IBM® DB2® ユニバーサル・データベース

管理の手引き: パフォーマンス

バージョン 7

SC88-8512-01
(英文原典：SC09-2945-01)

IBM® DB2® ユニバーサル・データベース

管理の手引き: パフォーマンス

バージョン 7

SC88-8512-01
(英文原典：SC09-2945-01)

ご注意!

本書、および本書がサポートする製品をご使用になる前に、 643ページの『付録F. 特記事項』にある一般的な情報を必ずお読みください。

本書には、IBM の専有情報が含まれています。その情報は、使用許諾条件に基づき提供され、著作権により保護されています。本書に記載される情報には、いかなる製品の保証も含まれていません。また、本書で提供されるいかなる記述も、製品保証として解釈すべきではありません。

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)

原 典:	SC09-2945-01 IBM® DB2® Universal Database Administration Guide: Performance Version 7
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 2001.8

この文書では、平成明朝体™W3、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™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

本書について

3 巻で構成される本書には、 2000 年問題に対応した DB2* リレーショナル・データベース管理システム (RDBMS) 製品を使用し管理するために必要な、以下の情報が収められています。

- データベース設計についての情報 (*管理の手引き: 計画*)
- データベースの使用および管理についての情報 (*管理の手引き: インプリメンテーション*)
- パフォーマンスを向上させるための、データベース環境の構成およびチューニングについての情報 (*管理の手引き: パフォーマンス*)

本書に記載されているタスクの多くは、以下に示すさまざまなインターフェースを使用して実行することができます。

- **コマンド行プロセッサー**。グラフィカル・インターフェースから、データベースをアクセスし操作することができます。このインターフェースから、SQL ステートメントおよび DB2 ユーティリティー関数も実行することができます。本書にある例のほとんどが、このインターフェースを使った場合を例として取り上げています。コマンド行プロセッサーの使用に関する詳細は、*コマンド解説書* を参照してください。
- **アプリケーション・プログラミング・インターフェース**。アプリケーション・プログラム内で DB2 ユーティリティー関数を実行することができます。アプリケーション・プログラミング・インターフェースの使用についての詳細は、 *管理 API 解説書* を参照してください。
- **コントロール・センター**。システムの構成、ディレクトリーの管理、システムのバックアップとリカバリー、ジョブのスケジューリング、およびメディアの管理などの管理タスクをグラフィックを使用して実行することができます。またコントロール・センターには、システム間のデータの複製をグラフィックを使用してセットアップするための**複製管理**が含まれています。さらに、コントロール・センターでは、グラフィカル・ユーザー・インターフェースを介して DB2 ユーティリティー機能を実行できます。プラットフォームによっては、コントロール・センターを呼び出すための別の方法もあります。たとえば、コマンド行で `db2cc` コマンドを使用するか、 (OS/2 の場合) 「DB2」フォルダーから「コントロール・センター (Control Center)」アイコンを選択するか、または Windows プラットフォームの場合、「スタート」パネルを使用します。紹介のヘルプを表示するには、「コントロール・センター (Control Center)」ウィンドウの「**ヘルプ (Help)**」プルダウンから「**入門 (Getting started)**」を選択してください。 **Visual Explain** および**パフォーマンス測定プログラム**のツールは、コントロール・センターから呼び出されます。

他にも、管理タスクを行うために使用できるツールがあります。それらのツールには、以下のものがあります。

- スクリプト・センターは、スクリプトと呼ばれる小さなアプリケーションを保管します。これらのスクリプトには、オペレーティング・システムのコマンドだけでなく、SQL ステートメント、DB2 コマンドを含めることができます。
- アラート・センターは、他の DB2 操作から出されるメッセージをモニターします。
- ツール設定は、コントロール・センター、アラート・センター、および複製についての設定値を変更します。
- ジャーナルは、自動で実行するジョブをスケジュールします。
- データウェアハウスセンターは、ウェアハウス・オブジェクトを管理します。

本書の対象読者

本書は、ローカルまたはリモート・クライアントがアクセスするデータベースを設計、使用、および維持する必要のあるデータベース管理担当者、システム管理者、セキュリティー管理者、およびシステム・オペレーターを対象としています。 DB2 リレーショナル・データベース管理システムの管理および操作について理解しておくことが必要なプログラマーや、その他のユーザーも本書をご使用になれます。

本書の構成

本書には、以下の主な項目に関する情報が記載されています。

パフォーマンスの紹介

- 第1章 パフォーマンスの要素では、 DB2 UDB パフォーマンスの管理と改善に関する概念と考慮事項について紹介します。
- 第2章 アーキテクチャーとプロセスの概要では、基礎となる DB2 ユニバーサル・データベースの構造およびプロセスを紹介します。

アプリケーション・パフォーマンスのチューニング

- 第3章 アプリケーションについての考慮事項では、アプリケーションの設計時点で、データベースのパフォーマンスを向上させる手法をいくつか説明します。
- 第4章 環境についての考慮事項では、データベース環境の設定時点で、データベースのパフォーマンスを向上させる手法をいくつか説明します。
- 第5章 システム・カタログ統計では、最適なパフォーマンスを達成するために、データについての統計を収集し使用する方法について説明します。
- 第6章 SQL コンパイラーに関する解説では、 SQL コンパイラーを使用してコンパイルしたときに、 SQL ステートメントがどうなるかについて説明します。

- 第7章 SQL Explain 機能では、Explain 機能について説明します。この機能により、データにアクセスするために SQL コンパイラーが行った選択を調べることができます。

システムのチューニングと構成

- 第8章 操作上のパフォーマンスでは、データベース・マネージャーがメモリーを使用する方法の概要、および実行時のパフォーマンスに影響を与えるその他の考慮事項について説明します。
- 第9章 管理プログラムの使用法では、データベース管理のある局面を制御するための管理プログラムの使用について紹介します。
- 第10章 プロセッサーの追加による構成のスケーリングでは、データベース・システムのサイズの増加に関連する考慮事項と作業について説明します。
- 第11章 データベース区画間でのデータの再配分では、区画間でデータを再配分するために、区分データベース環境において必要な作業について説明します。
- 第12章 ベンチマーク・テストでは、ベンチマーク・テストの概要とベンチマーク・テストの実行方法について説明します。
- 第13章 DB2 の構成では、データベース・マネージャー、データベース構成ファイルおよび構成パラメーターの値について説明します。

付録

- 付録A. DB2 レジストリー変数と環境変数では、プロファイル・レジストリーの値と環境変数を示します。
- 付録B. Explain 表と定義では、 DB2 Explain 機能が使用する表についての情報を示し、それらの表の作成方法について説明します。
- 付録C. SQL EXPLAIN ツールでは、 DB2 EXPLAIN ツールである db2expln および dynexpln の使用法について説明します。
- 付録D. db2exfmt - Explain 表フォーマット・ツールでは、 DB2 Explain 表の内容をフォーマットします。
- 付録E. DB2 ライブラリーの使用法では、ウィザード、オンライン・ヘルプ、メッセージ、およびマニュアルを含む DB2 ライブラリーの構造について説明します。

管理の手引きの他の巻の概要

管理の手引き: 計画

管理の手引き: 計画 は、データベース設計を扱っています。論理設計および物理設計、分散トランザクション、および高可用性について説明しています。この巻のそれぞれの章と付録は、以下のように構成されています。

DB2 ユニバーサル・データベースの世界

- 『DB2 ユニバーサル・データベースの管理』では、 DB2 ユニバーサル・データベースを紹介し、概要を説明します。

データベースの概念

- 『関係データベースの基本的な概念』では、リカバリー・オブジェクト、記憶オブジェクト、およびシステム・オブジェクトを含むデータベース・オブジェクトの概要を説明します。
- 『連合システム』では、連合システム (連合データベース・システム: 複数のデータベースから構成されるが、単一のデータベース・イメージを提供するデータベース・システムを意味します) について説明します。連合システムはデータベース管理システム (DBMS) の一種で、アプリケーションやユーザーが 1 つのステートメント内で 2 つ以上の DBMS またはデータベースを参照する SQL ステートメントを実行依頼できます。
- 『並列データベース・システム』では、 DB2 で実現される並行性のタイプを紹介します。
- 『データウェアハウジングについて』では、データウェアハウジングおよびウェアハウジング・タスクについて概説します。
- 『地理情報エクステンダーについて』では、地理情報エクステンダーを紹介し、この目的とこれが処理するデータについて説明します。

データベースの設計

- 『論理データベースの設計』では、論理データベースの設計に関する概念と指針について説明します。
- 『物理データベースの設計』では、物理データベースの設計 (データ・ストレージに関する考慮事項を含む) の指針について説明します。

分散トランザクション処理

- 『分散データベースの設計』では、単一トランザクションで複数のデータベースをアクセスする方法を説明します。
- 『トランザクション・マネージャーの設計』では、 CICS などの分散トランザクション処理環境でデータベースを使用する方法について説明します。

高可用性システム

- 『高可用性とフェールオーバー・サポートの紹介』では、 DB2 が提供する高可用性障害リカバリー・サポートの概要を説明します。

付録

- 『データベース移行の計画』では、第 7 版へのデータベースの移行について説明します。
- 『リリースからリリースへの非互換性』では、第 7 版までに導入された、互換性のない機能について説明します。

- 『各国語サポート (NLS)』では、国、言語、およびコード・ページの情報を含む、DB2 各国語サポート (NLS) について説明します。

管理の手引き: インプリメンテーション

管理の手引き: インプリメンテーション では、データベース設計のインプリメントについて扱います。この巻のそれぞれの章と付録は、以下のように構成されています。

コントロール・センターによる管理

- 『GUI ツールを使用した DB2 の管理』では、データベースを管理するのに使用するグラフィカル・ユーザー・インターフェース (GUI) ツールについて説明します。

設計のインプリメント

- 『データベースを作成する前に』では、データベースを作成する際の前提条件について説明します。
- 『データベースの作成』では、データベースおよび関係するデータベース・オブジェクトの作成に関連したタスクを説明します。
- 『データベースの変更』では、データベースを変更する前に実行しなければならないタスクや、データベースまたは関係するデータベース・オブジェクトの変更または除去に関するタスクについて説明します。

データベースのセキュリティー

- 『データベース・アクセスの制御』では、データベース・リソースへのアクセスを制御する方法について説明します。
- 『DB2 アクティビティーの監査』では、データに対する不要なアクセスまたは予期せぬアクセスを検出してモニターする方法について説明します。

データの移動

- 『データの移動に使用するユーティリティー』は 1 ページで構成されており、データを移動するさまざまな方法について紹介し、 *データ移動ユーティリティー手引きおよび解説書* を読むための基礎的な情報を提供します。

リカバリー

- 『データベースのリカバリー』は、データベースのバックアップ、復元、およびロールフォワードの概念を簡潔に 1 ページで紹介しています。より詳細な情報は、*データ回復と高可用性の手引きと解説書* に記載されています。

付録

- 『分散コンピューティング環境 (DCE) ディレクトリー・サービスの使用』では、DCE ディレクトリー・サービスを使用する方法について説明します。

- 『データベース・リカバリー用のユーザー出口』では、ユーザー出口プログラムでデータベース・ログ・ファイルを使用する方法について説明し、いくつかのサンプル・ユーザー出口プログラムを示します。
- 『複数のデータベース区画サーバーに対するコマンドの発行』では、コマンドを区分データベース環境内のすべての区画に送るための *db2_all* および *rah* シェル・スクリプトの使用法について説明します。
- 『DB2 (Windows NT 版) が Windows NT セキュリティーを処理する方法』では、DB2 が Windows NT セキュリティーを処理する方法について説明します。
- 『Windows NT パフォーマンス・モニターの使用』では、 Windows NT パフォーマンス・モニタへの DB2 の登録についての情報と、パフォーマンス情報を使用する方法についての情報を示します。
- 『Windows NT または Windows 2000 データベース区画サーバーを使った作業』では、 Windows NT または Windows 2000 上でデータベース区画サーバーを動作させるために使用できるユーティリティーに関する情報が示されています。
- 『複数の論理ノードの構成』では、区分データベース環境で複数論理ノードを構成する方法について説明します。
- 『高速ノード間通信』では、仮想インターフェース・アーキテクチャーを DB2 ユニバーサル・データベースで使用できるようにする方法について説明します。
- 『Lightweight Directory Access Protocol (LDAP) ディレクトリー・サービス』では、LDAP ディレクトリー・サービスを使用する方法について説明します。
- 『コントロール・センターの拡張』では、新しいアクションを割り当てた新しいツールバー・ボタンの追加、新しいオブジェクト定義の追加、および新しいアクション定義の追加により、コントロール・センターを拡張する方法について説明します。

第1部 パフォーマンスの紹介

第1章 パフォーマンスの要素

パフォーマンス とは、特定の作業負荷がかかっているコンピューター・システムの動作の仕方のことです。パフォーマンスは、システムの応答時間、スループット、および可用性によって測定されます。また、次の事柄の影響も受けます。

- 使用可能なリソース
- リソースの使用頻度と共用の程度

一般的に、パフォーマンスのチューニングは、システムの費用効率を改善しようとするときに行います。 これには、次のような目標があります。

- 処理費用を増やすことなく、より大きい作業負荷または要求がより多い作業負荷を処理する。 (たとえば、新しいハードウェアを購入したりプロセッサー時間を長くしたりすることなく作業負荷を増やす。)
- 処理費用を増やすことなく、システムの応答時間を速くしたり、スループットを高くしたりする。
- ユーザーが利用するサービスにマイナスの影響を与えることなく、処理費用を削減する。

パフォーマンスを技術的な用語から経済学的な用語に翻訳することは簡単ではありません。パフォーマンスのチューニングには、確かに (人件費やプロセッサー時間などによる) 費用がかかるので、チューニングを行う前に、費用と見込まれる効果とを比較考量してください。 これらの効果には、次のようにはっきり分かるものがあります。

- リソースをより効率的に使用できるようになる
- システムにより多くのユーザーを追加できるようになる

応答時間が速くなったことによるユーザーの満足度の向上などのその他の効果は、はっきりとは分かりません。 これらすべての効果について考慮する必要があります。

DB2 には、パフォーマンスに関連した管理タスクのいくつかを実行するのに役立つウィザードが統合されています。一般にこれらのタスクでは、少しの時間でパフォーマンスを著しく改善することができます。ウィザードでは、それぞれのタスクを 1 ステップずつ実行します。ウィザードは、コントロール・センターおよびクライアント構成アシスタントから利用できます。

パフォーマンス構成 (Performance Configuration) ウィザードは、構成パラメーターを業務要件に合うように更新することによって、データベースのパフォーマンスをチューニングするのに役立ちます。このウィザードは、データベースのパフォーマンスを改善するのに役立ちます。データベース作成 (Create Database) ウィザードも、同様の点である程度役立ちます。その他のウィザードは、個々の表と一般のデータ・アクセスのパフォ

ーマンスを改善する点で役立ちます。これらのウィザードには、「表作成 (Create Table)」、「索引 (Index)」、および「マルチサイト更新の構成 (Configure Multisite Update)」ウィザードがあります。ウィザードは、コントロール・センターでオブジェクトの上でマウスの右ボタンをクリックすることによって、見つけることができます。

チューニングの指針

次の指針は、パフォーマンス・チューニングの全体的なアプローチを決定するのに役立ちます。

収穫逓減の法則を覚えておく: 通常、パフォーマンスの効果を最大にするには、最初の努力が重要です。一般にその後の変更では、得られる効果は少なくなり、必要な努力は多くなります。

チューニングのためのチューニングを行わない: チューニングは、識別されている制約を軽減するために行ってください。 パフォーマンス上の問題の主要な原因ではないリソースのチューニングを行っても、応答時間に対する効果は主要な制約を軽減するまでほとんどまたはまったくなく、それ以降のチューニング作業が行いにくくなります。 著しく改善できる可能性があるとすれば、それは応答時間の主要な要因となっているリソースのパフォーマンスを改善することにあります。

システム全体を考慮する: 他に影響を与えることなく 1 つのパラメーターまたはシステムのチューニングを行うことはできません。 調整を行う前に、その調整によってシステム全体がどのような影響を受けるかを考慮してください。

一度に 1 つのパラメーターを変更する: 一度に複数のパフォーマンス・チューニング・パラメーターを変更しないでください。 すべての変更が有益であることを確信している場合でも、それぞれの変更の効果を評価することができなくなります。 一度に複数のパラメーターを変更すると、行ったトレードオフを効果的に判断することもできません。 1 つのパラメーターを調整して 1 つの分野を改善すると、考慮に入れていなかった少なくとも 1 つの別の分野が影響を受けます。一度に 1 つずつ変更を行うことにより、希望どおりの改善が行われたかどうかをベンチマークを使用して評価することができます。

レベルごとに測定と再構成を行う: 一度に変更するパラメーターを 1 つにするのと同じ理由で、一度にチューニングするシステムのレベルは 1 つにしてください。 次のリストは、システム内の各レベルを示しています。

- ハードウェア
- オペレーティング・システム
- アプリケーション・サーバーとリクエスター
- データベース・マネージャー
- SQL ステートメント

- アプリケーション・プログラム

ハードウェアおよびソフトウェアの問題を検査する: 一部のパフォーマンス問題は、ハードウェアかソフトウェアのどちらか (あるいはその両方) にサービスを適用することによって修正することができます。 単にサービスを適用するだけで済む場合は、システムのモニターとチューニングに余分な時間がかかりません。

ハードウェアをアップグレードする前に問題を理解する: ストレージを増やしたりプロセッサーの能力を高めたりすることでパフォーマンスを即座に改善できるように思えても、時間を取って障害がどこに存在しているのかを理解してください。 費用をかけてディスク装置を追加しても、それを活用するだけの処理能力やチャネルがないことに気付く場合があります。

チューニングを始める前にフォールバック手順を実施する: 前述のとおり、チューニングを行うと予想外のパフォーマンス結果が生じることがあります。パフォーマンスが低下した場合は、そのチューニングを元に戻して別のチューニングを行う必要があります。簡単に元に戻せるように元の設定を保管しておけば、誤った情報を取り消すことはずっと簡単になります。

ディスク装置

すでに説明したとおり、システムのパフォーマンスには、システムを構成するハードウェアが影響を与える場合があります。ハードウェアがパフォーマンスに与える影響の例として、ディスク装置について考慮しましょう。

ディスク装置を管理する方法は、次の 4 通りの方法でパフォーマンスに影響を与えます。

- **ストレージを分割する方法**

  限られた量のストレージを、索引とデータ間で、表スペース間で、およびバッファー・プール間で分割する方法によって、さまざまな状況でそれぞれのストレージがどのように機能するかが大部分決まります。

- **ストレージのむだ**

  ストレージのむだは、それ自体が、そのストレージを使用しているシステムのパフォーマンスに影響を与えることはありませんが、他の場所でパフォーマンスを改善するのに使用できるリソースを示していることがあります。

- **ディスク入出力の配分**

  複数のディスク記憶装置とコントローラーにディスク入出力要求をバランスよく配分するかどうかは、データベース・マネージャーがディスクから情報を取り出す速度に影響を与えます。

- **ストレージの不足**

  使用可能なストレージの限界に達すると、全体のパフォーマンスが低下する可能性があります。

パフォーマンス改善のプロセス

システムのパフォーマンスを改善するには、次のプロセスを実行します。

1. パフォーマンスの目標を定義する。
2. パフォーマンスの指標を設定する。
3. パフォーマンスのモニター計画を立てる。
4. 計画を実行する。
5. 測定値を分析し、目標を達成できたかどうかを判断する。パフォーマンス・モニターそれ自体がシステム・リソースを使用するので、目標を達成できていれば、測定値の数を減らすことを考慮してください。目標を達成できていない場合は、次のステップに進んでください。
6. システムでの主要な制約を判別する。
7. トレードオフを行える場所と、余分の負荷を担えるリソースを決定する。 (ほとんどすべてのチューニングには、システム・リソースとパフォーマンスのさまざまな要素との間のトレードオフが関係しています。)
8. システムの構成を調整する。複数のチューニング・オプションを変更できると考える場合は、 1 つずつ変更してください。どのレベルにおいてもオプションが残っていない場合は、リソースの限界に達したことになり、ハードウェアをアップグレードする必要があります。
9. 上記のステップ 4 に戻り、システムのモニターを継続する。

定期的に、あるいはシステムまたは作業負荷に対して著しい変更を行った後は、次の事柄を実行してください。

- 上記のステップ 1 に戻る。
- 目標と指標を再検討する。
- モニターおよびチューニング戦略を改善する。

システムをどれほどチューニングできるか

システムの効率をどれほど改善できるかということについては限度があります。システム・パフォーマンスの改善にかかる時間と費用、また時間と費用をさらにかけることでシステムのユーザーにもたらされる利益を考慮してください。

チューニングをまったく行わなくてもシステムは十分に機能することもありますが、最大限に機能することはないでしょう。 すべてのデータベースは固有なものです。独自のデータベースとそれを使用するアプリケーションを開発したら、使用可能なチューニング・パラメーターを調査し、現在の状態を反映するようそれらのパラメーターの設定をカスタマイズする方法を調べてください。場合によってはシステムをチューニングしても得られる効果は小さいこともありますが、ほとんどの場合は著しい効果が得られます。

ウィザードは、コントロール・センター内から使用可能で、データベース・パラメーターをチューニングする点で役立ちます。パフォーマンス構成 (Performance Configuration) ウィザードを見付けるには、コントロール・センターでチューニングを行うデータベースの上でマウスの右ボタンをクリックしてください。

システムにパフォーマンス上の障害がある場合、チューニングの効果は高くなります。パフォーマンスの限界に近づいている場合に、システム上のユーザーの数を約 10 % 増やすと、応答時間は 10 % よりずっと長くなる可能性があります。この場合、システムをチューニングすることによって、このパフォーマンスの低下を相殺する方法を決定する必要があります。しかし、これ以上はチューニングを行っても効果がないという点が存在します。その点に達したときは、その環境内で目標と期待値を修正する必要があります。または、ディスク装置の増加、CPU の高速化、CPU の追加、メイン・メモリーの増加、通信リンクの高速化などを考慮に入れ、システム環境を変更する必要があります。

簡単なアプローチ

パフォーマンス目標を設定する時間や、包括的にモニターとチューニングを行う時間が十分にない場合は、ユーザーの意見を聞いてパフォーマンスの問題と取り組むことができます。ユーザーがパフォーマンスに関連した問題に直面していないかどうかを調べてください。 2、3 の簡単な質問をすることによって、問題を突き止めたり、問題を最初に探す場所を決定することができます。たとえば、ユーザーに次のような質問をしてみることができます。

- 「応答が遅い」とは、どのような意味でしょうか? 期待している速さより 10 % 遅いのでしょうか、それとも何十倍も遅いのでしょうか?
- 問題に気付いたのはいつですか? 最近ですか、それともこれまでずっとですか?
- 同じ問題を訴えている他のユーザーはいますか? 問題を訴えているのは 1 人か 2 人ですか、それともグループ全体ですか?
- (ユーザーのグループ全体が問題に直面している場合は、そのグループのユーザーは同じローカル・エリア・ネットワークに接続していますか?)
- 直面している問題は、特定のトランザクションまたはアプリケーション・プログラムに関連したものですか?
- 問題は、たとえば昼休みなどの決まった期間に生じますか、それとも連続的に生じますか?

まとめ

基本となる DB2 の構造は重要であり、この主要な概念およびプロセスを理解することが、他のパフォーマンスの問題を扱う上で役立ちます。ストレージの構造、データ管理、処理モデルおよびメモリー・モデルなどのトピックについては、次の章ではじめて説明されます。詳細については、11ページの『第2章 アーキテクチャーとプロセスの概要』を参照してください。

アプリケーション・パフォーマンスのチューニングには、アプリケーションに関連したパフォーマンスと、アプリケーションとデータベースとの対話に関連したパフォーマンスが関係します。アプリケーションそのものに関するトピックとしては、並行性、ロッキング、最適化クラス、照会の結果セットの制御、行ブロック化、複合 SQL の使用があります。さらに、文字変換 (アプリケーション・パフォーマンスに関連する部分)、ストアード・プロシージャー、データベースの活動化、および並列処理の利点についての短い説明もあります。詳細については、43ページの『第3章 アプリケーションについての考慮事項』を参照してください。

照会の最適化に関するトピックとしては、照会の最適化に影響を与える構成パラメーター、照会の最適化に対するノード・グループと表スペースの影響、および索引が照会の最適化に与える大きな影響があります。詳細については、91ページの『第4章 環境についての考慮事項』を参照してください。

システム・カタログ統計は、アプリケーションがデータにアクセスする性能に大きな影響を与えます。統計に関連したトピックとしては、RUNSTATS ユーティリティー、分散統計、索引統計、およびユーザーが更新できる統計があります。詳細については、113ページの『第5章 システム・カタログ統計』を参照してください。

SQL コンパイラーは、それぞれのアプリケーションを取り上げ、そのアプリケーションに最適なアクセス・プランを決定します。アプリケーション内のそれぞれの照会には、評価の後で、照会の目標を最も明確に定義するよう設計されたいくつかの異なる操作が行われます。次に、さまざまなアクセス方式 (走査および結合) がそれぞれの照会について検討され、照会が要求しているデータを取り出すのに最も速い方法が判別されます。並列性の影響についても考慮されます。詳細については、147ページの『第6章 SQL コンパイラーに関する解説』を参照してください。

DB2 製品とともに使用できる各種ツールは、アプリケーションの照会がどのように処理されているのかを理解するのに役立ちます。これらのツールでは、アプリケーションのパフォーマンスに影響を与えている要素を理解できます。詳細については、211ページの『第7章 SQL Explain 機能』を参照してください。

個々のアプリケーションのチューニングに加えて、それらのアプリケーションが実行されているデータベースのパフォーマンスも考慮する必要があります。データベースのパフォーマンスは、メモリーをどれほど効率的に使用するかによって大部分が決まります。パフォーマンスに関係しているメモリー周辺のトピックとしては、バッファー・プール、データの事前取り出し、並列入出力、分類機能、表内のデータの再編成、およびデータベース・エージェントの概念があります。詳細については、237ページの『第8章 操作上のパフォーマンス』を参照してください。

管理プログラムを設定すれば、アプリケーションがデータベースを使用する方法を管理することができます。詳細については、281ページの『第9章 管理プログラムの使用法』を参照してください。

プロセッサーの数とデータベース区画の数を増やせば、データベースのパフォーマンスを改善することができます。詳細については、297ページの『第10章 プロセッサーの追加による構成のスケーリング』を参照してください。

データベース区画の数を増やしたら、データベース区画間で、データベース内のデータが正しくスプレッドまたは再分配されていることを確認してください。詳細については、309ページの『第11章 データベース区画間でのデータの再配分』を参照してください。

データベースのパフォーマンスを判別するには、ベンチマーク・テストを実施することができます。ベンチマーク・テストの方法論、ベンチマーク・テストの準備の方法、ベンチマーク・プログラムの作成、およびベンチマーク・テストの実施は、すべて重要なトピックです。詳細については、317ページの『第12章 ベンチマーク・テスト』を参照してください。

331ページの『第13章 DB2 の構成』では、データベース・マネージャーおよびデータベースの構成パラメーターの非常に広範なセットが 1 つずつ説明されています。

これらのパフォーマンスのトピックには、関連した追加情報があります。付録には、次のトピックが含まれています。

- 501ページの『付録A. DB2 レジストリー変数と環境変数』
- 541ページの『付録B. Explain 表と定義』
- 573ページの『付録C. SQL EXPLAIN ツール』
- 619ページの『付録D. db2exfmt - Explain 表フォーマット・ツール』

第2章 アーキテクチャーとプロセスの概要

DB2 のデータベース操作のパフォーマンスを処理する際には、 DB2 のアーキテクチャーとプロセスについての基本的な理解が必要になります。この章では、 DB2 ユニバーサル・データベースの機能についての情報が十分に提供されています。ここで説明されているトピックのいくつかは後の章で詳しく扱われますが、この章にある情報は、後で理解するための背景として重要です。

最初の図に、DB2 UDB のアーキテクチャーとプロセスの概要を示します。

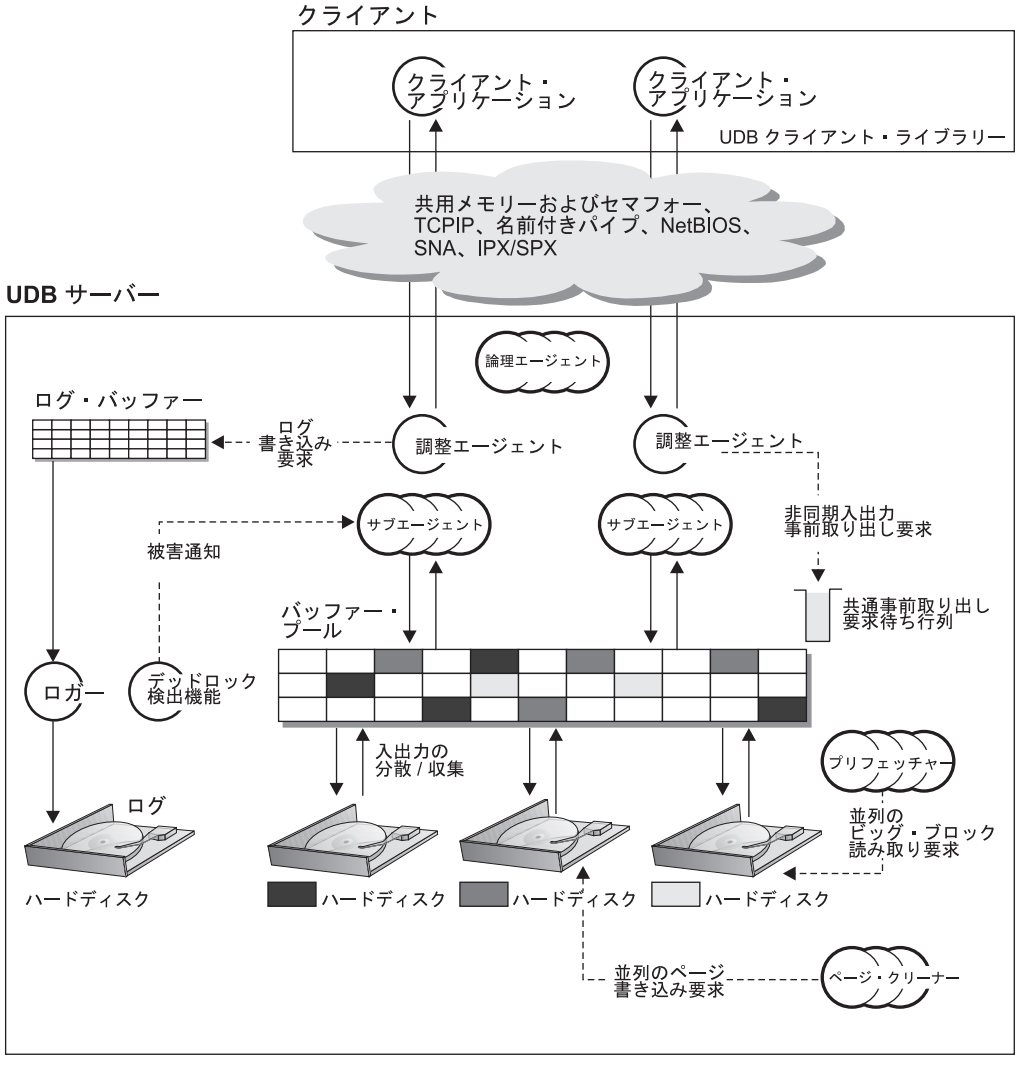


図 1. アーキテクチャーとプロセスの概要

クライアント側では、ローカル・アプリケーションやリモート・アプリケーションが、DB2 ユニバーサル・データベース・クライアント・ライブラリーにリンクされています。

クライアントと DB2 ユニバーサル・データベース・サーバーの間には『雲』があります。これは、ローカルまたはリモート・クライアントとサーバーとの間の通信を意味します。ローカル・クライアントは、共用メモリーおよびセマフォーを使用して通信します。リモート・クライアントは、名前付きパイプ (NPIPE)、TCP/IP、NetBIOS、IPX/SPX、または SNA などのプロトコルを使用します。

サーバー側では、アクティビティーは**エンジン・ディスパッチ可能単位** (EDU) により制御されます。この章のすべての図で、 EDU は、円または円グループとして示されています。 EDU は、Windows ベースのプラットフォームおよび OS/2 (すべて単一プロセス内) ではスレッドとして、 UNIX ではプロセスとしてインプリメントされています。最も一般的なタイプの EDU は、DB2 エージェントです。これらの EDU は、アプリケーションに代わって大量の SQL 処理を実行します。他の EDU としては、各種の入出力処理を実行する DB2 プリフェッチャーおよびページ・クリーナーがあります。詳しくは、269ページの『データベース・エージェント』を参照してください。

それぞれのクライアント・アプリケーションには 『調整エージェント』 と呼ばれる固有の EDU が割り当てられます。これは、アプリケーションの処理を調整し、アプリケーションと通信します。また、クライアント・アプリケーション要求を処理するために割り当てられるサブエージェントのセットもあります。サブエージェントは複数割り当てることができます。こうすると、対称マルチプロセス環境のように、サーバーが存在するマシンに複数のプロセッサーがある場合に、クライアント・アプリケーション要求でこれらのプロセッサーを利用できます。

すべてのエージェントおよびサブエージェントは、プール・アルゴリズムを使用して管理されます。これにより、EDU の作成や破棄の数を最小限にとどめることができます。

バッファー・プールは、ストレージ・メモリーの 1 つのエリアであり、ここで、ユーザー表データ、索引データ、およびカタログ・データが一時的に移動されたり、あるいは変更されたりします。データはディスクからよりもメモリーからの方がずっと速くアクセスできるため、バッファー・プールは、データベースの全体のパフォーマンスに影響を与える主要なものとなります。アプリケーションが必要とするデータのうち、バッファー・プールに存在しているものが多いほど、データにアクセスしてディスク記憶装置からデータを見つけるのにかかる時間は短くなります。詳しくは、245ページの『データベース・バッファー・プールの管理』を参照してください。

バッファー・プールの構成は、プリフェッチャーおよびページ・クリーナー EDU と共に、データへのアクセス・スピードと、その結果生じるアプリケーションで必要なデータの可用性を制御します。

プリフェッチャーは、データをディスクから取り出し、アプリケーションがそのデータを必要とする前にこれをバッファー・プールに移動するために存在しています。たとえば、データ・プリフェッチャーが存在しなければ、大量のデータ全体を走査する必要のあるアプリケーションは、データがディスクからバッファー・プールに移動するのを待機しなければなりません。アプリケーションのエージェントは、非同期読み取り先行要求を共通事前取り出しキューに送信します。使用可能になると、プリフェッチャーは大きなブロックを使用してこれらの要求をインプリメントするか、または読み取り入力操作を分散させてディスクからバッファー・プールに要求されたページを移動します。データベース・データのストレージに複数のディスクがあれば、データを複数ディスク間でストライピングすることができます。このデータ・ストライピングにより、プリフェッチャーは同時に複数のディスクを使用してデータを取り出すことができます。詳しく

は、
253ページの『バッファー・プールへのデータの事前取り出し』および 256ページの『事前取り出しおよび並列入出力を行うための入出力サーバーの構成』を参照してください。

プリフェッチャーは、データをバッファー・プールに入れるために使用されます。ページ・クリーナーは、データをバッファー・プールからディスクに戻すために使用されます。

ページ・クリーナーはバックグラウンド EDU であり、アプリケーション・エージェントからは独立して、バッファー・プールで必要でなくなったページを探して消去します。ページ・クリーナーにより、プリフェッチャーが取り出すページのスペースがバッファー・プール内に確保されます。

独立したプリフェッチャーやページ・クリーナー EDU がない場合には、バッファー・プールとディスク装置との間のデータの読み書きすべてをアプリケーション・エージェントが実行しなければなりません。

複数のアプリケーションがデータベースからのデータを処理している場合には、 2 つ以上のアプリケーションの間で 『デッドロック』 が発生する可能性があります。デッドロックについて、次の図に示します。

デッドロックの概念

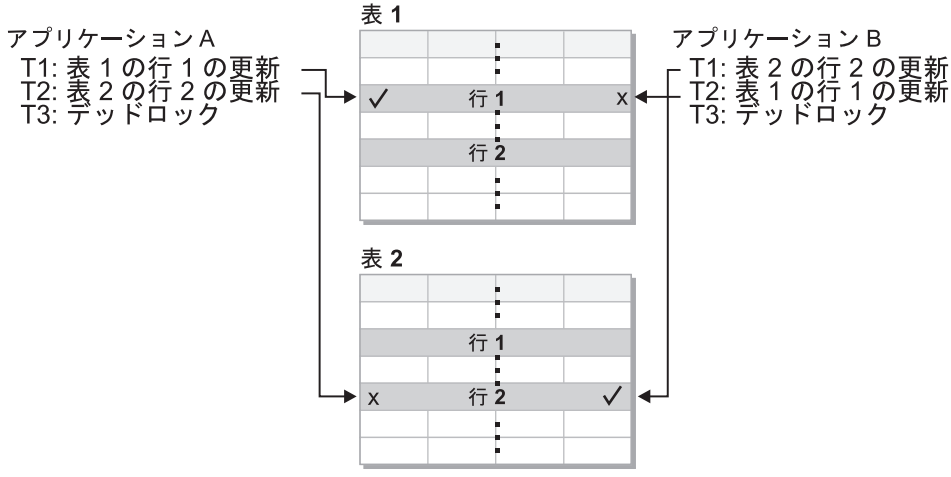


図 2. デッドロック検出機能

『デッドロック』 とは、複数のアプリケーションが、別のアプリケーションがデータのロックを解放するのを待機している状況です。待機しているアプリケーションのそれぞれが、他のアプリケーションで必要なデータをロックして保持しています。このロックされたデータが他の複数のアプリケーションで必要になっており、同時に、それらのア

プリケーションも他のアプリケーションで必要なデータを保持しています。他のアプリケーションが保持しているデータを解放するのを互いに待機している状態がデッドロックです。アプリケーションは、『他の』アプリケーションが保持データのロックを解放するのを永久に待機する可能性があります。他のアプリケーションは、自分の必要なデータを自発的には解放しません。これらのデッドロック状況を断ち切るプロセスが必要になります。

この名前で示すとおり、デッドロック検出機能は、ロックを待機しているエージェントについての情報をモニターします。デッドロック検出機能は、デッドロック状態のアプリケーションの 1 つを無作為に選択し、この 『志願した』 アプリケーションが保持しているロックを解放します。アプリケーションのロックを解放すると、他の待機しているアプリケーションで必要なデータが使用可能になります。これで、以前は待機中だったアプリケーションは、データベースでアクションを実行するのに必要なデータを自由に使用できるようになります。

バッファー・プールでのデータ・ページの変更はログに記録されます。ログ・バッファーが存在し、ロガー EDU に関連付けられます。データベースにあるデータ・レコードを更新しているエージェントは、バッファー・プールで関連するページを更新し、ログ・レコードを書き込みます。ログ・レコードには、変更の再実行または取り消しに必要な情報が含まれています。バッファー・プールのページも、ログ・バッファーにあるログ・レコードも、パフォーマンスを最適化するために即座にはディスクに書き込まれません。ロガー EDU およびバッファー・プール・マネージャーが協働して書き込み先行ロギング (WAL) プロトコルをインプリメントし、関連するログ・レコードがログに書き込まれる前にデータ・ページがディスクに書き込まれないようにします。 WAL プロトコルを使用すると、破損からリカバリーし、データベース整合性を復元するのに必要な情報が確実に用意されることになります。ページでの非コミット更新がディスクに書き込まれた場合、破損リカバリーでは、関連するログ・レコードにある取り消し情報を使用して、更新を取り消します。コミット更新がこれをディスクに作成しなかった場合、破損リカバリーでは、関連するログ・レコードにある再実行情報を使用して、更新を再実行します。

注: COMMIT 時には、トランザクションにあるすべてのログ・レコードがディスクにフラッシュされます (まだフラッシュされていない場合)。

ストレージのアーキテクチャー

ストレージ・アーキテクチャーの説明では、以下のことを考慮します。

- 『データベース・ディレクトリー』
- 17ページの『表スペース』

データベース・ディレクトリー

データベースを作成する際、デフォルト情報を含むデータベースについての情報がディレクトリーに入れられます。このディレクトリー構造は、データベース作成 (CREATE

DATABASE) コマンドで提供した情報に基づいた位置に作成されます。データベースを作成する際にパスまたはドライブの位置を指定しない場合には、デフォルト位置が使用されます。

データベースを作成する位置を明示的に指定するようにお勧めします。

CREATE DATABASE コマンドで指定するディレクトリーでは、インスタンスの名前を使用したサブディレクトリーが作成されます。このサブディレクトリーにより、同じディレクトリーの異なるインスタンスで作成されたデータベースが同じパスを使用しないことが保証されます。インスタンス名のサブディレクトリーの下に、 NODE0000 と名づけられたサブディレクトリーが作成されます。このサブディレクトリーは、複数の論理区分データベース環境で区画を区別するのに使用されます。ノード・ディレクトリーの後に、SQL00001 と名づけられたサブディレクトリーが作成されます。サブディレクトリーの名前は、データベース・トークンを使用して付けられ、作成中のデータベースを表します。また、CREATE DATABASE コマンドで指定したディレクトリーで、このインスタンス内で作成されたデータベースと区別するのにも使用できます。

ディレクトリー構造は、次のようになります。

```
<your_directory>/<your_instance>/NODE0000/SQL00001/
```

データベース・ディレクトリーには、データベース作成 (CREATE DATABASE) コマンドの一部として作成されたファイルがいくつか組み込まれます。バッファー・プール情報は、ファイル SQLBP.1 および SQLBP.2 に入れられます。表スペース情報は、ファイル SQLSPCS.1 および SQLSPCS.2 に入れられます。これらのファイルはそれぞれ 2 つずつあり、ファイル情報のバックアップが可能となっています。

データベース構成情報は、SQLDBCON に入れられます。ヒストリー・ファイル DB2RHIST.ASC およびそのバックアップ DB2RHIST.BAK は可読ファイルで、バックアップ、復元、表のロード、表の再編成、表スペースの変更、およびデータベースへの他の変更についてのヒストリー情報が含まれています。

ログ制御ファイル SQLOGCTL.LFH にはアクティブ・ログについての情報が入ります。リカバリー処理では、このファイルの情報を使用して、リカバリーのために戻るログ内の位置を判別します。 SQLOGDIR サブディレクトリーには、実際のログ・ファイルが入ります。

注: このログ・サブディレクトリーがご使用のデータに使用されているディスクとは異なるディスクにマップされていることを確認してください。こうすると、ディスクの問題が生じたときに、データとログの両方ではなく、データまたはログのいずれかに範囲を狭めることができます。また、ログ・ファイルおよびデータベース・コンテナーは、同じディスク・ヘッドの移動で競合することはないため、パフォーマンスにも大きな利点となります。ログ・サブディレクトリーの位置は、 *newlogpath* データベース構成パラメーターを使用すると変更できます。

SQLT* サブディレクトリーが作成され、作動データベースに必要なデフォルトのシステム管理スペース (SMS) 表スペースが含まれます。デフォルトの表スペースは 3 つ作成されます。

- SQLT0000.0 サブディレクトリーには、システム・カタログ表のカタログ表スペースが含まれます。
- SQLT0001.0 サブディレクトリーには、デフォルト一時表スペースが含まれます。
- SQLT0002.0 サブディレクトリーには、デフォルト・ユーザー・データ表スペースが含まれます。

表スペースを考慮する際には、『コンテナー』についても考慮しなければなりません。SMS 表スペースの場合、コンテナーはオペレーティング・システム・ディレクトリーです。

各サブディレクトリーまたはコンテナーには、 SQLTAG.NAM というファイルが作成されています。このファイルは、サブディレクトリーに使用中のマークを付け、後続の表スペース作成で、これらのサブディレクトリーが使用されないようにします。さらに、コンテナー・サブディレクトリーの下に他のファイルが作成されます。これらのファイルには、保管されているデータのタイプを識別するために異なる名前拡張子が付いています。拡張子は、以下のとおりです。

- SQL*.DAT (非ロング表データを含む)
- SQL*.LF (LONG VARCHAR または LONG VARGRAPHIC データを含む)
- SQL*.LB (BLOB、CLOB、または DBCLOB データを含む)
- SQL*.LBA (SQL*.LB ファイルについての割り当ておよびフリー・スペース情報を含む)
- SQL*.INX (索引表データを含む)
- SQL*.DTR (SQL*.DAT ファイルの再編成についての一時データを含む)
- SQL*.LFR (SQL*.LF ファイルの再編成についての一時データを含む)
- SQL*.RLB (SQL*.LB ファイルの再編成についての一時データを含む)
- SQL*.RBA (SQL*.LBA ファイルの再編成についての一時データを含む)

表スペース

サポートされる表スペースには 2 つのタイプがあります。システム管理スペース (SMS) とデータベース管理スペース (DMS) です。それぞれには、異なる環境で適した特性があります。表スペースの設計および選択について詳しくは、*管理の手引き: 計画* を参照してください。

SMS 表スペース

システム管理スペース (SMS) 表スペースは、オペレーティング・システム・ファイルのデータを保管します。表スペースのデータは、システム内のすべてのコンテナーにエクステント単位でストライピングされます。**エクステント**は、データベースに定義され

る連続ページのグループです。表スペースの各表には、すべてのコンテナーで使用される独自のファイル名が付けられます。ファイル拡張子は、ファイルに保管されているデータのタイプを示します。各表の先頭のエクステントは、コンテナー全体に 『ラウンドロビン』 形式で配置されます。このため、表スペースにあるすべてのコンテナーについて、スペース要件が均等に配分されることになります。これは、小さな表が数多くある場合に大変重要です。

追加のスペースが必要な場合には、スペースの割り当てが実行されます。デフォルトには、一度に 1 ページのスペースが割り当てられます。

DMS 表スペース

データベース・マネージャーは、データベース管理スペース (DMS) 表スペースを使用して記憶スペースを制御します。 DMS 表スペースが定義されるときには、表スペースに属するように装置またはファイルのリストが選択されます。これらの装置またはファイルのスペースは、 DB2 データベース・マネージャーにより管理されます。 SMS 表スペースおよびコンテナーと同様、 DMS 表スペースおよびデータベース・マネージャーもエクステント単位のストライピングを使用して、すべてのコンテナー間でデータが均等分配されるようにします。

DMS 表スペースと SMS 表スペースの相違点は、DMS 表スペースでは、必要なときではなく、表スペースが作成される際にスペースが割り当てられることです。

また、この 2 種類の表スペースでは、データの配置が異なることがあります。たとえば、効果的な表走査について考慮しましょう。エクステントのページが物理的に連続していることは重要です。 SMS の場合、オペレーティング・システムのファイル・システムが、各論理ページを物理的に配置する位置を決定します。ファイル・システムの他のアクティビティーのレベル、および配置を判別するのに使用されるアルゴリズムによっては、ページが連続して割り当てられることも、割り当てられないこともあります。ところが、DMS の場合、データベース・マネージャーがディスクとのインターフェースを直接とるため、確実にページが物理的に連続するようにできます。

このストレージでのページの連続配置については、この一般論が適用できない例外が 1 つあります。 DMS 表スペースを処理する際には、2 つのコンテナー・オプションがあります。ロー・デバイスとファイルです。ファイル・コンテナーを処理する際、データベース・マネージャーは、表スペースの作成時にコンテナー全体を割り当てます。表スペース全体のこの初期割り当ての結果として、物理的な割り当ては通常は連続していますが、ファイル・システムがこの割り当てを実行しているとしても、連続するとは保証されません。ロー・デバイス・コンテナーを処理する場合には、データベース・マネージャーがデバイス全体を制御し、常にエクステントのページを確実に連続配置できます。

SMS 表スペースとは異なり、DMS 表スペースを構成するコンテナーの場合は、容量を互いに均等に近づける必要はありません。ただし、コンテナーの容量は均等、あるいは

均等に近いようにすることが勧められています。また、コンテナーで満杯のものがあれば、他のコンテナーで使用可能なフリー・スペースを DMS 表スペースで使用することができます。

DMS 表スペースを処理する場合には、コンテナーを異なるディスクに関連付けることを考慮しなければなりません。これにより、表スペースの容量は大きくなり、並行入出力操作を利用する機能も改善されます。

CREATE TABLESPACE ステートメントは、データベースで新しい表スペースを作成し、この表スペースにコンテナーを割り当て、カタログに表スペース定義と属性を記録します。表スペースの作成にあたって定義されるもののなかに、エクステント・サイズがあります。エクステントは、表スペース内のスペース割り当ての単位です。これは、単なる連続ページのセットです。エクステント・サイズは、連続ページの数です。 1 つの表 (または、索引などのその他のオブジェクト) では、単一エクステント内のページしか使用できません。表スペースで作成されるすべてのオブジェクト (表、索引、その他) には、論理スペース・アドレス・マップでエクステントが割り当てられます。 1 つのエクステントは、一度に 1 つのオブジェクトにのみ属します。エクステントの割り当ては、スペース・マップ・ページ (SMP) により管理されます。

論理表スペース・アドレス・マップの先頭のエクステントは、表スペースのヘッダーで、これには内部制御情報が含まれます。 2 番目のエクステントは、表スペースのスペース・マップ・ページ (SMP) の最初のエクステントです。 SMP エクステントは、表スペースで等間隔に分散されます。それぞれの SMP エクステントは、現行の SMP エクステントから次の SMP エクステントへのエクステントのビットマップに過ぎません。このビットマップは、どの中間エクステントが使用中かまたは使用中でないかをトラックするのに使用されます。

SMP に続くエクステントは、表スペースのオブジェクト表です。オブジェクト表は、表スペースにどのユーザー・オブジェクトが存在するか、またどこに最初のエクステント・マップ・ページ (EMP) エクステントが配置されているかをトラックする内部表です。各オブジェクトには、それぞれ EMP があり、これは、論理表スペース・アドレス・マップに保管されているオブジェクトの各ページへのマップを提供します。

次の図に、DMS 表スペースの論理アドレス・マップを示します。

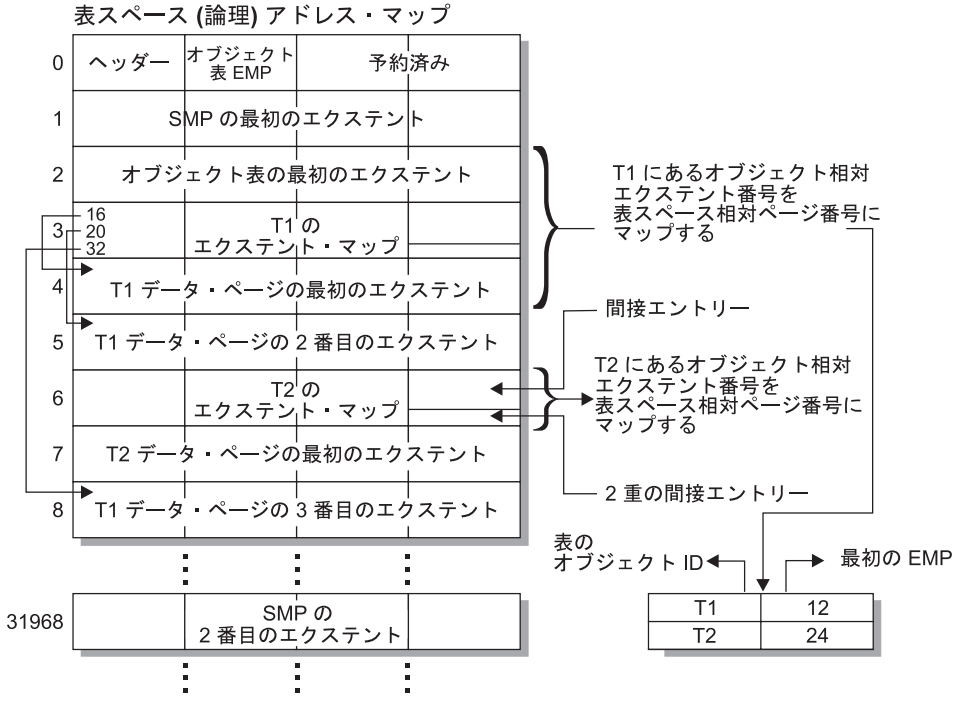


図 3. DMS 表スペース

オブジェクト表は内部リレーショナル表で、オブジェクト ID を表の最初の EMP エクステントのロケーションにマップします。この EMP エクステントは、直接的にも間接的にも、オブジェクトにあるすべてのエクステントをマップします。各 EMP には、エントリーの配列が含まれています。各エントリーは、オブジェクト相対エクステント番号を、オブジェクト・エクステントが配置されている表スペース相対ページ番号にマップします。直接 EMP エントリーは、オブジェクト相対アドレスをスペース相対アドレスに直接マップします。最初の EMP エクステントにある最後の EMP ページには、間接エントリーが含まれます。間接 EMP エントリーは EMP ページにマップし、続いて EMP ページがオブジェクト・ページにマップします。最初の EMP エクステントにある最後の EMP ページの末尾の 16 個のエントリーには、 2 重の間接エントリーが含まれます。

論理表スペース・アドレスからのエクステントは、表スペースに関連付けられているコンテナー全体にラウンドロビン方式でストライピングされます。

SMS および DMS 表スペースの比較

SMS と DMS 表スペースを比較すると、一般的には SMS 表スペースが適しています。 SMS 表スペースは、わずかな管理コストで大変優れたパフォーマンスを提供しています。最高のパフォーマンスを求める場合には、DMS 表スペースを選択するのが最善でしょう。ファイル・コンテナーまたは SMS 表スペースを使用してデータを移動す

る際に、ダブル・バッファリングが発生する可能性があるため、デバイス・コンテナーが最善のパフォーマンスを提供するといえます。 (ダブル・バッファリングは、最初にデータベース・マネージャー・レベルでデータがバッファーに入れられ、次いでファイル・システム・レベルでもバッファーに入れられた場合に発生します。)

