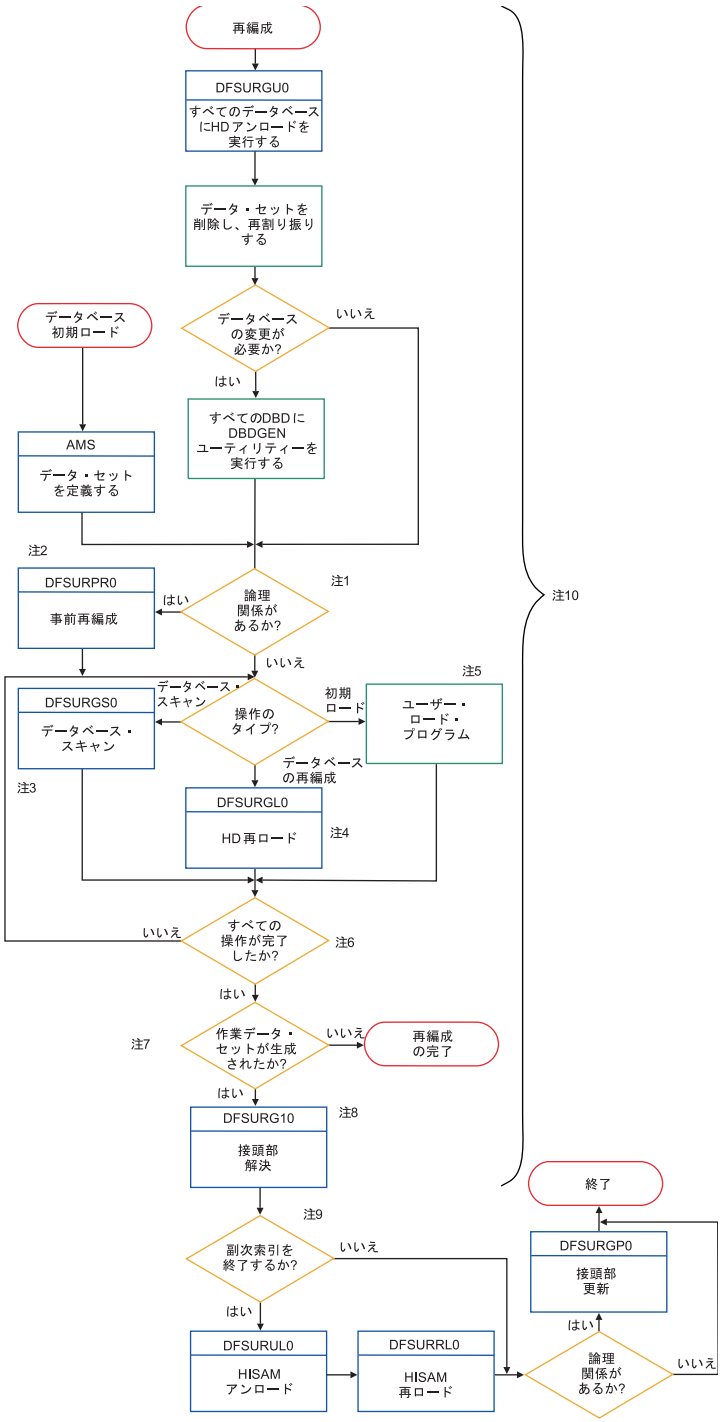


図 *294.* 複数データ・セット・グループを使用するために論理関係または副次索引を持つデータベースを変更する処理のフローチャート

このフローチャートに関する注:

1. 操作が行われているデータベースの一部または全部の中の 1 つ以上のセグメントが、論理関係または副次索引に関与している場合には、「はい」に分岐しなければなりません。また事前再編成ユーティリティーを用いて、どのデータベース操作を実行しなければならないかを決定することもできます。

2. データベース事前再編成ユーティリティーに与えた制御ステートメント上の情報に基づいて、初期ロード、再編成またはスキャンが必要なデータベースのリストをこのユーティリティーが提供します。事前再編成制御ステートメント上で指定したデータベースの数および順序は、再ロードと接頭部解決の間で変更しないでください。
3. 論理親の連結キーがアンロードすべき論理子のデータベースの中で仮想と定義されている場合には、データベースをアンロードする前に、DB スキャン・ユーティリティーを実行しなければなりません。

   このプログラムは、事前再編成ユーティリティーからの出力として示されたデータベースに対して、それぞれ実行しなければなりません。作業データ・セットは、このユーティリティーによってスキャンされたデータベースごとに生成されます。スキャンするデータベースは、事前再編成ユーティリティーの 1 つ以上の出力メッセージの中で、「DBS=」という文字の後ろに列挙されます。
4. HD 再編成再ロード・ユーティリティーは、後でデータベース接頭部解決ユーティリティーで使用されることになる作業データ・セットを生成させることがあります。 HD 再編成アンロード/再編成再ロード・ユーティリティーを使用して再編成するデータベースは、事前再編成ユーティリティーの 1 つ以上の出力メッセージの中で、「DBR=」という文字の後ろに列挙されます。
5. ユーザー提供の初期データベース・ロード・プログラムは、後で接頭部解決ユーティリティーで使用されることになる作業データ・セットを自動的に生成させることができます。作業データ・セットを生成するために、コードを初期ロード・プログラムに付け加える必要はありません。このコードの追加は、ユーザー・プログラムを通して ISRT 要求を出すことによって、IMS が自動的に行います。しかし、ユーザーは、このデータ・セットの DD ステートメントを、初期ロード・プログラムの実行に必要なその他の JCL ステートメントと共に提供しなければなりません。初期ロードするデータベースは、事前再編成ユーティリティーの出力メッセージの 1 つ以上の出力メッセージの中で、DBIL= という文字の後ろに列挙されます。
6. 「はい」に分岐する前に、事前再編成ユーティリティー (これを実行した場合には) が示した操作がすべて完了していることを確認しなければなりません。
7. ユーザーが実行したデータベース操作の間に、作業データ・セットが生成された場合には、「はい」に分岐しなければなりません。データベースの中に論理関係が存在しても、データベース操作中に作業データ・セットが生成されるとは限りません。作業データ・セットが必要となるかどうかは、再編成/ロード処理ユーティリティーがデータベース操作中に、提供される実際のセグメントに基づいて動的に決めます。論理関係に参加しているセグメントがロードされる場合には、作業データ・セットが生成されるので、「はい」に分岐しなければなりません。

   ある特定のデータベース操作について、データベースの作業データ・セットが生成されなかった場合には、そのデータベースの処理は完了しており、使用することができます。

   HIDAM データベースの初期ロードまたは再編成の場合には、その 1 次索引がデータベースのロード時に生成されます。
8. データベース接頭部解決ユーティリティーは、データベース・スキャン・ユーティリティーから出力された作業ファイルと、HD 再編成再ロード・ユーティ

リティーまたはユーザーによる初期データベース・ロードの実行のいずれか (両方ではありません) から出力された作業ファイルを組み合わせて、接頭部更新ユーティリティーで使用する出力データ・セット DFSURWF3 を作成します。次に、接頭部更新ユーティリティーは、操作が行われたデータベースに定義されているすべての論理関係を完結します。

9. 副次索引を作成するか、あるいは 2 つの副次索引を組み合わせる場合には、HISAM アンロード/再ロード・ユーティリティーを実行しなければなりません。 HISAM アンロード/再ロード・ユーティリティーの実行後に、論理関係がデータベース中にある場合には、接頭部更新ユーティリティーを実行しないと、再編成またはロード・プロセスが完了したとは見なされません。
10. フローチャートのこの部分は、操作が初期ロードか再編成、またはスキャンであっても、操作を行うデータベースごとに 1 回は通らなければなりません。操作はすべてのデータベースに並行して行うことも、一時点で 1 つのデータベースに行うこともできます。各種のデータベース操作を順次に実行すると、作業データ・セットのストレージ・スペースの節約が可能であり、各データベース操作に対応した DFSURWF1 DD ステートメントに、DISP=(MOD,KEEP) の指定がある場合には、処理効率が向上します。データベースの初期ロード、再編成、およびスキャンの各プログラムに関する作業データ・セットの属性は同じでなければなりません。

    HD 再編成再ロード・ユーティリティーを使用する場合には、論理親の連結キーが論理子の中で仮想として定義されているのであれば、まず論理関係にあるデータベースのアンロードおよびスキャンをすべて行います。

セグメント編集/圧縮出口ルーチンへの変換

IMS サンプル出口ルーチン DFSCMPX0 のようなセグメント編集/圧縮出口ルーチンを使用する前に、データベースに変更を加えることが必要な場合があります。

既存のデータベースをセグメント編集/圧縮出口ルーチンをサポートするように変換するには、以下のステップを実行してください。

1. 加えようとしている変更がアプリケーション・プログラムの中のコードに影響を及ぼすかどうかを判別します。コードに影響がある場合には、必ずそのコードを変更します。
2. 既存の DBD および HD アンロード・ユーティリティーを用いて、データベースをアンロードします。
3. 新しい DBD をコーディングします。新しい DBD は、編集する必要のあるセグメント・タイプにユーザーの編集ルーチンの名前を指定しなければなりません。
4. 加えようとしている変更がアプリケーション・プログラムの中のコードに影響を与える場合には、このアプリケーション・プログラムの PSB に必要な変更を行います。 DB/DC データ・ディクショナリーをもっている場合には、そのディクショナリーを使用して、DBD の変更の影響を受けるアプリケーション・プログラムと PCB を判別することができます。
5. ACB を動的に作成するのではなく、事前に作成しておく場合は、ACB を作成し直します。

6. データベース・スペースを計算し直します。それを行うのは、その変更を加えた結果として、データベース・スペースの必要量が変わってくるからです。
7. 古いデータベース・スペースを削除して新しいデータベース・スペースを定義します。 VSAM を使用している場合には、アクセス方式サービスの DEFINE CLUSTER コマンドを使用して VSAM データ・セットを定義します。
8. 新しい DBD を用いて、データベースを再ロードします。データベースを再ロードしたらすぐに、このデータベースのイメージ・コピーを作成することを忘れないでください。
9. データベースで論理関係または副次索引を使用している場合には、接頭部情報を解決するために、再ロードの前後で再編成ユーティリティーを実行する必要があります。

関連概念:

427 ページの『セグメント編集/圧縮出口ルーチン』

関連タスク:

711 ページの『再編成ユーティリティーを使用するオフライン再編成』

関連資料:

セグメント編集/圧縮出口ルーチン (出口ルーチン)

データ・キャプチャー出口ルーチンおよび非同期データ・キャプチャーのためのデータベースの変換

データ・キャプチャー出口ルーチンおよび非同期データ・キャプチャーのためにデータベースを変換する作業は、4 つの基本的なステップで実行できます。

このトピックには汎用プログラミング・インターフェース情報が含まれています。

データ・キャプチャー出口ルーチンまたは非同期データ・キャプチャーで使用するため、既存のデータベースを変換するには、次の手順を行ってください。

1. 何らかの変更を行うことにより、データベースの論理削除規則を改訂する必要があるかどうかを判別します。変更を行う場合には、削除規則を変更し、データベースを再編成する必要があります。
2. 新しい DBD をコーディングします。 DBD ステートメントまたは SEGM ステートメント上で、データベースの中のあるセグメントに対して呼び出したい各出口ルーチンの名前を指定します。
3. DBDGEN を実行します。
4. 動的に作成された ACB ではなく事前に作成された ACB を使用する場合には、ACB を作成し直します。

関連概念:

430 ページの『データ・キャプチャー出口ルーチン』

関連資料:

データベース記述 (DBD) 生成ユーティリティー (システム・ユーティリティー)

Database Manager 出口ルーチン (出口ルーチン)

オンライン・データベースの変更

IMS には、IMS データベースをオンラインで変更する方法がいくつかあります。例えば、タイプ 2 コマンドを使用する動的リソース定義のサポート、タイプ 1 コマンドを使用するオンライン変更機能、HALDB および DEDB の変更機能などがあります。

方法ごとに異なる前提条件とステップがあり、サポートされるデータベース変更のタイプが異なります。

例えば、オンライン変更機能には MODBLKS システム定義が必要ですが、動的リソース定義には必要ありません。HALDB Alter 関数は、INIT OLREORG コマンドを発行して開始しますが、DEDB Alter 関数は DEDB 変更ユーティリティー (DBFUDA00) を実行して開始します。

関連タスク:

591 ページの『オンライン・システムへのデータベースの導入』

オンライン HALDB データベースの定義の変更

INITIATE OLREORG コマンドの ALTER オプションを指定すると、データベースへのアクセスを停止せずに、特定のセグメントまたはフィールド定義の変更をオンライン HALDB データベースに適用できます。

再編成処理中、既存のデータベース構造を使用するアプリケーション・プログラムは、引き続きデータベースにアクセスできます。変更されるデータベース構造を使用するアプリケーション・プログラムは、再編成処理が完了し、オンライン変更機能が実行されて新しいセグメント定義およびフィールド定義にアクセスできるようになるまで、データベースにアクセスできません。

ALTER オプションは、現在以下のデータベース・アクセス・タイプでサポートされています。

- PHDAM
- PHIDAM

ALTER オプションを使用すると、データベース定義に対して以下の変更のみを行うことができます。

- セグメント長の増加。
- セグメント末尾にある既存の未定義スペースへの新規フィールドの追加。

ALTER オプションが指定されると、データベースは統合 HALDB オンライン再編成機能によって変更されます。そのため、特に断りのない限り、HALDB データベースを変更する場合は、統合 HALDB オンライン再編成機能について記載されているプロシージャー、ガイドライン、要件などが適用されます。

関連概念:

738 ページの『HALDB オンライン再編成』

オンライン HALDB データベースの変更の構成要件

オンライン HALDB データベースを変更する前に、IMS システムはいくつかの構成要件を満たしている必要があります。

タイプ 2 コマンド環境を使用可能にするために、IMS システムを IMS 共通サービス層 (CSL) のオペレーション・マネージャー (OM) コンポーネントと構造化呼び出しインターフェース (SCI) コンポーネントとともに構成する必要があります。HALDB 変更操作は、タイプ 2 コマンド INITIATE OLREORG NAME(*haldbmaster*) OPTION(ALTER) によってのみ開始することができます。

RECON データ・セットの MINVERS 値を、オンライン HALDB データベースの変更をサポートする IMS のバージョンに設定する必要があります。そのため、HALDB 変更機能を使用する IMS システムをサポートする RECON データ・セットを、HALDB 変更機能をサポートしないバージョンの IMS と共用することはできません。

オンライン HALDB データベースを変更するためのステップ

オンライン HALDB データベースの構造を変更するために必要なステップには、データベース定義の変更、DBD 生成と ACB 生成の実行、INIT OLREORG OPTION(ALTER) コマンドの発行、オンライン変更 (OLC) 手順の実行などがあります。

HALDB 変更機能は、動的割り振りを使用して、ステージング ACB ライブラリーを割り振ります。ステージング ACB ライブラリーの動的割り振りがまだ使用可能になっていない場合は、DFSMDA マクロおよび IMSDALOC プロシージャーを使用して IMSACB という名前の動的割り振りメンバーをビルドすることによって使用可能にします。

ALTER オプションを使用してデータベース変更を適用するために必要なステップの概要:

1. データベース生成 (DBDGEN) ユーティリティーの SEGM 入力ステートメント、および必要に応じて FIELD 入力ステートメントで、データベース構造に対する変更をコーディングします。
2. 変更したマクロを使用して DBDGEN ユーティリティーを実行し、DBD メンバーを生成します。
3. データベースにアクセスするアプリケーション・プログラムの PSB メンバーを生成します。 データベースのバージョン管理が使用可能になっている場合は、前のバージョンのデータベースにのみアクセスするアプリケーション・プログラムの PSB を更新する必要はありません。
4. 以下のいずれかの方法を使用して、ACB メンバーを ACB ステージング・ライブラリー内に生成します。
   - ご使用のシステムで IMS カタログが使用可能な場合は、ACB Generation and Catalog Populate ユーティリティー (DFS3UACB) を実行して、単一のジョブ・ステップで ACB メンバーの生成と IMS カタログへのデータの設定を行います。
   - ご使用のシステムで IMS カタログが使用可能でない場合は、ACB 保守ユーティリティーを実行します。
5. 必要に応じて、出力データ・セットの OSAM ブロックまたは VSAM CI サイズを変更するために RECON データ・セットの ALTERSZE 値を設定します。
6. IMS タイプ 2 コマンド INITIATE OLREORG OPTION(ALTER) を実行して、オンライン・データベースに変更を適用します。 コマンドの実行後、古い

データベース構造を使用するアプリケーション・プログラムは、オンライン変更が行われるまで、引き続きデータベースにアクセスできます。

7. メンバー・オンライン変更プロシージャーを使用して、新しいデータベース構造へのアクセスを使用可能にします。 ステージング・ライブラリーにある、変更されたデータベースの ACB メンバーがアクティブになるまで、HALDB オンライン変更操作は完了しません。

関連概念:

558 ページの『DBDGEN ユーティリティーの入力としてのデータベース記述のコーディング』

569 ページの『アプリケーション制御ブロック (ACBGEN) の作成』

ACBLIB メンバーのオンライン変更のための ACB ステージング・ライブラリーの動的割り振り (システム定義)

関連タスク:

872 ページの『オンライン変更機能を使用したデータベース変更の活動化』

852 ページの『HALDB の変更によるブロック・サイズまたは CI サイズの変更』

関連資料:

IMSDALOC プロシージャー (システム定義)

DFSMDA マクロ (システム定義)

INITIATE OLREORG コマンド (コマンド)

INITIATE OLC コマンド (コマンド)

CHANGE.PART コマンド (コマンド)

データベース記述 (DBD) 生成ユーティリティー (システム・ユーティリティー)

アプリケーション制御ブロック保守ユーティリティー (システム・ユーティリティー)

ACB ライブラリーおよび HALDB の変更

変更操作を開始する前に、データベースの新規構造を定義する ACB メンバーを生成して、ステージング ACB ライブラリーに入れる必要があります。オンライン変更機能によって RECON データ・セットのデータベース区画レコードから ALTER IN PROGRESS 状況がクリアされるまで、これらの ACB メンバーをステージング・ライブラリー内に維持する必要があります。

ステージング ACB ライブラリーは、データベースを共用するすべての IMS システム、およびデータベースにアクセスするバッチ・アプリケーション・プログラムまたはユーティリティーで使用可能でなければなりません。単一の共用ステージング ACB ライブラリーを使用することも、あるいはデータベースの変更が含まれている ACB メンバーの複製されたコピーがある多重ステージング ACB ライブラリ

ーを使用することもできます。 TYPE=IMSACB DFSMDA マクロを定義して、ステージング ACB ライブラリーの動的割り振りおよび動的割り振り解除を使用可能にする必要があります。

区画レコードが ALTER IN PROGRESS 状況であるときに、変更するデータベースの ACB メンバーが変更されたりステージング・ライブラリーから削除されたりすると、アプリケーション・プログラムおよびユーティリティーはデータベースにアクセスできず、オンライン変更機能は新しいデータベース構造をアクティブにすることができません。 ACB メンバーが変更または削除された場合は、ステージング ACB ライブラリーを変更プロセス開始時の状態に復元することにより、データベースへのアクセスを復元してオンライン変更を完了することができます。

すべての区画データ・セットが変更された後、ステージング・ライブラリーの ACB メンバーをアクティブ ACB ライブラリーに追加して、新しいデータベース構造にアクセスできるようにする必要があります。

ACB メンバーをアクティブ ACB ライブラリーに追加するには、ACB メンバー・オンライン変更機能を使用します。メンバー・オンライン変更機能は、変更されたデータベースの ACB メンバーのみをステージング ACB ライブラリーからアクティブ ACB ライブラリーにコピーします。

必要に応じて、ローカル・オンライン変更機能またはグローバル・オンライン変更機能を使用することができます。ただし、どちらの方式でも、オンライン変更プロセスを開始する前に、ACB メンバーをステージング・ライブラリーから非アクティブ ACB ライブラリーにコピーする必要があります。また、これらの方法は、変更された ACB メンバーだけでなく、ACB ライブラリー全体を置き換えます。

データベースのバージョン管理が使用可能で、特定のアプリケーション・プログラムが、変更しなくてもデータベースに引き続きアクセスできるようになっていない限り、新しい ACB メンバーをアクティブにする際に、データベース内の変更されたセグメントを参照するアプリケーション・プログラムもアクティブにする必要があります。

ACB メンバーがアクティブになった後は、もうステージング・ライブラリーは必要ありません。

関連概念:

- ACB ライブラリー・メンバー・オンライン変更 (システム管理)
- ACBLIB メンバーのオンライン変更のための ACB ステージング・ライブラリーの動的割り振り (システム定義)

関連タスク:

- オンラインでの IMS.ACBLIB メンバーの変更または追加 (システム管理)

オンライン変更および HALDB の変更

変更処理の完了後に HALDB データベースの新規構造にアクセスできるようにするには、IMS オンライン変更機能が必要です。

オンライン変更機能は、新しいデータベース構造が含まれている ACB メンバーをアクティブにするだけでなく、RECON データ・セット内のさまざまなフラグおよ

びカウンターをクリアします。オンライン変更機能以外の方法で ACB メンバーをアクティブにした場合、フラグおよびカウンターは自動的にはクリアされません。

ACB メンバーがアクティブになってフラグおよびカウンターがクリアされるまで変更手順は完了せず、新しいデータベース構造を使用することはできません。

オンライン変更手順を開始して変更操作を実行する前に、/DBRECOVERY DB *HALDB_master_name* コマンドまたは UPDATE DB NAME(*HALDB_master_name*) STOP(ACCESS) コマンドを実行して HALDB データベースへのアクセスを停止する必要があります。

メンバー・オンライン変更機能を使用して、変更手順を実行します。メンバー・オンライン変更機能は、ステージング ACB ライブラリーから直接読み取り、アクティブにする必要がある特定の ACB メンバーのみを処理することができます。ローカルおよびグローバルのオンライン変更機能では、ACB メンバーを非アクティブ ACB ライブラリーにコピーする必要があります。また、これらの機能は、新しいデータベース変更が含まれている ACB メンバーだけでなく、ACB ライブラリー全体を処理します。

データ共用環境で、ローカル・オンライン変更を使用する場合は、データベースを共用するすべての IMS システム間でローカル・オンライン変更を調整する必要があります。ローカル変更が行われた最初の IMS システムが RECON データ・セット内のすべてのフラグおよびカウンターをリセットするため、すべての IMS システムでデータベースに対して /DBRECOVER コマンドを発行した後、各システムでローカル・オンライン変更を行ってください。

関連概念:

899 ページの『オンライン変更と HALDB データベース』

ACB ライブラリー・メンバー・オンライン変更 (システム管理)

関連タスク:

872 ページの『オンライン変更機能を使用したデータベース変更の活動化』

オンラインでの IMS.ACBLIB メンバーの変更または追加 (システム管理)

関連資料:

INITIATE OLC コマンド (コマンド)

HALDB の変更によるブロック・サイズまたは CI サイズの変更

オンライン HALDB データベースを変更しているときにセグメントのサイズを増やす場合は、変更されたセグメントを保持する出力データベース・データ・セットの OSAM ブロック・サイズまたは VSAM CI サイズも増やさなければならない場合があります。

注: ALTER オプションが指定されていても、入力データベース・メンバー内でデータベース構造の変更が検出されない場合、INITIATE OLREORG コマンドは失敗します。その結果、データベースの構造変更も行っている場合を除き、ALTER オプションを使用してブロック・サイズまたは CI サイズを変更することはできません。

新しいブロック・サイズまたは CI サイズは、変更処理の開始時に出力データ・セットに適用されますが、INIT OLREORG OPTION(ALTER) コマンドを発行する前に RECON データ・セットに保管する必要があります。

新しいブロック・サイズまたは CI サイズを RECON データ・セットに保管するには、これらを CHANGE.PART コマンドの ALTERSZE キーワードで指定します。

VSAM データ・セットでは、変更処理用の出力データ・セットが存在する場合、変更処理を開始する前に、新しい CI サイズを必要とする出力データ・セットを削除する必要があります。必要な出力データ・セットは、ユーザーによって作成されない限り、変更処理によって新しい CI サイズで自動的に再作成されます。ユーザーが出力データ・セットを作成する場合には、CI サイズが、対応する ALTERSZE 値と一致していなければなりません。特定の VSAM データ・セット・グループに ALTERSZE 値が指定されておらず、出力データ・セットが存在する場合は、出力データ・セットの CI サイズが使用されます。 ALTERSZE 値が指定されておらず、出力データ・セットも存在しない場合は、入力データ・セットの CI サイズが使用されます。

OSAM データ・セットでは、ALTERSZE 値が設定されていない場合、出力データ・セットが存在していても、入力データ・セットのブロック・サイズが使用されます。

ブロック・サイズまたは CI サイズを変更する際には、バッファーのサイズも変更する必要があります。新しいブロック・サイズまたは CI サイズが現行のバッファー・サブプールに適合しない場合、IMS は、使用可能なサブプールのなかから、より大きいサイズのサブプールを見つけようとします。新しいブロック・サイズまたは CI サイズを保持するために十分な大きさの使用可能なサブプールがない場合は、出力データ・セットのオープンが失敗します。バッファー・サイズを調べるには、タイプ 2 コマンド QUERY POOL TYPE(DBAS) を発行します。

関連概念:

775 ページの『バッファーの調整』

関連タスク:

849 ページの『オンライン HALDB データベースを変更するためのステップ』

関連資料:

- CHANGE.PART コマンド (コマンド)
- LIST.DB コマンド (コマンド)
- QUERY POOL コマンド (コマンド)

HALDB の変更の ALTERSZE 値の設定:

HALDB 変更機能によってデータベースの構造を変更する場合に、データベース・データ・セットの OSAM ブロック・サイズまたは VSAM CI サイズを増やすには、RECON データ・セットの各区画レコードで、変更する各データ・セット・グループに ALTERSZE 値を設定する必要があります。

ALTERSZE 値を設定するには、DBRC コマンド CHANGE.PART または HALDB 区画定義ユーティリティー (%DFSHALDB) を使用します。

CHANGE.PART コマンドを使用して ALTERSZE 値を設定する場合、値を定位置のコンマ区切り値として指定する必要があります。 1 番目の位置の値は、1 番目のデータ・セット・グループに適用されます。 2 番目の位置の値は、2 番目のデータ・セット・グループに適用されます (以降同様)。

例えば、ALTERSZE(,,4096) という ALTERSZE 指定は、3 番目のデータ・セット・グループの新規ブロック・サイズまたは CI サイズを設定しますが、1 番目と 2 番目のデータ・セット・グループのサイズは変更しません。4 番目から 10 番目のデータ・セット・グループが存在する場合も同様にサイズは変更されません。

変更するセグメントを定義している SEGM ステートメントの DSGROUP キーワードを調べることにより、データ・セット・グループのサイズを入力する位置を判別することができます。DSGROUP=A は最初の位置を示し、DSGROUP=B は 2 番目の位置を示し、以降 10 番目の位置を示す DSGROUP=J まで同様です。

バッチ DBRC コマンドを使用して変更操作の出力データベース・データ・セットの ALTERSZE 値を設定するには、以下のコマンドを指定します。

コマンドが正常に処理された後、変更処理によって使用されるブロック・サイズまたは CI サイズは、区画の RECON レコードの「ALTER BLOCK SIZE」の下に表示されます。これは、DBRC コマンド LIST.DB DBD(*partitionname*) を実行することにより表示されます。

変更サイズ値の設定後、IMS タイプ 2 コマンド INIT OLREORG NAME(*masterdb*) OPTION(ALTER) を実行して、変更処理を開始することができます。

関連資料:

- CHANGE.PART コマンド (コマンド)
- LIST.DB コマンド (コマンド)

ALTERSZE 値の設定後の変更:

設定済みの ALTERSZE 値を訂正するには、CHANGE.PART コマンドまたは HALDB 区画定義ユーティリティー (%DFSHALDB) を使用して、誤った値を正しい値に置き換えます。

OSAM データ・セットの場合、ALTERSZE 値を入力データ・セットの元のブロック・サイズに戻すと、ALTERSZE 値が 0 と表示されます。これは、ブロック・サイズが変更されていないことを示します。すべての ALTERSZE 値が入力データ・セットの元のブロック・サイズに復元されると、LIST.DB コマンドの出力に「**ALTER SIZE**」フィールドは表示されません。

VSAM データ・セットの場合、ALTERSZE 値を元の CI サイズに戻すと、元の CI サイズが表示されます。すべての ALTERSZE 値が入力データ・セットの元の CI サイズに復元されると、「**ALTER SIZE**」フィールドに、ユーザーが入力した最後の値が表示されます。

CHANGE.PART PART(name) NOALTRSZ コマンドを指定して、区画の ALTERSZE 値をすべてクリアすることができます。 OSAM データ・セットおよび VSAM データ・セットのいずれの場合にも、NOALTRSZ キーワードを使用してすべての ALTERSZE 値をクリアすると、区画レコードを表示したときに「**ALTER SIZE**」フィールドは表示されません。

コマンドが正常に処理された後、変更処理によって使用されるブロック・サイズまたは CI サイズが、区画の RECON レコードの「**ALTER BLOCK SIZE**」の下に表示されます。これは、DBRC コマンド LIST.DB DBD(*partitionname*) を実行することにより表示されます。

変更サイズ値の修正後、IMS タイプ 2 コマンド INIT OLREORG NAME(*masterdb*) OPTION(ALTER) を実行して、変更処理を開始することができます。

関連資料:

- CHANGE.PART コマンド (コマンド)
- LIST.DB コマンド (コマンド)

ALTER オプションが指定されている場合のオンライン再編成処理

ALTER オプションが指定されている場合、INIT OLREORG コマンドにより HALDB データベース全体の再編成が開始され、すべてのデータベース区画にデータベース変更が適用されます。

INIT OLREORG OPTION(ALTER) コマンドを受け取ると、IMS システムは最大 10 個の区画を同時に変更することができます。10 個を超える区画がある場合は、最初に変更された区画のいずれかに対する処理が完了するとすぐに、残りの区画がキューに入れられて変更されます。

IMS システムが区画を再編成して変更する間、その IMS システムが区画の所有権を持ちます。変更処理の対象となる区画を所有する IMS システムのサブシステム ID は、区画レコードの OLRIMSID フィールドに記録されます。

データ共用環境では、区画の所有権は、区画を変更することができる最初の IMS システムに付与されます。他の IMS システムが使用可能になる前に、1 つの IMS システムが 10 個の区画を処理できる場合、10 個の区画はすべて単一の IMS システムによって処理されます。変更処理のために区画がキューに入れられると、その区画は、区画の所有権を最初に取得できた IMS システムによって処理されます。

変更処理の開始時に、IMS は、ALTER IN PROGRESS=YES が指定されているデータベース内のすべての区画 (キューに入っている区画を含む) の RECON レコードを更新します。 IMS システムが区画をアクティブに変更している間、その区画のレコードは OLREORG CURSOR ACTIVE=YES と表示されます。

区画に対する変更処理が完了したら、その区画レコードは PARTITION ALTERED=YES と表示されます。

データベース内のすべての区画の状況が PARTITION ALTERED=YES になったら、オンライン変更を実行して新しいデータベース構造をアクティブにすることができます。オンライン変更機能は、ALTER IN PROGRESS フィールドと PARTITION ALTERED フィールドの両方を NO にリセットします。

関連概念:

738 ページの『HALDB オンライン再編成』

関連資料:

- INITIATE OLREORG コマンド (コマンド)
- DB (HALDB) レコード・フィールド (コマンド)
- DB (PART) レコード・フィールド (コマンド)
- アクティブ・サイトの RECON データ・セットのサンプル・リスト (コマンド)
- LIST.DB コマンド (コマンド)

HALDB の変更およびオフライン再編成

データベースで HALDB の変更を開始した後にその変更操作が強制終了された場合、オフライン再編成プロセスを使用して、データベースへの DBD 変更の適用を完了できます。

オフライン再編成を開始する前に、IMS タイプ 2 コマンド UPDATE DB NAME(*partition_name*) STOP(ACCESS) または IMS タイプ 1 コマンド /DBRECOVERY を実行して、データベースへのアクセスを停止する必要があります。

オフライン再編成を実行するには、HD 再編成アンロード・ユーティリティー (DFSURGU0) を使用してデータベースをアンロードした後、HD 再編成再ロード・ユーティリティー (DFSURGL0) を使用してデータベースを再ロードします。

HD 再編成アンロード・ユーティリティーは、A-J データ・セットと M-V データ・セット両方のアクティブ部分からレコードを自動的にアンロードします。データベース・セグメントの一部またはすべてが HALDB 変更機能によって変更されているかどうかにかかわらず、HD 再編成アンロード・ユーティリティーの実行 JCL の IMS DD ステートメントで、データベースの変更されていないオリジナルの DBD が含まれている DBD ライブラリーを参照する必要があります。

また、HD 再編成再ロード・ユーティリティー (DFSURGL0) の実行 JCL の IMS DD ステートメントでも、データベースの変更されていないオリジナルの DBD が含まれている DBD ライブラリーを参照する必要があります。

変更された DBD は、ステージング ACB ライブラリーにあります。ステージング ACB ライブラリーの動的割り振りを使用可能にしている場合は、いずれのユーティリティーにおいても、ステージング ACB ライブラリーの DD ステートメントは必要ありません。

HD 再編成再ロード・ユーティリティーは、すべてのレコードを A–J データ・セットにロードします。HALDB 変更機能が M–V データ・セットに書き込みを行った場合、A–J データ・セットの特性が正しくない可能性があります。データベースを再ロードする前に、A-J データ・セットが、変更されたセグメント・サイズを保管するために十分な大きさに割り振られるようにしてください。また、VSAM データ・セットを変更する場合は、A–J データ・セットの VSAM CI サイズが正しいことを確認してください。

データベースが再ロードされた後、オンライン変更機能を実行して、変更されたデータベースをオンライン・システムで使用可能にし、RECON データ・セット内の HALDB 変更フラグとカウンターをリセットします。

オンライン変更の実行後、データベースを再始動できます。

関連タスク:

731 ページの『HALDB のオフライン再編成』

763 ページの『HALDB のオンライン再編成後のオフライン再編成』

関連資料:

HD 再編成アンロード・ユーティリティー (DFSURGU0) (データベース・ユーティリティー)

HD 再編成再ロード・ユーティリティー (DFSURGL0) (データベース・ユーティリティー)

共用 HALDB データベースの変更

データ共用環境では、参加している各 IMS システムに INIT OLREORG OPTION(ALTER) コマンドを発送することにより、複数の IMS システム間で変更処理を分散することができます。

変更操作に関与しているすべての IMS システムは、ステージング ACB ライブラリー内のメンバーからデータベース変更を読み取ります。ステージング ACB ライブラリーを共用することも、各 IMS システムで固有のステージング ACB ライブラリーを持つこともできます。ただし、ステージング ACB ライブラリーが共用されていない場合、各 IMS システムによって使用される ACB メンバーは、ACB 保守ユーティリティーの同じ実行によって作成されたコピーでなければなりません。

推奨事項**:** 可能であれば、関与する各 IMS システムで同じ共用ステージング ACB ライブラリーを割り振るようにしてください。

変更操作が開始されると、変更するデータベースにアクセスしない IMS システムも含めて、IMSplex 内のすべての IMS システムは、変更処理の開始を確認するために DFS3197I メッセージを発行します。

変更処理中、IMS システムは、変更中のデータベースの別の変更操作も、変更中のデータベースの HALDB オンライン再編成も開始することができません。

各 IMS システムは、開始した区画変更を停止するときに DFS3198I メッセージを発行します。データ共用環境で、IMS システムは、「DFS3198I: `DFS3198I HALDB ALTER ENDED...`」または「 `DFS3198I HALDB ALTER COMPLETED...`」のいずれかのバージョンのメッセージを発行する可能性があります。 IMS システムは、変更処理

が中断された場合、あるいは HALDB データベースの 1 つ以上の区画が他の IMS システムによって変更された場合に「DFS3198I HALDB ALTER ENDED...」メッセージを発行します。この場合、他の IMS システムが、開始した区画変更を完了していることも、あるいは完了していないこともあります。IMS システムは、HALDB データベース自体のすべての区画を変更した場合、および変更処理がデータベース内のすべての区画について完了した場合に、「DFS3198I HALDB ALTER COMPLETED...」メッセージを発行します。

データベースを共用するすべての IMS システムのステージング ACB ライブラリー内の ACB メンバーをアクティブにするには、メンバー・オンライン変更を使用します。

グローバルまたはローカルのオンライン変更を使用する場合は、変更処理の完了後、オンライン変更を行う前に、データベース変更が含まれている ACB メンバーを、各 IMS システムが使用する非アクティブ ACB ライブラリーにコピーする必要があります。

推奨事項: 変更処理の後、ACB メンバーをアクティブにするために、データ共用環境でローカル・オンライン変更を使用しないようにしてください。

HALDB の変更およびバッチ処理

バッチ・アプリケーション・プログラムとバッチ・ユーティリティーは、HALDB データベースが HALDB 変更機能によって再編成されている間、そのデータベースにアクセスできます。

バッチ・アプリケーション・プログラム

バッチ・アプリケーション・プログラムは、既存の実行 JCL を変更することなく、HALDB 変更操作の処理中に、HALDB データベースにアクセスできます。ただし、データベース内のすべてのセグメント・データが既に新しいデータベース定義に適合している場合でも、オンライン変更機能が変更操作を完了するまで、IMS は、比較を行うために、オリジナルの変更されていないデータベース定義を必要とします。そのため、DLIBATCH アプリケーション・プログラムは、実行 JCL の IMS DD ステートメントで、オリジナルの変更されていない DBD が含まれている DBD ライブラリーを参照する必要があります。 DBBBATCH アプリケーション・プログラムは、IMSACB DD ステートメントで、オリジナルの変更されていないデータベース定義が含まれているアクティブ ACB ライブラリーを参照する必要があります。ステージング ACB ライブラリーの動的割り振りを使用可能にしている場合は、変更された新しいデータベース定義が含まれているステージング ACB ライブラリーの DD ステートメントは必要ありません。

バッチ・ユーティリティー

バッチ・ユーティリティーは、HALDB 変更操作の処理中に、HALDB データベースにアクセスできます。ただし、データベース内のすべてまたは一部のセグメント・データが既に新しいデータベース定義に適合しているかどうかに関係なく、オンライン変更機能が変更操作を完了するまで、IMS は、比較を行うために、変更されていないオリジナルのデータベース定義を必要とします。

そのため、オンライン変更が実行されるまで、ユーティリティーの実行 JCL の IMS DD ステートメントで、変更されていないオリジナルのデータベース定義が含まれている DBD ライブラリーを参照する必要があります。

ステージング ACB ライブラリーの動的割り振りを使用可能にしている場合は、変更された新しいデータベース定義が含まれているステージング ACB ライブラリーの DD ステートメントは必要ありません。

HALDB 変更操作が開始されたが、変更処理を行うためにオンライン変更機能が実行されていない HALDB データベースの 1 つ以上の区画の再編成を行うために HD 再編成再ロード・ユーティリティーを使用する場合は、HD 再編成再ロード・ユーティリティーの実行 JCL で、変更されていないオリジナルの DBD が含まれている DBD ライブラリーを参照する IMS DD ステートメントを指定する必要があります。

関連資料:

- DBBBATCH プロシージャー (システム定義)
- DLIBATCH プロシージャー (システム定義)
- HD 再編成再ロード・ユーティリティー (DFSURGL0) (データベース・ユーティリティー)

HALDB の ALTER 処理の状況の照会

オンライン HALDB データベースの構造を変更しているときに変更処理が完了したかどうかを判別するために、IMS タイプ 2 コマンド QUERY OLREORG NAME(*partname* | *) STATUS(ALTER | ALTINPRG | ALTDONE) を使用できます。

HALDB データベース内の 1 つ、複数、またはすべての区画に対してコマンドを実行できます。

コマンドで STATUS(ALTER) とともに NAME(*) を指定すると、IMS は、データベース内のすべての区画の変更状況を返します。コマンドで STATUS(ALTINPRG) または STATUS(ALTDONE) とともに NAME(*) を指定すると、IMS は、データベース内の指定された状況の区画のみを返します。

1 つ以上の区画の名前を指定すると、IMS は、指定された区画のみの変更状況を返します。

変更中の HALDB 内の区画は、以下のいずれかの状態になります。

ALTINPRG
区画は、現在変更中であるか、または変更の待機中であることを示します。

ALTDONE
その区画の変更処理が完了していることを示します。区画の ALTER COMPLETE 状況は、オンライン変更が行われて新しいデータベース構造の ACB メンバーがアクティブになるまで引き続き有効です。

ALTINQUE
その区画に対する変更処理がまだ開始されていないことを示します。

オンライン HALDB データベースにおける変更処理の状況を照会するには、タイプ 2 コマンド QUERY OLREORG NAME(*partname* | *) STATUS(ALTER | ALTINPRG | ALTDONE) を実行します。

変更処理の完了前での停止

TERMINATE OLREORG コマンドを発行すると、HALDB データベースの変更処理をその完了前に停止できます。

タイプ 1 またはタイプ 2 のバージョンの TERMINATE OLREORG コマンドを実行できますが、複数の IMS システムに対して実行できるのはタイプ 2 バージョンのコマンドのみです。タイプ 1 コマンドは、コマンドが発行されたローカル IMS システムによってのみ処理されます。

TERMINATE OLREORG コマンドでは、HALDB マスター・データベースの名前の指定はサポートされていません。HALDB データベース全体に対する変更処理を停止する場合は、区画名の代わりにワイルドカード文字を指定します。あるいは、データベース内のすべての区画の名前を明示的に指定することもできます。複数の IMS システムでデータベースを変更している場合は、タイプ 2 コマンド TERMINATE OLREORG を使用してすべての IMS システムを同時に停止するか、または各 IMS システムで個々にタイプ 1 コマンドを実行する必要があります。

データベース内の区画のサブセットに対する変更処理が停止された場合、指定されたサブセットに含まれていない他の区画に対する変更処理は続行されます。

変更処理が停止されたときに、区画内のデータが入力データ・セットと出力データ・セットの両方に物理的に保管されている可能性があります。出力データ・セットは、変更されたデータベース構造に従っています。入力データ・セットは、古いデータベース構造に従っています。ただし、アプリケーション・プログラムは、データが物理的に保管されている場所を認識していません。変更処理が完了して、オンライン変更が行われるまで、アプリケーション・プログラムは古いデータベース構造のデータにしかアクセスできません。

HALDB データベース内のすべての区画が変更されてオンライン変更が行われるまで、変更されていないデータベース構造を使用するアプリケーション・プログラムのみがデータベースにアクセスできます。

