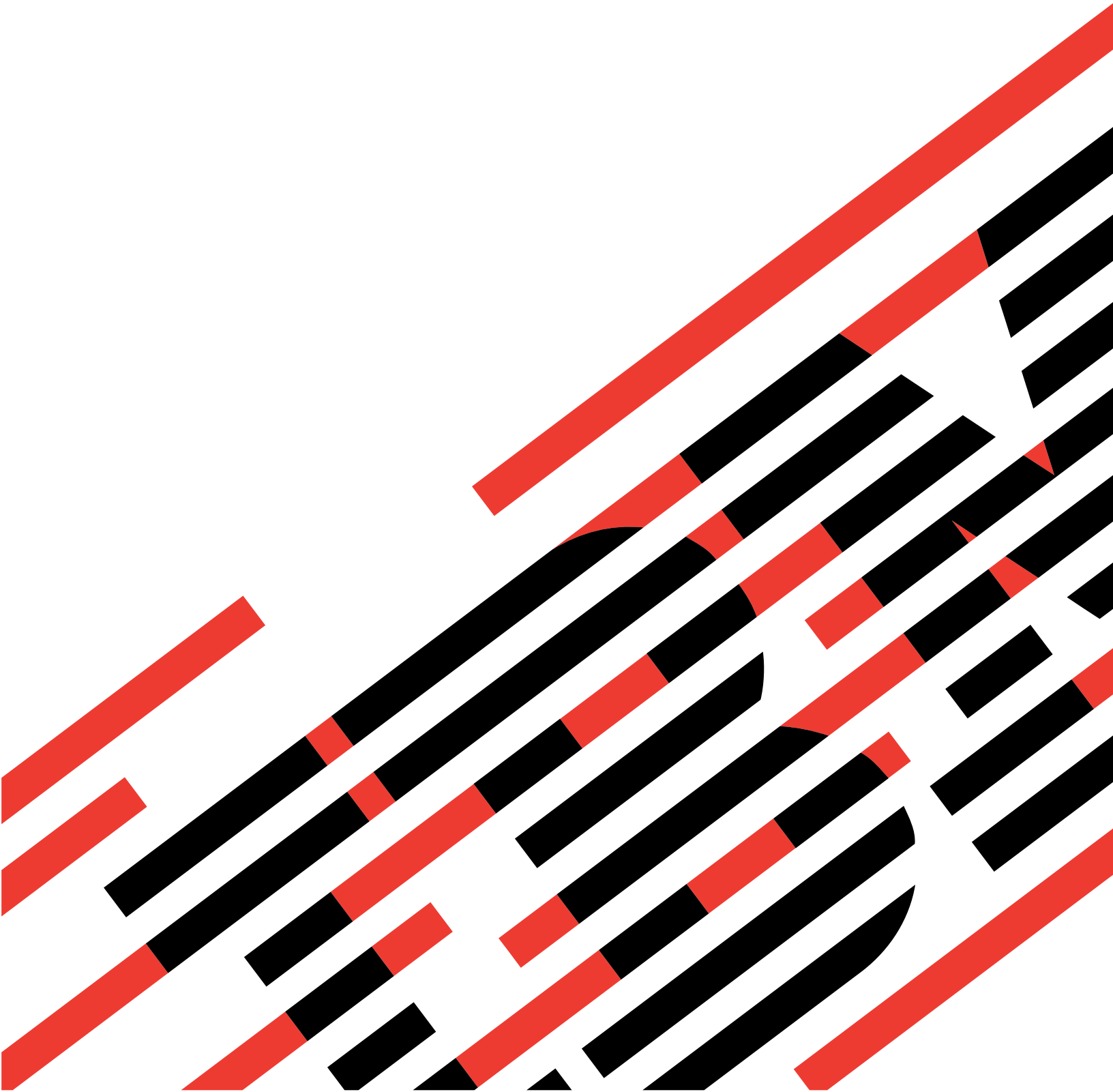
IBM

eserver

iSeries

DB2 UDB for iSeries データベース・プログラミング

バージョン 5 リリース 3



IBM

@server

iSeries

DB2 UDB for iSeries データベース・プログラミング

バージョン 5 リリース 3

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、 291 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、IBM OS/400 (プロダクト番号 5722-SS1) のバージョン 5、リリース 3、モディフィケーション 0 に適用されます。また、改訂版で断りがない限り、それ以降のすべてのリリースおよびモディフィケーションに適用されます。このバージョンは、すべての RISC モデルで稼動するとは限りません。また CISC モデルでは稼動しません。

本マニュアルに関するご意見やご感想は、次の URL からお送りください。今後の参考にさせていただきます。

`http://www.ibm.com/jp/manuals/main/mail.html`

なお、日本 IBM 発行のマニュアルはインターネット経由でもご購入いただけます。詳しくは

`http://www.ibm.com/jp/manuals/` の「ご注文について」をご覧ください。

(URL は、変更になる場合があります)

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： iSerier
DB2 UDB for iSeries Database programming
Version 5 Release 3

発　行： 日本アイ・ビー・エム株式会社

担　当： ナショナル・ランゲージ・サポート

第1刷 2005.8

この文書では、平成明朝体™W3、平成明朝体™W7、平成明朝体™W9、平成角ゴシック体™W3、平成角ゴシック体™W5、および平成角ゴシック体™W7を使用しています。この(書体*)は、(財) 日本規格協会と使用契約を締結し使用しているものです。フォントとして無断複製することは禁止されています。

注* 平成明朝体™W3、平成明朝体™W7、平成明朝体™W9、平成角ゴシック体™W3、平成角ゴシック体™W5、平成角ゴシック体™W7

目次

データベース・プログラミング

データベース・プログラミングに関するトピックでは、 DB2 UDB for iSeries (DB2 Universal Database for iSeries) データベース管理システムに関する説明と、従来のシステム・インターフェースを使用して iSeries サーバー上でデータベースをセットアップして使用する方法が記載されています。

OS/400® でのデータベース・プログラミングについて、詳しくは以下のトピックを参照してください。

V5R3 の新機能
このトピックでは、以前のリリースからの変更点を確認できます。

トピックの印刷
この情報の PDF バージョンを表示または印刷する方法。

データベース・ファイルの概念
OS/400 のデータベース・ファイルに関連した基本概念。

データベース・ファイルのセットアップ
データベース・ファイルを作成して記述する方法、論理ファイルをセットアップする方法、データベース・ファイルのアクセス・パスを記述する方法、およびデータベースのセキュリティーを保護する方法について説明します。

データベース・ファイルの処理
OS/400 でのデータベース・ファイルをより効率的に処理する方法、データベース・ファイルをオープンし、操作して、クローズする方法、データベース・ファイルに関連したエラー・メッセージをモニターおよび管理する方法について説明します。

データベース・ファイルの管理
データベース・ファイル、記述、および属性を管理することによって、システム上のデータベース・ファイルを制御する方法について説明します。さらに、データの喪失を防ぎ、保全性と制約を施行し、データベース変更時のトリガー・イベントをセットアップするために、データを制御する方法について説明します。

関連情報
データベース・プログラミングに関する追加のトピックが、 iSeries Information Center およびインターネットで提供されています。

V5R3 の新機能

今回のリリースでは、以下の点が追加または更新されています。

- 1 つの論理ファイル内に最大 256 までのメンバーを結合できます。
- パック 10 進数およびゾーン 10 進数の最大精度が 63 桁になりました。
- 物理ファイル・フィールドでの UTF-8 および UTF-16 のサポート。
- 物理メンバーの再編成で、並列処理、並行処理、および割り込み可能処理のオプションが提供されます。 196 ページの『物理ファイルの再編成』を参照してください。

- 物理ファイルを暗黙的にジャーナル処理できます。 40 ページの『物理ファイル作成時の暗黙的ジャーナル処理』を参照してください。

トピックの印刷

本書の PDF バージョンを表示またはダウンロードするには、『 データベース・プログラミング  』 (約 2500 KB) を選択します。

PDF ファイルの保存

表示または印刷のために PDF をワークステーションに保管するには、以下のようにします。

1. ブラウザーで PDF を右マウス・ボタン・クリックする (リンク上で右マウス・ボタン・クリック)。
2. Internet Explorer を使用している場合は、「**対象をファイルに保存...**」をクリックする。 Netscape Communicator を使用している場合は、「**リンクを名前を付けて保存...**」をクリックする。
3. PDF を保管したいディレクトリーに進む。
4. 「**保存**」をクリックする。

Adobe Acrobat Reader のダウンロード

これらの PDF を表示または印刷するには、Adobe Acrobat Reader が必要です。このアプリケーションは、Adobe Web サイト (www.adobe.com/products/acrobat/readstep.html)  からダウンロードできます。

データベース・ファイルの概念

このセクションでは、データベース概念と、 IBM® OS/400 データベース・ファイルをセットアップし作業するときに必要な考慮事項について説明します。

- 『DB2 Universal Database for iSeries』
- 3 ページの『DB2 UDB for iSeries へのインターフェース』
- 4 ページの『データベース・ファイル』
- 5 ページの『データベース・ファイルの記述方法』
- 8 ページの『データベース・データの保護およびモニター』
- 8 ページの『データベース・ファイルのサイズ』

DB2 Universal Database for iSeries

DB2 UDB for iSeries (DB2 Universal Database for iSeries) は、OS/400 で実行される、統合リレーショナル・データベース・マネージャーです。これは基本オペレーティング・システムの一部で、データへのアクセスとデータ保護の機能があります。また、参照保全およびデータベースの並列処理などの拡張機能もあります。

DB2 UDB for iSeries では、独立した補助記憶域プール (ASP) つまり独立ディスク・プールを利用して、各 ASP グループに対して 1 つまたは複数の別個のデータベースを関連付けることができます。さまざまなデータベースは、 1 次独立ディスク・プールを使用してセットアップされます。詳しくは、「独立ディスク・プール」のトピックを参照してください。

DB2 UDB for iSeries へのインターフェース

DB2 UDB for iSeries には、以下の、独立した、データベースへのインターフェースがあります。

- 『従来のシステム・インターフェース』
- 『SQL』
- 『iSeries ナビゲーター』
- 4 ページの『IBM Query for iSeries』

従来のシステム・インターフェース

OS/400 の従来のシステム・インターフェースは、システム・コマンドおよびその他の SQL 以外の機能の全セットで、ユーザーは、このインターフェースを使用して DB2 UDB for iSeries のデータにアクセスし変更することができます。従来のシステム・インターフェースは、データベース・オブジェクトの作成に使用できる制御言語 (CL) を提供します。システム・インターフェースには、また、データ記述仕様 (DDS) と呼ばれる、データを記述するための統合機能もあります。

IBM ライセンス・プログラムである WebSphere Development Studio for iSeries™ には、データを記述し処理するためのユーティリティーがあります。データ・ファイル・ユーティリティー (DFU) は、RPG/400®、DDS、および、対話式データ記述ユーティリティー (IDDU) によって記述されたデータベース・ファイルの中のデータを追加、変更、削除できます。原始ステートメント入力ユーティリティー (SEU) を使用すると、ファイルの中のデータを指定したり変更することができます。

SQL

構造化照会言語 (SQL) は、ホスト・プログラム言語内で、または、対話式に使用して、データベースにある情報にアクセスすることができる言語です。SQL は、リレーショナル・データベース・プロダクトにアクセスし変更するのに使用される、業界標準のデータベース・インターフェースです。SQL はデータのリレーショナル・モデルを使用します。つまり、すべてのデータは、表の中に存在すると考えます。DB2 UDB for iSeries データベースでは、 SQL 処理機能がシステムの中に組み込まれています。そして、SQL ステートメントが入っているコンパイル済みプログラムを処理します。SQL アプリケーションを作成するには、アプリケーションを作成するシステム上に、IBM ライセンス・プログラム DB2 UDB SQL 開発キット が必要です。

対話式 SQL は、DB2 UDB SQL 開発キット ライセンス・プログラムの機能の 1 つで、これにより、SQL ステートメントは、バッチ・モードでなく、動的に実行できます。対話式 SQL ステートメントのそれぞれが、ワークステーションから読み取られ、準備され、動的に実行されます。

SQL について詳しくは、 SQL プログラミングのトピック、および DB2® UDB for iSeries 構造化照会言語の概要のトピックを参照してください。

iSeries ナビゲーター

iSeries ナビゲーターは、 OS/400 に組み込まれている iSeries Access for Windows® の無料のフィーチャーです。 iSeries ナビゲーターは、データベースを含む、共通 OS/400 管理機能へのグラフィカルな Microsoft® Windows インターフェースを提供します。iSeries ナビゲーターを使用してアクセスできるデータベース操作のほとんどは、構造化照会言語 (SQL) 機能が基本になっています。ただし、操作の一部には、制御言語 (CL) コマンドのような従来のシステム・インターフェースを基にしているものがあります。

iSeries ナビゲーターについて詳しくは、 Information Center のiSeries ナビゲーターを参照してください。

IBM Query for iSeries

IBM Query for iSeries は、データベース・ファイルにある情報を選択、フォーマット、分析して、報告書および他のファイルを作成するのに使用できる IBM ライセンス・プログラムです。

データベース・ファイル

データベース・ファイルは、システムに保持されているシステム・オブジェクト・タイプ *FILE のいくつかのタイプのうちの 1 つで、ここには、入力データを内部記憶域からプログラムにどのように渡すか、および、出力データをプログラムから内部記憶域にどのように渡すかについての記述が入っています。データベース・ファイルには、以下のタイプがあります。

- 『ソース・ファイル』
- 『物理ファイル』
- 『論理ファイル』

データベース・ファイルには、メンバー (『メンバー』を参照) およびレコード (5 ページの『レコード』を参照) が含まれています。

ソース・ファイル

ソース・ファイルには、あるタイプのオブジェクトを作成するのに必要な、未コンパイルのプログラミング・コードと入力データが入っています。ソース・ファイルには、高水準言語プログラムおよびデータ記述仕様などのソース・ステートメントを入れることができます。ソース・ファイルは、ソース物理ファイル、ディスケット・ファイル、テープ・ファイル、またはインライン・データ・ファイルのいずれでもかまいません。

物理ファイル

物理ファイルは、アプリケーション・データを保管するデータベース・ファイルです。ここには、データをプログラムにどのように渡すかまたはプログラムからどのように受け取るか、および、データを実際にどのようにデータベースに保管するか、についての記述が入っています。物理ファイルは、可変長フィールドを入れることができる固定長レコードで構成されます。物理ファイルには、1 つのレコード様式と 1 つまたは複数のメンバーが入ります。SQL インターフェースから見れば、物理ファイルは表と同じです。

論理ファイル

論理ファイルは、1 つまたは複数の物理ファイルを論理的に表すデータベース・ファイルです。ここには、データをどのようにプログラムに渡すかまたはプログラムから受け取るかについての記述が入ります。このタイプのデータベース・ファイルは、1 つまたは複数の物理ファイルのレコード様式を定義するもので、データは入れません。論理ファイルによって、ユーザーは、物理ファイルとは異なる順序とフォーマットでデータにアクセスすることができるようになります。SQL インターフェースから見れば、論理ファイルは、ビューと索引に同じです。

メンバー

メンバーとは、1 つのデータベース・ファイルの中で、同じフォーマットを持った、異なるセットのデータのことです。ファイルに対して入力または出力操作を行うためには、ファイルにメンバーが少なくとも 1 つ含まれていなければなりません。一般に、データベース・ファイルにはメンバーが 1 つだけ、つまりファイルの作成時に作成されたメンバーだけがあります。ファイルにメンバーが複数個ある場合、おのおののメンバーはそのファイルのデータのサブセットとして機能します。

レコード

レコードとは、ファイル内の、関連したデータのグループのことです。SQL インターフェースから見れば、レコードは行と同じです。

データベース・ファイルの記述方法

データベース・ファイルのレコードは、次の 2 つの方法で記述することができます。

- フィールド・レベル記述。レコードのフィールドをシステムに記述します。各フィールドごとに記述するものとして、名前、長さ、データ・タイプ、妥当性検査、テキスト記述などがあります。フィールド・レベル記述によって作成されたデータベース・ファイルは、外部記述ファイルと呼ばれます。
- レコード・レベル記述。ファイルのレコードの長さだけをシステムに記述します。システムは、ファイル内のフィールドについては何も知りません。このようなデータベース・ファイルはプログラム記述ファイルと呼ばれます。

ファイルをフィールド・レベルまたはレコード・レベルのどちらで記述するかを問わず、まずファイルを記述し作成してからでなければ、そのファイルを使用するプログラムはコンパイルできません。つまり、ファイルは、使用する前にシステムに存在していなければなりません。

データの記述方法に関する詳細情報については、以下のトピックを参照してください。

- 『外部記述データおよびプログラム記述データ』
- 6 ページの『ディクショナリー記述データ』
- 7 ページの『レコード様式記述』
- 7 ページの『アクセス・パス記述』
- 7 ページの『データベース・ファイルの命名規則』

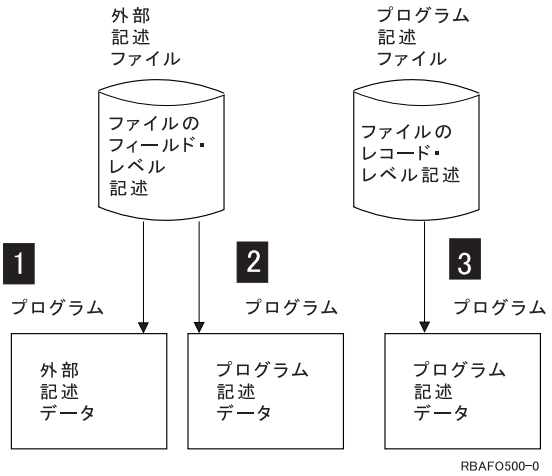
外部記述データおよびプログラム記述データ

プログラムでは、次の 2 つの方法でファイル記述を使用できます。

- プログラムは、ファイルの一部であるフィールド・レベル記述を使用します。フィールド記述はプログラム自身にとって外部のものであるため、データは外部記述データと呼ばれます。
- プログラムは、プログラム自身の中に記述されているフィールドを使用します。したがって、このようなデータはプログラム記述データと呼ばれます。レコード・レベルだけで記述されたファイルのフィールドは、そのファイルを使用するプログラム内で記述されなければなりません。

プログラムは、外部記述またはプログラム記述のどちらのファイルも使用できます。ただし、ファイルをフィールド・レベルで記述することを選択すると、システムでさらに多くのことが行えるようになります。たとえば、プログラムをコンパイルする際に、システムは外部記述ファイルから情報を抽出し、プログラムにフィールド情報を自動的に組み込むことができます。したがって、ファイルを使用するプログラムごとにフィールド情報をコーディングする必要がなくなります。

以下の図は、iSeries サーバーにおけるファイルとプログラムの典型的な関係を示しています。



1 外部記述データ

プログラムは、システムに定義されているファイルのフィールド・レベル記述を使用します。コンパイル時に言語コンパイラーは、ファイルの外部記述をプログラムにコピーします。

2 プログラム記述データ

プログラムは、フィールド・レベルでシステムに記述されているファイルを使用しますが、実際のフィールド記述は使用しません。コンパイル時に言語コンパイラーは、ファイルの外部記述をプログラムにコピーしません。ファイル内のフィールドは、プログラムで記述されます。この場合、プログラムで使用するフィールド属性 (例: フィールド長など) は、外部記述のフィールド属性と同一でなければなりません。

3 プログラム記述データ

プログラムは、レコード・レベルだけでシステムに記述されているファイルを使用します。ファイル内のフィールドはプログラムで記述しなければなりません。

外部記述ファイルは、プログラムの中でも記述することができます。前のシステムとの互換性を保つために、この方法を使用してもかまいません。たとえば、本来は従来のファイル・システムで使用していたプログラムを iSeries サーバーで実行したい場合があります。このようなプログラムはプログラム記述データを使用し、ファイル自身はレコード・レベルで記述されているにすぎません。将来的には、ファイルをフィールド・レベルで記述 (外部記述ファイル) し、システムで使用可能なデータベース機能をさらに活用するようにします。プログラム記述データを含んでいる古いプログラムはそのまま外部記述ファイルを使用することができ、新しいプログラムは、ファイルの一部であるフィールド・レベル記述を使用することになります。古いプログラムは時間をかけて、フィールド・レベル記述を使用するように変更することができます。

ディクショナリー記述データ

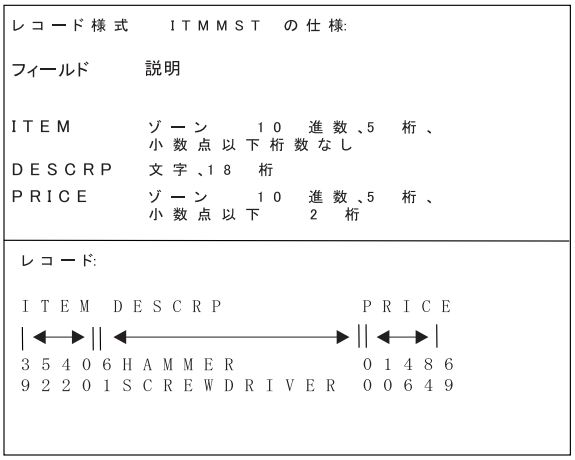
プログラム記述ファイルは、ディクショナリー記述ファイルにすることができます。対話式データ定義ユーティリティー (IDDU) を用いると、レコード様式情報を記述することができます。ファイルがプログラム記述である場合でも、IBM Query for iSeries、 iSeries Access、およびデータ・ファイル・ユーティリティー (DFU) は、データ・ディクショナリーに保管されているレコード様式記述を使用します。

外部記述ファイルも、ディクショナリー記述ファイルにすることができます。ファイルを記述するには IDDU を使用し、そして、同じく IDDU を使用してファイルを作成することができます。作成されたファイルは外部記述ファイルとなります。さらに、外部記述ファイルに保管されているファイル記述をデータ・ディクショナリーに移動させることもできます。システムは常に、データ・ディクショナリー内の定義と外部記述ファイルに保管されている記述が同一になるよう保ちます。

レコード様式記述

データベース・ファイルをシステムに記述する場合、ファイルの 2 つの重要な部分である、レコード様式とアクセス・パスを記述します。レコード様式は、各レコード内のフィールドの順序を記述します。また、レコード様式は各フィールドの詳細についても表します。この例として、長さ、データ・タイプ (例: パック 10 進数、文字など)、妥当性検査、テキスト記述、その他の情報などがあります。

以下の例は、レコード様式と物理ファイルのレコードとの関係を示しています。



RBAFO501-0

この例のレコード様式の仕様 ITMMST には、 3 つのフィールドがあります。フィールド *ITEM* はゾーン 10 進数、 5 桁、小数点以下の桁数なしです。フィールド *DESCRP* は文字で、18 桁です。フィールド *PRICE* はゾーン 10 進数、 5 桁、小数点以下 2 桁です。

物理ファイルには 1 つのレコード様式しかありません。物理ファイルのレコード様式は、データが実際に保管されている方法を表します。

論理ファイルにはデータが入っていません。論理ファイルは、1 つまたは複数の物理ファイルからのデータを別々の様式と順序で配列するために使用されます。たとえば、物理ファイル内のフィールドの順序を変更したり、物理ファイルに保管されているフィールドの一部だけをプログラムに渡したりするために使用できます。

論理ファイルのレコード様式は、物理ファイルに保管されているフィールドの長さやデータ・タイプを変更することができます。システムは、物理ファイルのフィールド記述と論理ファイルのフィールド記述との間で必要な変換を行います。たとえば、物理ファイルがフィールド FLDA を 5 桁のパック 10 進数フィールドと記述している場合、論理ファイルでは FLDA を再定義して、 7 桁のゾーン 10 進数フィールドとして使用できます。この場合、プログラムが論理ファイルを使用してレコードを読み取ると、システムは自動的に FLDA をゾーン 10 進数の形式に変換 (アンパック) します。

アクセス・パス記述

データベース・ファイルをシステムに記述する場合、ファイルの 2 つの重要な部分である、レコード様式とアクセス・パスを記述します。アクセス・パスは、レコードが取り出される順序を表します。アクセス・パスを記述する際には、アクセス・パスがキー順か、あるいは到着順かを記述します。アクセス・パスについては、 85 ページの『データベース・ファイルのアクセス・パスの記述』で詳しく説明しています。

データベース・ファイルの命名規則

ファイル名、レコード様式名、およびフィールド名は 10 文字以内であり、システムの命名規則すべてに従わなければなりません。ただし、高水準言語の中には、システムの規則よりも厳しい命名規則を持っている

ものがあることに注意してください。たとえば、RPG/400* 言語を使用した場合には、システムが 10 文字の名前を許していても、名前は 6 文字までしか許されなくなります。場合によっては、システム名を一時的に変更 (名前の変更) し、高水準言語の制限に合わせることもできます。プログラムでデータベース・フィールドの名前の変更をする方法の詳細については、高水準言語の手引きを参照してください。

この他にも、名前は以下のように固有のものでなければなりません。

- フィールド名はレコード様式の中で固有であること。
- レコード様式名とメンバー名はファイルの中で固有であること。
- ファイル名はライブラリーの中で固有であること。

データベース・データの保護およびモニター

システムには、データの保全性と整合性を向上するための 2 つの機能があります。ビジネス規則を実施することは、データを保護する 1 つの方法になります。以下の方法を使用して、ビジネス規則を実施できます。

- 参照制約を使用すると、相互従属関係にあると定義したファイルのデータに対して制御 (制約) を行うことができます。参照制約を使用すると、制約を受けるファイルに対して変更を加える際に守らなければならない規則を指定することができます。制約については、243 ページの『参照制約を使用したデータの保全性の保証』に詳しく説明されています。
- トリガーを使用すると、ファイルが変更されるときに、何らかの処置をとったり変更を評価するためのユーザー・プログラムを実行することができます。事前定義された変更を実行または試行すると、トリガー・プログラムが実行されます。トリガーについては、255 ページの『データベース内での自動イベントのトリガー』に詳しく説明されています。

データ・タイプ規則を実施することも、データを保護するもう 1 つの方法になります。ある場合には、システムでデータ・タイプのチェックを行って、たとえば、数字フィールドのデータが本当に数字であることを確かめるようにします。

さらに、システムは以下の方法を使用して、データが失われないようにします。

- ジャーナル処理およびコミットメント制御機能
- システム管理アクセス・パス保護 (SMAPP) サポート

これらのタイプのデータ保護について詳しくは、212 ページの『データベースの回復と復元』を参照してください。

データベース・ファイルのサイズ

iSeries サーバー上でファイルを設計するさいには、以下のデータベース・ファイルの最大値を念頭においてください。

説明	最大値
レコード内のバイト数	32,766 バイト
レコード様式内のフィールドの数	8,000 フィールド
ファイル内のキー・フィールドの数	120 フィールド
物理ファイルおよび論理ファイルのキーのサイズ	2000 文字[1]
ORDER BY (SQL) および KEYFLD (OPNQRYF) のキーのサイズ	10,000 バイト
ファイル・メンバーに入れるレコードの数	4,294,967,294 レコード[2]
ファイル・メンバー内のバイト数	1,869,162,846,624 バイト[3]
アクセス・パスのバイト数	1,099,511,627,776 バイト[3] [5]

説明	最大値
物理ファイル・メンバーに対して作成されるキー順論理ファイルの数	3,686 ファイル
論理ファイル・メンバー中の物理ファイル・メンバーの数	32 メンバー
結合できるメンバーの数	256 メンバー
文字フィールドまたは DBCS フィールドのサイズ	32,766 バイト[4]
ゾーン 10 進またはパック 10 進フィールドのサイズ	63 桁
同時に使用できる別個のデータベース・ファイルの最大数	~500,000
1 つの物理ファイルまたは論理ファイル内のメンバーの最大数	32,767
物理ファイルあたりの最大制約数	300 制約
物理ファイルあたりの最大トリガー数	300 トリガー
再帰挿入および更新トリガー呼び出しの最大数	200

:

[1] 先変更先出し (FCFO) アクセス・パスがファイルに指定されるとき、物理および論理ファイルのキーのサイズの最大値は 1995 文字です。

[2] キー順アクセス・パスを持つファイルの場合、メンバー中のレコードの最大数が変わり、以下の式で概算することができます。

$$\frac{2{,}867{,}200{,}000}{10 + (.8 \times \text{キーの長さ})}$$

これは概算値なので、実際のレコードの最大数は、この式から算定された数とかなり異なる場合があります。

[3] ファイル・メンバーのバイトの数とアクセス・パスのバイト数は、システムの最大オブジェクト・サイズに達したことを示すメッセージ CPF5272 が送られた時点で、調査する必要があります。

[4] 可変長文字または DBCS フィールドの最大サイズは 32,740 バイトです。DBCS グラフィック・フィールド長は、文字によって表されるので、最大値は 16,383 文字 (固定長) および 16,370 文字 (可変長) になります。

[5] 最大サイズ 4 ギガバイト (GB) つまり ACCPTHSIZE(*MAX4GB) でアクセス・パスが作成される場合、最大は 4,294,966,272 バイトになります。

上記の数は最大数です。実際に経験する限界値が上記の最大数より小さい場合があります。たとえば、特定の高水準言語では、上記の数よりもっと限定された数になる可能性があります。

上記の最大数に近づくほど、パフォーマンスに影響が及ぶ可能性があることに注意してください。たとえば、物理ファイル上に作成した論理ファイルの数が多いほど、システムのパフォーマンスは影響を受けやすくなります (多数の論理ファイル・アクセス・パスでの変更を引き起こす物理ファイルの中のデータを頻繁に変更した場合)。

通常、iSeries データベース・ファイルは、システム上で許されている最大値までサイズを大きくすることが可能です。システムは、通常一度にすべてのファイル・スペースを割り振ることはしません。むしろ、ファイルの大きさに応じて追加スペースを割り振ります。このような自動的に記憶域を割り振る方法により、適切なパフォーマンスと効率的な補助記憶域スペースが最良の組み合わせで管理されます。

ファイルのサイズ、記憶域の割り振り、およびファイルを補助記憶域と接続状態にするかどうかを制御する場合、物理ファイルの作成 (CRTPF) コマンドおよびソース・ファイルの作成 (CRTSRCPF) コマンドの SIZE、ALLOCATE、および CONTIG のそれぞれのパラメーターを使用できます。

以下の式を使用して、使用している物理ファイルおよび論理ファイルのディスク・サイズを概算できます。

- ヌルが可能なフィールドを持たない物理ファイル (アクセス・パスは除く) の場合

  ディスク・サイズ = (有効レコードおよび削除済みレコードの数 + 1) x
  (レコード長 + 1) + 12288 x (メンバーの数) + 4096

  物理ファイルのサイズは、CRTPF コマンドおよび CRTSRCPF コマンドの SIZE および ALLOCATE パラメーターにより異なります。ALLOCATE(*YES) を指定した場合は、レコード数ではなく SIZE キーワードでの初期割り振りおよび増分サイズを使用しなければなりません。
- ヌルが可能なフィールドを持つ物理ファイル (アクセス・パスは除く) の場合

  ディスク・サイズ = (有効レコードおよび削除済みレコードの数 + 1) x
  (レコード長 + 1) + 12288 x (メンバーの数) +
  4096 + ((フォーマット内のフィールドの数) ÷8) 切り上げる) x
  (有効レコードおよび削除済みレコードの数 + 1)

  物理ファイルのサイズは、CRTPF コマンドおよび CRTSRCPF コマンドの SIZE および ALLOCATE パラメーターにより異なります。ALLOCATE(*YES) を指定した場合は、レコード数ではなく SIZE キーワードでの初期割り振りおよび増分サイズを使用しなければなりません。
- 論理ファイルの場合 (アクセス・パスは除く)

  ディスク・サイズ = (12288) x (メンバーの数) + 4096
- キー順アクセス・パスの場合、メンバーごとの、索引サイズについての汎用式は、以下のとおりです。

```
a = (LimbPageUtilization - LogicalPageHeaderSize) *
    (LogicalPageHeaderSize - LeafPageUtilization - 2 * NodeSize)

b = NumKeys * (TerminalTextPerKey + 2 * NodeSize) *
    (LimbPageUtilization - LogicalPageHeaderSize + 2 * NodeSize)
    + CommonTextPerKey * [ LimbPageUtilization + LeafPageUtilization
    - 2 * (LogicalPageHeaderSize - NodeSize) ]
    - 2 * NodeSize * (LeafPageUtilization - LogicalPageHeaderSize
    + 2 * NodeSize)

c = CommonTextPerKey * [ 2 * NodeSize - CommonTextPerKey
    - NumKeys * (TerminalTextPerKey + 2 * NodeSize) ]
```

  とした場合、

```
NumberLogicalPages = ceil( [ -b - sqrt(b ** 2 - 4 * a * c) ]
                     / (2 * a))

TotalIndexSize = NumberLogicalPages * LogicalPageSize
```

  となります。

  この式は、式の中の定数のセットを以下のように変更することにより、 3 バイトおよび 4 バイトの両方について使用します。

定数	3 バイト索引	4 バイト索引
NodeSize (ノード・サイズ)	**3**	**4**

定数	3 バイト索引	4 バイト索引
LogicalPageHeaderSize (論理ページ・ヘッダー・サイズ)	**16**	**64**
LimbPageUtilization (リム・ページ使用率)	**.75 * LogicalPageSize**	**.75 * LogicalPageSize**
LeafPageUtilizations (リーフ・ページ使用率)	**.75 * LogicalPageSize**	**.80 * LogicalPageSize**

残りの定数、つまりキー当りの共通テキストおよびキー当りの端末テキストについては、以下の式で算定されます。

```
CommonTextPerKey = [ min(max(NumKeys - 256,0),256)
                   + min(max(NumKeys - 256 * 256,0),256 * 256)
                   + min(max(NumKeys - 256 * 256 * 256,0),
                     256 * 256 * 256)
                   + min(max(NumKeys - 256 * 256 * 256 * 256,0),
                     256 * 256 * 256 * 256) ]
                   * (NodeSize + 1) / NumKeys

TerminalTextPerKey = KeySizeInBytes - CommonTextPerKey
```

これにより、索引のタイプ (つまり、3 または 4 バイト) に対する索引のサイズ、全キー・サイズ、およびキーの数を計算するのに必要なことが軽減されます。共通テキスト概算は最小値になるので、概算は、実際の索引サイズより大きくなります。

索引サイズについてこの汎用式が指定された場合、LogicalPageSize (論理ページ・サイズ) は以下のようになります。

表 1. LogicalPageSize 値

キーの長さ	*MAX4GB (3 バイト) LogicalPageSize	*MAX1TB (4 バイト) LogicalPageSize
1 から 500	**4096 バイト**	**8192 バイト**
501 から 1000	**8192 バイト**	**16384 バイト**
1001 から 2000	**16384 バイト**	**32768 バイト**

表 1 の中の LogicalPageSizes (論理ページ・サイズ) は、以下の LimbPageUtilizations (リム・ページ使用率) を生成します。

キーの長さ	*MAX4GB (3 バイト) LimbPageUtilization	*MAX1TB (4 バイト) LimbPageUtilization
1 から 500	**3072 バイト**	**6144 バイト**
501 から 1000	**6144 バイト**	**12288 バイト**
1001 から 2000	**12288 バイト**	**24576 バイト**

表 1 の中の LogicalPageSizes (論理ページ・サイズ) は、以下の LeafPageUtilization (リーフ・ページ使用率) を生成します。

キーの長さ	*MAX4GB (3 バイト) LeafPageUtilization	*MAX1TB (4 バイト) LeafPageUtilization
1 から 500	3072 バイト	6554 バイト
501 から 1000	6144 バイト	13107 バイト
1001 から 2000	12288 バイト	26214 バイト

次に、索引サイズの汎用式を単純化するため、以下のようにします。

CommonTextPerKey = 0

これにより、次のようになります。
TerminalTextPerKey = KeySizeInBytes

```
b = NumKeys * (KeySizeInBytes + 2 * NodeSize) *
    (LimbPageUtilization - LogicalPageHeaderSize + 2 * NodeSize)
    - 2 * NodeSize * (LeafPageUtilization - LogicalPageHeaderSize
    + 2 * NodeSize)

c = 0

NumberLogicalPages = ceil( [ -b - sqrt(b ** 2 ) ]
                                / (2 * a))
                   = ceil[ (-2 * b) / (2 * a) ]
                   = ceil[ -b/a ]
```

例については、『例: データベース・ファイルのサイズ』を参照してください。

例: データベース・ファイルのサイズ

120 バイト・キーおよび 500,000 レコード TotalIndexSize を持つ *MAX1TB (4 バイト) アクセス・パスは、次のバイトの TotalIndexSize を持つことになります。

```
a = (LimbPageUtilization - LogicalPageHeaderSize) *
    (LogicalPageHeaderSize - LeafPageUtilization - 2 * NodeSize)
  = (6144 - 64) *
    (64 - 6554 - 2 * 4)
  = 6080 * -6498
  = -39,507,840

b = NumKeys * (KeySizeInBytes + 2 * NodeSize) *
    (LimbPageUtilization - LogicalPageHeaderSize + 2 * NodeSize)
    - 2 * NodeSize * (LeafPageUtilization - LogicalPageHeaderSize
    + 2 * NodeSize)
  = 500,000 * (120 + 2 * 4) *
    (6144 - 64 + 2 * 4)
    - 2 * 4 * (6554 - 64 + 2 * 4)
  = 500,000 * 128 *
    6088
    - 8 * 6498
  = 3.896319e+11

NumberLogicalPages = ceil[ -b/a ]
                   = ceil[ -3.896319e+11/-39507840 ]
                   = 9863
```

```
TotalIndexSize = NumberLogicalPages * LogicalPageSize
               = 9863 * 8192
               = 80,797,696 bytes
```

旧バージョンのオペレーティング・システムにあった索引サイズの式の結果は、以下のとおりです。

```
TotalIndexSize = (number of keys) * (key length + 8) *
                 (0.8) * (1.85) + 4096
               = (NumKeys) * (KeySizeInBytes + 8) *
                 (0.8) * (1.85) + 4096
               = 500000 * 128 *
                 .8 * 1.85 + 4096
               = 94,724,096
```

この概算値は、使用しているファイルにより大幅に異なる場合があります。キー順アクセス・パスは、使用しているレコードの中のデータと大きく関係します。正確なサイズを知るには、ユーザーのデータをロードして、ファイルの記述を表示してみるしかありません。

以下に最小のファイル・サイズのリストを示します。

説明	最小サイズ
メンバーのない物理ファイル	8192 バイト
単一メンバーの物理ファイル	20480 バイト
キー順アクセス・パス	12288 バイト

注: 到着順アクセス・パスの場合、追加スペースは不要です。

ファイル・サイズの他に、システムはデータベース・ファイル用の内部様式とディレクトリーを保持します。(これらの内部オブジェクトはユーザー・プロファイル QDBSHR が所有しています。) 以下に、これらのオブジェクトの概算サイズを示します。

- 別のファイルの様式を共用しないファイルの場合

  様式サイズ = (96 x フィールド数) + 4096
- 他のファイルと様式を共用するファイルの場合

  様式共用ディレクトリー・サイズ = (16 x 様式を共用するファイルの数) + 3856
- 論理ファイルまたはそれに対して作成された論理ファイル・メンバーを持つ個々の物理ファイルおよび物理ファイル・メンバーの場合

  データ共用ディレクトリー・サイズ = (16 x
  データを共用するファイルまたはメンバーの数) + 3856
- アクセス・パスを共用する論理ファイル・メンバーを持つ個々のファイル・メンバーの場合

  ディレクトリー・サイズを共用するアクセス・パス = (16 x アクセス・パスを
  共用するファイルまたはメンバーの数) + 3856

データベース・ファイルのセットアップ

第 2 部の各章では、 iSeries データベース・ファイルをセットアップする方法について詳しく説明します。

『データベース・ファイルの作成と記述』
以下のトピックでは、データベース・ファイル、ライブラリー、ソース・ファイル、および物理ファイルを作成するプロセスの概要を説明します。

- 19 ページの『データベース・ファイルの記述』
- 16 ページの『ソース・ファイルのセットアップ』
- 36 ページの『物理ファイルのセットアップ』

40 ページの『論理ファイルのセットアップ』
このトピックには、論理ファイルを記述し、作成するための指針が記載されています。これには、データ記述仕様 (DDS) を用いて論理ファイルのレコード様式と、さまざまなタイプのフィールドの使用法を記述するための情報が含まれます。さらに、論理ファイル・メンバーを定義してデータを論理グループに分割する方法も説明します。結合論理ファイルに関する項では、結合論理ファイルの使用に関する考慮事項が記載されています。これには、物理ファイルを結合する方法の例と、物理ファイルの結合に使用できるさまざまな方法が含まれます。また、結合論理ファイルのパフォーマンス、保全性、および規則の要約に関する情報も記載されています。

85 ページの『データベース・ファイルのアクセス・パスの記述』
このトピックには、システムにデータベース・ファイルおよびアクセス・パスを記述する方法と、使用可能なさまざまな方法が含まれています。 また、このようなファイル記述をプログラムで使用する方法や、使用するデータが別個のファイルに記述されている場合とプログラム自身の中で記述されている場合の違いについても説明します。 また、 DDS を使用したアクセス・パスの記述方法や、システム内にすでに存在するアクセス・パスの使用法についても説明します。

95 ページの『データベースの機密保護』
このトピックには、ファイルの機密保護、共通権限、ファイル内のデータを変更または削除する権限の制限、および論理ファイルによるデータの保護など、機密保護機能に関する情報が記載されています。 データベース・ファイルに関してユーザーに付与できるさまざまなタイプの権限と、物理ファイルに対して付与できる権限のタイプについても説明します。

データベース・ファイルの作成と記述

この章では、データベース・ファイル、ライブラリー、ソース・ファイル、および物理ファイルを作成するプロセスの概要を説明します。

システムでは、データベース・ファイルを記述し作成するためのいくつかのメソッドがサポートされています。

- IDDU

  WebSphere® Development Studio for iSeries ライセンス・プログラムの一部である対話式データ定義ユーティリティー (IDDU) を使用して、データベース・ファイルを作成することができます。IDDU を使用してデータベース・ファイルを記述する場合、ファイルの作成にも IDDU を使用することができます。
- OS/400 制御言語 (CL) (原始ステートメント入力ユーティリティー (SEU) またはデータ・ファイル・ユーティリティー (DFU) を使用してデータ記述仕様 (DDS) を指定する)。

  CL を使用してデータベース・ファイルを作成することができます。CL のデータベース・ファイル作成コマンドは、物理ファイルの作成 (CRTPF)、論理ファイルの作成 (CRTLF)、およびソース物理ファイルの作成 (CRTSRCPF) です。データベース・ファイルを作成すれば、 SEU または DFU を使用して、そのデータベース・ファイルにデータを記述することができます。SEU および DFU は、 IBM

WebSphere Development Studio for iSeries ライセンス・プログラムの一部です。本書では、これらのメソッドを使用したファイルの作成方法を中心に説明します。

- 構造化照会言語

  構造化照会言語 (SQL) ステートメントを使用して、データベース・ファイル (表) を作成し記述することができます。SQL は、IBM リレーショナル・データベース言語であり、 iSeries においてデータベース・ファイルを対話式で記述および作成するために使用できます。詳細については、 SQL プログラミング、特に表の作成と使用を参照してください。

- iSeries ナビゲーター

  iSeries ナビゲーターを使用してもデータベース・ファイル (表) を作成することができます。詳しくは、iSeries ナビゲーターを使用した表の作成と使用を参照してください。

CL および DDS を使用してデータベース・ファイルを作成し記述するには、以下のトピックを参照してください。

- 『ライブラリーの作成』
- 16 ページの『ソース・ファイルのセットアップ』
- 19 ページの『データベース・ファイルの記述』
- 36 ページの『物理ファイルのセットアップ』
- 40 ページの『論理ファイルのセットアップ』

ライブラリーの作成

ライブラリーは、他のオブジェクトへのディレクトリーとしての役割を果たすシステム・オブジェクトです。ライブラリーは関連したオブジェクトをグループ化し、ユーザーがオブジェクトを名前によって検索できるようになります。システムが認識できるオブジェクト・タイプの識別コードは *LIB です。データベース・ファイルを作成する前に、データベース・ファイルを保管するライブラリーが作成されていなければなりません。ライブラリーは、以下の方法で作成できます。

- iSeries ナビゲーターを使用して、ライブラリー (SQL ではスキーマと呼ばれる) を作成します。『iSeries ナビゲーターを使用したスキーマの作成』を参照してください。
- ライブラリー作成 (CRTLIB) コマンドを使用してライブラリーを作成できます。

  ライブラリーを作成するとき、ライブラリーの保管場所となる補助記憶域プール (ASP) を指定できます。これにより、複数の別個のデータベースを作成することができます。

iSeries ナビゲーターを使用したスキーマの作成: iSeries ナビゲーターを使用してライブラリーを作成することもできます。

1. iSeries ナビゲーター・ウィンドウで、使用したいシステムを拡張する。
2. 「**データベース**」を拡張し、操作したいデータベースを拡張する。
3. 「**スキーマ**」を右マウス・ボタン・クリックして、「**新規スキーマ (New Schema)**」を選択する。
4. 「**新規スキーマ (New Schema)**」ウィンドウで、スキーマ名を指定する。
5. 表示されるスキーマのリストにこのスキーマを追加するために、「**表示されたスキーマ・リストに追加する (Add to displayed list of schemas)**」を選択する。
6. 標準のスキーマとして作成するには、「**標準スキーマとして作成 (Create as a standard schema)**」を選択する (任意選択)。
7. データ・ディクショナリーを作成するには、「**データ・ディクショナリーを作成 (Create a data dictionary)**」を選択する (任意選択)。

8. スキーマを格納するディスク・プールを指定する。
9. 記述を指定する (任意選択)。
10. **OK** をクリックする。

ソース・ファイルのセットアップ

この章では、ソース・ファイルのセットアップ方法と、ソース・ファイルを使用する理由について説明します。

ソース・ファイルをセットアップするには、以下のトピックを参照してください。

- 『ソース・ファイルを使用する理由』
- 『ソース・ファイルの作成』

ソース・ファイルの使用について詳しくは、 230 ページの『ソース・ファイルの使用』を参照してください。

ソース・ファイルを使用する理由: ソース・ファイル (4 ページの『ソース・ファイル』を参照) は、コマンドだけでは、オブジェクトを作成するのに十分な情報が提供できないときに使用します。このファイルには、いくつかのタイプのオブジェクトの作成に必要な入力 (ソース) データが入っています。たとえば、制御言語 (CL) プログラムを作成するには、ソース・ステートメントの入ったソース・ファイルを使用しなければなりません。ソース・ステートメントはコマンドの形式になっています。論理ファイルを作成するには、DDS が入っているソース・ファイルを使用しなければなりません。

以下のオブジェクトを作成するには、ソース・ファイルが必要です。

- 高水準言語プログラム
- 制御言語プログラム
- 論理ファイル
- システム間通信機能 (ICF) ファイル
- コマンド

以下のオブジェクトを作成するため、ソース・ファイルを使用できますが、ソース・ファイルはなくてもかまいません 。

- 物理ファイル
- 表示装置ファイル
- 印刷装置ファイル
- 変換表

ソース・ファイルはデータベース・ファイル、ディスケット・ファイル、テープ・ファイル、またはインライン・データ・ファイルのどれかです。(インライン・データ・ファイルはジョブの一部として含まれます)。ソース・データベース・ファイルは、単に別のタイプのデータベース・ファイルにすぎません。ソース・データベース・ファイルは、システムにあるその他のデータベース・ファイルと同様に使用することができます。

ソース・ファイルの作成: ソース・ファイルを作成する前に、まず、ライブラリーを作成しておかなければなりません (15 ページの『ライブラリーの作成』を参照)。次に、以下のコマンドを使用して、ソース・ファイルを作成します。

- ソース物理ファイルの作成 (CRTSRCPF) コマンド

通常は、CRTSRCPF コマンドを使用してソース・ファイルを作成します。この理由は、パラメーターの多くが、ユーザーが一般的にソース・ファイルに必要とする値が省略時の値となっているためです。CRTSRCPF を使用してソース・ファイルを作成するには、『CRTSRCPF を使用したソース・ファイルの作成』を参照してください。

- 物理ファイルの作成 (CRTPF) コマンド、または 論理ファイルの作成 (CRTLF) コマンド

  ソース・ファイルを作成し、DDS を使用してレコード様式とフィールドを定義する必要がある場合は、物理ファイルの作成 (CRTPF) コマンドまたは論理ファイルの作成 (CRTLF) コマンドを使用します。

ソース・ファイルを作成するもう 1 つの方法として、 OS/400 その他のライセンス・プログラムに付属のソース・ファイルを利用できます。 『IBM 提供のソース・ファイル』を参照してください。

ほとんどのソース・ファイルに共通の属性については、 18 ページの『ソース・ファイルの属性』を参照してください。

CRTSRCPF を使用したソース・ファイルの作成: 以下の例は、CRTSRCPF コマンドおよびコマンドの省略時の値を使用してソース・ファイルを作成する方法を示しています。

```
CRTSRCPF FILE(QGPL/FRSOURCE) TEXT('Source file')
```

ソース物理ファイルの作成 (CRTSRCPF) コマンドは物理ファイルを作成しますが、属性はソース・ファイルに適したものとなります。たとえば、ソース・ファイルの省略時のレコード長は 92 です (ソース・データ・フィールドに 80、ソース順序番号フィールドに 6、またソース日付フィールドに 6 が使用されます)。

ソース・ファイルは、DDS を使用しても使用しなくても、作成することができます。以下のトピックを参照してください。

- 18 ページの『DDS を使用しないソース・ファイルの作成』
- 19 ページの『DDS を使用したソース・ファイルの作成』

IBM 提供のソース・ファイル: ユーザーの便宜を図るために、OS/400 プログラムおよびその他のライセンス・プログラムでは、各タイプのソース・ファイルごとにデータベース・ソース・ファイルを提供しています。ソース・ファイルは以下のとおりです。

ファイル名	ライブラリー名	作成に使用されるもの
QCBLSRC	QGPL	システム/38™ 互換 COBOL
QCSRC	QGPL	C プログラム
QCLSRC	QGPL	CL プログラム
QCMDSRC	QGPL	コマンド定義ステートメント
QDDSSRC	QGPL	ファイル
QFMTSRC	QGPL	分類ソース・ファイル
QLBLSRC	QGPL	COBOL/400® プログラム
QS36SRC	#LIBRARY	システム/36™ 互換 COBOL プログラム
QREXSRC	QGPL	プロシージャー言語 400/REXX プログラム
QRPGSRC	QRPG	RPG/400 プログラム
QAPLISRC	QPLI	PL/I プログラム
QPLISRC	QGPL	PL/I プログラム
QARPGSRC	QRPG38	システム/38 環境 RPG
QRPG3SRC	QRPG38	システム/38 環境 RPG
QRPG2SRC	#RPGLIB	システム/36 互換 RPG II
QS36PRC	#RPGLIB	システム/36 互換 RPG II

ファイル名	ライブラリー名	作成に使用されるもの
QS36SRC	#LIBRARY	システム/36 互換 RPG II (導入後)
QPASSRC	QPAS	Pascal プログラム
QTBLSRC	QGPL	変換表
QTXTSRC	QPDA	テキスト

ユーザーのソース・メンバーをこれらのファイルに追加するか、またはユーザー独自のソース・ファイルを作成できます。通常、IBM 提供のファイルと同じ名前を使用して、別のライブラリー内にユーザー独自のソース・ファイルを作成します (IBM 提供のファイルは、システムの新しいリリースが導入された時点で重ね書きされる場合があります)。IBM 提供のソース・ファイルは、対応する作成コマンドに使用されるファイル名を用いて作成されます (たとえば、CRTCLPGM コマンドは QCLSRC ファイル名を省略時の値として使用します)。さらに、IBM 提供のプログラマー・メニューも同じ省略時の名前を使用します。ユーザー独自のソース・ファイルを作成する場合は、そのソース・ファイルを IBM 提供のソース・ファイルと同じライブラリーに入れないようにしてください。(IBM 提供と同じファイル名を使用した場合、ライブラリー・リストでは、ユーザーのソース・ファイルが入っているライブラリーが、必ず IBM 提供のソース・ファイルが入っているライブラリーの前にくるように配置してください。)

ソース・ファイルの属性: ソース・ファイルには通常、以下のような属性があります。

- 92 文字のレコード長 (これには、6 バイトの順序番号、6 バイトの日付、および 80 バイトのソース・コードが収められます)。
- アクセス・パスが固有のキーを指定しなくても、それぞれ固有であるキー (順序番号)。ソース・ファイルにキーを指定する必要はありません。省略時のソース・ファイルはキーなし (到着順アクセス・パス) で作成されます。到着順アクセス・パスで作成されるソース・ファイルの方が、キー順アクセス・パスが指定されたソース・ファイルよりも、必要とする記憶域が少なくて済み、保管/復元時間が短縮できます。
- 複数のメンバー。
- メンバーを使用して作成されたオブジェクトの名前と同じメンバー名。
- すべてのレコードに対して同じレコード様式。
- 大多数のデータ・ファイルと比べて、各メンバーのレコード数が相対的に少ない。

以下のような制限があります。

- キーが指定されている場合には、ソース順序番号はキーとして使用しなければならない。
- キーが指定されている場合には、昇順でなければならない。
- アクセス・パスは固有のキーを指定できない。
- ALTSEQ キーワードは、ソース・ファイルの DDS で使用することができない。
- 最初のフィールドは、ゾーン 10 進データおよび 2 桁の小数部を含む 6 桁の順序番号フィールドでなければならない。
- 2 番目のフィールドは、ゾーン 10 進データおよびゼロ桁の小数部を含む 6 桁の日付フィールドでなければならない。
- 2 番目以降のフィールドはすべて、ゾーン 10 進または文字フィールドでなければならない。

DDS を使用しないソース・ファイルの作成: DDS を使用しないで、レコード長 (RCDLEN パラメーター) を指定して、ソース・ファイルを作成する場合、作成したソース・ファイルには 3 つのフィールド、つまり、 *SRCSEQ*、*SRCDAT*、および *SRCDTA* が含まれます。(レコード長には、レコードのデータ部分の長さがレコード長から 12 を差し引いた値と同じになるように、順序番号と最終変更日付のフィールド用の 12 文字が含まれていなければなりません。) レコードのデータ部分は、複数のフィールドが入るように定義付

けることができます (各フィールドごとに文字またはゾーン 10 進数を入れなければなりません)。レコードのデータ部分を複数のフィールドが入るように定義したい場合は、 DDS を使用してフィールドを定義しなければなりません。

ソース・ファイルの作成 (CRTSRCPF) コマンドを使用して作成するソース・ファイルに対しては、以下の 3 つのフィールドで構成されるレコード様式が自動的に使用されます。

フィールド	名前	データ・タイプおよび長さ	説明
1	SRCSEQ	ゾーン 10 進数、6 桁、小数部の桁数 2	レコードの順序番号
2	SRCDAT	ゾーン 10 進数、6 桁、小数部なし	レコードの最終更新日
3	SRCDTA	文字、任意の長さ	レコードのデータ部分 (テキスト)

注: IBM 提供のすべてのデータベース・ソース・ファイルは、データ部分の長さが 80 バイトです。IBM 提供の装置ソース・ファイルの場合は、データ部分の長さは関連装置の最大レコード長です。

DDS を使用したソース・ファイルの作成: レコード様式の定義が必要なソース・ファイルを作成する場合は、 物理ファイルの作成 (CRTPF) コマンドまたは 論理ファイルの作成 (CRTLF) コマンドを使用します。ソース論理ファイルを作成する場合は、重複キーを避けるため、論理ファイル・メンバーが物理ファイル・メンバーを 1 つだけ参照するようにします。

以下の例は、CRTPF を使用して、ソース・ファイルのレコード様式を定義するのに必要な DDS を示します。

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A*   R RECORD1
     A          F1             6S 2
     A          F2             6S
     A          F3            92A
```

データベース・ファイルの記述

この章では、iSeries データベース・ファイルの記述方法について説明します。

ファイルをレコード・レベルでのみ記述したい場合には、物理ファイルの作成 (CRTPF) およびソース物理ファイルの作成 (CRTSRCPF) コマンドでレコード長 (RCDLEN) パラメーターを使用することができます。

ファイルをフィールド・レベルで記述する必要がある場合は、データをデータベース・システムに記述するために、 IDDU、SQL コマンド、またはデータ記述仕様 (DDS) などのさまざまな方法を使用することができます。

対話式データ定義ユーティリティー (IDDU)

物理ファイルは、IDDU を使用して記述することができます。IDDU を使用する理由としては、IDDU がメニュー方式であり、対話式にデータを記述できることがあげられます。また、システム/36 で IDDU を使用してデータを記述した経験のある方もいることでしょう。さらに、IDDU を使用すると、Query、iSeries Access、および DFU で使用する複数様式物理ファイルを記述することもできます。

IDDU を使用してファイルを記述すると、ファイル定義は OS/400 データ・ディクショナリーの一部となります。

IDDU について詳しくは、「IDDU Use  」を参照してください。

DB2 UDB for iSeries 構造化照会言語 (SQL)

SQL (構造化照会言語) を使用して、iSeries データベース・ファイルを記述することができます。SQL 言語では、フィールドをデータベース・ファイルに記述し、ファイルを作成するためのステートメントをサポートします。

SQL は、標準かつ共通のデータベース言語の必要に応えて IBM で作成されたものです。SQL は、現在すべての IBM DB2 プラットフォーム、および多くの他社製のデータベース実装で使用されています。

データベース・ファイルが DB2 UDB for iSeries SQL 言語を用いて作成されると、ファイルの記述は自動的に SQL コレクション内のデータ・ディクショナリーに追加されます。このデータ・ディクショナリー (つまり、カタログ) は、システムによって自動的に保守されます。

SQL 言語は、他の多くのプラットフォームのデータベースへのアクセスに適した優れた言語であり、分散データベースおよび異種システム用の唯一の言語です。

SQL の詳細については、 SQL プログラミングおよび DB2 UDB for iSeries SQL 解説書を参照してください。

データ記述仕様 (DDS)

外部記述データ・ファイルは、DDS を使用して記述することができます。DDS を使用すると、フィールド、レコード、およびファイル・レベルの情報を記述することができます。

DDS を使用する理由としては、プログラマーがデータベースにデータを記述する上で、最も多くのオプションを提供していることがあげられます。たとえば、論理ファイルのキー・フィールドを記述できるのは、DDS だけです。

DDS 形式は、データを外部的に記述するための共通の様式を提供します。DDS データの列は重要です。本書の例では、列に番号が付けられており、データを正確な列に示しています。

プログラマーがデータを定義する上では DDS が最も多くのオプションを備えているので、本書では DDS を使用したデータベース・ファイルの記述に焦点をあてます。DDS を使用して、データベース・ファイルを記述するには、以下のトピックを参照してください。

- 『DDS を使用したデータベース・ファイルの記述』
- 29 ページの『データベース・ファイルとメンバー属性の指定』

データベース・ファイルが記述されれば、記述を表示することができます。202 ページの『データベース・ファイルに関する情報の表示』を参照してください。

DDS を使用したデータベース・ファイルの記述: DDS を使用してデータベース・ファイルを記述すると、以下のように、ファイル、レコード様式、結合、フィールド、キー、および選択/除外の各レベルで情報を記述できます。

- ファイル・レベルの DDS では、ファイル全体に関するシステム情報を指定できます。たとえば、ファイルのキー・フィールドのすべての値が固有でなければならないかどうかを指定できます。
- レコード様式レベルの DDS では、ファイルの特定のレコード様式に関するシステム情報を指定できます。たとえば、論理ファイルのレコード様式を記述するときに、基礎とする物理ファイルを指定できます。
- 結合レベルの DDS では、結合論理ファイルで使用される物理ファイルに関するシステム情報を指定できます。たとえば、2 つの物理ファイルを結合する方法を指定できます。
- フィールド・レベルの DDS では、レコード様式内の個々のフィールドに関するシステム情報を指定できます。たとえば、各フィールドの名前と属性を指定できます。

- キー・フィールド・レベルの DDS では、ファイルのキー・フィールドに関するシステム情報を指定できます。たとえば、レコード様式内のどのフィールドをキー・フィールドとして使用するかを指定できます。
- 選択/除外フィールド・レベルの DDS では、ファイルを処理するときにどのレコードをプログラムに戻すかといったシステム情報を指定できます。選択/除外仕様は、論理ファイルだけに適用されます。

以下のトピックは、 DDS を使用してデータベース・ファイルを記述する例を示しています。

- 『例: DDS を使用して物理ファイルを記述する』
- 24 ページの『例: DDS を使用して物理ファイルを記述する』

このほか、データベース・ファイルで DDS を使用する方法について、以下のトピックも参照してください。

- 24 ページの『DDS を使用して記述できる追加のフィールド定義機能』
- 25 ページの『既存のフィールド定義およびフィールド参照を使用したデータベース・ファイルの記述』
- 28 ページの『データベース・ファイルのフィールド参照用データ・ディクショナリーの使用』
- 28 ページの『データベース・ファイルの既存のレコード様式記述の共用』

DDS を使用したデータベース・ファイルの記述について詳しくは、DDS 解説書: 物理ファイルと論理ファイルを参照してください。

例: DDS を使用して物理ファイルを記述する: 次の例に示す物理ファイルの DDS は、以下の順序になっていなければなりません。

1 ファイル・レベル項目 (任意指定)。UNIQUE キーワードは、ファイル内の各レコードのキー・フィールドの値が固有でなければならないことを示すために使用されています。このファイルでは、重複キー値を使用することはできません。

2 レコード様式レベル項目。レコード様式名と任意指定のテキスト記述を指定しています。

3 フィールド・レベル項目。フィールド名とフィールド長、および任意指定のテキスト記述を各フィールドに指定しています。

4 キー・フィールド・レベル項目 (任意指定)。キー・フィールドとして使用するフィールド名を指定しています。

5 注釈 (任意選択)。

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A* ORDER HEADER FILE (ORDHDRP)
     A  5
     A                                     1  UNIQUE
     A       2  R ORDHDR                      TEXT('Order header record')
     A         3  CUST           5   0       TEXT('Customer number')
     A            ORDER          5   0       TEXT('Order number')
     A            .
     A            .
     A            .
     A          K CUST
     A     4    K ORDER
```

以下の例は、物理ファイル ORDHDRP (発注ヘッダー・ファイル) を示しています。このファイルには到着順のアクセス・パスが含まれ、キー・フィールドの指定はありません。さらに、このファイルを記述するのに必要な DDS をその下に示します。

物理ファイル ORDHDR のレコード様式

顧客番号 (CUST) — パック 10 進数、長さ 5、小数部なし
発注番号 (ORDER) — パック 10 進数、長さ 5、小数部なし
発注日付 (ORDATE) — パック 10 進数、長さ 6、小数部なし
購入発注番号 (CUSORD) — パック 10 進数、長さ 15、小数部なし
出荷指示 (SHPVIA) — 文字、長さ 15
注文状況 (ORDSTS) — 文字、長さ 1
...
州 (STATE) — 文字、長さ 2

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A* ORDER HEADER FILE (ORDHDRP)
     A          R ORDHDR                    TEXT('Order header record')
     A            CUST           5   0      TEXT('Customer Number')
     A            ORDER          5   0      TEXT('Order Number')
     A            ORDATE         6   0      TEXT('Order Date')
     A            CUSORD        15   0      TEXT('Customer Order No.')
     A            SHPVIA        15          TEXT('Shipping Instr')
     A            ORDSTS         1          TEXT('Order Status')
     A            OPRNME        10          TEXT('Operator Name')
     A            ORDAMT         9   2      TEXT('Order Amount')
     A            CUTYPE         1          TEXT('Customer Type')
     A            INVNBR         5   0      TEXT('Invoice Number')
     A            PRTDAT         6   0      TEXT('Printed Date')
     A            SEQNBR         5   0      TEXT('Sequence Number')
     A            OPNSTS         1          TEXT('Open Status')
     A            LINES          3   0      TEXT('Order Lines')
     A            ACTMTH         2   0      TEXT('Accounting Month')
     A            ACTYR          2   0      TEXT('Accounting Year')
     A            STATE          2          TEXT('State')
     A
```

17 列の R は、レコード様式を定義することを意味します。レコード様式名 ORDHDR は、19 列から 28 列に指定されています。

フィールドを記述する場合には、17 列には何も記入しません。17 列はブランクにしておき、19 列から 28 列にフィールド名を記入します。

35 列にはデータ・タイプを指定します。有効なデータ・タイプは、以下のとおりです。

記入項目

	意味
A	文字
P	パック 10 進数
S	ゾーン 10 進数
B	2 進数
F	浮動小数点数
H	16 進数
L	日付
T	時刻

Z 時刻スタンプ

注:

1. 2 バイト文字セット (DBCS) のデータ・タイプについては、2 バイト文字セット (DBCS) に関する考慮事項 を参照してください。
2. iSeries システムの算術処理は、ゾーン 10 進数の方が効率的です。
3. 高水準言語の中には、浮動小数点データをサポートしていないものもあります。
4. 浮動小数点フィールドを使用する場合には、以下の事柄を考慮する必要があります。
   - 浮動小数点フィールドの精度は、ビット数 (単精度または倍精度) および浮動小数点の値の内部表現が持つ機能によって異なります。この機能によって、浮動小数点フィールドで表される有効値と最大値は、サポート可能な 10 進数の桁数に変換されます。
   - 浮動小数点フィールドを定義するときに指定した桁数が、指定の精度の桁数よりも少ない場合には、指定した長さは単に表記だけのものとみなされ、内部計算の精度には一切影響しません。
   - 浮動小数点数の精度は、小数点以下の桁数 7 (単精度) または 15 (倍精度) ですが、 9 桁または 17 桁まで指定できます。補足の桁を使用して内部浮動小数点様式に固有の内部ビット・パターンを設定することができます。このように設定した場合、内部様式の浮動小数点値を 10 進数に変換し、再び内部様式に戻してもまったく同じ結果となります。

データ・タイプ (35 列) を指定しなかった場合、データ・タイプは小数点以下の桁数の記入項目で判別されます。小数点以下の桁数 (36 列から 37 列) が空白の場合には、データ・タイプは文字 (A) とみなされ、0 から 31 の数値の場合には、データ・タイプはパック 10 進数 (P) とみなされます。

フィールドの長さは 30 列から 34 列に指定し、小数点以下の桁数は 36 列から 37 列に指定します (数字フィールドの場合)。高水準言語プログラムでパック 10 進数またはゾーン 10 進数のフィールドを使用する場合には、フィールドの長さを、使用する言語で許される長さに限定しなければなりません。この長さは記憶域内のフィールドの長さではなく、記憶域の外部で指示する桁数または文字数です。たとえば、5 桁のパック 10 進数フィールドは、DDS では長さが 5 として指定しますが、記憶域は 3 バイトしか使用されません。

文字データまたは 16 進数データは、フィールド・レベルの VARLEN キーワードを指定することによって、可変長として定義することができます。データベースでは、可変長フィールドは一般的に、従業員の名前などに対して使用されます。名前は通常、30 バイトのフィールドに格納しますが、非常に長い名前の場合には 100 バイトを必要とすることがあります。しかし、フィールドを常に 100 バイトとして定義すると、記憶域を浪費することになります。また、フィールドを常に 30 バイトとして定義すると、切り捨てられる名前もでてきます。

DDS の VARLEN キーワードを使用すると、文字フィールドを可変長として定義することができます。このようなフィールドは、以下のように定義することができます。

- 長さを割り振らない可変長。この定義方法では、データと同じバイト数 (長さの値を入れるために各フィールドにつき 2 バイト、およびレコードごとにオーバーヘッドのための数バイトが追加される) だけでフィールドを格納できます。ただし、すべてのデータがファイルの可変長部分に格納され、取り出しには 2 つのディスクの読み取り操作が必要となるので、パフォーマンスに影響を及ぼすことがあります。
- データのおおよそのサイズを割り振った可変長。この定義方法では、大半のフィールド・データはファイルの一定の部分に格納でき、固定長フィールド定義に共通する未使用の記憶域割り当ては最小化されます。割り振ったフィールド長より短い長さのフィールド・データを取り出す場合、読み取り操作は 1 回で済みます。割り振った長さより長いフィールド・データは、ファイルの可変長部分に格納され、データの取り出しには 2 回の読み取り操作が必要となります。

例: DDS を使用して物理ファイルを記述する: 次の例に示す論理ファイルの DDS は、以下の順序になっていなければなりません。

1 ファイル・レベル項目 (任意指定)。この例では、UNIQUE キーワードは、このファイルの各レコードのキー値が固有でなければならないことを示しています。つまり、重複キー値を使用することはできません。

各レコード様式には、以下のものを指定します。

2 レコード様式レベル項目。この例では、レコード様式名、関連する物理ファイル、および任意指定のテキスト記述を指定しています。

3 フィールド・レベル項目 (任意指定)。この例では、レコード様式で使用する各フィールド名を指定しています。

4 キー・フィールド・レベル項目 (任意指定)。この例では、*Order* フィールドをキー・フィールドとして使用しています。

5 選択/除外フィールド・レベル項目 (任意指定)。この例では、*Opnsts* フィールドに N の値が入っているすべてのレコードをファイルのアクセス・パスから除外しています。つまり、このファイルからレコードを読み取るプログラムは、 *OPNSTS* フィールドの値が N であるレコードを見ないことになります。

6 注釈。

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A* ORDER HEADER FILE (ORDHDRP)
     A   6
     A                                  1  UNIQUE
     A      2  R ORDHDR                    PFILE(ORDHDRP)
     A        3  ORDER                     TEXT('Order number')
     A           CUST                     TEXT('Customer number')
     A           .
     A           .
     A           .
     A      4    K ORDER
     A           O OPNSTS              5    CMP(EQ 'N')
     A           S                          ALL
```

論理ファイルは、基礎となる物理ファイルをすべて作成した後で作成しなければなりません。前の例の PFILE キーワードは、論理ファイルの基礎となる物理ファイルを指定するために使用されています。

論理ファイルのレコード様式には、以下のものがあります。

- 物理ファイルからのフィールドを基とした、新しいレコード様式。
- すでに記述してある物理ファイルまたは論理ファイルと同一のレコード様式 (28 ページの『データベース・ファイルの既存のレコード様式記述の共用』を参照)。

論理ファイルのレコード様式のフィールドは、最低 1 つの物理ファイルのレコード様式に指定されているもの、または論理ファイルが基とする物理ファイルのフィールドから派生したものの、どちらかでなければなりません。

論理ファイルの記述の詳細については、『論理ファイルのセットアップ』を参照してください。

DDS を使用して記述できる追加のフィールド定義機能: 機能キーワード (DDS 形式の 45 列から 80 列) を使用すると、物理ファイルと論理ファイルのレコード様式のフィールドに関する追加の情報を記述することができます。指定することができるものとして、以下のものがあります。

- フィールドのデータが基準を満たしているかどうかを検査するための、妥当性検査用キーワード。たとえば、500 から 900 までがフィールドの有効範囲であることを記述することができます。(この検査が行われるのは、キーボードにタイプしたデータが表示されるときだけです。)
- フィールドの表示方法または印刷方法を制御するための編集用キーワード。たとえば、EDTCDE(Y) キーワードを使用すると、日付フィールドが MM/DD/YY として表示されるように指定することができます。EDTCDE および EDTWRD キーワードは、編集を制御するために使用できます。(この編集が行われるのは、表示装置ファイルまたは印刷装置ファイルで使用されたときだけです。)
- フィールド記述およびフィールド名を制御するための、文書化、見出し、および名前制御キーワード。たとえば、TEXT キーワードを使用すると、各フィールドの記述を文書化することができます。このテキスト記述は、プログラムで使用されるファイルをより分かりやすく文書化するために、コンパイラー・リストに組み込まれます。TEXT および COLHDG キーワードは、テキストおよび列見出しの定義を制御します。ALIAS キーワードは、フィールドにより記述的な名前を付けるために使用できます。(高水準言語で別名をサポートしている場合は) この別名は、プログラムで使用されます。
- フィールドの NULL 内容と省略時データを制御するための内容および省略時値キーワード。ALWNULL キーワードは、フィールド内で NULL が許可されるかどうかを指定します。 ALWNULL が使用される場合、フィールドの省略時値は NULL です。ALWNULL がフィールド・レベルに存在しない場合、NULL は許可されません。DFT (省略時) キーワードを使用して別の値を指定しない限り、文字フィールドと 16 進フィールドはブランク、数字フィールドはゼロが省略時になります。

既存のフィールド定義およびフィールド参照を使用したデータベース・ファイルの記述: あるフィールドがすでに既存のファイルに記述されており、新しく作成するファイルでそのフィールド記述を使用したい場合には、その記述を新しいファイル記述にコピーするようにシステムに要求することができます。DDS キーワード REF および REFFLD を使用すると、既存のファイル内のフィールドを参照できます。このようにすると、DDS ステートメントをコーディングする手間を省くことができます。また、そのフィールドを使用するすべてのファイルにおいて、フィールド属性の一貫性が保証されることになります。

さらに、単にフィールド記述を使用する目的だけで物理ファイルを作成することもできます。つまり、ファイルを作成してもデータを入れずに、別のファイルでフィールド記述を参照するためだけにそれを使用します。このようなタイプのファイルのことを、フィールド参照ファイルといいます。**フィールド参照ファイル**は、データを含まず、フィールド記述だけを含む物理ファイルです。

フィールド参照ファイルを使用すると、レコード様式の記述を簡素化することができ、一貫したフィールド記述が保証されることになります。フィールド参照ファイルでは、あるアプリケーションに必要なすべてのフィールドや、任意のファイルのグループを定義することができます。フィールド参照ファイルを作成するには、 DDS および物理ファイルの作成 (CRTPF) コマンドを使用します。

フィールド参照ファイルを作成した後は、各ファイルの個々のフィールドの特性を記述しなくても、このフィールド参照ファイルから物理ファイルのレコード様式を作成することができます。物理ファイルを作成する際に必要なことは、 (REF キーワードおよび REFFLD キーワードを用いて) フィールド参照ファイルを参照し、変更内容を指定することだけです。新しいファイルで指定するフィールド記述とキーワードへの変更内容により、フィールド参照ファイル内の記述が一時変更されます。

以下の例では、DSTREFP という名前のフィールド参照ファイルが配布アプリケーション用に作成されます。以下の例は、DSTREFP の記述に必要とされた DDS を示します。

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A* FIELD REFERENCE FILE (DSTREFP)
     A          R DSTREF                    TEXT('Field reference file')
     A
     A* FIELDS DEFINED BY CUSTOMER MASTER RECORD (CUSMST)
     A            CUST           5  0       TEXT('Customer numbers')
```

```
A                                      COLHDG('CUSTOMER' 'NUMBER')
A            NAME          20          TEXT('Customer name')
A            ADDR          20          TEXT('Customer address')
A
A            CITY          20          TEXT('Customer city')
A
A            STATE          2          TEXT('State abbreviation')
A                                      CHECK(MF)
A            CRECHK         1          TEXT('Credit check')
A                                      VALUES('Y' 'N')
A            SEARCH         6  0       TEXT('Customer name search')
A                                      COLHDG('SEARCH CODE')
A            ZIP            5  0       TEXT('Zip code')
A                                      CHECK(MF)
A            CUTYPE         15         COLHDG('CUSTOMER' 'TYPE')
A                                      RANGE(1 5)
A
A* FIELDS DEFINED BY ITEM MASTER RECORD (ITMAST)
A            ITEM           5          TEXT('Item number')
A                                      COLHDG('ITEM' 'NUMBER')
A                                      CHECK(M10)
A            DESCRP        18          TEXT('Item description')
A            PRICE          5  2       TEXT('Price per unit')
A                                      EDTCDE(J)
A                                      CMP(GT 0)
A                                      COLHDG('PRICE')
A            ONHAND         5  0       TEXT('On hand quantity')
A                                      EDTCDE(Z)
A                                      CMP(GE 0)
A                                      COLHDG('ON HAND')
A            WHSLOC         3          TEXT('Warehouse location')
A                                      CHECK(MF)
A                                      COLHDG('BIN NO')
A            ALLOC     R               REFFLD(ONHAND *SRC)
A                                      TEXT('Allocated quantity')
A                                      CMP(GE 0)
A                                      COLHDG('ALLOCATED')
A
A* FIELDS DEFINED BY ORDER HEADER RECORD (ORDHDR)
A            ORDER          5  0       TEXT('Order number')
A                                      COLHDG('ORDER' 'NUMBER')
A            ORDATE         6  0       TEXT('Order date')
A                                      EDTCDE(Y)
A                                      COLHDG('DATE' 'ORDERED')
A            CUSORD        15          TEXT('Cust purchase ord no.')
A                                      COLHDG('P.O.' 'NUMBER')
A            SHPVIA        15          TEXT('Shipping instructions')
A            ORDSTS         1          TEXT('Order status code')
A                                      COLHDG('ORDER' 'STATUS')
A            OPRNME    R               REFFLD(NAME *SRC)
A                                      TEXT('Operator name')
A                                      COLHDG('OPERATOR NAME')
A            ORDAMT         9  2       TEXT('Total order value')
A                                      COLHDG('ORDER' 'AMOUNT')
A            INVNBR         5  0       TEXT('Invoice number')
A                                      COLHDG('INVOICE' 'NUMBER')
A            PRTDAT         6  0       EDTCDE(Y)
A                                      COLHDG('PRINTED' 'DATE')
A            SEQNBR         5  0       TEXT('Sequence number')
A                                      COLHDG('SEQ' 'NUMBER')
A            OPNSTS         1          TEXT('Open status')
A                                      COLHDG('OPEN' 'STATUS')
A            LINES          3  0       TEXT('Lines on invoice')
A                                      COLHDG('TOTAL' 'LINES')
A            ACTMTH         2  0       TEXT('Accounting month')
A                                      COLHDG('ACCT' 'MONTH')
A            ACTYR          2  0       TEXT('Accounting year')
```

```
     A                                     COLHDG('ACCT' 'YEAR')
     A
     A* FIELDS DEFINED BY ORDER DETAIL/LINE ITEM RECORD (ORDDTL)
     A            LINE           3  0       TEXT('Line no. this item')
     A                                     COLHDG('LINE' 'NO')
     A            QTYORD         3  0       TEXT('Quantity ordered')
     A                                     COLHDG('QTY' 'ORDERED'
     A                                     CMP(GE 0)
     A            EXTENS         6  2       TEXT('Ext of QTYORD x PRICE')
     A                                     EDTCDE(J)
     A                                     COLHDG('EXTENSION')
     A
     A* FIELDS DEFINED BY ACCOUNTS RECEIVABLE
     A            ARBAL          8  2       TEXT('A/R balance due')
     A                                     EDTCDE(J)
     A
     A* WORK AREAS AND OTHER FIELDS THAT OCCUR IN MULTIPLE PROGRAMS
     A            STATUS        12          TEXT('status description')
     A     A
```

上記の例の DDS が、ソース・ファイル FRSOURCE に入力されるとします (メンバー名は DSTREFP)。この後でフィールド参照ファイルを作成するには、次の物理ファイルの作成 (CRTPF) コマンドを使用します。

```
CRTPF FILE(DSTPRODLB/DSTREFP)
      SRCFILE(QGPL/FRSOURCE) MBR(*NONE)
      TEXT('Distribution field reference file')
```

パラメーター MBR (*NONE) は、メンバーをファイルに追加しないようにシステムに指示するためのものです (これは、フィールド参照ファイルにはデータがなく、メンバーとする必要がないからです)。

DSTREFP を参照して物理ファイル ORDHDRP を記述するには、以下の例のような DDS を使用します。

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A* ORDER HEADER FILE (ORDHDRP) - PHYSICAL FILE RECORD DEFINITION
     A                                     REF(DSTREFP)
     A          R ORDHDR                   TEXT('Order header record')
     A            CUST      R
     A            ORDER     R
     A            ORDATE    R
     A            CUSORD    R
     A            SHPVIA    R
     A            ORDSTS    R
     A            OPRNME    R
     A            ORDAMT    R
     A            CUTYPE    R
     A            INVNBR    R
     A            PRTDAT    R
     A            SEQNBR    R
     A            OPNSTS    R
     A            LINES     R
     A            ACTMTH    R
     A            ACTYR     R
     A            STATE     R
     A
```

フィールド記述の基礎とするファイルを指示するために、 DSTREFP (フィールド参照ファイル名) を付けて REF キーワード (45 列から 80 列) を指定しています。各フィールドの 29 列の R は、フィールド記述を参照ファイルから得ることを示します。

ORDHDRP ファイルを作成すると、システムは ORDHDR レコード様式に含めるフィールドの属性を判別するために、DSTREFP ファイルを使用します。ORDHDRP ファイルを作成するには、物理ファイルの作成 (CRTPF) コマンドを使用します。上記の例の DDS が、ソース・ファイル QDDSSRC に入力されたとします (メンバー名は ORDHDRP)。

```
CRTPF FILE(DSTPRODLB/ORDHDRP)
      TEXT('Order Header physical file')
```

注: 本書の例で使用しているファイルは、このフィールド参照ファイルを参照しています。

データベース・ファイルのフィールド参照用データ・ディクショナリーの使用: DDS のフィールド参照ファイルの代わりに、データ・ディクショナリーと IDDU を使用することができます。IDDU を使用すると、フィールドをデータ・ディクショナリーに定義することができます。詳しくは、「IDDU Use  」を参照してください。

データベース・ファイルの既存のレコード様式記述の共用: レコード様式を物理ファイルまたは論理ファイル (結合論理ファイルを除く) で一度記述すると、それを多くのファイルで使用することができます。新しいファイルを記述する際には、新しいファイルで既存のファイルのレコード様式を使用することを指定することができます。このようにすると、新しいファイルにレコード様式を記述するときにコーディングする DDS ステートメントの数を削減することができ、補助記憶装置のスペースを節約することができます。

レコード様式を本来記述しているファイルは、レコード様式を共用するファイルに影響を与えることなく削除することができます。レコード様式を使用する最後のファイルを削除すると、システムは自動的にレコード様式記述を削除します。

以下の例は、2 つのファイルで使用される DDS を示しています。最初のファイルはレコード様式を記述しており、2 番目のファイルは最初のファイルのレコード様式を共用しています。

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A          R RECORD1                  PFILE(CUSMSTP)
     A            CUST
     A            NAME
     A            ADDR
     A            SEARCH
     A          K CUST
     A

|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A          R RECORD1                  PFILE(CUSMSTP)
     A                                     FORMAT(CUSMSTL)
     A          K NAME
     A
```

最初の例はファイル CUSMSTL を示しており、レコード様式は *Cust*、 *Name*、 *Addr*、および *Search* フィールドから構成されています。 *Cust* フィールドがキー・フィールドとして指定されています。

2 番目の例の DDS はファイル CUSTMSTL1 を示しており、 FORMAT キーワードでは、レコード様式を提供するための CUSMSTL が指定されています。レコード様式名は RECORD1 であり、これは最初の例に示されたレコード様式の名前と同じでなければなりません。ファイルは同じ様式を共用するので、両ファイルともレコード様式には *Cust*、*Name*、*Addr*、および *Search* のフィールドがあることになります。 ファイル CUSMSTL1 では、別のキー・フィールドとして *Name* が指定されています。

レコード様式を共用する場合には、以下の制限が適用されます。

- 物理ファイルは論理ファイルの様式を共用することができません。

- 結合論理ファイルは別のファイルの様式を共用することができず、また、別のファイルも結合論理ファイルの様式を共用することができません。
- ビューは別のファイルの様式を共用することができず、別のファイルもビューの様式を共用することができません。(SQL においては、**ビュー** (視点) とは、 1 つまたは複数の表からのデータの代替表現です。ビューには、その定義の基礎となる表 (1 つまたは複数) に含まれるすべての列、またはいくつかの列を含めることができます。)

関連するファイルをすべて削除し、元のファイルおよび関連するすべてのファイルを再び作成することによって元のレコード様式を変更すると、そのレコード様式は、それを共用するすべてのファイルで変更されます。元の様式を定義したファイルだけを削除し、そのファイルを新しいレコード様式で再作成すると、そのファイルの前の様式を共用していたファイルはすべて、元の様式を引き続き使用することになります。

論理ファイルを定義する際に、フィールド記述を一切指定せず、さらに FORMAT キーワードも指定しなかった場合には、最初の物理ファイル (論理ファイルの PFILE キーワードに最初に指定したもの) のレコード様式が自動的に共用されます。論理ファイルに指定するレコード様式名は、物理ファイルに指定したレコード様式名と同じでなければなりません。

あるファイルが別のファイルと様式を共用しているかどうかを調べるには、データベース関係の表示 (DSPDBR) コマンドで RCDFMT パラメーターを使用します。

レコード様式と物理ファイルおよび論理ファイルについて、詳しくは以下のトピックを参照してください。
- 『物理ファイルおよび論理ファイル間のレコード様式の関係』
- 『物理データベースおよび論理データベースのレコード様式の共用制限』

物理ファイルおよび論理ファイル間のレコード様式の関係: 物理ファイルの変更 (CHGPF) コマンドを使用してフィールドを変更、追加、および削除する場合、同じレコード様式を共用する物理ファイルと論理ファイルとの間には、以下の関係があります。
- 物理ファイル内のフィールドの長さを変更すると、論理ファイルのフィールドの長さも変更される。
- 物理ファイルにフィールドを追加すると、そのフィールドは、論理ファイルにも追加される。
- 物理ファイル内のフィールドを削除すると、DDS 内にキー・フィールド、選択ステートメント、または省略ステートメントなどの別の依存関係がない限り、フィールドは論理ファイルからも削除される。

物理データベースおよび論理データベースのレコード様式の共用制限: 1 つのレコード様式は、32K のオブジェクトによってのみ共用できます。制限に達すると、エラー・メッセージが出されます。同じデータベース・オブジェクトを複数回コピーする場合は、この制限が検出されることがあります。

注: コピーしたファイルに対して、様式共用処理が実行されます。同じ様式は、最高 32,767 回まで共用できます。それを超えても同じ様式を共用するファイルがコピーされる場合は、コピーしたファイルのために新しい様式が作成されます。

データベース・ファイルとメンバー属性の指定: データベース・ファイルを作成する場合、データベース属性はファイルやメンバーと一緒に保管されます。属性は、データベース・コマンド・パラメーターによって指定します。以下のトピックを参照してください。
- 30 ページの『ファイル名およびメンバー名 (FILE および MBR) パラメーターの指定』
- 31 ページの『物理ファイル・メンバー制御 (DTAMBRS) パラメーターの指定』
- 31 ページの『ソース・ファイルおよびソース・メンバー (SRCFILE および SRCMBR) パラメーターの指定』
- 31 ページの『データベース・ファイル・タイプ (FILETYPE) パラメーターの指定』

- 31 ページの『メンバーの最大数 (MAXMBRS) パラメーターの指定』
- 31 ページの『データの記憶位置 (UNIT) パラメーターの指定』
  - 32 ページの『UNIT パラメーター使用に関するヒント』
- 32 ページの『補助記憶装置への書き出し頻度 (FRCRATIO) パラメーターの指定』
  - 32 ページの『FRCRATIO パラメーターに関するヒント』
- 32 ページの『アクセス・パスの書き出し頻度 (FRCACCPTH) パラメーターの指定』
  - 32 ページの『FRCACCPTH パラメーターに関するヒント』
- 32 ページの『レコード様式記述変更の検査 (LVLCHK) パラメーターの指定』
  - 32 ページの『レベル検査の例』
- 33 ページの『現行アクセス・パス・メインテナンス (MAINT) のパラメーターの指定』
  - 33 ページの『MAINT パラメーターの比較』
  - 34 ページの『MAINT パラメーターに関するヒント』
- 34 ページの『回復 (RECOVER) パラメーターの指定』
  - 35 ページの『RECOVER パラメーターに関するヒント』
- 35 ページの『ファイル共用 (SHARE) パラメーターの指定』
- 35 ページの『ロックされたファイルまたはレコードの待ち時間 (WAITFILE および WAITRCD) パラメーターの指定』
- 35 ページの『共通権限 (AUT) パラメーターの指定』
- 36 ページの『ファイルを作成するシステム (SYSTEM) パラメーターの指定』
- 36 ページの『ファイルおよびメンバー・テキスト (TEXT) パラメーターの指定』
- 36 ページの『コード化文字セット識別子 (CCSID) パラメーターの指定』
- 36 ページの『分類順序 (SRTSEQ) パラメーターの指定』
- 36 ページの『言語識別コード (LANGID) パラメーターの指定』

属性を指定する方法、および可能な値についての詳細は、 制御言語 (CL) の以下のコマンドを参照してください。

- 物理ファイルの作成 (CRTPF)
- 論理ファイルの作成 (CRTLF)
- ソース物理ファイルの作成 (CRTSRCPF)
- 物理ファイル・メンバーの追加 (ADDPFM)
- 論理ファイル・メンバーの追加 (ADDLFM)
- 物理ファイルの変更 (CHGPF)
- 論理ファイルの変更 (CHGLF)
- 物理ファイル・メンバーの変更 (CHGPFM)
- ソース物理ファイルの変更 (CHGSRCPF)
- 論理ファイル・メンバーの変更 (CHGLFM)

ファイル名およびメンバー名 (FILE および MBR) パラメーターの指定: ファイルの命名は、作成コマンド中の FILE パラメーターで行います。また、ファイルが常駐することになるライブラリーも命名します。物理ファイルまたは論理ファイルを作成する場合、システムは通常、そのファイルと同じ名前を持つメ

ンバーを作成します。しかし、作成コマンド中の MBR パラメーターでメンバー名を指定することができます。また、作成コマンド中で MBR(*NONE) を指定することによって、メンバーを作成しないという選択もできます。

注: システムは、ソース物理ファイルのメンバーを自動的に作成しません。

物理ファイル・メンバー制御 (DTAMBRS) パラメーターの指定: 物理ファイル・メンバーの読み取りは、論理ファイルの作成 (CRTLF) コマンドの DTAMBRS パラメーターによって制御することができます。以下のように指定することができます。

- 物理ファイル・メンバーが読み取られる順序
- 使用される物理ファイル・メンバーの数

論理ファイルをこのように使用する場合の詳細については、 45 ページの『論理ファイル・メンバーの定義』を参照してください。

ソース・ファイルおよびソース・メンバー (SRCFILE および SRCMBR) パラメーターの指定: SRCFILE と SRCMBR パラメーターは、作成されるファイルを記述する DDS ステートメントを含んだソース・ファイルとソース・メンバーの名前を指定します。名前を指定しない場合は、以下のようになります。

- 省略時のソース・ファイル名は QDDSSRC です。
- 省略時のメンバー名は FILE パラメーターで指定された名前です。

データベース・ファイル・タイプ (FILETYPE) パラメーターの指定: データベース・ファイル・タイプは、データ (*DATA) またはソース (*SRC) のいずれかです。 物理ファイルの作成 (CRTPF) コマンドと論理ファイルの作成 (CRTLF) コマンドでは、省略時としてデータのファイル・タイプ (*DATA) が使用されます。

メンバーの最大数 (MAXMBRS) パラメーターの指定: MAXMBRS パラメーターは、ファイルが保持できるメンバーの最大数を指定します。物理ファイルと論理ファイルのメンバーの最大数の省略時は 1 であり、ソース・ファイルの省略時は *NOMAX です。

データの記憶位置 (UNIT) パラメーターの指定:

注: バージョン 3 リリース 6 で有効な UNIT パラメーターは、次のコマンドのためのノーオペレーション (NOP) 機能です。

- CRTPF
- CRTLF
- CRTSRCPF
- CHGPF
- CHGLF
- CHGSRCPF

このパラメーターを組み込んでも、エラーは起こりません。その場合、このパラメーターは無視されます。

システムは、補助記憶装置上でファイルのための記憶位置を見つけます。 ファイルの記憶位置を指定するには、UNIT パラメーターを使用してください。しないという選択もできます。UNIT パラメーターは次のものを指定します。

- 物理ファイル中のデータ・レコードの位置。
- 物理ファイルと論理ファイルの両方のアクセス・パス。

次の場合、データは別の装置に置かれます。

- 装置上に十分なスペースがない場合。
- 装置がユーザーのシステムにとって無効な場合。

ファイル・メンバーが追加されると、要求された装置にファイルが保管されなかったことを示す通知メッセージが送信されます。 (ファイル・メンバーを拡張したときには、メッセージは送信されません。)

UNIT パラメーター使用に関するヒント: 通常、UNIT パラメーターは指定すべきではありません。システムが選択したディスク装置にファイルを置くようにしてください。この方法は、パフォーマンスのために良い方法であり、またユーザーが補助記憶装置を管理する手間を省きます。

装置番号と補助記憶域プールを指定する場合は、装置番号は無視されます。補助記憶域プールについて詳しくは、システム管理、ストレージ･ソリューションに関するトピックの独立ディスク・プールを参照してください。

補助記憶装置への書き出し頻度 (FRCRATIO) パラメーターの指定: データベース・ファイルの作成、変更、または一時変更コマンドで強制書き出し頻度 (FRCRATIO) パラメーターを使用すると、データベースへの変更が補助記憶装置に書き出される時を制御することができます。通常、変更されたデータを主記憶域から補助記憶装置に書き出す時は、システムによって決定されます。ファイルのクローズ (共用クローズを除く) およびデータの強制終了操作では、残りの更新、削除、および追加が補助記憶装置に強制的に行われます。ファイルのジャーナル処理を行っている場合には、FRCRATIO パラメーターは通常、*NONE となっていなければなりません。

FRCRATIO パラメーターに関するヒント: FRCRATIO パラメーターを使用する場合には、システムのパフォーマンスと回復を考慮しなければなりません。このような考慮事項を把握するには、 212 ページの『データベースの回復と復元』を参照してください。

アクセス・パスの書き出し頻度 (FRCACCPTH) パラメーターの指定: アクセス・パスの強制書き出し (FRCACCPTH) パラメーターは、アクセス・パスが補助記憶装置に書き出される時を制御します。FRCACCPTH(*YES) は、アクセス・パスが変更されるたびに、アクセス・パスを補助記憶装置に強制的に書き出します。これにより、システムに障害が発生したときにアクセス・パスを再構築する必要性が削減されます。

FRCACCPTH パラメーターに関するヒント: FRCACCPTH(*YES) と指定すると、アクセス・パスの変更があったときにパフォーマンスが低下することがあります。アクセス・パスの強制処理の代わりに、アクセス・パスのジャーナルをとることもできます。アクセス・パスの強制とアクセス・パスのジャーナルについての詳細は、 212 ページの『データベースの回復と復元』を参照してください。

レコード様式記述変更の検査 (LVLCHK) パラメーターの指定: ファイルのオープン時、システムはデータベース・ファイル定義への変更がないかチェックします。ユーザーのプログラムで処理できないかもしれないほどにファイルが変更されている場合は、システムがプログラムに通知します。省略時の値はレベル検査の実行です。次の場合に、レベル検査を行うかどうかを指定することができます。

- ファイルを作成する場合。
- データベース・ファイルの変更コマンドを使用する場合。

システムを一時変更してレベル検査を無視するには、データベース・ファイルによる一時変更 (OVRDBF) コマンドを使用します。

レベル検査の例: たとえば、ユーザーが 2 か月前にプログラムをコンパイルし、その時点では、ファイルの各レコードに 3 つのフィールドがあることが定義されていたとします。先週、別のプログラマーがレコード様式に新しいフィールドを追加したため、現在、各レコードには 4 つのフィールドがあります。ユー

ザーのプログラムがファイルをオープンしようとすると、システムは、プログラムが最後にコンパイルされてからファイル定義に重大な変更がなされたことをプログラムに通知します。この通知は、レコード様式レベル検査と呼ばれます。

現行アクセス・パス・メインテナンス (MAINT) のパラメーターの指定: MAINT パラメーターは、クローズしているファイルについてアクセス・パスが保守される方法を指定します。ファイルがオープンしている間、システムは、ファイル内のデータに変更が加えられるごとにアクセス・パスを保守します。ただし、同じデータについて複数のアクセス・パスが存在することがあるため、あるファイルでデータを変更すると、現在オープンしていない (使用中でない) 別のファイルのアクセス・パスに変更が生じる可能性があります。クローズしているファイルのアクセス・パスを保守する 3 つの方法は、次のとおりです。

- アクセス・パスの**即時**メインテナンスでは、ファイルがオープンしているかどうかを問わず、関連するデータに変更があれば、直ちにアクセス・パスが保守されます。参照制約によって使用されるアクセス・パスは、常に即時メインテナンスになります。
- アクセス・パスの**再作成**メインテナンスでは、アクセス・パスはファイルがオープンしている間のみ保守され、ファイルがクローズしているときには保守されません。アクセス・パスは、ファイルが次にオープンされるときに再作成されます。再作成メインテナンスが指定されたファイルがクローズされると、システムはアクセス・パスの保守を停止します。ファイルがもう一度オープンすると、アクセス・パス全体が再作成されます。再作成メインテナンスが指定された特定のファイル・メンバーが 1 つまたは複数のプログラムによってオープンされると、システムは最後のユーザーがファイル・メンバーをクローズするまで、そのメンバーのアクセス・パスを保守します。
- アクセス・パスの**遅延**メインテナンスでは、アクセス・パスの保守は、ファイル・メンバーが次にオープンされ、オープン状態を保っている間に行われます。ただし、アクセス・パスは再作成メインテナンスの場合のように再作成されません。アクセス・パスへの更新は、メンバーがクローズされた時点から、それがもう一度オープンされるまで収集されます。メンバーがオープンされると、収集された変更がアクセス・パスに組み合わされます。

ファイルのメインテナンスのタイプを指定しなかった場合、省略時は即時メインテナンスとなります。

MAINT パラメーターの比較: 34 ページの表 2 は、即時、再作成、および遅延メインテナンスが、ファイルのオープンおよび処理にどのように影響するかを比較しています。

表 2. MAINT 値

機能	即時メインテナンス	再作成メインテナンス	遅延メインテナンス
オープン	アクセス・パスは現行のものなので、オープンは速い。	アクセス・パスを再作成しなければならないので、オープンは遅い。	アクセス・パスの再作成は必要ないが、変更が必要なので、オープンは中程度の速さ。 変更が多い場合には、オープンは遅い。
処理	データの変更中に即時メインテナンスのアクセス・パスが多数作成される場合は、更新/出力操作は遅い (システムがアクセス・パスを保守しなければならない)。	データの変更中に再作成メインテナンスのアクセス・パスが多数作成され、オープンされない場合は、更新/出力操作は速い (システムはアクセス・パスを保守する必要がない)。	データの変更中に遅延メインテナンスのアクセス・パスが多数作成され、オープンされない場合は、更新/出力操作は中程度の速さ (システムは変更を記録するが、アクセス・パス自体は保守されない) 。

注:

1. 遅延メインテナンスまたは再作成メインテナンスは、固有のキーを持つファイルには指定できません。
2. 再作成メインテナンスは、アクセス・パスをジャーナルするファイルには指定できません。

MAINT パラメーターに関するヒント: 指定すべきアクセス・パスのメインテナンス・タイプは、レコードの数と、ファイルがクローズしているときのファイルへの追加、削除、および更新の頻度によって異なります。

ファイル・メンバーがクローズしている間のアクセス・パスの変更が比較的少ないファイルについては、遅延メインテナンスを使用します。遅延メインテナンスでは、即時に保守されるアクセス・パスの数が削減されるので、システムのオーバーヘッドが削減されます。また、アクセス・パスが再作成される必要がないので、結果としてオープン処理がより速くなる場合もあります。

頻繁に使用されるアクセス・パスの場合や、ファイルのオープン時にアクセス・パスが再作成されるのを待てないような場合には、即時メインテナンスの指定を推奨します。頻繁に使用されないアクセス・パスの場合や、アクセス・パスを構成するレコードのキーへの変更が少ない場合には、遅延メインテナンスの指定を推奨します。

一般的に、対話式で使用するファイルについて即時メインテナンスを使用すると、応答時間が改善されます。バッチ・ジョブに使用するファイルについては、メンバーのサイズと変更の頻度に応じて、即時、遅延、または再作成のいずれかのメインテナンスを使用します。

回復 (RECOVER) パラメーターの指定: 障害の後、補助記憶装置に強制的に書き出されなかったか、またはジャーナル処理されなかった変更アクセス・パスは再構成する必要があります。アクセス・パスを再構成してデータを回復するためには、以下のコマンドで RECOVER パラメーターを使用します。これらのコマンドを使用して、いつアクセス・パスを再構成するかを指定します。

- 物理ファイルの作成 (CRTPF)
- 論理ファイルの作成 (CRTLF)
- ソース物理ファイルの作成 (CRTSRCPF)

注: アクセス・パスは、初期プログラム・ロード (IPL) の間、IPL の後、またはファイルのオープン時に再構築されます。

データの回復についての詳細は、 212 ページの『データベースの回復と復元』を参照してください。

表 3 は、重複キー・オプションとメインテナンス・オプションの可能な組み合わせについての選択肢を示しています。

表 3. 回復オプション

重複キー・オプション	メインテナンス・オプション	回復オプション
固有	即時	IPL 時に再作成 (*IPL)、IPL 後に再作成 (*AFTIPL、省略時) 、 IPL 時に再作成せず最初のオープンを待機 (*NO)
固有でない	即時または遅延	IPL 時に再作成 (*IPL)、IPL 後に再作成 (*AFTIPL)、 IPL 時に再作成せず最初のオープンを待機 (*NO、省略時)
固有でない	再作成	IPL 時に再作成せず最初のオープンを待機 (*NO、省略時)

RECOVER パラメーターに関するヒント: 回復させる必要のあるアクセス・パスを持つファイルのリストが、次回の初期プログラム・ロード (IPL) 中に、「アクセス・パスの再作成編集」画面に示されます (IPL が手動モードの場合)。この画面で希望のオプションを選択することにより、ファイルの元の回復オプションを編集することができます。IPL の完了後、アクセス・パスの再作成の編集 (EDTRBDAP) コマンドを使用して、アクセス・パスが再作成される順序を設定することができます。IPL が自動の場合は、「アクセス・パスの再作成の編集」画面は表示されず、RECOVER パラメーターによって決定された順序でアクセス・パスは再作成されます。*AFTIPL および *NO (オープン) アクセス・パスしか表示されません。

データの回復について詳しくは、「バックアップおよび回復の手引き  」を参照してください。

ファイル共用 (SHARE) パラメーターの指定: データベース・システムは、同時に複数のユーザーによる、ファイルのアクセスや変更を可能にします。SHARE パラメーターで、同一ジョブにおけるオープン・ファイルの共用ができます。たとえば、あるジョブのファイルを共用することによって、そのジョブ中のプログラムはファイルの状況、レコード位置、またバッファーを共用することができます。ジョブのファイルを共用することにより、次のものを削減し、パフォーマンスを向上させることができます。

- ジョブが必要とする記憶域の量
- ファイルのオープンとクローズに必要な時間

同一ジョブ内でのファイルの共用の詳細については、 111 ページの『同一ジョブまたは活動化グループ内のデータベース・ファイルの共用』 を参照してください。

ロックされたファイルまたはレコードの待ち時間 (WAITFILE および WAITRCD) パラメーターの指定:
ファイルの作成時には、別のジョブがファイルまたはファイル内のレコードをロックしている場合にプログラムがファイルまたはレコードを待つ時間を指定することができます。ファイルまたはレコードが解放される前に待ち時間が終了すると、ジョブがファイルの使用またはレコードの読み取りができないことを示すメッセージがプログラムに送られます。レコードとファイルのロックおよび待ち時間の詳細については、107 ページの『レコードのロック』と 108 ページの『ファイルのロック』を参照してください。

共通権限 (AUT) パラメーターの指定: ファイルの作成時に、共通認可を指定することができます。共通認可とは、あるユーザーがファイルに対する特定の権限を持っていない場合や、ファイルに対する特定の権限

があるグループに属していない場合に、そのユーザーがファイル (またはシステム上のその他のオブジェクト) に対して持つ権限のことです。共通認可の詳細については、 98 ページの『共通権限の指定』を参照してください。

ファイルを作成するシステム (SYSTEM) パラメーターの指定: ファイルをローカル・システムで作成するか、あるいは、分散データ管理 (DDM) をサポートするリモート・システムで作成するかを指定することができます。DDM の詳細については、分散データ管理を参照してください。

ファイルおよびメンバー・テキスト (TEXT) パラメーターの指定: 作成するそれぞれのファイルとメンバーについて、テキスト記述を指定することができます。テキスト・データは、ファイルおよびメンバーに関する情報を記述する上で役立ちます。

コード化文字セット識別子 (CCSID) パラメーターの指定: 物理ファイルについて、コード化文字セット識別子 (CCSID) を指定することができます。CCSID は、このファイルに含まれる文字タイプ・フィールドのコード化スキーマと文字セットを記述します。 CCSID について詳しくは、 iSeries グローバリゼーションを参照してください。

分類順序 (SRTSEQ) パラメーターの指定: ファイルの分類順序を指定することができます。SRTSEQ パラメーターの値は、CCSID および LANGID パラメーターとともに、ファイルによって使用される分類順序表を判別します。物理ファイルと論理ファイルの両方について、SETSEQ パラメーターを設定することができます。

以下のように指定することができます。

- システム提供の固有または同順位分類順序表。サポートされる各言語について分類順序表があります。
- ユーザー作成の分類順序表。
- 文字セット中の文字の 16 進値。
- 現行ジョブの分類順序、または ALTSEQ パラメーターで指定されたもの。

分類順序表は、分類順序が *HEX の場合を除き、ファイルとともに保管されます。

言語識別コード (LANGID) パラメーターの指定: SRTSEQ パラメーター値が *LANGIDSHR または *LANGIDUNQ のときにシステムが使用すべき言語識別コードを指定できます。LANGID、CCSID、および SRTSEQ パラメーターの値は、ファイルによって使用される分類順序表を判別します。物理ファイルと論理ファイルの両方について、LANGID パラメーターを設定することができます。

システムでサポートされる任意の言語識別コードを指定することもできますし、現行ジョブの言語識別コードを使用するように指定することもできます。

物理ファイルのセットアップ

この章では、物理ファイルの作成に特有の考慮事項のいくつかを説明します。

- 37 ページの『物理ファイルの作成』
- 37 ページの『物理ファイル作成時の物理ファイルとメンバー属性の指定』
- 40 ページの『物理ファイル作成時の暗黙的ジャーナル処理』

物理ファイルのレコード様式の記述に関する詳細については、 21 ページの『例: DDS を使用して物理ファイルを記述する』を参照してください。

物理ファイルのアクセス・パスの記述に関する詳細については、 85 ページの『データベース・ファイルのアクセス・パスの記述』を参照してください。

物理ファイルの作成: 物理ファイルを作成するには、まず、ライブラリーを作成 (15 ページの『ライブラリーの作成』を参照) し、ソース・ファイルを作成 (16 ページの『ソース・ファイルの作成』を参照) しておかなければなりません。次に、以下のステップを実行します。

1. DDS を使用する場合、物理ファイルの DDS をソース・ファイルに入力します。これは、原始ステートメント入力ユーティリティー (SEU) を使用して行うことができます。SEU は、IBM WebSphere Development Studio for iSeries の一部です。ソース・ステートメントをソース・ファイルに入力する方法の詳細については、 230 ページの『ソース・ファイルの処理』を参照してください。データベース・ファイルを記述する方法について詳しくは、 19 ページの『データベース・ファイルの記述』を参照してください。
2. 物理ファイルを作成します。 CRTPF (物理ファイルの作成) コマンド、または、 CRTSRCPF (ソース物理ファイルの作成) コマンドを使用することができます。

以下のコマンドは、DDS を使用して 1 メンバーのファイルを作成し、それを DSTPRODLB と呼ばれるライブラリーに配置します。