データ管理

データベースの作成、表スペースの作成、表の作成、表へのデータの配置に続き、表の編成および表データを検索するための索引の使用方法について説明します。

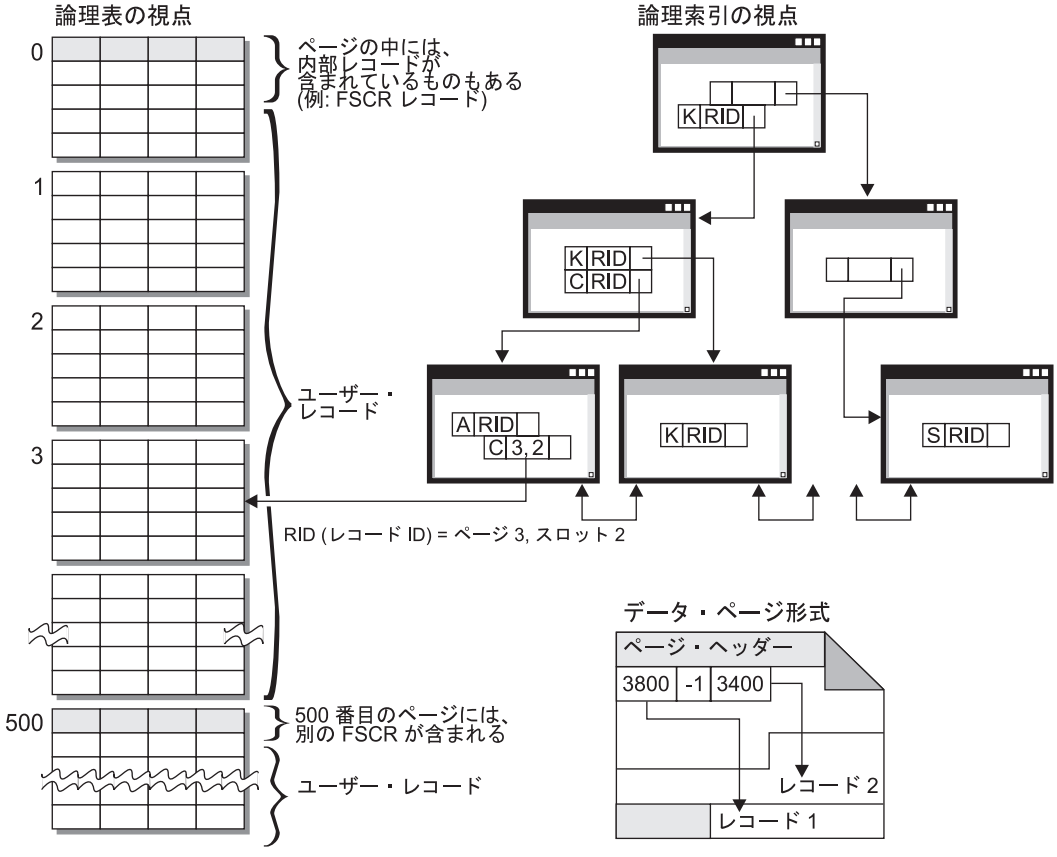


図 4. 表、レコード、および索引

論理的には、表データはデータ・ページのリストとして編成されます。そして、これらのデータ・ページは、表スペースのエクステント・サイズに基づいて論理的にグループ分けされます。たとえば、エクステント・サイズが 4 の場合、 0 ページから 3 ページが最初のエクステントに入り、 4 ページから 7 ページが 2 番目のエクステントに入るようになります。

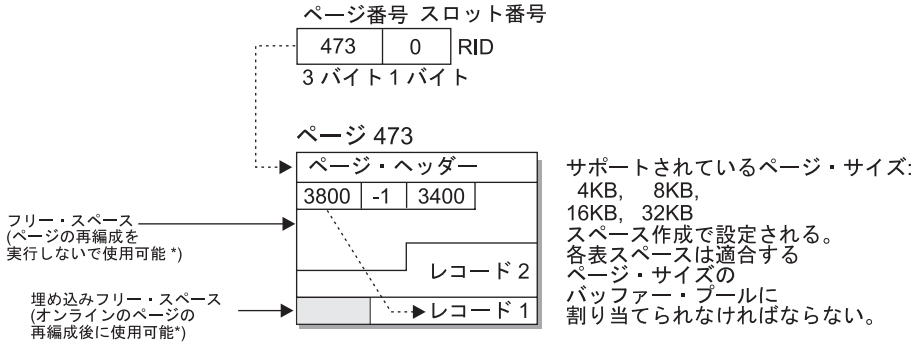
それぞれのデータ・ページに入るレコードの数は、データ・ページのサイズやレコードのサイズによって異なります。 1 ページに最大 255 個のレコードが入ります。大半のページには、ユーザー・レコードしか入りません。ただし、一部の少数のページには特殊な内部レコードが含まれます。これは表を管理するのに DB2 によって使用されます。たとえば、データ・ページで 500 ページごとにフリー・スペース制御レコード (FSCR) があるとします。これらのレコードは、 FSCR に続く 500 個のデータ・ページ (次の FSCR まで) に存在する新しいレコードのフリー・スペースの量をマップします。この使用可能なフリー・スペースは、表にレコードを挿入する際に使用されます。

論理的には、索引ページは B ツリーとして編成されています。 B ツリーでは、特定のキー値を持つ表データでレコードを効率的に配置できます。索引ページでのエンティティーの数は固定しておらず、キーのサイズによって異なります。 DMS 表スペースの表では、索引ページにあるレコード ID (RID) は、オブジェクト相対ページ番号ではなく表スペース相対ページ番号を使用します。これによって、**索引走査**は、マップのためにエクステント・マップ・ページ (EMP) を要求することなく、データ・ページに直接アクセスできるようになります。

データ・ページの形式は、次のとおりです。データ・ページの先頭にはページ・ヘッダーがあります。ページ・ヘッダーの後には、スロット・ディレクトリーがあります。スロット・ディレクトリーの各エントリーは、ページの異なるレコードに対応します。エントリー自体は、レコードが開始するデータ・ページへのバイト・オフセットです。マイナス 1 (-1) のエントリーは、削除されたレコードに対応します。

レコード ID とページ

レコード ID (RID) は、 3 バイトのページ番号の後に 1 バイトのスロット番号が付いたものです。索引を使用して RID が識別されると、その RID は正確なデータ・ページおよびそのページのスロット番号を取得するのに使用されます。スロットの内容は、ページ内のバイト・オフセットから、検索されるレコードの先頭まであります。レコードに割り当てられた RID は、表の再編成まで変わりません。



* 例外: 非コミット DELETE により予約されるスペースは私用できない。

図 5. データ・ページと RID の形式

表が再編成されるとき、レコードの削除の後でデータ・ページに残された埋め込みフリー・スペースは、使用可能なフリー・スペースに変換されます。 RID がデータ・ページのレコードの移動に基づいて再定義され、使用可能なフリー・スペースが活用されます。

DB2 では、さまざまなページ・サイズがサポートされています。行に順次アクセスする傾向のある作業負荷には大きいページ・サイズを使用します。たとえば、順次アクセスが意思決定支援アプリケーションに使用される場合や、一時表が集中的に使用される場合などです。アクセスがランダムである傾向の作業負荷には、小さいページ・サイズを使用してください。たとえば、ランダム・アクセスは OLTP 環境で使用されます。

表の再編成について詳しくは、264ページの『カタログとユーザー表の再編成』を参照してください。

スペース管理

新しい情報を表に入れるには SQL INSERT ステートメントを使用します。これを実行すると、この後に INSERT 探索アルゴリズムが実行されて作業が完了します。最初に、フリー・スペース制御レコード (FSCR) が使用されて、十分なスペースのあるページが見つけられます。ただし、 FSCR にフリー・スペースが十分にある場合でも、他のトランザクションの非コミット DELETE により 『予約』 されているために、このスペースを使用できない可能性もあります。結果として、すべてのトランザクションが頻繁に COMMIT されるようにしなければなりません。そうでないと、コミットされていないフリー・スペースが使用できなくなります。

表にあるすべての FSCR が検索されるわけではありません。 DB2MAXFSCRSEARCH レジストリー変数により、 INSERT の実行時に考慮される FSCR の数は制限されま

す。このレジストリー変数のデフォルト値は 5 です。 5 つの FSCR にスペースが見つからない場合、挿入しようとしているレコードは表の末尾に追加されます。また、INSERT の速度を最適化するために、 2 つのエクステントが満杯になるまで、後続のレコードは表の末尾に追加されます。この 2 つのエクステントが満杯になったら、次の INSERT から、先回終了した位置からの FSCR の検索が再開されます。

注: DB2MAXFSCRSEARCH の値は重要です。 INSERT の速度を最適化するには (表がすぐに大きくなる可能性がデメリットとしてある)、このレジストリー変数を小さい値に設定してください。スペースの再使用を最適化するには (INSERT 速度が遅くなる可能性がデメリットとしてある)、このレジストリー変数を大きい値に設定してください。

表全体が一度検索されると、挿入されるレコードは、検索をさらに実行せずに追加されます。 FSCR を使用する検索は、スペースが表のどこかに作成されるまで (たとえば、DELETE の後まで) 再実行されません。

他にも 2 つの検索アルゴリズム・オプションがあります。最初は、APPEND MODE です。このモードでは、新しい行が常に表の末尾に追加されます。 FSCR の検索や保守は実行されません。このオプションは ALTER TABLE APPEND ON ステートメントを使用して有効にすることができ、ジャーナルなどの大きくなるだけの表のパフォーマンスを改善することができます。 2 つ目の方法として、表でクラスター索引を定義することができます。この場合、データベース・マネージャーはレコードを、類似した索引キー値のある他のレコードと同じページに挿入しようとします。このページにスペースがない場合には、周辺のページにレコードを入れるように試みがなされます。それでも正常に実行されない場合には、上記の FSCR 検索アルゴリズムが使用されます。ただし、ここでは、最初に見つかったスペースが使用されるのではなく、最も大きさが異なるスペースが使用されるという違いがわずかにあります。このアプローチは、フリー・スペースがより大きいページを選択する傾向があります。これが実行されるのは、このキー値の行の新しいクラスター域を確立するためです。

表にクラスター索引を定義する場合には、表のロードまたは再編成の前に ALTER TABLE... PCTFREE を使用します。 PCTFREE 文節は、ロードおよび再編成の後、指定されたパーセンテージ値をこの表のデータ・ページでのフリー・スペースとして残しておきます。これにより、クラスター索引走査が望むページにフリー・スペースを検出できる可能性を増すことになります。

データ・ページとオーバーフロー・レコード

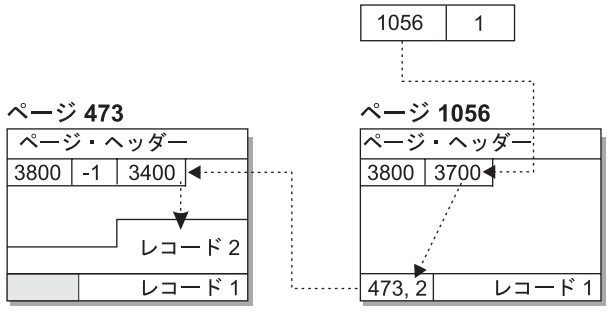


図6. データ・ページとオーバーフロー・レコード

オーバーフロー・レコードは、更新要求で既存のレコードが大きくなり、現行のページに入らなくなる場合に起こります。大きくなったレコードは、オーバーフロー・レコードとして十分なスペースのある他のページに挿入されます。元の RID は、ポインター・レコードに変換され、ここに、オーバーフロー・レコードの新しい RID が組み込まれます。表の索引では、元の RID が保持されるため、要求されたデータ・レコードを取得するには、余分のページ読み取りが必要です。オーバーフロー・レコードがたくさんあると、余分のページ読み取りも多くなるために、表にアクセスする際のパフォーマンスが低下します。オーバーフロー・レコードは、表の再編成でなくすことができます。ただし、可能な場合はいつでも、レコードを大きくする更新要求を避けることにより、オーバーフロー・レコードを回避してください。

索引管理

DB2 索引は、書き込み先行ログを使用した効率的かつ並行性の高い索引管理メソッドに基づく、最適化された B ツリー・インプリメンテーションです。

最適化された B ツリー・インプリメンテーションでは、双方向のポインターがリーフ・ページにあるため、単一索引で前方または後方のいずれの方向でも走査できます (両方向を同時に走査することはできません)。通常、索引は、ちょうど半々に分割されます。例外として、上位キー・ページでは、90/10 の分割が使用されます。つまり、索引キーの上位 10 % は、新しいページに入れられます。このタイプの索引ページの分割は、新しい上位キーを使用して INSERT 要求が頻繁に実行される作業負荷の場合に役立ちます。

ページの最後の索引キーが除去されると、索引内のページが解放されます。ただし、索引を作成する際に MINPCTUSED 文節が選択された場合は例外です。この文節を使用した場合、この索引はオンラインで再編成できます。このとき、この文節で指定された値は、索引リーフ・ページで使用されるスペースの最小パーセンテージのしきい値です。索引ページからキーが削除された後で、ページで使用されているスペースが指定された値以下である場合には、残りのキーを近隣のページのキーにマージする試みが実行され

ます。空きが十分にある場合には、マージが実行され、リーフ・ページが削除されます。この文節を使用すると、スペース再利用が改善される可能性があります。ただし、使用される値が高すぎると、マージを実行するのにかかる時間が長くなるだけでなく、成功する可能性も低くなります。この文節の値は、常に 50 % 未満にするようにお勧めします。

CREATE INDEX ステートメントの INCLUDE 文節を使用すると、指定されたカラムをキー・カラムに加えて索引リーフ・ページに組み込むことができます。これで、索引のみのアクセスで適格である照会の数を増やすことができます。ただし、同時に索引スペースの要件も増やすことになり、組み込まれた列が頻繁に更新される場合には、索引の保守にかかるコストも増える可能性があります。索引 B ツリーの順序付けは、キー・カラムを使用することによってのみ実行できます。組み込みカラムでは実行できません。

ロック

データベース・マネージャーは、並行性制御を可能にし、ロックを使うことによって、リソースおよびデータに無秩序にアクセスがなされることがないようにします。**ロック**は、アプリケーションをデータベース・マネージャー・リソースまたはデータ・レコードに関連付けます。ロックは、他のアプリケーションが同じリソースまたはデータ・レコードにアクセスする方法を制御します。

データベース・マネージャーは、以下に基づいてレコード・レベルのロッキングおよび表レベルのロッキングを適宜使用します。

- プリコンパイル時、またはアプリケーションがデータベースにバインドされるときに指定される分離レベル。分離レベルは、非コミット読み取り (UR)、カーソル固定 (CS)、読み取り固定 (RS)、または反復可能読み取り (RR) のいずれかです。これらの異なる分離レベルは、非コミット・データへのアクセス、更新消失の防止、データの反復不能読み取りの許可、および幻像読み取りの防止を制御するために使用されます。ご使用のアプリケーションが必要とする最低の分離レベルを使用してください。
- 最適化プログラムにより選択されるアクセス・プラン。表走査、索引走査、および他のデータ・アクセス方式のそれぞれについて、異なるタイプのデータ・アクセスが必要です。
- 表の LOCKSIZE 属性。 ALTER TABLE ステートメントの LOCKSIZE 文節で、表にアクセスする際に使用するロックの細分性を示します。 ROW (行ロックの場合) または TABLE (表ロックの場合) を選択できます。読み取り先頭の表には ALTER TABLE... LOCKSIZE TABLE を使用してください。これにより、データベース・アクティビティーで必要なロックの数を減らすことができます。
- ロッキング専用のメモリーの量。ロッキング専用のメモリーの量は、 *locklist* データベース構成パラメーターにより制御されます。ロック・リストが満杯になると、ロック・エスカレーションおよびデータベースでの共用オブジェクトの並行性が悪化することが原因で、パフォーマンスが低下する可能性があります。ロック・エスカレーションが頻繁に発生する場合には、 *locklist* や *maxlocks* の値を大きくしてください。

すべてのトランザクションが頻繁に COMMIT され、保持されているロックが解放されるようにしてください。

通常、以下の場合以外は、レコード・レベルのロッキングが使用されます。

- 分離レベルに非コミット読み取り (UR) が選択されている場合。
- 選択された分離レベルが反復可能読み取り (RR) で、アクセス・プランに、述部なしの走査が必要な場合。
- 表の LOCKSIZE 属性が 『TABLE』 である場合。
- ロック・リストが満杯で、ロック・エスカレーションが発生している場合。
- LOCK TABLE ステートメントを介して獲得された明示的な表ロックがある場合。LOCK TABLE ステートメントを使用すると、並行アプリケーション・プロセスが表を変更したり表を使用したりできないようになります。

ロック・エスカレーションとは、 1 つまたは複数のレコード・ロックが表ロックに変換されることです。排他ロック・エスカレーションとは、獲得された表ロックが排他ロックであるロック・エスカレーションのことです。ロック・エスカレーションは並行性を低下させるので、回避する必要があります。

レコード・ロッキングの期間は、使用されている分離レベルによって異なります。

- 非コミット読み取り (UR) 走査。レコードが変更しないかぎり、レコード・ロックはありません。
- カーソル固定 (CS) 走査。カーソルがレコードに位置している場合にのみレコード・ロックが保持されます。
- 読み取り固定 (RS) 走査。トランザクション中、限定したレコード・ロックのみ保持されます。
- 反復可能読み取り (RR) 走査。トランザクションの間、すべてのレコード・ロックが保持されます。この分離レベルが必要でないか、望ましくない環境では、DB2_RR_TO_RS レジストリー表を使用してください。これは、データベース・マネージャーが RR セマンティクスを使用可能にするのに必要な余分のロッキングを避けるように指示します。その結果としてパフォーマンスは向上します。

このトピックについて詳しくは、51ページの『ロック』を参照してください。

ロギング

ロギング・ストラテジーは、以下の 2 つから選択できます。

- 循環ログ。ログ・レコードがログ・ファイルを満たすと、最初のログ・ファイルの最初のログ・レコードが上書きされます。上書きされたログ・レコードをリカバリーすることはできません。
- ログ・ファイルがログ・レコードでいっぱいになったら、このログ・レコードは保存され、アーカイブされます。新しいログ・ファイルが、ログ・レコード用に使用可能になります。ログ・ファイルの保存により、**ロールフォワード・リカバリー**が可能に

なります。ロールフォワード・リカバリーは、ログに記録されている完了した作業単位 (トランザクション) に基づいてデータベースに変更を適用しなおします。このロールフォワード・リカバリーは、ログの最後まで実行するか、またはそれより前の特定の時点で終了するかを指定することができます。

いずれを選択する場合でも、正規データおよび索引ページになされた変更はすべて、ログ・バッファーに書き込まれます。以下の場合にのみ、ログ・バッファーにあるデータはディスクに強制書き込みされます。

- 対応するデータ・ページがディスクに強制書き込みされる前。これは、『書き込み先行ロギング』 と呼ばれます。
- COMMIT 時または、データベース構成パラメーターをグループ化する COMMITS の数の値 (*mincommit*) に到達した後。
- ログ・バッファーがほぼ満杯になった場合。ロガー・プロセスは、ログ・データをディスクに書き込み、『ログ・バッファー・フル』 の状態にならないようにします。

注: COMMIT ステートメントを使用してトランザクションが完了すると、変更されたページすべてがディスクにフラッシュされてリカバリー可能性が保証されます。

トランザクションが短い場合、 COMMIT 時のフラッシュが頻繁に起きるために、ログの入出力が 『障害』 になります。このような環境では、 *mincommit* 構成パラメーターを 1 より大きい値に設定すると、『ボトルネック』を取り除くことができます。 1 より大きい値が使用される場合には、複数のトランザクションの COMMIT が保持されるか、または 『バッチ』 されます。 COMMIT される最初のトランザクションは、続く (*mincommit* - 1) 個のトランザクション COMMIT が実行されるまで待機します。それから、ログがディスクに強制書き込みされ、すべてのトランザクションがアプリケーションに応答します。結果として、個々のログ入出力が複数実行されるのではなく、 1 つだけログ入出力が実行されることになります。

応答時間が極端に遅れるのを避けるために、各トランザクションは、 (*mincommit* - 1) 個の他のトランザクションが COMMIT されるのを 1 秒までしか待機しません。 1 秒が経過すると、待機しているトランザクションはログ自体に強制書き込みされ、このアプリケーションに応答します。これにより、 *mincommit* を設定すると同時に、数の減ったトランザクションが処理されている時間のパフォーマンスを心配しなくてもよくなります。

ラージ・オブジェクト (LOB) および LONG VARCHAR への変更は、シャドー・ページングによりトラックされます。ログ保存が使用され、 LOB カラムが CREATE TABLE ステートメントで NOT LOGGED 文節を使用しないものとして定義されていないかぎり、 LOB カラムの変更はログ記録されません。 LONG または LOB データ・タイプの割り当てページへの変更は、正規データ・ページと同様にログ記録されます。

更新時の処理

エージェントがページを更新すると、ログやデータ・ページはどうなるでしょうか? ここで説明されているプロトコルは、トランザクションで必要な入出力を最小化し、同時にリカバリー可能性を保証します。

最初に、更新されるページがピンされ、排他ロックでラッチされます。ログ・バッファーにログ・レコードが書き込まれ、この変更を再実行したり、取り消したりする方法が説明されます。このアクションの一部として、ログ順序番号 (LSN) が取得され、更新されているページのページ・ヘッダーに保管されます。この時点で、ページが変更されます。最後に、ページがアンラッチおよび固定解除されます。このページは、「ダーティー」ページと見なされます。このページに変更があるものの、ディスクにこれが書き出されていないためです。ログ・バッファーも更新されています。

ログ・バッファーおよび「ダーティー」データ・ページの両方とも、ディスクに強制書き込みされる必要があります。パフォーマンスが低下しないようにするため、これらの入出力は、適当なとき (たとえば、システム負荷が落ち着いたとき) まで、あるいはリカバリー可能性を保証しなければならなくなったときまで、または限界リカバリー時間まで後回しにされます。厳密に言うと、以下の場合に「ダーティー」ページがディスクに強制書き込みされます。

- 他のエージェントがこれを被害ページとして選択した場合。
- 次の結果として、ページでページ・クリーナーが実行される場合。
  - 他のエージェントがこれを被害ページとして選択している。
  - *chngpgs_thresh* データベース構成パラメーターのパーセンテージ値を超えた。この値を超えると、非同期ページ・クリーナーが 『起動し』、変更されたページをディスクに書き込みます。
  - *softmax* データベース構成パラメーターのパーセンテージ値を超えた。この値を超えると、非同期ページ・クリーナーが 『起動し』、変更されたページをディスクに書き込みます。
- NOT LOGGED INITIALLY 文節が呼び出された表の一部としてページが更新されており、 COMMIT が発行される場合。 COMMIT 時には、変更されたページすべてがディスクにフラッシュされてリカバリー可能性が保証されます。

以下の場合に、ログ・バッファーはロガー・エンジン・ディスパッチ可能単位 (EDU) によりディスクに強制書き込みされます。

- 対応するデータ・ページがディスクに強制書き込みされる前。これは、『書き込み先行ログ』 と呼ばれます。
- COMMIT 時、または、データベース構成パラメーターをグループ化する COMMITS の数の値 (*mincommit*) に到達した後。
- ログ・バッファーがほぼ満杯になった場合。ロガー・プロセスは、ログ・データをディスクに書き込み、『ログ・バッファー・フル』 の状態にならないようにします。

プロセス・モデル

ローカルおよびリモート・アプリケーション・プロセスは、同一のデータベースを処理できます。リモート・アプリケーションとは、データベース・マシンから離れているマシンからデータベース・アクションを開始するアプリケーションのことです。ローカル・アプリケーションは、サーバー・マシンでデータベースに直接接続されています。

以下の図にある円は、エンジン・ディスパッチ可能単位 (EDU) を示します。これらは、UNIX プラットフォームでは 『プロセス』、 Windows NT および OS/2 プラットフォームでは 『スレッド』 と呼ばれます。

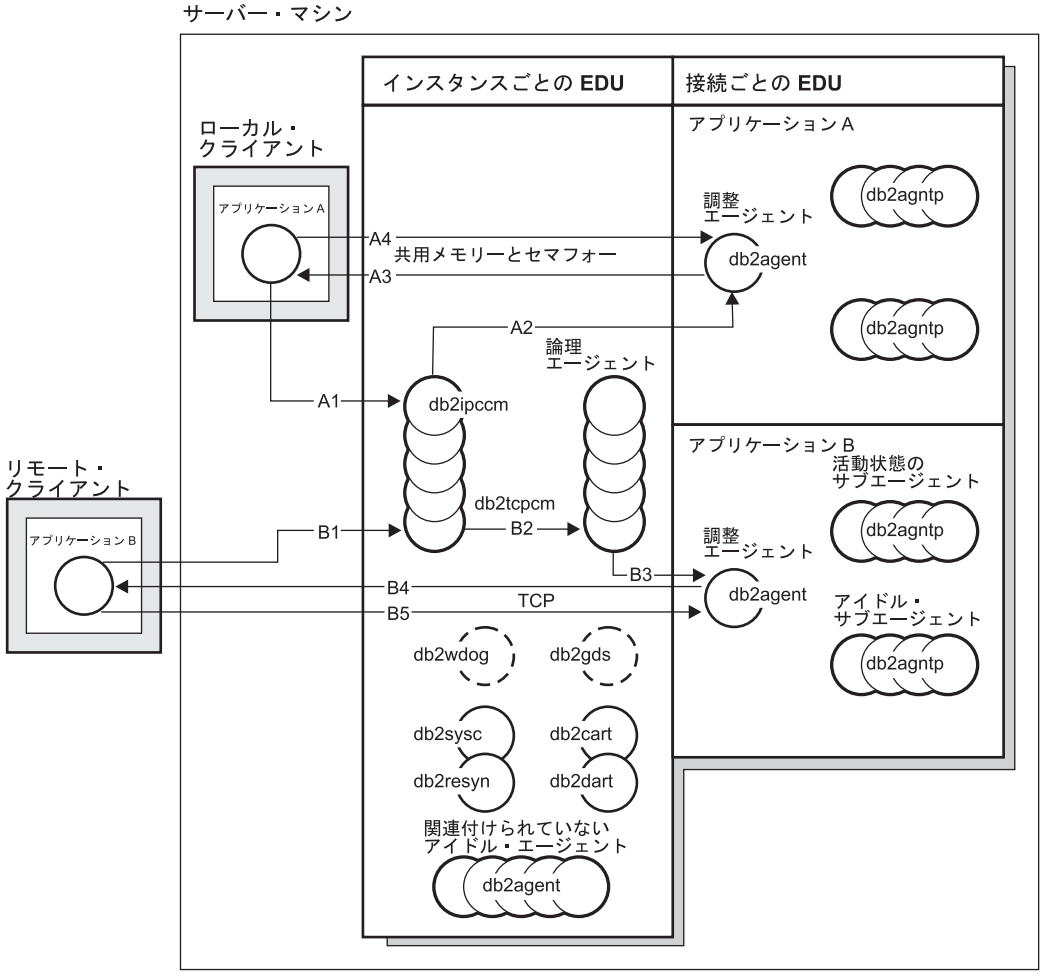


図 7. プロセス・モデルの概要

アプリケーションとデータベース・マネージャーとの間の通信の手段は、アプリケーションがデータベースで実行しようとしている作業が完了する前に確立しておかなければなりません。

上図の A1 では、ローカル・クライアントは、最初に db2ipccm エンジン・ディスパッチ可能単位 (EDU) を処理して、通信を確立します。 A2 で、この EDU は db2agent EDU を処理します。これは、ローカル・クライアントからのアプリケーション要求の調整エージェントになります。調整エージェントは、A3 でクライアント・アプリケーションに接触して、 A4 でクライアント・アプリケーションとデータベースとの間の共用メモリーおよびセマフォー通信を確立します。こうして、ローカル・クライアントのアプリケーションは、データベースに接続されます。

上図の B1 では、リモート・クライアントは、最初に db2tcpcm EDU を処理して、通信を確立します。他の通信プロトコルが選択された場合には、適切な EDU が使用されます。 B2 では、db2tcpcm EDU が論理エージェントを処理します。 B3 で、この EDU は db2agent EDU を処理します。これは、リモート・クライアントからのアプリケーション要求の調整エージェントになります。調整エージェントは、B4 でクライアント・アプリケーションに接触して、 B5 でクライアント・アプリケーションとデータベースとの間の TCP/IP 通信を確立します。こうして、リモート・クライアントのアプリケーションは、データベースに接続されます。

この図で他に注意することは以下のとおりです。

- エージェントには、2 つのクラスがあります。論理エージェントと作業エージェントです。論理エージェントは、データベース・マネージャーへの接続されたアプリケーションを表します。作業エージェントは、アプリケーション要求を実行しますが、特定のアプリケーションに永続的に接続されるものではありません。
- 作業エージェントには、4 つのタイプがあります。活動状態の調整エージェント、サブエージェント、非活動状態のエージェント、およびアイドル・エージェントです。
- 論理エージェントで表されるクライアント・アプリケーションの各プロセスまたはスレッドは、活動状態の調整エージェントにリンクされます。
- 区分データベース環境、および区画内並行性環境では、調整エージェントは、データベース要求をサブエージェント (db2agntp) に配布します。サブエージェントは、アプリケーションへの要求を実行します。
- エージェント・プール (db2agent) では、アイドル・エージェントが新しい作業が来るのを待機します。
- クライアント接続、2 フェーズ COMMIT、バックアップおよび復元タスク、およびその他のタスクを管理するその他の EDU もあります。

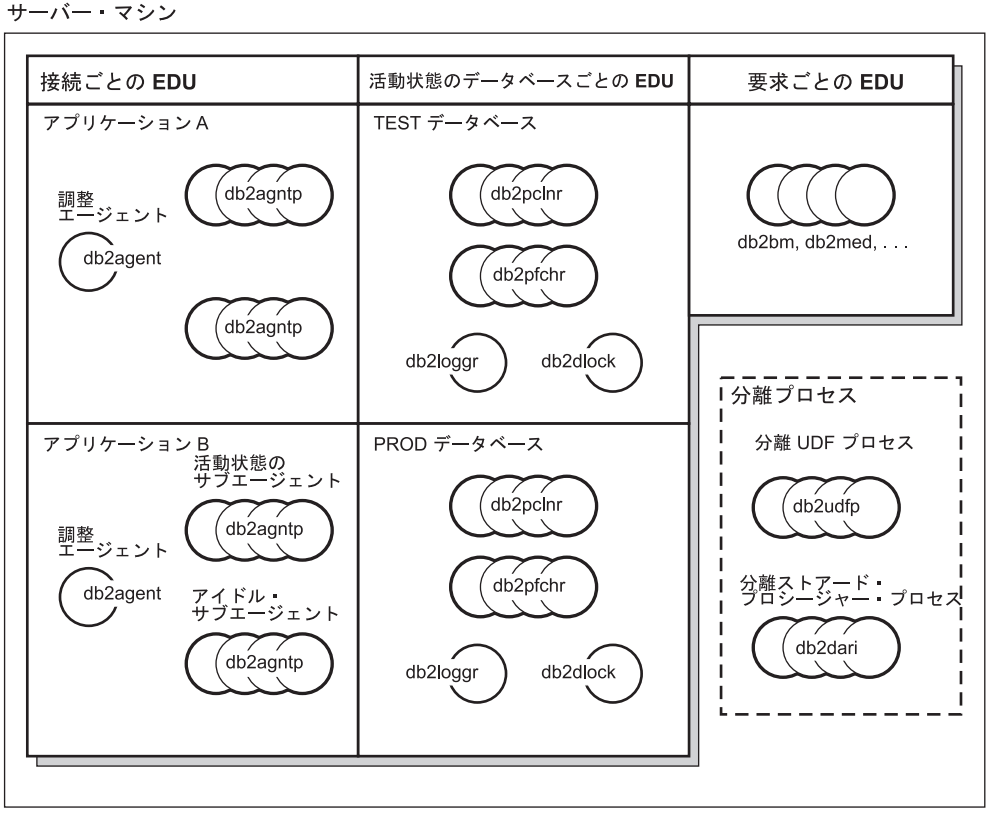


図 8. プロセス・モデル 2

この図は、サーバー・マシン環境の一部である追加のエンジン・ディスパッチ可能単位 (EDU) を示します。活動状態のデータベースには、それぞれプリフェッチャー (db2pfchr) とページ・クリーナー (db2pclnr) の共用プール、そして独自のロガー (db2loggr) およびデッドロック検出機能 (db2dlock) があります。

図の右下にある 『db2udfp』 および 『db2dari』 の円は、それぞれ、分離ユーザー定義関数 (UDF) およびストアード・プロシージャーとして DB2 ユニバーサル・データベース内で実行されるプロセスを表します。これらのプロセスは、作成および破棄に関連したコストを最小限に押さえるために管理されます。 *keepdari* データベース構成パラメーターのデフォルトは 『YES』 であり、ストアード・プロシージャー・プロセスを、次のストアード・プロシージャー呼び出しで再使用できます。

注: さらに、エージェントのアドレス・スペースで直接実行する非分離 UDF およびストアード・プロシージャーもあります。このように処理すると、パフォーマンスが改善されます。ただし、エージェントのアドレス・スペースへのアクセスに制限がないため、使用前には厳しくテストする必要があります。

詳しくは、*アプリケーション開発の手引き* のストアード・プロシージャーの章を参照してください。

複数区画処理モデルは、単一の区画処理モデルの論理拡張です。実際、いずれのモードの操作も、共通のコード・ベースでサポートされています。以下の図では、上記の 2 つの図に示されたような単一区画処理モデルと、複数区画処理モデルとの類似点や相違点を示します。

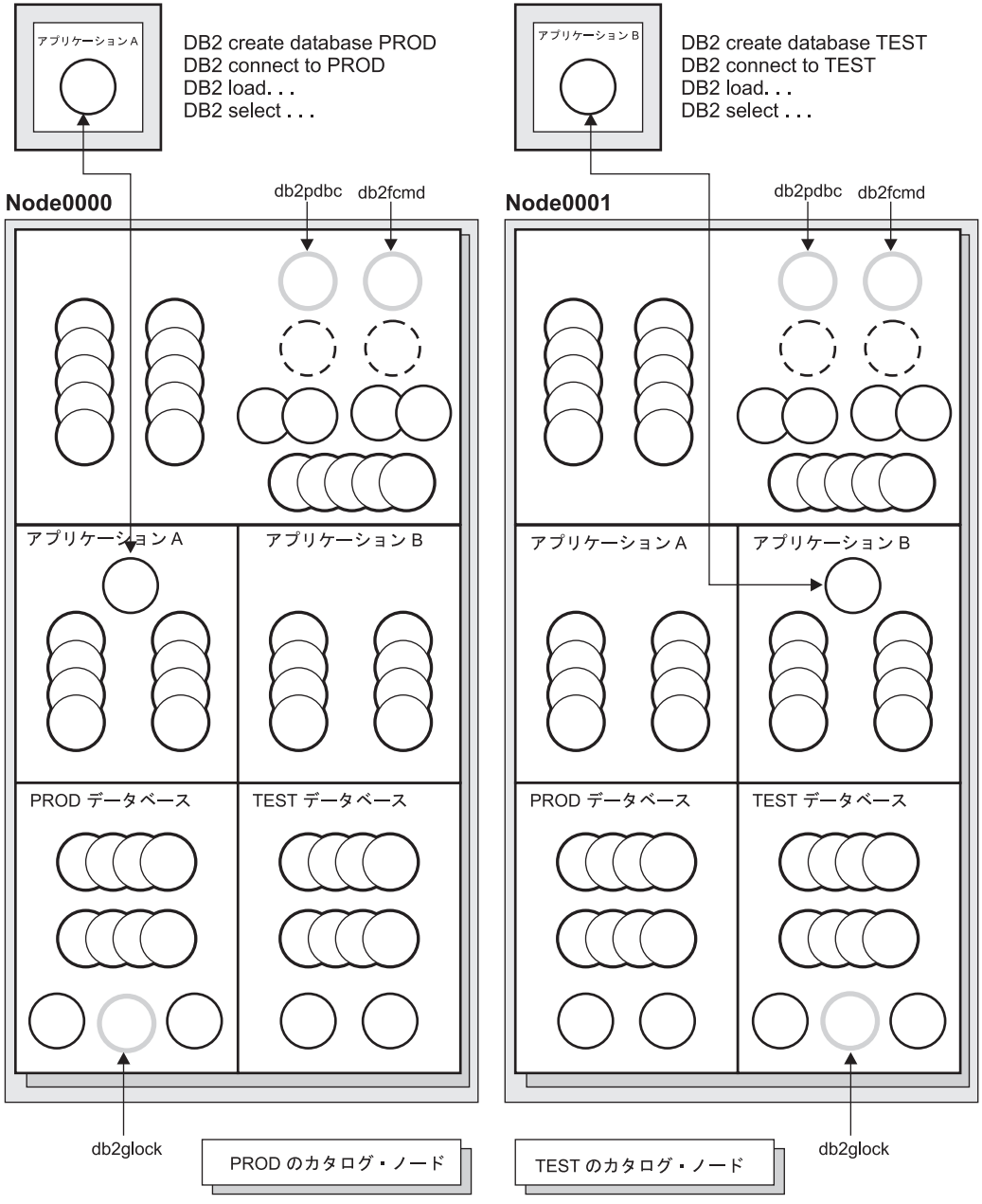


図 9. プロセス・モデルと複数区画

エンジン・ディスパッチ可能単位 (EDU) の大半は、単一区画処理モデルと、複数区画処理モデルで同じです。

複数区画 (またはノード) 環境では、区画の 1 つがカタログ・ノードになります。カタログには、データベース内のオブジェクトに関する情報すべてが保持されています。

上の図で示すとおり、アプリケーション A は Node0000 で PROD データベースを作成するため、 PROD データベースのカタログがこのノードに作成されます。同様に、アプリケーション B が Node0001 に TEST データベースを作成するため、 TEST データベースのカタログがこのノードに作成されます。ご使用のシステム環境のノード間で各データベースのカタログに関連するアクティビティーの均衡を保つために、異なる複数ノード上にデータベースを作成することもできます。

ここには、インスタンスに関連付けられる追加の EDU (db2pdbc および db2fcmd) があり、これらは、複数区分データベース環境の各ノードに見つかります。これらの EDU は、データベース区画間の要求を調整し、高速コミュニケーション・マネージャー (FCM) を使用可能にするのに必要です。

さらに、データベースのカタログ・ノードに関連した追加の EDU (db2glock) があります。この EDU は、活動状態のデータベースがあるノード間のデッドロック状況を制御します。

アプリケーションからの各 CONNECT は、データベースの論理エージェントによって示され、結果として単一調整エージェントになります。調整エージェントは、アプリケーションの接続先の区画に存在します。この区画は、アプリケーションの 『調整プログラム・ノード』 になります。調整プログラム・ノードは、 *SET CLIENT CONNECT_NODE* コマンドを使用することによっても設定できます。アプリケーションからのデータベース要求のパーツは、調整プログラム・ノードにより他の区画のサブエージェントに送られます。次いで、他の区画からの結果すべてが調整プログラム・ノードで統合されてから、アプリケーションに戻されます。

CREATE DATABASE コマンドが出されたデータベース区画は、データベースの 『カタログ・ノード』 と呼ばれます。カタログ表は、このデータベース区画に保管されます。通常、すべてのユーザー表は一連のノード間で区画化されます。

注: 複数の区画を、同じマシンで稼働するように構成できます。これは、『複数論理区画』、あるいは『複数論理ノード』構成と呼ばれます。このような構成は、巨大なメイン・メモリーのある大きい対称マルチプロセッサー (SMP) マシンで大変役立ちます。この環境では、区画間の通信は、共用メモリーおよびセマフォーを使用するように最適化されます。

メモリー・モデル

メモリーは、データベース内での作業に大きな影響を与える点で重要です。データベース内のこれらのエリア間で使用可能なメモリーをどのように分割するかによって、データベースの実行状況を制御することができます。この異なるヒープ間でのメモリーの分割は、構成パラメーターで制御します。このセクションでは、主要な構成パラメーター

およびこれらが制御するメモリーについて説明します。このトピックの詳細については、237ページの『DB2 でのメモリーの使用方法』を参照してください。

区画にあるエンジン・ディスパッチ可能単位 (EDU) はすべて、インスタンス共用メモリーに接続されています。データベース内で作業するすべての EDU は、そのデータベースのデータベース共用ディレクトリーに接続されています。特定のアプリケーションに代わって作業するすべての EDU は、そのアプリケーションのアプリケーション共用メモリーに接続されています。このタイプの共用メモリーは、区画内または区画間の並行性が使用可能になっている場合にのみ割り当てられます。これで、各 EDU には私用メモリーがあります。

データベースが開始されると、インスタンス共用メモリー (データベース・マネージャー共用メモリーともいう) が割り当てられます。インスタンス共用メモリーから、その他すべてのメモリーが接続 (または割り当て) されます。高速コミュニケーション・マネージャー (FCM) が使用される場合には、このメモリーから取り出されたバッファーが使用されます。 FCM は、特定のデータベース環境におけるデータベース・サーバー間およびデータベース・サーバー内での内部通信 (主にメッセージ) に使用されます。最初のアプリケーションがデータベースに接続すると、データベース共用、アプリケーション共用、およびエージェント私用メモリー・エリアが割り当てられます。

データベースが最初に活動化または接続されると、データベース共用メモリー (データベース・グローバル・メモリーともいう) が割り当てられます。このメモリーは、このデータベースに接続するすべてのアプリケーションが使用できます。データベース共用メモリーには、以下のものを含む数多くの異なるメモリー・エリアがあります。

- バッファー・プール
- ロック・リスト
- データベース・ヒープ - これには、ログ・バッファーとカタログ・キャッシュが含まれます。
- ユーティリティー・ヒープ
- パッケージ・キャッシュ

データベース・マネージャー構成パラメーター *numdb* は、並行して活動化できるローカル・データベースの数を指定します。区分データベース環境では、このパラメーターにより、データベース区画サーバー上の活動状態のデータベース区画数が制限されます。 *numdb* パラメーターの値は、割り当てられるメモリーの合計量に影響を与えます。

アプリケーションがデータベースに接続すると、アプリケーション共用メモリー (アプリケーション・グローバル・メモリーともいう) が割り当てられます。この割り当てが実行されるのは、区分データベース環境の場合、あるいは、データベース・マネージャー構成パラメーター *intra_parallel* がオンの場合だけです。このメモリーは、データを共用し、アプリケーションの代わりに作業するエージェントがデータを共用してアクティビティーを調整するために使用されます。

データベース構成パラメーター *maxappls* は、データベースに接続するアプリケーションの数の上限を設定します。データベースに接続するアプリケーションごとに私用メモリーがいくらか割り当てられるので、使用可能な並行アプリケーションの数を多くすると、使用されるメモリーも増える可能性があります。

アプリケーションの最大数は、ある程度 *maxagents* (区画環境の場合は *max_coordagents*) によっても制御されています。 *maxagents* パラメーターは、区画にあるデータベース・マネージャー・エージェントの合計数の上限を設定します。これらのデータベース・マネージャー・エージェントには、活動状態の調整エージェント、サブエージェント、非活動状態のエージェント、およびアイドル・エージェントが含まれます。

エージェント私用メモリーは、エージェントが特定のアプリケーションでの作業を割り当てられる際に、このエージェントに対して割り当てられます。エージェント私用メモリーはエージェントに対して割り当てられるもので、その特定のエージェントによってのみ使用される割り当てメモリー (分類ヒープやアプリケーション・ヒープなど) を含んでいます。

共用メモリーには、いくつか特別なタイプがあります。

- エージェント / ローカル・アプリケーション共用メモリー。このメモリーは、エージェントとそのクライアント・アプリケーションとの間の SQL 要求および応答通信に使用されます。
- UDF/エージェント共用メモリー。このメモリーは、分離 UDF またはストアード・プロシージャーを実行するエージェントにより接続されます。これは、通信エリアとして使用されます。
- 拡張記憶。拡張バッファー・プールとして使用される、通常は大変大きい (4 GB を超える) 共用メモリーの領域。エージェント / プリフェッチャー / ページ・クリーナーは永続的にはこれに接続されないものの、必要に応じて個々のセグメントに接続されます。

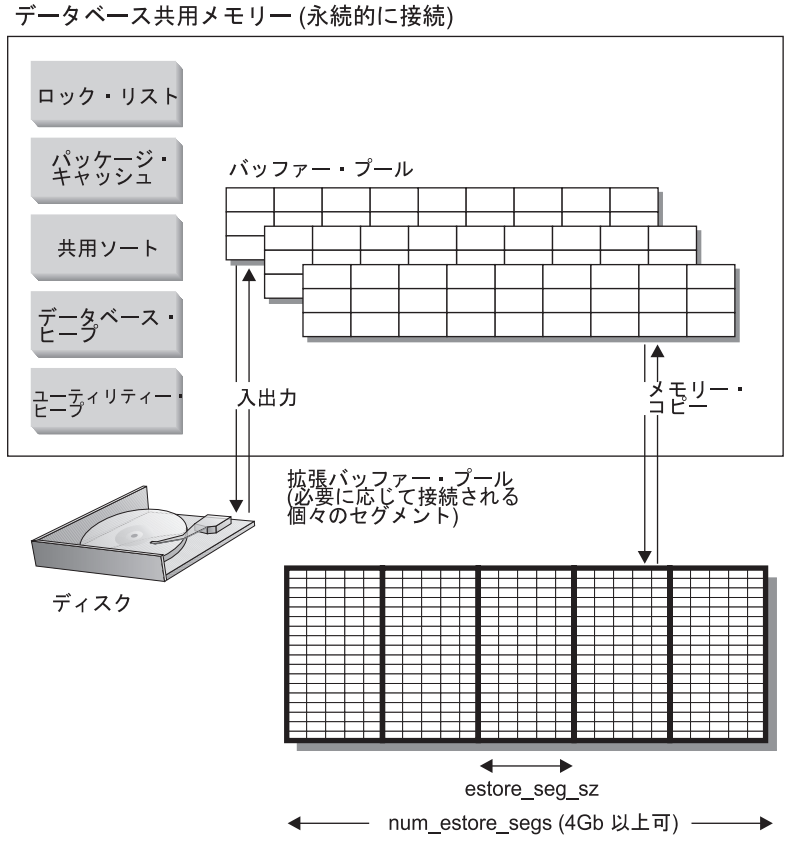


図 10. バッファー・プールと拡張記憶

拡張記憶は、メイン・バッファー・プールの拡張検索バッファーとして機能します。このトピックの詳細については、278ページの『メモリーの拡張』を参照してください。これは 4 GB よりもずっと大きく、主メモリーが大きなマシンを活用するのに最適です。拡張記憶キャッシュは、メモリー・セグメントとして定義されます。

実アドレス可能メモリーの一部を拡張記憶域キャッシュとして使用しようとする場合には、ジャーナル・ファイルシステム・キャッシュまたはプロセス専用アドレス・スペースなどの他の目的ではこのメモリーをマシン上で使用できなくなることに注意してください。拡張記憶キャッシュに追加の実アドレス可能メモリーを割り当てると、システム・ページングが増加する可能性があります。

以下に示すデータベース構成パラメーターは、拡張記憶で使用可能なメモリーの量とサイズに影響を与えます。

- *num_estore_segs* は、拡張記憶メモリー・セグメントの数を定義します。
- *estore_seg_sz* は、各拡張メモリー・セグメントのサイズを定義します。

各表スペースには、バッファー・プールが割り当てられます。拡張記憶キャッシュは、常に 1 つまたは複数の特定のバッファー・プールに関連付けておく必要があります。拡張記憶キャッシュのページ・サイズは、関連付けられたバッファー・プールのページ・サイズに適合しなければなりません。

拡張記憶キャッシュについて詳しくは、278ページの『メモリーの拡張』を参照してください。

第2部 アプリケーション・パフォーマンスのチューニング

第3章 アプリケーションについての考慮事項

アプリケーション実行時のパフォーマンスに影響を与える要素がいくつかあります。この章では、アプリケーションの設計とコーディングのときに考慮する必要のある以下のトピックについて説明します。

- 並行性
- ロック
- 最適化クラスの調整
- 結果セットの制約によるパフォーマンス向上
- 行のブロック化
- 照会の調整
- 複合 SQL
- パフォーマンスについての考慮事項および文字変換
- ストアード・プロシージャー
- データベースの活動化
- アプリケーションの並列処理

また、アプリケーションのパフォーマンスに影響を及ぼす以下のような点に関する情報をさらに知りたい場合は、 *アプリケーション開発の手引き* と *コール・レベル・インターフェースの手引きおよび解説書* を参照してください。

- 組み込み静的 SQL を使用するプログラムの作成
- 組み込み動的 SQL を使用するプログラムの作成
- DB2 コール・レベル・インターフェース (CLI) を使用するプログラムの作成

並行性

リレーショナル・データベースにおいて、複数のユーザーがデータにアクセスし変更したとしても、データの保全性は保たれなければなりません。*並行性* とは、複数の対話式ユーザーまたはアプリケーション・プログラムが同時にリソースを共用することです。データベース・マネージャーはこのアクセスを制御し、次のような望ましくない結果を防ぎます。

- *更新の消失*。 2 つのアプリケーション A および B が、両方ともデータベースから同じ行を読み取り、読み取ったデータに基づいてその 1 つの列の新しい値をそれぞれが計算することがあるかもしれません。 A がその行を新しい値で更新し、次に B もその行を更新すると、 A によって行われた更新が失われてしまいます。

- コミットしていないデータへのアクセス。アプリケーション A がデータベース内の値を更新し、それがコミットされる前に、アプリケーション B がその値を読み取ることがあるかもしれません。この場合、後になって A による値がコミットされずに取り消されるなら、 B が実行する計算は、この非コミットの (そしておそらく無効な) データに基づくものとなってしまいます。
- *反復不能読み取り*。アプリケーションによっては、次のような順序でイベントが生じるものがあります。つまり、まずアプリケーション A がデータベースからデータを読み取り、次いで他の SQL 要求を処理します。その間、アプリケーション B はその行を修正または削除し、その変更をコミットします。その後、アプリケーション A が元の行を再び読み取ろうとすると、修正された行を受け取り、元の行はすでに削除されていることがわかります。
- *幻像読み取り現象*。幻像読み取り現象は、以下の場合に生じます。
  1. アプリケーションが照会を実行し、一定の検索基準に従って行のセットを読み取る。
  2. 別のアプリケーションが、新しいデータを挿入するか、現在のアプリケーションの照会を満足する既存データを更新する。
  3. 最初のアプリケーションは (同じ作業単位内で) ステップ 1 から照会を繰り返す。

  照会を繰り返すと (ステップ 3)、最初に照会を実行したとき (ステップ 1) には戻されなかった追加の行 (『単独読み取り行』) が結果表の一部として戻されます。

分離レベル は、データがアクセスされている間、データが他の処理からどのようにロックされ分離されるかを決めるものです。分離レベルは、作業単位の持続期間中で有効です。 WITH HOLD 文節を使用して DECLARE CURSOR ステートメントによって宣言されたカーソルを使用するアプリケーションは、 OPEN CURSOR が実行された作業単位の持続期間中、選択した分離レベルを保ちます。 (詳細については、*SQL 解説書* を参照してください。) 分離レベルの指定方法については、49ページの『分離レベルの指定』を参照してください。

DB2 でサポートする分離レベルは、次のとおりです。
- 反復可能読み取り
- 読み取り固定
- カーソル固定
- 非コミット読み取り

(一部の DRDA データベース・サーバーはコミットなし 分離レベルをサポートすることに注意してください。その他のデータベースでは、コミットされない読み取りの分離レベルと同様に動作します。この分離レベルの説明については、 *SQL 解説書* を参照してください。)

次のものも参照してください。
- 48ページの『分離レベルの選択』
- 49ページの『分離レベルの指定』

*連合データベース・システム*では、アプリケーションやユーザーが 1 つのステートメント内で 2 つ以上のデータベース管理システム (DBMS) またはデータベースを参照する SQL ステートメントを実行依頼できます。DB2 連合システムは、データベース・オブジェクトの*位置透過性*を提供します。たとえば、表および視点についての情報が移動する場合に、その情報への参照 (ニックネームと呼ばれる) を、その情報を要求するアプリケーションに変更を加えることなく更新することができます。アプリケーションがニックネームにアクセスすると、 DB2 はデータ・ソース・データベース・マネージャーのデータの並行性制御プロトコルにより、分離レベルを保証します。 (データ・ソースは、DBMS とデータで構成されています。)DB2 は要求されたデータ・ソースでの分離レベルを論理的に同等のものと一致させようと試みますが、結果はデータ・ソースの機能によって異なることがあります。ニックネームにアクセスするアプリケーションの作成については、 *アプリケーション開発の手引き* を参照してください。

これ以降では、それぞれの分離レベルの詳細について、影響の大きい順に説明されています。ただし、データにアクセスしたりデータを変更したりする場合には、影響が小さいほど注意が必要になります。

反復可能読み取り

反復可能読み取り (RR) では、ある作業単位の中でアプリケーションが参照する行すべてにロックがかけられます。反復可能読み取りを使用した場合、カーソルがオープンされたのと同じ作業単位の中でアプリケーションが同じ SELECT ステートメントを 2 回発行しても、それぞれ同じ結果になります。反復可能読み取りを使用すると、更新が失われている場合、コミットされていないデータおよび単独読み取り行へのアクセスはできなくなります。

反復可能読み取りアプリケーションは、その作業単位が完了するまでに、必要な回数だけ行の検索および操作を行えます。 しかしそれ以外のアプリケーションは、その作業単位が完了するまで、結果表に影響を与える可能性のある行を更新、削除、または挿入することができなくなります。反復可能読み取りアプリケーションでは、他のアプリケーションの非コミット変更を表示することはできません。

反復可能読み取りを使用すると、検索中の行だけでなく、参照される行すべてにロックがかかります。適正にロックがかけられることによって、最初の照会で参照した後で再び照会を実行したときに、その間に別のアプリケーションが行を挿入または追加するということはできなくなります。これにより、単独読み取り行が発生しなくなります。つまり、10 000 個の行を走査してそれらに述部を適用する場合、 10 行しか適格でなくても、それら 10 000 個の行すべてにロックがかけられます。

注: 反復可能読み取りの分離レベルでは、返されるデータすべてをアプリケーションが*見る* までは、一時表や行ブロック化が使用されている場合であっても、それらの行はすべて未変更のままになっています。

反復可能読み取りでは、相当数のロックを獲得し保持するため、それらのロックが、*locklist* と *maxlocks* の構成パラメーターの結果として使用可能なロックの数を超えることがあります。 (389ページの『自動調整前のロック・リストの最大パーセント (maxlocks)』と 357ページの『ロック・リスト用最大ストレージ (locklist)』を参照。) ロック・エスカレーションが非常に発生しやすいと判断した場合、ロック・エスカレーションを避けるために、最適化プログラムは索引の走査のために単一表レベル・ロックを直接獲得することがあります。 (ロック・エスカレーションについては、57ページの『ロック・エスカレーション』を参照してください。) これは、データベース・マネージャーが LOCK TABLE ステートメントを発行したかのように機能します。表レベルのロックをかけたくない場合は、トランザクションに十分な数のロックを使用できるようにするか、または読み取り固定分離レベルを使用します。

読み取り固定

読み取り固定 (RS) では、ある作業単位においてアプリケーションが検索する行にロックをかけます。これにより、ある作業単位で適格とされて読み取られた行は、その作業単位が完了するまで他のアプリケーションによって変更されないようになり、また他のアプリケーションのプロセスによって変更されたすべての行は、そのプロセスによって変更がコミットされるまで読み取られないようになります。つまり、「反復不能読み取り」の処理は**あり得なくなります**。

反復可能読み取りとは異なり、読み取り固定では、アプリケーションが同じ照会を 2 回以上出すと、新たな*単独読み取り* 行 (*幻像読み取り現象*) が生じる可能性があります。前述の 10 000 個の行を走査する例を考えると、読み取り固定を使う場合は適格な行だけがロックされます。したがって、読み取り固定を使用すると、10 行だけが検索され、ロックはそれら 10 行だけにしかかけられません。これとは対照的に反復可能読み取りを使用すると、この例の場合は、ロックが 10 000 行すべてにかけられます。保持することができるロックは、共用、次回共用、更新、または排他ロックです。 (ロック属性については、52ページの『ロックの属性』 を参照してください。)