- HALDB データベースの 1 つ以上の区画の変更処理を停止するには、TERMINATE OLREORG NAME(*partnm* | *) コマンドを実行します。
- 変更処理を再開するには、INITIATE OLREORG OPTION(ALTER) コマンドを実行します。

関連概念:

743 ページの『HALDB オンライン再編成の終了段階』

関連タスク:

754 ページの『HALDB オンライン再編成の停止』

関連資料:

- INITIATE OLREORG コマンド (コマンド)
- TERMINATE OLREORG コマンド (コマンド)

論理関係のある HALDB データベースの変更

HALDB データベースが他のデータベースと論理的に関連する場合、HALDB オンライン変更機能を使用して、論理親セグメントまたは論理子セグメントを除くデータベース内の任意のセグメントを変更できます。

論理親または論理子であるセグメントを変更する必要がある場合は、HD 再編成アンロード・ユーティリティー (DFSURGU0) および HD 再編成再ロード・ユーティリティー (DFSURGL0) を使用して、両方のデータベースに同時に変更を適用できるように、両方のデータベースをオフラインにする必要があります。HALDB データベースのオンライン変更機能は、一度に 1 つのデータベースにのみ変更を適用することができます。

副次索引を持つ HALDB データベースの変更

HALDB データベースが 1 つ以上の副次索引によって索引を付けられている場合、HALDB オンライン変更オプションを使用してデータベースを変更できますが、HALDB オンライン変更オプションを使用して副次索引を変更することはできません。

DEDB 変更ユーティリティーを使用したオンライン DEDB データベースの定義の変更

DEDB 変更ユーティリティー (DBFUDA00) を使用して、2 ステージ・ランダマイザーを使用する DEDB データベースのデータベースまたはエリアの定義を変更でき、その際に DEDB エリアまたは DEDB データベースはオンラインでアクセス可能な状態に保つことができます。

DEDB 変更ユーティリティーを使用する場合は、PROCLIB データ・セットの DFSDFxxx メンバーの高速機能セクションで、ALTERHLQ をデフォルトとして設定します。

複数の IMS が呼び出される場合は、PROCLIB データ・セットの DFSDFxxx メンバーの高速機能セクションで、ALTERGRP を設定する必要があります。

要件:

- 2 ステージ・ランダマイザーを使用する必要があります。これにより、エリアを個々に処理することができます。
- すべての IMS データ共用システムは、IMS バージョン 13 以降でなければなりません。 RECON データ・セットの MINVERS 値は 13.1 以降であることが必要です。
- DEDB データベースが DBRC に登録されている必要があります。
- XRF ペアのアクティブ IMS システムが DEDB データベースの変更に関与する場合は、XRF ペアのアクティブ IMS システムとトラッキング IMS システムの両方が、同じ ACB ライブラリーを共用する必要があります。
- FDBR 領域によってトラッキングされる IMS システムが DEDB データベースの変更に関与する場合は、トラッキングされる IMS システムと FDBR 領域の両方が、同じ ACB ライブラリーを共用する必要があります。

DEDB 変更機能は、単一エリア・データ・セット (ADS) と多重エリア・データ・セット (ADS) の両方をサポートします。 DEDB データベース用に DBRC に定義するシャドー・エリア・データ・セットの数に応じて、DEDB 変更機能の出力を単一 ADS または多重 ADS として制御します。

DEDB 変更機能は、高速機能仮想記憶オプション (VSO) モードまたは共用仮想記憶オプション (SVSO) モードの DEDB データベースをサポートしません。VSO オプションまたは SVSO オプションを使用している DEDB データベースを、まず /VUNLOAD コマンドを使用して DASD にアンロードしてから、DEDB 変更ユーティリティーを実行する必要があります。

関連資料:

DEDB 変更ユーティリティー (DBFUDA00) (データベース・ユーティリティー)

DEDB 変更ユーティリティーを使用した、アクティブ DEDB エリアのサイズ属性の変更

DEDB 変更ユーティリティー (DBFUDA00) を使用して、DEDB エリアをオンラインでアクセス可能な状態に保ちながら、アクティブ DEDB エリアの SIZE、UOW、および ROOT パラメーターの DEDB エリア定義を変更して、DEDB エリアのサイズを増やすことができます。

1. 必要に応じて、SIZE、UOW、および ROOT パラメーターを変更し、AREA ステートメントを更新します。パラメーターの説明については、AREA ステートメント (システム・ユーティリティー) を参照してください。
2. SIZE、UOW、および ROOT の各パラメーターに対する更新を使用して DBDGEN ユーティリティーを実行し、アクティブ DEDB エリアの新規 DEDB DBD 定義を変更します。
3. 変更した DEDB DBD 定義を参照するすべての PSB について ACBGEN ユーティリティーを実行し、ステージング ACBLIB に保管します。

   IMS カタログを使用している場合は、DEDB 変更ユーティリティーの終了後、以下のいずれかの方法でカタログを更新します。

   - ACB 保守ユーティリティーと IMS Catalog Populate ユーティリティー (DFS3PU00) の両方を実行します。
   - IMS Catalog ACBGEN ユーティリティー (DFS3UACB) を実行します。このユーティリティーは、ACB の作成と IMS カタログのデータの設定を単一のジョブ・ステップで実行します。
4. 変更対象アクティブ DEDB エリアの 1 つ以上のシャドー DEDB エリア・データ・セットを割り振ります。
   - アクティブ DEDB エリアが多重エリア・データ・セットとして定義されている場合は、2 つから 6 つの新しいシャドー DEDB エリア・データ・セットを割り振ります。1 つのシャドー・イメージ・コピー・データ・セットが必要であるため、DEDB 変更ユーティリティーは、6 つのシャドー DEDB エリア・データ・セットをサポートします。多重エリア・データ・セットに 7 つのエリア・データ・セットが必要な場合は、DEDB 変更ユーティリティーの完了後に DEDB 作成ユーティリティーを実行して、7 つ目のエリア・データ・セットを作成します。

- DEDB 変更処理時に、単一エリア・データ・セットから多重エリア・データ・セットに変更できます。
  - 多重エリア・データ・セットに変更する場合は、2 つから 6 つの新しいシャドー DEDB エリア・データ・セットを割り振ります。
  - 1 つのシャドー・イメージ・コピー・データ・セットが必要であるため、DEDB 変更ユーティリティーは、6 つのシャドー DEDB エリア・データ・セットをサポートします。
- DEDB 変更処理時に、多重エリア・データ・セットから単一エリア・データ・セットに変更できます。
  - 単一エリア・データ・セットに変更する場合は、1 つの新しいシャドー DEDB エリア・データ・セットを割り振ります。

VSAM 制御インターバル・サイズは、新しい DEDB DBD 定義内の変更するアクティブ DEDB エリアの AREA ステートメントの SIZE= パラメーターと一致している必要があります。

5. DBRC INIT.ADS コマンド (SHADOW オプションを指定) を使用して、ステップ 4 (862 ページ) で割り振った、変更するアクティブ DEDB エリアのシャドー・データ・セットを DBRC に登録します。 1 つは SHADOW IC データ・セットとして DBRC に登録します。DEDB 変更ユーティリティーが実行されてアクティブ・エリア・データ・セットがシャドー・エリア・データ・セットに置き換えられるまで、シャドー・エリア・データ・セットは使用不可になります。

   ```
   INIT.ADS ADDN(name) ADSN(name) AREA(name) DBD(name) SHADOW
   ```

   データ・セットの DD 名の ADDN(*name*) およびデータ・セットの DS 名の ADSN(*name*) は、各シャドー・エリア・データ・セットでユニークで、かつ、関連するアクティブ・エリア・データ・セットのこれらの名前と異なる必要があります。
6. 変更するアクティブ DEDB エリアについて、更新した DEDB エリア定義を使用してデータ・セットをフォーマットします。DEDB 初期設定ユーティリティー (DBFUMIN0) で SHADOW 制御ステートメントを使用して、ステージング ACBLIB 内のデータ・セットをフォーマットします。 DEDB 初期設定ユーティリティーの実行後、これらのシャドー DEDB エリア・データ・セットは、まだ IMS オンライン・システムからアクセスできませんが、DBRC では「SHADOW ADS LIST:」で使用可能とマークされます。
7. 変更するアクティブ DEDB エリアのシャドー・イメージ・コピー・データ・セットとして、ステップ 4 (862 ページ) で割り振られたシャドー・エリア・データ・セットの 1 つを DBRC に登録します。 SHADOW オプションおよび IC オプションを指定して DBRC INIT.ADS コマンドを使用し、シャドー・イメージ・コピー・データ・セットを登録します。 DEDB 変更ユーティリティーが実行されてアクティブ DEDB エリア・データ・セットがシャドー・エリア・データ・セットに置き換えられるまで、シャドー・イメージ・コピー・データ・セットは使用不可になります。イメージ・コピーは、DEDB 変更ユーティリティーの実行中に作成されます。

   ```
   INIT.ADS   ADDN(name) ADSN(name)  AREA(name) DBD(name) SHADOW IC
   ```

シャドー・イメージ・コピー・データ・セットの DD 名の ADDN(name) およびシャドー・イメージ・コピー・データ・セットの DS 名の ADSN(name) は、関連するエリア・データ・セットのこれらの名前と異なっていなければなりません。

8. DEDB 初期設定ユーティリティー (DBFUMIN0) を使用して、ステージング ACBLIB 内の、変更するアクティブ DEDB エリアに関連した DEDB データベースの更新された DEDB エリア定義を用いて、変更するアクティブ DEDB エリアの 1 つのシャドー・イメージ・コピー・データ・セットをフォーマットします。

   DEDB 初期設定ユーティリティーの実行後、シャドー・イメージ・コピー・データ・セットは、まだ IMS オンライン・システムからアクセスできません。

   DEDB 初期設定ユーティリティーの一度の実行で、変更する DEDB エリアのシャドー・エリア・データ・セットとシャドー・イメージ・コピー・データ・セットをフォーマットすることができます。
9. ALTERAREA 関数を使用して DEDB 変更ユーティリティーを実行し、DEDB エリアがオンラインでアクセス可能な状態のまま、アクティブ DEDB エリアを変更します。

DEDB 変更ユーティリティーの ALTERAREA が正常終了したら、シャドー・エリア・データ・セットは、アクティブ・エリア・データ・セットにプロモートされます。以前アクティブであったエリア・データ・セットは、シャドー・エリア・データ・セットにデモートされ、IMS オンライン・システムからアクセスできなくなります。

以前アクティブであった DEDB エリア・データ・セットがもう不要である場合は、DELETE.ADS コマンドを使用して、DBRC から削除できます。

```
DELETE.ADS ADDN(name) AREA(name) DBD(name)
```

ADDN(name) は、古いアクティブ・エリア・データ・セットの DD 名です。

関連資料:

- DEDB 変更ユーティリティー (DBFUDA00) (データベース・ユーティリティー)
- DEDB 初期設定ユーティリティー (DBFUMIN0) (データベース・ユーティリティー)
- AREA ステートメント (システム・ユーティリティー)
- FIELD ステートメント (システム・ユーティリティー)
- SEGM ステートメント (システム・ユーティリティー)
- ACB Generation and Catalog Populate ユーティリティー (DFS3UACB) (システム・ユーティリティー)
- INIT.ADS コマンド (コマンド)

関連情報:

DEDB 変更ユーティリティーおよび IMS 生成ユーティリティーを使用した、アクティブ DEDB エリアを変更するときのランダマイザーの変更

DEDB 変更ユーティリティー (DBFUDA00) の REPLRAND 関数を使用する代わりに、ALTERAREA 関数でアクティブ DEDB のランダマイザー名を変更することもできます。 DEDB の変更では 2 ステージ・ランダマイザーのみを使用できます。ランダマイザーは、DEDB 内の既存のエリアのデータ位置を変更してはなりません。この機能により、ランダマイザーの名前変更や、将来新しい DEDB エリアを追加するための準備を行うことができます。

ランダマイザーを変更する場合、DEDB 変更ユーティリティーの ALTERAREA 関数は、順次従属 (SDEP) セグメントが定義されていない DEDB データベースをサポートします。

ランダマイザーを変更する場合、DEDB 変更ユーティリティーの REPLRAND 関数は、SDEP セグメントが定義されているかどうかに関係なく、DEDB データベースをサポートします。

1. ALTERAREA を使用して DEDB ランダマイザーをオンラインで変更するための準備をします。
   a. 変更したランダマイザーをアセンブルおよびリンク・エディットして、IMS.SDFSRESL または IMS.SDFSRESL STEPLIB 連結内のいずれかのライブラリーに格納します。
   b. 変更する DEDB エリアに関連付けられた DEDB データベースの定義の RMNAME パラメーターに新規の 2 ステージ・ランダマイザーの名前を指定します。
   c. DEDB DBD 定義で、変更するアクティブ DEDB エリアの AREA ステートメントの SIZE、UOW、および ROOT パラメーターを変更します。
   d. アクティブ DEDB エリアの DEDB DBD 定義を新しい定義に変更するために更新された SIZE、UOW、および ROOT パラメーター、および更新された RMNAME で DBDGEN ユーティリティーを実行します。
   e. 変更した DEDB DBD 定義を参照するすべての PSB について ACBGEN ユーティリティーを実行し、ステージング ACBLIB に保管します。
   f. 変更するアクティブ DEDB エリアの 1 つ以上のシャドー DEDB エリア・データ・セットを割り振ります。
      - アクティブ DEDB エリアが単一エリア・データ・セットとして定義されている場合は、1 つの新しいシャドー DEDB エリア・データ・セットを割り振ります。
      - アクティブ DEDB エリアが多重エリア・データ・セットとして定義されている場合は、2 つから 6 つの新しいシャドー DEDB エリア・データ・セットを割り振ります。1 つのシャドー・イメージ・コピー・データ・セットが必要であるため、DEDB 変更ユーティリティーは、6 つのシャドー DEDB エリア・データ・セットをサポートします。多重エリア・データ・セットに 7 つのエリア・データ・セットが必要な場合は、DEDB 変更ユーティリティーの完了後に DEDB 作成ユーティリティーを実行して、7 つ目のエリア・データ・セットを作成します。

DEDB 変更処理時に、単一エリア・データ・セットから多重エリア・データ・セットに変更できます。

- 多重エリア・データ・セットに変更する場合は、2 つから 6 つの新しいシャドー DEDB エリア・データ・セットを割り振ります。
- 1 つのシャドー・イメージ・コピー・データ・セットが必要であるため、DEDB 変更ユーティリティーは、6 つのシャドー DEDB エリア・データ・セットをサポートします。

DEDB 変更処理時に、多重エリア・データ・セットから単一エリア・データ・セットに変更できます。単一エリア・データ・セットに変更する場合は、1 つの新しいシャドー DEDB エリア・データ・セットを割り振ります。

VSAM 制御インターバル・サイズは、新しい DEDB DBD 定義内の変更するアクティブ DEDB エリアの AREA ステートメントの SIZE= パラメーターと一致している必要があります。

g. SHADOW オプションを指定した DBRC INIT.ADS コマンドを使用して、変更するアクティブ DEDB エリアの 1 つ以上のシャドー DEDB エリア・データ・セットを DBRC に登録します。 DEDB 変更ユーティリティーが実行されてアクティブ DEDB エリア・データ・セットがシャドー DEDB エリア・データ・セットに置き換えられるまで、シャドー DEDB エリア・データ・セットは使用不可になります。

```
INIT.ADS ADDN(name) ADSN(name) AREA(name) DBD(name) SHADOW
```

シャドー・エリア・データ・セットの DD 名の ADDN(name) およびシャドー・エリア・データ・セットの DS 名の ADSN(name) は、各シャドー・エリア・データ・セットでユニークで、かつ、関連するアクティブ・エリア・データ・セットのこれらの名前と異なる必要があります。

h. DEDB 初期設定ユーティリティー (DBFUMIN0) を使用して、ステージング ACBLIB 内の、変更するアクティブ DEDB エリアに関連した DEDB データベースの更新された DEDB エリア定義 (SIZE、UOW、および ROOT) で、変更するアクティブ DEDB エリアの 1 つ以上のシャドー DEDB エリア・データ・セットをフォーマットします。 DEDB 初期設定ユーティリティーの実行後、シャドー DEDB エリア・データ・セットは、まだ IMS オンライン・システムからアクセスできません。

i. 変更するアクティブ DEDB エリアのシャドー・イメージ・コピー・データ・セットとして、ステップ 1f (865 ページ) で割り振られたシャドー・エリア・データ・セットの 1 つを DBRC に登録します。 SHADOW オプションおよび IC オプションを指定して DBRC INIT.ADS コマンドを使用し、シャドー・イメージ・コピー・データ・セットを登録します。 DEDB 変更ユーティリティーが実行されてアクティブ DEDB エリア・データ・セットがシャドー DEDB エリア・データ・セットに置き換えられるまで、シャドー・イメージ・コピー・データ・セットは使用不可になります。

```
INIT.ADS ADDN(name) ADSN(name) AREA(name) DBD(name) SHADOW IC
```

シャドー・イメージ・コピー・データ・セットの DD 名の ADDN(name) およびシャドー・イメージ・コピー・データ・セットの DS 名の

ADSN(name) は、関連するアクティブ・エリア・データ・セットのこれらの名前と異なっていなければなりません。

j. DEDB 初期設定ユーティリティー (DBFUMIN0) を使用して、ステージング ACBLIB 内の、変更するアクティブ DEDB エリアに関連した DEDB データベースの更新された DEDB エリア定義 (SIZE、UOW、および ROOT) を用いて、変更するアクティブ DEDB エリアの 1 つのシャドー・イメージ・コピー・データ・セットをフォーマットします。

DEDB 初期設定ユーティリティーの実行後、シャドー・イメージ・コピー・データ・セットは、まだ IMS オンライン・システムからアクセスできません。

DEDB 初期設定ユーティリティーの一度の実行で、変更する DEDB エリアのシャドー・エリア・データ・セットとシャドー・イメージ・コピー・データ・セットをフォーマットすることができます。

2. 以下の手順を実行して、DEDB データベースの DEDB ランダマイザーをオンラインで変更します。
   a. 変更したランダマイザーをアセンブルおよびリンク・エディットして、IMS.SDFSRESL または IMS.SDFSRESL STEPLIB 連結内のいずれかのライブラリーに格納します。
   b. DEDB DBD 定義で、変更するアクティブ DEDB エリアに関連した DEDB データベースの DBD ステートメントの RMNAME パラメーターを別の 2 ステージ DEDB ランダマイザー名に変更します。
   c. DEDB DBD 定義内にある、新しい変更済みの DEDB AREA ステートメントおよび DEDB DBD ステートメントのランダマイザーの変更を使用して、DBDGEN ユーティリティーを実行します。
   d. 変更した DEDB DBD 定義を参照するすべての PSB について ACBGEN ユーティリティーを実行し、ステージング ACBLIB に保管します。
3. ALTERAREA 関数を使用して DEDB 変更ユーティリティーを実行し、DEDB エリアがオンラインでアクセス可能な状態のまま、アクティブ DEDB エリアを変更してランダマイザーも変更します。 DEDB 変更ユーティリティーは、実行されると 1 つのアクティブ DEDB エリアを変更してランダマイザーも変更します。DEDB ランダマイザーは、DEDB 変更ユーティリティーの ALTERAREA 関数の一環として変更されます。ランダマイザーの変更は、DEDB データベース・ランダマイザーを変更するための DEDB 変更ユーティリティーによって変更されない DEDB エリア間でのデータ分散には影響しません。新しいランダマイザーは、それが他の DEDB データベースで既存のランダマイザーでない限り、他の DEDB データベースと共用されません。

   DEDB 変更ユーティリティーが正常に実行された後、シャドー・エリア・データ・セットがアクティブ・エリア・データ・セットにプロモートされます。以前アクティブであったエリア・データ・セットは、シャドー・エリア・データ・セットにデモートされます。以前アクティブであったエリア・データ・セットは、IMS オンライン・システムからアクセスできなくなります。

   以前アクティブであった DEDB エリア・データ・セットがもう不要である場合は、DELETE.ADS コマンドを使用して、DBRC から削除できます。

```
DELETE.ADS ADDN(name) AREA(name) DBD(name)
```

ADDN(name) は、古いアクティブ・エリア・データ・セットの DD 名です。

4. IMS カタログを使用している場合は、DEDB 変更ユーティリティーの終了後、以下のいずれかの方法でカタログを更新します。
   - ACB 保守ユーティリティーと IMS Catalog Populate ユーティリティー (DFS3PU00) の両方を実行します。
   - IMS Catalog ACBGEN ユーティリティー (DFS3UACB) を実行します。このユーティリティーは、ACB の作成と IMS カタログのデータの設定を単一のジョブ・ステップで実行します。

関連資料:

DEDB 変更ユーティリティー (DBFUDA00) (データベース・ユーティリティー)

DEDB 初期設定ユーティリティー (DBFUMIN0) (データベース・ユーティリティー)

AREA ステートメント (システム・ユーティリティー)

DBD ステートメント (システム・ユーティリティー)

FIELD ステートメント (システム・ユーティリティー)

SEGM ステートメント (システム・ユーティリティー)

IMS Catalog Populate ユーティリティー (DFS3PU00) (システム・ユーティリティー)

ACB Generation and Catalog Populate ユーティリティー (DFS3UACB) (システム・ユーティリティー)

INIT.ADS コマンド (コマンド)

DEDB 変更ユーティリティーを使用したアクティブ DEDB データベース・ランダマイザーのオンラインでの置換

アクティブ DEDB データベース・ランダマイザーの置換は、DEDB 変更ユーティリティー (DBFUDA00) の REPLRAND 関数を使用して、DEDB データベースがオンラインでアクセス可能な状態の間に実行できます。DEDB 変更ユーティリティーのランダマイザーの変更では、SDEP セグメントが定義されている DEDB データベースも定義されていない DEDB データベースもサポートされます。

DEDB ランダマイザーが ALTERAREA 関数の一環として変更されており、オリジナルの DEDB ランダマイザー名に戻したい場合は、REPLRAND 関数を使用して DEDB 変更ユーティリティーを実行します。これにより、DEDB 変更ユーティリティーの ALTERAREA 関数の終了後に、新しいランダマイザー名がオリジナルの DEDB ランダマイザー名に変更されます。

1. DEDB ランダマイザーをオンラインで置き換えるための準備をします。
   a. 変更したランダマイザーをアセンブルおよびリンク・エディットして、IMS.SDFSRESL または IMS.SDFSRESL STEPLIB 連結内のいずれかのライブラリーに格納します。

b. DEDB DBD 定義で、変更するアクティブ DEDB エリアに関連した DEDB データベースの DBD ステートメントの RMNAME パラメーターを別の 2 ステージ DEDB ランダマイザー名に変更します。
c. DEDB DBD 定義の DBD ステートメントの RMNAME パラメーターを使用して、DBDGEN ユーティリティーを実行します。
d. 変更した DEDB DBD 定義を参照するすべての PSB について ACBGEN ユーティリティーを実行し、ステージング ACBLIB に保管します。

2. REPLRAND 関数を使用して DEDB 変更ユーティリティーを実行し、DEDB データベースがオンラインでアクセス可能な状態のまま、ランダマイザーを置き換えます。
3. IMS カタログを使用している場合は、DEDB 変更ユーティリティーの終了後、以下のいずれかの方法でカタログを更新します。
   - ACB 保守ユーティリティーと IMS Catalog Populate ユーティリティー (DFS3PU00) の両方を実行します。
   - IMS Catalog ACBGEN ユーティリティー (DFS3UACB) を実行します。このユーティリティーは、ACB の作成と IMS カタログのデータの設定を単一のジョブ・ステップで実行します。

DEDB 変更ユーティリティーが正常に完了した後、DEDB データベースがオンラインでアクセス可能な状態のまま、DEDB データベースの古いランダマイザーが新しいランダマイザーに置き換わります。

関連資料:

- DEDB 変更ユーティリティー (DBFUDA00) (データベース・ユーティリティー)
- DBD ステートメント (システム・ユーティリティー)
- FIELD ステートメント (システム・ユーティリティー)
- SEGM ステートメント (システム・ユーティリティー)

オンライン・システムでのデータベースの動的な変更

動的リソース定義によって、オンライン IMS システムに新しいデータベースを追加する作業、データベースの特定の属性をオンラインで更新する作業、あるいはオンライン IMS システムからデータベースを除去する作業を、IMS をシャットダウンしたり、システム定義処理を実行したりすることなく行えます。

動的リソース定義を使用するには、IMS システムでこの機能を使用可能にする必要があります。

動的リソース定義を使用したデータベースの変更をサポートするタイプ 2 コマンドは次のとおりです。

- CREATE DB
- DELETE DB
- UPDATE DB

QUERY DB コマンドを使用して、他のコマンドによって変更できるデータベース属性を表示することができます。

また、動的リソース定義を使用してデータベース・リソース記述子を定義して保管し、これを共通の属性をもつ新しいデータベースを作成するためのテンプレートとして使用することもできます。例えば、CREATE DB コマンドを使用してオンライン・システムにデータベースを追加する際に、各属性を個別に指定する代わりに `CREATE DB NAME(db_name) LIKE(DESC(db_desc_name))` を指定できます。 LIKE() パラメーターは、既存のデータベースまたはデータベース記述子を基に新しいデータベースを作成する場合にも使用できます。

動的リソース定義を使用してデータベースにオンラインで加えた変更をコールド・スタート後も維持するには、データベース定義をリソース定義データ・セット (RDDS) (または、IMS でリポジトリーが使用可能な場合は IMSRSC リポジトリー) にエクスポートするか、またはシステム定義によって IMS MODBLKS データ・セットに追加してから、コールド・スタート時にインポートする必要があります。ウォーム・スタートおよび緊急時再始動後は、IMS によって動的リソース定義の変更がログからリカバリーされます。

IMS で IMSRSC リポジトリーの使用が有効になっている場合、IMS の変更リストのリソース定義は、ウォーム・リスタート時または緊急時再始動時の最後に読み取られます。IMS は、リソース・マネージャー (RM) を呼び出し、IMS の変更リストを読み取ります。IMS 変更リストは、RM によってリポジトリー内で保守されます。

変更リストが存在する場合、IMS は、変更リストにあるすべてのリソースと記述子の、保管されているリソース定義を、IMSRSC リポジトリーからインポートします。IMS 変更リストは、IMS ログが処理された後に処理されます。変更リスト内のリソースと記述子は静止され、保管済みリソース定義がリポジトリーからインポートされるまで使用することができません。IMS システム用の変更リストは、ウォーム・リスタートまたは緊急時再始動の終わり、または IMS コールド・スタートの終わりに削除されます。

関連概念:

- IMSRSC リポジトリーのリソース・リスト (システム定義)
- IMSRSC リポジトリーの概要 (システム定義)

関連タスク:

591 ページの『オンライン・システムへのデータベースの導入』

オンライン IMS システムでのデータベース属性の動的な変更

UPDATE DB コマンドを使用して、オンライン・データベースの属性と状況を変更することができます。

UPDATE DB コマンドを使用することにより、例えば、データベースを使用可能にしたりオフラインにしたりするほか、スケジューリングの停止、更新の停止、およびデータベースのロックとアンロックを行うことができます。データベースの属性を変更する場合は、事前にデータベースへのアクセスを停止する必要があります。

UPDATE DB コマンドでは、複数のデータベース名を指定できます。指定したデータベースは個別に処理され、個々のデータベースで処理が失敗すると、そのデータベースでの失敗について説明する条件コードが IMS から返されます。

データベースの現行の属性と状況を表示するには、QUERY DB コマンドを実行します。

UPDATE DB コマンドを使用して次の操作を行うことができます。

- データベース、区画、および DEDB エリアの開始と停止
- データベースのアクセス・タイプの設定
- データベースの RESIDENT 属性の変更
- データベースのロックおよびアンロック
- IMSplex 環境でのデータベースのグローバルな状況の設定

オンライン・データベースの定義属性に対する変更は、データベース定義をリソース定義データ・セット (RDDS) または IMSRSC リポジトリーにエクスポートしないかぎり、IMS をコールド・スタートすると失われます。

データベース定義を RDDS またはリポジトリーにエクスポートするには、EXPORT DEFN コマンドを使用します。また、定義がシステム・チェックポイントで自動的に RDDS にエクスポートされるように IMS を構成することもできます。自動エクスポートを使用可能にするには、AUTOEXPORT=AUTO または AUTOEXPORT=RDDS を IMS PROCLIB データ・セット内の DFSDFxxx メンバーの DYNAMIC_RESOURCES セクションに指定します。

関連資料:

UPDATE DB コマンド (コマンド)

QUERY DB コマンド (コマンド)

オンライン IMS システムからのデータベースの動的な除去

オンライン IMS システムからデータベースを除去するには、DELETE DB コマンドを使用します。

MSDB データベースをオンライン・システムから除去する場合は、他のデータベース・タイプとは異なる手順が必要です。

DB の状況および DB が PSB から参照されているかどうかを表示するには、コマンド QUERY DB SHOW(WORK) および QUERY DB SHOW(PGM) を使用します。

MSDB データベース以外のデータベースをオンライン IMS システムから除去するには、次のようにします。

1. 以下のいずれかを実行してデータベースへのアクセスを停止します。
   - UPD DB NAME(name) STOP(ACCESS)
   - /DBR DB name
2. データベースが論理関係を使用している場合は、データベースをオンライン・システムから除去する前に論理関係を除去する必要があります。

a. 関係を除去するように DBD を変更します。
b. 変更した DBD ステートメントで DBDGEN および ACBGEN を実行します。
c. 新しい定義をオンラインにします。

3. プログラム・ディレクトリー制御ブロック (PDIR) 内のすべての PSB から、データベースへの参照を除去するために、PDIR を完全に削除するか、または PSB からデータベースへの参照を削除して ACBLIB のオンライン変更機能を実行する。
4. DELETE DB コマンドを発行します。

オンライン **IMS** システムからの **MSDB** データベースの動的な除去**:**

MSDB データベースは拡張共通ストレージ域 (ECSA) 内にあるため、MSDB データベースをオンライン・システムから除去するには、MSDBLOAD を指定して IMS を再始動する必要があります。

MSDB データベースをオンライン IMS システムから除去するには、次のようにします。

1. MSDB 保守ユーティリティー (DBFDBMA0) を使用して、MSDBINIT データ・セットから MSDB データベースを削除する。
2. オンライン変更機能を使用して、ACB ライブラリーから MSDB データベース (BHDR 制御ブロック) を除去する。
3. すべての PSB から、MSDB データベースへの参照を除去する。
4. IMS を通常どおりにシャットダウンする。
5. MSDBLOAD キーワードを指定して IMS をウォーム・スタートする。MSDBINIT データ・セットは既に存在せず、ACBLIB にもこのデータ・セットに対応する項目がないため、IMS はデータベースを認識しますが、MSDB データベースとしては認識しません。
6. データベースに対して /DBR コマンドを実行する。/DBR コマンドをデータベースに対して実行できるのは、IMS がこのデータベースを MSDB と見なさなくなったためです。
7. コマンド DELETE DB NAME(*MSDB_name*)を出す。

オンライン変更機能を使用したデータベース変更の活動化

オンライン変更機能を使用すると、IMS を停止せずにオンラインで、HALDB マスター・データベースなどの全機能データベース、および高速機能 DEDB データベースを追加、変更、および削除することができます。

MSDB は、オンライン変更機能をサポートしません。

DEDB 用のオンライン変更機能を使用すれば、データベース・レベルの変更とエリア・レベルの変更を行うことができます。データベース・レベルの変更は DEDB の構造に影響を与えます。このレベルの変更には、エリアの追加または削除、セグメント・タイプの追加、あるいはランダム化ルーチンの変更などがあります。エリア・レベルの変更には、エリア (IOVF、DOVF、CI) のサイズの増減が含まれます。エリア・レベルの変更では、ユーザーは /DBRECOVERY コマンドを使用してその

エリアのみを停止する必要があります。データベース・レベルの変更では、ユーザーは DEDB のすべてのエリアを停止する必要があります。

2 ステージ・ランダマイザーは、DEDB 全体にデータベース・レコードを分散する標準ランダマイザーと異なり、データベース・レコードをエリア内に分散します。2 ステージ・ランダマイザーを使用すると、各エリアのルート・アドレス可能割り振りへの変更はエリア・レベルの変更であり、影響を受けるエリアのみを停止すれば十分です。

全機能データベース、HALDB マスター・データベース、および DEDB データベースに対してオンライン変更を使用するには、以下のステップを実行します。

1. オンライン変更のためのデータ・セットを割り振る。方法については、「*IMS V13* インストール」を参照してください。
2. システム定義の DATABASE ステートメント (場合によっては APPLCTN ステートメントにも) に対して追加、変更、削除を行う必要がある場合は、MODBLKS システム定義を実行する (詳しくは、*IMS V13* システム管理 を参照)。
3. 必要な DBDGEN、PSBGEN、および ACBGEN (「*IMS V13* システム・ユーティリティー」を参照) を実行する。

   注**:** ACBGEN 実行の結果に伴う ACBLIB メンバーへのすべての変更は、オンライン変更後、オンライン・システムで使用することができます (ただし、変更された PSB リソースおよび DBD リソースがオンライン・システムで定義されている場合)。
4. IMS システムのセキュリティー機能のセキュリティー定義を、新規データベースを含むよう更新する。セキュリティー機能には、RACF、別の外部セキュリティー製品、および出口ルーチンがあります。IMS のセキュリティーについて詳しくは、*IMS V13* システム管理 を参照してください。
5. 追加しようとしているデータベースに、データベース・データ・セットを割り振る。
6. データベースをロードする。
7. 高速機能の場合には、データベースをロードする前に、オンライン変更を完了する必要がある。また、高速機能はデータベースをオンラインでのみロードできます。バッチ・ジョブは使用できません。
8. z/OS 環境で動的な割り振りを使用する場合には、動的割り振りユーティリティーを実行する。
9. オンライン変更コピー・ユーティリティー (DFSUOCU0) を使用して、更新されたステージング・ライブラリーを非アクティブ・ライブラリーにコピーする (このユーティリティーの実行方法については、「*IMS V13* システム・ユーティリティー」を参照)。
10. オペレーター・コマンドを出して、ユーザーの非アクティブ・ライブラリーをアクティブ・ライブラリーにする (オンライン変更に使用されるコマンドについては、「*IMS V13* オペレーションおよびオートメーション」を参照)。

アクティブ ACBLIB データ・セットを変更したために z/OS 環境のデータベースを再編成する必要がある場合は、`/DBR` を発行してデータベースの割り振りを解除してから、`/MODIFY COMMIT` コマンドを実行して ACBGEN の変更を取り入れる必要

があります。変更または削除するデータベースのエリアをオフラインで取得するには、 /MODIFY COMMIT コマンドを出す前に /DBR コマンド、/DBD コマンド、または /STA DATABASE ACCESS= コマンドの実行が完了していなければなりません。

関連概念:

569 ページの『アプリケーション制御ブロック (ACBGEN) の作成』

899 ページの『オンライン変更と HALDB データベース』

851 ページの『オンライン変更および HALDB の変更』

オンラインでの ACB ライブラリー・メンバーの変更

IMS の稼働中に ACB ライブラリー・メンバー・オンライン変更機能を使用して、ACB ライブラリーの個々のメンバーを追加または変更し、オンラインにすることができます。

ACB ライブラリー・メンバー・オンライン変更機能は、オンライン変更の影響を受けるリソースのみを静止するので、変更の影響を受けないリソースはオンライン変更処理の実行中もアクティブなままとなります。

オンライン変更機能で変更または削除しようとする ACB ライブラリー・メンバーに、IMS で使用するためにオンライン・ストレージにロードされたものがある場合は、オンライン変更時に IMS によってそれらがストレージから除去されます。非常駐の ACB メンバーは、それらを必要とするアプリケーション・プログラムが次回スケジュールされるときにストレージに再ロードされます。

関連概念:

ACB ライブラリー・メンバー・オンライン変更 (システム管理)

関連タスク:

オンラインでの IMS.ACBLIB メンバーの変更または追加 (システム管理)

オンライン変更機能、DEDB、および IFP 領域と MPP 領域の可用性

DEDB にアクセスする IFP 領域および MPP 領域の可用性を保持したまま、DEDB に変更を加えることができます。IFP/MPP の実行中に DEDB に対してデータベース・レベルの変更が行われた場合、アプリケーションは疑似異常終了し、DEDB に対する次の DL/I 呼び出し時に PSB がスケジュール変更されます。

IFP/MPP の実行中に DEDB に対してデータベース・レベルの変更が行われた場合、アプリケーションは疑似異常終了し、DEDB に対する次の DL/I 呼び出し時に PSB がスケジュール変更されます。

DEDB には 2 つのレベルの変更が可能です。データベース・レベルの変更では、以下のことが可能です。

1. DEDB の追加または削除
2. セグメント・タイプの追加または削除
3. セグメントとそのフィールドの追加、変更、または削除
4. セグメント圧縮ルーチンの追加、変更、または削除
5. データ・キャプチャー出口ルーチンの追加、変更、または削除

6. ランダマイザーの変更
7. エリアの追加または削除
8. エリア RAP スペース割り振りの変更 (ランダマイザーが 2 ステージ・ランダマイザーではない場合)

エリア・レベルの変更では、以下のことが可能です。

1. エリア RAP スペース割り振りの変更 (ランダマイザーが 2 ステージ・ランダマイザーである場合)
2. DOVF または IOVF スペース割り振りの変更
3. SDEP スペース割り振りの変更
4. CI サイズの変更

エリア・レベルの変更、およびデータベース・レベルの変更の項目 4 から 8 の場合、BUILD DBD (BUILD PSB ではなく) が必要になります。このケースでは、項目 4 および 5 で、定義済みの PSB SENSEG が追加または削除される出口ルーチンを参照している場合を除いて、PSB は変更されません。スケジュールされた PSB に変更がない場合に限り、オンライン変更を使用すれば、DEDB にアクセスする IFP 領域および MPP 領域の可用性を保持しながら、DEDB の変更を行うことができます。 DEDB に対する次の DL/I 呼び出し時に、アプリケーションは ABENDU0777 で疑似異常終了し、PSB がスケジュール変更されます。メッセージ DFS2834I が出されます。 DEDB データベース変更の項目 1 から 5 のような PSB に対するその他の変更、全機能データベースの変更、またはオンライン変更を使用した PSB 変更の場合、IFP および MPP 領域を停止させる必要があります。

以下の手順は、IFP/MPP の実行中に、DEDB に対するデータベース・レベルの変更を行うために必要なステップを示しています。

1. ユーザーが開発した特定のアプリケーション・プログラムまたは OEM ユーティリティーを使用し、既存のシステム定義を介して DEDB をアンロードする。
2. DBDGEN、PSBGEN および ACBGEN を使用して、アプリケーション制御ブロックを生成し、DEDB 構造の変更をインプリメントする。変更後のアプリケーション制御ブロックまたは新しいアプリケーション制御ブロックは、アクティブ IMS システムの、オフラインである ACBLIB のステージング・コピーに組み込む必要があります。
3. オンライン変更ユーティリティー DFSUOCU0 を実行して、変更済みの ACBLIB をステージング ACBLIB から、アクティブ IMS システムに対してオンラインである ACBLIB の非アクティブ (A または B) コピーに移動する。
4. 通常の /DBR コマンド・シーケンスを入力して、DEDB へのアクセスをアクティブ IMS システムから除去する。/DBR で GLOBAL オプションが使用された場合、DBRC に追加許可禁止フラグが設定されます。これにより、PSB のスケジュールが変更されると、U0458 異常終了が発生します。通常は GLOBAL オプションを使用するデータ共用カスタマーの場合、NOPFA キーワードを追加することが推奨されます (/DBR DB NOFPA)。
5. ACBLIB 変更のための PREPARE 処理用のオンライン変更コマンド・シーケンスを入力して実行する。

6. ACBLIB 変更のための COMMIT/ABORT 処理用のオンライン変更コマンド・シーケンスを入力して実行する。オンライン IMS システムは、ACBLIB のアクティブ (A または B) コピーの使用から非アクティブ (A または B) コピーの使用に切り替えます。
7. 新しいシステム定義を使用して、すべての DEDB エリア・データ・セットを削除し、定義し、初期設定する。
8. 通常の /START DATABASE および /START AREA コマンドを入力して、DEDB およびそのエリアをアクティブ IMS システムでアクセス可能にする。
9. ユーザーが開発した特定のアプリケーション・プログラムまたは OEM ユーティリティーを使用し、DEDB に関する変更システム定義を介して DEDB を再ロードする。
10. 新しく変更した DEDB への最初のアクセス時に、アプリケーションは疑似異常終了し、PSB がスケジュール変更されます。メッセージ DFS2834I が表示されます。

    PSB がスケジュール変更されると、IFP および MPP に関するトランザクションがもう一度試行されます。コミット処理の完了前にアプリケーションが DEDB へのアクセスを試みると、'FH' 状況がアプリケーションに戻されます。 DEDB は、DEDB 用のランダマイザーが /DBR コマンドによってアンロードされるので、アクセス不能になります。

BMP または DBCTL スレッドがアクティブな間にデータベース・レベルまたはエリア・レベルの変更が DEDB に対して行われると、コミット処理は失敗し、メッセージ DFS3452 が出されます。

関連情報:

DFS3452 (メッセージおよびコード)

オンライン変更と DEDB ランダマイザーと出口ルーチン

ランダム化ルーチンはデータベース・レコードの位置を、DEDB 内の AREA と AREA 内のルート・アンカー・ポイント (RAP) によって決定します。 DEDB ランダマイザーの変更は、データベース・レベルの変更です。

新しいランダム化ルーチンは、DEDB 内のすべてのデータベース・レコードの位置 (AREA および RAP) に影響を与えます。ランダマイザーは、DBD パラメーター RMNAME= によって DEDB に対して定義します。

ランダマイザーの変更には、アクティブ IMS への新しい特定のランダマイザーの取り込み、または使用中の既存のランダマイザーの 1 つ以上の DEDB による変更が関係します。

ランダマイザーの名前は、DBD パラメーター RMNAME= で指定します。既存の DEDB のために新しいランダマイザーを取り込む場合には、以下の手順の各ステップだけでなく、新しいランダマイザー名を指定したデータベースの DBDGEN および ACBGEN が必要になります。