```
CRTPF FILE(DSTPRODLB/ORDHDRP)
      TEXT('Order header physical file')
```

示されているように、このコマンドでは省略時が使用されています。システムは SRCFILE パラメーターと SRCMBR パラメーターに対して、 QDDSSRC と呼ばれるソース・ファイルと ORDHDRP (ファイル名と同じ) という名前のメンバーの中にある DDS を使用します。ファイルと同じ名前のメンバーが 1 つあるファイル ORDHDRP がライブラリー DSTPRODLB に配置されます。

表は、物理ファイルと同様です。 iSeries ナビゲーターまたは CREATE TABLE SQL ステートメントを使用して表を作成できます。詳しくは、以下を参照してください。

- iSeries ナビゲーターを使用した表の作成と使用
- CREATE TABLE

物理ファイル作成時の物理ファイルとメンバー属性の指定: 物理ファイルの作成 (CRTPF)、ソース物理ファイルの作成 (CRTSRCPF)、物理ファイルの変更 (CHGPF)、ソース物理ファイルの変更 (CHGSRCPF)、物理ファイル・メンバーの追加 (ADDPFM)、物理ファイル・メンバーの変更 (CHGPFM) コマンドで物理ファイルおよびメンバーについて指定できる属性には、以下のものがあります。

- 『満了日』
- 38 ページの『物理ファイル・メンバーのサイズ』
- 38 ページの『記憶域の割り振り』
- 38 ページの『記憶領域を割り振る方法』
- 39 ページの『レコード長』
- 39 ページの『削除済みレコード』
- 40 ページの『物理ファイルの処理可能性』
- 40 ページの『ソース・タイプ』

満了日: **EXPDATE パラメーター。**このパラメーターでは、ファイル内の各メンバーの満了日を指定します (ADDPFM、CHGPFM、CRTPF、CHGPF、CRTSRCPF、および CHGSRCPF コマンド)。満了日が経過している場合、ファイルがオープンされたときに、システム操作員にその旨が通知されます。その時点でシステム操作員は、満了日を指定変更して続行したり、またはジョブを停止したりすることができます。各メンバーは別個の満了日を持つことができ、この日付はメンバーをファイルに追加するときに指定します。(満了日検査は指定変更することができます。 107 ページの『ファイルの満了日の検査』を参照してください。)

物理ファイル・メンバーのサイズ: **SIZE パラメーター。**このパラメーターには、各メンバーに入れることができる最大レコード数を指定します (CRTPF、CHGPF、CRTSRCPF、および CHGSRCPF コマンド)。最大値を判別するには、以下の式を使用することができます。

R + (I * N)

各文字の意味は以下のとおりです。

R 開始レコード数

I 各回に加算するレコード (増分) 数

N 増分を加算する回数

SIZE パラメーターの省略時は以下のとおりです。

R 10,000

I 1,000

N 3 (CRTPF コマンド)

499 (CRTSRCPF コマンド)

たとえば、5000 レコード用に作成されたファイルに、それぞれ 1000 レコードの増分を 3 回加えたものが R であるとします。システムは、初期レコード数 5000 に 1000 を 3 回追加して、合計最大値を 8000 にすることができます。合計最大値に達すると、システム操作員は、ジョブを停止するか、またはシステムにレコードの増分をもう一度追加し、続行するよう指示します。増分が追加されると、システム活動記録ログにメッセージが送られます。ファイルがその最大サイズを超えて拡張されるときには、最小拡張は現行サイズの 10% です (それが指定された増分より大きい場合でも)。

省略時のサイズを使用するかまたはサイズを指定する代わりに、 *NOMAX を指定することができます。1 つのファイルに許される最大レコード数については、 データベース・ファイルのサイズを参照してください。

記憶域の割り振り: **ALLOCATE パラメーター。**このパラメーターは、メンバーをファイルに追加するとき、それらに対して割り振る記憶域を制御するためのものです (CRTPF、CHGPF、CRTSRCPF、および CHGSRCPF コマンド)。割り振る記憶域は 1 メンバーの初期レコード数を含むのに十分な大きさとします。メンバーを追加するときにユーザーが記憶域を割り振らなかった場合には、必要に応じてシステムが自動的に記憶域割り振りを拡張します。SIZE パラメーターで最大サイズを指定した場合に限って、ALLOCATE パラメーターを使用することができます。SIZE(*NOMAX) を指定すると、ALLOCATE(*YES) は指定することができません。

記憶領域を割り振る方法: **CONTIG パラメーター。**このパラメーターは、メンバーに対して物理記憶域を割り振る方法を制御します (CRTPF コマンドと CRTSRCPF コマンド)。記憶域を割り振る場合、メンバーの開始レコード数の記憶域が連続するよう要求することができます。すなわち、メンバー内のすべてのレコードは物理的にまとまって配置されます。連続した記憶域が十分にない場合は、連続した記憶域割り振りは使用されず、メンバーを追加するときに、割り振りを要求するジョブに情報メッセージが送られます。

注: 物理ファイルを初めて作成する場合、システムは常に初期の記憶域を連続して割り振ろうとします。CONTIG(*NO) を使用するときと、CONTIG(*YES) を使用するときの唯一の相違点は、CONTIG(*YES) を使用すると、ファイルを作成するときに連続した記憶域を割り振ることができない場合にシステムがメッセージをジョブ・ログに送るということです。作成後にファイルを拡張するときは、CONTIG パラメーターで何を指定していても、メッセージは送られません。

レコード長: **RCDLEN パラメーター**。このパラメーターには、ファイルのレコードの長さを指定します (CRTPF および CRTSRCPF コマンド)。ファイルをレコード・レベルだけで記述する場合には、ファイルを作成するときに RCDLEN パラメーターを指定します。DDS、IDDU または SQL を用いてファイルを記述する場合には、このパラメーターを指定することはできません (システムはフィールド・レベル記述からファイルのレコード長を自動的に判別します)。

削除済みレコード: **DLTPCT パラメーター**。このパラメーターには、システムがシステム活動記録ログにメッセージを送る前に、ファイルに含めることのできる削除済みレコードのパーセントを指定します (CRTPF、CHGPF、CRTSRCPF、および CHGSRCPF コマンド)。ファイルがクローズするとき、システムは削除済みレコードのパーセントを判別するためにメンバーを検査します。そのパーセントが DLTPCT パラメーターで指定した値を超えている場合には、メッセージが活動記録ログに送られます。(活動記録ログの処理について詳しくは、「CL Programming (CL プログラミング、 SD88-5038)」のメッセージ処理の章を参照してください。) ファイル内の削除済みレコードが一定のパーセンテージに達したことを知る理由の 1 つは、削除済みレコードに使用されていたスペースを再利用することです。削除済みレコードについてのメッセージを受け取った後、スペースを再使用するために物理ファイル・メンバーの再編成 (RGZPFM) コマンドを使用することができます。(RGZPFM の詳細については、 196 ページの『物理ファイルの再編成』を参照してください。) また、DLTPCT パラメーターに *NONE 値を使用することによって、削除済みレコードの検査を省略するよう指定することもできます。DLTPCT パラメーターの省略時は *NONE です。

REUSEDLT パラメーター。このパラメーターには、削除済みレコードのスペースを、今後の書き込み操作で再使用するかどうかを指定します (CRTPF および CHGPF コマンド)。REUSEDLT パラメーターに *YES と指定すると、そのファイルに対するすべての挿入要求で削除済みレコードのスペースが再使用されます。削除済みレコードのスペースを再使用することによって、RGZPFM コマンドを出さなくても、削除済みレコードで使われていたスペースを再使用することができます。CHGPF コマンドを使用して削除済みレコードを再使用するようにファイルを変更する場合、特にファイルが大きく、その中にすでにたくさんの削除済みレコードがあると、コマンドの実行時間が長くなります。以下の点に注意しておくことは重要です。

- *到着順* は、削除済みレコードのスペースを再使用するファイルについては意味がなくなります。削除済みレコードのスペースが再使用されると、レコードは必ずしもファイルの終わりに挿入されなくなります。
- 削除済みレコードのスペース再使用属性を付けて新しい物理ファイルを作成した場合に、そのファイルがキー順のときは、 FIFO または LIFO アクセス・パス属性をその物理ファイルに指定することはできません。また、FIFO または LIFO アクセス・パス属性を持つどのようなキー順論理ファイルも、その物理ファイルを基として作成することはできません。
- 重複キーについて FIFO または LIFO の順序付けが指定された物理ファイルを基礎としている論理ファイルがある場合、または物理ファイルの重複キーの順序付けが FIFO または LIFO である場合には、その既存の物理ファイルを、削除済みレコードのスペースを再使用するように変更することはできません。
- 削除済みレコードのスペースは、直接ファイルとして処理されるファイル、あるいは相対レコード番号を用いて処理されるファイルに指定してはなりません。

  注: 削除済みレコードの再使用については、 104 ページの『削除済みレコードの再使用』を参照してください。

REUSEDLT パラメーターの省略時は *NO です。

***物理ファイルの処理可能性:* ALWUPD パラメーターと ALWDLT パラメーター。**ファイルの処理可能性を使用すると、データベース・ファイルの権限から独立して、データベース・ファイルについて許可される入出力操作を制御することができます。データベース・ファイルの処理可能性および権限の詳細については、『データベースの機密保護』を参照してください。

***ソース・タイプ:* SRCTYPE パラメーター。**このパラメーターは、ソース・ファイル (ADDPFM および CHGPFM コマンド) の中のメンバーのソース・タイプを指定します。このソース・タイプは、メンバーのために使用される構文検査機能、プロンプト、および様式設定を決定します。ユーザーが固有のソース・タイプ (COBOL および RPG のような iSeries でサポートされているタイプ以外) を指定する場合は、ユーザーはその固有のタイプを扱うプログラミングを行わなければなりません。

ソース・タイプが変更される場合、メンバーが次にオープンされたときに初めて反映されます。現在オープンしているメンバーには影響しません。

物理ファイル作成時の暗黙的ジャーナル処理: 物理ファイルの作成時に、自動的にジャーナル処理を開始することができます。物理ファイルが作成されるのと同じライブラリー内に QDFTJRN というデータ域が存在し、このデータ域を使用する権限がユーザーに与えられている場合、以下のすべてが該当するなら、データ域で指定されたジャーナルにジャーナル処理が開始されます。

- 物理ファイルに関連して識別されるライブラリーは、QSYS、QSYS2、 QRECOVERY、 QSPL、 QRCL、 QRPLOBJ、 QGPL、または QTEMP であってはなりません。
- データ域によって指定されるジャーナルがすでに存在する必要があり、ユーザーにはそのジャーナルに対してジャーナル処理を行う権限が必要です。
- データ域の最初の 10 バイトには、ジャーナルが格納されているライブラリーの名前が含まれなければなりません。
- 2 番目の 10 バイトには、ジャーナルの名前が含まれる必要があります。
- 3 番目の *n* バイトには、値 *FILE が含まれる必要があります。ジャーナル処理を開始しないようにするために、値 *NONE を使用できます。

論理ファイルのセットアップ

この章では、論理ファイルの作成に特有の考慮事項のいくつかを説明します。 論理ファイルをセットアップするための規則の大半は、すべての種類の論理ファイルに適用されます。本書では、1 種類の論理ファイルにしか適用されない規則については、それがどの種類に適用されるかを明記しています。すべての種類の論理ファイルに適用できる規則については特定の種類を明記していません。

論理ファイルを作成するには、以下のトピックを参照してください。

- 『論理ファイルの作成』
- 47 ページの『論理ファイルのレコード様式の記述』
- 53 ページの『論理ファイルのアクセス・パスの記述』
- 60 ページの『結合論理ファイルのセットアップ』

論理ファイルの作成

論理ファイルを作成する前に、論理ファイルが基礎とする物理ファイルがすでに存在していなければなりません。

論理ファイルを作成するには、以下の手順に従います。

1. 論理ファイルの DDS をソース・ファイルに入力します。これには、SEU またはその他の手段を使用します。ソース・コードがどのようにソース・ファイルに配置されるかについては、 230 ページの『ソース・ファイルの処理』を参照してください。論理ファイル ORDHDRL (発注ヘッダー・ファイル) の DDS を以下に示します。

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A* ORDER HEADER LOGICAL FILE (ORDHDRL)
     A          R ORDHDR                    PFILE(ORDHDRP)
     A          K ORDER
```

   このファイルは、キー・フィールド *Order* (発注番号) を使用してアクセス・パスを定義しています。レコード様式は、関連する物理ファイル ORDHDRP と同一です。論理ファイルのレコード様式名は、フィールド記述を指定していないので、物理ファイルのレコード様式名と同じでなければなりません。
2. 論理ファイルを作成します。 CRTLF (論理ファイルの作成) コマンドを使用することができます。

   CRTLF コマンドのタイプ方法を以下に示します。

```
CRTLF   FILE(DSTPRODLB/ORDHDRL)
        TEXT('Order header logical file')
```

示されているように、このコマンドではいくつかの省略時が使用されています。たとえば、SRCFILE および SRCMBR パラメーターが指定されていないため、システムは IBM 提供のソース・ファイル QDDSSRC からの DDS を使用し、ソース・ファイル・メンバー名は ORDHDRL (CRTLF コマンドで指定されているファイル名と同じ) になります。同じ名前のメンバーを持つファイル ORDHDRL はライブラリー DSTPRODLB に配置されます。

1 つの物理ファイルに複数の論理ファイルを作成することができます。1 つの物理ファイルに作成できる論理ファイルの最大数は 32K です。

論理ファイルに対するこのほかの操作については、以下のトピックを参照してください。

- 『複数のレコード様式を持つ論理ファイルの作成』
- 185 ページの『複数の様式を持つファイルに追加するレコード様式の識別』

ビューは、論理ファイルと同様です。 iSeries ナビゲーターまたは CREATE VIEW SQL ステートメントを使用して、ビューを作成できます。詳しくは、以下を参照してください。

- iSeries ナビゲーターを使用したビューの作成と使用
- CREATE VIEW

複数のレコード様式を持つ論理ファイルの作成: 複数様式論理ファイルを用いると、1 つの論理ファイルを参照するだけで複数の物理ファイルからの関連レコードが使用できるようになります。おのおののレコード様式は、常に 1 つまたは複数の物理ファイルと関連しています。複数のレコード様式で同じ物理ファイルを使用することができます。

以下は、フィールド参照ファイルから作成される物理ファイル ORDDTLP の DDS を示しています。

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A* ORDER DETAIL FILE (ORDDTLP) - PHYSICAL FILE RECORD DEFINITION
     A                                     REF(DSTREF)
     A          R ORDDTL                   TEXT('Order detail record')
     A            CUST      R
     A            ORDER     R
     A            LINE      R
     A            ITEM      R
     A            QTYORD    R
     A            DESCRP    R
     A            PRICE     R
```

```
     A            EXTENS    R
     A            WHSLOC    R
     A            ORDATE    R
     A            CUTYPE    R
     A            STATE     R
     A            ACTMTH    R
     A            ACTYR     R
     A
```

以下の例は、物理ファイル ORDHDRP の DDS を示しています。

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A* ORDER HEADER FILE (ORDHDRP) - PHYSICAL FILE RECORD DEFINITION
     A                                     REF(DSTREFP)
     A          R ORDHDR                   TEXT('Order header record')
     A            CUST      R
     A            ORDER     R
     A            ORDATE    R
     A            CUSORD    R
     A            SHPVIA    R
     A            ORDSTS    R
     A            OPRNME    R
     A            ORDMNT    R
     A            CUTYPE    R
     A            INVNBR    R
     A            PRTDAT    R
     A            SEQNBR    R
     A            OPNSTS    R
     A            LINES     R
     A            ACTMTH    R
     A            ACTYR     R
     A            STATE     R
     A
```

次の例では、2 つのレコード様式を持った論理ファイル ORDFILL を作成する方法を示します。1 つのレコード様式は物理ファイル ORDHDRP からの発注ヘッダー・レコードに対して定義し、もう 1 つのレコード様式は物理ファイル ORDDTLP からの発注明細レコードに対して定義しています。

論理ファイルのレコード様式 ORDHDR は、順序付けのために 1 つのキー・フィールド *Order* を使用しています。また、論理ファイルのレコード様式 ORDDTL は、順序付けのために *Order* と *Line* の 2 つのキー・フィールドを使用しています。

以下の例は、論理ファイル ORDFILL の DDS を示しています。

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A* ORDER TRANSACTION LOGICAL FILE (ORDFILL)
     A          R ORDHDR                   PFILE(ORDHDRP)
     A          K ORDER
     A
     A          R ORDDTL                   PFILE(ORDDTLP)
     A          K ORDER
     A          K LINE
     A
```

2 つの関連する物理ファイルを持つ論理ファイル ORDFILL を作成するためには、以下のように論理ファイルの作成 (CRTLF) コマンドを使用します。

```
CRTLF  FILE(DSTPRODLB/ORDFILL)
       TEXT('Order transaction logical file')
```

DDS ソース・コードは、ファイル QDDSSRC のメンバー ORDFILL の中にあります。同じ名前のメンバーを持つファイル ORDFILL は、DSTPRODLB ライブラリーに配置されます。論理ファイル・メンバー ORDFILL のアクセス・パスは、 ORDHDRP と ORDDTLP の両ファイルからのレコードを配列します。

両方の物理ファイルのレコード様式は、共通のフィールドである *Order* がキーとなっています。それらのレコード様式は、それらが論理ファイル記述で指定された順序のために、最初にヘッダー・ファイル ORDHDRP から、次に明細ファイル ORDDTLP から取り出されたファイル間の重複を使用して、*Order* の順序で組み合わせられます。FIFO、LIFO、あるいは FCFO を指定していないので、同じファイル内の重複キーを取り出す順序は保証されません。

注: ある条件下では、複数様式論理ファイルを使用するよりも、複数の論理ファイルを使用した方が良い場合があります。たとえば、キー順アクセスが複数様式論理ファイルで使用されたとき、ファイルの 1 つにほとんどレコードがない場合には、パフォーマンスが低下する可能性があります。複数様式であっても、論理ファイルは 1 つの索引 (それぞれの物理ファイルからの項目が入った) しか持ちません。アプリケーション・プログラムにより行われている処理の種類により (たとえば、小さいファイルを処理するキーとともに RPG SETLL と READE を使用している場合) 、システムは小さいファイルの項目を見つけるためにすべての索引項目を検索しなければならないかもしれません。索引が多くの項目をもつ場合は、各ファイルのキーの数と索引のキーの順序によっては、索引検索にかなりの時間がかかるかもしれません。(小さいファイルにレコードがない場合は、パフォーマンスは影響を受けません。なぜなら、システムは高速パスを使用し、索引を検索しないからです。)

複数様式のファイルについて、詳しくは以下のトピックを参照してください。

- 『複数の様式を持つファイル内でのレコードの取り出し方法の制御』
- 45 ページの『複数の様式を持つファイルへのレコードの追加方法の制御』

複数の様式を持つファイル内でのレコードの取り出し方法の制御: 論理ファイルに複数のレコード様式がある場合、キー・フィールド定義が必要となります。おのおののレコード様式には独自のキー定義があり、レコード様式のキー・フィールドを別の様式のレコードと組み合わせるように定義することができます。おのおののレコード様式ではキーにすべてのキー・フィールドを含める必要はありません。たとえば、以下のようなレコードがあるとします。

ヘッダー・レコードの様式

レコード	Order	Cust	Ordate
1	41882	41394	050688
2	32133	28674	060288

明細レコードの様式

レコード	Order	Line	Item	Qtyord	Extens
A	32133	01	46412	25	125000
B	32133	03	12481	4	001000
C	41882	02	46412	10	050000
D	32133	02	14201	110	454500
E	41882	01	08265	40	008000

DDS では、ヘッダー・レコードの様式をまず定義してから明細レコードの様式を定義します。アクセス・パスが両方のレコード様式の最初のキー・フィールドとして *Order* フィールドを使用し、 2 番目のレコード様式だけの 2 番目のキー・フィールドとして *Line* フィールドを使用する (両方とも昇順で) 場合、アクセス・パスのレコードの順序は以下のようになります。

レコード 2
レコード A
レコード D

レコード B
レコード 1
レコード E
レコード C

注: 重複キー値を持つレコードは、まず、物理ファイルが指定されている順序で配列されます。その後、1 つのレコード様式の中にまだ重複が存在する場合には、重複するレコードは FIFO、LIFO、または FCFO キーワードに指定されている順序で配列されます。たとえば、論理ファイルに DDS キーワード FIFO を指定している場合、その様式内で重複するレコードは先入れ先出しの順序で示されることになります。

複数レコード様式の論理ファイルの場合、キー・フィールドについて *NONE DDS 機能を使用して、 1 つのレコード様式のレコードを、同じアクセス・パスにある他のレコード様式のレコードから分離することができます。一般的に、すべてのレコード様式からのレコードはキー値に基づいて組み合わされます。 ただし、DDS で 1 つのキー・フィールドに *NONE を指定している場合には、*NONE の前のすべてのレコード様式に現れるキー・フィールドを持つレコードだけが組み合わされます。複数のレコード様式からキーによってそのようなレコードが検索されるとき、 *NONE の前のすべてのレコード様式に含まれるキー・フィールドだけが使用されます。使用されるキー・フィールドの数を増やすには、扱うレコード様式の数を制限してください。

次の例の論理ファイルは 3 つのレコード様式を含んでおり、それぞれ異なる物理ファイルと関連しています。

レコード様式	物理ファイル	キー・フィールド
EMPMSTR	EMPMSTR	Empnbr (従業員番号) 1
EMPHIST	EMPHIST	Empnbr、Empdat (採用日) 2
EMPEDUC	EMPEDUC	Empnbr、Clsnbr (クラス番号) 3

注: すべてのレコード様式には、1 つの共通のキー・フィールドである *Empnbr* フィールドがあります。

この例の DDS は以下のようになります。

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
      A
      A          K EMPNBR   1
      A
      A          K EMPNBR   2
      A          K EMPDAT
      A
      A          K EMPNBR   3
      A          K *NONE
      A          K CLSNBR
      A
```

*NONE は、EMPMSTR の 2 番目および 3 番目のフィールドに、および EMPHIST の 3 番目のフィールドにあるものとみなされます。なぜなら、これらのキー・フィールドの位置の後にはキー・フィールドが何もないからです。

レコードの配列は、以下のようになります。

Empnbr	Empdat	Clsnbr	レコード様式名
426			EMPMSTR
426	6/15/74		EMPHIST
426		412	EMPEDUC

Empnbr	Empdat	Clsnbr	レコード様式名
426		520	EMPEDUC
427			EMPMSTR
427	9/30/75		EMPHIST
427		412	EMPEDUC

*NONE は、レコード様式 EMPHIST と EMPEDUC の区切り記号として働きます。同じ *Empnbr* フィールドのある EMPHIST のすべてのレコードはまとめてグループ化され、*Empdat* フィールドによって分類されます。同じ *Empnbr* フィールドのある EMPEDUC のすべてのレコードはまとめてグループ化され、*Clsnbr* フィールドによって分類されます。

注: 上記の順序付けを保証するために、キー順アクセス・パスに追加のキー・フィールド値が入れられているため、重複キー値は予測不可能です。

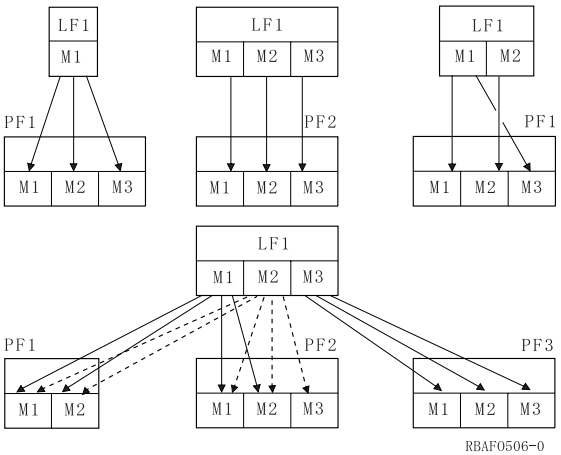
*NONE DDS 機能の別の例については、DDS 解説書を参照してください。

複数の様式を持つファイルへのレコードの追加方法の制御: 複数様式論理ファイルにレコードを追加するには、レコードを書き込みたい、基礎となっている物理ファイルのメンバーを識別します。使用しているアプリケーション・プログラムでは様式中の特定メンバーを指定できない場合には、論理ファイルのおのおのの様式を 1 つの物理ファイル・メンバーと関連付けなければなりません。基礎とする 1 つ以上の物理ファイルに複数のメンバーがある場合、 『論理ファイル・メンバーの定義』で述べている DTAMBRS パラメーターを使用して、 1 つのメンバーをおのおのの様式に固有に関係付けるようにします。最後に、複数様式論理ファイルのおのおのの様式に固有の名前を付けます。複数様式論理ファイルをこのような方法で定義すると、追加操作で様式名を指定するとき、それはレコードを追加する特定の物理ファイル・メンバーとなります。

複数様式の論理ファイルにレコードを追加する場合、アプリケーション・プログラムがレコード様式名ではなくファイル名を使用するときは、様式選択プログラムを作成する必要があります。様式選択プログラムの詳細については、 185 ページの『複数の様式を持つファイルに追加するレコード様式の識別』を参照してください。

論理ファイル・メンバーの定義: データを論理グループに分けるために、論理ファイルにメンバーを定義することができます。論理ファイル・メンバーは、1 つの物理ファイル・メンバーあるいは数個の物理ファイル・メンバーと関連付けることができます。

以下にこの概念を示します。



論理ファイルの中ですべての論理メンバーとともに使用されるレコード様式は、ファイルを作成するとき DDS で定義しなければなりません。新しいレコード様式が必要となった場合は、別の論理ファイルまたはレコード様式を作成しなければなりません。

アクセス・パスの属性は、論理ファイルの作成時に DDS およびコマンドで指定された情報によって判別されます。データ・メンバーの選択は、論理ファイルの作成 (CRTLF) および論理ファイル・メンバーの追加 (ADDLFM) コマンドの DTAMBRS パラメーターで指定されます。

論理ファイルを定義するときには、レコード・レベルの PFILE キーワードまたは JFILE キーワードを用いて、その論理ファイルが使用する物理ファイルを DDS で指定します。複数レコード様式を DDS に定義している場合は、おのおののレコード様式に対して PFILE キーワードを指定しなければなりません。おのおのの PFILE キーワードには複数の物理ファイルを指定することができます。

論理ファイルを作成するか、またはファイルにメンバーを追加する際には、論理ファイルの作成 (CRTLF) または論理ファイル・メンバーの追加 (ADDLFM) コマンドで DTAMBRS パラメーターを使用して、論理ファイルによって使用される物理ファイルのどのメンバーがデータに使用されるかを指定することができます。 物理ファイルのどのメンバーもデータとして使用しない場合には、物理ファイル・メンバー名として *NONE を指定することができます。

次の例では、論理ファイルで 2 つのレコード様式が定義されています。

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A
00010A          R LOGRCD2                   PFILE(PF1 PF2)
     A            .
     A            .
     A            .
00020A          R LOGRCD3                   PFILE(PF1 PF2 PF3)
     A            .
     A            .
     A            .
     A
```

ここで、以下のように、DTAMBRS パラメーターを CRTLF コマンドまたは ADDLFM コマンドで指定したとします。

```
DTAMBRS((PF1 M1) (PF2 (M1 M2)) (PF1 M1) (PF2 (*NONE)) (PF3 M3))
```

これによって、レコード様式 LOGRCD2 は PF1 の物理ファイル・メンバー M1、および PF2 の M1 と M2 に関連付けられます。レコード様式 LOGRCD3 は PF1 の M1、および PF3 の M3 に関連付けられます。PF2 のどのメンバーも LOGRCD3 とは関連付けられません。複数の PFILE キーワードで同じ物理ファイル名を指定すると、それらの物理ファイル名のおのおのは異なる物理ファイルとして扱われます。

PFILE キーワードのファイルに対してライブラリー名を指定していなかった場合、論理ファイルが作成されるときに、ライブラリー・リストを用いて物理ファイルが見つけ出されます。これ以降は、物理ファイル名とライブラリー名が論理ファイル記述の一部分となります。DTAMBRS パラメーターで指定する物理ファイル名とライブラリー名は、論理ファイル記述に保管されているものと同じでなければなりません。

DTAMBRS パラメーターでファイル名がライブラリー名により修飾されていない場合、ライブラリー名は省略時値の *CURRENT となり、システムはそれぞれの物理ファイル名に対して、論理ファイル記述に保管されているライブラリー名を使用します。このライブラリー名は、DDS の PFILE キーワードでファイルについて指定されたライブラリー名か、または論理ファイルの作成時にライブラリー・リストを用いてファイルが検出されたライブラリーの名前です。

メンバーを論理ファイルに追加する場合、以下のようにデータ・メンバーを指定することができます。

- 関連する物理ファイル・メンバーを指定しない (省略時値は DTAMBRS (*ALL))。論理ファイル・メンバーは、論理ファイルの DDS で指定しているすべての PFILE キーワード中の、すべての物理ファイルのすべての物理ファイル・メンバーに関連付けられます。
- 関連する物理ファイル・メンバーを指定する (DTAMBRS パラメーター)。ライブラリー名を指定しなかった場合、使用するライブラリーを論理ファイルが決定します。1 つの物理ファイルに複数の物理ファイル・メンバーを指定している場合、このようなメンバーの間で重複キー値が発生したときにレコードが取り出される順序でメンバー名を指定しなければなりません。特定の物理ファイルからはメンバーを一切取り込みたくない場合、物理ファイル名を指定しないか、または物理ファイル名を指定した上でメンバー名に *NONE を指定します。この方法は、論理ファイルに定義するレコード様式のサブセットを含む論理ファイル・メンバーを定義するのに使用することができます。

論理ファイルを作成する際には、論理ファイルの作成 (CRTLF) コマンドを使用して最初のメンバーを作成することができます。これ以降のメンバーは、論理ファイル・メンバーの追加 (ADDLFM) コマンドを使用して追加しなければなりません。ただし、今後もメンバーをさらに追加する予定がある場合には、CRTLF コマンドで MAXMBRS パラメーターに 1 より大きい数を指定しなければなりません。論理ファイルにメンバーを追加する次の例では、 40 ページの『論理ファイルの作成』で使用されている CRTLF コマンドを使用しています。

```
CRTLF   FILE(DSTPRODLB/ORDHDRL)
        MBR(*FILE)  DTAMBRS(*ALL)
        TEXT('Order header logical file')
```

*FILE は MBR パラメーターの省略時であり、メンバーの名前がファイルの名前と同じであることを意味します。関連する物理ファイル (ORDHDRP) のすべてのメンバーは、論理ファイル (ORDHDRL) のメンバーで使用されます。テキスト記述はメンバーのテキスト記述です。

論理ファイルのレコード様式の記述

DDS で記述するそれぞれの論理ファイル・レコード様式について、レコード様式名と、 PFILE キーワード (単一または複数様式論理ファイルの場合) または JFILE キーワード (結合論理ファイルの場合) を指定しなければなりません。 PFILE あるいは JFILE キーワードで指定するファイル名は、論理ファイルの基礎とする物理ファイルです。 単一あるいは複数様式論理ファイルのレコード様式は、以下のいずれかの方法を用いて DDS に指定することができます。

- 単純な論理ファイル・レコード様式では、レコード様式名と PFILE キーワードだけを指定します。PFILE キーワードで指定された唯一 (または最初) の物理ファイルのレコード様式が論理ファイルのレコード様式となります。論理ファイルで指定するレコード様式名は、唯一 (または最初) の物理ファイルのレコード様式名と同一でなければなりません。以下の単純な論理ファイルの例を見てください。

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A
     A          R ORDDTL                    PFILE(ORDDTLP)
     A
```

- 含めたいフィールド名をリストすることによって、独自のレコード様式を記述できます。フィールド名を別の順序で指定したり、RENAME キーワードを用いてフィールドの名前を変更したり、CONCAT キーワードを用いてフィールドを連結したり、SST キーワードを用いてフィールドの特定の桁を使用したりすることができます。 また、論理ファイルに別の属性を指定することによって、フィールドの属性を指定変更することもできます。以下は、フィールドが指定された単純な論理ファイルの例です。

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A
     A          R ORDHDR                    PFILE(ORDHDRP)
```

```
A            ORDER
A            CUST
A            SHPVIA
A
```

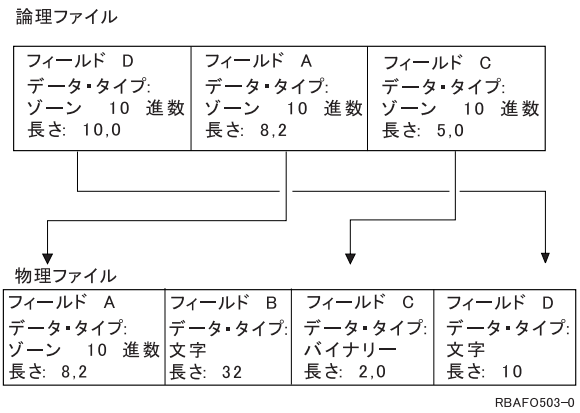
- ファイル名に対するデータベース・ファイル名を FORMAT キーワードで指定します。レコード様式は、記述されている論理ファイルにより、このデータベース・ファイルから共用されます。ファイル名はライブラリー名で修飾することができます。ライブラリー名を指定していない場合は、ファイルを見つけ出すのにライブラリー・リストが使用されます。記述しているファイルを作成するには、ファイルが必ず存在していなければなりません。さらに、論理ファイルで指定したレコード様式名は、FORMAT キーワードで指定したファイルのレコード様式名の 1 つと同じでなければなりません。以下に、例を示します。

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A
     A          R CUSRCD                    PFILE(CUSMSTP)
     A                                      FORMAT(CUSMSTL)
     A
```

次の例で、プログラムは以下のことを行う必要があります。

- フィールドを別の順番に配置する。
- 物理ファイルからフィールドのサブセットを取り出す。
- 一部のフィールドのデータ・タイプを変更する。
- 一部のフィールドのフィールド長を変更する。

論理ファイルを使用すると、これらの変更を行うことができます。



論理ファイルの DDS は以下のようになります。

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A
     A          R LOGREC                    PFILE(PF1)
     A            D             10S 0
     A            A
     A            C              5S 0
     A
```

物理ファイルの DDS は以下のようになります。

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A
     A          R PHYREC
     A            A              8S 2
```

```
A            B            32
A            C             2B 0
A            D            10
A
```

論理ファイルからレコードが読み取られるとき、物理ファイルのフィールドは論理ファイルの記述に適合するように変更されます。プログラムがレコードを更新または追加すると、フィールドは元に戻されます。論理ファイルを用いて追加または更新の操作を行うためには、プログラムは論理ファイルが使用する様式に適合するデータを提供しなければなりません。

次の図は、物理ファイルと論理ファイルの間で有効なデータ・マッピングのタイプをまとめたものです。

	論理ファイルのデータ・タイプ							
物理ファイルのデータ・タイプ	**文字または 16 進数**	**ゾーン 10 進数**	**パック 10 進数**	**2 進数**	**浮動小数点数**	**日付**	**時刻**	**時刻スタンプ**
文字または 16 進数	有効	注 1 を参照	無効	無効	無効	無効	無効	無効
ゾーン 10 進数	注 1 を参照	有効	有効	注 2 を参照	有効	無効	無効	無効
パック 10 進数	無効	有効	有効	注 2 を参照	有効	無効	無効	無効
2 進数	無効	注 2 を参照	注 2 を参照	注 3 を参照	注 2 を参照	無効	無効	無効
浮動小数点数	無効	有効	有効	注 2 を参照	有効	無効	無効	無効
日付	無効	有効	無効	無効	無効	有効	無効	無効
時刻	無効	有効	無効	無効	無効	無効	有効	無効
時刻スタンプ	無効	無効	無効	無効	無効	有効	有効	有効
注: 1. 文字数またはバイト数が桁数と等しい場合のみ有効。 2. 2 進数フィールドの小数点以下の桁数がゼロの場合のみ有効。 3. 両方の 2 進数フィールドの小数点以下の桁数が等しい場合のみ有効。								

注: DBCS フィールドについては、2 バイト文字セット (DBCS) に関する考慮事項を参照してください。

論理ファイルのレコード様式の記述について、以下の関連トピックも参照してください。

- 『論理ファイルのフィールドの用途の記述』
- 51 ページの『既存のフィールドからの新しいフィールドの抽出』
- 53 ページの『論理ファイル内の浮動小数点フィールドの記述』

論理ファイルのフィールドの用途の記述: データベース・ファイルのフィールドが入力専用か、入出力共用か、または非入出力フィールドかを指定することができます。このためには、38 列に以下のうち 1 つを指定します。

記入項目
意味

ブランク
単一または複数様式論理ファイルの場合、省略時値は B (入出力共用) になります。結合論理ファイルの場合、省略時値は I (入力専用) になります。

B 入力と出力の両方が許可されます。結合論理ファイルについては無効です。『論理ファイルのフィールドの用途の記述: 入出力共用』を参照してください。

I 入力専用 (読み取り専用)。『論理ファイルのフィールドの用途の記述: 入力のみ』を参照してください。

N 非入出力。結合論理ファイルについてのみ有効です。『論理ファイルのフィールドの用途の記述: 非入出力』を参照してください。

注: 用途値 (38 列) は参照機能では使用されません。他のファイルが論理ファイルのフィールドを (REF または REFFLD キーワードを用いて) 参照するときには、用途値はそのファイルにコピーされません。

論理ファイルのフィールドの用途の記述: 入出力共用: 入出力共用フィールドは入力と出力の両方の操作に使用することができます。プログラムでは、フィールドからデータを読み取ったり、フィールドにデータを書き込んだりすることができます。入出力共用フィールドは、結合論理ファイルについては無効です。これは、結合論理ファイルが読み取り専用ファイルであるためです。

論理ファイルのフィールドの用途の記述: 入力のみ: 入力専用フィールドは読み取り操作だけに使用することができます。プログラムでは、フィールドからデータを読み取ることはできても、ファイルのフィールドを更新することはできません。入力専用フィールドの典型的な例としては、キー・フィールド (キー・フィールド値の変更を防ぐことによってアクセス・パスのメインテナンスを減らすため)、ユーザーが参照できても更新できない重要なフィールド (たとえば、給与)、および変換表 (TRNTBL) キーワードまたはサブストリング (SST) キーワードが指定されているフィールドがあります。

入力専用フィールドのあるレコードをプログラムで更新しても、入力専用フィールドはファイル内で更新されません。入力専用フィールドのあるレコードをプログラムで追加すると、入力専用フィールドは省略時値 (DFT キーワード) を取ります。

論理ファイルのフィールドの用途の記述: 非入出力: 非入出力フィールドは、入力にも、出力にも使用されません。これは結合論理ファイルについてのみ有効です。非入出力フィールドは結合論理ファイル内の結合フィールドとして使用できますが、プログラムでは非入出力フィールドを読み取ったり、更新したりすることはできません。

物理ファイルの結合フィールドの属性が一致しないときに、非入出力フィールドを使用します。このような場合、一方または両方の結合フィールドを定義し直さなければなりません。ただし、このように再定義したフィールドは、レコード様式に含めることはできません (アプリケーション・プログラムは再定義されたフィールドを参照しません)。したがって、再定義する結合フィールドは、レコード様式の中に現れないように N とコーディングすることができます。

38 列が N であるフィールドは、プログラムが使用するバッファーには現れません。ただし、フィールド記述はファイル・フィールド記述の表示 (DSPFFD) コマンドで表示することができます。

非入出力フィールドは、選択/除外フィールドあるいはキー・フィールドとして使用することはできません。

非入出力フィールドの例については、74 ページの『レコード様式に現れないフィールドの記述 (例 5)』を参照してください。

既存のフィールドからの新しいフィールドの抽出: 論理ファイルのフィールドは、論理ファイルの基礎となる物理ファイルのフィールドから、または同じ論理ファイルのフィールドから抽出することができます。たとえば、物理ファイルから複数のフィールドを CONCAT キーワードを使用して連結し、論理ファイルの 1 つのフィールドのように表示することができます。 同様に、物理ファイルの 1 つのフィールドを分割し、SST キーワードを使用して論理ファイルに複数のフィールドを表示することもできます。

キーワードを使用してフィールドを抽出する方法については、以下のトピックを参照してください。

- 『連結フィールド』
- 52 ページの『サブストリング・フィールド』
- 53 ページの『名前変更フィールド』
- 53 ページの『変換フィールド』

連結フィールド: CONCAT キーワードを使用して物理ファイルのレコード様式からの複数のフィールドを連結し、論理ファイルのレコード様式の 1 つのフィールドとすることができます。たとえば、物理ファイルのレコード様式が、*Month*、*Day*、および *Year* のフィールドを含んでいるとします。論理ファイルでは、これらのフィールドを *Date* という 1 つのフィールドに連結することができます。

この結果、連結フィールドのフィールド長は、含まれたフィールドの長さを合計したものとなります (ただし物理ファイルのフィールドが 2 進数またはパック 10 進数の場合は、ゾーン 10 進数に変更されます) 。結果として生じたフィールドの長さは、システムで自動的に計算されます。連結フィールドには以下のものを指示することができます。

- 列見出し
- 妥当性検査
- テキスト記述
- 編集コードまたは編集ワード (数字の連結フィールドのみ)

注: この編集および妥当性検査情報は、データベース管理システムで使用されるのではなく、データベース・ファイルからのフィールド記述が表示装置ファイルまたは印刷装置ファイルで参照されるときに検索されます。

フィールドが連結されると、データ・タイプが変更されることがあります (結果のデータ・タイプはシステムで自動的に決定されます)。次の規則と制限事項が適用されます。

- OS/400 プログラムは、連結するフィールドのデータ・タイプに基づきデータ・タイプを割り当てます。
- 連結フィールドの最大長は、連結フィールドのデータ・タイプと連結するフィールドの長さによって変化します。連結フィールドがゾーン 10 進数 (S) ならば、合計長は 31 バイトを超えることはできません。文字 (A) ならば、合計長は 32766 バイトを超えることはできません。
- 結合論理ファイルでは、連結するフィールドは同じ物理ファイルからのものでなければなりません。CONCAT キーワードに指定する最初のフィールドにより、使用される物理ファイルが識別されます。このため、最初のフィールドは、論理ファイルの基礎となっている物理ファイルの中で固有でなければなりません。もしくは、どの物理ファイルを使用するか指定するために、 JREF キーワードも指定しなければなりません。
- 連結フィールドの用途は、連結フィールドが可変長ならば I (入力専用) でなければなりません。そうでない場合には、B (入出力共用) でもかまいません。
- O または J のデータ・タイプを割り当てた連結フィールドでは、 REFSHIFT を指定することはできません。
- いずれかのフィールドが NULL を含んでいる場合、連結の結果は NULL となります。

注: DBCS フィールドの連結については、2 バイト文字セット (DBCS) に関する考慮事項を参照してください。

数字フィールドだけを連結する場合は、グループの最後のフィールドの符号が連結フィールドの符号として使用されます。

注:

1. ゼロ以外の 10 進数精度を持つ数字フィールドは、連結フィールドに含めることができません。
2. 日付、時刻、時刻スタンプ、および浮動小数点フィールドは連結フィールドに含めることができません。

以下に、連結を行うための DDS でのフィールド記述を示します。(連結するフィールドを指定するために CONCAT キーワードを使用しています。)

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A
00101A            MONTH
00102A            DAY
00103A            YEAR
00104A            DATE                      CONCAT(MONTH DAY YEAR)
     A
```

この例では、論理ファイルのレコード様式には、連結フィールドの *Date* だけでなく、 *Month*、*Day*、および *Year* という別個のフィールドも含まれています。以下のいずれの様式も使用することができます。

- *Month*、*Day*、および *Year* という別個のフィールドがある様式。
- 連結フィールドの *Date* だけがある様式。
- *Month*、*Day*、および *Year* という別個のフィールドと連結フィールドの *Date* がある様式。

別個のフィールドと連結フィールドの両方が様式の中にあるときは、フィールドの更新は必ず DDS で指定された順序で処理されます。前の例で *Date* フィールドが 103188 であり、 *Month* フィールドを 12 に変更したとすると、レコードを更新するときには *Date* フィールド内の月が使用されます。更新したレコードには、103188 が入ることになります。*Date* フィールドを最初に指定した場合には、更新したレコードには 123188 が入ることになります。

連結フィールドは、キー・フィールドや選択/除外フィールドとして使用することもできます。

サブストリング・フィールド: SST キーワードを使用すると、サブストリングに入るフィールド (文字、16 進数、またはゾーン 10 進数) を指定することができます。(論理ファイルでデータ・タイプとして S (ゾーン 10 進数) を指定することにより、物理ファイル内のパック 10 進数フィールドを含むサブストリングを使用することもできます。) たとえば、物理ファイル *PF1* で 6 文字の長さの *Date* フィールドを定義したとします。この場合、3 つのフィールド (それぞれ 2 文字の長さ) を持つ論理ファイルを記述することができます。SST キーワードを使用して、 MM を *Date* フィールドの 1 桁目から始まる 2 文字として、 DD を *Date* フィールドの 3 桁目から始まる 2 文字として、 YY を *Date* フィールドの 5 桁目から始まる 2 文字として定義することができます。

以下に、これらのサブストリング・フィールドについての DDS のフィールド記述を示します。SST キーワードは、サブストリングのフィールドを指定するために使用されます。

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A          R REC1                       PFILE(PF1)
     A
     A            MM                  I      SST(DATE 1 2)
     A            DD                  I      SST(DATE 3 2)
     A            YY                  I      SST(DATE 5 2)
     A
```

サブストリングの開始位置が、ファイル内の位置ではなく、操作対象のフィールド *(Date)* 内の位置によって指定されていることに注意してください。「用途」列の I は、入力専用を示しています。

サブストリング・フィールドは、キー・フィールドおよび選択/除外フィールドとしても使用することができます。

名前変更フィールド: RENAME キーワードを使用すると、物理ファイルとは異なる名前を論理ファイルのフィールドに付けることができます。 プログラムが異なるフィールド名を使用して作成されていたり、元のフィールド名が、使用している高水準言語の命名制限に適合しない場合、論理ファイル内でフィールドの名前を変更することが必要な場合もあります。

変換フィールド: TRNTBL キーワードを使用して、フィールドの変換表を指定することができます。 論理ファイルのレコードを読み取ったときに、論理ファイルの複数のフィールドに対して変換表が指定されていると、システムは物理ファイルのフィールド値から変換表で決められた値へとデータを変換します。

論理ファイル内の浮動小数点フィールドの記述: 論理ファイルのマップ・フィールドとして浮動小数点フィールドを使用することができます。単精度または倍精度の浮動小数点フィールドは、ゾーン、パック、ゼロ精度の 2 進数、または他の浮動小数点フィールドへのマッピングや、このようなフィールドからのマッピングを行うことができます。浮動小数点フィールドと、非ゼロ精度 2 進数フィールド、文字フィールド、 16 進数フィールド、または DBCS フィールド間ではマッピングを行うことはできません。

精度の異なる (単精度または倍精度) 浮動小数点フィールド間のマッピング、または浮動小数点フィールドと他の数字フィールドの間のマッピングは、丸めや精度の損失につながります。倍精度浮動小数点数を単精度浮動小数点数にマッピングすると、関与した数値とその内部表現によっては、丸めが生じる可能性があります。丸めが生じると、最も近い (偶数の) ビットになります。結果は常に、できるだけ高い精度に保たれます。精度の桁数が減少する場合には、2 つの 10 進数の間でも精度が失われる可能性があります。

プログラムで明示的に変更しなかったフィールド値が意図せずに変更されることがあります。浮動小数点フィールドの場合、物理ファイルが倍精度フィールドとなっており、それが論理ファイルの単精度フィールドにマッピングされ、論理ファイルを通してレコードが更新されると、このようなことが起こります。浮動小数点数の内部表現が、論理ファイルにマッピングされたとき丸められると、論理レコードを更新することによって物理ファイルの精度を永久に失うことになります。丸められた数値が物理レコードのキーならば、物理ファイルのレコードの順序も同様に変更されることになります。

精度が論理ファイルで低下すると、固定小数点フィールドも不注意に更新されることがあります。

論理ファイルのアクセス・パスの記述

論理ファイルのレコード様式のアクセス・パスは、以下の方法のいずれかで指定することができます。

- キー順アクセス・パス仕様。最後のレコードまたはフィールド・レベル仕様の後にキー・フィールドを指定します。レコード様式にはキー・フィールド名がなければなりません。結合論理ファイルの場合、キー・フィールドは最初または基本の物理ファイルからのものでなければなりません。

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A          R CUSRCD                    PFILE(CUSMSTP)
     A          K ARBAL
     A          K CRDLMT
     A
```

- コード化ベクトル・アクセス・パス仕様。コード化ベクトル・アクセス・パスは、SQL CREATE INDEX ステートメントを使用して定義します。

- 到着順アクセス・パス仕様。キー・フィールドは指定しません。PFILE キーワードには物理ファイルを 1 つのみ指定することができます (論理ファイルのメンバーを追加するときは、物理ファイルのメンバーの中の 1 つのみ指定することができます)。

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A          R CUSRCD                      PFILE(CUSMSTP)
```

- 事前に定義されたキー順アクセス・パス仕様 (単一および複数様式論理ファイルの場合のみ)。REFACCPTH キーワードをファイル・レベルで指定して、アクセス・パスおよび選択/除外仕様がこの論理ファイルにコピーされる、事前に作成されたデータベース・ファイルを識別します。 REFACCPTH キーワードで個々のキーまたは選択/除外フィールドを指定することはできません。

  注: 指定したファイルのアクセス・パス仕様が使用されたとしても、どのファイルのアクセス・パスを実際に共用するかは、システムが決定します。 REFACCPTH キーワードが使用されているかどうかを問わず、システムは常にアクセス・パスを共用しようとします。

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
                                     REFACCPTH(DSTPRODLIB/ORDHDRL)
     A          R CUSRCD                      PFILE(CUSMSTP)
```

別のファイルのアクセス・パスのキー・フィールド仕様を共用する論理ファイルのレコード様式を定義する場合 (DDS キーワードの REFACCPTH を使用して) には、関連する物理ファイルのレコード様式からどのようなフィールドでも使用することができます。このようなフィールドは、アクセス・パスを記述するファイルでは使用する必要はありません。ただし、アクセス・パスを記述するファイルで使用されるキーおよび選択/除外フィールドはすべて、新しいレコード様式で使用されなければなりません。

論理ファイルのアクセス・パスの記述について、詳しくは以下のトピックを参照してください。

- 『論理ファイルを用いてのレコードの選択と除外』
- 58 ページの『既存のアクセス・パスの使用』

論理ファイルを用いてのレコードの選択と除外: システムは、論理ファイルを使用するときにレコードの選択と除外を行うことができます。このことは、処理の便宜上あるいは安全を目的とするために、ファイルのレコードを除外する際に役立ちます。

レコードの選択と除外の処理は、論理ファイルの DDS 形式の 17 列で識別された比較に基づきます。これは、高水準言語プログラムでコーディングする一連の比較によく似ています。たとえば、発注明細レコードを含む論理ファイルでは、注文量が出荷量よりも多いレコードだけを使用するということを指定できます。その他のレコードはすべてアクセス・パスから除外されます。除外されたレコードは物理ファイルに残りますが、論理ファイルのために取り出されません。物理ファイルにレコードを追加する場合、すべてのレコードが追加されますが、選択/除外基準に適合する選択されたレコードしか、選択/除外アクセス・パスを用いて取り出すことができません。

DDS で選択または除外を指定するためには、DDS 形式の 17 列に S (選択) または O (除外) を指定します。さらに、選択または除外処理で使用されるフィールドの名前を指定します (19 列から 28 列)。45 列から 80 列には比較を指定します。

注: (キーを指定している場合) 選択/除外仕様は、キー仕様の後に表示されます。

レコードは、以下のようないくつかのタイプの比較によって選択および除外することができます。

- VALUES。フィールドの内容は、100 個以下の値のリストと比較されます。 一致するものが見つかると、レコードが選択または除外されます。 次の例では、VALUES キーワードに指定された値の 1 つが *Itmnbr* フィールドで見つかれば、レコードが選択されます。

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A          S ITMNBR                    VALUES(301542 306902 382101 422109 +
     A                                      431652 486592 502356 556608 590307)
     A
```

- RANGE。フィールドの内容は、下限値および上限値と比較されます。 内容が下限値以上で、上限値以下であれば、レコードが選択または除外されます。 次の例では、*Itmnbr* フィールドが 301000から 599999 の範囲となっているレコードがすべて選択されます。

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A          S ITMNBR                    RANGE(301000 599999)
     A
```

- CMP。 フィールドの内容は、ある値または他のフィールドの内容と比較されます。有効な比較コードは、EQ、NE、LT、NL、GT、NG、LE、および GE です。比較の条件が満たされると、レコードが選択または除外されます。次の例では、*Itmnbr* フィールドが 599999 以下の場合に、レコードが選択されます。

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A          S ITMNBR                    CMP(LE 599999)
     A
```

CMP、VALUES、または RANGE キーワードが指定された数字フィールドの値は、そのフィールドについて指定された小数点以下の桁数に基づいて位置合わせされ、必要であればゼロで埋められます。フィールドについて小数点以下の桁数が指定されていない場合、小数点は値の右端の数字の右側に置かれます。たとえば、長さが 5 で小数点以下の桁数が 2 の数字フィールドでは、値 1.2 は 001.20 として解釈され、値 100 は、100.00 と解釈されます。

レコードの状況は、選択/除外ステートメントを指定した順序で評価することによって判別されます。レコードが選択または除外に当てはまると、それ以降のステートメントは無視されます。

通常、選択と除外の比較は相互に独立して取り扱われ、比較は OR の関係にあります。すなわち、選択または除外の比較の条件が満たされると、レコードが選択または除外されます。条件が満たされない場合、システムは次の比較に進みます。比較を接続するためには、DDS 形式の 17 列を NULL にしておきます。この場合、この形で接続されたすべての比較の条件が満たされなければ、レコードは選択または除外されません。すなわち、比較は AND の関係にあります。

比較が少ないほど、作業はより効率的になります。選択/除外の比較をいくつか必要とする場合には、最初に大半のレコードを選択または除外してしまうような比較を指定するようにします。

以下の例には、選択/除外機能をコーディングする方法が示されています。これらの例では、*Rep* フィールドが JSMITH となっているレコードはほとんどありません。これらの例は、ニューヨーク州の JSMITH という名前の販売担当者についての 1988 年以前のレコードをすべて選択するために DDS を使用する方法を示しています。すべては同じ結果ですが、効率は異なります。最も効率的な方法は **3** です。

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A          S ST                        CMP(EQ 'NY')        1
     A            REP                       CMP(EQ 'JSMITH')
     A            YEAR                      CMP(LT 88)
     A

|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A          O YEAR                      CMP(GE 88)          2
     A          S ST                        CMP(EQ 'NY')
     A            REP                       CMP(EQ 'JSMITH')
     A
```

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A          O REP                       CMP(NE 'JSMITH')    3
     A          O ST                        CMP(NE 'NY')
     A          S YEAR                      CMP(LT 88)
     A
```

1 すべてのレコードは、選択フィールドの *St*、*Rep*、および *Year* のすべてと比較された後に、選択または除外されます。

2 すべてのレコードは *Year* フィールドと比較され、その後、1988 年以前のレコードが *St* および *Rep* フィールドと比較されます。

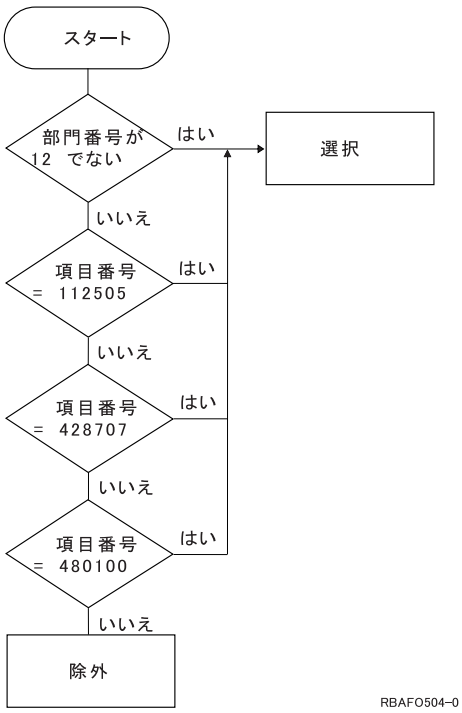
3 すべてのレコードが *Rep* フィールドと比較されます。その後、JSMITH というわずかのレコードが *St* フィールドと比較されます。さらに、残ったわずかのレコードが *Year* フィールドと比較されます。

別の例として、以下のものを選択したいとします。

- 部門が 12 以外のすべてのレコード。
- 部門が 12 であり、かつ、項目番号が 112505、428707、または 480100 となっているレコード。部門 12 のその他のレコードは選択しない。

前述の例を分類順序表を用いて作成する場合は、比較の前に、選択/除外フィールドが分類表に従って変換されます。たとえば、大文字小文字を共通の等価のものとして使用する分類順序表を用いると、NY と ny は同等になります。詳しくは、 DDS に関する情報を参照してください。

次の図は、この例に含まれるロジックを示しています。



RBAFO504-0

以下に、DDS の選択/除外機能を使用して、この例をどのようにコーディングするかを示します。

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A          S DPTNBR                    CMP(NE 12)
     A          S ITMNBR                    VALUES(112505 428707 480100)
     A
```

アクセス・パスに選択/除外値を付けて、ファイルを到着順に処理することもできます。たとえば、高水準言語プログラムでは、キー順のアクセス・パスを無視するように指定することができます。この場合、すべてのレコードが到着順でファイルから読み取られますが、ファイルで指定された選択/除外値を満たすレコードだけが高水準言語プログラムに返されます。

キー・フィールドと選択/除外値が指定されている論理ファイルは、到着順に、または相対レコード番号を用いてランダムに処理することができます。選択/除外値によって除外されたレコードは処理されません。すなわち、除外されたレコードが相対レコード番号で要求された場合には、そのレコードは高水準言語プログラムに返されません。

論理ファイルを介して追加または変更が行われた場合に、同じ論理ファイルの中でレコードに再びアクセスできるかどうかは、システムによって保証されません。たとえば、論理ファイルの選択値で *Fld1* が A となっているレコードだけが指定されており、プログラムでレコードを更新して *Fld1* を B にすると、プログラムはこの論理ファイルを使用してそのレコードを再び取り出すことができません。

注: 浮動小数点フィールドの値に基づいて選択あるいは除外を行うことはできません。

選択/除外の操作には 2 種類があります。この 2 種類とは、アクセス・パスの選択/除外と動的選択/除外です。省略時値はアクセス・パスの選択/除外です。選択/除外の仕様そのものは各種共通ですが、システムが実際にレコードの選択と除外の処理を行う時点が異なります。以下のトピックを参照してください。

- 『アクセス・パスの選択/除外』
- 『動的選択/除外』

さらに、 Query ファイルのオープン (OPNQRYF) コマンドを使用してレコードを選択または除外することもできます。 58 ページの『レコードを選択/除外するための Query ファイルのオープン・コマンドの使用』を参照してください。

アクセス・パスの選択/除外: アクセス・パスの選択/除外を使用すると、アクセス・パスには論理ファイルで指定された選択/除外値を満たすキーだけが含まれます。ファイルにキー・フィールドを指定していた場合には、アクセス・パスはそのファイルに対して保持され、論理ファイルが使用する物理ファイルのレコードをユーザーが追加または変更したときには、システムが保守を行います。アクセス・パス内の索引項目だけが選択/除外値を満たすものになります。

動的選択/除外: 動的選択/除外を使用すると、プログラムがファイルからレコードを読み取るときに、システムは選択/除外値を満たすレコードだけを返します。つまり、実際の選択/除外処理はプログラムがレコードを読み取るときに行われ、レコードが追加または変更されるときには行われません。ただし、キー順アクセス・パスには、選択されたレコードからのキーだけでなく、すべてのキーが含まれています。動的選択/除外を使用するアクセス・パスでは、アクセス・パスの共用がより多く許され、これによってパフォーマンスを向上させることができます。アクセス・パスの共用の詳細については、 58 ページの『既存のアクセス・パスの使用』を参照してください。

動的選択/除外を指定するには、動的選択 (DYNSLT) キーワードを使用します。 動的選択/除外にはキー・フィールドは必要ありません。

更新が頻繁であり、読み取りがあまり行われないファイルがある場合には、プログラムがファイルを読み取るまでは、選択/除外の目的でアクセス・パスを更新する必要はありません。このような場合には、動的選択/除外を選択すると良いでしょう。このことを示すために、以下のような例を挙げます。

変更の頻度が低いコード・フィールド (A = 活動状態、I = 非活動状態) を使用して、レコードの選択/除外を行います。プログラムは活動状態のレコードを処理します。レコードの大半 (80% 以上) は活動状態で

す。コード・フィールドを変更するときにはアクセス・パスを保守するよりも、 DYNSLT を使用して処理時にレコードを動的に選択した方がより効率的です。

レコードを選択/除外するための Query ファイルのオープン・コマンドの使用: レコードを選択するための別の手法として、Query ファイルのオープン (OPNQRYF) コマンドの QRYSLT パラメーターの使用があります。 OPNQRYF コマンドが作成するオープン・データ・パスは一時論理ファイルのようなものであり、クローズしたときに自動的に削除されます。 これとは逆に論理ファイルは、特定して削除しない限り存在し続けます。OPNQRYF コマンドの詳細については、126 ページの『Query ファイルのオープン (OPNQRYF) コマンドの使用』を参照してください。

既存のアクセス・パスの使用: 2 つ以上のファイルが同一の物理ファイルと同一のキー・フィールドを基礎としており、その順序が同じ場合には、同じキー順アクセス・パスを自動的に共用します。アクセス・パスが共用されると、アクセス・パスを保守するのに必要なシステムの活動量、およびファイルが使用する補助記憶域の量が削減されます。

キー順アクセス・パスを持つ論理ファイルを作成する場合、システムは常に既存のアクセス・パスを共用しようとします。アクセス・パスを共用するためには、アクセス・パスは以下の条件を満たしているシステム上になければなりません。

- 追加する論理ファイル・メンバーは、既存のアクセス・パスの基礎である同じ物理ファイルのメンバーを基礎としていなければなりません。
- それぞれのキー・フィールドについて指定する長さ、データ・タイプ、および小数点以下の桁数は、新しいファイルと既存のファイルの両方で同じでなければなりません。
- FIFO、LIFO、または FCFO キーワードを指定していない場合、新しいファイルのキー・フィールドの数は既存のアクセス・パスより少なくてもかまいません。つまり、キーの先頭部分が同一ならば、新しい論理ファイルは既存のアクセス・パスを共用することができます。ただし、ファイルが既存のアクセス・パスの一部分のキーを共用するときは、共用されるアクセス・パスのキーの一部分のフィールドが更新されると、そのアクセス・パス内でのレコード位置が変わることもあります。そのような状況の説明については、59 ページの『暗黙共用アクセス・パスの例』を参照してください。
- アクセス・パスの属性 (たとえば、UNIQUE、LIFO、FIFO、および FCFO) およびキー・フィールドの属性 (たとえば、DESCEND、ABSVAL、UNSIGNED、および SIGNED) は同一でなければなりません。

  例外:

  1. FIFO アクセス・パスは、アクセス・パス共用のための他のすべての要件を満たしている場合、UNIQUE キーワードが指定されているアクセス・パスを共用することができます。
  2. UNIQUE アクセス・パスは、アクセス・パス共用のための他のすべての要件を満たしている場合、再作成する必要のある (たとえば、 *REBLD メインテナンスが指定されている) FIFO アクセス・パスを共用することができます。
- 新しい論理ファイルに選択/除外仕様がある場合には、その仕様は既存のアクセス・パスの選択/除外仕様と同一でなければなりません。ただし、新しい論理ファイルで DYNSLT を指定している場合は、既存のアクセス・パスが以下のいずれかであるときに、既存のアクセス・パスを共用することができます。
  - 動的選択 (DYNSLT) キーワードが指定されている
  - 選択/除外キーワードが指定されていない
- 新しい論理ファイル・メンバーに代替照合順序 (ALTSEQ キーワード) および変換表 (TRNTBL) を指定している場合は、既存のアクセス・パスの代替照合順序および変換表と同一でなければなりません。

注: 連結フィールドあるいはサブストリング・フィールドを含む論理ファイルは、物理ファイルとアクセス・パスを共用することはできません。

アクセス・パスの所有者は、アクセス・パスを最初に作成した論理ファイル・メンバーです。共用アクセス・パスの場合、アクセス・パスを所有する論理メンバーが削除されたときに、アクセス・パスを共用する最初のメンバーが新しい所有者となります。既存のアクセス・パスを共用するために、論理ファイルの作成 (CRTLF) コマンドの FRCACCPTH、MAINT、および RECOVER パラメーターを、そのアクセス・パスの同じパラメーターと一致させる必要はありません。アクセス・パスがいくつかの論理ファイル・メンバーによって共用され、 FRCACCPTH、MAINT、および RECOVER パラメーターが同じでないときには、システムは、共用メンバーによって指定されているそれぞれのパラメーターについて、最も制限の厳しい値によってアクセス・パスを保守します。以下に、このことがどのように起こるかを示します。

MBRA は以下を指定:	FRCACCPTH (*NO) MAINT (*IMMED) RECOVER (*AFTIPL)
MBRB は以下を指定:	FRCACCPTH (*YES) MAINT (*DLY) RECOVER (*NO)
システムによって以下が行われる:	FRCACCPTH (*YES) MAINT (*IMMED) RECOVER (*AFTIPL)

アクセス・パスの共用は、メンバー間での共用には依存しません。したがって、メンバーを削除する順序には制限を与えません。

ファイル記述の表示 (DSPFD) コマンドとデータベース関係の表示 (DSPDBR) コマンドは、アクセス・パス共用の関係を表示します。

暗黙共用アクセス・パスの例: この例は、暗黙のアクセス・パス共用について十分に理解するために役立ちます。

2 つの論理ファイル LFILE1 と LFILE2 が物理ファイル PFILE を基礎にして作成されています。LFILE1 が最初に作成され、これは 2 つのキー・フィールド KFD1 と KFD2 を持ちます。LFILE2 には 3 つのキー・フィールド KFD1、KFD2、および KFD3 があります。2 つの論理ファイルは、2 つの同一キー・フィールドを使用しますが、2 つのキー・フィールドを持つファイルの後に、3 つのキー・フィールドを持つ論理ファイルが作成されたため、アクセス・パスは共用しません。

表 4. 保管と復元の前の物理ファイルと論理ファイル

	物理ファイル (PFILE)	**論理ファイル 1 (LFILE1)**	**論理ファイル 2 (LFILE2)**
アクセス・パス		**KFD1、KFD2**	**KFD1、KFD2、KFD3**
フィールド	**KFD1、KFD2、KFD3、 A、B、C、D、E、F、G**	**KFD1、KFD2、KFD3、 F、C、A**	**KFD1、KFD2、KFD3、 D、G、F、E**

アプリケーションは LFILE1 を使用してレコードにアクセスし、KFD3 フィールドをブランク (C が入っている場合) または C (ブランクの場合) に変更します。このアプリケーションでは、アクセス・パスが共用されないので、ユーザーの予期しない結果は生じません。しかし、物理ファイルと両方の論理ファイルの保管と復元の後は、プログラムが何もしていないように見え、処理に時間がかかります。

復元に何も変更を加えない限り、iSeries システムは次のように作動します。

- 最大数のキーを持つ論理ファイルを最初に復元します。
- 不必要なアクセス・パスは作成しません。

3 つのキー・フィールドを持つので、LFILE2 が最初に復元されます。回復の後、LFILE1 は LFILE2 のアクセス・パスを暗黙に共用します。暗黙に共用されるアクセス・パスについて理解していないユーザーは、回復後に LFILE1 を使用するとき、実際には LFILE2 のキーを使用していることに気付きません。

表 5. 保管と復元の後の物理ファイルと論理ファイル： 保管と復元の前との違いは、論理ファイルが同一のアクセス・パスを共用するようになったことだけであることに注意してください。

	物理ファイル (PFILE)	論理ファイル 1 (LFILE1)	論理ファイル 2 (LFILE2)
アクセス・パス		**KFD1、KFD2、KFD3**	**KFD1、KFD2、KFD3**
フィールド	**KFD1、KFD2、KFD3、A、B、C、D、E、F、G**	**KFD1、KFD2、KFD3、F、C、A**	**KFD1、KFD2、KFD3、D、G、F、E**

テストされ変更されるレコードには、次のものがあります。

相対レコード	KFD1	KFD2	KFD3
001	**01**	**01**	**<ブランク>**
002	**01**	**01**	**<ブランク>**
003	**01**	**01**	**<ブランク>**
004	**01**	**01**	**<ブランク>**

最初のレコードは、 0101<ブランク> の最初のキーを介して読み取られ、 0101C に変更されます。レコードは次のようになります。

相対レコード	KFD1	KFD2	KFD3
001	**01**	**01**	**C**
002	**01**	**01**	**<ブランク>**
003	**01**	**01**	**<ブランク>**
004	**01**	**01**	**<ブランク>**

アプリケーションが次のキーを取得するとき、0101<ブランク> の上にある次の高位のキーは 0101C です。これは変更されたばかりのレコードです。しかし、この時点でアプリケーションは KFD3 フィールドを C からブランクへ変更します。

ユーザーは暗黙のアクセス・パス共用を理解していないので、アプリケーションはすべてのレコードを 2 度アクセスして変更します。その結果、アプリケーションの実行に時間がかかり、レコードが変更されなかったように見えます。

結合論理ファイルのセットアップ

この節では、以下の事柄について説明します。

- 61 ページの『2 つの物理ファイルを結合する場合の基本概念 (例 1)』
- 69 ページの『結合論理ファイルのセットアップ』
- 70 ページの『複数のフィールドを使用したファイル結合 (例 2)』
- 71 ページの『2 次ファイルの重複レコードの読み取り (例 3)』
- 73 ページの『属性が異なる結合フィールドの使用 (例 4)』
- 74 ページの『レコード様式に現れないフィールドの記述 (例 5)』
- 76 ページの『結合論理ファイルのキー・フィールドの指定 (例 6)』
- 76 ページの『結合論理ファイルの選択/除外ステートメントの指定』

- 77 ページの『3 つ以上の物理ファイルの結合 (例 7)』
- 79 ページの『物理ファイル自身の結合 (例 8)』
- 80 ページの『2 次ファイルに欠落しているレコードへの省略時データの使用 (例 9)』
- 81 ページの『複合結合論理ファイル (例 10)』
- 83 ページの『結合論理ファイルに関する考慮事項』

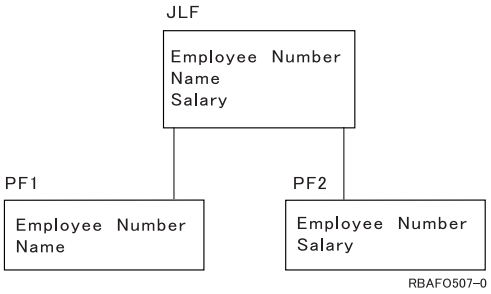
一般に、この節の例では、ファイルの図解、ファイルの DDS、およびサンプル・データが含まれています。例 1 では、ケースをいくつか設定しながら、ファイルを異なった状況 (物理ファイルのデータが異なる) でどのように結合するかを示します。

例では、便宜上および簡単に理解できるように、結合論理ファイルには JLF のラベルを付け、物理ファイルには PF1、PF2、PF3 などのラベルを付けています。

結合について、詳しくは、 SQL プログラミングに関するトピックの複数の表からデータを結合するを参照してください。

2 つの物理ファイルを結合する場合の基本概念 (例 1): 結合論理ファイルは、2 つ以上の物理ファイルからのフィールドを結合した (1 つのレコード様式で) 論理ファイルです。 レコード様式では、すべてのフィールドが物理ファイルに存在する必要はありません。

次の例では、2 つの物理ファイルを結合する結合論理ファイルを図示します。この例は、例 1 で述べる 5 つのケースで使用されます。



この例では、結合論理ファイル (JLF) に *Employee Number* (従業員番号)、 *Name* (名前)、および *Salary* (給与) フィールドがあります。物理ファイル 1 (PF1) には *Employee Number* と *Name* があり、物理ファイル 2 (PF2) には *Employee Number* と *Salary* があります。 2 つの物理ファイル (PF1 および PF2) の間で *Employee Number* は共通していますが、 *Name* は PF1 のみ、 *Salary* は PF2 のみに存在します。

結合論理ファイルを使用すると、アプリケーション・プログラムは (結合論理ファイルのレコード様式に対して) 1 回の読み取り操作を行い、両方の物理ファイルから必要なデータをすべて入手します。結合仕様がないと、論理ファイルは 2 つのレコード様式 (1 つは PF1 に基づくもので、もう 1 つは PF2 に基づくもの) を含むことになり、アプリケーション・プログラムは 2 つの物理ファイルから必要なデータをすべて入手するために 2 回の読み取り操作を行うことが必要になります。したがって、結合することにより、データベース設計がより柔軟なものとなります。

ただし、結合論理ファイルに対しては以下のような制限があります。

- 結合論理ファイルを通して物理ファイルを変更することはできません。更新、削除、あるいは書き込み (追加) 処理を行うには、別の複数様式論理ファイルを作成し、それを用いて物理ファイルを変更しなければなりません。変更処理を行うために、直接物理ファイルを使用することもできます。
- DFU を使用して結合論理ファイルを表示することはできません。
- 結合論理ファイルで指定できるレコード様式は 1 つだけに限られています。

- 結合論理ファイルのレコード様式は共用できません。
- 結合論理ファイルは他のファイルのレコード様式を共用できません。
- キー・フィールドは結合レコード様式で定義したフィールドでなければならず、JFILE キーワードに最初に指定したファイル (基本ファイル) からのフィールドでなければなりません。
- 選択/除外フィールドは結合レコード様式で定義したフィールドでなければなりませんが、任意の物理ファイルからも持ってくることができます。
- 結合論理ファイルではコミットメント制御を使用することはできません。

以下に、例 1 の DDS を示します。

```
JLF
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A          R JOINREC                   JFILE(PF1 PF2)
     A          J                           JOIN(PF1 PF2)
     A                                      JFLD(NBR NBR)
     A            NBR                       JREF(PF1)
     A            NAME
     A            SALARY
     A          K NBR
     A

PF1
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A          R REC1
     A            NBR           10
     A            NAME          20
     A          K NBR
     A

PF2
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A          R REC2
     A            NBR           10
     A            SALARY         7  2
     A          K NBR
     A
```

例 1 の結合論理ファイルの DDS について説明します (特定のキーワードの詳細については、DDS 解説書を参照してください)。

レコード・レベルの仕様では、結合論理ファイルで使用するレコード様式名を識別します。

R レコード様式であることを指示します。結合論理ファイルで指定できるのは 1 つのレコード様式に限られています。

JFILE 単一および複数様式論理ファイルで使用される PFILE キーワードを置換します。最低 2 つの物理ファイルを指定しなければなりません。JFILE キーワードに最初に指定するファイルが**基本ファイル**です。 JFILE キーワードで指定するもう一方のファイルが **2 次ファイル**です。

結合仕様では、1 対の物理ファイルを結合する方法を記述します。対のうち 2 番目のファイルが常に 2 次ファイルであり、各 2 次ファイルに対して 1 つの結合仕様がなければなりません。

J 結合仕様の開始を指示します。結合論理ファイル内では、最低 1 つの結合仕様を指定しなければなりません。結合仕様は、19 列から 28 列に指定した最初のフィールド名、または 17 列に指定した次の J で終わります。

JOIN 結合仕様でどの 2 つのファイルを結合するかを指示します。結合論理ファイルで 2 つの物理ファ

イルだけを結合する場合には、 JOIN キーワードは任意指定です。このキーワードの使用法の例については、この節の後の 77 ページの『3 つ以上の物理ファイルの結合 (例 7)』を参照してください。

JFLD JOIN で指定した物理ファイルからのレコードを結合するための**結合フィールド**を指示します。各結合仕様において、JFLD を最低 1 回は指定しなければなりません。結合フィールドは、物理ファイルに共通するフィールドです。 最初の結合フィールドは、JOIN キーワードに最初に指定するファイルのフィールドであり、2 番目の結合フィールドは、JOIN キーワードに 2 番目に指定するファイルのフィールドです。

文字タイプのフィールドを除いて、結合フィールドは同じ属性 (データ・タイプ、長さ、および小数点以下の桁数) でなければなりません。 フィールドが文字タイプならば、同じ長さである必要はありません。属性の異なる物理ファイルのフィールドを結合するときは、結合論理ファイルで使用するために再定義することができます。記述と例については、 73 ページの『属性が異なる結合フィールドの使用 (例 4)』を参照してください。

フィールド・レベル仕様では、結合論理ファイルに含まれるフィールドを指示します。

フィールド名 アプリケーション・プログラムでどのフィールドを使用するか (この例の場合は、*Nbr*、*Name*、および *Salary*) を指定します。最低 1 つのフィールド名が必要です。論理ファイルが使用する物理ファイルのフィールド名であれば、どの名前も指定することができます。RENAME、CONCAT、または SST といったキーワードを、単一および複数様式論理ファイルで使用するのと同様に、使用することもできます。

JREF レコード様式 (結合仕様レベルの後に指定し、キー・フィールド・レベルがある場合にはそれより先行) では、フィールドがどの物理ファイルからのものかを、フィールド名ではっきり識別しなければなりません。この例では、*Nbr* フィールドは PF1 と PF2 に存在します。したがって、どちらのファイルの *Nbr* フィールド記述を使用するかを指示するために、JREF キーワードが必要となります。

キー・フィールド・レベル仕様は任意指定であり、結合論理ファイルのキー・フィールド名が入ります。

K キー・フィールド仕様であることを指示します。K は 17 列に示されます。キー・フィールド仕様は任意指定です。

キー・フィールド名
キー・フィールド名 (この例では、*Nbr* が唯一のキー・フィールド) は任意指定であり、結合論理ファイルを (キー順の) 索引ファイルにします。キー・フィールドがないと、結合論理ファイルは到着順ファイルとなります。結合論理ファイルでは、キー・フィールドは基本ファイルのフィールドでなければなりません。また、キー・フィールド名は論理ファイルのレコード様式の 19 列から 28 列に指定しなければなりません。

選択/除外フィールド・レベル仕様は任意指定であり、結合論理ファイルの選択/除外フィールド名が入ります。

S または O 選択または除外の仕様であることを指示します。S または O は 17 列に示されます。選択/除外の仕様は任意指定です。

選択/除外フィールド名
選択/除外値を満たすレコードだけが、論理ファイルを使用するプログラムに返されます。選択/除外フィールドは、論理ファイルのレコード様式の 19 列から 28 列に指定しなければなりません。

以下のトピックには、物理ファイルの結合に関する特定のケースが説明されています。

- 『結合論理ファイルの読み取り』
- 65 ページの『基本ファイルと 2 次ファイルのレコードの一致 (ケース 1)』
- 65 ページの『2 次ファイルに欠落するレコード; JDFTVAL キーワードを指定していない場合 (ケース 2A)』
- 67 ページの『基本ファイルの 1 レコードに対して 2 次ファイルに複数の一致レコードがある場合 (ケース 3)』
- 67 ページの『2 次ファイルの余分なレコード (ケース 4)』
- 68 ページの『ランダム・アクセス (ケース 5)』

結合論理ファイルの読み取り: 以下に、 61 ページの『2 つの物理ファイルを結合する場合の基本概念 (例 1)』 に示す結合論理ファイルが、レコードをアプリケーション・プログラムに渡す方法をケースごとに説明します。

PF1 ファイルは JFILE キーワードで最初に指定されており、したがって、基本ファイルです。アプリケーション・プログラムがレコードを要求すると、システムは以下のことを行います。

1. 基本ファイルの最初の結合フィールドの値を使用します (PF1 の *Nbr* フィールド)。
2. 2 次ファイルの中から、結合フィールドと一致する最初のレコードを見つけ出します (PF2 の *Nbr* フィールドは PF1 の *Nbr* フィールドと一致しています)。
3. 各一致ごとに、物理ファイルからのフィールドを 1 つのレコードに結合し、このレコードをプログラムに渡します。物理ファイルにあるレコード数によって、以下の条件のうち 1 つが発生します。
   a. 基本ファイルのすべてのレコードに対して、一致レコードが 1 つだけ 2 次ファイルに見つかった場合。結果の結合論理ファイルには、基本ファイルの各レコードに対して 1 つのレコードが入ります。 65 ページの『基本ファイルと 2 次ファイルのレコードの一致 (ケース 1)』を参照してください。
   b. 基本ファイルのレコードで、2 次ファイルに一致レコードが見つからないものがある場合。

JDFTVAL キーワードを指定すると、以下のようになります。

- 2 次ファイルに一致レコードのある基本ファイルのレコードについては、システムは 2 次ファイルまたは複数の 2 次ファイルに結合します。その結果、基本ファイルの各レコードに対して、1 つ以上のレコードが存在することになります。
- 2 次ファイルに一致レコードのない基本ファイルのレコードについては、システムは 2 次ファイルに省略時値フィールドを追加して、結合処理を続行します。物理ファイルで DFT キーワードを使用すると、どの省略時が使用されるかを定義することができます。 65 ページの『2 次ファイルに欠落するレコード; JDFTVAL キーワードを指定していない場合 (ケース 2A)』 と 66 ページの『2 次ファイルに欠落するレコード; JDFTVAL キーワードを指定している場合 (ケース 2B)』を参照してください。

  注: DFT キーワードが 2 次ファイルで指定されている場合は、 DFT キーワードに指定された値が結合で使用されます。その結果、各基本レコードに対して最低 1 つの結合レコードが存在することになります。
- レコードは 2 次ファイルに存在しているものの、一致する値が基本ファイルにない場合は、レコードはプログラムに返されません。基本ファイルと 2 次ファイルの順番が逆転する 2 番目の結合論理ファイルを使用して、基本ファイルに一致レコードが存在しない 2 次ファイルのレコードがあるかを判別することができます。

JDFTVAL キーワードを指定しないと、以下のようになります。

- 2 次ファイルに一致レコードが存在する場合、システムは 2 次ファイルまたは複数の 2 次ファイルに結合します。その結果、基本ファイルの各レコードに対して、1 つ以上のレコードが存在することになります。
- 2 次ファイルに一致レコードが存在しない場合は、システムはレコードを返しません。

注: JDFTVAL を指定していない場合、システムは基本ファイルのレコードに対して、すべての 2 次ファイルで一致が見つかったときにだけ、レコードを返します。

以下の例では、ケース 1 からケース 4 までは順次読み取り処理を説明し、ケース 5 はキーによる読み取りを説明します。

基本ファイルと 2 次ファイルのレコードの一致 (ケース 1): ここで、結合論理ファイルを 61 ページの『2 つの物理ファイルを結合する場合の基本概念 (例 1)』 のように指定し、 PF1 と PF2 の両方に以下の 4 つのレコードが含まれているとします。

物理ファイル 1 (PF1)

235	Anne
440	Doug
500	Mark
729	Sue

物理ファイル 2 (PF2)

235	1700.00
440	950.50
500	2100.00
729	1400.90

プログラムは読み取り処理を 4 回行い、以下のレコードを得ます。

結合論理ファイル (JLF)

235	Anne	1700.00
440	Doug	950.50
500	Mark	2100.00
729	Sue	1400.90

2 次ファイルに欠落するレコード; JDFTVAL キーワードを指定していない場合 (ケース 2A): ここで、結合論理ファイルを 61 ページの『2 つの物理ファイルを結合する場合の基本概念 (例 1)』 のように指定し、 PF1 には 4 つのレコードが、PF2 には 3 つのレコードがあるとします。

物理ファイル 1 (PF1)

235	Anne
440	Doug
500	Mark
729	Sue

物理ファイル 2 (PF2)

235	1700.00
440	950.50
729	1400.90

PF2 では、番号 500 のレコードがありません。

プログラムは結合論理ファイルを読み取り、以下のレコードを取得します。

結合論理ファイル (JLF)

235	Anne	1700.00
440	Doug	950.50
729	Sue	1400.90

JDFTVAL キーワードを指定せず、かつ 2 次ファイル中の結合フィールドに一致レコードがない場合は、レコードは結合論理ファイルの中に含まれません。

2 次ファイルに欠落するレコード; JDFTVAL キーワードを指定している場合 (ケース 2B): ここで、結合論理ファイルを 61 ページの『2 つの物理ファイルを結合する場合の基本概念 (例 1)』 のように指定しますが、以下の DDS で示すように、JDFTVAL キーワードを指定しています。

```
JLF
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A                                     JDFTVAL
     A          R JOINREC                  JFILE(PF1 PF2)
     A          J                          JOIN(PF1 PF2)
     A                                     JFLD(NBR NBR)
     A            NBR                      JREF(PF1)
     A            NAME
     A            SALARY
     A          K NBR
     A
```

プログラムは結合論理ファイルを読み取り、以下のレコードを取得します。

結合論理ファイル (JLF)

235	Anne	1700.00
440	Doug	950.50
500	Mark	0000.00
729	Sue	1400.90

JDFTVAL を指定していると、システムは番号 500 のレコードを、たとえそのレコードが PF2 で欠落していたとしても返します。レコードがないために、結合レコードでフィールド値の一部が欠落することがあります。この場合、*Salary* フィールドが欠落します。JDFTVAL を指定していると、欠落した文字フィールドには通常、ブランクが使用され、欠落した数字フィールドにはゼロが使用されます。したがって、この場

合、結合レコードの中で欠落しているレコードに対する値は 0 です。ただし、物理ファイルでフィールドについて DFT キーワードが指定されている場合は、 DFT キーワードで指定された省略時値が使用されます。

基本ファイルの 1 レコードに対して 2 次ファイルに複数の一致レコードがある場合 (ケース 3): ここで、結合論理ファイルを 61 ページの『2 つの物理ファイルを結合する場合の基本概念 (例 1)』 のように指定し、 PF1 には 4 つのレコードが、PF2 には 5 つのレコードがあるとします。

物理ファイル 1 (PF1)

235	Anne
440	Doug
500	Mark
729	Sue

物理ファイル 2 (PF2)

235	1700.00
235	1500.00
440	950.50
500	2100.00
729	1400.90

PF2 では、235 のレコードが重複しています。

プログラムは 5 つのレコードを得ます。

結合論理ファイル (JLF)

235	Anne	1700.00
235	Anne	1500.00
440	Doug	950.50
500	Mark	0000.00
729	Sue	1400.90

結合レコードでは、235 のレコードが重複しています。 JDUPSEQ キーワードを使用しない場合、重複したレコードから取得されるレコードの順序は予測不能になります。詳細については、 71 ページの『2 次ファイルの重複レコードの読み取り (例 3)』を参照してください。

2 次ファイルの余分なレコード (ケース 4): ここで、結合論理ファイルを 61 ページの『2 つの物理ファイルを結合する場合の基本概念 (例 1)』 のように指定し、 PF1 には 4 つのレコードが、PF2 には 5 つのレコードがあるとします。

301 のレコードは PF2 にのみ存在します。

プログラムは結合論理ファイルを読み取り、 4 つのレコードだけを取得します。 301 のレコードは含まれません。

結合論理ファイル (JLF)

235	Anne	1700.00
440	Doug	950.50
500	Mark	2100.00
729	Sue	1400.90

JDFTVAL キーワードを指定していても同じ結果になります。結合レコードを取得するには、必ず基本ファイルにレコードが含まれる必要があるためです。

ランダム・アクセス (ケース 5): ここで、結合論理ファイルを 61 ページの『2 つの物理ファイルを結合する場合の基本概念 (例 1)』 のように指定しているとします。結合論理ファイルにはキー・フィールドを定義していることに注意してください。このケースは、結合論理ファイルを使用したランダム・アクセス読み取り処理に対して、どのレコードが返されるかを示しています。

PF1 と PF2 には以下のレコードがあるとします。

物理ファイル 1 (PF1)

235	Anne
440	Doug
500	Mark
729	Sue
997	Tim

物理ファイル 2 (PF2)

235	1700.00
440	950.50
729	1400.90
984	878.25
997	331.00
997	555.00

PF2 では、レコード 500 のレコードがありません。レコード 984 は PF2 にのみ存在し、 997 のレコードが重複しています。

プログラムは以下のレコードを得ます。

論理ファイルの *Nbr* フィールドに対してプログラムから 235 の値が与えられると、システムは以下のレコードを提供します。

235	Anne	1700.00

論理ファイルの *Nbr* フィールドに対してプログラムから 500 の値が与えられ、さらに JDFTVAL キーワードを指定していると、システムは以下のレコードを提供します。

500	Mark	0000.00

注: 結合論理ファイルに JDFTVAL キーワードを指定していないと、 2 次ファイルに一致レコードがないので、500 の値に対するレコードは見つかりません。

論理ファイルの *Nbr* フィールドに対してプログラムから 984 の値が与えられると、レコード 984 は基本ファイルにないので、システムは何のレコードも提供せず、**レコードなし**の例外が発生します。

論理ファイルの *Nbr* フィールドに対してプログラムから 997 の値が与えられると、システムは以下のレコードのうち 1 つを返します。

997	Tim	331.00

または

997	Tim	555.00

プログラムにどちらのレコードが返されるかは予測できません。どのレコードを返すかを指定するためには、結合論理ファイルに JDUPSEQ キーワードを指定します。 71 ページの『2 次ファイルの重複レコードの読み取り (例 3)』を参照してください。

注:

1. ランダム・アクセスでは、アプリケーション・プログラマーは、PF2 に重複レコードが存在する可能性があることを知っていなければなりません。また、プログラムが重複キーのあるレコードに対して複数回の読み取り処理を確実に行うようにしなければなりません。プログラムで順次アクセスが使用されている場合には、 2 回目の読み取り処理で 2 つ目のレコードが入手されます。
2. JDUPSEQ キーワードを指定すると、システムは結合論理ファイルに対して別個のアクセス・パスを作成することができます (なぜなら、共用することができる既存のアクセス・パスを見つけ出すことはほとんど不可能に近いからです)。JDUPSEQ キーワードを省略すると、システムは他のファイルのアクセス・パスを共用することができます。(この場合、システムは PF2 のアクセス・パスを共用することになります。)

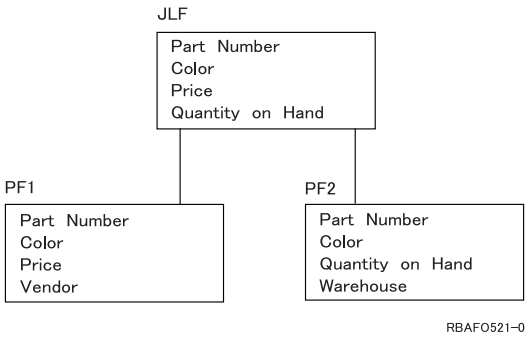
結合論理ファイルのセットアップ: 結合論理ファイルをセットアップするには、以下のことを行います。

1. 論理ファイルのレコード様式で必要とする、すべての物理ファイル・フィールドのフィールド名を検出します。(ファイル・フィールド記述の表示 [DSPFFD] コマンドを用いると、ファイルに含まれているフィールドを表示することができます。)
2. レコード様式にフィールドを記述します。縦方向リストにフィールド名を記入します。これが結合論理ファイルのレコード様式の開始を示します。

   注: フィールド名はどのような順番でも指定することができます。異なる物理ファイルに同じフィールド名がある場合、このようなフィールドには JREF キーワードを用いて、物理ファイル名を指定します。RENAME キーワードを用いるとフィールド名を変更することができます。さらに CONCAT キーワードを用いると同じ物理ファイルのフィールドを連結することもできます。既存の文字、16 進数、またはゾーン 10 進数のそれぞれのフィールドのサブセットは、SST キーワードを用いて定義することができます。文字フィールドまたはゾーン 10 進数フィールドのサブストリングは文字フィールドです。16 進数フィールドのサブストリングもやはり 16 進数です。データ・タイプ、長さ、または小数点以下の桁数を変更して、フィールドを再定義することができます。

3. JFILE キーワードのパラメーター値として物理ファイル名を指定します。最初に指定したファイル名が基本ファイルです。それ以外はすべて 2 次ファイルです。最高のパフォーマンスを得るために、基本ファイルの後には最もレコードの少ない 2 次ファイルをまず指定します。
4. 各 2 次ファイルに対して、結合仕様をコーディングします。各結合仕様において、どの組のファイルを結合するか (JOIN キーワードを使用しますが、2 次ファイルが 1 つだけなら任意指定です)、また、どのフィールドを使用して結合するか (JFLD キーワードを使用します。各結合仕様に最低 1 つは必要です) を指示します。
5. 任意選択として、以下のものを指定します。
   a. JDFTVAL キーワード。2 次ファイルに一致レコードが存在しない場合であっても、基本ファイルの各レコードに対してレコードを返したいのであれば、このキーワードを指定します。
   b. JDUPSEQ キーワード。2 次ファイルに重複する値が存在する可能性のあるフィールドに指定します。JDUPSEQ には、重複を分類するフィールド (結合フィールドの 1 つ以外のフィールド)、および使用する順序を指定します。
   c. キー・フィールド。キー・フィールドは 2 次ファイルからコピーすることはできません。キー・フィールドを除外すると、基本ファイルに現れるとおりの到着順でレコードが返されます。
   d. 選択/除外フィールド。状況によっては、ファイル・レベルで動的選択 (DYNSLT) キーワードも指定しなければなりません。
   e. 非入出力フィールド。説明については、74 ページの『レコード様式に現れないフィールドの記述 (例 5)』を参照してください。

複数のフィールドを使用したファイル結合 (例 2): 2 つのファイルを結合するために、複数の結合フィールドを指定することができます。以下に示すのは論理ファイルと 2 つの物理ファイルのフィールドです。



結合論理ファイル (JLF) には、 *Part Number* (部品番号)、 *Color* (色)、 *Price* (価格)、および *Quantity on Hand* (在庫量) フィールドがあります。物理ファイル 1 (PF1) には *Part Number*、 *Color*、 *Price*、 *Vendor* (ベンダー) が含まれ、物理ファイル 2 (PF2) には *Part Number*、 *Color*、 *Quantity on Hand*、および *Warehouse* (倉庫) が含まれます。これらのファイルの DDS は、以下のとおりです。

```
JLF
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A          R JOINREC                   JFILE(PF1 PF2)
     A          J                           JOIN(PF1 PF2)
     A                                      JFLD(PTNBR PTNBR)
     A                                      JFLD(COLOR COLOR)
     A            PTNBR                     JREF(PF1)
     A            COLOR                     JREF(PF1)
     A            PRICE
     A            QUANTOH
     A

PF1
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A          R REC1
```

```
     A            PTNBR          4
     A            COLOR         20
     A            PRICE          7  2
     A            VENDOR        40
     A
```

PF2

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A          R REC2
     A            PTNBR          4
     A            COLOR         20
     A            QUANTOH        5  0
     A            WAREHSE       30
     A
```

物理ファイルには以下のレコードがあるとします。

物理ファイル 1 (PF1)

100	Black	22.50	ABC Corp.
100	White	20.00	Ajax Inc.
120	Yellow	3.75	ABC Corp.
187	Green	**110.95**	ABC Corp.
187	Red	**110.50**	ABC Corp.
190	Blue	40.00	Ajax Inc.

物理ファイル 2 (PF2)

100	Black	23	ABC Corp.
100	White	15	Ajax Inc.
120	Yellow	102	ABC Corp.
187	Green	**0**	ABC Corp.
187	Red	**2**	ABC Corp.
190	Blue	**2**	Ajax Inc.

ファイルを順次に処理すると、プログラムは以下のレコードを受け取ります。

結合論理ファイル (JLF)

100	Black	22.50	23
100	White	20.00	15
120	Yellow	3.75	102
187	Green	**110.95**	**0**
187	Red	**110.50**	**2**

部品番号 190 で色が青のレコードはプログラムでは使用できないことに注意してください。なぜなら、2 次ファイルの 2 つのフィールドで一致レコードが見つからないからです。JDFTVAL を指定していないので、レコードは返されません。

2 次ファイルの重複レコードの読み取り (例 3): 2 次ファイルを結合する場合に、2 次ファイルから複数のレコードが生じることがあります。 このようなことが発生する場合には、その 2 次ファイルの結合仕様

に JDUPSEQ キーワードを指定することによって、重複レコードの順番を 2 次ファイルで指定しているフィールドに基づくようシステムに指示することができます。

物理ファイルと結合論理ファイルの DDS を以下に示します。

```
JLF
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A          R JREC                      JFILE(PF1 PF2)
     A          J                           JOIN(PF1 PF2)
     A                                      JFLD(NAME1 NAME2)
     A                                      JDUPSEQ(TELEPHONE)
     A            NAME1
     A            ADDR
     A            TELEPHONE
     A

PF1
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A          R REC1
     A            NAME1         10
     A            ADDR          20
     A

PF2
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A          R REC2
     A            NAME2         10
     A            TELEPHONE      8
     A
```

物理ファイルには以下のレコードがあります。

物理ファイル 1 (PF1)

Anne	120 1st St.
Doug	40 Pillsbury
Mark	2 Lakeside Dr.