注: 読み取り固定分離レベルを使う場合、一時表や行ブロック化を使用する場合でも、返されるデータすべてをアプリケーションが見る までは、それらのデータはすべて未変更になっています。

読み取り固定分離レベルの目的は、データの表示を一定にするとともに、高度の並行性を提供することにあります。この目的を達成するため、最適化プログラムは、ロック・エスカレーションが発生するまで表レベル・ロックがかけられないようにします。 (ロック・エスカレーションについては、57ページの『ロック・エスカレーション』を参照してください。)

次のことすべてを行うアプリケーションでは、読み取り固定分離レベルが最適です。
- 並行環境で操作する
- 作業単位の間、適格となる行を一定にしておく必要がある

- 作業単位において同じ照会を 2 回以上発行しない、または同じ作業単位において同じ照会を 2 回以上発行するときに、同じ応答を入手することを要求しない。

カーソル固定

カーソル固定 (CS) は、アプリケーションのトランザクションがアクセスする行にカーソルがある間、その行をロックします。このロックは、次の行が取り出されるか、またはトランザクションが終了する時まで有効です。しかし、行の中の何らかのデータが変更された場合、変更がデータベースにコミットされるまで、ロックが保持されていなければなりません。

カーソル固定アプリケーションが検索した行に更新可能なカーソルがある間、他のアプリケーションはその行を更新したり削除したりできません。カーソル固定のアプリケーションでは、他のアプリケーションによる非コミット変更を見ることができません。

前述の 10 000 行を走査する例を考えてみると、カーソル固定を使う場合は、現在カーソルが位置している行だけにロックがかかります。ロックは、その行から移動すると (その行を更新しなければ) 解除されます。

カーソル固定を使うと、反復不能読み取りと幻像読み取り現象は両方とも発生する可能性があります。カーソル固定はデフォルトの分離レベルですが、コミットされた行だけを他のアプリケーションから見る間は並行性を最大にしたいという場合に、ぜひ使用するようにしてください。

非コミット読み取り

非コミット読み取り (UR) では、アプリケーションが他のアプリケーションの非コミットの変更にアクセスできます。アプリケーションは、他のアプリケーションが表を除去または変更しようとするのでない限り、読み取り中の行について他のアプリケーションに対するロックをかけません。非コミット読み取りの動作は、読み取り専用カーソルと更新可能カーソルとで違います。

読み取り専用カーソルは、他のトランザクションのほとんどの非コミットの変更にアクセスすることができます。しかし、トランザクションの処理中に、他のトランザクションによって作成または除去されている表、視点、および索引にアクセスすることはできません。他のトランザクションによるその他の変更は、コミットまたはロールバックされる前に読み取ることができます。

注: 非コミット読み取り分離レベルで更新可能な操作を行っているカーソルは、カーソル固定分離レベルの場合と同じ働きをします。

分離レベル UR を使用してアプリケーション・プログラムを実行するとき、分離レベル CS を使用してアプリケーションを実行することができます。これは、アプリケーション・プログラムで使用されるカーソルが未確定であるために起こります。 BLOCKING オプションにより、未確定カーソルを分離レベル CS に自動調整することができます。

BLOCKING オプションのデフォルトは、UNAMBIG です。これは、未確定カーソルが更新可能として扱われ、分離レベルが CS に自動調整されることを意味します。この自動調整が行われないようにするための選択肢が 2 つあります。 1 つ目の選択肢は、SELECT ステートメントに FOR READ ONLY 文節を組み込んで、アプリケーション・プログラム内のカーソルが未確定とならないように変更することです。 2 つ目の選択肢は、カーソルがアプリケーション・プログラム内で未確定のままになっている場合、プログラムを、BLOCKING ALL オプションを指定してプリコンパイルまたは結合することです。これにより、未確定カーソルはすべて、プログラムの実行時に読み取り専用として扱われます。

前述の 10 000 行を走査する例を考えてみると、非コミット読み取りを使う場合は、ロックが獲得されません。つまり行にロックがかからなくなります。

非コミット読み取りを使用すると、反復不能読み取りと幻像読み取り現象の両方とも発生する可能性があります。

非コミット読み取り分離レベルは、読み取り専用表に対して、あるいはＳＥＬＥＣＴステートメントのみを実行しており、かつ他のアプリケーションから非コミット・データを見るかどうかが問題にはならない場合に、最もよく用いられています。

分離レベルの選択

表1 には、 *アプリケーション開発の手引き* に示されている望ましくない影響に関連して、さまざまな分離レベルが要約されています。

表 1. 分離レベルのまとめ

分離レベル	非コミット・データへのアクセス	反復不能読み取り	幻像読み取り現象
反復可能読み取り (RR)	不可能	不可能	不可能
読み取り固定 (RS)	不可能	不可能	可能
カーソル固定 (CS)	不可能	可能	可能
非コミット読み取り (UR)	可能	可能	可能

表2 には、ご使用のアプリケーションの初期分離レベルを選択する上で役立つ簡単な発見的手法が収められています。次の表を参考に、必要な要件により適した値を選択する上で考慮すべきさまざまなレベルの要因について述べたこれまでの説明を参照してください。

表 2. 分離レベルを選択する指針

アプリケーションのタイプ	高度のデータ安定度が必要	高度のデータ安定度が不要
読み書きトランザクション	RS	CS

表 2. 分離レベルを選択する指針 (続き)

アプリケーションのタイプ	高度のデータ安定度が必要	高度のデータ安定度が不要
読み取り専用トランザクション	RR または RS	UR

アプリケーションに適した分離レベルを選択することは、そのアプリケーションに許容されていない現象を回避するためにとても重要です。分離レベルは、アプリケーション間の分離の程度に影響を与えるだけでなく、ロックの獲得と解放に必要な CPU とメモリーのリソースが分離レベルごとに異なるため、個々のアプリケーションのパフォーマンス特性にも影響を与えます。デッドロック状態になる可能性は、分離レベルごとに異なります。

分離レベルの指定

分離レベルは、プリコンパイル時、またはアプリケーションがデータベースにバインドされるときに指定されます。サポートされるコンパイル言語で作成されたアプリケーションの場合、コマンド行プロセッサーの PREP または BIND コマンドの ISOLATION オプションを使用します。分離レベルは、PREP または BIND の API を使って指定することもできます。

プリコンパイル時にバインド・ファイルが作成される場合、分離レベルはそのバインド・ファイル内に保管されます。バインド時に分離レベルが指定されない場合のデフォルトは、プリコンパイル時に使用された分離レベルです。

分離レベルを指定しない場合、デフォルトとしてカーソル固定が使用されます。

次の照会を実行すると、パッケージの分離レベルを調べることができます。

```
SELECT ISOLATION FROM SYSCAT.PACKAGES
  WHERE PKGNAME = 'XXXXXXXX'
  AND PKGSCHEMA = 'YYYYYYYY'
```

XXXXXXXX はパッケージの名前、 *YYYYYYYY* はパッケージのスキーマ名です。これらの名前は両方ともすべて大文字でなければなりません。

データベースを作成すると、 REXX に含まれる SQL のさまざまな分離レベルのサポートに使用される複数のバインド・ファイルがデータベースにバインドされます (REXX をサポートするサーバーで)。他のコマンド行プロセッサー・パッケージも、データベースの作成時にデータベースにバインドされます。 バインド・ファイルの詳細については、 *アプリケーション開発の手引き* を参照してください。

データベースへの REXX とコマンド行プロセッサーによる接続では、カーソル固定のデフォルト分離レベルが使用されます。異なる分離レベルに変更しても、接続状態は変更しません。このコマンドは、CONNECTABLE AND UNCONNECTED 状態または

IMPLICITLY CONNECTABLE 状態のときに実行する必要があります。 (接続状態の詳細については、 *SQL 解説書* の『CONNECT TO ステートメント』を参照してください。)

REXX アプリケーションで SQLISL REXX 変数の値を調べることによって、使用される分離レベルを調べることができます。その値は、CHANGE SQLISL コマンドが実行されるたびに更新されます。

DB2_RR_TO_RS プロファイル・レジストリー変数を使用すると、次のキーのロックを最小限に抑えるので、結果として、並行性とパフォーマンスが向上することになります。ただし、これを使用することによって、反復可能読み取り (RR) 分離レベルは使用できなくなります。データベースにアクセスするアプリケーションにおいて RR のセマンティクスが必要ない場合には、この変数を "Yes" に設定することをお薦めします。有効にするには、データベース・マネージャーを停止してから開始する必要があります。db2start の後で、この変更はインスタンス全体に影響を与えるようになります。この登録値がいったん設定されたならば、ユーザー表への RR を使用したアクセスが要求されても要求の内容が内部的に修正されて、読み取り固定 (RS) 分離レベルが代わりに使用されます。この処理を行う前に警告は出されません。

アプリケーションを準備またはバインドするときに分離レベルをパッケージ・レベルで設定することに加え、分離レベルをステートメント・レベルで設定することができます。ステートメント・レベルの分離レベルは、WITH 文節を使用して指定します。

ステートメント・レベル分離をサポートしているのは以下の SQL ステートメントです。

- SELECT ステートメント
- SELECT INTO
- 探索条件付き DELETE
- INSERT
- 探索条件付き UPDATE
- DECLARE CURSOR

ステートメント・レベル分離の使用に関しては、いくつかの条件があります。

- WITH 文節を副照会で使用することはできません。
- WITH UR オプションは、読み取り専用の操作にのみ適用されます。このオプションが他の状況で使用された場合、ステートメントは 『UR』 から 『CS』 に自動的に変更されます。
- ステートメントのデフォルト分離レベルは、ステートメントがバインドされるパッケージの分離レベルです。
- ステートメント・レベルの分離レベルは、ステートメントがあるパッケージに指定された分離レベルをオーバーライドします。

コマンド行プロセッサーを使用している場合には、 CHANGE ISOLATION LEVEL コマンドを使用して分離レベルを変更できます。詳細については、*コマンド解説書* を参照してください。

DB2 コール・レベル・インターフェース (DB2 CLI) の場合は、 DB2 CLI 構成の一部として分離レベルを変更できます。実行時の CLI では、 SQLSetConnectAttr 関数を SQL_ATTR_TXN_ISOLATION 属性と組み合わせて使用します。これによって、*ConnectionHandle* で参照される現行接続のトランザクション分離レベルが設定されます。 db2cli.ini ファイルでは、TXNISOLATION キーワードも使用できます。

注: JDBC および SQLJ は、DB2 上の CLI を使用してインプリメントされています。つまり、db2cli.ini の設定は、 JDBC や SQLJ を使用して作成および実行されることに影響します。詳細については、*コール・レベル・インターフェースの手引きおよび解説書* を参照してください。

実行時に JDBC または SQLJ を使用する場合は、 java.sql インターフェース接続の setTransactionIsolation メソッドを使用して、分離レベルを確立することができます。詳しくは、*アプリケーション開発の手引き* の『Java でのプログラミング』の章を参照してください。

SQLJ を使用していて、 db2profc SQLJ 最適化プログラムを実行している場合には、パッケージが作成されます。このパッケージの最後のオプションに、使用する分離レベルを指定することができます。詳しくは、*アプリケーション開発の手引き* の『Java でのプログラミング』の章を参照してください。

さらに、多くの商業目的で作成されたアプリケーションでも、分離レベルを選択するための手段が提供されています。詳細については、*コール・レベル・インターフェースの手引きおよび解説書* を参照してください。

宣言済み一時表および並行性

宣言済み一時表は、これらを宣言したアプリケーションでのみ使用できるため、並行性の問題はありません。このタイプの表は、アプリケーションがこれを宣言してから、これを完了または切断するまでのあいだのみ存在します。

ロック

データベース・マネージャーは、並行性制御が可能になっており、ロックを使うことによって、無秩序にアクセスがなされることがないようになっています。*ロック* はデータベース・マネージャー・リソースをアプリケーションに関連付けて、そのリソースへの他のアプリケーションからのアクセス方法を制御する手法です。リソースが関連付けられているアプリケーションは、ロックを保持または所有していると言われます。

データベース・マネージャーはロックをかけることで、アプリケーションが他のアプリケーションによって作成された非コミットのデータにアクセスすることがないようにし

ます (ただし、非コミット読み取り分離レベルが使用されている場合は除きます)。 この原則によりデータの整合性 (データの一貫性とセキュリティー) が保たれます。さらにロックを使って、行の更新を禁止することができます (反復可能読み取りアプリケーションの場合など)。

データ保全性を保つため、データベース・マネージャーはデータベース・マネージャー制御の下で暗黙のうちにロックを獲得します。非コミット読み取り分離レベルを除いては、非コミット・データを他のプロセスから隠すためにアプリケーションが明示的にロックを要求する必要は全くありません。

ロックの基本原則により、ほとんどの場合ロックを制御するためのアクションを取る必要はありません。それでも、アプリケーションは特定の一般的なパラメーターに基づいてロックを獲得します。ローカルな状況について把握しておくなら、それらのパラメーターを変更してシステム・リソースを効果的に使用するうえで役立ちます。ここでは、ロックに関する以下のトピックを説明します。

- ロックの属性
- ロックとアプリケーションのパフォーマンス
- ロックに影響する要因
- LOCK TABLE ステートメント
- CLOSE CURSOR WITH RELEASE
- ロックについての考慮事項のまとめ

ロックの属性

データベース・マネージャーのロック機能には、以下のような基本属性があります。

モード ロックの所有者に許可されるアクセスの種類、そしてロックの対象の並行ユーザーに許可されるアクセスの種類。これは、しばしばロックの *状態* と呼ばれます。

オブジェクト
ロックするリソース。明示的なロック可能オブジェクトのタイプは、表だけです。このほかにもデータベース・マネージャーでは、行、表、表スペースなど、他のタイプのリソースにもロックをかけます。ロックされているオブジェクトは、ロックの *細分性* を表します。

期間 ロックが保持される時間的長さ。ロック期間は分離レベルの影響を受けます (それについては、 43ページの『並行性』で説明されています)。

以下の表では、モードとその効果が示されています。ここに示す順に、リソースへの制御が大きくなります。

表 3. ロック・モードの要約

ロック・モード	適用できるオブジェクト・タイプ	説明
IN (意図なし)	表スペース、表	ロックの所有者は、非コミット・データを含め、表内のすべてのデータを読み取ることができますが、更新はできません。ロック所有者によって行ロックは獲得されません。同時に実行される他のアプリケーションは、その表を読み取ったり更新したりできます。
IS (意図共用)	表スペース、表	ロック所有者は、ロックされている表のデータを読み取ることはできますが、更新はできません。アプリケーションが IS 表ロックを保持している場合には、そのアプリケーションは各行の読み取りごとに S ロックまたは NS ロックを獲得します。いずれにしても、他のアプリケーションはその表を読み取ったり更新したりできます。
NS (次キー共用)	行	ロック所有者とすべての並行アプリケーションは、ロックされた行を読み取ることはできますが、更新はできません。このロックは、S ロックの代わりに、表の行に対して獲得されます。この場合、アプリケーションの分離レベルは、RS か CS のいずれかです。
S (共用)	行、表	ロック所有者とすべての並行アプリケーションは、ロックされたデータを読み取ることはできますが、更新はできません。表の個々の行は S ロックされます。表が S ロックされている場合、行ロックは必要ではありません。
IX (意図排他)	表スペース、表	ロック所有者と同時に実行されるアプリケーションとは、表のデータを読み取ったり更新したりできます。ロック所有者がデータを読み取るときには、行ごとに S、NS、X、または U ロックが獲得されます。また、ロック所有者が更新する行ごとに X ロックも獲得されます。同時に実行される他のアプリケーションは、その表を読み取ったり更新したりできます。
SIX (意図排他共用)	表	ロック所有者は、表のデータを読み取ったり更新したりできます。ロック所有者は、更新する行ごとに X ロックを獲得しますが、読み取る行のロックは獲得しません。同時に実行される他のアプリケーションは、その表を読み取ることができます。
U (更新)	行、表	ロック所有者は、ロックされた行または表のデータを更新できます。ロック所有者は、行を更新する前にこの行に対して X ロックを獲得します。他の作業単位はロックされた行または表のデータを読み取ることはできますが、更新は行えません。

表 3. ロック・モードの要約 (続き)

ロック・モード	適用できるオブジェクト・タイプ	説明
NX (次キー排他)	行	ロック所有者は、ロックされた行の読み取りはできますが更新はできません。このモードは、NS ロックと互換性があることを除けば、X ロックと類似した働きをします。
NW (次キー弱排他)	行	このロックは、行を非カタログ表の索引に挿入するときに、その行の次の行に対して獲得されます。ロック所有者は、ロックされた行の読み取りはできますが更新はできません。このモードは、W ロックおよび NS ロックと互換性があることを除けば、 X ロックおよび NX ロックと類似した働きをします。
X (排他)	行、表	ロック所有者は、ロックされた行または表のデータを読み取ったり更新したりできます。表は、排他ロックできます。つまり、これらの表の行について行ロックが獲得されることはありません。ロックされた表にアクセスできるのは、非コミット読み取りアプリケーションだけです。
W (弱排他)	行	このロックは、行を非カタログ表に挿入するときに、その行に対して獲得されます。ロック所有者は、ロックされた行を変更することができます。このロックは、NW ロックと互換性があることを除けば、X ロックと類似した働きをします。ロックされた行にアクセスできるのは、非コミット読み取りアプリケーションだけです。
Z (超排他)	表スペース、表	このロックが表上で獲得されるのは、表が変更または除去されるとき、表の索引が作成または除去されるとき、あるいは表が再編成されるときなど、特定の状況においてです。同時に実行される他のアプリケーションは、その表を読み取ったり更新したりできません。

注: 『意図』 ロック・モードを取得するのは、表と表スペースだけです。つまり、行には意図ロックは取得されません。

ロックとアプリケーションのパフォーマンス

アプリケーション・プログラマーは、ロックの使用に関係するいくつかの関連した要因と、それがアプリケーションのパフォーマンスに与える影響について知っておく必要があります。そのような要因には、次のものが含まれます。

- 並行性と細分性
- ロックの互換性
- ロックの変換
- ロック・エスカレーション

- ロックの待機とタイムアウト
- デッドロック

並行性と細分性

あるアプリケーションがロックを保持していると、別のアプリケーションによるアクセスはできなくなります。したがって、並行性を最大にするには、表ロックよりも行レベル・ロックが適しています。しかし、ロックにはストレージが必要になり、また管理のための処理時間も必要になります。したがって、ストレージと処理時間を最小限にするには、多くの行ロックよりも 1 つの表ロックの方が適しています。

ALTER TABLE ステートメントの LOCKSIZE 文節を使用すると、行または表レベルでロックのサイズ (細分性) を定義できます。デフォルトでは、行ロックが使用されます。 ALTER TABLE で定義した永続表ロックを使うと、S および X 表ロックのみが使用されます。アプリケーションが多くの行ロックを獲得したり、解放したりする必要がなくなるので、パフォーマンスは向上します。次のような場合には、 LOCK TABLE ステートメントを使用して単一トランザクション表をロックするよりも、 ALTER TABLE を使用して永続表ロックをすることができます。

- 使用している表が読み取り専用であり、S ロックが必ず必要な場合。表レベル・ロックを使用すると、他のユーザーが表に対して S ロックをかけることを許可しながら、パフォーマンスが向上します。
- 表を保守するためにアクセスするユーザーは 1 人だけです。保守を行うユーザーは、短い期間の間 X ロックをかける必要があります。表の ALTER TABLE で表レベルのロックを変更すると、このユーザーには表レベルの X ロックが与えられます。このユーザーが保守を終えると、 ALTER TABLE を使用して表を行レベルのロックに戻せます。

ALTER TABLE ステートメントを使用しても、通常のロック・エスカレーションに支障はありません。

さらに、ALTER TABLE を使用して表レベルでロックするというのはグローバルな方法であることに注目してください。これは、その表をアクセスするすべてのアプリケーションおよびユーザーに影響を与えます。個々のアプリケーションで使えるもう 1 つの方法は、 LOCK TABLE ステートメントを使用することです。これにより、データベース・レベルではなくアプリケーション・レベルで表ロックをかけることができます (上の 2 番目の黒丸を参照)。

ロックの互換性

表4 に、特定の状態で別のプロセスが同じリソースにロックを保持または要求している場合に、ロック要求が認可されるかどうかを示します。**いいえ**は、非互換ロックが他のプロセスによってすべて解除されるまで、要求側が待機しなければならないことを示します。ロック待機中に、タイムアウトになることがあるので注意してください。**はい**は、ロックが認可されることを示します (ただし、他のだれかがそのリソースを待っていない場合に限ります)。

表 4. ロック・タイプの互換性

	保持されているリソースの状態												
要求されている状態	**なし**	**IN**	**IS**	**NS**	**S**	**IX**	**SIX**	**U**	**NX**	**X**	**Z**	**NW**	**W**
なし	はい	はい	はい	はい	はい	はい	はい	はい	はい	はい	はい	はい	はい
IN	はい	はい	はい	はい	はい	はい	はい	はい	はい	はい	いいえ	はい	はい
IS	はい	はい	はい	はい	はい	はい	はい	はい	いいえ	いいえ	いいえ	いいえ	いいえ
NS	はい	はい	はい	はい	はい	いいえ	いいえ	はい	はい	いいえ	いいえ	はい	いいえ
S	はい	はい	はい	はい	はい	いいえ	いいえ	はい	いいえ	いいえ	いいえ	いいえ	いいえ
IX	はい	はい	はい	いいえ	いいえ	はい	いいえ	いいえ	いいえ	いいえ	いいえ	いいえ	いいえ
SIX	はい	はい	はい	いいえ	いいえ	いいえ	いいえ	いいえ	いいえ	いいえ	いいえ	いいえ	いいえ
U	はい	はい	はい	はい	はい	いいえ	いいえ	いいえ	いいえ	いいえ	いいえ	いいえ	いいえ
NX	はい	はい	いいえ	はい	いいえ	いいえ	いいえ	いいえ	いいえ	いいえ	いいえ	いいえ	いいえ
X	はい	はい	いいえ	いいえ	いいえ	いいえ	いいえ	いいえ	いいえ	いいえ	いいえ	いいえ	いいえ
Z	はい	いいえ	いいえ	いいえ	いいえ	いいえ	いいえ	いいえ	いいえ	いいえ	いいえ	いいえ	いいえ
NW	はい	はい	いいえ	はい	いいえ	いいえ	いいえ	いいえ	いいえ	いいえ	いいえ	いいえ	はい
W	はい	はい	いいえ	いいえ	いいえ	いいえ	いいえ	いいえ	いいえ	いいえ	いいえ	はい	いいえ

注:

I 意図
N なし
NS 次キー共用
S 共用
NX 次キー排他
X 排他
U 更新
Z 超排他
NW 次キー弱排他
W 弱排他

これらのロック・タイプについては、52ページの『ロックの属性』の説明を参照してください。

注:

- はい - 要求されたロックがただちに**付与される**
- いいえ - 保持しているロックを解放するまで、またはタイムアウトになるまで**待機する**

アプリケーション A がある表のロックを保持していて、その表へのアクセスをアプリケーション B も待っているとします。データベース・マネージャーはアプリケーション B のために、ある特定のモードのロックを要求します。 A が保持しているロックのモードが、 B の要求しているロックを許可するものである場合、2 つのロック (またはモード) は、互換性があると言います。

アプリケーション B が要求したロック・モードとアプリケーション A が保持しているロックとの間に互換性がない場合には、アプリケーション B はそれ以上継続できません。アプリケーション B は、アプリケーション A がロックを解放し、さらに既存の非互換のロックがすべて 解放されるまで待機する必要があります。

ロックの変換

ロックの変換が行われるのはあるプロセスがすでにロックを保持しているデータ・オブジェクトにアクセスする場合に、そのアクセスのモードが、すでに保持しているロックよりさらに制約の大きいロックを必要とするものである場合です。 1 プロセスの中で同じデータ・オブジェクトに (照会によって間接的に) 何度もロックを要求できますが、1 つのデータ・オブジェクトのロックは 1 回に 1 つしか保持できません。すでに保持しているロックのモード変更操作は、変換 と呼ばれます。

行の変換の場合は単純です。たとえば、X が必要なときに S または U を保持しているといった場合に、変換が行われます。

表および行には、いくつかのロック・モードがあります。しかし、IX (意図排他) と S (共用) ロックは、ロック変換に関しては特殊なケースです。 S と IX はどちらも他方よりも制約が大きいとは見なされないため、これらのロックの一方を保持しているときに他方が必要になった場合には、結果として SIX (意図排他共用) ロックに変換されることになります。他のすべての変換の結果は、要求されているモードの制限がよりゆるい場合は、要求されたロック・モードが、保持するロックのモードになります。

行を更新するための照会では、変換が二重に行われる可能性もあります。たとえば、その行が索引アクセスによって読み取られ、 S ロックがかけられていたとします。その行が入っている表には、それをカバーするような意図的ロックがかけられているはずです。ロック・タイプが IX ロックではなく IS であるとします。後で行が変更されると、表ロックは IX に変換され、行ロックは X に変換されることになります。

通常、ロックの適用は照会の実行時に暗黙のうちに行われます。各種の照会および表と索引の組み合わせの下で発行されるロックの種類について知っておくことは、アプリケーションの設計と調整に役立つでしょう。そのことについては、61ページの『ロックに影響する要因』を参照してください。

ロック・エスカレーション

ロック・エスカレーションは、保持されるロックの数を減らすための内部機構です。エスカレーションは、多くの行ロック (単一の表内) から単一の表ロックまであります。

ロック・エスカレーションが起きるのは、データベース内で現在保持しているロックが多くなりすぎたときです。

ロック・エスカレーションは、特定のデータベース・エージェントがロック・リストのそのエージェントに割り振られた分を超過した場合にも、起こることがあります (389ページの『自動調整前のロック・リストの最大パーセント (maxlocks)』を参照してください)。

このような自動調整は内部的に処理されるので、 1 つまたは複数の表への並行アクセスが少なくなるのは、外部検出結果のみです。通常、適正に構成されたデータベースでは、ロック・エスカレーションはほとんど行われません。

ロック・エスカレーションが起きる場合の例としては、アプリケーション設計者が大きな表で索引を使用してパフォーマンスと並行性を向上させようとしたが、そのアプリケーションはその表の大部分のレコードにアクセスするというような場合があげられます。 (この場合、) データベース・マネージャーは、表の大部分がロックされることを予測できないため、 S または X の表だけをロックするのではなく、各レコードを個別にロックします。このような場合の解決策として、データベース設計者は、アプリケーション設計者とも相談した上で、このトランザクションに LOCK TABLE ステートメントを使用するように推奨することができます。

しばしば、内部的に自動調整要求を受け取るプロセスは、表に対してほとんど、またはまったくレコード・ロックをかけていません。この自動調整が行われる理由は、1 つのプロセスが多くのロックをかけているが、ロックの数が、1 プロセス当たりのロック数を指定するデータベース構成パラメーター値より小さく、自動調整要求が引き起こされるほどではないためです。そのプロセスは、トランザクション終了時を除いて、別のロックを要求したりデータベースにアクセスしたりしないかもしれません。そして、別のプロセスが、自動調整要求を引き起こすロックを要求する場合があります。

ロック・エスカレーションによって並行性が受諾不能なレベルまで下がると、以下のことができるようになります。

- *db2diag.log* の内容を検査して、自動調整についての情報がないかどうかを調べる。情報は、自動調整されているそれぞれの表について記録されます。 記録される情報のタイプには、次のものが含まれています。
  - 現在保持されているロックの数。
  - ロック・エスカレーションが完了する前に必要なロックの数。
  - 自動調整されているそれぞれの表の表 ID 情報と表名。
  - 現在保持されている非表ロックの数。
  - 自動調整の一部として獲得される新しい表レベルのロック。 一般的に、これは 『S』 (共用ロック) または 『X』 (排他ロック) です。
  - 新しい表ロック・レベルを獲得した結果として生じる内部戻りコード。

  現在の動的 SQL ステートメントも記録される場合があります。記録される場合に、**もし** DIAGLEVEL データベース・マネージャー構成パラメーターが 4 であれば、記録される情報には、どの表ロックの自動調整が行われるよりも前の現在の SQL ステートメントが含まれます。ロック・エスカレーションが失敗する場合、**もし** DIAGLEVEL が 2 以上であれば、記録される情報には、自動調整が失敗した表と現在の SQL ステートメント (利用可能で、以前に作成されたのではない場合) が含まれます。

  この情報があれば、以下で説明されているその他の点に基づいて、適切なアクションを実行できます。

  このタイプの情報を記録し始めるには、データベース・マネージャー構成パラメーター DIAGLEVEL を 3 (デフォルト) または 4 に設定します。

- データベース構成ファイルの *maxlocks* パラメーターまたは *locklist* パラメーター (あるいはその両方) の値を大きくすることによって、許可されるロックの数を増やす。(389ページの『自動調整前のロック・リストの最大パーセント (maxlocks)』と 357ページの『ロック・リスト用最大ストレージ (locklist)』を参照。) 他のプロセスからの表への並行アクセスが最重要である場合には、これが適切な処置となります。しかし、レコード・レベルのロックの取得によるオーバーヘッドのために、表への並行アクセスによって節約できる量を超えて、他のプロセスに遅延が誘発されることもあります。 (区分データベースでこれらのパラメーターを変更する場合には、すべての区画でパラメーターが更新されるように注意してください。)
- 妨げとなっているプロセス (自動調整するものまたはロールバックするものの場合もあれば、そうでない場合もある) を見つけて調整し、明示的に LOCK TABLE ステートメントを発行する。
- 分離の程度を変更する。この場合、並行性が低下する可能性があります。
- コミットの頻度を増やす。こうすると、特定の時間に存在するロックの数を減らすことができます。分離レベルと並行性の詳細については、 43ページの『並行性』を参照してください。

ロックの待機とタイムアウト

ロック・タイムアウト検出を指定していない場合、異常終了時には、ロックが解放されるまでアプリケーションの無期限待機が必要になることがあります。たとえば、あるトランザクションが別のユーザーによって保持されているロックを待機しており、一方でそのユーザーが実行中のトランザクションをアプリケーションがコミットできるように対話を実行してロックを解放するということをせずに作業場から席をはずしてしまった場合に、そうなります。明らかに、この結果アプリケーションのパフォーマンスが低下します。こうしたケースでプログラムが*停止* しないようにするには、 *locktimeout* 構成パラメーターを使用して、アプリケーションがロックを獲得するまで待機する最大待ち時間を設定することができます。 (390ページの『ロック・タイムアウト (locktimeout)』を参照してください。)

このパラメーターを使用すると、特に分散作業単位 (DUOW) アプリケーションにおいては、グローバル・デッドロックを避けることができます。ロックがタイムアウトになる、つまり、ロック要求が保留にされている時間が *locktimeout* 値より長くなると、アプリケーションはエラーを受け取り、トランザクションがロールバックされます。たとえば、*program1* が、すでに *program2* によって保持されているロックを獲得しようとするときにタイムアウトになると、 *program1* は SQLCODE -911 と理由コード 68 を返します。

データベース・マネージャー構成パラメーター *diaglevel* が 4 に設定され、ロック要求がタイムアウトになった場合、 db2diag.log にはさらに情報が追加されている可能性があります。この情報には、オブジェクト、ロック・モード、およびこのオブジェクトへのロックを保持しているアプリケーションが含まれます。また、現行の動的 SQL ステートメントまたは静的パッケージも見つかる可能性があります。

デッドロック

データベース・マネージャーでは、データベースを使用するプロセスによってロックの競合が発生すると、デッドロックになる可能性があります。たとえば、プロセス 1 が X (排他) モードで表 A にロックをかけ、プロセス 2 が X モードで表 B にロックをかけるとします。次にプロセス 1 が X モードで表 B にロックをかけようとし、さらにプロセス 2 が X モードで表 A にロックをかけようとすると、それらのプロセスはデッドロックに陥ります。デッドロック状態になると、プロセスは両方とも、それらの 2 度目のロック要求が受け入れられるまで中断状態になり、しかもどちらの要求も、いずれかのプロセスがコミットまたはロールバックを実行するまで受け入れられることはありません。外部エージェントがいずれかのプロセスを活動化し、強制的にロールバックを実行させるまで、この状態が無限に続きます。

ロック・システムでのデッドロックは、データベース・マネージャーにおいて、デッドロック検出機能と呼ばれる非同期システム背景プロセスによって処理されます。デッドロック検出機能は、 *dlchktime* 構成パラメーターによって指定された周期で活動状態になります (388ページの『デッドロック検査の時間間隔 (dlchktime)』を参照)。デッドロック検出機能が活動状態になると、ロック・システムがデッドロック状態になっていないかどうかを調べます。データベースが区分データベースである場合、各区画は、システム・カタログ視点のあるデータベース区画にロック・グラフ を送ります。このシステム・カタログ視点では、グローバルなデッドロック検出が行われます。

デッドロックが検出されると、デッドロック検出機能はデッドロック状態のプロセスのうちどれをロールバックするかを選択します。選択されたプロセスは活動状態にされ、呼び出し側アプリケーションに SQLCODE -911 (SQLSTATE 40001) 理由コード 2 で戻されます。データベース・マネージャーは選択されたプロセスを自動的にロールバックします。ロールバックが完了すると、選択されたプロセスに属するロックは解除され、それによってそのデッドロックにかかわっていた他のプロセスが先に進めるようになります。

デッドロック検出機能の適切な時間間隔を選択することは、適正なパフォーマンスを確保するうえで必要なことです。時間間隔が短すぎると不必要なオーバーヘッドが生じ、長すぎるとデッドロックによるプロセスの遅延が長くなってしまいます。たとえば、起動時間間隔を 30 分に設定すると、デッドロックがほとんど 30 分間存在し続ける可能性があります。 アプリケーション設計者は、デッドロックを解決するための遅延と、デッドロックを検出するオーバーヘッドとの間でうまくバランスを保つ必要があります。

区分データベースの場合は、時間間隔は、すべての区画で同じにする必要があります (*dlchktime* 構成パラメーターを、すべての区画で同じ値に更新しなければなりません)。カタログ・ノードでの値が他の区画での値よりも小さかった場合には、実際には起きていないのにデッドロックと見なされ検出される場合があります。カタログ・ノードでの値が他の区画での値よりも大きい場合には、 2 回分より長い時間間隔が経過してからデッドロックが検出されるようなことが起こりえます。大量のデッドロックが区分データ

ベースで検出される場合には、 *dlchktime* パラメーターの値を大きくして、ロック待機や通信待機を解決するようにしてください。

データベースにアクセスする独立したプロセスが複数個あるアプリケーションが、デッドロックが生じやすい構造になっている場合は、別の問題が起きる可能性があります。その一例は、いくつかのプロセスが同じ表にアクセスして、読み取りを行ってから書き込みを行うようになっているアプリケーションです。それらのプロセスが最初に読み取り専用 SQL 照会を行い、次に同じ表に対する SQL 更新を行うと、プロセス相互間で同じデータに対する競合が生じる可能性があるため、デッドロックの起きる可能性が大きくなります。たとえば、2 つのプロセスが同じ表を読み取った後でその表の更新を行うと、プロセス A がある行に対する X ロックを入手しようとしているが、プロセス B がその行に対する S ロックを持っている (あるいは、その反対) という状況になります。結果として、デッドロックになってしまう可能性があります。こうしたデッドロックを避けるため、修正する目的でデータにアクセスするアプリケーションは、選択を実行するときに FOR UPDATE OF 文節を使用するようにしてください。それによって、プロセス A がデータを読み取ろうとするとき、 U ロックがかけられるようになります。

注: デッドロックが生じるときに記録されるモニターを定義することもできます。モニターを作成するには、 *SQL 解説書* で説明されている CREATE EVENT ステートメントを使用します。

アプリケーションがニックネームにアクセスする連合システム環境では、アプリケーションによって要求されたデータが、データ・ソースでデッドロックが起こっているために使用できない可能性があります。 このことが起こると、 DB2 はデータ・ソースでのデッドロック処理機能により、ロックを解決します。 複数のデータ・ソースにまたがるデッドロックが生じる場合は、 DB2 はデータ・ソースのタイムアウト機構により、デッドロックを解除します。

データベース・マネージャー構成パラメーター *diaglevel* が 4 に設定され、ロック要求がデッドロックのために失敗した場合、 db2diag.log にはさらに情報が追加されている可能性があります。この情報には、オブジェクト、ロック・モード、およびこのオブジェクトへのロックを保持しているアプリケーションが含まれます。また、現行の動的 SQL ステートメントまたは静的パッケージも見つかる可能性があります。

ロックに影響する要因

データベース・マネージャーのロックのモードと細分性は、いくつかの要因によって、つまり、アプリケーションが実行する処理の種類、アプリケーションがデータをアクセスする方法、および指定可能なパラメーターによって決まります。

アプリケーションの処理

ロックの属性を決めるために、処理を次の 4 つのタイプのいずれかに分類することができます。

読み取り専用

このタイプには、本質的に読み取り専用であるＳＥＬＥＣＴステートメント (カーソルについては *SQL 解説書* を参照)、明示的な FOR READ ONLY 文節を含むＳＥＬＥＣＴステートメント、あるいは、明示はされていないが PREP コマンドまたは BIND コマンドに指定された BLOCKING オプションの値に基づいて SQL コンパイラーが読み取り専用と見なしたＳＥＬＥＣＴステートメントがすべて含まれます。この場合、必要なのは共用ロック (S または IS) だけです。

変更を意図

このタイプには、FOR UPDATE 文節を含むすべてのＳＥＬＥＣＴステートメント、または不明確なステートメントの解釈の結果として変更を意図したものと SQL コンパイラーが見なしたＳＥＬＥＣＴステートメントが含まれます。この場合、共用および更新ロック (行では S、U、および X、表では IX、U、X) を使います。

変更　このタイプには UPDATE、INSERT、および DELETE が含まれますが、UPDATE WHERE CURRENT OF または DELETE WHERE CURRENT OF は含まれません。これには排他ロック (X または IX) が必要です。

カーソル制御

このタイプには UPDATE WHERE CURRENT OF および DELETE WHERE CURRENT OF が含まれます。これにも排他ロック (X または IX) が必要です。

副選択ステートメントの結果に基づいて目的の表に挿入、更新、または削除を行うステートメントは、2 種類の処理を行います。副選択に返される表のロックは、読み取り専用処理の規則によって決められます。目的の表のロックは、変更処理の結果によって決まります。

アクセス・パス

アクセス・パス とは、特定の表参照からデータを取り出すために最適化プログラムによって選択される方式です。 (160ページの『データ・アクセス概念および最適化』を参照。) 最適化プログラムによって選択されるアクセス・パスは、ロック・モードに重大な影響を与える可能性があります。たとえば、索引走査を使ってある特定の行を見つける場合、最適化プログラムはおそらく表については行レベル・ロック (IS) を選択するでしょう。このようなアクセスは、次のようなステートメントによって、 EMPLOYEE 表 (従業員番号 EMPNO に基づく索引がある) から単一の従業員についての情報を選択する場合に使用されます。

```
SELECT *
     FROM EMPLOYEE
    WHERE EMPNO = '000310';
```

同様に、索引を用いない場合には、表全体を順次に走査して選択した行を検索しなければならないので、単一表レベル・ロック (S) を獲得することになります。たとえば、

SEX 列の索引がない状況で、次のようなステートメントを使用してすべての男性従業員を選択するという場合には、このタイプのアクセスが使用されます。

```
SELECT *
     FROM EMPLOYEE
    WHERE SEX = 'M';
```

以下の表に、アクセス・プランの種類ごとにどのようなロックが獲得されるかの概説を示します。列見出しの定義については、61ページの『アプリケーションの処理』を参照してください。アクセス方式の定義については、 160ページの『データ・アクセス概念および最適化』も参照してください。 *カーソル制御* 型の処理では、アプリケーションが更新または削除する行を見つけるまで、基礎となるカーソルのロック・モードを使います。 この種の処理では、カーソルのロック・モードが何であっても、更新や削除を行うときは必ず排他ロックが獲得されます。

以下の表では、ロック・モードが 1 つだけ表示されている場合、それは表レベルのロック・モードです。 2 つのロック・モードが表示されている場合、最初のモードは表レベルのロック・モードで、 2 番目のモードは行レベルのロック・モードです。

表 5. 表走査のロック・モード

分離レベル	読み取り専用	変更を意図	変更
アクセス方式: *述部なしの表走査*			
RR	S	U	X
RS	IS / NS	IX / U	IX / X
CS	IS / NS	IX / U	IX / X
UR	IN	IX / U	IX / X
アクセス方式: *述部ありの表走査*			
RR	S	U	U
RS	IS / NS	IX / U	IX / U
CS	IS / NS	IX / U	IX / U
UR	IN	IX / U	IX / U

表 6. 索引走査のロック・モード

分離レベル	読み取り専用	変更を意図	変更
アクセス方式: *述部なしの索引走査*			
RR	S	IX / U	X
RS	IS / NS	IX / U	IX / X
CS	IS / NS	IX / U	IX / X
UR	IN	IX / U	IX / X
アクセス方式: *単一修飾行の索引走査*			

表 6. 索引走査のロック・モード (続き)

分離レベル	読み取り専用	変更を意図	変更
RR	IS / S	IX / U	IX / X
RS	IS / NS	IX / U	IX / X
CS	IS / NS	IX / U	IX / X
UR	IN	IX / U	IX / X
アクセス方式: *開始述部と停止述部のみの索引走査*			
RR	IS / S	IX / S	IX / X
RS	IS / NS	IX / U	IX / X
CS	IS / NS	IX / U	IX / X
UR	IN	IX / U	IX / X
アクセス方式: *述部ありの索引走査*			
RR	IS / S	IX / S	IX / U
RS	IS / NS	IX / U	IX / U
CS	IS / NS	IX / U	IX / U
UR	IN	IX / U	IX / U

表7 には、データ・ページの読み取りの際のロック・モードが据え置かれ、行のリストに対して以下の処理ができることが示されています。

- 複数の索引を使用して、さらに修飾する。詳細は、 167ページの『複数の索引アクセス』を参照してください。
- 効果的な事前取り出しが行えるように分類する。詳細は、 256ページの『リスト事前取り出しについて』を参照してください。

データ・ページの据え置きアクセスでは、行に対するアクセスが 2 つのステップで行われ、それによりロックのシナリオがさらに複雑になることを示唆しています。分離レベルによって 2 つの主要な区分があります。反復可能読み取り分離レベルは獲得したすべてのロックをトランザクションの終了まで保持するため、最初のステップで獲得したロックは保持されているので、 2 番目のステップでロックをさらに獲得する必要はありません。読み取り固定分離レベルおよびカーソル固定分離レベルでは、 2 番目のステップでロックを獲得する必要があります。並行性を最大化するには、最初のステップでロックを獲得しないで、修飾行だけが確実に戻されるように、すべての述部を必ず再度適用します。

表 7. 据え置きデータ・ページ・アクセスに使用される索引走査のロック・モード

分離レベル	読み取り専用	変更を意図	変更
アクセス方式: *述部なしの索引走査*			
RR	IS / S	IX / S	X

表 7. 据え置きデータ・ページ・アクセスに使用される索引走査のロック・モード (続き)

分離レベル	読み取り専用	変更を意図	変更
RS	IN	IN	IN
CS	IN	IN	IN
UR	IN	IN	IN
アクセス方式: *据え置きデータ・ページ・アクセス (述部なしの索引走査後)*			
RR	IN	IX / S	X
RS	IS / NS	IX / U	IX / X
CS	IS / NS	IX / U	IX / X
UR	IN	IX / U	IX / X
アクセス方式: *述部ありの索引走査*			
RR	IS / S	IX / S	IX / S
RS	IN	IN	IN
CS	IN	IN	IN
UR	IN	IN	IN
アクセス方式: *開始述部と停止述部のみの索引走査*			
RR	IS / S	IX / S	IX / X
RS	IN	IN	IN
CS	IN	IN	IN
UR	IN	IN	IN
アクセス方式: *据え置きデータ・ページ・アクセス (述部ありの索引走査後)*			
RR	IN	IX / S	IX / S
RS	IS / NS	IX / U	IX / U
CS	IS / NS	IX / U	IX / U
UR	IN	IX / U	IX / U

アクセス・パスはユーザーからは制御されません。最適化プログラムによって選択されます。

使用されるアクセス・パスは、ロックのモードと細分性に影響する可能性があります。たとえば、反復可能読み取り (RR) 分離レベルを使用しているアプリケーションでは、述部なしの表走査を使用する UPDATE 照会は表に対して X ロックを使用することになります。索引を使って行を検索する場合、データベース・マネージャーは表の個々の行にロックをかけることができます。

宣言済み一時表およびロッキング

宣言済み一時表は、これらを宣言したアプリケーションのみ使用できるため、ロックされていません。このタイプの表は、アプリケーションがこれを宣言してから、これを完了または切断するまでのあいだのみ存在します。

LOCK TABLE ステートメント

アプリケーションで LOCK TABLE ステートメントを使用すると、初期ロック・モードを獲得するための規則を指定変更できます。

このステートメントは表全体にロックをかけます。 LOCK TABLE ステートメントに指定された表だけがロックされます。指定された表の親表と従属表はロックされません。アクセス可能な他の表をロックすることが望ましい結果になるかどうかを、並行性とパフォーマンスの観点から判断する必要があります。その作業単位がコミットまたはロールバックされるまで、ロックは解除されません。

表が複数のユーザー間で共用されることが多い場合、次のような理由から表をロックすることもできます。

LOCK TABLE IN SHARE MODE

時間の点で一貫した データにアクセスしたい、つまり特定の時点での表の現行データにアクセスしたい場合。表への活動が頻繁な場合、表全体を一定であるようにするための唯一の方法は、表をロックすることです。たとえば、アプリケーションで表のスナップショットを取りたいとします。しかし、アプリケーションが表のいくつかの行を処理する必要があるときに、他のアプリケーションが未処理の表を更新しています。反復可能読み取りではこれが可能ですが、この処置は希望するものとは異なります。

別の手段として、アプリケーションは LOCK TABLE IN SHARE MODE ステートメントを発行できます。行が取り出されたかどうかに関係なく、行は変更できません。それら取り出した行が検索前と比べて変更されていないことが分かっているので、必要なだけの行を取り出すことができます。

LOCK TABLE IN SHARE MODE を使用すると、他のユーザーは表からデータを検索できますが、表内の行を更新、削除、または挿入することはできません。

LOCK TABLE IN EXCLUSIVE MODE

表の大部分を更新したい場合。他のすべてのユーザーがその表をアクセスできないようにすることは、更新する各行ごとにロックをかけ、変更をすべてコミットした後で行をアンロックするより安価で効率的です。

LOCK TABLE IN EXCLUSIVE MODE を使用すると、他のユーザーはすべてロックアウトされます。他のアプリケーションは、非コミット読み取りアプリケーションでない限り、表にアクセスすることはできません。

LOCK TABLE ステートメントの詳細については、 *SQL 解説書* を参照してください。

LOCK TABLE ステートメントを使用する代わりに、 LOCKSIZE パラメーターを指定した ALTER TABLE ステートメントを使用できます。 LOCKSIZE パラメーターでは、 ROW ロックまたは TABLE ロックのどちらかを選択できます。 どちらを選択しても、次に表がアクセスされるときに、それが選択されたロックの細分性となります。 ROW ロックを選択することは、表が作成されるときにデフォルト・ロック・サイズを選択することと何ら変わりありません。 TABLE ロックを選択すると、獲得する必要のあるロックの数が限定されることによって、照会のパフォーマンスが向上します。ただし、すべてのロックが表全体にわたって保持されるので、並行性は低くなります。 どちらを選択しても、通常のロック・エスカレーションに支障はありません。 ALTER TABLE ステートメントの詳細については、 *SQL 解説書* を参照してください。

CLOSE CURSOR WITH RELEASE

CLOSE CURSOR ステートメント (その中に WITH RELEASE 文節が入っている) を使用してカーソルをクローズすると、データベース・マネージャーは、そのカーソルについて保持された読み取りロックがあるなら、それらをすべて解放しようとします。読み取りロックには、表ロック IS、S、U と行ロック S、NS、U があります。ロック・モードについては、52ページの『ロックの属性』 を参照してください。

WITH RELEASE 文節は、CS または UR の分離レベルで作動しているカーソルには影響がありません。 RS または RR 分離レベルで作動するカーソルに関して WITH RELEASE 文節を指定すると、それらの分離レベルの保証はいくつか終了してしまいます。特に、RS カーソルでは*非反復可能読み取り* 現象が起こり、 RR カーソルでは*非反復可能読み取り* か*幻像読み取り* 現象が起こることがあります。

もともと RR または RS であるカーソルが、 WITH RELEASE 文節を使ってクローズされた後に再度オープンされた場合は、新たに読み取りロックが獲得されます。

CLOSE CURSOR ステートメントの他の基本文節との比較については、 79ページの『DECLARE CURSOR WITH HOLD ステートメント』を参照してください。

DB2 CLI 接続属性 SQL_ATTR_CLOSE_BEHAVIOR を CLI アプリケーションで使用すると、 CLOSE CURSOR WITH RELEASE と同じ結果を得ることができます。詳しくは、 *コール・レベル・インターフェースの手引きおよび解説書* の SQLSetConnectAttr() のセクションを参照してください。

ロックについての考慮事項のまとめ

ロックについて注意すべき点は次のことです。

- 小さな作業単位 (頻繁な COMMIT ステートメント) では、多くのユーザーによるデータの並行アクセスが促進されます。アプリケーションが論理的に一貫性ポイントにあるとき、つまり変更したデータが一貫したものになっているときには、 COMMIT ステートメントを含めます。 COMMIT が発行されると、ロックは解除されます (ただし、 WITH HOLD と宣言されたカーソルに関連する表ロックは除きます)。

- アプリケーションが行を読み取るだけでもロックは獲得されるので、読み取り専用の作業単位をコミットすることは非常に重要です。これは、共用ロックが、読み取り専用アプリケーション内で反復可能読み取り、読み取り固定、およびカーソル固定によって獲得されるからです。反復可能読み取りと読み取り固定の場合は、 WITH RELEASE 文節を使用してカーソルをクローズしない限り、すべてのロックは COMMIT が出されるまで保持されて、ロックされたデータが他のプロセスによって更新されることのないようにします。加えて、カタログ・ロックは動的 SQL を使用している非コミット読み取りアプリケーション内であっても獲得されます。
- データベース・マネージャーでは、非コミット読み取り分離レベルが使用されていない限り、アプリケーションが非コミットのデータ (他のアプリケーションによって更新されたが、まだコミットされていない行) を検索しないようにしています。
- 次の場合には、LOCK TABLE ステートメントを発行することによって、保護したい表全体をロックできます。
  - 他のアプリケーションが、行の検索はできても、更新、削除、または挿入を行えないようにする。
  - 他のアプリケーション (非コミット読み取り分離レベルのもの以外) が表の行にアクセスしないようにする。
- CLOSE CURSOR ステートメント (その中に WITH RELEASE 文節が入っている) を使用してカーソルをクローズすると、データベース・マネージャーは、そのカーソルについて保持された読み取りロックがあるなら、それらをすべて解放しようとします。
- 区分データベース内のロックに影響を与える構成パラメーターを変更するときは、データベース内のすべての区画に変更を加えたかどうか確認してください。

最適化クラスの調整

SQL 照会がコンパイルされると、多数の最適化技法を使用して、その照会に最も効果的なアクセス・プランを決定することができます。 複数の最適化技法を使用することにより、以下の効果があります。

1. 実行時パフォーマンスが向上する
2. 照会コンパイル時間が増加する
3. 使用できるシステム・リソースが増加する

このため、最適化クラスを設定することにより、照会の最適化に適用される技法の数を制限することができます。 これは、特に以下の場合に便利です。

- 非常に小さなデータベースまたは非常に単純な動的照会
- コンパイル時にデータベース・サーバーで使用できるメモリーが制限されている
- 照会コンパイルの (たとえば PREPARE) 時間を少なくしたい場合

照会最適化クラスとしては、以下に説明するもののいずれかを選択できますが、クラス 0 とクラス 9 は特殊な環境でのみ使用するようにしてください。クラス 5 はデフォル

トです。クラス 0、1、2 は、貪欲型結合列挙アルゴリズムを使用しています。複雑な照会の場合には、このアルゴリズムでは代替計画はほとんど考慮に入れないので、クラス 3 以上に比べてコンパイル時間が大幅に短縮されます。 クラス 3 以上では、動的プログラミング結合列挙アルゴリズムを使用しています。このアルゴリズムではより多くの代替計画を考慮に入れようとするので、表の数が増えるにつれて、クラス 0、1、2 に比べてコンパイル時間が大幅に長くなる可能性があります。

0 - このクラスは、最適化プログラムが最小限の最適化を使ってアクセス・プランを生成するよう指示します。たとえば:

- 最適化プログラムは、非均一分布統計を考慮しない。
- 基本的な照会書き直し規則のみを適用する (照会書き直しについては、151ページの『SQL コンパイラーによる照会書き直し』を参照)。
- 貪欲型結合列挙を行う (177ページの『最適結合を選択するための探索方式』を参照)。
- ネスト・ループ結合および索引走査アクセス方式のみが使用可能になる (172ページの『結合の概念』および 161ページの『索引走査の概念』を参照)。
- リスト事前取り出しおよび索引 AND 結合を使用不能にし、生成されたアクセス方式で使用されないようにする。
- スター型結合方式は考慮に入れない。

このクラスを使用するのは、照会コンパイル・オーバーヘッドを最小限にすることが必要な特殊な環境の場合だけにしてください。適切に索引付けされた表にアクセスする非常に単純な動的 SQL ステートメントだけで構成されるアプリケーションでは、照会最適化クラス 0 が適しています。

1 - このクラスは、DB2/6000 バージョン 1 の機能に、バージョン 1 にはない追加の低コスト機能をいくつか合わせたものにほぼ匹敵する最適化を使用するよう、最適化プログラムに指示します。特に、以下のようになります。

- 最適化プログラムは、非均一分布統計を考慮しない。
- DB2/6000 バージョン 1 で提供されるものを含めて、照会書き直し規則のサブセットのみを適用する。
- 貪欲型結合列挙を行う (177ページの『最適結合を選択するための探索方式』を参照)。
- リスト事前取り出しおよび索引 AND 結合を使用不能にし、生成されたアクセス方式で使用されないようにする。

  注: 索引 AND 結合は、スター型結合で見つかった半結合を処理する際に引き続き使用されます。

最適化クラス 1 は、マージ走査結合と表走査も使用可能である点を除けば、クラス 0 と同じ働きをします。

2 - このクラスは、最適化をクラス 1 よりも大幅に向上させ、複雑な照会の場合のコンパイル・コストをクラス 3 以上に比べてはるかに低く抑えるような最適化を使用するよう、最適化プログラムに指示します。特に、

- 使用可能な統計すべて (頻度と変位値の両方の非均一分布統計を含む) を使用する。
- 非常にまれな場合にしか適用できない、計算を多用する規則を除き、すべての照会再作成規則が適用されます。これには、要約表に対する照会の経路指定が含まれます。
- 貪欲結合列挙が使用されます。
- リスト・プリフェッチおよび要約表経路指定を含む広い範囲のアクセス方式が考慮されます。
- 該当する場合には、スター型結合方式が考慮される。