オンライン変更を使用した新規 DEDB ランダム化ルーチンの導入

1. ユーザーが開発した特定のアプリケーション・プログラムまたは相手先商標製造会社 (OEM) ユーティリティーを使用して、DEDB および現行ランダマイザーをアンロードする。
2. 新しいランダマイザーをアセンブルおよびリンク・エディットして、IMS SDFSRESL または STEPLIB 連結内のいずれかのライブラリーに入れる。
3. DBD パラメーター RMNAME= に新しいランダマイザーを指定して、DEDB 用の DBDGEN を実行する。
4. 新しいランダマイザーを含む DEDB 定義をインプリメントするアプリケーション制御ブロックを作成するためにも、ACBGEN が必要です。変更後のアプリケーション制御ブロックまたは新しいアプリケーション制御ブロックは、アクティブ IMS システムの、オフラインである ACBLIB のステージング・コピーに組み込む必要があります。
5. オンライン変更ユーティリティー DFSUOCU0 を実行して、変更済みの ACBLIB をステージング ACBLIB から、アクティブ IMS システムに対してオンラインである ACBLIB の非アクティブ (A または B) コピーに移動する。
6. タイプ 1 の /DBR DB オペレーター・コマンド・シーケンスまたはタイプ 2 の `UPDATE DB STOP(ACCESS)` オペレーター・コマンド・シーケンスを入力して、アクティブ IMS システムから DEDB へのアクセスを除去し、ランダマイザーをアンロードする。タイプ 2 コマンド `UPDATE DB STOP(ACCESS)` を実行する場合は OPTION(NORAND) パラメーターを使用しないでください。NORAND パラメーターを指定するとランダマイザーがアンロードされなくなります。
7. ACBLIB 変更のための PREPARE 処理用のオンライン変更コマンド・シーケンスを入力して実行する。
8. ACBLIB 変更のための COMMIT/ABORT 処理用のオンライン変更コマンド・シーケンスを入力して実行する。オンライン IMS システムは、ACBLIB のアクティブ (A または B) コピーの使用から非アクティブ (A または B) コピーの使用に切り替えます。
9. 新しいシステム定義を使用して、すべての DEDB エリア・データ・セットを削除し、定義し、初期設定する。
10. タイプ 1 コマンド `/START DATABASE` と `/START AREA`、またはタイプ 2 コマンド `UPDATE DB START(ACCESS) AREA(*)` を入力して、DEDB とそのエリアをアクティブ IMS システムでアクセス可能にする。
11. ユーザーが開発した特定のアプリケーション・プログラムまたは OEM ユーティリティーを使用し、新しいランダム化ルーチンが有効である DEDB を再ロードする。

オンライン変更を使用した **DEDB** ランダム化ルーチンの変更**:**

1 つ以上の DEDB がすでに使用しているランダマイザーに対して変更を行った場合、当該ランダマイザーを使用中のすべての DEDB を、変更処理に組み込む必要があります。

アクティブ IMS システム内のいずれかの DEDB のためにランダマイザーの既存のバージョンがすでにロードされている場合、変更したランダマイザーは取り込まれません。既存のバージョンがもはや使用されていないことは、メッセージ DFS2838I

内にキーワード GONE があることで判別できます。また、ランダマイザー・モジュールがいずれかのライブラリーからストレージへロードされたことは、メッセージ DFS2842I 内にキーワード LOADED があることで判別できます。

DEDB ランダマイザーの変更には、下記の手順が必要です。ランダマイザーの名前は変わらないので、DBDGEN、ACBGEN およびオンライン変更コマンド・シーケンスは適用できません。

1. ユーザーが開発した特定のアプリケーション・プログラムまたは OEM ユーティリティーを使用し、DEDB および既存のランダマイザーをアンロードする。これは、変更するランダマイザーを使用するすべての DEDB について実行する必要があります。
2. タイプ 1 コマンド `/DBR DATABASE` またはタイプ 2 コマンド `UPDATE DB STOP(ACCESS)` を入力して、アクティブ IMS システムから DEDB へのアクセスを除去する。`/DBR DATABASE` コマンドは、オペランドで指定された DEDB のランダマイザーをアンロードします。 `UPDATE DB STOP(ACCESS)` コマンドも、OPTION(NORAND) パラメーターが指定されない限り、オペランドとして指定されている DEDB のランダマイザーをアンロードします。そのランダマイザーを参照するすべての DEDB が停止されると、ランダマイザーはアクティブ IMS システムから除去されます。停止されていない DEDB が IMS システムから除去されたランダマイザーを参照すると、次の DL/I 呼び出しで U1021 異常終了が発生します。
3. 変更したランダマイザーをアセンブルおよびリンク・エディットして、IMS SDFSRESL または IMS SDFSRESL STEPLIB 連結内のいずれかのライブラリーに入れる。
4. すべての DEDB エリア・データ・セットを削除、定義、初期設定して、変更したランダマイザーを使用する DEDB を再ロードできるように準備する。
5. タイプ 1 コマンド `/START DATABASE` またはタイプ 2 コマンド `UPDATE DB START(ACCESS)` を実行して、変更されたランダマイザーを使用する DEDB へのアクセスを再開する。DEDB の場合、`/START DATABASE` コマンドまたは `UPDATE DB START(ACCESS)` コマンドによってランダマイザーがロードされます。全エリアへのアクセスを同時に再開するには、タイプ 2 コマンド `UPDATE DB START(ACCESS) AREA(*)` を実行します。
6. ユーザーが開発した特定のアプリケーション・プログラムまたは OEM ユーティリティーを使用し、変更したランダム化ルーチンが有効である DEDB を再ロードする。

オンライン変更を使用した **DEDB** ランダム化ルーチンの削除**:**

古いランダム化ルーチンを使用する DEDB をすべてアンロードし、それらの DEDB に対してタイプ 1 コマンド /DBR またはタイプ 2 コマンド UPDATE DB STOP(ACCESS) を実行すると、古いランダム化ルーチンを削除できるようになります。

アクティブ IMS システムからランダム化ルーチンを削除するには、 876 ページの『オンライン変更と DEDB ランダマイザーと出口ルーチン』のステップに従います。ランダム化ルーチンが削除されると、メッセージ DFS2838 が生成されます。

タイプ 2 コマンド UPDATE DB STOP(ACCESS) を使用する場合は、OPTION(NORAND) パラメーターを指定しないでください。NORAND パラメーターを指定すると DEDB ランダム化ルーチンがストレージからアンロードされなくなるためです。

SDFSRESL を共用していないデータ共用 IMS システムを使用しているユーザーは、両方のシステムからランダム化ルーチンを削除するよう注意してください。

オンライン変更と **DEDB** セグメント圧縮ルーチン**:**

セグメント圧縮ルーチンは、セグメントに固有であり、DBD SEGM パラメーター ("COMPRTN=") によって DEDB に対して定義されます。

セグメント圧縮ルーチンの追加、変更、または削除の手順および制約事項は、DEDB ランダム化ルーチンの場合と同じです。

オンライン変更と **DEDB** データ・キャプチャー出口ルーチン**:**

データ・キャプチャー出口ルーチンは、DBD ステートメントで DEDB について定義するか、SEGM ステートメント ("EXIT=") で特定のセグメントについて定義するか、あるいはその両方を行うことができます。

単一の DBD または SEGM ステートメントに複数の出口ルーチンを指定することができます。

オンライン変更を使用した新しい ***DEDB*** データ・キャプチャー出口ルーチンの追加*:*

オンライン変更機能を使用して、新しいデータ・キャプチャー出口ルーチンを DEDB に追加することができます。

新しいデータ・キャプチャー出口ルーチンを追加するには、以下の手順を行ってください。

1. 新しい出口ルーチンをアセンブルおよびリンク・エディットして、IMS.SDFSRESL、または IMS.SDFSRESL STEPLIB 連結内のいずれかのライブラリーに格納する。
2. DBD または SEGM パラメーター "EXIT=" に新しい出口ルーチンを指定して、DEDB 用の DBDGEN を実行する。
3. 新しい出口ルーチンを組み込む DEDB 定義をインプリメントするアプリケーション制御ブロックを作成するために、ACBGEN も必要です。変更後のアプリケーション制御ブロックまたは新しいアプリケーション制御ブロックは、アクティブ IMS システムの、オフラインである ACBLIB のステージング・コピーに組み込む必要があります。
4. オンライン変更コピー・ユーティリティー DFSUOCU0 を実行して、変更済みの ACBLIB をステージング ACBLIB から、アクティブな IMS システムに対してオンラインである ACBLIB の非アクティブ (A または B) コピーに移動する。
5. 通常の /DBR コマンド・シーケンスを入力して、DEDB へのアクセスをアクティブ IMS システムから除去する。

6. ACBLIB 変更のための PREPARE 処理用のオンライン変更コマンド・シーケンスを入力して実行する。
7. ACBLIB 変更のための COMMIT/ABORT 処理用のオンライン変更コマンド・シーケンスを入力して実行する。オンライン IMS システムは、ACBLIB のアクティブ (A または B) コピーの使用から非アクティブ (A または B) コピーの使用に切り替えます。
8. 通常の /START DATABASE および /START AREA コマンドを入力して、DEDB およびそのエリアをアクティブ IMS システムでアクセス可能にする。

オンライン変更を使用した既存の ***DEDB*** データ・キャプチャー出口ルーチンの変更*:*

オンライン変更機能を使用して、既存の DEDB データ・キャプチャー出口ルーチンを変更することができます。

既存のデータ・キャプチャー出口ルーチンを変更するには、以下のステップを行ってください。

1. 特定のデータ・キャプチャー出口ルーチンを使用している DEDB にアクセスしている従属領域が正常に終了できるようにする。
2. 変更した出口ルーチンをアセンブルおよびリンク・エディットして、IMS SDFSRESL または IMS SDFSRESL STEPLIB 連結内のいずれかのライブラリーに入れる。
3. 従属領域を開始する。データ・キャプチャー出口ルーチンは、従属領域の初期設定時にロードされるので、出口ルーチンの新しいバージョンは、領域の開始時に有効になります。再入可能または再使用可能としてリンクされたデータ・キャプチャー出口ルーチンは、従属領域の終了時にアンロードされます。それ以外のデータ・キャプチャー出口ルーチンは、各 DL/I 呼び出し後にアンロードされます。

オンライン変更を使用した ***DEDB*** データ・キャプチャー出口ルーチンの削除*:*

オンライン変更機能を使用して、DEDB データ・キャプチャー出口ルーチンを削除することができます。

データ・キャプチャー出口ルーチンを削除するには、以下のステップを行ってください。

1. DBDGEN を実行して、古い出口ルーチンを使用する DEDB を DBD または SEGM ステートメントから除去する。
2. アプリケーション制御ブロック保守ユーティリティーを使用して、古い出口ルーチンを除外する DEDB 定義をインプリメントするためのアプリケーション制御ブロックを作成する。変更後のアプリケーション制御ブロックまたは新しいアプリケーション制御ブロックは、アクティブ IMS システムの、オフラインである ACBLIB のステージング・コピーに組み込む必要があります。
3. オンライン変更コピー・ユーティリティー DFSUOCU0 を実行して、変更済みの ACBLIB をステージング ACBLIB から、アクティブな IMS システムに対してオンラインである ACBLIB の非アクティブ (A または B) コピーに移動する。

4. 通常の /DBR コマンド・シーケンスを入力して、DEDB へのアクセスをアクティブ IMS システムから除去する。
5. ACBLIB 変更のための PREPARE 処理用のオンライン変更コマンド・シーケンスを入力して実行する。
6. ACBLIB 変更のための COMMIT/ABORT 処理用のオンライン変更コマンド・シーケンスを入力して実行する。オンライン IMS システムは、ACBLIB のアクティブ (A または B) コピーの使用から非アクティブ (A または B) コピーの使用に切り替えます。
7. 通常の /START DATABASE および /START AREA コマンドを入力して、DEDB およびそのエリアをアクティブ IMS システムでアクセス可能にする。

オンライン変更を使用した、**2** ステージ・ランダム化ルーチンによる **DEDB** ルート・アドレス可能スペースの変更**:**

オンライン変更を使用して、2 ステージ・ランダム化ルーチンによる DEDB ルート・アドレス可能スペースを変更することができます。

UOW 構造およびルート・アドレス可能割り振りは、各 DEDB 内の各エリアに固有です。ただし、1 つのエリア内のルート・アドレス可能 CI の数の変更は、DEDB 全体のルート・アンカー・ポイントの数に影響を与える可能性があります。DEDB が、データベース全体にわたってデータベース・レコードをランダムに分散する標準のランダム化ルーチンを使用している場合、ルート・アドレス可能割り振りの変更は データベース・レベル の変更になり、変更手順はそのレベルの変更として対処する必要があります。このトピックは、このような変更には適用できません。

ただし、2 ステージ・ランダム化ルーチンを DEDB に使用する場合、各エリアの UOW ルート・アドレス可能定義の変更は、*AREA* レベル の変更になります。2 ステージ・ランダム化ルーチンは、DEDB 全体内のルート・アンカー・ポイントの合計数に基づいてデータベース・レコードをすべてのエリアにわたって均等に分散しようとはしません。2 ステージ・ランダム化ルーチンは、ランダム化ルーチン名を以下のようにコーディングすることによって、DBDGEN に指定します。

```
RMNAME=(mmmmmmmm,2)
```

IMS の以前のリリースでは、DEDB DBD が、RMNAME パラメーターに複数のオペランドをもっていると、以下のエラー・メッセージを受け取っていました。

```
8, DBD130 - RMNAME OPERAND IS OMITTED OR INVALID
```

IMS の現在のリリースでも、RMNAME の第 2 オペランドとして 2 以外を指定すると、同じメッセージが表示されます。標準のランダム化ルーチンには、今でも RMNAME=(mmmmmmmm) を指定できます。

オンライン変更を使用した **DEDB AREA UOW** 構造定義の変更**:**

DEDB エリアの UOW 構造定義を変更するには、以下に示すいくつかの手順のステップを実行する必要があります。

DEDB エリアの UOW 構造定義を変更するには、以下のステップを実行します。

1. ユーザーが開発した特定のアプリケーション・プログラムまたは相手先商標製造会社 (OEM) ユーティリティーを使用して、既存のシステム定義を介してエリアをアンロードする。
2. DBDGEN、PSBGEN および ACBGEN を使用して、アプリケーション制御ブロックを生成し、DEDB 構造の変更をインプリメントする。 AREA DBDGEN マクロ・ステートメントの "UOW=(x,y)" パラメーターは、DEDB UOW 内のオーバーフロー域のために割り振られるスペースの量を定義します。 AREA DBDGEN マクロ・ステートメントの "ROOT=(nnn,mmm)" パラメーターは、独立オーバーフロー域のために割り振られるスペースの量を定義します。 変更後のアプリケーション制御ブロックまたは新しいアプリケーション制御ブロックは、アクティブ IMS システムの、オフラインである ACBLIB のステージング・コピーに組み込む必要があります。
3. オンライン変更ユーティリティー DFSUOCU0 を実行して、変更済みの ACBLIB をステージング ACBLIB から、アクティブ IMS システムに対してオンラインである ACBLIB の非アクティブ (A または B) コピーに移動する。
4. /DBR AREA コマンドを入力して、エリアへのアクセスをアクティブ IMS システムから除去する。
5. ACBLIB 変更のための PREPARE 処理用のオンライン変更コマンド・シーケンスを入力して実行する。
6. ACBLIB 変更のための COMMIT/ABORT 処理用のオンライン変更コマンド・シーケンスを入力して実行する。
7. 新しいシステム定義を使用して、エリアを削除し、定義し、初期設定する。
8. /START AREA コマンドを入力して、エリアをアクティブ IMS システムでアクセス可能にする。
9. ユーザーが開発した特定のアプリケーション・プログラムまたは OEM ユーティリティーを使用して、DEDB に関する変更したシステム定義を介して DEDB を再ロードする。

関連タスク:

886 ページの『オンライン変更と DEDB の従属および独立オーバーフロー・スペース割り振り』

DEDB レベルおよびエリア・レベルでのオンライン変更の実行

DEDB データベースに対するオンライン変更は、データベース・レベルまたはエリア・レベルで実行できます。

オンライン変更を使用した DEDB データベースの追加または削除:

オンライン変更機能を使用して、DEDB データベースを追加または削除することができます。

以下の図は、オンライン変更を使用したデータベースの追加の全体的な処理を示しています。

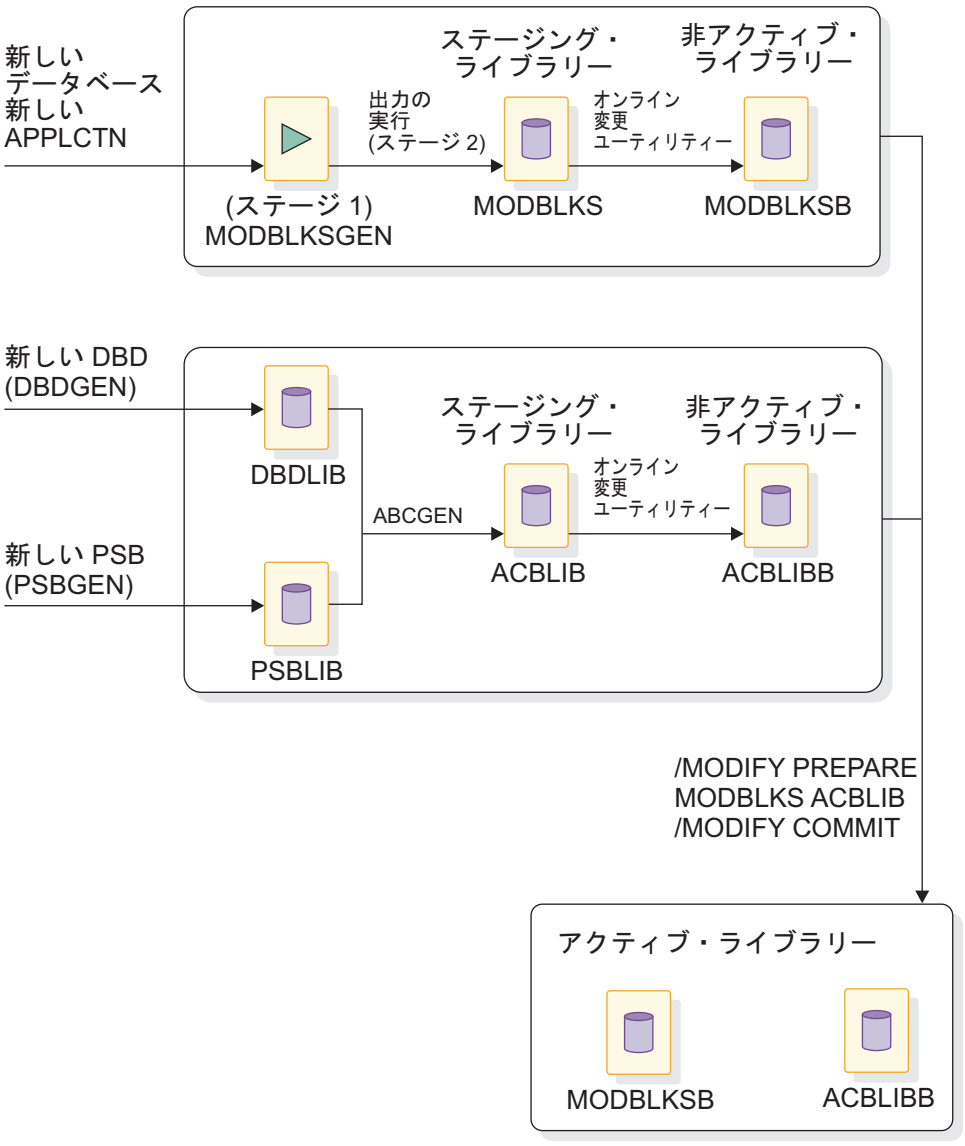


図 *295.* オンライン変更によるデータベースの追加

IMS オンライン変更機能による DEDB の追加または削除、および変更のインプリメントを行うには、以下のステップを実行する必要があります。

1. MODBLKS レベルのシステム定義 (ステージ 1 およびステージ 2) を実行して、DEDB を追加または削除する。IMS.MODBLKS データ・セットの変更は、アクティブ IMS システムの、オフラインの IMS.MODBLKS データ・セットのステージング・コピーに生成する必要があります。
2. DBDGEN、PSBGEN および ACBGEN を使用して、アプリケーション制御ブロックを生成し、それにアクセスする DEDB および PSB を追加または削除する。変更後のアプリケーション制御ブロックまたは新しいアプリケーション制御ブロックは、アクティブ IMS システムの、オフラインである IMS.ACBLIB データ・セットのステージング・コピーに生成する必要があります。
3. オンライン変更ユーティリティー DFSUOCU0 を実行して、変更済みの IMS.MODBLKS データ・セットおよび IMS.ACBLIB データ・セットをステー

ジング・ライブラリーから、アクティブ IMS システムにオンラインであるこれらのライブラリーの非アクティブ (A または B) コピーに移動する。

4. PREPARE 処理用のオンライン変更コマンド・シーケンスを入力して実行する。高速機能がインストールされていない IMS システムに DEDB を追加すると、DFS2833 エラー・メッセージが表示され、PREPARE 処理は打ち切られます。

   CI サイズがシステム・バッファー・サイズ (BSIZ=) を超えているエリアを持つ DEDB が追加されると、メッセージ DFS2832 が表示され、PREPARE 処理は打ち切られます。

   最後に、DEDB を指定せずに初期設定した IMS システムに DEDB を追加すると、メッセージ DFS2837 が表示され、PREPARE 処理は打ち切られます。

   出力スレッドは、DEDB がシステム内に現在生成されている場合のみ、高速機能初期設定時に初期設定されます。ユーザーがオンライン変更で DEDB を追加できるためには、まず DEDB を使用して IMS 初期設定する必要があります。
5. DEDB を削除するには、DEDB に対するアクセスをスケジュールしたすべての BMP 領域または DBCTL スレッドを最初に停止する必要があります。 DEDB に対するアクセスをスケジュールした全機能トランザクションは QSTOP 状態に置かれ、その結果として、MPP または IFP 従属領域は、オンライン変更をインプリメントして DEDB を削除するために停止する必要はありません。
6. DEDB を削除する場合、アクティブ状態の IMS システムからのアクセスは、`/DBR DB` コマンドによって除去する必要があります。 DEDB に対する `/DBR` コマンドの実行が前もって成功していない場合、コミットはメッセージ DFS3452 を出して失敗します。
7. COMMIT/ABORT 処理用のオンライン変更コマンド・シーケンスを実行する。
8. DEDB を新しく追加する場合、通常のユーザー・アクセスが可能になる DEDB 一般出荷可能日以前の適切な時期に、以下の追加のステップを行う。
   a. 新しい DEDB とそのエリアおよびエリア・データ・セットを DBRC データ・セットおよび RECON データ・セットに対して定義する、通常の手順を行う。
   b. 新しい DEDB に属するすべてのエリア・データ・セットを定義し、初期設定する。
   c. アクティブ IMS システムへの DEDB およびそのエリアの割り振りを可能にするために必要な動的割り振り定義を組み込む手順を行う。あるいは、DEDB およびそのエリアを DBRC に登録すれば、DBRC は IMS の初期設定時に、それらを動的に割り振ります。
   d. `/START DATABASE` および `/START AREA` コマンドを入力して、DEDB およびそのエリアをアクティブ IMS システムでアクセス可能にする。
   e. 必要なアプリケーション・ロード・プログラムを実行する。

オンライン変更を使用した **DEDB** セグメントの追加または削除**:**

オンライン変更機能を使用して、DEDB セグメントを追加または削除することができます。

セグメント・タイプの追加または削除、あるいはセグメント・フォーマットの変更は、DEDB の構造に影響を与え、データベース・レベルの変更になります。セグメ

ント・タイプ (DEDB 順次従属セグメント・タイプを含む) の追加または削除は、階層構造およびこの構造をインプリメントするためのセグメント接頭部のレイアウトに影響を与えます。同様に、各セグメント・フォーマットの変更は、データベース全体の構造および DEDB の各エリア内のスペース割り振りを変更します。

既存の DEDB に対して構造的な変更を行うには、下記の手順のステップを行ってください。

1. ユーザーが開発した特定のアプリケーション・プログラムまたは OEM ユーティリティーを使用し、既存のシステム定義を介して DEDB をアンロードする。
2. DBDGEN、PSBGEN および ACBGEN を使用して、アプリケーション制御ブロックを生成し、DEDB 構造の変更をインプリメントする。変更後のアプリケーション制御ブロックまたは新しいアプリケーション制御ブロックは、アクティブ IMS システムの、オフラインでの ACBLIB のステージング・コピーに組み込む必要があります。
3. オンライン変更コピー・ユーティリティー (DFSUOCU0) を実行して、変更済みの ACBLIB をステージング ACBLIB から、アクティブな IMS システムに対してオンラインである ACBLIB の非アクティブ (A または B) コピーに移動する。
4. 通常の /DBR コマンド・シーケンスを入力して、DEDB へのアクセスをアクティブ IMS システムから除去する。このコマンドは、/MODIFY COMMIT の前であればいつでも出すことができます。
5. ACBLIB 変更のための PREPARE 処理用のオンライン変更コマンド・シーケンスを入力して実行する。
6. ACBLIB 変更のための COMMIT/ABORT 処理用のオンライン変更コマンド・シーケンスを入力して実行する。
7. 新しいシステム定義を使用して、DEDB に属するすべてのエリア・データ・セットを削除し、定義し、初期設定する。
8. 通常の /START DATABASE および /START AREA コマンドを入力して、DEDB およびそのエリアをアクティブ IMS システムでアクセス可能にする。
9. ユーザーが開発した特定のアプリケーション・プログラムまたは OEM ユーティリティーを使用して、DEDB に関する変更したシステム定義を介して DEDB を再ロードする。

オンライン変更を使用した **DEDB** エリアの追加または削除**:**

エリアの追加または削除は、DEDB 内のすべてのデータベース・レコードの位置に影響を与える可能性があります。エリアの数を変更すると、DEDB 内のルート・アンカー・ポイント (RAP) の数が変更されます。

DEDB ランダム化ルーチンは、最初はエリアに基づいて、次にエリア内のルート・アンカー・ポイント (RAP) に基づいて、DEDB 全体にわたってデータベース・レコードをランダムに分散させます。

1 つ以上のエリアを DEDB に追加または削除することは、セグメント・タイプの追加のような、構造的な変更を行うことになります。 DEDB 内に定義済みのエリアの数を変更するには、 884 ページの『オンライン変更を使用した DEDB セグメントの追加または削除』で説明したステップを行ってください。エリアを新しく追加する場合、アクティブ IMS システムが新しいエリアにアクセスできるようにするに

は、エリアおよびエリア・データ・セットに必要な DBRC 定義を処理し、動的割り振りブロックを作成する必要があります。

オンライン変更を使用した **DEDB** ルート・アドレス可能スペース割り振りの変更**:**

DEDB レコードをランダムに分散するか、または標準のランダム化ルーチンを使用して DEDB レコードを均等に分散するかに応じて、異なる影響が生じます。いずれの場合でも、DEDB 全体または 1 つの DEDB エリアのみにわたって DEDB レコードを分散することができます。

すべての **DEDB** エリアにわたるデータベース・レコードのランダム分散

ルート・アドレス可能部分に対して定義される DEDB 制御インターバルの数に影響を与える DEDB 作業単位 (UOW) 構造に対する変更は、DEDB 全体のルート・アンカー・ポイントの数に影響を与えます。このタイプの変更は、DEDB 内のすべてのデータベース・レコードの位置に影響を与える可能性があります。

標準 **DEDB** ランダム化ルーチン

標準の DEDB ランダム化ルーチンは、すべてのエリアにわたって、また選択したエリア内で、データベース・レコードを均等に分散しようとします。このようなランダム化ルーチンは、DEDB 全体内のルート・アンカー・ポイントの合計数に基づいて、レコードの位置を決めます。

ルート・アドレス可能域に対して定義される CI の数を変更する UOW 構造の変更は、標準の DEDB ランダム化ルーチンを使用している場合、データベース・レベルの変更になります。このタイプの変更は、オンライン変更手順については、DEDB の構造的な変更と同様に取り扱う必要があります。

オンライン変更と **DEDB** の従属および独立オーバーフロー・スペース割り振り**:**

高速機能では、DEDB エリアの独立オーバーフロー・スペース割り振りの拡張を限定的にサポートします。さらに、DEDB オンライン変更によって、各 UOW (従属オーバーフロー) 内とエリアのルート・アドレス可能部分 (独立オーバーフロー) の外側の両方でオーバーフロー・スペース割り振りに対する変更が可能になります。

従属オーバーフロー変更と独立オーバーフロー変更の両方ともエリア・レベルの変更と見なされます。ただし、このような変更はルート・アドレス可能部分に対して定義されている CI の数を変更してはなりません。ルート・アドレス可能 CI の数を変更すると、ルート・アンカー・ポイントの数が変更され、DEDB ランダム化ルーチンがデータベース・レコードの位置を決める場合に影響を与えます。

DEDB エリアのオーバーフロー割り振りの変更には、ルート・アドレス可能域の変更のために定義した手順のステップと同じものが必要です。

関連タスク:

881 ページの『オンライン変更を使用した DEDB AREA UOW 構造定義の変更』

オンライン変更を使用した **DEDB CI** サイズの変更**:**

DEDB エリア制御インターバル・サイズを変更するために、DEDB オンライン変更を使用することができます。ただし、CI サイズの変更は、エリアのルート・アドレ

ス可能部分に割り振られている CI の数を変更してはなりません。この変更を行うと、DEDB ランダム化ルーチンが DEDB 内のデータベース・レコードの位置を決める場合に影響が及びます。

エリアを構成するデータ・セットの CI サイズを定義するには、DBDGEN の AREA ステートメントの SIZE= パラメーターを使用します。

DEDB 独立オーバーフロー・オンラインの拡張

IMS がオンラインであっても、DEDB エリアの独立オーバーフロー (IOVF) 部分を拡張することができます。

この手順が完了した後に初めてそのエリアがオープンされるときに、メッセージが出て高速機能がそのエリアへの変更を認識して受け入れることを検証し、通常のオープン処理が完了します。 DEDB オンライン変更機能を使用すれば、DEDB の IOVF 部分を変更することもできます。

この手順では IOVF のサイズを小さくすることはできません。しかし、そのエリアに割り振られたスペースの総量に応じて、順次従属部のサイズを大きくしたり小さくしたりすることができます。この手順の各ステップは、エリアの再編成も行います。

DEDB の IOVF 部のサイズをオンラインのまま大きくするには、以下のようにします。

1. DBDGEN ユーティリティーを実行して、更新された DBD を取得します。AREA ステートメントの ROOT= キーワードの *number2* オペランドと *overflow2* オペランドを更新します。AREA ステートメント (システム・ユーティリティー)を参照してください。

   他の制御ステートメントはすべて、既存の DBD のステートメントと同じものにします。他の制御ステートメントを変更すると、データが損傷し予測できない結果が生じることがあります。
2. 更新された DBD を用いて、ACBGEN ユーティリティーを実行します。新しい ROOT= パラメーターを使って新規の完全な ACBLIB を作成するため PSB=ALL を実行してください。その出力は、制限エリアで現在使用されているデータ・セットとは異なるデータ・セットになるはずです。 ROOT= の変更を除き、新しい ACBLIB は古い ACBLIB と同一です。ユーザーは、ステージング ACBLIB を使用することができます。しかし、オンライン変更機能を切り替えることはできません。
3. エリアの状態が良好であることを確認します。このエリアには、未確定事項やリカバリーが必要な状態であってはなりません。また、このエリアには、エラー・キュー・エレメント (EQE) のないコピーが少なくとも 1 つは必要です (1 つのエリア・データ・セット)。 `/DIS AREA` コマンドを使用して、EQE と状態を表示します。 `/DIS CCTL INDOUBT` コマンドを使用して、未確定のスレッドすべてを表示します。データが失われたり破壊されたりしないように、次のステップに進む前に、欠陥となる可能性があるものを排除します。
4. SDEP スキャン・ユーティリティーおよび削除ユーティリティーを用いて、SDEP を処理します。 このステップが必要なのは、IOVF 拡張手順にはエリアのアンロードとロードが必要だからです。 SDEP を処理できないアンロード・

ユーティリティーおよび再ロード・ユーティリティーもあります。 SDEP を処理できないアンロード/ロード・ユーティリティーは、これを時間順序ではなくルート・セグメントの順序で再ロードする可能性があり、これがその後の SDEP スキャンおよび削除操作の障害となることがあります。

5. エリアに複数のコピー (MADS) がある場合は、/STOP ADS コマンドを使用して 1 つを除いたすべてのコピーを停止します。残りのコピーに EQE がなく、リカバリーが必要な状態でもないことを確認します。 IOVF が拡張された後にエリアがオンラインされたとき、DBRC に正確な情報があるようにするために、複数の ADS を停止しなければなりません。
6. エリアに対して、/DBR コマンドまたは /STO AREA コマンドを出します。
7. エリアのイメージ・コピーをとります。
8. エリアを DBRC に登録する場合には、このエリアのために要リカバリー・フラグを立てます。 DEDB 初期設定ユーティリティーには、このフラグが必要です。このフラグは、CHANGE.DBDS RECOV コマンドを使用して設定できます。
9. このエリアをアンロードします。
10. IDCAMS ユーティリティーを実行して、データ・セットを削除し、定義し直します。 この定義手順でエリアに割り振るスペースの量は、増加した IOVF のサイズを反映している必要があります。エリアの SDEP CI の数は、エリアに割り振られたスペースの総量と他の部分で使用されている量との差を示すので、変化することがあります。他の部分とは、ルート・アドレス可能部、IOVF、再編成 UOW、および 2 つの制御 CI です。

    関連資料: IDCAMS の削除機能と定義機能については、*z/OS DFSMS* カタログのためのアクセス方式サービス・プログラム を参照してください。
11. 新しい ACBLIB を用いて、新しいエリアを対象として高速機能初期設定ユーティリティーを実行します。
12. /START AREA コマンドを出して、そのエリアをオンラインにします。
13. エリアを再ロードします。

    推奨事項: エリアはバッチで再ロードしてください。BMP を用いてエリアを再ロードすると、この BMP は、メッセージ DFS3709A および理由コード 5 が出されて失敗することがあります。この失敗が起こった場合は、CHANGE.DBDS コマンドを出して ICOFF を設定し、BMP を再始動してください。
14. 再ロードの後にエリアのイメージ・コピーをとります。

このエリアに次にアクセスすると、メッセージ DFS3703I が出ます。このメッセージは、オープン処理中に矛盾が発見されたことを警告しています。ただし、この矛盾はユーザーが IOVF のサイズを増やすために認められた手順を用いたことを IMS に示しているので、オープン処理は継続されます。 IMS がオンラインである限り、その後のオープン時に DFS3703I は出ません。 DFS3703I は、IOVF が拡張された後に共用サブシステム上で最初にこのエリアをオープンしたときにも、このサブシステムによって出されます。

緊急時再始動時または拡張回復機能 (XRF) テークオーバー時には、更新されたエリア情報がログから取り出されます。したがって、DFS3703I は発行されません。

その後オンライン・システムの正常再始動を行うときに、新しい ACBLIB を使用してください。この新しい ACBLIB は ROOT= キーワードに加えられた変更だけ を反映していることを確認してください。ユーザーが加える他の変更が、エリアに損傷を与えることがあります。新しい ACBLIB を使用しない場合は、オープン論理によって古い ACBLIB の情報とエリア・データ・セットの情報の間に矛盾が生じますが、このような矛盾が生じるたびにメッセージ DFS3703I が出されます。

注: オンライン変更機能を使用して、変更した ROOT= パラメーターで ACBLIB を更新することはできません。

関連概念:

522 ページの『DEDB エリアの設計の指針』

DBRC 管理 (システム管理)

関連資料:

アプリケーション制御ブロック保守ユーティリティー (システム・ユーティリティー)

データベース記述 (DBD) 生成ユーティリティー (システム・ユーティリティー)

DEDB 順次従属スキャン・ユーティリティー (DBFUMSC0) (データベース・ユーティリティー)

高速処理データベース順次従属スキャン・ユーティリティー出口ルーチン (DBFUMSE1) (出口ルーチン)

HALDB データベースの変更

既存の HALDB データベースは他の全機能データベースを変更する場合と同じように変更することができます。ただし、HALDB に対しては、実行方法が異なる変更作業もあるし、 HALDB に固有の変更作業もあります。

このトピックで説明する変更作業を実行する前に、HALDB の多くの固有な概念と特性に精通しておくことも必要です。

関連概念:

934 ページの『HDAM および HIDAM データベースから HALDB への変換』

584 ページの『HALDB 設計のインプリメント』

733 ページの『HALDB データベースのオフライン再編成のオプション』

HALDB データベースの変更の概説

HALDB データベースに実施できる変更の多くは、他の IMS 全機能データベース・タイプに実施できるものと同じです。ただし、HALDB データベース内の区画全体にわたってレコードの分散を調整するなどの、HALDB データベースに固有な変更もあります。

HALDB データベースへのほとんどの変更を、HALDB マスター・データベースを定義するデータベース定義 (DBD) ステートメントまたは区画定義が保管される DBRC RECON データ・セットに指定します。変更をどこに指定するかは、実施す

る変更のタイプによって異なります。例えば、レコードの階層構造を変更するには、DBDGEN プロセスを使って DBD に変更を指定します。しかし、区画に保管されているレコードの範囲を変更するには、区画定義ユーティリティーまたはバッチ DBRC コマンドを使用して DBRC RECON データ・セットに変更を指定します。

HALDB データベースに対する変更の有効範囲

HALDB データベースを変更するときは、変更の有効範囲を意識する必要があります。実施できる変更には、1 つの区画だけに影響を与えるものもあるし、区画のサブセットに影響を与えるものもあります。 HALDB データベースに対するそれ以外の変更は、データベース全体およびそのすべての区画に影響を与えます。

以下の表に、代表的な HALDB データベースの変更、変更を指定する場所、および変更の有効範囲を示します。

表 *83. HALDB* データベースの変更例、変更を指定する場所、および変更の有効範囲

変更するもの	変更を指定する場所	変更の有効範囲	注
フィールド定義、追加または削除	DBD ステートメント	HALDB データベース全体とその区画	HALDB データベースのアンロードは、フィールド・オフセットまたはセグメントの長さを変更する場合にのみ必要
データ・キャプチャー出口ルーチン	DBD ステートメント	HALDB データベース全体とその区画	HALDB データベースのアンロードは不要
セグメント・タイプ、追加または削除	DBD ステートメント	HALDB データベース全体とその区画	HALDB データベースのアンロードが必要
ポインター・オプション	DBD ステートメント	HALDB データベース全体とその区画	HALDB データベースのアンロードが必要
セグメント編集/圧縮出口ルーチン	DBD ステートメント	HALDB データベース全体とその区画	HALDB データベースのアンロードが必要
HALDB 区画選択出口ルーチンの選択基準	DBD ステートメントおよび RECON データ・セット	区画選択出口ルーチンによるレコード分散の変更によって影響を受ける区画のみ	レコード分散の変更によって影響を受ける区画のみアンロードが必要
Data set name prefix	RECON データ・セット	指定された区画のみ	影響を受ける区画だけが無許可、他のすべての区画へのアクセスは影響を受けない
ランダム化モジュールまたはランダム化パラメーター	DBD ステートメント、RECON データ・セット、または両方	DBD 内で実施される場合、データベース内の全区画 RECON データ・セット内で実施される場合、指定された区画のみ	RECON データ・セット内の指定への変更の場合、影響を受ける区画だけが無許可、他のすべての区画へのアクセスは影響を受けない

表 83. *HALDB* データベースの変更例、変更を指定する場所、および変更の有効範囲 (続き)

変更するもの	変更を指定する場所	変更の有効範囲	注
区画のハイ・キー	RECON データ・セット	新規または変更されたハイ・キーによって導入されたレコード分散の変更の影響を受ける全区画	レコード分散の変更によって影響を受ける区画のみアンロードが必要

HALDB データベース内の全区画に影響を与える変更**:**

HALDB 区画の一部の特性は、HALDB データベース内のすべての区画で共用されます。

これらの特性を変更するためには、その前にタイプ 1 コマンド `/DBRECOVERY DB *HALDB_master_name*` またはタイプ 2 コマンド `UPDATE DB NAME(*HALDB_master_name*) STOP(ACCESS)` を実行し、その HALDB データベース内のすべての区画をオフラインにしなければなりません。

HALDB 区画の以下の特性は、HALDB データベース内の全区画をオフラインにした後でのみ変更することができます。

- DBD 定義
- HALDB 区画選択出口ルーチン
- Share level
- リカバリー不能属性状況
- RSR GSG 名またはトラッキング・レベル
- PHDAM データベースおよび PHIDAM データベースの OSAM データ・セットの最大サイズ

注**:** 上記の変更の一部は、実施の際に HALDB データベース内の区画をオフラインにするだけでなく、全区画をアンロードし、適切な変更を加えて区画を再ロードし、区画へのアクセスを復元する必要があります。

例えば HALDB データベースにある既存の HALDB 区画選択出口ルーチンを、新規アルゴリズムに基づいて区画を選択する HALDB 区画選択出口ルーチンで置き換える必要があるものとします。区画選択出口ルーチンはデータベース内のすべての区画内のレコードの配置に影響を与える可能性があるので、このような変更を行うには、HALDB データベース全体をオフラインにする必要があります。データベースをオフラインにした後に、データベースをアンロードし、出口ルーチンに変更を加え、データベースを再ロードし、データベースへのアクセスを復元する必要があります。

単一区画に適用できる変更**:**

HALDB の一部の特性は、データベース内の各区画に固有のものです。これらの特性を変更するために HALDB データベース全体をオフラインにする必要はありません。オフラインにする必要があるのは、変更を加える区画だけです。

区画を変更する前に、その区画に対してタイプ 1 コマンド /DBRECOVERY DB *partition_name* またはタイプ 2 コマンド UPDATE DB NAME(*partition_name*) STOP(ACCESS) を実行することにより、その区画へのアクセスを停止します。

例えば、HALDB 区画の以下の特性を変更するには、その単一の区画のみをオフラインにします。

- Data set name prefix
- ランダム化モジュール名
- ルート・アンカー・ポイントの数 (RAP)
- BYTES パラメーター
- OSAM ブロック・サイズ
- VSAM CI サイズ

注: 上記の変更を行うには、区画をオフラインにするだけでなく、区画をアンロードし、適切な変更を加えて区画を再ロードし、区画へのアクセスを復元する必要があります。

HALDB データベース内のレコード分散と区画境界

HALDB データベース内のレコードの分散は、区画が保持するレコードの範囲を再定義することにより調整することができます。

区画のレコードの範囲を再定義するには、区画間の境界を変更します。区画が保持するレコードの数は時間が経つと変る可能性があるので、HALDB データベース内のレコードの分散は、一般的には、HALDB データベース内の 1 つ以上の区画が大きくなりすぎたか、小さくなりすぎたときに調整します。

区画の境界をどのように変更するかは、HALDB データベースが区画の選択をどのように行うかによって異なります。HALDB データベースは区画の選択を、区画のハイ・キーを使用するか (レコードのルート・キーに基づきます)、区画選択出口ルーチンを使用して (ユーザー定義の区画選択ストリングに基づきます) 行います。