物理ファイル 2 (PF2)

Anne	555-1111
Anne	555-6666
Anne	555-2222
Doug	555-5555

結合論理ファイルは以下のレコードを返します。

結合論理ファイル (JLF)

Anne	120 1st St.	555-1111
Anne	120 1st St.	555-2222
Anne	120 1st St.	555-6666
Doug	40 Pillsbury	555-5555

プログラムは Anne の使用可能なレコードをすべて読み取り、それから Doug、Mark へと続きます。Anne には 1 つの住所しかありませんが、電話番号は 3 つあります。そのため、Anne のレコードは 3 つ返されることになります。

Anne のレコードは、電話番号で昇順に分類されます。これは、キーワード・パラメーターとして *DESCEND を指定していない限り、 JDUPSEQ キーワードが昇順に分類するからです。以下に DDS で *DESCEND を使用した例を示します。

```
JLF
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A          R JREC                       JFILE(PF1 PF2)
     A          J                            JOIN(PF1 PF2)
     A                                       JFLD(NAME1 NAME2)
     A                                       JDUPSEQ(TELEPHONE *DESCEND)
     A            NAME1
     A            ADDR
     A            TELEPHONE
     A
```

JDUPSEQ で *DESCEND を指定すると、以下のようなレコードが返されます。

結合論理ファイル (JLF)

Anne	120 1st St.	555-6666
Anne	120 1st St.	555-2222
Anne	120 1st St.	555-1111
Doug	40 Pillsbury	555-5555

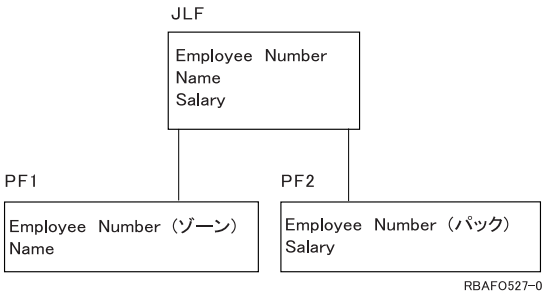
注: JDUPSEQ キーワードは、キーワードが指定されている結合仕様に対してのみ適用されます。複数の結合仕様のある結合論理ファイルにおける JDUPSEQ キーワードを示す例については、81 ページの『複合結合論理ファイル (例 10)』を参照してください。

属性が異なる結合フィールドの使用 (例 4): 結合フィールドとして使用する物理ファイルからのフィールドは、通常、同じ属性 (長さ、データ・タイプ、および小数点以下の桁数) を持っています。たとえば、71 ページの『2 次ファイルの重複レコードの読み取り (例 3)』 のように、 *Name1* フィールドは物理ファイル PF1 にある長さ 10 文字の文字フィールドであり、物理ファイル PF2 にある長さ 10 文字の文字フィールド *Name2* に結合することができます。*Name1* と *Name2* のフィールドは同じ特性を持っているため、結合フィールドとして簡単に使用することができます。

異なる長さの文字タイプ・フィールドも、フィールドの再定義をしなくても結合フィールドとして使用することができます。たとえば、PF1 の Name1 フィールドが 10 文字の長さで、 PF2 の Name2 フィールドが 15 文字の長さである場合に、どちらかのフィールドを再定義しなくても、これらのフィールドを結合フィールドとして使用することができます。

以下に示すのは、結合フィールドが同じ属性を持っていない例です。どちらの物理ファイルにも、従業員番号フィールドがあります。物理ファイル PF1 の *Nbr* フィールドと物理ファイル PF2 の *Nbr* フィールドは、どちらも 34 列で指定しているように長さが 3 です。しかし、PF1 ファイルではフィールドはゾーン 10 進数であり (35 列に S)、PF2 ファイルではフィールドはパック 10 進数です (35 列に P)。これらのフィールドを結合フィールドとして用いて 2 つのファイルを結合するためには、同じ属性を持つようにどちらか一方あるいは両方のフィールドを再定義しなければなりません。

以下に、論理ファイルと物理ファイルのフィールドを示します。



結合論理ファイル (JLF) には、 *Employee Number* (従業員番号)、 *Name* (名前)、 *Salary* (給与) フィールドがあります。物理ファイル 1 (PF1) には、 *Employee Number* (ゾーン) および *Name* が含まれます。物理ファイル 2 (PF2) には、 *Employee Number* (パック) および *Salary* が含まれます。これらのファイルの DDS は、以下のとおりです。

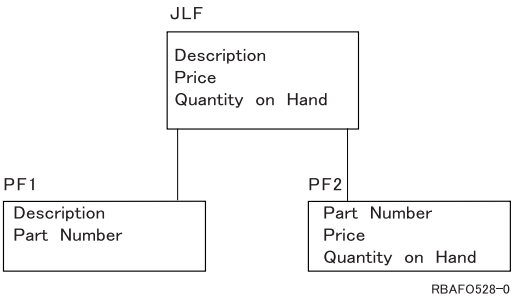
```
JLF
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A          R JOINREC                   JFILE(PF1 PF2)
     A          J                           JOIN(PF1 PF2)
     A                                      JFLD(NBR NBR)
     A            NBR             S         JREF(2)
     A            NAME
     A            SALARY
     A

PF1
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A          R REC1
     A            NBR            3S 0 <-Zoned
     A            NAME          20
     A          K NBR
     A

PF2
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A          R REC2
     A            NBR            3P 0 <-Packed
     A            SALARY         7  2
     A          K NBR
     A
```

注: この例では、論理ファイルの *Nbr* フィールドは PF2 からコピーしたものです。これは、JREF(2) が指定されているからです。物理ファイル名を指定する代わりに、JREF キーワードで相対ファイル番号を指定することができます。この例では、2 が PF2 を示しています。

PF1 ファイルと PF2 ファイルの *Nbr* フィールドは、結合フィールドとして使用されているため、同じ属性を持っていなければなりません。この例では同じ属性を持っていません。したがって、どちらか一方または両方を再定義して、同じ属性を持つようにしなければなりません。この例では、2 の従業員番号フィールドの属性の違いを解決するために、JLF の *Nbr* フィールド (PF2 ファイルからのもの) がゾーン 10 進数として再定義されています (JLF の 35 列に S)。

レコード様式に現れないフィールドの記述 (例 5): 非入出力フィールド (38 列で N と指定) は入力も出力も行わない結合論理ファイルで使用することができます。結合論理ファイルを使用するプログラムは非入出力フィールドを見ることも読み取ることもできません。非入出力フィールドはレコード様式には含まれません。非入出力フィールドはキー・フィールドにはできませんし、結合されたファイルの選択/除外ステートメントにも使用できません。非入出力フィールドは結合フィールド (JFLD キーワードを使用して結合仕様レベルで指定) に使用することができます。このフィールドは、結合が可能なレコード・レベルだけで再定義しますが、プログラムでは必要とされません。

次の例では、プログラムは在庫部品の説明、価格、および在庫量を読み取ります。部品番号自身は、部品のレコードをまとめるため以外は必要ではありません。しかし、部品番号は異なった属性となっているので、最低 1 つは再定義しなければなりません。



結合論理ファイル (JLF) には、 *Description* (説明)、 *Price* (価格)、および *Quantity on Hand* (在庫量) フィールドがあります。物理ファイル 1 (PF1) には *Description* および *Part Number* (部品番号) があり、物理ファイル 2 (PF2) には *Part number*、 *Price*、および *Quantity on Hand* があります。これらのファイルの DDS は、以下のとおりです。

JLF
```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A          R JOINREC                   JFILE(PF1 PF2)
     A          J                           JOIN(PF1 PF2)
     A                                      JFLD(PRTNBR PRTNBR)
     A            PRTNBR          S  N      JREF(1)
     A            DESC
     A            PRICE
     A            QUANT
     A          K DESC
     A
```

PF1
```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A          R REC1
     A            DESC          30
     A            PRTNBR         6P 0
     A
```

PF2
```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A          R REC2
     A            PRTNBR         6S 0
     A            PRICE          7  2
     A            QUANT          8  0
     A
```

PF1 では、*Prtnbr* フィールドはパック 10 進数フィールドですが、PF2 では、*Prtnbr* フィールドはゾーン 10 進数フィールドです。結合論理ファイルでは、これらを結合フィールドとして使用します。PF1 の *Prtnbr* フィールドは、フィールド・レベルで 35 列に S と指定することで、ゾーン 10 進数フィールドに再定義されます。JREF キーワードはフィールドがどの物理ファイルからコピーされたものかを識別します。しかし、フィールドはレコード様式に含まれていません。そのため、38 列に N を指定して非入出力フィールドとします。このファイルを使用するプログラムはそのフィールドを見ることはありません。

この例では、販売員が部品の説明を入力することができます。プログラムは、突き合わせあるいは近似突き合わせを行うために結合論理ファイルを読み取り、ユーザーが説明、価格、在庫量を調査できるように 1 つ以上の部品を表示することができます。このアプリケーション・プログラムでは、顧客注文や倉庫用の部品の発注には部品番号が必要ではないことを前提としています。

結合論理ファイルのキー・フィールドの指定 (例 6): 結合論理ファイルにキー・フィールドを指定するときは、以下の規則が適用されます。

- キー・フィールドは、1 次物理ファイルに存在しなければなりません。
- キー・フィールドの名前は、論理ファイルの結合レコード様式の 19 列から 28 列で指示しなければなりません。
- キー・フィールドは論理ファイルの非入出力フィールド (フィールドの 38 列に N と指定) として定義されたフィールドであってはなりません。

以下のものはキー・フィールドの規則を示しています。

JLF

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A          R JOINREC                   JFILE(PF1 PF2)
     A          J                           JOIN(PF1 PF2)
     A                                      JFLD(NBR NUMBER)
     A                                      JFLD(FLD3 FLD31)
     A            FLD1                      RENAME(F1)
     A            FLD2                      JREF(2)
     A            FLD3          35   N
     A            NAME
     A            TELEPHONE                 CONCAT(AREA LOCAL)
     A          K FLD1
     A          K NAME
     A
```

PF1

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A          R REC1
     A            NBR            4
     A            F1            20
     A            FLD2           7  2
     A            FLD3          40
     A            NAME          20
     A
```

PF2

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A          R REC2
     A            NUMBER         4
     A            FLD2           7  2
     A            FLD31         35
     A            AREA           3
     A            LOCAL          7
     A
```

以下のフィールドはキー・フィールドには**なりません**。

Nbr (19 列から 28 列に名前が指示されていません。)
Number (19 列から 28 列に名前が指示されていません。)
F1 (19 列から 28 列に名前が指示されていません。)
FLd31 (2 次ファイルからコピーされたものです。)
FLd2 (2 次ファイルからコピーされたものです。)
FLd3 (非入出力フィールドです。)
Area と *Local* (19 列から 28 列に名前が指示されていません。)
Telephone (2 次ファイルからのフィールドに基づいています。)

結合論理ファイルの選択/除外ステートメントの指定: 結合論理ファイルに選択/除外ステートメントを指定する場合は、以下の規則が適用されます。

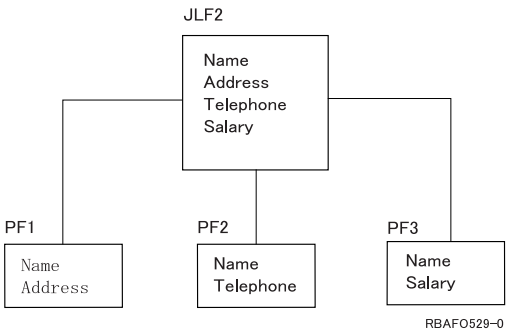
- フィールドは論理ファイルが使用するどの物理ファイルからコピーしたものであってもかまいません (JFILE キーワードで指定します)。
- 選択/除外ステートメントで指定するフィールドは、非入出力フィールド (フィールドの 38 列に N と指定) として定義されたフィールドであってはなりません。
- 状況によっては、結合論理ファイルで選択/除外ステートメントを指定する際に、DYNSLT キーワードを指定しなければなりません。詳細と例については、DDS 解説書 の DYNSLT キーワードを参照してください。

結合論理ファイルの選択/除外ステートメントを示す例については、 81 ページの『複合結合論理ファイル (例 10)』を参照してください。

3 つ以上の物理ファイルの結合 (例 7): 結合論理ファイルを使用して 32 個までの物理ファイルを結合することができます。これらのファイルは、JFILE キーワードで指定しなければなりません。JFILE キーワードで最初に指定したファイルが基本ファイルであり、それ以外のファイルはすべて 2 次ファイルです。

物理ファイルは対で結合しなければならず、おのおのの対は結合仕様で記述します。各結合仕様では 1 つ以上の結合フィールドを識別しなければなりません。

以下にファイルのフィールドと、論理ファイルのすべての物理ファイルに共通する 1 つのフィールドを示します。



結合論理ファイル (JLF2) には、 *Name*、 *Address* (住所)、 *Telephone* (電話番号)、および *Salary* (給与) が含まれています。物理ファイル 1 (PF1) には *Name* と *Address* が含まれ、物理ファイル 2 (PF2) には *Name* と *Telephone* が含まれます。物理ファイル 3 (PF3) には、 *Name* と *Salary* が含まれます。この例では、*Name* フィールドがすべての物理ファイル (PF1、PF2、および PF3) に共通しており、結合フィールドとなっています。

以下に、物理ファイルと論理ファイルの DDS を示します。

JLF

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A          R JOINREC                   JFILE(PF1 PF2 P3)
     A          J                           JOIN(PF1 PF2)
     A                                      JFLD(NAME NAME)
     A          J                           JOIN(PF2 PF3)
     A                                      JFLD(NAME NAME)
     A            NAME                      JREF(PF1)
     A            ADDR
     A            TELEPHONE
     A            SALARY
     A          K NAME
     A
```

PF1

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A          R REC1
```

```
     A            NAME          10
     A            ADDR          20
     A          K NAME
     A
```

PF2

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A          R REC2
     A            NAME          10
     A            TELEPHONE      7
     A          K NAME
     A
```

PF3

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A          R REC3
     A            NAME          10
     A            SALARY         9  2
     A          K NAME
     A
```

物理ファイルには、以下のレコードがあるとします。

物理ファイル 1 (PF1)

Anne	120 1st St.
Doug	40 Pillsbury
Mark	2 Lakeside Dr.
Tom	335 Elm St.

物理ファイル 2 (PF2)

Anne	555-1111
Doug	555-5555
Mark	555-0000
Sue	555-3210

物理ファイル 3 (PF3)

Anne	1700.00
Doug	950.00
Mark	2100.00

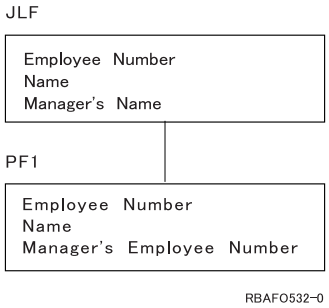
プログラムは以下の論理ファイル・レコードを読み取ります。

結合論理ファイル (JLF)

Anne	120 1st St.	555-1111	1700.00
Doug	40 Pillsbury	555-5555	950.00
Mark	2 Lakeside Dr..	555-0000	2100.00
Doug	40 Pillsbury	555-5555	

Tom のレコードは返されません。これは、Tom のレコードが PF2 と PF3 で見つからず、JDFTVAL キーワードの指定もないからです。Sue のレコードも返されません。これは、基本ファイルに Sue のレコードがないからです。

物理ファイル自身の結合 (例 8): 1 つのファイル内の複数のレコードを連結することによって形成されたレコードを読み取り、物理ファイルをそれ自身に結合することができます。以下にこれらの例を示します。



結合論理ファイル (JLF) には、 *Employee Number* (従業員番号)、 *Name* (名前)、および *Manager's Name* (管理者の名前) が含まれます。物理ファイル (PF1) には、 *Employee Number*、*Name*、および *Manager's Employee Number* (管理者の従業員番号) が含まれます。以下にこれらのファイルの DDS を示します。

JLF
```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A                                     JDFTVAL
     A          R JOINREC                  JFILE(PF1 PF1)
     A          J                          JOIN(1 2)
     A                                     JFLD(MGRNBR NBR)
     A            NBR                      JREF(1)
     A            NAME                     JREF(1)
     A            MGRNAME                  RENAME(NAME)
     A                                     JREF(2)
     A
```

PF1
```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A          R RCD1
     A            NBR             3
     A            NAME           10          DFT('none')
     A            MGRNBR          3
     A
```

注:

1. 同じファイル名を JFILE キーワードに 2 度指定しているので、 JOIN キーワードに相対ファイル番号を指定しなければなりません。相対ファイル番号 1 は JFILE キーワードに最初に指定した物理ファイルを指します。2 は 2 番目のものを指し、3 以降も同様となります。
2. JFILE キーワードに同じ物理ファイルを指定しているので、フィールド・レベルで指定した各フィールドに JREF キーワードが必要となります。

PF1 には以下のレコードがあるとします。

物理ファイル 1 (PF1)

235	Anne	440
440	Doug	729
500	Mark	440
729	Sue	888

プログラムは以下の論理ファイル・レコードを読み取ります。

結合論理ファイル (JLF)

235	Anne	Doug
440	Doug	Sue
500	Mark	Doug
729	Sue	none

JDFTVAL キーワードを指定しているので、Sue の管理者の名前のレコードが返されることに注意してください。さらに PF1 物理ファイルの *Name* フィールドに DFT キーワードを使用しているので、値 none が返されることに注意してください。

2 次ファイルに欠落しているレコードへの省略時データの使用 (例 9): 3 つ以上のファイルを結合する場合に JDFTVAL キーワードを指定すると、 2 次ファイルに欠落している結合フィールドにはシステム提供の省略時値が使用され、他の 2 次ファイルと結合されます。DFT キーワードを 2 次ファイルで指定すると、 DFT キーワードに指定した値が論理ファイルで使用されます。

ファイルの DDS は以下のとおりです。

```
JLF
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A                                     JDFTVAL
     A          R JRCD                     JFILE(PF1 PF2 PF3)
     A          J                          JOIN(PF1 PF2)
     A                                     JFLD(NAME NAME)
     A          J                          JOIN(PF2 PF3)
     A                                     JFLD(TELEPHONE TELEPHONE)
     A            NAME                     JREF(PF1)
     A            ADDR
     A            TELEPHONE                JREF(PF2)
     A            LOC
     A

PF1
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A          R RCD1
     A            NAME          20
     A            ADDR          40
     A            COUNTRY       40
     A

PF2
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A          R RCD2
     A            NAME          20
     A            TELEPHONE      8          DFT('999-9999')
     A

PF3
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A          R RCD3
     A            TELEPHONE      8
     A            LOC           30          DFT('No location assigned')
     A
```

PF1、PF2、および PF3 には以下のレコードがあるとします。

物理ファイル 1 (PF1)

Anne	120 1st St.	
Doug	40 Pillsbury	Canada
Mark	2 Lakeside Dr.	Canada
Sue	120 Broadway	

物理ファイル 2 (PF2)

Anne	555-1234
Doug	555-2222
Sue	555-1144

物理ファイル 3 (PF3)

555-1234	Room 312
555-2222	Main lobby
999-9999	No telephone number

結合論理ファイルに JDFTVAL を指定することによって、プログラムは以下の論理ファイル・レコードを読み取ります。

結合論理ファイル (JLF)

Anne	120 1st St.	555-1234	Room 312
Doug	40 Pillsbury	555-2222	Main lobby
Mark	2 Lakeside Dr.	999-9999	No telephone number
Sue	120 Broadway	555-1144	No location assigned

この例では、Anne と Doug には完全なデータがあります。しかし、Mark と Sue のデータは一部分が欠落しています。

- PF2 には Mark のレコードが欠落しています。これは、電話番号がないからです。PF2 の *Telephone* フィールドの省略時値は、 DFT キーワードを用いて 999-9999 と定義されています。したがって、この例では、電話番号がないときには 999-9999 の電話番号が返されます。結合論理ファイルに JDFTVAL キーワードを指定することによって、 PF2 の *Telephone* フィールドの省略時値 (999-9999) が PF3 のレコードとの突き合わせに使用されます。 (電話番号 999-9999 を説明するためのレコードが PF3 に含まれています。) JDFTVAL キーワードがないと、Mark のレコードは何も返されません。
- Sue の電話番号にはまだ場所が割り当てられていません。したがって、 PF3 には 555-1144 のレコードが欠落しています。JDFTVAL を指定していなかった場合、Sue についてのレコードが返されません。JDFTVAL キーワードを指定しているので、システムは、 PF3 の DFT キーワードに指定された省略時値 (`No location assigned`) を *Loc* フィールドに提供します。

複合結合論理ファイル (例 10): 次の例では、さらに複雑な結合論理ファイルを示します。データが以下の 3 つの物理ファイルにあるとします。

ベンダー・マスター・ファイル (PF1)

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A          R RCD1                    TEXT('VENDOR INFORMATION')
```

```
     A            VDRNBR         5          TEXT('VENDOR NUMBER')
     A            VDRNAM        25          TEXT('VENDOR NAME')
     A            STREET        15          TEXT('STREET ADDRESS')
     A            CITY          15          TEXT('CITY')
     A            STATE          2          TEXT('STATE')
     A            ZIPCODE        5          TEXT('ZIP CODE')
     A                                      DFT('00000')
     A            PAY            1          TEXT('PAY TERMS')
     A
```

注文ファイル (PF2)

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A          R RCD2                     TEXT('VENDORS ORDER')
     A            VDRNUM         5S 0       TEXT('VENDOR NUMBER')
     A            JOBNBR         6          TEXT('JOB NUMBER')
     A            PRTNBR         5S 0       TEXT('PART NUMBER')
     A                                      DFT(99999)
     A            QORDER         3S 0       TEXT('QUANTITY ORDERED')
     A            UNTPRC         6S 2       TEXT('PRICE')
     A
```

部品ファイル (PF3)

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A          R RCD3                     TEXT('DESCRIPTION OF PARTS')
     A            PRTNBR         5S 0       TEXT('PART NUMBER')
     A                                      DFT(99999)
     A            DESCR         25          TEXT('DESCRIPTION')
     A            UNITPRICE      6S 2       TEXT('UNIT PRICE')
     A            WHSNBR         3          TEXT('WAREHOUSE NUMBER')
     A            PRTLOC         4          TEXT('LOCATION OF PART')
     A            QOHAND         5          TEXT('QUANTITY ON HAND')
A
```

結合論理ファイルのレコード様式には、以下のフィールドがなければなりません。

Vdrnam (ベンダー名)
Street、*City*、*State*、*Zipcode* (ベンダーの住所)
Jobnbr (作業番号)
Prtnbr (部品番号)
Descr (部品の説明)
Qorder (発注量)
Untprc (単価)
Whsnbr (倉庫番号)
Prtloc (部品の場所)

この結合論理ファイルの DDS は以下のとおりです。

結合論理ファイル (JLF)

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
      A                                  1  DYNSLT
      A                                  2  JDFTVAL
      A          R RECORD1                  JFILE(PF1 PF2 PF3)
      A      3   J                          JOIN(1 2)
      A                                     JFLD(VDRNBR VDRNUM)
      A                                  4  JDUPSEQ(JOBNBR)
      A      5   J                          JOIN(2 3)
      A                                  6  JFLD(PRTNBR PRTNBR)
      A                                     JFLD(UNTPRC UNITPRICE)
      A        7   VDRNUM          5A  N      TEXT('CHANGED ZONED TO CHAR')
      A            VDRNAM
      A            ADDRESS                8  CONCAT(STREET CITY STATE +
      A                                        ZIPCODE)
      A            JOBNBR
      A            PRTNBR                 9  JREF(2)
```

```
A            DESCR
A            QORDER
A            UNTPRC
A            WHSNBR
A            PRTLOC
A    10    S VDRNAM                  COMP(EQ 'SEWING COMPANY')
A          S QORDER                  COMP(GT 5)
A
```

1 JDFTVAL キーワードと選択フィールドを指定しているので、DYNSLT キーワードが必要です。

2 物理ファイルにある省略時値を取り出すために JDFTVAL を指定しています。

3 最初の結合仕様です。

4 PF2 に重複するベンダー番号があるので、JDUPSEQ キーワードを指定しています。

5 2 番目の結合仕様です。

6 PF2 と PF3 ファイルからの正しいレコードを確実に結合するために、 2 つの JFLD キーワードを指定しています。

7 *Vdrnum* フィールドをゾーン 10 進数から文字に再定義しています (これは、結合フィールドとして使用するのに、PF1 と PF2 で同じ属性となっていないからです)。

8 CONCAT キーワードは同じ物理ファイルの 4 つのフィールドを 1 つのフィールドに連結します。

9 *PRTNBR* フィールドが 2 つの物理ファイルにあり、 PF2 にある方を使用するので、JREF キーワードを指定しなければなりません。

10 選択/除外フィールドは *Vdrnam* と *Qorder* です。(別個の 2 つの物理ファイルからコピーしたものであることに注意してください。)

結合論理ファイルに関する考慮事項: 結合論理ファイルに関する考慮事項については、以下のトピックを参照してください。

- 『パフォーマンスの考慮事項』
- 『データ保全性の考慮事項』
- 84 ページの『規則のまとめ』

パフォーマンスの考慮事項: 結合論理ファイルのパフォーマンスを向上させるために、以下の事柄を行うことができます。

- 結合しようとしている物理ファイルのレコード数がそれぞれ異なっている場合、最も少ないレコード数の物理ファイルを最初 (JOIN キーワードの後の最初のパラメーター) に指定します。
- DYNSLT キーワードを使用することを考慮します。詳細については、 57 ページの『動的選択/除外』を参照してください。
- 結合論理ファイルが自動的に既存のアクセス・パスを共用できるように記述することを考慮します。詳細については、 58 ページの『既存のアクセス・パスの使用』を参照してください。

  注: 結合論理ファイルは常に、JFLD キーワードに指定した対のフィールドの 2 番目のフィールドを使用するアクセス・パスを持っています。このフィールドは単純な論理ファイルのキー・フィールドのように働きます。アクセス・パスがまだ存在していない場合には、即時メインテナンスによってアクセス・パスが暗黙に作成されます。

データ保全性の考慮事項: 結合論理ファイルが使用する物理ファイルをロックしない限り、以下のことが発生する可能性があります。

- 2 次ファイルに 2 つ以上のレコードが存在するようなレコードを、プログラムが読み取る。システムは 1 つのレコードをプログラムに提供します。
- プログラムが読み取ったばかりの基本ファイルのレコードを他のプログラムが更新して、結合フィールドを変更する。
- プログラムが別の読み取り要求を出す。システムは基本ファイルの結合フィールドの現在の (新しい) 値に基づいて次のレコードを渡します。

このような考慮事項は、2 次ファイルにも同様に当てはまります。

規則のまとめ: データベース・ファイルを結合する場合に必要な規則を以下に要約します。

- 『要件』
- 『結合フィールド』
- 85 ページの『結合論理ファイルのフィールド』
- 85 ページの『その他の規則』

要件: 結合論理ファイルの主な要件は以下のとおりです。

- おのおのの結合論理ファイルには以下のものがなければなりません。
  - JFILE キーワードを指定した、唯一のレコード様式。
  - JFILE キーワードに指定した最低 2 つの物理ファイル。(JFILE キーワードの物理ファイル名は異なるファイルでなくてもかまいません。)
  - 最低 1 つの結合仕様 (JFLD キーワードを指定した上で 17 列に J と指定します。)
  - 最大 255 個の 2 次ファイル。
  - フィールド・レベルで N (非入出力フィールド) 以外に使用する最低 1 つのフィールド名。
- JFILE キーワードに 2 つの物理ファイルしか指定していない場合、 JOIN キーワードは必要ありません。この場合には 1 つの結合仕様だけを入れることができ、これによって 2 つの物理ファイルが結合されます。
- JFILE キーワードに 2 つ以上の物理ファイルを指定している場合、以下の規則が適用されます。
  - 基本ファイルは、最初の JOIN キーワードに指定する対のファイルの最初のファイルであること (基本ファイルは別の JOIN キーワードに指定する対のファイルの最初のものでもかまいません)。

    注: JFILE キーワードに同じファイル名を 2 度指定する場合には、JOIN キーワードと JREF キーワードには相対ファイル番号を指定しなければなりません。
  - すべての 2 次ファイルは、JOIN キーワードの対のファイルの 2 番目のファイルとして 1 度しか指定できません。つまり、JFILE キーワードのすべての 2 次ファイルには 1 つの結合仕様を指示しなければならないことを意味します (2 つの 2 次ファイルは 2 つの結合仕様を意味し、3 つの 2 次ファイルは 3 つの結合仕様を意味することになります)。
  - 2 次ファイルが結合仕様に現れる順序は、JFILE キーワードに指定する順序と一致していなければなりません。

結合フィールド: 結合フィールドについて覚えておくべき規則は、以下のとおりです。

- 結合する物理ファイルはすべて、最低 1 つの結合フィールドで別の物理ファイルと結合しなければなりません。結合フィールドは、結合仕様で JFLD キーワードのパラメーター値として指定するフィールドです。
- 複数の結合フィールド (JFLD キーワードに指定) は同じ属性 (長さ、データ・タイプ、小数点以下の桁数) となっているか、同じ属性を持つように結合論理ファイルのレコード様式で再定義しなければなりません。結合フィールドが文字タイプであれば、フィールドの長さは違っていてもかまいません。

- 結合フィールドは、結合論理ファイルのレコード様式で指定する必要はありません (ただし、属性が同じになるようにどちらか一方または両方を再定義しなければならない場合を除きます)。
- 結合フィールドを再定義する場合、38 列に N と指定して (非入出力フィールドにします)、結合論理ファイルを使用するプログラムが、再定義したフィールドを使用しないようにすることができます。
- 結合物理ファイルで使用されるフィールドの最大長は、物理および論理ファイルのキーの最大サイズと同じです (データベース・ファイルのサイズを参照してください)。

結合論理ファイルのフィールド: 結合論理ファイルのフィールドについて覚えておくべき規則は、以下のとおりです。

- 結合論理ファイルのレコード様式のフィールドは、論理ファイルが使用する物理ファイルの 1 つになければなりません。または、フィールドに CONCAT、RENAME、TRNTBL、または SST を指定している場合は、フィールドの結果を物理ファイルの 1 つに入れるようにします。
- CONCAT キーワードにパラメーター値として指定するフィールドは、同じ物理ファイルからコピーしたものでなければなりません。CONCAT キーワードに最初に指定するフィールド名が物理ファイルの中で固有のものでない場合には、使用するフィールド記述がどのファイルに入っているかを指示するために、フィールドに JREF キーワードを指定しなければなりません。
- 結合論理ファイルのレコード様式のフィールド名を複数の物理ファイルで指定している場合、フィールドがどのファイルからコピーしたものかを JREF キーワードで固有に指定しなければなりません。
- キー・フィールドを指定する場合、基本ファイルからコピーしたものでなければなりません。結合論理ファイルのキー・フィールドは、基本ファイルのキー・フィールドである必要はありません。
- 選択/除外フィールドは、結合論理ファイルが使用するどの物理ファイルからコピーしたものであってもかまいません。ただし状況によっては、 DYNSLT キーワードが必要となります。
- キー・フィールドと選択/除外フィールドを指定する場合、レコード様式で定義しなければなりません。
- 物理ファイル名を JFILE キーワードに複数回指定している場合、 JOIN と JREF キーワードには相対ファイル番号を使用しなければなりません。

その他の規則: 結合論理ファイルを使用する際に気を付けなければならないその他の規則として、以下のものがあります。

- 結合論理ファイルは読み取り専用ファイルです。
- 結合レコード様式を共用することはできず、さらに、他のレコード様式を共用することもできません。
- 結合論理ファイルでは、以下のものは許されません。
  - REFACCPTH キーワードと FORMAT キーワード
  - 共用フィールド (38 列に B と指定)

データベース・ファイルのアクセス・パスの記述

この章では、データベース・ファイルのアクセス・パスの記述方法について説明します。

アクセス・パスは、レコードが取り出される順序を表します。物理ファイルまたは論理ファイルのレコードは、到着順アクセス・パスまたはキー順アクセス・パスを用いて取り出すことができます。論理ファイルの場合、各レコードの 1 つまたは複数のフィールドの値に基づいて、レコードを選択したり、除外したりすることができます。キー・フィールドは、ファイル・メンバー内の特定タイプのレコードを配列するために使用されるフィールドです。

以下のような方法で、アクセス・パスを記述できます。

- 86 ページの『データベース・ファイルの到着順アクセス・パスの使用』
- 86 ページの『データベース・ファイルのキー順アクセス・パスの使用』

- 94 ページの『既存のアクセス・パス仕様の使用』
- 94 ページの『データベース・ファイルのアクセス・パスでの浮動小数点フィールドの使用』

データベース・ファイルの到着順アクセス・パスの使用

到着順アクセス・パスは、レコードがファイルに到着し、保管された順序に基づきます。読み取りまたは更新を行う場合、以下のようにレコードをアクセスすることができます。

- 順次アクセス。各レコードはファイル内の次の順次物理位置から読み取られます。
- 相対レコード番号による直接アクセス。レコードはファイルの先頭からの位置によって識別されます。

外部記述ファイルは、キー・フィールドがファイルに指定されていない場合に限って、到着順アクセス・パスとなります。

到着順アクセス・パスは、以下のものに対してのみ有効です。

- 物理ファイル
- 論理ファイルの各メンバーが 1 つの物理ファイル・メンバーを基としている場合、その論理ファイル
- 結合論理ファイル
- ビュー

到着順アクセス・パスは、以下のように使用することができます。

- 削除済みレコードが以前占有していた記憶域のスペースに別のレコードを配置することによって、プログラムがそのスペースを使用できるようにするのは、到着順の処理方法によってだけ可能です。この方法では、ユーザーが相対レコード番号を指定することによって、明示的にレコードを挿入する必要があります。レコードを削除することによって発生したスペースをシステムが管理する別の方法としては、削除済みレコードの再使用属性を物理ファイルに指定することがあります。削除済みレコードの再使用属性に関する詳細およびヒントについては、 104 ページの『削除済みレコードの再使用』を参照してください。 削除済みレコードの処理の詳細については、 188 ページの『データベース・レコードの削除』を参照してください。
- 高水準言語の物理ファイル・メンバーの表示 (DSPPFM) コマンドおよびファイルのコピー (CPYF) コマンドを使用すると、キー順ファイルを到着順で処理することができます。 この機能は、物理ファイル、1 つの物理ファイルを基とする単純な論理ファイル、または結合論理ファイルに対して使用することができます。
- 高水準言語においては、相対レコード番号で直接、キー順ファイルを処理することができます。この機能は、物理ファイル、1 つの物理ファイルを基とする単純な論理ファイル、または結合論理ファイルに対して使用することができます。
- 到着順アクセス・パスは追加の記憶域を必要とせず、常にファイルと一緒に保存または復元されます。(これは、到着順アクセス・パスが、データが保管された物理的な順序にほかならないからです。データを保存するということは、到着順アクセス・パスを保存するということです。)

データベース・ファイルのキー順アクセス・パスの使用

キー順アクセス・パスは、DDS で定義されたキー・フィールドの内容に基づきます。この種のアクセス・パスは、レコードの追加または削除が行われるたびに、あるいはレコードが更新されてキー・フィールドの内容が変わったときに更新されます。キー順アクセス・パスは、物理ファイルと論理ファイルの両方で有効です。ファイル内のレコード順序は、ファイルの作成時に DDS で定義され、システムにより自動的に保守されます。

文字フィールドとして定義したキー・フィールドは、EBCDIC 文字用に定義されている順序に基づいて配列されます。数字フィールドとして定義したキー・フィールドは、そのフィールドについて UNSIGNED

(符号なしの値) または ABSVAL (絶対値) DDS キーワードが指定されない限り、代数の値に基づいて配列されます。 キー・フィールドを 2 バイト文字セットとして定義することもできますが、ビット表現に基づく、1 バイト文字として配列されるにすぎません。

キー・フィールドの配列については、以下のトピックを参照してください。

- 『代替照合順序を使用したキー・フィールドの配列』
- 88 ページの『SRTSEQ パラメーターを使用したキー・フィールドの配列』
- 89 ページの『キー・フィールドの昇順または降順の配列』
- 90 ページの『複数のキー・フィールドの使用』
- 91 ページの『重複キー値の防止』
- 92 ページの『重複キーの配列』

代替照合順序を使用したキー・フィールドの配列: 文字フィールドとして定義されたキー・フィールドは、EBCDIC 文字の順序または代替照合順序のどちらかに基づいて配列することができます。 たとえば、以下のようなレコードがあるとします。

レコード	Empname	Deptnbr	Empnbr
1	Jones, Mary	45	23318
2	Smith, Ron	45	41321
3	JOHNSON, JOHN	53	41322
4	Smith, ROBERT	27	56218
5	JONES, MARTIN	53	62213

Empname がキー・フィールドであり、文字フィールドの場合に、EBCDIC 文字の順序を使用すると、レコードは以下のように配列されます。

レコード	Empname	Deptnbr	Empnbr
1	Jones, Mary	45	23318
3	JOHNSON, JOHN	53	41322
5	JONES, MARTIN	53	62213
2	Smith, Ron	45	41321
4	Smith, ROBERT	27	56218

ここで、EBCDIC 順序により、予期しなかった順番で分類がなされたことに注意してください。これは、小文字が大文字よりも前に分類されるからです。したがって、`Smith, Ron` は `Smith, ROBERT` よりも前に分類されることになります。代替照合順序を使用すると、大文字と小文字を使用して入力されたレコードを、以下の例のように分類することができます。

レコード	Empname	Deptnbr	Empnbr
3	JOHNSON, JOHN	53	41322
5	JONES, MARTIN	53	62213
1	Jones, Mary	45	23318
4	Smith, ROBERT	27	56218
2	Smith, Ron	45	41321

文字のキー・フィールドに対して代替照合順序を使用するには、 DDS の ALTSEQ キーワードを指定し、代替照合順序が入っている表の名前を指定します。表を作成する場合、表の各 2 バイトを 1 つの文字に対応させます。文字を分類する順番を変更するには、2 桁の値を、分類させるときに等価とする文字と同じ値

に変更します。 ALTSEQ キーワードの詳細については、 DDS 解説書 を参照してください。大文字と小文字を区別しないで分類する場合の説明については、ライブラリー QUSRSYS の QCASE256 表に用意されています。

SRTSEQ パラメーターを使用したキー・フィールドの配列: SRTSEQ パラメーターで使用可能ないくつかの分類順序に従って、文字データを含むキー・フィールドを配列することができます。たとえば、以下のようなレコードがあるとします。

レコード	Empname	Deptnbr	Empnbr
1	Jones, Marilyn	45	23318
2	Smith, Ron	45	41321
3	JOHNSON, JOHN	53	41322
4	Smith, ROBERT	27	56218
5	JONES, MARTIN	53	62213
6	Jones, Martin	08	29231

Empname フィールドがキー・フィールドで、かつ文字フィールドである場合は、*HEX 順序 (EBCDIC 順序) により次のようにレコードが配列されます。

レコード	Empname	Deptnbr	Empnbr
1	Jones, Marilyn	45	23318
6	Jones, Martin	08	29231
3	JOHNSON, JOHN	53	41322
5	JONES, MARTIN	53	62213
2	Smith, Ron	45	41321
4	Smith, ROBERT	27	56218

*HEX 順序では、すべての小文字が大文字より前に分類されることに注意してください。したがって、`Smith, Ron` は、`Smith, ROBERT` より前に置かれ、 `JOHNSON, JOHN` は、小文字と大文字の Jones の間に置かれます。レコードが大文字小文字の混合形式で入力されている場合は、レコードの分類のために *LANGIDSHR 分類順序を使用することができます。大文字と小文字に対して同じ照合の重みを使用する *LANGIDSHR 順序では、次のような配列になります。

レコード	Empname	Deptnbr	Empnbr
3	JOHNSON, JOHN	53	41322
1	Jones, Marilyn	45	23318
5	JONES, MARTIN	53	62213
6	Jones, Martin	08	29231
4	Smith, ROBERT	27	56218
2	Smith, Ron	45	41321

*LANGIDSHR 順序では、大文字と小文字が等価として扱われることに注意してください。したがって、`JONES, MARTIN` と `Jones, Martin` は等価であり、基礎となるファイルと同じ順序で分類されます。これは誤りではありませんが、報告書では、小文字の Jones をすべて大文字の JONES の前に置く方が、見た目に良いかもしれません。レコードが大文字や小文字をそろえずに使用して入力されている場合、レコードを分類するために、*LANGIDUNQ 分類順序を使用することができます。小文字と大文字に対して異なる (ただし、順次の) 照合の重みを使用する *LANGIDUNQ 順序では、次のような配列になります。

レコード	Empname	Deptnbr	Empnbr
3	JOHNSON, JOHN	53	41322
1	Jones, Marilyn	45	23318
6	Jones, Martin	08	29231
5	JONES, MARTIN	53	62213
4	Smith, ROBERT	27	56218
2	Smith, Ron	45	41321

*LANGIDSHR および *LANGIDUNQ 分類順序は、ご使用のシステムでサポートされるあらゆる言語に対して使用できます。 LANGID パラメーターは、*LANGIDSHR と *LANGIDUNQ のどちらの分類順序を使用するかを決定します。 SRTSEQ パラメーターは分類順序を指定し、また LANGID パラメーターは言語を指定するために使用します。

キー・フィールドの昇順または降順の配列: キー・フィールドは昇順または降順に配列することができます。たとえば、以下のようなレコードがあるとします。

レコード	Empnbr	Clsnbr	Clsnam	Cpdate
1	56218	412	Welding I	032188
2	41322	412	Welding I	011388
3	64002	412	Welding I	011388
4	23318	412	Welding I	032188
5	41321	412	Welding I	051888
6	62213	412	Welding I	032188

Empnbr フィールドがキー・フィールドの場合、このようなレコードの編成方法として以下の 2 つが考えられます。

- 昇順。アクセス・パスのレコードの順番は以下のようになります。

レコード	Empnbr	Clsnbr	Clsnam	Cpdate
4	23318	412	Welding I	032188
5	41321	412	Welding I	051888
2	41322	412	Welding I	011388
1	56218	412	Welding I	032188
6	62213	412	Welding I	032188
3	64002	412	Welding I	011388

- 降順。アクセス・パスのレコードの順番は以下のようになります。

レコード	Empnbr	Clsnbr	Clsnam	Cpdate
3	64002	412	Welding I	011388
6	62213	412	Welding I	032188
1	56218	412	Welding I	032188
2	41322	412	Welding I	011388
5	41321	412	Welding I	051888
4	23318	412	Welding I	032188

キー・フィールドを記述する際には、昇順が省略時です。ただし、DDS キーワード DESCEND を使用すれば、キー・フィールドを降順で配列することを指定できます。

複数のキー・フィールドの使用: ファイルのレコードを配列するのに、複数のキー・フィールドを使用することができます。キー・フィールドはみな同じ順序を使用する必要はありません。たとえば、2 つのキー・フィールドを使用する場合、1 つのフィールドが昇順を使用し、もう一方が降順を使用してもかまいません。たとえば、以下のようなレコードがあるとします。

レコード	Order	Ordate	Line	Item	Qtyord	Extens
1	52218	063088	01	88682	425	031875
2	41834	062888	03	42111	30	020550
3	41834	062888	02	61132	4	021700
4	52218	063088	02	40001	62	021700
5	41834	062888	01	00623	50	025000

アクセス・パスが、キー・フィールドとしてまず *Order* フィールド、次に *Line* フィールドを両方とも昇順で使用すると、アクセス・パスのレコードの順番は以下のようになります。

レコード	Order	Ordate	Line	Item	Qtyord	Extens
5	41834	062888	01	00623	50	025000
3	41834	062888	02	61132	4	021700
2	41834	062888	03	42111	30	020550
1	52218	063088	01	88682	425	031875
4	52218	063088	02	40001	62	021700

アクセス・パスが、キー・フィールドとしてまず *Order* を昇順で使用し、次に *Line* フィールドを降順で使用すると、アクセス・パスのレコードの順番は以下のようになります。

レコード	Order	Ordate	Line	Item	Qtyord	Extens
2	41834	062888	03	42111	30	020550
3	41834	062888	02	61132	4	021700
5	41834	062888	01	00623	50	025000
4	52218	063088	02	40001	62	021700
1	52218	063088	01	88682	425	031875

1 つのファイル内で、あるレコードのキー・フィールドの内容が、別のレコードのキー・フィールドの内容と同じであった場合には、重複キー値を持つレコードがあるファイルということになります。ただし、重複キー値と呼べるのは、1 つのレコードのすべての キー・フィールドで重複している場合だけです。たとえば、レコード様式に *Order* と *Ordate* の 2 つのキー・フィールドがある場合は、 *Order* と *Ordate* の両方のフィールドの内容が複数のレコードで同一のときに、重複キー値が生じます。以下のレコードには、重複キー値があります。

Order	Ordate	Line	Item	Qtyord	Extens
41834	062888	03	42111	30	020550
41834	062888	02	61132	04	021700
41834	062888	01	00623	50	025000

Line フィールドを 3 番目のキー・フィールドとして使用してファイルを定義すると、重複キーがなくなります。

(第 1 キー・フィールド) Order	(第 2 キー・フィールド) Ordate	(第 3 キー・フィールド) Line	Item	Qtyord	Extens
41834	062888	03	42111	30	020550
41834	062888	02	61132	04	021700
41834	062888	01	00623	50	025000

複数のレコード様式を持つ論理ファイルには、それらのレコード様式が異なる物理ファイルに基づくものであっても、重複キー値を持つレコードを含むことができます。つまり、キー値は、別々のレコード様式からのものであっても、重複キー値であるとみなされます。

重複キー値の防止: DB2 UDB for iSeries では、ファイルに重複キー値を持つレコードがあってもかまいません。ただし、ファイルによっては、重複キー値がないようにしたい場合もあります。たとえば、キー・フィールドを顧客番号フィールドとして定義するファイルを作成する場合、ファイルの各レコードが固有の顧客番号を持つことが望まれます。

ファイルに重複キー値がないようにするには、DDS で UNIQUE キーワードを指定します。 UNIQUE キーワードを指定すると、すでにファイルに存在するレコードのキー値と同じキー値のレコードを入力したり、ファイルにコピーしたりできなくなります。 また固有制約を用いても、強制的に固有キーの保全性を保つことができます。 サポートされている制約についての詳細は、237 ページの『制約によってユーザーのデータベースの保全性を制御する』を参照してください。

物理ファイルにすでに存在する複数のレコードが重複キー値となっている場合には、関連する論理ファイルでは UNIQUE キーワードが指定できません。関連する物理ファイルに重複キー値が入っているのに、UNIQUE キーワードを指定して論理ファイルを作成しようとしても、論理ファイルは作成されません。システムからは、その旨を示すメッセージが送られ、さらに、どのレコードに重複キー値が入っているかを示すメッセージ (最大 20 個) が送られます。

ファイルに UNIQUE キーワードを指定した場合、新しいレコードを追加するためにどのようなファイルを使用したとしても、ファイル内の既存のレコードと重複するキー値を持つレコードをファイルに追加することはできません。たとえば、LF1 と LF2 の 2 つの論理ファイルが物理ファイル PF1 を基礎としているとします。 LF1 には、UNIQUE キーワードが指定されています。 PF1 にレコードを追加するのに LF2 を使用した場合に、 LF1 に重複キー値が発生するようなときはレコードが追加できなくなります。

キー・フィールドのいずれかで NULL が許可されている場合、このようなフィールドに挿入された NULL は、ファイルの作成時にアクセス・パスをどのように定義したかによって、重複となる場合とならない場合があります。 UNIQUE キーワードの *INCNULL パラメーターは、固有のアクセス・パスに重複キーが存在するかどうかを判別するときに、NULL を含めることを示します。 *EXCNULL パラメーターは、重複キーが存在するかどうかを判別するときに、 NULL を含めないことを示します。詳細については、DDS 解説書 を参照してください。

以下の例は、固有のキー値を必要とする論理ファイルの DDS を示しています。

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A* ORDER TRANSACTION LOGICAL FILE (ORDFILL)
     A                                     UNIQUE
     A          R ORDHDR                   PFILE(ORDHDRP)
     A          K ORDER
     A
     A          R ORDDTL                   PFILE(ORDDTLP)
     A          K ORDER
     A          K LINE
     A
```

この例では、キー・フィールド (ORDHDR レコード様式の *Order* フィールド、および ORDDTL レコード様式の *Order* フィールドと *Line* フィールド) の内容は、レコードが ORDHDRP ファイル、ORDDTLP ファイル、あるいはここで定義されている論理ファイルのどれから追加されるかを問わず、固有でなければなりません。 *Line* フィールドを ORDDTL レコード様式の 2 番目のキー・フィールドとして定義しているので、両方の物理ファイルの *Order* キー・フィールドには同じ値が存在することができます。物理ファイル ORDDTLP には 2 つのキー・フィールドがあり、物理ファイル ORDHDRP には 1 つのキー・フィールドしかないので、この 2 つのファイルのキー値は対立しません。

重複キーの配列: DDS で UNIQUE キーワードを指定しない場合には、重複キー値を持つレコードがあったときに、これらをシステムがどのように保管するかを指定することができます。重複キー値を持つレコードは、以下のいずれかの方法でアクセス・パスに保管するよう指定できます。

- 後入れ先出し (LIFO)。 LIFO キーワードを指定すると (**1**)、重複キー値を持つレコードは、レコードの物理的な順序に従って後入れ先出し順で取り出されます。 以下の例は、LIFO キーワードを使用する DDS を示しています。

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
      A* ORDERP2
      A                                  1  LIFO
      A          R ORDER2
      A            .
      A            .
      A            .
      A          K ORDER
      A
```

- 先入れ先出し (FIFO)。 FIFO キーワードを指定すると、重複キー値を持つレコードは、レコードの物理的な順序に従って先入れ先出し順で取り出されます。
- 先変更先出し (FCFO)。 FCFO キーワードを指定すると、重複キー値を持つレコードは、キーの物理的な順序に従って先変更先出し順で取り出されます。
- 重複キー・フィールドの特定順序なし (省略時)。 FIFO、FCFO、または LIFO キーワードを指定しない場合には、重複キーを持つレコードを取り出すための保証された順序はありません。これによって、さらに多くのアクセス・パスが共用されることになり、パフォーマンスが向上する場合もあります。アクセス・パスの共用の詳細については、 58 ページの『既存のアクセス・パスの使用』を参照してください。

単一または複数様式論理ファイルが 1 つまたは複数の物理ファイル・メンバーを基礎としている場合、重複キー値を持つレコードは、論理ファイルの作成 (CRTLF) または論理ファイル・メンバーの追加 (ADDLFM) コマンドの DTAMBRS パラメーターでファイルおよびメンバーが指定された順序で読み取られます。 複数のレコード様式を持つ論理ファイルの例は、DDS 解説書 に記載されています。

重複キー値を持つレコードの LIFO または FIFO 順は、キー・フィールドの内容が更新された順序では決まらず、ファイル・メンバー内のレコードの物理的な順序だけで決まります。ここで、FIFO キーワードが指定された (重複キー値を持つレコードが先入れ先出し順になっている) 物理ファイルがあり、レコードが以下の順序でファイルに追加されたとします。

レコードがファイルに追加された順序	キー値
1	A
2	B
3	C
4	C
5	D

アクセス・パスの順序 (FIFO の昇順キー) は以下のようになります。

レコード番号	キー値
1	A
2	B
3	C
4	C
5	D

重複キー値を持つレコード 3 と 4 は、FIFO 順になっています。つまり、レコード 3 はレコード 4 より先にファイルに追加されたため、レコード 4 の前に読み取られます。レコードを降順で読み取ると、このことがはっきり分かります。このためには、この物理ファイルに基づいて、論理ファイルを作成しますが、このときに論理ファイルに DESCEND キーワードを指定します。

アクセス・パスの順序 (FIFO の降順キー) は以下のようになります。

レコード番号	キー値
5	D
3	C
4	C
2	B
1	A

キー値が C となるように物理レコード 1 を変更すると、物理ファイルのアクセス・パスの順序 (FIFO の昇順キー) は以下のようになります。

レコード番号	キー値
2	B
1	C
3	C
4	C
5	D

最後に、降順に変更すると、論理ファイルのアクセス・パスの順序 (FIFO の降順キー) は以下のようになります。

レコード番号	キー値
5	D
1	C
3	C
4	C
2	B

レコード 4 を追加した後にレコード 1 のキー・フィールドの内容を更新するという変更後であっても、レコード 1 はレコード 4 の後には現れません。

重複キー値を持つレコードの FCFO 順序は、キー・フィールドの内容を更新した順序で決まります。上の例では、キー値が C となるようにレコード 1 を変更した後のアクセス・パスの順序 (FCFO の昇順キー) は以下のようになります。

レコード番号	キー値
2	B
3	C
4	C
1	C
5	D

FCFO の場合、FCFO アクセス・パスが再作成されたとき、またはロールバック操作が行われたときに、重複キーの順序が変わることがあります。場合によっては、キー・フィールドが変わっても、物理的なキーが変わらないことがあります。このような場合には、キー・フィールドが変わっても、FCFO の順序付けは変わりません。たとえば、索引の順序付けがキーの絶対値に基づくように変わっても、 FCFO の順序付けは変わりません。キーが負数から正数に変更しても、キーの物理的な値は変わりません。物理的なキーが変わっていないので、FCFO の順序付けは変わらないことになります。

物理ファイルについて削除済みレコードの再使用属性が指定されている場合、重複キーの順序付けは、省略時が取られるようにするか、あるいは FCFO にしなければなりません。物理ファイルのキーの順序付けが FIFO または LIFO となっている場合、あるいは物理ファイルに基づいて定義された論理ファイルの重複キーの順序付けが FIFO または LIFO となっている場合には、物理ファイルについて削除済みレコードの再使用属性は使用できません。

既存のアクセス・パス仕様の使用

別のファイルのアクセス・パス仕様を使用するために、 DDS キーワードの REFACCPTH を使うことができます。ファイルの作成時に、システムはどのアクセス・パスを共用するかを判別します。 REFACCPTH キーワードを使用するファイルは、 REFACCPTH キーワードで指定されたファイルのアクセス・パスを必ずしも共用するとは限りません。 REFACCPTH キーワードは、単に、指定しなければならない DDS ステートメントの数を削減するために使用されます。 つまり、ファイルのキー・フィールド仕様をコーディングする代わりに、 REFACCPTH キーワードを指定できるということです。 ファイルの作成時に、システムは REFACCPTH キーワードに指定されたファイルから、作成されるファイルにキー・フィールドと選択/除外仕様をコピーします。

データベース・ファイルのアクセス・パスでの浮動小数点フィールドの使用

キー順序データベース・ファイルのレコード照合順序は、DDS の SIGNED、 UNSIGNED、および ABSVAL のキーワードの有無によって異なります。 浮動小数点フィールドの場合、符号は最も左端のビットであり、指数はその次のビットで、最後が有効桁です。 UNSIGNED を指定した場合の照合順序は、以下のとおりです。

- 正の実数－正の無限大
- 負の実数－負の無限大

DDS で SIGNED キーワードが指定されているかまたは省略時となっている浮動小数点キー・フィールドは、代数の数値順になります。照合順序は、負の無限大－実数－正の無限大です。

DDS で ABSVAL キーワードが指定されている浮動小数点のキー・フィールドは、絶対値数値順序を持っています。

以下の浮動小数点の照合順序に注意してください。

- ゼロ (正または負) は、他のすべての正/負の実数と同じように照合されます。
- 負のゼロは、SIGNED の順序においては正のゼロの前に照合されます。
- 正および負のゼロは、ABSVAL の順序においては同じものとして照合されます。

キー・フィールドでは数値以外 (*NAN) の値は使用できません。このような値を使用しようとし、ファイル作成時にキー・フィールドで *NAN 値が検出されると、ファイルは作成されません。

データベースの機密保護

以下のトピックは、データベースの機密保護を行うためのアクションを示しています。

- 『ファイル権限およびデータ権限の付与』
- 98 ページの『共通権限の指定』
- 100 ページの『入出力操作を制御するデータベース・ファイル処理可能性の使用』
- 100 ページの『データベース・ファイルの特定のフィールドへのアクセスの制限』
- 100 ページの『データを機密保護するための論理ファイルの使用』

iSeries システムでの機密保護の実施について詳しくは、「機密保護解説書  」を参照してください。

ファイル権限およびデータ権限の付与

ファイル権限およびデータ権限を付与するには、以下の方法があります。

- iSeries ナビゲーターを使用して、ユーザーまたはグループを許可することができます。『iSeries ナビゲーターを使用したユーザーまたはグループの許可』を参照してください。
- オブジェクト権限の付与 (GRTOBJAUT) コマンドを使用して、データベース・ファイルのデータへアクセスするユーザーの権限を指定することができます。
- SQL GRANT ステートメントを使用することができます。

付与できる権限のタイプについては、 『データベース・ファイルのオブジェクト権限のタイプ』および 97 ページの『データベース・ファイルのデータ権限のタイプ』を参照してください。

iSeries ナビゲーターを使用したユーザーまたはグループの許可: ユーザーによっては、オブジェクトに対して、共通権限によって許可されている許可とは異なる権限が必要な場合があります。オブジェクトに対して、ユーザーまたはグループを許可するには、以下のようにします。

1. iSeries ナビゲーター・ウィンドウで、使用したいシステムを拡張する。
2. 許可を編集する対象のオブジェクトが見えるまで、ナビゲートする。
3. 許可を追加する対象のオブジェクトを右クリックし、「**許可**」を選択する。
4. 「**許可**」ウィンドウで、「**追加**」をクリックする。
5. 「**追加**」ウィンドウで、1 つまたは複数のユーザーまたはグループを選択するか、ユーザー名フィールドまたはグループ名フィールドに、ユーザー名またはグループ名を入力する。
6. **OK** をクリックする。これで、ユーザーまたはグループが、リストの始めに追加されます。

注: ユーザーまたはグループには、オブジェクトに対するデフォルトの権限が付与されます。ユーザーの権限は、システムで定義されているタイプのどれか 1 つに変更できます。あるいは、権限をカスタマイズすることもできます。

iSeries ナビゲーターを使用して、権限を除去したり、カスタマイズすることもできます。

データベース・ファイルのオブジェクト権限のタイプ: 以下に、データベース・ファイルについて、ユーザーに付与できる権限のタイプについて説明します。

オブジェクト操作権限

以下のことを行う場合、ユーザーにはオブジェクト操作権限が必要になります。

- ファイルを処理のためにオープンする。(データ権限も最低 1 つは持っていなければなりません。)
- ファイル記述を使用するプログラムをコンパイルする。
- ファイルの活動メンバーに関する記述情報を表示する。
- Query 処理のためにファイルをオープンする。たとえば、Query ファイルのオープン (OPNQRYF) コマンドは、Query 処理のためにファイルをオープンします。

注: この他にも、オープン操作で指定するオプションによって必要とされる該当のデータ権限も持っていなければなりません。

オブジェクト存在権限

以下のことを行う場合、ユーザーにはオブジェクト存在権限が必要になります。

- ファイルを削除する。
- ファイルの記憶域を保管、復元、および解放する。オブジェクト存在権限がユーザーに明示的に付与されていない場合には、ユーザーは *SAVSYS 特殊ユーザー権限を使用して、ファイルの記憶域の保管、復元、および解放を行うことができます。*SAVSYS は、オブジェクト存在権限と同じものではありません。
- ファイルからメンバーを除去する。
- ファイルの所有権を移動する。

注: このような機能 (保管/復元を除く) を実行するには、ファイルに関するオブジェクト操作権限も必要です。

オブジェクト管理権限

以下のことを行う場合、ユーザーにはオブジェクト管理権限が必要になります。

- キー順アクセス・パスを持つ論理ファイルを作成する (論理ファイルが参照する物理ファイルに対するオブジェクト管理権限が必要です)。
- 権限を付与/取り消しする。付与および取り消しを行うことができるのは、自分がすでに持っている権限だけです。(ファイルに対するオブジェクト操作権限も持っていなければなりません。)
- ファイルを変更する。
- ファイルにメンバーを追加する。(ファイルの所有者が新しいメンバーの所有者となります。)
- ファイルのメンバーを変更する。
- ファイルを移動する。
- ファイルの名前を変更する。
- ファイルのメンバーの名前を変更する。
- ファイルのメンバーを消去する。(データの削除権限も必要です。)
- ファイルのメンバーを初期設定する。(省略時レコードで初期設定するためにはデータの追加権限も必要です。削除済みレコードで初期設定するためにはデータの削除権限も必要です。)
- ファイルのメンバーを再編成する。(すべてのデータ権限も必要です。)

オブジェクト更新権限

ユーザーは、オブジェクト管理権限と同じ操作の大半のオブジェクト変更権限が必要になります (前項を参照)。オブジェクト変更権限は、オブジェクト管理権限に取って代わる権限です。

オブジェクト参照権限

あるオブジェクトを別のオブジェクトが参照する権限が必要な場合には、ユーザーはオブジェクト参照権限が必要です (この場合、そのオブジェクトの操作は参照側オブジェクトによって制限を受けます)。

物理ファイルの参照制約を追加すると、親ファイルに対するオブジェクト管理権限またはオブジェクト参照権限のどちらかの存在が検査されます。物理ファイルの制約については、237 ページの『制約によってユーザーのデータベースの保全性を制御する』 および 243 ページの『参照制約を使用したデータの保全性の保証』に説明があります。

データベース・ファイルのデータ権限のタイプ: 以下のデータ権限または許可を使用して、ユーザーに物理ファイルおよび論理ファイルへのアクセス権限を付与することができます。

読み取り権限

ユーザーは、ファイル内のレコードを読み取ることができます。

追加権限

ユーザーは、ファイルに新しいデータを追加することができます。

更新権限

ユーザーは、既存のレコードを更新することができます。(更新のためにレコードを読み取るには、読み取り権限も持っていなければなりません。)

削除権限

ユーザーは、既存のレコードを削除することができます。(削除のためにレコードを読み取るには、読み取り権限も持っていなければなりません。)

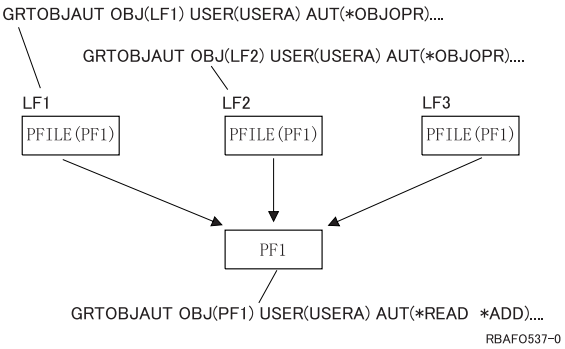
実行権限

実行権限を使用してライブラリーを使用したり、プログラムを呼び出したりできます。たとえば、トリガーに関連したファイルを変更するには、トリガー・プログラムに対する実行権限を持っていなければなりません。実行権限がない場合には、システムはトリガー・プログラムを呼び出しません。トリガーについての詳細は、255 ページの『データベース内での自動イベントのトリガー』を参照してください。

通常、ユーザーがファイル内のデータに対して持っている権限は、実際に入出力操作を行うまで検査されません。ただし、Query ファイルのオープン (OPNQRYF) コマンドとデータベース・ファイルのオープン (OPNDBF) コマンドを使用する場合には、ファイルのオープン時にもデータ権限が検査されます。

ファイルに対するオブジェクト操作権限が付与されていないユーザーの場合、そのユーザーはファイルをオープンすることができません。

以下の例は、論理ファイルに対して付与された権限と、論理ファイルが使用する物理ファイルに対して付与された権限との関係を示しています。論理ファイル LF1、LF2、および LF3 は、物理ファイル PF1 が基礎となっています。USERA は、PF1 中のデータに対する読み取り権限 (*READ) および追加権限 (*ADD)、および LF1 と LF2 に対するオブジェクト操作権限 (*OBJOPR)、読み取り権限 (*READ) および追加権限 (*ADD) を持っています。USERA は、PF1 に対するオブジェクト操作権限 (*OBJOPR) を持っていない ので、PF1 をオープンしたり、 PF1 のデータを直接に使用することはできません。代わりに、USERA は、LF1 および LF2 をオープンして、 LF1 および LF2 を介して PF1 からレコードを読み取ったり、 PF1 にレコードを追加することができます。このユーザーには LF3 に関する権限は与えられていないので、 LF3 を使用することはできない点に注意してください。



共通権限の指定

ファイルを作成するときに、 物理ファイルの作成コマンド、または、 ソース物理ファイルの作成コマンドに AUT パラメーターを使用して、共通権限を指定することができます。**共通認可**とは、該当のファイルに関する特定権限を持っていないユーザー、あるいは該当のファイルに対する特定権限を持つグループに属していないユーザーが、だれでも利用できる権限です。共通認可は、検査の対象としては最後に検査される権限です。すなわち、ユーザーがあるファイルに対して特定権限を持っている場合、あるいは、ユーザーが特定権限を持つグループに属するメンバーである場合には、共通認可の検査は行われません。共通認可は以下のように指定することができます。

- *LIBCRTAUT。該当のファイルを作成するときには、ファイルの共通認可を判別するために、そのファイルが作成されるライブラリーが検査されます。権限はおのおののライブラリーに与えられます。権限はライブラリーを作成する際に指定し、 *LIBCRTAUT 値をファイル作成 (CRTLF、CRTPF、および CRTSRCPF) コマンドの AUT パラメーターに指定した場合には、そのライブラリー内に作成されるファイルにはすべてこの共通認可が与えられます。*LIBCRTAUT 値は省略時の共通認可です。
- *CHANGE。該当のファイルに対する特定のユーザー権限またはグループ権限を持っていないすべてのユーザーが、そのファイル中のデータを変更する権限を持ちます。
- *USE。該当のファイルに対する特定のユーザー権限または、グループ権限を持っていないすべてのユーザーが、そのファイル中のデータを読み取る権限を持ちます。
- *EXCLUDE。該当のファイルの所有者、機密保護担当者、特定権限を持つユーザー、または特定権限を持つグループに属するユーザーだけが、そのファイルを使用することができます。
- *ALL。該当のファイルに対する特定のユーザー権限またはグループ権限を持っていないすべてのユーザーが、すべてのデータ権限と、オブジェクト操作、オブジェクト管理、およびオブジェクト存在権限を持ちます。
- 権限リスト名。権限リストは、ユーザーとその権限のリストです。このリストを用いて、ユーザーとユーザーが持つ種々の権限をグループにまとめることができます。

注: 論理ファイルの作成時に、データ権限の付与は行われません。したがって、*CHANGE を指定しても *USE を指定するのと同じであり、また *ALL を指定してもデータ権限は付与されません。

共通権限は、以下の方法で認可することができます。

- iSeries ナビゲーターを使用して、共通権限を定義する。 99 ページの『iSeries ナビゲーターを使用したファイルの共通権限の定義』を参照してください。
- オブジェクト権限の編集 (EDTOBJAUT) コマンド、 オブジェクト権限の付与 (GRTOBJAUT) コマンド、または、 オブジェクト権限の取り消し (RVKOBJAUT) コマンドを使用して、ファイルの共通権限の付与または取り消しを行う。

さらに、 iSeries ナビゲーターを使用して、新規ファイルに対するデフォルトの共通権限を設定することもできます。 『iSeries ナビゲーターを使用した新規ファイルに対するデフォルトの共通権限の設定』を参照してください。

iSeries ナビゲーターを使用したファイルの共通権限の定義: 共通権限は、システム上のすべてのオブジェクトに定義され、オブジェクトに対して特定のアクセス権限を持っていないユーザーのアクセスのタイプを記述します。共通権限を定義するには、以下のようにします。

1. iSeries ナビゲーター・ウィンドウで、使用したいシステムを拡張する。
2. 許可を編集する対象のオブジェクトが見えるまで、ナビゲートする。
3. 許可を追加する対象のオブジェクトを右クリックし、「**許可**」を選択する。
4. 「**許可**」ウィンドウで、グループ・リストから「**共通**」を選択する。
5. 「**詳細**」ボタンをクリックして、詳細許可を実施する。該当するチェック・ボックスをチェックして、共通に対する必要な許可を適用する。
6. **OK** をクリックする。

iSeries ナビゲーターを使用した新規ファイルに対するデフォルトの共通権限の設定: デフォルトの共通権限を設定すると、ユーザーは、新規オブジェクトがライブラリーに作成されるときに割り当てられる共通の権限を持つことができます。異なるレベルの機密保護が必要なオブジェクトには、許可を個々に編集することができます。デフォルトの共通権限を設定するには、以下のようにします。

1. iSeries ナビゲーター・ウィンドウで、使用したいシステムを拡張する。
2. **データベース**を拡張する。
3. 操作したいデータベースおよびライブラリーを拡張する。
4. 共通権限を設定したいライブラリーを右クリックし、「**許可**」を選択する。
5. 「**許可**」ウィンドウで、「**新規オブジェクト**」をクリックする。
6. 「**新規オブジェクト**」ウィンドウで、「デフォルト共通権限」を選択する。
7. 許可リストを割り当てるために、許可リストの名前を入力するか**ブラウズする**。許可リストのプロパティーを表示するために、「**オープン**」を選択する。
8. **OK** をクリックする。

システム値
新規オブジェクトのデフォルト共通権限にシステム値を使用することを指定します。

使用
オブジェクト属性へのアクセスとオブジェクトの使用を許可します。公衆 (一般ユーザー) はオブジェクトを表示することはできても、変更はできません。

変更
オブジェクトの内容を変更することを許可します (例外あり)。

すべて
オブジェクトに対するすべての操作 (所有者に限定されているものを除く) を許可します。ユーザーまたはグループは、オブジェクトの存在を制御し、オブジェクトの機密保護を指定し、オブジェクトを変更し、オブジェクトに基本機能を実行することができます。ユーザーまたはグループは、オブジェクトの所有権を変更することもできます。

除外
オブジェクトに対するすべての操作が禁止されます。この許可を持つユーザーまたはグループは、オブジェクトに対してアクセスも操作も行えません。公衆はオブジェクトを使用できないことを指定します。

許可リストの使用

ユーザーは、オブジェクトを機密保護するために使用する許可リストを指定することができます。

入出力操作を制御するデータベース・ファイル処理可能性の使用

ファイルの処理可能性を使用すると、データベース・ファイルの権限から独立して、データベース・ファイルについて許可される入出力操作を制御することができます。

物理ファイルを作成するときに、 物理ファイルの作成 (CRTPF) コマンドおよび ソース物理ファイルの作成 (CRTSRCPF) コマンドで ALWUPD パラメーターおよび ALWDLT パラメーターを使用することによって、ファイルが更新可能であるかまたは削除可能であるかを指定することができます。更新不能で、かつ削除不能であるファイルを作成すれば、ファイルに一度書き込まれたデータを変更または削除できない環境を効率的に実現することができます。

ファイルの処理可能性は、論理ファイルに対して明示的に設定することはできません。論理ファイルの処理可能性は、その基礎となる物理ファイルの処理可能性によって判別されます。

ファイルを作成した後には、ファイルの処理可能性を変更することはできません。ファイルを削除してから、希望の処理可能性を持つファイルを再作成しなければなりません。ファイル記述の表示 (DSPFD) コマンドを使用して、ファイルの処理可能性を調べることができます。

データベース・ファイルの特定のフィールドへのアクセスの制限

物理データベース・ファイルの特定のフィールドに対するユーザーの更新および読み取り要求を制限することができます。これを行うためには、以下の 2 つの方法があります。

- ユーザーにアクセスさせたいフィールドだけを含む、物理ファイルの論理ビューを作成する。詳細については、『データを機密保護するための論理ファイルの使用』を参照してください。
- SQL GRANT ステートメントを使用して、SQL 表の特定の列に対する更新権限を付与する。詳しくは、SQL プログラミングに関するトピックを参照してください。

データを機密保護するための論理ファイルの使用

論理ファイルを使用して、物理ファイル内のあるフィールドが表示されないようにすることができます。これは、ユーザーに見せたくないフィールドが入っていない論理ファイル・レコード様式を記述することによって実現されます。この主題の詳細については、 47 ページの『論理ファイルのレコード様式の記述』 を参照してください。

また、論理ファイルを使用して、物理ファイル内の 1 つまたは複数のフィールドが変更されないようにすることもできます。そのためには、保護したいフィールドについて、DDS 形式の 38 列に I (入力専用) を指定します。この主題の詳細については、 49 ページの『論理ファイルのフィールドの用途の記述』 を参照してください。

論理ファイルを使用して、物理ファイルのレコードの 1 つまたは複数のフィールドの内容に基づいて、そのレコードを保護することができます。フィールドの内容に基づいてレコードを保護するためには、論理ファイルを記述するときに選択/除外キーワードを使用します。この主題の詳細については、 54 ページの『論理ファイルを用いてのレコードの選択と除外』 を参照してください。

データベース・ファイルの処理

以下のトピックでは、プログラムでデータベース・ファイルを処理する方法について説明しています。

『データベース・ファイル処理: 実行時の考慮事項』
このトピックには、プログラムまたはジョブでのファイルの使い方の計画と、プログラムのパフォーマンスの向上に関する説明が含まれています。さらに、ファイル処理の効率を高めるために指定できるファイル処理パラメーターと実行時オプションについても説明されています。このトピックには、データベース・ファイルを同時に複数のユーザーによってアクセスできるようにする、複数のジョブによるデータベース・ファイルの共用についての説明もあります。さらに、ファイル、レコード、あるいはメンバーがジョブ間で共用されることを防止するためのロックについても説明されています。

124 ページの『データベース・ファイルのオープン』
このトピックでは、プログラムの中でデータベース・ファイル・メンバーをオープンするための、Query ファイルのオープン (OPNQRYF) コマンド、および、データベース・ファイルのオープン (OPNDBF) コマンドを使用する方法が説明されています。高水準言語プログラムを作成する際の例、パフォーマンスの考慮事項、および従うべき指針も記載されています。さらに、発生する可能性のある代表的なエラーについても説明されています。

177 ページの『プログラム内での基本データベース・ファイル操作』
このトピックでは、基本データベース操作について説明されています。これには、データベース・ファイルにおける位置の設定と、データベース・ファイルのレコードの読み取り、更新、追加、および削除が含まれます。データベース・レコードを読み取るためのいくつかの方法も説明されています。更新に関する説明では、論理ファイルまたは物理ファイルの中の既存のデータベース・レコードを変更する方法が説明されています。書き込み操作を用いて物理データベース・メンバーに新しいレコードを追加する方法の説明も含まれています。

189 ページの『データベース・ファイルのクローズ』
このトピックには、プログラムがデータベース・ファイル・メンバーの処理を完了し、プログラムをファイルから切断するときに、データベース・ファイルをクローズする方法が記載されています。

189 ページの『プログラム内でのデータベース・ファイル・エラーのモニター』
このトピックには、プログラムでデータベース・ファイル・エラーを処理するときにモニターするメッセージについての説明があります。

データベース・ファイル処理: 実行時の考慮事項

ファイルをオープンして処理を始める前に、プログラムやジョブでファイルをどのように使用するかを考えておく必要があります。実行時のファイル処理パラメーターを十分に把握すれば、予期しない結果となることを避けることができ、さらに、プログラムのパフォーマンスを向上させることにもなります。

ファイルのオープン時には、データベース・ファイル記述にある属性がプログラム中のパラメーターと組み合わされます。通常、プログラムがファイルをオープンして処理するためにシステムで必要とされる情報の大部分は、ファイル属性や、アプリケーション・プログラム自体の中で検出されます。

しかし、場合によっては、ファイルやプログラム内で検出された処理パラメーターの一時変更が必要となることがあります。たとえば、ファイルの最初のメンバー以外のメンバーを処理したいときには、処理したいメンバーを使用するようシステムに指示する方法が必要となります。データベース・ファイルによる一時変更 (OVRDBF) コマンドを使用すれば、これを行うことができます。OVRDBF コマンドを使用すると、ジョブのパフォーマンスの向上に役立ち、ファイル属性やプログラムでは指定できない処理パラメーターを、

指定できます。OVRDBF コマンドのパラメーターは、ファイルやプログラムの属性よりも優先されます。統合言語環境（Integrated Language Environment®）における一時変更の作用の仕方について、詳しくは、

「ILE 概念  」を参照してください。

この章では、ファイル処理パラメーターについて説明し、データベース・ファイルの操作を制御するその他の方法や考慮事項を示します。パラメーターの値は、高水準言語プログラム、ファイル属性、および高水準プログラムの呼び出しの前に処理されるオープン・コマンドまたは一時変更コマンドによって決まります。

これらのパラメーターおよび方法について、以下のトピックを参照してください。

- 『ファイル名およびメンバー名』
- 103 ページの『ファイル処理オプション』
- 106 ページの『データの回復と保全性』
- 107 ページの『共用データのロック』
- 111 ページの『同一ジョブまたは活動化グループ内のデータベース・ファイルの共用』
- 118 ページの『データベース・ファイルの順次のみの処理』

これらのパラメーターの要約と、これらのパラメーターを指定する場所については、 121 ページの『データベース・ファイル処理実行時の考慮事項の要約』を参照してください。記憶域プールのページング・オプション、およびそれがデータベース・ファイル処理に与える影響については、 124 ページの『データベースのパフォーマンスに与える記憶域プールのページング・オプションの影響』を参照してください。

コマンドの処理パラメーターの詳細については、 制御言語 (CL) のトピックに含まれる以下のコマンドを参照してください。

- 物理ファイルの作成 (CRTPF)
- 論理ファイルの作成 (CRTLF)
- ソース物理ファイルの作成 (CRTSRCPF)
- 物理ファイル・メンバーの追加 (ADDPFM)
- 論理ファイル・メンバーの追加 (ADDLFM)
- 物理ファイルの変更 (CHGPF)
- 物理ファイル・メンバーの変更 (CHGPFM)
- 論理ファイルの変更 (CHGLF)
- 論理ファイル・メンバーの変更 (CHGLFM)
- ソース物理ファイルの変更 (CHGSRCPF)
- データベース・ファイルによる一時変更 (OVRDBF)
- データベース・ファイルのオープン (OPNDBF)
- Query ファイルのオープン (OPNQRYF)
- ファイルのクローズ (CLOF)

ファイル名およびメンバー名

FILE パラメーターおよび MBR パラメーター。データベース・ファイルのデータを処理する前には、どのファイルとメンバーを使用したいかを指示しなければなりません。通常、高水準言語プログラムの中でファイル名と任意指定のメンバー名を指定します。システムは、ユーザーのプログラムがファイルをオープンするように要求したときに、この指定された名前を使用します。プログラムで指定されているファイル名を一時変更して別のファイルをオープンするには、データベース・ファイルによる一時変更 (OVRDBF) コマ

ンドで TOFILE パラメーターを使用することができます。プログラムの中でメンバー名が指定されていない場合には、ファイルの最初のメンバー (作成の日付および時刻によって定義される) が処理されます。

高水準言語プログラムでメンバー名を指定できない場合 (高水準言語の中にはメンバー名の指定ができないものがあります)、または最初のメンバー以外のメンバーを処理したい場合は、データベース・ファイルによる一時変更 (OVRDBF) コマンドまたはオープン・コマンド (OPNDBF または OPNQRYF) を使用して、処理したいファイルおよびメンバーを指定できます (FILE パラメーターおよび MBR パラメーターを使用します)。

あるファイルのすべてのメンバーを処理したい場合には、 MBR(*ALL) パラメーターを指定した OVRDBF コマンドを使用します。たとえば、FILEX に 3 つのメンバーがあり、そのメンバーを 3 つとも処理したい場合には、以下のように指定します。

```
OVRDBF   FILE(FILEX)  MBR(*ALL)
```

OVRDBF コマンドで MBR(*ALL) を指定すると、プログラムは作成された順序でメンバーを読み取ります。ファイルがキー順または到着順のどちらのファイルであるかによって、プログラムは各メンバーのレコードをキー順または到着順で読み取ります。

ファイル処理オプション

以下の項では、実行時処理オプションをいくつか説明します。プログラムが使用するファイル操作の指示、ファイルの開始位置の指定、削除済みレコードの再使用、キー順アクセス・パスの無視、ファイル終わり処理の処理方法、およびファイルのレコード長の指定などを行うためのオプションがあります。

これらのトピックを参照してください。

- 『処理のタイプの指定』
- 104 ページの『初期ファイル位置の指定』
- 104 ページの『削除済みレコードの再使用』
- 104 ページの『キー順アクセス・パスの無視』
- 105 ページの『ファイル終わり処理の遅延』
- 105 ページの『レコード長の指定』
- 105 ページの『レコード様式の無視』
- 105 ページの『重複キーが存在するかどうかの判別』

処理のタイプの指定: OPTION パラメーター。プログラム中でファイルを使用する場合、そのファイルに対して予定している操作のタイプをシステムに知らせる必要があります。たとえば、ファイル内のデータを単に読み取ろうとしているのか、それともデータを読み取って更新しようとしているのかをシステムに知らせる必要があります。使用できる操作オプションとしては、入力、出力、更新、および削除があります。システムは、高水準言語プログラムに指定されている情報、あるいはデータベース・ファイルのオープン (OPNDBF) コマンドや Query ファイルのオープン (OPNQRYF) コマンドの OPTION パラメーターから、ユーザーが使用しようとしているオプションを判別します。

システムはこのようなオプションを使用して、プログラムの中で使用できる操作を決めます。たとえば、ファイルを入力専用にオープンしているのに、プログラムで出力操作をしようとすると、プログラムはエラーを受け取ることになります。

通常、システムは、ユーザーのプログラムで入出力操作が行われるときに、ユーザーに必要なデータ権限があるかどうかを検査します。ただし、Query ファイルのオープン (OPNQRYF) またはデータベース・ファイルのオープン (OPNDBF) コマンドを使用する際には、システムは、ファイルのオープン時に、ユーザー

に OPTION パラメーターで指定された操作を実行するのに必要なデータ権限があるかどうかを検査します。データ権限の詳細については、 97 ページの『データベース・ファイルのデータ権限のタイプ』を参照してください。

さらに、システムは、使用するロックを判別するのにも、これらのオプションを使用します。ロックは、プログラムで処理するファイルやレコードのデータ保全性を保護するのに使用されます。ロックに関する詳細については、 107 ページの『共用データのロック』を参照してください。

初期ファイル位置の指定: POSITION パラメーター。ファイルをオープンした後、ファイルの処理をどこから開始するかをシステムは知る必要があります。省略時では、ファイルの最初のレコードの直前から開始されます (最初の順次読み取り操作で最初のレコードが読み取られます)。しかし、データベース・ファイルによる一時変更 (OVRDBF) コマンドを使用すると、ファイルの最後から開始したり、ファイルの途中の特定レコードから開始したりするように、システムに指示することができます。さらに、プログラム中でファイルの位置を動的に設定することもできます。プログラム中でのファイルの位置の設定の詳細については、 177 ページの『ファイル内での位置の設定』を参照してください。

削除済みレコードの再使用: REUSEDLT パラメーター。物理ファイルの作成 (CRTPF) コマンドまたは物理ファイルの変更 (CHGPF) コマンドで REUSEDLT(*YES) と指定すると、以下の操作が異なった動作をすることがあります。

- 到着順は、削除済みレコードのスペースを再使用するファイルについては意味がなくなります。レコードがファイルの終わりに追加されなくなる可能性があるためです。
- 削除済みレコードのスペースを再使用するファイルについては、ファイル終わり遅延は働きません。
- 削除済みレコードのスペースの 100% の再使用は保証されません。削除済みレコードのスペースがファイルにまだあるにもかかわらず、*ファイル満杯* の状態となったり、ファイルが拡張されたりすることがあります。

| システムは一定の方法で削除済みレコードのスペースを再使用するため、削除済みレコードのスペースを再
| 使用するような、以下のタイプのファイルを作成または変更しないでください。

| • 相対レコード番号を用いて処理されるファイルと、別のファイルへのキーとして使用される相対レコー
| ド番号を判別するためにアプリケーション・プログラムで使用されるファイル。
| • 待ち行列として使用されるファイル
| • 新しいレコードがファイルの終わりに挿入されるものとみなしているアプリケーション・プログラムに
| よって使用されるファイル。
| • 行が更新、挿入、または削除された場合に並列索引を維持する対象のファイル (DB2 UDB 対称マルチプ
| ロセスのインストール時)

削除済みレコードのスペースを再使用するよう既存の物理ファイルを変更する場合に、物理ファイルに対する重複キーの順序付けが LIFO または FIFO であるアクセス・パスを持つ論理ファイルにあるときには、以下の方法によって、FIFO または LIFO 属性を持たない論理ファイルを再作成し、既存のアクセス・パスの再作成を避けることができます。

1. FIFO または LIFO 属性を持つ既存の論理ファイルの名前を変更します。
2. 名前を変更したファイルと同じ 2 番目の論理ファイルを作成します。ただし、重複キーの順序付けはファイルに指定しないことに注意してください。新しいファイルに元のファイル名を付けます。新しいファイルは名前が変更されたファイルのアクセス・パスを共用します。
3. 名前を変更したファイルを削除します。

キー順アクセス・パスの無視: ACCPTH パラメーター。キー順アクセス・パスを持つファイルを処理する場合、通常、データの取り出し検索にそのアクセス・パスが使用されます。キー・フィールドをファイル

に定義している場合、システムは自動的にキー順アクセス・パスを使用します。ただし、場合によっては、キー順アクセス・パスを無視して到着順でファイルを処理することによって、さらに高いパフォーマンスが得られることがあります。

一部の高水準言語や、データベース・ファイルのオープン (OPNDBF) コマンドでは、キー順アクセス・パスを無視するようにシステムに指示することができます。キー順アクセス・パスを無視すると、キーでデータを読み取る操作はできなくなります。操作は到着順アクセス・パスにより順次に行われます。(選択/除外値が定義されている論理ファイルにこのオプションを指定している場合、到着順アクセス・パスが使用され、選択/除外値に合致するレコードだけがプログラムに戻されます。つまり、操作は、ファイルに DYNSLT キーワードを指定しているかのように行われます。)

注: 複数の物理ファイル・メンバーを基礎としている論理ファイル・メンバーについては、キー順アクセス・パスを無視できません。

ファイル終わり処理の遅延: EOFDLY パラメーター。データベース・ファイルの読み取り中にプログラムがデータの終わりに達すると、システムは通常、読み取るデータがないことをプログラムに知らせます。しかし、場合によっては、プログラムにデータがないことを知らせる代わりに、ファイルにさらにデータが到着するまでシステムがプログラムを保留する方が望ましい場合があります。ファイルの中にさらにデータが到着すると、プログラムは新しく到着したレコードを読み取ることができます。データベース・ファイルによる一時変更 (OVRDBF) コマンドで EOFDLY パラメーターを使用すると、このタイプの処理を行うことができます。このパラメーターの詳細については、181 ページの『ファイル終わりに達した時のレコード待ち』を参照してください。

注: ファイル終わり遅延は、削除済みレコードを再使用するファイルには使用するべきではありません。

レコード長の指定: システムはプログラムで処理するレコードの長さを知る必要がありますが、ユーザーがプログラム中でレコード長を指定する必要はありません。システムはプログラム中に指示されているファイルの属性や記述からの情報で、レコード長を自動的に判別します。ただし、高水準言語プログラムでは任意選択としてレコード長を指定することができます。

オープンされるファイル内のレコードがプログラムで指定した長さよりも長い場合、システムはファイル・メンバーのレコード長に見合う記憶域を割り振り、このオプションを無視します。この場合、レコード全体がプログラムに渡されます。(ただし高水準言語の中には、プログラムに指定したレコード長で定義されているレコードの一部だけをアクセスすることができるものもあります。) オープンされるファイル内のレコードがプログラムで指定した長さよりも短い場合、システムはプログラムに指定されているレコード長に対して記憶域を割り振ります。プログラムは余分な記憶域を使用することができますが、ファイル・メンバーに定義されたレコード長だけが入出力操作に使用されます。

レコード様式の無視: 複数様式論理ファイルを使用する場合には、システムは、そのファイルで定義されているすべての様式をユーザーが使用するものとみなします。しかし、すべての様式を使用する必要がない場合には、どの様式を使用し、どの様式を無視するかを指定できます。様式を無視するためにこのオプションを使用しない場合には、プログラムはファイルで定義されているすべての様式を処理できます。この処理オプションの詳細については、高水準言語の手引きを参照してください。

重複キーが存在するかどうかの判別: DUPKEYCHK パラメーター。キーが重複キーかどうかを判別するのに使用されるキー順アクセス・パスのセットは、実行する入出力操作によって異なります。

入力 (読み取り) 操作においては、ファイルがオープンされたときのキー順アクセス・パスが使用されます。物理ファイルに対して存在するこれ以外のキー順アクセス・パスはどれも考慮されません。また、選択/除外仕様のために除外されたキー順アクセス・パスは、キー操作の重複を調べる場合に考慮されません。

出力 (書き込み) 操作および更新操作においては、この出力操作または更新操作のキーが重複キーであるかどうかの判定に、物理ファイルに対して存在する *IMMED メインテナンスの非固有キー順のすべてのアクセス・パスが検索されます。アクセス・パスがフィードバック時にファイルに対して活動状態でオープンする場合は、*RBLD および *DLY メインテナンスがあるキー順アクセス・パスのみが考慮対象となります。

COBOL プログラムでキー順ファイルを処理するときには、 COBOL 言語を使用して、あるいはデータベース・ファイルのオープン (OPNDBF) または Query ファイルのオープン (OPNQRYF) コマンドを使用して、重複キー・フィードバックがプログラムに返されるように指定できます。ただし、COBOL で重複キー・フィードバックが返されるようにすると、パフォーマンスが低下する場合があります。

データの回復と保全性

以下の項では、データ保全性に関する実行時の考慮事項について説明します。これらのトピックを参照してください。

- 『ジャーナル処理およびコミットメント制御によるファイルの保護』
- 『データおよびアクセス・パスの補助記憶装置への書き出し』
- 『レコード様式記述に対する変更の検査』
- 107 ページの『ファイルの満了日の検査』
- 107 ページの『ジョブによるファイル内データの変更の防止』

ジャーナル処理およびコミットメント制御によるファイルの保護: COMMIT パラメーター。ジャーナル処理およびコミットメント制御は、iSeries システムでのデータおよびトランザクションを回復するための望ましい手法です。データベース・ファイルのジャーナル処理は、ファイルについてジャーナル物理ファイルの開始 (STRJRNPF) コマンドを実行することによって開始されます。アクセス・パスのジャーナル処理は、ファイルについてジャーナル・アクセス・パスの開始 (STRJRNAP) コマンドを実行するか、またはシステム管理のアクセス・パス保護 (SMAPP) を使用することによって開始されます。ファイルをコミットメント制御のもとで実行することをシステムに伝えるには、コミットメント制御の開始 (STRCMTCTL) コマンドと、高水準言語の仕様を使用します。また、データベース・ファイルのオープン (OPNDBF) コマンドおよび Query ファイルのオープン (OPNQRYF) コマンドでコミットメント制御 (COMMIT) パラメーターを指定することもできます。ジャーナル処理とコミットメント制御について詳しくは、 213 ページの『ジャーナル管理』、および、 220 ページの『コミットメント制御を使用したデータの保全性の保証』を参照してください。

参照制約に関連するファイルに対して挿入、更新、または削除を実行する場合に、削除規則、更新規則、あるいはその両方が RESTRICT 以外であるときには、ジャーナル処理を使用しなければなりません。参照制約の詳細については、 243 ページの『参照制約を使用したデータの保全性の保証』を参照してください。

データおよびアクセス・パスの補助記憶装置への書き出し: FRCRATIO パラメーターおよび FRCACCPTH パラメーター。変更されたデータをいつ主記憶域から補助記憶装置へ書き出すかは、通常、DB2 UDB for iSeries によって決定されます。データベースの変更が補助記憶装置に書き出される時を制御したい場合には、データベース・ファイルの作成、変更、または一時変更コマンドで強制書き出し頻度 (FRCRATIO) パラメーターを使用し、データベース・ファイルの作成および変更コマンドでアクセス・パスの強制書き出し (FRCACCPTH) パラメーターを使用することができます。FRCRATIO パラメーターおよび FRCACCPTH パラメーターを使用するときには、システムのパフォーマンスと回復について考慮する必要があります。このような考慮事項を把握するには、 212 ページの『データベースの回復と復元』を参照してください。

レコード様式記述に対する変更の検査: LVLCHK パラメーター。ファイルのオープン時に、システムは、ユーザーが使用するレコード様式の記述が、プログラムのコンパイル以降に、プログラムがファイルを

処理できないほどに変更されたかどうかを検査します。このような状況の場合には、システムは通常、プログラムに通知します。この状態はレベル検査と呼ばれます。ファイルの作成または変更コマンドを使用する際に、このレベル検査を実行することを指定できます。また、データベース・ファイルによる一時変更 (OVRDBF) コマンドで LVLCHK パラメーターを使用して、ファイルについて定義されているレベル検査属性を一時変更することもできます。このパラメーターの詳細については、208 ページの『ファイル記述内のフィールドの変更の影響』を参照してください。

ファイルの満了日の検査: EXPDATE パラメーターおよび EXPCHK パラメーター。システムは、指定されたファイル中のデータが現行のものかどうかを検査できます。ファイルの作成または変更コマンドに EXPDATE パラメーターを使用すると、ファイルまたはメンバーの満了日を指定することができます。さらに、データベース・ファイルによる一時変更 (OVRDBF) コマンドで EXPCHK パラメーターを使用すると、システムが指定の日付を検査するかどうかを指定することもできます。満了日の検査を行い、現在の日付が満了日より後である場合には、ファイルのオープン時にメッセージがシステム操作員に送られます。

ジョブによるファイル内データの変更の防止: INHWRT パラメーター。プログラムをテストしたいものの、テストに使用するファイル中のデータを実際に変更したくない場合は、システムに、プログラムによる変更をファイルへ書き込まない (禁止する) ように指示できます。ファイルへの変更をすべて禁止するためには、データベース・ファイルによる一時変更 (OVRDBF) コマンドで INHWRT (*YES) を指定します。

共用データのロック

すべてのデータベース・ファイルは、定義上、複数のユーザーにより同時に使用できます。しかし、一部の操作では、ファイル、メンバー、またはメンバー内のデータ・レコードが複数のジョブで共用されないようにするために、ロックすることができます。ファイル、メンバー、またはレコードがロックされている間は、他のジョブがその同じデータを読み取って更新を行うことはできません。また、最初のジョブによる更新を、別のジョブが不用意に削除することも防げます。

表を開いてロックしたい行を編集することで、 iSeries ナビゲーターの行をロックすることができます。また、 SQL LOCK TABLE ステートメントを使用することもできます。または、以下の操作を使用してファイル、メンバー、あるいはデータ・レコードをロックすることができます。

- 『レコードのロック』
- 108 ページの『ファイルのロック』
- 109 ページの『メンバーのロック』
- 109 ページの『レコード様式データのロック』

ロックされたレコードは、以下のメソッドのどちらかを使用して表示することができます。

- 108 ページの『DSPRCDLCK を使用したロックされたレコードの表示』
- 108 ページの『iSeries ナビゲーターを使用したロックされた行の表示』

一般に使用されるデータベース機能と、その機能がデータベース・ファイルに対して行うロックのタイプの一覧表については、データベース・ロックに関する考慮事項を参照してください。

レコードのロック: WAITRCD パラメーター。DB2 UDB for iSeries には、レコードの保全性が組み込まれています。たとえば、PGMA がレコードを更新のために読み取る場合、そのレコードをロックします。別のプログラムは、PGMA がこのレコードを解放するまで、同じレコードを更新のために読み取ることはできません。しかし単に照会のためだけであれば、レコードを読み取ることができます。このような方法でシステムは、データベースの保全性を確保します。

システムは、プログラムで指定されているファイル処理のタイプと、要求された操作に基づいてロック条件を判別します。たとえば、オープン・オプションに更新または削除が含まれている場合、複数ユーザーが同時にレコードを読み取ることができても、 1 人のユーザーしかレコードを更新することができないように、レコードの読み取りのたびにロックされます。

システムは通常、ロックされたレコードが解放されるのを指定された秒数だけ待ち、その後、ユーザーが要求しているレコードを入手することができないというメッセージをプログラムに送ります。省略時のレコード待ち時間は 60 秒です。しかし、ファイルの作成コマンド、ファイルの変更コマンド、およびデータベース・ファイルの一時変更コマンドに WAITRCD パラメーターを指定して、独自の待ち時間を設定することができます。必要とするレコードが別の操作によってロック中であるということがプログラムに通知された場合、プログラムに適切な処置を取らせる (たとえば、要求されたレコードが現在使用できないというメッセージを操作員に送る) ことができます。

| 参照保全 CASCADE DELETE、SET NULL、または SET DEFAULT 削除規則の結果としてレコード・ロ
| ックが暗黙的に獲得される場合には、ロック待ち時間が 30 秒に制限されます。

ロックされたレコードが更新または削除されると、システムは自動的にロックを解除します。ただし、レコードを更新しないでレコード・ロックを解除することもできます。レコード・ロックの解除方法については、該当の高水準言語の手引きを参照してください。

注: コミットメント制御を使用すると、レコード・ロックの規則が変わります。コミットメント制御、およびそれがレコード・ロック規則に与える影響についての詳細は、 コミットメント制御のトピックを参照してください。

ロックされたレコードは、以下のメソッドのどちらかを使用して表示することができます。

- 『DSPRCDLCK を使用したロックされたレコードの表示』
- 『iSeries ナビゲーターを使用したロックされた行の表示』

iSeries ナビゲーターを使用したロックされた行の表示: iSeries ナビゲーターを使用して、ロックされた行を表示することができます。

1. iSeries ナビゲーター・ウィンドウで、使用したいシステムを拡張する。
2. データベースを拡張する。
3. ライブラリーを拡張する。
4. ロックされた行の情報を表示したい表が入っているライブラリーをクリックする。
5. 表を右クリックして、「ロックされた行」を選択する。
6. 「ロックされた行」ウィンドウが、ロックされた行を表示します。

DSPRCDLCK を使用したロックされたレコードの表示: レコード・ロックの表示 (DSPRCDLCK) コマンドを使用すると、物理ファイル・メンバーのレコードの現在のロック状況 (待機または保留) を表示することができます。 制御言語 (CL) に関するトピックの DSPRCDLCK (レコード・ロックの表示) コマンドを参照してください。さらに、このコマンドは、どのタイプのロックが現在保留されているかを表示します。

(ロック・タイプの詳細については、「バックアップおよび回復の手引き」を参照してください。) 指定するパラメーターに応じて、このコマンドでは特定のレコードのロック状況またはメンバー内のすべてのレコードのロック状況が表示されます。さらに、「ジョブの処理 (WRKJOB)」画面からレコード・ロックを表示することもできます。

ファイルのロック: WAITFILE パラメーター。操作期間を通じて、ファイルを排他的に割り振るようなファイル操作もあります。ファイルが排他的に割り振られている間は、そのファイルをオープンしようとする

どのプログラムも、ファイルが解放されるまで待たなければなりません。ファイルが使用可能になるまでプログラムが待つ時間を制御するには、ファイルの作成コマンド、ファイルの変更コマンド、およびデータベース・ファイルの一時変更コマンドの WAITFILE パラメーターで待ち時間を指定します。待ち時間を特に要求していない場合には、システムはファイル待ち時間の省略時である 0 秒を使用します。

ファイルの属性を変更する操作の実行中は、そのファイルは排他的に割り振られています。このような操作 (移動、名前変更、権限の認可または取り消し、所有者の変更、削除など) は、同じファイルまたは同じファイルのメンバーに対する他の操作と同時に実行することはできません。他のファイル操作 (表示、オープン、ダンプ、オブジェクト検査など) はファイル定義だけを使用するので、ファイルのロックの排他性は低くなります。このような操作はお互いに同時に実行したり、または 1 つのメンバーの入出力操作と同時に実行したりすることができます。

メンバーのロック: メンバー操作 (追加、削除など) では、他のファイル操作が同時に起こらないように、自動的にファイルが排他的に割り振られます。同じメンバーに対する入出力操作は実行できませんが、同じファイルの他のメンバーに対する入出力操作は同時に実行できます。

レコード様式データのロック: RCDFMTLCK パラメーター。あるレコード様式に関連するレコードのセット全体 (たとえば、1 つの物理ファイル中のすべてのレコード) をロックしたい場合、 OVRDBF コマンドで RCDFMTLCK パラメーターを使用することができます。

データベース・ロックに関する考慮事項: 表 6 は、最も一般的に使用されるデータベース機能、およびデータベース・ファイルが位置するロックのタイプをいくつかまとめて示したものです。ロックのタイプは、次のページに説明されています。

表 6. データベース機能とロック

機能	コマンド	ファイル・ロック	メンバー/データ・ロック	アクセス・パス・ロック
メンバーの追加	ADDPFM、ADDLFM	*EXCLRD		*EXCLRD
ファイル属性の変更	CHGPF、CHGLF	*EXCL	*EXCLRD	*EXCLRD
メンバー属性の変更	CHGPFM、CHGLFM	*SHRRD	*EXCLRD	
オブジェクト・オーナーの変更	CHGOBJOWN	*EXCL		
オブジェクトの検査	CHKOBJ	*SHRNUPD		
物理ファイル・メンバーの消去	CLRPFM	*SHRRD	*EXCLRD[3]	
重複オブジェクトの作成	CRTDUPOBJ	*EXCL (新規オブジェクト) *SHRNUPD (オブジェクト)		
ファイルの作成	CRTPF、CRTLF、CRTSRCPF	*EXCL		
ファイルの削除	DLTF	*EXCL		*EXCLRD
権限の認可/取り消し	GRTOBJAUT、RVKOBJAUT	*EXCL		
物理ファイル・メンバーの初期設定	INZPFM	*SHRRD	*EXCLRD	
オブジェクトの移動	MOVOBJ	*EXCL		
ファイルのオープン	OPNDBF、OPNQRYF	*SHRRD	*SHRRD	*EXCLRD
アクセス・パスの再作成	EDTRBDAP、OPNDBF	*SHRRD	*SHRRD	*EXCLRD
メンバーの除去	RMVM	*EXCLRD	*EXCL	*EXCLRD
ファイル名の変更	RNMOBJ	*EXCL	*EXCL	*EXCL
メンバー名の変更	RNMM	*EXCLRD	*EXCL	*EXCL

表 6. データベース機能とロック (続き)

機能	コマンド	ファイル・ロック	メンバー/データ・ロック	アクセス・パス・ロック
物理ファイル・メンバーの再編成	RGZPFM	*SHRRD	*EXCL[4]	
ファイルの復元	RSTLIB、RSTOBJ	*EXCL		
ファイルの保管	SAVLIB、SAVOBJ、SAVCHGOBJ	*SHRNUPD[1]	*SHRNUPD[2]	

:

1 活動中保管の場合、ファイル・ロックは最初 *SHRUPD で、それから *SHRRD へ軽減されます。保管コマンドの活動中保管ロックについては、「バックアップおよび回復の手引き」を参照してください。

2 活動中保管の場合は、メンバー/データのロックは *SHRRD です。

3 このプロセスまたは他のプロセスでメンバーがオープンされる場合、消去は行われません。

4 ALWCANCEL(*YES) が指定されている場合、 LOCK キーワードで代わりに *SHRUPD または *EXCLRD ロックを指定できます。

以下の表は、有効なロックの組み合わせを示したものです。

ロック	*EXCL	*EXCLRD	*SHRUPD	*SHRNUPD	*SHRRD
*EXCL[1]					
*EXCLRD[2]					X
*SHRUPD[3]			X		X
*SHRNUPD[4]				X	X
*SHRRD[5]		X	X	X	X

:

1 排他ロック (*EXCL)。オブジェクトは要求するジョブの排他的使用のために割り振られますが、他のジョブはそのオブジェクトを使用できません。

2 排他ロック、読み取り可能 (*EXCLRD)。オブジェクトはそれを要求したジョブに割り振られますが、他のジョブはそのオブジェクトを読み取ることができます。

3 共用ロック、読み取りおよび更新可能 (*SHRUPD)。オブジェクトは、読み取りまたは変更のいずれかの場合に他のジョブとの共用が可能です。

4 共用ロック、読み取り専用 (*SHRNUPD)。オブジェクトは、読み取りの場合に他のジョブとの共用が可能です。

5 共用ロック (*SHRRD)。ジョブがオブジェクトの排他的使用を要求しない場合、オブジェクトは別のジョブとの共用が可能です。

表 7 は、制約が親ファイル (PAR) または従属ファイル (DEP) のどちらと関連付けられているかに対応する、データベース・ファイル制約用のデータベース・ロックを示しています。

表 7. データベース制約ロック： (括弧内の数字は、表の下の注を指します。)

機能タイプ	ファイル・タイプ	ファイル (5)	メンバー (5)	その他のファイル	その他のメンバー
ADDPFM (1)	DEP	*EXCL	*EXCL	*EXCL	*EXCL

表 7. データベース制約ロック (続き): (括弧内の数字は、表の下の注を指します。)

機能タイプ	ファイル・タイプ	ファイル (5)	メンバー (5)	その他のファイル	その他のメンバー
ADDPFM (1)	PAR	*EXCL	*EXCL	*EXCL	*EXCL
ADDPFCST (7)	*REFCST	*EXCL	*EXCL	*EXCL	*EXCL
ADDPFCST (6)	*UNQCST *PRIKEY	*EXCL	*EXCL	*EXCL	*EXCL
ADDPFCST	*UNIQUE *PRIKEY	*EXCL	*EXCL		
RMVM (2)	DEP	*EXCL	*EXCL	*EXCL	*EXCL
RMVM (2)	PAR	*EXCL	*EXCL	*EXCL	*EXCL
DLTF (3)	DEP	*EXCL	*EXCL	*EXCL	*EXCL
DLTF (3)	PAR	*EXCL	*EXCL	*EXCL	*EXCL
RMVPFCST (7)	*REFCST	*EXCL	*EXCL	*EXCL (4)	*EXCL
RMVPFCST (6)	*UNQCST *PRIKEY	*EXCL	*EXCL	*EXCL	*EXCL
RMVPFCST	*UNIQUE *PRIKEY	*EXCL	*EXCL		
CHGPFCST		*EXCL	*EXCL	*SHRRD	*EXCL

注:

1. 物理ファイル・メンバーを追加することで、参照制約が確定する場合。
2. 物理ファイル・メンバーを除去することで、確定済み参照制約が定義済みになる場合。
3. そのファイルについて確定済みまたは定義済みの制約を有する従属ファイルまたは親ファイルを削除する場合。
4. 確定済みまたは定義済みの制約を有する親ファイルに対して物理ファイル制約の除去 (RMVPFCST) コマンドを呼び出す場合は、親ファイルおよび、その親ファイルを基にした論理ファイルは、すべて *EXCL ロックされます。
5. 参照制約の場合、この欄は、従属ファイルまたは従属メンバーに関するものです。
6. 固有制約または基本キー制約は、他のファイルを従属ファイルとして持つ参照制約内の親キーです。
7. その他のファイルは、親ファイルです。

同一ジョブまたは活動化グループ内のデータベース・ファイルの共用

SHARE パラメーター。データベース管理システムの省略時では、多くのユーザーが 1 つのファイルを同時に読み取ったり、変更したりすることができます。また、データベース・ファイルをオープンすることによっても、以下のように同じジョブまたは活動化グループにおけるファイルの共用を行うことができます。

- 同じプログラムにおいて複数回
- 同じジョブまたは活動化グループにおける別個のプログラムにおいて

注: 統合言語環境における共用オープンについて詳しくは、「ILE 概念  」を参照してください。

ファイルの作成、ファイルの変更、およびデータベース・ファイルの一時変更コマンドの SHARE パラメーターを使用すると、ジョブまたは活動化グループで共用 (ファイル、その状況、その位置、およびその記憶域の共用を含む) を行うことができます。ジョブまたは活動化グループでファイルを共用するならば、必要とされる主記憶域の量や、ファイルのオープン/クローズに必要な時間を削減することになり、それによってパフォーマンスを向上させることができます。

SHARE(*YES) パラメーターを使用すれば、同じジョブまたは活動化グループで実行される 2 つ以上のプログラムは、オープン・データ・パス (ODP) を共用できます。オープン・データ・パスとは、ファイルに対するすべての入出力操作に使用されるパスであり、ある意味で、プログラムをファイルに結び付けるものです。SHARE(*YES) パラメーターを指定しない場合は、ファイルのオープン時に毎回、新規のオープン・データ・パスが作成されます。同一ジョブまたは活動化グループにおいて活動ファイルが複数回オープンされる場合は、現在オープンされているファイルの活動 ODP を使用できます。新規のオープン・データ・パスを作成する必要はありません。

これにより、初回のオープン後にファイルをオープンするのに必要な時間や、ジョブまたは活動化グループに必要な主記憶域の量が削減されます。オープン・データ・パスを共用するには、同じファイルの最初のオープンと他のオープンについて、 SHARE(*YES) を指定しなければなりません。十分に設計された (パフォーマンスの点で) アプリケーションは、通常、同じジョブまたは活動化グループの複数のプログラムでオープンされるファイルと、オープン・データ・パスを共用します。

SHARE(*NO) は、ファイルのオープン・データ・パスを共用しないことをシステムに指定します。通常、これはほとんど使用しないファイル、または特定のプログラムで固有の処理を必要とするファイルについてのみ指定します。

注: 高水準言語プログラムでは、ファイルが共用されていないかのようにオープンまたはクローズの操作が実行されます。ファイルの共用の指定は、高水準言語プログラムでは行わずに、 SHARE パラメーターを使用して、ファイルが同じジョブまたは活動化グループで共用されていることを示します。SHARE パラメーターは、データベース・ファイルの作成、変更、または一時変更コマンドだけで指定します。

データベース・ファイルの共用に関するその他の考慮事項については、以下のトピックを参照してください。

- 『ジョブまたは活動化グループで共用されるファイルのオープンの考慮事項』
- 113 ページの『ジョブまたは活動化グループで共用されるファイルの入出力の考慮事項』
- 114 ページの『ジョブまたは活動化グループで共用されるファイルのクローズの考慮事項』

ジョブまたは活動化グループで共用されるファイルのオープンの考慮事項: 同一ジョブまたは活動化グループで共用されるデータベース・ファイルをオープンするときに、次の項目を考慮してください。

- ジョブまたは活動化グループにおいて初めて共用ファイルをオープンするときには、必ずファイルのその後のオープンに必要なすべてのオープン・オプションを指定するようにしてください。共用ファイルのそれ以降のオープンについて指定したオープン・オプションが、共用ファイルの最初のオープンについて指定したオプションと一致しない場合には、エラー・メッセージがプログラムに送られます。(プログラム、あるいは OPNDBF コマンドまたは OPNQRYF コマンドのパラメーターに変更を加えて、矛盾したオプションを取り除くことによって、この状態を訂正することができます。)

  たとえば、PGMA はジョブまたは活動化グループにおいて FILE1 をオープンする最初のプログラムで、PGMA だけがファイルの読み取りを必要とします。ただし、PGMA は PGMB を呼び出し、PGMB は同じ共用ファイルからレコードを削除します。PGMB が共用ファイルからレコードを削除するので、PGMA は、それ自身もレコードを削除するかのようにファイルをオープンする必要があります。高水準言語で正しい指定を行うことによって、これを実現することができます。(高水準言語によっては、これを実現するために、実行されることのないファイル操作ステートメントを使用しなければならないこともあります。詳細については、高水準言語の手引きを参照してください。) ファイル処理オプションは、データベース・ファイルのオープン (OPNDBF) コマンドおよび Query ファイルのオープン (OPNQRYF) コマンドの OPTION パラメーターで指定することもできます。
- ジョブまたは活動化グループ中の共用ファイルは、望ましくない場合もあります。たとえば、1 つのプログラムが到着順でファイルからのレコードを必要とし、別のプログラムがキー順でレコードを必要とする場合です。このような状況では、オープン・データ・パスを共用してはなりません。データベース・ファイルによる一時変更 (OVRDBF) コマンドで SHARE(*NO) を指定して、ジョブまたは活動化グループ中でファイルが共用されないようにすることができます。
- 実行用ライブラリーにある共用ファイルの最初のオープンの後で UPDPROD(*NO) を指定してデバッグ・モードに入ると、そのファイルのそれ以降の共用オープンでは、元のオープン・データ・パスが共

用され、ファイルの変更が許可されます。これを防ぐためには、デバッグされるファイルをオープンする前に、 OVRDBF コマンドで SHARE(*NO) を指定してください。

- 共用ファイルの最初のオープンにコミットメント制御を使用すると、それに続くすべての共用オープンでもコミットメント制御を使用する必要があります。
- キー・フィードバック、挿入キー・フィードバック、または重複キー・フィードバックがそれ以降のファイルの共用オープンに望ましい場合には、これらのフィードバックのタイプのいずれかを全オープンに指定しなければなりません。
- プログラムで、またはデータベース・ファイルによる一時変更 (OVRDBF) コマンドでライブラリー名を指定しない (*LIBL が使用される) 場合には、システムは、*LIBL が指定されている同じ共用ファイルの最後のオープン以降、ライブラリー・リストが変更されていないものとみなします。ライブラリー・リストが変更されている場合には、OVRDBF コマンドにライブラリー名を指定して、正しいファイルがオープンされるようにしなければなりません。
- 全オープンに指定するレコード長は、ファイルの共用オープンでこれより大きいレコード長が指定されている場合でも、それ以降の共用オープンで使用されるレコード長となります。
- 共用ファイルの最初のオープンで指定された一時変更およびプログラム仕様は処理されます。それ以降のオープンで指定された一時変更およびプログラム仕様は、 OVRDBF コマンドの SHARE または LVLCHK パラメーターで指定されたファイル名または値を変更するものを除いて、無視されます。
- OPNQRYF コマンドを使用する最初のオープンで指定する一時変更は、 Query ファイルのオープン・コマンドにより処理されるファイル、ライブラリー、およびメンバーの名前を変更するのに使用できます。データベース・ファイルによる一時変更コマンド (OVRDBF) で指定されたパラメーター値は、TOFILE、MBR、 LVLCHK、および SEQONLY を除き、 OPNQRYF コマンドによって無視されます。
- データベース・ファイルのオープン (OPNDBF) および Query ファイルのオープン (OPNQRYF) コマンドは、次のように、オープン有効範囲 (OPNSCOPE) パラメーターで指定されたレベルに ODP の範囲を限定します。
  - システムは共用オープンを、まず活動化グループで、次にジョブで検索します。
  - 活動化グループに範囲が限定された共用オープンは、活動化グループ間で共用することはできません。
  - ジョブに範囲が限定された共用オープンは、一度にいくつの活動化グループであろうと、ジョブの間ずっと共用できます。

CPF4123 診断メッセージには、全オープンとそれ以降の共用オープンの間に発生する可能性のある不一致が示されます。これらの不一致は共用オープンの失敗の原因とはなりません。

注: Query ファイルのオープン (OPNQRYF) コマンドは、ジョブまたは活動化グループにおいて、既存の共用オープン・データ・パスを決して共用しません。共用 ODP がすでにジョブまたは活動化グループに、 Query ファイルのオープン・コマンドで指定したものと同じファイル、ライブラリー、およびメンバー名で存在する場合は、システムはエラー・メッセージを送信し、 Query ファイルはオープンされません。

ジョブまたは活動化グループで共用されるファイルの入出力の考慮事項: 同じジョブまたは活動化グループで共用されるデータベース・ファイルの処理の際には、次の項目を考慮してください。

- 1 つの共用ファイルに対して、データ・パスは 1 つしか許可されないため、ファイルを共用するジョブまたは活動化グループ内のすべてのプログラムに対して、ただ 1 つのレコード位置が保持されます。プログラムが読み取り操作または読み取り後更新操作を用いてレコードの位置を設定してから、この共用ファイルを使用する別のプログラムを呼び出しているという場合は、呼び出されたプログラムが呼び出しプログラムに戻る時点でレコード位置が移動していたり、レコード・ロックが解除されていたりすることがあります。この場合には、予期しないレコード位置またはロック状態のため、呼び出しプログラ

ムにエラーが起こることがあります。ファイルを共用する場合には、ユーザーが責任を持って、レコード位置やレコード・ロックを設定し直し、レコード位置とレコード・ロック状態を管理します。

- 共用ファイルが最初に更新のためにオープンされる場合、そのファイルを共用するそれ以降のすべてのプログラムがレコード・ロックを要求するとは限りません。ファイルを使用する各プログラムに必要なレコード・ロックのタイプは、システムによって判別されます。システムは、データ保全性を保ちながらロック競合を最小に抑えようとします。

  たとえば、PGMA が、ジョブまたは活動化グループにおいて共用ファイルをオープンする最初のプログラムだとします。PGMA はファイル中のレコードを更新しようとしているので、更新のためにレコードを読み取る時点でレコードをロックします。その後、PGMA は PGMB を呼び出し、 PGMB も共用ファイルを使用します。ただし、PGMB はファイルのレコードを更新せず、単に読み取るだけであるとします。PGMA が最初に共用ファイルを更新可能としてオープンしたとしても、PGMB は、その処理仕様のために、読み取るレコードのロックは行いません。このようにして、レコード・ロック競合を最小限に抑えながら、データ保全性を確保します。

ジョブまたは活動化グループで共用されるファイルのクローズの考慮事項: 同じジョブまたは活動化グループで共用されるデータベース・ファイルのクローズの際には、次の項目を考慮してください。

- クローズ操作の完全な処理 (ファイルおよびメンバーの解放、レコード・ロックの解除、変更の補助記憶装置への強制書き出し、およびオープン・データ・パスの破壊を含む) は、共用オープン・データ・パスをオープンする最後のプログラムがそれをクローズするときにだけ行われます。
- ファイルがデータベース・ファイルのオープン (OPNDBF) コマンドまたは Query ファイルのオープン (OPNQRYF) コマンドでオープンされた場合には、ファイルをクローズするためにファイルのクローズ (CLOF) コマンドを使用してください。次のいずれかが指定された場合には、資源の再利用 (RCLRSC) コマンドを使用して、 Query ファイルのオープン (OPNQRYF) コマンドによってオープンされたファイルをクローズすることができます。
  - OPNSCOPE(*ACTGRPDFN) が指定され、オープンが省略時の活動化グループから要求された場合
  - TYPE(*NORMAL)

  次のいずれかが指定された場合には、資源の再利用 (RCLRSC) コマンドが実行されても、ファイルはオープンしたままです。
  - OPNSCOPE(*ACTGRPDFN) が指定され、オープンが省略時以外の活動化グループから要求された場合
  - OPNSCOPE(*ACTGRP)
  - OPNSCOPE(*JOB)
  - TYPE(*PERM)

同一ジョブ内で共用されるファイルをクローズするときの考慮事項について、以下の例を参照してください。

- 『例 1: 類似した処理オプションを持つ単一セットのファイルの使用』
- 116 ページの『例 2: 類似した処理オプションを持つ複数セットのファイルの使用』
- 117 ページの『例 3: 異なる処理要件を持つ単一セットのファイルの使用』

例 1: 類似した処理オプションを持つ単一セットのファイルの使用: この例では、ユーザーがサインオンし、使用されるプログラムの大半が同じファイルのセットを処理します。

CL プログラム (PGMA) が最初のプログラムとして使用されます (アプリケーションをセットアップして、一時変更を組み込み、共用ファイルをオープンするため)。その後、PGMA は PGMB に制御権を渡し、PGMB はアプリケーション・メニューを表示します。この例では、ファイル A、B および C が使用

され、ファイル A と B が共用されるものとします。ファイル A とファイル B は SHARE(*NO) を指定して作成されているので、 SHARE(*YES) オプションを指定するために、おのおのの OPNDBF コマンドの前に OVRDBF コマンドを実行しなければなりません。ファイル C は SHARE(*NO) を指定して作成されていますが、この例ではファイル C は共用されません。