最適化クラス 2 は、動的プログラミングではなく貪欲型結合列挙を使用する点を除けば、クラス 5 と同様な働きをします。このクラスは、貪欲型結合列挙アルゴリズムを使用する最適化クラスの中では最も高度な最適化であり、複雑な照会の場合にあまり代替計画を考慮しないので、クラス 3 以上と比べてコンパイル時間は少なくて済みます。したがって、意思決定支援またはオンライン分析処理 (OLAP) 環境において非常に複雑な照会を行う場合には、このクラスをお勧めします。このようなケースでは、同じ照会は頻繁には行われないので、その照会が次に行われるまでそのアクセス・プランがキャッシュに残っていることはほとんどありません。

3 - このクラスでは、アクセス・プランの生成のために中程度の量の最適化を実行するよう要求します。このクラスは、DB2 (MVS/ESA 版) または DB2 (OS/390 版) の照会最適化の特性に最も近いものです。この最適化クラスには、次の特性があります。

- 使用可能であれば、頻繁に発生する値の追跡を行う非均一分布統計を使用する。
- 「副照会から結合への変換」を含めて、ほとんどの照会書き直し規則を適用する。
- 次に示す動的プログラミング結合列挙 (177ページの『最適結合を選択するための探索方式』を参照) を使用する。
  - 複合内部表の限定使用 (179ページの『複合表』を参照)
  - 『参照』表に関係するスタースキーマに対するカルテシアン積の限定使用 (178ページの『スター型結合のための探索方式』を参照)
- リスト事前取り出し、索引 AND 結合、スター型結合を含めた広範囲のアクセス方式が考慮される。

このクラスは、広い範囲のアプリケーションに適しています。 このクラスを使用すると、最適化プログラムが、 4 つ以上の結合の照会に最適のアクセス・プランを選択できるようになります。 しかし、最適化プログラムがより良いプランを考慮するのに失敗し、デフォルトの照会最適化クラスを選択することもあります。

5 - このクラスは、最適化プログラムが大幅な最適化を使ってアクセス・プランを生成するよう指示します。 たとえば、クラス 5 には以下のような特性があります。

- 使用可能な統計すべて (頻度と変位値の非均一分布統計を含む)。
- 照会の要約表への経路指定を含めて、すべての照会書き直し規則を適用する (ただし、まれなケースにしか適用されない計算量が多い規則を除く)。
- 次に示す動的プログラミング結合列挙 (177ページの『最適結合を選択するための探索方式』を参照) を使用する。
  - 複合内部表の限定使用 (179ページの『複合表』を参照)
  - 『参照』表に関係するスタースキーマに対するカルテシアン積の限定使用 (178ページの『スター型結合のための探索方式』を参照)
- リスト事前取り出し、索引の AND 結合、および要約表の経路指定を含めた広範囲のアクセス方式が考慮される。

最適化プログラムが、複合動的 SQL 照会に追加のリソースおよび処理時間が保証されないことを検出すると、最適化は縮小されます。縮小のエクステントつまりサイズは、マシンのサイズと述部の数によって決まります。

照会最適化プログラムが実行される照会最適化の量を縮小すると、通常は適用される照会書き直し規則をすべて適用し続けます。しかし、照会最適化プログラムは貪欲型結合列挙方法を使用するため、考慮されるアクセス・プランの組み合わせの数が少なくなります。

照会最適化クラス 5 は、トランザクションと複合的な照会の両方で構成される混合環境に適した選択です。この最適化クラスは、最も価値のある照会変換技法およびその他の照会最適化技法を、効率的な方法で適用させるよう設計されています。

7 - このクラスは、最適化プログラムが大幅な最適化を使ってアクセス・プランを生成するよう指示します。 複合動的 SQL 照会の照会最適化の量を縮小しないことを除けば、照会最適化クラス 5 と同じです。

9 - このクラスは、最適化プログラムが使用可能なすべての最適化技法を使用するよう指示します。それには、次のものが含まれます。

- すべての使用可能な統計。
- すべての照会書き直し規則。
- カルテシアン積および無制限の複合内部を含めて、結合列挙で可能なものすべて。
- すべてのアクセス方式。

このクラスでは、最適化プログラムによって考慮される可能なアクセス・プランの数が大幅に拡張されます。 このクラスは、より包括的な最適化によって、大型の表を使用する非常に複雑かつ非常に長時間実行する照会に適したアクセ

ス・プランを生成できるかどうかを調べるために使用します。 よりすぐれたプランが見つかったかどうかを、 Explain とパフォーマンスの測定値を使って調べる必要があります。

最適化クラスの設定方法

特定の照会最適化クラスを要求する方法は、静的 SQL を使用するか動的 SQL を使用するかによって違います。

- *静的 SQL* ステートメント では、 PREP コマンドおよび BIND コマンドに指定される最適化クラスが使用されます。 SYSCAT.PACKAGES カタログ表内の QUERYOPT 列には、パッケージをバインドするのに使われる最適化クラスが記録されています。パッケージが暗黙のうちに、または REBIND PACKAGE コマンドを使って再バインドされる場合、この同じ最適化クラスが静的 SQL ステートメントに使用されます。これらの静的 SQL ステートメントに使用される最適化クラスを変更したい場合は、BIND コマンドを使用しなければなりません。最適化クラスを指定しなかった場合には、 DB2 は *dft_queryopt* データベース構成パラメーターによって指定されたデフォルトの最適化技法を使用します。
- *動的 SQL* ステートメント では、 SQL の SET ステートメントを使用するよう設定されている CURRENT QUERY OPTIMIZATION 特殊レジスターで指定される最適化クラスが使用されます。 たとえば、次のステートメントは最適化クラス 1 の設定を行います。

  ```
  SET CURRENT QUERY OPTIMIZATION = 1
  ```

  動的 SQL ステートメントが常に同じ最適化クラスを使用するようにするには、この SET ステートメントをアプリケーション・プログラムに組み入れることもできます。詳細については、*SQL 解説書* を参照してください。

  CURRENT QUERY OPTIMIZATION レジスターが設定されていない場合、動的ステートメントは、デフォルトの照会最適化クラスでバインドされます。動的および静的の両方の SQL のデフォルト値は、データベース構成パラメーター *dft_queryopt* の値によって決まります。デフォルトを変更していない限り、クラス 5 がデフォルトの照会最適化クラスとなります。 (このパラメーターの詳細については、449ページの『デフォルトの照会最適化クラス (dft_queryopt)』を参照してください。) バインド・オプションおよび特殊レジスターのデフォルト値は、 *dft_queryopt* データベース構成パラメーターに入っているものが使われます。

どの程度の最適化が必要か

ほとんどのステートメントは、デフォルトの照会最適化クラスで、ある程度の量のリソースを使って、十分に最適化できます。 特定の最適化クラスでの照会コンパイル時間とリソース消費は、主として、照会の複雑度、特に結合と副照会の数によって影響を受けます。 しかし、コンパイル時間とリソースの使用量も、様々な最適化クラスに対して実

行される最適化の数によって影響を受けます。最適化クラスの場合、単一の照会と非常に複雑な照会とでは、照会コンパイル時間とリソース消費に歴然とした差が現れることを予想できます。

どの最適化クラスを使用するかを選択する際には、以下に示すことを参考にしてください。

- 始めは、デフォルトの照会最適化クラスを使用してみる。
- デフォルト以外のクラスを使用したい場合は、まず 1、2、3 のいずれかを試す。
- 実行時間が非常に短い照会、つまり 1 秒未満の照会には、低い最適化クラス (0 または 1) を使用する。 (下位の最適化クラスを選択する場合についての付加的な基準は、以下の説明をご覧ください。)
- 同じ列上にたくさんの結合述部を持つ表が多数ある場合で、コンパイル時間に制約があるという場合には、最適化クラス 1 または 2 を使用する。
- 実行時間の長い照会、つまり 30 秒以上かかる照会には、高い最適化クラス (3、5、または 7) を使用する。
- 通常の状況では、最適化クラス 9 は使用しない。
- 長い時間がかかる照会の場合には、その照会を db2batch を使用して実行し、コンパイルに使われる時間と実行に使われる時間を判別する必要がある。
  - ほとんどの時間がコンパイルに使われている場合には、最適化クラスを低くする。
  - ほとんどの時間が実行に使われている場合は、最適化クラスを高くすることを検討する。

照会最適化クラス 1、2、3、5、および 7 はすべて、汎用に適していることに注意してください。

照会コンパイル時間をさらに短くし、 SQL (たとえば、極めて単純なステートメント) の種類を知る必要がある場合のみ、クラス 0 を考慮するようにしてください。この SQL には、以下の特性があります。

- 単一の表または少しの表だけにアクセスする。
- 単一の行または少しの行だけを取り出す。
- 完全修飾した固有索引を使用する。

オンライン・トランザクション処理 (OLTP) のトランザクションは、この種の SQL の良い例です。

複雑な照会では、最適なアクセス・プランを選択するのにさまざまな量の最適化が必要になる場合があります。 以下の特性を示す照会については、より高度の最適化クラスを使用することもできます。

- 大きい表へのアクセス
- 述部の数が多い
- 副照会が多い
- 結合が多い

- UNION や INTERSECT などの集合演算子が多い
- 修飾行が多い
- GROUP BY および HAVING 操作
- ネストした表式
- 視点の数が多い

完全に正規化されたデータベースに対する判断支援照会または月末報告照会のようなものは、少なくともデフォルト照会最適化クラスが使用される複合照会の良い例です。

もっと高度な照会最適化クラスを使用するもう 1 つの理由は、照会生成プログラムによって生成される SQL です。 多くの照会生成プログラムが作成する SQL は効率的ではありません。 照会生成プログラムによって生成されたものも含めて、適切に作成されていない照会の場合は、さらに最適化を行って良好なアクセス・プランを選択できるようにしなければならない場合があります。 照会最適化クラス 2 以上を使用すると、あまり上手に作成されていない SQL 照会でも改善することができます。

静的 SQL または動的 SQL の使用、および同じ動的 SQL が繰り返し実行されるかどうか、ということも重要な考慮事項です。 静的 SQL の場合、照会コンパイル時間とリソースは 1 回だけ費やされ、その結果としての計画は大量の時間を使用する可能性があります。 一般に、静的 SQL は常にデフォルトの照会最適化クラスを使用します。 動的ステートメントは実行時にバインドされ実行されるので、動的ステートメントのための追加の最適化のオーバーヘッドが生じても総体的なパフォーマンスが向上するのかどうかということを検討してください。 ただし、同じ動的 SQL ステートメントが繰り返し実行される場合には、選択されたアクセス・プランはキャッシュされることになります。 したがって、照会最適化クラスを選択するときには、この種のステートメントは静的 SQL ステートメントと同様に扱うことができます。

(静的 SQL と動的 SQL をいつ使用するのかについての詳細は、 *アプリケーション開発の手引き* を参照してください。)

最適化を追加することで照会が便利になると考えられるが、その確証がない場合、またはコンパイル時間およびリソース使用のことを考慮している場合は、何らかのベンチマーク・テストを実行することができます。このテストによって、異なる最適化クラスから取得される利点を量で表すことができます。 一般的な技法および `db2batch` ツールの具体的な使用法については、 317ページの『第12章 ベンチマーク・テスト』を参照してください。ベンチマーク・テストを設計して実行するときは、以下に示すように、アプリケーション内の SQL ステートメントが静的か動的かに合わせて検討を行ってください。

- **動的** SQL ステートメントの場合は、そのテストで、ステートメントの平均実行時間を比較するようにしてください。以下の式を使って、平均実行時間を計算することができます。

$$\frac{\text{コンパイル時間 + すべての反復の実行時間の合計}}{\text{反復回数}}$$

ここで、**反復回数**は SQL ステートメントをコンパイルするたびに SQL ステートメントが実行すると予期されている回数を表します。

注: 初期コンパイルに続いて、環境の変化に伴ってステートメントの再コンパイルが必要になると、動的 SQL ステートメントは再コンパイルされます。 ある SQL ステートメントがいったんキャッシュされると、環境が変わらないと仮定すると、後続の PREPARE ステートメントはキャッシュされたステートメントを再使用するので、その SQL ステートメントを再度コンパイルする必要はなくなります。 (動的ステートメントを用いた処理時のパフォーマンスを改善できるキャッシュについては、
352ページの『カタログ・キャッシュ・サイズ (catalogcache_sz)』および
360ページの『パッケージ・キャッシュ・サイズ (pckcachesz)』を参照してください。)

- **静的** SQL ステートメントの場合は、そのテストで、ステートメント実行時間を比較するようにしてください。

  注: 静的 SQL のコンパイル時間にも関心があるとしても、ステートメントの合計時間 (コンパイルと実行のための時間) を、意味のあるコンテキストで使用するのは難しいことです。 合計時間の比較という方法では、静的 SQL ステートメントは 1 回バインドするたびに何度も実行可能であるという事実や、通常は実行時にはバインドしないという事実は考慮されていません。

結果セットの制約によるパフォーマンス向上

SELECT ステートメントには検索基準を満たす一連の行を定義します。 DB2 最適化プログラムは、そのアプリケーションが修飾行をすべて検索することを想定しています。 OLTP およびバッチ環境においては、この想定が最も適しています。しかし、『ブラウズ』アプリケーションにおいては、照会で定義されている結果の集合が大規模であっても、検索するのは最初のいくつかの行だけ、通常は画面をいっぱいにするのに十分な数の行だけであるのが一般的です。

最適化プログラムによるデフォルトの設定 (すべての修飾行を検索する) は、保管データの情報を更新または削除していないアプリケーションには最適とは言えません。

パフォーマンスが向上するように、 SELECT ステートメントを修正して結果表を限定または修正する方法は 5 つあります。それらは、以下のとおりです。

- FOR UPDATE 文節
- FOR READ/FETCH ONLY 文節
- OPTIMIZE FOR n ROWS 文節
- FETCH FIRST n ROWS ONLY 文節
- DECLARE CURSOR WITH HOLD ステートメント

FOR UPDATE 文節

FOR UPDATE 文節では、その後に置かれている UPDATE ステートメントが更新できる列を識別します。 FOR UPDATE 文節を列名なしで指定する場合は、表または視点のすべての更新可能な列が含まれることになります。列名を指定する場合、それぞれの名前は修飾されていてはならず、表または視点の 1 つの列を識別している必要があります。

FOR UPDATE 文節は、次のどちらかが当てはまる場合は使用できません。

- SELECT ステートメントに関連したカーソルを削除できない。
- 選択した列のうち少なくとも 1 つが、カタログ表の更新不能な列であって、 FOR UPDATE 文節に含まれている。

DB2 CLI 接続属性 SQL_ATTR_ACCESS_MODE を CLI アプリケーションで使用すると、同じ結果を得ることができます。詳しくは、*コール・レベル・インターフェースの手引きおよび解説書* の SQLSetConnectAttr() のセクションを参照してください。

FOR READ または FETCH ONLY 文節

FOR READ ONLY 文節は、結果表が読み取り専用になることを保証します。 FOR FETCH ONLY 文節も同じ意味です。

結果表の中には、定義によって読み取り専用となっているものがあります。たとえば、読み取り専用として定義されている SELECT ステートメントの結果表などです。そのような場合でも FOR READ ONLY 文節を指定できますが、文節の効果はありません。

更新と削除が許可されている結果表では、 FOR READ ONLY を指定すると、FETCH 操作のパフォーマンスが向上することがあります。パフォーマンスが向上する可能性があるのは、データベース・マネージャーがデータに対してブロック化 (排他ロックではない) を行うことができる場合です。指定した UPDATE または DELETE ステートメントで照会が使用されている場合を除き、パフォーマンスを向上させるには FOR READ ONLY 文節を使用する必要があります。

DB2 CLI 接続属性 SQL_ATTR_ACCESS_MODE を CLI アプリケーションで使用すると、同じ結果を得ることができます。詳しくは、*コール・レベル・インターフェースの手引きおよび解説書* の SQLSetConnectAttr() のセクションを参照してください。

OPTIMIZE FOR n ROWS 文節

OPTIMIZE FOR 文節では、結果のサブセット 1 つだけを検索するのが目的なのか、または最初の数行の検索を優先的に行うのが目的なのかを宣言する機構をアプリケーションに提供します。この目的が理解されたら、最適化プログラムは、最初の数行を検索するための応答時間を最小化するアクセス・プランを優先することができます。さらに、単一ブロックとしてクライアントに送られる行数 (79ページの『行のブロック化』を参照) は、 OPTIMIZE FOR 文節の 『n』という値によってバインドされます。したがっ

て、OPTIMIZE FOR は修飾行がサーバーによってデータベースから検索される方法と、修飾行がクライアントに戻される方法の両方に影響を与えます。

たとえば、従業員表で基本給が最高の従業員を照会するとします。

```
   SELECT LASTNAME,FIRSTNAME,EMPNO,SALARY
      FROM EMPLOYEE
ORDER BY SALARY DESC
```

SALARY 列では降順索引を定義しました。しかし、従業員の順序は従業員番号順になっているため、給与索引のクラスター化が不十分であることが考えられます。最適化プログラムは、多数のランダム同期入出力が行われないように、修飾行の行 ID の分類を必要とするリスト事前取り出しアクセス方式 (256ページの『リスト事前取り出しについて』を参照) を使用することを選択します。このため、最初の修飾行がアプリケーションに戻される前に遅延が起こります。 OPTIMIZE FOR 文節を次のようにステートメントに追加することにより、

```
   SELECT LASTNAME,FIRSTNAME,EMPNO,SALARY
      FROM EMPLOYEE
ORDER BY SALARY DESC
OPTIMIZE FOR 20 ROWS
```

最適化プログラムは、おそらく、最高給を得ている 20 人の従業員だけを取り出すことを認識して、 SALARY 索引を直接使用することを選択します。ブロック化された行数に関係なく、行のブロックは 20 行ごとにクライアントに戻されます。

OPTIMIZE FOR 文節を使用すると、最適化プログラムは、大量データ操作または行の流れを妨げる操作 (分類など) を行わないアクセス・プランを行おうとします。
OPTIMIZE FOR 1 ROW を使用すると、アクセス・パスに最も影響を与えることになります。結果として、この文節を使用すると、次のような効果があります。

- 複合内部表のある結合順序では一時表を必要とするため、使用することは少ない。
- 結合方法を変更できる。ネストされたループ結合はオーバーヘッド・コストが少なく、通常は数行を選択しない場合にだけ効果があるので、この結合が選択しやすくなっています。
- ORDER BY 文節に一致する索引が選ばれやすい。 ORDER BY では分類が不要になるためです。
- リスト事前取り出しは分類を必要とする方法であるため、このアクセス方式が選ばれることは少ない。
- 順次事前取り出しはわずかな行数だけを表示したいことを意味するため、 DB2 から要求されることはほとんどない。
- 結合照会では、ORDER BY 文節に必要な順序を指定した索引が外部表にある場合は、 ORDER BY 文節にある列で成る表が外部表として選ばれやすい。

OPTIMIZE FOR 文節はすべての最適化クラスに適用されますが (68ページの『最適化クラスの調整』を参照)、最適化クラス 3 かそれ以上のクラスで最も効果的に作業しま

す。最適化クラスが 3 より下のクラスで貪欲型結合列挙方法 (177ページの『最適結合を選択するための探索方式』を参照) を使用すると、最初の数行の素早い検索には役に立たない複数表結合のアクセス・プランになることがあります。

OPTIMIZE FOR 文節では、修飾行をすべて検索します。しかし、修飾行をすべて検索するための合計経過時間は、最適化プログラムが応答セット全体を最適化した場合よりも大幅に増える可能性があります。

コール・レベル・インターフェース (DB2 CLI または ODBC) を使用するパッケージ・アプリケーションがある場合は、 db2cli.ini 構成ファイルにある OPTIMIZEFORNROWS キーワードを用いて、 DB2 CLI に OPTIMIZE FOR 文節を各照会ステートメントの終わりに自動的に追加させることができます。詳細については、*コール・レベル・インターフェースの手引きおよび解説書* を参照してください。

データをニックネームから選択する場合、データ・ソース・サポートによって結果は異なります。ニックネームによって参照されるデータ・ソースが OPTIMIZE FOR 文節をサポートしており、 DB2 最適化プログラムがこの文節を含む照会全体をデータ・ソースに後入れ先出しする場合、この文節はデータ・ソースに送られたリモート SQL 内で生成されます。データ・ソースでこの文節がサポートされていない場合、または最適化プログラムがこの文節をローカルに実行することを決定する (コストが最低のプラン) 場合、 OPTIMIZE FOR 文節は DB2 でローカルに適用されます。この場合、DB2 最適化プログラムは引き続き、照会の最初の数行を検索する応答時間を最小限にするアクセス・プランを優先して選びますが、プランの生成に最適化プログラムが利用できるオプションははっきり区切られず、 OPTIMIZE FOR 文節によるパフォーマンスの向上もほとんどありません。

FETCH FIRST 文節と OPTIMIZE FOR 文節の両方が指定されている場合は、どちらか低い方の値が通信バッファー・サイズに影響を与えるものとして使用されます。この 2 つの値は、最適化という目的のために独立して考慮されます。これら 2 つの文節の相互関係について詳しくは、81ページの『SELECT ステートメントの使用』を参照してください。

FETCH FIRST n ROWS ONLY 文節

OPTIMIZE FOR *n* ROWS 文節は、すべての修飾行の検索を防止するわけではありません。 (修飾行をすべて検索するための合計経過時間は、最適化プログラムが応答セット全体を最適化した場合よりも大幅に増える可能性があります。)

FETCH FIRST *n* ROWS ONLY 文節は、 SELECT ステートメント内から検索できる最大行数を設定します。結果表を最初の数行に制限すると、パフォーマンスが向上します。 この文節が指定されていない SELECT ステートメントに基づく結果表に存在する行数にかかわりなく、 n 行だけが検索されます。

FETCH FIRST 文節と OPTIMIZE FOR 文節の両方が指定されている場合は、どちらか低い方の値が通信バッファー・サイズに影響を与えるものとして使用されます。この 2

つの値は、最適化という目的のために独立して考慮されます。これら 2 つの文節の相互関係について詳しくは、81ページの『SELECT ステートメントの使用』を参照してください。

DECLARE CURSOR WITH HOLD ステートメント

WITH HOLD 文節を含む DECLARE CURSOR ステートメントを指定してカーソルを宣言すると、トランザクションがコミットされるときに、すべてのオープン・カーソルは開いたままの状態になります。さらに、WITH HOLD オープン・カーソルの現行カーソル位置を保護しているロックを除き、すべてのロックが解放されます。

WITH HOLD 文節を含む DECLARE CURSOR ステートメントを指定してカーソルを宣言すると、トランザクションが ROLLBACK で終了するときに、すべてのオープン・カーソルは閉じます。さらに、すべてのロックが解除されて LOB ロケーターが解放されます。

CLOSE CURSOR ステートメントの他の基本文節との比較については、 67ページの『CLOSE CURSOR WITH RELEASE』を参照してください。

DB2 CLI 接続属性 SQL_ATTR_CURSOR_HOLD を CLI アプリケーションで使用すると、同じ結果を得ることができます。詳細については、*コール・レベル・インターフェースの手引きおよび解説書* の『SQLSetStmtAttr - ステートメントに関連したオプションの設定』を参照してください。

コール・レベル・インターフェース (DB2 CLI または ODBC) を使用するパッケージ・アプリケーションがある場合は、 `db2cli.ini` 構成ファイルにある CURSORHOLD キーワードを用いて、 DB2 CLI にすべての宣言されているカーソルについて WITH HOLD 文節が指定されているものと自動的に想定させることができます。詳しくは、*コール・レベル・インターフェースの手引きおよび解説書* のトランザクション構成キーワードのセクションを参照してください。

行のブロック化

行のブロック化は、単一の操作で複数の行からなるブロック を取り出し、データベース・マネージャーのオーバーヘッドを少なくするためのテクニックです。それらの行はキャッシュに格納され、アプリケーションの各 FETCH 要求ごとに一度に 1 行ずつ処理されます。ブロック内の全部の行が処理されると、別の行ブロックがデータベース・マネージャーによって取り出されます。

キャッシュは、アプリケーションが OPEN CURSOR 要求を発行する時に割り振られ、カーソルがクローズされるときに割り振り解除されます。キャッシュのサイズは、入出力ブロックのためのメモリーを割り振るときに使用される構成パラメーターによって決まります。使用されるパラメーターは、クライアントがローカルであるかリモートであるかによって異なります。

- *ローカル・アプリケーション* の場合、行のブロック化のためのキャッシュを割り振るのに、パラメーター *aslheapsz* が使用されます。 (このパラメーターについては、376ページの『アプリケーション・サポート層ヒープ・サイズ (aslheapsz)』を参照してください。)
- *リモート・アプリケーション* の場合、行ブロック化のためのキャッシュ を割り振るのに、クライアント・ワークステーション側のパラメーター *rqrioblk* が使用されます。キャッシュは、データベース・クライアントで割り振られます。 (このパラメーターについては、 379ページの『クライアント入出力ブロック・サイズ (rqrioblk)』を参照してください。)

*ローカル・アプリケーション*の場合、 1 ブロックごとに返される行の数は下記の式で計算されます。ただし、
- *aslheapsz* は、ページ数単位のメモリー領域
- 4 096 は、1 ページ当たりのバイト数
- *orl* は、出力行の長さ (バイト数)

```
Rows per block = aslheapsz * 4096 / orl
```

*リモート・アプリケーション*の場合、 1 ブロックごとに返される行の数は下記の式で計算されます。ただし、
- *rqrioblk* はバイト数単位のメモリー領域
- *orl* は、出力行の長さ (バイト数)

```
Rows per block = rqrioblk / orl
```

SELECT ステートメントの中で、FETCH FIRST *n* ROWS ONLY 文節、または OPTIMIZE FOR *n* ROWS 文節を使用した場合、 1 ブロック当たりの行数は、以下の 2 つの値のうち小さい方になります。
- 上記の式で計算される値
- FETCH FIRST 文節の *n* の値
- OPTIMIZE FOR 文節の *n* の値

PREP コマンドと BIND コマンドで BLOCKING オプションを使用すると、以下のいずれかの種類の行ブロック化を指定できます。

UNAMBIG
: 読み取り専用カーソルと 『FOR UPDATE OF』と指定されていないカーソルの場合に、ブロック化が行われる。未確定のカーソルは、更新可能として扱われます。

ALL
: 読み取り専用カーソルと 『FOR UPDATE OF』と指定されていないカーソルの場合に、ブロック化が行われる。未確定のカーソルは、読み取り専用として扱われます。

NO
: どのカーソルの場合もブロック化は行われない。未確定のカーソルは、読み取り専用として扱われます。

ここに示したタイプの行ブロック化の詳細については、 *コマンド解説書* の中の PREP コマンドと BIND コマンドの説明を参照してください。

PREP および BIND コマンドにオプションを何も指定しない場合、デフォルトの行ブロック化タイプは UNAMBIG になります。コマンド行プロセッサーおよびコール・レベル・インターフェースの場合、デフォルトの行ブロック化は ALL です。

カーソルの詳細については、*SQL 解説書* を参照してください。

照会の調整

ここでは、アプリケーション・プログラムにおいて SQL ステートメントを細かく調整するための考慮事項や指針について説明します。一般にこれらの指針は、非常に大きな表のデータにアクセスするために必要なシステム・リソースと時間の使用を最小限に抑えるプログラムを設計するのに役立ちます。 SQL ステートメントをコンパイルするときに起きる最適化の量によっては、 SQL ステートメントを細かく調整する必要がない場合もあります。 SQL コンパイラーでは、SQL をもっと効率的な形式で書き直すことができます。
151ページの『SQL コンパイラーによる照会書き直し』および 68ページの『最適化クラスの調整』を参照してください。

また、最適化プログラムによって選択されたアクセス・プランが他の要因 (環境についての考慮事項やシステム・カタログ統計を含む) からも影響を受けるということに注意しておくのも重要なことです。アプリケーションのパフォーマンスについてベンチマーク・テストを行うと、アクセス・プランを改善するように調整を行うことができます。

SELECT ステートメントの使用

SQL 言語は、非常に柔軟性の高い高水準言語です。そのため、同じデータを検索するのにも、いろいろと違う *SELECT* ステートメント を作成することができます。しかし、形式が異なり、最適化のクラスが異なると、パフォーマンスが変化する可能性があります。

SQL コンパイラー (書き直しと最適化のフェーズを含む) は、アクセス・プランを選択して、コーディングされた照会に対する結果セットを生成します。したがって、以下の指針の多くで示されているように、必要なデータだけが獲得されるように照会をコーディングする必要があります。

SELECT ステートメントを使用する際の指針

SELECT ステートメント を使用する際の指針は、以下のとおりです。

- 選択リスト内で必要な列だけを指定します。 1 つのアスタリスク (*) を使ってすべての列を指定するほうが簡単だとしても、不必要な処理がなされたり不要な列が戻されたりする可能性があります。

- 述部を使うことによって、選択される行の数を限定して、応答セットを必要な行だけに制限します。 (述部の種類とそれらがパフォーマンスに与える影響の違いについての詳細は、 170ページの『述部の用語』を参照してください。)
- 使用する行数が戻される行数の合計よりかなり少なくなる場合には、 *SELECT* ステートメント に OPTIMIZE FOR を指定してください。この文節は、アクセス・プランの選択と通信バッファーでブロック化される行数の両方に影響します。 (詳細については、79ページの『行のブロック化』を参照してください。)
- 検出される行数が少ない場合には、 FETCH FIRST n ROWS ONLY 文節の他に OPTIMIZE FOR k ROWS 文節を指定する必要はありません。ただし、n の値が大きいのではじめの k 行を取り出してから、後でまた k 行を取り出したい場合には、両方の値を指定してください。通信バッファーのサイズとして n と k の中で小さいほうが使われます。

  ```
  SELECT EMPNAME, SALARY FROM EMPLOYEE
     ORDER BY SALARY DESC
     FETCH FIRST 100 ROWS ONLY
     OPTIMIZE FOR 20 ROWS
  ```

- FOR READ ONLY (または FOR FETCH ONLY) 文節を指定するなら、照会が行ブロック化を利用できるようになりパフォーマンスが向上します。さらに、この文節が指定された照会によって検索される行には排他ロックがかけられないため、データの並行性も向上します。また、さらに照会書き直しを行うこともできます。
  BLOCKING ALL BIND と一緒に FOR READ ONLY (または FOR FETCH ONLY) 文節を指定するなら、連合システム内のニックネームに対する照会のパフォーマンスが向上します。
- また、FOR UPDATE OF 文節を指定すると、更新されるカーソルにとっては、データベース・マネージャーが最初に適切なロック・レベルを選択できるようになり、潜在的なデッドロック (60ページの『デッドロック』を参照) およびロック変換 (57ページの『ロックの変換』を参照) が回避されるため、パフォーマンスが向上します。
- 可能な限り、数値データ・タイプの変換はしないようにします。値を比較するとき、同じデータ・タイプの項目を使うようにするなら、さらに効率的です。変換が必要な場合、精度の低さのために正確でなくなったり、実行時変換のためにパフォーマンスが低下したりする可能性があります。

  可能なら、以下のデータ・タイプを使用してください。
  - 短い列では、可変長文字型ではなく、文字型
  - 浮動小数点数や 10 進数ではなく、整数
  - 文字ではなく、日時
  - 文字ではなく、数値
- 一般に DISTINCT、あるいは ORDER BY などの文節または操作を含む SQL ステートメントでは、その操作を実行するためにデータを並べ換えることが必要になります。分類操作が使用される回数を少なくしたい場合は、これらの文節の指定を必要でなければ省略してください。
- 表に行があるかどうかを調べるときに、次のステートメントは使用せず、

```
SELECT COUNT(*) FROM  TABLENAME
```

その表が非常に小さいことを知らなければ、ゼロ以外の値であるかを調べてください。表が大きくなるにつれ、全行カウントはパフォーマンスに影響するようになります。むしろ、単一行を選択してみるようにしてください。これを行うには、カーソルをオープンして 1 行を取り出すか、または単一行選択 (SELECT INTO) を行います。 (*SELECT* ステートメント から複数の行が検出される場合は、 SQLCODE -811 のエラーを必ず調べてください。)

- 更新活動が低調で表が非常に大きい場合には、述部として頻繁に使用する列に索引を定義してください。
- 複数の述部文節に同じ列が存在する場合には、 IN リストを使用することを考慮してください。
- 大きい IN リストをホスト変数と組み合わせて使用すると、ホスト変数のサブセットのループインでパフォーマンスが向上する可能性があります。

複数の表にアクセスする *SELECT* ステートメント には、特に以下のことが適用されます。

- 表を結合するときには、結合述部を使用します。 (結合述部とは、1 つの結合において異なる表の 2 つの列を比較することです。)
- 結合述部内で列に対して索引を定義し、それによって、その結合をさらに効率的に処理できるようにしてください。これは、複数の表にアクセスする SELECT ステートメントを含む UPDATE ステートメントおよび DELETE ステートメントにも、効果があります。
- 可能なら、結合述部で式または OR 文節を使用しないようにします。この場合、結合技法の中にはデータベース・マネージャーで使用できないものもあるため、結果として最も効率的な結合方式を選択できない場合があります。
- 可能ならば、区分データベース環境で、結合された表が両方とも結合列上で区画化されるようにしてください。

詳細については、 172ページの『結合の概念』を参照してください。

さらに、結合や副照会を含む SQL ステートメントのコーディング方法の詳細については、 *アプリケーション開発の手引き* を参照してください。

複合 SQL

複合 SQL を使用すると、複数の SQL ステートメントを単一の実行可能ブロックにまとめることができます。このブロックの中の SQL ステートメント (*sub-statements*) は、個々に実行することもできますが、ステートメント・ブロックごとに作成および実行すればデータベース・マネージャーのオーバーヘッドが少なくなります。リモート・クライアントの場合は、複合 SQL を使うことによって、ネットワークで送信する必要のある要求の数を少なくすることもできます。

複合 SQL には、次の 2 つの種類があります。

- **アトミック**

  アプリケーションは、すべてのサブステートメントが正常完了したとき、または 1 つのサブステートメントがエラー終了したときに、データベース・マネージャーから応答を受信します。 1 つのサブステートメントがエラー終了すると、そのブロック全体がエラー終了したと見なされ、そのブロックの中でのデータベースに対する変更がすべてロールバックされます。

- **非アトミック**

  アプリケーションは、すべてのサブステートメントが完了したときにデータベース・マネージャーから応答を受信します。先行するサブステートメントが正常完了したかどうかに関係なく、ブロックの中のすべてのサブステートメントが実行されます。このグループのステートメントをロールバックできるのは、「非アトミック」複合 SQL を含む作業単位がロールバックされる場合だけです。

- アトミック複合 SQL は DB2 コネクトではサポートされていません。
- 複合 SQL はストアード・プロシージャー (DARI ルーチンとも呼ばれる) 内でサポートされています。
- 複合 SQL は、以下のものによってサポートされています。
  - 組み込み静的 SQL (*SQL 解説書* を参照)。
  - DB2 コール・レベル・インターフェース (*コール・レベル・インターフェースの手引きおよび解説書* を参照)。
  - JDBC (*アプリケーション開発の手引き* を参照)。

動的複合ステートメント

動的複合ステートメントは、他の SQL ステートメントを 1 つの実行可能なブロックにグループ化したものです。動的複合ステートメントでは、SQL 変数の宣言、SQLSTATE に関連する条件の宣言、および 1 つ以上の SQL 手続きステートメントの指定が可能です。動的複合ステートメントでエラーが発生した場合、前の SQL ステートメントはすべてロールバックされ、動的複合ステートメントの残りの SQL ステートメントは処理されません。

動的複合ステートメントはトリガー、SQL 関数、または SQL メソッドに組み込むことができ、また、動的 SQL ステートメントを使用して発行することができます。この実行可能ステートメントは、動的に準備することができます。ステートメントを呼び出す特権は必要ありませんが、そのステートメントに関連する許可 ID は複合ステートメント内の組み込み SQL ステートメントを呼び出すために必要な特権を持っていなければなりません。

変数は変数宣言のサブステートメントにあります。条件は、条件宣言の SQLSTATE 値に基づくサブステートメントにあります。動的複合ステートメントは DB2 によって単一ステートメントとしてコンパイルされます。このステートメントは、制御フロー・ロ

ジックをほとんど含まない、有効なデータ・フローを含む短いスクリプトに対して効果的に使用することができます。ネストした複雑な制御フローを持つ大きな構成の場合、SQL プロシージャーの使用を検討してください。

動的複合ステートメント内では、複数の制御フロー・ロジック・ステートメントを使用できます。このステートメントには、FOR ステートメント、IF ステートメント、ITERATE ステートメント、および WHILE ステートメントがあります。これらのステートメント、および、サポートされているこれ以外のステートメントの詳細については、 *SQL 解説書* を参照してください。

パフォーマンスについての考慮事項および文字変換

アプリケーションとデータベースで使用されているコード・ページが同じでない場合、可能であれば、一方のコード・ページと他方のコード・ページの間でデータのマッピングが実行されます。アプリケーションとデータベースの間でのコード・ページのデータ・マッピングが正しく行われるためには、いくらかのデータ変換が必要になることがあります。

このマッピングおよびデータ変換のために、データベースのコード・ページとは違うコード・ページで実行しているアプリケーションの処理時間には、一定量のオーバーヘッドが発生します。アプリケーションとデータベースとで同じコード・ページか、または同じ照合順序を使用すると、アプリケーションのパフォーマンスが向上する場合があります。

コード・ページの変換

文字変換は次の場合に行われることがあります。

- データベースにアクセスするクライアントまたはアプリケーションが、データベースのコード・ページとは別のコード・ページで実行されているとき。

  データベースの変換はデータベース・サーバー・マシンで次のように行われます。つまり、アプリケーション・コード・ページからデータベース・コード・ページへの変換、またはデータベース・コード・ページからアプリケーション・コード・ページへの変換です。
- ファイルをインポート (またはロード) するクライアントまたはアプリケーションが、インポート (またはロード) されるファイルと異なったコード・ページで実行されるとき。
- DB2 コネクトを使用して DRDA サーバーのデータにアクセスするとき。

文字変換は次の事柄については**行われません**。

- ファイル名。
- FOR BIT DATA 属性を割り当てた列に指定したり、そこから派生したデータ、または SQL 操作によって FOR BIT または BLOB データに変換されるデータ。

- EUC または UCS-2 間での変換機能サポートがインストールされていない DB2 製品またはプラットフォーム。アプリケーションを実行すると、SQLCODE -332 (SQLSTATE 57017) が出されます。

EUC コード・ページ・サポートまたは各国語サポート (NLS) の考慮事項については、*管理の手引き: 計画* を参照してください。

オペレーティング・システムの環境によっては、マルチバイト・コード・ページを変換する際に、 DB2 データベース・マネージャーは、変換機能と変換表を使用したり、DBCS 変換 API を使用したりします。

注: マルチバイト・コード・ページ間の文字ストリング変換を行うと、 EUC を用いる DBCS と同様に、ストリングの長さが大きくなったり、小さくなったりすることがあります。

ある国の PC DBCS、EUC、および UCS-2 コード・セット中の別の文字に割り当てられたコード・ポイントは、同じ文字を分類したときに、別の結果になることがあります。さまざまな国のコード・セットで分類が必要な場合は、 *管理の手引き: 計画* を参照してください。

拡張 UNIX コード (EUC) のコード・ページ・サポート

C または C++ アプリケーションでグラフィック・データ (2 バイト文字) を使用するホスト変数を用いるには、特殊な考慮事項が必要です。それには、特殊なプリコンパイラー、アプリケーションのパフォーマンス、およびアプリケーション設計に関する事柄が含まれます。

EUC コード・セットを必要とするアプリケーションを開発する場合は、 *管理 API 解説書* を参照してください。

データベースおよびクライアント・アプリケーションがグラフィック・データ (つまり、2 バイト文字) をサポートするには、日本語と中国語 (繁体字) の EUC コード・ページの両方にある多数の文字を処理するときに、 2 バイト幅の制約事項を克服する必要があります。それらの EUC コード・ページのグラフィック・データの保管および操作には、UCS-2 コード・セットを使用します。

ストアード・プロシージャー

データベース・アプリケーション環境では、操作が繰り返し行われることがよくあります。たとえば、一定のデータを受信する、データベースに対して複数の同じ要求を実行する、または一定のデータを戻すなどです。ストアード・プロシージャーでは、リモート・データベースに呼び出しを一度出すだけで、事前プログラムされたプロシージャーを実行することができます。 1 回の呼び出しで、データベースに複数回アクセスできることになります。

リモート・データベースのための単一 SQL ステートメントを処理するには、 2 回の送信 (1 回の要求と 1 回の受信) が必要になります。しかし、1 つのアプリケーションにたくさんの SQL ステートメントが含まれている場合があります。ストアード・プロシージャーを使わないとすれば、アプリケーションでの作業を完了するために送信が何回も必要になってしまいます。

データベース・クライアントでストアード・プロシージャーを使用するなら、プロセス全体で送信は 2 回だけなので、ネットワーク送信の回数が少なくなります。ストアード・プロシージャーを呼び出すには、要求側のアプリケーションが、そのプロシージャーの含まれているデータベースに接続されていることが必要です。

通常、これらのストアード・プロシージャーはデータベース・エージェントとは別のプロセスで実行されます。このように分けているため、ストアード・プロシージャーとエージェント・プロセスとはルーターを介して通信する必要があります。ストアード・プロシージャーで可能なかぎり最高のパフォーマンスを得るには、ストアード・プロシージャーを「承認された」、または「分離していない」として指定し、その結果として、このプロシージャーを直接データベース・エージェント・プロセスで実行させることができます。「承認された」および「分離していない」とはどういう意味でしょうか?

- *分離していない* とは、ストアード・プロシージャーと、データベース・エージェントが使用するデータベース制御構造の間を分離するものが何もないということです。
- *承認された* は、偶然であれ故意であれ、データベース制御構造がこのストアード・プロシージャーにより壊されることはないと管理者が確信していることを表します。つまり、データベースの保全性を危うくしないような方法で、それらが動作するということを信頼しているということです。

これらの用語は両方とも、同じことを意味します。つまり、ストアード・プロシージャーが「分離していない」場合は、ストアード・プロシージャーは「承認された」状態になります。データベースを壊す危険性を考えると、最高のパフォーマンスを得る必要がある場合は、分離していないストアード・プロシージャー**だけを使用するようにしてください**。さらに、分離していないストアード・プロシージャーとして実行するためには、プロシージャーが正しくコーディングされ、完全にテストされていることを確認する必要があります。これらの、分離していないストアード・プロシージャーのいずれかを実行中に致命的エラーが発生した場合は、データベース・マネージャーは、このエラーがアプリケーションで発生したのかデータベース・マネージャーのコードで発生したのかを調べ、適切なリカバリー処置を実行します。

分離していないストアード・プロシージャーがデータベース・マネージャーを壊してリカバリー不能にし、結果としてデータの消失やデータベースの破壊を招く可能性もあります。承認されたストアード・プロシージャーを実行する場合には、細心の注意を払う必要があります。たいていの場合、アプリケーションのパフォーマンス分析を適切に実行すれば、このオプションを使用せずに望ましいパフォーマンスを得ることができます。たとえば、トリガーを使用するとパフォーマンスが改善される場合があります。

ストアード・プロシージャーを分離していないとして作成するためには、次の 2 つの方法があります。

- CREATE PROCEDURE コマンドを使用し、NOT FENCED 文節を指定する。
- ご使用のプラットフォームの*概説およびインストール* で定義されている特殊なディレクトリーにプロシージャーを入れる。 (この方法は、Java ストアード・プロシージャーには有効ではありません。)

ストアード・プロシージャーを実行するためには、プロシージャーを呼び出すアプリケーションを実行するエンド・ユーザーには、以下のいずれかの特権が実行時に必要です。

- ストアード・プロシージャーに関連するパッケージについての EXECUTE または CONTROL 特権
- SYSADM または DBADM 権限

ストアード・プロシージャーを使用するプログラムの作成の説明については、 *アプリケーション開発の手引き* を参照してください。

データベースの活動化

データベースを始動すると、いくつかの種類のデータはキャッシュに入れられます。たとえば、データ・バッファーはバッファー・プールでキャッシュに入れられ、パッケージおよび動的 SQL ステートメントはパッケージ・キャッシュでキャッシュに入れられます。

データベースにユーザーが誰も接続していない短い時間が頻繁に発生し、さらにこの時間と少数のユーザーしかデータベースに接続していない時間とが散らばって発生する場合は、キャッシュの内容が頻繁に壊されるため、キャッシュを利用する利点は失われます。この状態を避けるためには、次のコマンドを出してデータベースを活動化することを考慮してください。

```
DB2 ACTIVATE DATABASE database
```

このコマンドは指定したデータベースを活動化して必要なすべてのサービスを開始するため、データベースは任意のアプリケーションから接続され、使用できるようになります。 ACTIVATE DATABASE により初期設定したデータベースは、 DEACTIVATE DATABASE または *db2stop* によりシャットダウンできます。これらのコマンドの詳細については、 *コマンド解説書* を参照してください。

アプリケーションの並列処理

DB2 がサポートする並列環境の 1 つのタイプは、対称マルチプロセッサー (SMP) マシンを必要とする環境です。この環境では、複数のプロセッサーがデータベースへのアクセスを共用します。これにより、複雑な SQL 要求は複数のプロセッサーに分割され、並列に実行できるようになります。

アプリケーションのコンパイル時に CURRENT DEGREE 特殊レジスターまたは DEGREE バインド・オプションを使用することで、実行する並列化の度合い (並列度) を指定できます。「度合い」は、単に、並行して実行される照会の部分の数を指します。プロセッサーの数と並列化の度合いに選択する値との間には、厳密な関係はありません。ハードウェア・プラットフォームで使用可能な総プロセッサー数が、アプリケーションの実行中に要求されるわけではありません。つまり、選択する値はこの数より多くても少なくてもかまいません。

並列度が大きくなれば、システム・メモリーと CPU のオーバーヘッドは増えます。

並列化を活用する際には、パフォーマンスを最適化するために一部の構成パラメーターを修正する必要があることに注意してください。並列度が高い環境では、共用メモリー量と事前取り出しを制御する構成パラメーターを見直し、必要に応じて修正する必要があります。 並列操作と区分データベース環境に関するパラメーターのリストについては、 466ページの『区画データベース』を参照してください。

区画内並列処理の制御や管理に使用する構成パラメーターは、3 つあります。 *intra_parallel* データベース・マネージャー構成パラメーターは、並列サポートをオンにしたりオフにしたりします。 *max_querydegree* データベース・マネージャー構成パラメーターは、データベースでの照会における並列度の上限を設定します。この値は、CURRENT DEGREE 特殊レジスターおよび DEGREE バインド・オプションを上書きします。 3 番目の構成パラメーターは、*dft_degree* データベース構成パラメーターです。これは、CURRENT DEGREE 特殊レジスターおよび DEGREE バインド・オプションのデフォルト値を設定します。

2 つ以上の並列度を使用するアプリケーションの用途と意味の詳細については、 *アプリケーション開発の手引き* を参照してください。

照会で DEGREE = ANY を指定して実行すると、データベース・マネージャーによって区画内の並列度が選ばれます。この並列度は、プロセッサーの数や照会の特性などを示すいくつかの係数に基づいて選ばれます。ですから、係数の値によっては、実際に実行時に使用される度合いの値がプロセッサーの数よりも少ない場合があります。

並列度は、ステートメントのコンパイル時に SQL 最適化プログラムによって判別され、データベース活動に応じて、照会が実行される前に調整されます。システムがよく使用されている場合には、並列度は SQL 最適化プログラムで選ばれた値よりも小さい

可能性があります。なぜなら、区画内並列処理は照会にかかる時間を節約するためにシステム・リソースを酷使するからです。このままでは、他のデータベース・ユーザーのパフォーマンスも低下してしまいます。

SQL 最適化プログラムによって選ばれた並列度は、 SQL Explain 機能を使用してアクセス・プランを表示することによって検出できます。また、実行時での並列度は、データベースのシステム・モニターを使って検出できます。 SQL Explain 機能と関連ツールの詳細については、 211ページの『第7章 SQL Explain 機能』と 573ページの『付録C. SQL EXPLAIN ツール』を参照してください。 さらに詳細については、システム・モニター *手引きおよび解説書* を参照してください。

注: 並列度は、ハードウェア環境とは関係なく設定できます。つまり、SMP マシンがなくても、並列度を使用できます。たとえば、単一プロセッサー・マシンで ″入出力制約″ の照会を実行する場合でも、 ″2″ 以上の並列度を指定しておいた方が有利です。この場合、単一プロセッサーは入力または出力タスクの完了を待たずに、次の照会を処理できます。並列度 ″2″ 以上を宣言しても、単一プロセッサー・マシン上の入出力並列化は制御しません。 Load などのユーティリティーでは、このような宣言とは関係なく、入出力の並列化を制御することができます。キーワード `ANY` は、*dft_degree* を変更するときにも使用できます。 `ANY` を使用することは、最適化プログラムが区画内並列処理度を判別することを意味します。

多くの場合、並列実行を調整するためにデータベース・エージェント が使用されます。詳細および各種のデータベース・マネージャー構成パラメーターのリストについては、269ページの『データベース・エージェント』を参照してください。

第4章 環境についての考慮事項

アプリケーションを設計しコーディングするときに考慮すべき要因 (43ページの『第3章 アプリケーションについての考慮事項』を参照) のほかにも、アプリケーションのために選択したアクセス・プランに影響を及ぼす可能性のある環境による要因がいくつかあります。

- 照会最適化に影響する構成パラメーター
- 照会最適化に対するノードグループの影響
- 照会最適化に対する表スペースの影響
- 照会最適化に対する索引付けの影響
- 連合データベース照会に影響を与えるサーバー・オプション

SQL 最適化プログラムに影響を与える要因については、 113ページの『第5章 システム・カタログ統計』を参照してください。

アプリケーションと環境のチューニングを行う場合は、上記の分野のどれに変更を加えた場合であっても、その後にアプリケーションの再バインドを行うようにしてください。 こうすると、最適なアクセス・プランを使用できます。

照会最適化に影響する構成パラメーター

構成パラメーターの中には、 SQL コンパイラーによって選択されるアクセス・プランに影響を与えるものがいくつかあります。それらのパラメーターの多くは単一区画データベースに適用されるものですが、一部には区分データベースのみに適用されるものもあります。区分データベースにおいて構成パラメーターを用いて処理を行う場合には、各パラメーターに使用する値をすべての区画で同じにすることをお勧めします。

連合システムで作業している場合、照会の大部分がニックネームにアクセスするときは、環境を変更する前に、送信している照会のタイプを考慮してください。たとえば、バッファー・プールはデータ・ソースからページをキャッシュしません。したがって、*buffpage* パラメーター値を増やしても、ニックネームを含む照会のアクセス・プランを作成しているとき、追加の代替プランが最適化プログラムによって考慮されることが保証されるわけではありません。 (データ・ソースとは、連合システム内の DBMS とデータのことです。) また、最適化プログラムは、データ・ソース表をローカルに具体化することが、最もコストを低くする手段、または分類操作に必要なステップであると判断することがあります。この場合、DB2 ユニバーサル・データベースが利用できるリソースを増やすと、パフォーマンスが高速になります。詳細については、105ページの『連合データベース照会に影響を与えるサーバー・オプション』および 347ページの『データベース共用メモリー』を参照してください。

以下に、SQL コンパイラーによって選択されるアクセス・プランに影響を与える構成パラメーターのリストを示します。

- 348ページの『バッファー・プール・サイズ (buffpage)』

  アクセス・プランを選択するとき、最適化プログラムはディスクからページをバッファー・プールに取り出すときの入出力コストを考慮します。その計算の中で、最適化プログラムは照会に必要な入出力の数を見積もります。この見積もりには、バッファー・プール使用率の予測も含まれています。すでにバッファー・プールの中にあるページに含まれている行を読み取るには、追加の物理的な入出力が必要ではないためです。最適化プログラムは、ページがバッファー・プールの中にあるかどうかを見積もるときに、 BUFFERPOOLS システム・カタログ表の *npages* 列の値を考慮します。

  表を読み取るときの入出力コストは、以下のものに影響を与える可能性があります。
  - 2 つの表の結合方法 (176ページの『外部か内部かの判別』を参照)
  - クラスター化されていない索引を使ってデータを読み取るかどうか (168ページの『クラスター索引』を参照)

  1 つのデータベースには複数のバッファー・プールを入れることができます。また、1 つの区分データベースに複数のバッファー・プールを入れることも可能です。新しいバッファー・プールは、データベースの区画にそれぞれ追加することも、すべての区画にまたがって追加することも選択できます。最適化プログラムによる区分データベースでの見積もりには、システム・カタログ表 BUFFERPOOLS および BUFFERPOOLSNODE の *npages* 列が使用されます。

- 449ページの『デフォルトの程度 (dft_degree)』

  dft_degree 構成パラメーターは、 CURRENT DEGREE 特殊レジスターおよび DEGREE バインド・オプションのデフォルト値を指定します。値 1 は、区画内並列処理でないことを意味しています。値 -1 は、プロセッサー数と照会のタイプに基づいて、区画内並列処理の程度を最適化プログラムが決定することを意味します。

- 449ページの『デフォルトの照会最適化クラス (dft_queryopt)』

  SQL 照会をコンパイルするときには、照会最適化クラスを使用して、最適化プログラムにどの程度の最適化技法を使用するのかを指示することができます。適切な照会最適化クラスの選択に関する詳細については、 68ページの『最適化クラスの調整』を参照してください。

- 401ページの『活動アプリケーションの平均数 (avg_appls)』

  avg_appls パラメーターは、 SQL 最適化プログラムが、選択したアクセス・プランの実行時にどれだけのバッファー・プールが使用可能になるかを見積もるために使われます。 このパラメーターに高い値を指定すると、最適化プログラムがそのバッファー・プールを控えめに使用するようなアクセス・プランを選択する影響が出る可能性があります。このパラメーターの値を 1 にすると、最適化プログラムはバッファー・プール全体をアプリケーションに利用可能なものとして処理します。

- 363ページの『分類ヒープ・サイズ (sortheap)』

最終的に分類されたデータ・リストを保管するために一時表が必要ではない場合は、分類は「パイプ処理」されるものと見なされます。つまり、分類の結果を 1 つの順次アクセスで読み取ることができます。パイプ処理された分類は、パイプ処理ではない分類の場合よりもパフォーマンスが向上するので、可能ならそれが使用されます。(パイプ式分類と非パイプ式分類の定義については、 189ページの『最適化プログラムでの分類の影響』を参照してください。)

アクセス・プランを選択するとき、最適化プログラムは分類操作のコストを見積もります。それには、分類が次のものによってパイプ処理可能かどうかの評価も含まれます。

- 分類するデータの量を見積もる。
- *sortheap* パラメーターを見て、分類のパイプ処理のために十分なスペースがあるかどうかを調べる。

- 357ページの『ロック・リスト用最大ストレージ (locklist)』 および 389ページの『自動調整前のロック・リストの最大パーセント (maxlocks)』