HALDB データベースが区画をハイ・キーに基づいて選択する場合、区画内のレコードの範囲は、区画のハイ・キーを変更することにより変更します。

HALDB データベースが区画選択出口ルーチンを使用して区画を選択する場合、区画内のレコードの範囲は、この出口ルーチン自体を変更することにより変更します。このような変更は一般にインストール済み環境によって異なるため、このトピックでは取り上げていません。

区画境界の変更は、1 つ以上の区画に影響する可能性があります。 1 つの区画の変更によって他の区画との間でレコードの移動が起こる場合は、変更がそれらの区画にも影響を及ぼします。区画選択にハイ・キーを使用する場合は、IMS が自動的に、境界変更によって影響を受ける区画に初期設定が必要なことを示すフラグを設定します。HALDB 区画選択出口ルーチンを使用する場合は、ユーザーが、境界変更によって影響を受ける区画に初期設定が必要なことを示すフラグを設定する必要があります。いずれの場合も、区画境界を変更する前に、変更によって影響を受けることになるすべての区画に対して /DBR コマンドを実行しなければなりません。

HALDB データベースに区画を追加するか、使用不可にするか、または削除すると、レコードの分散も影響を受ける可能性があります。

関連概念:

906 ページの『HALDB 区画の使用不可および使用可能の設定』

関連タスク:

901 ページの『既存の HALDB データベースへの区画の追加』

909 ページの『既存の HALDB データベースからの区画の削除』

レコード分散とハイ・キーの区分化:

キー範囲の区分化を使用する場合、1 つ以上の区画のハイ・キーを変更することにより、HALDB 区画全体にわたるレコードの分散を変更できます。区画のハイ・キーは、区画が保持することができるレコードの最大ルート・キーを指定します。

区画のハイ・キーは、そのハイ・キーを所有する区画と、次に大きいハイ・キーを所有する区画 (そのような区画が存在する場合) との間の境界も定義します。 2 つの区画間の境界を定義するハイ・キーを変更すると、両方の区画がその変更の影響を受けます。例えば、区画のハイ・キーを小さくすると、通常、ハイ・キーを持つ区画内のレコードの数は減少し、次に大きいハイ・キーを持つ区画内のレコードの数は増加します。

HALDB データベース内のレコードの分散を調整する前に、以下を行ってください。

- HALDB データベース内の区画全体にわたってレコードの現在の分散を分析します。

  既存の HALDB データベース内のレコードの正確な報告書を入手するには、IBM IMS HALDB Toolkit for z/OS を使用してください。これは、個別にライセンス交付を受けるソフトウェア・ツールです。このツールについて詳しくは、IBM Knowledge Center (www.ibm.com/support/knowledgecenter) で IMS ツールに関する情報を参照してください。
- HALDB 再編成番号検査機能を使用可能にします。これを使用可能にすると、区画の再編成番号が区画間のレコードの移動によって後退しなくなります。

区画のハイ・キーは、区画定義ユーティリティーを使用するか、バッチ DBRC コマンドを使用して変更することができます。この変更を行うには、一般的に、区画定義を変更する前に、影響を受ける区画をアンロードし、変更が終了したら、その区画を初期設定して、再ロードする必要があります。影響を受ける区画とは、ユーザーが区画の定義を直接に変更しなかった場合でも、キーの範囲が大きくなる、または小さくなる区画です。

区画のハイ・キーを変更するのは、通常は、HALDB データベース内のデータ量が区画間で均衡が取れなくなったときです。区画のハイ・キーを変更すると、区画の境界が変更され、一部のレコードが 1 つの区画から他の区画へ移動します。

以下の図に、区画 B のハイ・キーを 400 から 500 に変更する例を示します。この変更の結果、401 から 500 のキーを持つレコードが区画 C から区画 B に移動します。

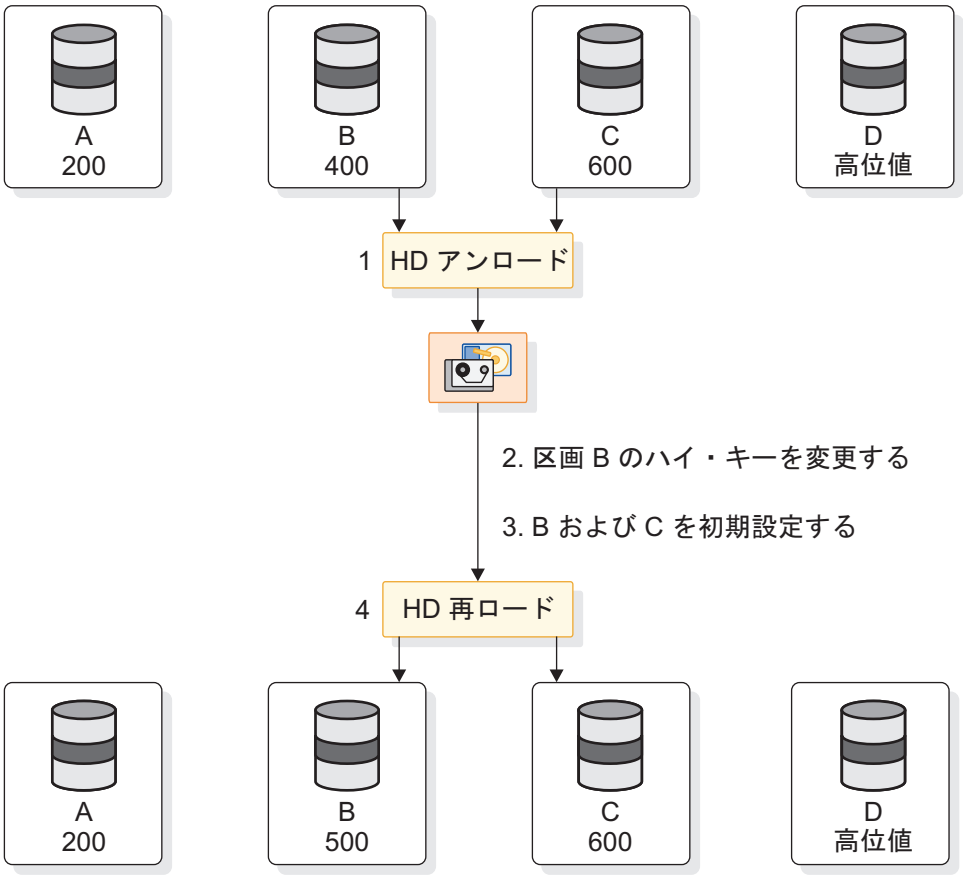


図 *296.* キー範囲の区分化によるハイ・キーの変更

したがって、前の図の区画 B のハイ・キーの変更は、区画 B と区画 C の両方に影響します。区画 B の定義が変更されると、IMS は、区画 B と区画 C の両方に、初期設定が必要であることを示すフラグを設定します。区画 A および D は、この変更による影響を受けません。

オンライン IMS システムは、以下のイベントのいずれかが発生するまで、前の図の区画 B および C の区画定義の変更を認識しないという点に注意してください。

- /START DB *HALDB_Master* OPEN コマンドの実行。
- UPDATE DB NAME(*HALDB_Master*) START(ACCESS) OPTION(OPEN) コマンドの実行。
- DL/I 呼び出しに起因する、区画 B または C のための DBRC への許可呼び出しの実行。最初の DL/I 呼び出しによって HALDB 区画選択が実行され、区画 B または C のいずれかが選択され、許可されます。

関連概念:

196 ページの『HALDB 区画再編成番号』

関連タスク:

900 ページの『区画のハイ・キーの変更』

レコード分散と区画選択出口ルーチン:

区画選択出口ルーチンを使用する場合、区画全体にわたるレコードの分散は、その出口ルーチンによって管理されるのであって、IMS によって管理されるのではありません。

区画全体にわたってレコードの分散を調整するには、この出口ルーチンを変更する必要があります。

IMS は、出口ルーチン内で行われるレコードの分散に対する変更を検出することはできません。また、IMS は、どの区画がこれらの変更の影響を受けるかを認識しません。したがって、ユーザーが行う変更によってどの区画が影響を受けるかを理解して、それに応じて区画を初期設定します。

変更の結果、レコードが 1 つの区画から他の区画に移動する場合、レコードの移動の影響を受けるすべての区画に対して、ユーザーは HD 再編成アンロード・ユーティリティー (DFSURGU0) の実行、区画の初期設定が必要なことを示す (PINIT) フラグの設定、初期設定、および HD 再編成再ロード・ユーティリティー (DFSURGL0) を行う必要があります。

また、使用している区画選択出口ルーチンに、構造の初期設定呼び出しおよび変更呼び出しを処理するコードが含まれていることを確認してください。これらの呼び出しは、区画が追加または変更された後でオンライン区画構造を更新するときに IMS が発行します。IMS が提供するサンプルの区画選択出口ルーチン (DFSPSE00) には、これらの呼び出しをサポートするコードが含まれています。

関連タスク:

920 ページの『HALDB 区画選択出口ルーチンの追加または変更』

関連資料:

HALDB 区画選択出口ルーチン (DFSPSE00) (出口ルーチン)

区画定義制御ブロックと RECON データ・セット内の区画定義

オンライン IMS システムは、ユーザーが定義する各 HALDB 区画に対して内部の区画定義制御ブロックを作成します。

ユーザーが RECON データ・セット内の HALDB 区画定義を変更するたびに、IMS は、その変更の影響を受けるすべての区画の区画定義制御ブロックを再作成しなければなりません。これを行うまでは、変更の影響を受ける区画は使用できません。

IMS がいつ区画定義制御ブロックを再作成する必要があるかを検出するために、IMS は、変更バージョン番号 を使って HALDB データベースへの変更を追跡します。ユーザーが RECON データ・セットに記録されている区画定義を変更するたびに、DBRC は、HALDB マスター・データベース用の変更バージョン番号を 1 だけ大きくします。オンライン IMS システムが HALDB データベース用の変更バージョン番号の増加を検出すると、IMS は、変更の影響を受ける各区画の区画定義制御ブロックを再作成します。

HALDB 区画定義の変更、または区画定義制御ブロックの再作成には、オンライン変更は使用しません。IMS は HALDB 変更バージョン番号の増加を認識して、オンライン IMS システムに新規の定義を動的に反映させます。

XRF を使用している場合には、代替 IMS システムが動的変更を検出し、代替システム内の定義を自動的に更新します。

関連概念:

195 ページの『HALDB 区画の名前と番号』

区画定義制御ブロックの再作成を起動するイベント:

IMS は、HALDB 変更バージョン番号を検査して、区画定義の変更を検出します。

IMS は、以下の場合に、必要に応じて内部の区画定義制御ブロックを再作成します。

- 区画の使用が許可されている場合。既存の区画またはまだ許可されていない区画での変更が検出されます。このような場合は、一般的には、区画がオフラインにされ、RECON データ・セット内の定義に変更が加えられ、その区画が再び使用可能になったときに行われます。更新された区画を最初に使用する時点で、区画定義制御ブロックの再作成がトリガーされます。
- 区画選択または区画選択出口ルーチンによって無効キーが検出された場合など、最後の区画のハイ・キーを超えて新規区画が追加され、既存のすべての区画がすでに許可されている場合に起こります。この場合は、IMS 区画選択または区画選択出口ルーチンによって新しい区画が検出され、IMS が自動的に区画定義制御ブロックを再作成します。
- /START DB *HALDB_Master* OPEN または UPDATE DB NAME (*HALDB_Master*) OPTION(OPEN) コマンドが実行された場合。例えば、最後の区画のハイ・キーを超えて新規区画が追加され、既存のすべての区画がすでに許可されている場合、これらのコマンドは区画定義制御ブロックの再作成を開始します。

関連概念:

『区画定義制御ブロックに関する追加の考慮事項』

区画定義制御ブロックに関する追加の考慮事項:

HALDB 区画定義を変更する場合は、オンライン区画定義制御ブロックが再作成されるまでに発生する可能性があるいくつかの状況に注意する必要があります。

以下の点を考慮してください。

- HALDB 区画選択出口ルーチンを使用する場合、/DBRECOVERY コマンドまたは UPDATE STOP(ACCESS) コマンドを実行し、次に、区画定義を変更した後、/START コマンドまたは UPDATE START(ACCESS) コマンドを実行する必要があります。HALDB 区画選択出口ルーチンが HALDB 区画を選択しても、IMS は内部の区画定義制御ブロックの再作成が完了するまで HALDB 区画の境界を認識しません。
- HALDB 区画選択出口ルーチンを使用していて、IMS から構造の変更が通知される場合は、出口ルーチンで現行区画構造に基づいた正しい区画を選択する必要があります。

- 区画構造の定義が変更された後に、/START DB *partition_name* OPEN コマンドが失敗する場合があります。これは、対応する区画定義制御ブロックの再作成が必要となるためです。制御ブロックの再作成を呼び出すためには、その区画を使用するアプリケーション・プログラムを実行するか、タイプ 1 のコマンド /START DB *HALDB_Master* OPEN を実行しなければなりません。
- 新規に追加された区画は、対応する区画定義制御ブロックの再作成が呼び出しされて新規制御ブロックが作成されるまで、オンライン IMS システムによって認識されません。
- 区画選択出口ルーチンを使用している場合、そのルーチンに構造の初期設定呼び出しおよび変更呼び出しを処理するコードが含まれていることを確認してください。これらの呼び出しは、区画が追加または変更された後で区画定義制御ブロックを更新するときに IMS が発行します。IMS と一緒に配布されるサンプルの区画選択出口ルーチン (DFSPSE00) には、これらの呼び出しをサポートするコードが含まれています。

関連概念:

896 ページの『区画定義制御ブロックの再作成を起動するイベント』

IMS が区画 ID 番号を割り当てる方法

各区画が定義されるとき、IMS はその区画に区画 ID 番号を割り当てます。

新しい区画の区画 ID は、IMS に定義された最後の区画の区画 ID 番号を増分することにより生成されます。したがって、HALDB データベース内の新しい区画は、それぞれ、大きくなった新しい番号を割り当てられます。割り当てられた最後の区画 ID が RECON データ・セットに保管されます。前に割り当てられた区画 ID 番号は再利用されません。

区画の定義を削除すると、その区画の区画 ID は永久に失われます。その区画を再定義して区画を復元しようとすると、再定義された区画は、大きくなった新しい区画 ID を与えられます。

区画を削除して再定義するときに新しい区画 ID が割り当てられることは、そのような変更をバックアウトできる点で重要な意味を持ちます。区画 ID は区画データ・セットの中に物理的に保管されます。したがって、ある区画用に作成されたデータ・セットは、その区画が削除されて再定義されると、他の区画で、または同じ区画で使用することはできません。同様に、イメージ・コピー内の区画 ID が、イメージ・コピーを適用しようとしている区画の区画 ID と一致しない場合、イメージ・コピーは使用できません。

推奨事項**:** 区画を削除する前に、その区画を使用不可にしてください。使用不可にされた区画はアクティブな使用からは除外されますが、再度使用可能にすることができます。再度使用可能にされた区画は、元の区画 ID を保持します。使用不可にした区画は、アクティブな HALDB データベースからの除去が完全にテストされた後で削除してください。

以下の 3 つの図は、3 つの区画を持つ HALDB データベースでの区画 ID の割り当てを示しています。

以下の図は、A、B、C という 3 つの区画を持つ HALDB データベースを示しています。3 つの区画の区画 ID はそれぞれ 001、002、003 です。

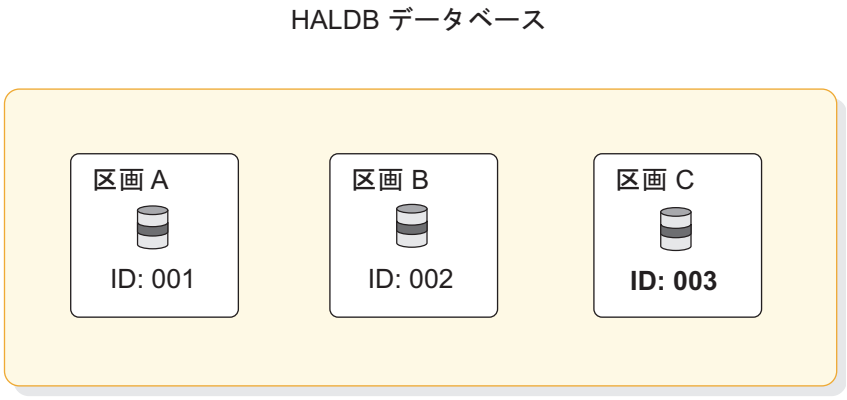


図 *297.* 区画を削除して再定義する前の区画 *ID*

以下の図は、区画 C が削除された後の同じ HALDB データベースを示しています。区画 C を削除するには、区画 B と C のアンロード、区画 C の削除、区画 B の初期設定、区画 B と C にあったレコードの区画 B への再ロードが必要です。これにより、データベースの区画が 3 つから 2 つに変更されます。

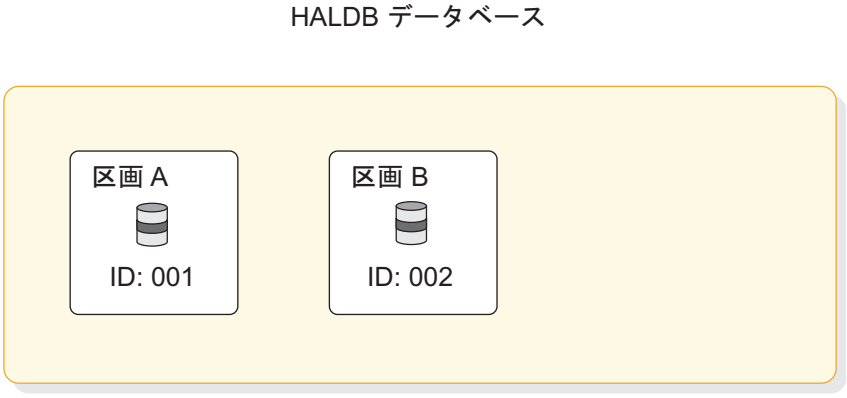


図 *298.* 区画を削除して再定義した後の区画 *ID*

以下の図は、区画 C が削除されたときに元の区画 ID が永久に失われたために、区画 C を再定義してもリカバリーできないことを示しています。区画 C が再定義されるとき、IMS は区画 ID 004 を割り当てますが、これは、前に割り当てられた区画 ID 番号を増分することにより得られたものです。区画 C の区画 ID は 004 になったので、区画 C が削除される前に取られた区画 C のデータ・セットのイメージ・コピーは、使用できません。これらのイメージ・コピーは、区画 ID 003 を持っています。

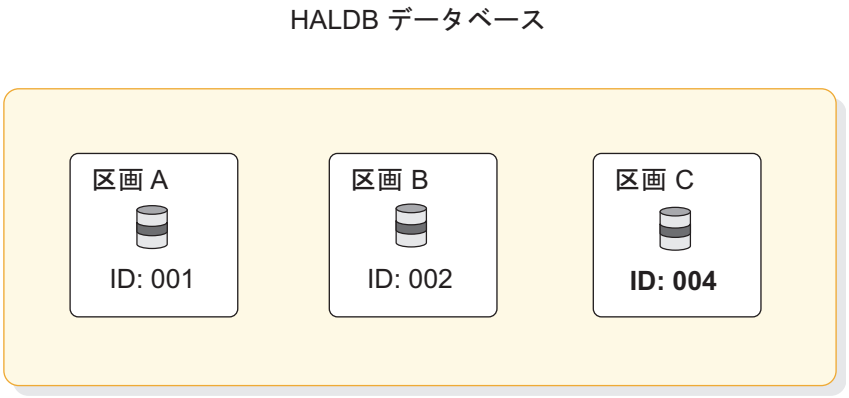


図 *299.* 区画を削除して再定義した後の区画 *ID*

関連概念:

906 ページの『HALDB 区画の使用不可および使用可能の設定』

関連タスク:

909 ページの『既存の HALDB データベースからの区画の削除』

914 ページの『削除された HALDB 区画の復元』

HALDB 副次索引および論理関係ポインターの自動更新

論理関係または副次索引を持つアプリケーション・プログラムが HALDB データベースを更新する場合には、IMS は、HALDB 自己回復ポインター処理を使用してポインターを自動的に訂正します。

論理関係で使用されるポインターをユーザーが更新する必要はないし、副次索引をユーザーが再作成する必要もありません。

例えば、副次索引定義の変更を必要としない変更を索引付きデータベースに加えた場合、副次索引をアンロード、再ロード、または再作成する必要はありません。ポインターが使用されるときに、HALDB 自己回復ポインター処理によってポインターが個別に調整されます。索引付きデータベースのための再ロード・プロセスを実行すると、ILDS が更新されます。移動したターゲット・セグメントを検索するために副次索引ポインターを確実に使用できるようにするために、ほかに必要な作業はありません。

また、論理的に関連付けられたデータベースの区画を変更しても、他の論理的に関連付けられたデータベースに変更を加える必要はありません。データベース内の区画を追加、削除、または変更したとき、論理的に関連付けられているデータベースをアンロードして、再ロードする必要はありません。ポインターが使用されるときに、HALDB の自己回復ポインター処理によってポインターが個別に調整されます。

関連概念:

764 ページの『HALDB 自己回復ポインター処理』

オンライン変更と HALDB データベース

HALDB データベースに対する DBD 変更は、オンライン・システムでオンライン変更を使ってインプリメントできます。

オンライン変更処理中は、データベースは使用できません。 /DBRECOVERY DB *HALDB_master_name* コマンドまたは UPDATE DB NAME(*HALDB_master_name*) STOP(ACCESS) コマンドを実行して HALDB データベース全体へのアクセスを停止してください。

DBD 変更がコミットされた後は、/START DB *HALDB_master_name* コマンドまたは UPDATE DB NAME(*HALDB_master_name*) START(ACCESS) コマンドを実行してデータベースへのアクセスを再開できます。 DBD 変更はすべての区画に影響するので、これらのコマンドには、マスター・データベース名を使用しなければなりません。

制約事項**:** オンライン変更を使用して IMS からデータベース名を削除すると、最初にコールド・スタートを行わなければ、その名前を他の HALDB 区画に再利用できません。同様に、RECON データ・セットから区画名を削除する場合、オンライン変更を使用すると、最初にコールド・スタートを行わなければ、その名前を他のデータベースに再利用できません。

関連概念:

851 ページの『オンライン変更および HALDB の変更』

関連タスク:

872 ページの『オンライン変更機能を使用したデータベース変更の活動化』

区画のハイ・キーの変更

区画のハイ・キーを変更することで、その区画に入れられるレコードの数を増やしたり、または減らしたりすることができます。

推奨事項**:** HALDB データベース内の区画のハイ・キーを変更する前に、HALDB 再編成番号検査機能を使用可能にしておいてください。 HALDB 再編成番号検査機能は、レコードがハイ・キーの変更によって、またはデータベース再編成によって移動したとき、データの保全性を保護します。

区画のハイ・キーを変更するには、次のようにします。

1. /DBRECOVERY DB *partition_name* コマンドまたは UPDATE DB NAME(*partition_name*) STOP(ACCESS) コマンドを実行することにより、すべての区画の場合は、ハイ・キーの変更の影響を受けます。
2. ハイ・キーの変更の影響を受けるすべての区画をアンロードします。
3. HALDB 区画定義ユーティリティー (%DFSHALDB) または DBRC コマンドを使用して区画のハイ・キー定義を変更します。
4. 次のユーティリティーのどちらかを実行して、変更の影響を受けるすべての区画を初期設定します。
   - HALDB 区画データ・セット初期設定ユーティリティー (DFSUPNT0)
   - データベース事前再編成ユーティリティー (DFSURPR0)
5. ステップ 2 の出力を使用して、変更によって影響を受けたすべての区画を再ロードします。再ロードされた区画内のデータ・セットに対して、イメージ・コピーが必要なことを示すフラグが設定されます。

6. 影響を受ける区画内のデータ・セットのイメージ・コピーを作成します。イメージ・コピーが必要なことを示すフラグが設定された後、RECON データ・セット内にイメージ・コピーが記録されるまで、DBRC はデータの更新を許可しません。
7. /START DB *partition_name* コマンドまたは UPDATE DB NAME (*partition_name*) START(ACCESS) コマンドを実行することにより、これらの区画を再度使用可能にします。

オンライン IMS システムは、以下のイベントの 1 つが発生するまで、RECON データ・セット内の区画定義の変更を認識しません。

- /START DB *HALDB_Master* OPEN コマンドの実行。
- UPDATE DB NAME(*HALDB_Master*) START(ACCESS) OPTION(OPEN) コマンドの実行。
- DL/I 呼び出しに起因する、変更の影響を受けた区画のための DBRC への許可呼び出しの実行。最初の DL/I 呼び出しによって HALDB 区画選択が再び実行され、正しい区画が選択され、許可されます。

関連概念:

893 ページの『レコード分散とハイ・キーの区分化』

196 ページの『HALDB 区画再編成番号』

既存の HALDB データベースへの区画の追加

既存の HALDB データベースに区画を追加することができます。一般的には、区画が大きくなりすぎたときに行います。区画を追加すると、通常、既存の区画内のレコードは新しい区画に移動します。

新しい区画を追加するには、次のようにします。

- HALDB データベースは区画の選択にハイ・キーを使用しているのか、区画選択出口ルーチンを使用しているのかを判別します。HALDB データベースが区画選択出口ルーチンを使用している場合、区画を追加するとき、補足ステップを実行する必要があります。
- 新しい区画に再分散されるレコードが入っているすべての既存の区画を特定します。新しい区画を追加するには、これらの区画をアンロード、初期設定、および再ロードする必要があります。
- 区画を追加した後、オンライン IMS システムが新しい区画を認識し、そのオンライン・データベース構造を更新することを確認してください。IMS は、以下のイベントのいずれかが発生したとき、そのオンライン構造を再作成します。
  - /START DB *HALDB_Master* OPEN コマンドの実行。
  - UPDATE DB NAME(*HALDB_Master*) START(ACCESS) OPTION(OPEN) コマンドの実行。
  - DL/I 呼び出しに起因する、変更の影響を受けた区画のための DBRC への許可呼び出しの実行。最初の DL/I 呼び出しによって HALDB 区画選択が再び実行され、正しい区画が適切に選択され、許可されます。

以下の図では、区画の選択にハイ・キーを使用するデータベースに区画 D が追加されています。区画 D のハイ・キーは 300 です。以前に定義されていた区画 A、B、および C のハイ・キーは 200、400、および高位値です。この新しい区画を追加すると、200 より大きく、300 までのキーを持つレコードは区画 B から区画

D に移動する必要があります。これは、区画 B は変更の影響を受けることを意味します。区画 D が 300 のハイ・キーで定義されると、IMS は区画 B および D に対して PINIT フラグを立てます。区画の初期設定では、この 2 つの区画が初期設定されます。区画 A および C は、この変更による影響はありません。

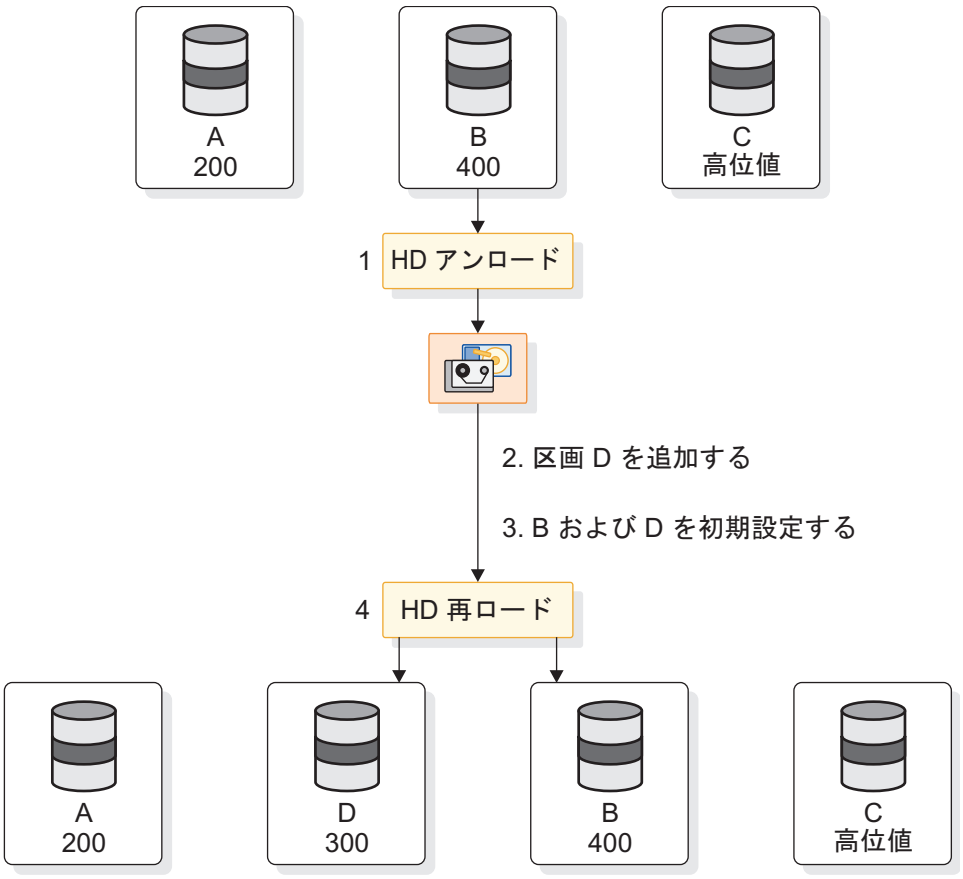


図 *300.* キー範囲の区分化による区画の追加

関連概念:

892 ページの『HALDB データベース内のレコード分散と区画境界』

関連タスク:

914 ページの『削除された HALDB 区画の復元』

区画の選択にハイ・キーを使用する HALDB データベースへの区画の追加

ユーザーが区画選択出口ルーチンを使用しなければ、HALDB データベースは区画の選択にハイ・キーを使用します。IMS は、ユーザーが追加する新しい区画と、その新しい区画へのレコードの再分散によって影響を受ける既存の区画に対して区画初期設定 (PINIT) フラグを立てます。

区画の選択にハイ・キーを使用する HALDB データベースに区画を追加するには、次のようにします。

1. /DBRECOVERY DB *partition_name* コマンドまたは UPDATE DB NAME(*partition_name*) STOP(ACCESS) コマンドを実行することにより、すべての区画の場合は、再分散されるレコードが入っています。
2. 新しい区画に再分散されるレコードが入っているすべての区画をアンロードします。
3. 区画定義ユーティリティーまたは DBRC コマンドを使用して新しい区画を定義します。
4. 新しい区画にデータ・セットを割り振ります。
5. 新しい区画と、その新しい区画へのレコードの再分散の影響を受ける既存の区画を初期設定します。
6. ステップ 1 の出力を使ってすべての区画を再ロードします。再ロードされたデータ・セットに対して、イメージ・コピーが必要なことを示すフラグが設定されます。
7. /START DB *partition_name* コマンドまたは UPDATE DB NAME (*partition_name*) START(ACCESS) コマンドを実行することにより、これらの区画を再度使用可能にします。
8. オンライン IMS システムが新しい区画を直ちに認識しない場合、/START DB *HALDB_Master* OPEN コマンドまたは UPDATE DB NAME(*HALDB_Master*) OPTION(OPEN) コマンドを実行して、コマンドに指定した HALDB データベースのオンライン区画構造をすべて再作成します。

新しい最大ハイ・キーを定義する区画の追加

HALDB データベースの中のキーが時間に基づいているか、または昇順の値に基づいている場合、どの既存の区画よりも大きいハイ・キーを持つ区画を追加することも可能です。キー値が大きくなるにつれて、新しいレコードを保持する新しい区画を追加することができます。

HALDB データベースに新しい最大のハイ・キーを定義する区画を追加するには、次のようにします。

1. レコードが他の区画から新しい区画に移動する場合、/DBRECOVERY DB *partition_name* コマンドまたは UPDATE DB NAME(*partition_name*) STOP(ACCESS) コマンドを実行することにより、レコードが移動する元の区画へのアクセスを停止します。
2. レコードが他の区画から新しい区画に移動する場合、レコードが移動する元の区画をアンロードします。
3. X'FF' のような適切なハイ・キーを指定して、新しい区画を定義します。
4. 必要な場合、以前に最大のハイ・キーを持っていた区画のハイ・キーを変更して、2 つの区画がハイ・キー X'FF' を持つことがないようにしてください。
5. 新しい区画と、レコードが移動した元の区画を初期設定します。
6. 必要な場合、新しい区画と、レコードが移動した元の区画をロードします。新しい区画内のデータ・セットに対して、イメージ・コピーが必要なことを示すフラグが設定されます。
7. /START DB *partition_name* コマンドまたは UPDATE DB NAME (*partition_name*) START(ACCESS) コマンドを実行することにより、これらの区画を再度使用可能にします。

8. オンライン IMS システムが新しい区画を直ちに認識しない場合、/START DB *HALDB_Master* OPEN コマンドまたは UPDATE DB NAME(*HALDB_Master*) OPTION(OPEN) コマンドを実行して、コマンドに指定した HALDB データベースのオンライン区画構造をすべて再作成します。

以下の図に、HALDB データベースに新しい最大のハイ・キーを定義する区画を追加する例を示します。このデータベースのキーは時間に基づきます。キーの高位部分は年を示します。年ごとに 1 つの区画があります。新年ごとに新しい区画が必要です。 2007zzz より上のキーを持つレコードはありません。そのようなレコードがデータベースに追加される前に、ハイ・キーが 2008zzz の区画 D が追加されます。既存の区画から移動するレコードはないので、既存の区画は影響を受けません。

図 *301.* キー範囲の区分化による、より大きいキーの区画の追加

区画選択出口ルーチンを使用する HALDB データベースへの区画の追加

区画選択出口ルーチンを使用する HALDB データベースに区画を追加する場合、レコードが新しい区画に移動する元の区画に PINIT フラグをユーザーが手動で設定する必要があります。

出口ルーチンで使用される区画選択基準を変更するだけでなく、新しい区画を認識するように出口ルーチンを変更が必要になる場合もあります。

区画選択出口ルーチンを使用する場合、IMS は、区画の追加によってどの既存の区画が影響を受けるかを知ることはできません。そのため、IMS は、新しい区画へのレコードの再分散の影響を受ける既存の区画に PINIT フラグを設定しません。既存の区画に対しては、ユーザーが PINIT フラグを設定しなければなりません。

また、使用している区画選択出口ルーチンに、構造の初期設定呼び出しおよび変更呼び出しを処理するコードが含まれていることを確認してください。これらの呼び

出しは、区画が追加または変更された後でオンライン区画構造を更新するときに IMS が発行します。IMS と一緒に配布されるサンプルの区画選択出口ルーチン (DFSPSE00) には、これらの呼び出しをサポートするコードが含まれています。

区画選択出口ルーチンを使用する HALDB データベースに新しい区画を追加するには、次のようにします。

1. 適宜、データベースへのアクセスを停止します。
   - 区画選択出口ルーチンを変更する必要がある場合、/DBRECOVERY DB *HALDB_master_name* コマンドまたは UPDATE DB NAME(*HALDB_master_name*) STOP(ACCESS) コマンドを実行することにより、HALDB マスター・データベースへのアクセスを停止します。HALDB マスターへのアクセスを停止すると、既存の区画選択出口ルーチンはストレージからアンロードされます。
   - 区画選択出口ルーチンを変更する必要がない場合、/DBRECOVERY DB *partition_name* コマンドまたは UPDATE DB NAME(*partition_name*) STOP(ACCESS) コマンドを実行することにより、レコードが新しい区画に移動する元の区画へのアクセスのみを停止します。
2. 必要な場合、区画選択出口ルーチンを、新しい区画を使用するように変更します。
3. レコードが移動する元の区画 (1 つまたは複数) をアンロードします。
4. 新しい区画を定義します。 IMS が新しい区画に対して PINIT フラグを立てます。
5. 該当する既存の区画に PINIT フラグを設定します。
6. PINIT フラグが設定されている区画を初期設定します。
7. 新しい区画と、レコードが移動した元の区画をロードします。
8. 再ロードされたデータ・セットのイメージ・コピーを取ります。初期設定されたか、HD 再ロードによってロードされた区画内のデータ・セットに対して、イメージ・コピーが必要なことを示すフラグが設定されます。
9. 適宜、データベースへのアクセスを再開します。
   - データベース全体を停止した場合、/START DB *HALDB_master_name* コマンドまたは UPDATE DB NAME(*HALDB_master_name*) START(ACCESS) コマンドを実行することにより、アクセスを再開します。 HALDB マスターへのアクセスを停止すると、既存の区画選択出口ルーチンはストレージからアンロードされます。
   - 選択した区画のみを停止した場合、/START DB *partition_name* コマンドまたは UPDATE DB NAME (*partition_name*) START(ACCESS) コマンドを実行することにより、アクセスを再開します。
10. データベース全体へのアクセスを停止していないときにオンライン IMS システムが新しい区画を直ちに認識しない場合、/START DB *HALDB_Master* OPEN コマンドまたは UPDATE DB NAME(*HALDB_Master*) OPTION(OPEN) コマンドを実行して、コマンドに指定した HALDB データベースのオンライン区画構造をすべて再作成します。

サンプルの HALDB 区画選択出口ルーチン (DFSPSE00) について詳しくは、「*IMS V13* 出口ルーチン」を参照してください。

HALDB 区画の使用不可および使用可能の設定

区画を使用不可にする ことにより、アクティブな HALDB データベースから区画を一時的に除去することができます。

使用不可にされた区画は HALDB データベースから削除されたように見え、IMS および他のユーティリティーで使用することはできません。しかし、区画 ID 番号およびイメージ・コピー・レコードなど、区画に関連付けられたすべてのレコードは DBRC によって RECON データ・セットに保存されています。

通常、削除する予定の区画を使用不可にすることになります。区画を使用不可にすると、実際に区画およびその区画に関連付けられたリカバリー情報を削除せずに、提案された削除をテストすることができます。区画を使用不可にした後、それを削除することもできるし、区画を使用可能にし、リカバリーすることによって、その区画を HALDB データベースに復元することもできます。

関連概念:

892 ページの『HALDB データベース内のレコード分散と区画境界』

897 ページの『IMS が区画 ID 番号を割り当てる方法』

使用不可にされた区画について

RECON リストにおいて、フラグ PARTITION DISABLED=YES は、区画が使用不可であることを表します。 HALDB 区画定義ユーティリティーは、区画の状況を DISABLED と表示することによって、使用不可にされた区画を表します。

多くの場合、使用不可にされた区画は IMS で認識されず、RECON データ・セットに登録されていないかのように扱われます。しかし、その場合でも、DBRC LIST コマンドまたは HALDB 区画定義ユーティリティー (%DFSHALDB) を使用することによって、使用不可にされた区画に関するすべての情報を表示することができます。しかし、使用不可にされた区画は、HALDB データベースに対して定義できる区画の最大数にはカウントされます。

使用不可にされた区画は、DBRC のグループ CAGROUP、DBDSGRP、DBGROUP、および RECOVGRP から除去されませんが、そのグループが DBRC コマンドで使用された場合、一般的に、そのグループの一部として処理されません。これは、コマンドで使用された暗黙のグループについても当てはまります。これに対する例外は、GENJCL.CA コマンドを発行し、IMS データベース変更累積ユーティリティーを実行したときに、変更累積グループのメンバーである、使用不可にされた区画に対しては変更が累積されることです。

GENJCL.IC コマンドおよび GENJCL.RECOV コマンドは、使用不可にされた区画に対しては失敗します。これらのコマンドがどの種類のグループに指定されても、使用不可にされた区画は単にスキップされ、JCL は生成されず、メッセージは出されません。

以下のコードは、DBRC CHANGE.PART コマンドを使用することにより、区画を使用不可にする例です。

```
//CHGPART JOB
...
...
//SYSIN DD *
CHANGE.PART DBD(DB3) PART(PART3) DISABLE
/*
```

HALDB 区画の使用不可の設定

区画を使用不可にすると、アクティブな HALDB データベースからその区画が除去されますが、すべての区画レコードは RECON データ・セットに保持されます。使用不可にした区画は、後で使用可能にすることができます。

HALDB 区画を使用不可にするには、次のようにします。

1. 使用不可にする予定の区画と、使用不可にされた区画からレコードが移動する先の区画のイメージ・コピーを取ります。
2. /DBRECOVERY DB *partition_name* コマンドまたは UPDATE DB NAME(*partition_name*) STOP(ACCESS) コマンドを実行することにより、使用不可にしようとしている区画と、影響を受ける他の区画へのアクセスを停止します。
3. 使用不可にしようとしている区画と、影響を受ける他の区画をアンロードします。
4. 次の方法の 1 つを使って区画を使用不可にします。
   - HALDB 区画定義ユーティリティー
   - DBRC コマンド CHANGE.PART DBD(*HALDB_master_name*) PART (*partition_name*) DISABLE
5. HALDB データベースがハイ・キー区分化を使用している場合、DBRC に定義されている各区画のハイ・キー値が依然として適切であることを確認します。
6. 変更の影響を受ける残りの区画を初期設定します。
7. 変更の影響を受ける残りの区画を再ロードします。
8. 再ロードされたデータ・セットのイメージ・コピーを取ります。初期設定されたか、HD 再編成再ロード・ユーティリティー (DFSURGL0) によってロードされた区画内のデータ・セットに対して、イメージ・コピーが必要なことを示すフラグが設定されます。
9. /START DB *partition_name* コマンドまたは UPDATE DB NAME(*partition_name*) START(ACCESS) コマンドを実行して、影響を受けた区画を再度使用可能にします。

区画を使用可能にする設定について

使用不可にされた区画を使用可能にすると、その区画は IMS で再度使用できるようになります。

区画を HALDB データベースで使用可能にする手順は、新しい区画を HALDB データベースに追加する手順と実質的に同じです。使用不可になっている区画を使用可能にするには、DBRC コマンド CHANGE.PART に ENABLE パラメーターを指定して実行します。

HALDB 区画を使用可能にするときに、アンロードおよび再ロード処理を共に実行するか、またはリカバリー処理を使用して HALDB データベースを区画が使用不可になる前の状態に復元することができます。

区画が使用不可になった後でデータベースが更新された場合は、タイム・スタンプ・リカバリーを使用して HALDB データベースをリカバリーする必要があります。通常、タイム・スタンプ・リカバリーでは、HALDB データベース全体とそのすべての区画、および関連するすべての HALDB データベースをリカバリーする必要があります。