```
PGMA:   PGM      /* PGMA - Initial program  */
        OVRDBF   FILE(A) SHARE(*YES)
        OVRDBF   FILE(B) SHARE(*YES)
        OPNDBF   FILE(A) OPTION(*ALL) ....
        OPNDBF   FILE(B) OPTION(*INP) ...
        TFRCTL   PGMB
        ENDPGM



PGMB:   PGM      /* PGMB - Menu program */
        DCLF     FILE(DISPLAY)
BEGIN:  SNDRCVF  RCDFMT(MENU)
        IF       (&RESPONSE *EQ '1') CALL PGM11
        IF       (&RESPONSE *EQ '2') CALL PGM12
        .
        .
        IF       (&RESPONSE *EQ '90') SIGNOFF
        GOTO     BEGIN
        ENDPGM
```

PGMA でオープンされるファイルは、ジョブに、あるいは同じ活動化グループで実行される PGMA、PGM11、および PGM12 に範囲が限定され、ファイルのオープンはその活動化グループに範囲が限定されます。

この例では、以下を想定します。

- PGM11 は、ファイル A と B をオープンします。これらのファイルは PGMA の中で OPNDBF コマンドによって共用ファイルとしてオープンされているので、オープンの時間が短縮されます。共用オープン・データ・パスがクローズされるときには、クローズの時間も短縮されます。データベース・ファイルによる一時変更 (OVRDBF) コマンドは、制御権が (PGMA の中で制御の転送 [TFRCTL] コマンドによって) PGMB に渡される場合にもそのまま効力を保ちます。
- PGM12 は、ファイル A、B、および C をオープンします。ファイル A と B は共用ファイルとしてすでにオープンされているので、オープンの時間が短縮されます。ファイル C はこのプログラムだけで使用されるので、共用ファイルとしてはオープンされません。

この例では、必要なファイルは 1 セットだけなので、ファイルのクローズ (CLOF) コマンドは使用されていません。操作員がサインオフすると、ファイルは自動的にクローズされます。PGMA (初期プログラム) はジョブの開始時だけに呼び出されるものとします。統合言語環境でのリソースのレクラメーション処理について詳しくは、「ILE 概念 」を参照してください。

注: PGMB の表示装置ファイル (DISPLAY) も共用ファイルとして指定できます。その場合は、後で表示装置ファイルを使用するどのプログラムでも、このファイルをオープンする際のパフォーマンスが向上します。

例 1 では、OPNDBF コマンドは別個のプログラム (PGMA) に入っているので、ジョブ内の他の処理プログラムは可能な限り効率よく実行されます。すなわち、ジョブ内の他のプログラムで使用される重要なファイルは、 PGMA でオープンされます。PGMA によりファイルがオープンされた後、主処理プログラム (PGMB、PGM11、および PGM12) でこれらのファイルを共用できるため、オープンおよびクローズ要求の処理がより速くなります。さらに、データベース・ファイルのオープン (OPNDBF) コマンドを PGMB ではなく PGMA に置くことによって、PGMB のために使用される主記憶域の量が少なくてすみます。

一時変更およびオープンはすべて初期プログラム (PGMA) の中で指定することができます。その後、このプログラムをジョブから取り除く (たとえば、ジョブの外へ転送する) ことができます。しかし、プログラムがファイルをオープンした時点でこのプログラムが作成したオープン・データ・パスはそのまま残り、ジョブ内の他のプログラムが使用することができます。

OPNDBF コマンドとの関係における OVRDBF コマンドの取り扱いに注意してください。一時変更は、ファイルがオープンされる前に指定しなければなりません。OVRDBF コマンドのパラメーターのいくつかは、OPNDBF コマンドにもあります。矛盾が生じると、OVRDBF の値が使用されます。統合言語環境で一時変更がいつ有効になるかについて詳しくは、「ILE 概念」を参照してください。

例 2: 類似した処理オプションを持つ複数セットのファイルの使用: 必要なファイルをオープンするためにデータベース・ファイルのオープン (OPNDBF) コマンドを使用するアプリケーション・プログラム (たとえば、売掛管理プログラムや買掛管理プログラム) を指定するように、メニューで操作員に要求するとします。アプリケーションが終了すると、ファイルのクローズ (CLOF) コマンドによりファイルがクローズされます。CLOF コマンドは、ジョブに必要とされる主記憶域の量を減らすために使用されています。この例では、アプリケーションごとに異なるファイルが使用されています。ユーザーは、通常、新しいアプリケーション・プログラムを選択する前に、相当に長い時間、1 つのアプリケーションで処理を行います。

売掛管理プログラムの例を以下に示します。

```
PGMC:   PGM      /* PGMC PROGRAM */
        DCLF     FILE(DISPLAY)
BEGIN:  SNDRCVF  RCDFMT(TOPMENU)
        IF       (&RESPONSE *EQ '1') CALL ACCRECV
        IF       (&RESPONSE *EQ '2') CALL ACCPAY
        .
        .
        IF       (&RESPONSE *EQ '90') SIGNOFF
        GOTO     BEGIN
        ENDPGM



ACCREC: PGM      /* ACCREC PROGRAM */
        DCLF     FILE(DISPLAY)
        OVRDBF   FILE(A) SHARE(*YES)
        OVRDBF   FILE(B) SHARE(*YES)
        OPNDBF   FILE(A) OPTION(*ALL) ....
        OPNDBF   FILE(B) OPTIONS(*INP) ...
BEGIN:  SNDRCVF  RCDFMT(ACCRMENU)
        IF       (&RESPONSE *EQ '1') CALL PGM21
        IF       (&RESPONSE *EQ '2') CALL PGM22
        .
        .
        IF       (&RESPONSE *EQ '88') DO /* Return */
                 CLOF FILE(A)
                 CLOF FILE(B)
                 RETURN
               ENDDO
        GOTO     BEGIN
        ENDPGM
```

買掛管理メニュー用のプログラムもこれと似ていますが、異なるセットの OPNDBF コマンドと CLOF コマンドを使用します。

この例では、ファイル A と B は SHARE(*NO) を指定して作成されています。したがって、OPNDBF コマンドの前には OVRDBF コマンドがなければなりません。例 1 の場合と同様に、OPNDBF コマンドを別個のプログラムに入れ、それを呼び出すことによって、各ジョブによって使用される主記憶域の量を減らす

ことができます。CLOF コマンド用にも別個のプログラムを作成することができます。OPNDBF コマンドは、メニューから呼び出され、特定のアプリケーション・プログラムに制御権を移動するアプリケーション・セットアップ・プログラムの中に入れることができます (このセットアップ・プログラムで指定された一時変更はすべて保持されます)。しかし、これらの機能のために別個のプログラムを呼び出す場合でもシステム資源が使用されるので、各種のメニューが使用される頻度によっては、この例のように、各アプリケーション・プログラム・メニューに OPNDBF および CLOF コマンドを組み込む方が良いこともあります。

ファイルのクローズ (CLOF) コマンドを使用する代わりに、 PGMC (セットアップ・プログラム) の中で資源の再利用 (RCLRSC) コマンドを使用することもできます。RCLRSC コマンドは、すべてのファイルをクローズし、さらに、呼び出された後に呼び出し側のプログラムに戻されたすべてのファイルおよびプログラムに関連する残りの記憶域をすべて解放します。ただし、RCLRSC では、データベース・ファイルのオープン (OPNDBF) コマンドまたは Query ファイルのオープン (OPNQRYF) コマンドで次のものを指定してオープンされたファイルはクローズされません 。

- OPNSCOPE(*ACTGRPDFN) が指定され、オープンが省略時以外の活動化グループから要求された場合。
- オープンの活動化グループ番号よりも低い番号を持つ活動化グループから RCLRSC コマンドが出された場合は、OPNSCOPE(*ACTGRP) が再利用されます。
- OPNSCOPE(*JOB)
- TYPE(*PERM)

以下の例は、ファイルのクローズに使用される RCLRSC コマンドを示しています。

```
 .
 .
IF              (&RESPONSE *EQ '1') DO
                CALL ACCRECV
                RCLRSC
                ENDDO
IF              (&RESPONSE *EQ '2') DO
                CALL ACCPAY
                RCLRSC
                ENDDO
 .
 .
```

例 3: 異なる処理要件を持つ単一セットのファイルの使用: いくつかのプログラムが読み取り専用のファイル処理を必要とし、他のプログラムがいくつかのオプションまたはすべてのオプション (入力/更新/追加/削除) を必要としている場合には、以下の方法のうち 1 つを使用できます。ファイルの処理にある種のコマンド・パラメーターを使用するプログラムと使用しないプログラムとがある場合 (たとえば、時に応じてコミット・オプションを使用する必要がある場合) にも、これらの方法が適用されます。

単一のデータベース・ファイルのオープン (OPNDBF) コマンドを使用して OPTION(*ALL) を指定することができ、これを行うと、オープン・データ・パスが共用としてオープンされます (たとえば、前の OVRDBF コマンドを使用して SHARE(*YES) を指定した場合)。その後で、各プログラムはオプションのサブセットをオープンできます。プログラムは、プログラム内での指定に応じて、オープンのタイプを要求します。入力専用のオープンを指定しているプログラムが、共用オープンを行わなかったかのように動作する (たとえば、レコードの読み取り時に追加のレコード・ロックが行われない) ため、それ以上考慮する必要がない場合もあります。

しかし、OPNDBF コマンドで指定したオプションが、プログラムの操作方法に影響する場合もあります。たとえば、プログラム内でファイルのオープン・コマンドに SEQONLY(*NO) を指定した場合などです。OPNDBF コマンドで SEQONLY(*YES) を指定し、順次のみの処理にとっては無効な操作をプログラムが試みると、エラーが起こります。

ACCPTH パラメーターも、プログラムがアクセス・パスを使用する方法 (到着順またはキー順) と一貫したものでなければなりません。

データベース・ファイルのオープン (OPNDBF) コマンドで COMMIT(*YES) を指定し、コミットメント制御の開始 (STRCMTCTL) コマンドで LCKLVL(*ALL) または LCKLVL(*CS) を指定した場合には、レコードの読み取り操作があると、そのレコードがロックされます (コミットメント制御のレコード・ロック規則により)。これは、予期しないレコード・ロックおよびプログラム内でのエラーの原因となることがあります。

同じデータに対して 2 つの OPNDBF コマンドを使用する (たとえば、一方に OPTION(*ALL) を指定し、他方に OPTION(*INP) を指定する) ことができます。この 2 番目のコマンドの指定は、同じ物理ファイル (複数も可) を指す論理ファイルでなければなりません。この論理ファイルは、SHARE(*YES) としてオープンでき、同じジョブの間で複数回使用できます。

データベース・ファイルの順次のみの処理

SEQONLY パラメーターおよび NBRRCDS パラメーター。プログラムが入力のみまたは出力のみのためにデータベース・ファイルを順次に処理する場合には、データベース・ファイルによる一時変更 (OVRDBF) またはデータベース・ファイルのオープン (OPNDBF) コマンドで順次のみの処理 (SEQONLY) パラメーターを使用することによって、パフォーマンスを向上させることができます。SEQONLY 処理を使用するには、ファイルが入力専用または出力専用としてオープンされていなければなりません。NBRRCDS パラメーターは、オープン・オプションのどのような組み合わせとも使用できます。(Query ファイルのオープン [OPNQRYF] コマンドでは、可能な場合には常に順次のみの処理が行われます。) 高水準言語の仕様によっては、高水準言語でも省略時として順次のみの処理が使用されることがあります。たとえば、ファイルを入力専用にオープンし、高水準言語プログラムに指定されている唯一のファイル操作が順次読み取り操作である場合には、高水準言語は自動的に順次のみの処理を要求します。

注: ファイルの位置付け操作は順次読み取り操作とはみなされないので、位置付け操作を含む高水準言語プログラムは、順次のみの処理を自動的には要求しません 。(ファイルの位置付け操作の例としては、RPG/400 の SETLL 操作や COBOL/400 の START 操作があります。) 高水準言語プログラムが順次のみの処理を自動的に要求できなくても、 OVRDBF コマンドの SEQONLY パラメーターを使用して要求できます。

順次のみの処理を指定する場合には、システムのデータベース主記憶域とジョブの内部データ主記憶域との間で 1 単位として転送されるレコードの数を指定することもできます。順次のみの処理で転送するレコードの数を指定しない場合には、システムが、4096 バイト・バッファーに見合うレコードの数に基づいて、レコードの数を計算します。

さらに、システムでは、補助記憶装置と主記憶域との間で 1 単位として転送されるレコードの数を制御する方法も用意されています。データが物理的に保管されている順序でデータを読み取る場合には、OVRDBF コマンドで NBRRCDS パラメーターを使用することによって、ジョブのパフォーマンスを向上させることができます。

注: 物理データがアクセス・パスと同じ順番でない限り、順次のみの処理をキー順アクセス・パス・ファイルで使用するべきではありません。SEQONLY(*YES) 処理は物理データがアクセス・パスの順番に再編成されるまで、アプリケーションのパフォーマンスを低下させる原因となる場合があります。

順次のみの処理を行うときの考慮事項について、以下のトピックを参照してください。

- 119 ページの『順次のみの処理でのオープンに関する考慮事項』
- 120 ページの『順次のみの処理での入出力に関する考慮事項』

- 121 ページの『順次のみの処理でのクローズに関する考慮事項』

順次のみの処理でのオープンに関する考慮事項: 以下の考慮事項は、順次のみの処理を指定した場合のファイルのオープンに関するものです。システムが順次のみの処理を行うことができないと判定した場合には、順次のみの処理の要求は受け入れられないことを示すメッセージがプログラムに送られますが、その場合でも、ファイルは処理に備えてオープンされます。

- プログラムが出力専用でメンバーをオープンし、SEQONLY(*YES) を指定しており (レコード数は指定していない) 、しかもオープンされるメンバーが論理メンバーまたは固有キーを持つ物理メンバーであるか、あるいは物理メンバーへのアクセス・パスが他にある場合には、 SEQONLY(*YES) は SEQONLY(*NO) に変更されるので、出力操作時に、起こり得るエラー (たとえば、重複キー、変換マッピング・エラー、選択/除外エラー) をプログラムが処理できるようになります。システムに順次のみの処理を実行させたい場合には、*YES の値とレコード数の指定の両方を含むように SEQONLY パラメーターを変更してください。
- 順次のみの処理は、入力専用 (読み取り) または出力専用 (追加) の操作についてのみ指定することができます。プログラムで更新または削除の操作を指定している場合には、システムは順次のみの処理を認めません。
- ファイル出力用にオープンする場合には、そのファイルは、物理ファイルであるか、または 1 つの物理ファイル・メンバーが基礎になっている論理ファイルでなければなりません。
- メンバーが出力専用にオープンされている場合しか、順次のみの処理をコミットメント制御とともに指定することはできません。
- 順次のみの処理が、コミットメント制御を用いてオープンされたファイルについて使用されている場合に、ジョブのロールバック操作が実行されると、ロールバック操作時にジョブの記憶域内にあるレコードは、システム記憶域に書き込まれず、コミットメント制御トランザクションのジャーナルに反映されません。ロールバック操作が特定のコミットメント制御トランザクションに対して実行される前に、システム記憶域にレコードが書き込まれていない場合は、コミットメント制御トランザクション全体がジャーナルに反映されません。
- 出力専用の場合には、1 単位として移動するよう指定したレコード数と強制書き出し率が比較され、必要に応じて自動的に調整されます。レコード数が強制書き出し頻度よりも大きい場合は、レコード数が強制書き出し頻度と等しくなるように減らされます。逆の場合には、強制書き出し頻度がレコード数と等しくなるように削減されます。
- メンバーが出力専用としてプログラムによってオープンされ、 SEQONLY(*YES) が指定され (レコード数は指定されていない) 、しかも重複キー・フィードバックまたは挿入キー・フィードバックが要求されている場合には、レコードがファイルに挿入される時点でレコードごとのフィードバックが提供されるように、SEQONLY(*YES) は SEQONLY(*NO) に変更されます。
- 以下の条件をすべて満たすならば、1 ブロック内のレコード数は 1 に変更されます。
  - メンバーが出力専用のためにオープンされた。
  - 順次のみの処理を指定した有効な一時変更操作がない。
  - オープンされているファイルは、レコードの増分数がゼロにセットされたため、拡張できないファイルである。
  - ファイル内の使用できるバイト数が、1 ブロックのレコードに収まるバイト数より小さい。

順次のみの処理を指定せず、Query ファイルのオープン (OPNQRYF) コマンドを使用してファイルをオープンする場合には、下記の事項を考慮しなければなりません。これらの条件が満たされる場合には、順次のみの処理が行われることを示すメッセージが送られ、Query ファイルがオープンされます。

- OPNQRYF コマンドのグループ・フィールド (GRPFLD) パラメーターに 1 つまたは複数のフィールド名を指定している場合、あるいは OPNQRYF でグループ処理を要求している場合。

- OPNQRYF コマンドの UNIQUEKEY パラメーターに 1 つまたは複数のフィールド、あるいは *ALL を指定している場合。
- SQL SELECT ステートメントの DISTINCT オプションでビューを使用する場合。この場合は、SEQONLY(*YES) 処理が自動的に実行されます。

OPNQRYF コマンドの詳細については、 126 ページの『Query ファイルのオープン (OPNQRYF) コマンドの使用』を参照してください。

順次のみの処理での入出力に関する考慮事項: 以下の考慮事項は、順次のみの処理を指定した場合のファイルに対する入出力操作に関するものです。

- 入力については、プログラムは入力バッファーから一度に 1 つのレコードを受け取ります。入力バッファーのすべてのレコードが処理されると、システムは自動的に次の 1 組のレコードを読み取ります。

  注: レコードが入力バッファーに読み取られた後の変更は、入力バッファーには反映されません。
- 出力については、プログラムは一度に 1 つのレコードを出力バッファーに転送しなければなりません。出力バッファーがフルになると、システムは自動的にレコードをデータベースに追加します。

  注: ジャーナルを使用している場合には、すべての項目が論理的に一緒に起こったかのように、バッファー全体が一度にジャーナルに書き込まれます。このジャーナル処理は、レコードがデータベースに追加される前に行われます。

  出力について順次のみの処理を使用する場合には、ファイルに対するすべての変更をそれが起こった時点で見ることができない場合があります。たとえば、PGMA が使用するファイルに対して順次のみの処理が指定されていて、 PGMA がファイルに新しいレコードを追加するとします。さらに、バッファー内のレコードの数として SEQONLY パラメーターで 5 を指定していたとすれば、新しく追加されたレコードはバッファーがフルになった時点で初めてデータベースに転送されます。この例では、5 番目のレコードが追加されてから、初めて最初の 5 つのレコードがデータベースに転送され、システム内の他のジョブでの処理に使用されることができるようになります。

  さらに、出力について順次のみの処理を使用するときには、レコードがバッファーからデータベースへ移される時点でエラーが発生した場合、それを処理しない限り、レコードがデータベースに追加されないことがあります。たとえば、バッファーに 5 つのレコードが入っていて、バッファー内の 3 番目のレコードがファイル内の別のレコードと重複するキーを持っており、しかも、ファイルが固有キー・ファイルとして定義されているとします。この場合、システムがバッファーの内容をデータベースに転送すると、最初の 2 つのレコードはデータベースに追加されますが、 3 番目のレコードで重複キー・エラーが起こります。このエラーのため、バッファー内の 3 番目、4 番目、および 5 番目のレコードはデータベースに追加されません 。
- 出力操作にデータ強制終了機能を使用して、バッファー内のすべてのレコードをデータベースに強制書き出しをすることができます (ただし、上の例のように、固有キーを持つものとして定義されているファイルの中で重複キーの原因となったレコードは除かれます)。データ強制終了機能は、ある種の高水準言語でしか使用することができません。
- 以下の条件をすべて満たすならば、1 ブロック内のレコード数は 1 に変更されます。
  - メンバーが出力専用処理または順次のみの処理のためにオープンされた。
  - 順次のみの処理を指定した有効な一時変更操作がない。
  - オープンされているファイルは、レコードの増分数がゼロにセットされたため、拡張されている。
  - ファイル内の使用できるバイト数が、1 ブロックのレコードに収まるバイト数より小さい。

順次のみの処理でのクローズに関する考慮事項: 順次のみの処理が指定されているファイルがクローズされると、出力バッファー内にまだ残っているレコードがすべてデータベースに追加されます。しかし、あるレコードでエラーが発生すると、そのレコードに続くすべてのレコードはデータベースに追加されません。

同じジョブ内の複数のプログラムが 1 つの順次のみの出力ファイルを共用している場合には、出力バッファーは最後のクローズが起こるまで NULL になりません。したがって、クローズが起こっても (ジョブでの最後のクローズの場合を除いて) まだバッファー内にあるレコードは、このジョブあるいは別のジョブのデータベースに書き出されません。

データベース・ファイル処理実行時の考慮事項の要約

以下の表は、プログラムによるデータベース・ファイル・メンバーの使用を制御するパラメーターのリストであり、これらのパラメーターをどこで指定できるかを示しています。2 か所以上で指定できるパラメーターの場合には、システムによって値が組み合わされます。データベース・ファイルによる一時変更 (OPNDBF) コマンドのパラメーターはプログラムのパラメーターに優先し、データベース・ファイルのオープン (OPNDBF) または Query ファイルのオープン (OPNQRYF) コマンドのパラメーターはファイルの作成または変更パラメーターに優先します。

注: TOFILE、MBR、LVLCHK、SEQONLY、SHARE、WAITRCD、および INHWRT 以外の一時変更パラメーターは、 OPNQRYF コマンドではいずれも無視されます。

以下は、制御言語 (CL) コマンドで指定するデータベース処理オプションの表です。

表 8. CL コマンドで指定するデータベース処理オプション

		コマンド				
説明	**パラメーター**	**CRTPF、CRTLF**	**CHGPF、CHGLF**	**OPNDBF**	**OPNQRYF**	**OVRDBF**
ファイル名	FILE	X	X[1]	X	X	X
ライブラリー名		X	X[2]	X	X	X
メンバー名	MBR	X		X	X	X
メンバー処理オプション	OPTION			X	X	
レコード様式のロック状態	RCDFMTLCK					X
オープン後の開始ファイル位置	POSITION					X
プログラムによる順次処理のみの実行	SEQONLY			X	X	X
キー順アクセス・パスの無視	ACCPTH			X		
ファイル・ロックの待機時間	WAITFILE	X	X			X

表 8. CL コマンドで指定するデータベース処理オプション (続き)

		コマンド				
説明	**パラメーター**	**CRTPF、CRTLF**	**CHGPF、CHGLF**	**OPNDBF**	**OPNQRYF**	**OVRDBF**
レコード・ロックの待機時間	WAITRCD	X	X			X
一時変更の防止	SECURE					X
補助記憶装置から主記憶域へ転送されるレコードの数	NBRRCDS					X
他のプログラムとのオープン・データ・パスの共用	SHARE	X	X			X
様式選択プログラム	FMTSLR	X[3]	X[3]			X
強制書き出し頻度	FRCRATIO	X	X			X
書き込み禁止	INHWRT					X
レコード様式のレベル検査	LVLCHK	X	X			X
満了日の検査	EXPCHK					X
満了日	EXPDATE	X[4]	X[4]			X
アクセス・パスの強制書き出し	FRCACCPTH	X	X			
コミットメント制御	COMMIT			X	X	
ファイル終わり遅延	EOFDLY					X
重複キーの検査	DUPKEYCHK			X	X	
削除済みレコードのスペースの再使用	REUSEDLT	X[4]	X[4]			
コード化文字セット識別子	CCSID	X[4]	X[4]			
分類順序	SRTSEQ	X	X		X	
言語識別コード	LANGID	X	X		X	

表 8. CL コマンドで指定するデータベース処理オプション (続き)

説明	パラメーター	コマンド				
		CRTPF、CRTLF	CHGPF、CHGLF	OPNDBF	OPNQRYF	OVRDBF
注:						
[1]	ファイル名: CHGPF コマンドおよび CHGLF コマンドではファイル名は識別のためだけに使用されます。ファイル名を変更することはできません。					
[2]	ライブラリー名: CHGPF コマンドおよび CHGLF コマンドではライブラリー名は識別のためだけに使用されます。ライブラリー名を変更することはできません。					
[3]	様式選択プログラム: CRTLF コマンドおよび CHGLF コマンドだけで使用されます。					
[4]	満了日、削除済みレコードのスペースの再使用、およびコード化文字セット識別子: CRTPF コマンドおよび CHGPF コマンドだけで使用されます。					

以下は、プログラムで指定するデータベース処理オプションの表です。

表 9. プログラムで指定するデータベース処理オプション

説明	RPG/400 言語	COBOL/400 言語	iSeries BASIC	iSeries PL/I	iSeries Pascal
ファイル名	X	X	X	X	X
ライブラリー名			X	X	X
メンバー名			X	X	X
プログラム・レコード長	X	X	X	X	X
メンバー処理オプション	X	X	X	X	X
レコード様式のロック状態			X	X	
プログラムが使用するレコード様式	X		X		
レコードの物理ファイル・メンバーの消去		X	X		X
プログラムによる順次処理のみの実行	X	X		X	X
キー順アクセス・パスの無視	X	X	X	X	X
他のプログラムとのオープン・データ・パスの共用				X	X
レコード様式のレベル検査	X	X	X	X	
コミットメント制御	X	X		X	
重複キーの検査		X			

表 9. プログラムで指定するデータベース処理オプション (続き)

説明	RPG/400 言語	COBOL/400 言語	iSeries BASIC	iSeries PL/I	iSeries Pascal
: 制御言語 (CL) プログラムでも、これらのパラメーターの多くを指定することができます。CL コマンドで指定できるデータベース処理オプションの詳細については、 121 ページの表 8を参照してください。					

データベースのパフォーマンスに与える記憶域プールのページング・オプションの影響

共用プールのページング・オプションは、データベース・ファイルの読み取りと変更のパフォーマンスに大きな影響を与えます。

- ページング・オプション *FIXED を指定すれば、プログラムにより使用されるメモリーの量が最小限になります。これは、プログラムが次のことを行うためです。
  - 補助記憶装置から主メモリーへ、より小さなブロックでデータを転送する。
  - ファイルの変更 (既存のレコードへの更新や新たに追加されたレコード) を頻繁に補助記憶装置に書き出す。

  このオプションにより、システムは、ページング・オプションが追加される前とほぼ同じことを実行できます。
- ページング・オプション *CALC を指定すれば、プログラムがデータベース・ファイルの読み取りや更新を行うときの実行方法を改善することができます。共用プール内で十分なメモリーが使用可能である場合には、プログラムは次のことを行うことができます。
  - 補助記憶装置からメモリーへ、より大きなデータ・ブロックを転送する。
  - 変更されたデータをより低い頻度で補助記憶装置に書き出す。

  データベース・ファイルに対して行われるページング操作は、ファイルの使用法とメモリーの使用可能度に基づいて、動的に変化します。頻繁に参照されるファイルは、アクセスされることが少ないファイルより、常駐する可能性が高くなります。メモリーは、一般的なデータのためのキャッシュのように使用されます。入出力操作の総数は、*CALC ページング・オプションを使用して削減することができます。

ページング・オプションについて詳しくは、 Information Center のパフォーマンス を参照してください。

データベース・ファイルのオープン

この章では、データベース・ファイルのオープンについて説明します。さらに、データベース・ファイルのオープン (OPNDBF) および Query ファイルのオープン (OPNQRYF) の CL コマンドについても説明します。以下のトピックを参照してください。

- 『データベース・ファイル・メンバーのオープン』
- 125 ページの『データベース・ファイルのオープン (OPNDBF) コマンドの使用』
- 126 ページの『Query ファイルのオープン (OPNQRYF) コマンドの使用』

データベース・ファイル・メンバーのオープン

プログラムの中でデータベース・ファイルを使用するためには、プログラムがデータベース・ファイルに対してオープン命令を出さなければなりません。プログラミング言語によってはオープン操作を指定しない場合に、ファイルを自動的にオープンするものもあります。プログラムで、またはデータベース・ファイルによる一時変更 (OVRDBF) コマンドでメンバー名を指定しない場合には、ファイルの最初のメンバー (作成の日時によって定義される) が使用されます。

メンバー名を指定する場合は、正しいファイル名を持つがそのメンバー名を含まないファイルは無視されます。FILEA という名前のデータベース・ファイルが、別々のライブラリーに複数ある場合は、オープンさ

れるメンバーは、ライブラリー・リストの中で要求に適合する最初のものです。たとえば、LIB1、LIB2、および LIB3 がユーザーのライブラリー・リストにあり、すべてに FILEA という名前のファイルが入っているとします。LIB3 の FILEA だけに、オープンされる MBRA という名前のメンバーがあります。LIB3 の FILEA のメンバー MRBA がオープンされ、他の FILEA は無視されます。

メンバーを見つけ出した後、システムはプログラムをデータベース・ファイルに接続します。これによって、プログラムはファイルに対する入出力操作を実行できます。高水準言語プログラムでのファイルのオープンの詳細については、該当する高水準言語の手引きを参照してください。

高水準言語プログラムでは、ステートメントを使用してデータベース・ファイルをオープンできます。また、データベース・ファイルのオープン (OPNDBF) および Query ファイルのオープン (OPNQRYF) という CL コマンドを使用することもできます。OPNDBF コマンドは、ジョブ内の初期プログラムの中で共用ファイルをオープンするのに効果的です。OPNQRYF コマンドは、プログラムの外でレコードの選択および編成を行う上で非常に効果的です。この場合には、プログラムは OPNQRYF コマンドによって与えられた情報を使用して、必要なデータだけを処理できます。 制御言語 (CL) のトピックに含まれる、 OPNDF (データベース・ファイルのオープン) コマンドおよび OPNQRYF (Query ファイルのオープン) コマンドを参照してください。

データベース・ファイルのオープン (OPNDBF) コマンドの使用

通常、OPNDBF コマンドを使用する場合には、コマンド・パラメーターの省略時値を使用できます。下記のパラメーターについては、省略時値を使用する代わりに、特定の値を指定することが必要になる場合があります。

OPTION パラメーター。アプリケーション・プログラムで入力専用の処理 (レコードを更新せず、読み取るだけ) を使用する場合には、*INP オプションを指定してください。このようにすると、システムは、各レコードを更新に備えてロックしようとすることなく、レコードを読み取ることができます。アプリケーション・プログラムが出力専用の処理 (既存のレコードの読み取りまたは更新を行わずに、レコードをファイルに書き込む) を使用する場合には、 *OUT オプションを指定してください。

注: プログラムがアクティブ・レコードに対する直接出力操作 (相対レコード番号による更新) を行う場合には、*OUT の代わりに *ALL を指定しなければなりません。プログラムが削除済みレコードに対して直接出力操作を行う場合には、 *OUT を指定しなければなりません。

MBR パラメーター。ファイルの最初のメンバー以外のメンバーをオープンしたい場合には、オープンしたいメンバーの名前を指定するか、あるいはデータベース・ファイルのオープン (OPNDBF) コマンドの前にデータベース・ファイルによる一時変更 (OVRDBF) コマンドを指定しなければなりません。

注: 後続のプログラムの中でオープンするメンバー (最初のメンバー以外) を使用するためには、OVRDBF コマンドでメンバー名を指定しなければなりません。

OPNID パラメーター。ファイル名以外の識別コードを使用する場合には、その識別コードを指定しなければなりません。このオープン識別コードは、ファイルを処理するための他の CL コマンドで使用されることがあります。たとえば、ファイルのクローズ (CLOF) コマンドでは、クローズするファイルを示すためにこの識別コードを使用します。

ACCPTH パラメーター。ファイルがキー順アクセス・パスを持ち、しかも (1) オープン・オプションが *OUT、または (2) オープン・オプションが *INP または *ALL であるがプログラムがキー順アクセス・パスを使用していない、のいずれかの場合には、OPNDBF パラメーターで ACCPTH(*ARRIVAL) を指定できます。キー順アクセス・パスを無視すると、ジョブのパフォーマンスが向上することもあります。

SEQONLY パラメーター。後続のアプリケーション・プログラムがファイルを順次に処理する場合には、*YES を指定してください。システム・データ・バッファーとプログラム・データ・バッファーとの間で転送されるレコードの数を指定する際にも、このパラメーターを使用できます。データベース・ファイルのオープン (OPNDBF) コマンドに OPTION(*INP) または OPTION(*OUT) を指定していない場合には、SEQONLY(*YES) を指定することはできません。物理データがアクセス・パスと同じ順番でない限り、順次のみの処理をキー順アクセス・パス・ファイルで使用するべきではありません。

COMMIT パラメーター。アプリケーション・プログラムがコミットメント制御を使用する場合には、*YES を指定してください。*YES を指定した場合には、コミットメント制御環境 (コミットメント制御の開始 [STRCMTCTL] コマンドが処理された) で実行していないと、 OPNDBF コマンドは正常に実行されません。アプリケーション・プログラムがコミットメント制御を使用しない場合には、省略時の *NO を使用してください。

OPNSCOPE パラメーター。オープン・データ・パス (ODP) の有効範囲を指定します。要求が省略時の活動化グループから出された場合は、*ACTGRPDFN を指定してください。すると、ODP はコマンドを発行したプログラムの呼び出しレベルに範囲限定されます。要求が他の活動化グループから出された場合は、ODP はその活動化グループに範囲限定されます。ODP がコマンドを発行するプログラムの活動化グループに範囲限定される場合は、*ACTGRP を指定してください。ODP がジョブに範囲限定される場合は、*JOB を指定してください。このパラメーターと TYPE パラメーターを指定すると、エラー・メッセージを受け取ります。

DUPKEYCHK パラメーター。重複キー・フィードバックが必要かどうかを指定してください。*YES を指定した場合には、入出力操作で重複キー・フィードバックが返されます。*NO を指定した場合には、入出力操作で重複キー・フィードバックは返されません。アプリケーション・プログラムが COBOL/400 または C/400 言語で作成されていない場合、あるいは COBOL または C プログラムで、返される重複キー・フィードバック情報を使用しない場合には、省略時値 (*NO) を使用してください。

TYPE パラメーター。アプリケーション・プログラムにおいて監視されない例外が生じた場合に必要な処置を指定してください。*NORMAL を指定すると、次のいずれかが生じる可能性があります。

- ユーザーのプログラムが資源の再利用 (RCLRSC) コマンドを発行して、 RCLRSC コマンドを発行しているプログラムよりも高いレベル (呼び出しスタックにおいて) でオープンされているファイルをクローズする。
- ユーザーの使用している高水準言語がクローズ操作を実行する。

ファイルをもう一度オープンしないでアプリケーションを続行したい場合には、 *PERM を指定してください。TYPE(*NORMAL) を指定すると、次の両方が生じた場合にファイルがクローズされます。

- ユーザーのプログラムがエラー・メッセージを受け取る。
- ファイルが呼び出しスタック内でより高いレベルでオープンされている。

TYPE(*PERM) を指定すれば、エラー・メッセージを受け取っても、ファイルをオープンしたままにしておくことができます。このパラメーターは、OPNSCOPE パラメーターを指定した場合は、指定しないでください。

Query ファイルのオープン (OPNQRYF) コマンドの使用

Query ファイルのオープン (OPNQRYF) コマンドは、データベース・ファイルに対して多くのデータ処理機能を実行することのできる CL コマンドです。本質的には、OPNQRYF コマンドは処理プログラムとデータベース・レコードとの間のフィルターとして働きます。データベース・ファイルは、物理ファイルまたは論理ファイルのどちらでもかまいません。物理ファイルの作成 (CRTPF) コマンドまたは論理ファイルの作成 (CRTLF) コマンドとは異なり、 OPNQRYF コマンドは、データを処理するための一時ファイルだけを作成します。永続ファイルは作成されません。

OPNQRYF コマンドには、DDS にある機能や、CRTPF コマンドおよび CRTLF コマンドの機能と類似した機能があります。DDS は、ファイルを作成するためのソース・ステートメントと、個別のステップを必要とします。OPNQRYF を用いれば、DDS を使用せずに動的に定義を行うことができます。OPNQRYF コマンドは DDS の機能のすべてはサポートしていませんが、DDS の機能を上回る優れた機能をサポートしています。さらに、OPNQRYF コマンドで実行される機能の一部は、 Query for iSeries を使用して実行することもできます。しかし、プログラミング・ツールとしては OPNQRYF コマンドの方が有用です。

OPNQRYF コマンド・パラメーターにはまた、SQL の SELECT ステートメントと類似した多数の機能もあります。たとえば、 FILE パラメーターは SQL の FROM ステートメントに類似し、 QRYSLT パラメーターは SQL の WHERE ステートメントに類似し、 GRPFLD パラメーターは SQL の GROUP BY ステートメントに類似し、 GRPSLT パラメーターは SQL の HAVING ステートメントに類似しています。SQL の詳細については、 SQL プログラミングのトピックを参照してください。

OPNQRYF が提供する主な機能のリストを以下に示します。

- 動的レコード選択
- 動的キー順アクセス・パス
- 結合ファイルに対する動的キー順アクセス・パス
- 動的結合
- 2 次結合ファイル内での脱落レコードの処理
- 固有キー処理
- マップ・フィールド定義
- グループ処理
- 最終合計のみの処理
- パフォーマンスの向上
- オープン・ファイル識別コード (ID)
- 分類順序の処理

OPNQRYF コマンドの構文およびパラメーターについての説明は、 制御言語 (CL) に関するトピックを参照してください。

OPNQRYF コマンドを使用して Query を作成する方法については、 128 ページの『OPNQRYF コマンドを用いた Query の作成』を参照してください。

OPNQRYF コマンドを理解するためには、ファイル内のレコード様式の使用と、異なるレコード様式を持つファイルの使用という 2 つの処理アプローチを把握しておく必要があります。OPNQRYF コマンドの代表的な使用法は、高水準言語プログラムによる順次の読み取りが可能になるようにデータの選択、編成、および形式設定を行うことです。これらの処理アプローチについては、以下のトピックを参照してください。

- 128 ページの『ファイル内の既存のレコード様式の使用』
- 130 ページの『異なるレコード様式のファイルの使用』

OPNQRYF の主要機能のパラメーターを指定する方法、および Query ファイルのオープン・コマンドを高水準言語プログラムと一緒に使用する方法について、詳しくは以下のトピックを参照してください。

- 132 ページの『OPNQRYF コマンドを用いた CL プログラムのコーディング』
- 133 ページの『長さゼロのリテラルと内容 (*CT) 関数』
- 133 ページの『DDS を用いないレコードの選択』

これらのトピックには例が含まれています。例に関する注意事項については、 132 ページの『OPNQRYF の例』をご覧ください。

これらの主要機能に関連して OPNQRYF を使用するときの考慮事項について、以下のトピックを参照してください。

- 160 ページの『ファイルの作成および FORMAT パラメーターの使用に関する考慮事項』
- 161 ページの『レコードの配列に関する考慮事項』
- 161 ページの『DDM ファイルに関する考慮事項』
- 161 ページの『高水準言語プログラムの作成に関する考慮事項』

OPNQRYF 使用時に出されるメッセージについては、 162 ページの『Query ファイルのオープン (OPNQRYF) コマンドの実行中に送られるメッセージ』を参照してください。

その他の OPNQRYF 使用方法については、以下のトピックを参照してください。

- 164 ページの『入力だけに限定しない Query ファイルのオープン (OPNQRYF) コマンドの使用』
- 164 ページの『OPNQRYF コマンドを使用した日付、時刻、および時刻スタンプの比較』
- 165 ページの『OPNQRYF コマンドを使用した日付、時刻、および時刻スタンプの算術演算』
- 169 ページの『ランダム処理のための Query ファイルのオープン (OPNQRYF) コマンドの使用』

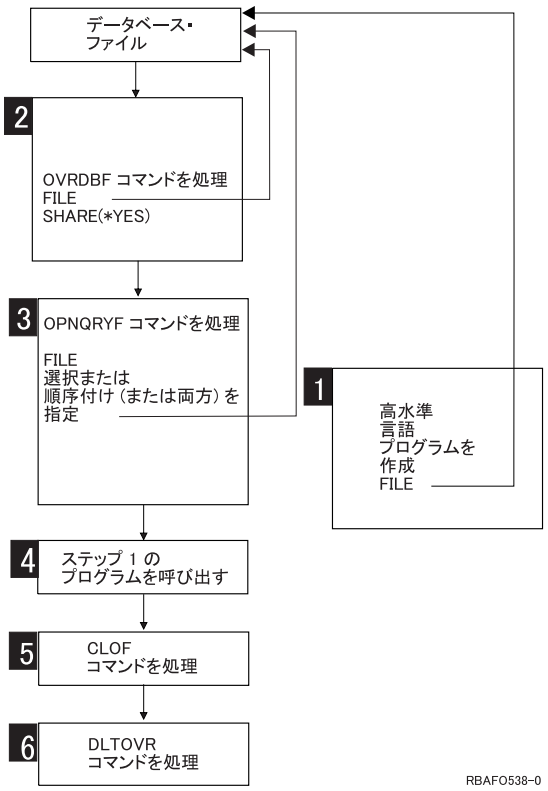
OPNQRYF のパフォーマンス、その他の考慮事項については、以下のトピックを参照してください。

- 169 ページの『Query ファイルのオープン・コマンド: パフォーマンスの考慮事項』
- 171 ページの『Query ファイルのオープン・コマンド: 分類順序表のパフォーマンスの考慮事項』
- 172 ページの『他のデータベース機能とのパフォーマンスの比較』
- 172 ページの『フィールドの使用に関する考慮事項』
- 173 ページの『ジョブで共用されるファイルに関する考慮事項』
- 174 ページの『レコード様式記述が変更されたかどうかの検査に関する考慮事項』
- 174 ページの『OPNQRYF コマンドに関するその他の実行時の考慮事項』

OPNQRYF 使用時のエラーについては、 176 ページの『Query ファイルのオープン (OPNQRYF) コマンドの使用時の代表的なエラー』を参照してください。

OPNQRYF コマンドを用いた Query の作成: Query を作成するには、 OPNQRYF コマンドを使用できます。あるいは、 iSeries ナビゲーターの「Run SQL スクリプト」ウィンドウを使用して、 Query を作成することもできます。 Run SQL スクリプトを使用してスクリプト (Query) を作成を参照してください。

ファイル内の既存のレコード様式の使用: *Code* フィールドが D であるレコードだけをプログラムで処理したいとします。プログラムは、 *Code* フィールドが D であるレコードしかないかのように作成します。つまり、プログラムでは選択操作をまったくコーディングしません。その後で、OPNQRYF コマンドを実行し、*Code* フィールドが D であるレコードだけがプログラムに返されるように指定します。OPNQRYF コマンドによってレコードの選択が行なわれるので、プログラムでは選択値を満たすレコードだけが処理されます。この方法を使用して、レコードのセットを選択したり、保管されていた順序とは異なった順序でレコードを返したり、あるいはその両方を行うことができます。以下に示すのは、OPNQRYF コマンドを使用して、レコードの選択および順序付けを行う例です。



1 外部記述データを使用する通常のプログラムと同様に、データベース・ファイルを処理する高水準言語プログラムを作成します。レコード様式は 1 つだけ使用できますが、そのレコード様式はファイルに存在するものでなければなりません。

2 処理するファイルとメンバー、および SHARE(*YES) を指定した上で OVRDBF コマンドを実行します。(メンバーが永久的に SHARE(*YES) に変更されており、使用したいメンバーが最初のメンバーまたは唯一のメンバーである場合には、このステップは必要ありません。)

OPNQRYF コマンドで指定したファイル名を一時変更したい場合を除いて、 OVRDBF コマンドは OPNQRYF コマンドの後に実行できます。この説明と実例では、OVRDBF コマンドを最初に示しています。

OPNQRYF コマンドと一緒に OVRDBF コマンドを使用する場合には、制約事項があります。たとえば、MBR(*ALL) を指定すると、エラー・メッセージが出されて、ファイルはオープンされません。詳細については、 173 ページの『ジョブで共用されるファイルに関する考慮事項』を参照してください。

3 データベース・ファイル、メンバー、様式名、選択オプション、順序付けオプション、およびオープン・ファイルの影響の有効範囲を指定して、OPNQRYF コマンドを実行してください。

4 ステップ 1 で作成した高水準言語プログラムを呼び出します。高水準言語を使用するほかに、Query ファイルからのコピー (CPYFRMQRYF) コマンドを使用しても、OPNQRYF コマンドによって作成されたファイルを処理することができます。その他の CL コマンド (たとえば、ファイルのコピー [CPYF] コマンドおよび物理ファイル・メンバーの表示 [DSPPFM] コマンド) およびユーティリティー (たとえば、Query) は、OPNQRYF コマンドで作成されたファイルを処理しません。

5 ファイルをオープンしたままにしておきたい場合を除いて、ステップ 3 でオープンしたファイルをクローズします。ファイルをクローズするには、ファイルのクローズ (CLOF) コマンドを使用できます。

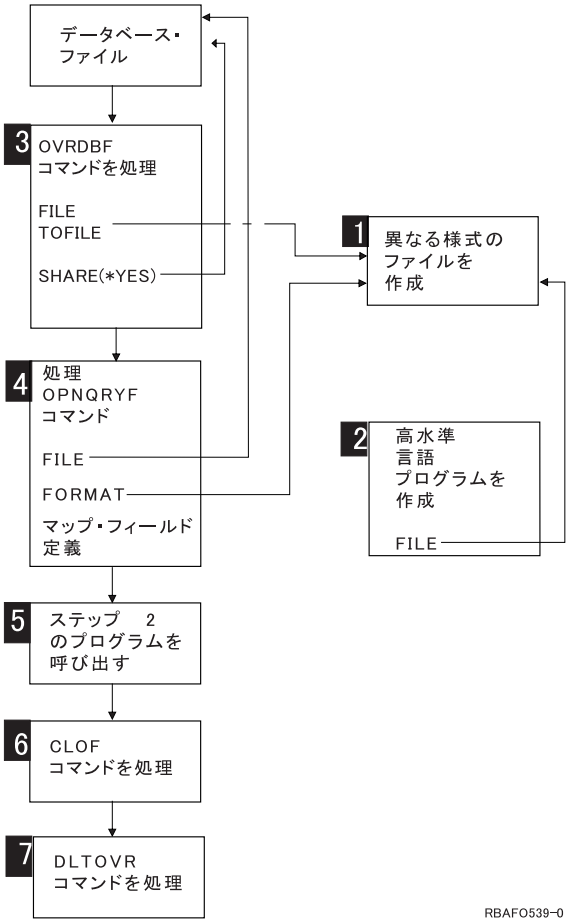
6 一時変更の削除 (DLTOVR) コマンドを使用して、ステップ 2 で指定した一時変更を削除します。一時変更の削除が常に必要とは限りませんが、一貫性を持たせるために、すべての例にこのコマンドを示しています。

異なるレコード様式のファイルの使用: Query ファイルのオープン (OPNQRYF) コマンドのより高度な機能 (別々のファイルからのレコードの動的結合など) を使用するためには、異なるレコード様式を含む新しいファイルを定義しなければなりません。この新しいファイルは、処理しようとしているファイルとは別のファイルです。この新しいファイルには、OPNQRYF コマンドで作成しようとするフィールドが入っています。この強力な機能を使用すると、現在データベース・レコードには存在していないものの、それらのレコードから抽出することのできるフィールドを定義できます。

高水準言語プログラムをコーディングする場合には、既存のフィールドと抽出フィールドの両方の外部記述フィールド定義をプログラムで処理できるように、異なる様式が入っているファイルの名前を指定します。

高水準言語プログラムを呼び出す前に、プログラム・ファイル名を Query ファイルのオープン・コマンドに指示するために、データベース・ファイルによる一時変更 (OVRDBF) コマンドを指定しなければなりません。OPNQRYF コマンドでは、データベース・ファイルと、高水準言語プログラムによって使用される特殊な様式の新しいファイルの両方を指定してください。Query の対象となるファイルに SHARE(*YES) を指定していない場合は、 OVRDBF コマンドで SHARE(*YES) を指定しなければなりません。

以下は、処理の流れを示しています。



1 異なるレコード様式を持つファイルの DDS を指定し、そのファイルを作成します。このファイルには、高水準言語プログラムで処理したいフィールドが入っています。通常、このファイルにはデータがなく、メンバーは必要ありません。このファイルは、通常、キーのない物理ファイルとして作成します。フィールドを記述するために、フィールド参照ファイルを使用できます。レコード様式名は、指定したデータベース・ファイルのレコード様式名と異なっていてもかまいません。この機能では、どのデータベース・ファイルまたは DDM ファイルでも使用できます。このファイルは、論理ファイルにすることも、索引付きファイルにすることもできます。このファイルは、データの有無を問わず、1 つまたは複数のメンバーを持つことができます。

2 ステップ 1 で作成したレコード様式を持つファイルを処理するための高水準言語プログラムを作成します。このプログラムでは、データが入っているデータベース・ファイルの名前を指定しないでください。

3 データベース・ファイルによる一時変更 (OVRDBF) コマンドを実行します。FILE パラメーターを用いて、異なる (新しい) レコード様式を持つファイルの名前を指定します。TOFILE パラメーターを用いて、Query の対象としたいデータベース・ファイルの名前を指定します。MBR パラメーターでメンバー名を指定することもできます。Query の対象とするデータベース・メンバーに SHARE(*YES) が指定されていない場合は、OVRDBF コマンドで SHARE(*YES) も指定しなければなりません。

4 Query ファイルのオープン (OPNQRYF) コマンドを実行します。Query の対象とするデータベース・ファイルを FILE パラメーターで指定し、ステップ 1 で作成した異なる (新しい) 様式を持つファイルの名前を FORMAT パラメーターで指定します。OPNQRYF コマンドでは、データベース・ファイルからのデータを、ステップ 1 で作成された様式にマップする方法を記述するため

に、マップ・フィールド定義が必要になる可能性があります。さらに、オープンされるファイルについての選択オプション、順序付けオプション、および影響の範囲を指定することができます。

5 ステップ 2 で作成した高水準言語プログラムを呼び出します。

6 ステップ 4 で FILE パラメーターに指定した最初のファイルは、 OPNQRYF コマンドで SHARE(*YES) としてオープンされ、オープンされたままになっています。このファイルをクローズしなければなりません。ファイルのクローズ (CLOF) コマンドを使用できます。

7 ステップ 3 で指定した一時変更を削除します。

上記のステップは、外部記述データを使用する通常の流れを示しています。各 OPNQRYF コマンドごとに固有の DDS およびレコード様式を作成する必要はありません。既存のレコード様式を再使用することができます。しかし、レコード様式の中のフィールドはすべて、実際のデータベース・ファイルの実際のフィールドであるか、あるいはマップ・フィールド定義で定義されているフィールドでなければなりません。プログラム記述データを使用する場合は、プログラムをいつでも作成できます。

ステップ 1 で作成したファイルを使用して、 Query ファイルのオープン (OPNQRYF) コマンドで作成されたデータを保存できます。たとえば、異なるレコード様式を持つファイルにデータをコピーする高水準言語処理プログラムとステップ 5 を置き換えたり、 Query ファイルからのコピー (CPYFRMQRYF) コマンドを使用できます。ファイルのコピー (CPYF) コマンドは使用できません。この場合、ステップ 5 の後に CPYF コマンドまたは Query を続けることができます。

OPNQRYF の例: 以下の項では、前述の主要な機能のそれぞれについて OPNQRYF パラメーターを指定する方法と、Query ファイルのオープン・コマンドを高水準言語プログラムと一緒に使用する方法を説明します。

注:

1. OPNSCOPE(*ACTGRPDFN) または TYPE(*NORMAL) パラメーター・オプションを指定して、コマンド入力行から OPNQRYF コマンドを実行すると、 OPNQRYF コマンドが正常に実行した後で出たエラー・メッセージにより、ファイルがクローズされません。そのようなメッセージは、バージョン 2 リリース 3 より前では、TYPE(*NORMAL) が使用された場合にファイルをクローズしていました。エラー・メッセージ (システムがコマンド内でエラーを検出した場合に送信されるメッセージ CPF0001 は除く) が出された場合、システムは、自動的に資源の再利用 (RCLRSC) コマンドを実行します。ただし、RCLRSC コマンドによってクローズされるのは、 RCLRSC コマンドが実行されたレベルよりも高いレベル (呼び出しスタックにおいて) で、省略時活動化グループからオープンされたファイルだけです。
2. 順次処理のために Query ファイルのオープン・コマンドを使用するプログラムを実行した後のファイル位置は、通常、ファイルの終わりです。同じファイルを用いて同じプログラムまたは別のプログラムを実行したい場合には、ファイルの配置を行うか、あるいはファイルをクローズしてから、そのファイルを同じ OPNQRYF コマンドでオープンしなければなりません。ファイルの配置を行うには、データベース・ファイルの配置 (POSDBF) コマンドを使用することができます。場合によっては、高水準言語プログラム・ステートメントを使用することができます。

OPNQRYF の例については、以下のセクションを参照してください。

- 『OPNQRYF コマンドを用いた CL プログラムのコーディング』
- 133 ページの『長さゼロのリテラルと内容 (*CT) 関数』
- 133 ページの『DDS を用いないレコードの選択』

OPNQRYF コマンドを用いた CL プログラムのコーディング: Query ファイルのオープン (OPNQRYF) コマンドには、コーディング・エラーを防止するための 3 つの基本規則があります。

1. アンパーサンド (&) を付けないで、データベース・ファイルからの選択フィールドを指定する。DCL または DCLF を指定して CL プログラム内で宣言されたフィールドには、アンパーサンドが必要です。
2. DCL または DCLF を指定して CL プログラム内で定義されたフィールドは、単一引用符で囲む (たとえば、'&testfld')。
3. すべてのパラメーター比較は、文字フィールドと比較される場合は二重引用符で、数字フィールドと比較される場合は単一引用符で囲む。

以下の例では、INVCUS および INVPRD のフィールドは文字データとして定義されています。

```
QRYSLT('INVCUS *EQ "' *CAT &K1CUST *CAT '" *AND +
        INVPRD *GE "' *CAT &LPRD *CAT '" *AND +
        INVPRD *LE "' *CAT &HPRD *CAT '"')
```

これらのフィールドが数値データと定義された場合には、QRYSLT パラメーターは次のようになります。

```
QRYSLT('INVCUS *EQ ' *CAT &K1CUST *CAT ' *AND +
        INVPRD *GE ' *CAT &LPRD *CAT ' *AND +
        INVPRD *LE ' *CAT &HPRD *CAT ' ')
```

長さゼロのリテラルと内容 (*CT) 関数: *長さゼロのリテラル* の概念は、バージョン 2、リリース 1、モディフィケーション・レベル 1 で取り入れられました。OPNQRYF コマンドでは、長さゼロのリテラルは、何も含まない、つまり複数の引用符 (″″) の間にブランクもない引用符付き文字列として表されます。

長さゼロのリテラルのサポートは、内容 (*CT) 関数の比較引き数として使用されるとき、比較の結果を変更します。次のステートメントについて考慮してみてください。

```
QRYSLT('field *CT ""')
```

長さゼロのリテラルがサポートされている場合は、ステートメントは何かが入っているレコードを返します。これは本質的に、任意の数の文字が続く任意の数の文字のためのワイルドカード比較です。これは、次のステートメントと等価です。

```
'field = %WLDCRD("**")'
```

長さゼロのリテラルがサポートされる以前は (バージョン 2 リリース 1 モディフィケーション・レベル 1 より前)、引き数 (″″) は単一バイトのブランクとして解釈されました。このステートメントは、フィールドのどこかに単一ブランクを入れたレコードを返しました。それは本質的に、ブランクと任意の数の文字が後続する、任意の数の文字のワイルドカード比較でした。次のステートメントと等価でした。

```
'field = %WLDCRD("* *")'
```

バージョン 2 リリース 1 モディフィケーション・レベル 1 より前の結果を、内容関数で得るには、ブランクがはっきりと分かるようにして QRYSLT をコーディングしなければなりません。

```
QRYSLT('field *CT " "')
```

DDS を用いないレコードの選択: 動的レコード選択を行うと、DDS を使用しないでファイル中のレコードのサブセットを要求することができます。たとえば、特定の値または特定の値の範囲を持つレコードを選択することができます (たとえば、1000 ～ 1050 の範囲の得意先番号)。Query ファイルのオープン (OPNQRYF) コマンドを使用すると、これらを他の選択機能を組み合わせて、強力なレコード選択機能を作り出すことができます。

例については、以下のトピックを参照してください。

- 134 ページの『Query ファイルのオープン (OPNQRYF) コマンドを使用したレコード選択』
- 145 ページの『DDS を用いないキー順アクセス・パスの指定』

- 146 ページの『異なるファイルにあるキー・フィールドの指定』
- 147 ページの『DDS を用いないデータベース・ファイルの動的結合』
- 150 ページの『2 次結合ファイルから欠落しているレコードの処理』
- 151 ページの『固有キーの処理』
- 152 ページの『既存のフィールド定義から抽出されるフィールドの定義』
- 155 ページの『ゼロによる除算の処理』
- 155 ページの『データベース・ファイル・レコードからのデータの集約 (グループ化)』
- 158 ページの『最終合計のみの処理』
- 159 ページの『システムが Query ファイルのオープン・コマンドを実行する方法の制御』

Query ファイルのオープン (OPNQRYF) コマンドを使用したレコード選択: 以下の例では、いずれも、単一様式のデータベース・ファイル (物理ファイルまたは論理ファイル) を処理するものと仮定しています。(OPNQRYF コマンドの FILE パラメーターを使用すれば、ファイルが複数様式の論理ファイルである場合にレコード様式名を指定することができます。)

- 『例 1: OPNQRYF コマンドを使用したレコードの選択』
- 136 ページの『例 2: OPNQRYF コマンドを使用したレコードの選択』
- 137 ページの『例 3: OPNQRYF コマンドを使用したレコードの選択』
- 138 ページの『例 4: OPNQRYF コマンドを使用したレコードの選択』
- 138 ページの『例 5: OPNQRYF コマンドを使用したレコードの選択』
- 139 ページの『例 6: OPNQRYF コマンドを使用したレコードの選択』
- 139 ページの『例 7: OPNQRYF コマンドを使用したレコードの選択』
- 140 ページの『例 8: OPNQRYF コマンドを使用したレコードの選択』
- 142 ページの『例 9: OPNQRYF コマンドを使用したレコードの選択』
- 143 ページの『例 10: OPNQRYF コマンドを使用したレコードの選択』
- 144 ページの『例 11: OPNQRYF コマンドを使用したレコードの選択』

QRYSLT パラメーターで使用される式の形式の詳細については、 制御言語 (CL) に関するトピックの OPNQRYF コマンドを参照してください。

例 1: OPNQRYF コマンドを使用したレコードの選択: 特定の値を持つレコードの選択

FILEA から、*Code* フィールドの値が D であるレコードをすべて選択したいとします。処理プログラムは PGMB です。PGMB は、選択値を満たすレコードだけを取り出します (ユーザーがプログラム中でテストする必要はありません)。

注: OPNQRYF コマンドの場合、プロンプト機能を使用するとパラメーターの指定が容易になります。たとえば、QRYSLT パラメーターの式を区切り文字で囲まずに指定できます (システムによりアポストロフィが追加されるため)。

以下のように指定してください。

```
OVRDBF     FILE(FILEA) SHARE(*YES)
OPNQRYF    FILE(FILEA) QRYSLT('CODE *EQ "D" ')
CALL       PGM(PGMB)
CLOF       OPNID(FILEA)
DLTOVR     FILE(FILEA)
```

注:

1. QRYSLT パラメーターの式は、全体をアポストロフィで囲まなければなりません。
2. OPNQRYF コマンドでフィールド名を指定する場合には、レコード中の名前はアポストロフィで囲みません。
3. 文字リテラルは、引用符または二重アポストロフィで囲まなければなりません。(例では引用符文字を使用しています。) 引用符の中の文字 (複数の場合もある) は、データベース内で見つけ出したい値と合わせて大文字または小文字のいずれかにすることが大切です。(例では、すべて大文字になっています。)
4. 数値定数に対する選択を要求するためには、以下のように指定してください。

   ```
   OPNQRYF    FILE(FILEA) QRYSLT('AMT *GT 1000.00')
   ```

   数値定数は二重アポストロフィ (引用符) では*囲まない* ことに注意してください。
5. フィールド値と CL 変数を比較するとき、以下のアポストロフィを使用してください (ただし、CL 可変長文字は使用できます)。
   - 文字、日付、時刻、時刻スタンプのいずれかのフィールドに対し選択を行うとき、以下のように指定してください。

     ```
     OPNQRYF    FILE(FILEA) QRYSLT('"' *CAT &CHAR *CAT '" *EQ FIELDA')
     ```

     または以下のように逆の順序を指定してください。

     ```
     OPNQRYF    FILE(FILEA) QRYSLT('FIELDA *EQ "' *CAT &CHAR *CAT '"')
     ```

     アポストロフィと引用符で CL 変数と *CAT 演算子を囲むことに注意してください。
   - 数字フィールドに対し選択を行うとき、以下のように指定してください。

     ```
     OPNQRYF    FILE(FILEA) QRYSLT(&CHARNUM *CAT ' *EQ NUM')
     ```

     または以下のように逆の順序を指定してください。

     ```
     OPNQRYF    FILE(FILEA) QRYSLT('NUM *EQ ' *CAT &CHARNUM);
     ```

     アポストロフィがフィールドと演算子のみを囲むことに注意してください。

2 つのフィールドまたは定数を比較するときは、データ・タイプに互換性がなければなりません。以下の表は、有効な比較を示しています。

表 10. OPNQRYF コマンドの有効なデータ・タイプの比較

	任意の形式の数値	**文字**	**日付[1]**	**時刻[1]**	**時刻スタンプ[1]**
任意の形式の数値	有効	無効	無効	無効	無効
文字	無効	有効	有効[2]	有効[2]	有効[2]
日付[1]	無効	有効[2]	有効	無効	無効
時刻[1]	無効	有効[2]	無効	有効	無効
時刻スタンプ[1]	無効	有効[2]	無効	無効	有効

:

1 日付、時刻、時刻スタンプのデータ・タイプは、フィールドおよび式によって表現できますが、定数では表現できません。しかし、文字定数は日付、時刻、または時刻スタンプ値を表現することができます。

2 文字フィールドまたは定数は、日付データ・タイプと比較される場合は有効な日付値を、時刻データ・タイプと比較される場合は有効な時刻値を、時刻スタンプ・データ・タイプと比較される場合は有効な時刻スタンプ値を表していなければなりません。

注: DBCS の情報については、2 バイト文字セット (DBCS) に関する考慮事項を参照してください。

キー順アクセス・パスで選択されるフィールドを使用するファイルがシステム上にあると、レコード選択のパフォーマンスが大幅に向上します。このような場合には、システムは選択値を満たすレコードだけを迅速にアクセスすることができます。こうしたアクセス・パスがない場合には、システムはレコードをすべて読み取って、それが選択値を満たすかどうかを調べなければなりません。

選択元とするフィールドにアクセス・パスが存在していても、システムがそのアクセス・パスを使用しないことがあります。たとえば、到着順でデータを処理する方が速い場合には、システムはその方法を採用します。詳細については、169 ページの『Query ファイルのオープン・コマンド: パフォーマンスの考慮事項』を参照してください。

例 2: OPNQRYF コマンドを使用したレコードの選択: 特定の日付の値を持つレコードの選択

レコード中の *Date* フィールドが現在の日付と同じであるすべてのレコードを処理したいとします。また、*Date* フィールドがシステム日付と同じ形式となっているとします。CL プログラムでは以下のように指定することができます。

```
DCL        VAR(&CURDAT);  TYPE(*CHAR)  LEN(6)
RTVSYSVAL  SYSVAL(QDATE) RTNVAR(&CURDAT);
OVRDBF     FILE(FILEA) SHARE(*YES)
OPNQRYF    FILE(FILEA) QRYSLT('"' *CAT &CURDAT *CAT '" *EQ DATE')
CALL       PGM(PGMB)
CLOF       OPNID(FILEA)
DLTOVR     FILE(FILEA)
```

CL 変数には先行アンパーサンド (&) が付けられ、アポストロフィで囲まずに指定されています。式全体が、アポストロフィで囲まれています。CAT 演算子および CL 変数は、アポストロフィと引用符の両方で囲まれています。

データベース中のデータが、文字、日付、時刻、時刻スタンプ、または数値のどのタイプとして定義されているかを知ることは重要です。先にあげた例では、*Date* フィールドは文字であると仮定しています。

Date フィールドが日付データ・タイプとして定義された場合は、先にあげた例は、以下のように指定することになります。

```
OVRDBF FILE(FILEA) SHARE(*YES)
OPNQRYF FILE(FILEA) QRYSLT('%CURDATE *EQ DATE')
CALL PGM(PGMB)
CLOF OPENID(FILEA)
DLTOVR FILE(FILEA)
```

注: 日付フィールドはシステム日付と同じ形式である必要はありません。
例は、以下のようにも指定できます。

```
DCL VAR(&CVTDAT); TYPE(*CHAR) LEN(6)
DCL VAR(&CURDAT); TYPE(*CHAR) LEN(8)
RTVSYSVAL SYSVAL(QDATE) RTNVAR(&CVTDAT);
CVTDAT DATE(&CVTDAT); TOVAR(&CURDAT); TOSEP(/)
OVRDBF FILE(FILEA) SHARE(*YES)
OPNQRYF FILE(FILEA)
       QRYSLT('"' *CAT &CURDAT *CAT '" *EQ DATE')
CALL PGM(PGMB)
CLOF OPNID (FILEA)
DLTOVR FILE(FILEA)
```

ここで、*DATE* は FILEA で日付データ・タイプとなっており、ジョブの省略時の日付形式は MMDDYY、ジョブの省略時の日付区切り文字はスラッシュ (/) です。

注: MMDDYY (月日年)、DDMMYY (日月年)、YYMMDD (年月日)、または年間通算日のいずれかの形式であるすべての日付の文字表記が認識されるためには、ジョブの省略時の日付形式および区切り文字が同じものでなければなりません。

代わりに定数を用いた場合、QRYSLT は以下のように指定されます。

```
QRYSLT('"12/31/87" *EQ DATE')
```

ジョブの省略時の日付形式は MMDDYY でなければならず、ジョブの省略時の区切り文字はスラッシュ (/) でなければなりません。

データベースに数字フィールドが存在し、それを変数と比較したい場合には、文字変数だけが使用できます。たとえば、パック 10 進数の *Date* フィールドが変数より大きいすべてのレコードを選択するためには、その変数を文字形式にしなければなりません。通常、これは、Query ファイルのオープン (OPNQRYF) コマンドの前に変数の変更 (CHGVAR) コマンドを使用して、変数を 10 進数フィールドから文字フィールドに変更することを意味します。CHGVAR コマンドは、以下のように指定します。

```
CHGVAR VAR(&CHARVAR);  VALUE('123188')
```

QRYSLT パラメーターは、以下のように指定します (先にあげた例との違いに注意してください)。

```
QRYSLT(&CHARVAR  *CAT ' *GT DATE')
```

代わりに定数を用いた場合、QRYSLT ステートメントは、以下のように指定されます。

```
QRYSLT('123187 *GT DATE')
```

例 3: OPNQRYF コマンドを使用したレコードの選択: 値の範囲のレコードの選択

Date フィールドが文字形式 YYMMDD および区切り文字 『.』 で指定されており、 1988 年のすべてのレコードを処理したいとします。以下のように指定することができます。

```
OVRDBF     FILE(FILEA) SHARE(*YES)
OPNQRYF    FILE(FILEA) QRYSLT('DATE *EQ %RANGE("88.01.01" +
               "88.12.31") ')
CALL       PGM(PGMC)
CLOF       OPNID(FILEA)
DLTOVR     FILE(FILEA)
```

DATE フィールドが日付データ・タイプとなっており、ジョブの省略時の日付形式が YYMMDD で、ジョブの省略時の日付区切り文字がピリオド (.) である場合にも、この列は有効です。.

注: MMDDYY (月日年)、DDMMYY (日月年)、YYMMDD (年月日)、または年間通算日のいずれかの形式であるすべての日付の文字表記が認識されるためには、ジョブの省略時の日付形式および区切り文字が同じものでなければなりません。

範囲が文字データ・タイプとして定義された変数で、DATE フィールドが文字データ・タイプとして定義された場合、QRYSLT パラメーターを次のように指定してください。

```
QRYSLT('DATE *EQ %RANGE("' *CAT &LORNG *CAT '"' *BCAT '"'  +
   *CAT &HIRNG *CAT '")')
```

しかしながら、DATE フィールドが数値データ・タイプとして定義された場合は、QRYSLT パラメーターを次のように指定してください。

```
QRYSLT('DATE *EQ %RANGE(' *CAT &LORNG *BCAT &HIRNG *CAT ')')
```

注: QRYSLT パラメーターが CL プログラムにある場合、*BCAT が使用できます。しかし対話式コマンドでは許可されません。

例 4: OPNQRYF コマンドを使用したレコードの選択: 内容関数を用いたレコードの選択

Addr フィールドに BROADWAY という通りが含まれているすべてのレコードを処理したいとします。内容 (*CT) 関数は、指定のフィールドの中にこれらの文字が入っているかどうかを調べることができます。以下のように指定することができます。

```
OVRDBF     FILE(FILEA) SHARE(*YES)
OPNQRYF    FILE(FILEA) QRYSLT('ADDR *CT "BROADWAY" ')
CALL       PGM(PGMC)
CLOF       OPNID(FILEA)
DLTOVR     FILE(FILEA)
```

この例では、データベース・レコード中のデータが大文字になっていると仮定しています。データが小文字または大文字小文字混合である場合には、比較を行う前に、それらのデータを大文字に変換するために変換関数を指定することができます。システム提供の表 QSYSTRNTBL により、a 〜 z の英字は大文字に変換されます。(変換を行うには、任意の変換表を使用することができます。) したがって、以下のように指定することができます。

```
OVRDBF     FILE(FILEA) SHARE(*YES)
OPNQRYF    FILE(FILEA) QRYSLT('%XLATE(ADDR QSYSTRNTBL) *CT +
                "BROADWAY" ')
CALL       PGM(PGMC)
CLOF       OPNID(FILEA)
DLTOVR     FILE(FILEA)
```

QRYSLT ステートメントで %XLATE 関数を使用すると、高水準言語プログラムに渡されるフィールドの値は、データベースに存在するものと同じになります。MAPFLD パラメーターで %XLATE 関数を使用することによって、このフィールドを強制的に大文字にすることができます。

例 5: OPNQRYF コマンドを使用したレコードの選択: 複数のフィールドを用いたレコードの選択

Amt フィールドがゼロに等しいか、あるいは *Lstdat* フィールド (YYMMDD の文字形式) が 88-12-31 より小さいすべてのレコードを処理したいとします。以下のように指定することができます。

```
OVRDBF     FILE(FILEA) SHARE(*YES)
OPNQRYF    FILE(FILEA) QRYSLT('AMT *EQ 0 *OR LSTDAT +
                *LE "88-12-31" ')
CALL       PGM(PGMC)
CLOF       OPNID(FILEA)
DLTOVR     FILE(FILEA)
```

LSTDAT フィールドが日付データ・タイプとなっている場合にも、この例は有効です。*LSTDAT* フィールドは、どのような有効日付形式でもかまいません。しかし、ジョブの省略時の日付形式は YYMMDD でなければならず、ジョブの省略時の日付区切り文字はダッシュ (-) でなければなりません。

注: MMDDYY (月日年)、DDMMYY (日月年)、YYMMDD (年月日)、または年間通算日のいずれかの形式であるすべての日付の文字表記が認識されるためには、ジョブの省略時の日付形式および区切り文字が同じものでなければなりません。

変数を使用する場合には、QRYSLT パラメーターは以下のように入力します。

```
QRYSLT('AMT *EQ ' *CAT &VARAMT *CAT ' *OR +
   LSTDAT *LE "' *CAT &VARDAT *CAT '"')
```

または逆の順序で入力します。

```
QRYSLT('"' *CAT &VARDAT *CAT '" *GT LSTDAT *OR ' +
   *CAT &VARAMT *CAT ' *EQ AMT')
```

&VARAMT 変数は、文字タイプとして定義しなければならないことに注意してください。変数が CL プログラムに数値タイプとして渡される場合には、連結が可能になるように文字タイプに変換しなければなりません。変数の変更 (CHGVAR) コマンドを使用すると、この変換を行うことができます。

例 6: OPNQRYF コマンドを使用したレコードの選択: プログラム内での Query ファイルのオープン (OPNQRYF) コマンドの複数回の使用

高水準言語プログラムの中で OPNQRYF コマンドを 2 回以上使用できます。たとえば、ユーザーにいくつかの選択値の入力を求めるプロンプトを表示し、その後で 1 ページまたは複数ページのレコードを表示したいとします。レコードの最初の要求の後で、ユーザーが別の選択値を指定し、それらのレコードを表示したい場合があります。それには以下のことを行います。

1. 高水準言語プログラムを呼び出す前に、データベース・ファイルによる一時変更 (OVRDBF) コマンドを使用して SHARE(*YES) を指定します。
2. 高水準言語プログラムの中で、ユーザーに選択値の入力を求めるプロンプトを出します。
3. OPNQRYF コマンドを出す CL プログラムに選択値を渡します (または、プログラム QCMDEXC への呼び出しがあるコマンドを実行します)。プログラムが OPNQRYF コマンドを実行する前に、ファイルがクローズされていなければなりません。通常、ファイルのクローズ (CLOF) コマンドを使用して、ファイルがオープンされていないことをモニターします。
4. 高水準言語プログラムに戻ります。
5. 高水準言語プログラムでファイルをオープンします。
6. レコードを処理します。
7. プログラムでファイルをクローズします。
8. ステップ 2 に戻ります。

プログラムが完了した時点で、ファイルのクローズ (CLOF) コマンドまたは資源の再利用 (RCLRSC) コマンドを実行してファイルをクローズし、次いで、ステップ 1 で指定したデータベース・ファイルによる一時変更コマンドを削除します。

注: 呼び出された CL プログラム中の一時変更コマンドは、主プログラムでのオープンには影響を及ぼしません。すべての一時変更は、プログラムが終了した時点で暗黙に削除されます。(ただし、必要であれば、高水準言語プログラムからプログラム QCMDEXC への呼び出しを使用して、一時変更を指定することができます。)

例 7: OPNQRYF コマンドを使用したレコードの選択: パック 10 進数データ・フィールドのフィールド・マッピング

MMDDYY という形式のパック 10 進数の *Date* フィールドがあり、 1988 年のすべてのレコードを選択したいとします。パック 10 進数フィールドの一部分から直接にレコードを選択することはできませんが、OPNQRYF コマンドの MAPFLD パラメーターを使用して新しいフィールドを作成し、その新しいフィールドを使用して元のフィールドの一部分を選択することができます。

各マップ・フィールド定義の形式は以下のとおりです。

(結果のフィールド '式' 属性)

各文字の意味は以下のとおりです。

結果のフィールド	=	結果のフィールドの名前。

式	=	結果のフィールドを導く方法。式には、サブストリング、他の組み込み関数、または数学ステートメントを含めることができます。
属性	=	結果のフィールドの任意選択属性。属性を指定しない場合 (またはフィールドがファイルに定義されていない場合) には、OPNQRYF コマンドが、式に指定されたフィールドによって決まるフィールド属性を計算します。

```
OVRDBF     FILE(FILEA) SHARE(*YES)
OPNQRYF    FILE(FILEA) QRYSLT('YEAR *EQ "88" ') +
             MAPFLD((CHAR6 '%DIGITS(DATE)')  +
               (YEAR '%SST(CHAR6 5 2)' *CHAR 2))
CALL       PGM(PGMC)
CLOF       OPNID(FILEA)
DLTOVR     FILE(FILEA)
```

この例では、DATE が日付データ・タイプの場合には、以下のように指定することができます。

```
OPNQRYF FILE(FILEA) +
QRYSLT ('YEAR *EQ 88') +
MAPFLD((YEAR '%YEAR(DATE)'))
```

最初のマップ・フィールド定義では、パック 10 進数の *Date* フィールドから *Char6* というフィールドを作成することが指定されています。%DIGITS 関数は、パック 10 進数から文字への変換を行いますが、小数部の定義は無視します (すなわち、1234.56 は 123456 に変換されます)。*Char6* フィールドの定義が指定されていないため、システムは長さ 6 を割り当てます。2 番目のマップ・フィールドでは、 *Year* フィールドがタイプ *CHAR (文字)、長さ 2 として定義されています。式では、*Char6* フィールドの最後の 2 文字を *Year* フィールドにマップするためのサブストリング関数が使用されています。

マップ・フィールド定義は指定された順序で処理される点に注意してください。この例でいえば、*Date* フィールドが文字に変換されて *Char6* フィールドに割り当てられ、次いで、*Char6* フィールドの最後の 2 桁 (年) が *Year* フィールドに割り当てられています。この順序を変更すると、違った結果が生じることになります。

注: マップ・フィールド定義は、常に QRYSLT パラメーターの評価より先に処理されます。

以下のように、QRYSLT パラメーターでサブストリングを指定し、マップ・フィールド定義の 1 つを除去しても、同じ結果を得ることができます。

```
OPNQRYF    FILE(FILEA) +
             QRYSLT('%SST(CHAR6 5 2) *EQ "88" ') +
             MAPFLD((CHAR6 '%DIGITS(DATE)'))
```

例 8: OPNQRYF コマンドを使用したレコードの選択: 『ワイルドカード』関数の使用

MMDDYY という形式のパック 10 進数の *Date* フィールドがあり、 1988 年 3 月のレコードを選択したいとします。そのためには、以下のように指定できます。

```
OVRDBF     FILE(FILEA) SHARE(*YES)
OPNQRYF    FILE(FILEA) +
             QRYSLT('%DIGITS(DATE) *EQ %WLDCRD("03__88")')
CALL       PGM(PGMC)
CLOF       OPNID(FILEA)
DLTOVR     FILE(FILEA)
```

%DIGITS 関数の結果用のデータベース・フィールドを定義するために MAPFLD パラメーターが必要となるのは、引き数として単純なフィールド名 (関数または式でなく) だけをサポートする関数で、結果を使用することが必要な場合だけである点に注意してください。%WLDCRD 演算には、*EQ 演算子の前に置かれたオペランドに対するこのような制約はありません。

さらに、データベース内のフィールドが数値形式であっても、リテラルが二重アポストロフィで囲まれているため、このフィールドの定義は *Char6* フィールドと同じになる点に注意してください。ワイルドカード関数は、DATE、TIME、または TIMESTAMP データ・タイプについてはサポートされません。

%WLDCRD 関数を使用すると、選択値に合致するレコードを選択することができます。選択値の中では、下線 (_) が任意の 1 つの文字値に相当します。例 8 では 2 つの下線文字が使用されているため、 3 月のすべての日が選択されることになります。%WLDCRD 関数を使用すれば、ワイルドカード文字を指定することもできます (下線は省略時)。

ワイルドカード関数では、2 つの異なる形式がサポートされます。

- 先の例で示されたような、固定位置ワイルドカード。この場合、下線 (またはユーザーの指定した文字) は、次の例のように任意の 1 つの文字に相当します。

  ```
  QRYSLT('FLDA *EQ %WLDCRD("A_C")')
  ```

  ABC、ACC、ADC、AxC といった文字列は、いずれも基準を満たします。この例では、分析されるフィールドの長さが正確に 3 文字である場合にだけ、比較の対象となります。フィールドが 3 文字より長い場合は、ワイルドカード・サポートの 2 番目の形式も必要となります。
- 可変位置ワイルドカードは、任意のゼロ個以上の文字に相当します。Query ファイルのオープン (OPNQRYF) コマンドでは、このタイプのワイルドカード可変文字にアスタリスク (*) を使用します。あるいは、ユーザーが独自の文字を指定することもできます。次の例ではアスタリスクが使用されています。

  ```
  QRYSLT('FLDB *EQ %WLDCRD("A*C*") ')
  ```

  AC、ABC、AxC、ABCD、AxxxxxxxC といった文字列は、いずれも基準を満たします。アスタリスクにより、このコマンドでは文字が挿入されていても無視されます。この例では、後でフィールドに入る可能性のある文字 (複数の場合もある) の前と後の両方にアスタリスクが指定されている点に注意してください。検索引き数の終わりのアスタリスクを取り除いた場合には、フィールドが C の文字で終わる場合にだけ選択されます。

  パターンがフィールドの途中で始まるレコードを選択したい場合は、ワイルドカード・ストリングの先頭にアスタリスクを指定しなければなりません。同様に、指定するパターンの残り部分がフィールドの途中で終わるレコードを選択したい場合には、パターン・ストリングの終わりにアスタリスクを指定しなければなりません。

  たとえば、以下のように指定することができます。

  ```
  QRYSLT('FLDB *EQ %WLDCRD("*ABC*DEF*") ')
  ```

  ABCDEF、ABCxDEF、ABCxDEFx、ABCxxxxxxDEF、ABCxxxDEFxxx、xABCDEF、 xABCxDEFx といった文字列が基準を満たします。

以下の例のように、2 つのワイルドカード関数を組み合わせることができます。

```
QRYSLT('FLDB *EQ %WLDCRD("ABC_*DEF*") ')
```

ABCxDEF、ABCxxxxxxDEF、ABCxxxDEFxxx といった文字列が基準を満たします。下線があるため、ABC と DEF の間には少なくとも 1 文字がなければなりません (たとえば、ABCDEF は基準を満たしません)。

以下の名前が入っている *Name* というフィールドがあるとします。

JOHNS
JOHNS SMITH
JOHNSON
JOHNSTON

以下のように指定すると、最初のレコードだけが選択されます。

```
QRYSLT('NAME *EQ "JOHNS"')
```

指定した値にブランクが追加された上で比較が行われるので、他のレコードは選択されません。4 つの名前をすべて選択するには、以下のように指定します。

```
QRYSLT('NAME *EQ %WLDCRD("JOHNS*")')
```

注: DBCS に対する %WLDCRD 関数の使用については、『2 バイト文字セット (DBCS) に関する考慮事項』を参照してください。

例 9: OPNQRYF コマンドを使用したレコードの選択: 複合選択ステートメントの使用

複合選択ステートメントを指定することもできます。たとえば、以下のように指定することができます。

```
QRYSLT('DATE *EQ "880101" *AND AMT *GT 5000.00')

QRYSLT('DATE *EQ "880101" *OR AMT *GT 5000.00')
```

また、以下のように指定することもできます。

```
QRYSLT('CODE *EQ "A" *AND TYPE *EQ "X" *OR CODE *EQ "B"')
```

演算子の処理の優先順位を制御する規則については、 制御言語 (CL) に関するトピックに記載されています。以下に規則の一部を示します。

- *AND 演算が最初に処理されます。したがって、以下の場合にレコードが選択されます。

  Code フィールド = "A" で、かつ *Type* フィールド = "X"
  または
  Code フィールド = "B" の場合

- 次の例のように、式が処理される方法を制御するために括弧を使用することができます。

  ```
  QRYSLT('(CODE *EQ "A" *OR CODE *EQ "B") *AND TYPE *EQ "X" +
          *OR CODE *EQ "C"')
  ```

  Code フィールド = "A" で、かつ *Type* フィールド = "X" の場合
  または
  Code フィールド = "B" で、かつ *Type* フィールド = "X" の場合
  または
  Code フィールド = "C" の場合

また、次の例のように、省略形の代わりに制御言語 (CL) に関するトピックで説明されている記号を使用することもできます (たとえば、 *EQ の代わりに = を使用できます)。

```
QRYSLT('CODE = "A" & TYPE = "X" | AMT > 5000.00')
```

このコマンドでは、以下のレコードがすべて選択されます。

Code フィールド = "A" で、かつ *Type* フィールド = "X"
または
Amt フィールド > 5000.00 の場合

複合選択ステートメントは、以下の例のようにコーディングすることもできます。

```
QRYSLT('CUSNBR = %RANGE("60000" "69999") & TYPE = "B" +
       & SALES>0 & ACCRCV / SALES>.3')
```

このコマンドでは、以下のレコードがすべて選択されます。

Cusnbr フィールドが 60000 ～ 69999 の範囲内にあり、
Type フィールド = "B" で、
Sales フィールドが 0 より大きく、かつ
Accrcv を *Sales* で除算した結果が 30 パーセントより大きい

例 10: OPNQRYF コマンドを使用したレコードの選択: コード化文字セット識別子 (CCSID) の使用

CCSID に関する一般情報については、iSeries グローバリゼーション を参照してください。

すべてのデータベース・ファイル中の個々の文字と DBCS フィールドには、 CCSID のタグが付けられています。この CCSID を使用すると、フィールドの比較、結合、または表示が意味のある方法で実行されるように、ファイルに保存されているデータをさらに定義することができます。たとえば、FILE1 の中に CCSID が 37 (USA) の FIELD1 があり、 FILE2 の中に CCSID が 273 (オーストラリア、ドイツ) の FILED2 がある場合に、これらのフィールドを比較すると、その比較を意味のあるものにするために適切なマッピングが行われます。

```
OPNQRYF FILE(FILEA FILEB) FORMAT(RESULTF) +
       JFLD((FILEA/NAME FILEB/CUSTOMER))
```

フィールド NAME の CCSID が 37 であり、フィールド CUSTOMER の CCSID が 273 である場合、OPNQRYF コマンドの処理中に、NAME または CUSTOMER のどちらかのマッピングが行われ、この 2 つのフィールドの結合が意味のある結果となります。

通常、MAPFLD、QRYSLT、および GRPSLT パラメーターで定義した定数には、現在のジョブに定義されている CCSID のタグが付けられます。すなわち、2 人のユーザーが異なるジョブ CCSID で同じ OPNQRYF コマンド (または OPNQRYF コマンドを含むプログラム) を実行し、 OPNQRYF コマンドの中に定数が定義されている場合には、ユーザーがそれぞれ別個の結果を得ることがあるということです。これは、定数にタグとして付けられた CCSID によって定数の扱い方が異なる可能性があるためです。

MAPFLD パラメーターを使用すると、定数を特定の CCSID でタグ付けすることができます。定義が定数だけで構成される MAPFLD を指定し、その後で MAPFLD の CCSID を指定すると、定数は MAPFLD パラメーターで指定された CCSID でタグ付けされます。たとえば、次の場合があります。

```
OPNQRYF FILE(FILEA) FORMAT(RESULTF) QRYSLT('NAME *EQ MAP1') +
       MAPFLD((MAP1 '"Smith"' *CHAR 5 *N 37))
```

定数 "Smith" には、OPNQRYF コマンドを出したユーザーのジョブ CCSID が何であるかを問わず、CCSID 37 のタグが付けられます。この例では、すべてのユーザーが得るレコードは同じものとなります (ただし、結果のレコードは、ユーザーのジョブ CCSID にマップされます)。逆に、Query を以下のように指定したとします。

```
OPNQRYF FILE(FILEA) FORMAT(RESULTF) QRYSLT('NAME *EQ "Smith"')
```

この Query の結果は、OPNQRYF コマンドを出したユーザーのジョブ CCSID に応じて、異なったものになる場合があります。

例 11: OPNQRYF コマンドを使用したレコードの選択: 分類順序と言語識別コードの使用

分類順序の使用方法を調べるために、表 11 で示された STAFF ファイルに対して、この項の例を実行してください。

表 11. STAFF ファイル

ID	NAME	DEPT	JOB	YEARS	SALARY	COMM
10	Sanders	20	Mgr	7	18357.50	0
20	Pernal	20	Sales	8	18171.25	612.45
30	Merenghi	38	MGR	5	17506.75	0
40	OBrien	38	Sales	6	18006.00	846.55
50	Hanes	15	Mgr	10	20659.80	0
60	Quigley	38	SALES	00	16808.30	650.25
70	Rothman	15	Sales	7	16502.83	1152.00
80	James	20	Clerk	0	13504.60	128.20
90	Koonitz	42	sales	6	18001.75	1386.70
100	Plotz	42	mgr	6	18352.80	0

例において、結果は次のおのおのを使用した特別なステートメントに示されています。

- *HEX 分類順序
- 言語識別コード ENU に対する同順位分類順序
- 言語識別コード ENU に対する固有分類順序

注: ENU は、OPNQRYF コマンドで SRTSEQ(*LANGIDUNQ) または SRTSEQ(*LANGIDSHR) のいずれか、および LANGID(ENU) を指定することによって、言語識別コードとして選択されます。

次のコマンドは、JOB フィールドに値 MGR を持つレコードを選択します。

```
OPNQRYF FILE(STAFF) QRYSLT('JOB *EQ "MGR"')
```

表 12 は、*HEX 分類順序によるレコード選択を示します。JOB フィールドのレコード選択基準に合致するレコードは、 QRYSLT ステートメントで指定されたそのとおりに選択されます。大文字の MGR だけが選択されます。

*表 12. *HEX 分類順序の使用*: OPNQRYF FILE(STAFF) QRYSLT('JOB *EQ ″MGR″') SRTSEQ(*HEX)*

ID	NAME	DEPT	JOB	YEARS	SALARY	COMM
30	Merenghi	38	MGR	5	17506.75	0

145 ページの表 13 は、同順位分類順序によるレコード選択を示します。JOB フィールドのレコード選択基準に合致するレコードは、大文字と小文字を同等に扱って選択されます。この分類順序では、mgr、Mgr、および MGR が選択されます。

表 13. 同順位分類順序の使用： OPNQRYF FILE(STAFF) QRYSLT('JOB *EQ ″MGR″') SRTSEQ(LANGIDSHR) LANGID(ENU)

ID	NAME	DEPT	JOB	YEARS	SALARY	COMM
10	**Sanders**	**20**	**Mgr**	**7**	**18357.50**	**0**
30	**Merenghi**	**38**	**MGR**	**5**	**17506.75**	**0**
50	**Hanes**	**15**	**Mgr**	**10**	**20659.80**	**0**
100	**Plotz**	**42**	**mgr**	**6**	**18352.80**	**0**

表 14 は、固有分類順序によるレコード選択を示します。JOB フィールドのレコード選択基準に合致するレコードは、大文字と小文字を固有に扱って選択されます。この分類順序では、mgr、Mgr、および MGR 値はすべて異なります。MGR 値が選択されます。

表 14. 固有分類順序の使用： OPNQRYF FILE(STAFF) QRYSLT('JOB *EQ ″MGR″') SRTSEQ(LANGIDUNQ) LANGID(ENU)

ID	NAME	DEPT	JOB	YEARS	SALARY	COMM
30	**Merenghi**	**38**	**MGR**	**5**	**17506.75**	**0**

DDS を用いないキー順アクセス・パスの指定: 動的アクセス・パス機能を使用すると、処理するデータのキー順アクセス・パスを指定することができます。共用可能なアクセス・パスがすでに存在している場合には、システムはそのアクセス・パスを共用できます。新しいアクセス・パスが必要な場合には、プログラムにレコードが渡される前にアクセス・パスが作成されます。例については、以下のトピックを参照してください。

- 『例 1: DDS を用いないキー順アクセス・パスの指定』
- 『例 2: DDS を用いないキー順アクセス・パスの指定』
- 146 ページの『例 3: DDS を用いないキー順アクセス・パスの指定』
- 146 ページの『例 4: DDS を用いないキー順アクセス・パスの指定』

例 1: DDS を用いないキー順アクセス・パスの指定: 1 つのキー・フィールドを使用するレコードの編成

プログラム PGMD で、*Cust* フィールドの値によって編成された FILEA 中のレコードを処理したいとします。以下のように指定することができます。

```
OVRDBF     FILE(FILEA) SHARE(*YES)
OPNQRYF    FILE(FILEA) KEYFLD(CUST)
CALL       PGM(PGMD)
CLOF       OPNID(FILEA)
DLTOVR     FILE(FILEA)
```

注: 処理されるファイルとして FILEA を指定することにより PGMD が作成されているため、Query ファイルのオープン (OPNQRYF) コマンドの FORMAT パラメーターは必要ありません。FILEA は到着順ファイルとキー順ファイルのどちらでもかまいません。FILEA がキー順ならば、キー・フィールドは *Cust* フィールドでも、まったく別のフィールドでもかまいません。

例 2: DDS を用いないキー順アクセス・パスの指定: 複数のキー・フィールドを使用するレコードの編成

レコードを *Cust* フィールドの順に処理し、その後で *Cust* ごとに *Date* の順に処理したい場合は、以下のように指定します。

```
OPNQRYF    FILE(FILEA) KEYFLD(CUST DATE)
```

Date を降順に表示したい場合は、以下のように指定します。

```
OPNQRYF    FILE(FILEA) KEYFLD((CUST) (DATE *DESCEND))
```

この 2 つの例には、FORMAT パラメーターが使用されていません。(異なる様式を定義する場合には、すべてのキー・フィールドがその様式の中に存在していなければなりません。)

例 3: DDS を用いないキー順アクセス・パスの指定: 固有分類順序を使用したレコードの配列

144 ページの表 11 の STAFF ファイルを使用して、固有分類順序で JOB フィールド値によりレコードを処理するには、次のように指定してください。

```
OPNQRYF FILE(STAFF) KEYFLD(JOB) SRTSEQ(*LANGIDUNQ) LANGID(ENU)
```

この Query は、JOB フィールドを次のような順序にします。

Clerk
mgr
Mgr
Mgr
MGR
sales
Sales
Sales
Sales
SALES

例 4: DDS を用いないキー順アクセス・パスの指定: 同順位分類順序を使用したレコードの配列

144 ページの表 11 の STAFF ファイルを使用して、固有分類順序で JOB フィールド値によりレコードを処理するには、次のように指定してください。

```
OPNQRYF FILE(STAFF) KEYFLD(JOB) SRTSEQ(*LANGIDSHR) LANGID(ENU)
```

この Query の結果は、例 3 と同様の結果になります。*mgr* と *sales* 項目は任意の順序になります。大文字と小文字が同等に扱われるからです。つまり、共用順位分類順序は mgr、Mgr、および MGR を等価として扱います。同様に、sales、Sales、および SALES は等価として扱われます。

異なるファイルにあるキー・フィールドの指定: 結合論理ファイルに対する動的キー順アクセス・パスを使用すると、異なる物理ファイルのキーを使用できる処理順序を指定することができます (DDS では、基本ファイルのキーしか使用できないという制約があります)。

指定は前の方法の場合と同じです。アクセス・パスは、どのキー・フィールドが必要であっても、そのキー・フィールドを使用して指定されます。キー・フィールドが入っている物理ファイルに対する制約はありません。しかし、キー・フィールドが結合仕様の基本ファイル以外のファイルにある場合には、システムが結合レコードを一時的にコピーしなければなりません。さらに、システムは、Query ファイルをオープンする前に、コピーされたレコードに対するキー順アクセス・パスを作成しなければなりません。キー・フィールドは、FORMAT パラメーターで指定した様式になければなりません。

例については、『例: 異なるファイルにあるキー・フィールドの指定』を参照してください。

例: 異なるファイルにあるキー・フィールドの指定: 2 次ファイル中のフィールドをキー・フィールドとして使用

JOINLF という名前の結合論理ファイルがすでにあるとします。FILEX が基本ファイルとして指定され、FILEY に結合されています。FILEY にある *Descrp* フィールドを使用して、JOINLF 中のレコードを処理したいとします。

ファイルのレコード様式には、以下のフィールドが入っているとします。

FILEX	FILEY	JOINLF
Item	Item	Item
Qty	Descrp	Qty
		Descrp

以下のように指定することができます。

```
OVRDBF    FILE(JOINLF) SHARE(*YES)
OPNQRYF   FILE(JOINLF) KEYFLD(DESCRP)
CALL      PGM(PGMC)
CLOF      OPNID(JOINLF)
DLTOVR    FILE(JOINLF)
```

Descrp の中の *Qty* に従ってレコードを編成したい場合には (*Descrp* は基本キー・フィールドであり、*Qty* は 2 次キー・フィールドです)、以下のように指定することができます。

```
OPNQRYF   FILE(JOINLF) KEYFLD(DESCRP QTY)
```

DDS を用いないデータベース・ファイルの動的結合: 動的結合機能を用いると、最初に DDS を指定して結合論理ファイルを作成しなくても、ファイルを結合することができます。結合のためのレコード様式を指定するためには、 Query ファイルのオープン (OPNQRYF) コマンドで FORMAT パラメーターを使用しなければなりません。結合論理ファイルおよびビューを含め、どのような物理ファイルまたは論理ファイルでも結合することができます (DDS では論理ファイルを結合することはできません)。キー順アクセス・パスまたは到着順アクセス・パスのどちらかを指定することができます。キーを指定する場合には、結合に含まれるどのファイルからのものであってもかまいません (DDS では、キーは基本ファイルだけに制限されています)。

以下の例では、FORMAT パラメーターに指定されているファイルが作成されたものと仮定します。通常は、外部記述データ定義を使用できるように、処理プログラムを作成する前にファイルを作成する必要があります。

以下のすべての例では、結合順序 (JORDER) パラメーターの省略時値が使用されます。JORDER パラメーターの省略時値は *ANY で、これは、ファイルを結合する順序がシステムによって判別されることを示します。すなわち、どのファイルを基本ファイルとして使用し、どのファイルを 2 次ファイルとして使用するかをシステムが判別することになります。これによって、システムは、結合機能のパフォーマンスの向上を試みることができます。

結合基準はレコード選択基準と同様、指定された分類順序 (SRTSEQ) と言語識別コード (LANGID) によって影響を受けます (144 ページの『例 11: OPNQRYF コマンドを使用したレコードの選択』を参照)。

例については、以下のトピックを参照してください。

- 『例 1: DDS を用いないデータベース・ファイルの動的結合』
- 149 ページの『例 2: DDS を用いないデータベース・ファイルの動的結合』
- 150 ページの『例 3: DDS を用いないデータベース・ファイルの動的結合』

例 1: DDS を用いないデータベース・ファイルの動的結合: ファイルの動的結合

FILEA と FILEB を結合したいとします。ファイルには以下のフィールドが含まれているものとします。

FILEA	FILEB	JOINAB
Cust	Cust	Cust

FILEA	FILEB	JOINAB
Name	Amt	Name
Addr		Amt

結合フィールドは、両方のファイルに存在する *Cust* です。結合ファイルについての Query ファイルのオープン (OPNQRYF) コマンドでは、任意のレコード様式名を指定することができます。ファイルはメンバーを必要としません。レコードはキー順である必要はありません。

以下のように指定することができます。

```
OVRDBF     FILE(JOINAB) TOFILE(FILEA) SHARE(*YES)
OPNQRYF    FILE(FILEA FILEB) FORMAT(JOINAB) +
             JFLD((FILEA/CUST FILEB/CUST)) +
             MAPFLD((CUST 'FILEA/CUST'))
CALL       PGM(PGME) /* Created using file JOINAB as input */
CLOF       OPNID(FILEA)
DLTOVR     FILE(JOINAB)
```

ファイル JOINAB は、データのない物理ファイルです。これは、Query ファイルのオープン (OPNQRYF) コマンドの FORMAT パラメーターで指定されるレコード様式を含むファイルです。

データベース・ファイルによる一時変更 (OVRDBF) コマンドの TOFILE パラメーターは、結合操作の基本ファイル (OPNQRYF コマンドの FILE パラメーターに指定する最初のファイル) の名前を指定する点に注意してください。この例では、Query ファイルのオープン (OPNQRYF) コマンドの FILE パラメーターには、結合される順序 (A から B へ) でファイルが指定されています。ファイルの様式は JOINAB ファイルに含まれています。

JFLD パラメーターは、FILEA の *Cust* フィールドを FILEB の *Cust* フィールドに結合することを指定しています。*Cust* フィールドは、結合されるすべてのレコード様式に対して固有ではないので、JFLD パラメーターで修飾しなければなりません。Query ファイルのオープン (OPNQRYF) コマンドに JFLD パラメーターを指定しない場合でも、システムが最も効果的な値を判別しようとすることがあります。たとえば、前の例を使用して、以下のように指定したとします。

```
OPNQRYF    FILE(FILEA FILEB) FORMAT(JOINAB) +
             QRYSLT('FILEA/CUST *EQ FILEB/CUST') +
             MAPFLD((CUST 'FILEA/CUST'))
```

システムは、QRYSLT パラメーターに値が指定されているので、 *Cust* フィールドを使用して FILEA と FILEB を結合します。この例では、コマンドに JFLD パラメーターが指定されていない点に注意してください。しかし、OPNQRYF コマンドに JDFTVAL(*ONLYDFT) または JDFTVAL(*YES) のどちらかを指定した場合には、JFLD パラメーターを指定しなければなりません。

Query ファイルのオープン (OPNQRYF) コマンドでは、ファイル JOINAB のレコード様式内の *Cust* フィールドへのデータとして使用されるファイルを記述するために、 MAPFLD パラメーターが必要です。MAPFLD パラメーターにフィールドが定義されている場合には、 OPNQRYF コマンドの他の場所では、どこでもそのフィールドの非修飾名 (この例ではファイル名の指定のない *Cust* フィールド) を使用することができます。*Cust* フィールドが MAPFLDパラメーターで定義されているので、JFLD パラメーターの最初の値は修飾する必要がありません。たとえば、以下のように指定しても同じ結果が得られます。

```
JFLD((CUST FILEB/CUST)) +
MAPFLD((CUST 'FILEA/CUST'))
```

これ以外に、MAPFLD パラメーターで定義されたファイル以外のファイルからのフィールドを示すために Query ファイル・オープン (OPNQRYF) コマンドで同じフィールド名を使用する場合には、フィールド名をファイル名で修飾しなければなりません。

KEYFLD パラメーターが指定されていないので、レコードは、 Query ファイルのオープン (OPNQRYF) コマンドでレコードが選択される方法に従い、任意の順序で配列されます。基本ファイルと同じにレコードを編成することをシステムに強制することができます。そのためには、KEYFLD パラメーターに *FILE を指定してください。この指定は、基本ファイルが到着順の場合にも行うことができます。

Query ファイルのオープン (OPNQRYF) コマンドでは、 2 次ファイルからレコードの 1 つが脱落している場合にシステムの取るべき処置を記述するために、JDFTVAL パラメーター (DDS の JDFTVAL キーワードと類似した) を指定することもできます。この例では、JDFTVAL パラメーターが指定されていないので、両方のファイルに存在するレコードだけが選択されます。

Query の結果の改善を (OPNQRYF コマンドのパラメーターを介して) システムに指示した場合には、システムは、通常、レコード数の最も少ないファイルを基本ファイルとして使用しようとします。しかし同時に、システムは、一時ファイルの作成を避けようともします。

JORDER(*FILE) を指定することによって、 Query ファイルのオープン (OPNQRYF) コマンドの FILE パラメーターで指定した結合のファイル順序に従うよう、システムを強制することができます。JDFTVAL(*YES) または JDFTVAL(*ONLYDFT) を指定した場合には、別の順序では別の結果となるため、システムはファイル結合順序を変更しません。

例 2: DDS を用いないデータベース・ファイルの動的結合: 2 次ファイル・レコードのあるファイルだけの読み取り

ファイル FILEAB、FILECD、FILEEF を結合して、2 次ファイル中のレコードと合致するレコードだけを選択したいとします。ファイル JOINF を定義し、使用する様式を記述します。ファイルのレコード様式には以下のフィールドが入っているとします。

FILEAB	**FILECD**	**FILEEF**	**JOINF**
Abitm	Cditm	Efitm	Abitm
Abord	Cddscp	Efcolr	Abord
Abdat	Cdcolr	Efqty	Cddscp
			Cdcolr
			Efqty

この場合、結合ファイルを構成するファイルのフィールド名はすべて、2 文字の接頭部 (該当ファイルのすべてのフィールドに共通) で始まり、すべてのファイルに共通の接尾部で終わっています (たとえば、*xxitm*)。このようにすると、すべてのフィールド名が固有となり、修飾の必要がなくなります。

xxitm フィールドは、FILEAB から FILECD への結合を可能にします。2 つのフィールド *xxitm* と *xxcolr* によって、FILECD から FILEEF への結合が可能となります。これらのファイルには、キー順アクセス・パスの存在は必要ありません。しかし、システムは可能な限り、レコードの編成および選択に既存のアクセス・パスを使用しようとするので、キー順アクセス・パスが存在している場合には、パフォーマンスが大幅に向上します。アクセス・パスが存在しない場合には、システムが自動的に作成し、ファイルがオープンされている間は保守されます。