  使用する分離レベル (43ページの『並行性』を参照) が**反復可能読み取り (RR)** である場合、 SQL 最適化プログラムは *locklist* パラメーターと *maxlocks* パラメーターの値を考慮して、行レベルのロックが表レベルのロックに自動調整される可能性があるかどうかを調べます。 最適化プログラムは、表アクセスに関してロック自動調整が起きると予測した場合、照会実行時のロック自動調整のオーバーヘッドが生じないよう、そのアクセス・プランには表レベル・ロックを選択します。

- 481ページの『CPU 速度 (cpuspeed)』

  SQL 最適化プログラムが特定の操作を実行するコストを見積もるときには、 CPU の速度が使用されます。最適化プログラムは、各種入出力コスト見積もりとともにそれらの CPU コスト見積もりを使用して、照会に最適のアクセス・プランを選択します。

  マシンの CPU 速度は、選択されるアクセス・プランに重大な影響を与える可能性があります。この構成パラメーターは、データベースをインストールまたは移行した時点で、自動的に適切な値に設定されます。このパラメーターを調整するのは、テスト・システムにおいて実動環境のモデル化を行っている場合か、またはハードウェア変更の影響を見積もる場合**だけに**してください。このパラメーターを使用して異なるハードウェア環境のモデル化を行う場合は、その環境のために選択されるアクセス・プランを観察することができます。

- 366ページの『ステートメント・ヒープ・サイズ (stmtheap)』

  ステートメント・ヒープのサイズは、最適化プログラムがさまざまなアクセス・パスを選択する際には影響がありません。しかし、複合 SQL ステートメントに関して実行される最適化の量には影響します。

  stmtheap パラメーターの設定値が十分大きくない場合は、使用可能なメモリーが足りないのでステートメントを処理できないことを示す SQL 警告を受け取ることがあります。たとえば、 SQLCODE +437 (SQLSTATE 01602) により、ステートメントのコ

ンパイルに使った最適化の量が、照会最適化クラスの指定時に要求した量より少ないことを示す場合があります。 (詳しくは、68ページの『最適化クラスの調整』を参照してください。)

- 473ページの『照会の最大並列処理度 (max_querydegree)』

  このパラメーターの値が ″ANY″ のときには、最適化プログラムが、使用する並列化の程度 (並列度) を選択します。 ″ANY″ 以外の値が示されている場合には、ユーザー指定の値を使用して、アプリケーションの並列度を決定します。

- 480ページの『通信帯域幅 (comm_bandwidth)』

  通信帯域幅は、最適化プログラムがアクセス・プランを決めるのに使用されます。最適化プログラムはこのパラメーターの値を使用して、区分データベースのデータベース区画サーバー間で一定の操作を実行する際のコストを見積もります。

詳細については、332ページの『構成パラメーターの調整』を参照してください。

照会最適化に対するノードグループの影響

区分データベースの場合、最適化プログラムによって表の併置が認識され、それが照会の最適なアクセス・プランを判別する際に使用されます。結合照会に頻繁に関係する表が区分データベースの複数の区画にまたがって分割されている場合、結合される各表にある行が同じデータベース区画上にあることが前提になります。結合を実行するときに、結合される両方の表のデータが同じ区画に入れられていると、データをある区画から別の区画に移す必要がなくなります。同じノードグループに両方の表を置くと、それらの表のデータは同じ区画に入れられます。

表の併置に関する詳細については、*管理の手引き: 計画* を参照してください。

また、区分データベース内で表のサイズに応じて、データをより多くの区画に配分すると、照会の実行にかかる見積時間 (つまりコスト) が減少します。表の数、表のサイズ、それらの表のデータがある場所、および照会のタイプ (上述した結合が必要かどうか) はすべて照会のコストに影響を与えます。

照会最適化に対する表スペースの影響

表スペースの特性のうちのあるものは、 SQL コンパイラーによって選択されるアクセス・プランに影響を与える可能性があります。

- コンテナー特性

  コンテナー特性は、照会の実行時に関連付けられた入出力のコストに重大な影響を与える可能性があります。アクセス・プランを選択するとき、SQL 最適化プログラムは、異なる複数の表スペースのデータにアクセスする場合のコストの相違も含めて、それらの入出力コストを考慮に入れます。最適化プログラムが表スペースのデータにアクセスするための入出力コストを見積もるときに、 SYSCAT.TABLESPACES システム・カタログの 2 つの列が使用されます。

– OVERHEAD。データがメモリーに読み込まれるまでにコンテナーで必要な時間の見積値 (ミリ秒)。このオーバーヘッド活動には、ディスク待ち時間以外にも、コンテナーの入出力コントローラーのオーバーヘッド (ディスクのシーク時間を含む) が含まれます。

  以下の式を使って、オーバーヘッドのコストを見積もることができます。

  ```
  OVERHEAD = ミリ秒単位の平均シーク時間
             + (0.5 * 回転待ち時間)
  ```

  - 0.5 は半回転した場合の平均オーバーヘッドを示します。
  - 1 回転ごとの回転待ち時間はミリ秒単位で以下のように計算されます。

    ```
    (1 / RPM) * 60 * 1000
    ```

    ここで、
    - 1 分当たりの回転数で除算し、1 回転当たりの分数を求めます。
    - 60 (1 分間の秒数) で乗算します。
    - 1000 (1 秒間のミリ秒数) で乗算します。

  たとえば、ディスクの 1 分当たりの回転数が 7 200 であるとします。この場合、回転待ち時間は次の式のようになります。

  ```
  (1 / 7200) * 60 * 1000 = 8.328 ミリ秒
  ```

  この値は、想定される 11 ミリ秒の平均シーク時間と共に、次のように OVERHEAD 見積もりの計算に使用することができます。

  ```
  OVERHEAD = 11 + (0.5 * 8.328)
           = 15.164
  ```

  見積もられた OVERHEAD 値が約 15 ミリ秒となります。

– TRANSFERRATE。1 ページのデータをメモリーに読み込むのに必要な時間の見積値 (ミリ秒)。

  それぞれの表スペース・コンテナーが単一の物理ディスクである場合は、以下の式を使って、転送コストを 1 ページ当たりのミリ秒数で見積もることができます。

  ```
  TRANSFERRATE = (1 / spec_rate) * 1000 / 1 024 000 * page_size
  ```

  - spec_rate は、転送速度に関するディスクの仕様を、 1 秒当たりの MB 数で表します。
  - spec_rate で除算し、MB 当たりの秒数を求めます。
  - 1000 (1 秒間のミリ秒数) で乗算します。
  - MB 当たり 1 024 000 バイトで除算します。
  - ページ・サイズ (バイト単位) で乗算します (たとえば、4 KB のページの場合は 4 096 バイト)。

  たとえば、ディスクの指定率が 1 秒当たり 3 MB であるとします。この場合、次の計算が行われます。

```
TRANSFERRATE = (1 / 3) * 1000 / 1024000 * 4096
             = 1.333248
```

見積もられた TRANSFERRATE 値は、ページ当たり約 1.3 ミリ秒になります。

表スペース・コンテナーが単一の物理ディスクではなく、 (RAID などの) ディスク・アレイである場合、使用する TRANSFERRATE を決定しようとするときに追加の考慮事項があります。アレイが比較的小さい場合は、障害はディスク・レベルにあると想定して、 spec_rate にディスクの数を乗算することができます。しかし、コンテナーを構成しているアレイ内のディスクの数が多ければ、障害はディスク・レベルではなく、ディスク・コントローラー、入出力バス、またはシステム・バスなどのその他の入出力サブシステム構成装置の 1 つに存在する可能性もあります。この場合、入出力スループット能力は、 spec_rate とディスクの数の積であるとは想定できません。順次走査中に、実際の入出力率 (MB 単位) を測定する必要があります。たとえば、順次走査は select count(*) from big_table となり、サイズは MB 単位となります。この数を、big_table が存在している表スペースを構成するコンテナーの数で除算します。上記の式の spec_rate をこの計算結果で置き換えます。たとえば、 4 つのコンテナー表スペースの中の表を走査している間に測定された 100 MB の順次入出力率は、コンテナー当たり 25 MB となるか、 TRANSFERRATE がページ当たり (1/25) * 1000 / 1024000 * 4096 = 0.16 ミリ秒となります。

表スペースに割り当てられたコンテナーは、それぞれ異なる物理ディスク上に存在しています。最適の結果を得るためには、所定の表スペースに使用されるすべての物理ディスクの OVERHEAD および TRANSFERRATE の特性が同じでなければなりません。これらの特性が同じでない場合には、 OVERHEAD および TRANSFERRATE の値を設定するときには平均値を使用する必要があります。

これらの列のメディア特有の値は、ハードウェア仕様から、または実験によって得ることができます。これらの値は、CREATE TABLESPACE および ALTER TABLESPACE ステートメントで指定できます。

コンテナーとしてディスク・アレイを使用している上記のような環境では、実験は特に重要です。データを移動させる単純な照会を作成して、その照会をプラットフォーム固有の測定ユーティリティーと一緒に使用する必要があります。その後、表スペース内の異なるコンテナー構成についてその照会を再実行します。 CREATE および ALTER TABLESPACE ステートメントを使用して、現在の環境でデータが転送された方法を変更することができます。

これら 2 つの値による入出力コスト情報は、いくつかの方法で最適化プログラムに影響を与える可能性があります。データにアクセスするのに索引を使用するかどうか、または 1 つの結合の中で内部と外部にどの表を選択するか、などがこれに含まれます。

- 事前取り出し

表スペースのデータにアクセスするための入出力コストを考慮するとき、最適化プログラムは、ディスクからデータおよび索引ページを事前取り出しすることによって、照会のパフォーマンスに与える潜在的な影響も考慮します。データおよび索引ページの事前取り出しを行うと、データをバッファー・プールに読み込むことに関連するオーバーヘッドと待ち時間が少なくなります。 詳細については、253ページの『バッファー・プールへのデータの事前取り出し』を参照してください。

最適化プログラムは SYSCAT.TABLESPACES の PREFETCHSIZE 列および EXTENTSIZE 列の情報を使用して、表スペースに関して生じる取り出しの量を見積もります。

- EXTENTSIZE を設定できるのは、(たとえば CREATE TABLESPACE ステートメントを使って) 表スペースを作成するときだけです。デフォルトのエクステント・サイズは 32 ページ (それぞれが 4 KB) で、普通はこれで十分です。
- PREFETCHSIZE は、表スペースを作成するときと、さらに ALTER TABLESPACE ステートメントを使用するときに設定できます。デフォルトの事前取り出しサイズは DFT_PREFETCH_SZ データベース構成パラメーターの値によって決まります。このパラメーターはオペレーティング・システムによって異なります。 396ページの『デフォルト事前取り出しサイズ (dft_prefetch_sz)』を参照して、このパラメーターの推奨サイズを検討し、データの移動を改善するために必要な変更を行う必要があります。

次に、RESOURCE 表スペースの特性を変更するための構文の例を示します。

```
ALTER TABLESPACE RESOURCE
  PREFETCHSIZE 64
  OVERHEAD     19.3
  TRANSFERRATE 0.9
```

表スペースに対して変更を行った後は、アプリケーションの再バインドを考慮して、最適なアクセス・プランが使用されるように、 RUNSTATS ユーティリティーを用いて索引に関する最新の統計を収集する必要があります。

照会最適化に対する索引付けの影響

索引をいつ使用するかを自分で決定することはない、という点は重要です。最適化プログラムが、使用可能な表および索引の情報に基づいて決定を行います。しかし、パフォーマンスを向上できるものとして必要な索引を作成するなら、そのプロセスにおいて重要な役割を果たすことになります。また、索引を作成した後、索引についての統計を収集する (RUNSTATS ユーティリティーを使用) か、事前取り出しサイズを変更し、状況に応じてその統計を最新のものに保つことも重要です。つまり、管理者は、作成可能な索引の種類、およびそれらを作成する方法を理解していなければなりません。

索引付きと索引なし

データベース照会に関係するどの列にも索引が存在していない場合、その照会において参照される各表ごとに、表走査を実行しなければなりません。表が大きいほど、表走査

にかかる時間は長くなります。*表走査* が行われるのは、データベース・マネージャーが表の各行に順次アクセスするときです。これは、データベース・マネージャーが索引を使用してデータにアクセスするときに行われる*索引走査* と比較することができます。 (161ページの『索引走査の概念』を参照。)

索引列が SELECT ステートメント内で参照され、最適化プログラムが表走査よりも索引走査を使用したほうが速くなると見積もった場合は、索引を使うよう選択します。特に表が大きくなるにつれ、索引ファイルは一般にファイル全体に比べて小さくなり、読み取りのための時間が少なくなります。さらに、索引全体を走査することが必要になることはまずありません。索引に述部を適用すると、データ・ページから読み取られる行数が減ります。

各索引項目は、検索キー値と、その値が入った行を指すポインターで構成されています。値を逆方向に検索できるのは、 CREATE INDEX ステートメントで ALLOW REVERSE SCANS パラメーターが指定されている場合だけです。したがって、適切な述部を使用して、検索を一まとめにすることができます。索引を使ってオーダー順序で行を入手することにより、データベース・マネージャーが行を表から読み取ってからその行を分類する必要がなくなります。 ALLOW REVERSE SCANS を指定すると、順方向および逆方向に行を順次に直接読み取るよう索引を使用できます。詳細については、*SQL 解説書* を参照してください。

固有索引には、組み込み列と検索キー値および行ポインターが含まれます。

注: ユーザーは、最適化プログラムが索引を選択して、データベース・マネージャーが使用するかどうかについては制御することができません。たとえば、照会中の表に索引があるというだけで、照会結果がオーダー順序で作成されるという保証はありません。データベース・マネージャーは照会時に、最適化プログラムによってこの索引を選択して、それを使用することはできますが、索引の使用が必須ではありません。 ORDER BY 文節を使用することによってのみ、結果セットを順序づけることが 『できます』。

索引により、アクセス時間を大幅に短縮することができます。しかし、索引は、パフォーマンスに悪い影響を与えることもあります。索引を作成する前に、複数の索引を作成することが、ディスク・スペースや処理時間に対してどんな影響を与えるかを考慮してください。

- 各索引は、ストレージまたはディスク・スペース上で特定の容量を占めます。正確な量は、表のサイズと索引に含まれる列の数およびサイズによって異なります。
- 表に対して実行される INSERT または DELETE の各操作では、その表の各索引を更新することもさらに必要になります。さらに、索引キーを変更する UPDATE 操作についても同じことが言えます。
- LOAD ユーティリティーでは既存の索引が再作成されたり、既存の索引に索引が追加されます。
- 索引が作成された時に使用される索引 PCTFREE を指定変更するために、 LOAD コマンドで `indexfreespace` MODIFIED BY パラメーターを指定できます。

- 各索引は照会の代替アクセス・パスを潜在的に追加するものとなり、最適化プログラムでそれを考慮するため、照会のコンパイル時間が長くなります。

索引は、アプリケーション・プログラムの要件に合わせて慎重に選択する必要があります。

特定のパッケージで索引を使用するかどうかを決めるには、 SQL Explain 機能を使用できます (211ページの『第7章 SQL Explain 機能』を参照)。

インデックス・アドバイザーの使用

DB2 インデックス・アドバイザーは、表データにとって最適の索引のセットを選択するのに役立ちます。このツールは、次のような複数の方法で呼び出すことができます。

- コントロール・センターで「索引 (Indexes)」フォルダーを選択し、マウス・ボタン 2 をクリックし、**「作成 (Create)」—>「索引-ウィザード使用 (Index using wizard)」**を選択します。
- このツールは、コマンド行から `db2advis` と入力してアクセスできます。

DB2 インデックス・アドバイザーの詳細については、230ページの『SQL アドバイス機能』で説明されています。

より大きな索引キーの使用

255 バイトよりも長い列を索引キーの一部として指定できます。DB2_INDEX_2BYTEVARLEN レジストリー変数を 1 バイトでなく 2 バイトにして、索引キーの長さを保管できます。

レジストリー変数を変更すると、複数の SQL ステートメントが影響を受けます。それらは、以下のとおりです。

- CREATE TABLE。可変キー部分を持つ 1 次キー、外部キー、および固有キーは、255 バイトよりも大きなサイズにすることができます。
- CREATE INDEX。可変キー部分を持つすべての索引 (固有索引および組み込み列を含む) は、 255 バイトより大きなサイズにすることができます。
- ALTER TABLE。可変キー部分を持つ 1 次キー、外部キー、および固有キーは、 255 バイトよりも大きなサイズにすることができます。可変キー部分を持つすべての索引 (固有索引および組み込み列を含む) は、 255 バイトより大きなサイズにすることができます。

  レジストリー変数の値に関係なく、255 バイトの外部キー制限はなくなります。外部キーに対応する 1 次キーの真理条件では、制限が強制されます。

より大きな索引キーを使用するために既存の索引を変換するには、索引をドロップして、DB2_INDEX_2BYTEVARLEN レジストリー変数をオンに設定し、索引を (より大きな列を使用して) 作成します。

構文記述を含め、SQL ステートメントについて詳しくは、 *SQL 解説書* を参照してください。

索引付けの指針

どんな索引を作成するかは、データおよびその用途によって異なります。どの索引が最も有用かを判断する上で、以下に示す指針が役立つでしょう。

- 該当する場合は、CREATE UNIQUE INDEX ステートメントを使用して、基本キーと固有キーを定義します。 (詳細については、*SQL 解説書* を参照してください。) 固有索引は、最適化プログラムが分類などの特定の操作を実行しないようにするのに役立ちます。
- データ検索のパフォーマンスを向上するために列を組み込んでいる固有索引を定義します。 列は、次の場合に、固有索引の INCLUDE 列のふさわしい候補です。
  - 頻繁にアクセスされるので、索引のみのアクセスによって益が生じる場合
  - 索引走査の範囲を限定するために必要でない場合
  - 索引キーの順序または固有性に影響を与えない場合

  INCLUDE 列の詳細については、 *管理の手引き: 計画* の『索引、索引の拡張、または索引の指定の作成』という章を参照してください。
- 索引を使用することによって、データ・ページが 2、3 ページより多い表への頻繁な照会を最適化します。このことは、 SYSCAT.TABLES カタログ視点の NPAGES 列に記録されます。次のことを行う必要があります。
  - 表を結合するときに使用するすべての列に索引を作成します。
  - 通常、特定の値を検索する対象となる列に索引を作成します。
- キーの配列を昇順にするか降順にするかを、どちらの順序が主に使用されるかまたは要求されるかに基づいて決定します。値を逆方向に検索できるのは、 CREATE INDEX ステートメントで ALLOW REVERSE SCANS パラメーターが指定されている場合だけです。索引は順方向と逆方向の両方向に走査できますが、索引の順方向走査 (つまり、索引の作成時に指定された順) の方が、索引の逆方向走査よりもパフォーマンスは多少よくなります。詳細については、*SQL 解説書* を参照してください。
- 列上の他の索引キーの部分キーとなるような索引は作成しないようにします。たとえば、列 a、b、および c に対して 1 つの索引がある場合、列 a と b についての 2 番目の索引は通常有用ではありません。
- 外部キーについての索引を使用して、親表での削除操作と更新操作のパフォーマンスを向上させます。
- データ分類時に頻繁に使われる列についての索引を使います。
- 複数列索引を作成するとき、最初のキー列に関して複数の選択の余地がある場合は、『=』 (equijoin) 述部に指定されることの最も多いものを選択するか、またはまず列に個別値の最大個数を指定します。
- すべての列に無制限に索引を作成すると、ディスク・スペースを大量に使うだけでなく、準備時間も長くなります。このことは、複合照会、つまりダイナミック・プログラミングにより結合されて列挙された最適化クラスが使われる照会の場合に特に当てはまります (68ページの『最適化クラスの調整』を参照してください)。

- 表に対して定義する索引の数に関する*経験則* を以下に示します。この数は、データベースの主な用途に基づいています。
  - オンライン・トランザクション処理 (OLTP) 環境では、 1 つまたは 2 つの索引だけにしてください。
  - 照会 (読み取り専用) 環境では、6 個以上の索引を使用できます。
  - 照会 /OLTP の混合環境では、2〜5 個の索引を使用できます。
- 新しく挿入された行がその索引に応じてクラスター化され続けるのに役立つ、クラスター化索引を定義することを考慮してください。クラスター化索引は、表を再編成する必要性を大幅に少なくします。

  注: クラスター化索引が定義されると、データ・ページで挿入が行われるようにするために、フリー・スペースが予約された状態で表が各データ・ページにロードされる必要があります。 (フリー・スペースは、ALTER TABLE ステートメントで PCTFREE キーワードを使用して、または LOAD コマンドの pagefreespace MODIFIED BY 文節を使用して予約します。)
- 索引を作成するときには、PCTFREE キーワードを使用することを考慮してください。 PCTFREE は、索引に対する将来の更新に備えて索引ページにスペースを予約します。このようにすることによって、ページ分割の頻度が下がり、パフォーマンスが向上します。
- 索引の作成時に MINPCTUSED オプションを使用することを考慮します。 MINPCTUSED は、索引葉ページ上で使用される最小スペースの限界値を指定し、オンライン索引再編成を可能にします。これによって、データと索引のオフライン再編成の必要性が少なくなります。

注: 宣言済み一時表では索引はサポートされません。

索引を作成するとパフォーマンスが向上するのは、一般に次のような環境の場合です。

- 最も頻繁に処理される照会およびトランザクションの WHERE 文節の中で使用される列には、索引を作成することができます。

  次のような WHERE 文節の場合、

  ```
  WHERE WORKDEPT='A01' OR WORKDEPT='E21'
  ```

  一般に WORKDEPT に対する索引があれば便利ですが、それらの値が頻繁に発生する場合はそうでもありません。
- 照合順序の中での行の並べ替えにおいては、 1 つまたは複数の列に索引を作成することができます。 ORDER BY 文節においてだけでなく、DISTINCT 文節や GROUP BY 文節など、他の機能でも並べ替えが必要になります。

  以下の例では、DISTINCT 文節を使用しています。

  ```
  SELECT DISTINCT WORKDEPT
       FROM EMPLOYEE
  ```

データベース・マネージャーでは、 WORKDEPT に対して昇順または降順に定義された索引を使用することによって、重複値を削除することができます。以下の GROUP BY 文節の例のようにして、この同じ索引を使用して値をグループ化することもできます。

```
SELECT WORKDEPT, AVERAGE(SALARY)
     FROM EMPLOYEE
GROUP BY WORKDEPT
```

- ステートメントで参照される各列に名前を指定するためにも、索引を作成することができます。この方法で索引を指定すると、その結果、索引のみのアクセスによって、データをより効率的に取り出せるようになります。

  たとえば、以下の SQL ステートメントが発行されたとします。

```
SELECT LASTNAME
     FROM EMPLOYEE
  WHERE WORKDEPT IN ('A00','D11','D21')
```

  EMPLOYEE 表の WORKDEPT 列と LASTNAME 列に索引が定義されている場合、表全体を走査するよりも索引を走査する方が、ステートメントをより効率的に処理することができます。述部は WORKDEPT についてであるため、この列は索引の最初の列でなければなりません。

- 索引の組み込み列は、表での索引の使用を改善する別の方法です。 前の例を使用し、次のようにして固有索引を定義できます。

```
CREATE UNIQUE INDEX x ON employee (workdept) INCLUDE (lastname)
```

  lastname を索引キーの一部としてではなく、組み込み列として指定することは、lastname が索引の葉ページでのみ保管されるということを意味します。

索引の管理に関するパフォーマンスについてのヒント

以下は、索引を正しく使用、管理することがパフォーマンスにどのように影響するかを理解する上で役に立つ情報です。

1. **索引の作成**

   大きな表に索引を作成する場合、SMP マシンを使用しているならば、 *intra_parallel* を YES (1) か SYSTEM (-1) に設定して、並列パフォーマンスの機能を利用することを考慮してください。

   複数のプロセッサーを使用して、データの走査や分類を行うことができます。索引の作成時に複数のプロセッサーを有していても特に利点がないのは、 *indexsort* データベース構成パラメーターが NO の場合だけです (このパラメーターのデフォルトは YES です)。このパラメーターは、索引の作成時に索引キーの分類を行うかどうかを制御します。

2. **索引表スペース**

索引は、他の表データを保管するのに使用するのとは別の表スペースに保管することができます。これにより、読み書きヘッドの移動が少なくなり、ディスク装置をさらに効率的に使用できるようになります。索引表スペースを、高速な物理装置に保管されるように作成することもできます。

また、表スペースに別のバッファー・プールを割り当てて、大量のデータ・ページによりバッファーが満杯になって索引ページが押し出されることのないようにすることもできます。

索引が別々の表スペースに置かれていないと、データと索引ページの両方が同じエクステント・サイズと事前取り出し数量を使用します。索引用に別の表スペースを使用する場合は、表スペースのすべての特性に別の値を選択するオプションがあります。索引は一般に表より小さく、いくつかのコンテナーに分散しているため、 8 や 16 のような小さいエクステント・サイズを選ぶのが普通です。詳細については、170ページの『索引ページの事前取り出し』を参照してください。 SQL 最適化プログラムでは、表スペースに高速の装置を使用することが考慮されます (94ページの『照会最適化に対する表スペースの影響』を参照)。表スペースの詳細については、*管理の手引き: 計画* を参照してください。

3. **クラスター化の程度**

   SQL ステートメントで並べ替え (たとえば、ORDER BY、GROUP BY、DISTINCT) が必要であり、さらにその並べ替えを行うのに適した索引がある場合、データベース・マネージャーが索引を*選択しない* 可能性があります。こういうことが起きるのは、次のような場合です。
   - 索引クラスター化が不十分である場合 (SYSCAT.INDEXES の CLUSTERRATIO 列および CLUSTERFACTOR 列を参照)
   - 表が小さいので、表の走査や応答セットの分類をメモリーで行う方が低コストになるような場合
   - 表へのアクセスを競合する索引がある場合

   クラスター化索引を作成した後で、ユーザーが REORG、つまり分類、および LOAD を実行することが勧められています。ただし、一般に 1 つの索引では 1 つの表しかクラスター化できないことに注意してください。表と索引は、その表に対するクラスター化索引の順序で構築する必要があります。 RUNSTATS ユーティリティーによって集められた CLUSTERRATIO または CLUSTERFACTOR 統計を向上させて、クラスター化索引はデータの特定順序を維持しようと試みます。

   表のロードまたは再編成を行う前にその表を変更するときは、 PCTFREE を使用することも考慮するべきです。クラスター化を維持するために、各ページは、追加の挿入に備えて使用可能なスペースをそれぞれのデータ・ページに設ける必要があります。このスペースが使用できると、既存のデータで追加の挿入をクラスター化できます。そうすると、追加データのクラスター化のために各ページに、あるパーセンテージのフリー・スペースを残して、データをその表にロードすることを考慮したいことでしょう。これは、最初に表を作成してから、 PCTFREE パラメーターを使用してその表を変更することによって行うことができます。同じように、データを再編成す

る前に、 PCTFREE パラメーターを使用して表を変更することを考慮する必要があります。そうしないと、PCTFREE が設定されていない場合は、再編成によってすべての余分のスペースが除去されてしまいます。

クラスター化は、現状では更新の間維持されません。つまり、クラスター化索引のキー値が変更されるような方法でレコードを更新する場合、レコードはクラスター化順序を維持するために改ページに必ずしも移動されるとは限りません。クラスター化を維持するために、UPDATE を使用する代わりに、 DELETE とそれから INSERT を使用してください。

4. **RUNSTATS ユーティリティー**

   新規索引を作成した後は、RUNSTATS ユーティリティーを使用して索引統計を収集するようにしてください。それらの統計を使うことによって、索引の使用によってアクセス・パフォーマンスが向上するかどうかを最適化プログラムが判断することが可能になります。このトピックの詳細については、114ページの『RUNSTATS ユーティリティーを使用しての統計収集』を参照してください。

5. **索引の再編成**

   索引から最高のパフォーマンスを得るには、索引を周期的に再編成することを考慮してください。表を更新すると、索引ページの事前取り出しの効果が減少します。索引ページの事前取り出しの効果を保持するには、索引を再編成する必要があります。

   索引の再編成には、索引を消去してから再作成するか、または REORG ユーティリティーを使用します。詳細については、264ページの『カタログとユーザー表の再編成』を参照してください。

   頻繁に再編成しなくても済むよう、索引の作成時に PCTFREE を指定することができます。 索引の作成中に PCTFREE パラメーターを指定すると、索引葉ページが作成される時にフリー・スペースが各索引葉ページに残されます。その結果、索引に関連した将来の活動で、索引ページ分割をほとんど生じさせることなくレコードを索引に挿入できます。索引ページ分割が生じると、索引ページは隣接したものとならず、連続したものともなりません。これは、索引ページの事前取り出しを実行する機能が低下する原因になります。索引についてふさわしい PCTFREE を選択することで、索引を再編成する必要をなくしたりその頻度を下げることができます。

   注: 索引の作成時に指定される PCTFREE は、再編成中に索引が再作成されるときに使用されます。

   索引を消去してから再作成すると、ほぼ隣接し連続した一連の新しいページが作成されます。これにより、索引ページの事前取り出しが行われるときに、効果性が改善されます。

   REORG ユーティリティーの実行にはコストがかかりますが、データ・ページを確実にクラスター化します。 このようにしてクラスター化を行うと、大量のデータ・ページにアクセスする索引走査を行う場合に大きな利点があります。

対称マルチプロセッサー (SMP) 環境で処理を行っている場合には、 REORG ユーティリティーは、 *intra_parallel* が YES または ANY のときに複数のプロセッサーを使用します。

6. **EXPLAIN の使用**

   定期的に、使用頻度が最も高い照会に対して EXPLAIN を実行して、それぞれの索引が最低でも 1 回は使用されているかどうかを調べるようにしてください。どの照会でも使用されていない索引がある場合には、その索引の除去を検討する必要があります。

   また、EXPLAIN を使用すると、大きな表の表走査がネスト・ループ結合の内部表として処理されているかどうかを調べることもできます。それはすなわち、結合述部列の索引が欠落しているか、またはその索引が結合述部を適用するのに効果的でないと考えられることを示しています。あるいは、結合述部がないことも考えられます。

7. **揮発性表**

   揮発性 表の特徴として、実行時に表の内容が空であるものから内容が非常に大きなものに至るまでさまざま存在する、という点があります。このタイプの表のアクセス・プランを生成すると、最適化プログラムは (間違って)、内容 (カーディナリティー) がそれぞれ大きく異なる表のアクセスに、索引走査ではなく表走査を優先的に使用します。

   ALTER TABLE...VOLATILE ステートメントを使用して表を「揮発性」として宣言すると、最適化プログラムは揮発性表に対して索引走査を使用できるようになります。最適化プログラムは、統計に関係なく、 (表走査ではなく) 索引走査を使用します。

   - 参照される列がすべて索引内にある場合。
   - 索引が索引走査で述部を適用できる場合。

   表がタイプ付き表である場合、タイプ付き表階層のルート表で、 ALTER TABLE...VOLATILE ステートメントの使用のみがサポートされます。このトピックの詳細については、 *管理の手引き: 計画* または *SQL 解説書* を参照してください。

連合データベース照会に影響を与えるサーバー・オプション

連合システムは、DB2 DBMS (連合データベース) と 1 つまたは複数のデータ・ソースから構成されています。データ・ソースは、CREATE SERVER ステートメントを発行したときに連合システムに識別されます。このステートメントを発行すると、 DB2 および指定したデータ・ソースに関係する連合システム操作のさまざまな局面を洗練して制御するサーバー・オプションを提供することもできます。サーバー・オプションは、ALTER SERVER ステートメントを使用して後で変更できます。 CREATE SERVER ステートメントおよび ALTER SERVER ステートメントの詳細については、 *SQL 解説書* を参照してください。

注: サーバーを作成してサーバー・オプションを指定する前に、分散結合インストール・オプションをインストールし、データベース・マネージャー・パラメーター *federated* を YES に設定する必要があります。

サーバー・オプションとその値は、照会の後入れ先出し分析、グローバル最適化、および連合データベース操作のその他の局面に影響します。たとえば、CREATE SERVER ステートメントでは、特定のパフォーマンス統計をサーバー・オプション値として指定できます。つまり、*cpu_ratio* オプションを、データ・ソースおよび連合サーバーの CPU の相対速度を表す値に設定することができます。また、*io_ratio* オプションを、データ・ソースおよび連合サーバーの入出力装置の相対速度を表す値に設定できます。CREATE SERVER を実行すると、このデータはカタログ視点 SYSCAT.SERVEROPTIONS に追加され、最適化プログラムはデータ・ソースのアクセス・プランを立てるときにそのデータを使用します。 (たとえばデータ・ソース CPU がアップグレードされたときなどに) 統計が変更されると、 ALTER SERVER ステートメントを使用して、この変更を反映するよう SYSCAT.SERVEROPTIONS を更新できます。その後、最適化プログラムは更新された情報を使用して、データ・ソースの次のアクセス・プランを立てます。

表 8. サーバー・オプションとその設定値

オプション	有効な設定値	デフォルト設定
collating_sequence	データ・ソースが連合データベースと同じデフォルト照合順序を使用しているかどうかを、コード・セットと国別情報に基づいて指定する。データ・ソースの照合順序が DB2 の照合順序と異なっている場合、 DB2 の照合順序に依存しているほとんどの操作は、データ・ソースにおいてリモートに評価できません。たとえば、照合順序が異なるデータ・ソースにおいて、ニックネーム文字列に対して MAX 列関数を実行する場合がそうです。 MAX 関数がリモート・データ・ソースにおいて評価される場合は結果が異なるので、 DB2 は集約操作と MAX 関数をローカルに実行します。 照会に等号が含まれている場合は、照合順序が異なっていても ('N' に設定されている)、照会のその部分を後入れ先出しすることができます。たとえば、述部 C1 = 'A' はデータ・ソースに後入れ先出しすることができます。もちろん、データ・ソースでの照合順序で大文字小文字が区別されない場合、そのような照会を後入れ先出しすることはできません。データ・ソースが大文字小文字を区別しない場合、 C1= 'A' と C1 = 'a' の結果は同じです。これは大文字小文字の区別を行う環境 (DB2) では受け入れられません。 管理者は、データ・ソースの照合順序と一致する特定の照合順序の連合データベースを作成できます。この方法を用いると、すべてのデータ・ソースが同じ照合順序を使用する場合、または、ほとんどあるいはすべての列関数が同じ照合順序を使用するデータ・ソースに向けられている場合は、パフォーマンスが高速になります。 'Y' データ・ソースの照合順序は、連合データベースの照合順序と同じ。 'N' データ・ソースの照合順序は、連合データベースの照合順序と異なる。 'I' データ・ソースの照合順序は、連合データベースの照合順序と異なっており、大文字小文字を区別しない (たとえば、'TOLLESON' と 'TolLESon' は同じものと見なされる) 。	'N'

表 8. サーバー・オプションとその設定値 (続き)

オプション	有効な設定値	デフォルト設定
comm_rate	連合サーバーとその関連データ・ソース間の通信速度を指定する。秒当たりの MB 単位で表されます。 有効な値は、 0 より大きく 2147483648 より小さい数です。値は、12 などのように、自然数でのみ表すことができます。	'2'
connectstring	OLE DB Provider への接続に必要な初期化特性を指定する。接続ストリングの完全な構文と意味については、 *Microsoft OLE DB 2.0 Programmer's Reference and Data Access SDK (Microsoft Press, 1998)* の『Data Link API of the OLE DB Core Components』を参照してください。	なし
cpu_ratio	データ・ソースの CPU が連合サーバーの CPU より、どれほど速いまたは遅いかを表す。 有効な値は、 0 より大きく $1x10^{23}$ より小さい数です。値は、たとえば 123E10、123、または 1.21E4 のように、任意の有効な倍精度表記で表すことができます。	'1.0'
dbname	連合サーバーがアクセスするデータ・ソース・データベースの名前。 DB2 ファミリー・データ・ソースでは必須ですが、Oracle データ・ソースには適用されません。これは、Oracle インスタンスに 1 つのデータベースしか含まれないためです。DB2 の場合、この値はインスタンス内の特定のデータベースに対応します。 DB2 (OS/390 版) の場合は、データベースの LOCATION 値に対応します。	なし。

表 8. サーバー・オプションとその設定値 (続き)

オプション	有効な設定値	デフォルト設定
fold_id (この表の最後にある注 1 および 4 を参照。)	連合サーバーが認証のためにデータ・ソースに送信するユーザー ID に適用される。有効な値は次のとおりです。 'U' 連合サーバーは、ユーザー ID をデータ・ソースに送信する前に、ユーザー ID を大文字に変換します。これは、DB2 ファミリーおよび Oracle データ・ソースについて論理的に選択できます。(この表の最後にある注 2 を参照。) 'N' 連合サーバーは、ユーザー ID をデータ・ソースに送信する前に、ユーザー ID に対して何の処理も行いません。 (この表の最後の注 2 を参照。) 'L' 連合サーバーは、ユーザー ID をデータ・ソースに送信する前に、ユーザー ID を小文字に変換します。 これらの設定値のいずれも使用しない場合は、連合サーバーはユーザー ID を大文字にしてデータ・ソースに送信しようとします。そのユーザー ID を正常に送信できない場合は、サーバーはユーザー ID を小文字で送信しようとします。	なし。
fold_pw (この表の最後にある注 1、3 および 4 を参照。)	連合サーバーが認証のためにデータ・ソースに送信するパスワードに適用される。有効な値は次のとおりです。 'U' 連合サーバーは、パスワードをデータ・ソースに送信する前に、パスワードを大文字に変換します。これは、DB2 ファミリーおよび Oracle データ・ソースについて論理的に選択できます。 'N' 連合サーバーは、パスワードをデータ・ソースに送信する前に、パスワードに対して何の処理も行いません。 'L' 連合サーバーは、パスワードをデータ・ソースに送信する前に、パスワードを小文字に変換します。 これらの設定値のいずれも使用しない場合は、連合サーバーはパスワードを大文字にしてデータ・ソースに送信しようとします。そのパスワードを正常に送信できない場合は、サーバーはパスワードを小文字で送信しようとします。	なし。

表 8. サーバー・オプションとその設定値 (続き)

オプション	有効な設定値	デフォルト設定
io_ratio	データ・ソースの入出力システムが連合サーバーの入出力システムより、どれほど速いまたは遅いかを表す。 有効な値は、 0 より大きく $1x10^{23}$ より小さい数です。値は、たとえば 123E10、123、または 1.21E4 のように、任意の有効な倍精度表記で表すことができます。	'1.0'
node	データ・ソースが RDBMS でインスタンスとして定義される名前。すべてのデータ・ソースについて必須です。 DB2 ファミリー・データ・ソースの場合、これは連合データベースの DB2 ノード・ディレクトリーで指定されているノードです。このディレクトリーを表示するには、 **db2 list node directory** コマンドを発行します。 Oracle データ・ソースの場合、この名前は Oracle tnsnames.ora ファイルで指定されているサーバー名です。Windows NT プラットフォームでこの名前にアクセスするには、 Oracle SQL Net Easy Configuration ツールの**「構成情報の表示 (View Configuration Information)」**オプションを指定します。	なし。
password	パスワードがデータ・ソースに送信されるかどうかを指定する。 'Y' パスワードは常にデータ・ソースに送信されて妥当性検査されます。これがデフォルト値です。 'N' パスワードはデータ・ソースに送信されず (ユーザー・マッピングに関係なく)、妥当性検査されません。 'ENCRYPTION' パスワードは常に暗号化された形式でデータ・ソースに送信されて妥当性検査されます。暗号化されたパスワードをサポートする DB2 ファミリー・データ・ソースについてのみ有効。	'Y'

表 8. サーバー・オプションとその設定値 (続き)

オプション	有効な設定値	デフォルト設定
plan_hints	プラン・ヒント を使用可能にするかどうかを指定する。プラン・ヒントはステートメントの一部分であり、データ・ソース最適化プログラムについての追加情報を提供します。特定の照会タイプについてこの情報を利用すれば、照会パフォーマンスを改善することができます。プラン・ヒントは、データ・ソース最適化プログラムが索引を使用するかどうか、どの索引を使用するか、またはどの表結合順序を使うかを判別するのに役立ちます。 'Y' データ・ソースがプラン・ヒントをサポートしている場合は、プラン・ヒントが使用可能になります。 'N' プラン・ヒントはデータ・ソースで使用可能になりません。	'N'
pushdown	'Y' DB2 はデータ・ソースに操作を評価させることを考慮します。 'N' DB2 はリモート・データ・ソースからの列しか検索せず、データ・ソースに結合などのその他の操作を評価させません。	'Y'
varchar_no_trailing_blanks	このデータ・ソースが、ブランクが埋め込まれていない varchar 比較セマンティクスを使用するかどうかを指定する。後書きブランクを含んでいない可変長文字ストリングについて、一部の DBMS のブランク埋め込みなしの比較セマンティクスでは、 DB2 の比較セマンティクスと同じ結果が戻されます。データ・ソースにあるすべての VARCHAR 表 / 視点の列に後書きブランクが含まれていないことが確かである場合は、データ・ソースについてこのサーバー・オプションを 'Y' に設定することを考慮してください。このオプションは、Oracle データ・ソースでしばしば使用されます。ニックネームを持つ可能性のあるすべてのオブジェクトを考慮に入れてください。 'Y' このデータ・ソースのブランク埋め込みなしの比較セマンティクスは、DB2 と同じです。 'N' このデータ・ソースのブランク埋め込みなしの比較セマンティクスは、DB2 と同じではありません。	'N'

107ページの表8 の注:

1. このフィールドは、認証に指定される値に関係なく適用されます。

2. DB2 はユーザー ID を大文字で格納するので、値 'N' と 'U' は論理的に互いに同等です。
3. パスワードの設定が 'N' である場合は、 fold_pw を設定しても効果はありません。パスワードが送信されないので、大文字小文字の区別は意味をなしません。
4. いずれかのオプションについてヌル設定を使用することは避けてください。 DB2 がユーザー ID とパスワードの解決を複数回試行することになるので、ヌル設定は有用に思えますが、パフォーマンスが低下する場合があります (DB2 が、データ・ソース認証に成功するまでにユーザー ID とパスワードを 4 回送信するということもあり得ます)。

第5章 システム・カタログ統計

SQL 照会を最適化するとき、 SQL コンパイラーの下す決定は、データベースの内容の最適化プログラムのモデルによって大きく影響を受けます。このデータ・モデルは、最適化プログラムが、特定の照会を解決するのに使用できる代替アクセス・パスのコストを見積もるのに使用されます。

データ・モデルの中でかぎとなる要素は、データベースに含まれるデータに関して収集された統計のセットであり、それはシステム・カタログ表に保管されます。それには、表、ニックネーム、索引、列、およびユーザー定義関数 (UDF) の統計が含まれます。データ統計に変更があると、希望するデータにアクセスするための最も効率的な方法として選択されたアクセス・プランの選択も変更される可能性があります。

最適化プログラムに対してデータ・モデルを定義するために使われる統計の例としては、次のようなものがあります。
- 表内のページの数、および空でないページの数
- 元のページから他の (オーバーフロー) ページに移動された行の割合
- 1 つの表内の行数
- 1 列の中の個別値の数
- 索引のクラスター化の程度。つまり、表内の行の物理的順序が索引のとおりになっている割合
- 各索引内の索引レベルの数および葉ページの数
- 頻繁に使用される列値の出現数 (121ページの『分配統計の収集と使用』を参照)
- 列に表示されている値の範囲での列値の分布 (121ページの『分配統計の収集と使用』を参照)
- ユーザー定義関数 (UDF) のコスト見積もり

システム・カタログ内でオブジェクトの統計が更新されるのは、明示的に要求があった場合だけです。統計の一部または全部は、次のようにして更新できます。
- RUNSTATS (統計の実行) ユーティリティーを使用します (114ページの『RUNSTATS ユーティリティーを使用しての統計収集』を参照)。
- 統計収集オプションを指定して LOAD を使用します。
- 一連の定義済みカタログ視点に対して操作を行う SQL UPDATE ステートメントをコーディングします (133ページの『ユーザー更新が可能なカタログ統計』を参照)。ユーザー定義関数の統計は、この技法を使って更新しなければなりません (139ページの『ユーザー定義関数の統計の更新』を参照)。テスト・システムの実稼働環境をモデル化したり、「what-if 分析」を行う場合は、カタログは手動で更新する必要があります (UDF を除く)。実動システムでは統計を更新することはできません。

連合データベース・システムの場合、データ・ソースからニックネームの新しい統計を収集するには、ニックネームを除去し、データ・ソースで RUNSTATS と等価のユーティリティーを実行してから、ニックネームを再作成するという方法しかありません。ニックネームが作成される度に、基礎表上の統計がデータ・ソース・カタログから収集されます。

基礎表の中のデータ定義が変更された場合は、ニックネームを除去して再作成する必要があります。たとえば、表定義に列が追加された場合などです。

さらに、照会パフォーマンスが低下する場合にもニックネームの再作成を考慮する必要があります。別の方法として、SYSSTAT.TABLES の中の統計を手操作で更新することもできます。

視点のニックネームを作成するときは注意してください。ニックネームが戻す行数などの統計情報は、この視点の評価にかかる実際のコストを反映していないことがあります。 SELECT リスト上で列関数が適用されていない単一の基礎表上に視点が定義されている場合、最適化プログラムが利用できる統計情報は正確です。複合視点の場合は、最適化プログラムがデータにアクセスする効率的なプランを立てることができるように、連合データベース・システム内の DB2 ユニバーサル・データベースで、視点基礎表のニックネームに対して新しい視点を作成することを考慮してください。

追加情報:

SYSCAT カタログと SYSSTAT カタログには、収集される統計に関する情報が入ります。 *SQL 解説書* を参照してください。

- すべてのカタログ視点とその中に入っている列についての情報。
- すべての更新可能なカタログ視点とその中に入っている列についての情報。このほかにも、カタログ表の統計的な列にのみ関心がある場合は、以下の部分を参照することができます。
- 表統計についての情報。
- 列統計についての情報。
- 列分布統計についての情報。
- 索引統計についての情報。
- ユーザー定義関数統計についての情報。

RUNSTATS ユーティリティーを使用しての統計収集

RUNSTATS ユーティリティーは、照会最適化プロセス処理のために、システム・カタログ表内の統計を更新します。それらの統計がないと、データベース・マネージャーが、 SQL ステートメントのパフォーマンスを低下させるような決定を下す可能性があります。 RUNSTATS ユーティリティーを使うと、表、索引、または表と索引の両方に入っているデータに関する統計を収集することができます。

表データと索引データの両方に基づく統計を収集して、以下の状況においてアクセス・プラン選択プロセスに対し正確な情報を提供するには、 RUNSTATS ユーティリティーを使います。

- 表にデータがロードされており、該当する索引が作成済みの場合。
- 表が REORG ユーティリティーによって再編成されている場合。
- 表とその索引に影響する大量の更新、削除、および挿入が行われた場合。 (この場合「大量」とは、表データと索引データの 10 ～ 20 % 程度が影響を受けたことを意味します。)
- パフォーマンスが重要なアプリケーション・プログラムをバインドする前。
- 以前の統計と比較したい場合。定期的に統計を実行すると、パフォーマンス問題を早期発見できます。
- 事前取り出し数量が変更される場合。
- REDISTRIBUTE NODEGROUP ユーティリティーを使用している場合。

区分データベースで処理を行う場合には、単一ノードで RUNSTATS 操作を実行することによって、表とその索引に関連する統計を収集します。 (ユーティリティーを実行するノードは、コマンドを出すノードに表データが含まれているか否かによって決まります。詳細については、『RUNSTATS を実行するデータベース区画』を参照してください。) カタログに保管されている統計は表レベルの情報を表すので、データベース・マネージャーによって収集されたノード・レベルの統計に、必要に応じて、表の区分化先のノード数を乗算します。これによって、各ノードで RUNSTATS を実行してその結果を集計して得られる実際の統計の近似値を求めることができます。

注: DB2 照会最適化プログラムは、属性値 (データ) が、システムのデータベース区画全体に等量かつ均等に入っていると想定しています。データの配置が均等ではなかった場合には、代表的な表分配がなされていると考えられるデータベース区画に対して、このコマンドを実行する必要があります。

RUNSTATS を実行するデータベース区画

ある表上で RUNSTATS を起動する場合は、表を保管するデータベースに接続する必要はありますが、コマンドを出すデータベース区画には必ずしもその表の区画が含まれていなくてもかまいません。

- 表の区画を含むデータベース区画から RUNSTATS を出す場合には、ユーティリティーはそのデータベース区画で実行されます。
- 表区画が含まれないデータベース区画から RUNSTATS を出す場合には、出した要求は、その表の区画を保持するノードグループの中の先頭のデータベース区画に送られます。したがってユーティリティーは、そのデータベース区画で実行されます。

統計の分析

統計を分析すると、再編成が必要であることがわかる場合があります。それは、次のようなことに表れます。

- 索引のクラスター化

クラスター率の統計が収集される場合、値は 0 ～ 100 の範囲です。クラスター係数の統計が収集される場合、値は 0 と 1 の間です。この 2 つのクラスター統計のうち 1 つが SYSCAT.INDEXES カタログに記録されます。一般に、クラスター化率が高くなるのは表内の 1 つの索引だけです。値 -1 は、使用可能な統計がないことを示す場合に使います。

クラスター化率の値を比較する場合は、クラスター係数に 100 を乗算して、クラスター化された量のパーセンテージを求めてください。

索引専用アクセス**ではない**索引走査は、クラスター率が高い方がよくなることがあります。クラスター率が低いと、この種の走査では各データ・ページの最初のアクセスの後、次にアクセスが行われるまでバッファー・プール内にページが残っている可能性が少なくなるため入出力が増えます。バッファー・サイズを大きくすると、クラスター化されていない索引のパフォーマンスが向上することがあります。 (データベース・マネージャーでクラスター率の低い索引の索引走査パフォーマンスを向上させる方法については、256ページの『リスト事前取り出しについて』、また最適化プログラムが索引統計を使用する方法については、168ページの『クラスター索引』を参照してください。)

特定の索引に関して表データが最初にクラスター化されていて、上記のクラスター化に関する情報から、この索引に関するデータのクラスター化が不十分であることがわかった場合は、表を再編成して、その索引に関するデータを再びクラスター化することができます。

- 行のオーバーフロー

  オーバーフローの数は、元のページに収まらない行の数を示しています。オーバーフローは、VARCHAR 列が現行より長い値で更新された場合に起きることがあります。この場合、ポインターは行の元の位置のままになっています。これはパフォーマンスに悪い影響を与える可能性があります。データベース・マネージャーはポインターに従って行の内容を検出しなければならないので、それにより処理時間が長くなり、入出力の回数が多くなる可能性もあるためです。

  オーバーフロー行の数が多いほど、表データを再編成する方が効果的である可能性が高くなります。表データを再編成すると、オーバーフロー行は除去されます。

- ファイル・ページの比較

  行の含まれているページの数を、表に含まれているページの合計数と比較することができます。空ページを読み取って表走査することができます。行の範囲が全部削除されると、空ページが生じることがあります。

  空ページの数が多いほど、表を再編成する必要が大きくなります。表を再編成すると、空ページを再利用するので、表に使われるスペースの量を圧縮できます。未使用ページを再利用すると、ディスク・スペースを有効利用できるだけでなく、バッファー・プール中で読み取られるページ数が減るので、表走査のパフォーマンスを向上させることもできます。

- 葉ページの数

葉ページの数は、索引の完全な走査に必要な索引ページ入出力の回数を予測したものです。

ランダム更新アクティビティーによりページ分割が行われ、必要なスペースの最少量よりも索引のサイズが大きくなります。表の再編成中に索引を再作成すると、各索引を使用できるスペースの最少量で作成することができます。索引の最小スペース要件については、 97ページの『照会最適化に対する索引付けの影響』または、 *管理の手引き: 計画* の『索引、索引の拡張、または索引の指定の作成』というセクションを参照してください。

注: 索引の再作成を行う場合、デフォルトにより、各索引ページあたり 10% のフリー・スペースが残っています。最初に索引を作成する時に、PCTFREE パラメーターを使用してフリー・スペースの量を増やすことができます。そうすれば、索引を再編成するたびに PCTFREE 値が使用されます。索引を再編成しなければならない回数を減らしたい場合は、フリー・スペースを 10 % 以上にすることが重要です。フリー・スペースは、追加の索引挿入を受け入れるのに使用されます。

RUNSTATS は、データベースの変更とパフォーマンスの関係を調べるのにも役立ちます。統計は、表内でのデータ分布を示します。 RUNSTATS を規則的に使用することによって、一定期間内の表と索引に関するデータが提供され、それによって時間の経過に伴うパフォーマンスの傾向をデータ・モデルに識別させることができます。

理想的には、統計を実行した後でアプリケーション・プログラムを再バインドするようにします。統計が新しくなることによって、照会最適化プログラムが、異なるアクセス・プランを選択する可能性があるためです。

一度にすべての統計を収集する時間がない場合には、定期的に RUNSTATS を実行して収集できる分の統計のみを更新することもできます。表で行われた活動の結果として、RUNSTATS を実行して選択的に部分更新を行ったある期間と次の期間の間に不整合が見つかった場合は、警告メッセージ (SQL0437W、理由コード 6) が出されます。たとえば、最初は RUNSTATS を使用して表分布統計を収集したとし、次に、RUNSTATS を使用して索引統計を収集したとします。その場合に、表で行われた活動の結果として不整合が検出されたとすると、その表の分布統計は除去され、警告メッセージが出されます。こういう状況が生じたときには、RUNSTATS を実行して表分布統計を収集することをお勧めします。

RUNSTATS を定期的に使用する場合には、索引統計を表統計に同期化することができるように、表統計と索引統計の両方を一度に収集することをお勧めします。索引統計では、最後に実行された RUNSTATS で収集された表統計および列統計のほとんどを保持しています。表統計を最後に収集した時点以降にその表が広範囲に変更された場合には、その表の索引統計のみを収集すると、それらの 2 つの統計のセットは同期しないことになってしまいます。

次の状況では、索引データにのみ基づく統計を収集したいと思われるでしょう。

- RUNSTATS ユーティリティーが実行されてから新たに索引が作成されたため、表データの統計を再収集する必要がなくなった場合。
- 索引の最初の列に影響を与える大量のデータ変更がなされた場合。

RUNSTATS ユーティリティーを使うと、各種レベルの統計を収集することができます。表の場合は、基本レベルの統計を収集するか、または表内の列値に関する分布統計を収集することもできます (121ページの『分配統計の収集と使用』を参照)。索引の場合は、基本レベルの統計を収集するか、または詳細な統計を収集することもできます。これは、最適化プログラムが索引走査の入出力コストの見積もりを正確なものにするのに役立ちます。 (「詳細」 統計については、168ページの『クラスター索引』を参照してください。)

注: LONG またはラージ・オブジェクト (LOB)、または構造型列の場合は、統計は収集されません。行タイプの場合、表レベルの統計 NPAGES、FPAGES、および OVERFLOW は、副表に対して収集されません。拡張索引、または宣言済み一時表の場合、統計は収集されません。

以下の表に、RUNSTATS ユーティリティーによって更新されるカタログ統計を示します。

表 9. 表統計 (SYSCAT.TABLES と SYSSTAT.TABLES)

統計	説明	RUNSTATS オプション	
		表	索引
FPAGES	表が使用しているページの数	○	○
NPAGES	行が含まれているページの数	○	○
OVERFLOW	オーバーフローしている行の数	○	×
CARD	表内の行の数 (カーディナリティー)	○	○ (注 2)