区画を使用可能にすると、DBRC はその区画にリカバリーが必要なことを示すフラグを立てます。HALDB データベースが区画選択出口ルーチンを使用している場合、DBRC は、データベース内のすべての区画に区画の初期設定が必要なことを示すフラグを立てます。 HALDB データベースがハイ・キー区分化を使用している場合、DBRC は、次に大きいハイ・キー値を持つ区画にその区画の初期設定が必要なことを示すフラグを立てます。

HALDB 区画の使用可能の設定

使用不可にされた区画を使用可能にすると、その区画は IMS で再度使用できるようになります。

HALDB 区画の使用可能化をアンロードおよび再ロード処理と共に実行するには、次のようにします。

1. /DBRECOVERY DB *partition_name* コマンドまたは UPDATE DB NAME(*partition_name*) STOP(ACCESS) コマンドを実行することにより、すべての区画の場合は、再分散された後にレコードが入ります。
2. オフラインの区画をアンロードします。
3. 次の方法の 1 つを使って区画を使用可能にします。
   - HALDB 区画定義ユーティリティー
   - DBRC コマンド CHANGE.PART DBD(*HALDB_master_name*) PART(*partition_name*) ENABLE

   DBRC は、使用可能にされた区画内のすべてのデータベース・データ・セットにリカバリーが必要であるというマークを付けます。
4. HALDB データベースがハイ・キー区分化を使用している場合、DBRC に定義されている各区画のハイ・キー値が適切であることを確認します。
5. 使用可能にされた区画および影響を受けた他のすべての区画にデータ・セットを割り振ります。
6. オフラインの区画を初期設定します。
7. 区画を再ロードします。データ・セットに対して、イメージ・コピーが必要なことを示すフラグが設定されます。
8. 再ロードされた区画のイメージ・コピーを取ります。
9. /START DB *partition_name* コマンドまたは UPDATE DB NAME (*partition_name*) START(ACCESS) コマンドを実行することにより、これらの区画を再度使用可能にします。
10. オンライン IMS システムが使用可能にされた区画を直ちに認識しない場合、/START DB *HALDB_Master* OPEN コマンドまたは UPDATE DB NAME(*HALDB_Master*) OPTION(OPEN) コマンドを実行して、HALDB データベースのオンライン区画構造をすべて再作成します。

区画を使用可能にする際に HALDB データベースをリカバリーする

区画を使用可能にすると、RECON データ・セットでその区画にリカバリーが必要なことを示すフラグが設定されます。使用可能にした区画をリカバリーする場合は、その区画が使用不可になったときに影響を受けたすべての区画もリカバリーする必要があります。

区画が使用不可になった後で HALDB データベースが更新された場合は、その HALDB データベース全体、および使用可能にする区画が含まれる HALDB データベースに論理的に関連するすべての HALDB データベースのリカバリーが必要になる可能性があります。

リカバリーの一環として区画を使用可能にするには、次のようにします。

1. 次の方法の 1 つを使って区画を使用可能にします。
   - HALDB 区画定義ユーティリティー
   - DBRC コマンド `CHANGE.PART DBD(HALDB_master_name) PART (partition_name) ENABLE`

   DBRC は、使用可能にされた区画内のすべてのデータベース・データ・セットにリカバリーが必要であるというマークを付けます。
2. 区画が使用不可になる前に取られたイメージ・コピーを使用して、区画が使用不可になったときに変更されたすべての区画、または区画が使用不可になった後にアプリケーションによって変更されたすべての区画をリカバリーします。

   区画が使用不可になった後でデータベースが更新された場合は、タイム・スタンプ・リカバリーを実行する必要があります。
3. `/START DB partition_name` コマンドまたは `UPDATE DB NAME (partition_name) START(ACCESS)` コマンドを実行することにより、これらの区画を再度使用可能にします。
4. オンライン IMS システムが使用可能にされた区画を直ちに認識しない場合、`/START DB HALDB_Master OPEN` コマンドまたは `UPDATE DB NAME(HALDB_Master) OPTION(OPEN)` コマンドを実行して、HALDB データベースのオンライン区画構造をすべて再作成します。

関連概念:

661 ページの『データベースのリカバリー』

既存の HALDB データベースからの区画の削除

既存の HALDB データベースから区画を削除することができます。これは、一般的は、区画にデータがほとんど入っていないときに行います。通常は、区画を削除する際にそのレコードを別の区画に移動します。

区画を削除する前に、以下を行ってください。

1. HALDB データベースは区画の選択にハイ・キーを使用しているのか、区画選択出口ルーチンを使用しているのかを判別します。HALDB データベースが区画選択出口ルーチンを使用している場合、区画を削除するとき、補足ステップを実行する必要があります。

2. 削除された区画に以前保管されていたレコードを受け取る他のすべての区画を特定します。削除された区画に加えて、これらの区画をアンロード、初期設定、および再ロードする必要があります。
3. HALDB データベースが副次索引 (PSINDEX) を使用するかを判別します。区画を削除した後、PSINDEX の再作成が必要になる場合があります。
4. 区画のイメージ・コピーを取ります。
5. HALDB 区画定義ユーティリティー (%DFSHALDB) を使用することにより、削除しようとしている区画の区画定義をエクスポートします。
6. 削除する予定の区画を使用不可にします。 区画を使用不可にすると、区画に関する情報を RECON データ・セットから削除せずに、その区画は、事実上、HALDB データベースから除去されます。区画 ID 番号を含めて、区画定義は RECON データ・セットに保存されています。その区画を復元する必要がある場合、その区画を簡単に再び使用可能にすることができ、以前のイメージ・コピーはすべて、まだ使用可能です。その区画を復元する必要がないことが確実になったら、RECON データ・セットから該当の区画レコードを削除してください。

以下の図に、ハイ・キーを使用して区画の選択を行う HALDB データベースから区画を削除する例を示します。HALDB データベースに、区画 A、B、C、および D があります。ハイ・キー 400 を持つ区画 B が削除されます。区画 B のレコードは区画 C に移動し、このことは、区画 C は、区画 B の削除の影響を受けることを意味します。区画 B の定義が削除されると、IMS は区画 C に対して PINIT フラグを設定します。区画の初期設定は、区画 C を初期設定します。区画 A および D は、この変更の影響を受けません。

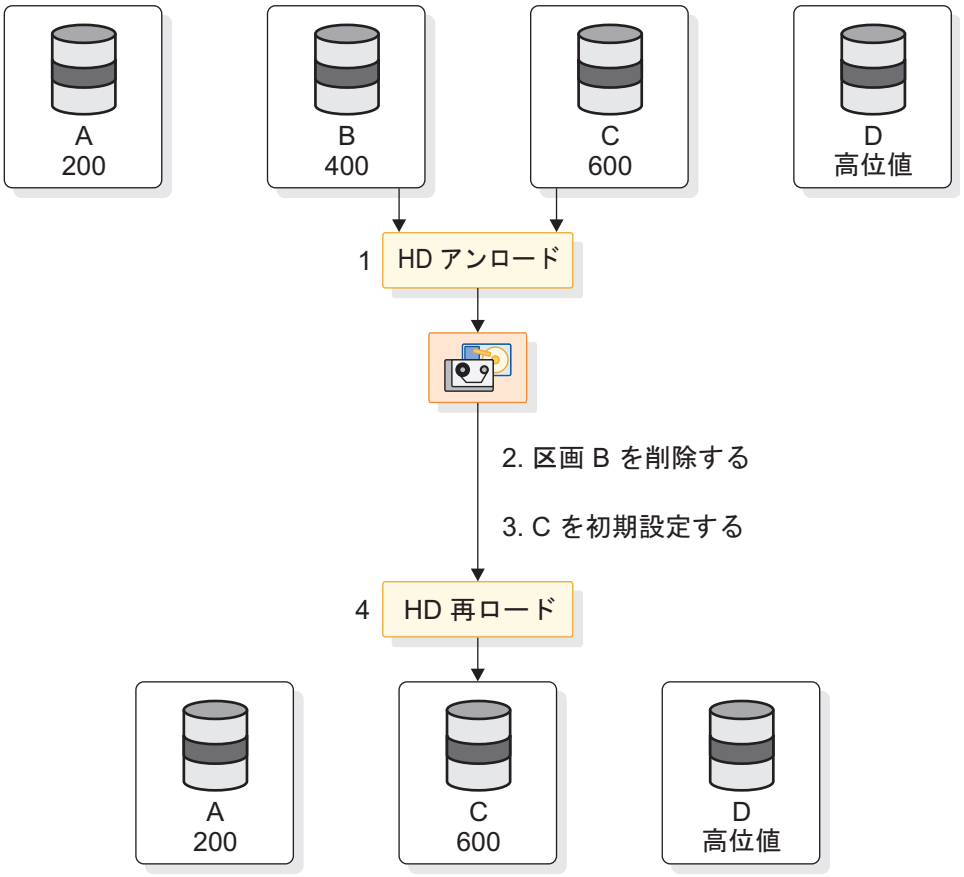


図 *302.* キー範囲の区分化による区画の削除

関連概念:

892 ページの『HALDB データベース内のレコード分散と区画境界』

897 ページの『IMS が区画 ID 番号を割り当てる方法』

ハイ・キー区分化を使用する HALDB データベースからの区画の削除

区画を永久に削除すると、区画およびそこに含まれるすべてのレコードが HALDB データベースから除去されます。削除された区画に関連するレコードは、RECON データ・セットからも永久に除去されます。

ハイ・キーを使用して区画を選択する HALDB データベースから区画を削除するには、次のようにします。

1. 削除する予定の区画と、削除された区画からレコードが移動する先の区画のイメージ・コピーを取ります。
2. `/DBRECOVERY DB partition_name` コマンドまたは `UPDATE DB NAME(partition_name) STOP(ACCESS)` コマンドを実行することにより、削除しようとしている区画と、影響を受ける他の区画へのアクセスを停止します。
3. 削除しようとしている区画と、削除の影響を受ける他の区画の両方をアンロードします。
4. 次の方法の 1 つを使って、区画の定義を RECON データ・セットから削除します。
   - HALDB 区画定義ユーティリティー

- DBRC コマンド DELETE.PART DBD(*HALDB_master_name*) PART (*partition_name*)

5. RECON データ・セットに定義されている各区画のハイ・キー値が依然として適切であることを確認します。
6. 削除された区画のレコードを入れる、残っている区画を初期設定します。
7. ステップ 1 の出力を使用して、残っている区画を再ロードします。
8. 再ロードされた区画のイメージ・コピーを取ります。再ロードされた区画のデータ・セットに対して、イメージ・コピーが必要なことを示すフラグが設定されます。
9. /START DB *partition_name* コマンドまたは UPDATE DB NAME(*partition_name*) START(ACCESS) コマンドを実行して、影響を受けた区画を再度使用可能にします。

削除処理の完了後、IMS が HALDB データベースにオンライン区画構造を再作成するとき、IMS は、削除された区画について理由コード 90 を示したメッセージ DFS0415W を出します。

最小キーを持つ区画とその全レコードの削除

最小のキー範囲を持つ区画をその区画に含まれるすべてのレコードと共に削除すると便利な場合があります。

このような状況は、HALDB データベース内のキーが、時間または昇順の値に基づいている場合に生じる可能性があります。キー値が大きくなるにつれて、空になった区画、またはそこに含まれるレコードにもう意味がなくなった区画を削除できます。

例えば以下の図では、データベースのキーは時間に基づきます。キーの高位部分は年を示し、年ごとに区画が存在します。この例では、2005zzz のハイ・キーを持つ区画 A は不要になりました。その中のレコードがすべて削除されているか、またはその中のレコードが不要になりました。実際、これは、区画 A の中の全レコードを削除するのに効率のよい方法です。他の区画との間で移動するレコードはないので、影響を受ける他の区画はありません。アンロード、再ロード、または初期設定が必要な区画はありません。この処理のステップは、区画の削除だけです。

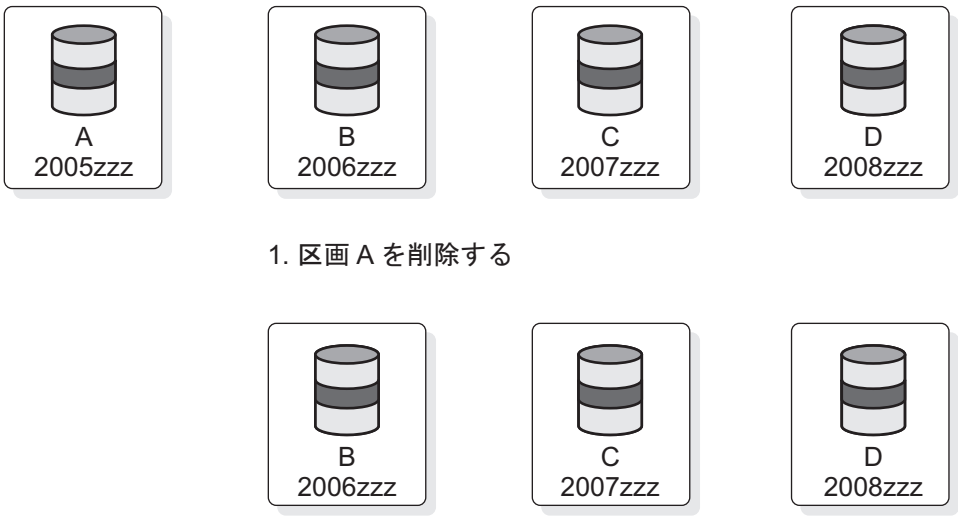


図 *303.* キー範囲の区分化を使用する *HALDB* での最小キーを持つ区画の削除

副次索引を使用する HALDB データベースから区画を削除する場合の考慮事項

HALDB データベースが副次索引を使用している場合は、区画を削除するときに注意が必要です。

削除された区画を指す項目が副次索引の中にある場合、その副次索引は無効になります。IMS Index Builder のようなツールで副次索引を再作成する必要があります。削除された区画を指す項目が副次索引の中にない場合、副次索引は影響を受けないので、再作成する必要はありません。削除された区画の中にレコードがなければ、副次索引を再作成する必要はありません。その副次索引は区画削除の影響を受けません。

区画選択出口ルーチンを使用する HALDB データベースからの区画の削除

区画選択出口ルーチンを使用する場合、IMS は、区画の削除によってどの区画が影響を受けるかを認識できません。

そのため、IMS は、削除された区画からのレコードの再分散の影響を受ける区画に PINIT フラグを設定しません。これらの区画に対しては、ユーザーが PINIT フラグを手動で設定しなければなりません。

一部の区画の中の区画ストリングを変更するだけでなく、新しい区画を認識するように出口ルーチンを変更する必要がある場合もあります。

区画選択出口ルーチンを使用する HALDB データベースから区画を削除するには、次のようにします。

1. `/DBRECOVERY DB partition_name` コマンドまたは `UPDATE DB NAME(partition_name) STOP(ACCESS)` コマンドを実行することにより、削除しようとしている区画と、削除された区画からのレコードが移動する先の区画をオフラインにします。

2. HD 再編成アンロード・ユーティリティー (DFSURGU0) を実行して、削除しようとしている区画と、削除された区画からのレコードが移動する先の区画の両方をアンロードします。
3. 削除しようとしている区画の定義を RECON データ・セットから削除します。
4. 必要な場合、区画選択出口ルーチンを変更します。
5. 必要に応じて、区画ストリングを区画用に変更します。
6. 削除された区画からのレコードを受け取る区画に PINIT フラグを設定します。
7. PINIT フラグが設定されている区画を初期設定します。
8. HD 再編成再ロード・ユーティリティー (DFSURGL0) を実行して、区画をロードします。
9. 再ロードされたデータ・セットのイメージ・コピーを取ります。初期設定されたか、DFSURGL0 ユーティリティーによってロードされた区画内のデータ・セットに対して、イメージ・コピーが必要なことを示すフラグが設定されます。
10. `/START DB partition_name` コマンドまたは `UPDATE DB NAME (partition_name) START(ACCESS)` コマンドを実行することにより、影響を受けた区画を再度使用可能にします。

削除された HALDB 区画の復元

削除された HALDB 区画の復元方法は、区画を削除する前に特定の予防措置を講じたかどうかによって異なります。

区画を削除する前に、イメージ・コピーを作成し、HALDB 区画定義ユーティリティー (%DFSHALDB) を使って区画 ID を含めて区画定義をエクスポートした場合、区画定義をインポートし、イメージ・コピーを適用することによって、削除された区画を復元することができます。

HALDB 区画定義ユーティリティーを使って、削除された区画のみをインポートするには、区画定義をインポートするときに「Import a Database」パネルで処理オプションとして「`Try all partitions`」 を指定します。削除された区画の定義のみがインポートされます。削除されなかった区画の定義はインポートされません。復元された区画には、削除前と同じ区画 ID が割り当てられるため、ユーザーは、区画を削除する前に作成した区画データ・セットのイメージ・コピーを使用できます。

区画を削除する前にイメージ・コピーの作成と区画定義のエクスポートを行わなかった場合は、削除された区画を復元する代わりに、削除された区画が保持していたレコードを保持する新しい区画を定義することができます。ただし、この場合は、削除された区画とは異なる区画 ID が新しい区画に割り当てられ、削除された区画の既存のイメージ・コピーを使用することはできません。

関連概念:

897 ページの『IMS が区画 ID 番号を割り当てる方法』

関連タスク:

901 ページの『既存の HALDB データベースへの区画の追加』

HALDB 区画の名前の変更

HALDB 区画の名前は、区画を削除して、新しい名前を付けてその区画を再定義することによって変更できます。

ただし、区画を削除して再定義する場合、注意すべき結果が次のようにいくつかあります。

- 区画 ID 番号が変更されます。
- 区画のイメージ・コピー・レコードが RECON データ・セットから削除されます。
- 区画の ALLOC レコードが RECON データ・セットから削除されます。

注: オンライン変更機能を使用すると、RECON データ・セットから削除された HALDB 区画名は、IMS のコールド・スタートも実行しなければ、データベースに命名するためにオンライン変更機能が再利用することはできません。逆も言えます。すなわち、IMS から削除されたデータベース名は、コールド・スタートも実行しなければ、HALDB 区画に命名するために再利用することはできません。

HALDB 区画の名前を変更するには、次のようにします。

- /DBRECOVERY DB *partition_name* コマンドまたは UPDATE DB NAME(*partition_name*) STOP(ACCESS) コマンドを実行することにより、区画をオフラインにします。
- 区画をアンロードします。
- 区画定義を RECON データ・セットから削除します。
- RECON データ・セットに新しい名前で区画を再定義します。
- 区画を初期設定します。
- 区画を再ロードします。
- /START DB *partition_name* コマンドまたは UPDATE DB NAME (*partition_name*) START(ACCESS) コマンドを実行することにより、区画を使用可能にします。

名前変更処理が完了すると、IMS は、HALDB データベースのオンライン区画構造を再作成するときに、削除された区画について理由コード 90 を示したメッセージ DFS0415W を出します。

関連概念:

897 ページの『IMS が区画 ID 番号を割り当てる方法』

関連タスク:

869 ページの『オンライン・システムでのデータベースの動的な変更』

PHDAM 区画内のルート・アンカー・ポイント数の変更

PHDAM 区画に同じルート・アンカー・ポイント (RAP) にランダム化される多数のルートがあると、パフォーマンスに悪影響を及ぼすロック競合の問題が発生するおそれがあります。そこでこの問題を修正するために、その区画内の RAP の数を増やすことができます。

PHDAM 区画内の RAP の数を変更するには、次のようにします。

1. /DBRECOVERY DB *partition_name* コマンドまたは UPDATE DB NAME(*partition_name*) STOP(ACCESS) コマンドを実行することにより、区画をオフラインにします。
2. その区画からデータをアンロードします。
3. HALDB 区画定義ユーティリティー (%DFSHALDB) または DBRC コマンドを使用して区画の RAP の数を変更します。
4. 区画を初期設定します。
5. 区画を再ロードします。
6. 区画のデータ・セットのイメージ・コピーをとります。
7. /START DB *partition_name* コマンドまたは UPDATE DB NAME (*partition_name*) START(ACCESS) コマンドを実行することにより、区画を使用可能にします。

レコード・セグメントの変更

HALDB データベース内のセグメントおよびセグメントの階層の変更は、DBDGEN 処理の DBD ステートメントで行います。

HALDB データベース内のセグメントおよびセグメントの階層のほとんどの変更で、ユーザーは、最初に HALDB データベース内のすべての区画をアンロードし、変更が完了した後でその区画を再ロードする必要があります。

以下のリストに、HALDB セグメントおよびセグメントの階層に行うことができる代表的な変更と、その変更は HALDB データベースの完全なアンロードが必要かどうかを示します。

- セグメントのサイズを大きくしないフィールド定義を追加する場合、データベースのアンロードは必要ありません。
- セグメントのサイズを小さくせず、また副次索引で使用されないフィールド定義を削除する場合、データベースのアンロードは必要ありません。
- セグメント・タイプを追加または削除する場合、データベースのアンロードが必要です。
- セグメント・サイズの定義を変更する場合、データベースのアンロードが必要です。
- シーケンス・フィールドの定義を変更する場合、データベースのアンロードが必要です。
- 階層内のセグメントの場所を変更する場合、データベースのアンロードが必要です。
- ポインター・オプションを変更する場合、データベースのアンロードが必要です。

関連概念:

801 ページの『レコード・セグメントの変更』

HALDB 区画のデータ・セットの変更

HALDB データベースでは、各区画はそれ自身のデータ・セットを持っています。1 つの区画のデータベース・データ・セットを変更しても、他の区画内のデータベース・データ・セットには影響しません。

変更中の区画のみが処理に使用不可になっている必要があるだけです。 HALDB データベース内の他の区画はすべて、使用可能なままでかまいません。

HALDB 区画データ・セットに以下の変更のいずれかを行う前に、区画に対して /DBRECOVERY DB *partition_name* コマンドまたは UPDATE DB NAME(*partition_name*) STOP(ACCESS) コマンドを実行することにより、その区画へのアクセスを停止しなければなりません。 HALDB データベース内の他の区画はすべて、使用可能なままでかまいません。

HALDB データ・セット名接頭部の変更

各区画のデータ・セット名接頭部が RECON データ・セットに保管されています。これは、HALDB 区画定義ユーティリティー (%DFSHALDB) を使用するか、または DBRC バッチ・コマンド `CHANGE.PART` の `DSNPREFX(`*`string`*`)` パラメーターを使用して変更することができます。

データ・セット名接頭部を変更すると、その区画に対して区画の初期設定が必要なことを示す (PINIT) フラグが設定されます。

区画データ・セットのフリー・スペース・パラメーターの変更

HALDB 区画データ・セットのフリー・スペースの指定は RECON データ・セットに保管されています。

これは、HALDB 区画定義ユーティリティー (%DFSHALDB) を使用するか、または DBRC バッチ・コマンド `CHANGE.PART` のフリー・スペース・パラメーター `FBFF(`*`value`*`)` または `FSPF(`*`value`*`)` を使用して変更することができます。

区画が統合 HALDB オンライン再編成機能によって再編成中 (データ・セット内のレコードの OLREORG CURSOR ACTIVE = YES で示される) であるか、または区画が統合 HALDB オンライン再編成処理用に IMS システムによって所有されている間は、その区画のフリー・スペース・パラメーターを変更することはできません。

HALDB データベース・データ・セット用の OSAM ブロック・サイズの変更

HALDB データベース・データ・セットの OSAM ブロック・サイズは、RECON データ・セットに保管されています。

これは、HALDB 区画定義ユーティリティー (%DFSHALDB) を使用するか、または DBRC バッチ・コマンド `CHANGE.PART` の `BLOCKSZE(`*`nnnnn`*`)` パラメーターを使用して変更することができます。データ・セットの OSAM ブロック・サイズを変更した後、その区画に対して区画の初期設定が必要なことを示す (PINIT) フラグが設定されます。

HALDB データベース・データ・セット用の VSAM CI サイズの変更

HALDB データベース・データ・セットVSAM CI サイズは、z/OS DFSMS コマンドでのみ変更できます。VSAM CI サイズを変更した後、その区画に対して区画の初期設定が必要なことを示す (PINIT) フラグが設定されます。

VSAM CI サイズは DBRC RECON データ・セットに保管されないため、HALDB 区画定義ユーティリティー (%DFSHALDB) または DBRC CHANGE コマンドで変更することはできません。

VSAM データ・セットに使用する z/OS DFSMS コマンドについては、「*z/OS DFSMS カタログのためのアクセス方式サービス・プログラム*」を参照してください。

OSAM データ・セットおよび HALDB データベースの最大サイズ

HALDB データベースの OSAM PHDAM データ・セットと PHIDAM データ・セットの最大サイズは、RECON データ・セット内で設定します。

HALDB データベースの OSAM PHDAM データ・セットおよび PHIDAM データ・セットの最大サイズは、4 ギガバイトまたは 8 ギガバイトにすることができます。 HALDB データベースが 8 GB の OSAM データ・セットをサポートする場合、HALDB オンライン再編成 (OLR) 機能を使用してデータベースを再編成することはできません。代わりに、データベース再編成用のオフライン・ユーティリティーを使用するか、4 GB の OSAM データ・セット (OLR をサポートします) を使用してください。

新規 HALDB データベースの OSAM データ・セットの最大サイズの定義

新規 HALDB マスター・データベースを DBRC に登録する際に、OSAM PHDAM または PHIDAM データ・セットの最大サイズを指定できます。

HALDB データベースは、OSAM PHDAM または PHIDAM のデータベース・データ・セット内で最大 4 ギガバイトまたは 8 ギガバイトのデータをサポートすることができます。

1. DBRC バッチ・コマンド INIT.DB DBD(*master_HALDB_name*) TYPHALDB を発行します。
2. デフォルトの 4 GB の OSAM データ・セットを使用しない場合、および HALDB オンライン再編成 (OLR) 機能を使用できるようにする必要がない場合は、コマンドで以下のキーワードを指定します。
   - 8 GB の OSAM データ・セットを使用する場合、OLRNOCAP および OSAM8G を指定します。
   - オンラインで再編成できない 4 GB の OSAM データ・セットを使用する場合、OLRNOCAP および NOOSAM8G を指定します。

HALDB データベースが DBRC に登録され、マスター・データベース・レコードは、HALDB データベースが 4 GB または 8 GB いずれの OSAM データ・セットをサポートするかを示します。

HALDB の最大 OSAM データ・セット・サイズの 4 GB から 8 GB への変更

CHANGE.DB コマンドを使用して、HALDB データベースの OSAM PHDAM および PHIDAM データ・セットの最大サイズを 4 GB から 8 GB に増やします。

HALDB OSAM データ・セットの最大サイズを 8 GB に増やすには、以下のようにします。

1. データベースがオンライン IMS システムで開いている場合、/DBRECOVERY DB *haldb_master* コマンドまたは UPDATE DB NAME(*haldb_master*) STOP(ACCESS) コマンドを発行して、データベースへのアクセスを停止します。
2. データベースに対して HALDB オンライン再編成が実行された場合、すべてのデータベース区画が A から J のデータ・セットを使用することを確認します。
3. OSAM8G キーワードを指定して、CHANGE.DB コマンドを発行します。データベースがオンラインで再編成された場合、OLRNOCAP キーワードも指定する必要があります。
4. /START DB *haldb_master* コマンドまたは UPDATE DB NAME (*partition_name*) START(ACCESS) OPTION(OPEN) コマンドを実行することにより、これらの区画を使用可能にします。

RECON データ・セット内のマスター・データベース・レコードが更新され、HALDB が 8 GB の OSAM データ・セットをサポートすることを示します。

HALDB の最大 OSAM データ・セット・サイズの 8 GB から 4 GB への変更

CHANGE.DB コマンドを使用して、HALDB データベースの OSAM PHDAM および PHIDAM データ・セットの最大サイズを減らします。

OSAM データ・セットのデータ容量を 8 GB から 4 GB に減らす際には、特にデータベースに 4 GB を超えるデータが含まれる場合、データベース内のデータを管理する方法を検討する必要があります。

OSAM データ・セットの最大サイズを 4 GB に減らすには、以下のようにします。

1. データベースが既にオンライン IMS システムで開いている場合、/DBRECOVERY DB *haldb_master* コマンドまたは UPDATE DB NAME(*haldb_master*) STOP(ACCESS) コマンドを発行して、データベースへのアクセスを停止します。
2. データベース内のすべての区画からデータをアンロードします。
3. NOOSAM8G キーワードを指定して CHANGE.DB コマンドを発行します。
4. 4 GB を超えるデータを持つデータ・セットが含まれる区画を分割します。新規区画を定義するか、HALDB 区画定義ユーティリティー (%DFSHALDB) または DBRC コマンドを使用して既存の区画のハイ・キー定義を再調整します。
5. HALDB 区画データ・セット初期設定ユーティリティー (DFSUPNT0) またはデータベース事前再編成ユーティリティー (DFSURPR0) を使用して、変更の影響を受けるすべての区画を初期設定します。
6. データベースからのデータのアンロードからの出力を使用して、変更の影響を受けるすべての区画を再ロードします。

7. 影響を受ける区画内のデータ・セットのイメージ・コピーを作成します。イメージ・コピーが必要なことを示すフラグが設定された後、RECON データ・セット内にイメージ・コピーが記録されるまで、DBRC はデータの更新を許可しません。
8. /START DB *haldb_master* コマンドまたは UPDATE DB NAME (*partition_name*) START(ACCESS) OPTION(OPEN) コマンドを実行することにより、これらの区画を使用可能にします。

RECON データ・セット内のマスター・データベース・レコードが更新され、HALDB が 4 GB の OSAM データ・セットをサポートすることを示します。

出口ルーチンの変更と HALDB データベース

HALDB データベースで使用できるほとんどの出口ルーチンについて、それらを変更する方法は、区分化されていないデータベース用に出口ルーチンを変更する場合と違いはありません。

HALDB 出口ルーチンに対するほとんどの指定は、RECON データ・セットの中ではなく、データベース定義ステートメントに書きます。この規則の例外は区画選択出口ルーチンまたは PHDAM ランダム化モジュールへの変更であって、この変更は、DBD と RECON データ・セットの両方に指定できます。RECON データ・セット内の指定は、データベース定義で書かれた指定に優先します。

PHDAM ランダム化ルーチンを除いて、データベース出口ルーチンに行う変更は、HALDB データベース全体とそのすべての区画に適用されます。

PHDAM ランダム化ルーチンまたは区画選択出口ルーチンのような出口ルーチンを変更した後、データベースの再編成を検討してください。

HALDB データベースで一般的に使用される出口ルーチンには、次のようなものがあります。

- HALDB 区画選択出口ルーチン
- データ・キャプチャー出口ルーチン
- データ変換ユーザー出口ルーチン
- セグメント編集/圧縮出口ルーチン
- 副次索引データベース保守出口ルーチン
- HDAM ランダム化ルーチンおよび PHDAM ランダム化ルーチン

関連概念:

430 ページの『データ・キャプチャー出口ルーチン』

427 ページの『セグメント編集/圧縮出口ルーチン』

関連資料:

Database Manager 出口ルーチン (出口ルーチン)

HALDB 区画選択出口ルーチンの追加または変更

ご使用のシステムが HALDB 区画選択出口ルーチンを使用する場合、この出口ルーチンを、HALDB データベース内の区画全体にわたってレコードの分散を調整するように変更する必要が生ずることがあります。

一般的に、各区画内のデータの量が HALDB データベース内のすべての区画にわたって均衡が取れなくなったときです。HALDB データベース全体がオフラインの間に出口ルーチンを切り替えるか、または既存の出口ルーチンを変更することにより、レコードの分散を変更することができます。

HALDB 区画選択出口ルーチンを追加または変更するには、以下の手順が必要です。

1. タイプ 1 コマンド /DBRECOVERY DB またはタイプ 2 コマンド UPDATE DB NAME(*dbname*) STOP(ACCESS) を実行することにより、HALDB データベース全体をオフラインにします。
2. 新規または変更された区画選択出口ルーチンの影響を受けるすべての区画をアンロードします。
3. 必要に応じて、区画ストリングまたはランダム化パラメーターのような区画定義を変更します。区画の定義を変更すると、その区画に PINIT フラグが設定され、そのため、次のステップで時間が節約できる可能性があります。
4. 必要に応じて、HALDB 区画定義ユーティリティー (%DFSHALDB) またはバッチ DBRC コマンド CHANGE.DB を使用することにより、各区画の PINIT フラグを設定または設定解除します。
   - 新規の区画選択出口ルーチンを追加しようとしている場合で、そのため、全区画にわたってレコードの新しい分散を取り入れている場合は、PINIT フラグはどれも変更する必要はありません。すべての区画で PINIT フラグが正しく設定されています。
   - 既存の区画選択出口ルーチンの名前のみを変更しようとしている場合で、全区画にわたるレコードの分散は同じままの場合、すべての区画の PINIT フラグをオフにしてください。出口ルーチンの名前だけが変更された場合でも、IMS はその出口ルーチンを新しい区画選択出口ルーチンと見なし、全区画に PINIT フラグを設定します。
   - 既存の区画選択出口ルーチンの名前を変更しようとしている場合で、区画選択出口ルーチンがすべての区画にわたってレコードをどのように分散するかも変更した場合、レコードの分散方法の変更の影響を受けないすべての区画の PINIT フラグをオフにしてください。
   - 既存の区画選択出口ルーチンによるすべての区画にわたるレコードの分散方法のみを変更しようとしている場合で、区画選択出口ルーチンの名前は変更していない場合、レコードの分散方法の変更の影響を受けるすべての区画に PINIT フラグを設定しなければなりません。
5. レコードの分散方法の変更の影響を受ける区画を初期設定します。
6. HD 再編成再ロード・ユーティリティー (DFSURGL0) を使用して、レコードの分散方法の変更の影響を受ける区画を再ロードします。
7. 再ロードされた区画のイメージ・コピーを取ります。初期設定されたか、DFSURGL0 ユーティリティーによってロードされた区画内のデータ・セットに対して、イメージ・コピーが必要なことを示すフラグが設定されます。
8. タイプ 1 コマンド /START DB またはタイプ 2 コマンド UPDATE DB NAME(dbname) START(ACCESS) を実行することにより、HALDB データベースへのアクセスを復元します。

詳しくは、

- サンプルの HALDB 区画選択出口ルーチン (DFSPSE00) については、「*IMS V13* 出口ルーチン」を参照してください。
- HD 再編成再ロード・ユーティリティーについては、「*IMS V13* データベース・ユーティリティー」を参照してください。

関連概念:

895 ページの『レコード分散と区画選択出口ルーチン』

PHDAM 区画用のランダム化モジュールまたはランダム化パラメーターの変更

PHDAM ランダム化モジュールに関する指定は 2 カ所に保管することができます。すなわち、HALDB データベース用の DBD と RECON データ・セット内の各区画用レコードです。

DBD 内の指定は、HALDB データベース内のすべての区画に適用されます。RECON データ・セット内の区画レコードでの指定は、その区画のみに適用され、DBD での指定よりも優先されます。

PHDAM 区画のランダム化モジュールまたはランダム化パラメーターを変更するには、以下の手順が必要です。

1. 区画に対して DBRC コマンド `LIST.DB DBD(`*`partition_name`*`)` を実行し、出力を保管することにより、現行のランダム化指定をバックアップします。
2. 区画に対してタイプ 1 コマンド `/DBRECOVERY` またはタイプ 2 コマンド `UPDATE DB NAME(`*`partition_name`*`) STOP(ACCESS)` を実行することにより、その区画へのアクセスを停止します。データベース内の他の区画はすべて、使用可能なままでかまいません。
3. 区画をアンロードします。
4. ランダム化パラメーターを変更しようとしている場合、区画定義ユーティリティーまたは DBRC コマンド `CHANGE.PART` を使用して変更します。
5. ランダム化モジュールを置き換えようとしている場合、区画定義ユーティリティーまたは DBRC コマンド `CHANGE.PART DBD(`*`name`*`) PART(`*`name`*`) RANDOMZR(`*`name`*`)` を使用して新しいランダム化モジュールの名前を指定します。
6. HALDB 区画データ・セット初期設定ユーティリティー (DFSUPNT0) またはデータベース事前再編成ユーティリティー (DFSURPR0) を使用して区画を初期設定します。パラメーターを変更した後、その区画に対して区画の初期設定が必要なことを示す (PINIT) フラグが設定されます。
7. 区画を再ロードします。
8. タイプ 1 コマンド `/START DB` またはタイプ 2 コマンド `UPDATE DB NAME(`*`partition_name`*`) START(ACCESS)` を実行することにより、その区画を再始動します。

HALDB データベースへの副次索引の追加

既存の HALDB データベースに副次索引を追加することができます。

IMS は、副次索引を作成するためのユーティリティーを提供していません。副次索引を追加する最も簡単な方法は、IBM IMS Index Builder のようなツールを使用す

ることです。 IMS Index Builder は、既存の HALDB データベースを読み取って、それに関する 1 つ以上の副次索引を作成します。

新しく作成された副次索引内のポインターは正確なので、副次索引を追加する際に ILDS に項目を追加することはありません。後で区画が再編成されるときに、IMS がターゲット・セグメント用の項目を ILDS に追加します。

副次索引を追加するには、索引付きデータベース DBD に新しい定義が必要ですが、データベース・データ・セットの変更は必要ありません。

副次索引を手動で HALDB に追加するには、次のようにします。

1. 索引付きデータベース用のアンロード・ファイルを作成します。 HD 再編成アンロード・ユーティリティー (DFSURGU0) またはユーザーが作成したアプリケーション・プログラムを使用できます。
2. 索引付きデータベース DBD に副次索引定義を追加します。
3. 副次索引用の DBD を作成します。
4. 副次索引用の区画を定義します。
5. 副次索引用のデータ・セットを割り振ります。
6. 副次索引の区画を初期設定します。
7. 索引付きデータベースをロードします。これを行うためのプログラムはユーザーが準備する必要があります。そのプログラムは、ステップ 1 で作成されたファイルを読み取ります。索引付きデータベースがロードされると、副次索引の項目が作成されます。副次索引項目の作成はランダム処理で、ロード時間が大幅に増大する可能性があります。

ステップ 1 で DFSURGU0 ユーティリティーを使用した場合、出力ファイルには、ヘッダー・レコード、各セグメントごとに 1 つのレコード、およびトレーラー・レコードが入っています。セグメント・レコードには、セグメント名とセグメント・データが含まれます。ステップ 7 のユーザーのロード・プログラムは、SDFSMAC 内の IMS DFSURGUP マクロで DSECT を使用することにより、このファイル内のレコードをマップすることができます。

関連概念:

『HALDB 区分副次索引の変更』

HALDB 区分副次索引の変更

HALDB 区分副次索引 (PSINDEX) の変更は、他の HALDB データベースの変更と類似しています。つまり、PSINDEX の区画を追加、削除、または変更することができます。

影響を受けた区画をアンロードして初期設定し、再ロードしなければなりません。索引付きデータベースは、副次索引の区画への変更の影響を受けません。

区分化されていない全機能データベースの場合と同じように、副次索引への変更が索引付きデータベースの定義の変更を必要とする場合、索引付きデータベースのアンロードおよび再ロードが必要になる場合があります。

副次索引への変更の例として、サブシーケンス・フィールドの変更があります。サブシーケンス・フィールドを追加または変更する場合、索引付きデータベースの

DBD も変更しなければなりません。索引付きデータベースにサブシーケンス・フィールドが既に存在しているか、または DBD への他の変更を必要としない場合、索引付きデータベースをアンロードして、再ロードする必要はありません。当然ながら、新しい定義を使って副次索引を再作成する必要があります。

索引付きデータベースを再編成しても、その副次索引は再作成されません。再作成するには、IBMIMS Index Builder ツールなどの他の手段を使用する必要があります。あるいは、副次索引を追加するときに使用した手法と類似の手法を使用することができます。

HALDB 区画データ・セット初期設定ユーティリティー (DFSUPNT0) を使用して、PSINDEX 内の区画を、PHDAM または PHIDAM データベース内の区画を初期設定するのと同様に初期設定します。DFSUPNT0 は、PSINDEX 用のリカバリー・ポイントを自動的に生成します。ユーザーが自分の PSINDEX 区画を削除し、再定義し、その後でその区画の PINIT フラグをオフにすると、リカバリー・ポイントは作成されません。

副次索引定義の変更を必要としない変更を索引付きデータベースに加えた場合、副次索引を変更する必要はありません。副次索引をアンロード、再ロード、または再作成する必要はありません。 HALDB の自己回復ポインター方式がこの機能を提供しています。索引付きデータベースのための再ロード・プロセスを実行すると、ILDS が更新されます。移動したターゲット・セグメントを検索するために副次索引ポインターを確実に使用できるようにするために必要なことは、これだけです。

関連タスク:

922 ページの『HALDB データベースへの副次索引の追加』

第 31 章 データベース・タイプの変換

時間が経過してアプリケーションの特性が変化してしまった場合には、別の DL/I アクセス方式に切り替えることにより、パフォーマンスが向上することがあります。

DL/I アクセス方式 (またはデータベースのタイプ) は、一般に以下のような変動要素に基づいて選択されます。

- 実行する必要のある処理の種類 (順次処理、直接処理、またはその両方)
- データの変化の度合い

このトピックでは、アクセス方式を変更するという決定を行ったものとして、以下の事項について説明します。

- 既存の DL/I アクセス方式があるとして、別の DL/I アクセス方式に変換するためには、何を変更する必要があるか。
- この変換をどのようにして行うか。

関連トピックで説明した再編成ユーティリティーを使用して、DL/I アクセス方式を HISAM、HDAM、および HIDAM アクセス方式の間で変更することができます。これには例外が 1 つあります。HDAM データベースの物理レコードがルート・キーの順序で並んでいない限り、HDAM を HISAM または HIDAM に変更することはできません。この例外があるのは、HISAM および HIDAM データベースには、データベース・レコードをルート・キーの順序でロードしなければならないからです。 HD 再編成アンロード・ユーティリティーが HDAM データベースをアンロードするときには、GN 呼び出しを用いてアンロードします。 HDAM データベースを対象として GN 呼び出しを出すと、ランダム化モジュールによってデータベース・レコードが保管されている物理的順序でデータベース・レコードがアンロードされます。順次ランダム化モジュール (物理的なルート・キー・シーケンスでデータベース・レコードをデータベースの中に入れるもの) を使用しない限り、これはルート・キー・シーケンスにはなりません。

関連概念:

801 ページの『第 30 章 データベースの変更』

155 ページの『指定することのできるポインターの種類』

497 ページの『CI とブロックのサイズの決定』

501 ページの『バッファーの数』

490 ページの『HDAM または PHDAM オプションの選択』

489 ページの『使用するランダム化モジュールの決定 (HDAM および PHDAM のみ)』

関連タスク:

487 ページの『フリー・スペースの指定 (HDAM、PHDAM、HIDAM、および PHIDAM のみ)』

496 ページの『HD データベースの論理レコード長の選択』

603 ページの『データベースの最小サイズの見積もり』

データベースの HISAM から HIDAM への変換

データベースを HISAM から HIDAM に変換する作業は数回のステップで実行できますが、いくつかの準備ステップも実行する必要があります。

DL/I アクセス方式を HISAM から HIDAM に変更する前に、以下の事項を行う必要があります。

- HIDAM データベースの中にフリー・スペースを確保するかどうかを決めます。(フリー・スペースとは、データベースの初期ロード時にデータベース・レコードがロードされないスペースです。)

  HISAM とは異なり、HIDAM データベースは、(ESDS または OSAM データ・セットの中の) ある周期のブロックまたは CI、あるいは各ブロックまたは CI の中での一定の割合のスペースを、フリー・スペースとして確保しておくことができます。 こうしておけば、初期ロードの後、このデータベースにデータベース・レコードまたはセグメントを挿入するために、このフリー・スペースを使用することができます。
- データベースで使用するポインターの種類を決定します。 HISAM とは異なり、HIDAM ではデータベースの中のあるセグメントから次のセグメントを指すのに、直接アドレス・ポインターを使用します。
- 先に選定した論理レコード・サイズを再検討します。 HISAM の 1 つの論理レコードは、同一のデータベース・レコードのセグメントしか収容できません。HIDAM においては、1 つの論理レコードは複数のデータベース・レコードからのセグメントを収容することができます。
- 先に選定した CI サイズまたはブロック・サイズを再検討します。 HISAM では、選択した CI またはブロック・サイズは、データベース・レコードの平均サイズの倍数となっています。 HIDAM においては、装置の特性、および予定している処理の種類に基づいて、このサイズを選定すべきです。
- 先に選定したデータベース・バッファー・サイズ、および先に割り振ったバッファーの数を再検討します。 CI サイズまたはブロック・サイズを変更した場合には、その新しいサイズに合わせたバッファーを割り振る必要があります。
- データベース・スペースを計算し直します。それを行う必要があるのは、上述の変更によってデータベース・スペースの必要量が変わってくるからです。

いったんどのような変更を行う必要があるかを決めたら、DL/I アクセス方式を HISAM から HIDAM に変更することができます。これを行うには、次のようにします。

1. 既存の DBD および HD 再編成アンロード・ユーティリティーを用いて、データベースをアンロードします。
2. 必要な変更を反映した DBD を新しくコーディングします。 HIDAM 索引に DBD もコーディングしなければなりません。
3. DBD では指定されていない変更を行う必要がある場合 (例えば、データベース・バッファー・サイズの変更、データベースに割り振られるスペース量の変更など)、これらの変更を行います。

4. 非 VSAM データ・セットの場合は、古いデータベース・スペースを削除して、新しいデータベース・スペースを定義します。 VSAM データ・セットでは、古いクラスターに割り振られているスペースを削除して、新しいクラスターのスペースを定義します。
5. 新しい DBD および HD 再編成再ロード・ユーティリティーを用いて、データベースを再ロードします。データベースを再ロードしたらすぐに、このデータベースのイメージ・コピーを作成することを忘れないでください。

   論理関係または副次索引を使用している場合には、データベースを再ロードする前後で、他のユーティリティーを実行する必要があります。

関連タスク:

711 ページの『再編成ユーティリティーを使用するオフライン再編成』

データベースの HISAM から HDAM への変換

データベースを HISAM から HDAM に変換する作業は数回のステップで実行できますが、いくつかの準備ステップも実行する必要があります。

DL/I アクセス方式を HISAM から HDAM に変更する前に、以下の事項を行う必要があります。

- データベースで使用するポインターの種類を決定します。 HISAM とは異なり、HDAM ではデータベースの中のあるセグメントから次のセグメントを指すのに、直接アドレス・ポインターを使用します。
- 使用するランダム化モジュールを決定します。 HISAM とは異なり、HDAM ではランダム化モジュールが使用されます。ランダム化モジュールとは、データベース・レコードをどこに保管するか決める情報を生成するためのものです。
- 使用する HDAM オプションを決定します。 HISAM とは異なり HDAM データベースは 2 つの部分に別れています。ルート・アドレス可能域とオーバーフロー域です。ルート・アドレス可能域は、すべてのルート・セグメントを収容している区域であり、データベース・レコードの中の従属セグメントの 1 次ストレージ域です。オーバーフロー域は、ルート・アドレス可能域に収まらない従属セグメントを保管するための場所です。ここで説明している HDAM オプションとは、ルート・アドレス可能域に関してユーザーが行う選択に関係するものです。これらは、次のとおりです。
  - データベース・レコードのセグメントを (処理操作を介在させずに) 連続して挿入するときに、ルート・アドレス可能域に入れるデータベース・レコードの最大バイト数
  - ルート・アドレス可能域の中のブロックまたは CI の数
  - ルート・アドレス可能域の中のブロックまたは CI の RAP (ルート・アンカー・ポイント) の数。 (RAP とは、ルート・セグメントを指すフィールドです。)
- 先に選定した論理レコード・サイズを再検討します。 HISAM の 1 つの論理レコードは、同一のデータベース・レコードのセグメントしか収容できません。HDAM では、1 つの論理レコードは複数のデータベース・レコードからのセグメントを収容することができます。さらに、HDAM 論理レコードには、RAP および 2 つのスペース管理フィールド (FSE と FSEAP) が入っています。

- 先に選定した CI サイズまたはブロック・サイズを再検討します。 HISAM では、選択した CI またはブロック・サイズは、データベース・レコードの平均サイズの倍数となっています。 HDAM においては、装置の特性、および予定している処理の種類に基づいて、このサイズを選定してください。
- 先に選定したデータベース・バッファー・サイズ、および先に割り振ったバッファーの数を再検討します。 CI サイズまたはブロック・サイズを変更した場合には、その新しいサイズに合わせたバッファーを割り振る必要があります。
- データベース・スペースを計算し直します。それを行う必要があるのは、上述の変更によってデータベース・スペースの必要量が変わってくるからです。

いったんどのような変更を加える必要があるかを決めたら、DL/I アクセス方式を HISAM から HDAM に変更することができます。これを行うには、次のようにします。

1. 既存の DBD および HD 再編成アンロード・ユーティリティーを用いて、データベースをアンロードします。
2. 必要な変更を反映した DBD を新しくコーディングします。
3. DBD では指定されていない変更を行う必要がある場合 (例えば、データベース・バッファー・サイズの変更、データベースに割り振られるスペース量の変更など)、これらの変更を行います。 HDAM には 1 つのデータ・セットしか必要でありませんが、HISAM には 2 つのデータ・セットが必要です。
4. 非 VSAM データ・セットの場合は、古いデータベース・スペースを削除して、新しいデータベース・スペースを定義します。 VSAM データ・セットでは、古いクラスターに割り振られているスペースを削除して、新しいクラスターのスペースを定義します。
5. 新しい DBD および HD 再編成再ロード・ユーティリティーを用いて、データベースを再ロードします。データベースを再ロードしたらすぐに、このデータベースのイメージ・コピーを作成します。

   論理関係または副次索引を使用している場合には、データベースを再ロードする前に、他のユーティリティーを実行する必要があります。

関連タスク:

711 ページの『再編成ユーティリティーを使用するオフライン再編成』

データベースの HIDAM から HISAM への変換

データベースを HIDAM から HISAM に変換する作業は数回のステップで実行できますが、いくつかの準備ステップも実行する必要があります。

DL/I アクセス方式を HIDAM から HISAM に変更する前に、以下の事項を行う必要があります。

- 先に選定した論理レコード・サイズを再検討します。 HISAM の 1 つの論理レコードは、同一のデータベース・レコードのセグメントしか収容できません。 HIDAM においては、1 つの論理レコードは複数のデータベース・レコードからのセグメントを収容することができます。
- 先に選定した CI サイズまたはブロック・サイズを再検討します。 HIDAM においては、装置の特性、および予定している処理の種類に基づいて、CI サイズ

またはブロック・サイズを選定してください。 HISAM では、このサイズはデータベース・レコードの平均サイズの倍数とします。

- 先に選定したデータベース・バッファー・サイズ、および先に割り振ったバッファーの数を再検討します。 CI サイズまたはブロック・サイズを変更した場合には、その新しいサイズに合わせたバッファーを割り振る必要があります。
- データベース・スペースを計算し直します。それを行う必要があるのは、上述の変更によってデータベース・スペースの必要量が変わってくるからです。

いったんどのような変更を加える必要があるかを決めたら、さらに DL/I アクセス方式を HIDAM から HISAM に変更することができます。これを行うには、次のようにします。

1. 既存の DBD および HD 再編成アンロード・ユーティリティーを用いて、データベースをアンロードします。
2. 必要な変更を反映した DBD を新しくコーディングします。 DBD では直接アドレス・ポインターやフリー・スペースを指定しないことを忘れないでください。HIDAM とは異なり、HISAM ではこれらの使用は許されないからです。また、HIDAM には 2 つの DBD がありますが、HISAM には 1 つの DBD しかありません。
3. DBD では指定されていない変更を行う必要がある場合 (例えば、データベース・バッファー・サイズの変更、データベースに割り振られるスペース量の変更など)、これらの変更を行います。
4. 非 VSAM データ・セットの場合は、古いデータベース・スペースを削除して、新しいデータベース・スペースを定義します。 VSAM データ・セットでは、古いクラスターに割り振られているスペースを削除して、新しいクラスターのスペースを定義します。
5. 新しい DBD および HD 再編成再ロード・ユーティリティーを用いて、データベースを再ロードします。データベースを再ロードしたらすぐに、このデータベースのイメージ・コピーを作成することを忘れないでください。

   論理関係または副次索引を使用している場合は、データベースの再ロードの直前および直後に他のユーティリティーを実行する必要があります。

関連タスク:

711 ページの『再編成ユーティリティーを使用するオフライン再編成』

データベースの HIDAM から HDAM への変換

データベースを HIDAM から HDAM に変換する作業は数回のステップで実行できますが、いくつかの準備ステップも実行する必要があります。

DL/I アクセス方式を HIDAM から HDAM に変更する前に、以下の事項を行う必要があります。

- 先に選定した直接アドレス・ポインターを再検討します。 HIDAM と HDAM の両方で直接アドレス・ポインターを使用しますが、以下のような理由により使用する直接アドレス・ポインターの種類を変更する必要が生じることがあります。
  - アプリケーションの要件の変更のため。

  - ポインターを選定した理由の一部が、使用するデータベース・タイプであったため。例えば、ルート・セグメントの高速順次処理を実現するために、HIDAM データベースのルート・セグメントにおいて物理兄弟逆方向ポインターを使用した場合には、HDAM データベースではこれは何の役にも立ちません。
- 使用するランダム化モジュールを決定します。 HIDAM とは異なり、HDAM ではランダム化モジュールが使用されます。ランダム化モジュールとは、データベース・レコードをどこに保管するか決める情報を生成するためのものです。
- 使用する HDAM オプションを決定します。 HIDAM とは異なり、HDAM データベースには別個の索引データベースはありません。その代わり、データベースが 2 つの部分に別れています。ルート・アドレス可能域とオーバーフロー域です。ルート・アドレス可能域は、すべてのルート・セグメントを収容している区域であり、データベース・レコードの中の従属セグメントの 1 次ストレージ域です。オーバーフロー域は、ルート・アドレス可能域に収まらない従属セグメントを保管するための場所です。ここで説明している HDAM オプションとは、ルート・アドレス可能域に関してユーザーが行う選択に関係するものです。これらは、次のとおりです。
  - データベース・レコードのセグメントを (処理操作を介在させずに) 連続して挿入するときに、ルート・アドレス可能域に入れるデータベース・レコードの最大バイト数
  - ルート・アドレス可能域の中のブロックまたは CI の数
  - ルート・アドレス可能域の中のブロックまたは CI のルート・アンカー・ポイント (RAP) の数
- 先に選定した論理レコード・サイズを再検討します。
- 先に選定した CI サイズまたはブロック・サイズを再検討します。
- 先に選定したデータベース・バッファー・サイズ、および先に割り振ったバッファーの数を再検討します。 CI サイズまたはブロック・サイズを変更した場合には、その新しいサイズに合わせたバッファーを割り振る必要があります。
- データベース・スペースを計算し直します。それを行う必要があるのは、上述の変更によってデータベース・スペースの必要量が変わってくるからです。

どのような変更を加える必要があるかを決めた後、DL/I アクセス方式を HIDAM から HDAM に変更することができます。これを行うには、次のようにします。

1. 既存の DBD および HD 再編成アンロード・ユーティリティーを用いて、データベースをアンロードします。
2. 必要な変更を反映した DBD を新しくコーディングします。おそらく、フリー・スペースは指定しないでしょうが、HDAM オプションは指定することになります。 HIDAM には 2 つの DBD が必要ですが、HDAM には 1 つの DBD しか必要でないということにも注意してください。
3. DBD では指定されていない変更を行う必要がある場合 (例えば、データベース・バッファー・サイズの変更、データベースに割り振られるスペース量の変更など)、これらの変更を行います。 HDAM には 1 つのデータ・セットしか必要でありませんが、HIDAM には 2 つのデータ・セットが必要です。

4. 非 VSAM データ・セットの場合は、古いデータベース・スペースを削除して、新しいデータベース・スペースを定義します。 VSAM データ・セットでは、古いクラスターに割り振られているスペースを削除して、新しいクラスターのスペースを定義します。
5. 新しい DBD および HD 再編成再ロード・ユーティリティーを用いて、データベースを再ロードします。データベースを再ロードしたらすぐに、このデータベースのイメージ・コピーを作成することを忘れないでください。

   論理関係または副次索引を使用する場合には、データベースの再ロードの直前および直後に他のいくつかのユーティリティーを実行する必要があります。

関連概念:

155 ページの『指定することのできるポインターの種類』

関連タスク:

711 ページの『再編成ユーティリティーを使用するオフライン再編成』

データベースの HDAM から HISAM への変換

データベースを HDAM から HISAM に変換する作業は数回のステップで実行できますが、いくつかの準備ステップも実行する必要があります。

DL/I アクセス方式を HDAM から HISAM に変更する前に、以下の事項を行う必要があります。

- 先に選定した論理レコード・サイズを再検討します。 HISAM の 1 つの論理レコードは、同一のデータベース・レコードのセグメントしか収容できません。 HISAM においては、1 つの論理レコードは複数のデータベース・レコードからのセグメントを収容することができます。
- 先に選定した CI サイズまたはブロック・サイズを再検討します。 HDAM においては、装置の特性、および予定している処理の種類に基づいて、CI サイズまたはブロック・サイズを選定してください。 HISAM では、このサイズはデータベース・レコードの平均サイズの倍数とします。
- 先に選定したデータベース・バッファー・サイズ、および先に割り振ったバッファーの数を再検討します。 CI サイズまたはブロック・サイズを変更した場合には、その新しいサイズに合わせたバッファーを割り振る必要があります。
- データベース・スペースを計算し直します。データベース・スペースを計算し直す必要があるのは、上述の変更によってデータベース・スペースの必要量が変わってくるからです。

どのような変更を加える必要があるかを決めた後、DL/I アクセス方式を HDAM から HISAM に変更することができます。 HDAM データベースの中のデータベース・レコードが物理ルート・キー・シーケンスで並んでいるのでない限り、ユーザー自身がアンロード・プログラムおよび再ロード・プログラムを作成しなければならないことに注意してください。ユーザー独自のロード・プログラムを作成する際は、HDAM データベースが論理関係を使用しているならば、情報を削除バイトの中に保持しなければなりません (例えば、データベースの中で論理的に削除されたセグメントが物理的に削除されていないことがあります)。

HDAM から HISAM に変更するには以下のようにします。

1. 既存の DBD および次のプログラムを用いて、データベースをアンロードします。
   - ユーザーのアンロード・プログラム、または
   - データベース・レコードが物理ルート・キー・シーケンスで並んでいる場合は、HD 再編成アンロード・ユーティリティー
2. 必要な変更を反映した DBD を新しくコーディングします。直接アドレス・ポインターや HDAM オプションを指定することにはなりません。
3. DBD では指定されていない変更を行う必要がある場合 (例えば、データベース・バッファー・サイズの変更、データベースに割り振られるスペース量の変更など)、これらの変更を行います。 HDAM には 1 つのデータ・セットしか必要でありませんが、HISAM には 2 つのデータ・セットが必要です。
4. 非 VSAM データ・セットの場合は、古いデータベース・スペースを削除して、新しいデータベース・スペースを定義します。 VSAM データ・セットでは、古いクラスターに割り振られているスペースを削除して、新しいクラスターのスペースを定義します。
5. 新しい DBD および次のプログラムを用いて、データベースを再ロードします。
   - ユーザーのロード・プログラム
   - データベース・レコードが物理ルート・キー・シーケンスで並んでいる場合は、HD 再編成再ロード・ユーティリティー

     データベースを再ロードしたらすぐに、このデータベースのイメージ・コピーを作成することを忘れないでください。

   論理関係または副次索引を使用する場合には、データベースの再ロードの直前および直後に他のいくつかのユーティリティーを実行する必要があります。

関連タスク:

711 ページの『再編成ユーティリティーを使用するオフライン再編成』

データベースの HDAM から HIDAM への変換

データベースを HDAM から HIDAM に変換する作業は数回のステップで実行できますが、いくつかの準備ステップも実行する必要があります。

DL/I アクセス方式を HDAM から HIDAM へ変更する前に、以下の事項を実行する必要があります。

- HIDAM データベースの中にフリー・スペースを確保するかどうかを決めます。(フリー・スペースとは、データベースの初期ロード時にデータベース・レコードがロードされない スペースです。) HIDAM データベースにおいては、(ESDS または OSAM の各データ・セットの中の) ある周期のブロックまたは CI、あるいは各ブロックまたは CI の中のある割合のスペースを、フリー・スペースとして確保しておくことができます。こうしておけば、初期ロードの後、このデータベースにデータベース・レコードまたはセグメントを挿入するために、このフリー・スペースを使用することができます。 HDAM データベースにおいては、通常、HDAM オプションを慎重に選択することにより必要なフリー・スペースを取得します。

- 先に選定した直接アドレス・ポインターを再検討します。 HIDAM と HDAM の両方で直接アドレス・ポインターを使用しますが、以下のような理由により使用する直接アドレス・ポインターの種類を変更する必要が生じることがあります。
  - アプリケーションの要件の変更のため。
  - ポインターを選定した理由の一部が、使用するデータベース・タイプであったため。例えば、ルート・セグメントの高速順次処理を実現するために HIDAM データベースのルート・セグメントにおいて、物理兄弟順方向ポインターと物理兄弟逆方向ポインターを使用しなければならないことがあります。
- 先に選定した論理レコード・サイズを再検討します。
- 先に選定した CI サイズまたはブロック・サイズを再検討します。
- 先に選定したデータベース・バッファー・サイズ、および先に割り振ったバッファーの数を再検討します。 CI サイズまたはブロック・サイズを変更した場合には、その新しいサイズに合わせたバッファーを割り振る必要があります。
- データベース・スペースを計算し直します。データベース・スペースを計算し直す必要があるのは、上述の変更によってデータベース・スペースの必要量が変わってくるからです。

いったんどのような変更を加えるかを決めたら、DL/I アクセス方式を HDAM から HIDAM に変更することができます。 HDAM データベースの中のデータベース・レコードが物理ルート・キー・シーケンスで並んでいるのでない限り、ユーザー自身がアンロード・プログラムおよび再ロード・プログラムを作成しなければならないことに注意してください。ユーザー独自のロード・プログラムを作成する際は、HDAM データベースが論理関係を使用しているならば、情報を削除バイトの中に保持しなければなりません (例えば、データベースの中で論理的に削除されたセグメントが物理的に削除されていないことがあります)。

HDAM から HIDAM に変更するには以下のことを行います。

1. 既存の DBD および次のプログラムを用いて、データベースをアンロードします。
   - ユーザーのアンロード・プログラム、または
   - データベース・レコードが物理ルート・キー・シーケンスで並んでいる場合は、HD 再編成アンロード・ユーティリティー
2. 必要な変更を反映した DBD を新しくコーディングします。 HIDAM 索引に DBD もコーディングしなければなりません。 HDAM オプションは指定しませんが、フリー・スペースは指定することになります。
3. DBD では指定されていない変更を行う必要がある場合 (例えば、データベース・バッファー・サイズの変更、データベースに割り振られるスペース量の変更など)、これらの変更を行います。 HDAM には 1 つのデータ・セットしか必要でありませんが、HIDAM には 2 つのデータ・セットが必要です。
4. 非 VSAM データ・セットの場合は、古いデータベース・スペースを削除して、新しいデータベース・スペースを定義します。 VSAM データ・セットでは、古いクラスターに割り振られているスペースを削除して、新しいクラスターのスペースを定義します。

5. 新しい DBD および次のプログラムを用いて、データベースを再ロードします。
   - ユーザーのロード・プログラム
   - データベース・レコードが物理ルート・キー・シーケンスで並んでいる場合は、HD 再編成再ロード・ユーティリティー

   データベースを再ロードしたらすぐに、このデータベースのイメージ・コピーを作成することを忘れないでください。

   論理関係または副次索引を使用している場合には、データベースを再ロードする前に、他のユーティリティーを実行する必要があります。

関連タスク:

711 ページの『再編成ユーティリティーを使用するオフライン再編成』

HDAM および HIDAM データベースから HALDB への変換

IMS 基本ユーティリティーを用いて、HDAM または HIDAM データベースを HALDB データベースに変換することができます。

HDAM および HIDAM データベースからそれぞれ PHDAM および PHIDAM への変更の前後の論理ビューについては、以下の図を参照してください。

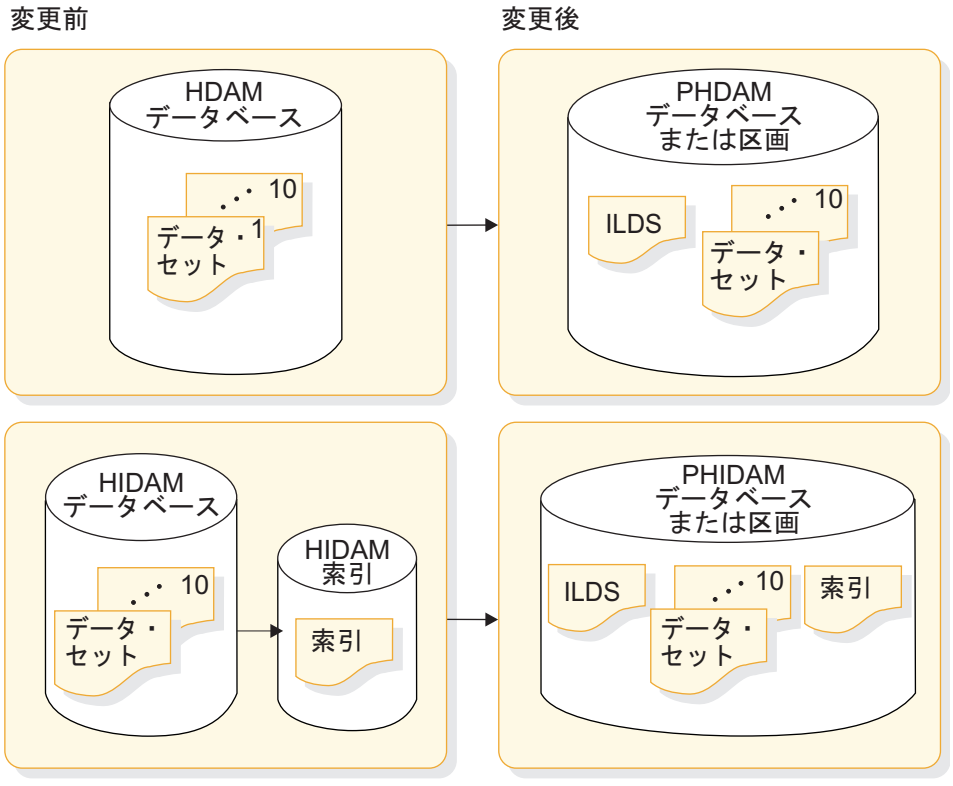


図 *304. HDAM* および *HIDAM* データベースからそれぞれ *PHDAM* および *PHIDAM* への変更の前後

制約事項: 論理関係についての HALDB のサポートには、仮想対のサポートは含まれません。論理的に関連付けられたデータベースを HALDB に変換するときは、以下のようにして論理関係を変換できます。

- 変換されるデータベース内の単一方向関係は、HALDB データベース内でも単一方向のまま残すことができます。
- 変換されるデータベース内の物理対の論理関係は、HALDB データベース内でも物理対のまま残すことができます。
- 変換されるデータベース内の仮想対の論理関係は、HALDB データベース内では物理対に変換することができます。

このセクションでは、データベースを HALDB に変換する基本ステップに加えて、副次索引を持つデータベースを HALDB に変換するステップ、および論理関係を持つデータベースを HALDB に変換するステップを示します。それぞれの作業、および変換処理に関連するその他の概念や事項を、以下のトピックで説明していきます。

関連タスク:

889 ページの『HALDB データベースの変更』

HALDB へのマイグレーションのための並列アンロード

ラージ・データベースを HALDB にマイグレーションする場合、HD アンロード・ユーティリティー (DFSURGU0) を複数の並列ジョブで実行し、それぞれのジョブで指定されたキー範囲のデータベース・レコードをアンロードすることにより、アンロードのパフォーマンスを向上させることができます。

キー範囲のデータベース・レコードのアンロードは、MIGRATE=YES オプションが使用されている場合にのみサポートされます。

マイグレーションのパフォーマンスをさらに向上させるには、1 つのステップで単一パーティションの 1 つのキー範囲をアンロードした後、対応する HALDB 区画のマイグレーション再ロードを実行します。

マイグレーション再ロードでは、DBRC=Y を使用する必要があり、JCL を使用するか、または動的割り振りの場合は DBRC RECON MDA メンバーを使用して、新しい HALDB 区画定義を含んでいる RECON データ・セットを割り振る必要があります。

関連資料:

HD 再編成アンロード・ユーティリティー (DFSURGU0) (データベース・ユーティリティー)

既存のデータベース情報のバックアップ

変換処理を開始する前に、重要なデータベース・エレメントをバックアップします。

既存のデータベースの DBD ステートメントの変更、RECON データ・セットからのデータベース情報の削除、およびデータベースのアンロードと再ロードを行うため、非 HALDB データベース構造の復元が必要となる場合に備えて、これらのエレメントを個々にバックアップする必要があります。

変換処理の開始にあたって、以下のステップを実行して重要なデータベース・エレメントをバックアップします。

- IMS イメージ・コピー・ユーティリティーのいずれか、またはその他の手段を用いて、データベース・データ・セットのイメージ・コピーを作成します。
- DBD を変更する前に、既存のデータベースの DBD ソースをバックアップします。
- RECON データ・セットから情報を削除する前に、RECON データ・セットをバックアップします。RECON データ・セットをバックアップするには 2 つの方法があります。
  - HALDB に変換するデータベースに固有の情報のみを保管するには、DBRC コマンド LIST.DB を実行します。変換後にデータベースを非 HALDB の状態に戻す必要が生じた場合は、RECON データ・セットに非 HALDB データベースの情報を再入力するだけで済みますが、この再入力はおそらく手動で行う必要があります。
  - RECON データ・セット全体をバックアップするには、DBRC コマンド BACKUP.RECON を実行します。変換後にデータベースを非 HALDB の状態に戻す必要が生じた場合は、RECON データ・セットのバックアップ・コピーを復元し、バックアップ・コピー作成後に行われた RECON データ・セットへの変更をすべてリカバリーする必要があります。

  使用するリカバリーの方法に関係なく、データベースを非 HALDB の状態に戻す必要が生じた場合にどのオプションでも選択できるようにするために、LIST.DB コマンドと BACKUP.RECON コマンドの両方を実行することを検討してください。

関連タスク:

957 ページの『論理的に関連した HDAM または HIDAM データベースから HALDB への変換』

943 ページの『副次索引を持つ HDAM または HIDAM データベースから HALDB への変換』

『単純な HDAM または HIDAM データベースから HALDB PHDAM または PHIDAM への変換』

単純な HDAM または HIDAM データベースから HALDB PHDAM または PHIDAM への変換

副次索引や論理関係を持たないデータベースは単純データベース と呼ばれます。データベースが副次索引または論理関係を使用している場合は、それらを変換するための追加のステップが必要です。

データベースを HALDB に変換する前に、データベース、サポートするデータ・セット、および関連する DBD ステートメントをバックアップしてください。

単純な HDAM または HIDAM データベースを HALDB データベースに変換するには、以下に挙げるトピックで詳述しているステップを実行します。

関連タスク:

935 ページの『既存のデータベース情報のバックアップ』

957 ページの『論理的に関連した HDAM または HIDAM データベースから HALDB への変換』

943 ページの『副次索引を持つ HDAM または HIDAM データベースから HALDB への変換』

既存のデータベースのアンロード

IMS HD 再編成アンロード・ユーティリティー (DFSURGU0) を用いて既存のデータベースをアンロードする。

MIGRATE=YES 制御ステートメントを指定します。MIGRATE=YES 制御ステートメントは、このアンロードが HALDB 変換処理の一部であることを示します。IMS HD 再編成アンロード・ユーティリティーでは、非 HALDB データベースを定義する DBD を使用する必要があります。

推奨事項:

- OSAM データベース・データ・セットを使用している場合は、データベースをアンロードする際に OSAM 順次バッファリングを使用してください。OSAM 順次バッファリングを使用すると、一般にアンロードのパフォーマンスが向上します。
- マイグレーションしているデータベースがラージ・データベースであり、MIGRATE 制御ステートメントを使用している場合、DFSURGU0 ユーティリティーでキーの範囲のレコードをアンロードできます。アンロードのパフォーマンスを向上させるため、複数のキー範囲を並列ジョブでアンロードできます。

関連概念:

505 ページの『OSAM 順次バッファリング』

関連資料:

HD 再編成アンロード・ユーティリティー (DFSURGU0) (データベース・ユーティリティー)

RECON データ・セットからのデータベース情報の削除

DBRC コマンド DELETE.DB を用いて、RECON データ・セットからデータベース情報を削除します。

次のステップで新しい HALDB データベースを DBRC に登録する前に、データベース情報を削除しておく必要があります。

HALDB データベースの DBD ステートメントの定義

新しい HALDB の DBD は現行の非 HALDB データベースの DBD に基づくため、データベースを HALDB に変換するには既存の DBD にいくつか変更を加える必要がある。

DBD を変更するには、以下のステートメントを変更する必要があります。

- 変換されるデータベースの DBD ステートメント
- HIDAM 1 次索引の DBD ステートメント
- DATASET ステートメント
- SEGM ステートメント

- LCHILD ステートメント
- FIELD ステートメント
- XDFLD ステートメント

DBD ステートメントの変更

HALDB の DBD ステートメントに対して必要となる可能性がある変更は以下のとおりです。

- 変換後の HALDB データベースのタイプに応じて、ACCESS パラメーターに PHDAM、PHIDAM、または PSINDEX を指定します。
- 区画選択出口ルーチンを使用する場合は、HALDB データベースの DBD ステートメントで PSNAME パラメーターを使用して出口ルーチンのモジュール名を指定します。出口ルーチンを使用しない場合は、PSNAME パラメーターを指定しないでください。

  区画定義処理の中で、区画選択出口ルーチンを挿入または変更することもできます。区画定義処理で区画選択出口ルーチンに対して行った指定は、DBD ステートメントでの指定に優先します。
- PHDAM データベースに変換する場合は、RMNAME パラメーターを指定します。HALDB の場合は、RMNAME パラメーターで HALDB データベース内の全区画に対してデフォルトのランダマイザー値を定義します。各区画を定義する際に、区画ごとに別のランダマイザー値を指定することもできます。

ヒント**:** データベースを HALDB データベースに変換する場合は、できればデータベース名を変えないでください。名前を変更すると、そのデータベースを参照するすべての PSB の変更が必要になります。ここで説明する変換処理では、いずれも変換処理の前と後で同じデータベース名を使用しています。

HIDAM 索引の ***DBD*** ステートメントの省略

データベースが HIDAM データベースである場合は、1 次索引の DBD ステートメントを削除します。PHIDAM 1 次索引は DBD を必要としません。HIDAM データベースが PHIDAM にマイグレーションされると、HIDAM 索引の DBD は廃棄されます。IMS は、PHIDAM 1 次索引の生成に必要な情報を PHIDAM DBD から取得します。

DATASET ステートメントの削除

HALDB データベースでは、DATASET ステートメントは使用されません。既存のデータベースの DBD を変更して HALDB データベースの DBD を作成する場合は、DATASET ステートメントを削除してください。DATASET ステートメントで定義されていたエレメントは、以下の方法で指定することができます。

- 区画を定義するときに DD 名を作成します。
- SEGM ステートメントで、DSGROUP パラメーターを使用してデータ・セット・グループを定義します。
- 区画を定義するときに、フリー・スペースおよび OSAM ブロックのサイズを指定します。

- HALDB データベースの SCAN パラメーターを省略します。HALDB は、セグメント挿入時に、使用可能なストレージ・スペースを現行のシリンダーだけで検索します。

SEGM ステートメントの変更

SEGM ステートメントに次のような変更を加えることが必要な場合があります。

- PTR パラメーターのポインター指定を変更します。

  HALDB では、階層ポインターは使用されません。PTR パラメーターに指定されている HIER、H、HIERBWD、または HB キーワードを、TWIN、T、TWINBWD、または TB に変更する必要があります。

  PHIDAM では、ルート・セグメントに対する兄弟順方向専用 (TWIN または T) ポインターの使用がサポートされません。兄弟順方向専用ポインターを使用する HIDAM ルートがある場合は、SEGM ステートメントで PTR パラメーターのキーワードを NOTWIN、NT、TWINBWD、または TB に変更します。

  HALDB では、シンボリック・ポインターは使用されません。論理関係に関して必要な変更については、 959 ページの『論理的に関連した HALDB データベースに対する DBD ステートメントの定義』で説明します。
- 複数データ・セット・グループを使用する場合は、SEGM ステートメントで DSGROUP パラメーターを指定します。

  複数データ・セット・グループを定義するには、1 番目のデータ・セット・グループにないセグメントの SEGM ステートメントで DSGROUP パラメーターを指定する必要があります。DSGROUP の有効な値は文字 A から J です。A は 1 番目のデータ・セット・グループ、B は 2 番目のデータ・セット・グループ、J は 10 番目のデータ・セット・グループです。
- /SX フィールドをサブシーケンス・フィールドとして使用している場合は、副次索引セグメントの BYTES パラメーターの値を 4 バイト増やす必要があります。詳しくは、 948 ページの『PSINDEX の DBD ステートメントの定義』を参照してください。

LCHILD ステートメントの変更

LCHILD ステートメントでは、PHIDAM データベースの 1 次索引が定義されません。HIDAM DBD を PHIDAM に変換する場合は、LCHILD ステートメントを削除します。

副次索引 DBD を HALDB PSINDEX DBD に変換する場合は、以下の変更が必要となることがあります。

- 副次索引で PTR=SYMB が指定されている場合は、その指定を PTR=SNGL に変更するか、または PTR= キーワードを省略する必要があります。PTR=SNGL はデフォルトです。PSINDEX の LCHILD ステートメントでは、この指定のみが有効です。
- HDAM または HIDAM 索引付きデータベースに PTR=SYMB が指定されている場合は、その指定を PTR=INDX に変更する必要があります。

- データベースに副次索引がある場合は、副次索引 DBD の LCHILD ステートメントで RKSIZE パラメーターを使用して、ターゲット・データベース内のルート・セグメントのキー・サイズを指定する必要があります。

シンボリック・ポインターを使用して論理関係を変換する場合は、論理関係の LCHILD ステートメントで PTR=SYMB の指定を省略する必要があります。

FIELD ステートメントの変更

仮想対を使用する論理関係を定義するために使用されている LCHILD ステートメントの変更については、 959 ページの『論理的に関連した HALDB データベースに対する DBD ステートメントの定義』で説明します。

/SX フィールドがサブシーケンス・フィールドとして使用されている場合は、副次索引 DBD で、副次索引セグメントのシーケンス・フィールドに対する FIELD ステートメントの BYTES パラメーターの値を 4 バイト増やす必要があります。/SX フィールドは、索引付きデータベース DBD で定義されます。詳しくは、 943 ページの『副次索引を持つ HDAM または HIDAM データベースから HALDB への変換』を参照してください。

XDFLD ステートメントの変更

HALDB は、共用副次索引をサポートしません。索引付きデータベースの XDFLD ステートメントに CONST パラメーターが指定されている場合は、それを削除する必要があります。CONST パラメーターは、共用副次索引に関連付けられる文字を指定します。HALDB 副次索引は、各索引専用の副次索引データベースに保管する必要があります。HALDB に変換する共用副次索引ごとに別々の PSINDEX データベースを定義する必要があります。

関連タスク:

959 ページの『論理的に関連した HALDB データベースに対する DBD ステートメントの定義』

947 ページの『PSINDEX で索引付けされた HALDB データベースの DBD ステートメントの定義』

DBRC による HALDB マスター・データベースの登録

HALDB マスター・データベースは、DBRC に登録する必要があります。

HALDB マスター・データベースを DBRC に登録するには、区画定義ユーティリティーまたは DBRC バッチ・コマンド INIT.DB を使用します。

DBRC に対する区画の定義

HALDB 区画は、DBRC RECON データ・セットで定義されます。

区画を定義するときには、RECON データ・セットに対する更新権限が必要です。

DBRC に対して区画を定義するには、区画定義ユーティリティーまたは DBRC バッチ・コマンド `INIT.PART` を使用します。

データベース・データ・セットの割り振り

データベースで使用されるデータベース・データ・セットを割り振る必要があります。

データ・セット名は、区画を定義するときに指定したデータ・セット名接頭部と、データ・セット名アルゴリズムから作成されます。データ・セットを割り振るときには、この名前を使用する必要があります。

各区画に、間接リスト・データ・セット (ILDS) をはじめとするデータベース・データ・セットを割り振ります。

データ・セットの大きさを決定する際には、HALDB マイグレーション・エイド・ユーティリティー (DFSMAID0) で生成される情報が役立ちます。このユーティリティーからは、各区画で既存のセグメント・データが使用しているバイト数が報告されます。追加するセグメント、挿入するフリー・スペース、およびビットマップのスペースを確保するために追加のバイト数を加算する必要があります。

HALDB にマイグレーションする際に、フリー・スペース・パラメーターを増やしたい場合があります。非 HALDB データベースでは、データ・セット・サイズの制限のために、フリー・スペースの値が十分でない場合があります。フリー・スペースを増やすと、データベースや区画を再編成する頻度が低くなります。場合によっては、フリー・スペースを増やすことで再編成が完全に不要となる可能性もあります。

論理関係または副次索引を持たないデータベースは ILDS を使用しませんが、いずれにしても HALDB で各区画に ILDS が必要となる場合があります。オンライン IMS システムは、データベースで必要なければ ILDS を割り振りませんが、バッチ・ジョブが ILDS を割り振るため、ILDS を定義しておく必要があります。空の ILDS がオープンされると IMS からメッセージ IEC161I 152-061 が出されますが、このメッセージは問題を通知するものではありません。

副次索引または論理関係が定義されていなければ、HD 再編成再ロード・ユーティリティー (DFSURGL0) は ILDS を更新しません。

ILDS を使用しない場合は、データ・セット用にごく少量のスペース (1 トラックなど) を割り振ることができます。

どの ILDS も、そのレコードは 50 バイトの固定長です。キーはゼロ・オフセットで 9 バイトです。以下の例は、ILDS を割り振るための IDCAMS DEFINE ステートメントを示しています。

```
DEFINE CLUSTER(                    -
   NAME(JOUKO3.HALDB.DB.PEOPLE.L00001) -
   INDEXED                         -
   CYL(1 1)                        -
   RECORDSIZE(50 50)               -
   SHAREOPTIONS(3 3)               -
   SPEED                           -
   KEY(9,0)                        -
   FREESPACE(10,10)                -
   CONTROLINTERVALSIZE(8192)       -
   VOLUMES(TOTIMN)                 -
)
```

ILDS 以外のすべての HALDB VSAM データ・セットには、DEFINE ステートメントで REUSE を指定する必要があります。

PHIDAM 1 次索引を割り振るときは、DBDGEN ユーティリティーの出力が役立ちます。PHIDAM データベースに対する DBDGEN ユーティリティーの出力では、IDCAMS (データ操作ユーティリティー) 定義に必要なパラメーターが、「RECOMMENDED VSAM DEFINE CLUSTER PARAMETERS」というラベルの下にリストされます。必要なパラメーターは、INDEXED パラメーター、RECORDSIZE パラメーターの値、REUSE パラメーター、および KEY パラメーターの値です。以下の例は、PHIDAM 1 次索引データ・セットを割り振るための IDCAMS DEFINE ステートメントを示しています。

```
DEFINE CLUSTER(                    -
   NAME(JOUKO3.HALDB.DB.RZL.X00001) -
   INDEXED                         -
   CYL(10 5)                       -
   RECORDSIZE(20 20)               -
   SHAREOPTIONS(3 3)               -
   REUSE                           -
   KEY(14,5)                       -
   FREESPACE(25,10)                -
   CONTROLINTERVALSIZE(8192)       -
   VOLUMES(TOTIMN)                 -
)
```

関連概念:

29 ページの『HALDB 区画、DD 名、およびデータ・セットのための命名規則』

関連タスク:

963 ページの『論理的に関連したデータベース・データ・セットの割り振り』

954 ページの『索引付きデータベース・データ・セットの割り振り』

区画の初期設定

HALDB 区画を使用する前に、区画を初期設定する必要があります。

区画を初期設定するには、IMS データベース事前再編成ユーティリティー (DFSURPR0) または IMS HALDB データベース・データ・セット初期設定ユーティリティー (DFSUPNT0) を使用します。または、IBM IMS High Performance Load ツールを使用して区画を初期設定することもできます。

関連概念:

197 ページの『HALDB 区画の初期設定』

HALDB データベースとしてのデータベースのロード

HD 再編成再ロード・ユーティリティー (DFSURGL0) を用いて、データベースを HALDB データベースとして再ロードします。

HD 再編成再ロード・ユーティリティーへの入力となるのは、HD 再編成アンロード・ユーティリティーからの出力です。HD 再編成再ロード・ユーティリティーを実行するときに ILDSINGLE 制御ステートメントを指定すると、ILDS に確実にフリー・スペースが挿入されます。