```
OVRDBF     FILE(JOINF) TOFILE(FILEAB) SHARE(*YES)
OPNQRYF    FILE(FILEAB FILECD FILEEF) +
             FORMAT(JOINF) +
             JFLD((ABITM CDITM)(CDITM EFITM) +
```

```
            (CDCOLR EFCOLR))
CALL        PGM(PGME) /* Created using file JOINF as input */
CLOF        OPNID(FILEAB)
DLTOVR      FILE(JOINF)
```

結合フィールドの対は、上の例と同じ順序で指定する必要はありません。たとえば、JFLD パラメーターの値を以下のように指定しても、同じ結果が得られます。

```
JFLD((CDCOLR EFCOLR)(ABITM CDITM) (CDITM EFITM))
```

結合フィールドの各対の属性は、同一である必要はありません。文字フィールドの通常の埋め込みおよび数字フィールドの小数点の桁合せは自動的に行われます。

JDFTVAL パラメーターは指定されないので、 *NO が想定され、結合レコードを構成するために省略時値は使用されません。JDFTVAL(*YES) を指定し、ファイル FILEAB のレコードと同じ結合フィールド値を持つレコードが FILECD にない場合には、ファイル FILEEF に結合するために *Cddscp* および *Cdcolr* に省略時が使用されます。これらの省略時によって、ファイル FILEEF に一致するレコードが見つかることがあります (省略時値が 2 次ファイル中のレコードに一致するかどうかによって)。見つからない場合には、これらのファイルおよび *Efqty* フィールドに対して省略時値が表示されます。

例 3: DDS を用いないデータベース・ファイルの動的結合: 結合フィールドとしてのマップ・フィールドの使用

MAPFLD パラメーターで定義したフィールドを、結合フィールドの対のいずれか 1 つに使用することができます。これは、2 次ファイル中のキーが単一フィールド (たとえば、6 文字の日付フィールド) として定義されており、基本ファイルの中に同じ情報 (たとえば、月、日、および年) を示す別個のフィールドがあるときに役立ちます。FILEA に文字フィールド *Year*、*Month*、および *Day* があり、このファイルを、YYMMDD 形式の *Date* フィールドがある FILEB に結合したいとします。希望する様式でファイル JOINAB を定義しているとします。以下のように指定することができます。

```
OVRDBF      FILE(JOINAB) TOFILE(FILEA) SHARE(*YES)
OPNQRYF     FILE(FILEA FILEB) FORMAT(JOINAB) +
               JFLD((YYMMDD FILEB/DATE)) +
               MAPFLD((YYMMDD 'YEAR *CAT MONTH *CAT DAY'))
CALL        PGM(PGME) /* Created using file JOINAB as input */
CLOF        OPNID(FILEA)
DLTOVR      FILE(JOINAB)
```

MAPFLD パラメーターでは、*YYMMDD* フィールドが FILEA からのいくつかのフィールドの連結として定義されています。*YYMMDD* フィールドの属性 (たとえば、長さ、タイプなど) は、システムが式から計算するので、MAPFLD パラメーターで指定する必要はありません。

2 次結合ファイルから欠落しているレコードの処理: ユーザーは、 2 次ファイル内の脱落レコードに関する省略時 (結合論理ファイルの場合の JDFTVAL DDS キーワードのような) が認められるかどうかを制御することができます。また、ユーザーは、省略時を持つレコードだけが処理されるように指定することもできます。これにより、2 次ファイルに脱落レコードがあるレコードだけを選択することができます。

例については、『例: 2 次結合ファイルから欠落しているレコードの処理』を参照してください。

例: 2 次結合ファイルから欠落しているレコードの処理: 2 次ファイルの中にレコードを持たない基本ファイルからのレコードの読み取り

147 ページの『例 1: DDS を用いないデータベース・ファイルの動的結合』 においては、JDFTVAL パラメーターが指定されていないので、読み取られるレコードは、FILEA から FILEB への結合が正しく行わ

れたものだけです。FILEB に一致するものを持たない FILEA のレコードのリストが必要な場合、以下の例に示すように、JDFTVAL パラメーターで *ONLYDFT を指定することができます。

```
OVRDBF     FILE(FILEA) SHARE(*YES)
OPNQRYF    FILE(FILEA FILEB) FORMAT(FILEA) +
              JFLD((CUST FILEB/CUST)) +
              MAPFLD((CUST 'FILEA/CUST')) +
              JDFTVAL(*ONLYDFT)
CALL       PGM(PGME) /* Created using file FILEA as input */
CLOF       OPNID(FILEA)
DLTOVR     FILE(FILEA)
```

JDFTVAL(*ONLYDFT) を指定すると、2 次ファイル (FILEB) に等価レコードがないときにだけ、レコードがプログラムに戻されます。

FILEB のフィールドに対する結合操作によって返される値は、いずれも省略時値であるため、FILEA の様式だけを使用するのが通常の方法です。表示されるレコードは、FILEB に一致するもののないレコードです。FILE パラメーターで複数のファイルを記述する場合には、常に FORMAT パラメーターが必須ですが、ファイル名には、FILE パラメーターで指定したファイルのうちいずれでも指定することができます。プログラムは FILEA を使用して作成されます。

逆に、FILEB にレコードがあって、FILEA に一致するもののないすべてのレコードのリストを受け取ることもできます。そのためには、すべての指定で 2 次ファイルを基本ファイルにします。以下のように指定できます。

```
OVRDBF     FILE(FILEB) SHARE(*YES)
OPNQRYF    FILE(FILEB FILEA) FORMAT(FILEB) JFLD((CUST FILEA/CUST)) +
              MAPFLD((CUST 'FILEB/CUST')) JDFTVAL(*ONLYDFT)
CALL       PGM(PGMF) /* Created using file FILEB as input */
CLOF       OPNID(FILEB)
DLTOVR     FILE(FILEB)
```

注: この例のデータベース・ファイルによる一時変更 (OVRDBF) コマンドでは、 FILE(FILEB) が使用されています。これは、OPNQRYF コマンドの FILE パラメーターで最初のファイルを指定しなければならないからです。ファイルのクローズ (CLOF) コマンドでも FILEB が指定されています。JFLD パラメーターと MAPFLD パラメーターも変更されています。プログラムは FILEB を使用して作成されます。

固有キーの処理: 固有キー処理を行うと、グループの最初のレコードだけを処理することができます。グループは、同じキー値のセットを持つ 1 つまたは複数のレコードを用いて定義されます。最初のレコードを処理するということは、受け取るレコードが固有キーを持っていることを意味します。

固有キー処理を使用する場合には、ファイルを順次にしか読み取ることができません。キー・フィールドは、指定された分類順序 (SRTSEQ) と言語識別コード (LANGID) に従って分類されます (146 ページの『例 3: DDS を用いないキー順アクセス・パスの指定』 と 146 ページの『例 4: DDS を用いないキー順アクセス・パスの指定』を参照してください)。

固有キー処理を指定したときに、実際にはファイルが重複キーを持っている場合、同じキー値を持つレコードのそれぞれのグループから受け取れるのは 1 つのレコードだけです。

例については、以下のトピックを参照してください。

- 『例 1: 固有キーの処理』
- 152 ページの『例 2: 固有キーの処理』

例 1: 固有キーの処理: 固有キー・レコードだけの読み取り

Cust フィールドに重複キーを持つレコードがある FILEA を処理したいとします。*Cust* フィールドのそれぞれの固有値に対して最初のレコードだけをプログラム PGMF で処理したいとします。以下のように指定することができます。

```
OVRDBF     FILE(FILEA) SHARE(*YES)
OPNQRYF    FILE(FILEA) KEYFLD(CUST) UNIQUEKEY(*ALL)
CALL       PGM(PGMF)
CLOF       OPNID(FILEA)
DLTOVR     FILE(FILEA)
```

例 2: 固有キーの処理: キー・フィールドの一部だけを使用するレコードの読み取り

同じファイルを *Slsman*、*Cust*、*Date* の順序で処理しますが、 *Slsman* および *Cust* ごとに 1 つのレコードだけが必要であるとします。ファイル中のレコードは、以下のようになっているとします。

Slsman	Cust	Date	Record #
01	5000	880109	1
01	5000	880115	2
01	4025	880103	3
01	4025	880101	4
02	3000	880101	5

最初のキー・フィールドから始めて、固有のキー・フィールドの数を指定します。

```
OVRDBF     FILE(FILEA) SHARE(*YES)
OPNQRYF    FILE(FILEA) KEYFLD(SLSMAN CUST DATE) UNIQUEKEY(2)
CALL       PGM(PGMD)
CLOF       OPNID(FILEA)
DLTOVR     FILE(FILEA)
```

以下のレコードがプログラムによって取り出されます。

Slsman	Cust	Date	Record #
01	4025	880101	4
01	5000	880109	1
02	3000	880101	5

注: 複数の NULL は等価として扱われます。したがって、最初の NULL だけが戻されます。

既存のフィールド定義から抽出されるフィールドの定義: マップ・フィールド定義を使用すると、以下のことが行えます。

- 選択値を指定する内部フィールドを作成する (139 ページの『例 7: OPNQRYF コマンドを使用したレコードの選択』を参照)。
- 複数のファイルに同じフィールド名が存在する場合に混同を避ける (147 ページの『例 1: DDS を用いないデータベース・ファイルの動的結合』を参照)。
- 処理したい様式にだけ存在し、データベース自体には存在しないフィールドを作成する。このようにすると、変換、サブストリング、連結、および複雑な数学操作を行うことができます。以下の例において、この機能を説明しています。

処理対象の様式にだけ存在するフィールドを作成する例については、以下のトピックを参照してください。

- 153 ページの『例 1: 既存のフィールド定義から抽出されるフィールドの定義』
- 153 ページの『例 2: 既存のフィールド定義から抽出されるフィールドの定義』
- 154 ページの『例 3: 既存のフィールド定義から抽出されるフィールドの定義』

例 1: 既存のフィールド定義から抽出されるフィールドの定義: 抽出されたフィールドの使用

レコード様式に *Price* および *Qty* フィールドがあるとします。Query ファイルのオープン (OPNQRYF) コマンドを使用して 1 つのフィールドと他のフィールドを乗算し、抽出されたフィールド *Exten* を作成することができます。FILEA を処理する必要があり、すでに FILEAA を作成しているとします。ファイルのレコード様式には以下のフィールドが入っているとします。

FILEA	**FILEAA**
Order	Order
Item	Item
Qty	Exten
Price	Brfdsc
Descrp	

Exten フィールドはマップ・フィールドです。その値は、*Price* を *Qty* 倍することによって求められます。新しい様式に *Qty* または *Price* のどちらかがなければならないということはありませんが、望む場合にはその様式にすることもできます。*Brfdsc* フィールドは、*Descrp* フィールドの簡単な説明です (最初の 10 文字を使用します)。

PGMF で新しい様式を処理することを指定したとします。このプログラムを作成するためには、読み取るファイルとして FILEAA を使用します。以下のように指定することができます。

```
OVRDBF     FILE(FILEAA) TOFILE(FILEA) SHARE(*YES)
OPNQRYF    FILE(FILEA) FORMAT(FILEAA) +
              MAPFLD((EXTEN 'PRICE * QTY') +
              (BRFDSC 'DESCRP'))
CALL       PGM(PGMF) /* Created using file FILEAA as input */
CLOF       OPNID(FILEA)
DLTOVR     FILE(FILEAA)
```

Exten フィールドの属性は、FILEAA のレコード様式で定義されている属性である点に注意してください。フィールドの計算された値が大きすぎる場合は、例外がプログラムに送られます。

フィールドの最初の数文字だけが必要な場合には、サブストリング関数を使用して *Brfdsc* フィールドへマップする必要はありません。*Brfdsc* フィールドの長さは、FILEAA レコード様式の中で定義されています。

FORMAT パラメーターで指定した様式の中のすべてのフィールドは、 OPNQRYF コマンドで記述しなければなりません。すなわち、出力様式の中のすべてのフィールドは、FILE パラメーターで指定したファイルのレコード様式の 1 つに存在しているか、または MAPFLD パラメーターで定義されていなければなりません。FORMAT パラメーターで指定された様式に、プログラムで使用されないフィールドがある場合には、 MAPFLD パラメーターを使用して、そのフィールドにゼロまたはブランクを入れることができます。*Fldc* フィールドは文字フィールドであり、*Fldn* フィールドは出力様式の数値フィールドであって、プログラムではどちらの値も使用しないこととします。以下のように指定することによって、OPNQRYF コマンドでのエラーを防ぐことができます。

```
MAPFLD((FLDC ' " " ')(FLDN 0))
```

引用符でブランク値を囲んでいる点に注意してください。使用しないフィールドの定義に定数を使用することによって、OPNQRYF コマンドの使用のたびに固有の様式を作成する手間を省くことができます。

例 2: 既存のフィールド定義から抽出されるフィールドの定義: 組み込み関数の使用

FILEA の中の *Fldm* フィールドの正弦である数学関数を計算したいとします。まず、以下のフィールドを含むレコード様式のあるファイル (FILEAA と呼ぶことにします) を作成します。

FILEA	**FILEAA**
Code	Code
Fldm	Fldm
	Sinm

その後で、FILEAA を入力として使用するプログラム (PGMF) を作成して以下のように指定することができます。

```
OVRDBF     FILE(FILEAA) TOFILE(FILEA) SHARE(*YES)
OPNQRYF    FILE(FILEA) FORMAT(FILEAA) +
             MAPFLD((SINM '%SIN(FLDM)'))
CALL       PGM(PGMF) /* Created using file FILEAA as input */
CLOF       OPNID(FILEA)
DLTOVR     FILE(FILEAA)
```

組み込み関数 %SIN は、この関数の引き数として指定されたフィールドの正弦を計算します。*Sinm* フィールドは、FORMAT パラメーターに指定した様式の中で定義されているので、OPNQRYF コマンドは正弦値の内部定義 (浮動小数点形式で) を *Sinm* フィールドの定義に変換します。この手法を使用すると、浮動小数点フィールドの用法に関する高水準言語の制約を避けることができます。たとえば、*Sinm* フィールドをパック 10 進数フィールドとして定義した場合には、値が浮動小数点フィールドを使用して計算されている場合でも、PGMF の作成にはどの高水準言語でも使用することができます。

正弦の他にも、使用できる関数は数多くあります。組み込み関数の詳細なリストについては、 制御言語 (CL) に関するトピックの OPNQRYF コマンドを参照してください。

例 3: 既存のフィールド定義から抽出されるフィールドの定義: 抽出フィールドおよび組み込み関数の使用

前の例で、FILEA に *Fldx* というフィールドも存在し、この *Fldx* フィールドが、*Fldm* フィールドの正弦を保持するのに適切な属性を持っているとします。また、*Fldx* フィールドの内容は使用しないものとします。MAPFLD パラメーターを使用して、フィールドの内容が高水準言語プログラムに渡される前に、その内容を変更することができます。たとえば、以下のように指定することができます。

```
OVRDBF     FILE(FILEA) SHARE(*YES)
OPNQRYF    FILE(FILEA) MAPFLD((FLDX '%SIN(FLDM)'))
CALL       PGM(PGMF) /* Created using file FILEA as input */
CLOF       OPNID(FILEA)
DLTOVR     FILE(FILEA)
```

この場合、FORMAT パラメーターで別のレコード様式を指定する必要はありません。(省略時では、FILE パラメーターの最初のファイルの様式が使用されます。) したがって、プログラムは FILEA を使用して作成されます。この手法を使用するときには、定義し直すフィールドが、計算された値を正しく処理することができるような属性を持つようにしなければなりません。最も簡単な方法は、各 Query ごとに処理したい特定のフィールドを持つ別個のファイルを作成することです。

この手法をマップ・フィールド定義および %XLATE 関数で使用して、フィールドがデータベースの中にあるときとは異なるようにプログラムに見えるように、フィールドを変換することができます。たとえば、小文字フィールドを変換して、プログラムには大文字だけが見えるようにすることができます。

分類順序と言語識別コードは、 %MIN および %MAX 組み込み関数の結果に影響を与えることがあります。たとえば、文字の大文字と小文字のバージョンは、選択された分類順序および言語識別コードに応じ

て、等価であったりなかったりします。変換されたフィールド値は、最小と最大を判別するために使用されますが、変換されなかった値は結果レコードに返却されることに注意してください。

ここの例では、FILEA は入力ファイルとして使用されています。OPNQRYF コマンドを使用してデータを更新することもできます。しかし、マップ・フィールド定義を使用してフィールドを変更する場合には、フィールドに対する更新は無視されます。

ゼロによる除算の処理: ゼロによる除算は、Query ファイルのオープン (OPNQRYF) コマンドではエラーとみなされます。

通常、レコード選択はフィールド・マッピング・エラーが発生 (たとえば、フィールド・マッピングによる除算エラーが発生) する前に行われます。したがって、ゼロによる除算エラーの原因となったレコードを (QRYSLT パラメーター値およびレコード内の有効データに基づいて) 除外することができます。こういった場合、レコードが除外されて、OPNQRYF コマンドによる処理が続行されます。

答えをゼロにしたい場合には、通常の商業データに実際に使用できる以下のような方法があります。

A を B で除算して C を得たいとします (式で表すと、A / B = C となります)。以下の定義で、B はゼロの場合があるとします。

フィールド	桁数	10 進数
A	6	2
B	3	0
C	6	2

以下の算法を使用することができます。

```
(A * B) / %MAX((B * B) .nnnn1)
```

%MAX 関数は、B * B または最小値のうち、大きい方を返します。最小値には十分な桁数のゼロを先行させ、B がゼロでない限り B * B で計算されるすべての値よりも小さくなるようにしなければなりません。この例では、B の小数点以下の桁数がゼロなので、.1 を使用できます。先行ゼロの数は、B の小数点以下の桁数の 2 倍にする必要があります。たとえば、B の小数点以下の桁数が 2 の場合には、.00001 を使用します。

以下の MAPFLD 定義を指定します。

```
MAPFLD((C '(A * B) / %MAX((B * B) .1)'))
```

最初の乗算の目的は、B がゼロの場合に被除数をゼロにすることです。このようにすると、除算が行われたときに、結果がゼロになります。B がゼロの場合には除数として .1 が使用されるので、ゼロによる除算は起こりません。

データベース・ファイル・レコードからのデータの集約 (グループ化): グループ処理機能を使用すると、既存のデータベース・レコードからのデータを集約することができます。以下のように指定することができます。

- グループ化フィールド
- グループ化前とグループ化後の両方の選択値
- 新しいレコードに対するキー順アクセス・パス
- マップ・フィールド定義。これによって、合計、平均、標準偏差、および分散を求めたり、各グループ中のレコードをカウントしたりすることができます。

- フィールド値をグループ化するための重みを与える分類順序と言語識別コード

通常、最初に、以下のタイプのフィールドだけを含むレコード様式を持つファイルを作成します。

- グループ化フィールド。GRPFLD パラメーターに指定するものであり、グループを定義します。各グループには、すべてのグループ化フィールドの値の一定セットが含まれます。グループ化フィールドは、FORMAT パラメーターで指定されるレコード様式には入れる必要はありません。
- 集約フィールド。以下の組み込み関数のうち 1 つまたは複数を指定した MAPFLD パラメーターで定義します。

  %COUNT
  グループ内のレコードのカウント

  %SUM
  グループ内の該当フィールドの値の合計

  %AVG
  グループ内の該当フィールドの値の算術平均

  %MAX
  グループ内での該当フィールドの最大値

  %MIN
  グループ内での該当フィールドの最小値

  %STDDEV
  グループ内の該当フィールドの標準偏差

  %VAR
  グループ内の該当フィールドの分散
- 定数フィールド。フィールド値の中に定数を入れるために使用できます。Query ファイルのオープン (OPNQRYF) コマンドが出力様式中のすべてのフィールドを認識していなければならないという制約は、グループ化機能にもあてはまります。

グループ処理を使用する場合は、ファイルを順次にしか読み取ることができません。

グループ処理の例については、 『例: データベース・ファイル・レコードからのデータの集約 (グループ化)』を参照してください。

例: データベース・ファイル・レコードからのデータの集約 (グループ化): グループ処理の使用

得意先番号によってデータをグループ化し、金額フィールドを分析したいとします。FILEA というデータベース・ファイルがあり、以下のようなフィールドを持つレコード様式を含む FILEAA というファイルを作成するものとします。

FILEA	FILEAA
Cust	Cust
Type	Count (得意先ごとのレコードのカウント)
Amt	Amtsum (金額フィールドの合計)
	Amtavg (金額フィールドの平均)
	Amtmax (金額フィールドの最大値)

新しいファイルでフィールドを定義する場合には、結果を保持するのに十分な大きさを確保しなければなりません。たとえば、*Amt* フィールドを 5 桁と定義した場合には、*Amtsum* フィールドは 7 桁として定義します。算術オーバーフローは、いずれもプログラムの異常終了の原因となります。

FILEA のレコードの値は以下のとおりであるとします。

Cust	Type	Amt
001	A	500.00
001	B	700.00
004	A	100.00
002	A	1200.00
003	B	900.00
001	A	300.00
004	A	300.00
003	B	600.00

次に、レコードを印刷するために、FILEAA を入力として使用してプログラム (PGMG) を作成します。

```
OVRDBF     FILE(FILEAA) TOFILE(FILEA) SHARE(*YES)
OPNQRYF    FILE(FILEA) FORMAT(FILEAA) KEYFLD(CUST) +
              GRPFLD(CUST) MAPFLD((COUNT '%COUNT') +
                (AMTSUM '%SUM(AMT)') +
                (AMTAVG '%AVG(AMT)') +
                (AMTMAX '%MAX(AMT)'))
CALL       PGM(PGMG) /* Created using file FILEAA as input */
CLOF       OPNID(FILEA)
DLTOVR     FILE(FILEAA)
```

プログラムによって取り出されるレコードは以下のようになります。

Cust	Count	Amtsum	Amtavg	Amtmax
001	3	1500.00	500.00	700.00
002	1	1200.00	1200.00	1200.00
003	2	1500.00	750.00	900.00
004	2	400.00	200.00	300.00

注: GRPFLD パラメーターを指定した場合は、グループが昇順で表示されないことがあります。特定の順序を確実にするためには、KEYFLD パラメーターを使用してください。

この例の中で、*Amtsum* の値が 700.00 よりも大きい集約レコードだけを印刷したいとします。*Amtsum* フィールドは特定の得意先に関する集計フィールドであるため、GRPSLT パラメーターを使用してグループ化後の選択を指定しなければなりません。以下の GRPSLT パラメーターを追加します。

```
GRPSLT('AMTSUM *GT 700.00')
```

プログラムによって取り出されるレコードは、以下のとおりです。

Cust	Count	Amtsum	Amtavg	Amtmax
001	3	1500.00	500.00	700.00
002	1	1200.00	1200.00	1200.00
003	2	1500.00	750.00	900.00

Query ファイルのオープン (OPNQRYF) コマンドでは、グループ化前の選択 (QRYSLT パラメーター) とグループ化後の選択 (GRPSLT パラメーター) の両方がサポートされます。

Type フィールドが A と等しい、追加の得意先レコードを選択したいとします。*Type* はファイル FILEA のレコード様式内のフィールドであり、集約フィールドではないため、次のように QRYSLT ステートメントを追加してグループ化前の選択を指定します。

```
QRYSLT('TYPE *EQ "A" ')
```

選択に使用するフィールドは、プログラムによって処理される様式中に表示される必要がない点に注意してください。

プログラムによって取り出されるレコードは、以下のとおりです。

Cust	Count	Amtsum	Amtavg	Amtmax
001	2	800.00	400.00	500.00
002	1	1200.00	1200.00	1200.00

グループ化が行われる前に選択されるため、CUST 001 の値が変わっていることに注意してください。

前の QRYSLT パラメーターの値に加えて、出力を *Amtavg* の降順に編成したいとします。これには、以下のように、OPNQRYF コマンドの KEYFLD パラメーターを変更します。

```
KEYFLD((AMTAVG *DESCEND))
```

プログラムによって取り出されるレコードは、以下のとおりです。

Cust	Count	Amtsum	Amtavg	Amtmax
002	1	1200.00	1200.00	1200.00
001	2	800.00	400.00	500.00

最終合計のみの処理: 最終合計のみの処理は、グループ化フィールドの指定が不要な、グループ化の特殊な形式です。レコードは 1 つだけ出力されます。グループ化用の特殊な組み込み関数は、すべて指定することができます。最終合計を構成するレコードの選択も指定することができます。

最終合計のみの処理の例については、以下のトピックを参照してください。

- 『例 1: 最終合計のみの処理』
- 159 ページの『例 2: 最終合計のみの処理』
- 159 ページの『例 3: 最終合計のみの処理』

例 1: 最終合計のみの処理: 単純な合計処理

FILEA というデータベース・ファイルがあり、以下のような最終合計レコード用のファイル FINTOT を作成したいとします。

FILEA	FINTOT
Code	Count (選択されたすべてのレコードのカウント)
Amt	Totamt (金額フィールドの合計)
	Maxamt (金額フィールドの最大値)

FINTOT ファイルは、最終合計で作成される単一レコードを保存するという特別の目的で作成されます。以下のように指定できます。

```
OVRDBF     FILE(FINTOT) TOFILE(FILEA) SHARE(*YES)
OPNQRYF    FILE(FILEA) FORMAT(FINTOT) +
              MAPFLD((COUNT '%COUNT') +
                 (TOTAMT '%SUM(AMT)') (MAXAMT '%MAX(AMT)'))
CALL       PGM(PGMG) /* Created using file FINTOT as input */
CLOF       OPNID(FILEA)
DLTOVR     FILE(FINTOT)
```

例 2: 最終合計のみの処理: レコード選択を伴う合計のみの処理

前の例を変更して、*Code* フィールドが B に等しいレコードだけを最終合計に入れたいとします。以下のように、QRYSLT パラメーターを追加することができます。

```
OVRDBF     FILE(FINTOT) TOFILE(FILEA) SHARE(*YES)
OPNQRYF    FILE(FILEA) FORMAT(FINTOT) +
              QRYSLT('CODE *EQ "B" ') MAPFLD((COUNT '%COUNT') +
                (TOTAMT '%SUM(AMT)') (MAXAMT '%MAX(AMT)'))
CALL       PGM(PGMG) /* Created using file FINTOT as input */
CLOF       OPNID(FILEA)
DLTOVR     FILE(FINTOT)
```

最終合計関数とともに GRPSLT キーワードを使用することができます。指定した GRPSLT 選択値によって、最終合計レコードが受け取られるかどうかが決まります。

例 3: 最終合計のみの処理: 新しいレコード様式を用いた合計のみの処理

CL プログラムで新しいファイル/様式を処理したいとします。ファイルを読み取り、最終合計と一緒にメッセージを送ることにします。以下のように指定することができます。

```
DCLF       FILE(FINTOT)
DCL        &COUNTA *CHAR LEN(7)
DCL        &TOTAMTA *CHAR LEN(9)
OVRDBF     FILE(FINTOT) TOFILE(FILEA) SHARE(*YES)
OPNQRYF    FILE(FILEA) FORMAT(FINTOT) MAPFLD((COUNT '%COUNT') +
                (TOTAMT '%SUM(AMT)'))
RCVF
CLOF       OPNID(FILEA)
CHGVAR     &COUNTA &COUNT
CHGVAR     &TOTAMTA &TOTAMT
SNDPGMMSG  MSG('COUNT=' *CAT &COUNTA *CAT +
               ' Total amount=' *CAT &TOTAMTA);
DLTOVR     FILE(FINTOT)
```

数字フィールドを即時メッセージに含めるには、文字フィールドに変換しなければなりません。

システムが *Query* ファイルのオープン・コマンドを実行する方法の制御: 最適化機能を使用すると、Query の結果の使用方法を指定することができます。

Query ファイルのオープン (OPNQRYF) コマンドを使用する場合、 2 つのステップでパフォーマンスに関して考慮しなければなりません。最初のステップは、OPNQRYF コマンド自体を実際に実行するときです。このステップでは、OPNQRYF で既存のアクセス・パスを使用するのか、またはこの Query 要求のために新しいアクセス・パスを作成するかを決めます。パフォーマンスの考慮が必要な 2 番目のステップは、アプリケーション・プログラムが OPNQRYF の結果を使用してデータを処理するときです。詳細については データベース・パフォーマンスおよび Query 最適化を参照してください。

大半のバッチ・タイプの機能では、通常、上記の 2 つのステップの合計時間だけが問題となります。したがって、OPNQRYF の省略時値は OPTIMIZE(*ALLIO) です。これは、両方のステップに必要な合計時間が OPNQRYF で考慮されることを意味します。

対話式環境で OPNQRYF を使用する際には、ファイル全体を処理する必要がない場合があります。最初の画面に収まるだけのレコードを、できるだけ速く表示したいときなどがこれにあたります。このためには、可能であれば最初のステップではアクセス・パスを作成しない方が望ましいといえます。こういった場合には、OPTIMIZE(*FIRSTIO) を指定します。

複数のプログラムで OPNQRYF の同じ結果を処理したい場合には、最初のステップで効果的なオープン・データ・パス (ODP) を作成する必要があります。すなわち、OPNQRYF コマンドに OPTIMIZE(*MINWAIT) を指定することによって、 2 番目のステップで処理プログラムが読み取る必要のあるレコードの数を最小にする必要があります。

OPNQRYF コマンドの KEYFLD パラメーターまたは GRPFLD パラメーターで、共用するアクセス・パスがないときにアクセス・パスを作成するように要求している場合は、OPTIMIZE の指定が何であるかにかかわらず、アクセス・パスの全体が作成されます。最適化は、主として選択処理に影響を及ぼします。

例については、以下のトピックを参照してください。

- 『例 1: システムが Query ファイルのオープン・コマンドを実行する方法の制御』
- 『例 2: システムが Query ファイルのオープン・コマンドを実行する方法の制御』

例 1: システムが Query ファイルのオープン・コマンドを実行する方法の制御: レコードの最初のセットに関する最適化

対話式ジョブで、操作員が、*Code* フィールドが B に等しいすべてのレコードを要求したとします。プログラムのサブファイルには、1 画面あたり 15 のレコードが入っています。操作員が結果の最初の画面をできるだけ速く受け取るようにするには、以下のように指定することができます。

```
OVRDBF    FILE(FILEA) SHARE(*YES)
OPNQRYF   FILE(FILEA) QRYSLT('CODE = "B" ') +
            SEQONLY(*YES 15) OPTIMIZE(*FIRSTIO)
CALL      PGM(PGMA)
CLOF      OPNID(FILEA)
DLTOVR    FILE(FILEA)
```

システムは、Query の処理を最適化して、*Code* フィールドに対するアクセス・パスがすでに存在するかどうかにかかわらず、Query 全体が完了する前に最初のバッファーをレコードで満たします。

例 2: システムが Query ファイルのオープン・コマンドを実行する方法の制御: 読み取るレコードの数を最小にするための最適化

Query ファイルのオープン (OPNQRYF) コマンドで作成された同じファイルをアクセスする複数のプログラムがあるとします。この場合、アプリケーション・プログラムが関係のあるデータだけを読み取るように、パフォーマンスを最適化したいことでしょう。つまり、できるだけ効率的に選択を実行する OPNQRYF が必要になります。以下のように指定できます。

```
OVRDBF   FILE(FILEA) SHARE(*YES)
OPNQRYF  FILE(FILEA) QRYSLT('CODE *EQ "B"') +
         KEYFLD(CUST) OPTIMIZE(*MINWAIT)
CALL     PGM(PGMA)
POSDBF   OPNID(FILEA) POSITION(*START)
CALL     PGM(PGMB)
CLOF     OPNID(FILEA)
DLTOVR   FILE(FILEA)
```

ファイルの作成および FORMAT パラメーターの使用に関する考慮事項: FILE パラメーターに複数の項目を指定して結合処理を要求する場合には、FORMAT パラメーターにレコード様式名を指定しなければなりません (すなわち、FORMAT(*FILE) を指定することはできません)。また、グループ化機能を使用する場合、あるいは抽出フィールドを定義するために MAPFLD パラメーターに複雑な式を指定する場合にも、通常、レコード様式名を指定します。以下の事項を考慮してください。

- レコード様式名は、ユーザーが選択する任意の名前です。Query の対象とするデータベース・ファイル内の様式名と異なっていてもかまいません。

- フィールド名は、ユーザーが選択する任意の名前です。Query の対象とするデータベース・ファイルの中でフィールド名が固有である場合には、システムは、 Query の対象となるファイルのレコード様式 (FILE パラメーター) および Query の結果のレコード様式 (FORMAT パラメーター) に同じ名前のフィールドがあれば、そのフィールドの値を暗黙のうちにマッピングされます。詳細については、 147 ページの『例 1: DDS を用いないデータベース・ファイルの動的結合』を参照してください。
- フィールド名は固有であるが、属性が FILE パラメーターで指定したファイルと FORMAT パラメーターで指定したファイルとで違っているという場合には、データは暗黙のうちにマップされます。
- MAPFLD パラメーターを使用して抽出フィールドを定義する場合には、正しいフィールド属性を使用しなければなりません。たとえば、グループ化 %SUM 関数を使用する場合には、合計を入れるのに十分な大きさのフィールドを定義しなければなりません。このようにしないと、算術オーバーフローが起こり、プログラムに例外が送られます。
- FORMAT パラメーターで指定されているレコード様式にマップされるすべてのフィールド値に対して小数点位置合せが行われます。Query の結果のレコード様式に小数部のない 5 桁のフィールドがあり、計算された値またはそのフィールドにマップされる値が 0.12345 であるとすると、小数点の右側の桁が切り捨てられるので、フィールドの値は 0 という結果になります。

レコードの配列に関する考慮事項: Query ファイルのオープン (OPNQRYF) コマンドの省略時の処理では、レコードは、パフォーマンスを向上させ、KEYFLD パラメーターに指定されている順序と矛盾しない任意の順序で提供されます。したがって、KEYFLD パラメーターを使用して特定のキー・フィールドまたは KEYFLD(*FILE) を指定しない限り、同じQuery ファイルのオープン (OPNQRYF) コマンドを実行するたびに、プログラムに返されるレコードの順序が異なる可能性があります。

Query ファイルのオープン (OPNQRYF) コマンドに KEYFLD(*FILE) パラメーター・オプションを指定する場合に、ジョブの省略時値または OPNQRYF コマンドの SRTSEQ パラメーターで Query について *HEX 以外の分類順序が指定されていると、実際のファイル順序を反映しない順序でレコードを受け取ることがあります。ファイルがキー順の場合、Query の分類順序はファイルのキー・フィールドに適用され、通知メッセージ CPI431F が送信されます。ファイルの分類順序と代替照合順序表が存在する場合、それらは順序付けの際に無視されます。これにより、すべてのフィールド名をリストしなくても、どのフィールドに分類順序を適用するかを示すことができます。分類順序が Query に指定されていない場合 (たとえば *HEX) には、順序付けはバージョン 2 リリース 3 より前と同様に行われます。

DDM ファイルに関する考慮事項: Query ファイルのオープン (OPNQRYF) コマンドで DDM ファイルを処理することができます。FILE パラメーターに指定するファイルはすべて、同じ IBM iSeries システムまたはシステム/38 受動システムに存在していなければなりません。グループ処理を指定し、DDM ファイルを使用する場合には、起動システムおよび受動システムが同じタイプ (両方ともシステム/38、または両方とも iSeries システム)でなければなりません。

高水準言語プログラムの作成に関する考慮事項: 128 ページの『ファイル内の既存のレコード様式の使用』で説明した方法 (FORMAT パラメーターは除外される) では、データベース・ファイルに直接にアクセスするかのように高水準言語プログラムがコーディングされています。選択や順序付けはプログラムの外部で行われ、プログラムは選択されたデータを指定の順序で受け取ります。プログラムは、選択値から除外されたレコードは受け取りません。選択/除外基準で論理ファイルを処理する場合には、これと同じ機能が働きます。

FORMAT パラメーターを使用する場合には、FORMAT パラメーターで使用したのと同じファイル名をプログラムで指定します。プログラムは、このファイルに実際のデータが入っているかのように作成されています。

ファイルを順次に読み取る場合は、高水準言語はキー・フィールドを無視するよう自動的に指定することができます。通常は、到着順でレコードを読み取るかのようにプログラムを作成します。Query ファイルのオープン (OPNQRYF) コマンドで KEYFLD パラメーターを使用した場合は、診断メッセージが出されますが、このメッセージは無視することができます。

ファイルをキーによってランダムに処理する場合は、高水準言語ではキー仕様が必要になる場合があります。選択値がある場合は、データベースに存在するレコードへのプログラムのアクセスが妨げられることもあります。OPNQRYF コマンドを使用したのか、DDS の選択/除外論理を用いて作成された論理ファイルを使用したのかを問わず、ランダム読み取りでは、レコードが見つかりませんという状態が発生する可能性があります。

場合によっては、マッピング・エラーが原因となった例外 (算術オーバーフローなど) をモニターすることができますが、すべてのフィールドの属性を定義して結果を正しく処理する方が好都合です。

Query ファイルのオープン (OPNQRYF) コマンドの実行中に送られるメッセージ: OPNQRYF コマンドの実行時には、OPNQRYF 要求の状況を対話ユーザーに知らせるメッセージが送られます。たとえば、要求を満たすために OPNQRYF によってキー順アクセス・パスが作成された場合、ユーザーにはメッセージが送られます。OPNQRYF コマンドの実行時に送られる可能性のあるメッセージは、以下のとおりです。

メッセージ識別コード	説明
CPI4301	Query プログラム実行中。
CPI4302	Query プログラム実行中。アクセス・パス作成中。
CPI4303	Query プログラム実行中。ファイルのコピーを作成中。
CPI4304	Query プログラム実行中。選択は完了しました。
CPI4305	Query プログラム実行中。*N のファイル *N のコピー分類中。
CPI4306	Query プログラム実行中。ファイルからアクセス・パス作成中。
CPI4307	Query プログラム実行中。&2 のファイル &1 からハッシュ表作成中。
CPI4011	Query プログラム実行中。レコードが処理されました。

これらの状況メッセージが表示されないようにするには、 CL Programming (CL プログラミング、SD88-5038) のメッセージ処理に関する説明を参照してください。

ジョブがデバッグ・モードで (STRDBG コマンドを使用して) 実行されている場合、または Query オプション・ファイルのオプション DEBUG_MESSAGES *YES を使用して要求された場合は、メッセージがユーザーのジョブ・ログに送られます。これらのメッセージは、OPNQRYF 要求を処理するために使用される実行方法を説明しています。このようなメッセージでは、行われた最適化処理についての情報が提供されます。これらのメッセージは、最高のパフォーマンスが得られるように OPNQRYF 要求を調整するためのツールとして使用することができます。メッセージには、以下のものがあります。

CPI4321
ファイルのアクセス・パスが作成されました。

CPI4322
アクセス・パスがキー付きファイルから作成されました。

CPI4324
ファイルに一時ファイルが作成されました。

CPI4325
Query 用に一時結果ファイルが作成されました。

CPI4326
ファイルが結合位置で処理されました。

CPI4327
ファイルが結合位置 1 で処理されました。

CPI4328
ファイルのアクセス・パスが使用されました。

CPI4329
ファイルに到着順アクセスが使用されました。

CPI432A
Query 最適化はタイムアウトになりました。

CPI432C
ファイルのすべてのアクセス・パスが考慮されました。

CPI432E
選択フィールドが異なる属性にマップされました。

CPI432F
ファイルのアクセス・パスに関する示唆。

CPI433B
Query オプション・ファイルを更新できません。

CPI4330
ファイル &1 の並列 &10 スキャンに &6 タスクが使用されました。

CPI4332
ファイルで作成された並列索引に &6 タスクが使用されました。

CPI4333
結合処理にハッシュ・アルゴリズムが使用されました。

CPI4338
ファイルのビットマップ処理に &1 アクセス・パスが使用されました。

CPI4339
Query オプションがライブラリー &1 内のファイル &2 を検索しました。

CPI4341
配布 Query を実行中。

CPI4342
Query 用の配布結合を実行中。

CPI4345
Query 用の一時配布結果ファイル &4 が構築されました。

CPI4346
&2 の Query 結合ステップ &1 に対する最適化プログラムのデバッグ・メッセージが続きます。

CPI4347
Query は複数ステップで処理されています。

ほとんどのメッセージは、特定のオプションが実行された理由を提供しています。各メッセージの第 2 レベルのテキストには、そのオプションが選択された理由が説明されています。メッセージの中には、OPNQRYF 要求のパフォーマンスを向上させるのに役立つ情報を提供しているものもあります。

入力だけに限定しない Query ファイルのオープン (OPNQRYF) コマンドの使用: OPNQRYF コマンドでは、処理のタイプを決めるために OPTION パラメーターを使用することができます。省略時値は OPTION(*INP) であり、この場合にはファイルは入力専用にオープンされます。OPNQRYF コマンドのその他の OPTION 値および高水準言語プログラムを使用すると、 Query ファイルのオープンによってレコードの追加、更新、または削除を行うこともできます。しかし、UNIQUEKEY、GRPFLD、または GRPSLT パラメーターを指定する場合、集約関数の 1 つを使用する場合、あるいは FILE パラメーターで複数のファイルを指定する場合には、ファイルの使用は入力だけに限定されます。

結合論理ファイルは、入力専用の処理に制限されます。ビューの定義でグループ処理、結合処理、合併処理、区別処理、またはユーザー定義の表関数が指定されている場合、ビューの処理は入力専用に制限されます。 Query 最適化プログラムが Query を実行するために一時ファイルを作成する必要がある場合は、ファイルの使用は入力専用に制限されます。

ファイルの中の一部のレコードについてフィールド値を現在の値から別の値に変更したい場合には、OPNQRYF コマンドと特定の高水準言語プログラムを組み合わせて使用することができます。たとえば、*Flda* フィールドが ABC に等しいすべてのレコードを変更して、*Flda* フィールドが XYZ に等しくなるようにしたいとします。以下のように指定することができます。

```
OVRDBF    FILE(FILEA) SHARE(*YES)
OPNQRYF   FILE(FILEA) OPTION(*ALL) QRYSLT('FLDA *EQ "ABC" ')
CALL      PGM(PGMA)
CLOF      OPNID(FILEA)
DLTOVR    FILE(FILEA)
```

プログラム PGMA は読み取り可能なすべてのレコードを処理しますが、 Query 選択により、*Flda* フィールドが ABC に等しいレコードだけに限定されます。プログラムは、各レコードのフィールドの値を XYZ に変更して、レコードを更新します。

OPNQRYF コマンドを使用して、データベース・ファイルの中のレコードを削除することもできます。レコード中のあるフィールドが X に等しい場合には、そのレコードを削除したいとします。読み取ったレコードをすべて削除するようにプログラムを作成しておき、以下のように OPNQRYF コマンドを使用して、削除するレコードを選択します。

```
OVRDBF    FILE(FILEA) SHARE(*YES)
OPNQRYF   FILE(FILEA) OPTION(*ALL) QRYSLT('DLTCOD *EQ "X" ')
CALL      PGM(PGMB)
CLOF      OPNID(FILEA)
DLTOVR    FILE(FILEA)
```

OPNQRYF コマンドを使用してレコードを追加することもできます。しかし、Query 仕様に選択値が含まれている場合には、その選択値のために、プログラムが追加されたレコードを読み取ることができない場合もあります。

OPNQRYF コマンドを使用した日付、時刻、および時刻スタンプの比較: 日付、時刻、および時刻スタンプ値は、同じデータ・タイプの他の値、またはそのデータ・タイプのストリング表記と比較できます。すべての比較は日時の順に行われます。つまり、0001 年 1 月 1 日から始まって、時がたつにつれ値が大きくなる という意味です。

時刻の値および時刻の値のストリング表記が関係する比較は必ず秒数が含まれます。ストリング表記が秒数を省略する場合、ゼロ秒が暗黙のうちに指定されます。

時刻スタンプ値を伴う比較は、同等とみなされる可能性がある表記に関係なく、発生順に行われます。したがって、次の述部は真です。

TIMESTAMP('1990-02-23-00.00.00') > '1990-02-22-24.00.00'

文字、DBCS 混用、または DBCS 択一フィールドあるいは定数が、日付、時刻、または時刻スタンプとして表されたとき、以下の規則があてはまります。

日付: 日付形式が *ISO、*USA、*EUR、*JIS、*YMD、*MDY、または *DMY の場合、フィールドまたはリテラルの長さは少なくとも 8 でなければなりません。日付形式が *JUL (yyddd) の場合、変数の長さは少なくとも 6 (yy と ddd の区切り記号を含む) でなくてはなりません。フィールドまたはリテラルには、ブランクを埋め込むことができます。

時刻: すべての時刻形式 (*USA、*ISO、*EUR、*JIS、*HMS) で、フィールドまたはリテラルの長さは少なくとも 4 でなければなりません。フィールドまたはリテラルにはブランクが埋め込まれています。

時刻スタンプ: 時刻スタンプ形式 (yyyy-mm-dd-hh.mm.ss.uuuuuu) では、フィールドまたはリテラルの長さは少なくとも 16 でなければなりません。フィールドまたはリテラルには、ブランクを埋め込むことができます。

OPNQRYF コマンドを使用した日付、時刻、および時刻スタンプの算術演算: 日付、時刻、および時刻スタンプ値は、増分、減分、または減算することができます。これらの操作には、*期間* と呼ばれる 10 進数が関係することがあります。期間の定義については、 『期間』を参照してください。日付、時刻、および時刻スタンプ値に対する算術演算の規則については、 166 ページの『日付、時刻、および時刻スタンプの演算の規則』を参照してください。

このほか、これらの操作に関連した以下のトピックも参照してください。

- 日付の算術演算:
  - 167 ページの『日付の減算』
  - 167 ページの『日付の増分および減分』
- 時刻の算術演算:
  - 168 ページの『時刻の減算』
  - 168 ページの『時刻の増分および減分』
- 時刻スタンプの算術演算:
  - 168 ページの『時刻スタンプの減算』
  - 169 ページの『時刻スタンプの増分および減分』

期間: **期間**は時間の間隔を表す数値です。期間の 4 つのタイプには以下のものがあります。

ラベル付き期間

ラベル付き期間は、7 つの期間組み込み関数 %DURYEAR、%DURMONTH、%DURDAY、%DURHOUR、%DURMINUTE、%DURSEC、または %DURMICSEC のうちの 1 つのオペランドとして使用される数値 (式の結果でもよい) により表現される、特定の時間の単位です。7 つの関数はそれぞれ、年、月、日、時、分、秒、マイクロ秒の期間のためのものです。指定された数値は、DECIMAL(15, 0) に割り当てられたかのように変換されます。ラベル付き期間を算術演算子のオペランドとして使用できるのは、もう一方のオペランドがデータ・タイプ *DATE、*TIME、*TIMESTP の値であるときだけです。したがって、式 HIREDATE + %DURMONTH(2)

+ %DURDAY(14) は有効ですが、式 HIREDATE + (%DURMONTH(2) + %DURDAY(14)) は無効です。これらの式のいずれにおいても、ラベル付き期間は、%DURMONTH(2) と %DURDAY(14) です。

日付期間

日付期間は、DECIMAL(8, 0) で表現される、年、月、日の数値を表します。正しく解釈されるために、数値は *yyyymmdd* の形式でなければなりません。*yyyy* には年、*mm* には月、 *dd* には日が入ります。ある日付値を他のものから引いた結果 (式 HIREDATE - BRTHDATE など) が日付期間です。

時刻期間

時刻期間は、DECIMAL(6, 0) で表現される、時、分、秒の数値を表します。正しく解釈されるために、数値は *hhmmss* の形式でなければなりません。*hh* には時、*mm* には分、 *ss* には秒が入ります。ある時刻値を他のものから引いた結果が時刻期間です。

時刻スタンプ期間

時刻スタンプ期間は、DECIMAL(20, 6) として表現される、年、月、日、時、分、秒、マイクロ秒の数値を表します。正しく解釈されるためには、数値が *yyyymmddhhmmsszzzzzz* の形式でなければなりません。ここで、*yyyy*、 *mm*、*dd*、*hh*、*mm*、*ss* および *zzzzzz* には、それぞれ、年、月、日、時、分、秒、マイクロ秒が入ります。ある時刻スタンプ値を他の時刻スタンプ値から引いた結果が、時刻スタンプ期間です。

日付、時刻、および時刻スタンプの演算の規則: 日付と時刻値に対して実行される算術演算は、加算と減算のみです。日付値または時刻値が加算のオペランドであれば、他のオペランドは期間でなければなりません。日付値と時刻値に関する加算演算子の使用に関する規則は、以下のとおりです。

- 一方のオペランドが日付の場合、他方のオペランドは期間またはラベル付き期間 (年、月、または日の) でなければならない。
- 一方のオペランドが時刻の場合、他方のオペランドは時間またはラベル付き期間 (時、分、または秒の) でなければならない。
- 一方のオペランドが時刻スタンプの場合、他方のオペランドは期間でなければならない。どのタイプの期間も有効です。

日付値および時刻値の減算演算子の使用についての規則は、加算の場合と同じではありません。なぜなら、日付値または時刻値は期間から引くことはできず、日付値や時刻値から日付値や時刻値を引く演算は、日付値または時刻値から期間を引く演算とは同じではないからです。日付値と時刻値に関する減算演算子の使用に関する規則は、以下のとおりです。

- 最初のオペランドが日付の場合、2 番目のオペランドは日付、期間、日付のストリング表記、またはラベル付き期間 (年、月、または日の) でなければならない。
- 2 番目のオペランドが日付の場合、最初のオペランドは日付または日付のストリング表記でなければならない。
- 最初のオペランドが時刻の場合、2 番目のオペランドは時刻、時間、時刻のストリング表記、またはラベル付き期間 (時、分、または秒の) でなければならない。
- 2 番目のオペランドが時刻の場合、最初のオペランドは時刻または時刻のストリング表記でなければならない。
- 最初のオペランドが時刻スタンプの場合、2 番目のオペランドは時刻スタンプ、時刻スタンプのストリング表記、または期間でなければならない。
- 2 番目のオペランドが時刻スタンプの場合、最初のオペランドは時刻スタンプまたは時刻スタンプのストリング表記でなければならない。

日付の減算: ある日付 (DATE2) を他の日付 (DATE1) から引いた結果は、 2 つの日付の間の年数、月数、および日数を指定する期間です。結果のデータ・タイプは DECIMAL(8, 0) です。DATE1 が DATE2 よりも大きいあるいは等しい場合は、DATE1 から DATE2 を引きます。しかし、 DATE1 が DATE2 よりも小さい場合、DATE2 から DATE1 を引くと、結果の符号は負になります。以下の手続きの記述は、式 RESULT = DATE1 - DATE2 に伴うステップを明確に示しています。

```
%DAY(DATE2) <= %DAY(DATE1) の場合、
    %DAY(RESULT) = %DAY(DATE1) - %DAY(DATE2)

%DAY(DATE2) > %DAY(DATE1) の場合、
    %DAY(RESULT) = N + %DAY(DATE1) - %DAY(DATE2)
    (ここで、N = %MONTH(DATE2) の最終日)
     %MONTH(DATE2) は 1 増加します。
%MONTH(DATE2) <= %MONTH(DATE1) の場合、
     %MONTH(RESULT) = %MONTH(DATE1) - %MONTH(DATE2)

%MONTH(DATE2) > %MONTH(DATE1) の場合、
     %MONTH(RESULT) = 12 + %MONTH(DATE1) - %MONTH(DATE2)
     %YEAR(DATE2) は 1 増加します。
%YEAR(RESULT) = %YEAR(DATE1) - %YEAR(DATE2)
```

たとえば、%DATE('3/15/2000') - '12/31/1999' の結果は 215 (すなわち、0 年、2 か月、15 日の期間) です。

日付の増分および減分: 日付に期間を加えた結果、または日付から期間を引いた結果は、日付になります。(この演算のために、月はカレンダーのページと同等のものを表します。日付に月を加えれば、カレンダーのページをめくったようになり、その日付の入っているページから始まります。) 結果は、0001 年 1 月 1 日から 9999 年 12 月 31 日までの日付にならなければなりません。年の期間が加算または減算される場合、日付の年の部分のみが影響を受けます。月は変更されず、計算の結果がうるう年でない年なのに 2 月 29 日とならない限り、日もそのままです。この場合には、日は 28 に変更されます。

同様に、月の期間が加算または減算される場合は、月のみが (さらに、必要であれば年も) 影響を受けます。日付の日の部分は、計算結果が無効 (たとえば、9 月 31 日) とならない限り、変更されません。この場合には、日はその月の最後の日に設定されます。

日の期間を加算または減算すれば、もちろん日付の日の部分が影響を受け、場合によっては月および年も影響を受けます。

期間は、正または負にかかわらず、日付に加えたり日付から引いたりすることができます。ラベル付き期間の場合と同様に、結果は有効な日付になります。

正の期間が日付に加えられるか、または負の期間が日付から引かれると、日付は指定された年数、月数、および日数だけ、その順序で増分されます。したがって、DATE1 + X (X は正の DECIMAL(8,0) の数値) は、次の式と同等です。DATE1 + %DURYEAR(%YEAR(X)) + %DURMONTH(%MONTH(X)) + %DURDAY(%DAY(X))

正の期間が日付から引かれるか、または負の期間が日付に加えられると、日付は指定された日数、月数、および年数だけ、その順序で減分されます。したがって、DATE1 - X (X は正の DECIMAL(8,0) の数値) は、次の式と同等です。DATE1 - %DURDAY(%DAY(X)) - %DURMONTH(%MONTH(X)) - %DURYEAR(%YEAR(X))

期間を日付に加えるとき、指定された日付に 1 か月を加えると、 1 月遅れの同じ日になります。ただし、その日付が 1 か月後の月に存在しない場合は除きます。その場合、日付は次の月の最後の日に設定されま

す。たとえば、1 月 28 日に 1 カ月後を加えると 2 月 28 日になり、 1 月 29、30、または 31 日に 1 カ月を加えた結果は、いずれも 2 月 28 日になります (うるう年の場合は 2 月 29 日です) 。

注: ある日付に 1 か月または数か月を加え、その結果から同じ月数だけ引いた場合、最終的な日付は必ずしも最初の日付と同じになるとは限りません。

時刻の減算: ある時刻 (TIME2) を他の時刻 (TIME1) から引いた結果は、 2 つの時刻の間の時間数、分数、および秒数を指定する時間です。結果のデータ・タイプは DECIMAL(6, 0) です。TIME1 が TIME2 より大きいか等しい場合は、TIME2 が TIME1 より引かれます。しかし、TIME1 が TIME2 より小さい場合は、TIME1 が TIME2 より引かれ、結果の符号は負になります。以下の手続きの記述は、式 RESULT = TIME1 - TIME2 に伴うステップを明確に示しています。

%SECOND(TIME2) <= %SECOND(TIME1) の場合、
    %SECOND(RESULT) = %SECOND(TIME1) - %SECOND(TIME2)

%SECOND(TIME2) > %SECOND(TIME1) の場合、
    %SECOND(RESULT) = 60 + %SECOND(TIME1) - %SECOND(TIME2)
    %MINUTE(TIME2) は 1 増加します。
%MINUTE(TIME2) <= %MINUTE(TIME1) の場合、
    %MINUTE(RESULT) = %MINUTE(TIME1) - %MINUTE(TIME2)

%MINUTE(TIME2) > %MINUTE(TIME1) の場合、
    %MINUTE(RESULT) = 60 + %MINUTE(TIME1) - %MINUTE(TIME2)
    %HOUR(TIME2) は 1 増加します。
%HOUR(RESULT) = %HOUR(TIME1) - %HOUR(TIME2)。

たとえば、%TIME('11:02:26') - '00:32:56' の結果は 102930 (10 時間、29 分、30 秒の期間) です。

時刻の増分および減分: 時刻に期間を加えた結果、または時刻から期間を引いた結果は、時刻になります。時間のオーバーフローまたはアンダーフローは廃棄されるので、結果は常に時刻となります。時の期間が加算または減算される場合、時刻の時の部分のみが影響を受けます。分や秒は変更されません。

同様に、分の期間が加算または減算される場合は、分のみが (さらに、必要であれば時も) 影響を受けます。秒の部分は変更されません。

秒の期間を加算または減算すれば、もちろん時刻の秒の部分が影響を受け、場合によっては分および時も影響を受けます。

時間も、正か負かにかかわりなく、時刻に加えたり時刻から引いたりできます。結果は、指定された時間数、分数、秒数だけ、その順序で増分または減分された時刻です。したがって、TIME1 + X (X は DECIMAL(6,0) の数値)は、次の式と同等です。TIME1 + %DURHOUR(%HOUR(X)) + %DURMINUTE(%MINUTE(X)) + %DURSEC(%SECOND(X))

時刻スタンプの減算: ある時刻スタンプ (TS2) を別の時刻スタンプ (TS1) から引いた結果は、 2 つの時刻スタンプの間の年、月、日、時、分、秒、およびマイクロ秒の数を指定する時刻スタンプ期間です。結果のデータ・タイプは DECIMAL(20, 6) です。TS1 が TS2 より大きいか等しい場合には、TS2 が TS1 より引かれます。しかし、TS1 が TS2 より小さい場合は、TS1 から TS2 を引くと、結果の符号は負になります。以下の手続きの記述は、式 RESULT = TS1 - TS2 に伴うステップを明確に示しています。

%MICSEC(TS2) <= %MICSEC(TS1) の場合、
    %MICSEC(RESULT) = %MICSEC(TS1) -
    %MICSEC(TS2)

%MICSEC(TS2) > %MICSEC(TS1) の場合、

%MICSEC(RESULT) = 1000000 +
%MICSEC(TS1) - %MICSEC(TS2)
%SECOND(TS2) は 1 増加します。

時刻スタンプの秒と分の部分は、時刻の減算に関する規則で指定されたとおりに減算されます。

%HOUR(TS2) <= %HOUR(TS1) の場合、
%HOUR(RESULT) = %HOUR(TS1) - %HOUR(TS2)

%HOUR(TS2) > %HOUR(TS1) の場合、
%HOUR(RESULT) = 24 + %HOUR(TS1) - %HOUR(TS2)
%DAY(TS2) は 1 増加します。

時刻スタンプの日の部分は、日付の減算に関する規則で指定されたとおりに減算されます。

時刻スタンプの増分および減分: 時刻スタンプに期間を加えた結果、または時刻スタンプから期間を引いた結果は、時刻スタンプになります。日付と時刻の算術計算は、前述のように実行されます。ただし、時間のオーバーフローとアンダーフローが結果の日付部分に取り込まれる点を除きます (結果の日付は、有効な日付の範囲内でなければなりません)。マイクロ秒は秒へオーバーフローします。

ランダム処理のための Query ファイルのオープン (OPNQRYF) コマンドの使用: これまでの例の大部分は、順次処理を使用する OPNQRYF コマンドを示しています。ほとんどの場合に、ランダム処理操作 (たとえば、RPG/400 言語の CHAIN 操作または COBOL/400 言語の READ 操作) を使用することができます。しかし、グループ機能または固有キー機能を使用する場合には、ファイルをランダムに処理することはできません。

Query ファイルのオープン・コマンド: パフォーマンスの考慮事項: Query アプリケーションのパフォーマンスを最適化するためのヒント、および技法については、データベース・パフォーマンスおよび Query 最適化 を参照してください。

Query ファイルのオープン (OPNQRYF) コマンドで既存のキー順アクセス・パスを使用するときに、最高のパフォーマンスを得ることができます。たとえば、*Code* フィールドが B に等しいすべてのレコードを選択したい場合に、*Code* フィールドに対するアクセス・パスが存在すれば、システムは実行時にレコードを読み取って選択する (動的選択) 代わりに、アクセス・パスを使用して選択 (キー位置による選択) を実行することができます。

次のいずれかが真の場合、Query ファイルのオープン (OPNQRYF) コマンドは、既存の索引を使用することができません。

- アクセス・パスのキー・フィールドが、サブストリング関数から抽出された場合。
- アクセス・パスのキー・フィールドが、連結関数から抽出された場合。
- Query と関連した分類順序表 (SRTSEQ パラメーターで指定される) が、次の両方に該当する場合。
  - 同順位分類順序表である。
  - アクセス・パスに関連した順序表 (分類順序表または代替照合順序表) と一致しない。
- Query と関連した分類順序表 (SRTSEQ パラメーターで指定される) が、次の両方に該当する場合。
  - 固有分類順序表である。

– 次のいずれかのときに、アクセス・パスに関連した順序表 (分類順序表または代替照合順序表) と一致しない。
  - 順序付けが指定されている (KEYFLD パラメーター) 。
  - *EQ、*NE、*CT、%WLDCRD、または %VALUES を使用しないレコード選択が存在する (QRYSLT パラメーター)。
  - *EQ または *NE 演算子を使用しない結合選択が存在する (JFLD パラメーター)。

OPNQRYF 処理の一部は、要求を満たす最も速い方法を判別します。使用するファイルが大きく、ほとんどのレコードに B と等しい *Code* フィールドがある場合には、既存のキー順アクセス・パスを使用するよりも、到着順処理を行う方が速くなります。その場合にも、プログラムはやはり同じレコードを参照することになります。*Code* フィールドにアクセス・パスが存在する場合には、OPNQRYF はこのタイプの決定を行うことができます。一般に、ファイル中のレコードの約 20% 以上が要求に該当する場合には、OPNQRYF は既存のアクセス・パスを無視して、ファイルを到着順に読み取る傾向があります。

Code フィールドに対するアクセス・パスが存在していない場合、プログラムはファイル中のすべてのレコードを読み取り、選択されたレコードだけをプログラムに渡します。すなわち、ファイルは到着順で処理されます。

システムは、アプリケーション・プログラムよりも速く選択を行うことができます。該当するキー順アクセス・パスが存在しない場合には、プログラムまたはシステムのいずれかが、処理するレコードの選択を行います。システムに選択処理を実行させれば、すべてのレコードをアプリケーション・プログラムに渡すよりも大幅に速くなります。

このことは、ファイルを更新操作のためにオープンする場合に特に当てはまります。その理由は、個々のレコードをプログラムに渡さなければならず、レコードの読み取りのたびにロックが行われるからです (プログラムがレコードを更新する必要がある場合)。システムにレコード選択を実行させると、選択値を満たすレコードだけがプログラムに渡されてロックされることになります。

KEYFLD パラメーターを使用して、レコードを読み取る特定の順序を要求するときには、同じキー仕様を使用するアクセス・パスがすでに存在している場合、または仕様と類似したキー順アクセス・パス (指定したすべてのフィールドが含まれていて、さらにキーの終わりにいくつかのフィールドが追加されているキーなど) が存在している場合に、パフォーマンスは最も速くなります。これは、GRPFLD パラメーターおよび JFLD パラメーターの TO フィールドにも当てはまります。こうしたアクセス・パスが存在しない場合には、システムがアクセス・パスを作成し、ジョブでファイルがオープンされている間、そのアクセス・パスを保守します。

整列されるレコードの数 (必ずしも、ファイル内のレコードの総数とは限らない) が 1000 を超え、かつファイル内のレコードの 20% よりも多い場合には、まだ存在していないアクセス・パスによってファイル内のすべてのレコードを処理することは、通常、全レコード分類を使用することより効率的ではありません。一般的には、分類を行うよりも、キー順アクセス・パスを作成する方が高速ですが、ジョブの合計時間から見れば、通常、到着順処理を使用することによってより高速な処理が可能となるので、データの分類を行う方が有利になります。使用可能なアクセス・パスがすでに存在している場合には、そのアクセス・パスを使用する方が、データの分類を行うよりも速くなります。レコード処理の最も速い方法が全レコード分類であるようなら、 Query ファイルのオープン (OPNQRYF) コマンドの ALWCPYDTA(*OPTIMIZE) パラメーターを使用して、システムがその方法を使用できるようにすることができます。

Query レコードのすべては読み取る予定がない場合、および OPTIMIZE パラメーターが *FIRSTIO または *MINWAIT の場合、取り出すレコードの数を示す数値を指定することができます。レコード数が、Query によって返されると予想されている総数よりかなり少ない場合、システムはより速いアクセス方式を選択する場合があります。

グループ化機能を使用する場合は、グループ化後の選択 (GRPSLT パラメーター) の代わりにグループ化前の選択 (QRYSLT パラメーター) を指定することによって、より速いパフォーマンスが得られます。GRPSLT パラメーターは、集約関数を用いる比較にだけ使用してください。

OPNQRYF コマンドを使用するほとんどの場合では、データにアクセスし、それをプログラムに渡すために、新しいアクセス・パスまたは既存のアクセス・パスが使用されます。OPNQRYF コマンドでは、システムが一時ファイルを作成しなければならない場合もあります。いつ一時ファイルが作成されるかについての規則は複雑ですが、典型的な例として、以下の場合があります。

- 動的結合を指定しているのに、KEYFLD パラメーターに異なる物理ファイルからのキー・フィールドが記述されている場合。
- 動的結合を指定しているのに、GRPFLD パラメーターに異なる物理ファイルからのキー・フィールドが記述されている場合。
- GRPFLD パラメーターおよび KEYFLD パラメーターの両方を指定したが、それらが同じでない場合。
- KEYFLD パラメーターに指定したフィールドの長さが 2000 バイトを超える場合。
- 動的結合を指定し、OPTIMIZE パラメーターに *MINWAIT を指定した場合。
- 結合論理ファイルを使用する動的結合を指定しているときに、結合論理ファイルの結合タイプ (JDFTVAL) が動的結合の結合タイプに一致しない場合。
- 1 つの論理ファイルを指定しているときに、その論理ファイルの様式が複数の物理ファイルを参照している場合。
- SQL ビューを指定しているときに、システムがビューの結果を入れるための一時ファイルを要求する場合。
- ALWCPYDTA(*OPTIMIZE) パラメーターが指定されており、一時結果を使用すれば Query のパフォーマンスが向上する場合。

動的結合が生じると (JDFTVAL(*NO))、OPNQRYF は、ファイルを再配置し、選択されたレコードが最小数のファイルを、選択されたレコードが最大数のファイルと結合させることにより、パフォーマンスの向上を図ります。OPNQRYF がファイルを再配置しないようにするためには、 JORDER(*FILE) を指定します。これは OPNQRYF コマンドに指定された順序でファイルを結合するよう、 OPNQRYF に強制します。

Query ファイルのオープン・コマンド: 分類順序表のパフォーマンスの考慮事項: 分類順序表のパフォーマンスの考慮事項については、以下のトピックを参照してください。

- 『グループ化、結合、および選択: OPNQRYF パフォーマンスの考慮事項』
- 172 ページの『順序: OPNQRYF パフォーマンスの考慮事項』

グループ化、結合、および選択: OPNQRYF パフォーマンスの考慮事項: 既存の索引を使用する際、最適化プログラムは、選択、結合、およびグループ化フィールドの属性が、既存の索引内のキーの属性と一致することを確認します。さらに、Query と関連した分類順序表が、既存の索引のキー・フィールドと関連した順序表 (分類順序表または代替照合順序表) と一致していなければなりません。順序表が一致しない場合、既存の索引は使用できません。

ただし、Query に関連した分類順序表が同順位分類順序表 (*HEX を含む) である場合は、追加の最適化が可能です。最適化プログラムは、グループ化フィールド、あるいは次の演算子または関数を使用する選択または結合述部について、分類順序表が指定されていないかのように動作します。

- *EQ
- *NE
- *CT
- %WLDCRD
- %VALUES

この利点は、キーがフィールドと一致し、かつアクセス・パスが次のいずれかの場合に、最適化プログラムは自由に任意の既存のアクセス・パスを使用できるということです。

- 順序表を含まない。
- 固有分類順序表を含まない (表は Query と関連した固有分類順序表と一致する必要はありません)。

順序: OPNQRYF パフォーマンスの考慮事項: 順序付けフィールドに対して、最適化プログラムは自由に任意の既存のアクセス・パスを使用することはできません。索引と Query のそれぞれに関連した分類順序表は、最適化プログラムが順序付け要求を満たすために分類を行うことを選択した場合を除き、一致しなければなりません。分類が使用されると、分類中に変換が行われ、最適化プログラムが選択基準に適合する既存のアクセス・パスを自由に使用できるようになります。

他のデータベース機能とのパフォーマンスの比較: Query ファイルのオープン (OPNQRYF) コマンドでは、論理ファイルおよび結合論理ファイルと同じデータベース・サポートが使用されます。したがって、キー順アクセス・パスの作成や結合操作の実行といった機能のパフォーマンスは、同じになります。

OPNQRYF コマンドで実行される選択機能 (QRYSLT パラメーターと GRPSLT パラメーターの場合) は、論理ファイルの選択/除外と似ています。主な相違は、OPNQRYF コマンドにおいては、システム上で使用可能なアクセス・パスおよび OPTIMIZE パラメーターに指定されている値の結果として、アクセス・パス選択または動的選択を使用するかどうかをシステムが決めるという点です (論理ファイルの DDS 中の DYNSLT キーワードを除外するか、指定するかということと同様です)。

フィールドの使用に関する考慮事項: グループ化機能を使用する場合には、Query ファイルのオープン用のレコード様式 (FORMAT パラメーター) に含まれるすべてのフィールド、およびすべてのキー・フィールド (KEYFLD パラメーター) は、グループ化フィールド (GRPFLD パラメーターで指定された) 、あるいはグループ化フィールド、定数、および集約関数だけを用いて定義されたマップ・フィールド (MAPFLD パラメーターで指定された) のいずれかでなければなりません。集約関数は、%AVG、%COUNT、%MAX (1 つのオペランドだけを使用) 、 %MIN (1 つのオペランドだけを使用)、%STDDEV、%SUM、および %VAR です。グループ処理が必要となるのは、以下の場合です。

- GRPFLD パラメーターでグループ化フィールド名を指定する場合。
- GRPSLT パラメーターでグループ選択値を指定する場合。
- MAPFLD パラメーターで指定するマップ・フィールドの定義の中で集約関数が使用されている場合。

ラージ・オブジェクト・データ・タイプ、すなわち BLOB、CLOB、または DBCLOB を持つフィールドは、Query ファイルからのコピー **(CPYFRMQRYF)** コマンドを使用するか、または構造化照会言語 (SQL) を使用してのみ読み取ることができます。ラージ・オブジェクト・フィールド・データは、Query オープン・ファイルから直接アクセスできません。 **CPYFRMQRYF** コマンドを使用してオープン Query ファイルからラージ・オブジェクト・フィールドにアクセスします。ラージ・オブジェクト・データ・タイプ (BLOB、 CLOB または DBCLOB) を持つフィールドは、KEYFLD、UNIQUEKEY、JFLD、および GRPFLD の OPNQRYF パラメーターでは指定できません。

タイプ DATALINK のフィールドは、選択、グループ化、配列、または結合では表示されません。DATALINK フィールドがその様式に現された場合には、データベース内に存在しているため、未処理で戻されます。

FILE パラメーターで識別されたレコード様式に含まれていて、 N (非入出力) の用途値を用いて定義されている (ファイルの作成に使用された DDS で) フィールドは、 OPNQRYF コマンドのどのパラメーターにも指定することができません。I (入力専用) または B (入出力共用) として定義されたフィールドだけが指定できます。FORMAT パラメーターで識別されたレコード様式で用途が N として定義されているフィールドは、すべて OPNQRYF コマンドによって無視されます。

オープンした Query ファイルのレコード内のフィールドは、通常、 FORMAT パラメーターで識別されたレコード様式内のフィールドと同じ用途属性 (入力専用または入出力共用) を持ちますが、下記の例外があります。ファイルが、出力や更新および任意の用途を含む任意のオプション (OPTION パラメーター) のためにオープンされている場合に、 FORMAT パラメーターで指定されたレコード様式内の B (入出力共用) フィールドが、 Query ファイルのオープンのレコード様式で I (入力専用) に変更されると、 OPNQRYF コマンドによって通知メッセージが送られます。

結合処理またはグループ処理を要求した場合、あるいは UNIQUEKEY 処理を指定した場合には、Query レコード内のすべてのフィールドが入力専用となります。処理されるファイル (FILE パラメーターで識別された) の入力専用フィールドからのマッピングは、いずれも Query ファイルのオープンのレコード様式で入力専用となります。MAPFLD パラメーターを使用して定義されたフィールドは、 Query ファイルのオープンでは、通常、入力専用となります。MAPFLD パラメーターで定義されたフィールドは、以下の条件がすべて満たされる場合に、その構成フィールドの用途に一致した値となります。

- この節で前に説明した条件のすべてが満たされているため、入力専用の必要がない。
- MAPFLD パラメーターで指定したフィールド定義式がフィールド名である (演算子や組み込み関数が入っていない)。
- フィールド定義式で使用したフィールドが、FILE パラメーターで指定したファイル、メンバー、またはレコード様式の 1 つに存在している (MAPFLD パラメーターを使用して定義した別のフィールドには存在しない)。
- 基礎フィールドとマップ・フィールドが互換性のあるフィールド・タイプである (同じ長さのゾーン 10 進数フィールドと文字フィールドの間のマッピングを除いて、マッピングで数字と文字のフィールド・タイプが混合されることはありません)。
- 基礎フィールドがゼロでない小数部精度を持つ 2 進数フィールドである場合には、マップ・フィールドも同じ精度の 2 進数フィールドでなければならない。

ジョブで共用されるファイルに関する考慮事項: アプリケーション・プログラムが Query ファイルのオープン (OPNQRYF) コマンドによって作成されたオープン・データ・パスを使用するには、Query ファイルを共用しなければなりません。プログラムが Query ファイルを共用としてオープンしない場合、実際には、それが使用するようにコンパイルされていたファイルの全オープンを行うことになります (OPNQRYF コマンドで作成された Query オープン・データ・パスを使用せずに)。プログラムは、以下の条件に応じて、Query オープン・データ・パスを共用します。

- アプリケーション・プログラムは、ファイルを共用としてオープンしなければなりません。プログラムは、Query の対象となる最初のメンバーまたは唯一のメンバー (FILE パラメーターで指定) が SHARE(*YES) の属性を持っているときに、この条件を満たします。最初のメンバーまたは唯一のメンバーが SHARE(*NO) の属性を持っている場合は、プログラムを呼び出す前に、データベース・ファイルによる一時変更 (OVRDBF) コマンドで SHARE(*YES) を指定しなければなりません。
- アプリケーション・プログラムでオープンされるファイルは、 OPNQRYF コマンドでオープンされるファイルと同じ名前を持っていなければなりません。プログラムは、プログラムの中に指定されているフ

ファイルが、Query の対象となる最初のメンバーまたは唯一のメンバー (FILE パラメーターで指定) と同じファイルおよびメンバー名を持っている場合に、この条件を満たします。最初のメンバーまたは唯一のメンバーが異なる名前を持っている場合には、データベース・ファイルによる一時変更 (OVRDBF) コマンドを指定して、プログラムがコンパイルされているファイル名を、 Query の対象となる最初のメンバーまたは唯一のメンバーの名前に一時変更しなければなりません。

- ユーザーのプログラムは、Query オープン・データ・パス (ODP) が範囲限定されるのと同じ活動化グループで実行されなければなりません。Query ODP がジョブに範囲限定される場合は、プログラムはジョブ内の任意の活動化グループで実行することができます。

OPNQRYF コマンドは、ジョブまたは活動化グループ内の既存のオープン・データ・パスを決して共用しません。オープン・データ・パスがオープン要求にあるものと同じライブラリー、ファイル、およびメンバー名を持つ場合、かつ次のいずれかが真の場合は、 Query ファイルのオープン要求は失敗し、エラー・メッセージが送られます。

- OPNQRYF コマンドに対して OPNSCOPE(*ACTGRPDFN) または OPNSCOPE(*ACTGRP) が指定され、オープン・データ・パスが、OPNQRYF コマンドが実行されるのと同じ活動化グループまたはジョブに範囲限定される。
- OPNQRYF コマンドに対して OPNSCOPE(*JOB) が指定され、オープン・データ・パスが、OPNQRYF コマンドが実行されるのと同じジョブに範囲限定される。

それ以後の共用オープンは、OPNQRYF コマンドの実行時に有効であったのと同じオープン・オプション (SEQONLY など) を受け継ぎます。

ジョブまたは活動化グループ内のファイルの共用の詳細については、 111 ページの『同一ジョブまたは活動化グループ内のデータベース・ファイルの共用』を参照してください。

レコード様式記述が変更されたかどうかの検査に関する考慮事項: レコード様式のレベル検査を指定した場合は、Query ファイルのオープンのレコード様式 (FORMAT パラメーターで識別された) の様式レベル数が、プログラムがコンパイルされたレコード様式に対して検査されます。これは、プログラムが前にオープンされた Query ファイルを共用する場合に起こります。以下の条件が満たされていると、プログラムの共用オープンでレコード様式レベルに関する検査が行われます。

- Query の対象となる最初または唯一のファイル (FILE パラメーターで指定) が、 LVLCHK(*YES) 属性を持つ。
- Query の対象となる最初または唯一のファイルが、LVLCHK(*NO) に一時変更されていない。

OPNQRYF コマンドに関するその他の実行時の考慮事項: 以下は、OPNQRYF 実行時のその他の考慮事項です。

- 『一時変更と OPNQRYF コマンド』
- 『オープンしている Query ファイルからのコピー』

一時変更と OPNQRYF コマンド: 一時変更により、Query ファイルのオープンで処理したいファイル、ライブラリー、およびメンバーの名前を変更することができます。(ただし、データベース・ファイルによる一時変更 (OVRDBF) で指定したパラメーター値は、TOFILE、MBR、LVLCHK、INHWRT、または SEQONLY を除いて、 OPNQRYF コマンドによって無視されます。) Query の対象となる最初のメンバーまたは唯一のメンバーに名前の一時変更が適用される場合には、それ以降の一時変更は、 OPNQRYF コマンドの FILE パラメーターに指定されている名前ではなく、この新しい名前に対して行わなければなりません。

オープンしている Query ファイルからのコピー: Query ファイルからのコピー (CPYFRMQRYF) コマンドを使用すると、オープンしている Query ファイルを別のファイルへコピーしたり、レコードの定様式リ

ストを印刷したりすることができます。Query ファイルのオープン (OPNQRYF) コマンドの FILE パラメーターで指定された、入力、更新、または全操作用にオープンしている Query ファイルは、分散データ管理機能 (DDM) ファイルを使用しているものを除き、 CPYFRMQRYF コマンドを用いてコピーすることができます。CPYFRMQRYF コマンドは、論理ファイルへのコピーには使用できません。詳細については、ファイル管理 を参照してください。

CPYFRMQRYF コマンドはオープンしている Query ファイルのオープン・データ・パスを使用しますが、ファイルはオープンしません。したがって、コピーするデータベース・ファイルには SHARE(*YES) を指定する必要がありません。

以下の例は、OPNQRYF コマンドと CPYFRMQRYF コマンドの使用方法を示したものです。

- 『例 1: オープンしている Query ファイルからのコピー』
- 『例 2: オープンしている Query ファイルからのコピー』
- 『例 3: オープンしている Query ファイルからのコピー』
- 『例 4: オープンしている Query ファイルからのコピー』

例 1: オープンしている Query ファイルからのコピー: レコードのサブセットを持つファイルの作成

STATE フィールドの値が TEXAS であるレコードだけから成るファイルを CUSTOMER/ADDRESS ファイルから作成したいとします。以下のように指定することができます。

```
OPNQRYF FILE(CUSTOMER/ADDRESS) QRYSLT('STATE *EQ "TEXAS"')
CPYFRMQRYF FROMOPNID(ADDRESS) TOFILE(TEXAS/ADDRESS) CRTFILE(*YES)
```

例 2: オープンしている Query ファイルからのコピー: 選択したレコードの印刷

CITY フィールドの値が CHICAGO であるすべてのレコードを FILEA から選択して印刷したいとします。以下のように指定することができます。

```
OPNQRYF FILE(FILEA) QRYSLT('CITY *EQ "CHICAGO"')
CPYFRMQRYF FROMOPNID(FILEA) TOFILE(*PRINT)
```

例 3: オープンしている Query ファイルからのコピー: レコードのサブセットのディスケットへのコピー

FIELDB の値が 10 であるすべてのレコードを FILEB からディスケットへコピーしたいとします。以下のように指定することができます。

```
OPNQRYF FILE(FILEB) QRYSLT('FIELDB *EQ "10"') OPNID(MYID)
CPYFRMQRYF FROMOPNID(MYID) TOFILE(DISK1)
```

例 4: オープンしている Query ファイルからのコピー: 動的結合による出力のコピーの作成

様式およびデータが FILEA と FILEB を結合したものである物理ファイルを作成したいとします。ファイルには以下のフィールドが含まれているものとします。

```
FILEA        FILEB      JOINAB
Cust         Cust       Cust
Name         Amt        Name
Addr                    Amt
```

結合フィールドは、両ファイルにある Cust です。ファイルを結合し、その結果のコピーを新しい物理ファイル MYLIB/FILEC に保存するには、以下のように指定することができます。

```
OPNQRYF FILE(FILEA FILEB) FORMAT(JOINAB) +
  JFLD((FILEA/CUST FILEB/CUST)) +
  MAPFLD((CUST 'FILEA/CUST')) OPNID(QRYFILE)
CPYFRMQRYF  FROMOPNID(QRYFILE) TOFILE(MYLIB/FILEC) CRTFILE(*YES)
```

ファイル MYLIB/FILEC は、CPYFRMQRYF コマンドによって作成されます。一部のファイル属性が変更されることはありますが、ファイルは FILEA と同様のファイル属性を持つことになります。ファイルの様式は JOINAB と同様になります。ファイルには、Cust フィールドを使用することによって FILEA と FILEB を結合したデータが含まれます。ライブラリー MYLIB 内のファイル FILEC は、他の物理ファイルと同様に、物理ファイル・メンバーの表示 (DSPPFM) コマンドなどの CL コマンドや、 Query などのユーティリティーを用いて処理することができます。CPYFRMQRYF コマンドとその他のコピー・コマンドの詳細については、ファイルの管理 を参照してください。

Query ファイルのオープン (OPNQRYF) コマンドの使用時の代表的なエラー: OPNQRYF コマンドおよびプログラムが正しい結果を得るためには、いくつかの機能を正しく指定しなければなりません。問題が起きた場合には、ジョブの表示 (DSPJOB) コマンドを使用するのが最良の方法です。このコマンドは、ファイルのオープン・オプションとファイルの一時変更オプションの両方をサポートします。問題がある場合には、これらのオプションの両方を見てください。

以下に示すのは、最も起こりやすい問題とその訂正方法です。

- 共用オープン・データ・パス (ODP)。OPNQRYF コマンドは、共用 ODP を通して働きます。ファイルを正しく処理するには、メンバーが共用 ODP 用にオープンされなければなりません。問題がある場合には、DSPJOB コマンドでファイルのオープン・オプションを使用して、メンバーがオープンされているかどうか、および共用 ODP を持っているかどうかを調べてください。

  通常、ファイルがオープンされない理由は以下の 2 つです。
  - 処理されるメンバーが、SHARE(*YES) でなければならない。データベース・ファイルによる一時変更 (OVRDBF) コマンドを使用するか、あるいはメンバーを永久的に変更してください。
  - ファイルがクローズされている。エラー・メッセージを受け取ったプログラムまたは単に資源の再利用 (RCLRSC) コマンドを実行したプログラムよりも呼出スタック内で高いレベルにある、省略時活動化グループ内で実行されていたプログラムから、OPNSCOPE(*ACTGRPDFN) または TYPE(*NORMAL) パラメーター・オプションを指定して OPNQRYF コマンドを実行しました。これにより、オープンしている Query ファイルはクローズされます。これは、そのファイルが、RCLRSC コマンドを実行したプログラムよりも呼び出しスタック内で高いレベルにあるプログラムからオープンされたためです。オープンしている Query ファイルがクローズされた場合は、OPNQRYF コマンドを再度実行しなければなりません。バージョン 2 リリース 3 より前のリリースで、TYPE(*NORMAL) パラメーター・オプションを指定した OPNQRYF コマンドを使用すると、オープンしている Query ファイルは、たとえ資源を再利用するのと同じプログラムからオープンされた場合であっても、クローズされることに注意してください。
- レベル検査。レベル検査を行うと、プログラムがコンパイルされたときと同じレコード様式でプログラムを実行できるようになるので、この検査は一般に使用されます。レベル検査上の問題がある場合には、通常、以下の 1 つが原因です。
  - プログラムが作成された後でレコード様式が変更された。プログラムを作成し直して、問題を訂正してください。
  - 間違ったファイルへの一時変更がプログラムに指示されている。DSPJOB コマンドでファイルの一時変更オプションを使用して、一時変更が正しく指定されているかどうかを確認してください。
  - FORMAT パラメーターが必要であるのに、指定されていないか、または間違って指定されている。FORMAT パラメーターを指定してファイルを処理する場合は、以下のことが必要です。
    - TOFILE パラメーターを指定したデータベース・ファイルによる一時変更 (OVRDBF) コマンドで、Query ファイルのオープン (OPNQRYF) コマンドの FILE パラメーターの最初のファイルを記述する。
    - FORMAT パラメーターで、プログラムの作成に使用された様式を含むファイルを指定する。

– 異なるファイルからの様式を処理するため (例：グループ処理) に FORMAT パラメーターが使用されているが、OVRDBF コマンドで SHARE(*YES) が要求されていなかった。

- 処理するファイルが、ファイルの終わりになっている。OPNQRYF コマンドの通常の用法は、ファイルを 1 回しか処理できない順次処理です。処理が行われた時点で、ファイルの位置はファイルの終わりとなり、このファイルをもう一度処理しようとしても、レコードを受け取ることはできません。ファイルを再び始めから処理するには、OPNQRYF コマンドを再び実行するか、または処理の前にファイルを再配置しなければなりません。ファイルの再配置は、データベース・ファイルの配置 (POSDBF) コマンドを使用するか、または高水準言語プログラム・ステートメントを用いて行うことができます。
- レコードがない。これは、FORMAT キーワードを使用しているのに、OVRDBF コマンドを指定していないときに起こる可能性があります。
- 構文エラー。システムが OPNQRYF コマンドの指定にエラーを見つけました。
- 無効な操作。Query の定義に KEYFLD パラメーターが含まれていないのに、高水準言語プログラムがキー・フィールドを使用して Query ファイルを読み取ろうとしています。
- 無効な GET オプション。高水準言語プログラムが現在のレコード位置より前のレコードを読み取ろうとしたか、または現在のレコード位置より前にレコード位置を設定しようとしており、さらに Query ファイルは、 SQL ステートメントでグループ別オプション、固有キー・オプション、または区別オプションを使用しています。

プログラム内での基本データベース・ファイル操作

この章では、プログラム内で実行することのできる基本データベース・ファイル操作について説明します。以下のトピックを参照してください。

- 『ファイル内での位置の設定』
- 178 ページの『データベース・レコードの読み取り』
- 184 ページの『データベース・レコードの更新』
- 185 ページの『データベース・レコードの追加』
- 188 ページの『データベース・レコードの削除』

ファイル内での位置の設定

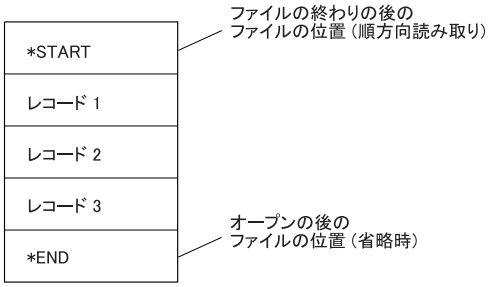
ジョブによってファイルがオープンされた後、システムはそのジョブのためにファイル内での位置を維持します。ファイル位置は、ファイルの処理で使用されます。たとえば、プログラムが次の順次レコードを要求する読み取り操作を実行する場合には、システムはファイル位置を使用して、プログラムに返すレコードを決めます。その後で、システムは、読み込まれたばかりのレコードにファイル位置を移し、次の順次レコードを要求する別の読み取り操作が実行されたときに正しいレコードが返されるようにします。システムは、各ジョブごとにすべてのファイル位置のトラックを保持しています。さらに、各ジョブは同じファイルの中に複数の位置を持つことができます。

ファイル位置は、データベース・ファイルによる一時変更 (OVRDBF) コマンドの POSITION パラメーターで指定されている位置に最初に設定されます。 制御言語 (CL) に関するトピックの、 OVRDBF (データベース・ファイルによる一時変更) コマンドを参照してください。 OVRDBF コマンドを使用しない場合、または POSITION パラメーターの省略時の値を使用する場合には、ファイル位置はメンバーのアクセス・パスの最初のレコードのすぐ前に設定されます。

プログラムは、該当の高水準言語プログラムのファイル位置決め命令 (たとえば、RPG/400 言語の SETLL または COBOL/400 言語の START) を使用して現在のファイル位置を変更できます。また、プログラムは、データベース・ファイルの位置決め (POSDBF) CL コマンドを使用してファイル位置を変更することもできます。

注: データベース・ファイルによる一時変更 (OVRDBF) コマンドによるファイルの位置決めは、次にファイルがオープンするまで起きません。ファイルは CL プログラム内で 1 回だけしかオープンできないので、このコマンドを 1 つの CL プログラムで使用して、RCVF コマンドを介して読み取られるものに影響を与えることはできません。

ファイルの終わりでは、最後の読み取りの後、プログラムがファイルを順方向に読み取るか逆方向に読み取るかに応じて、ファイル・メンバーは *START ファイル位置または *END ファイル位置に位置付けられます。以下の図は、*START ファイル位置および *END ファイル位置を示しています。先頭に *START があり、その後に 3 つのレコード、さらに末尾に *END があります。



RBAFO540-0

ファイル位置は、読み取り操作、データ強制終了操作、高水準言語の位置決め操作、またはファイル位置を変更する特定の CL コマンドによってのみ変更できます。追加、変更、または削除の操作ではファイル位置は変更されません。読み取り操作の後で、ファイルは新しいレコードに位置付けられます。その後、このレコードはプログラムに戻されます。読み取り操作が完了した後、ファイルはプログラムに戻されたばかりのレコードに位置付けされます。メンバーが入力用にオープンされている場合には、ファイルはデータ強制終了操作によってファイル内の最後のレコードの後 (*END) に位置付けられ、ファイルの終わりメッセージがプログラムに送られます。

順次読み取り操作では、アクセス・パス上の次のレコードまたは前のレコードを見つけるために現在のファイル位置が使用されます。キーによる読み取り操作または相対レコード番号による読み取り操作では、ファイル位置は使用されません。オープン時に POSITION(*NONE) を指定した場合、開始ファイル位置は設定されません。この場合、順次読み取りを行うには、ユーザーはプログラムの中でファイル位置を設定しなければなりません。

データベース・ファイルによる一時変更 (OVRDBF) コマンドでファイルに対してファイル終わり遅延が指定されている場合には、プログラムが最後のレコードを読み取った時点でもファイルは *START または *END に位置付けられず、読み取られた最後のレコードに位置付けられたままになります。ファイル終わり遅延処理が指定されたファイルは、データ強制終了 (FEOD) が起こるか、あるいはジョブの強制終了が起こったときにだけ *START または *END に位置付けられます。ファイル終わり遅延の詳細については、181 ページの『ファイル終わりに達した時のレコード待ち』を参照してください。

データベース・ファイルのオープン (OPNDBF) コマンドまたは Query ファイルのオープン (OPNQRYF) コマンドを使用してオープンされるファイルの場合には、ファイル中の現在のファイル位置を設定または変更するのにデータベース・ファイルの位置決め (POSDBF) コマンドも使用できます。

データベース・レコードの読み取り

iSeries システムは、データベース・レコードを読み取る複数の方法を提供しています。以下の節では、それらの方法を詳しく説明しています。(高水準言語によっては、システムで使用できる読み取り操作のすべてをサポートしないものもあります。データベース・レコードの読み取りの詳細については、該当の高水準言語の手引きを参照してください。)

以下のトピックを参照してください。

- 『到着順アクセス・パスを使用したデータベース・レコードの読み取り』
- 180 ページの『キー順アクセス・パスを使用したデータベース・レコードの読み取り』
- 181 ページの『ファイル終わりに達した時のレコード待ち』
- 184 ページの『ロックされたレコードの解放』

到着順アクセス・パスを使用したデータベース・レコードの読み取り: システムは、高水準言語を使用して指定される操作に基づいて、以下の読み取り操作を実行します。これらの操作は、ファイルが到着順アクセス・パスで定義されていて、プログラム内のデータベース・ファイルのオープン (OPNDBF) コマンド、または Query ファイルのオープン (OPNQRYF) コマンドでキー順アクセス・パス無視オプションが指定されているという場合に実行できます。キー順アクセス・パスを無視するためのオプションの詳細については、 104 ページの『キー順アクセス・パスの無視』を参照してください。

注: 高水準言語によっては、以下の読み取り操作のすべてを使用できないものもあります。該当の高水準言語で使用できる操作を調べるには、その言語の手引きを参照してください。

読み取り操作に関する以下のトピックを参照してください。

- 『Read next 操作』
- 『Read previous 操作』
- 『Read first 操作』
- 『Read last 操作』
- 『Read same 操作』
- 『相対レコード番号による読み取り操作』

Read next 操作: 到着順アクセス・パスの中で削除されていない次のレコードにファイルを位置付け、そのレコードを取り出します。ファイル内での現在の位置と次のアクティブ・レコードとの間にある削除済みレコードはスキップされます。(RPG/400 言語の READ ステートメントおよび COBOL/400 言語の READ NEXT ステートメントがこの例です。)

Read previous 操作: 到着順アクセス・パスの中で削除されていない前のレコードにファイルを位置付け、そのレコードを取り出します。ファイル内での現在の位置と前のアクティブ・レコードとの間にある削除済みレコードはスキップされます。(RPG/400 の言語 READP ステートメントおよび COBOL/400 言語の READ PRIOR ステートメントがこの例です。)

Read first 操作: 到着順アクセス・パスの中の最初のアクティブ・レコードにファイルを位置付け、そのレコードを取り出します。

Read last 操作: 到着順アクセス・パスの中の最後のアクティブ・レコードにファイルを位置付け、そのレコードを取り出します。

Read same 操作: ファイル内の現在位置によって識別されるレコードを取り出します。ファイル位置は変更されません。

相対レコード番号による読み取り操作: 到着順アクセス・パスの中にあって、相対レコード番号によって識別されるレコードにファイルを位置付け、そのレコードを取り出します。この相対レコード番号は、アクティブ・レコードを示すものでなければならず、メンバー中の最大の活動相対レコード番号以下の値でなければなりません。到着順アクセス・パスの中にある現在のファイル位置よりも、指定のレコード数だけ大きいかまたは小さいレコードも、この操作によって読み取られます。(RPG/400 言語の CHAIN ステートメントおよび COBOL/400 言語の READ ステートメントがこの例です。) 相対レコード処理を行うファイルの

場合には、削除済みレコードを再使用するようにファイルを作成または変更するときに特に考慮が必要です。詳細については、 104 ページの『削除済みレコードの再使用』を参照してください。

キー順アクセス・パスを使用したデータベース・レコードの読み取り: システムは、高水準言語を使用して指定したステートメントに基づいて、以下の読み取り操作を実行します。これらの操作をキー順アクセス・パスで使用して、データベース・レコードを取り出すことができます。

キー順アクセス・パスを使用している場合には、読み取り操作では、削除済みレコードが占めていた記憶域にファイルを位置付けることはできません。

注: 高水準言語によっては、以下の読み取り操作のすべてを使用することはできないものもあります。該当の高水準言語で使用できる操作を調べるには、その言語の手引きを参照してください。

読み取り操作に関する以下のトピックを参照してください。

- 『Read next 操作』
- 『Read previous 操作』
- 『Read first 操作』
- 『Read last 操作』
- 『Read same 操作』
- 『キーによる読み取り操作』
- 181 ページの『相対レコード番号による読み取り操作』
- 181 ページの『論理ファイルがさらに多くのキーとアクセス・パスを共用する時の読み取り操作』

Read next 操作: キー順アクセス・パス上の次のレコードを取り出します。レコード様式名が指定されている場合には、キー順アクセス・パスの中にあってそのレコード様式に一致する次のレコードが取り出されます。次のレコードを見つけるために、ファイル内での現在の位置が使用されます。(RPG/400 言語の READ ステートメントおよび COBOL/400 言語の READ NEXT ステートメントがこの例です。)

Read previous 操作: キー順アクセス・パス上の前のレコードを取り出します。レコード様式名が指定されている場合には、キー順アクセス・パスの中にあってそのレコード様式に一致する前のレコードが取り出されます。前のレコードを見つけるために、ファイル内での現在の位置が使用されます。(RPG/400 の言語 READP ステートメントおよび COBOL/400 言語の READ PRIOR ステートメントがこの例です。)

Read first 操作: キー順アクセス・パス上の最初のレコードを取り出します。レコード様式名が指定されている場合には、アクセス・パス上にあって、指定されたレコード様式名を持つ最初のレコードが取り出されます。

Read last 操作: キー順アクセス・パス上の最後のレコードを取り出します。レコード様式名が指定されている場合には、アクセス・パス上にあって、指定されたレコード様式名を持つ最後のレコードが取り出されます。

Read same 操作: 現在のファイル位置によって識別されるレコードを取り出します。ファイル内での位置は変更されません。

キーによる読み取り操作: キー値によって識別されるレコードを取り出します。キーに等しい、キーに等しいかその後、キーに等しいかその前、以前のキー読み取り (等しい)、次のキー読み取り (等しい、後、または前) のキー操作を指定できます。様式名が指定されている場合には、システムは指定されたキー値とレコード様式名を持つレコードを検索します。様式名が指定されていない場合には、指定のキー値を見つけるためにキー順アクセス・パス全体が検索されます。ファイルのキー定義に複数のキー・フィールドが含まれている場合には、部分キーを指定できます (キー・フィールドの数または使用するキー長のいずれかを指定

できます)。このようにすると、総称キー検索を行うことができるようになります。プログラムにキー・フィールドの数が指定されていない場合には、システムは、省略時のキー・フィールド数が指定されているとみなします。この省略時の値は、レコード様式名がプログラムから渡されたかどうかによって異なります。レコード様式名が渡された場合には、省略時のキー・フィールド数は、そのレコード様式名に定義されているキー・フィールドの合計数です。レコード様式名が渡されない場合には、省略時のキー・フィールド数は、アクセス・パス上のすべてのレコード様式に共通のキー・フィールドの最大数です。プログラムは、システムによって仮定されるキー・フィールド数に適合する十分なキー・データを提供しなければなりません。(RPG/400 言語の CHAIN ステートメントおよび COBOL/400 言語の READ ステートメントがこの例です。)

相対レコード番号による読み取り操作: キー順アクセス・パスの場合には、相対レコード番号を使用できます。この相対レコード番号は、オープンされたメンバーがキー順アクセス・パスを持つ場合でも、到着順の番号です。メンバーに複数のレコード様式がある場合、レコード様式名を指定しなければなりません。この場合には、関連する物理ファイル・メンバーの中にあって指定のレコード様式と合致するレコードを要求することになります。オープンされたメンバーが選択/除外ステートメントを含んでいて、相対レコード番号で識別されたレコードがキー順アクセス・パスから除外されている場合には、プログラムにエラー・メッセージが送られ、操作は行われません。操作が完了した後、ファイルは、相対レコード番号によって識別された物理レコードの中に含まれているキー順アクセス・パスのキー値に位置付けられます。キー順アクセス・パスの中にあって、現在のファイル位置よりレコード番号がいくつか大きいかまたは小さいレコードも、この操作によって取り出されます。(RPG/400 言語の CHAIN ステートメントおよび COBOL/400 言語の READ ステートメントがこの例です。)

論理ファイルがさらに多くのキーとアクセス・パスを共用する時の読み取り操作: FIFO、LIFO または FCFO キーワードが論理ファイルのデータ記述仕様 (DDS) 内で指定されていないとき、論理ファイルは、作成されている論理ファイルより多いキーを持つアクセス・パスを暗黙のうちに共用できます。このように、既存のアクセス・パスのキーの部分的な共用をするキー順アクセス・パスを使用してデータベースの読取操作を行う際に問題が生じることがあります。次のような問題が生じるかもしれません。

- 読み取られるべきレコードが、ユーザーのプログラムへ戻らない。
- レコードが、何回もユーザーのプログラムへ戻ってしまう。

実際に起こっている現象は、ユーザーのプログラムまたは現在活動している他のプログラムが、部分的に共用されたキー順アクセス・パス内のキーである物理ファイル・フィールドを更新中にもかかわらず、そのキーがユーザーのプログラムに使用されている論理ファイルの実際のキーではないということです。(更新されているフィールドは、ユーザーのプログラムに使用されている論理ファイルに知らされているキー数を超えています。) ユーザーのプログラムまたは他のプログラムによる論理ファイルの実際のキー・フィールドの更新は、常に上記の結果を生じます。部分的に共用されたキー順アクセス・パスとの相違点は、論理ファイルに知らされている数を超えたキーである物理ファイル・フィールドの更新が、同じ結果を生じさせることがあるという点です。

部分的に共用されたキー順アクセス・パスによりもたらされたそれらの結果が受け入れられない場合、FIFO、LIFO または FCFO キーワードを論理ファイルの DDS へ加え、論理ファイルをもう一度作成することができます。

ファイル終わりに達した時のレコード待ち: ファイル終わり遅延は、ファイル終わり状態になった後で、データベース・ファイル (論理ファイルまたは物理ファイル) からの順次読み取りを続行する方法です。順次読み取り (たとえば、次の/前のレコード) を実行中のファイルでファイル終わり状態になり、ファイル終わり遅延時間を (データベース・ファイルによる一時変更 [OVRDBF] コマンドの EOFDLY パラメーターで) 指定している場合には、システムは指定された時間だけ待機します。遅延時間が終わると、ファイルに追加された新しいレコードがあるかどうかを調べるために、別の読み取りが行われます。レコードが追加さ

れた場合には、再びファイル終わり状態になるまで、通常のレコード処理が行われます。ファイルに追加されたレコードがない場合には、システムは再び指定された時間だけ待ちます。キー順アクセス・パスを使用しないようにオープンされた選択/除外仕様のある論理ファイルに対してファイル終わり遅延を使用する場合には、特別の考慮が必要です。この場合、ファイルの終わりに達すると、システムは、基礎となる物理ファイルに追加されたレコードのうち、論理ファイルの選択/除外仕様に一致するものだけを取り出します。

さらに、キー順アクセス・パスを使用するようにオープンされたキー順アクセス・パスを持つファイルに対してファイル終わり遅延を使用する場合にも、特別の考慮が必要です。この場合、ファイルの終わりに達すると、システムは、ファイルに追加されたレコードまたはファイル中で更新されたレコードのうち、キー順アクセス・パスを使用する読み取り操作の仕様に一致するものだけを取り出します。

たとえば、昇順の数値キー・フィールドを持つキー付きファイルでファイル終わり遅延が使用されます。アプリケーション・プログラムはキー順アクセス・パスを使用してファイル中のレコードを読み取ります。アプリケーション・プログラムは次の読み取り操作を行い、キー値が 99 であるレコードを取り出します。アプリケーション・プログラムがもう一度次の読み取りを行って、ファイルにもはやレコードが見つからなくなると、システムは指定のファイル終わり遅延時間の経過後に再度ファイルの読み取りを試みます。ファイルに追加されたレコードまたは更新されたレコードがあって、そのレコードのキー値が 99 より小さい場合には、システムはレコードを取り出しません。ファイルに追加されたレコードまたは更新されたレコードがあって、そのレコードのキー値が 99 以上の場合には、システムはそのレコードを取り出します。

ファイル終わり遅延時間が 10 秒以上の場合には、ジョブは、待ち時間の間、主記憶域から除外適格となります。ジョブを主記憶域から除外適格としたくない場合には、ジョブが使用するクラスに関するクラスの作成 (CRTCLS) コマンドで PURGE(*NO) と指定してください。

ファイル終わり遅延が行われているジョブを示すために、「活動ジョブ処理 (WRKACTJOB)」画面の状況フィールドに、レコード待機中のジョブのファイル終わり待ちまたはファイル終わり活動レベルが表示されます。

ジョブがファイル終わり遅延およびコミットメント制御を使用している場合には、レコード・ロックを保持する時間が長くなることがあります。そのため、他のジョブが同じレコードをアクセスしようとして、ロックされる頻度が増えます。したがって、同じジョブの中でファイル終わり遅延とコミットメント制御の両方を使用する場合には注意が必要です。

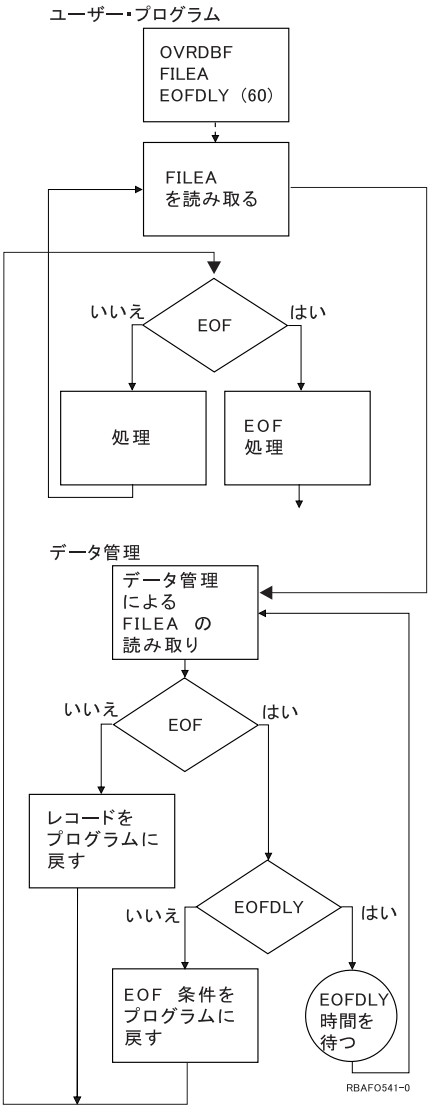
ファイルを共用する場合には、共用ファイルのオープン後に指定した一時変更は無視されるので、ファイル終わり遅延を指定するデータベース・ファイルによる一時変更 (OVRDBF) コマンドは、ファイルの*最初の*オープンより前に要求しなければなりません。

データベース・ファイルによる一時変更 (OVRDBF) コマンドでファイル終わり遅延を指定しているためにレコード待ちとなっているジョブを終了するには、以下のいくつかの方法があります。

- ファイル終わり遅延を指定したファイルに、アプリケーション・プログラムによって最後のレコードとして認識されるレコードを書き込む。アプリケーション・プログラムには、その後にデータ強制終了 (FEOD) 操作を指定しておくことができます。FEOD 操作によって、プログラムは通常のファイル終わり遅延を完了できます。
- ジョブの終了 (ENDJOB) コマンドで OPTION(*CNTRLD) を指定し、DELAY パラメーターに EOFDLY 時間より長い時間を指定することによって、ジョブの強制終了を行う。DELAY パラメーターに指定する時間には、EOFDLY 時間が経過するための時間、ファイルに入れられた新しいレコードをすべて処理するための時間、およびアプリケーションで必要とするファイル終了処理を行うための時間を見込んでおかなければなりません。新しいレコードの処理の後、システムはファイルの終わりを知らせ、通常のファイル終わり状態となります。

- ジョブの終了 (ENDJOB) コマンドに OPTION(*IMMED) を指定する。ファイル終わり処理は行われません。
- ジョブが対話式の場合には、システム要求キーを押して、最後の要求を終了する。

以下に示すものは、ファイル終わり遅延操作の例です。



長い遅延時間を指定してジョブの終了 (ENDJOB) コマンドで OPTION(*CNTRLD) を使用した場合には、真のファイル終わりが強制されることがあるため、EOFDLY パラメーターの実際の処理は、上記の説明より複雑になります。

ジョブは、ファイルに新しいレコードが追加された時点では活動状態になりません。指定したファイル終わり遅延時間が終わった後、ジョブは活動状態になります。ジョブが活動状態になると、システムは新しいレコードがあるかどうかを検査します。新しいレコードが追加されている場合には、アプリケーション・プログラムが制御権を受け取って、新しいレコードをすべて処理し、その後で再び待ちます。このため、ジョブは実行中にはバッチ・ジョブの性質を帯びます。たとえば、ジョブは通常、要求をバッチとして処理します。バッチが完了すると、ジョブは非活動状態になります。遅延が短い場合には、ジョブの開始および新しいレコードの検査に必要な内部処理のため、システム・オーバーヘッドが過剰になることがあります。通常は、ファイル終わり遅延を待つジョブには小さなオーバーヘッドしか使用されません。

注: ジョブが非活動 (待ち) 状態のときには、そのジョブは長時間待ち状態にあります。これは、そのジョブが活動レベルから解放されていることを意味します。長い待ち時間が経過した後で、システムはこのジョブを活動レベルにスケジュールし直します。(活動レベルについて詳しくは、実行管理機能を参照してください。)

ロックされたレコードの解放: レコードが更新されたり、削除されたり、またはファイル内の別のレコードが読み取られたりすると、システムは自動的にレコード・ロックを解除します。しかし、こうした操作を行わなくてもレコード・ロックを解除できます。高水準言語によっては、レコード・ロックの解除をサポートしているものがあります。レコード・ロックの解除の詳細については、該当の高水準言語の手引きを参照してください。

注: ジョブがコミットメント制御下で実行中の場合には、ロックの規則が異なります。詳しくは、 コミットメント制御のトピックを参照してください。

データベース・レコードの更新

更新操作によって、論理ファイルまたは物理ファイル中の既存のデータベース・レコードを変更できます。(RPG/400 言語の UPDAT ステートメントおよび COBOL/400 言語の REWRITE ステートメントがこのタイプの操作の例です。) データベース・レコードを更新する前に、まずレコードを読み取り、ロックしなければなりません。 179 ページの『到着順アクセス・パスを使用したデータベース・レコードの読み取り』、および 180 ページの『キー順アクセス・パスを使用したデータベース・レコードの読み取り』にリストされている読み取り操作のいずれかで更新オプションを指定すると、ロックが獲得されます。

更新オプションを指定して複数の読み取り操作を出す場合には、読み取り操作ごとに、前のレコードのロックが解除され、その後、新しいレコードが見つけられてロックされます。更新操作を行う場合には、システムは現在ロックされているレコードを更新するものとみなします。したがって、更新操作では更新するレコードを指定する必要はありません。更新操作が行われた後、システムはロックを解除します。

注: ジョブがコミットメント制御下で実行中の場合には、ロックの規則が異なります。詳しくは、 コミットメント制御のトピックを参照してください。

即時メインテナンスが指定されているアクセス・パスの中のキー・フィールドが更新操作によって変更されると、高水準言語が認める場合にはアクセス・パスが更新されます。(高水準言語によっては、更新操作でキー・フィールドを変更することを認めないものがあります。)

すでに更新のためにロックされているレコードに関する読み取り操作を要求し、かつユーザーのジョブが *ALL または *CS (カーソルの安定度) のコミットメント制御レベル下で作動している場合には、レコードのロックが解除されるか、あるいはファイルの作成コマンドまたは一時変更コマンドの WAITRCD パラメーターに指定した時間が経過するまで待たなければなりません。ロックが解除されずに WAITRCD 時間が経過した場合には、プログラムに例外が戻され、ファイル、メンバー、相対レコード番号、およびロックを行っているジョブを示すメッセージが、ジョブに送られます。レコードを読んでいるジョブがコミットメント制御レベル *ALL または *CS で作動していない場合、そのジョブは更新のためにロックされているレコードを読むことができます。

更新しようとしているファイルがそれに関連した更新トリガーを持っている場合は、レコードの更新の前または後にトリガー・プログラムが呼び出されます。トリガー・プログラムについての詳細は、 255 ページの『データベース内での自動イベントのトリガー』を参照してください。

更新しようとするファイルが参照制約と関連している場合は、更新操作に影響が出ることがあります。参照制約についての詳細は、 243 ページの『参照制約を使用したデータの保全性の保証』を参照してください。

データベース・レコードの追加

物理データベース・ファイル・メンバーに新しいレコードを追加するには、書き込み操作を使用します。(RPG/400 言語の WRITE ステートメントおよび COBOL/400 言語の WRITE ステートメントがこの操作の例です。) 新しいレコードは、物理ファイル・メンバー、あるいは物理ファイル・メンバーが基礎となっている論理ファイル・メンバーに追加できます。複数様式の論理ファイルを使用している場合には、システムにレコードの追加先の物理ファイル・メンバーを知らせるためにレコード様式名を指定しなければなりません。

通常、新しいレコードは物理ファイル・メンバーの終わりに追加されます。次に使用可能な相対レコード番号 (削除済みレコードも含む) が新しいレコードに割り当てられます。高水準言語によっては、削除済みレコードの位置に新しいレコードを書き込むことができる場合もあります。(たとえば、ファイル編成を RELATIVE と定義している場合の COBOL/400 の WRITE ステートメント) 削除済みレコード位置へのレコードの書き込みの詳細については、該当の高水準言語の手引きを参照してください。

レコードを追加する物理ファイルが削除済みレコードを再使用する場合には、システムはレコードを削除済みのレコードが占めていたスペースに挿入しようと試みます。削除済みレコードを再使用するようにファイルを作成したり、変更したりする前に、制約事項や使用法のヒントをよく検討し、そのファイルが削除済みレコードのスペースを再使用するための候補となるかを決めなければなりません。削除済みレコードのスペースの再使用の詳細については、104 ページの『削除済みレコードの再使用』を参照してください。

キー順アクセス・パスを持つファイル・メンバーに新しいレコードを追加する場合には、新しいレコードは、キー順アクセス・パスの中でレコード・キーによって定義された位置にすぐに現れます。選択/除外値を含む論理メンバーにレコードを追加する場合には、メンバーのアクセス・パスに新しいレコードが現れてくるのを除外値で防止できます。

追加しようとするファイルがそれに関連した挿入トリガーを持っている場合には、レコードの挿入の前または後にトリガー・プログラムが呼び出されます。トリガー・プログラムについての詳細は、255 ページの『データベース内での自動イベントのトリガー』を参照してください。

追加しようとするファイルが参照制約と関連している場合は、レコード挿入に影響が出ることがあります。参照制約についての詳細は、243 ページの『参照制約を使用したデータの保全性の保証』を参照してください。

物理ファイル・メンバーに追加することのできるレコードの数は、物理ファイルの作成 (CRTPF) コマンドおよびソース・ファイルの作成 (CRTSRCPF) コマンドの SIZE パラメーターによって決まります。

レコードの追加に関する詳細は、以下のトピックを参照してください。

- 『複数の様式を持つファイルに追加するレコード様式の識別』
- 187 ページの『データ強制終了操作の使用』

複数の様式を持つファイルに追加するレコード様式の識別: アプリケーションがデータベースに追加されるレコードに対してレコード様式名の代わりにファイル名を使用しており、しかも使用するファイルが複数のレコード様式を持つ論理ファイルである場合には、データベースのどこにレコードを入れるかを決めるために、様式選択プログラムを作成する必要があります。様式選択プログラムは、CL プログラムまたは高水準言語プログラムとして作成できます。

以下の条件がすべて真の場合には、様式選択プログラムを使用しなければなりません。

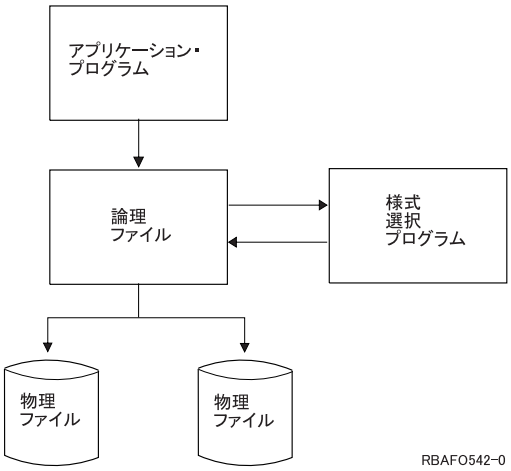
- 論理ファイルが結合論理ファイルではなく、ビュー論理ファイルでもない。
- 複数の物理ファイルが論理ファイルの基礎となっている。

- プログラムは追加操作でレコード様式名の代わりにファイル名を使用している。

この状況で様式選択プログラムを作成しない場合、アプリケーション・プログラムは、データベースにレコードを追加しようとした時点でエラーにより終了します。

注: ファイルに複数のメンバーがある場合は、様式選択プログラムを使用してメンバーを選択できません。様式選択プログラムはレコード様式の選択しかできません。

アプリケーション・プログラムがデータベース・ファイルにレコードを追加しようとすると、システムは様式選択プログラムを呼び出します。様式選択プログラムは、レコードを調べて、使用するレコード様式を指定します。その後で、システムは、指定されたレコード様式名を使用してデータベース・ファイルにレコードを追加します。



以下の例は、RPG/400 言語で作成された様式選択プログラムのプログラミング・ステートメントを示しています。