注:
1. 区分データベースの場合は、各統計の値は、そのデータベース区画でのカウント値にデータベース区画の数を乗じて推定値を得ます。
2. 表に定義された索引がないときに、索引の統計を要求した場合には、 CARD 統計が新たに更新されることはありません。前の CARD 統計は引き続き保持されます。

表 10. 列統計 (SYSCAT.COLUMNS と SYSSTAT.COLUMNS)

統計	説明	RUNSTATS オプション	
		表	索引
COLCARD	列カーディナリティー	○ (注 1)	○ (注 2)

表 10. 列統計 (SYSCAT.COLUMNS と SYSSTAT.COLUMNS) (続き)

統計	説明	RUNSTATS オプション	
		表	索引
AVGCOLLEN	列の平均長	○	○ (注 2)
HIGH2KEY	列内で 2 番目に高い値	○	○ (注 2)
LOW2KEY	列内で 2 番目に低い値	○	○ (注 2)
NUMNULLS	列内のヌルの数	○	○ (注 2)

注:

1. COLCARD は、表内のすべての列に関して見積もられます。区分データベースの場合、列がその表の単一列区分化キーであるときには、カウントの値は、データベース区画でのカウントにデータベース区画の数を乗ずることによって見積もられます。
2. 列統計は、索引キーの中の最初の列に関して収集されます。

表 11. 索引統計 (SYSCAT.INDEXES と SYSSTAT.INDEXES)

統計	説明	RUNSTATS オプション	
		表	索引
NLEAF	索引葉ページの数	×	○ (注 3)
NLEVELS	索引レベルの数	×	○
CLUSTERRATIO	表データのクラスター化の程度	×	○ (注 2)
CLUSTERFACTOR	クラスター化の詳細の程度	×	詳細説明 (注 1、2)
DENSITY	索引の対象となるページ範囲にあるページ数に対する SEQUENTIAL_PAGES の比率 (パーセンテージ) (注 4)	×	○
FIRSTKEYCARD	索引の最初の列の個別値の数	×	○ (注 3)
FIRST2KEYCARD	索引の最初の 2 つの列の個別値の数	×	○ (注 3)
FIRST3KEYCARD	索引の最初の 3 つの列の個別値の数	×	○ (注 3)
FIRST4KEYCARD	索引の最初の 4 つの列の個別値の数	×	○ (注 3)
FULLKEYCARD	索引のすべての列の個別値の数	×	○ (注 3)
PAGE_FETCH_PAIRS	異なるバッファー・サイズでのページ取り出し見積もり	×	詳細説明 (注 1、2)

表 11. 索引統計 (SYSCAT.INDEXES と SYSSTAT.INDEXES) (続き)

統計	説明	RUNSTATS オプション	
		表	索引
SEQUENTIAL_PAGES	索引キーの順序で、間をあまり空けずにディスクに位置づけられた葉ページの数。	×	○

注:

1. 詳細索引統計は、RUNSTATS コマンドに DETAILED 文節を指定するか、または RUNSTATS API を呼び出すときに `statsopt` パラメーターに `A`、 `Y`、または `X` を指定することによって収集します。
2. 表のサイズが相当大きいものでない限り、 DETAILED 文節を指定しても CLUSTERFACTOR および PAGE_FETCH_PAIRS は収集されません。 表のページ数が約 25 より大きいと、 CLUSTERFACTOR または PAGE_FETCH_PAIRS 統計が収集されます。この場合、CLUSTERRATIO は -1 です (収集されません)。表が比較的小さいと、RUNSTATS は CLUSTERRATIO だけを記入し、 CLUSTERFACTOR および PAGE_FETCH_PAIRS は記入しません。 DETAILED 文節を指定しないと、CLUSTERRATIO 統計だけが収集されます。
3. 区分データベースの場合は、この値は、そのデータベース区画でのカウント値にデータベース区画の数を掛け合わせたものから見積もられます。
4. この統計は、その表に属する索引を含むページがファイルに対してどのくらいの比率 (パーセンテージ) を占めるかを測定します。表に定義された索引が 1 つしかない表の場合には、通常、DENSITY は 100 になります。DENSITY は、索引ページが事前取り出しされたときに、他の索引から不適切なページが平均してどのくらい読み取られたのかについて、最適化プログラムが見積もるのに使用されます。

表 12. 列分布統計 (SYSCAT.COLDIST と SYSSTAT.COLDIST)

統計	説明	RUNSTATS オプション	
		表	索引
DISTCOUNT	TYPE が Q の場合は、COLVALUE 統計以下の個別値の数	分布 (注 2)	×
TYPE	行の統計が頻出値統計または変位値統計かの標識	分布	×
SEQNO	表の行を固有に識別するのに役立つ順序番号の頻度のランク	分布	×
COLVALUE	頻出値統計または変位値統計を収集する際のデータ値	分布	×
VALCOUNT	列内でデータ値が発生する頻度、または変位数の場合は、データ値 (COLVALUE) 以下の数値	分布	×

表 12. 列分布統計 (SYSCAT.COLDIST と SYSSTAT.COLDIST) (続き)

統計	説明	RUNSTATS オプション	
		表	索引

注:

1. 列分布統計は、RUNSTATS コマンドに WITH DISTRIBUTION 文節を指定するか、または RUNSTATS API を呼び出すときに、 `statsopt` パラメーターに `A`、`D`、あるいは `Y` を指定することによって収集します。列の値が十分に不均一でないかぎり、分布統計は**収集されません**。
2. DISTCOUNT は、索引の最初のキー列である列でのみ収集されます。
3. 区分データベースの場合は、VALCOUNT の値は、そのデータベース区画でのカウントにデータベース区画の数を掛け合わせて見積もられます。例外として、TYPE が 'F' で、さらにその列が表の単一列区分化キーである場合には、 VALCOUNT は単純にデータベース区画のカウントになります。

列分布統計についての詳細は、『分配統計の収集と使用』を参照してください。

ユーザー定義関数の統計は、RUNSTATS ユーティリティーでは収集されません。それらの関数の統計は、手動で更新しなければなりません。 133ページの『ユーザー更新が可能なカタログ統計』と 139ページの『ユーザー定義関数の統計の更新』を参照してください。

分配統計の収集と使用

データベース・マネージャーでは、列内でのデータ値の分布を簡潔に記述する 2 種類の統計として、「頻出値統計」と 「変位値」を、収集、保守、および使用することができます。最適化プログラムでそれらの統計を使用すると、ある列に関して、所定の等号述部または範囲述部を満たす行の数をかなり正確に見積もりできるようになります。こうしたより正確な見積もりにより、最適化プログラムが最適の計画を選択する可能性が大きくなります。

RUNSTATS コマンドで WITH DISTRIBUTION 文節を使用すると、こうしたデータ値の分布に関する統計を収集することができます。そのような付加的な統計を収集すると、 RUNSTATS ユーティリティーには追加のオーバーヘッドが生じますが、 SQL コンパイラーはこの情報を使用して最適なアクセス・プランを選択できるようになります。

場合によっては、データベース・マネージャーは分布統計を収集せずエラーも戻らないことがあります。たとえば:

- 構成パラメーター *num_freqvalues* および *num_quantiles* をゼロ (0) に設定して、分布統計を収集しないことを指示している場合。これらのパラメーターに関する詳細は、次を参照してください。
  - 125ページの『保持する統計の数』
  - 450ページの『頻度の高い値の保存数 (num_freqvalues)』
  - 451ページの『列変位数の数 (num_quantiles)』

- データの分布が分布統計を使わなくてもわかる場合。たとえば、列の中で同じデータ値が複数個ない場合、つまり列の中の各データ値が固有の場合。
- データが統計に収集されない種類のものである場合。つまり、長フィールドまたはラージ・オブジェクトの種類のデータを使って列が定義されている場合。
- 変位数の場合に、列の中に非 NULL 値が 1 つしかない場合。

分布統計は、索引の最初の列の場合には正確です。この列以降の各列については、正確な統計を計算するには余りにも多くの時間とメモリーが必要となるため、データベース・マネージャーは、ハッシュおよびサンプリングの技法を使って、分布統計を見積もります。これらの技法は、かなり正確な統計方法とされています。

分布統計は、 SYSSTAT.COLDIST を更新し、分布統計の必要がなくなった列のすべての COLVALUE 値および VALCOUNT 値に 0 または -1 のいずれかを設定することにより、除去することができます。

以下のトピックは、管理者が上記の分布統計を理解し、使用することについてのものです。
- 分布統計の理解
- いつ分布統計を使用するべきか
- 保持する統計の数
- 最適化プログラムが分布統計を使用する方法
- 実動データベースのモデル化
- 列分布統計の更新の規則

分布統計の理解

N を N>=1 の定数とすると、列内の *N 個の最大頻出値* は、頻度 (重複の数) が最高のデータ値、 2 番目に頻度の高いデータ値、そして、N 番目に頻度の高いデータ値に至るまでのデータ値によって構成されるものです。対応する*頻出値統計* は、それらの『N』個のデータ値と、列内のそれらの値の頻度とで構成されるものです。

列の *K 変位値* とは、さまざまなデータ値のうち、データ値がそれ以下である行が『K』個以上あるような最小のデータ値 V のことです。 K 変位値は、データ値の昇順により列内の行を分類することで計算できます。 K 変位値は、その分類された列の K 番目の行にあるデータ値です。

たとえば、ある列のデータが次のようであるとします。

```
C1
--
 B
 E
 Y
 B
 F
 G
 E
```

```
A
J
K
E
L
```

この列を分類すると、次のようになります。

```
C1'
--
 A
 B
 B
 E
 E
 E
 F
 G
 J
 K
 L
 Y
```

列 C1 の中には 9 種類の異なるデータ値が存在しています。 N = 2 の場合、頻出値統計は次のようになります。

```
SEQNO    COLVALUE     VALCOUNT
-----    ---------    --------
  1          E            3
  2          B            2
```

収集される変位数が 5 の場合 (451ページの『列変位数の数 (num_quantiles)』を参照)、K = 1、3、6、9、および 12 のときのこの列の K 変位数は、次のようになります。

```
SEQNO    COLVALUE     VALCOUNT
-----    ---------    --------
  1          A            1
  2          B            3
  3          E            6
  4          J            9
  5          Y           12
```

この例では、分類後の列の 6 番目の行のデータ値が E なので、 6 変位値は E ということになります (また元の列でデータ値が E 以下の行は 6 行ある、ということです)。

変位値が共通の値の場合、同じ変位値が複数生じることがあります。ある 1 つの値について 2 つの変位数のうちの最大値が保管されます。この 2 つの変位数のうち 1 番目のものには VALCOUNT があり、これは厳密に COLVALUE 未満の行数になります。一方、2 番目のものは COLVALUE 以下の行数になります。

いつ分布統計を使用するべきか

所定の表に関して分布統計を維持すべきかどうかを判断するには、次の 2 つの要因を考慮する必要があります。

1. 静的 SQL または動的 SQL の使用。

   分布統計は、ホスト変数を使用しない動的 SQL および静的 SQL で最も便利です。ホスト変数と一緒に SQL を使用すると、最適化プログラムは分布統計を限定された方法で使用することになります。

2. データ分布での均一性の欠如。

   表内の少なくとも 1 つの列に、かなり 「不均一」なデータ分布があり、その列が下記のような等号述部または範囲述部に頻繁に現れる場合、分布統計を維持することをお勧めします。

   ```
   WHERE C1 = KEY;
   WHERE C1 IN (KEY1, KEY2, KEY3);
   WHERE (C1 = KEY1) OR (C1 = KEY2) OR (C1 = KEY3);
   WHERE C1 <= KEY;
   WHERE C1 BETWEEN KEY1 AND KEY2;
   ```

   データ分布における不均一性には、次の 2 種類がありますが、これらは一緒に発生する可能性があります。

   - 不均一性の 1 つのタイプは、データが最高データ値と最低データ値の間に均等に分布しておらず、何らかの副次的なデータ範囲の中にクラスター化されている場合です。以下の例では、データは範囲 (5、10) の中にクラスター化されています。

     ```
        C1
     -----
       0.0
       5.1
       6.3
       7.1
       8.2
       8.4
       8.5
       9.1
      93.6
     100.0
     ```

     このタイプの不均一がある場合は、変位値を維持するのが便利です。

     以下の例は、列の不均一性が高いかどうかを調べるための照会です。

     ```
     SELECT C1, COUNT(*) AS OCCURRENCES
       FROM T1
     GROUP BY C1
     ORDER BY OCCURRENCES DESC;
     ```

- もう 1 つのタイプの不均一は、特定のデータ値の頻度が、別のデータ値の頻度よりはるかに高い場合です。たとえば、データ値の頻度が次のようになっている場合です。

```
   データ値      頻度
----------   ---------
      20           5
      30          10
      40          10
      50          25
      60          25
      70          20
      80           5
```

  このタイプの不均一がある場合は、変位値と頻出値の両方を維持するのが便利です。

分布統計は、RUNSTATS コマンドで WITH DISTRIBUTION 文節を使用するか、または RUNSTATS API を呼び出すときに `statsopt` パラメーターに `D`、`E`、あるいは `A` を指定することによって収集します。アプリケーション・プログラミング・インターフェースについて詳しくは、 *管理 API 解説書* を参照してください。

保持する統計の数

列分布統計をたくさん保持するなら、最適化プログラムによるアクセス・プランの選択は改善されるかもしれませんが、それらの統計を収集し照会をコンパイルするためのコストも大きくなってしまいます。統計ヒープのサイズ (368ページの『統計ヒープ・サイズ (stat_heap_sz)』を参照) により、計算し保管できる統計の数が制限される場合があります。

分布統計が要求されると、データベース・マネージャーは、1 つの列に関してデフォルトとして 10 個の最頻出値を保管します。ほとんどの場合、 10 ～ 100 個の頻出値で十分です。頻出値の頻度とそれ以外の値の頻度が互いにほぼ等しいか、または頻出値以外の値の頻度が最頻出値の頻度に比べて無視できる程度になるよう、頻出値統計を維持するのが理想です。

収集する頻出値の数を設定するには、 *num_freqvalues* 構成パラメーターを使います (450ページの『頻度の高い値の保存数 (num_freqvalues)』を参照)。データベース・マネージャーはこの頻出値統計の数より小さい数を収集することがあります。その理由は、複数個あるデータ値に限りこの統計は収集されるからです。変位値統計だけを収集する場合は、このパラメーターをゼロに設定できます。

分布統計が要求されるデータベース・マネージャーは、 1 つの列に関してデフォルトとして 20 個の変位値を保管します。この値では、最大見積エラーは、不等号範囲述部 (>、>=、<、または <=) に関しては約 2.5%、 BETWEEN 述部では 5% が保証されます。変位値の数を決めるときの経験則は、次のとおりです。

- 範囲照会の行数を見積もるときに許容される最大エラーを決めます (これを P とします)。
- 述部が BETWEEN 述部である場合は変位値の数を約 100/P にし、述部がその他の種類の範囲述部 (<、<=、>、または >=) である場合には約 50/P にします。

たとえば、変位値を 25 にすると、BETWEEN 述部の場合の見積エラーは最大で 4%、″>″ 述部の場合の見積エラーは最大で 2% ということになります。一般に、最低 10 個の変位値を保持するようにします。 50 以上の変位値が必要になるのは、極端に不均一なデータの場合です。

変位値の数を設定するには、 *num_quantiles* 構成パラメーターを使います (451ページの『列変位数の数 (num_quantiles)』を参照)。頻出値統計だけを収集する場合は、このパラメーターをゼロに設定できます。このパラメーターを「1」に設定すると、値の範囲が全部 1 つの変位数に収まるので、この場合も変位数統計は収集されません。

最適化プログラムが分布統計を使用する方法

なぜ分布統計を収集し、保管するのでしょうか。それは、最適化プログラムが、コストが最低のアクセス・プランを選択するために、列の中で等号述部または範囲述部を満たす行の数を見積もる必要があるためです。見積もりが正確になるほど、最適化プログラムが最適のアクセス・プランを選択する可能性が大きくなります。たとえば、次の照会を考えてみます。

```
SELECT C1, C2
  FROM TABLE1
  WHERE C1 = 'NEW YORK'
  AND C2 <= 10
```

C1 に索引が 1 つ、C2 に索引が 1 つあるとします。この場合に可能なアクセス・プランの一つとしては、 C1 の索引を使用して `C1 = 'NEW YORK'` の行をすべて検索してから、取り出された行ごとに `C2 <= 10` かどうかを調べるというものが考えられます。別の計画としては、 C2 の索引を使用して `C2 <= 10` の行をすべて検索してから、取り出された行ごとに `C1 = 'NEW YORK'` であるかどうか調べる、という方法があります。一般に、こうした照会を実行するときの主なコストは、行の検索に要するコストなので、最小限の数の検索で済むプランを選択することが望ましくなります。最善のプランを選択するには、各述部を満たす行の数を見積もることが必要です。

分布統計を要求しないなら、最適化プログラムは、ある列について、2 番目に高いデータ値 (HIGH2KEY)、2 番目に低いデータ値 (LOW2KEY)、値の種類の数 (COLCARD)、および行数 (CARD) だけを維持します。これにより、等号または範囲の述部を満たす行の数は、列内の各データ値の頻度はすべて等しく、データ値が区間 (LOW2KEY、HIGH2KEY) にわたって均等に分布する、という仮定の下に見積もられます。具体的には、等号述部 `C1 = KEY` を満たす行の数は、 CARD/COLCARD として見積もられ、また範囲述部 `C1 BETWEEN KEY1 AND KEY2` を満たす行の数は、次のように見積もられます。

```
    KEY2 - KEY1
------------------  x CARD       (1)
 HIGH2KEY - LOW2KEY
```

以上の見積もりが正確な見積もりとなるのは、列内のデータ値の真の分布が、十分に均一である場合だけです。使用できる分布統計がなく、データ値の頻度が相互に大きく異なっているか、またはデータ値が幅の狭い区間 (LOW_KEY、HIGH_KEY) の中にクラスター化されている場合は、見積もりはけた違いのものになり、最適化プログラムが最適でないアクセス・プランを選択する可能性があります。

分布統計が使用できるときは、等価述部を満たす行数を計算するのに頻出値統計を使用し、範囲述部を満たす行数を計算するのに頻出値統計と変位数とを使用することによって、前述のエラーを大幅に小さくすることができます。

等価述部への影響の例:

まず、C1 = KEY の形式の述部を考えます。 KEY が、N 最大頻出値のいずれかである場合、最適化プログラムは単に、カタログ内に保管されている KEY の頻度を使用します。KEY が N 最大頻出値のどれでもない場合、最適化プログラムは、 (COLCARD - N) 個の非頻出値が均一に分布しているという仮定の下に、述部を満たす行の数を見積もります。つまり、行の数は、次のように見積もられます。

```
CARD - NUM_FREQ_ROWS
--------------------          (2)
   COLCARD - N
```

NUM_FREQ_ROWS は N 最大頻出値のいずれかと値の等しい行の合計数です。

たとえば、ある列 (C) のデータ値の頻度が以下のようになっているとします。

データ値	頻度
1	2
2	3
3	40
4	4
5	1

最頻出値 (つまり N = 1) にのみ基づいた頻出値統計が使用可能であるとします。この列の場合、CARD = 50 および COLCARD = 5 です。述部 C = 3 の場合、ちょうど 40 個の行がそれを満たしています。データ分布が均一であると仮定すると、述部を満たす行の数は 50/5 = 10 と見積もられ、エラーは -75% になります。頻出値統計を使用すると、行の数は 40 と見積もられ、エラーなしになります。

同様に、述部 C = 1 を満たす行は 2 行あります。頻出値統計を使用しないと、述部を満たす行の数は 10 で 400% のエラーと見積もられます。以下の式を使って、見積エラーを (パーセンテージで) 計算できます。

```
見積もられた行数 - 実際の行数
------------------------------  X 100
         実際の行数
```

頻出値統計 (N = 1) を使用すると、最適化プログラムは、たとえば以下のように前述の式 (2) を使用して、この値の行の数を見積もります。

```
(50 - 40)
--------- = 3
 (5 - 1)
```

また、エラー率は、次のように 1 桁減ります。

```
 3 - 2
------- = 50%
       2
```

範囲述部を満たす行の数は、変位値を使用して見積もることができます。以下の例でこれを示します。次のような列 (C) があるとします。

```
   C
-------
   0.0
   5.1
   6.3
   7.1
   8.2
   8.4
   8.5
   9.1
  93.6
 100.0
```

K 変位値は、K = 1、4、7、および 10 のものが使用可能であるとします。

```
 K      K 変位値
---  ----------
 1       0.0
 4       7.1
 7       8.5
10     100.0
```

まず最初に述部 C <= 8.5 について考えます。前述のデータでは、正確には 7 つの行がこの述部を満たしています。データ分布が均一であると仮定し、前述の式 (1) で KEY1 を LOW2KEY に置き換えると、述部を満たす行の数は、次のように見積もられます。

```
 8.5 - 5.1
---------- x 10 *= 0
93.6 - 5.1
```

*= は 「ほぼ等しい」という意味です。この見積もりのエラーは、約 -100% です。

変位値を使用する場合は、まず K 変位値の 1 つである 8.5 を見つけて、それに対応する K の値 7 を見積もりとして使用することによって、同じ述部 (C <= 8.5) を満たす行の数が見積もられます。この場合、エラーは 0 になります。

次に、述部 C <= 10 について考えます。正確には 8 行がこの述部を満たしています。前述の例とは異なり、値 10 は、保管された K 変位値のどれでもありません。 一様なデータ分布を想定し、(1) の公式を使用すると、 その述部を満たす行数は -87.5% の誤りで 1 と見積もられます。

変位値を使用する場合は、最適化プログラムは述部を満たす行の数を r_1 + r_2 と見積もります。ここで、r_1 は、述部 C <= 8.5 を満たす行の数、 r_2 は、述部 C > 8.5 かつ C <= 10 を満たす行の数です。 前述の例のとおり、r_1 = 7 です。 r_2 を見積もるため、最適化プログラムは線形補完を使用します。

```
       10 - 8.5
r_2 *= ---------- x (number of rows with value > 8.5 and <= 100.0)
       100 - 8.5
       10 - 8.5
r_2 *= ---------- x (10 - 7)
       100 - 8.5
       1.5
r_2 *= ---- x (3)
       91.5
r_2 *= 0
```

最終的な見積もりは、r_1 + r_2 *= 7 であり、エラーは -12.5% のみです。

前述の例で変位値を使うと見積もりの正確さが増したのは、実際のデータ値は範囲 5〜10 でクラスター化しているのに、標準の見積式ではそのデータ値が 0〜100 の間で均一に分布していると想定されているためです。

このほかにも、変位値を使うと、データ値ごとの頻度の差が非常に大きい場合に正確さを向上させることができます。ある列のデータ値の頻度が次のようになっているものとします。

データ値	頻度
20	5
30	5
40	15
50	50
60	15
70	5
80	5

K 変位値は、K = 5、25、75、95、および 100 のものが使用可能であるとします。

K	K 変位値
5	20

```
 25       40
 75       50
 95       70
100       80
```

さらに、3 個の最頻出値に基づく頻出値統計が使用可能であるとします。

述部 C BETWEEN 20 AND 30 を考えてみましょう。データ値の分布状況からは、正確には 10 個の行がこの述部を満たしていることが分かります。データ分布が均一であると仮定し、式 (1) を使うと、述部を満たす行の数は次のように見積もられます。

```
  30 - 20
-------  x 100 = 25
70 - 30
```

エラーは 150% です。

頻出値統計と変位値を使用すると、述部を満たす行の数は r_1 + r_2 と見積もられます。ここで、r_1 は述部 (C = 20) を満たす行数、 r_2 は述部 C > 20 AND C <= 30 を満たす行数となります。式 (2) を使用すると、r_1 は次のように見積もられます。

```
100 - 80
-------- = 5
  7 - 3
```

線形補完を使用すると、r_2 は次のように見積もられます。

```
  30 - 20
  ------- x (> 20 かつ <= 40 を満たす行の数)
  40 - 20
  30 - 20
= ------- x (25 - 5)
  40 - 20
= 10,
```

最終見積もりは 15 になり、エラーは 3 分の 1 になりました。

詳細な索引統計の収集と使用

オプションとして、索引に関するより詳細な統計の収集を行うと、最適化プログラムが、その索引を使用して、表にアクセスする際のコストをより正確に見積もることができるようになります。これを行うには、2 つの方法のいずれかを選択できます。 1 つは、 RUNSTATS コマンドで DETAILED 文節を使用する方法で、もう 1 つは、RUNSTATS API の呼び出し時に statsopt パラメーターに A、Y、X のいずれかを指定する方法です。 DETAILED 統計の PAGE_FETCH_PAIRS と CLUSTERFACTOR は、表のサイズが十分に大きい (約 25 ページ以上) 場合にのみ収集されます。この場合には、CLUSTERFACTOR の値は 0 から 1 の間になり、 CLUSTERRATIO の値は -1 (収集されないことを示す) になります。 25 ページより小さい表の場合には、

CLUSTERFACTOR の値は -1 (収集されないことを示す) になり、その表の索引に対して DETAILED 文節を指定した場合であっても、 CLUSTERRATIO の値は 0 から 100 になります。

詳細な索引統計の理解

DETAILED 統計は、簡潔な方法で、完全な索引走査がさまざまなバッファー・サイズの下で行われるとしたときに、表のデータ・ページにアクセスするのに必要になる物理入出力の数を把握しようとするものです。 RUNSTATS は索引のページ全体を走査するとして、さまざまなバッファー・サイズのモデルを作り、ページ不在が起こる頻度の見積もりを収集します。たとえば、使用可能なバッファー・ページが 1 ページだけという場合には、索引によって新しくページが参照されるたびに (状況によっては、各行が異なるページを参照するたびに) ページ不在が生じることになるので、 CARDINALITY 入出力が非常に多くなります。他方の極端な例をあげると、バッファーが表全体を入れられるくらい大きい (ただし最大バッファー・サイズ以下) 場合には、その表の NPAGES ページのそれぞれが、完全に 1 回で物理的に読み取られることになります。したがって、物理入出力の数は必ずバッファー・サイズの単調で非増加の関数になります。

RUNSTATS は、これらの見積もりに断片線形曲線を当てはめ、この曲線は、PAGE_FETCH_PAIRS 統計の 11 対のストリングとして保管されます。各対の最初の値は仮定バッファー・サイズを表し、2 番目の値は、そのサイズのバッファーが全部索引の走査に使用可能であるとして、 1 回の索引走査でデータ・ページを取り出す際の物理入出力の見積数を表します。次に、最適化プログラムは、PAGE_FETCH_PAIRS 統計を使用して、その索引を使用した全索引走査または部分索引走査におけるデータ・ページ取り出しの物理入出力の数を見積もります。

PAGE_FETCH_PAIRS に保管されている索引の曲線の形状は、その索引のクラスター化の動作によって異なります。

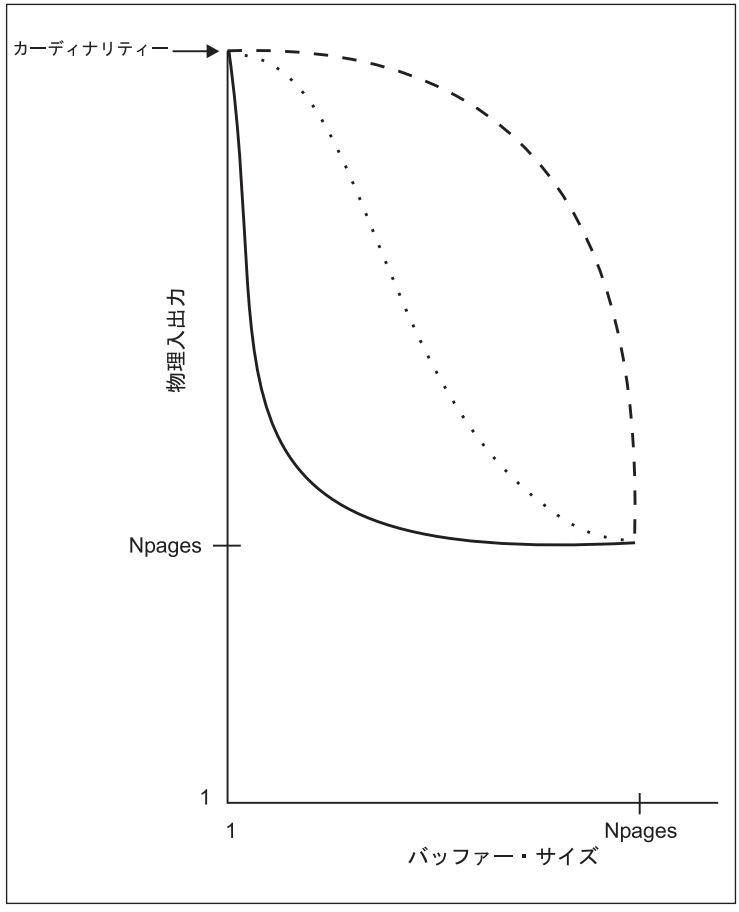


図 11. クラスター化索引および非クラスター化索引の 3 種類の曲線

考えられる曲線には 3 つのタイプがあります。

1. 曲線 1 (破線) は、再参照されるページがバッファー内で見つかる前に、表とほぼ同じ大きさのバッファーを必要とする、高度非クラスター化索引です。これは、同じページへの参照が索引のキー値全体に広範囲に広がっているため、中程度のサイズのバッファーでは不十分で、同じページへの多数回の再参照が避けられないという状況を表しています。これは、首尾良く実行するために非常に大きなバッファー・スペースを必要とするという意味で、最悪のシナリオです。最適化プログラムは、このような索引に対してリスト事前取り出しアクセス方式を使用し、索引の適格キー値を求めて、データ・ページ・アクセスをクラスター化しようと試みます。この索引が頻繁に使用される場合には、その索引は再編成の第 1 候補と言えます。
2. 曲線 2 (実線) は、よりローカルに非クラスター化されています。非常に小さなバッファーの場合には、曲線 1 と同程度にクラスター化されていない状態ですが、最後に参照されたデータを入れることができるバッファー・ページが数ページ使用可能に

なると、データ・ページのヒット率は大きく上昇します。これは、索引が特別にクラスター化されていないにもかかわらず、同じデータ・ページの参照同士が索引のキー値においてごく近くにあり、若干改善された状況を表しています。

3. 曲線 3 (点線) は、2 つの極端の中間に位置し、バッファーの増大にともなった一様な率での改善が見られます。これが通常、非クラスター化索引の場合の一般的なケースであり、 DETAILED 索引がないときに最適化プログラムが使用するものを表しています。

詳細な索引統計の使用

照会において参照する列がすべて索引内に収まっているわけではない場合には、DETAILED 索引統計を使用するようにしてください。また、DETAILED 索引統計は以下のような場合に使用する必要があります。

- クラスター化の程度が異なる複数の非クラスター化索引が存在する場合
- クラスター化の程度がキー値間で一様ではない場合
- 索引の値が不均一に更新される場合

以前の状況に関する知識がなく、さらに、バッファー・サイズを変えて索引走査の施行を試行し、その結果の物理入出力を観察するモニターを使用しなければ、これらの状況を判別するのは非常に難しいと言えます。これらの状況が生じているか否かを最も簡単に判別する方法は、索引に関する DETAILED 統計を収集して、その結果の PAGE_FETCH_PAIRS が非線形であった場合にはそれらを保存することです。

ユーザー更新が可能なカタログ統計

選択されたシステム・カタログを更新できるという機能によって、次のことが可能になります。

- 開発システム上で、実動システム統計を使って照会パフォーマンスをモデル化する。
- 『what if』照会パフォーマンス分析を実行する。

実動システム上では統計を更新**しないで**ください。最適化プログラムが照会に最適のアクセス・プランを検出する妨げになってしまうためです。

それらの統計列の値を更新するには、 SYSSTAT スキーマに定義された視点に対して SQL UPDATE ステートメントを使用します。更新できる統計は、次のものです。

- 明示的な CONTROL 特権を付与されている表。この表の列と索引に関する統計を更新することもできます。
- 連合データベース・システム内で明示的な CONTROL 特権を付与されているニックネーム。これらのニックネームの列と索引に関する統計を更新することもできます。更新が影響を与えるのは、ローカル・メタデータだけであることに注意してください (データ・ソース表統計は変更されません)。これらの変更が影響を与えるのは、 DB2 最適化プログラムが生成するグローバル・アクセス戦略だけです。
- 独自のユーザー定義関数 (UDF) (139ページの『ユーザー定義関数の統計の更新』を参照)。

さらに、自分のユーザー ID に、データベースに対して明示的な DBADM 権限が与えられている場合、つまり、そのユーザー ID が DBADM 権限を持つものとして SYSCAT.DBAUTH 表に記録されている場合は、これらの統計を更新することができます。 DBADM グループに属していることは、明示的にこの権限を付与することにはなりません。

これらの視点を使用すると、DBADM ですべてのユーザーの統計行を表示することができます。 DBADM 権限のないユーザーは、CONTROL 特権を持つオブジェクトの統計が入った行しか表示できません。

以下に、EMPLOYEE 表の表統計を更新する例を示します。

```
UPDATE SYSSTAT.TABLES
SET  CARD   = 10000,
     NPAGES = 1000,
     FPAGES = 1000,
     OVERFLOW = 2
WHERE TABSCHEMA = 'userid'
  AND TABNAME   = 'EMPLOYEE'
```

カタログ統計を更新するときは慎重に行ってください。勝手な更新は、以降の照会のパフォーマンスに重大な影響を与える可能性があります。以下のいずれかの方法を使って、これらの表に適用した更新をすべて置換できます。

- 変更が加えられた作業単位の ROLLBACK を実行します (その作業単位がコミットされていないことが前提です)。
- RUNSTATS ユーティリティーを使用すると、カタログ統計を計算し直し、最新表示することができます。
- カタログ統計を更新して、その統計が収集されていないことを示します。 (たとえば、列 NPAGES を -1 に設定すると、ページ数の統計は収集されなかったことが示されます。)
- カタログ統計を更新前のデータで置換します。この方法は、*db2look* ツールを使って (141ページの『実動データベースのモデル化』を参照)、変更前の統計をキャプチャーする場合に限り使えます。

場合によっては、最適化プログラムが特定の統計値か値の組み合わせを無効と判断して、デフォルト値を使い、警告を出すことがあります。しかし、統計の更新時には大部分の妥当性検査が行われるので、このような状況はまれです。

追加情報: カタログ統計の更新については、以下を参照してください。

- 135ページの『カタログ統計の更新規則』
- 135ページの『表統計とニックネーム統計の更新の規則』
- 136ページの『列統計の更新の規則』
- 137ページの『列分布統計の更新の規則』
- 138ページの『索引統計の更新の規則』
- 139ページの『ユーザー定義関数の統計の更新』
- 141ページの『実動データベースのモデル化』

カタログ統計の更新規則

カタログ統計を更新する場合、一般的な規則のうち最も重要なのは、各種統計の有効な値、範囲、および形式を確実に統計視点に保管するということです。さらに、各種統計相互間の関係の一貫性を保つことも重要です。

たとえば、SYSSTAT.COLUMNS 内の COLCARD は SYSSTAT.TABLES 内の CARD より小さくなければなりません (列内の個別値の数は、行の数より多くすることはできません)。ここで、COLCARD を 100 から 25 に減らし、CARD を 200 から 50 に減らしたいと仮定します。このとき最初に SYSCAT.TABLES を更新したとすると、エラーを受け取ることになります (CARD が COLCARD より小さくなってしまうため)。正しい順序は、最初に SYSCAT.COLUMNS 内の COLCARD を更新し、その後で SYSSTAT.TABLES 内の CARD を更新するという順序です。逆に COLCARD を 100 から 250 に増やし、CARD を 200 から 300 に増やしたい場合にも、同じことが言えます。その場合には、最初に CARD を更新してから、その後で COLCARD を更新しなければなりません。

更新された統計と別の統計との間に矛盾が検出された場合には、エラーが出されます。ただし、矛盾が生じたときに必ずエラーが出されるとは限りません。特に 2 つの関連する統計が別々のカタログにある場合など、状況によっては、矛盾を検出したりエラーを報告したりするのが難しいことがあります。したがって、こういった矛盾を起こさないように十分注意が必要です。

カタログ統計を更新する前に検査する必要のある規則のうち、最も一般的なのは次のものです。

1. 数値統計は -1 もしくは 0 以上でなければなりません。
2. パーセンテージを表す数値統計 (たとえば SYSSTAT.INDEXES 内の CLUSTERRATIO) は、 0 ～ 100 の範囲でなければなりません。

注: 行タイプの場合、表レベルの統計 NPAGES、FPAGES、および OVERFLOW は、副表に対して更新されません。

表統計とニックネーム統計の更新の規則

SYSTAT.TABLES 内で更新可能な統計値は、 CARD、 FPAGES、 NPAGES、 OVERFLOW の 4 つだけです。更新の際には、以下に留意してください。

1. CARD は、その表に対応する SYSSTAT.COLUMNS 内のすべての COLCARD 値よりも大きくなければなりません。
2. CARD は、NPAGES より大きくなければなりません。
3. FPAGES は、NPAGES より大きくなければなりません。
4. NPAGES は、(この統計が索引に関連している場合) どの索引の PAGE_FETCH_PAIRS 列内のどの ″Fetch (取り出し)″ 値よりも小さいか、等しくなければなりません。

5. CARD は、(この統計が索引に関連している場合) どの索引の PAGE_FETCH_PAIRS 列内のどの ″Fetch (取り出し)″ 値よりも大きくなければなりません。

連合データベース・システムで作業している場合、リモート視点に対するニックネームについての統計を手操作で提供 / 更新するときには、注意が必要です。ニックネームが戻す行数などの統計情報は、このリモート視点の評価にかかる実際のコストを反映していないことがあるので、 DB2 最適化プログラムが正しく動作しないことがあります。統計の更新が役立つ状況には、リモート視点が SELECT リスト上で列関数が適用されていない単一の基礎表上に定義されている場合が含まれます。複合視点では、それぞれの照会の調整が必要な複合チューニング・プロセスが必要になることがあります。 DB2 最適化プログラムが視点のコストをより正確に導き出す方法を知ることができるように、ニックネームに対してローカル視点を作成することを考慮してください。

列統計の更新の規則

SYSSTAT.COLUMNS 内の統計を更新する場合には、以下の指針に従ってください。列分布統計の更新の詳細については、137ページの『列分布統計の更新の規則』を参照してください。

1. (SYSSTAT.COLUMNS 内にある) HIGH2KEY と LOW2KEY は、以下の規則に従う必要があります。
   - HIGH2KEY、LOW2KEY のどの値のデータ・タイプも、統計が取得されるユーザー列のデータ・タイプに対応していなければなりません。なお、HIGH2KEY は VARCHAR 列なので、値は引用符で囲む必要があります。たとえば、INTEGER ユーザー列の HIGH2KEY を 25 に設定するには、更新ステートメントには SET HIGH2KEY = '25' と指定します。
   - HIGH2KEY、LOW2KEY 値の長さは、33 または目標列のデータ・タイプの最大長の、いずれか小さい方でなければなりません。
   - HIGH2KEY は、対応する列に異なる値が 4 つ以上含まれるときは必ず LOW2KEY よりも大きい値にする必要があります。 列に含まれる値のうち異なる値が 3 つ以下の場合、 HIGH2KEY は LOW2KEY と同じにすることができます。
2. 列のカーディナリティー (SYSSTAT.COLUMNS 内の COLCARD 統計) は、対応する表のカーディナリティー (SYSSTAT.TABLES 内の CARD 統計) より大きくすることはできません。
3. 列のヌルの数 (SYSSTAT.COLUMNS 内の NUMNULLS 統計) は、対応する表のカーディナリティー (SYSSTAT.TABLES 内の CARD 統計) より大きくすることはできません。
4. データ・タイプが LONG VARCHAR、 LONG VARGRAPHIC、 BLOB、 CLOB、 DBCLOB の列の統計はサポートされていません。

列分布統計の更新の規則

カタログ統計を更新する方法に関する一般的な情報については、 133ページの『ユーザー更新が可能なカタログ統計』で説明されています。列分散統計の更新を試みる前に、この部分をお読みください。

カタログ内のすべての統計を矛盾のない状態にするためには、分布統計を更新するときに十分に注意を払わなければなりません。具体的には、各列ごとに、頻出データ統計と変位値のカタログ項目は、以下の制約を満たしていなければなりません。

- 頻出値統計 (SYSSTAT.COLDIST カタログ中)。この制約には次のものが含まれます。
  - 列 VALCOUNT の値は、SEQNO の値の増加に対して、不変または減少でなければなりません。
  - 列 COLVALUE の値の数は、その列の個別値の数 (この数は、カタログ視点 SYSSTAT.COLUMNS 内の列 COLCARD に保管されている) 以下でなければなりません。
  - 列 VALCOUNT の値の合計数は、その列の行の数 (カタログ表 SYSSTAT.TABLES の列 CARD に保管されているもの) 以下でなければなりません。
  - ほとんどの場合、列 COLVALUE の値は、その列の 2 番目に高いデータ値と 2 番目に低いデータ値 (それぞれカタログ表 SYSSTAT.COLUMNS の列 HIGH2KEY および LOW2KEY に保管されているもの) との間にあります。 HIGH2KEY より大きい頻出値が 1 つと LOW2KEY より小さい頻出値が 1 つ存在してもかまいません。
- 変位数 (SYSSTAT.COLDIST カタログ内の)。この制約には次のものが含まれます。
  - 列 COLVALUE の値は、SEQNO の値の増加に対して、不変または減少でなければなりません。
  - 列 VALCOUNT の値は、SEQNO の値が増加するにつれて、単調増加でなければなりません。
  - 列 COLVALUE 内の最大値に対応する列 VALCOUNT の項目は、その列の行の数と等しいものでなければなりません。
  - ほとんどの場合、列 COLVALUE の値は、その列の 2 番目に高いデータ値と 2 番目に低いデータ値 (それぞれカタログ表 SYSSTAT.COLUMNS の列 HIGH2KEY および LOW2KEY に保管されているもの) との間にあります。

行の数が『R』個である列 C1 で分布統計が使用可能である場合に、データ値の相対比率を同じに保ったまま、行数を『(F × R)』にした列に対応するように統計を変更したいものとします。頻出値統計を F 倍するには、列 VALCOUNT の各項目を F 倍しなければなりません。同様に、変位値を F 倍するには、列 VALCOUNT 内の各項目を F 倍しなければなりません。これらの規則に従わないなら、照会の実行時に最適化プログラムが誤ったフィルター要因を使用することになり、予測不能なパフォーマンスになる可能性があります。

索引統計の更新の規則

SYSSTAT.INDEXES 内の統計を更新する場合には、以下で説明する規則に従ってください。

1. PAGE_FETCH_PAIRS (SYSSTAT.INDEXES 内にある) は、以下の規則に従う必要があります。
   - PAGE_FETCH_PAIRS 統計内の個々の値は、一連のブランク区切り文字によって区切る必要があります。
   - PAGE_FETCH_PAIRS 統計内の個々の値は 10 桁より長くすることはできず、かつ最大整数値 (MAXINT = 2147483647) より小さい値でなければなりません。
   - CLUSTERFACTOR が 0 より大きい場合は、必ず有効な PAGE_FETCH_PAIRS 値が存在していなければなりません。
   - 単一の PAGE_FETCH_PAIR 統計に含まれるのは、ちょうど 11 対でなければなりません。
   - PAGE_FETCH_PAIRS のバッファー・サイズ項目は、値の昇順で並んでいなければなりません。
   - PAGE_FETCH_PAIRS 項目のバッファー・サイズ値を、 MIN (NPAGES, 524287) より大きくすることはできません (NPAGES は、対応する表 (SYSSTAT.TABLES) のページ数)。
   - PAGE_FETCH_PAIRS の「取り出し」項目は、値が降順でなければなりません。この場合、個々の「取り出し」項目は NPAGES 以上です。
     PAGE_FETCH_PAIRS 項目内の「取り出し」サイズ値は、対応する表の CARD (カーディナリティー) 統計より大きくすることはできません。
   - バッファー・サイズの値が 2 つの連続した対の値と同じ場合は、ページ取り出しの値も両方の対 (SYSSTAT.TABLES 中) の値と同じでなければなりません。

   有効な PAGE_FETCH_UPDATE は次のとおりです。

   ```
   PAGE_FETCH_PAIRS =
    '100 380 120 360 140 340 160 330 180 320 200 310 220 305 240 300
     260 300 280 300 300 300'
   ```

   ここで、

   ```
   NPAGES = 300
   CARD   = 10000
   CLUSTERRATIO  =  -1
   CLUSTERFACTOR = 0.9
   ```

2. CLUSTERRATIO と CLUSTERFACTOR (SYSSTAT.INDEXES 内にある) は、以下の規則に従わなければなりません。
   - CLUSTERRATIO の有効な値は -1、または 0 と 100 の間です。
   - CLUSTERFACTOR の有効な値は -1、または 0 と 1 の間です。
   - CLUSTERRATIO 値と CLUSTERFACTOR 値のうち少なくとも 1 つは、常に -1 でなければなりません。
   - CLUSTERFACTOR が正の値の場合は、有効な PAGE_FETCH_PAIR 統計が伴わなければなりません。

3. 以下の規則は、 FIRSTKEYCARD、 FIRST2KEYCARD、 FIRST3KEYCARD、 FIRST4KEYCARD、および FULLKEYCARD に適用されます。
   - FIRSTKEYCARD は、単一列索引の場合には FULLKEYCARD と等しくなければなりません。
   - FIRSTKEYCARD は、対応する列の COLCARD (SYSSTAT.COLUMNS 内) と等しくなければなりません。
   - これらの索引統計のいずれかが必要ない場合には、それらを -1 に設定する必要があります。たとえば、索引に 3 行しか列がない場合には、 FIRST4KEYCARD を -1 に設定します。
   - 複数の列索引の場合、すべての統計が必要ならば、それらの間の関係は以下のようにしなければなりません。

     ```
     FIRSTKEYCARD <= FIRST2KEYCARD <= FIRST3KEYCARD <= FIRST4KEYCARD
        <= FULLKEYCARD <= CARD
     ```

4. 以下の規則は、SEQUENTIAL_PAGES および DENSITY に適用されます。
   - SEQUENTIAL_PAGES の有効な値は -1、または 0 と NLEAF の間です。
   - DENSITY の有効な値は -1、または 0 と 100 の間です。

ユーザー定義関数の統計の更新

SYSSTAT.FUNCTIONS カタログ視点を使用すると、ユーザー定義関数 (UDF) の統計を更新することができます。これらの統計が使用可能であれば、最適化プログラムは各種アクセス・プランのコストを見積もる際にそれらを使用します。使用できる統計がないなら、統計の列の値は -1 になり、最適化プログラムは単純な UDF を前提とするデフォルト値を使用します。

UDF に関して更新可能な統計列についての情報を、次の表にまとめます。

表 13. 関数統計 (SYSCAT.FUNCTIONS と SYSSTAT.FUNCTIONS)

統計	説明
IOS_PER_INVOC	関数が実行されるたびに実行される読み取り / 書き込み要求の数の見積値。
INSTS_PER_INVOC	関数が実行されるたびに実行されるマシン語命令の数の見積値。
IOS_PER_ARGBYTE	入力引き数バイトごとに実行される読み取り / 書き込み要求の数の見積値。
INSTS_PER_ARGBYTES	入力引き数バイトごとに実行されるマシン語命令数の見積もり。
PERCENT_ARGBYTES	関数が実際に処理する入力引き数バイトの平均比率 (%) の見積値。
INITIAL_IOS	関数が最初または最後に呼び出されるときにだけ実行される読み取り / 書き込み要求の数の見積値。

表 13. 関数統計 (SYSCAT.FUNCTIONS と SYSSTAT.FUNCTIONS) (続き)

統計	説明
INITIAL_INSTS	関数が最初または最後に呼び出されるときにだけ実行されるマシン語命令の数の見積値。
CARDINALITY	表関数によって生成される行数の見積値。

たとえば、米国製の靴のサイズを、それに対応する欧州製の靴のサイズに変換する UDF (EU_SHOE) を考えてみましょう。 (これらの 2 つの靴のサイズは、UDT になります。) この UDF の場合、統計列を次のように設定します。

- INSTS_PER_INVOC は、次のことを行うのに必要なマシン語命令数の見積もりに設定します。
  - EU_SHOE の呼び出し
  - 出力ストリングの初期化
  - 結果の戻り
- INSTS_PER_ARGBYTE を、入力ストリングを欧州製靴サイズに変換するのに必要な、マシン語命令数の見積もりに設定する必要があります。
- PERCENT_ARGBYTES を 100 に設定すると、入力ストリング全体を変換することになります。
- INITIAL_INSTS、IOS_PER_INVOC、IOS_PER_ARGBYTE、および INITIAL_IOS はすべて 0 に設定する必要があります。この UDF は計算しか実行しないからです。

PERCENT_ARGBYTES は、必ずしも入力ストリング全体を処理するとは限らない関数によって使用されます。たとえば、2 つの引き数を入力とし、第 2 引き数の中で、第 1 引き数が現れる最初の出現の開始位置を返す UDF (LOCATE) を考えてみましょう。第 1 引き数の長さが第 2 引き数と比べると十分に小さく、第 2 引き数の平均 75 % が探索されるものとします。この情報に基づくと、PERCENT_ARGBYTES は 75 に設定されます。この平均 75 % の見積もりは、以下に示す追加の前提事項に基づいています。

- 2 回に 1 回は、最初の引き数は検出されず、2 番目の引き数全体が探索されることになります。
- 最初の引き数は 2 番目の引き数内のどの場所にも現れる可能性があるため、最初の引き数が検出されるとき、平均して 2 番目の引き数の半分が探索される結果となります。

INITIAL_INSTS または INITIAL_IOS を使用すると、最初または最後に関数が呼び出されるときにだけ実行されるマシン語命令または読み取り / 書き込み要求の数の見積もりを記録することができます。これは、たとえば、スクラッチパッド域をセットアップするコストを記録するときなどに使用できます。

ユーザー定義関数によって使用される入出力と命令についての情報を得るには、プログラム言語コンパイラーによって、またはオペレーティング・システムで使用可能なモニター・ツールによって提供される出力を使用することができます。

実動データベースのモデル化

テスト・システムに、実動システムのデータのサブセットを入れる必要が生じる場合があります。しかし、テスト・システムのカタログ統計と構成パラメーターを実動システムのカタログ統計と構成パラメーターと一致するように更新したのでない限り、この種のテスト・システムで選択されたアクセス・プランは、実動システムで選択されるアクセス・プランと必ずしも同じにはなりません。

生産性向上ツール *db2look* を実動データベースに対して実行すると、テスト・データベースのカタログ統計を実動のものと一致させるのに必要な更新ステートメントを生成することができます。これらの更新ステートメントを生成する場合は、模擬モード (-m オプション) で *db2look* を使用します。そのようにした場合、*db2look* は、実動データベースのカタログ統計を模造するのに必要なステートメントをすべて含むコマンド・プロセッサー・スクリプトを生成します。これは、テスト環境で Visual Explain を使用して SQL ステートメントを分析するときに便利です。

db2look -e を用いて DDL ステートメントを抽出すると、表、視点、索引およびデータベース内の他のオブジェクトを含むデータベース・データ・オブジェクトを再作成することができます。このコマンドから作成されたコマンド・プロセッサー・スクリプトを別のデータベースに対して実行すると、データベースを再作成することができます。 -e オプションは -m オプションと一緒に使用することができます。

db2look によって生成された更新ステートメントをテスト・システムに対して実行したなら、テスト・システムを使用して、実動で生成されるアクセス・プランを妥当性検査できるようになります。最適化プログラムが表スペースのタイプと構成を使用して入出力コストの見積もりを行うので、テスト・システムには、同じ表スペース形状つまり表スペース・レイアウトがなければなりません。すなわち、同じタイプ (SMS か DMS のいずれか) のコンテナーが同じ数だけなければなりません。

db2look ツールは、 *bin* サブディレクトリーにあります。

この生産性向上ツールの使用方法について知りたい場合は、コマンド行で次のように入力してください。

```
db2look -h
```

このツールの詳細については、 *コマンド解説書* も参照できます。

コントロール・センターにも、この *db2look* ユーティリティー (「SQL の生成 - オブジェクト名 (Generate SQL - Object Name)」と呼ばれる) へのインターフェースが備わっています。コントロール・センターを使用すると、このユーティリティーからの結果ファイルをスクリプト・センターに統合することができます。コントロール・センターから、 *db2look* コマンドをスケジュールすることもできます。コントロール・センターを使用する場合の違いは、 *db2look* コマンドを使用する場合は、 1 つの呼び出しで最

大 30 の表を分析できるのに対して、 1 つの表の分析しか行えないということです。コントロール・センターからは、 LaTex および図形出力はサポートされていないことも知っておく必要があります。

db2look ユーティリティーは OS/390 データベースに対して実行することもできます。db2look ユーティリティーは、 OS/390 オブジェクトの DDL および UPDATE 統計ステートメントを抽出します。これは、OS/390 オブジェクトを抽出して、 DB2 ユニバーサル・データベース (UDB) データベース内にそのオブジェクトを再作成したい場合に非常に役立ちます。 db2look ユーティリティーに関する追加情報については、 *コマンド解説書* を参照してください。

DB2 UDB の統計と OS/390 の統計の間にはいくらかの違いがあります。 db2look は、適切な場合に DB2 (OS/390 版) から DB2 UDB への適切な変換を実行し、 DB2 (OS/390 版) で対応するものが存在しない DB2 UDB の統計についてはデフォルト値 (-1) を設定します。以下に、db2look ユーティリティーが、 DB2 (OS/390 版) の統計を DB2 UDB の統計にマップする方法を示します。以下の説明で、 『UDB_x』は DB2 UDB の統計列を、『S390_x』は DB2 (OS/390 版) の統計列を表します。

1. 表レベル統計。

   UDB_CARD = S390_CARDF
   UDB_NPAGES = S390_NPAGES

   S390_FPAGES はありません。ただし、DB2 (OS/390 版) には、PCTPAGES という別の統計があり、表の行を含んでいる活動状態の表スペース・ページのパーセンテージを表します。それで、以下のように、 S390_NPAGES および S390_PCTPAGES に基づいて UDB_FPAGES を計算することができます。

   ```
   UDB_FPAGES=(S390_NPAGES * 100)/S390_PCTPAGES
   ```

   UDB_OVERFLOW にマップする S390_OVERFLOW はありません。そのため、db2look ユーティリティーは、この値をデフォルト値に設定します。

   ```
   UDB_OVERFLOW=-1
   ```

2. 列レベル統計。

   UDB_COLCARD = S390_COLCARDF
   UDB_HIGH2KEY = S390_HIGH2KEY
   UDB_LOW2KEY = S390_LOW2KEY

   UDB_AVGCOLLEN にマップする S390_AVGCOLLEN がないため、 db2look ユーティリティーは、この値をデフォルト値に設定します。

   ```
   UDB_AVGCOLLEN=-1
   ```

3. 索引レベル統計。

UDB_NLEAF = S390_NLEAF
UDB_NLEVELS = S390_NLEVELS
UDB_FIRSTKEYCARD= S390_FIRSTKEYCARD
UDB_FULLKEYCARD = S390_FULLKEYCARD
UDB_CLUSTERRATIO= S390_CLUSTERRATIO