データベース・データ・セットのイメージ・コピー

再ロード処理で、ILDS および PHIDAM 1 次索引以外の各データベース・データ・セットに、イメージ・コピーが必要 であることを示すフラグが設定されます。

フラグが設定されたデータ・セットのイメージ・コピーを作成します。

DFSMDA メンバーおよび HIDAM 1 次索引 DBD のクリーンアップ

HALDB データベースは、動的割り振りに DFSMDA メンバーを使用せず、また、PHIDAM データベースの 1 次索引用に別の DBD を必要としません。

HALDB データベースは、DBRC RECON データ・セットに保管されている情報を使用してデータベース・データ・セットの動的割り振りを行うため、DFSMDA メンバーは使用しません。変換処理が完了し、データベースを非 HALDB の状態に戻す必要がないことが確実となった時点で、DFSMDA メンバーは削除してかまいません。

IMS は、PHIDAM 1 次索引の生成に必要な情報を PHIDAM DBD から取得します。 HIDAM データベースを HALDB PHIDAM データベースに変換した場合は、変換処理が完了してデータベースを非 HALDB の状態に戻す必要がないことが確実となった時点で、古い HIDAM データベースの 1 次索引の DBD は廃棄してかまいません。

関連タスク:

965 ページの『DFSMDA メンバーおよび HIDAM 1 次索引 DBD のクリーンアップ』

957 ページの『PSINDEX に変換した後の DFSMDA メンバーおよび HIDAM 1 次索引 DBD のクリーンアップ』

副次索引を持つ HDAM または HIDAM データベースから HALDB への変換

区分副次索引 (PSINDEX) は、HALDB データベースがサポートする唯一の副次索引タイプです。副次索引を持つ HDAM または HIDAM データベースを HALDB PHDAM または PHIDAM データベースに変換する場合は、同時に副次索引データベースを HALDB PSINDEX データベースに変換する必要があります。

データベース・データ・セットのイメージ・コピーを作成し、RECON データ・セットと既存の DBD 定義をバックアップします。

ターゲットの HDAM または HIDAM データベースとその副次索引を HALDB に変換するためのステップは、 936 ページの『単純な HDAM または HIDAM データベースから HALDB PHDAM または PHIDAM への変換』で詳しく説明しているステップと基本的には同じですが、副次索引の変換と ILDS へのデータの追加に関連する変更があります。

PSINDEX データベースには区画の境界を定義する必要があります。既存の副次索引に対して HALDB マイグレーション・エイド・ユーティリティー (DFSMAID0) を実行すると、PSINDEX の区画化方式を決定する際に役立ちます。

変換処理に IMS の基本ユーティリティーのみを使用する場合は、以下の 2 とおりの方法でデータベースとその副次索引を変換できます。

- HD 再編成アンロード・ユーティリティー (DFSURGU0) でターゲット・データベースをアンロードするときに副次索引出力ファイルを作成するには、HD 再編成アンロード・ユーティリティー実行時に MIGRATX=YES 制御ステートメントを指定します。MIGRATX=YES 制御ステートメントを指定すると、ユーティリティーは副次索引を読み取らずに、索引付きデータベースのターゲット・セグメントから副次索引出力ファイルを作成します。MIGRATX=YES 制御ステートメントを使用する方法では、副次索引内のユーザー・データが維持されません。
- 副次索引をスタンドアロン・データベースとして変換する場合は、索引付きデータベースと各副次索引に対して HD 再編成アンロード・ユーティリティーを実行するときに MIGRATE=YES 制御ステートメントを指定します。この方法では、HD 再編成アンロード・ユーティリティーが各副次索引を個別に読み取ってアンロードすることが必要です。このユーティリティーは、副次索引を読み取るときに索引付きデータベースも読み取る必要があります。

  ここでは、副次索引を HALDB に変換する場合に MIGRATE=YES 制御ステートメントを指定する方法については説明しません。

推奨事項**:** MIGRATX=YES を指定する方法を使用してください。

関連タスク:

935 ページの『既存のデータベース情報のバックアップ』

936 ページの『単純な HDAM または HIDAM データベースから HALDB PHDAM または PHIDAM への変換』

957 ページの『論理的に関連した HDAM または HIDAM データベースから HALDB への変換』

既存のデータベースのアンロード

HD 再編成アンロード・ユーティリティーを MIGRATX=YES 制御ステートメントと共に用いて、既存のターゲット・データベースをアンロードします。MIGRATX=YES 制御ステートメントを指定すると、索引付きデータベースのアンロード・ファイルに加えて、副次索引のアンロード・ファイルが作成されます。

MIGRATX=YES 制御ステートメントを用いてデータベースをアンロードすると、HD 再編成アンロード・ユーティリティーは副次索引を読み取りません。副次索引データ・セットの DD ステートメントは不要です。副次索引の項目を作成するために必要な情報は、ソース・セグメントから生成されます。副次索引のアンロード・ファイルは、副次索引のロードに使用する前にソートする必要があります。HD 再編成アンロード・ユーティリティーの出力には、副次索引のアンロード・ファイルをソートするためのソート制御ステートメントが含まれています。

MIGRATX=YES 制御ステートメントを使用する場合は、副次索引のユーザー・データが維持されませんが、それが問題となることはまれです。ほとんどのシステム環境では副次索引のユーザー・データが維持されません。これは、非 HALDB データベースの再編成で副次索引の再作成が必要となるたびにユーザー・データが失われるためです。

以下の図は、HD 再編成アンロード・ユーティリティーを MIGRATX=YES 制御ステートメントと共に用いて、データベースとその 2 つの副次索引をマイグレーションする場合を示しています。

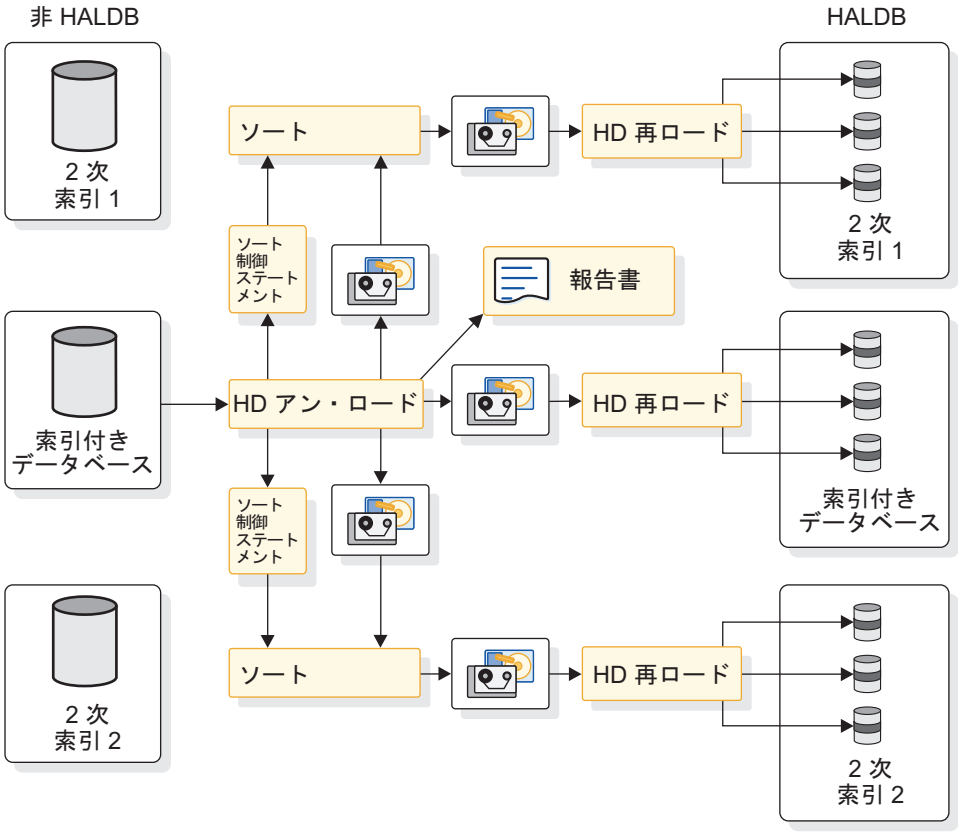


図 *305. MIGRATX=YES* 制御ステートメントを使用した、2 つの副次索引を持つデータベースのマイグレーション

OSAM データベース・データ・セットを使用する場合は、OSAM 順次バッファリングを使用することによってアンロード処理のパフォーマンスが向上します。

HD 再編成アンロード・ユーティリティーの SYSPRINT リストには、作業ファイル統計報告書が示されます。HD 再編成アンロード・ユーティリティーは、各副次索引の出力ファイルに DFSWORK1 や DFSWORK2 などの名前を割り当てます。作業ファイル統計報告書では、各副次索引に関連する出力ファイル名、ソート・ファイル名、およびその他のデータが 1 行にリストされます。この報告書には各出力ファイルと各副次索引の対応関係が示されます。この報告書を使用して、この関係を把握しておく必要があります。ユーティリティーは、アンロード処理で最初に検出した副次索引関係に DFSWRK01 という名前を割り当てます。報告書には、副次索引ファイル内のセグメント数、およびソート・フィールドのサイズとオフセットもリストされます。以下の例に示す報告書のサンプルは、作業ファイル DFSWRK01 に副次索引 STUNUM のレコード、作業ファイル DFSWRK02 に副次索引 STUCRT のレコードが入っていることを示しています。

```
W O R K   F I L E   S T A T I S T I C S

SINAME      WFNAME      SFNAME       RCDTOTAL          OFFSET    LENGTH
```

```
STUNUM      DFSWRK01    DFSSRT01     000087012         0124      0006

STUCRT      DFSWRK02    DFSSRT02     000010702         0124      0010
```

HD 再編成アンロード・ユーティリティーの出力のソート

HD 再編成アンロード・ユーティリティーの出力をソートする方法は、副次索引で固有キーを作成するときに /SX フィールドを使用するかどうかによって異なります。

固有キーの作成に */SX* フィールドを使用しない場合の出力のソート

/SX フィールドを使用していない場合は、DFSSRTnn DD ステートメントで定義されているデータ・セットで、HD 再編成アンロード・ユーティリティーによって生成されたソート制御ステートメントを用いて、副次索引出力データ・セットをソートします。

ソートは副次索引出力ファイルごとに行う必要があります。以下の例は、STUNUM 副次索引をソートする JCL のサンプルを示しています。SORTIN DD では、HD 再編成アンロード・ユーティリティーからの DFSWRK01 出力を指定しています。SYSIN DD では、HD 再編成アンロード・ユーティリティーからの DFSSRT01 出力を指定しています。このソートの出力は、STUNUM 副次索引に対する HD 再編成再ロード・ユーティリティー (DFSURGL0) への入力となります。

```
//*******************************************************************
//*   SORT THE STUNUM SEC. INDEX RECORDS FROM MIGRATX=YES
//*******************************************************************
//*
//SORT01   EXEC PGM=SORT,REGION=2048K,PARM='CORE=MAX'
//SORTLIB  DD DSN=SYS1.SORTLIB,DISP=SHR
//SYSOUT   DD SYSOUT=*
//SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,(50,5),,CONTIG)
//SORTWK02 DD UNIT=SYSDA,SPACE=(CYL,(50,5),,CONTIG)
//SORTWK03 DD UNIT=SYSDA,SPACE=(CYL,(50,5),,CONTIG)
//SYSIN    DD DSN=JOUKO3.STUDENT.MIGR.SRT01,DISP=OLD
//SORTIN   DD DSN=JOUKO3.STUDENT.MIGR.WRK01,DISP=OLD
//SORTOUT  DD DSN=JOUKO3.STUDENT.MIGR.WRK01.SORTED,DISP=(NEW,CATLG),
//         UNIT=3390,VOL=SER=TOTIMN,
//         SPACE=(CYL,(50,10),RLSE)
```

固有キーの作成に */SX* フィールドを使用する場合の出力のソート

固有キーの作成に /SX フィールドを使用している場合は、以下のステップを実行して、副次索引アンロード・レコードを正しいサブシーケンスの順序で配置します。

1. ファイルを、ヘッダー・レコード、アンロード・レコード、およびトレーラー・レコードが入った 3 つの別々のファイルに分割します。以下の例は、このステップのソート JCL と制御ステートメントを示しています。

   注**:** 次の例は、MIGRATX=YES が指定された場合に使用されます。MIGRATE=YES が使用された場合、トレーラー・レコードの OUTFIL INCLUDE ステートメントには、X'0290' でなく X'0090' が含まれます。

   ```
   //SORTIN     DD  DSN=UNLOAD.OUTPUT,DISP=SHR
   //HEADER     DD  DSN=HEADER.FILE,DISP=(NEW,PASS)
   //TRAILER    DD  DSN=TRAILER.FILE,DISP=(NEW,PASS)
   //ULCOPY     DD  DSN=UNLOAD.COPY,DISP=(NEW,PASS)
   //SYSIN      DD   *
   OPTION     COPY
   ```

```
OUTFIL INCLUDE=(5,2,CH,EQ,X'0080'),FNAMES=HEADER
OUTFIL INCLUDE=(5,2,CH,EQ,X'0290'),FNAMES=TRAILER
OUTFIL SAVE,FNAMES=ULCOPY
RECORD TYPE=V
```

2. アンロード・レコードが入ったファイルを、/SX サブシーケンスの順序でソートします。ソート・フィールドのオフセットとサイズを計算する必要があります。オフセットは、63 バイトに索引付きデータベースのルート・キーのサイズを加算して算出します。ソート・フィールドのサイズは、副次索引検索フィールドのサイズに 8 バイトを加算して算出します。以下の例は、このステップのソート JCL と制御ステートメントを示しています。この例では、オフセットが 73 バイト、ソート・フィールドのサイズが18 バイトになっています。

```
//SORTIN   DD DSN=UNLOAD.COPY,DISP=SHR
//SORTOUT  DD DSN=UNLOAD.SORTED1,DISP=(,CATLG),
//            UNIT=SYSDA,VOL=SER=000000,SPACE=(CYL,(1,5))
//SYSIN    DD  *
SORT FIELDS=(73,18,CH,A),FILSZ=E1000
RECORD TYPE=V
END
```

3. レコードを 1 つのファイルにマージします。以下の例は、このステップのソート JCL と制御ステートメントを示しています。

```
//SORTIN DD DSN=UNLOAD.HEADER,DISP=(OLD,DELETE),
//          UNIT=SYSDA,VOL=SER=000000,SPACE=(CYL,(1,5))
//       DD DSN=UNLOAD.SORTED1,DISP=(OLD,DELETE),
//          UNIT=SYSDA,VOL=SER=000000,SPACE=(CYL,(1,5))
//       DD DSN=UNLOAD.TRAILER,DISP=(OLD,DELETE),
//          UNIT=SYSDA,VOL=SER=000000,SPACE=(CYL,(1,5))
//SORTOUT DD DSN=UNLOAD.SORTED2,DISP=(NEW,KEEP),
//          UNIT=SYSDA,VOL=SER=000000,SPACE=(CYL,(1,5))
//SYSOUT   DD  SYSOUT=A
//SYSIN    DD  *
OPTION    COPY
END
```

RECON データ・セットからのデータベース情報の削除

次のステップで新しい HALDB データベースを DBRC に登録する前に、RECON データ・セットから古いデータベース情報を削除する必要があります。

DBRC コマンド DELETE.DB を用いて、RECON データ・セットからデータベース情報を削除します。

PSINDEX で索引付けされた HALDB データベースの DBD ステートメントの定義

索引付き HALDB マスター・データベースの属性は、DBD ステートメントを使用して定義します。

索引付き HALDB データベースの DBD ステートメントを定義するには、以下の DBD ステートメントを変更します。

- DBD ステートメント
- DATASET ステートメントを削除
- SEGM ステートメント
- LCHILD ステートメント
- FIELD ステートメント

- データベースが共用副次索引を使用している場合は、副次索引を HALDB に変換する前に、索引付きデータベースの XDFLD ステートメントから CONST パラメーターを削除します。

関連タスク:

937 ページの『HALDB データベースの DBD ステートメントの定義』

PSINDEX の DBD ステートメントの定義

PSINDEX マスター・データベースの属性は、DBD ステートメントを使用して定義します。

PSINDEX に対して以下の DBD ステートメントを定義します。

- DBD ステートメント

  ACCESS=PSINDEX を指定して、このデータベースを HALDB 区分副次索引として区別します。
- SEGM ステートメント
- LCHILD ステートメント

  PSINDEX に対して RKSIZE パラメーターを指定する必要があります。

  非 HALDB 副次索引でシンボリック・ポインターを使用していた場合は、索引付きデータベース DBD の PTR=SYMB の指定を PTR=INDX に変更し、副次索引 DBD で PTR パラメーターを省略するか、その DBD で PTR=SNGL を指定する必要があります。
- XDFLD ステートメント
- FIELD ステートメント

  既存の副次索引で非固有キーを使用している場合は、/SX フィールドを使用して PSINDEX に固有キーを作成することができます。HALDB PSINDEX には固有キーが必要です。

副次索引データベースを HALDB 副次索引としてロードする前に、そのデータベースの DBD を変更する必要があります。また、一部の索引付きデータベースの DBD を変更しなければならない場合もあります。

シンボリック・ポインターの除去

HALDB 副次索引はシンボリック・ポインターをサポートしていないため、シンボリック・ポインターを使用している場合は除去する必要があります。

シンボリック・ポインターは、非 HALDB 副次索引の LCHILD ステートメントおよび索引付きデータベースの DBD の、PTR=SYMB パラメーターで定義されています。索引付きデータベース DBD の PTR=SYMB の指定を PTR=INDX に変更し、副次索引 DBD で PTR パラメーターを省略するか、その DBD で PTR=SNGL を指定する必要があります。

以下の一連の図の例では、DBD ステートメントからシンボリック・ポインターを除去するために必要な変更を示しています。関係する DBD ステートメントは以下のとおりです。

- シンボリック・ポインターが定義されている非 HALDB 副次索引の DBD。
- 索引付き HDAM の DBD。
- HALDB に変換された後の副次索引の DBD。
- HALDB に変換された後の索引付きデータベースの DBD。

次の図は、シンボリック・ポインターが定義されている非 HALDB 副次索引の DBD を示しています。

図 *306.* 例*:* シンボリック・ポインターを使用する副次索引の *DBD*

```
DBD    NAME=CONTRSI,ACCESS=INDEX
DATASET DD1=CONTSI,DEVICE=3390,SIZE=8192
SEGM  NAME=CONTR,BYTES=26,PARENT=0
FIELD NAME=(CONTRNUM,SEQ,U),BYTES=8,START=1,TYPE=C
LCHILD NAME=(CONTRACT,ENGAGEM),INDEX=CONTRIDX,PTR=SYMB
DBDGEN
FINISH
END
```

次の図は、索引付き HDAM の DBD を示しています。

図 *307.* 例*:* シンボリック・ポインターを使用する索引付きデータベースの *DBD*

```
DBD    NAME=ENGAGEM,ACCESS=HDAM,RMNAME=(DFSHDC40,1,500,824)
DATASET DD1=ENGAHDAM,BLOCK=1648,SCAN=0
SEGM  NAME=CLIENT,BYTES=100,PTR=TWIN
FIELD NAME=(CLNUM,SEQ,U),BYTES=10,START=1,TYPE=C
SEGM  NAME=CONTRACT,PARENT=CLIENT,BYTES=60,PTR=TWIN
FIELD NAME=(CONTRNO,SEQ,U),BYTES=8,START=1,TYPE=C
LCHILD NAME=(CONTR,CONTRSI),PTR=SYMB
XDFLD NAME=CONTRIDX,SRCH=CONTRNO
DBDGEN
FINISH
END
```

次の図は、HALDB に変換された後の副次索引の DBD を示しています。LCHILD ステートメントから PTR=SYMB が削除されています。同じステートメントに RKSIZE が追加されています。シンボリック・ポインターは、索引セグメントのデータ域に保管されています。HALDB 副次索引内にはシンボリック・ポインターがないため、SEGM セグメントの BYTES パラメーターは、シンボリック・ポインターのサイズの分だけ小さくなっています。

図 *308.* 例*: HALDB* 副次索引の *DBD*

```
DBD    NAME=CONTRSI,ACCESS=PSINDEX
SEGM  NAME=CONTR,BYTES=8,PARENT=0
FIELD NAME=(CONTRNUM,SEQ,U),BYTES=8,START=1,TYPE=C
LCHILD NAME=(CONTRACT,ENGAGEM),INDEX=CONTRIDX,RKSIZE=10
DBDGEN
FINISH
END
```

次の図は、HALDB に変換された後の索引付きデータベースの DBD を示しています。LCHILD セグメントに、副次索引関係を表す PTR=INDX が指定されていま

す。

図 *309. 例: HALDB* 索引付きデータベースの *DBD*

```
DBD   NAME=ENGAGEM,ACCESS=PHDAM,RMNAME=(DFSHDC40,1,500,824)
SEGM  NAME=CLIENT,BYTES=100,PTR=TWIN
FIELD NAME=(CLNUM,SEQ,U),BYTES=10,START=1,TYPE=C
SEGM  NAME=CONTRACT,PARENT=CLIENT,BYTES=60,PTR=TWIN
FIELD NAME=(CONTRNO,SEQ,U),BYTES=8,START=1,TYPE=C
LCHILD NAME=(CONTR,CONTRSI),PTR=INDX
XDFLD NAME=CONTRIDX,SRCH=CONTRNO
DBDGEN
FINISH
END
```

シンボリック・ポインターをデータベースとして使用して副次索引を処理するアプリケーションがある場合は、DBD にほかの変更を加える必要があります。

非固有の副次索引キーから固有キーへの変換

HALDB 副次索引は固有キーを持たなければなりません。

副次索引が非固有キーを使用している場合は、以下のステップを実行してそれらを固有キーに変換します。

1. 索引付きデータベースの DBD で XDFLD ステートメントに /SX システム関連フィールドを追加することで、サブシーケンス・フィールドを追加します。このフィールドによって、索引のシーケンス・フィールドの長さが 8 バイト増えます。以下の例では、この変換で FIELD ステートメントの BYTES パラメーターが 8 バイト増えています。
2. FIELD ステートメントで、NAME パラメーターの 3 番目のサブパラメーターの値を M から U に変更します。U パラメーターは、セグメントのシーケンス・フィールドが固有値のみを受け入れることを示します。

以下の例は、非固有キーを持つ非 HALDB 副次索引の DBD を示しています。

図 *310. 例:* 非固有キーを持つ非 *HALDB* 副次索引の *DBD*

```
DBD     NAME=XSI3,ACCESS=INDEX
DATASET DD1=XSI301,OVFLW=XSI302
SEGM    NAME=XSNAM,BYTES=6,PARENT=0
FIELD   NAME=(XSNAME,SEQ,M),START=1,BYTES=6
LCHILD  NAME=(PERF,XPER01),INDEX=NAMX1,POINTER=SNGL
DBDGEN
FINISH
END
```

次の図は、HALDB に変換された後の副次索引の DBD を示しています。

図 *311. 例:* 固有キーを持つ *HALDB* 副次索引の *DBD*

```
DBD     NAME=XSI3,ACCESS=PSINDEX
SEGM    NAME=XSNAM,BYTES=14,PARENT=0
FIELD   NAME=(XSNAME,SEQ,U),START=1,BYTES=14
```

```
LCHILD  NAME=(PERF,XPER01),INDEX=NAMX1,POINTER=SNGL,RKSIZE=12
DBDGEN
FINISH
END
```

より大きな ***HALDB /SX*** フィールドに対する ***PSINDEX DBD*** の変更

HALDB に変換する副次索引の固有キーを作成するためにすでに /SX フィールドを使用している場合は、SEGM ステートメントと FIELD ステートメントの両方で BYTES パラメーターを 4 バイト増やします。非 HALDB 副次索引では、/SX フィールドに 4 バイトの RBA が入ります。HALDB 副次索引では、/SX フィールドに 8 バイトの ILK が入ります。

索引付き DBD の XDFLD ステートメントまたは /SX FIELD ステートメントを変更する必要はありません。/SX FIELD ステートメントに BYTES パラメーターは不要で、指定しても無視されます。

以下の例は、/SX フィールドが定義されている索引付き HDAM データベースの DBD を示しています。

図 *312. 例: /SX* フィールドが定義されている *HDAM* の *DBD*

```
DBD   NAME=VEHICLE,ACCESS=(HDAM,OSAM),                          X
      RMNAME=(DFSHDC40,2,500,)
DATASET DD1=VEHICLE1,BLOCK=1648,SCAN=0
SEGM  NAME=AUTO,BYTES=54,PTR=TB
FIELD NAME=(ID,SEQ,U),BYTES=10,START=1
FIELD NAME=MAKE,BYTES=20,START=11
FIELD NAME=MODEL,BYTES=20,START=31
FIELD NAME=YEAR,BYTES=4,START=51
FIELD NAME=/SX1
LCHILD NAME=(MAKEMOD,VEHSI),PTR=INDX
XDFLD NAME=MMIDX,SRCH=(MAKE,MODEL),SUBSEQ=/SX1
DBDGEN
FINISH
END
```

以下の例は、上記の図で定義されている /SX フィールドを使用する非 HALDB 副次索引の DBD を示しています。

図 *313. 例: /SX* フィールドを使用する非 *HALDB* 副次索引の *DBD*

```
DBD    NAME=VEHSI,ACCESS=INDEX
DATASET DD1=VEHSI1,DEVICE=3390,SIZE=8192
SEGM  NAME=MAKEMOD,BYTES=44,PARENT=0
FIELD NAME=(NAMES,SEQ,U),BYTES=44,START=1
LCHILD NAME=(AUTO,VEHICLE),INDEX=MMIDX
DBDGEN
FINISH
END
```

以下の例は、HALDB に変換された後の、図 312 の DBD を示しています。変更された点は、通常 HDAM DBD が PHDAM に変換されるときに加えられる変更だけです。つまり、DBD ステートメントの ACCESS パラメーターが PHDAM に変更され、DATASET ステートメントが削除されています。

図 *314. 例: /SX* フィールドが定義されている *PHDAM* の *DBD*

```
DBD   NAME=VEHICLE,ACCESS=(PHDAM,OSAM),                       X
      RMNAME=(DFSHDC40,2,500,)
SEGM  NAME=AUTO,BYTES=54,PTR=TB
FIELD NAME=(ID,SEQ,U),BYTES=10,START=1
FIELD NAME=MAKE,BYTES=20,START=11
FIELD NAME=MODEL,BYTES=20,START=31
FIELD NAME=YEAR,BYTES=4,START=51
FIELD NAME=/SX1
LCHILD NAME=(MAKEMOD,VEHSI),PTR=INDX
XDFLD NAME=MMIDX,SRCH=(MAKE,MODEL),SUBSEQ=/SX1
DBDGEN
FINISH
END
```

以下の例は、HALDB に変換された後の、 951 ページの図 313 の DBD を示しています。他の副次索引 DBD の変換と同様に、DBD ステートメントの ACCESS パラメーターが PSINDEX に変更され、DATASET ステートメントが削除され、LCHILD ステートメントに RKSIZE パラメーターが追加されています。/SX フィールドはサブシーケンス・フィールドとして使用されるため、SEGM ステートメントと FIELD ステートメントの BYTES パラメーターに 4 が加算されています。

図 *315. 例: /SX* フィールドを使用する *HALDB* 副次索引の *DBD*

```
DBD    NAME=VEHSI,ACCESS=PSINDEX
SEGM  NAME=MAKEMOD,BYTES=48,PARENT=0
FIELD NAME=(NAMES,SEQ,U),BYTES=48,START=1
LCHILD NAME=(AUTO,VEHICLE),INDEX=MMIDX,RKSIZE=10
DBDGEN
FINISH
END
```

索引付き HALDB マスター・データベースの DBRC への登録

索引付き HALDB マスター・データベースは、DBRC に登録する必要があります。

区画定義ユーティリティーまたは DBRC バッチ・コマンド INIT.DB を用いて、HALDB マスター・データベースを DBRC に登録します。

DBRC に対する索引付きデータベースの区画の定義

HALDB 区画定義が DBRC RECON データ・セットに保管される。

区画を定義するときには、RECON データ・セットに対する更新権限が必要です。

区画定義ユーティリティーまたは DBRC バッチ・コマンド INIT.PART を用いて、DBRC に対して区画を定義します。

DBRC による PSINDEX HALDB マスター・データベースの登録

索引付き PSINDEX HALDB マスター・データベースは、DBRC に登録する必要があります。

区画定義ユーティリティーまたは DBRC バッチ・コマンド INIT.DB を用いて、HALDB PSINDEX を DBRC に登録します。

DBRC に対する PSINDEX データベースの区画の定義

PSINDEX の HALDB 区画定義が DBRC RECON データ・セットに保管されます。

HALDB PSINDEX のレコードは、非 HALDB 副次索引のレコードに比べてはるかに大きい場合が少なくありません。これには 2 つの理由があります。

- PSINDEX レコードでは異なるポインターが使用されます。各 PSINDEX レコードには、28 バイトの拡張ポインター・セット (EPS) が含まれています。非 HALDB 副次索引レコードに含まれているのは、4 バイトの直接ポインターまたはシンボリック・ポインターです。シンボリック・ポインターの長さは連結キーの長さです。
- PSINDEX レコードにはターゲット・セグメントのルート・キーが保管されます。非 HALDB 副次索引レコードにはルート・キーが保管されません。HALDB 副次索引区画を割り振るときには、このサイズの増加を考慮に入れる必要があります。

副次索引全体のサイズは、通常は容易に見積もることができます。副次索引の項目はすべて固定長です。この長さは、DBDGEN ユーティリティーから報告されます。HALDB データベースに対する DBDGEN ユーティリティーの出力で、「RECOMMENDED VSAM DEFINE CLUSTER PARAMETERS」の下の「RECORDSIZE」に合計の長さが示されます。項目の数が変換で変わることはありません。現行の項目数は、複数のユーティリティーの出力で確認できます。再編成処理の一環として副次索引を再作成するときに、HISAM 再ロード・ユーティリティーまたは使用する任意のツールで、この情報が報告されます。

あるいは、z/OS の LISTCAT コマンドの REC-TOTAL 値を使用して、既存の副次索引データ・セット内のレコード数を確認することもできます。

さらに、HALDB マイグレーション・エイド・ユーティリティーでもレコードの数を確認できます。レコードの数と長さ、およびフリー・スペースの所要量を使用して、HALDB 副次索引全体のサイズを見積もることができます。例えば、レコード長が 48 バイトの項目が 100 万件あり、その 25% をフリー・スペースとして追加で確保する場合、索引に必要なサイズは 6000 万バイトです。

各区画に必要なサイズは、区画内の索引項目の数によって決まります。このサイズは、HALDB マイグレーション・エイド・ユーティリティー (DFSMAID0) の出力またはその他の手段によって見積もることができます。キー範囲での副次索引項目の数がわかっていれば、マイグレーション・エイド・ユーティリティーを使用する必要はありません。そうでない場合は、このユーティリティーを使用してください。

HALDB マイグレーション・エイド・ユーティリティーでは、キー範囲内のレコード数とキー範囲の境界に関する正確な情報が得られます。セグメントまたは区画に必要なバイト数については、正確な情報が得られません。このため、副次索引を読み取るときに MAX 制御ステートメントを使用しないでください。必要な区画の数がわかっている場合は、NBR 制御ステートメントを使用します。必要なハイ・キーがわかっている場合は、KR 制御ステートメントを使用します。NBR ステートメントを使用すると、各区画の正確なハイ・キーが報告されます。KR 制御ステートメントを使用すると、各区画の正確なセグメント数が報告されます。

以下の出力例は、副次索引に対する HALDB マイグレーション・エイド・ユーティリティーの出力の一部を示しています。この出力は NBR=4 制御ステートメントを指定して生成されています。区画 1、2、3 に関する報告はここでは省略されています。区画 4 およびデータベース全体に関する報告のみが示されています。
「segments」列で報告されているセグメントの数は正確です。区画 4 には 158518 個の副次索引セグメントが入り、索引全体では 634078 個のセグメントが入ることになります。「bytes」列と「prefix-incr」列の情報は、副次索引に関する同様の報告書と同じように、正確ではありません。ただし、区画内のセグメントの数さえわかれば、区画のスペース所要量は計算できます。

```
partition 4 :

   minimum key =

      +0000  f0f0f1f6 f0f8f1f1 f5f0f0f0 f0f0f1f3  |0016081150000013|
      +0010  f1f7                                |17              |

   maximum key =

      +0000  f0f0f2f1 f1f0f3f0 f0f0f0f0 f0f0f0f5  |0021103000000005|
      +0010  f4f6                                |46              |

                      segments         bytes   prefix-incr   length-incr
   1) 'ORDRCUST'         158518       3804432       1268144             0
 SUM)                    158518       3804432       1268144             0

--------------------------------------------------------------------------------

sum of partitions:
                      segments         bytes   prefix-incr   length-incr
   1) 'ORDRCUST'         634078      15217872       5072624             0
 SUM)                    634078      15217872       5072624             0


--------------------------------------------------------------------------------
```

DBDGEN ユーティリティーから報告されるレコード・サイズ、および区画のセグメント数を使用して、区画に必要なサイズを見積もることができます。必要に応じて、フリー・スペースおよび区画の拡張に備えたスペースを追加することができます。

重要**:** 区画に必要なサイズを見積もる際に、HALDB マイグレーション・エイド・ユーティリティーで報告される副次索引の「bytes」および「prefix-incr」の数値は使用しないでください。これらの数値は正確ではありません。報告が正確でない理由は、副次索引のセグメント接頭部が索引付きデータベース内のセグメントの接頭部より大きいためです。HALDB マイグレーション・エイド・ユーティリティーは、副次索引向けの調整を行いません。HALDB マイグレーション・エイド・ユーティリティーは、各区画のハイ・キーおよび副次索引項目の数を確認するために使用します。

索引付きデータベース・データ・セットの割り振り

索引付きデータベースの各区画にデータベース・データ・セットを割り振ります。

割り振る必要があるデータ・セットは、ILDS と 1 次索引です (1 次索引は HALDB データベースが PHIDAM データベースである場合のみ)。

関連概念:

29 ページの『HALDB 区画、DD 名、およびデータ・セットのための命名規則』

関連タスク:

『PSINDEX VSAM KSDS データ・セットの割り振り』
941 ページの『データベース・データ・セットの割り振り』

PSINDEX VSAM KSDS データ・セットの割り振り

IDCAMS DEFINE ステートメントを使用して、PSINDEX の各区画に VSAM KSDS データ・セットを割り振ります。

PSINDEX 区画にデータ・セットを割り振る際には、以下のパラメーターが必要です。

- キー値
- RECORDSIZE 値
- INDEXED
- REUSE

副次索引データ・セットを割り振るときは、DBDGEN ユーティリティーの出力が役立ちます。副次索引に対する出力では、IDCAMS 定義に必要なパラメーターが、「RECOMMENDED VSAM DEFINE CLUSTER PARAMETERS」の下にリストされます。以下の例は、副次索引データ・セットを割り振るための IDCAMS DEFINE ステートメントを示しています。

```
DEFINE CLUSTER(                    -
   NAME(JOUKO3.HALDB.DB.PEOSKSI.A00001) -
   INDEXED                         -
   RECORDSIZE(86 86)               -
   CYL(20 5)                       -
   SHAREOPTIONS(3 3)               -
   REUSE                           -
   KEY(29,50)                      -
   FREESPACE(10,10)                -
   CONTROLINTERVALSIZE(4096)       -
   VOLUMES(TOTIMN)                 -
)
```

関連タスク:

963 ページの『論理的に関連したデータベース・データ・セットの割り振り』
954 ページの『索引付きデータベース・データ・セットの割り振り』

区画の初期設定

HALDB 区画を使用する前に、区画を初期設定する必要があります。

区画を初期設定するには、IMS データベース事前再編成ユーティリティー (DFSURPR0) または IMS HALDB データベース・データ・セット初期設定ユーティリティー (DFSUPNT0) を使用します。または、IBM IMS High Performance Load ツールを使用して区画を初期設定することもできます。

関連概念:

197 ページの『HALDB 区画の初期設定』

ILDS 更新方法の選択

副次索引のターゲット・データベースをロードする際に ILDS データ・セットを更新する方法には 3 つのオプションがあります。選択するオプションは、ロード処理のパフォーマンスに影響する。

ILDS の更新方法を選択するには、制御ステートメントの指定に関して以下のいずれかを使用します。

- HD 再編成再ロード・ユーティリティー (DFSURGL0) によって各ターゲット・セグメントが索引付きデータベースにロードされるときに ILDS を更新する場合は、HD 再編成再ロード・ユーティリティー実行時に ILDS 制御ステートメントを指定しないようにします。これは最も時間のかかる方法です。
- HD 再編成再ロード・ユーティリティーで、索引付きデータベースが再ロードされた後に ILDS を更新する場合は、HD 再編成再ロード・ユーティリティー実行時に ILDSMULTI 制御ステートメントを指定します。この処理ではマルチスレッドの更新処理が使用されます。これは以下の理由から、ターゲット・セグメントがロードされるときに項目を更新する場合に比べてはるかに効率的です。
  - 複数の異なる区画に対して ILDS 項目の書き込みが並行して行われます。
  - 書き込みが順次です。
  - 書き込みがロード・モードで行われるので、CI および CA 分割のオーバーヘッドがありません。

  さらに、フリー・スペースを作成することができます。このフリー・スペースがあると、将来の再編成で ILDS を更新する際に役立つ場合があります。

  ILDSMULTI 制御ステートメントは、データベースを HALDB に変換する場合にのみ有効です。
- 索引付きデータベースが再ロードされた後に、HD 再編成再ロード・ユーティリティー以外の手段で ILDS を更新する場合は、NOILDS 制御ステートメントを指定します。このオプションを選択した場合は、HALDB 索引/ILDS 再作成ユーティリティー (DFSPREC0) を用いて ILDS を更新できます。

  NOILDS 制御ステートメントを使用すると最速で再ロードを実行でき、HALDB 索引/ILDS 再作成ユーティリティーによって各区画の ILDS を並行して更新できます。ただし、HALDB 索引/ILDS 再作成ユーティリティーは、区画を 1 つずつ読み取ってその ILDS を更新します。オプションとして、HALDB 索引/ILDS 再作成ユーティリティーでは ILDS を VSAM ロード・モードで再作成することができます。これにより、パフォーマンスが向上し、ILDS にフリー・スペースが組み込まれます。

  NOILDS 制御ステートメントは、HD 再編成再ロード・ユーティリティーで既存の HALDB データベースを再編成するときにも使用できます。

推奨事項**:** 2 次索引に多数の項目があるデータベースを HALDB に変換する場合は、HD 再編成再ロード・ユーティリティーでデータベースを再ロードするときに ILDSMULTI 制御ステートメントを用いてください。

関連タスク:

957 ページの『索引付きデータベースとその副次索引のロード』

索引付きデータベースとその副次索引のロード

索引付きデータベースとその副次索引をロードするには、HD 再編成再ロード・ユーティリティー (DFSURGL0) を使用する。

HD 再編成再ロード・ユーティリティーへの入力となるのは、HD 再編成アンロード・ユーティリティーからのソートされた出力です。

索引付きデータベースとその副次索引のロードについては、ILDS の更新方法の選択以外に特別な考慮事項はありません。

マイグレーション再ロードを行うと、副次索引内のポインターは正確でなくなるため、ポインターが修復されるまで IMS はポインターを使用して索引付きデータベース内のターゲット・セグメントを検索できません。ただし、ポインターを修復するためにユーザーが何らかの作業を行う必要はありません。IMS は、ポインターを修復するために、ターゲット・セグメントが必要となるまで待機し、ターゲット区画の ILDS を通じてターゲット・セグメントを検索する際に副次索引内のターゲット・セグメントのポインターを更新します。

関連タスク:

956 ページの『ILDS 更新方法の選択』

イメージ・コピーの作成

再ロード処理で、ILDS および PHIDAM 1 次索引以外の各データ・セットに、イメージ・コピーが必要 であることを示すフラグが設定されます。

フラグが立てられたデータベース・データ・セットと PSINDEX データ・セットのイメージ・コピーを作成します。

PSINDEX に変換した後の DFSMDA メンバーおよび HIDAM 1 次索引 DBD のクリーンアップ

変換処理が完了し、データベースを非 HALDB の状態に戻す必要がないことが確実となった時点で、古いデータベースの DFSMDA メンバーを削除し、HIDAM 1 次索引の DBD を廃棄してかまいません。

関連タスク:

943 ページの『DFSMDA メンバーおよび HIDAM 1 次索引 DBD のクリーンアップ』

論理的に関連した HDAM または HIDAM データベースから HALDB への変換

論理的に関連した HDAM または HIDAM データベースを HALDB PHDAM または PHIDAM に変換する処理は、単純データベースを HALDB に変換する場合とほぼ同じです。ただし、HALDB で論理関係をサポートするには、論理関係を変更する必要があるほか、論理的に関連したすべてのデータベースを同時に HALDB に変換する必要があります。

データベース・データ・セットのイメージ・コピーを作成し、RECON データ・セットと既存の DBD 定義をバックアップします。

このセクションでは、論理的に関連したデータベースを HALDB に変換するステップに加えて、変換処理を開始する前に考慮すべき概念および問題について説明します。

論理的に相互に関連したデータベースは、同じタイミングで HALDB に変換する必要があります。IMS は、HALDB と非 HALDB データベース間の論理関係をサポートしていません。

HALDB でサポートされる論理関係は、単一方向および物理対の両方向の 2 種類だけです。HALDB は仮想対をサポートしていません。仮想対を使用する両方向論理関係は、物理対を使用する両方向論理関係に変換する必要があります。

HALDB では、シンボリック・ポインターまたは階層ポインターは使用されません。HALDB PHIDAM データベースでは、兄弟順方向専用ポインター (TWIN または T) の使用がサポートされません。

論理データベースが参照している物理データベースをマイグレーションする際に、論理データベースの DBD を変更する必要はありません。論理データベースの DBD では、DBD ステートメントに ACCESS=LOGICAL が指定されています。

関連タスク:

935 ページの『既存のデータベース情報のバックアップ』

936 ページの『単純な HDAM または HIDAM データベースから HALDB PHDAM または PHIDAM への変換』

943 ページの『副次索引を持つ HDAM または HIDAM データベースから HALDB への変換』

既存のデータベースのアンロード

HD 再編成アンロード・ユーティリティー (DFSURGU0) を適切な制御ステートメントとともに使用して、既存のデータベースをアンロードします。

DFSURGU0 ユーティリティーとともに使用できる制御ステートメントは、次のとおりです。

- 論理的に関連したデータベースが副次索引を使用していない場合は、MIGRATE=YES 制御ステートメントを使用します。
- 論理的に関連したデータベースが副次索引を使用している場合は、MIGRATX=YES 制御ステートメントを用いて副次索引用のアンロード・ファイルを作成します。また、 943 ページの『副次索引を持つ HDAM または HIDAM データベースから HALDB への変換』に記載されているステップも実行する必要があります。