```
CL0N01N02N03Factor1+++OpcdeFactor2+++ResultLenDHHiLoEqComments+++...
++++*
C           *ENTRY    PLIST
C                     PARM           RECORD 80
C* The length of field RECORD must equal the length of
C* the longest record expected.
C                     PARM           FORMAT 10
C                     MOVELRECORD    BYTE    1
C           BYTE      IFEQ 'A'
C                     MOVEL'HDR'     FORMAT
C                     ELSE
C                     MOVEL'DTL'     FORMAT
C                     END
```

様式選択プログラムは最初のパラメーターでレコードを受け取るので、このフィールドでは、様式選択プログラムが予測する最長のレコードの長さであることを宣言しなければなりません。様式選択プログラムは、レコードのどの部分にもアクセスしてレコード様式名を調べることができます。この例では、様式選択プログラムはレコードの最初の文字が A であるかどうかを検査します。最初の文字が A である場合には、様式選択プログラムはレコード様式名 HDR を 2 番目のパラメーター (FORMAT) に移します。最初の文字が A でない場合には、様式選択プログラムはレコード様式名 DTL を 2 番目のパラメーターに移します。

様式選択プログラムは、10 文字のフィールドである 2 番目のパラメーターを使用して、システムにレコード様式名を渡します。レコード様式の名前が分かると、システムはデータベースにレコードを追加します。

以下の場合には、様式選択プログラムは必要ありません。

- 更新だけを行う場合。更新の場合、レコードはプログラムによってすでに取り出されており、そのレコードの取り出し元の物理ファイルがシステムに知られています。
- アプリケーション・プログラムが、追加または削除操作についてファイル名でなくレコード様式名を指定している場合。
- アプリケーション・プログラムで使用するすべてのレコードが、1 つの物理ファイルに入っている場合。

様式選択プログラムを作成するには、アプリケーション・プログラムの作成に使用した言語のプログラム作成コマンドを使用します。作成コマンドには USRPRF(*OWNER) を指定できません。様式選択プログラムは、所有者のユーザー・プロファイルではなく、ユーザーのユーザー・プロファイルの下で実行しなければなりません。

さらに、機密保護および保全性のため、およびパフォーマンスに大きい影響を与えるので、様式選択プログラムの中で呼び出しや入出力操作を行ってはなりません。

様式選択プログラムの名前は、論理ファイルの作成 (CRTLF) コマンド、論理ファイルの変更 (CHGLF) コマンド、またはデータベース・ファイルによる一時変更 (OVRDBF) コマンドの FMTSLR パラメーターで指定します。様式選択プログラムは、ファイルの作成時に存在している必要はありませんが、アプリケーション・プログラムの実行時には存在していなければなりません。

データ強制終了操作の使用: データ強制終了 (FEOD) 操作を用いると、プログラムがファイルに加えたすべての変更を補助記憶域に強制書き込みすることができます。通常、補助記憶域に変更を強制書き込みする時は、システムが決めます。しかし、FEOD 操作を使用すると、すべての変更を確実に補助記憶域に強制書き込みすることができます。

データの強制終了 (FEOD) 操作を使用すると、ファイルが入力操作のためにオープンされる場合に、ファイルの先頭と終わりのどちらかに位置決めすることもできます。*START により、メンバーの最初のレコードの直前に、現在オープンしているデータベース・ファイルの先頭または開始位置を設定します。(最初の順次読み取り操作は、現行のメンバーの最初のレコードを読み取ります。) MBR(*ALL) 処理がデータベース・ファイルによる一時変更 (OVRDBF) コマンドに有効な場合、前のレコードの読み取り操作は、1 つ前のメンバーの最後のレコードを読み取ります。前のレコードの読み取り操作がなされ、1 つ前のメンバーが存在していない場合、ファイルの終わりメッセージ (CPF5001) が送られます。*END は、現在オープンしているデータベース・ファイル・メンバーの位置をそのメンバーの最後のレコードの直後に設定します (前のレコードの読み取り操作は、現在のメンバーの最後のレコードを読み取ります)。MBR(*ALL) 処理がデータベース・ファイルによる一時変更 (OVRDBF) コマンドで有効な場合、次のレコードの読み取り操作は、次のメンバーの最初のレコードを読み取ります。次の読み取り操作がなされて、次のメンバーが存在しない場合、ファイルの終わりメッセージ (CPF5001) が発生します。

ファイルに削除トリガーがある場合、データの強制終了操作はできません。トリガーについての詳細は、255 ページの『データベース内での自動イベントのトリガー』を参照してください。ファイルが参照の親関係の一部である場合は、FEOD 操作はできません。参照制約についての詳細は、243 ページの『参照制約を使用したデータの保全性の保証』を参照してください。

FEOD 操作の詳細については、該当の高水準言語の手引きを参照してください (高水準言語によっては、FEOD 操作をサポートしないものもあります) 。

データベース・レコードの削除

削除操作を用いると、既存のデータベース・レコードを削除することができます。 (この操作の例には、RPG/400 言語の DELET ステートメント、および COBOL/400 言語の DELETE ステートメントがあります。) データベース・レコードを削除するには、まずレコードを読み取ってロックしなければなりません。レコードは、 179 ページの『到着順アクセス・パスを使用したデータベース・レコードの読み取り』または 180 ページの『キー順アクセス・パスを使用したデータベース・レコードの読み取り』でリストされている読み取り操作のいずれかで更新オプションを指定することによってロックされます。削除のためのレコードのロック方法および削除するレコードの指定の規則は、更新操作の場合と同じです。

注: 高水準言語によっては、最初にレコードを読み取る必要のない ものがあります。こうした言語では、削除したいレコードを単に削除ステートメントに指定するだけです。たとえば、RPG/400 言語では、レコードを最初に読み取らずに削除できます。

データベース・レコードを削除すると、物理レコードには削除済みのマークが付けられます。これは、削除操作が論理ファイルを通して行われた場合にも当てはまります。削除されたレコードを読み取ることはできません。レコードは、それが入っているすべてのキー順アクセス・パスから除去されます。削除済みレコードの相対レコード番号は同じままです。物理ファイル・メンバー中の他のすべての相対レコード番号も変更できません。

削除されたレコードが使用していたスペースはファイルの中に残りますが、下記の時点まで再使用されません。

- 物理ファイル・メンバーの再編成 (RGZPFM) コマンドを実行して、ファイル・メンバー中のこれらのスペースを圧縮して開放する時点。このコマンドの詳細については、 196 ページの『物理ファイルの再編成』を参照してください。
- プログラムが、削除されたレコードの相対レコード番号と同じ相対レコード番号を使用して、ファイルにレコードを書き込む時点。

注: ファイルに削除済みレコード・スペースの再使用属性がある場合には、システムは自動的に削除済みレコード・スペースを再使用しようとします。詳細については、 104 ページの『削除済みレコードの再使用』を参照してください。

システムは、削除済みレコードのデータの検索を認めません。しかし、削除済みレコードに対応する位置 (相対レコード番号) に新しいレコードを書き込むことは可能です。書き込み操作によって、新しいレコードが削除済みレコードに置き換わります。ファイル中の特定の位置 (相対レコード番号) にレコードを書き込む方法の詳細については、該当の高水準言語の手引きを参照してください。

削除済みレコードの相対レコード番号にレコードを書き込むためには、その相対レコード番号が物理ファイル・メンバーの中に存在していなければなりません。ファイル中のレコードは、高水準言語で削除操作を使用して削除できます。物理ファイル・メンバーの初期設定 (INZPFM) コマンドを使用して、ファイルからレコードを削除できます。INZPFM コマンドを使用すれば、レコードを削除するために物理ファイル・メンバー全体を初期設定できます。INZPFM コマンドの詳細については、 195 ページの『物理ファイル・メンバーのデータの初期設定』を参照してください。

削除を行おうとしているファイルがそれに関連した削除トリガーを持っている場合は、レコードの削除の前または後にトリガー・プログラムが呼び出されます。トリガーについての詳細は、 255 ページの『データベース内での自動イベントのトリガー』を参照してください。

ファイルが参照制約関係の一部である場合は、レコード削除に影響が出ることがあります。参照制約についての詳細は、 243 ページの『参照制約を使用したデータの保全性の保証』を参照してください。

データベース・ファイルのクローズ

プログラムは、データベース・ファイル・メンバーの処理を完了したときに、ファイルをクローズしなければなりません。データベース・ファイルをクローズすると、プログラムがファイルから切り離されます。クローズ操作は、すべてのレコード・ロックおよびすべてのファイル・メンバー・ロックを解除し、オープン・データ・パス (ODP) を介してなされたすべての変更を補助記憶域に書き込み、その後で ODP を破壊します。(共用ファイルをクローズしたものの、ODP がオープンしたままになっている場合は、機能が異なります。共用ファイルの詳細については、111 ページの『同一ジョブまたは活動化グループ内のデータベース・ファイルの共用』を参照してください。)

プログラムの中でデータベース・ファイルをクローズするには、以下のメソッドのうちのいずれかを使用します。

- 高水準言語のクローズ・ステートメント

  ほとんどの高水準言語では、データベース・ファイルのクローズを指定できます。高水準言語プログラムでデータベース・ファイルをクローズする方法の詳細については、該当の高水準言語の手引きを参照してください。
- ファイル・クローズ (CLOF) コマンド

  ファイル・クローズ (CLOF) コマンドを使用すると、データベース・ファイル・オープン (OPNDBF) コマンドまたは Query ファイルのオープン (OPNQRYF) コマンドでオープンされたデータベース・ファイルをクローズできます。 制御言語 (CL) に関するトピックの CLOF (ファイルのクローズ) コマンドを参照してください。
- 資源再利用 (RCLRSC) コマンド

  RCLRSC コマンドは、すべてのロック (コミットメント制御下では、変更されたものの、まだコミットされていないレコードのロックを除く) を解除し、すべての変更を補助記憶域に書き込み、その後で該当ファイルのオープン・データ・パスを破壊します。 制御言語 (CL) に関するトピックの RCLRSC (資源再利用) コマンドを参照してください。 RCLRSC コマンドを使用して、呼び出されたプログラムのファイルを呼び出し側プログラムにクローズさせることができます。 (たとえば、呼び出されたプログラムがそのファイルをクローズせずに呼び出しプログラムに戻った場合、呼び出しプログラムは呼び出されたプログラムのファイルをクローズできます。) しかし、通常は高水準言語のクローズ操作またはファイル・クローズ (CLOF) コマンドを使用して、プログラムのファイルをクローズします。統合言語環境における資源のレクラメーション処理について詳しくは、「ILE 概念  」を参照してください。

ジョブが正常に (たとえば、ユーザーのサインオフによって) 終了し、このジョブと関連するすべてのファイルがクローズされなかった場合、システムは、このジョブと関連する残りのオープン・ファイルをすべて自動的にクローズし、すべての変更を補助記憶域に強制的に書き込み、それらのファイルのすべてのレコード・ロックを解除します。ジョブが異常終了した場合、システムは、そのジョブと関連するすべてのファイルをクローズし、それらのファイルのすべてのロックを解除し、すべての変更を補助記憶域に強制的に書き込みます。

あるプロセスが、別のプロセスによってホールドされているファイルをロックしようとすると、データベース・ファイルをクローズする出口プログラムが呼び出されます。この出口ルーチンは、ロックをホールドしているプロセスの中で呼び出されます。詳しくは、API reference を参照してください。

プログラム内でのデータベース・ファイル・エラーのモニター

データベース・アプリケーションがデータベース・ファイルを処理する場合、プログラムが検出するファイル・エラーのメッセージのモニターを行う必要があります。それにより、これらのエラーを防ぐための適切

な行動を取ることができます。高水準言語はそれぞれ、これらのメッセージをモニターする独自のプロシージャーを持っているので、使用している言語の資料を参照してエラー・メッセージ・モニターを導入してください。

データベース・ファイルの処理中にエラーが検出された場合、以下のようなイベントが起こります。

- ファイルを処理しているプログラムのプログラム・メッセージ待ち行列にメッセージが送られる。
- システム操作員メッセージ待ち行列に照会メッセージが送られる。
- プログラムのファイル・フィードバック域の中にファイル・エラーおよび診断情報が戻りコードおよび状況情報として示される。

たとえば、COBOL 言語では、戻りコード (プログラムの中に定義されている場合) をファイル状況フィールドにセットします。

詳しくは、以下のトピックを参照してください。

- 『エラー・メッセージのシステム処理』
- 『ファイルの位置決めに対するエラー・メッセージの影響』
- 『モニターを行うメッセージの決定』

エラー・メッセージのシステム処理

メッセージをモニターしない場合は、システムがエラーを処理します。また、システムは、プログラムの中に該当のエラー戻りコードをセットします。エラーに応じて、システムはジョブを終了させたり、または操作員にメッセージを送って処置を要求したりすることがあります。

ファイルの位置決めに対するエラー・メッセージの影響

データベース・ファイル・メンバーの処理中にプログラムにメッセージが送られても、ファイル中の位置は失われません。以下の場合を除いて、ファイルの位置はメッセージが送られる前のレコードと同じ位置になっています。

- ファイルの終わり条件に達してプログラムにメッセージが送られた後には、ファイルは *START または *END に位置付けられます。
- 読み取り操作の変換マッピング・メッセージの後には、ファイルは、メッセージの原因となったデータが入っているレコードに位置付けられます。

モニターを行うメッセージの決定

エラー・メッセージのモニターが可能なプログラミング言語の場合には、モニターの対象としたいメッセージを選択できます。モニターできるエラー・メッセージのいくつかの例を以下に示します。どのメッセージをモニターできるかについては、ご使用の高水準言語の資料、または制御言語 (CL)に関するトピックのメッセージのモニターを参照してください。これらのメッセージの完全な記述を表示するには、メッセージ記述表示 (DSPMSGD) コマンドを使用してください。 制御言語 (CL) に関するトピックの DSPMSGD (メッセージ記述表示) コマンドを参照してください。

メッセージ識別コード	説明
CPF5001	ファイルの終わりに達した
CPF5006	レコードが見つからない
CPF5007	レコードが削除された
CPF5018	ファイルの最大サイズに達した
CPF5025	*START または *END を超えて読み取りが試みられた
CPF5026	重複キー
CPF5027	他のジョブによってレコードが使用中

メッセージ識別コード	説明
CPF5028	レコード・キーが変更された
CPF5029	データ・マッピング・エラー
CPF502B	トリガー・プログラム内のエラー
CPF502D	参照制約違反
CPF5030	メンバーに部分的損傷
CPF5031	レコード・ロックが最大数を超えた
CPF5032	レコードはすでにジョブに割り振られている
CPF5033	選択/除外エラー
CPF5034	別のメンバーのアクセス・パスに重複キーがある
CPF503A	参照制約違反
CPF5040	除外されたレコードが取り出されなかった
CPF5072	メンバー内で結合基準が変更された
CPF5079	コミットメント制御の資源の限界を超えた
CPF5084	コミットされていないキーに対する重複キー
CPF5085	別のアクセス・パスのコミットされていないキーが重複している
CPF5090	固有アクセス・パスが正しくないため追加または更新が阻止された
CPF5097	キー・マッピング・エラー

データベース・ファイルの管理

以下のトピックでは、データベース・ファイルの管理について説明します。

192 ページの『データベース・ファイルを管理するための基本操作』
このセクションでは、基本ファイル操作を使用してデータベース・ファイルを管理する方法について説明します。

193 ページの『データベース・メンバーの管理』
このセクションでは、データベース・ファイル・メンバーを管理する方法について説明します。説明には、メンバーの追加、メンバー属性の変更、メンバーの名前変更、メンバーの除去などのトピックが含まれます。また、物理ファイルに固有のメンバー操作についても説明します。

201 ページの『データベース属性および相互参照情報の使用』
このセクションでは、データベース・ファイル、物理ファイル、および論理ファイルの、各ファイル記述およびファイル属性の変更方法について説明します。

208 ページの『データベース・ファイルの記述および属性の変更』
このセクションには、データベース・ファイル属性、フィールドの関係、相互参照情報を変更する方法、および、表示し使用する方法についての説明があります。さらに、コマンド出力を直接データベース・ファイルに書き込む方法の説明もあります。

212 ページの『データベースの回復と復元』
このセクションには、以下のトピックを含む、システム障害発生時のデータベース・ファイルの回復計画に関する説明があります。

- 保管と復元
- ジャーナリング
- 補助記憶域の使用
- コミットメント制御の使用

230 ページの『ソース・ファイルの使用』
このセクションでは、ソース・ファイルにデータを入力し保守する方法、さらに、そのソース・ファイルを使用してシステム上に別のオブジェクトを作成する方法について説明します。

237 ページの『制約によってユーザーのデータベースの保全性を制御する』
このセクションでは、レコードを追加、変更、除去しても、制約を使用してデータベース内のデータの一貫性を保つようにする方法について説明します。

243 ページの『参照制約を使用したデータの保全性の保証』
このセクションでは、データベース内で参照制約を使用してデータベースに有効なデータだけを入れるようにする方法について説明します。

255 ページの『データベース内での自動イベントのトリガー』
このセクションでは、指定した物理データベース・ファイルに指定した変更操作または読み取り操作が行われるときに自動的に実行される一連のアクションを開始するためのトリガーの使用について説明します。

283 ページの『データベースの配布』
このセクションには、別売のフィーチャーである DB2 マルチシステムの概要が記載されています。このフィーチャーによって、疎結合環境内の複数システムにデータベース・ファイルを分散する単純で直接的なメソッドが提供されます。

データベース・ファイルを管理するための基本操作

この章では、基本的なデータベース・ファイル操作のいくつかを説明します。以下のトピックを参照してください。

- 『ファイルのコピー』
- 193 ページの『ファイルの移動』

ファイルのコピー

iSeries ナビゲーターで表のコピー操作を使用してファイルをコピーできます。『iSeries ナビゲーターを使用したファイル (表) のコピー』を参照してください。あるいは、ファイルのコピー (CPYF) コマンドを使用できます。 193 ページの『CPYF を使用したファイルのコピー』を参照してください。

iSeries ナビゲーターを使用したファイル (表) のコピー: 表を別のスキーマにコピーすると、同じ表の 2 つのインスタンスが作成されます。表を別のスキーマにコピーするには、以下のようにします。

1. iSeries ナビゲーター・ウィンドウで、使用したいシステムを拡張する。
2. **データベース**を拡張する。
3. 操作したいデータベースおよびスキーマを拡張する。
4. 「**表**」コンテナーをクリックする。
5. 表を右マウス・ボタンでクリックして、「**コピー**」を選択する。
6. 表のコピー先のスキーマを右クリックして、「**貼り付け**」をクリックする。

分散データ管理 (DDM) がiSeries ナビゲーターによって使用され、表が実際に移動またはコピーされます。起動システムがバージョン 4 リリース 4 以降であり、受動システムがバージョン 4 リリース 2 以降である場合は、 DDM over TCP/IP を使用して操作が実行されます。これ以外の場合は、操作は DDM over SNA を使用して実行されます。DDM over SNA を使用して行われる移動またはコピーの場合は、iSeries アクセスがシステムとして認識できる名前は、 DDM が使用する APPC または APPN 装置記述に指定された遠隔ロケーション名と同じ名前でなければなりません。DDM over TCP/IP を使用した移動また

はコピーの場合は、 TCP 通信がシステム間で使用可能になっていなければなりません。TCP/IP の場合、TCP/IP は、iSeries アクセスに認識されているように、システム間で使用可能になっていることが重要です。詳しくは、分散データベース管理を参照してください。

CPYF を使用したファイルのコピー: ファイルのコピー (CPYF) コマンドは、データベースまたは外部装置ファイルのすべてまたは一部を、データベースまたは外部装置ファイルにコピーします。詳しくは、 制御言語 (CL) に関するトピックの CPYF (ファイルのコピー) コマンドを参照してください。

ファイルの移動

iSeries ナビゲーターの表の移動操作を使用して、ファイルを 1 つのライブラリーから別のライブラリーに移動できます。『iSeries ナビゲーターを使用したファイル (表) の移動』を参照してください。あるいは、オブジェクトの移動 (MOVOBJ) コマンドを使用できます。『MOVOBJ コマンドを使用したファイルの移動』を参照してください。

iSeries ナビゲーターを使用したファイル (表) の移動: ファイル (表) を別のライブラリーに移動するには、以下のようにします。

1. iSeries ナビゲーター・ウィンドウで、使用したいシステムを拡張する。
2. **データベース**を拡張する。
3. 操作したいデータベースおよびスキーマを拡張する。
4. 「**表**」コンテナーをクリックする。
5. 表を右マウス・ボタンでクリックして、「**切り取り**」を選択する。
6. 表の移動先のスキーマを右クリックして、「**貼り付け**」をクリックする。
7. あるいは、表をドラッグして、同じシステムまたは別のシステムの別のライブラリー上にドロップすることもできます。

注: 表を別の位置に移動しても、その表がソース・ステムから除去されるわけではありません。たとえば、ユーザーがソース表に対して読み取り権限を持っているが削除権限を持っていない場合、表はターゲット・システムに移動されますが、 ソース・システムから削除されず、表の 2 つのインスタンスが存在することになります。

分散データ管理 (DDM) がiSeries ナビゲーターによって使用され、表が実際に移動またはコピーされます。起動システムがバージョン 4 リリース 4 以降であり、受動システムがバージョン 4 リリース 2 以降である場合は、 DDM over TCP/IP を使用して操作が実行されます。これ以外の場合は、操作は DDM over SNA を使用して実行されます。DDM over SNA を使用して行われる移動またはコピーの場合は、iSeries アクセスがシステムとして認識できる名前は、 DDM が使用する APPC または APPN 装置記述に指定された遠隔ロケーション名と同じ名前でなければなりません。DDM over TCP/IP を使用した移動またはコピーの場合は、 TCP 通信がシステム間で使用可能になっていなければなりません。TCP/IP の場合、TCP/IP は、iSeries アクセスに認識されているように、システム間で使用可能になっていることが重要です。詳しくは、分散データベース管理を参照してください。

MOVOBJ コマンドを使用したファイルの移動: オブジェクトの移動 (MOVOBJ) コマンドは、オブジェクトを、いま割り当てられているライブラリーから取り外し、別のライブラリーに入れます。移動されるオブジェクトのタイプは、OBJTYPE パラメーターに指定します。詳しくは、 制御言語 (CL) に関するトピックの MOVOBJ (オブジェクトの移動) コマンドを参照してください。

データベース・メンバーの管理

ファイルに対して入力または出力操作を行うためには、ファイルにメンバーが少なくとも 1 つ含まれていなければなりません。一般に、データベース・ファイルにはメンバーが 1 つだけ、つまりファイルの作成

時に作成されたメンバーだけがあります。このメンバーの名前は、ユーザーが別の名前を与えていない限り、ファイル名と同じです。ほとんどの場合、データベース・ファイル操作で使用されるメンバーはファイルの最初のメンバーであるとみなされ、また、大半のファイルにはメンバーが 1 つしかないので、通常、ユーザーがメンバー名について考慮したり、メンバー名を指定したりする必要はありません。

ファイルにメンバーが複数個ある場合、おのおののメンバーはそのファイルのデータのサブセットとして機能します。このため、データの類別が簡単になります。たとえば、売掛勘定ファイルを定義するとします。このファイルではデータを 1 年間保持しますが、一度に 1 か月分だけのデータの処理がたびたび必要になるとします。たとえば、月ごとに名前を付けた 12 のメンバーを持つ物理ファイルを作成するとします。こうすると、各月のデータを個別に (メンバーごとに) 処理できます。また、メンバーのうちのいくつかを、あるいはすべてを一緒に処理することもできます。

メンバーの管理については、以下のトピックを参照してください。

- 『すべてのデータベース・ファイルに共通なメンバー操作』
- 195 ページの『物理ファイル・メンバー操作』

すべてのデータベース・ファイルに共通なメンバー操作

システムには、ファイル定義を変更する手段が用意されています。CL コマンドを使用して、これらの操作のほとんどを実行できます。これらのコマンドの詳細については、 制御言語 (CL) に関するトピックを参照してください。

以下のトピックを参照してください。

- 『ファイルへのメンバーの追加』
- 『メンバー属性の変更』
- 195 ページの『メンバーの名前変更』
- 195 ページの『ファイルからのメンバーの除去』

ファイルへのメンバーの追加: 以下に示すどれかの方法で、ファイルにメンバーを追加できます。

- 自動追加。ファイルの作成が、物理ファイルの作成 (CRTPF) コマンドまたは 論理ファイルの作成 (CRTLF) コマンドを使用して行われる場合は、デフォルトでは、新しく作成されたファイルにメンバー (ファイルと同じ名前で) が自動的に追加されます。(ソース・ファイルの作成 (CRTSRCPF) コマンドの省略時値では、新しく作成されたファイルにメンバーは追加されません。) データベース・ファイル作成コマンドで MBR パラメーターを使用すると、別のメンバー名を指定できます。ファイルの作成時にメンバーを追加する必要がない場合には、MBR パラメーターに *NONE を指定します。
- 個別追加。ファイルの作成後、物理ファイル・メンバーの追加 (ADDPFM) コマンド、または 論理ファイル・メンバーの追加 (ADDLFM) コマンドを使用してメンバーを追加することができます。
- ファイルのコピー (CPYF) コマンド。コピーしたいメンバーがコピー先のファイルに存在しない場合には、 CPYF コマンドによってそのメンバーがファイルに追加されます。

メンバー属性の変更: 物理ファイル・メンバーの変更 (CHGPFM) コマンド、または、 論理ファイル・メンバーの変更 (CHGLFM) コマンドを使用して、物理ファイル・メンバーまたは論理ファイル・メンバーの特定の属性を変更することができます。物理ファイル・メンバーの場合には、SRCTYPE (メンバーのソース・タイプ)、 EXPDATE (メンバーの満了日)、SHARE (メンバーがジョブ内で共用されるかどうか)、および TEXT (メンバーのテキスト記述) の各パラメーターを変更できます。論理ファイル・メンバーの場合には、SHARE パラメーターおよび TEXT パラメーターを変更できます。

注: 物理ファイルの変更 (CHGPF) コマンド、および、 論理ファイルの変更 (CHGLF) コマンドを使用して、その他の多くのファイル属性を変更できます。たとえば、ファイル内のメンバーごとに許可されている最大サイズを変更するには、CHGPF コマンドの SIZE パラメーターを使用します。

メンバーの名前変更: メンバーの名前変更 (RNMM) コマンドは、物理ファイルまたは論理ファイルの既存メンバーの名前を変更するものです。ファイル名は変更されません。

ファイルからのメンバーの除去: メンバーの除去 (RMVM) コマンドを使用して、メンバーとその内容を除去できます。メンバーのデータと、メンバーそのものが両方とも除去されます。メンバーを除去すると、そのメンバーはシステムでは使用できなくなります。メンバーの除去は、メンバーからのデータの消去または削除とは異なります。メンバーがまだ存在している場合、プログラムはメンバーの使用 (例: メンバーへのデータの追加) を続行できます。

物理ファイル・メンバー操作

以下の項で、物理ファイル・メンバーに固有のメンバー操作について説明します。このような操作には、物理ファイル・メンバー中のデータの初期設定、データの消去、データの再編成、およびデータの表示があります。以下のトピックを参照してください。

- 『物理ファイル・メンバーのデータの初期設定』
- 196 ページの『物理ファイル・メンバーの中のデータの消去』
- 196 ページの『物理ファイルの再編成』
- 201 ページの『物理ファイル・メンバーのレコードの表示』

操作しようとするファイル・メンバーが参照制約と関連がある場合は、その操作に影響が出ることがあります。参照制約についての詳細は、 243 ページの『参照制約を使用したデータの保全性の保証』を参照してください。

物理ファイル・メンバーのデータの初期設定: プログラムで相対レコード処理を使用するためには、データベース・ファイルのレコード桁数がプログラムで使用する最高位の相対レコード番号に対応している必要があります。相対レコード番号処理を使用するプログラムでは、これらのレコードの初期設定をしなければならないこともあります。

物理ファイル・メンバーの初期設定 (INZPFM) コマンドを使用して、以下の 2 つのタイプのレコードのどちらかを持つメンバーを初期設定することができます。

- 省略時のレコード
- 削除済みレコード

物理ファイル・メンバーの初期設定 (INZPFM) コマンドの RECORDS パラメーターには、使用したいレコードのタイプを指定できます。

省略時のレコードを用いてレコードを初期設定する場合には、新しい各レコードのフィールドは、ファイルの作成時に定義された省略時のフィールド値に初期設定されます。省略時のフィールド値が定義されていない場合には、数字フィールドにはゼロが埋め込まれ、文字フィールドにはブランクが埋め込まれます。

可変長の文字フィールドの省略時の値は、長さがゼロです。NULL 可能フィールドの省略時の値は、NULL です。省略時の値が定義されていない日付、時刻、および時刻スタンプは、処理当日の日付、時刻、および時刻スタンプです。プログラム記述ファイルの省略時の値は、全桁ブランクです。

注: 物理ファイル・メンバーまたは関連する任意の論理ファイル・メンバーの DDS に UNIQUE キーワードを指定した場合には、 1 つの省略時のレコードを初期設定できます。そうでない場合には、一連の重複するキー・レコードが作成されます。

レコードを省略時のレコードに初期設定する場合は、相対レコード番号でレコードを読み取り、データを変更できます。

レコードを削除済みレコードに初期設定してある場合には、削除済みレコードの 1 つの相対レコード番号を用いてレコードを追加することで、変更できます。(削除されていない相対レコード番号を使用してレコードを追加することはできません。)

削除済みレコードは読み取り不能なうえ、メンバー内で場所を取るだけです。削除済みレコードに新しいレコードを重ね書きすることによって、削除済みレコードを変更できます。削除済みレコードの処理についての詳細は、 188 ページの『データベース・レコードの削除』を参照してください。

物理ファイル・メンバーの中のデータの消去: 物理ファイル・メンバーの消去 (CLRPFM) コマンドを使用して、物理ファイル・メンバーの中のデータを消去できます。消去操作の完了後は、メンバー記述は残りますが、データは消失します。

物理ファイルの再編成: 以下のトピックは、 OS/400 で物理ファイルを再編成する方法を説明しています。

- 『iSeries ナビゲーターを使用した表の再編成』
- 197 ページの『RGZPFM を使用した物理ファイルの再編成』
- 198 ページの『使用上の注意: ファイルの再編成』
- 198 ページの『再編成のタイプ』
- 200 ページの『再編成の一時停止または取り消し』

***iSeries* ナビゲーターを使用した表の再編成:** 表を再編成することにより、理想的な物理編成に復元することができます。データベース表の理想的な編成とは、行がページを基にしてレイアウトされ、その配列が頻繁に使用される索引のキー値の順になっているものです。表の再編成は、削除済みレコードを圧縮する、表キーまたは選択された索引を使用するなどの方法によって行います。表を再編成するには、以下のようにします。

1. iSeries ナビゲーター・ウィンドウで、使用したいシステムを拡張する。
2. **データベース**を拡張する。
3. 操作したいデータベース・フォルダーを拡張する。
4. 「**スキーマ (Schemas)**」フォルダーを拡張する。
5. 再編成したい表が入っているスキーマをクリックする。
6. 「**表 (Tables)**」をクリックする。
7. 詳細ペインで、再編成したい表を右クリックし、「**再編成**」を選択する。

「**再編成**」ウィンドウで、表の行を再編成する方法を指定するために、以下のオプションのいずれか 1 つを選択します。

- 行の到着順を保持せずに削除済み行を圧縮する。削除済み行が存在しなくなるまで、表の末尾にある有効な行が削除済み行に移動されます。
- 削除済み行を圧縮し、行の到着順を保持する。すべての削除済み行を圧縮するために、表の最初の削除済み行の後にあるすべての有効な行が、表の先頭方向に移動されます。
- 表キーによる編成。表のアクセス・パスのキー値に応じて、表の行が再編成されます。表に基本キーが存在するか、キー順物理ファイルでなければなりません。
- ライブラリー内の選択された索引による再編成。指定した表に対して構築された索引またはキー順論理ファイルのキー値に応じて、表の行が再配列されます。選択できるのは既存の索引だけです。索引のリストは、選択するライブラリーによって決定されます。

| 再編成操作のパフォーマンスと並行性を制御するために、「再編成」ウィンドウで、以下のような他のオプ
| ションを指定できます。

| • 区画化されたファイルのどの区画 (または、複数メンバーの物理ファイルのどのメンバー) を再編成する
| かを指定する。
| • 再編成を一時停止して後で再開できるようにするかどうかを指定する。

| 再編成を一時停止可能として指定しない場合、表は再編成操作の期間中にのみ割り振られ、即座にジョ
| ブを終えることによってのみ一時停止可能になります。

| 再編成を一時停止可能として指定した場合、以下のようになります。

| – 再編成操作が一時停止された場合に行の脱落を防ぐ目的で、コミットメント制御のもとで行が移動さ
| れるため、ファイルのジャーナル処理が必要になります。
| – さらに、再編成中に他のユーザーが表を読み取ったり変更したりできるよう指定することもできま
| す。再編成中に、移動対象の行に対するロックが短時間だけ獲得されます。並行するジョブが行に対
| するロックを同時に獲得した場合、レコード・ロック・タイムアウトが発生する可能性があります。
| その場合には、ファイルのレコード・ロック待機時間を変更するか、 OVRDBF コマンドを使用して
| 適切なレコード・ロック待機時間を指定することができます。詳細については、 107 ページの『レコ
| ードのロック』を参照してください。

| • 索引を保守する方法を以下のように指定できます。
| – 再編成を一時停止可能として指定した場合、再編成中にすべての索引を保守するよう指定できます。
| 索引を再構築する必要はありません。
| – それ以外の場合には、索引を同期または非同期で再構築するよう指定できます。非同期で構築される
| 索引の進行状況を確認するには、 EDTRBDAP コマンドを使用できます。

RGZPFM を使用した物理ファイルの再編成: 物理ファイル・メンバーの再編成 (RGZPFM) コマンドを使用して、以下のことが行えます。

| • 削除済みレコードを除去して、これらのレコードが占めていたスペースを使用可能にする。
• レコード検索の所要時間を最小限に抑えるよう、ファイルのレコードを順次正常にアクセスできる順序で再編成する。これには、KEYFILE パラメーターを使用します。このような再編成は、主に到着順以外の順序でアクセスするファイルの場合に有利です。以下のどちらかを使用して、メンバーを再編成することができます。
  – 物理ファイルのキー・フィールド
  – 物理ファイルに基づく論理ファイルのキー・フィールド
• ソース・ファイル・メンバーの再編成、新しいソース順序番号の挿入、およびソース日付フィールドのリセット (物理ファイル・メンバー再編成コマンドの SRCOPT パラメーターおよび SRCSEQ パラメーターを使用して)。
| • 再編成の取り消しが不可能と指定した場合、物理ファイル形式の可変長フィールドによって以前に使用
| され、現在は細分化されている、ファイルの可変部分のスペースを再利用します。

物理ファイル・メンバーを再編成するコマンドを使用して物理ファイルを再編成する例については、 『例: 物理ファイルの再編成』 を参照してください。また、コマンドを使用するときの注意事項については、198 ページの『使用上の注意: ファイルの再編成』を参照してください。

例: 物理ファイルの再編成: たとえば、以下の物理ファイル・メンバーの再編成 (RGZPFM) コマンドでは、論理ファイルからのアクセス・パスを使用して物理ファイルの最初のメンバーが再編成されます。

```
RGZPFM  FILE(DSTPRODLB/ORDHDRP)
        KEYFILE(DSTPRODLB/ORDFILL  ORDFILL)
```

物理ファイル ORDHDRP には、到着順アクセス・パスがあります。これは、論理ファイル ORDFILL 内のアクセス・パスを使用して再編成されたものです。キー・フィールドは *Order* フィールドであるとします。レコードの配列方法を、以下に示します。

以下は、元の ORDHDRP ファイル例です。RGZPFM コマンドの実行前にレコード 3 は削除済みであることに注意してください。

相対レコード番号	Cust	Order	Ordate. . .
1	41394	41882	072480. . .
2	28674	32133	060280. . .
3	deleted	レコード	
4	56325	38694	062780. . .

以下の例は、*Order* フィールドを昇順のキー・フィールドとして使用して再編成された ORDHDRP ファイルを示しています。

相対レコード番号	Cust	Order	Ordate. . .
1	28674	32133	060280. . .
2	56325	38694	062780. . .
3	41394	41882	072480. . .

使用上の注意: ファイルの再編成:

1. 到着順アクセス・パスを持つファイルを、キー順アクセス・パスを用いて再編成する場合には、到着順アクセス・パスが変更されます。すなわち、ファイル内のレコードは物理的に、使用されるキー順アクセス・パスの順に並べられます。データを再編成して、使用しているキー順アクセス・パスとほとんど同じ順序で並べることにより、データの順次処理のパフォーマンスを向上させることができます。
2. ファイルを再編成することによって削除済みレコードが圧縮されるため、後続の相対レコード番号が変更されます。
3. FCFO、FIFO、または LIFO のどちらかの DDS キーワードが指定されたアクセス・パスは、物理ファイルの中のレコードの物理的順序によって異なるため、キー順アクセス・パスを用いて物理ファイルを再編成した後には、重複するキー・フィールドを持つレコードの順序が変わる場合があります。重複するキー・フィールドを持つレコードの順序は、 KEYFILE パラメーターで指定されたアクセス・パスに関してのみ保守されます。 KEYFILE パラメーターで指定されたアクセス・パスに LIFO DDS キーワードが含まれる場合、再編成が取り消し可能 (一時停止可能) と指定した場合に限って、重複するキー・フィールドが保守されます。
4. 再編成を取り消しできないと指定した場合、 RGZPFM コマンドを実行中のジョブを終了すると、物理ファイル・メンバーのすべてのアクセス・パスを再構築する必要が生じるかもしれません。再編成を取り消し可能と指定した場合、 RGZPFM コマンドを取り消すと、再編成中に保守されなかったアクセス・パスを再構築する必要が生じるかもしれません。
5. 1 つの行で RGZPFM コマンドを 2 度使用した場合、最初の操作後のファイルの合計サイズが、 2 度目の操作後の合計サイズと異なる場合があります。これは、再編成対象のファイルに割り振られるスペースが見積もりであり、今後の追加を予想して余分なスペースが含まれるためです。最初にレコードが再編成された後、割り振られるスペースは正確に計算されます。

再編成のタイプ: データを再編成する方法には、基本的に以下の 2 つがあります。

- ALWCANCEL(*NO) - これは、従来のタイプの再編成です。データの完全なコピーが取られる可能性があるため、最大で 2 倍のスペースが必要です。このオプションは取り消し (一時停止) できず、完全には並列で実行できません。ファイルの排他的使用を必要とします。

- ALWCANCEL(*YES) - データ行がファイル内で移動されるため、データの完全なコピーは必要ありません。ただしファイルがジャーナル処理される可能性があるため、ジャーナル項目用の記憶域が必要です。特定のジャーナル・レシーバーで使用される記憶域を最小限にとどめるために、ジャーナル・レシーバーしきい値を使用できます。

  このオプションは、取り消し (一時停止) して後で再開することができます。 DB2 UDB 対称マルチプロセス・オプションがインストール済みであれば、このオプションは並列で実行可能です。再編成操作で使用されるリソース量を制御するために、 CHGQRYA CL コマンド、または iSeries ナビゲーターの「QUERY 属性の変更 (Change Query Attributes)」を使用して、 QUERY の属性を変更できます。

  このオプションの場合、再編成が完了した後、記憶域をシステムに戻すために、ほんの数秒間にわたって排他的使用を必要とします。排他ロックを獲得できない場合、スペースを回復できなかったことを示す警告メッセージがジョブ・ログに送られます。スペースを回復するには、ファイルに並行アクセスするユーザーが存在しないときに、再編成を再び実行することができます。その場合、再編成操作が開始される前に、ただちにスペースの回復が試行されます。最初の再編成の後で並行的なデータ変更が発生した場合には、スペースの一部だけが回復される可能性があります。

  LOCK(*EXCLRD) または LOCK(*SHRUPD) を指定した場合、他のユーザーが並行的にファイルの行をロックまたは変更する可能性があるため、再編成の結果が正確になることは保障されません。

どのタイプの再編成を選択すべきかは、いくつかの要素に依存します。たとえば、単にスペースを回復することが目的か、それとも行の順序が重要かという点です。さらに、再編成を取り消し (一時停止) 可能にする必要があるかどうか、ファイルへの並行アクセスを許可するかどうか、なども考慮する必要があります。これらの要素に基づき、どのオプションが最適かを判別するために、以下の表を使用してください。斜線の項目 (アスタリスクが付いている) は、キー・ファイルのオプションの特色で、それを選択するのが特に適しています。

	ALWCANCEL(*NO)		ALWCANCEL(*YES)		
	KEYFILE(*NONE)	**KEYFILE(*FILE またはキー・ファイル)**	**KEYFILE (*RPLDLTRCD)**	**KEYFILE(*NONE)**	**KEYFILE(*FILE またはキー・ファイル)**
取り消しと再開	いいえ	いいえ	はい*	はい*	はい*
並行アクセス	いいえ	いいえ	はい*	はい*	はい*
並列処理	索引の再構築のみ	索引の再構築のみ	データの移動および索引の再構築*	データの移動および索引の再構築*	データの移動および索引の再構築*
非並列パフォーマンス	非常に高速	高速	非常に高速*	やや低速*	非常に低速*
一時記憶域	データ記憶域の 2 倍	データ記憶域の 2 倍	ジャーナル・レシーバーの記憶域*	ジャーナル・レシーバーの記憶域*	ジャーナル・レシーバーの記憶域*
LIFO KEYFILE 索引の処理	該当せず	重複を反転	該当せず*	該当せず*	重複順序を保持*
索引の処理 (KEYFILE 以外)	同期または非同期の再構築*	同期または非同期の再構築*	索引の保守、または同期または非同期の再構築*	索引の保守、または同期または非同期の再構築*	索引の保守、または同期または非同期の再構築*
最終的な行位置の正確さ	はい*	はい*	LOCK(*EXCL) が再開されない場合のみ	LOCK(*EXCL) が再開されない場合のみ	LOCK(*EXCL) が再開されない場合のみ
使用される CPU および I/O の量	最も少ない*	次に少ない*	最も少ない	やや多い	最も多い
可変長セグメントの再編成	良好*	良好*	やや不良	やや不良	やや不良

	ALWCANCEL(*NO)		ALWCANCEL(*YES)		
	KEYFILE(*NONE)	KEYFILE(*FILE またはキー・ファイル)	KEYFILE (*RPLDLTRCD)	KEYFILE(*NONE)	KEYFILE(*FILE またはキー・ファイル)
参照保全の親および FILE LINK CONTROL DataLink を許可	はい*	はい*	いいえ	いいえ	いいえ
QTEMP およびデータベース相互参照ファイルを許可	はい*	はい*	いいえ	いいえ	いいえ
複製のコスト	最小 (1 つのジャーナル項目)*	最小 (1 つのジャーナル項目)*	やや大きい (移動されるすべての行のジャーナル項目)	最も大きい (移動されるすべての行のジャーナル項目)	最も大きい (移動されるすべての行のジャーナル項目)

再編成の一時停止または取り消し: 再編成を取り消しできないと指定した場合、物理ファイル・メンバーの再編成 (RGZPFM) コマンド実行中に以下のいずれかの条件が発生すると、レコードがまったく再編成されない可能性があります。

- システムが異常終了する。
- RGZPFM コマンドを含むジョブが *IMMED オプションで終了する。
- RGZPFM コマンドを実行中のサブシステムが、*IMMED オプションで終了する。
- システムが、*IMMED オプションで停止する。

さらに、物理ファイル・メンバーの再編成 (RGZPFM) コマンドの実行中、ラージ・オブジェクト (LOB) に関連したレコードは再編成されず、削除されたレコードがファイルの中に残る場合があります。

再編成を取り消し可能と指定し、以下のいずれかの条件が発生した場合、または再編成を取り消した場合には、再編成が部分的にしか完了しない可能性があります。後で同じ再編成操作を実行した場合、中断された箇所から単に継続するだけかもしれません。ただし、再編成の取り消し後に内容が大きく変更された場合には、再編成は継続されず、最初からやり直されます。

また、再編成したいメンバーの状況は、再編成の終了前にシステムが実行できる処理の量および SRCOPT パラメーターの指定値によっても異なります。

SRCOPT パラメーターを指定してある場合には、以下のいずれか (複数の場合もある) がメンバーに起こったと考えられます。

- メンバーが完全に再編成された。再編成操作が正常に完了したことを示す完了メッセージがユーザーのジョブ・ログに送られます。
- メンバーはまったく再編成されなかった、または部分的にしか再編成されなかった。再編成操作が正常に行われなかったことを示すメッセージがユーザーのジョブ・ログに送られます。このような場合は、物理ファイル・メンバーの再編成 (RGZPFM) コマンドをもう一度実行してください。
- メンバーは再編成されたが、変更されなかった順序番号がある。メンバーが再編成されたが、順序番号がすべて変更されたわけではないことを示す完了メッセージがユーザーのジョブ・ログに送られます。このような場合は、KEYFILE (*NONE) を指定して RGZPFM コマンドをもう一度出してください。

SRCOPT パラメーターを指定しなかった場合には、メンバーは完全に再編成されるか、まったく再編成されないかのどちらかです。物理ファイル・メンバーの表示 (DSPPFM) コマンドまたは iSeries ナビゲータ

ーの「クイック・ビュー」を使用してファイルの内容を表示し、ファイルのどの程度の部分が再編成されたか (再編成が実行された場合) を判別して、必要に応じて物理ファイル・メンバーの再編成 (RGZPFM) コマンドをもう 1 度実行できます。

物理ファイル・メンバー内に存在する削除済みレコードの数を減らすには、削除済みレコード・スペースを再使用するようにファイルを作成または変更できます。詳細については、104 ページの『削除済みレコードの再使用』を参照してください。

物理ファイル・メンバーのレコードの表示: 物理ファイル・メンバーの表示 (DSPPFM) コマンドを使用して、物理データベース・ファイル・メンバー内のデータを到着順で表示できます。このコマンドを使用して、以下のことを実行できます。

- 問題分析
- デバッグ
- レコード照会

ソース・ファイルまたはデータ・ファイルは、キー順または到着順のどちらであっても、表示できます。レコードは、ファイルがキー順ファイルであっても、到着順で表示されます。ファイルの表示画面を上下に移動させたり、レコード番号によって特定のレコードを見つけ出したり、あるいは表示画面を左右に移動させてレコードの他の部分を見たりすることができます。また、機能キーを押して、文字データまたは 16 進データを画面に表示することもできます。

Query プログラムを導入している場合には、Query 開始 (STRQRY) コマンドを使用して、レコードを選択し表示することもできます。

SQL 言語を導入している場合は、SQL 開始 (STRSQL) コマンドを使用して対話式にレコードを選択し表示することができます。

データベース属性および相互参照情報の使用

iSeries 統合データベースは、ファイルの属性および相互参照情報を提供します。相互参照情報には、以下の事項が含まれます。

- プログラムで使用されるファイル
- データまたはアクセス・パスにおいて他のファイルに従属するファイル
- ファイルの属性
- ファイルに対して定義されたフィールド
- ファイルに関連した制約
- ファイルのキー・フィールド

以下の項で説明するそれぞれのコマンドにより、情報を画面に表示したり、印刷出力を出したり、または相互参照情報をデータベース・ファイルに書き出したりすることができます。この相互参照情報は、後で分析のためにプログラムまたはユーティリティー (例: Query) で使用できます。

属性情報の表示についての説明は、202 ページの『データベース・ファイルに関する情報の表示』を参照してください。データベース・ファイルへの出力の書き込みについての説明は、206 ページの『コマンドからデータベース・ファイルへの出力の直接書き込み』を参照してください。

メンバー記述の検索 (RTVMBRD) コマンドを使用して、データベース・ファイルのメンバーに関する情報を取り出し、アプリケーションでの使用に備えることができます。特定のメンバーの記述を取り出すために

CL プログラムで RTVMBRD コマンドを使用する例については、 制御言語 (CL) に関するトピック、および「CL プログラミング  」の『メンバー記述情報の取り出し』に関するセクションを参照してください。

データベース・ファイルに関する情報の表示

iSeries ナビゲーターの表記述の表示操作を使用して、データベース・ファイルおよび装置ファイルのファイル属性を表示することができます。『iSeries ナビゲーターの表記述の表示を使用したファイルの属性の表示』を参照してください。または、ファイル記述の表示 (DSPFD) コマンドを使用することができます。『DSPFD を使用したファイルの属性の表示』を参照してください。

このほかに、データベース・ファイル情報の表示について、以下のトピックを参照してください。

- 『DSPFD を使用したファイルの属性の表示』
- 203 ページの『ファイル内のフィールドの記述の表示』
- 203 ページの『システム上の複数のファイル間の関係の表示』
- 204 ページの『プログラムが使用するファイルの表示』
- 205 ページの『システムの相互参照ファイルの表示』

iSeries ナビゲーターの表記述の表示を使用したファイルの属性の表示: iSeries ナビゲーターでは、「**記述**」ウィンドウによって、表 (データベース・ファイル) 属性情報を表示することができます。

1. iSeries ナビゲーター・ウィンドウで、使用したいシステムを拡張する。
2. **データベース**を拡張する。
3. 操作したいデータベースおよびライブラリーを拡張する。
4. 情報を表示したい表またはビューが入っているライブラリーをクリックする。
5. 表、ビュー、または索引を右クリックして、「**記述**」を選択する。

「記述」ウィンドウで、一般、割り振り、用途、活動、および詳細を選択することができます。

DSPFD を使用したファイルの属性の表示: ファイル記述の表示 (DSPFD) コマンドを使用して、データベース・ファイルの属性を表示します。情報は画面に表示したり、印刷したり、データベース出力ファイル (OUTFILE) に書き出したりすることができます。このコマンドによって提供される情報 (括弧内に示すパラメーター値) には、以下の事項が含まれます。

- 基本属性 (*BASATR)
- ファイルの属性 (*ATR)
- アクセス・パス指定 (*ACCPTH、論理ファイルおよび物理ファイルのみ)
- 選択/除外指定 (*SELECT、論理ファイルのみ)
- 結合論理ファイル指定 (*JOIN、結合論理ファイルのみ)
- 代替照合順序指定 (*SEQ、物理ファイルおよび論理ファイルのみ)
- レコード様式指定 (*RCDFMT)
- メンバー属性 (*MBR、物理ファイルおよび論理ファイルのみ)
- スプール属性 (*SPOOL、印刷装置ファイルおよびディスケット・ファイルのみ)
- メンバー・リスト (*MBRLIST、物理ファイルおよび論理ファイルのみ)
- ファイル制約 (*CST)
- トリガー (*TRG)

ファイル内のフィールドの記述の表示: ファイル・フィールド記述の表示 (DSPFFD) コマンドを使用して、データベース・ファイルと装置ファイルの両方のフィールド情報を表示することができます。情報は画面に表示したり、印刷したり、データベース出力ファイル (OUTFILE) に書き出したりすることができます。

システム上の複数のファイル間の関係の表示: データベース関係の表示 (DSPDBR) コマンドを使用して、データベースの編成に関する以下の情報を表示することができます。

- 特定のレコード様式を使用するデータベース・ファイル (物理および論理) のリスト
- データの共用に関して指定したファイルに従属するデータベース・ファイル (物理および論理) のリスト
- データの共用またはアクセス・パスに関して指定したメンバーに従属するメンバー (物理および論理) のリスト
- このファイルと参照制約関係にある従属ファイルの物理ファイルのリスト

この情報は、画面に表示したり、印刷したり、データベース出力ファイル (OUTFILE) に書き出したりすることができます。

たとえば、物理ファイルの ORDHDRP と対応付けられた、レコード様式が ORDHDR のすべてのデータベース・ファイルのリストを表示するには、以下の DSPDBR コマンドを入力します。

```
DSPDBR FILE(DSTPRODLB/ORDHDRP) RCDFMT(ORDHDR)
```

注: この画面の詳細については、 制御言語 (CL) に関するトピックの DSPDBR コマンドの説明を参照してください。

レコード様式名を RCDFMT パラメーターに指定すると、この画面に見出し情報が表示され、指定したレコード様式を使用するファイルに関する情報が表示されます。

DSPDBR コマンドの MBR パラメーターにメンバー名を指定すると、従属メンバーが表示されます。

データベース関係の表示 (DSPDBR) コマンドに省略時値の MBR(*NONE) パラメーター値を指定すると、従属データ・ファイルが表示されます。共用アクセス・パスを表示するには、メンバー名を指定する必要があります。

データベース関係の表示 (DSPDBR) コマンドの出力には、関連する共用のタイプが表示されます。コマンドの結果が表示される場合は、共用のタイプの名前が表示されます。コマンドの結果がデータベース・ファイルに書き込まれる場合は、出力ファイルのレコードの *WHTYPE* フィールドに、共用のタイプのコード (以下を参照) が入れられます。

タイプ	コード	説明
制約	C	物理ファイルは、制約を介して関連付けられた別の物理ファイル内のデータに依存します。
データ	D	ファイルまたはメンバーは、別のファイルのメンバーのデータに依存します。
アクセス・パス共用	I	ファイル・メンバーはアクセス・パスを共用します。

タイプ	コード	説明
アクセス・パス所有者	O	アクセス・パスを共用している場合は、ファイル・メンバーの 1 つがアクセス・パスの所有者とみなされます。アクセス・パスの所有者は、アクセス・パスが使用する記憶域についての責任を負います。表示されているメンバーが所有者として指示されている場合、 1 つまたは複数のファイル・メンバーにアクセス・パス共用を示す I が指定されます。
SQL ビュー	V	SQL ビューまたはメンバーは、別の SQL ビューに従属します。

プログラムが使用するファイルの表示: プログラム参照の表示 (DSPPGMREF) コマンドを使用して、プログラムが使用するファイル、データ域、および他のプログラムを判別できます。この情報は、コンパイル済みプログラムの場合にだけ使用可能です。

情報は画面に表示したり、印刷したり、データベース出力ファイル (OUTFILE) に書き出したりすることができます。

プログラムの作成時には、プログラムが使用する特定のオブジェクトに関する情報が保管されます。この情報は、後でプログラム参照の表示 (DSPPGMREF) コマンドで使用することができます。

以下の表に、高水準言語およびユーティリティーが情報を保管するオブジェクトを示します。

言語またはユーティリティー	ファイル	プログラム	データ域	参照する注
BASIC	はい	はい	いいえ	1
C/400® 言語	いいえ	いいえ	該当せず	
CL	はい	はい	はい	2
COBOL/400 言語	はい	はい	いいえ	3
CSP	はい	はい	いいえ	4
DFU	はい	該当せず	該当せず	
FORTRAN/400* 言語	いいえ	いいえ	該当せず	
Pascal	いいえ	いいえ	該当せず	
PL/I	はい	はい	該当せず	3
RPG/400 言語	はい	はい	はい	5
SQL 言語	はい	該当せず	該当せず	

言語またはユーティリティー	ファイル	プログラム	データ域	参照する注
:				

注:

1. 外部的に記述されたファイル参照、プログラム、およびデータ域が保管されます。
2. ファイル、プログラム、またはデータ域を参照するすべてのシステム・コマンドでは、コマンドを CL プログラム内でコンパイルする時に情報を保管することをコマンド定義に指定する必要があります。変数を使用する場合、変数の名前はオブジェクト名として使用されます (例: &FILE)。式を使用する場合、オブジェクトの名前は *EXPR として保管されます。ユーザー定義コマンドで、コマンドに指定されたファイル、プログラム、またはデータ域に関する情報を保管することもできます。 CL Programming (CL プログラミング、 SD88-5038) の PARM または ELEM コマンド・ステートメントの FILE、PGM、および DTAARA パラメーターの説明を参照してください。
3. プログラム名が保管されるのは、COBOL/400 識別コード (CALL PGM1 などの動的呼び出し) ではなく、リテラルがプログラム名に使用されている場合 (CALL 'PGM1' などの静的呼び出し) だけです。
4. CSP プログラムは、*MSGF、*CSPMAP および *CSPTBL タイプのオブジェクトに関する情報を保管します。
5. 内部データ域の用法は保管されません。

保管されたファイル情報には、用途を示す項目 (番号) が含まれています。プログラム参照の表示 (DSPPGMREF) コマンドのデータベース・ファイル出力 (OUTFILE パラメーターを使用した場合に作成される) では、以下のように指定します。

コード 意味

1 入力

2 出力

3 入力および出力

4 更新

8 未指定

コードは組み合わせて使用することもできます。たとえば、7 が指定されたファイルは、入力、出力、および更新に使用されます。

システムの相互参照ファイルの表示: システムは、以下の 8 つのデータベース・ファイルを管理します。

- 基本データベース・ファイルの属性情報 (QSYS/QADBXREF)
- システム上のすべてのデータベース・ファイル (QTEMP ライブラリーにあるデータベース・ファイルを除く) に関する相互参照情報 (QSYS/QADBFDEP)
- データベース・ファイルのフィールド情報 (QSYS/QADBIFLD)
- データベース・ファイルのキー・フィールド情報 (QSYS/QADBKFLD)
- 参照制約のファイル情報 (QSYS/QADBFCST)
- 参照制約のフィールド情報 (QSYS/QADBCCST)
- SQL パッケージ情報 (QSYS/QADBPKG)
- 遠隔データベースのディレクトリー情報 (QSYS/QADBXRDBD)

これらのファイルを使用すると、基本属性およびデータベース・ファイルの要件を判別することができます。これらのファイルに含まれるフィールドを表示するには、 ファイル・フィールド記述の表示 (DSPFFD) コマンドを使用します。

注: これらのファイルを使用する権限は、機密保護担当者に限定されています。しかし、各ファイル上に作成された論理ファイルのうちの 1 つ (または唯一のファイル) を使用してデータを表示する権限は、すべてのユーザーにあります。これらのファイルは常にオープンされているので、その権限を変更することはできません。

コマンドからデータベース・ファイルへの出力の直接書き込み

コマンドで OUTFILE パラメーターを指定することによって、出力物理ファイルに多数の CL コマンドからの出力を記憶できます。その後、プログラムまたはユーティリティー (たとえば Query) の出力ファイルを使用してデータを分析できます。たとえば、プログラム参照の表示 (DSPPGMREF) コマンドの出力を物理ファイルに送ってから、そのファイルに Query を実行し、特定のファイルを使用するプログラムを判別できます。

物理ファイルは、コマンドで OUTFILE パラメーターを指定した場合に作成されます。はじめに、ファイルは私用権限、つまり所有者 (コマンドの実行者) だけが使用できる権限を使用して作成されます。しかし、所有者は、他のすべてのデータベース・ファイルの場合と同様に、これらのファイルに対する権限を他のユーザーに認可することもできます。

システムには、OUTFILE パラメーターを指定できるコマンドごとにレコード様式を識別するモデル・ファイルが用意されています。まだ存在していないファイルの OUTFILE パラメーターにファイル名を指定した場合、システムは、モデル・ファイルと同じレコード様式を使用してファイルを作成します。既存の出力ファイルに対してファイル名を指定した場合は、システムはそのレコード様式がモデル・ファイルのレコード様式と同じかどうかを調べます。両方のレコード様式が一致しない場合、システムはジョブにメッセージを送り、コマンドは完了しません。

注: 出力ファイルには、OUTFILE パラメーターにシステム提供のモデル・ファイルを指定するのではなく、ユーザー独自のファイルを使用してください。

出力ファイルを扱うコマンドのリスト、およびそれらのコマンド用に提供されているモデル・ファイルの名前については、 制御言語 (CL) に関するトピックを参照してください。

注: すべてのシステム提供モデル・ファイルは QSYS ライブラリーに入っています。

ファイル・フィールド記述の表示 (DSPFFD) コマンドを使用すると、システム提供のモデル・ファイルのレコード様式に入っているフィールドを表示できます。

コマンド出力のファイルへの書き込みについて、詳しくは以下のトピックを参照してください。

- 『例: コマンド出力ファイルの使用』
- 207 ページの『ファイル記述の表示コマンドの出力ファイル』
- 207 ページの『ジャーナルの表示コマンドの出力ファイル』
- 207 ページの『問題の表示コマンドの出力ファイル』

例: コマンド出力ファイルの使用: 以下の例では、プログラム参照の表示 (DSPPGMREF) コマンドを使用して、すべてのライブラリーのすべてのコンパイル済みプログラムの情報を収集し、 DBROUT という名前のデータベース・ファイルに出力を入れます。

```
DSPPGMREF  PGM(*ALL/*ALL)  OUTPUT(*OUTFILE)  OUTFILE(DSTPRODLB/DBROUT)
```

Query を使用すると、出力ファイルを処理することができます。もう 1 つのファイル処理方法として論理ファイルを作成してそのファイルから情報を選択する方法もあります。このような論理ファイルの DDS を以下に示します。レコードはファイル名に基づいて選択されます。

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A* Logical file DBROUTL for query
     A
     A          R DBROUTL                   PFILE(DBROUT)
     A          S WHFNAM                    VALUES('ORDHDRL' 'ORDFILL')
     A
```

ファイル記述の表示コマンドの出力ファイル: ファイル記述の表示 (DSPFD) コマンドは、指定されたパラメーターにしたがって、固有の出力ファイルを作成します。 DSPFD コマンドのモデル・ファイルのリストについては、 制御言語 (CL) に関するトピックを参照してください。

注: すべてのシステム提供モデル・ファイルは QSYS ライブラリーに入っています。

LIBA ライブラリー内のすべてのファイルに関するアクセス・パス情報を収集するには、以下のように指定します。

```
DSPFD   FILE(LIBA/*ALL)  TYPE(*ACCPTH)  OUTPUT(*OUTFILE) +
        OUTFILE(LIBB/ABC)
```

ファイル ABC はライブラリー LIBB 内に作成され、システム提供ファイル QSYS/QAFDACCP と同じフィールド記述を使用して外部的に記述されます。ファイル ABC には、アクセス・パスを持つライブラリー LIBA に入っている各ファイルのそれぞれのキー・フィールドのレコードが含まれます。

ファイル記述の表示 (DSPFD) コマンドのコーディングは、以下のとおりです。

```
DSPFD   FILE(LIBX/*ALL)  TYPE(*ATR)  OUTPUT(*OUTFILE) +
        FILEATR(*PF)  OUTFILE(LIBB/DEF)
```

ファイル DEF はライブラリー LIBB の中に作成され、QSYS/QAFDPHY に存在するものと同じフィールド記述を使用して外部的に記述されます。ファイル DEF には、ライブラリー LIBX に入っている各物理ファイルのレコードが含まれます。

DSPFFD コマンドを使用して、 IBM 提供の各モデル・ファイルのフィールド名を表示できます。たとえば、アクセス・パス・モデル・ファイル (TYPE パラメーターに *ACCPTH を指定) のフィールド記述を表示するには、以下のように指定します。

```
DSPFFD   QSYS/QAFDACCP
```

ジャーナルの表示コマンドの出力ファイル: ジャーナルの表示 (DSPJRN) によって示すことのできる、システム上に提供されたモデル出力ファイルのリストについては、 制御言語 (CL) に関するトピックを参照してください。

問題の表示コマンドの出力ファイル: 問題の表示 (DSPPRB) コマンドによってシステムに提供されるモデル出力ファイルのリストについては、 制御言語 (CL) に関するトピックを参照してください。このコマンドは、レコードのタイプによって、固有の出力ファイルを提供します。

- 基本問題データ・レコード (*BASIC)。問題のタイプ、状況、マシンのタイプ/モデル/シリアル番号、製造識別コード、コンタクト・インフォメーション、追跡データが含まれます。
- 障害のポイント、分離、または回答 FRU レコード (*CAUSE)。回答 FRU は使用可能であれば使用されます。回答 FRU が使用不可である場合、分離 FRU が使用可能であれば使用されます。回答 FRU と分離 FRU が使用不可の場合、障害のポイント FRU が使用されます。
- PTF 修正レコード (*FIX)
- ユーザー入力テキスト (ノート・レコード) (*USRTXT)
- サポート・データ識別コード・レコード (*SPTDTA)

5 種類の出力ファイルすべてのレコードには、問題識別コードが付いているので、原因、修正、ユーザー・テキスト情報、およびサポート・データを基本問題データに相互に関係させることができます。特定の出力ファイルに書き込むことのできるデータ・タイプは、1 つだけです。原因、修正、ユーザー・テキスト、およびサポート・データの出力ファイルには、特定の問題について複数のレコードを記述できます。

データベース・ファイルの記述および属性の変更

この章では、データベース・ファイルの記述または属性の変更を計画する上での考慮事項を示します。以下のトピックを参照してください。

- 『ファイル記述内のフィールドの変更の影響』
- 209 ページの『物理ファイルの記述および属性の変更』
- 212 ページの『論理ファイルの記述および属性の変更』

ファイル記述内のフィールドの変更の影響

外部的に記述されたデータを使用するプログラムをコンパイルするときに、コンパイラーは、コンパイルするプログラムの中にファイルのファイル記述をコピーします。システムは、このプログラムを実行するとき、プログラムのコンパイル時のレコード様式が、ファイルに対して現在定義されているレコード様式と同じであるかを調べることができます。省略時の値はレベル検査の実行です。

システムは、レコード様式が結び付けられるファイルの作成時に、個々のレコード様式に固有のレベル識別コードを割り当てます。システムは、レコード様式記述内容の情報を使用して、レベル識別コードを決定します。この情報には、レコード様式の合計の長さ、レコード様式名、定義されたフィールドの数と順序、データ・タイプ、フィールドのサイズ、フィールド名、フィールド内の小数部の桁数、および、フィールドにヌル値が使用できるかどうか、が含まれています。レコード様式内のこの情報に変更を加えると、レベル識別コードが変更されます。

以下の DDS 情報はレベル識別コードに影響を及ぼさないので、ファイルを使用するプログラムをコンパイルし直さなくても、キーワードを変更できます。

- TEXT キーワード
- COLHDG キーワード
- CHECK キーワード
- EDTCDE キーワード
- EDTWRD キーワード
- REF キーワード
- REFFLD キーワード
- CMP キーワード、RANGE キーワード、および VALUES キーワード
- TRNTBL キーワード
- REFSHIFT キーワード
- DFT キーワード
- CCSID キーワード
- 結合仕様および結合キーワード
- キー・フィールド
- アクセス・パス・キーワード
- 選択/除外フィールド

キー・フィールドまたは選択/除外フィールドを変更してもレベル検査は行われませんが、この変更によって、新しいアクセス・パスを使用するプログラムに予期しない結果が生じる可能性があることに留意してください。たとえば、キー・フィールドを得意先番号から得意先名に変更すると、レコードの検索順序が変わり、そのファイルを処理するプログラムで予期しない問題が発生する場合があります。

レベル検査を指定した (または省略時値として指定されている) 場合は、使用するファイルのレベル識別コードが、ファイルのオープン時にプログラムのファイルのレベル識別コードと比較されます。両方の識別コードが異なる場合には、変更された状態を知らせるメッセージがプログラムに送られ、変更によりプログラムが影響を受ける可能性があります。この場合、変更を含めるように、プログラムをコンパイルし直す必要があります。

もう 1 つの方法はファイル記述を表示して、変更によりプログラムが影響を受けるかどうかを判別する方法です。ファイル・フィールド記述の表示 (DSPFFD) コマンドを使用して記述を表示することも、あるいは、SEU がある場合は、ファイルの DDS が入っているソース・ファイルを表示することができます。

ファイルに定義された様式レベル識別コードは、 ファイル記述の表示 (DSPFD) コマンドを使用して表示することができます。レベル識別コードを表示する場合には、ファイル識別コードではなく、レコード様式識別コードが比較されることに注意してください。

ファイルにおけるすべての変更が、必ずしもプログラムに影響を及ぼすわけではありません。たとえば、ファイルの終わりにフィールドを追加し、プログラムがその新しいフィールドを使用しない場合、プログラムをコンパイルし直す必要はありません。変更してもプログラムに影響が及ばない場合は、 LVLCHK(*NO) を指定した 物理ファイルの変更 (CHGPF) コマンドまたは 論理ファイルの変更 (CHGLF) コマンドを使用して、そのファイルのレベル検査を取り消すことができます。または、LVLCHK(*NO) を指定した データベース・ファイルの一時変更 (OVRDBF) コマンドを入力して、レベル検査なしでプログラムを実行することもできます。

レベル検査は実行することをお勧めします。LVLCHK(*YES) を指定すると、データベースの保全性は良くなります。LVLCHK(*NO) を指定することによって生じる結果は、必ずしも予期できません。

物理ファイルの記述および属性の変更

物理ファイルの記述に変更を加えてファイルを作成し直した場合は、レベル識別コードが変更される可能性があります。たとえば、ファイル記述にフィールドを追加したか、または既存のフィールドの長さを変更した場合、レベル識別コードは変更されます。レベル識別コードが変更された場合、ユーザーは、その物理ファイルを使用するプログラムを再びコンパイルする必要があります。プログラムは、再度コンパイルされた後は、新しいレベル検査識別コードを使用します。

論理ファイルの元のレコード様式でプログラムにデータを提供する論理ファイルを作成すれば、コンパイルし直さずに済みます。このような方法をとった場合、論理ファイルのレベル検査識別コードは、変更前の物理ファイルと同じになります。

たとえば、物理ファイル・レコード様式にフィールドを追加するとします。以下のようにすれば、プログラムをコンパイルし直さずに済みます。

1. DDS を変更し、新しいフィールドを入れるために新しい物理ファイル (LIBA 内の FILEB) を作成する。

   ```
   CRTPF FILE(LIBA/FILEB) MBR(*NONE)...
   ```

   FILEB にはメンバーがありません (古いファイル FILEA はライブラリー LIBA の中にあり、 MBRA というメンバーが 1 つあります)。
2. 古い物理ファイルのメンバーを新しい物理ファイルにコピーする。

```
CPYF    FROMFILE(LIBA/FILEA) TOFILE(LIBA/FILEB)
         FROMMBR(*ALL) TOMBR(*FROMMBR)
        MBROPT(*ADD) FMTOPT(*MAP)
```

新しい物理ファイルの中のメンバーは、自動的に、古い物理ファイル内のメンバーと同じ名前になります (FROMMBR(*ALL) と TOMBR(*FROMMBR) が指定されているため)。FMTOPT パラメーターでは、フィールド内のデータをフィールド名によってコピーするよう (*MAP) 指定されています。

3. 元の物理ファイルによく似た新しい論理ファイル (FILEC) を記述する (論理ファイル・レコード様式に、新しい物理ファイル・フィールドは *含まれません*)。PFILE キーワードに FILEB を指定します (レベル検査を実行すると、FILEA と FILEC は同一の様式になっているため、論理ファイルのレベル識別コードとプログラムのレベル識別コードが一致します)。
4. 新しい論理ファイルを作成する。

   ```
   CRTLF FILE(LIBA/FILEC)...
   ```

5. このようにすると、以下のどちらかを実行できます。
   a. 適切なジョブでデータベース・ファイルの一時変更 (OVRDBF) コマンドを使用して、プログラム内で参照される古い物理ファイルを論理ファイルで一時変更する (OVRDBF コマンド・パラメーターについては、データベース・ファイル処理: 実行時の考慮事項で詳しく説明しています)。

      ```
      OVRDBF FILE(FILEA) TOFILE(LIBA/FILEC)
      ```

   b. 古い物理ファイルを削除し、プログラム内のファイル名を一時変更しなくて済むように、論理ファイルの名前を古い物理ファイルの名前に変更する。

      ```
      DLTF FILE(LIBA/FILEA)
      RNMOBJ   OBJ(LIBA/FILEC) OBJTYPE(*FILE)
               NEWOBJ(FILEA)
      ```

3 つのファイルで使用されるレコード様式の関係を、以下に示します。

FILEA (古い物理ファイル)

FLDA	FLDB	FLDC	FLDD

FILEB では、FLDB1 が以下のようなレコード様式に追加された。

FILEB (新しい物理ファイル)

		FLDB1		

FILEC は、 FILEA のレコード様式を共有する。 FLDB1 は論理ファイルのレコード様式では使われない。

FILEC (論理ファイル)

FLDA	FLDB	FLDC	FLDD

ファイルを作成し直さなければならないような変更を物理ファイルに加えた場合は、この物理ファイルを参照するすべての論理ファイルをまず削除してから、新しい物理ファイルを作成します。ユーザーは、物理ファイルを再作成した後、それを参照する論理ファイルを再作成したり復元したりすることが可能になります。これを行うため 2 つの方法を、以下の例に示します。

- 211 ページの『例 1: 物理ファイルの記述および属性の変更』

- 『例 2: 物理ファイルの記述および属性の変更』

例 1: 物理ファイルの記述および属性の変更: 同じ名前の新しい物理ファイルを、別のライブラリーの中に作成する。

1. 古い物理ファイルが入っているライブラリーとは別のライブラリーに、レコード様式が異なる新しい物理ファイルを作成する。新しいファイルの名前は古いファイルの名前と同じにします (古い物理ファイル FILEPFC はライブラリー LIBB に入っており、このライブラリーに MBRC1 と MBRC2 の 2 つのメンバーがあります)。

   ```
   CRTPF FILE(NEWLIB/FILEPFC) MAXMBRS(2)...
   ```

2. 古い物理ファイルのメンバーを新しい物理ファイルにコピーする。TOMBR(*FROMMBR) および FROMMBR(*ALL) を指定しているため、新しい物理ファイルのメンバーには古い物理ファイルのメンバーと同じ名前が自動的に付けられます。

   ```
   CPYF      FROMFILE(LIBB/FILEPFC) TOFILE(NEWLIB/FILEPFC)
             FROMMBR(*ALL) TOMBR(*FROMMBR)
             FMTOPT(*MAP *DROP) MBROPT(*ADD)
   ```

3. 古い論理ファイルが入っているライブラリーとは別のライブラリーに、新しい論理ファイルを記述し、作成する。新しい論理ファイルの名前は、古い論理ファイルの名前と同じにします。レコード様式に変更を加える必要がない場合には、FORMAT キーワードを使用して、現行の論理ファイルに入っているレコード様式と同じ様式を使用できます。さらに、ライブラリー LIBB の古い論理ファイル FILELFC から別の論理ファイルを作成するときには、重複オブジェクトの作成 (CRTDUPOBJ) コマンドを使用できます。

   ```
   CRTLF FILE(NEWLIB/FILELFC)
   ```

4. 古い論理ファイルおよび物理ファイルを削除する。

   ```
   DLTF FILE(LIBB/FILELFC)
   DLTF FILE(LIBB/FILEPFC)
   ```

5. 以下のコマンドを使用して、新しく作成したファイルを元のライブラリーに移動する。

   ```
   MOVOBJ OBJ(NEWLIB/FILELFC) OBJTYPE(*FILE) TOLIB(LIBB)
   MOVOBJ OBJ(NEWLIB/FILEPFC) OBJTYPE(*FILE) TOLIB(LIBB)
   ```

例 2: 物理ファイルの記述および属性の変更: 同じライブラリー内に新しいバージョンのファイルを作成する。

1. 古い物理ファイルが入っているライブラリーと同じライブラリーにレコード様式が異なる新しい物理ファイルを作成する。ファイルの名前は別の名前にしてください。(古い物理ファイル FILEPFA はライブラリー LIBA に入っており、このライブラリーに 2 つのメンバー MBRA1 と MBRA2 があります。)

   ```
   CRTPF FILE(LIBA/FILEPFB) MAXMBRS(2)...
   ```

2. 古い物理ファイルのメンバーを新しい物理ファイルにコピーする。

   ```
   CPYF      FROMFILE(LIBA/FILEPFA) TOFILE(LIBA/FILEPFB)
             FROMMBR(*ALL) TOMBR(*FROMMBR)
             FMTOPT(*MAP *DROP) MBROPT(*REPLACE)
   ```

3. 古い物理ファイルが入っているライブラリーと同じライブラリーに、新しい論理ファイルを作成する。古いファイルと新しいファイルは、別の名前にしてください。(レコード様式に変更を加える必要がない場合には、FORMAT キーワードを使用して、現行の論理ファイルに入っているレコード様式と同じ様式を使用できます。) PFILE キーワードは、ステップ 1 で作成した新しい物理ファイルを参照しなければなりません。古い論理ファイル FILELFA はライブラリー LIBA に入っています。

   ```
   CRTLF FILE(LIBA/FILELFB)
   ```

4. 古い論理ファイルおよび物理ファイルを削除する。

   ```
   DLTF FILE(LIBA/FILELFA)
   DLTF FILE(LIBA/FILEPFA)
   ```

5. 新しい論理ファイルの名前を古い論理ファイルの名前に変更する。(物理ファイルの名前も変更する場合は、PFILE キーワードが新しい物理ファイル名を指すようにするため、論理ファイルの DDS を必ず指定してください。)

   ```
   RNMOBJ(LIBA/FILELFB) OBJTYPE(*FILE) NEWOBJ(FILELFA)
   ```

6. 論理ファイルの作成 (CRTLF) コマンドで省略時の値が使用されると仮定して、論理ファイル・メンバーの名前を変更する場合は、以下のコマンドを出す。

   ```
   RNMM FILE(LIBA/FILELFA) MBR(FILELFB) NEWMBR(FILELFA)
   ```

物理ファイルの変更 (CHGPF) コマンドを使用すると、物理ファイルとそのメンバーの属性の一部を変更できます。これらのパラメーターの詳細については、 制御言語 (CL) に関するトピックの『物理ファイルの変更 (CHGPF) コマンド』を参照してください。

論理ファイルの記述および属性の変更

一般に、レベル識別コードを変えるような変更 (たとえば、新しいフィールドの追加、フィールドの削除、またはフィールド長の変更) を論理ファイルに加える場合は、その論理ファイルを使用するプログラムを必ず コンパイルし直してください。レベル識別コードは変更しても、プログラムをコンパイルし直す必要のない変更 (たとえば、プログラムが使用しないフィールドをファイルの終わりに追加するなど) をファイルに加えることができる場合もあります。しかし、このような場合は、変更の対象となったファイルを使用するプログラムを実行するため、強制的にレベル検査をオフに切り替えなければなりません。これは望ましい方法ではありません。後で正しくないデータが発生する可能性が高くなるためです。

再コンパイルをしないで済ませるために、現行の論理ファイル (未変更の) を保持し、追加フィールドを加えた新しい論理ファイルを作成できます。プログラムは、まだ存在している古いファイルを参照します。

論理ファイルの変更 (CHGLF) コマンドを使用すると、 論理ファイルの作成 (CRTLF) コマンドで指定した論理ファイルおよびそのメンバーの属性のほとんどを変更することができます。

データベースの回復と復元

以下のトピックでは、システムがデータを消失した後に、データベースを回復または復元するための iSeries 機能について説明します。以下のトピックは、データの保護と回復方法について説明します。

- 『データベース・ファイルのデータの回復』 では、データベース・ファイルからデータを回復するための支援となる、 iSeries ジャーナル処理およびコミットメント制御について説明します。
- 221 ページの『アクセス・パスの回復時間の短縮』 では、iSeries アクセス・パスについて説明し、データベース回復を実行するときの効果的な使用方法を示します。
- 226 ページの『システムの異常終了後のデータベースの回復』 では、システムの異常終了が発生した場合に iSeries システムが行う処理の概要を説明します。

情報の保管および復元について、詳しくは「バックアップおよび回復の手引き  」、 バックアップおよび回復のトピック、および以下を参照してください。

- 229 ページの『データベースの保管と復元』
- 229 ページの『データベースの保管と復元の考慮事項』

データベース・ファイルのデータの回復

iSeries システムは、ジャーナル処理とコミットメント制御を使用してデータベース・ファイル内のデータの回復を支援します。

- 『ジャーナル管理』によって、データに発生したすべての変更点を 1 つまたは複数のデータベース・ファイルに記録できます。ジャーナルを用いて、回復を行うことができます。
- 220 ページの『コミットメント制御を使用したデータの保全性の保証』は、ジャーナル管理機能の拡張機能であり、ジョブまたはシステムが終了した場合でも、複雑なアプリケーション・トランザクションが論理的に同期化されるようにしま す。

ジャーナル管理: ジャーナル管理によって、データに発生したすべての変更点を 1 つまたは複数のデータベース・ファイルに記録できます。ジャーナルを用いて、回復を行うことができます。データベース・ファイルが破壊されたりまたは使用不可能になったときにジャーナルを使用していると、ファイルに対する活動の大半を再構成することができます。ジャーナルを使用すると、ファイルに対して行われた変更を除去することもできます。

ジャーナル管理の説明については、以下のトピックを参照してください。
- 『ジャーナル』
- 『ジャーナルの処理』

さらに、ジャーナル管理のトピックも参照してください。

ジャーナル: ジャーナルの使用時にファイルに変更を加えると、システムはジャーナル・レシーバーの変更を記録し、ファイルに記録される前にレシーバーを補助記憶域に書き出します。このため、ジャーナル・レシーバーには常に、最新のデータベース情報が入っていることになります。

ジャーナル項目は、特定のレコードまたはファイルに対する活動を全体として記録します。各項目には、活動のソース (ユーザー、ジョブ、プログラム、時間、および日付など) を識別する制御情報バイトが入っています。単一レコードに影響を及ぼす変更の場合には、制御情報の後にレコード・イメージが入れられます。変更前のレコード・イメージも入れることができます。物理ファイル・ジャーナルの開始 (STRJRNPF) コマンドに IMAGES パラメーターを指定することによって、変更前と変更後の両方のレコード・イメージのジャーナルを作成するのか、変更後レコード・イメージだけのジャーナルを作成するのかを制御できます。

すべてのジャーナル処理データベース・ファイルは、システムの開始時 (IPL 時) に自動的にジャーナルと同期化されます。システム・セッションが異常終了した場合には、データベースの変更のうちのいくつかがジャーナルに入っていますが、データベース・ファイルには反映されていないものがあります。このような場合には、システムは自動的にジャーナルからデータベース・ファイルを更新します。

ジャーナルを使用すると、データベースの保管がさらに容易に、迅速になります。たとえば、毎日ファイル全体を保管する代わりに、そのファイルに対する変更が入っているジャーナル・レシーバーを保管するだけで済みます。ファイル全体の保管は週単位で行います。この方法を用いると、日常の保管操作の実行に要する時間を削減できます。

ジャーナルの詳細については、 ジャーナル管理のトピックを参照してください。

ジャーナルの処理: 以下の CL コマンドを使用して、ジャーナルの処理を行うことができます。
- ジャーナル変更を含んでいる損傷を受けたあるいは使用不可能になったデータベース・ファイル・メンバーを回復するには、 ジャーナル変更の適用 (APYJRNCHG) コマンド、および ジャーナル変更の削除 (RMVJRNCHG) コマンドを使用します。
- ジャーナル・レシーバーに記録された変更点を、指定された物理ファイル・メンバーに適用するには、APYJRNCHG コマンドを使用します。ただし、物理ファイルに対する損傷のタイプと、ファイルが最後に保管されたときからの活動量によっては、RMVJRNCHG コマンドを使用してファイルから変更点を削除する方が簡単です。

- ジャーナル項目をデータベース・ファイルに変換するには、 ジャーナルの表示 (DSPJRN) コマンドを使用します。このファイルは、活動報告書、監査証跡、機密保護、およびプログラム・デバッグに使用できます。

ジャーナルを処理するための CL コマンドの使用について詳しくは、 ジャーナル管理のトピック、および Information Center の制御言語 (CL) に関するトピックを参照してください。

さらに、ジャーナルを処理するときに、以下のiSeries ナビゲーター機能を使用することができます。

- 『iSeries ナビゲーターを使用したジャーナルの作成』
- 215 ページの『iSeries ナビゲーターを使用したジャーナル・レシーバーの作成』
- 216 ページの『iSeries ナビゲーターを使用した遠隔ジャーナルの追加』
- 217 ページの『iSeries ナビゲーターを使用した遠隔ジャーナルの除去』
- 217 ページの『iSeries ナビゲーターを使用した遠隔ジャーナルの活動化』
- 217 ページの『iSeries ナビゲーターを使用した遠隔ジャーナルの非活動化』
- 218 ページの『iSeries ナビゲーターを使用した表のジャーナル情報の表示』
- 218 ページの『iSeries ナビゲーターを使用したレシーバーのスワッピング』
- 219 ページの『iSeries ナビゲーターを使用した、表 (ファイル) のジャーナルの開始/停止』

iSeries ナビゲーターを使用したジャーナルの作成: ジャーナルを 1 つ作成すると、ジャーナルの新しいインスタンスが 1 つ、システム上に作成されます。情報のジャーナル処理を始める前に、ジャーナルへの表のジャーナル処理を開始しなければなりません。

1. **iSeries ナビゲーター**・ウィンドウで、使用したいシステムを拡張する。
2. **データベース**を拡張する。
3. 新しいジャーナルを作成するデータベースおよびライブラリーを拡張する。
4. ライブラリー・オブジェクトを右クリックし、「**新規ジャーナル**」を選択する。
5. 「**新規ジャーナル**」ウィンドウで、「**名前**」フィールドに名前を指定する。
6. 「**記述**」フィールドに記述を入力する (オプション)。
7. ジャーナル・レシーバーを保管するライブラリーを選択する。

ここで、デフォルト値を使用してジャーナルを作成することができます。「アドバンスト」をクリックして、ジャーナルのデフォルト値を編集できます。デフォルト値を使用してジャーナルを作成する場合は、**OK** をクリックします。

ジャーナルのデフォルト値を編集するには、まず、ジャーナルを作成しなければなりません。

1. 「**新規ジャーナル**」ウィンドウで、「**アドバンスト**」をクリックする。
2. 「**ジャーナル・メッセージ・キュー**」フィールドで、ジャーナル・メッセージ・キューを選択する。デフォルトは、システム・オペレーター・メッセージ・キューです。別のメッセージ・キューを指定することができます。ただし、メッセージが送られたときに、ユーザーが指定したメッセージ・キューが利用不能である場合は、メッセージは、システム・オペレーター・メッセージ・キューに送られます。
3. ジャーナル・メッセージ・キューが常駐する予定のライブラリーを指定する。
4. 「**記述**」フィールドで、記述を編集するか指定する (オプション)。
5. オプションによって**管理されるレシーバー**を選択する。選択は、**システム**か**ユーザー**のどちらかになります。
6. ジョブ、プログラム、およびユーザー・プロファイル情報を記録しない場合は、「**項目の固定部分の最小化**」を選択する。これによって、各ジャーナル項目のサイズは最小化されますが、他のジャーナル・コマンドで使用できる選択基準が制約されます。

7. システムのリスタート/リカバリーに必要な内部ジャーナル項目の除去だけを自動的に行いたい場合は、「**内部項目の除去**」を選択する。これらの項目を除去すると、ジャーナル・レシーバーのサイズを減らすことができます。
8. 新しいジャーナル・レシーバーが、ジャーナルと同時に作成されます。「**新規レシーバー**」をクリックして、レシーバーのデフォルト値を編集することができます。
9. 「**アドバンスト**」オプションの入力を完了したら、**OK** をクリックして「**新規ジャーナル**」ウィンドウに戻る。
10. **OK** をクリックしてジャーナルを作成する。

iSeries ナビゲーターを使用したジャーナル・レシーバーの作成: ジャーナル・レシーバーとは、ジャーナルが記録中の情報が入っているファイルのことです。ジャーナル・レシーバーは、ジャーナルが作成されるときに自動的に作成されます。ただし、ユーザーが手動でレシーバーをスワップする必要がある場合には、まず、新規のジャーナル・レシーバーを作成する必要があります。

1. **iSeries ナビゲーター**・ウィンドウで、 使用したいシステムを拡張する。
2. **データベース**を拡張する。
3. レシーバーを追加したいジャーナルが入っているデータベースおよびライブラリーを拡張する。
4. レシーバーを追加したいジャーナルを右クリックし、「**プロパティー**」を選択する。
5. 「**レシーバー**」をクリックする。
6. 「**ジャーナルのためのレシーバー...**」ウィンドウで、「**新規**」をクリックする。
7. 名前 (10 文字まで)、レシーバーを入れるライブラリー、記述、ストレージ・スペースしきい値を指定する。
8. **OK** をクリックして、「**新規ジャーナル・レシーバー**」ウィンドウをクローズする。
9. **OK** をクリックして、「**ジャーナルのためのレシーバー..**」ウィンドウをクローズする。
10. **OK** をクリックして、「**ジャーナル・プロパティー**」ウィンドウをクローズする。

注: 新しいジャーナルを作成するときに、新しいレシーバーを作成することもできます。

1. 「**アドバンスト・ジャーナル属性**」ウィンドウで、「**新規レシーバー**」をクリックする。
2. 「**新規ジャーナル・レシーバー**」ウィンドウで、名前 (10 文字まで)、レシーバーを入れるライブラリー、記述、ストレージ・スペースしきい値を指定する。
3. **OK** をクリックして、「**新規ジャーナル・レシーバー**」ウィンドウをクローズする。
4. **OK** をクリックして、「**アドバンスト・ジャーナル属性**」ウィンドウをクローズする。ジャーナル・レシーバーの値を指定しない場合は、ジャーナル・レシーバーは、デフォルト値を使用して作成されます。

新しいジャーナルおよび新しいジャーナル・レシーバーを作成するときに使用される値: ジャーナルおよびジャーナル・レシーバーは、ユーザーが「アドバンスト・ジャーナル属性」ウィンドウ、または「ジャーナル・プロパティー」ダイアログで指定した値を使用して作成または変更することができます。値を指定しない場合は、ジャーナルおよびジャーナル・レシーバーはデフォルト値を使用して作成されます。ジャーナルの場合:

- 対象としているライブラリーにジャーナルが作成されます。
- ジャーナルのストレージ・スペースは、ジャーナルのライブラリーの ASP (補助記憶域プール) のストレージ・スペースと同じ ASP から割り振られます。この値は変更できません。
- このジャーナルに関連したメッセージ・キューは、システム・オペレーター・メッセージ・キューです。

- ジャーナル・レシーバーのスワッピングは、システムが自動的にスワッピングを行うように設定されます。
- ジャーナル・レシーバーは、スワップの処理中に、システムによって自動的に削除されることはありません。
- ジャーナル項目の固定部分は最小化されませんが、内部ジャーナル項目は除去されます。
- ジャーナルの共通権限は、ライブラリーのデフォルト共通権限からとられます。
- ジャーナルのデフォルト・テキスト記述は作成されません。

ジャーナル・レシーバーの場合:

- ジャーナル・レシーバーのストレージ・スペースは、ジャーナル・レシーバーのライブラリーの ASP (補助記憶域プール) のストレージ・スペースと同じ ASP から割り振られます。この値は変更できません。
- ジャーナル・レシーバーのストレージ・スペースしきい値は、500 メガバイト (MB) に設定されます。
- ジャーナル・レシーバーの共通権限は、ライブラリーのデフォルト共通権限からとられます。
- ジャーナル・レシーバーのデフォルト・テキスト記述が作成されます。

iSeries ナビゲーターを使用した遠隔ジャーナルの追加: 遠隔ジャーナルにより、ユーザーは、ジャーナル情報を別のシステムに複製することができます。遠隔ジャーナルは、既存のジャーナルに関連付けられます。ソース・システム上のジャーナルは、ローカル・ジャーナルでも、別の遠隔ジャーナルでもかまいません。

1. **iSeries ナビゲーター**・ウィンドウで、使用したいシステムを拡張する。
2. データベースを拡張する。
3. 操作したいデータベースを拡張して、「**ライブラリー**」を拡張する。
4. 遠隔ジャーナルを追加したいジャーナルが入っているライブラリーをクリックする。
5. 遠隔ジャーナルを追加したいジャーナルを右クリックし、「**プロパティー**」を選択する。
6. 「**ジャーナル・プロパティー**」ウィンドウで、「**遠隔ジャーナル**」をクリックする。
7. 遠隔ジャーナルをこのジャーナルに追加する (関連付ける) には、「**追加**」をクリックする。
8. リレーショナル・データベース (RDB) のディレクトリー項目のリストを表示するには、「**遠隔ジャーナルの追加**」ウィンドウの「**リレーショナル・データベースの名前ボックス**」内の下矢印をクリックする。
9. ジャーナル・タイプ (タイプ 1 またはタイプ 2) を選択する。
10. ターゲット・システム上の遠隔ジャーナル・レシーバーをソース・システムで使用されているものとは異なるライブラリーに関連付けるには、「**レシーバーのリダイレクト**」を選択する。
11. 「**ターゲット・レシーバー・ライブラリー**」フィールドで、遠隔ジャーナル・レシーバーを入れる予定のターゲット・システム上のライブラリーを指定する。
12. **OK** をクリックする。

遠隔ジャーナル・タイプは、リダイレクト機能、ジャーナル・レシーバーの復元操作、および、遠隔ジャーナルのアソシエーション特性に影響を与えます。

限定リダイレクト (タイプ 1)。追加される遠隔ジャーナルがタイプ 1 の場合は、ジャーナル・ライブラリー名はローカル・ジャーナルとは別の 1 つのライブラリーにリダイレクトできます。ある 1 つのローカル・ジャーナルに関連付けられたすべてのタイプ 1 遠隔ジャーナルは、同一のライブラリーに常駐しなければなりません。タイプ 1 の遠隔ジャーナルはタイプ 2 の遠隔ジャーナルに追加できません。

柔軟なリダイレクト (タイプ 2)。追加される遠隔ジャーナルがタイプ 2 の場合は、タイプ 2 遠隔ジャーナルの直前の追加で指定されたリダイレクト・ライブラリーと同じまたは別のリダイレクト・ライブラリーを指定することができます。タイプ 2 遠隔ジャーナルの後続追加では、前に追加された遠隔ジャーナルで指定されたものとは別のライブラリー・リダイレクトを指定できます。

遠隔ジャーナルは、追加した後で、アクティブにしなければなりません。

注: このタスクでは、ユーザーが RDB ディレクトリー項目を持ち、ターゲット・システム上にユーザー・プロファイルを持っていることを前提にしています。

iSeries ナビゲーターを使用した遠隔ジャーナルの除去: 遠隔ジャーナルは、除去されると、ソース・システム上のジャーナルから分離されます。これによって、ジャーナルまたはジャーナル・レシーバーが削除されることはありません。遠隔ジャーナルは、非活動化してからでないと、除去できません。

iSeries ナビゲーターを使用した遠隔ジャーナルの活動化: 遠隔ジャーナルは、追加したならば、アクティブにしなければなりません。

1. **iSeries ナビゲーター**・ウィンドウで、使用したいシステムを拡張する。
2. **データベース**を拡張する。
3. 操作したいデータベースを拡張して、「**ライブラリー**」を拡張する。
4. アクティブにしたい関連遠隔ジャーナルを持っているジャーナルが入っているライブラリーをクリックする。
5. ジャーナルを右クリックし、「**プロパティー**」を選択する。
6. 「**ジャーナル・プロパティー**」ウィンドウで、「**遠隔ジャーナル**」をクリックする。
7. 「**遠隔ジャーナル**」ウィンドウで、遠隔ジャーナルのリストにある遠隔ジャーナルを選択し、次に、「**活動化**」をクリックし、選択された遠隔ジャーナルをアクティブにする。
8. 「**活動化**」ウィンドウで、デリバリー・モード、開始レシーバー、および、活動化要求の処理モードを選択する。
9. **OK** をクリックする。

注:

1. 初めて遠隔ジャーナルを活動化すると、1 つ以上のジャーナル・レシーバーがターゲット・システム上に作成されます。
2. 遠隔ジャーナルを活動化すると、ソース・ジャーナルと遠隔ジャーナルとの間に接続が確立され、ジャーナル項目の複製が開始できます。
3. 遠隔ジャーナルは、アクティブにする前は、非アクティブになっていなければなりません。また、ユーザーがいまアクティブにしようとしている遠隔ジャーナルは、それ自身すでにジャーナル項目を他の遠隔ジャーナルに複製していてはなりません。

iSeries ナビゲーターを使用した遠隔ジャーナルの非活動化: 遠隔ジャーナルを非活動化すると、データの収集を停止します。

同期して保守されていたジャーナルを非活動化すると、要求した終了が即時終了か制御終了に関係なく、遠隔ジャーナル機能は即時に終了されます。遠隔ジャーナルがキャッチアップ・フェーズの処理を行っていた場合は、要求した終了が即時終了か制御終了に関係なく、遠隔ジャーナル機能は即時に終了されます。(キャッチアップ処理とは、遠隔ジャーナルがアクティブになる前にソース・ジャーナルのジャーナル・レシーバーにすでに存在していたジャーナル項目を複製するプロセスのことを指します。) 遠隔ジャーナルが非同期保留または同期保留のデリバリー・モードを持っている場合は、キャッチアップ状態にあります。