OS/390 で対応するものがない他の統計は、デフォルトに設定されます。つまり、以下のようになります。

```
UDB_FIRST2KEYCARD = -1
UDB_FIRST3KEYCARD = -1
UDB_FIRST4KEYCARD  = -1
UDB_CLUSTERFACTOR = -1
UDB_SEQUENTIAL_PAGES = -1
UDB_DENSITY = -1
```

4. 列分布統計。

DB2 (OS/390 版) SYSIBM.SYSCOLUMNS には、 2 つのタイプの統計があります。頻出値を表す「F」と、カーディナリティーを表す「C」です。 DB2 UDB に適用可能なのはタイプ「F」の項目だけです。考慮されるのは、これらの項目です。

```
UDB_COLVALUE = S390_COLVALUE
UDB_VALCOUNT = S390_FrequencyF * S390_CARD
```

また、DB2 (OS/390 版) の SYSIBM.SYSCOLUMNS には列 SEQNO がありませんが、 DB2 UDB では必要です。そのため、db2look はその列を自動的に生成します。

サブエレメント統計

サブエレメント統計を収集して使用するため、オプションが提供されます。ブランクで区切られた一連のサブフィールドまたはサブエレメントのフォームに構造がある場合、データ内容に関する統計が列にあります。

たとえば、データベースに、行ごとに文書の記述がある表 DOCUMENTS が含まれ、DOCUMENTS には KEYWORDS という列があり、この列には、テキスト検索用に文書に関連するキーワードのリストが含まれているものとします。 KEYWORDS の値としては次のものがある可能性があります。

```
'database simulation analytical business intelligence'
'simulation model fruitfly reproduction temperature'
'forestry spruce soil erosion rainfall'
'forest temperature soil precipitation fire'
```

この例では、各列の値は 5 個のサブエレメントから構成され、それぞれのエレメントにブランクで区切られたワード (キーワード) があります。

% match_all 文字を使用した列で LIKE 述部を指定する照会の場合、次のようになります。

```
SELECT .... FROM DOCUMENTS WHERE KEYWORDS LIKE '%simulation%'
```

次の用語のような列のサブエレメント構造に関して、最適化プログラムが基本統計を認識することが有効な場合がよくあります。

SUB_COUNT
サブエレメントの平均数。

SUB_DELIM_LENGTH
サブエレメントを区切っている区切り文字の平均長。区切り文字は、ここでは 1 つ以上の連続するブランク文字です。

KEYWORDS 列の例では、SUB_COUNT は 5 で、SUB_DELIM_LENGTH は 1 となります。これは、区切り文字が 1 個のブランク文字であるためです。

システム管理者は、DB2_LIKE_VARCHAR レジストリー変数の拡張により、この統計の収集と使用の制御を行います。このレジストリー変数は、次の書式の述部を DB2 UDB 最適化プログラムが処理する方法に影響します。

```
COLUMN LIKE '%xxxxxx'
```

xxxxxx には任意のストリングが入ります。つまり、この場合は、LIKE 述部の検索値は % 文字で始まります。 (% 文字で終了しないこともあります。) 下記の ″wildcard LIKE predicates″ として述べられます。すべての述部に対して、最適化プログラムが述部に一致する行の数を見積もる必要があります。ワイルドカード LIKE 述部では、最適化プログラムは一致する COLUMN に連結する一連のエレメントの構造があると推定し、前後の % 文字を含まないストリングの長さに基づいたエレメントの長さを見積もります。新規構文は次の通りです。

```
db2set  DB2_LIKE_VARCHAR=[Y|N|S|num1][,Y|N|num2]

説明
- 最初の用語 (コンマの前まで) は、
   正の値のサブエレメント統計を持たない列に対してのみ、
   以下の意味になります。
   S                 アルゴリズムを DB2 バージョン 2 で使用されていたように使用する。
   N                 固定長サブエレメント・アルゴリズムを使用する。
   Y    (デフォルト) 可変長サブエレメント・アルゴリズムを使用する。
                     アルゴリズム・パラメーターに対してデフォルト値を使用する。
   num1              可変長サブエレメント・アルゴリズムを使用し、
                     アルゴリズム・パラメーターに num1 を使用する。

- 2 番目の用語 (コンマの次) は、次の意味を持ちます。
   N (デフォルト)    サブエレメント統計を収集あるいは使用しない。
   Y                 サブエレメント統計を収集する。
                     列が正のサブエレメント統計を持つ場合、それらの統計を
                     使用する可変長サブエレメント・アルゴリズムを、パラメーター
                     にデフォルト値を指定して使用します。
   num2              サブエレメント統計を収集する。
```

列が正のサブエレメント統計を持つ場合、それらの統計を使用する可変長サブエレメント・アルゴリズムを、パラメーターに num2 値を指定して使用します。

DB2_LIKE_VARCHAR が、最初の用語にのみ含まれていると、サブエレメント統計は収集されず、前に収集されていた統計は無視されます。 指定された値は、以前と同じように、ワイルドカード LIKE 述部の選択肢の計算方法に影響を与えます。次の例で説明します。

- 値が S の場合、最適化プログラムは DB2 バージョン 2 で使用されたアルゴリズムと同じアルゴリズムを使用します。これはサブエレメント・モデルを想定しません。
- 値が N の場合、最適化プログラムは、サブエレメント・モデルを想定するアルゴリズムを使用し、COLUMN が可変長として定義されていても、固定長であると推定します。
- 値が Y (デフォルト) または浮動小数点定数の場合、最適化プログラムは、サブエレメント・モデルを想定するアルゴリズムを使用し、COLUMN が可変長として定義されている場合、可変長を認識します。データからよりも、照会自体からのサブエレメントの推測も行います。このアルゴリズムは、エレメントが、% 文字で囲まれたストリングよりどれくらい長いかを指定するパラメーター (アルゴリズム・パラメーター) を含みます。
- 値が Y の場合、最適化プログラムはアルゴリズム・パラメーター用の 1.9 のデフォルト値を使用します。
- 値が浮動小数点定数の場合、最適化プログラムはアルゴリズム・パラメーター用に指定された値を使用します。この定数は 0 から 6.2 の範囲にあるようにしてください。

DB2_LIKE_VARCHAR の値に 2 つの用語が含まれ、2 番目が Y または浮動小数点定数であると、CHAR、VARCHAR、GRAPHIC、または VARGRAPHIC の 1 バイト文字セット・ストリング列にあるサブエレメント統計は、RUNSTATS 操作中に収集され、ワイルドカード LIKE 述部を含む照会のコンパイル中に使用されます。 最適化プログラムがサブエレメント・モデルを想定するアルゴリズムを使用して SUB_COUNT と SUB_DELIM_LENGTH 統計と、アルゴリズム・パラメーターも使用して述部の選択肢を計算します。 アルゴリズム・パラメーターは、次の例のように、推測されるアルゴリズムが指定されるのと同じように指定されます。

- 値が Y の場合、最適化プログラムはアルゴリズム・パラメーター用の 1.9 のデフォルト値を使用します。
- 値が浮動小数点定数の場合、最適化プログラムはアルゴリズム・パラメーター用に指定された値を使用します。この定数は 0 から 6.2 の範囲にあるようにしてください。

最適化プログラムが、サブエレメント統計が照会に含まれる列に収集されていないことをコンパイル中に検出した場合、最適化プログラムは「推論上の」サブエレメント・アルゴリズムを使用します。つまり、DB2_LIKE_VARCHAR の最初の用語のみが指定され

ている時に使用されるアルゴリズムです。 こうして、サブエレメント統計が最適化プログラムで使用されるには、 DB2_LIKE_VARCHAR の 2 番目の用語が RUNSTATS およびコンパイルの両方で設定されている必要があります。

サブエレメント統計の値は、 SYSIBM.SYSCOLUMNS を照会して表示されます。たとえば:

```
select substr(NAME,1,16), SUB_COUNT, SUB_DELIM_LENGTH
   from sysibm.syscolumns where tbname = 'DOCUMENTS'
```

SUB_COUNT と SUB_DELIM_LENGTH 列が SYSSTAT.COLUMNS 統計表示にはないため、更新できません。

注: このオプションが使用されると、RUNSTATS には時間がかかる可能性があります。たとえば、DETAILED および DISTRIBUTION オプションを使用していないと、RUNSTATS は 5 文字の列のある表では 15 から 40% 長く時間がかかる可能性があります。 DETAILED または DISTRIBUTION オプションを指定していると、オーバーヘッドの絶対量が同じでも、パーセンテージのオーバーヘッドは少なくなります。このオプションの使用を考える場合、照会パフォーマンスの改善に対して、このオーバーヘッドを評価する必要があります。

第6章 SQL コンパイラーに関する解説

SQL 照会をコンパイルすると、「最適の」アクセス・プランが実行されるか、またはそれがシステム・カタログに保管されるまでに、いくつかのステップが実行されます。

区分データベース環境の場合には、SQL コンパイラーが SQL 照会上で実行する作業はすべて、接続しているデータベース区画で行われます。コンパイルされた照会は、実行される前に、データベース内のすべてのデータベース区画に送信されます。

SQL コンパイラーによって実行されるステップについて、以下のトピックで説明します。

- SQL コンパイラーの概説
- SQL コンパイラーによる照会書き直し
- 操作のマージ
- 操作の移動
- 述部の変換
- データ・アクセス概念および最適化
- 区画内並列操作の最適化方式
- 自動要約表
- 連合データベース照会コンパイラー・フェーズ

コンパイラーによって作成される結果に影響する可能性のある、コンパイラー外部の要因については、以下の各部分でも説明しています。

- 43ページの『第3章 アプリケーションについての考慮事項』
- 91ページの『第4章 環境についての考慮事項』
- 113ページの『第5章 システム・カタログ統計』

SQL コンパイラーによって選択されるアクセス・プランを調べる方法については、 211ページの『第7章 SQL Explain 機能』で説明されています。

SQL コンパイラーの概説

SQL コンパイラーは、実行可能なアクセス・プランを作成する前に、いくつかのステップを実行します。それらのステップは、148ページの図12 に示されています。

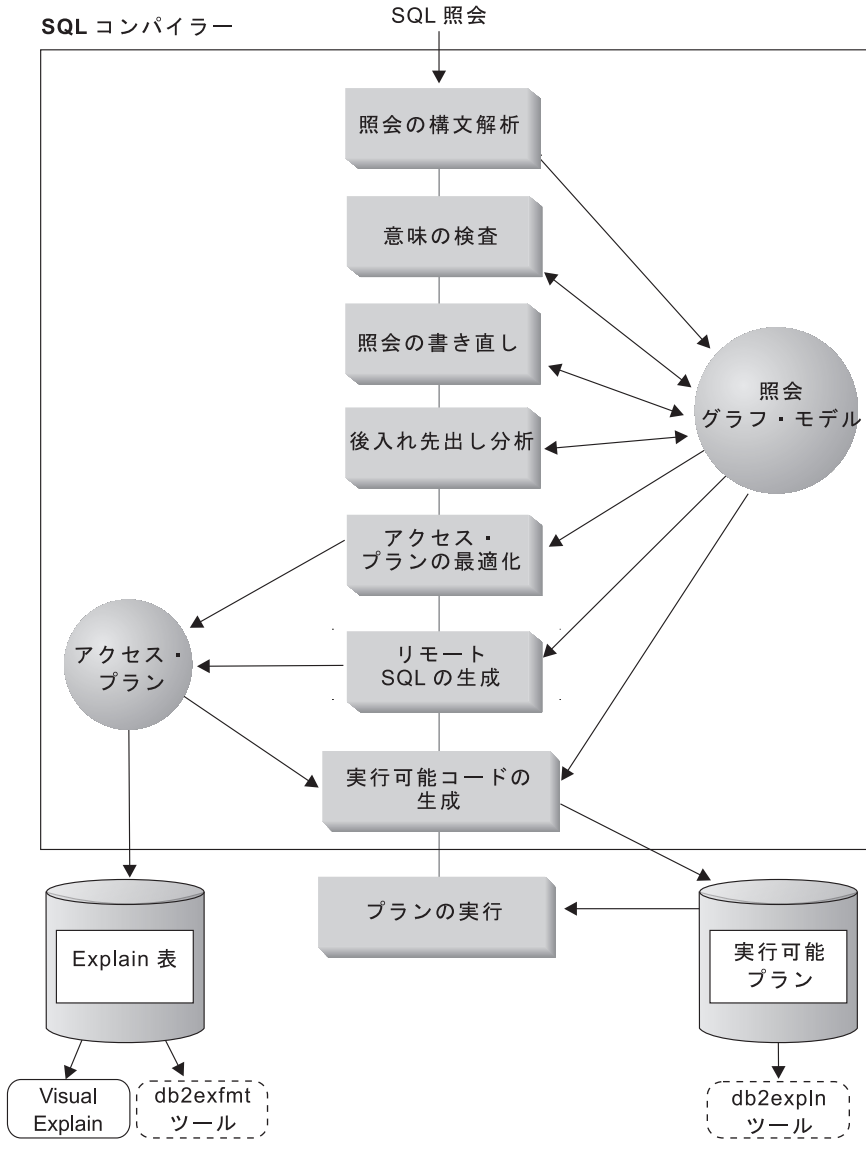


図 12. SQL コンパイラーによって実行されるステップ

この図は、**照会グラフ・モデル**が SQL コンパイラーの主要コンポーネントであることを示しています。*照会グラフ・モデル* とは、以下に説明するように、コンパイル処理全体を通じて照会を表示するのに使用される、メモリー内の内部的データベースです。

- **照会の構文解析**

  SQL コンパイラーの最初の作業は、SQL 照会を分析して構文の妥当性検査を行うことです。構文エラーが検出されると、SQL コンパイラーは処理を停止し、 SQL ステ

ートメントをコンパイルしようとしていたアプリケーションに対して、該当する SQL エラーが返されます。構文解析が完了すると、照会の内部表示が作成されます。

- **意味の検査**

  コンパイラーの 2 番目の作業は、ステートメント中に矛盾がないことを確認することです。この意味検査の 1 つの簡単な例は、 YEAR スカラー関数用に指定した列のデータ・タイプが日時データ・タイプであるか確認するものです。この 2 番目の段階でも、コンパイラーによって機能上の意味が照会グラフ・モデルに追加されます。それには、参照制約、表検査制約、トリガー、および視点などの結果が含まれます。

  照会グラフ・モデルには、照会ブロック、副照会、相関、派生表、式、データ・タイプ、データ・タイプ変換、コード・ページ変換、および区分化キーを含む、照会の意味すべてが含まれます。

- **照会の書き直し**

  SQL コンパイラーの第 3 フェーズでは、照会グラフ・モデルで提供されたグローバルな意味を使用して、照会をもっと最適化しやすい形に変形します。たとえば、コンパイラーは述部を移動することにより、適用されるレベルを変更し、照会のパフォーマンスを改善することがあります。このタイプの操作のことを、*一般述部後入れ先出し* といいます。詳細については、151ページの『SQL コンパイラーによる照会書き直し』を参照してください。

  区分データベース環境で作業している場合には、以下に関連するような一部の照会操作では、計算量が多くなります。

  – 集約
  – 行の再分配
  – 相関副照会

  相関副照会は、副照会の外側にある表の列への参照を含む副照会です。

  この環境では、一部の照会で、照会の書き直しの一部として非相関化が行われる場合があります。

  転送された照会は、照会グラフ・モデルに保管されます。

- **後入れ先出し分析 (連合データベース)**

  このステップの主な作業は、データ・ソースにおいて、ある操作をリモートで評価 (『後入れ先出し』) できるかどうかを、 DB2 最適化プログラムに通知することです。このタイプの後入れ先出しアクティビティーは、データ・ソース照会に特有のものであり、一般の述部の後入れ先出し操作を拡張するものとなります。

  連合データベース照会を実行するのでなければ、このステップは関係ありません。詳細については、197ページの『後入れ先出し分析』を参照してください。

- **アクセス・プランの最適化**

  SQL コンパイラーのうちの SQL 最適化プログラムの部分は、照会グラフ・モデルを入力として使用し、ユーザー要求を満たす多数の代替実行プランを生成します。ま

た、表、索引、列、および関数を使用する各代替プランの実行コストを見積もり、実行コストの見積もりが最も小さいプランを選択します。最適化プログラムは、照会グラフ・モデルを使用して照会の意味を分析したり、索引、基本表、派生表、副照会、相関、および再帰を含め、広範囲にわたる要因に関する情報を獲得したりします。

最適化プログラムの部分では、別のタイプの後入れ先出し操作である、*集約および分類* を考慮することもできます。この操作では、各操作の評価をデータ管理サービス・コンポーネントに知らせることにより、パフォーマンスを改善することができます。詳細については、190ページの『集約および分類の後入れ先出し操作』を参照してください。

さらに最適化プログラムでは、ページ・サイズの選択時に、別のサイズのバッファー・プールが存在するかどうかも考慮されます。環境に区分データベースが含まれる場合には、そのことも、対称マルチプロセッサー (SMP) 環境で照会内並列操作の可能性のために選択されたプランを拡張する機能と同様に、考慮されます。この情報は、最適化プログラムが照会にとって最適のアクセス・プランを選択するのに使用されます。詳細については、160ページの『データ・アクセス概念および最適化』を参照してください。

SQL コンパイラーのこのステップによる出力は、「アクセス・プラン」です。このアクセス・プランは、Explain 表にキャプチャーされる情報の基礎となるものです。この情報は、アクセス・プランを生成するのに使われ、 Explain スナップショットによってキャプチャーできます。 (Explain のトピックについては、211ページの『第7章 SQL Explain 機能』を参照してください。)

- **リモート SQL 生成 (連合データベース)**

  DB2 最適化プログラムで選択した最終プランは、リモート・データ・ソースに対して実行される一連のステップで構成されています。データ・ソースごとに実行されるこれらの操作では、リモート SQL 生成のステップにより、データ・ソース SQL ダイアレクトに基づく有効な SQL ステートメントが作成されます。

  連合データベース照会を実行するのでなければ、このステップは関係ありません。詳細については、204ページの『リモートでの SQL 生成とグローバル最適化』を参照してください。

- **「実行可能」コードの生成**

  SQL コンパイラーの最終ステップでは、アクセス・プラン と照会グラフ・モデルを使用して、照会の実行可能なアクセス・プラン、つまりセクションを作成します。このコード生成ステップでは、照会グラフ・モデルの情報を使用して、 1 つの照会で 1 回だけ計算するだけで済む式が繰り返し実行されないようにします。この最適化が行われる例としては、コード・ページ変換やホスト変数の使用を含むものがあげられます。

  静的 SQL のアクセス・プランに関する情報は、システム・カタログ表に保管されます。パッケージが実行されると、データベース・マネージャーはシステム・カタログ表に保管されている情報を使用して、データのアクセス方法を決め、照会結果を提供

します。 *db2expln* ツールによって使用されるのは、この情報です。 (Explain のトピックについては、211ページの『第7章 SQL Explain 機能』を参照してください。)

良いパフォーマンスを得たい照会において使用される表に対しては、 RUNSTATS を定期的に実行することをお勧めします。定期的に実行すると、最適化プログラムには、データの性質に関する適切な統計情報が提供されることになります。新規統計を利用するには、アプリケーションを再バインドする必要もあります。 RUNSTATS を実行しなかった (または最適化プログラムが、 RUNSTATS が空かほぼ空の表に対して実行されたと想定している) 場合には、最適化プログラムは、デフォルトを使用するか、あるいはディスク上に表を保管するのに使用されたファイル・ページ数 (FPAGES) に基づいて統計結果を引き出すように試みます。

SQL コンパイラーによる照会書き直し

SQL コンパイラーには、SQL ステートメントを最適化しやすい形式に変換し、その結果選択されたアクセス・パスを改善できるようにするための、照会書き直しの段階が含まれています。照会書き直しは、多くの副照会または結合を伴う照会を含む、非常に複雑な照会の場合、特に重要です。照会生成プログラム・ツールはしばしばこの種の非常に複雑な照会を作成します。

最適化クラスを変更すると、 SQL ステートメントに適用される照会書き直しの規則の数に影響することがあります (68ページの『最適化クラスの調整』を参照してください)。

Explain 機能または Visual Explain を使うと、照会書き直しの結果をいくつか表示させることができます。

SQL コンパイラーが実行する書き直しは、大きく分けて次の 3 つに分類されます。

- 操作のマージ
- 操作の移動
- 述部の変換

操作のマージ

SQL コンパイラーは照会を書き直すことによって照会操作をマージし、照会の SELECT 操作ができるだけ少なくなるよう試みます。以下の例に、SQL コンパイラーによってマージ可能な操作をいくつか示します。

- 例 - 視点のマージ

  SELECT ステートメントで視点を使用すると、表の結合順序が制限されることがあり、また余分な表結合が生成される可能性もあります。照会書き直し時に視点をマージするなら、これらの制限事項を除くことができます。

- 例 - 副照会から結合への変換

最適化プログラムが SELECT ステートメントの中に副照会を検出すると、表の処理順序の選択に関して制限を受ける場合があります。

- 例 - 余分な結合の除去

  照会書き直し時に、余分な結合を除去して、最適化対象の SELECT ステートメントをさらに簡略化することができます。

- 例 - 共用集約

  さまざまな関数を使用する場合には、照会の書き直しによって、行う必要のある計算の数を減らすことができます。

例 - 視点のマージ

EMPLOYEE 表に次の 2 つの視点でアクセスしているとしましょう。一方は高学歴の従業員を表示する視点で、他方は $35,000 以上の収入がある従業員を表示する視点です。

```
CREATE VIEW EMP_EDUCATION (EMPNO, FIRSTNME, LASTNAME, EDLEVEL) AS
SELECT EMPNO, FIRSTNME, LASTNAME, EDLEVEL
     FROM EMPLOYEE
 WHERE EDLEVEL > 17
CREATE VIEW EMP_SALARIES (EMPNO, FIRSTNAME, LASTNAME, SALARY) AS
SELECT EMPNO, FIRSTNME, LASTNAME, SALARY
     FROM EMPLOYEE
 WHERE SALARY > 35000
```

では、高学歴でかつ $35,000 以上の収入がある従業員をリストする照会を実行してみます。

```
SELECT E1.EMPNO, E1.FIRSTNME, E1.LASTNAME, E1.EDLEVEL, E2.SALARY
  FROM EMP_EDUCATION E1,
       EMP_SALARIES  E2
 WHERE E1.EMPNO = E2.EMPNO
```

照会書き直し時に、これらの 2 つの視点をマージすると、次の照会が作成されます。

```
SELECT E1.EMPNO, E1.FIRSTNME, E1.LASTNAME, E1.EDLEVEL, E2.SALARY
  FROM EMPLOYEE E1,
       EMPLOYEE E2
 WHERE E1.EMPNO = E2.EMPNO
   AND E1.EDLEVEL > 17
   AND E2.SALARY  > 35000
```

2 つの視点からの SELECT ステートメントをユーザー作成の SELECT ステートメントとマージすることによって、最適化プログラムはより多くのアクセス・プランの選択肢の中から選択できます。さらに、マージした 2 つの視点の使う基本表が同じである場合、 153ページの『例 - 余分な結合の除去』での説明にしたがって追加の書き直しを実行できます。

例 - 副照会から結合への変換

SQL コンパイラーは、次のような副照会を含む照会がある場合、

```
SELECT EMPNO, FIRSTNME, LASTNAME, PHONENO
     FROM EMPLOYEE
WHERE WORKDEPT IN
      (SELECT DEPTNO
  FROM DEPARTMENT
       WHERE DEPTNAME = 'OPERATIONS')
```

これを次の形式の結合照会に変換します。

```
SELECT DISTINCT EMPNO, FIRSTNME, LASTNAME, PHONENO
  FROM EMPLOYEE EMP,
       DEPARTMENT DEPT
 WHERE EMP.WORKDEPT = DEPT.DEPTNO
   AND DEPT.DEPTNAME = 'OPERATIONS'
```

一般に、結合は副照会を実行するよりはるかに効率的です。

例 - 余分な結合の除去

作成される、または生成される照会には、しばしば不要な結合が含まれています。 152 ページの『例 - 視点のマージ』に記述されている照会書き直し段階で、次のような照会が生成される可能性もあります。

```
SELECT E1.EMPNO, E1.FIRSTNME, E1.LASTNAME, E1.EDLEVEL, E2.SALARY
  FROM EMPLOYEE E1,
       EMPLOYEE E2
 WHERE E1.EMPNO = E2.EMPNO
   AND E1.EDLEVEL > 17
   AND E2.SALARY  > 35000
```

この照会では、SQL コンパイラーは結合を除去して、照会を次のように簡略化することができます。

```
SELECT EMPNO, FIRSTNME, LASTNAME, EDLEVEL, SALARY
     FROM EMPLOYEE
 WHERE EDLEVEL > 17
   AND SALARY  > 35000
```

次の例では、部門番号の EMPLOYEE サンプル表と DEPARTMENT サンプル表との間に、参照制約が存在すると想定しています。まず視点を作成します。

```
CREATE VIEW PEPLVIEW
   AS SELECT FIRSTNME, LASTNAME, SALARY, DEPTNO, DEPTNAME, MGRNO
         FROM EMPLOYEE E DEPARTMENT D
         WHERE E.WORKDEPT = D.DEPTNO
```

それから、次のような照会を作成します。

```
SELECT LASTNAME, SALARY
   FROM PEPLVIEW
```

この照会は、次のようになります。

```
SELECT LASTNAME, SALARY
     FROM EMPLOYEE
   WHERE WORKDEPT NOT NULL
```

この状態では、照会を書き直せることが分かっていても、基礎表へのアクセス権がないため、書き直すことはできません。持っているのは、視点へのアクセス権だけです (上記参照)。したがって、このタイプの最適化は、データベース・マネージャー内で実行する必要があります。

次のような場合、参照保全結合は冗長になる可能性があります。

- 視点が結合で定義される
- 照会が自動的に生成される

  たとえば、照会マネージャーには自動化されたツールがあるため、最適化された照会を書き込むことができません。

例 - 共用集約

照会の中で複数の関数を使用すると、複数の計算が生成されますが、これは場合によってはかなり時間を要します。照会内で行う計算の数を減らすことによって、計画を改善することができます。たとえば、以下のような複数の関数を使用する照会があるとします。

```
SELECT SUM(SALARY+BONUS+COMM) AS OSUM,
  AVG(SALARY+BONUS+COMM) AS OAVG,
  COUNT(*) AS OCOUNT
FROM EMPLOYEE;
```

SQL コンパイラーは、この照会を次のように変換します。

```
SELECT OSUM,
  OSUM/OCOUNT
  OCOUNT
FROM (SELECT SUM(SALARY+BONUS+COMM) AS OSUM,
  COUNT(*) AS OCOUNT
FROM EMPLOYEE) AS SHARED_AGG;
```

この書き直しによって、照会の関数は、2 つの SUM と 2 つの COUNT から 1 つの SUM と 1 つの COUNT に減らされます。

操作の移動

SQL コンパイラーは、照会を最小限の操作数と述部数で構成しようとするため、照会を書き直して照会操作を移動します。以下の例に、SQL コンパイラーによって移動可能な操作をいくつか示します。

- 例 - DISTINCT の除去

  照会書き直し時に最適化プログラムで DISTINCT 操作が実行される場所を移動すると、その操作のコストが少なくなることがあります。この例では、DISTINCT 操作は完全に除去されます。

- 例 - 一般的な述部後入れ先出し

  照会書き直し時に、述部を適用する順序を変更することによって、できるだけ早い機会に、もっと多くの選択述部が適用されるようにすることができます。

- 例 - 非相関化

  区分データベース環境にいる場合、データベース区画間での結果セットの移動は高コストになります。他のデータベース区画にブロードキャストする必要があるもののサイズ、またはブロードキャストの数 (あるいは、その両方) を減らすことも、照会の書き直し時の目標の 1 つです。

例 - DISTINCT の除去

EMPNO 列を EMPLOYEE 表の基本キーとして定義した場合、次の照会は、

```
SELECT DISTINCT EMPNO, FIRSTNME, LASTNAME
     FROM EMPLOYEE
```

DISTINCT 文節を除去することで書き直されます。

```
SELECT EMPNO, FIRSTNME, LASTNAME
     FROM EMPLOYEE
```

上記の例では、基本キーが選択されているため、 SQL コンパイラーには返される各行がすでに固有であることがわかっています。この場合、DISTINCT キーワードは不要です。照会を書き直さないとすると、最適化プログラムは必要な処理 (分類など) を含む計画を作成して、列を一意にすることになります。

例 - 一般的な述部後入れ先出し

述部が通常適用されるレベルを変更すると、パフォーマンスが向上する場合があります。たとえば、部署 『D11』 内の従業員全員のリストを示す以下のような視点があるとします。

```
CREATE VIEW D11_EMPLOYEE
 (EMPNO, FIRSTNME, LASTNAME, PHONENO, SALARY, BONUS, COMM)
AS SELECT EMPNO, FIRSTNME, LASTNAME, PHONENO, SALARY, BONUS, COMM
     FROM EMPLOYEE
    WHERE WORKDEPT = 'D11'
```

さらに、以下の照会があるとします。

```
SELECT FIRSTNME, PHONENO
  FROM D11_EMPLOYEE
 WHERE LASTNAME = 'BROWN'
```

コンパイラーの照会書き直し段階で、述部 `LASTNAME = 'BROWN'` が視点 D11_EMPLOYEE にプッシュされます。これにより、述部が早い時期におそらく効率的に適用されるようになります。この例で実行可能な実際の照会は、次のようになります。

```
SELECT FIRSTNME, PHONENO
     FROM EMPLOYEE
 WHERE LASTNAME = 'BROWN'
   AND WORKDEPT = 'D11'
```

述部の後入れ先出しは、視点に限定されません。述部が後入れ先出しされる可能性のあるこれ以外の状況には、 UNION、GROUP BY、派生表 (ネストされた表式や共通表式) があります。

例 - 非相関化

区分データベース環境では、 SQL コンパイラーは次のような照会の書き直しを行う場合があります。

次の例では、プログラミング・プロジェクトに従事している従業員のうち給与が低い人をすべて検索します。

```
SELECT P.PROJNO, E.EMPNO, E.LASTNAME, E.FIRSTNAME,
       E.SALARY+E.BONUS+E.COMM AS COMPENSATION
  FROM EMPLOYEE E, PROJECT P
WHERE P.EMPNO = E.EMPNO
  AND P.PROJNAME LIKE '%PROGRAMMING%'
  AND E.SALARY+E.BONUS+E.COMM <
     (SELECT AVG(E1.SALARY+E1.BONUS+E1.COMM)
        FROM EMPLOYEE E1, PROJECT P1
        WHERE P1.PROJNAME LIKE '%PROGRAMMING%'
          AND P1.PROJNO = A.PROJNO
          AND E1.EMPNO = P1.EMPNO)
```

この照会は相関しており、また PROJECT と EMPLOYEE の両方が PROJNO 上に区分化されていないので、各プロジェクトが各データベース区画にブロードキャストされます。さらに、この副照会の評価を何回も行わなければなりません。

SQL コンパイラーは、照会を以下に示すように書き直します。

- プログラミング・プロジェクトに従事している従業員の個別リストを決定して、それを DIST_PROJS とします。これが個別リストでなければならないのは、各プロジェクトに対して行われる集約が 1 度だけとなるようにするためです。

  ```
  WITH DIST_PROJS(PROJNO, EMPNO) AS
  (SELECT DISTINCT PROJNO, EMPNO
   FROM PROJECT P1
   WHERE P1.PROJNAME LIKE '%PROGRAMMING%')
  ```

- プログラミング・プロジェクトに従事している従業員の個別リストを使用して、これを従業員表に結合し、プロジェクトごとの平均報酬 AVG_PER_PROJ を求めます。

  ```
  AVG_PER_PROJ(PROJNO, AVG_COMP) AS
  (SELECT P2.PROJNO, AVG(E1.SALARY+E1.BONUS+E1.COMM)
   FROM EMPLOYEE E1, DIST_PROJS P2
   WHERE E1.EMPNO = P2.EMPNO
   GROUP BY P2.PROJNO)
  ```

- 最終的に、新しい照会は次のようになります。

```
SELECT P.PROJNO, E.EMPNO, E.LASTNAME, E.FIRSTNAME,
       E.SALARY+E.BONUS+E.COMM AS COMPENSATION
 FROM PROJECT P, EMPLOYEE E, AVG_PER_PROG A
WHERE P.EMPNO = E.EMPNO
  AND P.PROJNAME LIKE '%PROGRAMMING%'
  AND P.PROJNO = A.PROJNO
  AND E.SALARY+E.BONUS+E.COMM < A.AVG_COMP
```

書き直された SQL 照会では、プロジェクトごとの AVG_COMP (AVG_PRE_PROJ) を計算し、その結果を EMPLOYEE 表を含むデータベース区画すべてにブロードキャストすることができます。

述部の変換

SQL コンパイラーは照会を書き直して、特定の照会に関して既存の述部をより最適化された述部に変換します。以下の例に、SQL コンパイラーが変換できる述部をいくつか示します。

- 例 - 暗黙の述部の追加

  照会の書き直し時に述部をその照会に追加することによって、最適化プログラムが照会の最適アクセス・プランを選択するときに、追加の表結合についても検討できるようになります。

- 例 - OR から IN への変換

  照会書き直し時に、OR 述部を IN 述部に変換すると、より効率的なアクセス・プランを選択できるようになる場合があります。また、IN 述部を OR 述部に変換することが一層効率的なアクセス・プランの選択を可能にするのであれば、 SQL コンパイラーでそのような変換をすることもできます。

例 - 暗黙の述部の追加

以下の照会は、『E01』 に報告する部門のマネージャー、およびそれらのマネージャーが担当するプロジェクトのリストを生成するものです。

```
SELECT DEPT.DEPTNAME DEPT.MGRNO, EMP.LASTNAME, PROJ.PROJNAME
  FROM DEPARTMENT DEPT,
       EMPLOYEE   EMP,
       PROJECT    PROJ
 WHERE DEPT.ADMRDEPT = 'E01'
   AND DEPT.MGRNO = EMP.EMPNO
   AND EMP.EMPNO  = PROJ.RESPEMP
```

照会書き直しにより、以下の暗黙の述部が追加されます。

```
DEPT.MGRNO = PROJ.RESPEMP
```

この書き直しの結果、最適化プログラムがこの照会に最適のアクセス・プランを選択するとき、考慮できる結合の種類が増えることになります。

前述の述部移動のほかにも、照会書き直しでは、等号述部によって暗黙のうちに示される移動に基づいて、さらに別のローカル述部が導出されます。たとえば、以下の照会は、部門 (部門番号が 『E00』 より大きいもの) の名前、およびその部門で働く従業員の名前のリストを作成するものです。

```
SELECT EMPNO, LASTNAME, FIRSTNAME, DEPTNO, DEPTNAME
  FROM EMPLOYEE EMP,
       DEPARTMENT DEPT
 WHERE EMP.WORKDEPT = DEPT.DEPTNO
   AND DEPT.DEPTNO > 'E00'
```

この照会では、照会書き直し段階で、以下の暗黙の述部が追加されます。

```
EMP.WORKDEPT > 'E00'
```

この書き直しの結果として、最適化プログラムは結合する行の数を減らします。

例 - OR から IN への変換

OR 文節が、次の例のように同じ列にある 2 つ以上の単純等価述部を結合する場合を考えてみましょう。

```
SELECT *
     FROM EMPLOYEE
 WHERE DEPTNO = 'D11'
    OR DEPTNO = 'D21'
    OR DEPTNO = 'E21'
```

DEPTNO 列に索引が存在しない場合、OR 文節を次のような IN 述部に変換すると、照会をより効率的に処理できるようになります。

```
SELECT *
     FROM EMPLOYEE
 WHERE DEPTNO IN ('D11', 'D21', 'E21')
```

注: 場合によっては、データベース・マネージャーは IN 述部を一連の OR 述部に変換して、索引 OR 処理を実行できるようにすることがあります。索引 OR 処理については、 167ページの『複数の索引アクセス』を参照してください。

列の相関に関する説明

2 つの表を結合した結合述部が複数ある結合で構成される照会を含んだアプリケーションがあるかもしれません。これは複雑に聞こえますが、表の間で、類似した、関連のある列の間の関係を判別しようとする場合は、このような状況は珍しいことではありません。

たとえば、さまざまな色、伸縮性、品質の原料から製品が作成されます。完成品は、その原料と同じ色、伸縮性をしています。この場合、以下の照会を発行します。

```
SELECT PRODUCT.NAME, RAWMATERIAL.QUALITY FROM PRODUCT, RAWMATERIAL
   WHERE PRODUCT.COLOR       =  RAWMATERIAL.COLOR
     AND PRODUCT.ELASTICITY  =  RAWMATERIAL.ELASTICITY
```

この照会は、すべての製品の名前と原料の品質を戻します。以下の 2 つの結合述部があります。

```
PRODUCT.COLOR       =  RAWMATERIAL.COLOR
PRODUCT.ELASTICITY  =  RAWMATERIAL.ELASTICITY
```

DB2 UDB 最適化プログラムがこの照会を実行するプランを選択すると、上記の 2 つの述部の選択可能性をそれぞれ計算します。また 2 つの述部は独立していることが前提になっています。つまり、それぞれの色ごとにすべてのレベルの伸縮性が考慮され、また逆にそれぞれのレベルの伸縮性ごとにすべての色が考慮されます。個々の表にある色の数と伸縮性のレベルの数に関する統計を使用して、上記の述部の組みに関するあらゆる選択可能性を計算します。この結果に基づいて、たとえばネストしたループ結合とマージ結合のどちらを優先するかを選択します。

しかしながら、2 つの述部が独立していない場合もあります。たとえば、伸縮性の高い原料には 2、3 種類の色しかなく、伸縮性の非常に低い原料には他の (つまり伸縮性の高い原料の色とは別の) 2、 3 色しかない場合も考えられます。この場合、2 つの述部を組み合わせた場合の選択可能性は少なくなる (除去される行が少なくなる) ので、照会により戻される行は多くなります。このことを確認するには、極端なケースとしてそれぞれの色ごとに 1 つのレベルの伸縮性しかない場合やその逆の場合を考えてみてください。この場合、一方の述部は他方の述部によって暗黙指定されるので完全に省略することもできます。この場合、最適化プログラムによって選択されたプランは最善とはいえないこともあります。たとえば、ネストしたループ結合のプランが選択された場合でもマージ結合の方が高速になる場合があります。

他のデータベース製品の場合、データベース管理者はカタログ中の統計を更新して一方の述部の選択可能性を少なくすることによりこのパフォーマンス上の問題を解決していましたが、この方法は他の照会に思わぬ副次作用が及ぶことがありました。

次の場合に、DB2 UDB の最適化プログラムは、結合述部の相互関係の検出と補正を試みます。

1. 相関列、つまり相関述部中にある表列に関する固有索引を定義する。
2. レジストリー変数 DB2_CORRELATED_PREDICATES を 『NO』 に設定しない。

上記の例では、以下のいずれかを含む固有索引を定義する必要があります。

```
PRODUCT.COLOR, PRODUCT.ELASTICITY
```

または

```
RAWMATERIAL.COLOR, RAWMATERIAL.ELASTICITY
```

またはこの両方

相関が検出されるようにするには、この索引の非組み込み列は相関列でなければならず、またこれ以外の列はありません。任意でこの索引に組み込み列を入れることもできます。

通常は、結合述部には 2 つ以上の相関列があり得るので、固有索引の定義にすべての列が関係しているか確認する必要があります。

1 つの表中の相関列がその表の基本キーになっている場合もよくあります。基本キーは常に固有なので、相関列が基本キーになっている場合はそれとは別に固有索引を定義する必要はありません。

その後、表の統計が最新のものになっていることを確認し、何らかの理由で (たとえば、最適化プログラムを制御しようとして) 変更が加えられて真の値でなくなっていたりしないか確認してください。

最適化プログラムは固有索引の統計中の FIRSTnKEYCARD および FULLKEYCARD 情報を使用して相関事例を検出し、相関述部の組み合わせによる選択可能性を動的に調整するので、結合サイズとコストに関する見積もりの正確さが向上します。

結合述部相関に加えて、最適化プログラムは、タイプ COL = 『定数』の単純な等価述部に関する相関についても考慮します。たとえば、異なるタイプの車の表を考慮します。それぞれに、MAKE (製造メーカー)、MODEL、 STYLE (セダン、ステーション・ワゴン、スポーツ・カー)、 YEAR、および COLOR があります。 COLOR に関する述部は、 MAKE、MODEL、STYLE、または YEAR に関する述部から独立していると思われます。ほぼすべての製造メーカーは、毎年、それぞれのモデルおよびスタイルで標準色を使用できるようにするからです。しかし、特定の名前を持つモデルを製造するのは単一の車メーカーだけなので、述部 MAKE および MODEL が独立していないことは明らかです。複数の車メーカーによって同じモデル名が使用されることは非常にまれで、明らかに車メーカーもそれは望みません。 2 つの列 MAKE および MODEL の索引が存在する場合、最適化プログラムはその索引からの統計を使用して、異なる値を結合した数値を判別し、 2 つの列間の相関の選択可能性やカーディナリティーの見積もりを調整します。結合述部でない述部の場合、最適化プログラムが調整を行うための固有索引を持つ必要はありません。

データ・アクセス概念および最適化

SQL ステートメントをコンパイルするとき、 SQL 最適化プログラムは別の方法で要求を満たす場合の実行コストを見積もります。この評価に基づいて、最適化プログラムは最適のアクセス・プランと思われるものを選択します。アクセス・プラン とは、 SQL ステートメントの解決に必要な操作の順序を指定するものです。アプリケーション・プログラムがバインドされると、パッケージ が作成されます。このパッケージには、そのアプリケーション・プログラムの静的 SQL ステートメント全部に関するアクセス・プランが入れられます。動的 SQL のアクセス・プランは、アプリケーション実行時に作成されます。

表内のデータにアクセスする方法は 2 通りあります。表を直接読み取る (関係走査) か、または最初に表の索引にアクセスする (索引走査) かのいずれかです。

関係走査 が行われるのは、データベース・マネージャーが表の各行に順次アクセスするときです。索引走査がどのように機能するかについては、 『索引走査の概念』を参照してください。また、それぞれのタイプの走査がどのような条件で使用されるかについては、 170ページの『関係走査と索引走査』を参照してください。

以下のトピックでは、表のデータにアクセスし、照会の結果を生成するためのアクセス・プランで使用可能なその他の方法について説明します。

- 170ページの『述部の用語』
- 172ページの『結合の概念』
- 182ページの『区分データベースにおける結合方式』
- 189ページの『最適化プログラムでの分類の影響』

その他の関連トピック:

- 68ページの『最適化クラスの調整』では、 SQL コンパイラーによって評価される代替アクセス・プランの数の制御について説明されています。
- 211ページの『第7章 SQL Explain 機能』では、 SQL コンパイラーによって選択されるアクセス・プランについての情報を得る方法を説明しています。

索引走査の概念

索引走査 が行われるのは、データベース・マネージャーが以下のいずれかまたは全部を行うために索引にアクセスするときです。

- 基本表にアクセスする前に、 (索引の特定の範囲内にある行を走査することによって) 条件限定の行の集合の範囲を狭める。索引の*走査範囲* (走査の開始点と停止点) は、照会において索引列と比較する値によって判別されます。
- 出力を並べ替える。
- 要求されたデータを完全に取り出す。要求されたデータがすべて索引内にある場合、基本表にはアクセスしません。これは、*索引のみのアクセス* と呼ばれます。

索引に対する走査は、定義した方向とは逆方向に実行することもできます。詳細は、*SQL 解説書* に載せられている CREATE INDEX ステートメントの ALLOW REVERSE SCANS オプションを参照してください。

ここでは、以下の追加トピックを示します。

- 索引の構造
- 範囲を区切る索引走査
- データ並び替えのための索引走査
- 索引のみのアクセス

- 複数の索引アクセス
- クラスター索引
- 索引ページの事前取り出し

索引の構造

データベース・マネージャーは、索引の保管に B+ ツリー構造を使用します。 B+ ツリーには、1 つまたは複数のレベルがあり、それについては次の図に示します (ここで、RID は行 ID を意味します)。

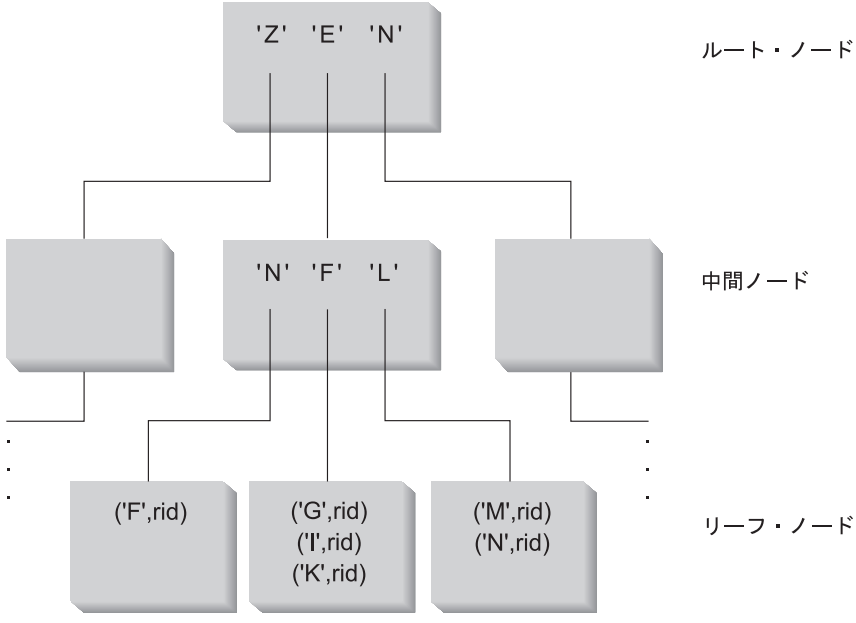


図 13. B+ ツリー構造

最上レベルは*ルート・ノード* と呼ばれます。最下レベルは、*リーフ・ノード* で構成され、ここには実際の索引キー値と、表の実際の行に対するポインターが保管されます。ルート・ノードとリーフ・ノードの間のレベルは、*中間ノード* と呼ばれます。

特定の索引キー値を検出するとき、索引マネージャーは、ルート・ノードから開始して、その索引ツリーを検索します。ルートには、その次のレベルの各ノードごとに 1 つずつキーが含まれています。それらの各キーの値は、その次のレベルの対応するノードに存在する最大のキー値です。たとえば、図13 に示すように、索引に 3 つのレベルがある場合に索引キー値を検出するには、索引マネージャーは、ルート・ノードから、目的のキー以上のキー値のうちの最初のものを探索します。このルート・ノード・キーは、多くの場合、特定の中間ノードを指すものです。その中間ノードで同じ手順に従い、どのリーフ・ノードに進むべきかを決めます。リーフ・ノードで最終索引キーが検出されます。 図13 の場合、検索するキーは 『I』 です。ルート・ノードの中で、

『I』 以上の最初のキーは 『N』 です。これは、その次レベルの中間ノードを指すものです。その中間ノードの中で、『I』 以上の最初のキーは 『L』 です。これは、『I』 の索引キーとそれに対応する行 ID (基本表の中の対応する行の行 ID) の含まれる特定のリーフ・ノードを指すものです。

注: リーフ・ノード・レベルでは、前のリーフ・ポインターが存在する場合があります。ツリーをくまなく検索することによって索引内に特定のキー値を見つけると、索引マネージャーはいずれかの方向にあるリーフ・ノードを走査して特定範囲の値を検索できるため、前のリーフ・ポインターがあることが非常に役立つことがあります。いずれかの方向に走査を実行するこの機能は、 ALLOW REVERSE SCANS パラメーターを使用して索引が作成された場合にのみ使用可能です。

詳細は、*SQL 解説書* に載せられている CREATE INDEX ステートメントのオプションを参照してください。

範囲を区切る索引走査

ある特定の照会に索引を使用できるかどうかを決定するとき、最適化プログラムは索引の各列を最初の列から順に評価し、次のものを満たすことができるかどうかを調べます。

- ステートメントの WHERE 文節内の EQUAL 述部。
- WHERE 文節内のその他の述部。

述部 は、WHERE 文節内で比較操作を明示する、または暗黙のうちに示す探索条件の 1 つの要素です。索引走査の範囲を区切るのに使用できる述部は、次の条件が真となる索引列の関係する述部です。

- 索引列は、定数、ホスト変数、評価結果が定数になる式、またはキーワードに対して等しいかどうかがテストされる。
- 索引列に対するテストは、『IS NULL』 または 『IS NOT NULL』。
- テストは、基本的な副照会 (つまり、ANY、ALL、または SOME が含まれていない) についての等価性のテストであり、その副照会には即時親照会ブロック (つまり、この副照会が副選択となる SELECT) への相関列参照がない。
- テストは、後述の条件を満たす不等号述部である。

たとえば、索引が次のように定義されているとします。

```
INDEX IX1:  NAME    ASC,
            DEPT    ASC,
            MGR     DESC,
            SALARY  DESC,
            YEARS   ASC
```

索引 IX1 の走査範囲を区切るのには、次の述部を使用することができます。

```
WHERE  NAME  = :hv1
 AND   DEPT  = :hv2
```

または

```
WHERE  MGR  = :hv1
  AND  NAME = :hv2
  AND  DEPT = :hv3
```

2 番目の例では、WHERE 述部を、索引内でキー列の現れる順序と同じ順序で指定する必要はありません。また、例の中ではホスト変数が使用されていますが、パラメーター・マーカー、式、または定数には同じ効果があります。

CREATE INDEX ステートメント上で ALLOW REVERSE SCANS パラメーターを使用して作成された単一索引は、前方向にも後方向にも走査することができます。つまり、それらの索引では、索引の作成時に定義した方向への走査と、反対方向 (逆方向) への走査がサポートされています。ステートメントは、以下のようになります。

```
CREATE INDEX iname ON tname (cname DESC) ALLOW REVERSE SCANS
```

この場合、索引 (iname) は、 cname の DESCending 値に基づいて形成されます。逆方向走査を許可することにより、列に対する索引が降順で走査するように定義されているとしても、昇順で走査することができます。索引を実際に両方向で使うことは、ユーザーが制御するのではなく、アクセス・プランの作成および考慮時に最適化プログラムによって制御されます。

以下の WHERE 文節では、NAME および DEPT の述部だけが索引走査の範囲を区切るのに使用され、 SALARY または YEARS の述部は使用されません。

```
WHERE  NAME   = :hv1
 AND   DEPT   = :hv2
  AND  SALARY = :hv4
  AND  YEARS  = :hv5
```

これは、これらの列を最初の 2 つの索引キー列から分離するキー列 (MGR) があるので、並べ替えがオフになるためです。しかし、いったん NAME = :hv1 および DEPT = :hv2 の述部によって範囲が決まったなら、残りの述部を残りの索引キー列に対して評価できるようになります。

前述の等号述部以外にも、特定の不等号述部を使用して、索引走査の範囲を区切ることができます。次に、2 種類の不等号述部 (狭義不等号と広義不等号) について説明します。

狭義不等号述部: 範囲を区切る述部として使用できる狭義不等号演算子は、> と < です。

索引走査の範囲を区切る場合、狭義不等号述部では 1 つの列だけが考慮されます。以下の例では、NAME 列と DEPT 列に対して述部を使用して範囲を区切ることはできますが、 MGR 列に対しては述部は使用できません。

```
WHERE  NAME  = :hv1
  AND  DEPT   > :hv2
  AND  DEPT   < :hv3
  AND  MGR    < :hv4
```

広義不等号述部: 以下は、範囲を区切る述部として使用できる広義不等号演算子です。

- >= と <=
- BETWEEN
- LIKE

索引走査の範囲を区切る場合、広義不等号述部については複数の列が考慮されます。次の例では、述部をすべて使用して索引走査の範囲を区切ることができます。

```
WHERE  NAME  = :hv1
  AND  DEPT  >= :hv2
  AND  DEPT  <= :hv3
  AND  MGR   <= :hv4
```

この例についてさらに考慮するために、:hv2 = 404、 :hv3 = 406、および :hv4 = 12345 を想定してください。データベース・マネージャーは部門 404 と 405 のすべての索引を走査しますが、 12345 より多い従業員数 (MGR 列) を持つ管理者を最初に検出した時点で、部門 406 の走査を停止します。

詳細については、171ページの『範囲区切り述部と索引検索引き数述部』を参照してください。

データ並び替えのための索引走査

照会に並び替えが関係している場合、並び替える列が、最初の索引キー列から始まって連続して索引内に現れる場合は、データを並び替えるのに索引を使用することができます。 (並び替えまたは分類は、 ORDER BY、 DISTINCT、 GROUP BY、『= ANY』副照会、 『> ALL』 副照会、 『< ALL』 副照会、 INTERSECT または EXCEPT、UNION などの操作の結果得られます。) ただし、索引キー列が 『定数値』 (つまり評価結果が定数の式) に等しいかどうかを比較する場合は例外です。この場合、並べ替えに使う列が最初の索引キー列以外のものである可能性があります。たとえば、

```
WHERE NAME = 'JONES'
  AND DEPT = 'D93'
ORDER BY MGR
```

という照会では、NAME と DEPT は必ず同じ値となり、結果としてその列については分類されることになるため、行を並び替えるのに索引を使用できます。つまり、これは先行する WHERE 文節と ORDER BY 文節とが次のようになっているのと同じことになります。

```
WHERE NAME = 'JONES'
  AND DEPT = 'D93'
ORDER BY NAME, DEPT, MGR
```

このほかにも固有索引を使用することによって、順序要件を切り捨てることもできます。たとえば、次のような索引の定義と、ORDER BY 文節とがあるとします。

```
UNIQUE INDEX IX0:  PROJNO  ASC
SELECT PROJNO, PROJNAME, DEPTNO
   FROM PROJECT
ORDER BY PROJNO, PROJNAME
```

IX0 索引により PROJNO が固有になるため、PROJNAME 列での分類は不要です。この固有性により、各 PROJNO 値には、PROJNAME 値が 1 つしかないことになります。

索引のみのアクセス

場合によっては、表にアクセスせずに、索引からすべての必須データを検索できることがあります。これは、*索引のみの* アクセスと呼ばれます。

以下の索引定義を使って、索引のみのアクセスについて説明します。

```
INDEX IX1:  NAME    ASC,
            DEPT    ASC,
            MGR     DESC,
            SALARY  DESC,
            YEARS   ASC
```

基本表を読み取らずに、索引にアクセスするだけで、以下の照会を実行することができます。

```
SELECT NAME, DEPT, MGR, SALARY
     FROM EMPLOYEE
  WHERE NAME = 'SMITH'
```

その他の場合は、索引内に現れない列が含まれていることもあります。それらの列のデータを得るには、基本表の行を読み取らなければなりません。たとえば、上記の IX1 索引で、以下の照会では、 PHONENO および HIREDATE 列データを取得するには、基本表にアクセスする必要があります。

```
SELECT NAME, DEPT, MGR, SALARY, PHONENO, HIREDATE
     FROM EMPLOYEE
  WHERE NAME = 'SMITH'
```