HALDB に変換するためにアンロードするデータベースに論理子がある場合は、そのデータベースと論理的に関連したデータベースも読み取られることがあります。HD 再編成アンロード・ユーティリティーは、以下のいずれかの条件が当てはまる場合に、論理的に関連したデータベースを読み取ります。

- 論理関係で物理対が使用されている。
- VIRTUAL オプションが指定されている (論理親の連結キーが論理子に保管されていない)。
- 論理関係でシンボリック・ポインターが使用されている。

- アンロードされるデータベースに仮想論理子が含まれている。

言い替えると、論理的に関連したデータベースが読み取られないのは、2 つの場合に限られます。それらは以下のとおりです。

- アンロードされるデータベースに、直接ポインターを使用する仮想対関係の実論理子が含まれ、かつ論理親の連結キーがこの論理子に保管されている。
- 論理関係が単一方向で、かつ直接ポインターを使用し、さらに論理親の連結キーが論理子に保管されている。

HD 再編成アンロード・ユーティリティーが論理的に関連したデータベースを読み取るときに、各論理子の論理親またはその論理子と対をなす論理子が読み取られます。つまり、論理関係のインスタンスごとに、関連データの読み取りが必要となります。この読み取りはランダムに行われます。各読み取りで物理的な入出力が必要となることがあるため、アンロードの時間が大幅に長くなるおそれがあります。

アンロードの際に読み取られる論理的に関連したデータベースには、バッファー・プールを用意する必要があります。HD 再編成アンロード・ユーティリティーの DFSVSAMP DD ステートメントに、これらのバッファーの定義を追加する必要があります。

以下の図は、論理的に関連した 2 つのデータベースを HALDB データベースにマイグレーションする処理を示しています。

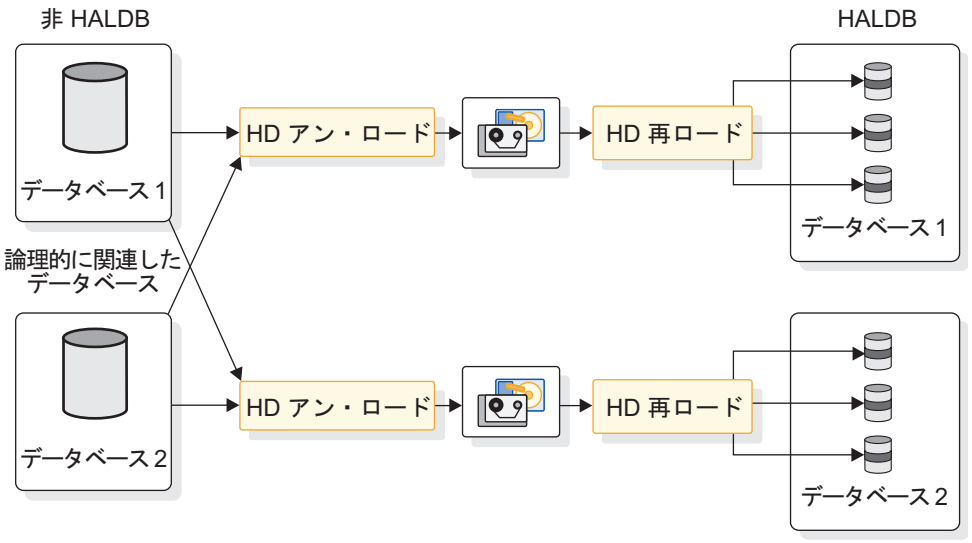


図 *316.* 論理関係を持つデータベースのマイグレーション

論理的に関連した HALDB データベースに対する DBD ステートメントの定義

論理的に関連した HALDB データベースごとに DBD ステートメントを定義する。

変更が必要なステートメントは以下のとおりです。

- DBD ステートメント

- SEGM ステートメント
- LCHILD ステートメント
- XDFLD ステートメント
- FIELD ステートメント

HALDB データベースでは、DATASET ステートメントは使用されません。

論理関係のためのポインター・オプションの変更

論理的に関連したデータベースでは、論理関係を持たないデータベースに必要な変更に加えて、ポインターへの変更が必要となる場合があります。

HALDB では、シンボリック・ポインターは使用されません。シンボリック・ポインターで実装されているすべての論理関係を変更する必要があります。そのためには、論理子に対する SEGM ステートメントの PTR パラメーターに LPARNT キーワードを追加します。

シンボリック・ポインターは使用されませんが、PARENT パラメーターで使用するキーワードに関係なく、論理親の連結キーは論理子に保管されます。つまり、VIRTUAL または V を指定しても、PHYSICAL または P が使用されます。VIRTUAL または V を指定した場合に出される警告メッセージが表示されないようにするには、PHYSICAL または P キーワードを指定する必要があります。

仮想対の認識

DBD を調べることで、仮想対があるかどうかを確認できます。HDAM または HIDAM データベースに SEGM ステートメントがあり SOURCE パラメーターが指定されていれば、そのセグメントは仮想論理子です。SOURCE パラメーターの 1 番目のサブパラメーターは、実論理子を示します。3 番目のサブパラメーターは、実論理子が存在するデータベースを示します。論理関係を持つ HIDAM DBD を示した以下の例では、DBD の NAMESKIL セグメントに SOURCE パラメーターが指定されています。これは仮想論理子です。

このソース・パラメーターは、 961 ページの図 318 の SKILLINV データベース内の SKILNAME セグメントを参照しており、このデータベースでは SKILNAME が実論理子となっています。

図 *317. 例:* 論理関係を持つ *HIDAM* の *DBD*

```
DBD    NAME=PAYROLDB,ACCESS=HIDAM
DATASET DD1=PAYHIDAM,BLOCK=1648,SCAN=3
SEGM   NAME=NAMEMAST,PTR=TWINBWD,RULES=(VVV),                X
      BYTES=150
LCHILD NAME=(INDEX,INDEXDB),PTR=INDX
LCHILD NAME=(SKILNAME,SKILLINV),PAIR=NAMESKIL,PTR=DBLE
FIELD  NAME=(EMPLOYEE,SEQ,U),BYTES=60,START=1,TYPE=C
SEGM   NAME=NAMESKIL,PARENT=NAMEMAST,PTR=PAIRED,             X
      SOURCE=((SKILNAME,DATA,SKILLINV))
FIELD  NAME=(TYPE,SEQ,U),BYTES=21,START=1,TYPE=C
DBDGEN

FINISH

END
```

図 *318.* 例: 論理関係を持つ *HDAM* の *DBD*

```
DBD    NAME=SKILLINV,ACCESS=HDAM,RMNAME=(DFSHDC40,1,500,824)
DATASET DD1=SKILHDAM,BLOCK=1648,SCAN=0
SEGM  NAME=SKILMAST,BYTES=31,PTR=TWINBWD
FIELD NAME=(TYPE,SEQ,U),BYTES=21,START=1,TYPE=C
SEGM  NAME=SKILNAME,                                          X
      PARENT=((SKILMAST,DBLE),(NAMEMAST,P,PAYROLDB)),         X
      BYTES=80,PTR=(LPARNT,LTWINBWD,TWIN)
FIELD NAME=(EMPLOYEE,SEQ,U),START=1,BYTES=60,TYPE=C
DBDGEN
FINISH
END
```

仮想対から物理対への変換

論理的に関連した既存のデータベースが仮想対を使用している場合、それらのデータベースを HALDB に変換するには DBD ステートメントを変更する必要があります。

仮想対の論理関係を物理対に変換するには、以下のステップを実行します。

1. 実論理子であったセグメントの親に LCHILD ステートメントを追加します。 960 ページの図 317 では、SKILMAST セグメントがこれに該当します。この LCHILD ステートメントを以下のように変更します。
   - 新しい LCHILD ステートメントに、SOURCE パラメーターが指定されていたセグメントを指定する NAME パラメーターを追加します。 960 ページの図 317 では、NAMESKIL がこれに該当します。
   - PAIR パラメーターで、SOURCE パラメーターに指定されていたセグメントを指定します。これは実論理子であったセグメントです。 960 ページの図 317 では、SKILNAME がこれに該当します。
2. 実論理子であったセグメントを参照する LCHILD ステートメントを探します。 960 ページの図 317 では、NAME=(SKILNAME,SKILLINV) が指定されている PAYROLDB データベースの LCHILD ステートメントがこれに該当します。この LCHILD ステートメントを以下のように変更します。
   - PTR パラメーターを削除します。
   - RULES パラメーターがある場合は削除します。
3. 仮想論理子を定義していた SEGM ステートメントを探します。 960 ページの図 317 では、NAMESKIL セグメントがこれに該当します。この SEGM ステートメントを以下のように変更します。
   - BYTES パラメーターを追加します。対をなす論理子では、固定交差データの長さが同じでなければなりません。BYTES パラメーターには、固定交差データと論理親の連結キー (LPCK) の両方を指定します。このセグメントの BYTES 値を計算するには、対をなす論理子の SEGM ステートメントの BYTES 値に対し、このセグメントの LPCK サイズを加算し、対をなす論理子の LPCK サイズを減算します。 960 ページの図 317 で NAMESKIL セグメントと対をなしている論理子は、SKILNAME セグメントです。そのサイズは 80 バイトです。NAMESKIL セグメントの LPCK は、SKILMAST セグメントの TYPE フィールドです。そのサイズは 21 バイトです。SKILNAME セグメントの LPCK は、NAMEMAST セグメントの

EMPLOYEE フィールドです。そのサイズは 60 バイトです。したがって NAMESKIL セグメントの BYTES パラメーターは 41 となります (80 + 21 - 60)。

- SOURCE パラメーターを削除します。
- PARENT パラメーターを変更して、このパラメーターがさらにその論理親を参照するようにします。 960 ページの図 317 では、SKILMAST セグメントがこれに該当します。PARENT パラメーターには、PHYSICAL または P キーワードも指定する必要があります。
- PTR パラメーターの指定を変更します。PAIRED を指定します。TWIN、T、TWINBWD、TB、NOTWIN、または NT を指定します。親の下にこのセグメント・タイプのインスタンスを複数作成する場合は、NOTWIN と NT を使用しないでください。

4. 実論理子であったセグメントの SEGM ステートメントを探します。 960 ページの図 317 では、SKILNAME セグメントがこれに該当します。この SEGM ステートメントを以下のように変更します。
   - PARENT パラメーターを変更して、SNGL と DBLE が指定されていない状態にします。 VIRTUAL または V が指定されている場合は、これを PHYSICAL または P に変更します。
   - PTR パラメーターの指定を変更します。LTWIN、LT、LTWINBWD、または LTB キーワードが指定されている場合は削除します。LPARNT キーワードと PAIRED キーワードを指定します。

図 319 と 図 320 は、 960 ページの図 317 と 961 ページの図 318 の DBD が物理対を使用する HALDB データベースに変換された後を示しています。論理関係に関する変更のほかに、HALDB への変換に関するその他の変更が加えられています。つまり、DBD ステートメントの ACCESS パラメーターが変更され、DATASET ステートメントが削除されています。

図 *319. 例:* 論理関係を持つ *PHIDAM* の *DBD*

```
DBD   NAME=PAYROLDB,ACCESS=PHIDAM
SEGM  NAME=NAMEMAST,PTR=TWINBWD,RULES=(VVV),                    X
      BYTES=150
LCHILD NAME=(SKILNAME,SKILLINV),PAIR=NAMESKIL
FIELD NAME=(EMPLOYEE,SEQ,U),BYTES=60,START=1,TYPE=C
SEGM  NAME=NAMESKIL,                                            X
      PARENT=((NAMEMAST),(SKILMAST,P,SKILLINV)),                X
      BYTES=41,PTR=(TWIN,PAIRED)
FIELD NAME=(TYPE,SEQ,U),BYTES=21,START=1,TYPE=C
DBDGEN
FINISH
END
```

図 *320. 例:* 論理関係を持つ *PHDAM* の *DBD*

```
DBD   NAME=SKILLINV,ACCESS=PHDAM,RMNAME=(DFSHDC40,1,500,824)
SEGM  NAME=SKILMAST,BYTES=31,PTR=TWINBWD
FIELD NAME=(TYPE,SEQ,U),BYTES=21,START=1,TYPE=C
LCHILD NAME=(NAMESKIL,PAYROLDB),PAIR=SKILNAME
SEGM  NAME=SKILNAME,                                            X
      PARENT=((SKILMAST),(NAMEMAST,P,PAYROLDB)),                X
```

```
      BYTES=80,PTR=(TWIN,PAIRED)
FIELD NAME=(EMPLOYEE,SEQ,U),START=1,BYTES=60,TYPE=C
DBDGEN
FINISH
END
```

関連タスク:

937 ページの『HALDB データベースの DBD ステートメントの定義』

RECON データ・セットからのデータベース情報の削除

RECON データ・セットから、論理的に関連したデータベースそれぞれのデータベース情報を削除する。

この情報は、次のステップで新しい HALDB データベースを登録する前に削除する必要があります。データベース情報を削除するには、DBRC コマンド DELETE.DB を使用します。

DBRC への各 HALDB マスター・データベースの登録

区画定義ユーティリティーまたは DBRC バッチ・コマンド INIT.DB を用いて、HALDB マスター・データベースを DBRC に登録します。

DBRC に対する区画の定義

DBRC RECON データ・セットで HALDB 区画を定義します。

区画を定義するときには、RECON データ・セットに対する更新権限が必要です。

区画定義ユーティリティーまたは DBRC バッチ・コマンド INIT.PART を用いて、DBRC に対して区画を定義します。

論理的に関連したデータベース・データ・セットの割り振り

データベース・データ・セットを割り振る。これには、各区画の間接リスト・データ・セットおよび PHIDAM データベースの 1 次索引が含まれる。

関連概念:

29 ページの『HALDB 区画、DD 名、およびデータ・セットのための命名規則』

関連タスク:

941 ページの『データベース・データ・セットの割り振り』

955 ページの『PSINDEX VSAM KSDS データ・セットの割り振り』

区画の初期設定

HALDB 区画を使用する前に、区画を初期設定する必要があります。

データベース事前再編成ユーティリティー (DFSURPR0) または HALDB 区画データ・セット初期設定ユーティリティー (DFSUPNT0) を用いて、データベース内の区画を初期設定します。

関連概念:

197 ページの『HALDB 区画の初期設定』

ILDS 更新方法の選択

論理的に関連したデータベースをロードする際に、3 つの方法のいずれかを使用して ILDS を更新することができます。

選択するオプションは、ロード処理のパフォーマンスに影響します。論理的に関連したデータベースのロードは、他のデータベースよりもはるかに時間がかかるため、選択するオプションがパフォーマンスに大きな影響を与える場合もあります。

ILDS を更新するには、以下のいずれかの方法を使用します。

- HD 再編成再ロード・ユーティリティー (DFSURGL0) によって、単一方向関係の各論理親または両方向関係の各論理子がデータベースにロードされるときに ILDS を更新する場合は、HD 再編成再ロード・ユーティリティー実行時に ILDS 制御ステートメントを指定しないようにします。これは最も時間のかかる方法です。
- HD 再編成再ロード・ユーティリティーで、論理的に関連したデータベースが再ロードされた後に ILDS を更新する場合は、HD 再編成再ロード・ユーティリティー実行時に ILDSMULTI 制御ステートメントを指定します。この処理ではマルチスレッドの更新処理が使用されます。これは以下の理由から、ターゲット・セグメントがロードされるときに項目を更新する場合に比べてはるかに効率的です。
  - 複数の異なる区画に対して ILDS 項目の書き込みが並行して行われます。
  - 書き込みが順次です。
  - 書き込みがロード・モードで行われるので、CI および CA 分割のオーバーヘッドがありません。

  さらに、フリー・スペースを作成することができます。このフリー・スペースがあると、将来の再編成で ILDS を更新する際に役立つ場合があります。

  ILDSMULTI 制御ステートメントは、データベースを HALDB に変換する場合にのみ有効です。
- 論理的に関連したデータベースが再ロードされた後に、HD 再編成再ロード・ユーティリティー以外の手段で ILDS を更新する場合は、NOILDS 制御ステートメントを指定します。このオプションを選択した場合は、HALDB 索引/ILDS 再作成ユーティリティー (DFSPREC0) を用いて ILDS を更新できます。

  NOILDS 制御ステートメントを使用すると最速で再ロードを実行でき、HALDB 索引/ILDS 再作成ユーティリティーによって各区画の ILDS を並行して更新できます。ただし、HALDB 索引/ILDS 再作成ユーティリティーは、区画を 1 つずつ読み取ってその ILDS を更新します。オプションとして、HALDB 索引/ILDS 再作成ユーティリティーでは ILDS を VSAM ロード・モードで再作成することができます。これにより、パフォーマンスが向上し、ILDS にフリー・スペースが組み込まれます。

  NOILDS 制御ステートメントは、HD 再編成再ロード・ユーティリティーで既存の HALDB データベースを再編成するときにも使用できます。

関連タスク:

965 ページの『HALDB データベースとしての各データベースのロード』

HALDB データベースとしての各データベースのロード

HD 再編成再ロード・ユーティリティー (DFSURGL0) を用いて、論理関係を持つ HALDB データベースをロードする。

特に指定しない限り、単一方向関係の論理親または両方向関係の論理子が区画にロードされるときに、その区画の ILDS に項目が作成されます。それ以外は、変換時の論理的に関連したデータベースのロードに関して特に考慮する事項はありません。

マイグレーション再ロードを行うと、論理的に関連したデータベース内のポインターは正確でなくなるため、ポインターが修復されるまで IMS はポインターを使用してターゲット・セグメントを検索できません。ただし、ポインターを修復するためにユーザーが何らかの作業を行う必要はありません。IMS は、ポインターを修復するために、ターゲット・セグメントが必要となるまで待機し、ターゲット区画の ILDS を通じてターゲット・セグメントを検索する際にターゲット・セグメントのポインターを修正します。

関連タスク:

964 ページの『ILDS 更新方法の選択』

イメージ・コピーの作成

データベース・データ・セットのイメージ・コピーを作成する。

ILDS および PHIDAM 1 次索引以外の各データベース・データ・セットには、イメージ・コピーが必要 であることを示すフラグが設定されています。フラグが設定されているデータ・セットのイメージ・コピーを作成する必要があります。

DFSMDA メンバーおよび HIDAM 1 次索引 DBD のクリーンアップ

HALDB データベースは、動的割り振りに DFSMDA メンバーを使用せず、また、PHIDAMデータベースの 1 次索引用に別の DBD を必要としない。

HALDB データベースは、DBRC RECON データ・セットに保管されている情報を使用してデータベース・データ・セットの動的割り振りを行うため、DFSMDA メンバーは使用しません。変換処理が完了し、データベースを非 HALDB の状態に戻す必要がないことが確実となった時点で、古いデータベースの DFSMDA メンバーを削除し、HIDAM 1 次索引の DBD を廃棄してかまいません。

IMS は、PHIDAM 1 次索引の生成に必要な情報を PHIDAM DBD から取得します。 HIDAM データベースを HALDB PHIDAM データベースに変換した場合は、変換処理が完了してデータベースを非 HALDB の状態に戻す必要がないことが確実となった時点で、古い HIDAM データベースの 1 次索引の DBD は廃棄してかまいません。

関連タスク:

943 ページの『DFSMDA メンバーおよび HIDAM 1 次索引 DBD のクリーンアップ』

単純データベースを HALDB に変換する際のデータベース名の変更

データベースを HALDB に変換する際に、データベース名を変更することができます。ただし、その場合は、新しい名前に合わせてすべての PCB を変更する必要があるため、余分な作業が生じることになります。同じデータベース名を引き続き使用することをお勧めします。

データベースを HALDB に変換する際にデータベース名を変更するには、以下のステップを実行します。

1. そのデータベースについてのレコードを削除する前に、RECON リストを作成します。旧情報は、RECON 内で必要な期間だけ保存されます。
2. 旧データベースをアンロードします。
3. DBDLIB および ACBLIB からその DBD を除去します。
4. 旧データベースを参照する MDA メンバーをすべて削除します。
5. 旧データベース名について、新規の HALDB データベースをソースにした論理データベースとして、DBDGEN を実行します。
6. DBDGEN、ACBGEN、および HALDB 区画定義ユーティリティーまたは DBRC コマンド INIT.DB と INIT.PART を使用して、HALDB データベースを定義します。

変換後の非 HALDB データベースの復元

PHDAM または PHIDAM へマイグレーション済みの HDAM または HIDAM データベースを復元する処理は、フォールバック と呼ばれます。

フォールバックでは、次のタイプの論理関係の変換がサポートされています。

- 単一方向の HALDB データベースから単一方向の非 HALDB データベースへ
- 物理対の HALDB データベースから物理対の非 HALDB データベースへ

HALDB からのフォールバックでは、キーを持たないセグメントおよび非固有キーを持つセグメントを含む、物理兄弟セグメントの順序が維持されます。

1 次索引は、アンロードされずに再作成されます。副次索引は、再ロード・ユーティリティーの処理により再作成されます。ユーザー・データは保存されません。

フォールバックの要件について

HALDB データベースを非 HALDB データベースに復元する前に、いくつかの要件と考慮事項を把握しておく必要があります。

フォールバックの要件には以下のものがあります。

- 論理的に関連したデータベースのすべてを並行してフォールバックする必要があります。
- 論理子または副次索引が存在する場合は、接頭部解決ユーティリティーまたは接頭部更新ユーティリティーをシステムにインストールする必要があります。
- 復元されたデータベースを使用するには、事前に関連するデータベースと副次索引データベースをすべて復元する必要があります。

推奨事項: データベースを使用する前であるが、接頭部解決または接頭部更新ユーティリティーを再ロードしていずれかを実行した後で、イメージ・コピー・ユーティリティーの 1 つを使用して、1 次索引を含む全データ・セットのイメージ・コピーを作成してください。イメージ・コピー・ユーティリティーは、DBRC を実行して入力および記録の結果を検証し、障害発生時に有効なリカバリー・ポイントとして役立つデータベースのバックアップ・コピーが作成されたことを確認します。

更新が行われる前のデータベースの復元

HALDB に変換してから更新していないデータベースを、その HDAM または HIDAM 構造に復元する作業はきわめて簡単です。

HALDB に変換してから更新していないデータベースを HDAM または HIDAM にフォールバックするには、以下のステップを実行します。

1. DBD を以前の指定に変更します。
2. RECON データ・セットから HALDB データベース情報を削除します。
3. 非 HALDB データベース・データ・セットが削除されている場合は、最後のイメージ・コピーおよび必要なログからデータ・セットを復元します。ログが必要となるのは、最後のイメージ・コピーが作成されてから変換処理が開始されるまでにデータベースが更新された場合だけです。
4. 非 HALDB データベースを DBRC に登録します。
5. データベース・データ・セットのイメージ・コピーを取ります。

更新が行われた後のデータベースの復元

HALDB に変換した後にデータベースを更新した場合は、データベースを非 HALDB の状態に復元するためのオプションが限定されます。

データが、非 HALDB データベースの VSAM のサイズ制限である 4 GB または OSAM のサイズ制限である 8 GB に収まらなくなった場合は、特別なステップを実行しないとそのデータベースを復元できません。データベースが非 HALDB のサイズ制限を超える原因には、データベースの自然な成長や、仮想対から物理対への変換が考えられます。非 HALDB データベース構造の復元が必要となり、かつデータが非 HALDB データベースのサイズ制限を超えるというきわめてまれな状況が発生した場合は、IBM ソフトウェア・サポートにお問い合わせください。

データ量が非 HALDB データベースのサイズ制限内であれば、以下のステップを完了することによってデータベースを復元できます。

1. HD 再編成アンロード・ユーティリティー (DFSURGU0) および FALLBACK=YES 制御ステートメントを用いて、データベースをアンロードします。HD 再編成再ロード・ユーティリティーで対をなす論理子を探し、フォールバックに必要な情報を出力データの接頭部に保管します。HD 再編成アンロード・ユーティリティーによって作成される接頭部には、データを再ロードするときに新しいセグメント接頭部を作成するために必要な情報が入っています。
2. 非 HALDB データベースの DBD を復元します。
3. RECON データ・セットから HALDB データベース情報を削除します。
4. 非 HALDB データベースを DBRC に登録します。
5. データベース事前再編成ユーティリティー (DFSURPR0) を実行し、DBR 制御ステートメントを指定します。

6. HD 再編成再ロード・ユーティリティー (DFSURGL0) を用いて、非 HALDB データベースを再ロードします。
7. データベースに副次索引がある場合は、索引付きデータベースに対するフォールバック処理が完了した後に、副次索引を再作成によって復元します。
8. データベースが別のデータベースと論理的に関連している場合は、論理的に関連したすべてのデータベースに対して同時にフォールバックを実行する必要があります。
9. データベース・データ・セットのイメージ・コピーを取ります。

関連タスク:

『副次索引データベースの復元』

『論理関係を使用するデータベースの復元』

副次索引データベースの復元

副次索引データベースを非 HALDB の状態に復元するには、索引付きデータベースに対するフォールバック処理の一環として副次索引を再作成します。

非 HALDB データベースを再編成するときと同じ再作成処理を使用できます。

副次索引データベースを再作成するには、以下のいずれかのユーティリティーを実行します。

- IMS Prefix Resolution
- HISAM アンロード
- HISAM 再ロード
- IMS Index Builder
- 索引付きデータベースから副次索引を作成するすべてのツール

索引付きデータベースのフォールバック処理は、以下のトピックで説明した処理と同じです。

- 967 ページの『更新が行われる前のデータベースの復元』
- 967 ページの『更新が行われた後のデータベースの復元』

関連タスク:

967 ページの『更新が行われた後のデータベースの復元』

論理関係を使用するデータベースの復元

論理的に関連したデータベースを復元するには、非 HALDB データベースの再編成の一環として論理関係を解決するための標準的なユーティリティーを使用します。

使用できるユーティリティーは以下のとおりです。

- データベース事前再編成ユーティリティー (DFSURPR0)
- データベース接頭部解決ユーティリティー (DFSURG10)
- データベース接頭部更新ユーティリティー (DFSURGP0)

非 HALDB データベースが仮想対を使用していた場合は、論理的に関連したデータベースを HALDB に変換する際におそらく仮想対を物理対に変換しています。これは HALDB が仮想対をサポートしていないためです。フォールバック処理で仮想対を復元することはできません。復元する非 HALDB データベースの DBD ステート

メントに、物理対の定義が含まれている必要があります。さらに、物理対をなすセグメントに必要なスペースが追加されたために、フォールバックを実行できない可能性があります。

フォールバック後に仮想対を復元するには、以下のステップを実行します。

1. 現行の物理対のデータベースをフォールバックします。
2. 現行のデータベースを再編成します。
3. 論理関係を仮想対のデータベースに変更します。

論理子セグメントについては、フォールバック処理で特別な配慮が必要となります。非 HALDB IMS データベースでは、論理親の連結キーを論理子に保管しない仮想キー・ストレージ・オプションを選択できます。通常の検索では、キーが作成され、ユーザー・アプリケーションはデータ内の連結キーにアクセスします。仮想キー・ストレージ・オプションを選択する場合は、アンロードされるすべての論理子セグメントについて、論理親の連結キーを除去する必要があります。アンロードされたセグメントは、物理対の関係の一部である実セグメントとして再ロードされます。アンロードで論理親の連結キーが除去されるのは、HD 再編成アンロード・ユーティリティー (DFSURGU0) からフォールバック・アンロードを実行した場合だけです。

物理対から仮想対に直接データベースを復元して、仮想論理子の論理的順序を保持することはできません。

関連タスク:

967 ページの『更新が行われた後のデータベースの復元』

データベースから DEDB への変換

データベースを DEDB に変換する作業は数回のステップで実行できますが、いくつかの準備ステップも実行する必要があります。

データベースが論理関係、副次索引、または固定長セグメントを必要とする場合には、DEDB を使用することはできません。

データベースを DEDB に変更する前に考慮する必要のある事項があります。

- ユーザーのアプリケーション・プログラムが FH (使用不能なデータ) 状況コードを許容することができるかどうかを決めます。
- ユーザーのデータベースがランダム化ルーチンを許容することができるかどうかを決めます (HDAM から変更する場合には問題とならないかもしれません)。
- データベース・スペースを計算し直します。特に、区分化およびデータ・セットの複製などの DEDB 機能を使用する場合には、再計算が必要です。
- どのポインターが使用可能であるか決めます。

DEDB に変更するには、以下の手順を行います。

1. 既存の DBD および次のプログラムを用いて、データベースをアンロードします。
   - ユーザーのアンロード・プログラム、または

- データベース・レコードが物理ルート・キー・シーケンスで並んでいる場合は、HD 再編成アンロード・ユーティリティー

2. DEDB に対して新しい DBD をコーディングします。
3. DBD 生成を実行します。
4. 非 VSAM データ・セットの場合は、古いデータベース・スペースを削除し、新しいデータベース・スペースを定義します。 VSAM データ・セットでは、古いクラスターに割り振られているスペースを削除して、新しいクラスターのスペースを定義します。
5. DEDB 初期設定ユーティリティー (DBFUMIN0) を実行します。
6. ユーザーの DEDB ロード・プログラムを実行します。

第 6 部 付録

特記事項

本書は米国 IBM が提供する製品およびサービスについて作成したものです。本書の他言語版を IBM から入手できる場合があります。ただし、ご利用にはその言語版の製品もしくは製品のコピーを所有していることが必要な場合があります。

本書に記載の製品、サービス、または機能が日本においては提供されていない場合があります。日本で利用可能な製品、サービス、および機能については、日本 IBM の営業担当員にお尋ねください。本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービスのみが使用可能であることを意味するものではありません。これらに代えて、IBM の知的所有権を侵害することのない、機能的に同等の製品、プログラム、またはサービスを使用することができます。ただし、IBM 以外の製品とプログラムの操作またはサービスの評価および検証は、お客様の責任で行っていただきます。

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む) を保有している場合があります。本書の提供は、お客様にこれらの特許権について実施権を許諾することを意味するものではありません。実施権についてのお問い合わせは、書面にて下記宛先にお送りください。

〒*103-8510*
東京都中央区日本橋箱崎町*19*番*21*号
日本アイ・ビー・エム株式会社
法務・知的財産
知的財産権ライセンス渉外

IBM およびその直接または間接の子会社は、本書を特定物として現存するままの状態で提供し、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。国または地域によっては、法律の強行規定により、保証責任の制限が禁じられる場合、強行規定の制限を受けるものとします。

この情報には、技術的に不適切な記述や誤植を含む場合があります。本書は定期的に見直され、必要な変更は本書の次版に組み込まれます。 IBM は予告なしに、随時、この文書に記載されている製品またはプログラムに対して、改良または変更を行うことがあります。

本書において IBM 以外の Web サイトに言及している場合がありますが、便宜のため記載しただけであり、決してそれらの Web サイトを推奨するものではありません。それらの Web サイトにある資料は、この IBM 製品の資料の一部ではありません。それらの Web サイトは、お客様の責任でご使用ください。

IBM は、お客様が提供するいかなる情報も、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で、使用もしくは配布することができるものとします。

本プログラムのライセンス保持者で、(i) 独自に作成したプログラムとその他のプログラム (本プログラムを含む) との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
US

本プログラムに関する上記の情報は、適切な使用条件の下で使用することができますが、有償の場合もあります。

本書で説明されているライセンス・プログラムまたはその他のライセンス資料は、IBM 所定のプログラム契約の契約条項、 IBM プログラムのご使用条件、またはそれと同等の条項に基づいて、 IBM より提供されます。

記載されている性能データとお客様事例は、例として示す目的でのみ提供されています。実際の結果は特定の構成や稼働条件によって異なります。

IBM 以外の製品に関する情報は、その製品の供給者、出版物、もしくはその他の公に利用可能なソースから入手したものです。IBM は、それらの製品のテストは行っておりません。したがって、他社製品に関する実行性、互換性、またはその他の要求については確証できません。 IBM 以外の製品の性能に関する質問は、それらの製品の供給者にお願いします。

IBM の将来の方向または意向に関する記述については、予告なしに変更または撤回される場合があり、単に目標を示しているものです。

本書には、日常の業務処理で用いられるデータや報告書の例が含まれています。より具体性を与えるために、それらの例には、個人、企業、ブランド、あるいは製品などの名前が含まれている場合があります。これらの名前はすべて架空のものであり、類似する個人や企業が実在しているとしても、それは偶然にすぎません。

著作権使用許諾:

本書には、様々なオペレーティング・プラットフォームでのプログラミング手法を例示するサンプル・アプリケーション・プログラムがソース言語で掲載されています。お客様は、サンプル・プログラムが書かれているオペレーティング・プラットフォームのアプリケーション・プログラミング・インターフェースに準拠したアプリケーション・プログラムの開発、使用、販売、配布を目的として、いかなる形式においても、IBM に対価を支払うことなくこれを複製し、改変し、配布することができます。このサンプル・プログラムは、あらゆる条件下における完全なテストを経ていません。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。これらのサンプル・プログラムは特定物として現存するままの状態で提供されるものであり、いかなる保証も提供されません。 IBM は、お客様の当該サンプル・プログラムの使用から生ずるいかなる損害に対しても一切の責任を負いません。

それぞれの複製物、サンプル・プログラムのいかなる部分、またはすべての派生的創作物にも、次のように、著作権表示を入れていただく必要があります。
© (お客様の会社名) (年).
このコードの一部は、IBM Corp. のサンプル・プログラムから取られています。
© Copyright IBM Corp. _年を入れる_.

プログラミング・インターフェース情報

この情報では、IMS が提供するプロダクト・センシティブ・プログラミング・インターフェースとそれに関連する情報と同時に、IMS が提供する診断、修正、またはチューニング情報についても記述しています。

プロダクト・センシティブ・プログラミング・インターフェースにより、お客様のインストール済み環境で、このソフトウェア製品の診断、修正、モニター、修復、調整、またはチューニングなどの作業を実行することができます。これらのインターフェースを使用すると、IBM のソフトウェア製品の詳細設計や実装に対する依存関係が生じます。このためプロダクト・センシティブ・プログラミング・インターフェースは上記の特別な目的にだけ使用してください。詳細設計やその実現方法に依存しているので、このようなインターフェースに合わせて作成したプログラムは、新しい製品のリリース、バージョンで実行するとき、または保守サービスの結果として、変更が必要になることがあります。プロダクト・センシティブ・プログラミング・インターフェースとそれに関連する情報は、セクションやトピックの単位の場合はその冒頭で識別され、それ以外の場合は「プロダクト・センシティブ・プログラミング・インターフェース」というマーキングで識別されます。IBM では、上記の冒頭部での識別の記述、およびその記述を参照する本書内のすべての記述を、そのような記述によって示される全体コピーまたは部分コピーに含めるよう求めています。

診断、修正、チューニングの情報は、 IMS の診断、変更、またはチューニングをお客さまが行う手助けをするために提供されます。診断、修正、またはチューニング情報は、プログラミング・インターフェースとしては使用しないでください。

診断、修正、またはチューニング情報は、節またはトピックの場合はその冒頭で識別され、それ以外の場合は次のようにマーク付けされています。診断、変更、またはチューニング情報。

商標

IBM、IBM ロゴおよび ibm.com® は、世界の多くの国で登録された International Business Machines Corporation の商標です。他の製品名およびサービス名等は、それぞれ IBM または各社の商標である場合があります。現時点での IBM の商標リストについては、http://www.ibm.com/legal/copytrade.shtml をご覧ください。

Adobe、Adobe ロゴ、PostScript ロゴは、Adobe Systems Incorporated の米国およびその他の国における登録商標または商標です。

Linux は、Linus Torvalds の米国およびその他の国における商標です。

Microsoft、Windows、Windows NT および Windows ロゴは、Microsoft Corporation の米国およびその他の国における商標です。

Java およびすべての Java 関連の商標およびロゴは Oracle やその関連会社の米国およびその他の国における商標または登録商標です。

UNIX は The Open Group の米国およびその他の国における登録商標です。

製品資料に関するご使用条件

これらの資料は、以下のご使用条件に同意していただける場合に限りご使用いただけます。

適用される条件

このご使用条件は、IBM Web サイトのすべてのご利用条件に追加して適用されます。

個人使用

これらの資料は、すべての著作権表示その他の所有権表示をしていただくことを条件に、非商業的な個人による使用目的に限り複製することができます。ただし、IBM の明示的な承諾をえずに、これらの資料またはその一部について、二次的著作物を作成したり、配布 (頒布、送信を含む) または表示 (上映を含む) することはできません。

商業的使用

これらの資料は、すべての著作権表示その他の所有権表示をしていただくことを条件に、お客様の企業内に限り、複製、配布、および表示することができます。ただし、IBM の明示的な承諾をえずにこれらの資料の二次的著作物を作成したり、お客様の企業外で資料またはその一部を複製、配布、または表示することはできません。

権利

ここで明示的に許可されているもの以外に、資料や資料内に含まれる情報、データ、ソフトウェア、またはその他の知的所有権に対するいかなる許可、ライセンス、または権利を明示的にも黙示的にも付与するものではありません。

資料の使用が IBM の利益を損なうと判断された場合や、上記の条件が適切に守られていないと判断された場合、IBM はいつでも自らの判断により、ここで与えた許可を撤回できるものとさせていただきます。

お客様がこの情報をダウンロード、輸出、または再輸出する際には、米国のすべての輸出入 関連法規を含む、すべての関連法規を遵守するものとします。

IBM は、これらの資料の内容についていかなる保証もしません。これらの資料は、特定物として現存するままの状態で提供され、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任なしで提供されます。

IBM オンライン・プライバシー・ステートメント

サービス・ソリューションとしてのソフトウェアも含めた IBM ソフトウェア製品 (「ソフトウェア・オファリング」) では、製品の使用に関する情報の収集、エンド・ユーザーの使用感の向上、エンド・ユーザーとの対話またはその他の目的のために、Cookie はじめさまざまなテクノロジーを使用することがあります。多くの場合、ソフトウェア・オファリングにより個人情報が収集されることはありません。IBM の「ソフトウェア・オファリング」の一部には、個人情報を収集できる機能を持つものがあります。ご使用の「ソフトウェア・オファリング」が、これらの Cookie およびそれに類するテクノロジーを通じてお客様による個人情報の収集を可能にする場合、以下の具体的事項をご確認ください。

この「ソフトウェア・オファリング」は、Cookie もしくはその他のテクノロジーを使用して個人情報を収集することはありません。

この「ソフトウェア・オファリング」が Cookie およびさまざまなテクノロジーを使用してエンド・ユーザーから個人を特定できる情報を収集する機能を提供する場合、 お客様は、このような情報を収集するにあたって適用される法律、ガイドライン等を遵守する必要があります。これには、エンドユーザーへの通知や同意の要求も含まれますがそれらには限られません。

このような目的での Cookie を含む様々なテクノロジーの使用の詳細については、IBM の『IBM オンラインでのプライバシー・ステートメント』(http://www.ibm.com/privacy/details/jp/ja/) の『クッキー、ウェブ・ビーコン、その他のテクノロジー』および『IBM Software Products and Software-as-a-Service Privacy Statement』(http://www.ibm.com/privacy/details) を参照してください。

参考文献

この参考文献のリストには、IMS バージョン 13 ライブラリーのすべての資料、補足資料、資料コレクション、および IMS バージョン 13 ライブラリーで引用されているアクセシビリティー関連の資料が記載されています。

表題	頭字語	資料番号
IMS V13 アプリケーション・プログラミング	APG	SA88-5465
IMS V13 アプリケーション・プログラミング *API*	APR	SA88-5463
IMS V13 コマンド 第 *1* 巻*: IMS* コマンド *A-M*	CR1	SA88-5466
IMS V13 コマンド 第 *2* 巻*: IMS* コマンド *N-V*	CR2	SA88-5467
IMS V13 コマンド第 *3* 巻*: IMS* コンポーネントおよび *z/OS* コマンド	CR3	SA88-5472
IMS V13 コミュニケーションおよびコネクション	CCG	SA88-5482
IMS V13 データベース管理	DAG	SA88-5485
IMS V13 データベース・ユーティリティー	DUR	SA88-5492
IMS Version 13 Diagnosis	DGR	GC19-3654
IMS V13 出口ルーチン	ERR	SA88-5494
IMS V13 インストール	INS	GA88-5697
IMS Version 13 Licensed Program Specifications	LPS	GC19-3663
IMS V13 メッセージおよびコード 第 *1* 巻*: DFS* メッセージ	MC1	GC43-1535
IMS V13 メッセージおよびコード 第 *2* 巻*: DFS* 以外メッセージ	MC2	GC43-1536
IMS V13 メッセージおよびコード 第 *3* 巻*: IMS* 異常終了コード	MC3	GC43-1537
IMS V13 メッセージおよびコード 第 *4* 巻*: IMS* コンポーネント・コード	MC4	GC43-1538
IMS V13 オペレーションおよびオートメーション	OAG	SA88-7059
IMS V13 リリース計画	RPG	GA88-7071
IMS V13 システム管理	SAG	SA88-7072
IMS V13 システム定義	SDG	GA88-7073
IMS V13 システム・プログラミング *API*	SPR	SA88-7077
IMS V13 システム・ユーティリティー	SUR	SA88-7100

補足資料

表題	資料番号
Program Directory for Information Management System Transaction and Database Servers V13.0	GI10-8914
Program Directory for Information Management System Transaction and Database Servers V13.0 Database Value Unit Edition V13R1	GI10-8966
Program Directory for Information Management System Transaction and Database Servers V13.0 Transaction Manager Value Unit Edition V13R1	GI10-9001
IRLM メッセージおよびコード	GC19-2666

資料コレクション

表題	フォーマット	資料番号
IMS バージョン 13 製品キット	CD	SK5T-8864

IMS バージョン 13 ライブラリーで引用されているアクセシビリティー関連の資料

表題	資料番号
z/OS TSO/E 入門	SA88-8632
z/OS TSO/E ユーザーズ・ガイド	SA88-8638
z/OS 対話式システム生産性向上機能 *(ISPF)* ユーザーズ・ガイド 第 *1* 巻	SC88-8965

索引

日本語, 数字, 英字, 特殊文字の順に配列されています。なお, 濁音と半濁音は清音と同等に扱われています。

[ア行]

[カ行]

[サ行]

[タ行]

[ナ行]

[ハ行]

[マ行]

[ヤ行]

[ラ行]

[ワ行]

[数字]

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

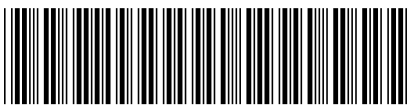
X

[特殊文字]

プログラム番号: 5635-A04
5655-DSM
5655-TM2

Printed in Japan

SA88-5485-04



日本アイ・ビー・エム株式会社
〒103-8510 東京都中央区日本橋箱崎町19-21

Spine information:

IMS バージョン **13**

データベース管理