制御

遠隔ジャーナル機能を制御方法で非活動化します。これは、遠隔ジャーナルを非活動化する前に、ソース・システムからターゲット・システムに送られるようにすでに待ち行列に入れられたジャーナル項目をすべて、システムが複製しなければならないことを意味します。待ち行列に入れられた項目がすべて複製されるまで、遠隔ジャーナルは非アクティブ保留状態になります。非アクティブ保留状態にある間は、「遠隔ジャーナル」ウィンドウは非アクティブ保留情報を提供します。待ち行列に入れられた項目がすべてターゲット・システムに送られると、システムはジャーナル・メッセージ・キューにメッセージを送り、遠隔ジャーナル機能が終了したことを伝えます。

即時

遠隔ジャーナル機能を即時に非活動化します。これは、遠隔ジャーナルを非活動化する前に待ち行列にすでに入れられていたジャーナル項目の複製をシステムが継続してはならないことを意味します。

***iSeries* ナビゲーターを使用した表のジャーナル情報の表示:**

1. **iSeries ナビゲーター**・ウィンドウで、使用したいシステムを拡張する。
2. **データベース**を拡張する。
3. 操作したいデータベースを拡張して、「**ライブラリー**」を拡張する。
4. ジャーナル情報を表示したい表が入っているライブラリーをクリックする。
5. 表を右クリックして、「**ジャーナル処理**」を選択する。
6. 表がまだ一度もジャーナル処理されたことがない場合は、該当するボックスにジャーナル名およびライブラリー名をタイプ入力するか、「**ブラウズ**」ボタンをクリックして、表用に使用したいジャーナルの位置までナビゲートして、使用したいジャーナルを選択することができます。
7. 変更前イメージをジャーナル処理するには、「**変更前イメージのジャーナル処理**」オプションを選択する。
8. オープン項目およびクローズ項目をジャーナル処理しないようにするには、「**オープン項目およびクローズ項目の除外**」オプションを選択する。

***iSeries* ナビゲーターを使用したレシーバーのスワッピング:** ジャーナル・レシーバーをスワップすると、現行のジャーナル・レシーバーが、システムによって自動的に作成されジャーナルに付加された新規のジャーナル・レシーバーで置き換えられます。ジャーナル処理のためにレシーバーをスワップするのに使用できるメソッドが 2 つあります。1 番目のメソッドでは、新しいレシーバーの属性を変更することはできませんが、 2 番目のメソッドでは変更できます。

1. **iSeries ナビゲーター**・ウィンドウで、使用したいシステムを拡張する。
2. **データベース**を拡張する。
3. 操作したいデータベースを拡張して、「**ライブラリー**」を拡張する。
4. 使用したいジャーナルを右クリックし、「**レシーバーのスワップ**」を選択する。システムは、レシーバーを作成するときに、新しい名前を生成します。

または

1. **iSeries ナビゲーター**・ウィンドウで、使用したいシステムを拡張する。
2. **データベース**を拡張する。
3. 操作したいデータベースを拡張して、「**ライブラリー**」を拡張する。
4. 使用したいジャーナルをダブルクリックする。
5. 「**レシーバー**」をクリックする。このウィンドウには、ジャーナルに関連付けられているすべてのレシーバーが表示されます。
6. 新しいレシーバーを追加するために、「**新規**」をクリックする。

7. **OK** をクリックする。
8. **OK** をクリックして、「**ジャーナル・レシーバー**」ウィンドウをクローズする。再び **OK** をクリックすると、新規ジャーナル・レシーバーは、その状況を「付加された」に変更します。

ジャーナル・レシーバーの状況は以下のいずれかになります。

付加された
ジャーナル・レシーバーは現在ジャーナルに付加されていることを示します。「空」は、レシーバーが一度もジャーナルに付加されたことがないことを示します。

解放された
ジャーナル・レシーバーは、切り離された後に、保管されたことを示します。レシーバーのストレージは、レシーバーが保管されたときに解放されます。

オンライン
ジャーナル・レシーバーがオンラインになっていることを示します。

部分的
以下の理由のいずれかのために、状況が部分的であることを示します。

- ジャーナル・レシーバーは、ジャーナルに付加されていたときに保管されたバージョンから復元されました。書き込まれた可能性のある追加のジャーナル項目が、復元されていません。
- ジャーナル・レシーバーがデュアル・ジャーナル・レシーバーのペアの一方で、ジャーナルに付加されていたときに損傷を受けたことがわかっています。それ以来、このジャーナル・レシーバーは切り離されています。追加のジャーナル項目がデュアル・ジャーナル・レシーバーに書き込まれている可能性があるので、このジャーナル・レシーバーは部分的と見なされます。
- ジャーナル・レシーバーは遠隔ジャーナルに関連付けられていますが、ソース・ジャーナルに付加されている関連ジャーナル・レシーバーの中にあるジャーナル項目のすべてが入っているとは限りません。保留は、ジャーナル・レシーバーがまだ作成されていないことを示します。ジャーナル・レシーバーは、「ジャーナル・プロパティー」ウィンドウで OK が選択された後で作成され付加されます。保管済みは、ジャーナル・レシーバーは、切り離された後で、保管されたことを示します。ジャーナル・レシーバーのストレージは、レシーバーが保管されたときに解放されていません。

iSeries ナビゲーターを使用した、表 (ファイル) のジャーナルの開始/停止: ジャーナルが作成されたならば、表のために開始しなければなりません。ジャーナルを削除する場合は、停止しなければなりません。

1. **iSeries ナビゲーター**・ウィンドウで、使用したいシステムを拡張する。
2. **データベース**を拡張する。
3. 編集したいジャーナルが入っているデータベースおよびライブラリーを拡張する。
4. 編集したいジャーナルが入っているライブラリーをクリックする。
5. ジャーナルを右クリックし、「**表のジャーナル処理の開始と終了**」を選択する。
6. 表 (ファイル) のジャーナル処理を開始するには、**表リスト**からジャーナル処理したい表を選択し、「**追加**」をクリックする。または、**表**リストにある表をドラッグし、**ジャーナル処理する表**のリストの上でドロップすることもできます。
7. 表のジャーナル処理を終了するには、「**すでにジャーナル処理中の表**」のリストから、もはやジャーナル処理しない表を選択し、「**除去**」をクリックする。
8. すべての表のジャーナル処理を同時に終了する場合は、「**すでにジャーナル処理中の表**」のリストにあるすべての表を選択するために、「**すべてを選択**」をクリックして、「**除去**」をクリックする。
9. 「**OK**」 をクリックして、「**ジャーナル処理の開始/停止**」ウィンドウをクローズする。

または

1. ライブラリー・ツリー・リストから、ジャーナル処理を開始または停止したいオブジェクトを右クリックし、「**ジャーナル処理**」を選択する。
2. 「**停止**」をクリックして、選択されたオブジェクトのジャーナル処理を停止する。
3. オブジェクトからジャーナル処理を開始するには、以下のようにします。
   a. オブジェクトに関連付けるジャーナルを選択する。「**ブラウズ**」を選択して、ジャーナルをブラウズすることもできます。
   b. ジャーナルが入っているライブラリーを選択する。「**ブラウズ**」からジャーナルを選択するときは、このフィールドは自動的に埋められます。
   c. 変更前イメージをジャーナル処理するには、「**変更前イメージのジャーナル処理**」オプションを選択する。
   d. オープン項目およびクローズ項目をジャーナル処理しないようにするには、「**オープン項目およびクローズ項目の除外**」オプションを選択する。
   e. 「**開始**」をクリックして、選択されたオブジェクトのジャーナル処理を開始する。

コミットメント制御を使用したデータの保全性の保証: コミットメント制御を使用すると、いくつかの変更点を単一の単位 (トランザクション) としてデータベース・ファイルに対して定義および処理できます。コミットメント制御は、ジョブまたはシステムが終了した場合でも、複雑なアプリケーション・トランザクションが必ず論理的に同期化されるようにします。2 フェーズのコミットメント制御により、複数のシステム上にあるデータベース・ファイルなどのコミット可能資源を同期化されたままにします。

コミットおよびロールバック処理を実行して、データベースにコミットメント制御を導入します。SQL を使用する場合、COMMIT および ROLLBACK ステートメントを使います。

コミットメント制御については、以下のトピックを参照してください。

- 『トランザクション』
- 221 ページの『コミットメント制御を使用する利点』
- 221 ページの『使用上の注意: コミットメント制御』

さらに、コミットメント制御のトピックも参照してください。

トランザクション: トランザクションは、たとえば貯蓄口座から当座預金口座への資金の振替のように、単一の変更として示される複数の変更点のグループのことです。トランザクションは以下のように類別できます。

- ファイルの変更が発生しない照会
- 実行キーを押すたびに 1 つのファイルが変更される単純なトランザクション
- 実行キーを押すたびに 2 つ以上のファイルが変更される複雑なトランザクション
- 実行キーを押すたびに 1 つまたは複数のファイルが変更される複雑なトランザクション。これらの変更は、トランザクションの論理グループの一部分しか表していません。

トランザクション処理中にファイルに加えられた変更点は、コミットメント制御使用時にジャーナル処理されます。

システムまたはジョブが異常終了した場合に、最終のレコード変更の消失だけで済むのは、ジャーナル処理を行った場合だけです。しかし、システムまたはジョブが複雑なトランザクション時に異常終了した場合には、ファイルは未完了の論理トランザクションを反映します。たとえば、ジョブがファイル A のレコード

を更新した可能性があり、このジョブがファイル B 内にある対応レコードを更新する前に異常終了したような場合です。この場合、論理トランザクションは 2 つの更新から成り立っていますが、ジョブが異常終了する前に更新は 1 つしか完了していません。

コミットメント制御を使用する利点: 複雑なアプリケーションを回復するためには、アプリケーション・プログラムについて詳しく知っていなければなりません。プログラムは、再起動できません。たとえば、最後の複合トランザクションが始まる直前の状態にファイルを戻すために、アプリケーション・プログラムまたはデータ・ファイル・ユーティリティーによりレコードを変更しなければならない場合もあります。この作業は、複数のユーザーが同時にフィールドにアクセスしている場合には、さらに複雑になります。

このような問題を解決する上で、コミットメント制御機能が役に立ちます。コミットメント制御は、複雑なトランザクション中に他のユーザーからレコードをロックします。このようにすると、トランザクションが完了するまで他のユーザーは、レコードを使用できなくなります。トランザクションが終了すると、プログラムはコミット操作を出して、レコードを解放します。ただし、コミット操作が実行される前にシステムが異常終了した場合には、前回のコミット操作の発生以降そのジョブに対して行われたレコードの変更は、すべてロールバックされます。コミットメント制御を受けたレコードで、まだロック状態にあるものは、ロック解除されます。言い換えると、データベース変更がロールバックして、問題のないトランザクション境界まで戻ります。

使用上の注意: コミットメント制御: コミット操作およびロールバック操作は、 RPG/400、COBOL/400、 PL/I、SQL、および OS/400 制御言語 (CL) を含むいくつかの iSeries プログラム言語で使用できます。

基本となる物理ファイルが別のジャーナルに対してジャーナル処理される場合、論理ファイルを開いてコミットメント制御下へ出力することができます。ただし、レコードの変更が、同一のジャーナルに対してジャーナル処理された基本となる物理ファイルに影響するかどうかにより、違反検査が行われます。同一のジャーナルにジャーナル処理されない基本となる物理ファイルにデータの変更が影響し、そのために重複キーまたは参照制約違反が起こる場合、入出力処理中にエラーが発生します。たとえば、固有のキーを持つ A という物理ファイルがジャーナル X に対してジャーナル処理され、一方、固有のキーを持つ物理ファイル B がジャーナル Y に対してジャーナル処理されたと仮定します。論理ファイル C は、物理ファイル A および B の上に作成され、コミットメント制御下で開かれます。論理ファイル C による削除処理では、キー K を使用して物理ファイル A からレコードが削除されます。トランザクションがコミットされる前に、キー K を使用して物理ファイル A にレコードを戻すこともできます。ただし、トランザクションがコミットされる前に、キー K を使用して物理ファイル B にレコードを追加しようとすると、物理ファイル A および B はそれぞれ異なるジャーナルに対してジャーナル処理されているため、トランザクションは失敗します。

コミットメント制御はバッチ環境でも使用することができます。コミットメント制御は、対話式のトランザクションの回復にもバッチ・ジョブの回復にも役立ちます。コミットメント制御の詳細については、 コミットメント制御のトピックを参照してください。

アクセス・パスの回復時間の短縮

システムは、ユーザーがアクセス・パスを使用する前に、そのパスの保全性を確保します。アクセス・パスが使用不能であるとシステムが判断した場合、システムはその回復を試みます。ユーザーは、アクセス・パスの回復時期を制御することができます。

アクセス・パスの回復には時間がかかります。再作成したいアクセス・パスが大きいか、その数が多い場合には特に時間がかかります。いくつかの方法により、回復時間を短縮できます。

ジャーナル処理アクセス・パスは、アクセス・パスを再作成するよりも時間が短縮できます。システム管理アクセス・パス保護 (SMAPP) サポートを使用すると、STRJRNAP などのジャーナル処理コマンドを使用しなくても、アクセス・パス・ジャーナル処理の利点が得られます。SMAPP は、IPL 時にアクセス・パスを再作成するのではなく、システムの異常終了後にアクセス・パスを回復するためのサポートです。

以下のトピックでは、アクセス・パス回復時間を短縮する方法を説明しています。

- 『アクセス・パスの保管』
- 『アクセス・パスの復元』
- 『アクセス・パスのジャーナル処理』
- 223 ページの『システム管理のアクセス・パス保護機能 (SMAPP)』
- 224 ページの『アクセス・パスの再作成』

アクセス・パスの保管: アクセス・パスの保管により、アクセス・パスの回復時間を短縮できます。SAVCHGOBJ、SAVLIB および SAVOBJ コマンドのアクセス・パス (ACCPTH) パラメーターにより、アクセス・パスを保管できます。通常、システムは論理ファイルの記述のみを保管しますが、以下の状況では、アクセス・パスを保管します。

- ACCPTH(*YES) が指定されている場合。
- 論理ファイル下のすべての物理ファイルが保管され、同じライブラリーに置かれる場合。
- 論理ファイルが、MAINT(*IMMED) または MAINT(*DLY) である場合。

ACCPTH(*YES) パラメーターを指定したときは、論理ファイル自体は保管されないことに注意してください。論理ファイルは、明示的に保管する必要があります。詳細については、「バックアップおよび回復の手引き」を参照してください。

アクセス・パスの復元: システムは以下の場合にアクセス・パスを復元できます。

- アクセス・パスが以前に保管されている場合。
- アクセス・パスの属するすべての物理ファイルが同時に復元される場合。

通常、アクセス・パスを再作成するよりも速くアクセス・パスを復元します。

たとえば、 500,000 レコードを含む物理ファイル上に論理ファイルが作成されるとします。ユーザーが、オブジェクト記述の表示 (DSPOBJD) コマンドを使用して論理ファイルのサイズを約 15 メガバイトに決定したとします。

この例では、論理ファイルのアクセス・パスを再作成するには約 50 分かかります。同じアクセス・パスをテープから復元するには、約 1 分しかかかりません。 (これはシステムが 1 分間におよそ 10,000 索引項目を作成すると想定した場合です。)

アクセス・パスの復元後、最新のジャーナル変更点を適用することによってファイルを最新の状態にすることが必要な場合があります。たとえば、システムは 1 時間におよそ 80,000 から 100,000 のジャーナル項目を適用できます。これは、項目が適用されている物理ファイルのそれぞれについて、アクセス・パスが 1 つだけ作成されていると想定した場合です。この速度は、その物理ファイルに対する *IMMED メインテナンスの各アクセス・パスごとに比例して低下します。この付加的な回復時間があるとしても、普通、アクセス・パスを再作成するよりも、復元する方が速いことがわかります。

追加情報については、 ジャーナル管理のトピックを参照してください。

アクセス・パスのジャーナル処理: アクセス・パスのジャーナル処理は、システムの異常終了後に再作成の必要のあるアクセス・パスの数を減らすので、回復時間を大幅に短縮することができます。 iSeries バー

ジョン 2 リリース 2 とそれ以降のリリースの場合、アクセス・パスのジャーナル処理をお勧めします。アクセス・パスが大きくなると、再作成により多くの時間が必要になるからです。

データベース・ファイルをジャーナル処理すると、ファイルに対する変更のイメージがジャーナルに記録されます。システムは、システムが異常終了した後に、これらのレコード・イメージを使用してファイルを回復します。

異常終了後に、作成されたファイルのアクセス・パスがファイル内のデータと同期化されていないことをシステムが発見する場合があります。アクセス・パスとそのデータが同期化されていない場合には、システムでは、この 2 つが同期化されて使用可能な状態になるよう、アクセス・パスを作成し直します。

アクセス・パスのジャーナル処理時に、システムはジャーナル内のアクセス・パスのイメージを記録し、アクセス・パスとそのデータの間に既知の同期点を設定します。ジャーナル内にこの情報が入れられると、システムはデータ・ファイルとアクセス・パスの両方を回復します。次にシステムはこの 2 つを同期化します。このような場合は、アクセス・パスを再作成するための時間を短縮できます。

さらに、その他のシステム回復機能がアクセス・パスのジャーナル処理とともに実行されます。たとえば、システムには、ディスク装置の障害および交換からの回復時間を短縮する上で役立つたくさんのオプションがあります。これらのオプションには、ユーザー補助記憶域プールおよびチェックサム保護が含まれます。これらのオプションは、さらにディスクの障害によるシステム全体の再ロードが必要となる可能性を減らします。ただし、障害のあるディスクの交換後にシステムが開始された時点でアクセス・パスの再作成が必要になる場合があることには変わりありません。アクセス・パスのジャーナル処理、および回復オプションをいくつか使用することによって、システム全体の再ロードやアクセス・パスの再作成が必要となる可能性を減らすことができます。

アクセス・パスのジャーナル処理を行う前に、ユーザーはアクセス・パスに関連した物理ファイルをジャーナルする必要があります。また、アクセス・パスおよびその関連物理ファイルに対して同じジャーナルを使用しなければなりません。アクセス・パスのジャーナル処理は、簡単に開始することができます。

- システム管理のアクセス・パス保護 (SMAPP) 機能を使用できます。
- アクセス・パスのジャーナル開始 (STRJRNAP) コマンドを使用して、ジャーナル処理環境をユーザー自身で管理することができます。
  - 指定したファイルのアクセス・パス・ジャーナル処理を開始するには、 STRJRNAP コマンドを使用します。即時 (*IMMED) または遅延 (*DLY) のメインテナンス属性を持つアクセス・パスのジャーナル処理を行うことができます。
  - いったんジャーナル処理が開始されると、アクセス・パスが削除されるか、ユーザーがこのアクセス・パスに対してアクセス・パス・ジャーナルの終了 (ENDJRNAP) コマンドを実行するまで、システムはアクセス・パスを保護します。

アクセス・パス・ジャーナル処理は、追加の出力操作を最小限に押さえます。たとえば、システムは、変更済みレコードおよび変更済みアクセス・パスのためのジャーナル・データを同じ出力操作で書き出します。しかし、ユーザーが自分のアクセス・パスのジャーナル処理を開始する場合には、ユーザー補助記憶域プールにおけるユーザーのジャーナル・レシーバーを必ず分離してください。ジャーナル・レシーバーを独自のユーザー補助記憶域プールに入れると、ジャーナル処理のパフォーマンスは最も良くなり、ディスク障害からの保護にも役立ちます。ジャーナル処理のアクセス・パスについて、詳しくはジャーナル管理のトピックを参照してください。

システム管理のアクセス・パス保護機能 (SMAPP): システム管理のアクセス・パス保護機能 (SMAPP) は、アクセス・パスの自動保護を行います。SMAPP サポートを使用すると、STRJRNAP などのジャーナル処理コマンドを使用しなくても、アクセス・パスのジャーナル処理の利点が得られます。SMAPP は、

IPL 時にアクセス・パスを再作成するのではなく、システムの異常終了後にアクセス・パスを回復するためのサポートです。システムの出荷時には、SMAPP サポートが自動的にオンになっています。SMAPP サポートの値は、70 分に設定されています。

システムは、どのアクセス・パスを保護するかを以下に基づいて判別します。

- ユーザーが定めるアクセス・パスの目標回復回数、または
- システム提供の省略時時間。

アクセス・パスの目標回復回数は、システム全体の値として指定するか、または ASP を基礎として指定します。ユーザー定義のジャーナルにすでに入っているアクセス・パスは、 SMAPP 保護の対象とはなりません。それらのアクセス・パスはすでに保護されているからです。 SMAPP の詳細については、 ジャーナル管理のトピックを参照してください。

アクセス・パスの再作成: データベース・アクセス・パスの再作成は、10,000 のレコードごとに 1 分ほどかかります。

以下の要素は、アクセス・パスの再作成時の概算時間に影響します。

- 記憶域プール・サイズ。大きな記憶域プールでジョブを実行する方が、再作成に要する時間は短くなります。
- システムの型式。処理装置の速度は、主な要素です。
- キーの長さ。キーの長さが長ければ、アクセス・パスはそれだけ多量のキー情報を作成して保管しなければならないため、アクセス・パスの再作成に要する時間は長くなります。
- 選択/除外基準。選択/除外処理は、システムが各レコードごとに比較して選択/除外基準と一致しているか調べるため、アクセス・パスの再作成は遅くなります。
- レコード長。レコードの長さが長いと、それだけ多量のデータを調べるため、アクセス・パスの再作成は遅くなります。
- データが入っている記憶装置。実際のデータが入っている記憶装置およびアクセス・パスが保管される装置の相対速度が、アクセス・パス再作成の所要時間に影響します。
- ファイル中のレコードの順序。システムは、アクセス・パスの使用時に迅速に情報が見つけられるように、アクセス・パスの再作成を試みます。ファイル中のレコードの順序は、システムが効率のよいアクセス・パスを保持するよう試みるので、アクセス・パスを作成する速度にわずかながら影響します。

以下のトピックでは、アクセス・パスの再作成に関する技法についての詳細を説明しています。

- 『アクセス・パス再作成時の制御』
- 225 ページの『アクセス・パスの再作成時間を短縮するためのファイル設計』
- 225 ページの『アクセス・パスの再作成を回避するための他の方法』

アクセス・パス再作成時の制御: システムが異常終了した場合、次の IPL の間に、システムは自動的にアクセス・パスの回復が必要なファイルのリストを表示します。ユーザーは、アクセス・パスを次のどの時点で再作成するかを決定できます。

- IPL 中
- IPL の後
- ファイルを最初に使用するとき

また次のことも行えます。

- アクセス・パスが再作成されるスケジューリングの順序を変更する。
- アクセス・パスの再作成を無期限に保留する。

- **IPL 限界値* 以下のシーケンス値を持つアクセス・パスの再作成中に、IPL プロセスを続行する。
- システムが IPL 処理を完了したのち、アクセス・パス再作成の編集 (EDTRBDAP) コマンドを使用して、アクセス・パスの再作成を制御する。

IPL 限界値は、IPL 中にどのアクセス・パスを再作成するかを決定します。IPL 限界値以下のシーケンス値を持つすべてのアクセス・パスは、IPL 中に再作成されます。IPL 限界値を 99 へ変更するということは、1 から 99 までのシーケンス値を持つすべてのアクセス・パス を IPL 中に再作成することを意味します。IPL 限界値を 0 に変更するということは、システムが IPL を完了するまでは、どのアクセス・パスも再作成しないということです。ただし、ジャーナル処理されていたアクセス・パスと、システム・ファイルのアクセス・パスは再作成されます。

ファイルのアクセス・パス回復値は、ファイル作成コマンドおよびファイル変更コマンドの RECOVER パラメーターに指定した値により決定されます。*IPL (IPL 中に再作成する) の省略時の回復値が 25 で、AFTIPL (IPL の後に再作成する) の省略時の値が 75 のとき、RECOVER(*IPL) は 25 と示されます。初期 IPL 限界値が 50 のとき、このパラメーターにより、いつアクセス・パスが再作成されるかに影響を与えることができます。「アクセス・パス再作成の編集」画面で、この値を変更できます。

IPL 直後にファイルが必要でない場合、後でファイルを再作成できるように指定してください。これは、IPL 時に再作成が必要なアクセス・パスを減らすのに役立ち、システムが IPL をかなり早く完了できるようにします。

たとえば、アクセス・パスを再作成しなければならないすべてのファイルは、ファイルが最初に使われるときアクセス・パスを再作成するよう指定できます。この場合、IPL 時にはアクセス・パスはまったく再作成されません。最初に再作成したいファイルを使うプログラムだけを実行することによって、アクセス・パスが再作成される順序を制御できます。この方法によって、 IPL 時間が短くなり、いくつかのアプリケーションが使用可能になる時点が早まります。ただし、アクセス・パスを再作成する総計時間は長くなります。(アクセス・パスの再作成中に他の作業が実行中で、アクセス・パスを再作成するために使用できる主記憶域が少ない可能性があるからです)。

アクセス・パスの再作成時間を短縮するためのファイル設計: ファイルの設計によっても、アクセス・パス回復時間を短縮できます。たとえば、大きなマスター・ファイルを、活動記録ファイルとトランザクション・ファイルに分けることができます。システムは、新しいデータの追加にトランザクション・ファイルを使用します。活動記録ファイルは、Query だけに使用します。毎日、トランザクション・データを活動記録ファイルの中に組み込んで、次の日のデータのためにトランザクション・ファイルを消去することができます。この設計により、アクセス・パスを再作成する時間を短縮できます。

ただし、システムが日中異常終了した場合は、比較的小さいトランザクション・ファイルへのアクセス・パスを再作成する必要がある場合があります。しかし、大きな活動記録ファイルへのアクセス・パスは、一日の大半は読み取りのみなので、データと非同期になることはほとんどありません。したがって、このアクセス・パスを再作成する必要が少なくなります。

アクセス・パス再作成の時間を短縮するためのファイル設計を行うことと、アクセス・パス・ジャーナル処理のようなシステムが提供する機能を使用することの利点を比較考慮してみてください。上記のファイル設計は、より複雑なアプリケーション設計を必要とします。ユーザーの状況を考慮した後、より複雑なアプリケーションを設計するよりも、アクセス・パスのジャーナル処理のようなシステムが提供する機能を使うこともできます。

アクセス・パスの再作成を回避するための他の方法: アクセス・パスをジャーナルしない場合や、SMAPP の利点を活用しない場合は、アクセス・パスを再作成しなくて済むシステム機能が他にないか考えてみます。

システムは、アクセス・パスを再作成する必要があるかどうかを判別するためにファイル同期化標識を採用します。通常、同期化標識がオンである場合には、アクセス・パスと、それに結び付けられるデータが同期化されていることを示します。アクセス・パスに影響するファイルがジョブによって変更される場合には、システムはファイル内の同期化標識をオフにします。システムが異常終了した場合には、同期化標識がオフになっているファイルを持つアクセス・パスを作成し直さなければなりません。

再作成しなければならないアクセス・パスの数を少なくするためには、定期的にデータをアクセス・パスと同期化しておく必要があります。ファイルをアクセス・パスと同期化する方法は、以下に示すようにいくつかあります。

- 全ファイル・クローズ。ファイルに対して実行された最後の全 (すなわち、共用されていない) システム規模クローズにより、アクセス・パスとデータが同期化されます。
- 強制アクセス・パス。アクセス・パスの強制書き出し (FRCACCPTH) パラメーターをデータベース・ファイル作成、変更または一時変更コマンドで指定します。
- 2 またはそれ以上の強制書き出し。強制書き出し (FRCRATIO) パラメーターをデータベース・ファイルの作成、変更または一時変更コマンドで指定します。
- データの強制終了。プログラム内でデータの強制終了操作を実行すると、ファイルのデータとそのアクセス・パスを同期化することができます。(高水準言語によってはデータの強制終了操作が行えないものもあります。詳細については、該当する高水準言語の解説書を参照してください。)

上記の方法の 1 つを実行すると、アクセス・パスとそのデータが同期化されます。ただし、ファイル内のデータに次の操作を行うと、同期化標識が再度オフになる可能性があります。

また、上記の方法は、いずれもパフォーマンスに多大な影響を及ぼすため、使用にあたっては注意が必要です。アクセス・パス保護の基本手段としてアクセス・パスの保管または SMAPP の使用だけでなく、アクセス・パスのジャーナル処理も検討してください。

システムの異常終了後のデータベースの回復

システムは、異常終了した後、いくつかの自動回復ステップを経て処置を継続します。システムはディレクトリーを再作成して、ジャーナル処理するファイルとジャーナルを同期化します。システムは、IPL 時と IPL 後に回復操作を実行します。

以下のトピックに特定のデータベース・ファイルの回復について説明しています。

- 『IPL 時のデータベース・ファイルの回復』
- 227 ページの『IPL 後のデータベース・ファイルの回復』
- 228 ページの『記憶域プール・ページング・オプションのデータベース回復に対する影響』
- 228 ページの『データベース・ファイルの回復オプションの表』

IPL 時のデータベース・ファイルの回復: IPL 時には、システム上で活動状態になっているのは回復機能だけです。データベース・ファイルの回復は以下のように行われます。

- システムの終了時に進行中であった以下の機能は完了します。
  - ファイル削除
  - メンバーの除去
  - メンバーの名前変更
  - オブジェクトの移動
  - オブジェクトの名前変更
  - オブジェクト所有者の変更
  - メンバーの変更
  - 権限の認可

  - 権限の取り消し
  - 物理ファイルのジャーナル処理開始
  - アクセス・パスのジャーナル処理開始
  - 物理ファイルのジャーナル処理終了
  - アクセス・パスのジャーナル処理終了
  - ジャーナルの変更
  - ジャーナルの削除
  - SQL ビューの回復
  - 物理ファイルの制約の除去
- システムの終了時に進行中であった以下の機能は、バックアウトされます (再実行しなければなりません)。
  - ファイルの作成
  - メンバーの追加
  - ファイルの変更
  - ジャーナルの作成
  - ジャーナルの復元
  - 物理ファイルの制約の追加
- 操作員が IPL を実行中の場合 (在席 IPL) は、「アクセス・パス再作成の編集」画面が操作員の表示装置に表示されます。この画面から操作員は、即時または遅延メインテナンスが指定された使用中だったファイルに対して、RECOVER オプションを編集することができます。アクセス・パスがすべて有効な場合、また IPL が不在で行われている場合には、画面が表示されません。
- アクセス・パスは、
  - 即時または遅延メインテナンスを持っています。
  - IPL 時の回復が指定されています (RECOVER オプションから指定されたかまたは「アクセス・パス再作成の編集」画面で変更された)。
  - 再作成され、アクセス・パスのジャーナル処理を開始したときにメッセージが送られます。ジャーナル・レシーバーを独自のユーザー補助記憶域プールに入れると、ジャーナル処理のパフォーマンスは最も良くなり、ディスク障害からの保護にも役立ちます。ジャーナル処理のアクセス・パスについて、詳しくはジャーナル管理のトピックを参照してください。

IPL 後のデータベース・ファイルの回復: データベース・ファイルのこの回復は、IPL が完了した後に実行されます。対話式ジョブおよびバッチ・ジョブがこのデータベース回復の手順と一緒に実行している場合があります。

IPL 後の回復は以下のように行われます。

- IPL 後の回復を指定する、即時メインテナンスまたは遅延メインテナンス・ファイルのアクセス・パスは、再作成されます。
- システム活動記録ログは、再作成操作の成否を示すメッセージを受け取ります。
- IPL の完了後、アクセス・パス再作成の編集 (EDTRBDAP) コマンドを使用して、アクセス・パスの再作成の順序を決めます。
- IPL の完了後、検査保留制約の編集 (EDTCPCST) コマンドで、検査保留になっている物理ファイル制約のリストが表示されます。このコマンドは、検査保留中の制約の検査順序を指定するために使用します。

注: ファイルのジャーナル処理を使用していない場合、以下のように IPL 後にレコードが存在する場合と存在しない場合とがあります。

- 追加レコードの場合、IPL 回復後追加された N 番目のレコードが存在していれば、N 以前に追加されたレコードも存在します。
- 更新および削除レコードの場合、N 番目のレコードに対する更新または削除が IPL 回復後に存在していても、N 番目のレコード以前に更新または削除されたレコードもデータベースに存在するという保証はありません。
- REUSEFLT(*YES) の場合、追加されたレコードは更新として扱われるので、IPL 回復後にレコードが存在しない場合があります。

記憶域プール・ページング・オプションのデータベース回復に対する影響: 共用プール・ページング・オプションは、最適パフォーマンスのために、システムが記憶域プールのページング特性を動的に調整するかどうかを制御します。

- システムは *FIXED のページング・オプションのページング特性を、動的には調整しません。
- システムは *CALC のページング・オプションのページング特性を、動的に調整します。
- ユーザーはまた、アプリケーション・プログラミング・インターフェースを介して、ページング特性を制御することもできます。詳細については、 アプリケーション・プログラミング・インターフェース (API) の『Change Pool Tuning Information API (QWCCHGTN)』を参照してください。

*FIXED 以外の共用プール・ページング・オプションは、システム障害のさいのジャーナル処理されていない物理ファイルのデータ消失に対して影響を与えることがあります。*CALC または USRDFN ページング・オプションの場合は、物理ファイルをジャーナル処理しないと、システム障害 (メモリーが保管されない) で消失されるデータが増加する可能性があります。ファイル変更が補助記憶域に書き込まれる頻度は、これらのオプションの場合は少なくなります。*FIXED オプションが指定されている、ジャーナル処理されていないファイルには、データ消失のリスクがありますが、*CALC またはユーザー定義 (USRDFN) ページング・オプションの場合の方がリスクが高くなる可能性があります。

ページング・オプションについて詳しくは、 パフォーマンス の、「自動システム調整」のセクションを参照してください。

データベース・ファイルの回復オプションの表: 以下の表は、ファイル回復オプションをまとめたものです。

指定された RECOVER パラメーター			
アクセス・パス/メインテナンス	***NO**	***AFTIPL**	***IPL**
キー順アクセス・パス/即時メインテナンスまたは遅延メインテナンス	• IPL 時のデータベース回復なし • ファイル即時使用可能 • ファイルの最初のオープン時にアクセス・パスが再作成される	• IPL 後アクセス・パス再作成	• IPL 中アクセス・パス再作成
キー順アクセス・パス再作成メインテナンス	• IPL 時のデータベース回復なし • ファイル即時使用可能 • ファイルの最初のオープン時にアクセス・パスが再作成される	• 適用外: 再作成メインテナンスの回復は行われない	• 適用外: 再作成メインテナンスの回復は行われない

指定された RECOVER パラメーター			
アクセス・パス/メインテナンス	***NO**	***AFTIPL**	***IPL**
到着順アクセス・パス	• IPL 時のデータベース回復なし • ファイル即時使用可能	• 適用外: 到着順アクセス・パスの回復は行われない	• 適用外: 到着順アクセス・パスの回復は行われない

データベースの保管と復元

データベース・ファイルとそれに関連するオブジェクトは、サポートされる装置および媒体または保管ファイルを使用して、保管および復元できます。保管ファイル (または媒体) は、特殊な形式で書き込まれた保管された情報のコピーを受け取ります。媒体は、ユーザーのシステムまたは別の iSeries システムで将来使用するために除去し、保管しておくことができます。復元された情報は、媒体または保管ファイルからシステム使用者がアクセスできる記憶装置へ読み込まれます。

保管ファイルは、保管操作の対象ファイルまたは復元操作のソース・ファイルとなるディスク常駐ファイルのことです。保管ファイルにより、不在時保管操作を行うことができます。保管ファイルに保管する場合、操作員がテープやディスケットをロードする必要がありません。ただし、 保管ファイル・データの保管 (SAVSAVFDTA) コマンドを使用して、テープやディスケットに保管ファイル・データを定期的に保管してください。定期的にテープやディスケットを装置から取り出して他の場所に、保管してください。これらの媒体は、設定場所で非常事態が生じたときに回復を行うのに役立ちます。

情報の保管および復元について、詳しくは「バックアップおよび回復の手引き」、およびバックアップおよび回復のトピックを参照してください。

データベースの保管と復元の考慮事項

以下のリストでは、保管および復元機能に関するヒントを提供します。

- オブジェクトを保管ファイルへ保管する場合、保管コマンドで UPDHST(*NO) を指定することにより、システムが保管操作の日付と時刻を更新しないようにすることができます。
- オブジェクトを復元するとき、システムは常に、復元操作の日付と時刻を使用して、オブジェクト記述を更新します。DETAIL(*FULL) を指定して オブジェクト記述の表示 (DSPOBJD) コマンドを使用することにより、オブジェクト記述と他の保管/復元に関連した情報を表示できます。
- 保管ファイルのオブジェクトを表示するには、 保管ファイルの表示 (DSPSAVF) コマンドを使用してください。
- 媒体上のオブジェクトを表示するには、 ディスケットの表示 (DSPDKT) コマンド、または テープの表示 (DSPTAP) コマンドで DATA(SAVRST) を指定してください。
- データベース・ファイルの最新の保管/復元日付を表示するには、DSPFD FILE(ファイル名) TYPE(*MBR) をタイプします。

さらに、レコードを補助記憶装置に自動的に書き込むこともできます。 230 ページの『補助記憶装置へのデータ強制書き込み』を参照してください。

情報の保管および復元について、詳しくは「バックアップおよび回復の手引き」、およびバックアップおよび回復のトピックを参照してください。

補助記憶装置へのデータ強制書き込み: ファイルの作成およびデータベース・ファイルの一時変更コマンドの強制書き出し頻度 (FRCRATIO) パラメーターは、レコードの補助記憶装置への書き込み頻度を示します。強制書き出し頻度に 1 を指定すると、考慮中のファイルに対するすべての追加、更新、削除要求が補助記憶装置へ即座に書き込まれます。しかし、このオプションを選択すると、システム・パフォーマンスが低下することがあります。したがって、ファイルの保管とファイルのジャーナル処理が、データベース・ファイルの保護のための基本的な方法とみるべきです。

ソース・ファイルの使用

この章では、ソース・ファイルにデータを入力し管理する方法、および、そのソース・ファイルを使用して、システム上に別のオブジェクト (たとえば、ファイルまたはプログラム) を作成する方法について説明します。以下のトピックを参照してください。

- 『ソース・ファイルの処理』
- 233 ページの『ソース・ファイルを使用したオブジェクトの作成』
- 235 ページの『ソース・ファイルの管理』

ソース・ファイルのセットアップ方法についての説明は、 16 ページの『ソース・ファイルのセットアップ』を参照してください。

ソース・ファイルの処理

以下のセクションで、さまざまなメソッドを使用してデータを入力し、管理する方法を説明します。

- 『原始ステートメント入力ユーティリティー (SEU) の使用』
- 『装置ソース・ファイルの使用』
- 231 ページの『ソース・ファイル・データのコピー』
- 232 ページの『iSeries 以外のシステムからのデータのロードおよびアンロード』
- 233 ページの『プログラムでのソース・ファイルの使用』

原始ステートメント入力ユーティリティー (SEU) の使用: 原始ステートメント入力ユーティリティー (SEU) を使用して、ソース・ファイルの中にソースを入力し変更することができます。SEU は、IBM WebSphere Development Studio for iSeries の一部です。SEU を使用してデータベース・ファイルに原始ステートメントを入力すると、 SEU は各ソース・レコードに順序番号フィールドと日付フィールドを追加します。

SEU を使用してソース・ファイルを更新する場合は、既存のレコードの間にレコードを追加できます。たとえば、レコード 0003.00 と 0004.00 の間にレコードを追加する場合は、追加されるレコードの順序番号を 0003.01 にすることができます。SEU は、新しく追加するステートメントをこのように自動的に配列します。

レコードが最初にソース・ファイルに入れられた場合、日付フィールドはすべてゾーン 10 進数のゼロです (DFT キーワードが指定された状態で DDS を使用しない場合)。SEU を使用した場合、レコードが変更されるとレコード内の日付フィールドが変わります。

データベース・ソース・ファイルの更新方法については、 V5R1 補足資料 Web サイトの「適用業務開発ツールセット AS/400® 用 原始ステートメント入力ユーティリティー」を参照してください。

装置ソース・ファイルの使用: テープ装置ファイルおよびディスケット装置ファイルは、ソース・ファイルとして作成できます。装置ファイルをソース・ファイルとして使用する場合は、レコード長に順序番号フィールドおよび日付フィールドを含める必要があります。最大レコード長の制約条件には、この追加の 12

文字を考慮に入れなければなりません。たとえば、テープ・レコードの最大レコード長は 32,766 です。データをソース入力として処理する場合、実際のテープ・データ・レコードは最大 32,754 (32,766 から 12 を差し引いたもの) の長さでなければなりません。

入力用にソース装置ファイルをオープンする場合には、システムは順序番号フィールドと日付フィールドを追加しますが、日付フィールドはゼロとなります。

装置ファイルを出力用にオープンし、さらに、そのファイルをソース・ファイルとして定義している場合、システムはデータを装置に書き出す前に順序番号と日付を削除します。

ソース・ファイル・データのコピー: ソース・ファイルのコピー (CPYSRCF) コマンド、およびファイルのコピー (CPYF) コマンドを使用して、複数のソース・ファイル・メンバー間でデータの書き込みを行うことができます。『ソース・ファイル間でコピーを行うソース・ファイルのコピー (CPYSRCF) コマンドの使用』 と 『ファイル間でコピーを行うファイルのコピー (CPYF) コマンドの使用』を参照してください。

データベース・ソース・ファイルから、それに関連した挿入トリガーを持つ別のデータベース・ソース・ファイルへコピーする場合は、コピーするレコードごとにトリガー・プログラムが呼び出されます。

コピーについては、このほかに 『コピーで使用されるソース順序番号』も参照してください。

ソース・ファイル間でコピーを行うソース・ファイルのコピー (CPYSRCF) コマンドの使用: ソース・ファイルのコピー (CPYSRCF) コマンドは、データベース・ソース・ファイルを操作します。機能においてはファイルのコピー (CPYF) コマンドと類似していますが、 CPYSRCF コマンドでは、通常、ソース・ファイルのコピー時に使用される省略時の値が提供されます。たとえば、このコマンドには、TOMBR パラメーターは FROMMBR パラメーターと同じであり、TOMBR レコードは常に置き換えを行う、と想定する省略時の値があります。TOFILE(*PRINT) を指定すると、CPYSRCF コマンドでは、固有の印刷形式もサポートされます。したがって、データベース・ソース・ファイルをコピーする場合には、 CPYSRCF コマンドを使用することになります。

CPYSRCF コマンドは自動的に取り出しファイル CCSID から受け入れファイルへ、データを変換します。

ファイル間でコピーを行うファイルのコピー (CPYF) コマンドの使用: ファイルのコピー (CPYF) コマンドには、 CPYSRCF コマンドの機能に加えて、以下のような機能があります。

- データベース・ソース・ファイルから装置ファイルへのコピー
- 装置ファイルからデータベース・ソース・ファイルへのコピー
- 非ソース・データベース・ファイルとソース・データベース・ファイル間のコピー
- 16 進数形式によるソース・メンバーの印刷
- 選択値を用いたソース仕様のコピー

コピーで使用されるソース順序番号: データベース・ソース・ファイルにコピーする場合は、SRCOPT パラメーターを使用して順序番号を更新し、日付をゼロに初期設定することができます。省略時の値では、システムは 1.00 という順序番号を最初のレコードに割り当て、残りのレコードに対して 1.00 ずつ順序番号を増やします。SRCSEQ パラメーターを使用して、小数部分の増分値を設定し、番号変更を開始するときの順序番号を指定できます。たとえば、SRCSEQ パラメーターに増分値が .10 で、順序番号が 100.00 から開始するよう指定した場合は、コピー済みのレコードには 100.00、100.10、100.20 という順序番号が付けられます。

.01 という開始値と、.01 という増分値を指定した場合、固有の順序番号が有効なレコードの最大数は 999,999 です。最大順序番号 (9999.99) に達すると、残りのレコードには 9999.99 という順序番号が付けられます。

同一ファイル内の 1 つのメンバーから別のメンバーへのソース・ステートメントのコピーの例を以下に示します。MBRB が存在しない場合には、追加されます。存在する場合には、すべてのレコードが置き換えられます。

```
CPYSRCF FROMFILE(QCLSRC) TOFILE(QCLSRC) FROMMBR(MBRA) +
        TOMBR(MBRB)
```

1 つのファイルから別のファイルへの総称メンバー名のコピーの例を以下に示します。PAY で始まるメンバーがすべてコピーされます。該当するメンバーが存在しない場合には、メンバーが追加されます。存在する場合には、すべてのレコードが置き換えられます。

```
CPYSRCF FROMFILE(LIB1/QCLSRC) TOFILE(LIB2/QCLSRC) +
        FROMMBR(PAY*)
```

メンバー PAY1 を印刷装置ファイル QSYSPRT (*PRINT の省略時の値) にコピーする例を以下に示します。ソース・ステートメントの印刷には、SEU が使用する様式に類似した様式が使用されます。

```
CPYSRCF FROMFILE(QCLSRC) TOFILE(*PRINT) FROMMBR(PAY1)
```

装置ソース・ファイルからデータベース・ソース・ファイルへコピーする場合は、順序番号が追加され、日付はゼロに初期設定されます。順序番号は 1.00 から始まり、1.00 ずつ増加します。コピーしたいファイルにあるレコードの数が 9999 より多い場合は、順序番号は 1.00 に戻り、SRCOPT パラメーターおよび SRCSEQ パラメーターを指定していない限り、増加し続けます。

データベース・ソース・ファイルから装置ソース・ファイルにコピーする場合、日付フィールドと順序番号フィールドは除去されます。

iSeries 以外のシステムからのデータのロードおよびアンロード: インポート・ファイルからのコピー (CPYFRMIMPF) コマンド、および インポート・ファイルへのコピー (CPYTOIMPF) コマンドを使用して、 iSeries との間でデータをインポート (ロード) またはエクスポート (アンロード) することができます。

iSeries 以外のデータベースから外部記述 DB2 UDB for iSeriesデータベース・ファイルにデータをインポートするには、以下のステップを実行してください。

1. コピーしたいデータのインポート・ファイルを作成する。1 つのフィールドを持つデータベース・ソース・ファイルまたは外部記述データベース・ファイルをインポート・ファイルにすることができます。フィールドのデータ・タイプは、CHARACTER、IGC OPEN、IGC EITHER、IGC ONLY、または UCS-2 でなければなりません。
2. インポート・ファイルに (またはインポート・ファイルから) データを送る。システムは、このプロセスの間に、必要な ASCII-EBCDIC 変換を実行します。以下のいくつかの方法でデータを送ることができます。
   - TCP/IP ファイル転送 (ファイル転送)
   - iSeries Access サポート (ファイル転送、ODBC)
   - テープ・ファイルからコピー (CPYFRMTAP) コマンド
3. データのコピー先となる外部記述 DB2 UDB for iSeries データベース・ファイルまたは DDM ファイルを作成する。

4. インポート・ファイルからコピー (CPYFRMIMPF) コマンドを使用して、インポート・ファイルから iSeries データベース・ファイルにデータをコピーする。システム上に DB2 UDB 対称マルチプロセス製品がインストールされている場合、システムは、ファイルを並列にコピーします。

iSeries データベース・データを別のシステムにエクスポートするには、インポート・ファイルへのコピー (CPYTOIMPF) コマンドを使用して、データベース・ファイルからインポート・ファイルにデータをコピーしてください。その後、データのエクスポート先になるシステムにデータを送ります。

プログラムでのソース・ファイルの使用: プログラムでソース・ファイルを処理できます。ソース・ファイルの外部記述を使用し、他のデータベース・ファイルの場合と同様に、そのファイルに対して入出力操作を実行できます。

ソース・ファイルは外部的に記述されたデータベース・ファイルです。プログラム内のソース・ファイルに名前を付けてコンパイルするような場合は、ソース・ファイル記述が自動的にプログラム印刷出力に入れられます。たとえば、ソース・ファイル QDDSSRC 中の FILEA というメンバーのレコードを読み取って更新するものとします。このファイルを処理するためにプログラムを作成すると、システムにはソース・ファイルから *SRCSEQ*、*SRCDAT*、および *SRCDTA* の各フィールドが入ります。

注: ファイル・フィールド記述の表示 (DSPFFD) コマンドを使用してファイルに定義されたフィールドを表示できます。このコマンドの詳細については、203 ページの『ファイル内のフィールドの記述の表示』を参照してください。

QDDSSRC ファイルの FILEA メンバーを処理するプログラムでは、以下のことができます。

- ファイル・メンバーのオープン (他のすべてのデータベース・ファイル・メンバーと同様)。
- ソース・ファイルからのレコードの読み取りおよび更新 (おそらく、実際のソース・データが保管されている *SRCDTA* フィールドを変更することになると考えられる)。
- ソース・ファイル・メンバーのクローズ (他のすべてのデータベース・ファイル・メンバーと同様)。

ソース・ファイルを使用したオブジェクトの作成

作成コマンドを使用すると、ソース・ファイルを使用してオブジェクトを作成できます。ソース・ファイルを使用してオブジェクトを作成する場合は、作成コマンドにソース・ファイルの名前を指定することができます。

たとえば、CL プログラムを作成するには、制御言語 (CL) プログラムの作成 (CRTCLPGM) コマンドを使用します。作成コマンドでは、SRCFILE パラメーターを用いて、ソース・ファイルが保管されている場所を指定します。

作成コマンドは、ユーザーが以下のことを実行する場合にソース・ファイル名およびメンバー名を指定しなくても済むような設計になっています。

1. 作成するオブジェクトのタイプに対して省略時のソース・ファイル名を使用する場合。(使用するコマンドの省略時のソース・ファイル名を見つけ出すには、17 ページの『IBM 提供のソース・ファイル』を参照してください。)
2. ソース・メンバーに、作成したいオブジェクトと同じ名前を付ける場合。

たとえば、コマンドの省略時の値を使用して CL プログラム PGMA を作成するには、以下のように入力します。

```
CRTCLPGM PGM(PGMA)
```

システムは PGMA のソース・ファイルが QCLSRC ソース・ファイル内の PGMA メンバーに入っているものと想定します。QCLSRC ファイルが入っているライブラリーは、ライブラリー・リストによって判別されます。

別の例として、以下のように物理ファイルの作成 (CRTPF) コマンドで、データベース・ソース・ファイル FRSOURCE を使用するファイル DSTREF を作成します。ソース・メンバーの名前は DSTREF です。SRCMBR パラメーターを指定していないので、システムは、メンバー名 DSTREF が、作成したいオブジェクトの名前と同じものであるとみなします。

```
CRTPF FILE (QGPL/DSTREF) SRCFILE(QGPL/FRSOURCE)
```

関連情報については、以下のトピックを参照してください。

- 『バッチ・ジョブのソース・ステートメントからのオブジェクトの作成』
- 235 ページの『オブジェクトの作成に使用されたソース・ファイル・メンバーの判別』

バッチ・ジョブのソース・ステートメントからのオブジェクトの作成: 作成コマンドがバッチ・ジョブに入っている場合には、インライン・データ・ファイルをコマンドのソース・ファイルとして使用できます。しかし、ソース・ファイルとして使用するインライン・データ・ファイルには、10,000 以上のレコードを入れてはなりません。インライン・データ・ファイルには名前を付けても、付けなくてもかまいません。名前を持つインライン・データ・ファイルは、//DATA コマンドに指定される独自のファイル名を持っています。インライン・データ・ファイルの詳細については、ファイル管理 を参照してください。

名前のないインライン・データ・ファイルは独自のファイル名を持たないファイルであり、すべてが QINLINE という名前になります。ソース・ファイルとして使用されるインライン・データ・ファイルの例を以下に示します。

```
//BCHJOB
CRTPF FILE(DSTPRODLB/ORD199) SRCFILE(QINLINE)
//DATA FILETYPE(*SRC)
  .
  .   (source statements)
  .
//
//ENDBCHJOB
```

この例では、//DATA コマンドにファイル名が指定されていません。スプール読み取りプログラムによってジョブが処理されたときに、名前のないスプール・ファイルが作成されます。名前のないファイルにアクセスするために、CRTPF コマンドにはソース・ファイル名として QINLINE を指定しなければなりません。//DATA コマンドも、インライン・ファイルがソース・ファイルであることを指定します (FILETYPE パラメーターに *SRC を指定します)。

//DATA コマンドにファイル名を指定する場合は、CRTPF コマンドの SRCFILE パラメーターにも、同じ名前を指定する必要があります。たとえば、次の場合があります。

```
//BCHJOB
CRTPF FILE(DSTPRODLB/ORD199) SRCFILE(ORD199)
//DATA FILE(ORD199) FILETYPE(*SRC)
  .
  .   (source statements)
  .
//
//ENDBCHJOB
```

プログラムでインライン・ファイルを使用する場合、システムは指定された名前の最初のインライン・ファイルを検索します。そのファイルが見つからなければ、プログラムは名前のない最初のファイル (QINLINE) を使用します。

作成コマンドにソース・ファイル名を指定しない場合は、 IBM 提供のソース・ファイルに必要なソース・データが入っているものとみなされます。たとえば、ソース・ファイル名を指定しないで CL プログラムを作成すると、 IBM 提供のソース・ファイル QCLSRC が使用されます。ソース・データは QCLSRC に入れておかなければなりません。

ソース・ファイルがデータベース・ファイルの場合は、必要なソース・データを入れるソース・メンバーを指定できます。ソース・メンバーを指定しない場合、ソース・データは、作成するオブジェクトと同じ名前のメンバーに入れる必要があります。

オブジェクトの作成に使用されたソース・ファイル・メンバーの判別: ソース・メンバーからオブジェクトを作成する場合は、ソース・ファイル、ライブラリー、およびメンバーに関する情報がオブジェクトの中に保持されます。ソース・メンバーがオブジェクトの作成前に最後に変更された日付/時刻も、オブジェクトの中に保管されます。

オブジェクトの中の情報は、 DETAIL(*SERVICE) を指定してオブジェクト記述の表示 (DSPOBJD) コマンドを使用することにより表示できます。

この情報は、使用されたソース・メンバーを判別し、また、オブジェクトの作成後に既存のソース・メンバーが変更されたかどうかを判別する上で役立ちます。

オブジェクトの作成に使用されたソース・メンバーが、ソース・メンバーの中に現在入っているソース・メンバーと同じものであることを、以下のコマンドを使用して確認できます。

- TYPE(*MBR) を用いた ファイル記述の表示 (DSPFD) コマンド。この画面には、ソース・メンバーの日付/時刻の両方が表示されます。**最終ソース更新日付/時刻**の値を使用して、DSPOBJD コマンドによって表示された**ソース・ファイル日付/時刻**の値と比較します。
- DETAIL(*SERVICE) を使用したオブジェクト記述の表示 (DSPOBJD) コマンド。この画面には、オブジェクトの作成に使用されたソース・メンバーの日付/時刻が表示されます。

注: 出力ファイルに書き出されたデータを使用してソース日付とオブジェクト日付が同じであるかどうかを判別する場合は、DSPOBJD DETAIL(*SERVICE) コマンドの出力ファイルからの *ODSRCD* (ソース日付) フィールドおよび *ODSRCT* (ソース時刻) フィールドを、 DSPFD TYPE(*MBR) コマンドの出力ファイルからの *MBUPDD* (メンバー更新日付) フィールドおよび *MBUPDT* (メンバー更新時刻) フィールドと比較できます。

ソース・ファイルの管理

この項では、ソース・ファイルの管理に関する考慮事項を示します。

- 『ソース・ファイル属性の変更』
- 236 ページの『ソース・ファイル・メンバーのデータの再編成』
- 236 ページの『ソース・ステートメントの変更時刻の判別』
- 237 ページの『文書化のためのソース・ファイルの使用』

ソース・ファイル属性の変更: データベース・ソース・ファイルを保守するために SEU を使用している場合には、データベース・ソース・ファイルの変更方法について、 V5R1 補足資料 Web サイトの「適用業務開発ツールセット AS/400 用 原始ステートメント入力ユーティリティー」を参照してください。データベース・ソース・ファイルの管理に SEU を使用しない場合には、既存のメンバー全体を置換しなければなりません。

ソース・ファイルがディスケット上にある場合は、このファイルをデータベース・ファイルにコピーし、SEU を使用して変更し、ディスケットにコピーし直してください。SEU を使用しない場合は、古いソース・ファイルを削除してから、新しいソース・ファイルを作成する必要があります。

ソース・ファイルを変更する場合、以前にソース・ファイルから作成したオブジェクトは、現行のソース・ファイルと一致しません。古いオブジェクトを削除し、変更済みのソース・ファイルを使用してオブジェクトを作成し直してください。たとえば、233 ページの『ソース・ファイルを使用したオブジェクトの作成』で作成したソース・ファイル FRSOURCE を変更する場合は、元のソース・ファイルから作成したファイル DSTREF を削除し、DSTREF が変更済みの FRSOURCE ソース・ファイルと一致するよう新しいソース・ファイルを使用して DSTREF を再び作成する必要があります。

ソース・ファイル・メンバーのデータの再編成: 到着順ソース・ファイルを使用している場合、通常は、ソース・ファイル・メンバーを再編成する必要はありません。

すべてのレコードに固有の順序番号を割り当てるには、 物理ファイル・メンバーの再編成 (RGZPFM) コマンドに以下のパラメーターを指定します。

- KEYFILE(*NONE)。レコードが再編成されないようにするため。
- SRCOPT(*SEQNBR)。順序番号を変更するため。
- SRCSEQ。すべての順序番号が独自のものであるように、.10 または .01 のような小数部付きとするため。

注: 削除済みレコードがある場合は、圧縮されます。

物理ファイルに対して、キー順アクセス・パスが指定されている論理ファイルが作成される場合、到着順アクセス・パスを持つソース・ファイルは、順序番号によって再編成されます。

ソース・ステートメントの変更時刻の判別: 各ソース・レコードは、ステートメントに変更が発生した場合に SEU によって自動的に更新される日付フィールドを含んでいます。これを使用して、ステートメントが最後に変更された時点を判別できます。大多数の高水準言語コンパイラーは、これらの日付をコンパイル・リストに印刷します。ファイルのコピー (CPYF) コマンドおよびソース・ファイルのコピー (CPYSRCF) コマンドも、これらの日付を印刷します。

各ソース・メンバー記述は、2 つの日付および時刻フィールドを含んでいます。最初の日付/時刻フィールドは、更新後にクローズされるときには必ず、メンバーに対する変更を反映します。

2 番目の日付/時刻フィールドには、メンバーの変更が反映されます。これには、SEU、コマンド (CRYF や CPYSRCF など)、許可変更、およびファイル状況の変更によって生じたすべての変更が含まれています。たとえば、物理ファイルの変更 (CHGPF) コマンドの FRCRATIO パラメーターにより、メンバー状況が変更されます。この日付/時刻フィールドは、変更オブジェクトの保管 (SAVCHGOBJ) コマンドによって、メンバーを保管すべきかどうかを判別するために使用されます。日付/時刻の両フィールドは、ファイル記述の表示 (DSPFD) コマンドで TYPE(*MBR) を指定して表示できます。ソース・メンバーと共に示される変更された日付/時刻は、2 つあります。

- 最終ソース更新日付/時刻。この値には、メンバー内のソース・データ・レコードへの変更が反映されます。ソース・ステートメントに更新が発生すると、 最終変更日付/時刻の値に 1 または 2 秒の差しかない場合でも、この値は更新されます。
- 最終変更日付/時刻。この値には、メンバーへの変更が反映されます。これには、SEU、コマンド (CPYF や CPYSRCF など)、許可変更、またはファイル状況の変更によって生じるすべての変更が含まれます。たとえば、CHGPF コマンドの FRCRATIO パラメーターによりメンバー状況が変更され、そのため、最終変更日付/時刻の値に反映されます。

文書化のためのソース・ファイルの使用: IBM 提供のソース・ファイル QTXTSRC を使用して、オンライン文書の作成および更新に役立てることができます。

QTXTSRC メンバーは、SEU で使用可能なこれ以外のアプリケーション・プログラム (QRPGSRC または QCLSRC など) と同様に作成および更新できます。QTXTSRC ファイルは叙述文書にはもっとも便利であり、オンラインで検索したり、印刷することができます。ソース・メンバーに入れたテキストは、SEU の追加、変更、移動、コピー、および組み込み操作を使用することにより、簡単に更新することができます。終了プロンプトの現行ソース・ファイルの印刷オプションに Yes と指定することにより、メンバー全体を印刷できます。ソース・メンバー全体または一部を印刷するプログラムを作成することもできます。

制約によってユーザーのデータベースの保全性を制御する

制約とは、レコードを追加、変更、除去してもデータベース内のデータの一貫性を保つようにするために、ファイルに付けられる制限または限度のことです。

- 固有制約および基本キー制約を使用すれば、ファイル・アクセス・パスに制約されずに、拡張固有キーを物理ファイルに対して作成することができます。 241 ページの『固有制約』 と 242 ページの『基本キー制約』を参照してください。
- 検査制約により、式の中のデータをテストして、ユーザーのデータの妥当性を検査する機能をもう 1 つ追加できます。 242 ページの『検査制約』を参照してください。

基本キーおよび固有制約は、参照制約を追加するさいに親キーとして使用できます。

制約を使用するには、以下のトピックを参照してください。

- 『データベースに制約を設定する』
- 238 ページの『固有制約、基本キー制約、または検査制約の除去』
- 239 ページの『制約グループの処理』
- 240 ページの『検査保留状況になっている制約の処理』

データベースに制約を設定する

物理ファイル制約を使用して、データベース内に保持されているデータの保全性を制御できます。

物理ファイル制約を追加するには、 物理ファイル制約の追加 (ADDPFCST) コマンドを使用します。

- 固有制約を追加するには、TYPE パラメーターに *UNQCST 値を指定します。KEY パラメーターに 1 つまたは複数のフィールド名を指定しなければなりません。
- 基本キー制約を追加するには、TYPE パラメーターに *PRIKEY 値を指定します。コマンドに指定したキーは、ファイルの基本アクセス・パスになります。ファイルに共有可能なキー順アクセス・パスがない場合は、システムが作成します。KEY パラメーターに 1 つまたは複数のフィールド名を指定しなければなりません。
- 検査制約を追加するには、TYPE パラメーターに *CHKCST 値を指定します。また、CHKCST パラメーターに検査制約式を指定しなければなりません。検査制約の式は、構造化照会言語 (SQL) を使用して定義された検査条件に使用される式と同じ構文になります。SQL を使用した制約の設定に関する詳細については、DB2 UDB for iSeries SQL 解説書を参照してください。

iSeries ナビゲーターを使用して、制約を追加することもできます。 SQL プログラミングの中の、以下のトピックを参照してください。

- iSeries ナビゲーターを使用したキー制約の追加
- iSeries ナビゲーターを使用した検査制約の追加

さらに、SQL の CREATE TABLE および ALTER TABLE ステートメントを使用するときにも、制約を追加できます。

制約の設定に関する追加情報については、『詳細: 制約の設定』を参照してください。

詳細: 制約の設定: すべての物理ファイル制約に以下の規則が適用されます。

- ファイルは、物理ファイルでなければなりません。
- ファイルは、最大 1 つのメンバー MAXMBR(1) を持つことができます。
- 制約は、ファイルのメンバーがゼロのときに定義できます。ファイルに 1 つ (複数は不可) のメンバーが入っていないと、制約は確定できません。
- 1 つのファイルは、基本キー制約は最大 1 つのみ持つことができますが、多くの固有制約を持つことができます。
- 1 ファイルあたりの最大制約関係は 300 です。この最大値は、次のものの総和です。
  - 固有制約
  - 基本キー制約
  - 検査制約
  - 参照制約 (親または従属のどちらで参与しているか、また制約が定義済みまたは確定済みのどちらかに関係なく)
- 制約名は、ライブラリーで固有名でなければなりません。
- 制約は、QTEMP ライブラリー内のファイルに追加することはできません。
- 参照制約は、親および従属ファイルを同じ補助記憶域プール (ASP) に持っていなければなりません。

固有制約、基本キー制約、または検査制約の除去

物理ファイル制約を除去するには、 物理ファイル制約の除去 (RMVPFCST) コマンドを使用します。コマンドのすべての効果は、除去する制約のタイプと使用方法に依存します。

- 固有制約を除去するには、TYPE パラメーターに *UNQCST 値を指定します。
- 基本キー制約を除去するには、TYPE パラメーターに *PRIKEY 値を指定します。
- 検査制約を除去するには、TYPE パラメーターに *CHKCST 値を指定します。

リストされているそれぞれの制約タイプに対して、制約 (CST) パラメーターに以下の値のいずれかを指定できます。

- CST(*ALL) を指定すると、TYPE パラメーターに指定したすべての制約が除去されます。
- CST(制約名) を指定すると、特定の制約が除去されます。
- CST(*CHKPND) を指定すると、検査保留状況になっている制約だけが除去されます。
- CST(*ALL) と TYPE(*ALL) を指定すると、すべての制約がファイルから除去されます。

以下のことを行って制約を除去することもできます。

- 構造化照会言語 (SQL) を使用して制約を除去する。DB2 UDB for iSeries SQL 解説書を参照してください。
- iSeries ナビゲーターを使用して制約を除去する。詳しくは、 SQL プログラミングの中の、 iSeries ナビゲーターを初めて使用される場合を参照してください。

制約の除去に関する追加情報については、 239 ページの『詳細: 制約の除去』を参照してください。

詳細: 制約の除去: 基本キー制約または固有制約を除去した場合に、それに関連したアクセス・パスを論理ファイルが共用していれば、共用パスの所有権はその論理ファイルに移ります。アクセス・パスを共用していない場合、アクセス・パスは除去されます。

RMVPFCST コマンドを使用して基本キー制約を除去すると、そのキー仕様をファイルから除去するかどうかを決定する照会メッセージが、システムから送信されます。「K」と答えると、ファイルのキー仕様は保持されます。ファイルはキー付きのままとなります。「G」と答えると、コマンドが完了するときに、ファイルは到着順アクセス・パスとなります。

注: 構造化照会言語 (SQL) ALTER TABLE ステートメントを使用して基本キー制約を除去すると、照会メッセージは送信されません。キー仕様は常に除去され、 ALTER TABLE が完了するときに、ファイルは到着順アクセス・パスとなります。

制約グループの処理

特定のファイルに存在する制約のリストを表示するには、 物理ファイル制約の処理 (WRKPFCST) コマンドを使用します。この画面から、制約を変更または除去することができます。また、ファイル制約が検査保留状況になる原因となったレコードのリストも表示できます。

制約グループの処理に関する詳細については、『詳細: 制約グループの処理』を参照してください。

詳細: 制約グループの処理:

```
                          物理ファイルの制約の処理

オプションを入力して，実行キーを押してください。
   2= 変更    4= 除去    6= 検査保留中レコードの表示


 Opt  制約          ファイル    ライブラリー タイプ    状態     保留中検査
  _   DEPTCST       EMPDIV7     EPPROD       *REFCST   EST/ENB  No
  _   ACCTCST       EMPDIV7     EPPROD       *REFCST   EST/ENB  Yes
  _   STAT84        EMPDIV7     EPPROD       *REFCST   DEF/ENB  No
  _   FENSTER       REVSCHED    EPPROD       *REFCST   EST/DSB  Yes
  _   IRSSTAT3      REVSCHED    EPPROD       *UNQCST
  _   IFRNUMBERO >  REVSCHED    EPPROD       *UNQCST
  _   EVALDATE      QUOTOSCHEM  EPPROD       *REFCST   EST/ENB  No
  _   STKOPT        CANSCRONN9  EPPROD       *PRIKEY
  _   CHKDEPT       EMPDIV2     EPPROD       *CHKCST   EST/ENB  No

                                                                 続く ...
 オプション 2，4，6 の場合のパラメーターまたはコマンド
 ===>
 F3= 終了   F4= プロンプト   F5= 最新表示   F9= コマンドの複写   F12= 取消し
 F15= 分類   F16= 位置指定の繰返し   F17= 位置指定   F22= 制約名の表示
```

物理ファイル制約の処理画面は、WRKPFCST コマンドで指定されたファイルに対して定義されたすべての制約を示しています。画面には、制約名、ファイル名、およびライブラリー名がリストされます。さらに、以下の情報が表示されます。

- タイプ欄は、制約が参照、検査、固有、または基本キーのどれであるかを識別します。
- 状態欄は、制約が定義済みまたは確定済みかどうか、および使用可能または使用不可能のどちらであるかを示します。状態欄は、参照制約および検査制約にのみ適用されます。

- 検査保留欄には、制約の検査保留状況が含まれています。固有制約および基本キー制約は、常に確定済みであり使用可能であるため、状態は表示されません。

リストされた制約のそれぞれに対して、以下のアクションを実行することができます。

- 参照制約または検査制約を変更して、使用可能状態にするには、変更 (オプション 2) を選択します。たとえば、現在使用不可能な制約を使用可能にすることができます。このオプションは、CHGPFCST コマンドと同じ働きをします。
- 制約を除去するには、除去 (オプション 4) を選択します。このオプションは、RMVPFCST コマンドと同じ働きをします。
- 検査保留状態になっているレコードを表示するには、表示 (オプション 6) を選択します。このオプションは、 DSPCPCST コマンドと同じ働きをします。DSPCPCST コマンドは、参照制約および検査制約にのみ適用されます。

検査保留状況になっている制約の処理: 参照制約または検査制約を追加すると、システムは自動的にデータベース・ファイル内のすべてのレコードを検査して制約定義を満たしていることを確認します。この検査は、システムが復元されたときにも実行されます。

制約が有効でない場合、または確認できない場合は、システムはそれを*検査保留* 状況に入れます。

検査保留状況にある制約を処理するには、以下の手順を実行してください。

1. 制約を使用不可能にします。物理ファイル制約の変更 (CHGPFCST) コマンドを実行して、制約状態パラメーターに *DISABLED を指定します。
2. 制約が検査保留としてマークされた原因となったレコードのリストを表示します。検査保留制約の表示 (DSPCPCST) コマンドを実行します。詳細については、 『制約を検査保留状況にしているレコードの表示』を参照してください。

   注: このコマンドの実行時間は、ファイルに含まれるレコードの数に依存します。
3. 検査保留状況になっている制約の確認をスケジュールします。検査保留制約の編集 (EDTCPCST) コマンドを実行します。詳細については、『検査保留状況になっている制約の処理』を参照してください。
4. 制約を使用可能にします。もう一度 CHGPFCST コマンドを実行して、制約状態パラメーターに *ENABLED を指定します。

制約を検査保留状況にしているレコードの表示: 制約が追加されると、システムはファイル内のレコードが制約の規則に従っているかどうかを確認します。レコードが有効でない場合は、システムは制約を検査保留状況に入れます。

制約の規則に従っていないレコードを検査することは有用です。レコードまたは制約のいずれかを必要に応じて変更することができます。

注: 以下のステップを実行する前に、 物理ファイル制約の変更 (CHGPFCST) コマンドを実行して制約を使用不可にしてください。

制約が検査保留状況になった原因であるレコードのリストを表示または印刷するには、 検査保留制約の表示 (DSPCPCST) コマンドを実行します。

検査保留状況になっている制約の処理: 大きなデータベース・ファイルに作成された制約の妥当性検査には、かなりの時間がかかることがあります。検査保留になっている制約をリストにして必要に応じて、検査にスケジュールすることができます。

検査保留状況にある制約を表示し、編集するには、以下の手順を実行してください。

1. 検査保留制約の編集 (EDTCPCST) コマンドを実行します。
2. 処理したい制約の状況をチェックします。
3. 制約の状況が RUN または READY 以外の場合は、SEQ フィールドの *HLD 値を 1 から 99 の範囲の値に変更します。
4. Enter を押します。

```
                              保留中制約の編集検査

順序を入力して、実行キーを押してください。
順序 :   1-99, *HLD

                  ---------------- 制約 ---------------- 検査       経過
 Seq  状況        CST          ファイル     ライブラリー  時間       時間
 1    RUN         EMP1         DEP          EPPROD        00:01:00   00:00:50
 1    READY       CONST    >   DEP          EPPROD        00:02:00   00:00:00
 *HLD CHKPND      FORTH    >   STYBAK       EPPROD        00:03:00   00:00:00
 *HLD CHKPND      CST88        STYBAK       EPPROD        00:10:00   00:00:00
 *HLD CHKPND      CS317        STYBAK       EPPROD        00:20:00   00:00:00
 *HLD CHKPND      KSTAN        STYBAK       EPPROD        02:30:00   00:00:00



                                                                        終わり
F3= 終了   F5= 最新表示   F11= タイプの表示   F12= 取消し   F13= すべての反復
F15= 分類   F16= 位置指定の繰返し   F17= 位置指定   F22= 制約名の表示
```

検査保留状況になっている制約の処理に関する詳細については、 『詳細: 検査保留状況になっている制約の処理』を参照してください。

詳細: 検査保留状況になっている制約の処理: 「検査保留制約の編集」画面の状況フィールドは、以下のいずれかの値になります。

- **RUN** は、制約が検査中であることを示します。
- **READY** は制約が検査できることを示します。
- **NOTVLD** は、制約に関連したアクセス・パスが無効であることを示します。アクセス・パスが再作成されると、システムは自動的に制約を検査します。この値は、参照制約にのみ適用します。
- **HELD** は、制約が検査されないことを示します。この状態を変更するには、順序の値を 1 から 99 の範囲の値に変更しなければなりません。
- **CHKPND** は、制約の検査が試行されたが、その制約がまだ検査保留になっていることを示します。この状態を変更するには、順序の値を 1 から 99 の範囲の値に変更しなければなりません。

制約欄には、制約名の最初の 5 文字が入ります。> 記号は、名前が 5 文字を超えた場合、名前の後に付きます。その行にカーソルを置いて、F22 キーを押すと、名前全体を表示できます。

検査時間欄は、システム上に他のジョブがない場合に、制約の検査にかかる時間を示します。経過時間欄は、制約を検査するのにすでに費やした時間を示します。

固有制約

固有制約は、データベース内でコントロールとして動作して行が固有であることを保証します。たとえば、顧客識別番号をデータベース内で固有制約として指定することができます。同じ顧客番号を使用して新規顧客を作成しようとした場合、エラー・メッセージがデータベース管理者に送られます。

固有制約は、値がファイル内のレコード間で固有でなければならないデータベース・ファイル内の 1 つまたは複数のフィールドを識別します。フィールドは、昇順で、NULL 可能になっていなければなりません。

1 つのファイルに複数の固有制約があってもかまいませんが、重複した固有制約があってはなりません。順序に関係なく、同一のキー・フィールドは重複制約になります。

固有制約は、参照制約を追加するさいに、親キーとして使用できます。

基本キー制約

基本キー制約は、キーをファイルへの基本アクセス・パスにする特殊属性を持つ固有キーです。

基本キー制約は、値がファイル内のレコード間で固有でなければならないデータベース・ファイル内の 1 つまたは複数のフィールドを識別します。フィールドは、昇順で、NULL 可能になっていなければなりません。NULL を使用できる場合は、検査制約が暗黙的に追加され、これらのフィールドに NULL を入力できないようにします。1 つのファイルに対して定義できる基本キー制約は 1 つだけです。

基本キー制約は、参照制約を追加する際に親キーとして使用できます。

検査制約

検査制約は、フィールドの値の制限を保持して、データベース要件に従うようにするために使用します。

検査制約は、データの挿入時や更新時に、ユーザー定義の検査制約式と対照させてそのデータを検査し、妥当性を確認します。

たとえば、あるフィールドに挿入される値の範囲が 1 から 100 の間でなければならないという検査制約を作成できます。値がこの範囲内にない場合は、データベースに対する挿入操作や更新操作は処理されません。

検査制約の状態は、以下の点で参照制約の状態に非常に類似しています。

- 定義済みかつ使用可能 - 制約定義がファイルに追加されていて、制約が確立された後でその制約が実施される。
- 定義済みかつ使用不可能 - 制約定義がファイルに追加されているが、その制約は実施されない。
- 確定済みかつ使用可能 - 制約がファイルに追加されていて、そのファイルに制約実施に関係するすべての部分がある。
- 確定済みかつ使用不可能 - 制約がファイルに追加されていて、そのファイルに制約実施に関係するすべての部分があるが、制約は実施されない。

検査制約は、参照制約と同様に、検査保留状況になります。いずれかのフィールド内のデータが検査制約式に違反している場合、その制約は検査保留になります。レコードの挿入または更新をするときに、データが検査制約式に違反している場合には、挿入または更新は処理されません。

1 つまたは複数のラージ・オブジェクト (LOB) フィールドを含む検査制約は、LOB フィールドを含まない検査制約よりも狭い範囲の操作に限定されます。検査制約に 1 つまたは複数の LOB フィールドを含む場合、LOB フィールドは以下の項目との直接比較にのみ関係します。

- 同じタイプおよび最大値の他の LOB フィールド
- リテラル値
- ヌル値

Substring 操作または Concat 操作などの派生操作として知られる操作は、 検査制約では LOB フィールドに対して許可されていません。LOB フィールドに対して派生操作を行う検査制約を追加しようとすると、診断メッセージ CPD32E6 が送信されます。

参照制約を使用したデータの保全性の保証

iSeries データベースで*参照制約* を使用して、ユーザーのシステムの参照保全を強化できます。*参照保全*は、データベースが有効なデータのみを含むようにするためにユーザーが使用するすべてのメカニズムおよび技術を含みます。

参照制約を使用するには、以下のトピックを参照してください。
1. 『参照制約の追加』
2. 247 ページの『参照制約の確認』
3. 247 ページの『参照制約を使用可能または使用不可にする』
4. 248 ページの『参照制約の除去』

また、以下のトピックでも参照保全に関する重要な情報を提供しています。
- 249 ページの『詳細: 参照制約を使用したデータ保全性の保証』
- 250 ページの『例: 参照制約を使用したデータ保全性の保証』
- 250 ページの『参照保全の用語』
- 251 ページの『参照保全の実行』
- 252 ページの『制約状態』
- 252 ページの『参照制約内の検査保留状況の検査』
- 253 ページの『参照保全と iSeries 機能』

参照制約の追加

最大 1 つのメンバーを持つ物理ファイルに参照制約を追加することができます。参照制約は、ファイル・レベル属性です。したがって、メンバーの存在よりも前に制約を作成できます。

参照制約を追加するには、以下のトピックを参照してください。
1. 『参照制約を追加する前に』
2. 244 ページの『参照制約内に親ファイルを定義する』
3. 245 ページの『参照制約内に従属ファイルを定義する』
4. 245 ページの『参照制約規則を指定する』

参照制約の追加に関する詳細については、以下のトピックを参照してください。
246 ページの『詳細: 参照制約の追加』
247 ページの『詳細: 制約サイクルの回避』

参照制約を追加する前に: 参照制約を追加する前に、以下の条件に合致していることを確認しなければなりません。
- 親キーになり得るキーを持つ親ファイルがなければならない。親ファイルが基本キーまたは固有制約を持っていないと、潜在親キーのフィールド属性が従属ファイルの外部キー・フィールド属性に合致する場合に、システムは基本キーまたは固有制約を親ファイルに追加しようとします。
- 親ファイルの属性に合致する特定の属性を持つ従属ファイルがなければならない。
  - 分類順序 (SRTSEQ) は、データ・タイプの CHAR、OPEN、EITHER、および HEX に関して一致しなければならない。
  - コード化文字セット識別コード CCSID の一方または両方が 65535 でない限り、 CCSID は各 SRTSEQ 表に関して一致しなければならない。

– 各分類順序表は、正確に一致しなければならない。
- 従属ファイルには、親キーの次のような属性に一致する外部キーが入っていなければならない。
  – データ・タイプ
  – 長さ
  – 精度 (パック、ゾーンまたは 2 進)
  – CCSID (どちらも 65535 の CCSID でない限り)
  – REFSHIFT (データ・タイプが OPEN、EITHER、または ONLY の場合)

参照制約内に親ファイルを定義する: 親ファイルは、最大 1 つのメンバーを持つ物理ファイルでなければなりません。親ファイルを定義するときに、新しいファイルを作成するかまたは既存のファイルを使用することができます。

親キーの概念は、参照制約に関してのみ適用します。参照制約が従属ファイルに追加されるさいは、親ファイルに親キーが必要です。このためには、最初に基本キー制約または固有制約のいずれかをキー用の適切なフィールド・セットを持つ親ファイルに追加しなければなりません。参照制約が追加されると、一致する固有制約 (および基本キー) の検索が実行されます。一致したものが検出されると、制約のアクセス・パスが参照制約関係内で親キーとして使用されます。

新しい物理ファイルを親ファイルとして作成するには、以下の手順を実行してください。
1. 物理ファイルの作成 (CRTPF) コマンドを使用してファイルを作成します。
2. 物理ファイル制約の追加 (ADDPFCST) コマンドを使用して、基本キー制約または固有制約のいずれかを追加します。基本キーは、NULL 可能ですが、システムは明示的な検査制約を作成してフィールドへの NULL 値の挿入を防ぎます。

注: SQL CREATE TABLE ステートメントを使用して上記の手順を 1 つの手順で実行することができます。

既存ファイルを親ファイルとして使用するには、以下のオプションから方法を選択してください。

- 物理ファイル制約の追加 (ADDPFCST) コマンドを使用して基本キー制約を追加できます。TYPE パラメーターに *PRIKEY を指定します。また、KEY パラメーターで、1 つまたは複数のキー・フィールドを指定しなければなりません。

  そのファイルの基本キー制約がすでに存在する場合は、ファイルは 1 つの基本キーしか持つことはできないため、TYPE(*PRIKEY) を指定した ADDPFCST コマンドは失敗します。別の基本キー制約を追加する場合は、まず、物理ファイル制約の除去 (RMVPFCST) コマンドを使用して、既存の基本キー制約を除去しなければなりません。除去した後で、新しい基本キー制約を追加できます。
- 物理ファイル制約の追加 (ADDPFCST) コマンドを使用してファイルに固有制約を追加できます。TYPE パラメーターに *UNQCST を指定します。また、KEY パラメーターで、1 つまたは複数のキー・フィールドを指定しなければなりません。また、構造化照会言語 (SQL) ALTER TABLE ステートメントを使用して固有制約を追加することもできます。

  親ファイルに親キーとして使用できる基本キーまたは固有制約がない場合、参照制約を追加するときにシステムは自動的に基本キー制約を追加します。

  アクセス・パス・フィールドが外部キーのフィールドと一致する (フィールドおよび一致する属性の数の両方) 親ファイルが固有キー・アクセス・パスを持っている場合、基本キー制約が明示的に親ファイルに追加されます。これは、参照制約の親キーになります。

親ファイルが到着順アクセス・パスで、親キーに指定されたフィールドが外部キーのフィールドと一致する (属性の一致) 場合、基本キー制約が明示的に親ファイルに追加されます。これは、参照制約の親キーになります。

親キーを追加できない場合には、 『親キーを定義できない場合』を参照してください。

親キーを定義できない場合: 基本キー制約または固有制約を持つ既存のファイルにおいて、どちらの制約も親キーとして十分でなければ、以下の方法があります。

- ファイルを削除し、適切なキーを指定して作成し直す。
- 作成したファイルに、固有キー制約または基本キー制約を追加する。

参照制約内に従属ファイルを定義する: 従属ファイルは、最大 1 つのメンバーを持つ物理ファイルでなければなりません。

従属ファイルを作成するには、物理ファイルと同様に作成するかまたは既存ファイルを使用します。

実際に制約を作成するときに、従属ファイルがキー順アクセス・パスを持っている必要はありません。既存のアクセス・パスが外部キー基準を満たさない場合、システムがアクセス・パスをそのファイルに追加します。

参照制約規則を指定する: 参照制約を使用すると、レコードの削除または更新を行うときにシステムが実行すべき規則を指定することができます。

参照制約に実行させる規則を指定するには、以下の手順を実行してください。

1. 物理ファイル制約の追加 (ADDPFCST) コマンドを実行します。
2. DLTRULE パラメーターの値を選択して、レコードを削除するときに実行させる規則 (*削除規則*) を指定します。
3. UPDRULE パラメーターの値を選択して、レコードを更新するときに実行させる規則 (*更新規則*) を指定します。

iSeries ナビゲーターを使用して、参照制約を追加することもできます。詳しくは、 SQL プログラミングの中の、 iSeries ナビゲーターを初めて使用される場合を参照してください。

参照制約に関する詳細については、以下のトピックを参照してください。

『詳細: 参照制約削除規則を指定する』
246 ページの『詳細: 参照制約更新規則を指定する』
246 ページの『詳細: 参照制約規則を指定する—ジャーナル処理要件』

詳細: 参照制約削除規則を指定する: DLTRULE パラメーターに指定できる値は 5 つあります。削除規則は、親キー値を削除したときにシステムが実行する処置を指定します。削除規則は、NULL 親キー値には影響しません。

- *NOACTION (省略時値)
  - 親キー値が、一致する外部キー値を持っている場合には、親ファイル内のレコード削除は行われません。
- *CASCADE
  - 親キー値が外部キー値に一致する場合、親ファイル内のレコードが削除されることにより、従属ファイル内のレコードが削除されます。
- *SETNULL

– 非 NULL の親キー値が外部キー値に一致する場合、親ファイルのレコードが削除されることで、従属ファイルのレコードが更新されます。上記の基準を満たす従属レコードの場合、外部キー内のすべての NULL 可能フィールドは NULL に設定されます。非 NULL 属性をもつ外部キー・フィールドは更新されません。

- *SETDFT
  – 非 NULL の親キー値が外部キー値に一致する場合、親ファイルのレコードが削除されることで、従属ファイルのレコードが更新されます。上記の基準を満たす従属レコードの場合、その 1 つまたは複数の外部キー・フィールドは、それぞれに応じた省略時値に設定されます。
- *RESTRICT
  – 親キー値が、一致する外部キー値を持っている場合には、親ファイル内のレコード削除は行われません。

  注: 削除が試みられると、システムはただちに削除の *RESTRICT 規則を適用します。システムは、操作の論理終了時に他の制約を実施します。他の制約の場合の操作には、削除の前または後に実行されるトリガー・プログラムが含まれます。トリガー・プログラムは、参照保全の違反が発生した場合に訂正できます。たとえば、トリガー・プログラムは、親レコードがない場合に親レコードを追加することがあります。*RESTRICT 規則では、違反を防止できません。

詳細: 参照制約更新規則を指定する: UPDRULE パラメーターに指定できる値は 2 つあります。UPDRULE パラメーターは、親ファイルと従属ファイルの制約関係の更新規則を識別します。更新規則は、システムが親ファイルの更新を試みたときにとる処置を指定します。

- *NOACTION (省略時値)
  – 従属ファイル内に一致する外部キー値がある場合には、親ファイル内のレコード更新は行われません。
- *RESTRICT
  – 外部キー値に一致する非 NULL の親キー値がある場合には、親ファイル内のレコード更新は行われません。

  注: 更新が試みられると、ただちにシステムは更新の *RESTRICT 規則を実施します。システムは、操作の論理終了時に他の制約を実施します。たとえば、トリガー・プログラムは、親レコードがない場合に親レコードを追加することがあります。*RESTRICT 規則では、違反を防止できません。

詳細: 参照制約規則を指定する—ジャーナル処理要件: 参照制約に関連するファイルに対して挿入、更新、または削除を実行する場合に、削除規則、更新規則、あるいはその両方が *RESTRICT 以外であるときには、ジャーナル処理を使用しなければなりません。親ファイルと従属ファイルの両方とも、同一のジャーナルにジャーナルしなければなりません。さらに、 物理ファイルのジャーナル処理開始 (STRJRNPF) コマンドを使用して、親ファイルと従属ファイルのジャーナル処理を開始する必要があります。

*RESTRICT 以外の削除規則、更新規則、またはこの両方に従う参照制約に関連したファイルに対して、レコードの挿入、更新、または削除を行う場合には、コミットメント制御が必要です。コミットメント制御を開始していない場合、システムが自動的に開始して終了します。

詳細: 参照制約の追加: 参照制約には、以下の制限が適用されます。

- 親ファイルは、物理ファイルでなければなりません。
- 親ファイルは、最大 1 つのメンバー MAXMBR(1) を持つことができます。
- 従属ファイルは、物理ファイルでなければなりません。
- 従属ファイルは、最大 1 つのメンバー MAXMBR(1) を持つことができます。

- 制約は、従属ファイルと親ファイルの両方または一方のメンバーがゼロのときに定義できます。両方のファイルにメンバーがない限り、制約を確定することはできません。
- ファイルは、最大 1 つの基本キーを持つことができますが、固有制約は複数持つことができます。
- 1 ファイルあたりの最大制約関係は 300 です。この最大値は、次のものの総和です。
  - 参照制約 (親または従属のどちらで参与しているか、また制約が定義済みまたは確定済みのどちらかに関係なく)。
  - 基本キー制約を含む固有制約。
  - 検査制約
- 参照制約においては、外部記述ファイルのみが許されます。ソース・ファイルは許されません。プログラム記述ファイルは許されません。
- *RESTRICT 関係においては、挿入、更新、または削除機能をもったファイルは許されません。
- 制約名は、ライブラリーで固有名でなければなりません。
- 制約は、QTEMP ライブラリー内のファイルに追加することはできません。
- 親ファイルがある ASP 内にあり、従属ファイルが別の ASP 内にある場合には、参照制約を追加できません。

詳細: 制約サイクルの回避: *制約サイクル* は、親ファイルの下位のファイルが、その親ファイルの親になるような制約関係順序です。

制約サイクルを DB2 UDB for iSeries データベース内で使用することができますが、その使用を回避してください。

参照制約の確認

ADDPFCST コマンドを使用して制約を追加すると、システムは自動的に参照制約の妥当性を確認します。システムは、外部キー内のすべての非 NULL 値に対して、親キー内に、対応する値があることを確認します。

検査が正常に終了した場合は、その後アクセスがあるごとに、制約規則がユーザーまたはアプリケーション・プログラムによって適用されます。検査が失敗した場合には、その制約には検査保留のマークが付けられます。制約が ADDPFCST コマンドを使用して追加された場合、その制約は検査保留になりますが、使用不能状態になります。

注: 大量のデータの入った既存ファイルへの参照制約の追加は異例ではありません。大量のレコードが関与するときには、ADDPFCST コマンドは完了までに数時間かかることがあります。追加プロセスでは、ファイルは排他にロックされます。参照制約を追加する前に、まず時間的要因とファイルの可用性を考慮に入れなければなりません。

参照制約を使用可能または使用不可にする

参照制約関係を使用可能または使用不可にするには、 物理ファイル制約の変更 (CHGPFCST) コマンドを使用します。参照制約を変更する際は、従属ファイルを指定しなければなりません。親ファイルを指定して制約を使用可能または使用不可能にすることはできません。

iSeries ナビゲーターを使用して、参照制約を使用可能または使用不可にすることもできます。詳しくは、SQL プログラミングの中の、 iSeries ナビゲーターを初めて使用される場合を参照してください。

制約を使用可能または使用不可能にするには、少なくとも従属ファイルに対するオブジェクト管理権限 (または ALTER 特権) がなければなりません。

参照制約を使用可能または使用不可能にすることに関する詳細については、 『詳細: 参照制約を使用可能または使用不可能にする』を参照してください。

詳細: 参照制約を使用可能または使用不可能にする: システムが制約を使用可能または使用不可能にするときには、親ファイルおよび従属ファイル、両方のメンバー、および両方のアクセス・パスをロックします。使用可能または使用不可能が完了すると、ロックは解除されます。

使用可能な制約を使用可能にしたり、使用不可能な制約を使用不可能にしても、通知メッセージが出されるだけです。

確定済み/使用不可能または検査保留中の制約関係を使用可能にすることができます。使用可能にすると、システムは制約を再び確認します。一致していない親キーと外部キーが検査で見つかった場合、その制約には検査保留のマークが付けられます。

制約関係を使用不可能にすると、ユーザーが正しい権限を持っている場合、親ファイルと従属ファイルの両方に対してすべてのファイル入出力操作を行えるようになります。制約の下部構造はすべて元のままです。親キーと外部キーのアクセス・パスは、そのまま保持されます。ただし、使用不可能関係にあるこの 2 つのファイルに対して、参照制約は実行されません。それ以外のすべての使用可能な制約は、以前のとおりに実行されます。

制約を使用不可能にすると、パフォーマンスが重視されるファイル入出力操作の速度を上げることができます。この種の状況では、常に代償について考慮してください。ファイルのデータが参照において無効になります。制約が使用可能になっていると、ファイルのサイズに依存して、システムが参照制約関係を再検査するのに時間がかかります。

注: 制約関係が確定済み使用不可能状態になっているファイルを変更する場合、ユーザーおよびアプリケーションは注意しなければなりません。制約がもう一度使用可能になるまで、関係違反が起きてもそれが検出されないことがあります。

制約関係が使用不可になっている間は、 オブジェクトの割り振り (ALCOBJ) コマンドで、ファイルを割り振る (ロックする) ことができます。この場合の割り振りは、その参照制約関係が使用不可能の間、他人がファイルを変更するのを防止します。排他的使用のためのロックを要求するのは、他のユーザーがファイルを読み取れるようにするためです。制約を再度使用可能にすると、 オブジェクトの割り振り解除 (DLCOBJ) コマンドでファイルのロックを解除できます。

複数の制約を使用可能または使用不可能にする場合、それらの制約は順番に処理されます。変更できない制約の場合には、診断メッセージが出され、処理はリスト内の次の制約へ進みます。すべての制約の処理が終わると、変更された制約数をリストした完了メッセージが出されます。

参照制約の除去

参照制約は、さまざまな方法で除去できます。除去の効果全体は、除去する制約、およびその制約を取り巻く特定条件によって異なります。

参照制約を除去するには、以下の手順を実行してください。

1. 物理ファイル制約の除去 (RMVPFCST) コマンドを実行します。
2. 以下のいずれかのパラメーターを使用して、除去したい制約 (複数可) を指定します。
   - CST パラメーターを指定してすべての制約を指定するかまたは特定の制約名を指定します。
   - TYPE パラメーターを使用して特定の制約タイプを指定します。

iSeries ナビゲーターを使用して、参照制約を除去することもできます。詳しくは、 SQL プログラミングの中の、 iSeries ナビゲーターを初めて使用される場合を参照してください。

参照制約を除去すると、システムはそれに関連した外部キーおよびアクセス・パスをファイルから除去します。システムは、システム上で論理ファイルまたはその他の制約が使用している外部キーのアクセス・パスは除去しません。

参照制約、基本キー制約、または固有制約を除去した場合に、それに関連したアクセス・パスを論理ファイルが共用していれば、共用パスの所有権はその論理ファイルに移ります。

参照制約の除去に関する詳細については、以下のトピックを参照してください。
『詳細: CST パラメーターを使用して制約を除去する』
『詳細: TYPE パラメーターを使用して制約を除去する』

詳細: CST パラメーターを使用して制約を除去する: CST パラメーターを使用して、以下の除去を指定できます。

- TYPE(*ALL) が指定されたファイルに関連したすべての制約 CST(*ALL)
- 特定の参照制約 CST(*制約名*)
- 検査保留中 CST(*CHKPND) の参照制約または検査制約
- 特定の制約タイプに関連したすべての制約 CST(*ALL)

詳細: TYPE パラメーターを使用して制約を除去する: TYPE パラメーターを使用して、除去する制約のタイプを指定できます。

- すべてのタイプ: TYPE(*ALL)
  - CST(*ALL) の場合は、すべての制約
  - CST(*CHKPND) の場合は、検査保留中のすべての制約
  - CST(制約名) の場合は、指定した制約
- 参照制約: TYPE(*REFCST)
  - CST(*ALL) の場合は、すべての参照制約
  - CST(*CHKPND) の場合は、検査保留中のすべての参照制約
  - CST(参照名) の場合は、指定した参照制約
- 固有制約: TYPE(*UNQCST)
  - CST(*ALL) の場合は、基本キー制約を除くすべての固有制約
  - CST(*CHKPND) の場合は、適用されない － 固有制約は検査保留にはなりません。
  - CST(制約名) の場合は、指定された固有制約
- 基本キー制約: TYPE(*PRIKEY)
  - CST(*ALL) の場合は、基本キー制約
  - CST(*CHKPND) の場合は、適用されない － 基本キー制約は、検査保留になりません。
  - CST(制約名) の場合は、指定された基本キー制約
- 検査制約: TYPE(*CHKCST)
  - CST(*ALL) の場合は、すべての検査制約
  - CST(*CHKPND) の場合は、検査保留中のすべての検査制約
  - CST(制約名) の場合は、指定された検査制約

詳細: 参照制約を使用したデータ保全性の保証

いくつかの参照保全をデータベース管理システムで使用したい場合があります。

- ファイル間のデータ値がユーザーのビジネスの規則に合致させる。たとえば、あるファイルに顧客リストを入れ、その会計を別のファイルに入れる業務の場合があります。会計追加は、該当する顧客が存在しないと無意味になります。同様に、顧客の削除は、その会計をすべて削除しない限り適切ではありません。
- データ値相互間の関係を定義できる。
- アプリケーションによる変更の内容に関係なく、システムにデータ関係を確保させる。
- 検査をデータベース内に移動させて 高水準言語 (HLL) または SQL レベルで行われる保全性検査のパフォーマンスを向上させる。

例: 参照制約を使用したデータ保全性の保証

データベースには、従業員ファイルと部門ファイルが入っています。両方のファイルに、DEPTNO という名前の部門番号フィールドがあります。これらのデータベース・ファイルの関連レコードは、従業員 DEPTNO が部門 DEPTNO に等しいレコードです。

この例の目的は、従業員ファイル内のすべての従業員が、それぞれが属する対応する部門を部門ファイル内に持っていることを確認することです。これは、参照制約によって行うことができます。

1. ADDPFCST コマンドを使用して、DEPTNO フィールド用の部門ファイルに基本キー制約または固有制約を追加します。これは、親キーになります。参照制約が追加されていないためにまだ親キーではありません。
2. ADDPFCST コマンドを使用して従業員ファイルに参照制約を追加します。従業員ファイルは、従属ファイルになります。外部キーは、従業員 DEPTNO になります。部門ファイルは、親キー、部門 DEPTNO を持つ親ファイルになります。これは、キーとして DEPTNO フィールドを持つ基本キー制約または固有制約のいずれかがあるため、制約が参照制約と関連した親キーとして働きます。

参照制約には、親ファイルまたは従属ファイルへのレコードの挿入、更新、および削除のさいに従わなければならない更新規則および削除規則があります。

参照保全の用語

参照保全の説明では、いくつかの用語を理解している必要があります。これらの用語は、用語相互間の関係を理解するのに便利な順番に並んでいます。

基本キー制約. 固有、昇順、および NULL を含むことはできない、データベース・ファイル内の 1 つまたは一連のフィールド。基本キーは、基本ファイルのアクセス・パスです。基本キー制約は、参照制約を追加するさいに親キーとして使用できます。基本キー制約は特殊属性を持つ固有制約です。

固有制約. 固有、昇順、および NULL を含むことのできる、データベース・ファイル内の 1 つまたは複数のフィールド。

親キー. 固有、昇順、および NULL を含む場合も含まない場合もある、データベース・ファイル内の 1 つまたは複数のフィールド。親ファイルの親キーは、従属ファイルに参照制約を追加するために使用されます。親キーは、基本キーまたは固有制約のいずれかでなければなりません。

外部キー. 関連親ファイルの親キー値と一致していなければならない非 NULL が入る 1 つまたは複数のフィールドのセット。

属性 (データ・タイプ、長さなど) は、親ファイルの親キーと同じでなければなりません。

親ファイル. 親キーを含む参照制約関係にあるファイル。

従属ファイル. 外部キーを含む参照制約関係にあるファイル。従属ファイルは、親ファイルに従属しています。つまり、従属ファイルの外部キー内のすべての非 NULL 値に対して、親ファイルの親キー内に、対応する非 NULL 値がなければなりません。

検査保留. 参照制約に関して、従属ファイルの外部キー内の非 NULL 値に対して親ファイルの親キー内に、対応する非 NULL 値がなければならないことが、真であるかどうかがデータベースにはっきり知られていないときに生じる状態。

削除規則. 親レコードの削除が試みられたときに、データベースがとるべき処置の定義。

更新規則. 親レコードの更新が試みられたときに、データベースがとるべき処置の定義。

参照保全の実行

確立され、使用可能になった制約に関連するファイルへの入出力アクセスは、多様です。そのファイルが制約関係におかれた親キーまたは外部キーを含んでいるかどうかによって異なります。システムは、親ファイルおよび従属ファイルのすべての入出力要求に対して、参照保全を実施します。

データベースは、アプリケーション・プログラムまたはシステム・コマンド (INZPFM コマンドなど)、あるいは SQL ステートメントまたはファイル入出力ユーティリティー (STRSEU など) からのすべての入出力要求に対して制約規則を適用します。

iSeries システムにおける参照保全の実施について詳しくは、以下のトピックを参照してください。

- 『外部キーの実行』
- 『親キーの実行』

外部キーの実行: 制約の作成時に指定した削除規則および更新規則は、親キーの変更に対して適用されます。データベースは、参照の保全性を保つために、外部キーの更新および挿入に対して非処置規則を適用します。システムは、この規則を外部キーの更新と挿入に対して実行して、すべての非 NULL 外部キーが親キーの値と一致するようにします。

新しい外部キー値に一致する親キーがない場合は、システムは参照制約違反を戻し、従属レコードは挿入も更新もされません。

親キーの実行: 参照制約に指定した規則は、データベースが親キーの削除と更新を処理する方法を決定します。システムは、すべての親ファイル入出力に対して親キーの固有属性を適用します。

削除および更新規則の実施に関する詳細については、以下のトピックを参照してください。

- 『削除規則の実行』
- 252 ページの『更新規則の実行』

削除規則の実行: 親ファイルからレコードを削除すると、システムは従属ファイル内に従属レコード (一致する非 NULL の外部キー値) があるかどうかを検査します。従属レコードが検出されると、削除規則により実行する処置が判別されます。

- **無処置**－従属レコードを検出した場合、システムは制約違反を戻し、レコードは削除されません。
- **カスケード**－システムは、従属ファイル内で検出した従属レコードを削除します。
- **NULL 設定**－見つけた従属レコードごとに、外部キー内の NULL 可能フィールドを NULL に設定します。
- **省略時値設定**－システムは、一致する親キーを削除するときに、外部キーのすべてのフィールドにそれぞれの省略時値を設定します。
- **制限**－即時実行であることを除いて、無処置の場合と同じです。

実行された削除規則の一部が失敗しても、削除操作全体が失敗し、すべての関連変更はロールバックされます。たとえば、削除のカスケード規則が原因で、データベースが 10 個の従属レコードを削除したが、最後のレコードの削除中にシステム障害が発生したとします。データベースは親キー・レコードの削除を許可せず、削除された従属レコードは再挿入されます。

参照制約の実行が原因で、レコードが変更された場合、それに関連したジャーナル項目には、参照制約が原因でレコードが変更されたことを示す標識が付けられます。たとえば、削除のカスケード規則で削除された従属レコードには、そのレコードの変更が参照制約の実行中に生成されたことを示すジャーナル項目標識が付けられます。

更新規則の実行: システムが親ファイル内で親キーを更新するときには、従属ファイル内に従属レコード (一致する非 NULL の外部キー値) があるかどうかを検査します。従属レコードが検出されると、更新規則により実行する処置が判別されます。

- **無処置**－従属レコードを検出した場合、システムは制約違反を戻し、レコードは更新されません。
- **制限**－システムは上記と同じ処置を実行しますが、即時に実施されます。

制約状態

ファイルは、3 つの制約状態のうちのどれか 1 つです。このうち 2 つの状態では、制約を使用可能にも使用不可能にもすることができます。

- **非制約関係状態。** この状態にあるファイルには、参照制約は存在しません。そのファイルにかつて制約関係が存在していた場合も、それに関する情報はすべて除去されています。
- **定義状態。** 従属ファイルと親ファイルの間に、制約関係が定義されています。制約関係を定義するのに、どちらかのファイルにメンバーを作成する必要はありません。定義状態では、制約は次のいずれかになります。
  - 定義済み使用可能。使用可能な定義済みの制約関係は、単に定義のためだけのものです。この制約の規則は適用されません。この状態にある制約は、確定状態に移っても使用可能のままです。
  - 定義済み使用不可能。使用不可能な定義済みの制約関係は、単に定義のためだけのものです。この制約の規則は適用されません。この状態にある制約は、確定状態に移っても使用不可能のままです。
- **確定状態。** 従属ファイルは、親ファイルと制約関係にあります。制約が確定するのは、外部キーと親キーの属性が一致している場合だけです。両方のファイルに、メンバーが存在しなければなりません。確定状態では、制約は次のいずれかになります。
  - 確定済み使用可能。使用可能な確定済みの制約関係があれば、データベースが参照保全を実行することになります。
  - 確定済み使用不可能。使用不可能な確定済みの制約関係があれば、データベースに参照保全を実行しないよう指示が出ます。

参照制約内の検査保留状況の検査

検査保留 は、親キーと外部キーが潜在的に一致していない可能性がある場合の、制約関係状態です。システムが、参照保全の違反がある可能性があると判断したときには、その制約関係には検査保留のマークが付けられます。たとえば、次の場合があります。

- 従属ファイルのデータだけを復元する復元操作が、システム上の親ファイルと同期しなくなっている (外部キーに親がない) 場合。
- 一致する外部キーが存在しており、システム障害が発生したため、親キー値が削除され得る場合。これは、従属ファイルと親ファイルがジャーナル処理されていない場合に限って発生することがあります。
- 外部キー値に、対応する親キー値がない場合。これは、それ以前に制約関係に関与したことのない既存ファイルに参照制約を追加する場合に発生する可能性があります。

検査保留状況は、*NO または *YES です。

検査保留は、確定状態にある制約に限って適用されます。確定済み使用可能参照制約には、*YES または *NO の検査保留状況があります。

制約関係を検査保留状態から解除するには、関係を使用不可能にし、キー (外部、親、または両方) ・データを訂正してから、制約をもう一度使用可能にしなければなりません。この後、データベースは制約関係をもう一度検査します。

関係が検査保留にあるときには、親ファイルと従属ファイルは、使用を制限された状態になります。親ファイルの入出力制限は、従属ファイルの制限事項とは異なります。検査保留制限事項は、確定済み使用不可能状況 (これは常に検査保留状況にあります) にある制約には適用されません。

検査保留状況および参照制約に関する詳細については、以下のトピックを参照してください。
『検査保留中の従属ファイルの制限事項』
『検査保留中の親ファイルの制限事項』

検査保留中の従属ファイルの制限事項: 検査保留中の確定済み使用可能参照制約には、以下の項目が適用されます。

制約関係に入っており、検査保留のマークが付けられている従属ファイルでは、ファイルの入出力操作を行うことはできません。従属ファイルと親ファイル間のファイルの不一致を訂正しなければなりません。また、システムが入出力操作を許可する前に、その関係を検査保留状態から解除する必要があります。ユーザーまたはアプリケーションが検査保留状況および制約違反を認識しない場合があるため、システムはこのようなファイルからレコードを読み取ることを許可しません。

検査保留中の使用可能参照制約を持つ従属ファイル上で入出力操作を実行するには、最初に制約を使用不可能にしてから入出力操作を実行します。

検査保留中の親ファイルの制限事項: 検査保留中の確定済み使用可能参照制約には、以下の項目が適用されます。

制約関係にあって、検査保留のマークの付けられた親ファイルは、オープンはできますが、いくつかのタイプの入出力において制限を受けます。レコードの読み取りおよび挿入はできますが、削除または更新はできません。

検査保留中の使用可能参照制約を持つ親ファイル上で更新および削除を実行するには、最初に制約を使用不可能にしてから入出力操作を実行します。

参照保全と iSeries 機能

参照保全は、以下の iSeries システム機能の特性に影響します。

- 物理ファイル・メンバーの追加 (ADDPFM):

  メンバーがゼロの従属ファイルと親ファイルの間で制約関係を定義する場合は、以下のようになります。
  - 最初にメンバーが親ファイルに追加される場合は、制約関係は定義された状態のままになります。
  - 次にメンバーが従属ファイルに追加される場合は、外部キーのアクセス・パスが作成され、親との制約関係が確定します。
- 物理ファイルの変更 (CHGPF):

ファイルの制約関係が存在するときには、CHGPF コマンド内で利用可能な特定のパラメーターを変更できません。以下のパラメーターが制限されています。

MAXMBRS
制約関係を結んだファイルの最大メンバー数は、1 です (MAXMBRS(1))。

CCSID
制約と関係のないファイルの CCSID は変更できます。ファイルが制約と関連している場合は、CCSID は 65535 にしか変更できません。

- 物理ファイル・メンバーの消去 (CLRPFM):

  レコードを持ち、使用可能な参照制約と関連した親ファイルに対して CLRPFM コマンドを出すと、失敗します。
- FORTRAN データの強制終了 (FEOD):

  使用可能な参照制約関係に関連する親ファイルに FEOD を出すと、失敗します。
- 重複オブジェクトの作成 (CRTDUPOBJ):

  CRTDUPOBJ コマンドでファイルを作成すると、取り出し元ファイルと関連した制約はすべて宛先ファイルに伝搬されます。

  親ファイルが同じライブラリーまたは異なるライブラリーに複製される場合は、システム相互参照ファイルが使用されて、定義済み参照制約の従属ファイルが探し出されます。また、システムは制約関係を確立しようとします。

  従属ファイルをコピーする場合、TOLIB が使用されて、制約関係が判別されます。
  - 親ファイルと従属ファイルが同じライブラリーにある場合は、 TOLIB の親ファイルとの参照制約関係が確立します。
  - 親ファイルと従属ファイルが別々のライブラリー内にある場合には、コピーした従属ファイルは、元の親ファイルとの参照制約関係を確立します。
- ファイルのコピー (CPYF):

  CPYF で新しいファイルを作成すると、元のファイルに制約があっても、その制約は新しいファイルにコピーされません。
- オブジェクトの移動 (MOVOBJ):

  MOVOBJ コマンドは、ライブラリー相互間でファイルを移動します。システムは、新ライブラリー内のファイルに関連し、かつ存在する可能性のある、定義済み参照制約を確定しようとします。
- オブジェクトの名前変更 (RNMOBJ):

  RNMOBJ コマンドは、同一のライブラリー内のファイルの名前を変更するか、またはライブラリーの名前を変更します。

  名前変更されたファイルまたはライブラリーに存在する可能性のある、定義済み参照制約を確定しようとする試みが行われます。
- ファイルの削除 (DLTF):

  DLTF コマンドには、参照制約関係の処理を指定するための任意選択キーワードがあります。RMVCST キーワードは、制約関係にある従属ファイルに対して適用されます。このキーワードは、親ファイルが削除されたときに、従属ファイルの制約関係をどの程度除去するかを指定します。

***RESTRICT**
親ファイルと従属ファイルの間で制約関係が定義または確定されている場合、親ファイルは削除されず、制約関係は除去されません。これはデフォルト値です。

***REMOVE**
親ファイルは削除され、制約関係と定義は除去されます。親ファイルと依存ファイルの間の制約関係は除去されます。従属ファイルの対応した外部キー・アクセス・パスおよび制約定義は除去されません。

***KEEP**
親ファイルは削除され、参照制約関係の定義は定義済み状態のままになります。従属ファイルの対応した外部キー・アクセス・パスおよび制約定義は削除されません。

- 物理ファイル・メンバーの除去 (RMVM):

  制約関係にある親ファイルのメンバーが削除されると、制約関係は、定義済み状態になります。外部キー・アクセス・パスおよび参照制約定義は、削除されません。親メンバーが除去されたので、親キーのアクセス・パスは除去されますが、親の制約定義はファイル・レベルにとどまります。

  制約関係にある従属ファイルのメンバーが削除されると、制約関係は定義済み状態になります。親キーのアクセス・パスおよび制約定義は除去されません。従属メンバーが除去されたので、外部キーのアクセス・パスは除去されますが、参照制約定義は除去されません。

- 保管/復元:

  親ファイルがライブラリーに復元される場合、システム相互参照ファイルが使用されて、定義済み参照制約の従属ファイルを探し出します。制約関係を確立しようとする試みが行われます。

  従属ファイルを復元する場合には、TOLIB が使用されて、制約関係が判別されます。
  - 親ファイルと従属ファイルが同じライブラリーにある場合は、TOLIB 内の親ファイルとの参照制約関係が確立されます。
  - 親ファイルと従属ファイルが別々のライブラリー内にある場合には、コピーされた従属ファイルは元の親ファイルとの参照制約関係を確立します。

  制約関係におかれた親ファイルと従属ファイルの復元順は問題ではありません (つまり、親ファイルが従属ファイルより前に復元されるか、またはこの逆)。制約関係は確立されます。

データベース内での自動イベントのトリガー

トリガー は、指定した物理データベース・ファイルで、指定した変更操作または読み取り操作が行われるときに自動的に実行される一連の処置です。変更操作には、アプリケーション・プログラムにおける高水準言語ステートメントの挿入、更新、または削除があります。読み取り操作には、アプリケーション・プログラムにおける高水準言語ステートメントのフェッチ、ゲット、または読み取りがあります。

iSeries では、一連のトリガー処置を、サポートされているどの高水準言語でも定義できます。以下のトピックでは、従来のシステム・インターフェースを使用してトリガーを処理する方法が説明されています。

- 256 ページの『トリガーの使用』
- 256 ページの『業務でトリガーを使用する利点』
- 256 ページの『トリガー・プログラムの作成』
- 278 ページの『ファイルへのトリガーの追加』
- 279 ページの『トリガーの表示』
- 280 ページの『トリガーの除去』

- 280 ページの『トリガーを使用可能または使用不可にする』
- 280 ページの『トリガーおよびその他の iSeries 機能との関係』
- 282 ページの『トリガーおよび参照保全との関係』

また、SQL トリガーも使用できます。詳しくは、 SQL プログラミングの中の、 SQL トリガーを参照してください。

トリガーの使用についてのさまざまな情報は、「ストアード・プロシージャーおよびトリガー DB2 Universal Database™ for iSeries  」レッドブックを参照してください。

トリガーの使用

データベース内でトリガーは次のように使用できます。

- 業務規則の実行
- 入力データの妥当性検査
- 別のファイルへの新規挿入行用の固有値の生成 (代理機能)
- 監査証跡を目的とした、他のファイルへの書き込み
- 相互参照を目的とした、他のファイルからの Query
- システム機能のアクセス (たとえば、規則違反のときの例外メッセージの印刷)
- データの一貫性を保つための、別のファイルへのデータのコピー

業務でトリガーを使用する利点

トリガーは、業務に次の利点を提供します。

- アプリケーション開発の高速化。トリガーはデータベースに保管されるため、それぞれのデータベース・アプリケーションごとにトリガーの処置をコード化する必要がありません。
- 業務規則のグローバルな実行。トリガーをいったん定義すると、データベースを使用するどのアプリケーションにも再使用できます。
- 保守の簡素化。業務方針が変更になった場合でも、各アプリケーション・プログラムではなく、該当するトリガー・プログラムを変更するだけで済みます。
- クライアント/サーバー環境でのパフォーマンスの向上。すべての規則はサーバー内で実行され、結果が戻されます。

トリガー・プログラムの作成

トリガー は、指定した物理データベース・ファイルで、指定した変更操作または読み取り操作が行われるときに自動的に実行される一連の処置です。トリガーを使用して、権限保護などの業務規則を実施することができます。トリガーは、監査証跡の保持、例外条件の検出、およびデータベース内の関係の維持に役立ちます。

物理ファイルにトリガーを追加するには、以下の手順を実行してください。

1. 最初にトリガー・プログラムを供給しなければなりません。トリガー・プログラムは高水準言語、構造化照会言語 (SQL) または制御言語 (CL) を使用して記述できます。C、COBOL、および RPG でコーディングされた 258 ページの『トリガー・プログラムの例』を参照してください。
2. 以下のメソッドのいずれか 1 つを使用してトリガーを追加します。
   - 物理ファイル・トリガーの追加 (ADDPFTRG) コマンド。コマンド上のトリガー・プログラム (PGM) パラメーターにトリガー・プログラムを指定しなければなりません。

- iSeries ナビゲーターを使用してトリガーを追加する。『iSeries ナビゲーターを使用したトリガーの追加』を参照してください。
- CREATE TRIGGER SQL ステートメント。

トリガーの説明については、以下のトピックを参照してください。

- 『トリガー・プログラムの動作』
- 258 ページの『トリガーの処理についてのその他の重要情報』

iSeries ナビゲーターを使用したトリガーの追加: トリガーは、指定した物理データベース・ファイルで、指定した変更操作が行われるときに自動的に実行される一連の処置です。ここでは、表は物理ファイルです。変更操作には、アプリケーション・プログラムにおける高水準言語ステートメントの挿入、更新、または削除があります。あるいは、SQL の INSERT、UPDATE、または DELETE ステートメントです。トリガーは、業務規則の実行、入力データの検証、監査証跡の保持などのタスクに役立ちます。

iSeries ナビゲーターを使用して、システム・トリガーおよび SQL トリガーを定義することができます。加えて、トリガーを使用可能または使用不可にすることができます。

トリガーを追加するには、以下の手順を実行してください。

1. **iSeries ナビゲーター**・ウィンドウで、ご使用のサーバー → 「**データベース**」 → 「**ライブラリー**」を拡張する。
2. トリガーを追加する表が入っているライブラリーをクリックする。
3. トリガーを追加する表を右マウス・ボタン・クリックして、「**プロパティー**」を選択する。「**表のプロパティー (Table Properties)**」ダイアログで、「**トリガー**」タブをクリックする。
4. システム・トリガーを追加するには、「**システム・トリガーの追加 (Add system trigger)**」を選択する。
5. SQL トリガーを追加するには、「**SQL トリガーの追加 (Add SQL trigger)**」を選択する。

システム・トリガーの詳細については、 255 ページの『データベース内での自動イベントのトリガー』を参照してください。

SQL トリガーの詳細については、 SQL プログラミングの SQL トリガーを参照してください。

トリガー・プログラムの動作: ユーザーまたはアプリケーションが、関連付けられたトリガーを持つ物理ファイルで変更操作または読み取り操作を実行すると、その操作は該当するトリガー・プログラムを呼び出します。

変更操作または読み取り操作によって、以下の表のような 2 つのパラメーターがトリガー・プログラムに渡されます。

パラメーター	説明	入力または出力	タイプ
1	トリガー・バッファー。このパラメーターには、このトリガー・プログラムを呼び出している現在の変更操作に関する情報が入っています。詳細については、 271 ページの『トリガー・バッファー・セクション』を参照してください。	入力	CHAR(*)
2	トリガー・バッファー長。	入力	BINARY(4)

これらの入力からトリガー・プログラムは、元のレコードまたは新しいレコードのコピーを参照できます。トリガー・プログラムをコード化して、これらのパラメーターを受け入れるようにします。

トリガーの処理についてのその他の重要情報: 以下のトピックでは、トリガー・プログラムのコーディングに関する追加情報を提供しています。

- 274 ページの『トリガー・プログラムに関する推奨事項』
- 274 ページの『トリガー・プログラムのコーディング時の指針』
- 277 ページの『トリガー・プログラム使用のモニター』
- 277 ページの『コミットメント制御の下で実行されるトリガーおよびアプリケーション・プログラム』
- 277 ページの『コミットメント制御の下で実行されないトリガーおよびアプリケーション・プログラム』
- 277 ページの『トリガー・プログラムのエラー・メッセージ』

トリガー・プログラムの例: コードの例については、 『コードについての特記事項』を参照してください。

ATMTRANS ファイルへの書き込み、更新、および削除操作によってトリガーを行うトリガー・プログラムの例を以下に示します。これらの例で使用されるデータベースの説明については、 271 ページの『トリガー・プログラム: 例で使用されるデータベースのデータ構造』をご覧ください。

| これらのトリガー・プログラムは、 ILE C、ILE COBOL、および RPG/400 で書かれています。 ILE RPG
| の例については、「ストアード・プロシージャーおよびトリガー DB2 Universal Database for iSeries
|  」レッドブックを参照してください。

このアプリケーションには、4 つのタイプのトランザクションが含まれます。

1. アプリケーションが 3 つのレコードを、挿入トリガーを実行する ATMTRANS ファイルに挿入します。挿入トリガー (259 ページの『例: RPG で作成された挿入トリガー』) でATMS ファイルに正しい金額が追加され、 ACCTS ファイルがその変更を反映します。
2. 次に、アプリケーションは更新トリガーを呼び出す 2 つの引き出しを実行します (262 ページの『例: ILE COBOL で作成された更新トリガー』)。
   a. アプリケーションは、口座番号 20001 および ATM 番号 10001 から $25.00 を引き出します。これによって更新トリガーが呼び出されます。更新トリガーは、ACCTS ファイルおよび ATMS ファイルから $25.00 を差し引きます。
   b. アプリケーションは、口座番号 20002 および ATM 番号 10002 から $900.00 を引き出します。これによって、更新トリガーが呼び出されます。更新トリガーは、トランザクションが失敗したことを示す例外をアプリケーションに送信します。
3. 最後に、アプリケーションは、ATMTRANS ファイルから ATM 番号を削除します。これによって、削除トリガーが呼び出されます。削除トリガー (266 ページの『例: ILE C で書かれた削除トリガー』) は、対応する ACCTID を ACCTS ファイルから、 ATMID を ATMS ファイルから削除します。

コードについての特記事項: 本書には、プログラミングの例が含まれています。

IBM は、お客様に、すべてのプログラム・コードのサンプルを使用することができる非独占的な著作使用権を許諾します。お客様は、このサンプル・コードから、お客様独自の特別のニーズに合わせた類似のプログラムを作成することができます。

すべてのサンプル・コードは、例として示す目的でのみ、IBM により提供されます。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。

ここに含まれるすべてのプログラムは、現存するままの状態で提供され、いかなる保証も適用されません。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の保証の適用も一切ありません。

例: RPG で作成された挿入トリガー: コードの例については、 258 ページの『コードについての特記事項』を参照してください。

以下の RPG トリガー・プログラムは、レコードを ATMTRANS ファイルに挿入します。

```
* Program Name : INSTRG
       * This is an insert trigger for the application
       * file.  The application inserts the following three
       * records into the ATMTRANS file.
       *
       * ATMID   ACCTID   TCODE    AMOUNT
       * -------------------------------
       * 10001   20001      D      100.00
       * 10002   20002      D      250.00
       * 10003   20003      D      500.00
       *
       * When a record is inserted into ATMTRANS, the system calls
       * this program, which updates the ATMS and
       * ACCTS files with the correct deposit or withdrawal amount.
       * The input parameters to this trigger program are:
       *  - TRGBUF : contains trigger information and newly inserted
       *             record image of ATMTRANS.
       *  - TRGBUF Length : length of TRGBUF.
       *
      H        1
       *
       * Open the ATMS file and the ACCTS file.
       *
      FATMS    UF  E                    DISK         KCOMIT
      FACCTS   UF  E                    DISK         KCOMIT
       *
       * DECLARE THE STRUCTURES THAT ARE TO BE PASSED INTO THIS PROGRAM.
       *
      IPARM1       DS
       * Physical file name
      I                                       1  10 FNAME
       * Physical file library
      I                                      11  20 LNAME
       * Member name
      I                                      21  30 MNAME
       * Trigger event
      I                                      31  31 TEVEN
       * Trigger time
      I                                      32  32 TTIME
       * Commit lock level
      I                                      33  33 CMTLCK
       * Reserved
      I                                      34  36 FILL1
       * CCSID
      I                                   B  37  400CCSID
       * Reserved
      I                                      41  48 FILL2
       * Offset to the original record
      I                                   B  49  520OLDOFF
       * length of the original record
      I                                   B  53  560OLDLEN
```

```
 * Offset to the original record null byte map
I                                   B  57  600ONOFF
 * length of the null byte map
I                                   B  61  640ONLEN
 * Offset to the new record
I                                   B  65  680NOFF
 * length of the new record
I                                   B  69  720NEWLEN
 * Offset to the new record null byte map
I                                   B  73  760NNOFF
 * length of the null byte map
I                                   B  77  800NNLEN
 * Reserved
I                                      81  96 RESV3
 * Old record ** not applicable
I                                      97 112 OREC
 * Null byte map of old record
I                                     113 116 OOMAP
 * Newly inserted record of ATMTRANS
I                                     117 132 RECORD
 * Null byte map of new record
I                                     133 136 NNMAP
IPARM2       DS
I                                   B   1   40LENG
 ******************************************************************
 * SET UP THE ENTRY PARAMETER LIST.
 ******************************************************************
C           *ENTRY    PLIST
C                     PARM           PARM1
C                     PARM           PARM2
 ******************************************************************
 * Use NOFF, which is the offset to the new record, to
 * get the location of the new record from the first
 * parameter that was passed into this trigger program.
 *   - Add 1 to the offset NOFF since the offset that was
 *     passed to this program started from zero.
 *   - Substring out the fields to a CHARACTER field and
 *     then move the field to a NUMERIC field if it is
 *     necessary.
 ******************************************************************
C                     Z-ADDNOFF      O       50
C                     ADD  1         O
 ******************************************************************
 *         - PULL OUT THE ATM NUMBER.
 ******************************************************************
C           5         SUBSTPARM1:O   CATM    5
 ******************************************************************
 *         - INCREMENT "O", WHICH IS THE OFFSET IN THE PARAMETER
 *           STRING.  PULL OUT THE ACCOUNT NUMBER.
 ******************************************************************
C                     ADD  5         O
C           5         SUBSTPARM1:O   CACC    5
 ******************************************************************
 *         - INCREMENT "O", WHICH IS THE OFFSET IN THE PARAMETER
 *           STRING.  PULL OUT THE TRANSACTION CODE.
 ******************************************************************
C                     ADD  5         O
C           1         SUBSTPARM1:O   TCODE   1
 ******************************************************************
 *         - INCREMENT "O", WHICH IS THE OFFSET IN THE PARAMETER
 *           STRING.  PULL OUT THE TRANSACTION AMOUNT.
 ******************************************************************
C                     ADD  1         O
C           5         SUBSTPARM1:O   CAMT    5
C                     MOVELCAMT      TAMT    52
 ************************************************************
 *  PROCESS THE ATM FILE.                    ****************
```

```
 ************************************************************
 * READ THE FILE TO FIND THE CORRECT RECORD.
C           ATMN      DOUEQCATM
C                     READ ATMS                     61EOF
C                     END
C   61                GOTO EOF
 * CHANGE THE VALUE OF THE ATM BALANCE APPROPRIATELY.
C           TCODE     IFEQ 'D'
C                     ADD  TAMT      ATMAMT
C                     ELSE
C           TCODE     IFEQ 'W'
C                     SUB  TAMT      ATMAMT
C                     ELSE
C                     ENDIF
C                     ENDIF
 * UPDATE THE ATM FILE.
C           EOF       TAG
C                     UPDATATMFILE
C                     CLOSEATMS
 ************************************************************
 *  PROCESS THE ACCOUNT FILE.                ****************
 ************************************************************
 * READ THE FILE TO FIND THE CORRECT RECORD.
C           ACCTN     DOUEQCACC
C                     READ ACCTS                    62  EOF2
C                     END
C   62                GOTO EOF2
 * CHANGE THE VALUE OF THE ACCOUNTS BALANCE APPROPRIATELY.
C           TCODE     IFEQ 'D'
C                     ADD  TAMT      BAL
C                     ELSE
C           TCODE     IFEQ 'W'
C                     SUB  TAMT      BAL
C                     ELSE
C                     ENDIF
C                     ENDIF
 * UPDATE THE ACCT FILE.
C           EOF2      TAG
C                     UPDATACCFILE
C                     CLOSEACCTS
 *
C                     SETON                     LR
```

アプリケーションによる挿入の後では、ATMTRANS ファイルには次のようなデータが入っています。

ATMID	ACCTID	TCODE	AMOUNT
10001	**20001**	**D**	**100.00**
10002	**20002**	**D**	**250.00**
10003	**20003**	**D**	**500.00**

挿入トリガー・プログラムによって ATMTRANS ファイルから更新が行われた後では、ATMS ファイルおよび ACCTS ファイルには次のようなデータが入っています。

ATMN	LOCAT	ATMAMT
10001	**MN**	**300.00**
10002	**MN**	**750.00**
10003	**CA**	**750.00**

ACCTN	BAL	ACTACC
20001	200.00	A
20002	350.00	A
20003	500.00	C

例: ILE COBOL で作成された更新トリガー: コードの例については、 258 ページの『コードについての特記事項』を参照してください。

以下の ILE COBOL トリガー・プログラムは、ATMTRANS ファイルでレコードが更新されるときに呼び出されます。

```
100        IDENTIFICATION DIVISION.
200        PROGRAM-ID. UPDTRG.
300 ************************************************************************
400 **** Program Name : UPDTRG                                              *
500 *****                                                                   *
600 ***** This trigger program is called when a record is updated          *
700 ***** in the ATMTRANS file.                                            *
800 ***** This program will check the balance of ACCTS and                 *
900 ***** the total amount in ATMS.If either one of the amounts            *
1000 ***** is not enough to meet the withdrawal, an exception             *
1100 ***** message is signalled to the application.                        *
1200 ***** If both ACCTS and ATMS files have enough money, this           *
1300 ***** program will update both files to reflect the changes.         *
1400 *****                                                                 *
1500 ***** ATMIDs of 10001 and 10002 will be updated in the ATMTRANS       *
1600 ***** file with the following data:                                   *
1700 *****                                                                 *
1800 ***** ATMID   ACCTID   TCODE    AMOUNT                               *
1900 ***** -------------------------------                                *
2000 ***** 10001   20001      W       25.00                               *
2100 ***** 10002   20002      W      900.00                               *
2200 ***** 10003   20003      D      500.00                               *
2300 *****                                                                 *
2400 ************************************************************************
2500       *************************************************************
2600        ENVIRONMENT DIVISION.
2700        CONFIGURATION SECTION.
2800        SOURCE-COMPUTER. IBM-AS400.
2900        OBJECT-COMPUTER. IBM-AS400.
3000        SPECIAL-NAMES. I-O-FEEDBACK IS FEEDBACK-JUNK.
3100        INPUT-OUTPUT SECTION.
3200        FILE-CONTROL.
3300            SELECT ACC-FILE ASSIGN TO DATABASE-ACCTS
3400                   ORGANIZATION IS INDEXED
3500                   ACCESS IS RANDOM
3600                   RECORD KEY IS ACCTN
3700                   FILE STATUS IS STATUS-ERR1.
3800
3900            SELECT ATM-FILE ASSIGN TO DATABASE-ATMS
4000                   ORGANIZATION IS INDEXED
4100                   ACCESS IS RANDOM
4200                   RECORD KEY IS ATMN
4300                   FILE STATUS IS STATUS-ERR2.
4400
4500       *************************************************************
4600       *                   COMMITMENT CONTROL AREA.                *
4700       *************************************************************
4800        I-O-CONTROL.
4900            COMMITMENT CONTROL FOR ATM-FILE, ACC-FILE.
5000
5100       *************************************************************
5200       *                  DATA DIVISION                            *
5300       *************************************************************
```

```
5400
5500        DATA DIVISION.
5600        FILE SECTION.
5700        FD  ATM-FILE
5800            LABEL RECORDS ARE STANDARD.
5900        01  ATM-REC.
6000        COPY DDS-ATMFILE OF ATMS.
6100
6200        FD  ACC-FILE
6300            LABEL RECORDS ARE STANDARD.
6400        01  ACC-REC.
6500        COPY DDS-ACCFILE OF ACCTS.
6600
7000
7100       ************************************************************
7200       *             WORKING-STORAGE SECTION                      *
7300       ************************************************************
7400        WORKING-STORAGE SECTION.
7500        01  STATUS-ERR1         PIC XX.
7600        01  STATUS-ERR2         PIC XX.
7700        01  TEMP-PTR USAGE IS POINTER.
7800
7900        01 NUMBERS-1.
8000          03 NUM1          PIC 9(10).
8100          03 NUM2          PIC 9(10).
8200          03 NUM3          PIC 9(10).
8300
8400        01  FEEDBACK-STUFF      PIC X(500) VALUE SPACES.
8500
8600       ************************************************************
8700       * MESSAGE FOR SIGNALLING ANY TRIGGER ERROR                 *
8800       * - Define any  message ID and message file in the following*
8900       *   message data.                                          *
9000       ************************************************************
9100        01 SNDPGMMSG-PARMS.
9200          03 SND-MSG-ID        PIC X(7)    VALUE "TRG9999".
9300          03 SND-MSG-FILE      PIC X(20)   VALUE "MSGF      LIB1    ".
9400          03 SND-MSG-DATA      PIC X(25)   VALUE "Trigger Error".
9500          03 SND-MSG-LEN       PIC 9(8) BINARY  VALUE 25.
9600          03 SND-MSG-TYPE      PIC X(10)   VALUE "*ESCAPE   ".
9700          03 SND-PGM-QUEUE     PIC X(10)   VALUE "*     ".
9800          03 SND-PGM-STACK-CNT PIC 9(8) BINARY  VALUE 1.
9900          03 SND-MSG-KEY       PIC X(4)    VALUE "    ".
10000         03 SND-ERROR-CODE.
10100           05 PROVIDED          PIC 9(8) BINARY VALUE 66.
10200           05 AVAILABLE         PIC 9(8) BINARY VALUE 0.
10300           05 RTN-MSG-ID        PIC X(7) VALUE " ".
10400           05 FILLER            PIC X(1) VALUE " ".
10500           05 RTN-DATA          PIC X(50) VALUE " ".
10600
10700      ************************************************************
10800      *             LINKAGE         SECTION                      *
10900      *  PARM 1 is the trigger buffer                            *
11000      *  PARM 2 is the length of the trigger buffer              *
11100      ************************************************************
11200       LINKAGE SECTION.
11300       01 PARM-1-AREA.
11400         03 FILE-NAME        PIC X(10).
11500         03 LIB-NAME         PIC X(10).
11600         03 MEM-NAME         PIC X(10).
11700         03 TRG-EVENT        PIC X.
11800         03 TRG-TIME         PIC X.
11900         03 CMT-LCK-LVL      PIC X.
12000         03 FILLER           PIC X(3).
12100         03 DATA-AREA-CCSID  PIC 9(8)  BINARY.
12200         03 FILLER           PIC X(8).
12300         03 DATA-OFFSET.
```

```
12400           05 OLD-REC-OFF      PIC 9(8)  BINARY.
12500           05 OLD-REC-LEN      PIC 9(8)  BINARY.
12600           05 OLD-REC-NULL-MAP PIC 9(8)  BINARY.
12700           05 OLD-REC-NULL-LEN PIC 9(8)  BINARY.
12800           05 NEW-REC-OFF      PIC 9(8)  BINARY.
12900           05 NEW-REC-LEN      PIC 9(8)  BINARY.
13000           05 NEW-REC-NULL-MAP PIC 9(8)  BINARY.
13100           05 NEW-REC-NULL-LEN PIC 9(8)  BINARY.
13200           05 FILLER           PIC X(16).
13300         03 RECORD-JUNK.
13400           05 OLD-RECORD       PIC X(16).
13500           05 OLD-NULL-MAP     PIC X(4).
13600           05 NEW-RECORD       PIC X(16).
13700           05 NEW-NULL-MAP     PIC X(4).
13800
13900         01 PARM-2-AREA.
14000         03  TRGBUFL             PIC X(2).
14100
14200         01 INPUT-RECORD2.
14300         COPY DDS-TRANS OF ATMTRANS.
14400
14500         05 OFFSET-NEW-REC2   PIC 9(8)  BINARY.
14600
14700 ****************************************************************
14800 ******                  PROCEDURE DIVISION                  *
14900 ****************************************************************
15000        PROCEDURE DIVISION USING PARM-1-AREA, PARM-2-AREA.
15100        MAIN-PROGRAM SECTION.
15200        000-MAIN-PROGRAM.
15300            OPEN I-O ATM-FILE.
15400            OPEN I-O ACC-FILE.
15500
15600            MOVE 0 TO BAL.
15700
15800       ************************************************************
15900       * SET UP THE OFFSET POINTER AND COPY THE NEW RECORD.        *
16000       ************************************************************
16100            SET TEMP-PTR TO ADDRESS OF PARM-1-AREA.
16200            SET TEMP-PTR UP BY NEW-REC-OFFSET.
16300            SET ADDRESS OF INPUT-RECORD2 TO TEMP-PTR.
16400            MOVE INPUT-RECORD2 TO INPUT-RECORD.
16500
16600       ************************************************************
16700       * READ THE RECORD FROM THE ACCTS FILE                      *
16800       ************************************************************
16900            MOVE ACCTID TO ACCTN.
17000            READ ACC-FILE
17100                 INVALID KEY PERFORM 900-OOPS
17200                 NOT INVALID KEY PERFORM 500-ADJUST-ACCOUNT.
17300
17400       ************************************************************
17500       * READ THE RECORD FROM THE ATMS FILE.                      *
17600       ************************************************************
17700            MOVE ATMID TO ATMN.
17800            READ ATM-FILE
17900                 INVALID KEY PERFORM 950-OOPS
18000                 NOT INVALID KEY PERFORM 550-ADJUST-ATM-BAL.
18100            CLOSE ATM-FILE.
18200            CLOSE ACC-FILE.
18300            GOBACK.
18400
18500 ****************************************************************
18600 ****************************************************************
18700 ****************************************************************
18800 ****************************************************************
18900 ****** THIS PROCEDURE IS USED IF THERE IS NOT ENOUGH MONEY IN THE  ****
19000 ****** ACCTS FOR THE WITHDRAWAL.                                    ****
```

```
19100 ****************************************************************
19200        200-NOT-ENOUGH-IN-ACC.
19300            DISPLAY "NOT ENOUGH MONEY IN ACCOUNT.".
19400            CLOSE ATM-FILE.
19500            CLOSE ACC-FILE.
19600            PERFORM 999-SIGNAL-ESCAPE.
19700            GOBACK.
19800
19900 ****************************************************************
20000 ****** THIS PROCEDURE IS USED IF THERE IS NOT ENOUGH MONEY IN THE
20100 ****** ATMS FOR THE WITHDRAWAL.
20200 ****************************************************************
20300        250-NOT-ENOUGH-IN-ATM.
20400            DISPLAY "NOT ENOUGH MONEY IN ATM.".
20500            CLOSE ATM-FILE.
20600            CLOSE ACC-FILE.
20700            PERFORM 999-SIGNAL-ESCAPE.
20800            GOBACK.
20900
21000 ****************************************************************
21100 ****** THIS PROCEDURE IS USED TO ADJUST THE BALANCE FOR THE ACCOUNT OF
21200 ****** THE PERSON WHO PERFORMED THE TRANSACTION.
21300 ****************************************************************
21400        500-ADJUST-ACCOUNT.
21500            IF TCODE = "W" THEN
21600               IF (BAL < AMOUNT) THEN
21700                  PERFORM 200-NOT-ENOUGH-IN-ACC
21800               ELSE
21900                  SUBTRACT AMOUNT FROM BAL
22000                  REWRITE  ACC-REC
22100            ELSE IF TCODE = "D" THEN
22200               ADD AMOUNT TO BAL
22300               REWRITE  ACC-REC
22400            ELSE DISPLAY "TRANSACTION CODE ERROR, CODE IS: ", TCODE.
22500
22600 ****************************************************************
22700 ****** THIS PROCEDURE IS USED TO ADJUST THE BALANCE OF THE ATM FILE ***
22800 ****** FOR THE AMOUNT OF MONEY IN ATM AFTER A TRANSACTION.          ***
22900 ****************************************************************
23000        550-ADJUST-ATM-BAL.
23100            IF TCODE = "W" THEN
23200               IF (ATMAMT < AMOUNT) THEN
23300                  PERFORM 250-NOT-ENOUGH-IN-ATM
23400               ELSE
23500                  SUBTRACT AMOUNT FROM ATMAMT
23600                  REWRITE  ATM-REC
23700            ELSE IF TCODE = "D" THEN
23800               ADD AMOUNT TO ATMAMT
23900               REWRITE  ATM-REC
24000            ELSE DISPLAY "TRANSACTION CODE ERROR, CODE IS: ", TCODE.
24100
24200 ************************************************************  *******
24300 ****** THIS PROCEDURE IS USED IF THERE THE KEY VALUE THAT IS USED IS **
24400 ****** NOT FOUND IN THE ACCTS FILE.                                  **
24500 ****************************************************************
24600        900-OOPS.
24700            DISPLAY "INVALID KEY: ", ACCTN, "  ACCOUNT FILE STATUS: ",
24800                     STATUS-ERR1.
24900            CLOSE ATM-FILE.
25000            CLOSE ACC-FILE.
25100            PERFORM 999-SIGNAL-ESCAPE.
25200            GOBACK.
25300
25400 ****************************************************************
25500 ****** THIS PROCEDURE IS USED IF THERE THE KEY VALUE THAT IS USED IS **
25600 ****** NOT FOUND IN THE ATM FILE.                                    **
25700 ****************************************************************
```

```
25800        950-OOPS.
25900            DISPLAY "INVALID KEY: ", ATMN, "  ATM FILE STATUS: ",
26000                     STATUS-ERR2.
26100            CLOSE ATM-FILE.
26200            CLOSE ACC-FILE.
26300            PERFORM 999-SIGNAL-ESCAPE.
26400            GOBACK.
26500
26600 ****************************************************************
26700 ****** SIGNAL ESCAPE TO THE APPLICATION                 ********
26800 ****************************************************************
26900        999-SIGNAL-ESCAPE.
27000
27100            CALL "QMHSNDPM" USING SND-MSG-ID,
27200                                  SND-MSG-FILE,
27300                                  SND-MSG-DATA,
27400                                  SND-MSG-LEN,
27500                                  SND-MSG-TYPE,
27600                                  SND-PGM-QUEUE,
27700                                  SND-PGM-STACK-CNT,
27800                                  SND-MSG-KEY,
27900                                  SND-ERROR-CODE.
28000       *DISPLAY RTN-MSG-ID.
28100       *DISPLAY RTN-DATA.
28200
```

更新トリガー・プログラムによって ATMTRANS ファイルから更新が行われた後では、ATMS ファイルおよび ACCTS ファイルには次のようなデータが入っています。ATMID 10002 への更新は、口座に十分な金額がないため失敗します。

ATMN	LOCAT	ATMAMT
10001	MN	275.00
10002	MN	750.00
10003	CA	750.00

ACCTN	BAL	ACTACC
20001	175.00	A
20002	350.00	A
20003	500.00	C

例: ILE C で書かれた削除トリガー: コードの例については、 258 ページの『コードについての特記事項』を参照してください。

以下の ILE C トリガー・プログラムは、ATMTRANS ファイルでレコードが削除されるときに呼び出されます。

```
/*************************************************************/
/* Program Name - DELTRG                                     */
/* This program is called when a delete operation occurs in  */
/* the ATMTRANS file.                                        */
/*                                                           */
/* This program will delete the records from ATMS and ACCTS  */
/* based on the ATM ID and ACCT ID that are passed in from   */
/* the trigger buffer.                                       */
/*                                                           */
/* The application will delete ATMID 10003 from the ATMTRANS */
/* file.                                                     */
/*                                                           */
/*************************************************************/
#include <stdio.h>
```

```
#include <stdlib.h>
#include <recio.h>
#include "applib/csrc/msghandler" /* message handler include       */
#include "qsysinc/h/trgbuf"       /* trigger buffer include without*/
                                  /* old and new records           */
Qdb_Trigger_Buffer_t *hstruct;      /* pointer to the trigger buffer */
char *datapt;

#define KEYLEN 5

/*************************************************************/
/* Need to define file structures here since there are non-   */
/* character fields in each file. For each non-character      */
/* field, C requires boundary alignment. Therefore, a _PACKED */
/* struct should be used in order to access the data that     */
/* is passed to the trigger program.                          */
/*                                                            */
/*************************************************************/

/** record area for ATMTRANS                **/
_Packed struct rec {
              char atmid[5];
              char acctid[5];
              char tcode[1];
              char amount[5];
                   } oldbuf, newbuf;

/** record area for ATMS                    **/
_Packed struct rec1{
              char atmn[5];
              char locat[2];
              char atmamt[9];
                   } atmfile;

/** record area for ACCTS                   **/
_Packed struct rec2{
              char acctn[5];
              char bal[9];
              char actacc[1];
                   } accfile;


/*********************************************************************/
/*********************************************************************/
/* Start of the Main Line Code.  *************************************/
/*********************************************************************/
/*********************************************************************/
main(int argc, char **argv)
{
_RFILE  *out1;                /* file pointer for ATMS        */
_RFILE  *out2;                /* file pointer for ACCTS       */
_RIOFB_T *fb;                 /* file feedback pointer        */
char record[16];             /* record buffer            */
_FEEDBACK fc;                 /* feedback for message handler */
_HDLR_ENTRY hdlr = main_handler;
                              /*********************************/
                              /* active exception handler      */
                              /*********************************/
CEEHDLR(&hdlr, NULL, &fc);;
                              /*********************************/
                              /* ensure exception handler OK   */
                              /*********************************/
if (fc.MsgNo != CEE0000)
 {
   printf("Failed to register exception handler.¥n");
   exit(99);
 }
```

```
/* set pointer to the input parameter     */
hstruct = (Qdb_Trigger_Buffer_t *)argv[1];
datapt  = (char *) hstruct;

/* Copy old and new record from the input parameter  */

 if ((strncmp(hstruct ->trigger_event,"2",1)== 0)|| /* delete event */
     (strncmp(hstruct -> trigger_event,"3",1)== 0))    /* update event */
   { obufoff = hstruct ->old_record_offset;
     memcpy(&oldbuf,datapt+obufoff,; hstruct->old_record_len);
   }
 if ((strncmp(hstruct -> trigger_event,"1",1)== 0) ||  /* insert event */
     (strncmp(hstruct -> trigger_event,"3",1)== 0))    /* update event */
   { nbufoff = hstruct ->new_record_offset;
     memcpy(&newbuf,datapt+nbufoff,; hstruct->new_record_len);
   }

/*****************************************************/
/* Open ATM and ACCTS    files                       */
/*                                                   */
/* Check the application's commit lock level. If it  */
/* runs under commitment control, then open both     */
/* files with commitment control. Otherwise, open    */
/* both files without commitment control.            */
/*****************************************************/
if(strcmp(hstruct->commit_lock_level,"0") == 0)    /* no commit    */
 {
   if ((out1=_Ropen("APPLIB/ATMS","rr+")) == NULL)
     {
       printf("Error opening ATM file");
   exit(1);
     }
   if ((out2=_Ropen("APPLIB/ACCTS","rr+")) == NULL)
     {
       printf("Error opening ACCTS file");
   exit(1);
     }
  }
 else                  /* with commitment control   */
 {
   if ((out1=_Ropen("APPLIB/ATMS","rr+,commit=Y")) == NULL)
     {
       printf("Error opening ATMS file");
   exit(1);
     }
   if ((out2=_Ropen("APPLIB/ACCTS","rr+,commit=Y")) == NULL)
     {
       printf("Error opening ACCTS file");
   exit(1);
     }
  }

/* Delete the record based on the input parameter  */
fb =_Rlocate(out1,&oldbuf.atmid,KEYLEN,__DFT);
if (fb->num_bytes != 1)
 {
   printf("record not found in ATMS¥n");
 _Rclose(out1);
   exit(1);
 }
 _Rdelete(out1);         /* delete record from ATMS */
 _Rclose(out1);

fb =_Rlocate(out2,&oldbuf.acctid,KEYLEN,__DFT);
if (fb->num_bytes != 1)
 {
```

```
   printf("record not found in ACCOUNTS¥n");
 _Rclose(out2);
   exit(1);
 }
 _Rdelete(out2);         /* delete record from ACCOUNTS  */
 _Rclose(out2);

 } /* end of main */
```

アプリケーションによる削除の後では、ATMTRANS ファイルには次のようなデータが入っています。

ATMID	ACCTID	TCODE	AMOUNT
10001	20001	W	25.00
10002	20002	W	900.00

削除トリガー・プログラムによって ATMTRANS ファイルから削除された後では、ATMS ファイルおよび ACCTS ファイルには次のようなデータが入っています。

ATMN	LOCAT	ATMAMT
10001	MN	275.00
10002	MN	750.00

ACCTN	BAL	ACTACC
20001	175.00	A
20002	350.00	A

```
/*********************************************************************/
/*   INCLUDE NAME : MSGHANDLER                                       */
/*                                                                   */
/*   DESCRIPTION  : Message handler to signal an exception message*/
/*                  to the caller of this trigger program.         */
/*                                                                   */
/*   Note: This message handler is a user defined routine.           */
/*                                                                   */
/*********************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <recio.h>
#include <leawi.h>

#pragma linkage (QMHSNDPM, OS)
void QMHSNDPM(char *,             /* Message identifier             */
              void *,             /* Qualified message file name    */
              void *,             /* Message data or text           */
              int,                /* Length of message data or text */
              char *,             /* Message type                   */
              char *,             /* Call message queue             */
              int,                /* Call stack counter             */
              void *,             /* Message key                    */
              void *,             /* Error code                     */
              ...);               /* Optionals:
                                       length of call message queue
                                        name
                                      Call stack entry qualification
                                      display external messages
                                       screen wait time              */
/*********************************************************************/
/********  This is the start of the exception handler function.      */
/*********************************************************************/
  void main_handler(_FEEDBACK *cond, _POINTER *token, _INT4 *rc,
```

```
                    _FEEDBACK *new)
    {
                            /****************************************/
                            /* Initialize variables for call to     */
                             /* QMHSNDPM.                            */
                            /* User defines any message ID and      */
                            /* message file for the following data  */
                            /****************************************/
   char      message_id[7] = "TRG9999";
   char      message_file[20] = "MSGF      LIB1      ";
   char      message_data[50] = "Trigger error               ";
   int       message_len = 30;
   char      message_type[10] = "*ESCAPE   ";
   char      message_q[10] = "_C_pep    ";
   int       pgm_stack_cnt = 1;
   char      message_key[4];
                            /****************************************/
                            /* Declare error code structure for     */
                            /* QMHSNDPM.                            */
                            /****************************************/
   struct error_code {
     int bytes_provided;
     int bytes_available;
     char message_id[7];
   } error_code;

   error_code.bytes_provided = 15;
                            /****************************************/
                            /* Set the error handler to resume and  */
                            /* mark the last escape message as      */
                            /* handled.                             */
                            /****************************************/
   *rc = CEE_HDLR_RESUME;
                            /****************************************/
                            /* Send my own *ESCAPE message.         */
                            /****************************************/
   QMHSNDPM(message_id,
            &message_file,
            &message_data,
            message_len,
            message_type,
            message_q,
            pgm_stack_cnt,
            &message_key,
            &error_code );
                            /****************************************/
                            /* Check that the call to QMHSNDPM      */
                            /* finished correctly.                  */
                            /****************************************/
  if (error_code.bytes_available != 0)
      {
        printf("Error in QMHOVPM : %s\n", error_code.message_id);
      }
   }

/*****************************************************************/
/*   INCLUDE NAME : TRGBUF                                       */
/*                                                               */
/*   DESCRIPTION  : The input trigger buffer structure for the   */
/*                  user's trigger program.                      */
/*                                                               */
/*   LANGUAGE     : ILE C                                        */
/*                                                               */
/*****************************************************************/
/*****************************************************************/
/*   Note: The following type definition only defines the fixed  */
/*         portion of the format. The data area of the original  */
/*         record, null byte map of the original record, the     */
```

```
/*         new record, and the null byte map of the new record  */
/*         is varying length and immediately follows what is    */
/*         defined here.                                        */
/************************************************************/
 typedef _Packed struct Qdb_Trigger_Buffer {
           char  file_name[10];
           char  library_name[10];
           char  member_name[10];
           char  trigger_event[1];
           char  trigger_time[1];
           char  commit_lock_level[1];
           char  reserved_1[3];
           int   data_area_ccsid;
           char  reserved_2]8];
           int   old_record_offset;
           int   old_record_len;
           int   old_record_null_byte_map;
           int   old_record_null_byte_map_len;
           int   new_record_offset;
           int   new_record_len;
           int   new_record_null_byte_map;
           int   new_record_null_byte_map_len;
                 } Qdb_Trigger_Buffer_t;
```

トリガー・プログラム: 例で使用されるデータベースのデータ構造: このアプリケーションで使用するデータ構造は、以下のとおりです。

- ATMTRANS : /* Transaction record */

```
ATMID      CHAR(5) (KEY)  /* ATM** machine ID number     */
ACCTID     CHAR(5)        /* Account number            */
TCODE      CHAR(1)        /* Transaction code          */
AMOUNT     ZONED          /* Amount to be deposited or */
                          /* withdrawn                 */
```

- ATMS : /* ATM machine record */

```
ATMN       CHAR(5)  (KEY) /* ATM machine ID number     */
LOCAT      CHAR(2)        /* Location of ATM           */
ATMAMT     ZONED          /* Total amount in this ATM  */
                          /* machine                   */
```

ATMN	LOCAT	ATMAMT
10001	**MN**	**200.00**
10002	**MN**	**500.00**
10003	**CA**	**250.00**

- ACCTS: /* Accounting record */

```
ACCTN      CHAR(5)  (KEY) /* Account number            */
BAL        ZONED          /* Balance of account        */
ACTACC     CHAR(1)        /* Status of Account         */
```

ACCTN	BAL	ACTACC
20001	**100.00**	**A**
20002	**100.00**	**A**
20003	**0.00**	**C**

トリガー・バッファー・セクション: トリガー・バッファーには、静的セクションと可変セクションの 2 つの論理セクションがあります。

- 静的セクションには、以下のものが入っています。

– 現行レコードと相対レコード番号の物理ファイル名、メンバー名、トリガー・イベント、トリガー時刻、コミット・ロック・レベル、および CCSID が入っているトリガー・テンプレート。
– レコード域および NULL バイト・マップのオフセットおよび長さ

このセクションは、0 から 95 までのオフセットを占めます (10 進数)。

可変セクションには、以下のものが入っています。

- 古いレコード、古い NULL バイト・マップ、新しいレコード、および新しい NULL バイト・マップ用の区域

次の表には、トリガー・バッファー内のフィールドの要約が示されています。これらのフィールドの詳細については、『トリガー・バッファーのフィールドの説明』を参照してください。

オフセット		タイプ	フィールド
10 進数	**16 進数**		
0	0	CHAR(10)	物理ファイル名
10	A	CHAR(10)	物理ファイルのライブラリー名
20	14	CHAR(10)	物理ファイル・メンバー名
30	1E	CHAR(1)	トリガー事象
31	1F	CHAR(1)	トリガー時間
32	20	CHAR(1)	コミット・ロック・レベル
33	21	CHAR(3)	予約済み
36	24	BINARY(4)	データの CCSID
40	28	BIN(4)	相対レコード番号
44	2C	CHAR(4)	予約済み
48	30	BINARY(4)	元のレコードのオフセット
52	34	BINARY(4)	元のレコードの長さ
56	38	BINARY(4)	元のレコードの NULL バイト・マップのオフセット
60	3C	BINARY(4)	元のレコードの NULL バイト・マップの長さ
64	40	BINARY(4)	新しいレコードのオフセット
68	44	BINARY(4)	新しいレコードの長さ
72	48	BINARY(4)	新しいレコードの NULL バイト・マップのオフセット
76	4C	BINARY(4)	新しいレコードの NULL バイト・マップの長さ
80	50	CHAR(*)	予約済み
*	*	CHAR(*)	元のレコード
*	*	CHAR(*)	元のレコードの NULL バイト・マップ
*	*	CHAR(*)	新しいレコード
*	*	CHAR(*)	新しいレコードの NULL バイト・マップ

トリガー・バッファーのフィールドの説明: 以下のリストには、トリガー・バッファーに含まれているフィールドが英字順で示されています。

データの CCSID. 新規レコードまたは元のレコードのデータの CCSID。データは、データベースによってジョブの CCSID へ変換されます。SBCS データは、単一バイトに関連した CCSID に変換されます。DBCS データは、2 バイトに関連した CCSID に変換されます。

コミット・ロック・レベル. 現在のアプリケーション・プログラムのコミット・ロック・レベル。指定できる値は以下のとおりです。

'0' *NONE

'1' *CHG

'2' *CS

'3' *ALL

新しいレコード. 変更操作の結果として、物理ファイル内で挿入または更新されるレコードのコピー。挿入操作または更新操作には、新しいレコードのみが適用されます。

新しいレコードの長さ. 最大長は、32766 バイトです。

新しいレコードの NULL バイト・マップ. この構造には、新しいレコードの各フィールドの NULL 情報が入っています。各バイトが、1 つのフィールドを表します。各バイトに指定できる値は、以下のとおりです。

'0' 非 NULL

'1' NULL

新しいレコードの NULL バイト・マップの長さ. 長さは、物理ファイル内のフィールド数に等しくなります。

新しいレコードの NULL バイト・マップのオフセット. 新しいレコードの NULL バイト・マップの位置。オフセット値は、トリガー・バッファーの先頭からの値です。レコードの新しい値が、たとえば削除操作などの変更操作に適用されない場合には、このフィールドは該当しません。

新しいレコードのオフセット. 新しいレコードの位置。オフセット値は、トリガー・バッファーの先頭からの値です。レコードの新しい値が、たとえば削除操作などの変更操作に適用されない場合には、このフィールドは該当しません。

元のレコード. 更新、削除、または読み取り前の元の物理レコードのコピー。元のレコードは、更新操作、削除操作、および読み取り操作にのみ適用されます。

元のレコードの長さ. 最大長は、32766 バイトです。

元のレコードの NULL バイト・マップ. この構造には、元のレコードの各フィールドの NULL 情報が入っています。各バイトが、1 つのフィールドを表します。各バイトに指定できる値は、以下のとおりです。

'0' 非 NULL

'1' NULL

元のレコードの NULL バイト・マップの長さ. 長さは、物理ファイル内のフィールド数に等しくなります。

元のレコードの NULL バイト・マップのオフセット. 元のレコードの NULL バイト・マップの位置。オフセット値は、トリガー・バッファーの先頭からの値です。レコードの元の値が、たとえば挿入操作などの変更操作に適用されない場合には、このフィールドは該当しません。

元のレコードのオフセット. 元のレコードの位置。オフセット値は、トリガー・バッファーの先頭からの値です。レコードの元の値が、たとえば挿入操作などの操作に適用されない場合には、このフィールドは該当しません。

物理ファイルのライブラリー名. 物理ファイルが常駐するライブラリー名。

物理ファイル・メンバー名. 物理ファイル・メンバー名。

物理ファイル名. 変更される物理ファイル名。

相対レコード番号. 更新または削除されるレコードの相対レコード番号 (*BEFORE トリガー)、あるいは、挿入、更新、削除、または読み取られたレコードの相対レコード番号 (*AFTER トリガー)。

トリガー事象. トリガー・プログラムが呼び出される原因となった事象。指定できる値は以下のとおりです。

'1' 挿入操作

'2' 削除操作

'3' 更新操作

'4' 読み取り操作

トリガー時刻. 物理ファイルに対して行われる操作に関して、トリガー・プログラムが呼び出される時刻を指定します。指定できる値は以下のとおりです。

'1' 変更操作または読み取り操作後

'2' 変更操作前

トリガー・プログラムに関する推奨事項: 以下では、トリガー・プログラムにおける推奨事項を説明します。

- 作成したユーザーのプロファイルの下で実行されるようにトリガー・プログラムを作成する。この方法では、そのプログラムへの同じレベルの権限を持たないユーザーに対してエラーが発生しません。
- USRPRF(*OWNER) および *EXCLUDE 共通認可を使用してプログラムを作成し、トリガー・プログラムに対する権限を USER(*PUBLIC) に認可しない。他のユーザーが、トリガー・プログラムの変更および置き換えをしないようにします。トリガー・プログラムを実行したユーザーが、そのトリガー・プログラムに対する権限を持っていてもいなくても、データベースは、トリガー・プログラムを呼び出します。
- プログラムが ILE 環境で実行される場合は、プログラムを ACTGRP(*CALLER) として作成する。これによって、トリガー・プログラムを、アプリケーションと同じコミットメント定義の下で実行できます。
- ファイルを、アプリケーションのコミット・ロック・レベルと同じコミット・ロック・レベルでオープンする。これによって、トリガー・プログラムを、アプリケーションと同じコミット・ロック・レベルの下で実行できます。
- プログラムを物理ファイルのライブラリー内に作成する。
- トリガー・プログラムが、アプリケーションとは異なる活動化グループの下で実行される場合は、トリガー・プログラムではコミットまたはロールバックを使用する。
- トリガー・プログラム内でエラーが起きたり、検出された場合は、例外を出す。エラー・メッセージがトリガー・プログラムから出されないと、データベースは、トリガーは正常に実行されたものとみなします。この場合、ユーザー・データが一貫していない状態で終了する可能性があります。

トリガー・プログラムのコーディング時の指針: トリガー・プログラムは、非常に強力なものになる場合があります。テープ装置などのシステム資源にアクセスするトリガー・プログラムの場合、設計に気を付ける必要があります。たとえば、レコード変更をテープ媒体に複写するトリガー・プログラムは便利ですが、プログラムそのものは、磁気テープ駆動機構が作動可能かどうか、またはそれに正しいデータが入っているかどうかを検出することはできません。トリガー・プログラムを設計する際、この種の資源側の問題を考慮に入れなければなりません。

さらに、読み取りトリガーを使用する場合には、特別な注意が必要です。読み取りトリガーを使用すると、読み取られるすべてのレコードごとにトリガーが呼び出される可能性があります。つまり、照会では、レコードが複数回にわたって処理され、トリガーが何度も呼び出される可能性があります。これは、システム・パフォーマンスに影響する可能性があります。

トリガー・プログラムについては、以下の関連トピックを参照してください。

- 『トリガー・プログラム内での使用に注意を要する機能』
- 『トリガー・プログラム内で使用できないコマンド、ステートメント、および操作』

トリガー・プログラム内での使用に注意を要する機能: 以下の CL コマンドと機能については、慎重に検討する必要があります。これらは、トリガー・プログラムにはお勧めできません。

- STRCMTCTL (コミットメント制御の開始)
- RCLSPLSTG (スプール・ストレージの再利用)
- RCLRSC (資源再利用)
- CHGSYSLIBL (システム・ライブラリー・リストの変更)
- DLTLICPGM、RSTLICPGM、および SAVLICPGM (ライセンス・プログラムの削除、復元、および保管)
- (*NO) 以外の SAVACT を指定した SAVLIB (ライブラリーの保管)
- DKT または TAP を指定された全コマンド
- すべての移行コマンド
- デバッグ・プログラム (機密保護エクスポージャー)
- 遠隔ジョブ入力 (RJE) に関連した全コマンド
- 別の CL または対話式項目の呼び出し − ロック資源限界に達することがあります。

トリガー・プログラム内で使用できないコマンド、ステートメント、および操作: トリガー・プログラムに、以下のようなコマンド、ステートメント、および操作を含めることはできません。これらを使用すると、システムが例外を返します。

- トリガーを呼び出した挿入、更新、削除、または読み取り操作に関連したコミットメント定義では、COMMIT 操作を行うことはできません。COMMIT 操作を、ジョブ内の他のコミットメント定義に使うことはできます。
- トリガーを呼び出した挿入、更新、削除、または読み取り操作に関連したコミットメント定義では、ROLLBACK 操作を行うことはできません。ROLLBACK 操作を、ジョブ内の他のコミットメント定義に使うことはできます。
- SQL CONNECT、DISCONNECT、SET CONNECTION、および RELEASE ステートメントは使用できません。
- トリガーを呼び出した挿入、更新、削除、または読み取り操作に関連したコミットメント定義では、ENDCMTCTL CL コマンドを使用することはできません。ENDCMTCTL CL コマンドを、ジョブ内の他のコミットメント定義に使うことはできます。
- ローカル API コミットメント資源 (QTNADDCR) を、トリガーを呼び出した挿入、更新、削除、または読み取り操作に関連した同一のコミットメント定義に追加することはできません。
- *SHARE を指定してトリガー・プログラムがオープンしたファイルであり、かつトリガー・プログラムを呼び出す原因となったファイルに対する入出力操作は無効です。
- トリガーを呼び出した挿入、更新、削除、および読み取り操作と同じコミットメント定義を使用し、すでに既存の遠隔資源を持っている、起動されたトリガー・プログラム。ただし、システムは以下の場合に、トランザクション全体をロールバックが必要な状態にします。
  - トリガー・プログラムが失敗し、エスケープ・メッセージ AND を出した場合。
  - iSeries 以外のロケーション、またはバージョン 3 リリース 2 以前のレベルにあるロケーションのための 1 次以外のコミット・サイクル中に、いずれかの遠隔資源が更新された場合。

- トリガー・プログラムは、トリガーを呼び出した挿入、更新、削除、または読み取り操作に関連したコミットメント定義に、遠隔資源を追加できます。この場合、LU 6.2 遠隔資源 (保護会話) および DFM 遠隔資源 (DDM ファイルのオープン) は追加できますが、 DRDA® 遠隔資源は追加できません。
- トリガー・プログラムから遠隔資源を変更しているときに障害が起きた場合は、トリガー・プログラムは、エスケープ・メッセージを出して終了しなければなりません。このようにして、すべての遠隔場所でのトランザクション全体を、システムが正しくロールバックできるようになっています。トリガー・プログラムがエスケープ・メッセージとともに終了しないと、遠隔場所ごとのデータベースに一貫性がなくなることがあります。
- アプリケーション・プログラムのコミット・ロック・レベルは、トリガー・プログラムに渡されます。トリガー・プログラムを、アプリケーション・プログラムと同じロック・レベルで実行してください。
- トリガー・プログラムとアプリケーション・プログラムは、同じ活動化グループ内で稼働することも、別個の活動化グループ内で稼働することもできます。トリガー・プログラムとアプリケーション・プログラムの間に一貫性をもたせるため、 ACTGRP(*CALLER) を使用してトリガー・プログラムをコンパイルしてください。
- トリガー・プログラムは、他のプログラムを呼び出すことも、ネストすることもできます (つまり、トリガー・プログラム内のステートメントで、別のトリガー・プログラムを呼び出す)。さらに、トリガー・プログラムは、自身もトリガー・プログラムを呼び出すことがあります。挿入、更新、削除、または読み取りの場合のトリガーの最大ネスト・レベルは 200 です。トリガー・プログラムがコミットメント制御の下で実行されている場合、以下の状態になっていると、エラーが発生します。
  - 変更操作によって、またはトリガー・プログラムでの操作によってすでに変更されている同一レコードを更新する。
  - 1 回の変更操作中に、同一レコードに対して相反する操作を行う。たとえば、変更操作でレコードを挿入してから、そのレコードをトリガー・プログラムで削除するなど。

  注：

  1. 変更操作がコミットメント制御下で実行されていない場合は、システムが常に変更操作を保護します。しかし、トリガー・プログラムでの同一のレコードの更新はモニターされません。
  2. 物理ファイル・トリガーの追加 (ADDPFTRG) コマンドの ALWREPCHG(*NO|YES) パラメーターは、コミットメント制御の下で実行される変更の繰り返しを制御します。省略時の値を ALWREPCHG(*YES) に変更すれば、同一のレコードまたはトリガー・プログラムに関連した更新レコードを繰り返し変更できます。
- 物理ファイル・トリガーの追加 (ADDPFTRG) コマンドの繰り返し変更可能 ALWREPCHG(*YES) パラメーターも、データベースの挿入操作および更新操作の前に呼び出されるように定義されたトリガー・プログラムに影響を及ぼします。トリガー・プログラムによりトリガー・バッファーの新規レコードが更新され、かつ ALWREPCHG(*YES) が指定される場合、修正された新規レコード・イメージは関連した物理ファイルで挿入操作または更新操作を実際に行うために使用されます。このオプションは、データの妥当性検査および訂正用に設計されたトリガー・プログラムで役に立ちます。トリガー・プログラムは、物理ファイル・レコード・イメージ (論理ファイルの場合でも) を受け取るので、そのレコード・イメージのどのフィールドでも変更できます。
- トリガー・プログラムは、物理ファイル内で変更される各行、または、物理ファイルから読み取られる各行ごとに呼び出されます。
- 物理ファイルまたは従属の論理ファイルが挿入 SEQONLY(*YES) 処理のためにオープンされていても、物理ファイルが挿入トリガー・プログラムに関連付けられていれば、変更される各行ごとにトリガー・プログラムが呼び出されるよう、システムはそのオープンを SEQONLY(*NO) に変更します。

コミットメント制御の下で実行されるトリガーおよびアプリケーション・プログラム: トリガー・プログラムとアプリケーション・プログラムを同じコミットメント定義の下で実行するときは、トリガー・プログラムに障害が起きると、そのトリガー・プログラムに関連したすべてのステートメントはロールバックされます。これには、ネストされたトリガー・プログラム中のすべてのステートメントが含まれます。元の変更操作もロールバックされます。この場合、トリガー・プログラムがエラーを検出したときに、例外を発信する必要があります。

トリガー・プログラムとアプリケーション・プログラムが別々のコミットメント定義の下で実行されるときには、アプリケーション・プログラム中の COMMIT ステートメントのみが、自らのコミットメント定義に影響を及ぼすだけです。プログラマーは、COMMIT ステートメントを発行してトリガー・プログラム内の変更をコミットしなければなりません。

挿入記録操作または更新記録操作がコミットメント制御の下で実行されるときは、特定の複写キー・エラーの検出は、その操作の論理終了まで据え置かれます。これは、論理終了時までにその種のエラーが解決される可能性があるからです。呼び出し側プログラムと同じコミットメント定義内でトリガー・プログラムが実行されている場合には、 1 つの、またはブロック化された挿入、更新、または削除レコード操作が呼び出し側プログラムによって実行された後、操作の論理が終了し、呼び出された変更前または変更後トリガー・プログラムから制御が戻されます。その結果として、トリガー・プログラムを呼び出した挿入、更新、または削除レコード操作と同じコミットメント定義を使用しているトリガー・プログラムでは複写キー・エラーは検出不可能になります。

コミットメント制御の下で実行されないトリガーおよびアプリケーション・プログラム: 両方のプログラムともコミットメント制御下で実行されていない場合は、トリガー・プログラムにエラーが生じると、ファイルはそのエラーの発生時のままの状態になります。ロールバックは行われません。

トリガー・プログラムがコミットメント制御の下ではなく、アプリケーション・プログラムがコミットメント制御の下で実行されている場合、次のいずれかの時点で、トリガー・プログラムからのすべての変更がコミットされます。

- トリガー・プログラム内でコミット操作が実行されるとき。
- 活動化グループの終了時。通常の場合、暗黙コミットは、活動化グループが終了するときにコミットされます。ただし、システムに異常障害が起きた場合は、ロールバックが行われます。

トリガー・プログラムのエラー・メッセージ: トリガー・プログラムの実行中に障害が起きた場合は、プログラムは、該当するエスケープ・メッセージを出してから終了しなければなりません。そうしなければ、トリガー・プログラムが正常に実行されたとアプリケーションがみなしてしまいます。メッセージは、システムから出される独自のメッセージでも、またはトリガー・プログラムの作成者によって作成されたメッセージでもかまいません。

トリガー・プログラム使用のモニター: DB2 UDB for iSeriesによって、トリガー・プログラムをデータベースのファイルに関連付けることができます。 トリガー・プログラム機能は、あらゆる業界で使用できる、高度なデータベース管理機能です。

トリガー・プログラムをデータベースのファイルと関連付ける場合、トリガー・プログラムをいつ起動するかを指定します。たとえばファイルに新しいレコードが追加された時には、必ず顧客順のファイルがトリガー・プログラムを起動するように設定できます。顧客の未払い残高が与信限度額を超えた時は、トリガー・プログラムは顧客に対して警告文を印刷し、与信責任者にメッセージを送ることができます。

トリガー・プログラムは、アプリケーション機能を提供し、情報を管理する上で生産的な方法です。またトリガー・プログラムを使用して、システムに『トロイの木馬』を作り、仕掛けをしかけることも可能です。システムのデータベース・ファイルにある一定のイベントが発生すると、有害なプログラムが起動しようと待ち構えていることがあります。

注: 歴史のなかで、トロイの木馬というのは巨大で中が空洞の木馬で、中には大勢のギリシャ兵が隠れていました。木馬がトロイの壁の内側に入れられると、兵士たちが木馬の中から出てきてトロイ人と闘ったのです。コンピューターの世界では、有害な機能が潜んだプログラムのことを、よくトロイの木馬とよびます。

システムが出荷されるときは、データベースのファイルにトリガー・プログラムを追加する機能は制限さています。オブジェクト権限を注意して管理していれば、一般のユーザーがトリガー・プログラムをデータベースのファイルに加える権限を持つことはありません。 *iSeries 機密保護解説書* の付録 D では、物理ファイル・トリガー追加 (ADDPFTRG) コマンドなど、あらゆるコマンドに必要な権限について説明しています。

トリガー・プログラムの印刷 (PRTTRGPGM) コマンドを使用して、特定のライブラリーや全ライブラリーの中のすべてのトリガー・プログラムのリストを印刷できます。以下は、報告書の例です。

トリガー・プログラム（全報告書）

```
指定したライブラリー . . . . . :   CUSTLIB
                                ---------------------------- トリガー -----------------------------
ライブラリー ファイル    タイプ ライブラリー プログラム     時刻         事象         条件
 CUSTLIB      MB106     *SYS    ARPGMLIB    INITADDR      前           更新         常に
 CUSTLIB      MB107     *SYS    ARPGMLIB    INITNAME      前           更新         常に
```

初期報告書を基準として使い、すでにシステムに存在するあらゆるトリガー・プログラムを評価することができます。さらに、変更した報告書を定期的に印刷して、システムに新しいトリガー・プログラムが追加されていないか調べることができます。

トリガー・プログラムを評価するときは、以下の項目について調べます。

- このトリガー・プログラムの作成者は誰か ? オブジェクト記述の表示 (DSPOBJD) コマンドを使用して、これを調べることができます。
- このプログラムは何を行うのか ? これを調べるには、ソース・プログラムを見たり、プログラム作成者に聞く必要があります。たとえばトリガー・プログラムは、ユーザーが誰であるかを確認しますか ? トリガー・プログラムはシステム資源にアクセスするために、おそらく特定のユーザー (QSECOFR) を待っているはずです。

基準になる情報を作ってしまえば、変更された報告書を定期的に印刷して、システムに新たに追加されたトリガー・プログラムをモニターできます。以下は、変更された報告書の例です。

トリガー・プログラム（変更された報告書）

```
指定したライブラリー . . . . . :   LIBX
最終変更報告書 . . . . . . . . :   03/11/21  14:33:37
                                ---------------------------- トリガー -----------------------------
ライブラリー ファイル    タイプ ライブラリー プログラム     時刻         事象         条件
 INVLIB       MB108     *SYS    INVPGM      NEWPRICE      後           削除         常に
 INVLIB       MB110     *SYS    INVPGM      NEWDSCNT      後           削除         常に
```

ファイルへのトリガーの追加

トリガーを追加するには、以下の手順を実行してください。

1. ユーザーが適切な権限を持っていること、およびファイルが適切なデータ機能を持っていることを確認します。これらの要件については、 279 ページの『トリガーに必要な権限およびデータ機能』を参照してください。

2. トリガー・プログラムを特定の物理ファイルに関連付けるには、以下のいずれかの方法を使用します。
   - iSeries ナビゲーターを使用して、新しい表を作成したり、既存の表のプロパティーを編集する
   - 物理ファイル・トリガーの追加 (ADDPFTRG) コマンドを使用する
   - SQL の CREATE TRIGGER ステートメントを使用する

注: トリガー・プログラムが QTEMP ライブラリーにある場合、そのトリガー・プログラムを物理ファイルに関連付けることはできません。

トリガー・プログラムとファイル間の関連付けが完了すると、対象の物理ファイル、物理ファイルのメンバー、およびその物理ファイルを元として作成された論理ファイルに対して変更操作が開始されたときに、システムはトリガー・プログラムを呼び出します。

1 つの物理ファイルには、最大 300 までのトリガーを関連付けることができます。それぞれの挿入、削除、または更新操作は、その操作が行われる前および行われた後に、複数のトリガーを呼び出すことができます。それぞれの読み取り操作は、その操作が行われた後に、複数のトリガーを呼び出すことができます。

Query によって実行された読み取り操作の後で呼び出されたトリガーの数は、実際に戻されたレコードの数に等しくない場合があります。これは、Query によってさまざまな数のレコードを読み取られ、それぞれの読み取り操作のたびにトリガーが呼び出されてから、レコードの数を戻すからです。

SQL 更新操作では、同時読み取り操作の後に書き込み操作を続けることが必要です。読み取りトリガーは、SQL 更新操作では実行されません。読み取りのすぐ後に書き込み操作を続けるには、更新トリガーを指定しなければなりません。

トリガーに必要な権限およびデータ機能: トリガーを追加するには、以下の権限が必要です。

- ファイルへのオブジェクト管理または更新権限
- ファイルへのオブジェクト操作権
- ファイルへの読み取りデータ権
- CRTPFTRG ALWREPCHG(*YES) が指定されている場合は、ファイルへのデータ更新権およびオブジェクト操作権
- ファイルのライブラリーへの実行権限
- トリガー・プログラムへの実行権限
- トリガー・プログラムのライブラリーへの実行権限

トリガーを追加する前に、ファイルには、適切なデータ機能がなければなりません。

- CRTPF ALWUPD(*NO) は、*UPDATE トリガーと対立する。
- CRTPF ALWDLT(*NO) は、*DELETE トリガーと対立する。

トリガーの表示

ファイル記述の表示 (DSPFD) コマンドを使用すると、ファイルに関連付けられているトリガーのリストを表示できます。このリストを表示するには、TYPE(*TRG) または TYPE(*ALL) を指定します。このコマンドは以下の情報を提供します。

- トリガー・プログラム数
- トリガー名およびライブラリー
- トリガー状況
- トリガー・プログラム名およびライブラリー

- トリガー事象
- トリガー時間
- トリガーの更新条件
- トリガーのタイプ
- トリガー・モード
- トリガーの方位
- トリガーの作成日時
- トリガー更新列の数
- トリガー更新列のリスト

トリガーの除去

物理ファイル・トリガーの除去 (RMVPFTRG) コマンドを使用して、ファイルとトリガー・プログラムのアソシエーションを除去します。アソシエーションが除去されると、物理ファイルに対して変更操作または読み取り操作が行われても、システムは何も処置を行いません。ただしトリガー・プログラムは、システム上に残ります。

また、 iSeries ナビゲーターを使用してトリガーを除去することもできます。詳しくは、 SQL プログラミングの中の、 iSeries ナビゲーターを初めて使用される場合を参照してください。

トリガーを使用可能または使用不可にする

物理ファイル・トリガーの変更 (CHGPFTRG) コマンドを使用して、指定したトリガーを使用可能または使用不可にする、あるいは、ファイルのすべてのトリガーを使用可能または使用不可にします。トリガーを使用不可にすると、物理ファイルに対して変更操作が行われても、トリガー・プログラムが呼び出されなくなります。トリガーを使用可能にすると、物理ファイルに対して変更操作が行われるときに、再びトリガー・プログラムが呼び出されるようになります。

iSeries ナビゲーターを使用して、トリガーを使用可能または使用不可にすることもできます。詳しくは、SQL プログラミングの中の、 iSeries ナビゲーターを初めて使用される場合を参照してください。

トリガーおよびその他の iSeries 機能との関係

トリガーは、以下の方法でシステムと相互作用します。

基本ファイルの保管/復元 (SAVOBJ/RSTOBJ)

- 保管/復元機能は、保管/復元時にトリガー・プログラムを検索しません。プログラムを管理するのは、ユーザーの責任です。実行時にトリガー・プログラムが復元されていない場合には、システムはトリガー・プログラム名、物理ファイル名、およびトリガー事象で重大エラーを戻します。
- ライブラリー (*ALL) 全体が保管され、物理ファイルおよびすべてのトリガー・プログラムが同じライブラリー内にあって、それぞれ別々のライブラリーに復元された場合は、物理ファイル内のすべてのトリガー・プログラム名は、新しいライブラリーを反映するよう変更されます。

トリガー・プログラムの保管/復元 (SAVOBJ/RSTOBJ)

- トリガー・プログラムを異なるライブラリーに復元すると、そのトリガー・プログラムは元のライブラリーにないため、変更操作は失敗します。トリガー・プログラム名、物理ファイル名、およびトリガー事象での重大エラーが戻されます。

  このような状態から回復するには、2 つの方法があります。

  – トリガー・プログラムを同じライブラリーに復元する。

– 新しいライブラリーに、同じ名前で新しいトリガー・プログラムを作成する。

ファイルの削除 (DLTF)

- トリガー・プログラムと削除済みファイルの間の関連は除去されます。トリガー・プログラムは、システムに残ります。

ファイルのコピー

- 受取先ファイルが挿入トリガーと関連付けられている場合には、それぞれの挿入レコードがトリガー・プログラムを呼び出します。
- 受け取り先ファイルが削除トリガー・プログラムと関連付けられていて、CPYF コマンドに MBROPT(*REPLACE) が指定されている場合は、コピー操作は失敗します。
- CREATE(*YES) を使用した場合は、トリガー情報をコピーしません。

重複オブジェクトの作成 (CRTDUPOBJ)

- 物理ファイルとそのトリガー・プログラムが、元々同じライブラリーにあるときは、トリガー・プログラムが新しいライブラリーに存在していない場合でも、トリガー・プログラム・ライブラリーは必ず新しいライブラリーに変更されます。さらに、以下のことが真になります。
  - CRTDUPOBJ コマンドが物理ファイルとそのトリガー・プログラムの両方を新しいライブラリーに複製している場合は、新しいトリガー・プログラムは新しい物理ファイルと関連付けられます。
  - CRTDUPOBJ コマンドが物理ファイルのみを複製している場合は、宛先ライブラリー内で同じプログラム名を持つトリガー・プログラムは、新しい物理ファイルに関連付けられます。宛先ライブラリー内に、その名前を持ったトリガー・プログラムがない場合でも、この関連付けは行われます。トリガー・プログラムのライブラリーは変更されます。
  - CRTDUPOBJ コマンドがトリガー・プログラムのみを複製している場合は、新しいトリガー・プログラムはどの物理ファイルにも関連付けられません。
- 物理ファイルおよびそのトリガー・プログラムが、元は別のライブラリー内にあるとき
  - 古いトリガー・プログラムは、新しい物理ファイルと関連付けられます。新しい物理ファイルが同じライブラリー内にトリガー・プログラムとして複製されたとしても、古いトリガー・プログラムは新しい物理ファイルと関連付けられます。
- QTEMP ライブラリー内に存在するトリガー・プログラムは、追加できません。データベース・ファイルの場合、CRTDUPOBJ コマンドは、 TO ライブラリーの中にトリガー・プログラムを見つけようとします。 CRTDUPOBJ コマンドで新しいライブラリーとして QTEMP が指定された場合、 CRTDUPOBJ はオブジェクトを可能な限り作成しようとします。ファイルが作成されても、トリガーが追加されないため、ファイルはメンバーが存在しないまま QTEMP に残ります。

物理ファイル・メンバー消去 (CLRPFM)

- 物理ファイルが削除トリガーと関連付けられている場合、CLRPFM 操作は失敗します。

物理ファイル・メンバー初期設定 (INZPFM)

- 物理ファイルが挿入トリガーと関連付けられている場合、INZPFM 操作は失敗します。

FORTRAN データの強制終了 (FEOD)

- 物理ファイルが削除トリガーと関連付けられている場合、FEOD 操作は失敗します。

ジャーナルされた変更の適用またはジャーナルされた変更の削除 (APYJRNCHG/RMVJRNCHG)

- 物理ファイルがどのタイプのトリガーと関連付けられていても、 APYJRNCHG 操作や RMVJRNCHG 操作ではトリガー・プログラムは呼び出されません。したがって、トリガー・プログラム内のすべての

ファイルをジャーナルしておかなければなりません。その後で、APYJRNCHG コマンドまたは RMVJRNCHG コマンドを使用するときには、必ずそれらのファイルをすべて指定します。このようにすれば、アプリケーション・プログラムですべての物理ファイルの変更が確保され、トリガー・プログラムは一貫性を保つことができます。

注: どのトリガー・プログラム機能もデータベース・ファイルと関連付けられていなく、明示的にジャーナル処理されていない場合は、関連した情報を記録するためにジャーナル項目を送信してください。ジャーナル項目の送信 (SNDJRNE) コマンドまたはジャーナル項目の送信 (QJOSJRNE) API を使用してください。一貫性を保つために、データベース・ファイルの回復中は、この情報を使用してください。

トリガーおよび参照保全との関係

物理ファイルに、トリガーと参照制約の両方を関係付けることができます。トリガー処置および参照制約の実行順は、ファイルに関連した制約とトリガーによって異なります。

ある場合には、システムは、変更前トリガー・プログラムを呼び出す前に、参照制約を評価します。RESTRICT 規則を指定する制約の場合が、これに当てはまります。

場合によっては、トリガー・プログラム内のすべてのステートメント (ネスト・トリガー・プログラムを含む) は、制約が適用される前に実行されます。NO ACTION、CASCADE、SET、NULL、および SET DEFAULT 参照制約の規則が、これに当てはまります。このような規則を指定すると、トリガー・プログラムのネスト結果に基づいて、システムはファイルの制約を評価します。たとえば次のように、制約およびトリガーをもった EMP ファイルに、アプリケーションが従業員レコードを挿入するとします。

- 参照制約では、EMP ファイルに挿入される従業員レコードの部門番号は、DEPT ファイル中に存在しなければならないと指定しています。
- EMP ファイルへの挿入が行われるごとに、トリガー・プログラムは、 DEPT ファイル内に部門番号が存在するかどうかを検査します。存在しない場合は、番号を追加します。

EMP ファイルに挿入されると、システムはまずトリガー・プログラムを呼び出します。部門番号が DEPT ファイルにない場合には、トリガー・プログラムは、 DEPT ファイルに新しい部門番号を挿入します。次にシステムは、参照制約を評価します。この場合、部門番号が DEPT ファイルにあるため、挿入は正常に行われます。

トリガーおよび参照制約が同一の物理ファイルに対して定義されている場合、特定の制限事項があります。

- 削除トリガーを物理ファイルに関連付ける場合、そのファイルは、 CASCADE の削除規則をもつ参照制約を受ける従属ファイルであってはなりません。
- 更新トリガーを物理ファイルに関連付ける場合、この物理ファイル内のフィールドは、SET NULL または SET DEFAULT の削除規則をもつ参照制約を受ける外部キーになることはできません。

トリガー・プログラムまたは参照制約の妥当性検査で障害が起きた場合、すべてのファイルを同一のコミットメント定義の下で実行していれば、変更操作に関連したすべてのトリガー・プログラムはロールバックします。参照制約が保証されるのは、トリガー・プログラムおよび参照保全のネットワーク内のすべてのファイルが、同一のコミットメント定義の下で実行されている場合です。コミットメント制御なしまたは混合シナリオでファイルを開いた場合、予期しない結果が生じることがあります。

トリガーを使用すれば、参照制約および業務規則を実施することができます。たとえば、トリガーを使用して、物理ファイルでのカスケード制約の更新をシミュレートできます。しかし、システムの参照保全機能を使用して定義した制約が提供したものと同じ機能的働きは得られません。トリガーを使用して制約を定義した場合は、参照保全の以下の利点が失われることがあります。

- 制約を検査保留にしながら、それでもファイルの操作を許容する 1 つ以上の参照制約に違反した行を、従属ファイルに入れておくことができます。
- 制約が検査保留になった場合に、ユーザーに知らせる機能。
- アプリケーションが COMMIT(*NONE) の下で実行されていて、カスケード削除時にエラーが生じた場合には、すべての変更はデータベースによってロールバックされます。
- 制約に関連したファイルを保管するとき、データベース・ネットワークはすべての従属ファイルを同じライブラリー内に保管します。

データベースの配布

別売の機能である DB2 マルチシステムは、結合の弱い環境にある複数システムにデータベース・ファイルを配布する、単純で直接的な方式を提供します。

DB2 マルチシステムにより、分散 iSeries システム上のユーザーは、分散データベースがすべてユーザーのシステムに存在しているかのように、リアルタイムで分散データベースへの Query と更新アクセスを行うことができます。ユーザー定義によるキー・フィールド (複数も可) に基づいて、 DB2 マルチシステムは新規レコードを適切なシステムに配置します。DB2 マルチシステムは、システム提供またはユーザー定義のハッシュ・アルゴリズムのどちらかに基づいて、システムを選択します。

環境内のノード数にほぼ等しい因数で Query のパフォーマンスが向上します。たとえば、4 つのシステムに分散されたデータベースに対する照会は、約 1/4 の時間で実行されます。しかし、照会に結合処理およびグループ化処理が関係していると、照会のパフォーマンスは大きく異なる場合があります。さらに、複数のノード間でのデータのバランスも、照会のパフォーマンスに影響を及ぼします。マルチ・システムは、照会をそれぞれのシステムで同時に実行します。DB2 マルチシステム では、大規模なデータベースへの照会にかかる時間がかなり短縮できます。詳細については、 DB2 マルチ・システムを参照してください。

---

2 バイト文字セット (DBCS) に関する考慮事項

2 バイト文字セット (DBCS) は、個々の文字を 2 バイトで表す文字セットです。DBCS は、固有の文字または記号 (1 バイトで表せる文字の最大数は 256文字です) を多数含む各国言語をサポートします。そのような言語の例としては、日本語、韓国語、および中国語があります。

このトピックでは、iSeries システム上のデータベースに適用される DBCS の考慮事項を示します。以下のトピックを参照してください。

- 『DBCS フィールドのデータ・タイプ』
- 284 ページの『DBCS フィールドのマッピングに関する考慮事項』
- 285 ページの『DBCS フィールドの連結』
- 286 ページの『DBCS フィールドのサブストリング操作』
- 286 ページの『論理ファイルでの DBCS フィールドの比較』
- 287 ページの『Query ファイルのオープン (OPNQRYF) コマンドでの DBCS フィールドの使用』

DBCS フィールドのデータ・タイプ

2 種類の一般的な DBCS データがあります。シフト文字付き DBCS データとグラフィック (シフト文字なし) DBCS データです。**シフト文字付き DBCS データ**は、DBCS シフトアウト文字がその前に付き、DBCS シフトイン文字がその後に付くものです。**グラフィック DBCS データ**は、シフトアウト文字とシフ

トイン文字によって囲まれていません。アプリケーション・プログラムでは、シフト文字付き DBCS データを扱うために、グラフィック DBCS データでは必要ないような特殊な処理が必要となる場合もあります。

特定の DBCS データ・タイプ (DDS コーディング形式上の 35 列で指定される) は、次のとおりです。

記入項目

意味

O DBCS 混用: 1 バイト・データとシフト文字付き 2 バイト・データの両方が含まれる文字ストリング。

E DBCS 択一: すべて 1 バイト・データか、すべてシフト文字付き 2 バイト・データのどちらかの文字ストリング

J DBCS 専用: シフト文字付き 2 バイト・データのみが含まれる文字ストリング。

G DBCS グラフィック: シフトなし 2 バイト・データのみが含まれる文字ストリング。

注: DBCS データ・タイプを含むファイルは、 1 バイト文字セット (SBCS) システム上で作成可能です。DBCS データ・タイプを含むファイルは、SBCS システムでオープンしたり使用したりすることができますが、システムが DBCS または混合 CCSID から SBCS CCSID へ変換しようとすると、コード化文字セット識別コード (CCSID) 変換エラーが発生する可能性があります。このエラーは、ジョブ CCSID が 65535 の場合は発生しません。

このほかに、『DBCS 定数』を参照してください。

DBCS 定数

定数は、使用される実際の文字ストリングを識別します。文字ストリングは、アポストロフィに囲まれ、DBCS 文字のストリングは、 DBCS シフトアウトおよびシフトイン文字 (以下の例では < および > で表されています) によって囲まれます。DBCS グラフィック定数は、その前に文字 *G* が付きます。DBCS 定数のタイプは、次のとおりです。

タイプ **例**

DBCS 専用 '<A1A2A3>'

DBCS 混用 '<A1A2A3>BCD'

DBCS グラフィック
G'<A1A2A3>'

DBCS フィールドのマッピングに関する考慮事項

以下の表は、DBCS フィールドの物理ファイルと論理ファイルの間で有効なデータ・マッピングのタイプを示しています。

	論理ファイルのデータ・タイプ								
物理ファイルのデータ・タイプ	**文字**	**16 進数**	**DBCS 混用**	**DBCS 択一**	**DBCS 専用**	**DBCS グラフィック**	**UCS2 グラフィック**	**UTF-8**	**UTF-16**
文字	有効	有効	有効	無効	無効	無効	有効	有効	有効
16 進数	有効	有効	有効	無効	無効	無効	無効	無効	無効
DBCS 混用	無効	有効	有効	無効	無効	無効	有効	有効	有効
DBCS 択一	無効	有効	有効	有効	無効	無効	無効[1]	無効[1]	無効[1]
DBCS 専用	有効	有効	有効	有効	有効	有効	無効[1]	無効[1]	無効[1]

	論理ファイルのデータ・タイプ								
物理ファイルのデータ・タイプ	**文字**	**16 進数**	**DBCS 混用**	**DBCS 択一**	**DBCS 専用**	**DBCS グラフィック**	**UCS2 グラフィック**	**UTF-8**	**UTF-16**
DBCS グラフィック	無効	無効	有効	有効	有効	有効	有効	有効	有効
UCS2 グラフィック	有効	無効	有効	無効[1]	無効[1]	有効	有効	有効	有効
UTF-8	有効	無効	有効	無効[1]	無効[1]	有効	有効	有効	有効
UTF-16	有効	無効	有効	無効[1]	無効[1]	有効	有効	有効	有効

注: この表の [1] は、変換後に置換文字が表示される可能性があるため、これらのマッピングがサポートされないことを示します。

DBCS フィールドの連結

フィールドが連結されると、データ・タイプが変更されることがあります (結果のデータ・タイプはシステムで自動的に決定されます)。

- OS/400 は、連結されるフィールドのデータ・タイプに基づいて、データ・タイプを割り当てます。DBCS フィールドが連結に含まれている場合は、以下の一般規則が適用されます。
  - 連結に 1 つまたは複数の 16 進数 (H) フィールドが含まれている場合、結果のデータ・タイプは 16 進数 (H) となります。
  - 連結中のフィールドがすべて DBCS 専用 (J) の場合、結果のデータ・タイプは DBCS 専用 (J) となります。
  - 連結に 1 つまたは複数の DBCS (DBCS 混用、DBCS 択一、DBCS 専用) フィールドが含まれていて、16 進数フィールドが含まれていない場合、結果のデータ・タイプは DBCS 混用 (O) となります。
  - 連結に 2 つ以上の DBCS 混用 (O) フィールドが含まれる場合、生じるデータ・タイプは可変長の DBCS 混用 (O) フィールドです。
  - 連結に、タイプを問わず 1 つまたは複数の可変長フィールドが含まれている場合、結果のデータ・タイプは可変長となります。
  - DBCS グラフィック (G) フィールドは、別の DBCS グラフィック・フィールドへのみ連結できます。結果のデータ・タイプは DBCS グラフィック (G) です。
  - UCS2 グラフィック (G) フィールドは、別の UCS2 グラフィック・フィールド、UTF-8 文字フィールド、または UTF-16 グラフィック・フィールドに連結可能です。結果としてできるデータ・タイプは、いずれかのオペランドが UTF-16 の場合には UTF-16、いずれかのオペランドが UTF-8 で UTF-16 のオペランドが存在しない場合には UTF-8、それ以外の場合には UCS-2 になります。
  - UTF-8 文字 (A) フィールドは、別の UTF-8 フィールド、UTF-16 フィールド、または UCS-2 フィールドに連結可能です。結果としてできるデータ・タイプは、いずれかのオペランドが UTF-16 の場合には UTF-16、いずれかのオペランドが UTF-8 で UTF-16 のオペランドが存在しない場合には UTF-8、それ以外の場合には UCS-2 になります。
  - UTF-16 グラフィック (G) フィールドは、別の UTF-16 フィールド、UTF-8 フィールド、または UCS-2 フィールドに連結可能です。結果としてできるデータ・タイプは、いずれかのオペランドが UTF-16 の場合には UTF-16、いずれかのオペランドが UTF-8 で UTF-16 のオペランドが存在しない場合には UTF-8、それ以外の場合には UCS-2 になります。
- 連結フィールドの最大長は、連結フィールドのデータ・タイプと、連結されるフィールドの長さによって異なります。連結フィールドがゾーン 10 進数 (S) フィールドの場合、その全長は 31 バイトを超え

ることはできません。連結されたフィールドが文字 (A)、DBCS 混用 (O)、または DBCS 専用 (J) の場合、その長さの総計は 32,766 バイトを超えることはできません (フィールドが可変長フィールドの場合は、32,740 バイト)。

DBCS グラフィック (G) フィールドの長さは、2 バイト文字 の数 (実際の長さは、文字数の 2 倍) として表現されます。このため、連結フィールドの全長は 16,383 文字 (フィールドが可変長フィールドの場合は 16,370 文字) を超えることはできません。

- 結合論理ファイルでは、連結するフィールドは同じ物理ファイルからのものでなければなりません。CONCAT キーワードに指定された最初のフィールドが、使用する物理ファイルを識別します。このため、最初のフィールドは、論理ファイルの基礎となっている物理ファイルの中で固有でなければなりません。もしくは、どの物理ファイルを使用するか指定するために、 JREF キーワードも指定しなければなりません。
- 連結フィールドの用途は I (入力専用) でなければなりません。
- O または J のデータ・タイプを割り当てた連結フィールドでは、 REFSHIFT を指定することはできません。

注:

1. シフト文字付き DBCS フィールドを連結する場合、あるフィールドの終わりのシフトイン文字と、次のフィールドの始めのシフトアウト文字が除去されます。連結に 1 つまたは複数の 16 進数フィールドが含まれる場合、シフトインとシフトアウトの対は、最初の 16 進数フィールドの前にある DBCS フィールドについてのみ排除されます。
2. DBCS フィールドを含む連結フィールドは、入力専用のフィールドでなければなりません。
3. Query ファイル・オープン (OPNQRYF) コマンドを使用する場合は、連結 DBCS フィールドの結果のデータ・タイプが異なることがあります。DBCS フィールドが連結に含まれる場合は、 288 ページの『OPNQRYF での DBCS フィールドとの連結の使用』における一般規則を参照してください。

DBCS フィールドのサブストリング操作

サブストリング操作により、論理ファイル中のフィールドまたは定数の一部を使用できます。シフト文字付き DBCS データ・タイプの場合は、その開始位置およびサブストリングの長さは *バイト* の数を引用します。ですから、各 2 バイト文字は 2 つの位置としてカウントされます。DBCS グラフィック (G) データ・タイプの場合は、開始位置およびサブストリングの長さは *文字* の数を引用します。ですから、各 2 バイト文字は 1 つの位置としてカウントされます。

論理ファイルでの DBCS フィールドの比較

2 つのフィールドまたは 1 つのフィールドと定数を比較するときは、データ・タイプの互換がある場合に限り、固定長フィールドと可変長フィールドを比較できます。論理ファイルの DBCS フィールドの有効な比較について、 表 15 で説明します。

表 15. 論理ファイルでの DBCS フィールドの有効比較

	任意の形式の数値	文字	16 進数	DBCS 混用	DBCS 択一	DBCS 専用	DBCS グラフィック	UCS-2 グラフィック	UTF-8	UTF-16	日付	時刻	時刻スタンプ
任意の形式の数値	有効	無効	無効	無効	無効	無効	無効	無効	無効	無効	無効	無効	無効
文字	無効	有効	有効	有効	有効	無効	無効	無効	有効	有効	無効	無効	無効
16 進数	無効	有効	有効	有効	有効	有効	無効	無効	無効	無効	無効	無効	無効
DBCS 混用	無効	有効	有効	有効	有効	有効	無効	無効	有効	有効	無効	無効	無効
DBCS 択一	無効	有効	有効	有効	有効	有効	無効	無効	無効	無効	無効	無効	無効

表 15. 論理ファイルでの DBCS フィールドの有効比較 (続き)

	任意の形式の数値	文字	16 進数	DBCS 混用	DBCS 択一	DBCS 専用	DBCS グラフィック	UCS-2 グラフィック	UTF-8	UTF-16	日付	時刻	時刻スタンプ
DBCS 専用	無効	無効	有効	有効	有効	有効	無効	無効	無効	無効	無効	無効	無効
DBCS グラフィック	無効	無効	無効	無効	無効	無効	有効	無効	有効	有効	無効	無効	無効
UCS2 グラフィック	無効	無効	無効	無効	無効	無効	無効	有効	有効	有効	無効	無効	無効
UTF-8	無効	有効	無効	有効	無効	無効	有効	有効	有効	有効	無効	無効	無効
UTF-16	無効	有効	無効	有効	無効	無効	有効	有効	有効	有効	無効	無効	無効
日付	無効	無効	無効	無効	無効	無効	無効	無効	無効	無効	有効	無効	無効
時刻	無効	無効	無効	無効	無効	無効	無効	無効	無効	無効	無効	有効	無効
時刻 スタンプ	無効	無効	無効	無効	無効	無効	無効	無効	無効	無効	無効	無効	有効

Query ファイルのオープン (OPNQRYF) コマンドでの DBCS フィールドの使用

この項では Query ファイルのオープン (OPNQRYF) コマンドで DBCS フィールドを使用するときの考慮事項について説明します。以下のトピックを参照してください。

- 『DBCS フィールドでのワイルドカード機能の使用』
- 『OPNQRYF での DBCS フィールドの比較』
- 288 ページの『OPNQRYF での DBCS フィールドとの連結の使用』
- 288 ページの『DBCS についての分類順序の使用』

DBCS フィールドでのワイルドカード機能の使用

DBCS フィールドでのワイルドカード (%WLDCRD) 機能の使用は、その機能がシフト文字付き DBCS フィールドで用いられるか、 DBCS グラフィック・フィールドで用いられるかによって異なります。

シフト文字付き DBCS フィールドでワイルドカード機能を使用する場合は、1 バイトと 2 バイトの両方のワイルドカード値 (アスタリスクおよび下線) を使用できます。以下の特別な規則が適用されます。

- 1 バイト下線は 1 つの EBCDIC 文字を参照し、 2 バイト下線は 1 つの 2 バイト文字を参照します。
- 1 または 2 バイトのアスタリスクは、任意のタイプの文字を任意の数だけ参照します。

DBCS グラフィック・フィールドでワイルドカード機能を使用するときは、 2 バイトのワイルドカード値 (アスタリスクおよび下線) のみ許可されます。以下の特別な規則が適用されます。

- 2 バイトの下線は、1 つの 2 バイト文字を参照します。
- 2 バイトのアスタリスクは、2 バイト文字を任意の数だけ参照します。

OPNQRYF での DBCS フィールドの比較

2 つのフィールドまたは 2 つの定数を比較するときは、データ・タイプの互換がある場合に限り、固定長フィールドと可変長フィールドを比較できます。 288 ページの表 16 で、OPNQRYF コマンドでの DBCS フィールドの有効な比較について説明します。

表 16. OPNQRYF コマンドでの DBCS フィールドの有効な比較

	任意の形式の数値	文字	16 進数	DBCS 混用	DBCS 択一	DBCS 専用	DBCS グラフィック	UCS2 グラフィック	日付	時刻	時刻スタンプ
任意の形式の数値	有効	無効	無効	無効	無効	無効	無効	無効	無効	無効	無効
文字	無効	有効	有効	有効	有効	無効	無効	有効	有効	有効	有効
16 進数	無効	有効	有効	有効	有効	有効	無効	無効	有効	有効	有効
DBCS 混用	無効	有効	有効	有効	有効	有効	無効	有効	有効	有効	有効
DBCS 択一	無効	有効	有効	有効	有効	有効	無効	有効	有効	有効	有効
DBCS 専用	無効	無効	有効	有効	有効	有効	無効	有効	無効	無効	無効
DBCS グラフィック	無効	無効	無効	無効	無効	無効	有効	有効	無効	無効	無効
UCS2 グラフィック	無効	有効	無効	有効	有効	有効	有効	有効	無効	無効	無効
日付	無効	有効	有効	有効	有効	無効	無効	無効	有効	無効	無効
時刻	無効	有効	有効	有効	有効	無効	無効	無効	無効	有効	無効
時刻 スタンプ	無効	有効	有効	有効	有効	無効	無効	無効	無効	無効	有効

OPNQRYF での DBCS フィールドとの連結の使用

Query ファイルのオープン (OPNQRYF) の連結機能を使用する場合、 OS/400 プログラムは、連結されるフィールドのデータ・タイプに基づいて結果のデータ・タイプを割り当てます。DBCS フィールドが連結に含まれている場合、結果のデータ・タイプは、わずかな違いはありますが、一般的には論理ファイルでの連結フィールドと同じようになります。以下の一般規則が適用されます。

- 連結に 1 つまたは複数の 16 進数 (H) フィールドが含まれている場合、結果のデータ・タイプは 16 進数 (H) となります。
- 連結に 1 つまたは複数の UCS2 グラフィック・フィールドが含まれている場合、結果のデータ・タイプは UCS2 グラフィックとなります。
- 連結中のフィールドがすべて DBCS 専用 (J) の場合、結果のデータ・タイプは可変長の DBCS 専用 (J) となります。
- 連結に 1 つまたは複数の DBCS (O、E、J) フィールドが含まれていて、16 進数フィールドや UCS2 グラフィック・フィールドが含まれていない場合、結果のデータ・タイプは可変長の DBCS 混用 (O) となります。
- 連結に 1 つまたは複数の任意のデータ・タイプの可変長フィールドが含まれている場合、結果のデータ・タイプは、可変長となります。
- DBCS グラフィック (G) フィールドを別の DBCS グラフィック・フィールドへ連結する場合、結果のデータ・タイプは DBCS グラフィック (G) になります。

DBCS についての分類順序の使用

分類順序が指定されると、DBCS データの変換が行われません。DBCS 択一または DBCS 混用フィールドの SBCS データだけが変換されます。UCS2 データは変換されます。

関連情報

以下の iSeries 関連資料および Information Center のトピックには、ユーザーが必要とする情報が記載されています。いくつかの資料は、正式な名称および資料番号と一緒にリストされています。本書の中でこれらの資料を参照する場合は、リストに記載されている簡単なタイトルを使用しています。

- 「アプリケーション・プログラミング・インターフェース (API)」この Information Center 資料は、アプリケーション・プログラマーがアプリケーション・プログラミング・インターフェースを使用してシステム・レベルのアプリケーション、または他の OS/400 アプリケーションを開発するのに必要な情報が含まれています。
- 「コミットメント制御」。この Information Center 資料には、データベースの変更を同期化するためのコミットメント制御に関する情報が含まれています。
- 「バックアップおよび回復」。この Information Center 資料には、バックアップおよび回復のストラテジーを計画する方法、データを格納したディスクの保護をセットアップする方法、システムをバックアップする方法、および障害が発生した場合にシステム・シャットダウンを制御する方法が説明されています。

- 「バックアップおよび回復の手引き  」。この資料には、 iSeries サーバーの回復および可用性オプションに関する一般情報が掲載されています。
- 制御言語 (CL)。この Information Center 資料は、 iSeries の制御言語 (CL)、および、OS/400 とライセンス・プログラムのコマンドに関する詳細説明を、アプリケーション・プログラマーおよびシステム・プログラマーに提供します。

- 「CL プログラミング  」。この手引きは、オブジェクトとライブラリーの一般説明、CL プログラミング、プログラム間の流れの制御と通信、 CL プログラムにおけるオブジェクトの処理、および CL プログラムの作成などの、広範囲にわたる iSeries プログラミング・トピックの説明をアプリケーション・プログラマーおよびプログラマーに提供します。
- 「DDS 概念」。この DDS 資料は、 Information Center 内の 5 冊から構成されている解説書で、データベース・ファイル (論理データベース・ファイルと物理データベース・ファイルの両方)、および、ユーザーのプログラムの外部にある特定の装置ファイル (表示装置、印刷装置、およびシステム間通信機能 (ICF)) を記述するのに必要な項目とキーワードの詳細説明をアプリケーション・プログラマーに提供します。
- 「分散データ管理」。この Information Center 資料は、遠隔ファイル処理に関する情報をアプリケーション・プログラマーに提供します。遠隔ファイルを OS/400 分散データ管理機能 (DDM) に定義する方法、DDM ファイルを作成する方法、DDM でサポートされているファイル・ユーティリティー、およびその他のシステムと関連させた場合の OS/400 DDM の要件を説明しています。
- データベース・ファイル管理。この Information Center 資料は、アプリケーション・プログラム内でのファイルの使用に関する情報をアプリケーション・プログラマーに提供します。ファイルのコピー (CPYF) コマンドおよび一時変更コマンドに関する事項が含まれています。
- 「OS/400 グローバリゼーション」。この Information Center 資料は、データ処理管理者、システム操作員とシステム管理者、アプリケーション・プログラマー、エンド・ユーザー、およびシステム技術者に、iSeries システムの各国語機能の理解と使用に関する情報を提供します。ユーザーが、 iSeries グローバリゼーションと多言語システムの計画、導入、構成、および使用の準備ができるようにするためのものです。さらに、多言語データのデータベース管理の説明、および多言語システムのアプリケーションに関する考慮事項も述べています。

- IDDU Use  。この手引きには、管理者の秘書、業務の専門家、またはプログラマーを対象に、対話式データ定義ユーティリティー (IDDU) を使用してデータ・ディクショナリー、ファイル、およびレコードをシステムに記述する方法が記載されています。
- 「Managing Disk Units in Disk Pools」。この Information Center 資料は、情報を常時利用できるようにするために、ディスク装置とディスク・プールを管理および保護するうえで役立ちます。

- 「ジャーナル管理」。この Information Center 資料は、システム管理アクセス・パス保護 (SMAPP)、ローカル・ジャーナル、およびリモート・ジャーナルをセットアップし、管理し、トラブルシューティングする方法について説明しています。
- パフォーマンス。この Information Center 資料には、システムの調整、レコード様式および収集されるデータの内容についての情報などのパフォーマンス・データの収集、システムの全操作を制御または変更するためのシステム値の処理、および、だれがシステムを使い、どの資源が使用されているかを判別するためのデータの収集方法に関する説明などが記載されています。
- 「Query for iSeries ご使用の手引き」。この Information Center 補足資料は、管理者の秘書、業務の専門家、またはプログラマーに、 IBM Query for iSeries を使用してデータベース・ファイルからデータを入手する方法についての説明を提供します。 Query にサインオンする方法、および選択したデータが入っている報告書を作成するための Query の定義と実行方法を説明しています。
- 「セキュリティー」。この Information Center 資料は、機密保護が必要な理由と主要な概念の定義、および、 iSeries システムで基本的な機密保護を計画、実施、およびモニターする方法について説明します。
- 「機密保護解説書  」。このマニュアルは、システム機密保護サポートが、システムやデータが正当な権限を有していない人物によって使われたり、故意または不慮で生じた損傷によって破壊されないよう保護し、機密保護情報を最新の状態に保持して、システム上に機密保護をセットアップする方法を示します。
- 「SQL プログラミング」。この Information Center 資料には、アプリケーション・プログラマー、プログラマー、またはデータベース管理者を対象に、 SQL ステートメントの設計、作成、実行、およびテストの仕方についての概要が記載されています。さらに、対話式構造化照会言語 (SQL) についても説明しています。
- DB2 UDB for iSeries SQL 解説書. この Information Center 資料には、アプリケーション・プログラマー、プログラマー、またはデータベース管理者を対象に、構造化照会言語 (SQL) のステートメントとそのパラメーターについての詳細説明が記載されています。
- 「システム値」。この Information Center 資料には、システム値のリストと説明が記載されています。
- 実行管理機能。この Information Center 資料は、実行管理機能の作成方法と変更方法についての説明をプログラマーに提供します。

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。

| 〒106-0032
| 東京都港区六本木 3-2-31
| IBM World Trade Asia Corporation
| Licensing

以下の保証は、国または地域の法律に沿わない場合は、適用されません。 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 Corporation
| Software Interoperability Coordinator, Department 49XA
| 3605 Highway 52 N
| Rochester, MN 55901
| U.S.A.

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。

この文書に含まれるいかなるパフォーマンス・データも、管理環境下で決定されたものです。そのため、他の操作環境で得られた結果は、異なる可能性があります。一部の測定が、開発レベルのシステムで行われた可能性がありますが、その測定値が、一般に利用可能なシステムのものと同じである保証はありません。さらに、一部の測定値が、推定値である可能性があります。実際の結果は、異なる可能性があります。お客様は、お客様の特定の環境に適したデータを確かめる必要があります。

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。これらの名称はすべて架空のものであり、名称や住所が類似する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。お客様は、IBM 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。

この情報をソフトコピーでご覧になっている場合は、写真やカラーの図表は表示されない場合があります。

商標

以下は、IBM Corporation の商標です。

AS/400
C/400
COBOL/400
DB2
DB2 Universal Database
DRDA
IBM
Integrated Language Environment
iSeries
Operating System/400
OS/400
RPG/400

System/36
System/38
WebSphere

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。

他の会社名、製品名およびサービス名等はそれぞれ各社の商標です。

資料に関するご使用条件

お客様がダウンロードされる資料につきましては、以下の条件にお客様が同意されることを条件にその使用が認められます。

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。

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。

これらの資料の著作権はすべて、IBM Corporation に帰属しています。

お客様が、このサイトから資料をダウンロードまたは印刷することにより、これらの条件に同意されたものとさせていただきます。

Printed in Japan

日本アイ・ビー・エム株式会社
〒106-8711　東京都港区六本木3-2-12