組み込み列のある固有索引を作成し、索引だけに基づくアクセス試行数を多くすると、データ検索のパフォーマンスを改善できます。

以下の索引定義を使って、組み込み列の使用法について説明します。

```
CREATE UNIQUE INDEX IX1 ON EMPLOYEE
   (NAME ASC)
    INCLUDE (DEPT, MGR, SALARY, YEARS)
```

この場合、NAME 列を強制的に固有なものにし、しかも DEPT、 MGR、 SALARY、および YEARS 列のデータも保管して保守する固有索引が作成されます。

基本表を読み取らずに、索引にアクセスするだけで、以下の照会を実行することができます。

```
SELECT NAME, DEPT, MGR, SALARY
     FROM EMPLOYEE
   WHERE NAME='SMITH'
```

複数の索引アクセス

前述のすべての例では、結果を引き出すのに単一索引走査が実行されました。 WHERE 文節の述部を満たすために、最適化プログラムでは複数索引の走査を選択することがあります。たとえば、以下の 2 つの索引定義があるとします。

```
INDEX IX2:  DEPT    ASC
INDEX IX3:  JOB     ASC,
            YEARS   ASC
```

この 2 つの索引を使って、以下の述部が解決されることになります。

```
WHERE DEPT = :hv1
  OR (JOB   = :hv2
  AND YEARS >= :hv3)
```

この例では、索引 IX2 を走査することによって、 `DEPT = :hv1` 述部を満たす行 ID (RID) のリストが作成されます。また、索引 IX3 を走査することによって、 `JOB = :hv2 AND YEARS >= :hv3` 述部を満たす RID のリストが作成されます。これらの 2 つの RID リストを組み合わせると、表にアクセスする前に重複が除かれます。これは、*索引 OR 操作* と呼ばれます。

索引 OR 操作は、次の例のように IN 式を使用している述部にも使用できます。

```
WHERE DEPT IN (:hv1, :hv2, :hv3)
```

索引 OR 操作の目的は、重複した RID を除去することですが、*索引 AND 結合* の目的は、共通 RID を検索することです。索引 AND 結合は、同じ表内の対応する列に複数の索引があり、さらに複数の AND 述部を使用する照会をこの表に対して実行するアプリケーションにおいて行われます。 その種の照会において、索引付けされた列ごとに複数の索引走査を行うと、ビットマップを作成するためにハッシュされた値が生成されます。 2 番目のビットマップは 1 番目のビットマップをプローブするのに使用され、最終的に戻されるデータ・セットを作成するために取り出される修飾行を生成します。

たとえば、以下の 2 つの索引定義があるとします。

```
INDEX IX4: SALARY   ASC
INDEX IX5: COMM     ASC
```

この 2 つの索引を使って、以下の述部が解決されることになります。

```
WHERE SALARY BETWEEN 20000 AND 30000
   AND COMM BETWEEN 1000 AND 3000
```

この例では、索引 IX4 を走査すると、 SALARY BETWEEN 20000 AND 30000 述部を満たすビットマップが生成されます。 IX5 の走査および IX4 のビットマップのプローブを行うと、両方の述部を満たす修飾 RID のリストが生成されます。これは、「動的ビットマップ AND 結合」と呼ばれます。これは、表が十分なカーディナリティーを持っていて、さらに列の修飾範囲内に十分な値がある (または、等号述部が使用される場合は多数の重複がある) 場合にのみ行われます。

重複索引走査時の、ダイナミック・ビットマップのパフォーマンスを認識するには、ソート・ヒープ・サイズ (*sortheap*) データベース構成パラメーターと、ソート・ヒープのしきい値 (*sheapthres*) データベース・マネージャー構成パラメーターとの値を変更する必要のある場合があります。

動的ビットマップがアクセス・プランの中で使用されている場合は、追加のソート・ヒープ・スペースが必要になります。 *sheapthres* が比較的、*sortheap* に近い (つまり、並列問い合わせの割合が 2、3 回の係数よりも少ない) 場合、重複索引アクセスに伴う動的ビットマップは、最適化プログラムが想定した値よりずっと少ないメモリーで作動しなくてはなりません。

これを解決するには、*sheapthres* の値を *sortheap* と比較して増加させます。

注: 単一表にアクセスする場合には、DB2 は、索引 AND 結合と索引 OR 結合の結合を行いません。

クラスター索引

アクセス・プランを選択するとき、最適化プログラムはディスクからページをバッファー・プールに取り出すときの入出力コストを考慮します。その計算の中で、最適化プログラムは照会に必要な入出力の数を見積もります。この見積もりには、バッファー・プール使用率の予測も含まれています。すでにバッファー・プールの中にあるページに含まれている行を読み取るには、追加の入出力が必要ないためです。

索引走査の場合、最適化プログラムは、システム・カタログ表 (SYSCAT.INDEXES) の情報を使用して、データ・ページをバッファー・プール内に読み込むための入出力コストを見積もります。 SYSCAT.INDEXES 表のうち次の列が使用されます。

- CLUSTERRATIO。この索引に関連して、表データのクラスター化の程度を示します。一般に、データのクラスター化が高くなるほど、多くの行が索引キー順でデータ・ページに配列されます。したがって、データ・ページがバッファー内にあるときに、このページの行をすべて読み取れます。この列の値が -1 の場合は、最適化プログラムは PAGE_FETCH_PAIRS および CLUSTERFACTOR 列を使おうとします。

  または

- PAGE_FETCH_PAIRS。CLUSTERFACTOR と共に、データ・ページをさまざまなサイズのバッファー・プールに読み込むのに必要な入出力の数をモデル化するための数値の対がいくつか含まれています。索引の統計を収集するとき、この情報は詳細統計と見なされます。

統計が利用不能な場合は、最適化プログラムは統計のデフォルト値を使います。この値は、索引に関するデータのクラスター化率が少ないことを想定しています。 113ページの『第5章 システム・カタログ統計』と 114ページの『RUNSTATS ユーティリティーを使用しての統計収集』も参照してください。

表の再編成時に行をクラスター化し、挿入処理の間中この特性を保持するのに使用するクラスター化索引を指定することができます。 (表の再編成については、264ページの『カタログとユーザー表の再編成』を参照してください。) 以降に更新および挿入を行うと、そのような索引のクラスター化が低下するため (RUNSTATS で収集される統計によって測定できます)、定期的に表を再編成することが必要な場合があります。 INSERT、UPDATE、および DELETES により頻繁に変更が加えられる表に対する再編成の頻度を下げるには、表の変更時に PCTFREE パラメーターを使用します。これで、追加の挿入が既存のデータでクラスター化されます。

索引のデータがどの程度クラスター化されているかによってパフォーマンスに重大な影響を与える可能性があるため、表の索引の 1 つを 100% クラスター化に近く維持してください。

一般的に、キーがクラスター索引のキーのスーパーセットである場合、または 2 つの索引のキー列間に事実上の相関がある場合には、 1 つの索引だけを 100% クラスター化することはできません。

クラスター索引を使用するパフォーマンス上の理由について詳しくは、 102ページの『索引の管理に関するパフォーマンスについてのヒント』を参照してください。クラスター化索引の作成方法の詳細については、 *SQL 解説書* で CREATE INDEX を参照してください。

事前取り出しを使用したクラスター化ページの読み取り: 最適化プログラムが索引を使用して行にアクセスする場合、すべての RID (行 ID) が索引から取得されるまで、データ・ページの読み取りを据え置くことがあります。たとえば、前に定義された索引 IX1、

```
INDEX IX1:  NAME    ASC,
            DEPT    ASC,
            MGR     DESC,
            SALARY  DESC,
            YEARS   ASC
```

および、次の探索基準があるとします。

```
WHERE NAME BETWEEN 'A' and 'I'
```

最適化プログラムは IX1 上で索引走査を実行することによって、取り出す行 (およびデータ・ページ) を決めます。この索引に従ってデータがクラスター化されていなかった場合、リスト事前取り出しには、索引操作から獲得された RID のリストを分類するステップが含まれることになります。詳細については、256ページの『リスト事前取り出しについて』を参照してください。

索引ページの事前取り出し

該当する場合、データベース・マネージャーは索引ページに対する順次アクセスを検出し、事前取り出し要求を生成します。これは、非選択索引走査および索引の重要な部分にアクセスする選択索引走査の経過時間を大幅に縮小するものです。

最適化プログラムは、DENSITY および SEQUENTIAL_PAGES などの索引統計、索引が常駐する表スペースの特性、および範囲区切り述部の結果を使用して、行われる索引ページ事前取り出しの量を見積もります。それらの見積もりが、特定の索引を使用するための合計コスト見積もりに入れられます。

詳細は、254ページの『順次事前取り出しについて』を参照してください。

関係走査と索引走査

照会に索引を使用できない場合、または索引走査の方がコストがかかると最適化プログラムが判断した場合、最適化プログラムは関係走査を選択します。次のような場合には、索引走査の方がコストが高くなります。

- 表が小さい場合。
- 索引クラスター化の程度が低い場合。
- 表のほとんどにアクセスする場合。

アクセス・プランで関係走査を使うかそれとも索引走査を使うかを、 SQL Explain 機能を使用して判別することもできます。 211ページの『第7章 SQL Explain 機能』を参照してください。

述部の用語

ユーザー・アプリケーションは、SQL ステートメントで、述部を使用して必要な特定の行を修飾することによって、データベースから一連の行を要求します。最適化プログラムが SQL ステートメントの評価方法を決定するとき、各述部は 4 つのカテゴリーのいずれかに分類されます。カテゴリーは、評価プロセスでその述部が使用される方法、および使用される時によって判別されます。それらのカテゴリーをパフォーマンスの点で高いものから順に示すと、次のようになります。

1. 範囲区切り述部
2. 索引検索引き数述部
3. データ検索引き数述部
4. その他の述部

検索引き数 (SARGable) とは、 *s*earch *arg*ument (検索引き数) として使用可能なものを表します。

172ページの『述部の使用法の要約』では、さまざまな述部カテゴリーのパフォーマンスに影響を与える特性を比較しています。

範囲区切り述部と索引検索引き数述部

範囲区切り述部は、索引走査の制限に使用するものです。これらの述部は、索引探索のキー値を開始または停止 (あるいはその両方) します。索引検索引き数述部は、探索の制限には使用しませんが、述部に関係する列は索引キーの一部であるため、索引から評価することはできます。たとえば、以前に定義した索引 IX1 (161ページの『索引走査の概念』を参照) と、次の WHERE 文節とがあるとします。

```
WHERE  NAME  = :hv1
 AND   DEPT  = :hv2
 AND   YEARS > :hv5
```

この例では、最初の 2 つの述部 (NAME = :hv1、DEPT = :hv2) は、範囲区切り述部であり、YEARS > :hv5 は索引検索引き数述部です。

データベース・マネージャーは、これらの述部を評価するときに、基本表を読み取るのではなく索引データを使用します。これらの*索引検索引き数* 述部によって、表からの読み取りに必要な一連の行が少なくなり、アクセスするデータ・ページの数が少なくなります。これらの種類の述部は、アクセスする索引ページの数には影響しません。

データ検索引き数述部

索引マネージャーによっては評価できないが、データ・マネージャー・サービスによって評価できる述部は、*データ検索引き数* 述部と呼ばれます。一般的に、これらの述部では、基礎表の個々の行をアクセスする必要があります。必要なら、データ・マネージャー・サービスは述部を評価するのに必要な列を取り出し、さらに SELECT リスト内の列を満たすもののうち獲得できなかったものを索引から取り出します。

たとえば、PROJECT 表について、次の単一の索引が定義されているものとします。

```
INDEX IX0:  PROJNO ASC
```

この場合、次の照会では、DEPTNO = 'D11' 述部がデータ検索引き数と見なされます。

```
SELECT PROJNO, PROJNAME, RESPEMP
  FROM PROJECT
WHERE DEPTNO = 'D11'
 ORDER BY PROJNO
```

その他の述部

一般にその他の述部は、基本表の単純アクセスを超えた入出力を必要とするものです。その他の述部の例としては、相関副照会を使用するもの、多値副照会 (ANY、ALL、SOME、または IN による副照会) を使用するもの、または LONG

VARCHAR あるいは LOB のデータ (表とは別個のファイル内に保管されるデータ) を読み取るものがあります。これらの述部は、リレーショナル・データ・サービスによって評価されます。

データ・ページにアクセスする際に、索引だけに適用される述部を再適用しなければならないことがあります。たとえば、索引 OR 結合または索引 AND 結合を使用するアクセス・プラン (167ページの『複数の索引アクセス』を参照) は、データ・ページにアクセスする際に、述部をその他の述部として再適用します。

述部の使用法の要約

照会に述部を使うと、その照会を満たすために読み取られるデータの量を減らすことができます。述部カテゴリーが異なると照会のパフォーマンスに与える影響も異なりますが、最適化プログラムはこの影響を考慮します。以下の表では、さまざまな種類の述部のランクを示し、それぞれの種類の述部がパフォーマンスに与える影響を記述しています。

表 14. 述部タイプの特性に関する要約

特性	述部タイプ			
	範囲区切り	索引検索引き数	データ検索引き数	その他
索引入出力の低減	はい	いいえ	いいえ	いいえ
データ・ページ入出力の低減	はい	はい	いいえ	いいえ
内部的に渡される行数の低減	はい	はい	はい	いいえ
修飾行の数の低減	はい	はい	はい	はい

結合の概念

結合 とは、 1 つの表の行を、他の 1 つまたは複数の表の行に連結するということです。たとえば、次の 2 つの表があるとします。

```
      TABLE1                    TABLE2
-----------------          -----------------
 PROJ     PROJ_ID          PROJ_ID     NAME
------    -------          -------    ------
  A          1               1        Sam
  B          2               3        Joe
  C          3               4        Mary
  D          4               1        Sue
                             2        Mike
```

表の ID 列が等しい場合に Table1 と Table2 を結合するには、次に示した SQL ステートメントを使用します。

```
SELECT PROJ, x.PROJ_ID, NAME
  FROM TABLE1 x, TABLE2 y
  WHERE x.PROJ_ID = y.PROJ_ID
```

さらに、以下の一連の結果行が生成されます。

```
 PROJ     PROJ_ID     NAME
------    -------    ------
  A          1        Sam
  A          1        Sue
  B          2        Mike
  C          3        Joe
  D          4        Mary
```

2 つの表を結合すると、1 つの表は外部表として選択され、もう一方の表は内部表として選択されます。外部表は最初にアクセスされ、1 回だけ走査されます。内部表が複数回走査されるかどうかは、結合の種類および索引がどんなものかによります。照会によって 2 つ以上の表が結合されるとしても、それに関係なく最適化プログラムは 1 回に 2 つの表だけを結合します。必要であれば、一時的な中間結果表を作成します。

最適化プログラムは、結合述部 (174ページの『マージ結合』を参照) が存在するかどうか、また表と索引の統計によって判別される必要な各種コストに応じて、 2 つの結合方式 (ネストしたループ結合とマージ結合) のいずれかを選択します。

ネストしたループ結合

ネストされたループ結合は、次の 2 つの方法のいずれかで実行されます。

1. 外部表のアクセスされる各行を、内部表を走査することによって検索する方法。

   たとえば、表 T1 と T2 の列 A の値が次のようになっている場合、

```
     外部表 T1: 列 A              内部表 T2: 列 A
-----------------------      -----------------------
          2                            3
          3                            2
          3                            2
                                       3
                                       1
```

   ネストしたループを行うステップは以下のとおりです。

   - T1 から最初の行を読みます。 A の値は 『2』 です。
   - 一致するもの (『2』) が見つかるまで T2 を走査してから、その 2 つの行を結合します。
   - 次に一致するもの (『2』) が見つかるまで T2 を走査してから、その 2 つの行を結合します。
   - 表の終わりまで T2 を走査します。
   - T1 に戻って、次の行 (『3』) を読みます。
   - T2 を最初の行から始めて一致するもの (『3』) が見つかるまで走査し、その 2 つの行を結合します。
   - 次に一致するもの (『3』) が見つかるまで T2 を走査してから、その 2 つの行を結合します。

- 表の終わりまで T2 を走査します。
- T1 に戻って、次の行 (『3』) を読みます。
- 前と同じように T2 を走査し、一致する行 (『3』) をすべて結合します。

2. 外部表のアクセスされる各行ごとに、内部表で索引検索を行う方法。

   この方法は、次の形式の述部が存在している場合に、指定された述部に使用できます。

   ```
   expr(outer_table.column)  relop  inner_table.column
   ```

   ここで、relop は比較演算子 (たとえば、=、>、>=、<、または <=) であり、expr は外部表で有効な式です。以下は、その例です。

   ```
   OUTER.C1 + OUTER.C2 <= INNER.C1
   ```

   および

   ```
   OUTER.C4 < INNER.C3
   ```

   この方法を使用すると、外部表の各アクセスごとに内部表でアクセスされる行の数を大幅に減らすことができます (結合述部の選択可能性を含め、いくつかの要因に依存しています)。

ネストしたループ結合を評価するとき、最適化プログラムは、結合を実行する前に外部表を分類するかどうかも決定します。結合列に基づいて外部表を並べ替えるなら、ページがすでにバッファー・プール内に存在する確率が高くなるため、内部表でディスクからページにアクセスするための読み取り操作の数が少なくなります。結合で高度にクラスター化された索引を使用して内部表にアクセスする場合、外部表が分類されているなら、アクセスされる索引ページの数を最小限にとどめることができます。

さらに、最適化プログラムは、結合後に分類を行うとコストが高くなると予期した場合に、結合前に分類を実行するよう選択することがあります。 GROUP BY、DISTINCT、ORDER BY、またはマージ結合をサポートするには、結合後の分類が必要になります。

マージ結合

マージ結合 (走査マージ結合または分類マージ結合ということもある) には、table1.column = table2.column. という形式の述部が必要です。これは、*等価結合述部* と呼ばれます。マージ結合には、索引アクセスによって、または分類によって、結合する列での入力の並べ替えが必要になります。マージ結合を使用するためには、結合列を LONG フィールド列やラージ・オブジェクト (LOB) 列にすることはできません。

結合された表は、同時に走査されます。マージ結合の外部表は 1 回だけ走査されます。外部表に繰り返される値がない場合には、内部表も 1 回だけ走査されます。外部表に繰り返される値がある場合には、内部表の中の行のグループが再度走査されることがあります。たとえば、表 T1 と T2 の列 A の値が次のようになっている場合、

```
   外部表 T1: 列 A              内部表 T2: 列 A
-----------------------      -----------------------
           2                            1
           3                            2
           3                            2
                                        3
                                        3
```

マージ結合を行うためのステップは、次のとおりです。

- T1 から最初の行を読みます。 A の値は 『2』 です。
- 一致するものが見つかるまで T2 を走査してから、その 2 つの行を結合します。
- 列が一致する間は T2 の走査を続け、列を結合します。
- T2 内で 『3』 を読んだら、T1 に戻って次の行を読みます。
- T1 内の次の値は 『3』 であり、これは T2 に一致するので、行を結合します。
- 列が一致する間は T2 の走査を続け、列を結合します。
- T2 の終わりに達します。
- T1 に戻り、次の行を獲得します。 T1 の次の値は T1 の直前の値と同じなので、 T2 の最初の 『3』 から再度 T2 の走査が開始されます (データベース・マネージャーはその位置を記憶しています)。

ハッシュ結合

ハッシュ結合には table1.columnX = table2.columnY の形式の述部が 1 つ以上必要で、これらの列のタイプは**同じ**でなければなりません。タイプが CHAR の列の場合は、長さが同じでなければなりません。タイプが DECIMAL の列の場合は、精度と位取りが同じでなければなりません。列のタイプを LONG フィールド列やラージ・オブジェクト (LOB) 列にすることはできません。

まず 1 つ目の表 (「内部」表) が走査され、分類ヒープ割り振りによって描画されたメモリー・バッファーに行がコピーされます (363ページの『分類ヒープ・サイズ (sortheap)』 のデータベース構成パラメーターを参照)。このメモリー・バッファーは、結合述部の列から計算された「ハッシュ・コード」に基づく区分に分割されています。1 つ目の表のサイズが使用可能な分類ヒープのスペースより大きい場合は、選択した区分から一時表にバッファーが書き込まれます。内部表の処理が終わると、2 つ目の表 (「外部」表) が走査されます。次に外部表の行と内部表の行が突き合わされます。そのためには、まず結合述部の列から生成された「ハッシュ・コード」が比較されます。次に、外部行の「ハッシュ・コード」と内部行の「ハッシュ・コード」が一致していると、実際の結合述部の列が比較されます。

一時表に書き込まれていない区分に対応した外部表の行は、メモリー内の内部表の行と直ちに突き合わされます。一方、対応する内部表の区分が一時表に書き込まれている場合は、外部行も一時表に書き込まれます。最後に、一致している区分の組みが一時表から読み取られ、それらの行の「ハッシュ・コード」が突き合わされ、結合述部が検査されます。

ハッシュ結合のパフォーマンス上の効果を理解するには、 *sortheap* データベース構成パラメーターの値、および *sheapthres* データベース・マネージャー構成パラメーターの値を変更する必要があります。

意思決定支援の照会の場合、ハッシュ結合アクセス・プランは、それ以外の場合よりも多くの分類ヒープ・スペースを使用します。 *sheapthres* が *sortheap* と比較的近い値に設定されている場合 (つまり、並行する照会当たり 2 か 3 になる係数より小さい値)、ハッシュ結合を使用すれば、最適化プログラムを使用するときよりも、使うメモリーの量は少なくて済みます。メモリーが限られている状況で実行するときには、ハッシュ結合は非常に遅くなる可能性があります。分類またはハッシュ結合では、使用可能メモリーのほとんどを使うため、複数の分類およびハッシュ結合を処理すると、照会中に問題が発生します。

この解決策は、*sheapthres* を (*sortheap* と比較して) 十分な大きさに構成することです。

外部か内部かの判別

結合時に、内部表と外部表はどのように判別されるのでしょうか。以下は、最適化プログラムが内部表および外部表を決定する方法についての一般的な指針です。

ハッシュ結合の場合は、内部表はメモリー・バッファーに保持されます。メモリー・バッファーが少な過ぎる場合は、ハッシュ結合が分割されます。最適化プログラムはこれを避けようとするので、 2 つの表の小さい方を内部表として、大きい方を外部表として用います。

表にアクセスする順序は、**ネストしたループ結合**では特に重要です。というのも、外部表は 1 回しかアクセスされませんが、内部表は外部表の各行ごとに 1 回ずつアクセスされるためです。最適化プログラムは、コスト見積もりに基づいて外部表と内部表を選択します。これらのコスト見積もりは、以下の要因によって影響を受けます。

- サイズ

  内部表に再アクセスしなければならない回数を少なくするため、多くの場合、小さい方の表が外部表として選択されます。しかし、事前取り出しを行うと、この逆になることがあります。事前取り出しを行うと、大きな表にアクセスするコストがかなり少なくなります。しかし通常は、事前取り出しが効果的なのは結合の外部表だけです。したがって、大きな表が最初にアクセスされることがあります。詳細は、253ページの『バッファー・プールへのデータの事前取り出し』を参照してください。

- 述部

  選択述部を表に適用できる場合、内部表にアクセスするのは外部表に適用される述部を満たす行についてだけなので、表は外部表として選択される可能性が高くなります。

- バッファリング

  外部表の各行ごとに内部表全体を走査しなければならない場合 (つまり、内部表に対して索引参照を実行できない場合) は、バッファリングを利用できるようにするた

め、 2 つの表のうち小さい方が内部表として選択されます。これは、表サイズやバッファー・プール・サイズによって影響を受けます。結合の決定は、バッファー・プール・サイズによって影響されるため、バッファー・プール・サイズを変更した後でアプリケーションをデータベースに再バインドすると、アプリケーションのアクセス・プランが変わる場合があります。

複数のバッファー・プールの作成、そのバッファー・プールのサイズの変更、さらにそのバッファー・プールを使用する表スペースの制御を行う能力は、内部表および外部表内でバッファリングが使用されると、影響を受けます。

- 索引

  どれか 1 つの表上で索引検索を実行できる場合、その表が内部表としての使用に最も適しています。その場合、その表は、外部表の結合キー述部をキー値の 1 つとして使用して、索引キー参照によってアクセスされます。表に索引がない場合は、外部表の各行ごとに内部表全体を走査しなければならないため、その表は内部表としてよい候補とは言えません。

- 順序の要件

  必要な順序を伴う表は、最初にアクセスするようにできます。たとえば、t1 と t2 の間の結合の出力が t1.c について分類されることになっていた場合、 t1 には t1.c についての索引を持つ外部表としてアクセスするのが良いでしょう。結合の出力は順序どおりになっていますが、分類の必要はありません。

```
SELECT * FROM t1, t2
  WHERE t1.a = t2.b
  ORDER BY t1.c
```

マージ結合においては、内部表と外部表は 1 回しか読み取られないため、表にアクセスする順序はさほど重要ではありません。しかし、外部表の重複結合値に対応する内部表の部分は、メモリー内バッファーに保持されます。外部表の次の行が外部表の直前の行と同じである場合にはバッファーは再読み取りされ、それ以外の場合にはバッファーはリセットされます。重複結合値の数がメモリー内バッファーの容量を超えた場合には、すべての重複が保持されるわけではなくなります。これは、ある値の重複が多くあり、その値と一致する値が外部表にある場合にのみ起こります。

重複値に関するこれらの考慮事項をすべて考えたうえで、多くの場合、重複が少ない表の方が結合における外部表として選択されます。しかし最終的には、最適化プログラムは詳細なコスト見積もりに基づいて外部表と内部表を選択します。

最適結合を選択するための探索方式

最適化プログラムは、さまざまな探索方式を使用して、最適な結合方法を判別します。使用される探索方式は、使用中の最適化クラスによって決まります
(68ページの『最適化クラスの調整』を参照してください)。探索方式とその特性は以下のとおりです。

- 貪欲型結合列挙
  - スペースおよび時間の観点から有効です。

  - 単一方向列挙。つまり、2 つの表の結合方法が一度選択されると、それは以降の最適化においても変更されません。
  - 多くの表を結合するときは、最善のアクセス・プランではない場合があります。照会で 2 つまたは 3 つ程度の表しか結合しない場合、貪欲型結合列挙によって選択されるアクセス・プランは、動的プログラミング結合列挙によって選択されるアクセス・プランと同じになります。このことは、照会の同一列に多数の結合述部がある (明示的に指定したか、または述部変位枠を使って暗黙に生成した) 場合に特に当てはまります。
- 動的プログラミング結合列挙
  - 結合する表の数が増えると、スペースと時間の必要性が指数関数的に大きくなります。
  - 最適なアクセス・プランのための効率的かつ完全な探索。
  - DB2 (OS/390 版) で使用される方法と似ています。

結合列挙アルゴリズムは、最適化プログラムが探索するプランの組み合わせの数の主要な決定要素です。

スター型結合のための探索方式

一般に、照会で参照される表は、結合述部によって接続されていなければなりません。結合述部を使わないで 2 つの表が結合していると、 2 つの表のカルテシアン積が形成されます。つまり、最初の表の該当する各行が 2 番目の該当する各行に結合され、 2 つの表のサイズの乗積から成る、通常は非常に大きい結果表が作成されます。そのようなプランがうまく実行するとは考えられないため、最適化プログラムはそうしたアクセス・プランのコストの判別でさえも行いません。唯一の例外として行われるのは、最適化クラスが 9 に設定される場合、または『スタースキーマ』という特殊な場合です。詳細については、 68ページの『最適化クラスの調整』を参照してください。

カルテシアン積を含むアクセス・プランがうまく実行されるのは、通常、スタースキーマ技法で設計された大規模な意思決定支援データベースです。スタースキーマとは大量のロー・データが多数の列を持つ 1 つの表に保持されたデータベース設計のことで、一般に 『ファクト』 表として知られています。多数の列には、ファクト表に格納された特定のデータの次元を特徴付けるエンコード値が入っています。ファクトのサブセットの一部を容易に分析できるように、寸法表を用いてエンコード値をデコードします。一般的な照会は寸法表のデコード値を参照する複数のローカル述部で成っており、次元表をファクト表に接続する結合述部が含まれています。このような照会では、複数の小さい寸法表のカルテシアン積を実行してから、大きいファクト表にアクセスするのが役に立ちます。この技法は、複数の結合述部を複数列索引に突き合わせる場合に役に立ちます。

DB2 には、少なくとも 2 つの寸法表を持つスタースキーマによって設計されたデータベースに対する照会を認識する機能や、寸法表のカルテシアン積の形成を含む潜在的な

プランを入れる探索スペースを大きくする機能があります。カルテシアン積を含むプランが見積もりで最低コストである場合は、そのプランが最適化プログラムによって選択されます。

上記で説明したスタースキーマ技法は、基本キー索引が結合内で使用されたと想定していました。それとは別に、外部キー索引に関係するシナリオもあり得ます。ファクト表の外部キー列が単一列索引で、全寸法表をまたがって相対的に高度な選択が行われる場合には、以下に示すようなスター型結合技法を使用することができます。

1. 各寸法表を次の方法で処理する。
   - 寸法表とファクト表の外部キー索引との間で半結合を実行する方法
   - 行 ID (RID) 値をハッシュして動的にビットマップを作成する方法
2. 各ビットマップを、 AND 述部を用いて直前のビットマップに対して使用する (167 ページの『複数の索引アクセス』を参照)。
3. 最後のビットマップを処理した後に、残す RID を判別する。
4. それらの RID を分類する (オプション)。
5. 基本表の行を取り出す。
6. SELECT 文節で必要とされる寸法表の列にアクセスして、ファクト表とそれらの各寸法表とを再結合する。
7. 述部 (その他の述部) を再適用する。

この技法を使用する場合、複数列索引を持つ必要はありません。この技法を選択するために、ファクト表と次元表の間の明示的な参照保全制約は必要ではありませんが、ファクト表と次元表の間の関係には、この特性がなければなりません。

スター型結合技法の一部によって、作成、使用された動的ビットマップはソート・ヒープ・メモリーを使用します。ソート・ヒープ・サイズ (*sortheap*) データベース構成パラメーターについて詳しくは、*管理の手引き: パフォーマンス* マニュアルの第 13 章、「DB2 の構成」をご覧ください。

複合表

別の重要なパラメーターが照会中の結合の順序の形状を決定します。一対の表を結合した結果は、複合表という新しい表になります。一般に、こうした複合表は別の内部表との結合の外部表となります。これを『複合外部表』と言います。ある場合、特に貪欲型結合列挙方法を使用している場合には、 2 つの表を結合した結果を選び、それを後の結合の内部表にすると役に立ちます。結合自体の内部表が 2 つ以上の表を結合した結果で成っていると、そのプランには『複合内部表』が含まれていると見なします。たとえば、次のような照会では、

```
SELECT COUNT(*)
FROM T1, T2, T3, T4
WHERE T1.A = T2.A AND
      T3.A = T4.A AND
      T2.Z = T3.Z
```

表 T1 と T2 を結合し (T1xT2)、T3 を T4 に結合します (T3xT4)。最後に、最初の結合結果を外部表として選択し、 2 番目の結合結果を内部表として選択すると役に立ちます。最後のプランでは ((T1xT2) x (T3xT4))、結合結果 (T3xT4) が複合内部表となります。照会最適化クラスに従って、最適化プログラムは結合の内部表となる最大数の表に異なる制約を課します。複合内部表は最適化クラス 5、7、および 9 で使用できます。

複製要約表

区分データベース環境で複製要約表を使用すると、基礎表データのデータベース管理事前計算済み値を持つことになるので、パフォーマンスを向上させることができます。たとえば、次の照会は以下の複製要約表を作成することから益を得ています。次のような前提事項があります。

- SALES 表は、複数区分表スペース REGIONTABLESPACE にあり、 REGION 列で区分されている。
- EMPLOYEE および DEPARTMENT 表が単一区画ノード・グループ。

その後、EMPLOYEE 表の情報に基づき、複製要約表を作成します。

```
CREATE TABLE R_EMPLOYEE
  AS (
      SELECT EMPNO, FIRSTNME, MIDINIT, LASTNAME, WORKDEPT
      FROM EMPLOYEE
     )
DATA INITIALLY DEFERRED REFRESH IMMEDIATE
IN REGIONTABLESPACE
REPLICATED;
```

作成後に、複製要約表の内容は、以下のステートメントを実行することにより更新されます。

```
REFRESH TABLE R_EMPLOYEE;
```

次の例は、売上を従業員ごと、部署の合計ごと、総合計ごとに計算しています。

```
SELECT d.mgrno, e.empno, SUM(s.sales)
FROM   department AS d, employee AS e, sales AS s
WHERE  s.sales_person = e.lastname
   AND e.workdept = d.deptno
GROUP BY ROLLUP(d.mgrno, e.empno)
ORDER BY d.mgrno, e.empno;
```

1 つのデータベース区画にしか存在しない EMPLOYEE 表を使用するのではなく、データベース・マネージャーは、 SALES 表が存在する各データベース区画で複製される R_EMPLOYEE 表を使用します。結合を計算するために、従業員の情報をネットワークを超えてそれぞれのデータベース区画に移動させる必要はないので、パフォーマンスが向上します。

複製要約表は結合の並びの中で補助機構として使用することができます。たとえば、ノードが 20 を超える大規模なファクト表スプレッドがあるスタースキーマを所有していた場合、これらの表が連結されていると、ファクト表間の結合とディメンション表が最も影響を受けます。

同一のノード・グループ内にすべての表を配置しようとすると、連結された結合の場合、多くても、正しく分割された一次元の表になります。すべての他のディメンション表は連結された結合の中では使用することができません。それは、ファクト表上の結合列がファクト表の区分化キーと対応しているためです。

たとえば、以下のようにコールされた表を所有することができます。 C1 で区分に分割された FACT (C1, C2, C3, ...) 表、C1 で区分に分割された DIM1 (C1, dim1a, dim1b, ...) 表、C2 で区分に分割された DIM2 (C2, dim2a, dim2b, ...) 表、など。

この例から、述部の DIM1.C1 = FACT.C1 が連結できるので、FACT と DIM1 間の結合が完全であることがご覧いただけます。これら表の両方は C1 列上で区分に分割されています。

述部 WHERE DIM2.C2 = FACT.C2 による DIM2 間の結合は連結できません。これは FACT が C1 列上で分割されており、C2 列上ではないからです。

このケースで、ファクト表のノード・グループ内で DIM2 を複製することは適しています。この方法でそれぞれの区分をローカルで結合できるようになります。

注: ここで述べられている、複製済み要約表は、データベース内レプリケーションと関係しています。データベース間レプリケーションはサブスクリプション、制御表、異なったデータベース、および異なったオペレーティング・システムに配置されたデータと関連しています。データベース間レプリケーションにご興味をお持ちの場合は、詳しくは *レプリケーションの手引きおよび解説書* を参照してください。

複製要約表を作成する際に、ソース表は、シングル・ノード・グループかマルチ・ノード・グループが可能です。ほとんどのケースで、表は小さく、シングル・ノード・グループ内に配置することができます。以下のような方法で複製するデータに対して制限がかけられます。表から列までサブセットのみ指定する方法、述部で使用された番号を通して行番号に制限をかける方法、複製要約表を作成する際に、2 つの方法を利用する方法です。

注: データ・キャプチャー・オプションは複製要約表の機能を必要としません。

複製要約表はマルチ・ノード・グループでも生成することができます。ノード・グループは、ユーザーの大規模な表のノード・グループと同じです。この場合、ソース表のコピーはノード・グループのすべての区分に作成されます。大規模なファクト表とディメンション表間の結合は、全区分に対し、ソース表をブロードキャストするよりも、この環境内で、ローカルで行うことのできるよい機会になります。

複製された表の索引は、自動的に作成されません。索引が作成されるとソース表で確認されているものとは異なる場合があります。

注: 複製された表では、固有索引の作成 (または制約の作成) をすることはできません。これはソース表上で、現在時が異なるなどの制約違反を防ぐためです。これらの制約はソース表上で同じ制約があっても許可されません。

REFRESH ステートメントを使用した後は、他の表と同様に複製表で RUNSTATS を実行する必要があります。

複製表は照会で直接参照することができます。ただし、特殊な区分で表データを参照するために複製表と一緒に述部 NODENUMBER() を使用することはできません。

参照するために複製要約表が使用されている場合 (ソース表を参照した照会がある場合) は、EXPLAIN 機能を使用することができます。始めに、EXPLAIN 表が存在することを確認します。それから、実行しようとしている SELECT ステートメントの説明計画を立てます。最後に、EXPLAIN 出力をフォーマットするため db2exfmt ユーティリティーを使用します。

最適化プログラムによって選択されたアクセス・プランは、結合する必要がある、などの情報に依存する複製要約表を使用する場合としない場合があります。最適化プログラムが、他の区分に対してオリジナル・ソース表をブロードキャストするほうがより安価であると決定した場合、複製要約表は使用されないこともあります。

区分データベースにおける結合方式

以下のセクションでは、区分データベース環境において可能な結合方式について説明します。 DB2 最適化プログラムは、各アプリケーションの要件に応じて、最適な結合方式を自動的に選択します。ここに記載されている結合方式を参照して、各方式で行われていることを理解するのに役立ててください。「表キュー」は、データベース区画間 (または、単一区画データベースの場合はプロセッサー間) で行を転送するための機構です。

以下の説明では、*指示* 表キューは、行が受け取り側のデータベース区画のいずれか 1 つに対してハッシュされる表キューのことを言います。*ブロードキャスト* 表キューは、行が受け取り側のデータベース区画すべてに送られる表キューのことを言います (つまり、ハッシュはされません)。このセクションの図では、q1、q2、q3 は、例に出てくる表キューと対応しています。同様に、図に示されている表は、これらのシナリオの目的に合わせて、 2 つのデータベース区画にまたがって分割されています。矢印は、表キューが送られる方向を示します。なお、調整プログラム・ノードは区画 0 です。

区分データベース内で頻繁な結合が行われる表についての考慮事項の一つに、表の併置があります。表の併置は、区分データベースにおいて、ある表のデータを、同じ区分化キーに基づいて、同じ区画にある別の表のデータを使用して見つけ出すための手段を提供します。併置が行われると、照会の中で結合されるデータは、照会作業の一部として

別のデータベース区画に移動せずに処理されます。結合の応答セットのみが調整プログラム・ノードに移動されます。この問題の詳細は、 *管理の手引き: 計画* の『表の併置』を参照してください。

結合の従属関係については、*SQL 解説書* を参照してください。

併置結合

最適化プログラムが併置結合を処理するためには、結合される表は併置され、対応する区分化キーのすべての対が等価結合述部に入れられなければなりません。 図14 に例が示されています。

注: 複製要約表は併置結合の可能性を高めます。詳細は、180ページの『複製要約表』を参照してください。

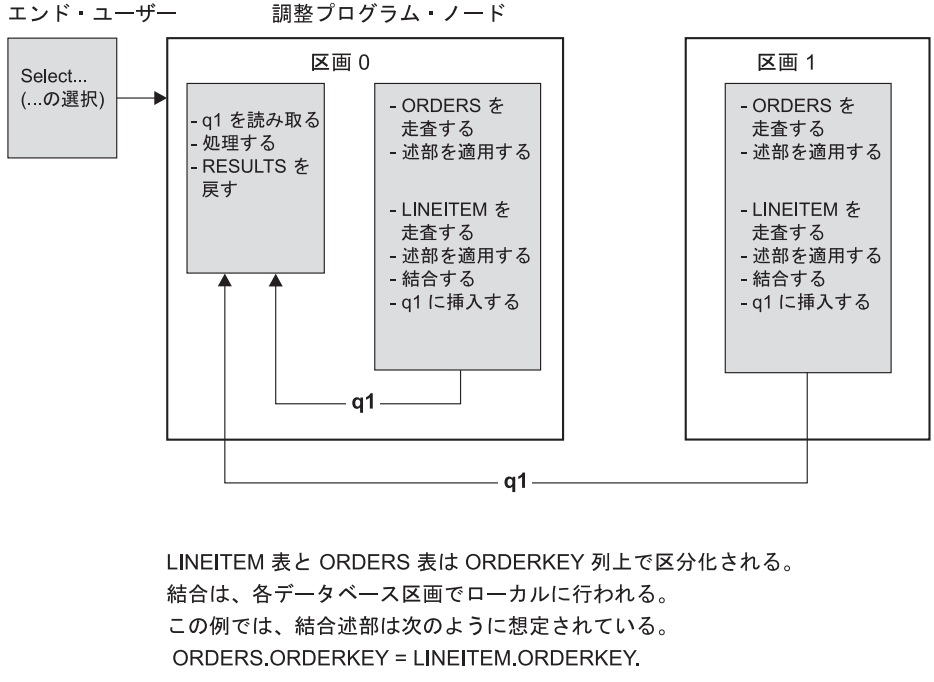


図 14. 併置結合の例

外部表のブロードキャスト結合

この並列結合方式は、結合される表の間に等価結合述部がない場合に使用することができます。また、この結合方式は、これが最も費用対効果が良い結合方式である状況でも使用されます。典型的には、非常に大きな表が 1 つと非常に小さな表が 1 つあり、どちらの表も結合述部列上で区分化されていない場合に使用されます。両方の表を区分化

するよりも、小さな表を大きな表にブロードキャストするほうが「安く」なる可能性があります。 図15 に例が示されています。

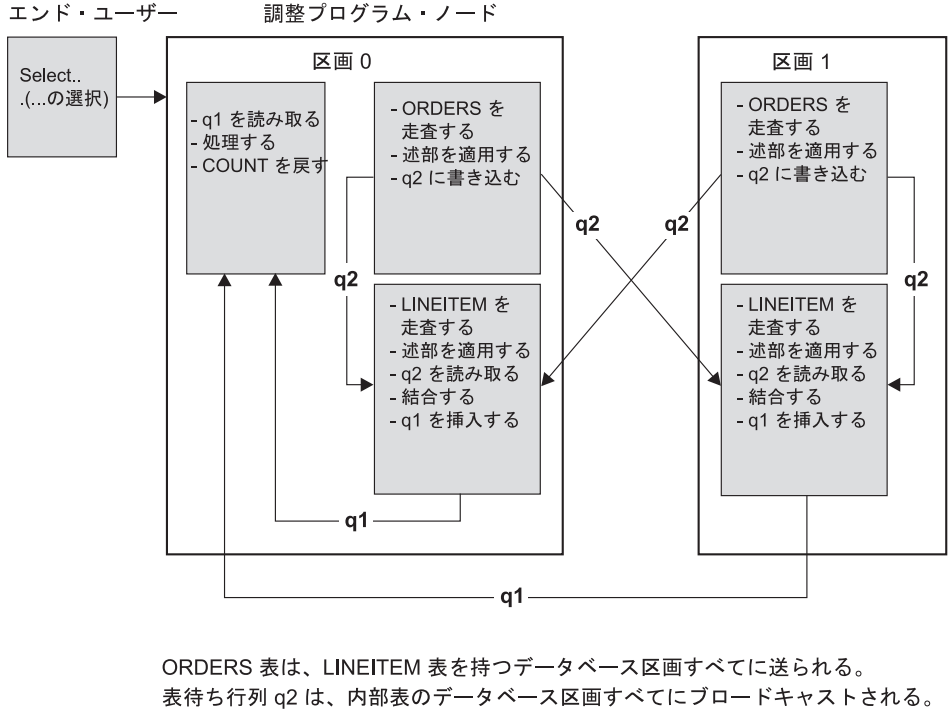
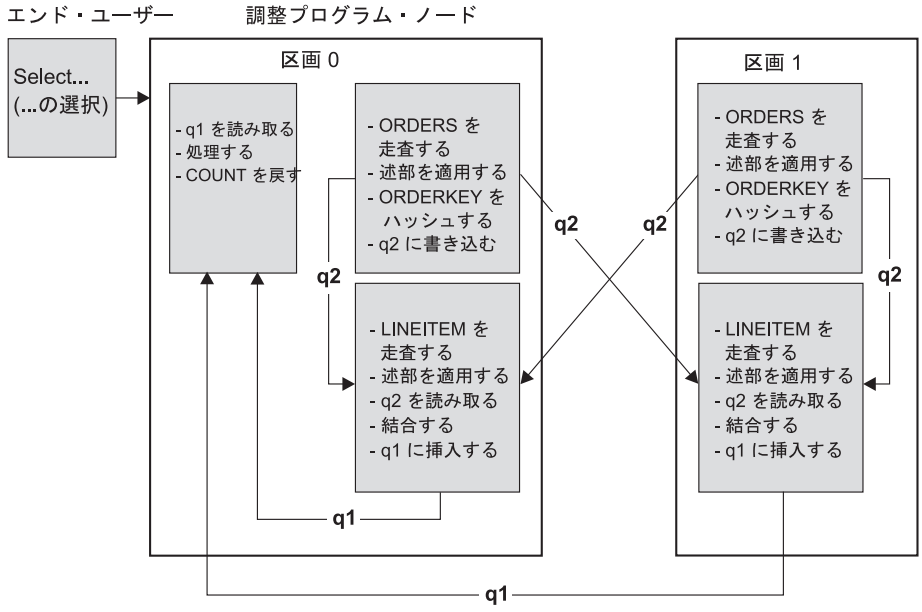


図 15. 外部表のブロードキャスト結合の例

外部表の指示結合

この結合方式では、外部表の各行を、 (内部表の区分化属性に基づいて) 内部表のデータベース区画のいずれか 1 つに送ります。結合は、このデータベース区画上で行われます。 185ページの図16 に例が示されています。



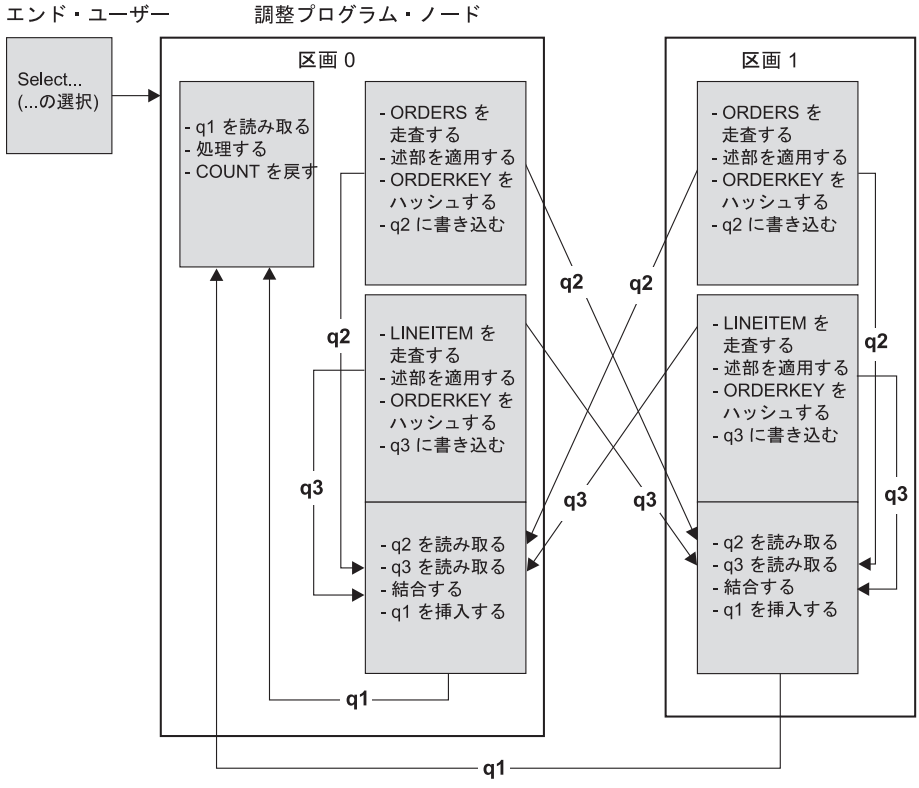
LINEITEM 表は ORDERKEY 列上で区分化される。
ORDERS 表は別の列で区分化される。
ORDERS 表がハッシュされて、適切な LINEITEM 表のデータベース区画に送られる。

この例では、結合述部は次のように想定されている。
ORDERS.ORDERKEY = LINEITEM.ORDERKEY.

図 16. 外部表の指示結合の例

内部表および外部表の指示結合

この方式では、結合を行う列の値に基づいて、外部表および内部表の行がデータベース区画のセットに送られます。結合は、これらのデータベース区画上で行われます。 186 ページの図17 に例が示されています。



いずれの表も、ORDERKEY 列上では区分化されない。
どちらの表もハッシュされ、新しいデータベースに送られて、その区画で結合される。
両方の表待ち行列 q2 と q3 が送られる。
この例では、結合述部は次のように想定されている。
ORDERS.ORDERKEY = LINEITEM.ORDERKEY

図 17. 内部表および外部表の指示結合の例

内部表のブロードキャスト結合

この方式では、内部表が外部結合表のすべてのデータベース区画に対してブロードキャストされます。 187ページの図18 に例が示されています。

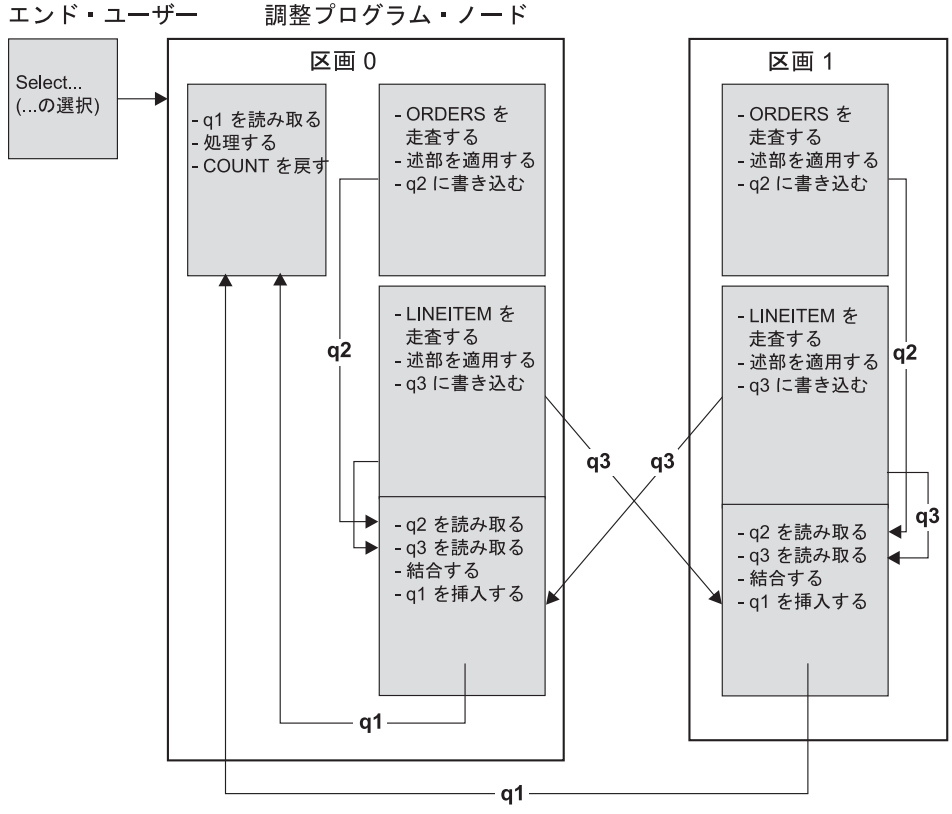
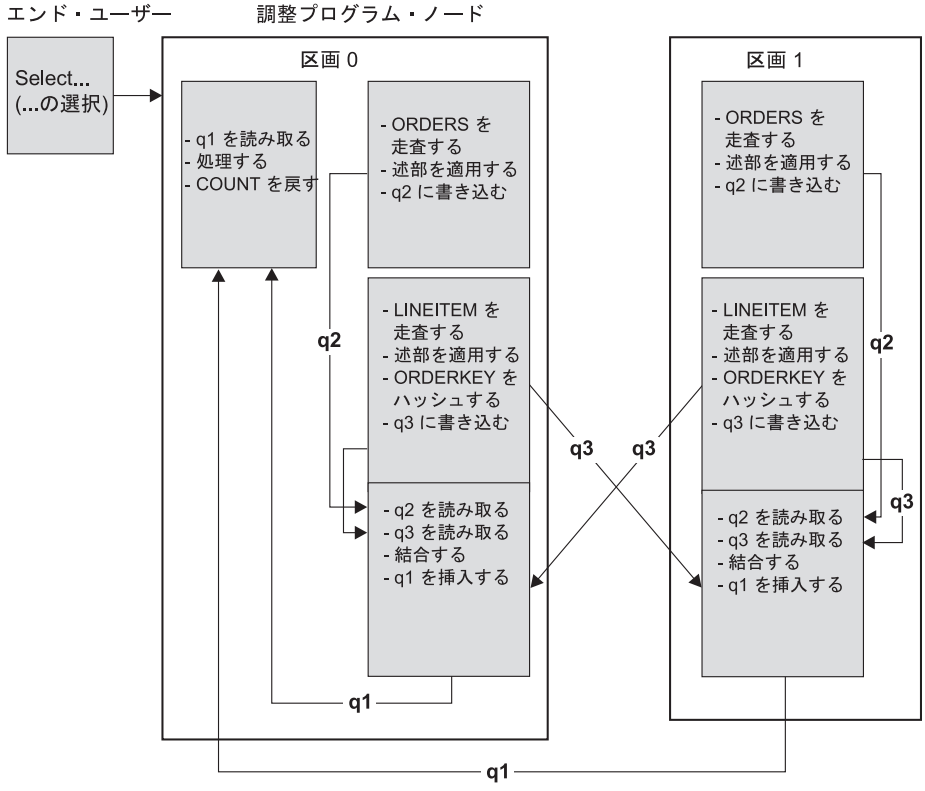


図 18. 内部表のブロードキャスト結合の例

内部表の指示結合

この方式では、内部表の各行を、 (外部表の区分化属性に基づいて) 外部結合表のデータベース区画のいずれか 1 つに送ります。結合は、このデータベース区画上で行われます。 188ページの図19 に例が示されています。



ORDERS 表は ORDERKEY 列上で区分化される。
LINEITEM 表は別の列で区分化される。
LINEITEM 表がハッシュされて、適切な ORDERS 表のデータベース区画に送られる。
この例では、結合述部は次のように想定されている。
ORDERS.ORDERKEY = LINEITEM.ORDERKEY.

図 19. 内部表の指示結合の例

表キュー

表キューは、以下の目的で使用されます。

- 区画間並列化の使用時に、あるデータベース区画の表データを別のデータベース区画に渡す
- 区画内並行化の使用時に、1 つのデータベース区画内で表データを渡す
- 単一区画データベースの使用時に、1 つのデータベース区画内で表データを渡す

各表キューはそれぞれ、単一方向にデータを渡すのに使用されます。

コンパイラーはどこで表キューが必要とされているかを判断し、それらをプランに組み込みます。プランが実行されると、データベース区画間の接続を行うとその表キューが開始されます。表キューがクローズされるのは、処理が終了したときです。

表キューには、以下に示すようにいくつかの種類があります。

- *非同期表*キュー。 これらの表キューが非同期と呼ばれるのは、アプリケーションによって FETCH が出される前に、行の読み取りを行うためです。 FETCH が出されたときには、行はこの表キューから取り出されます。

  非同期表キューは、SELECT ステートメントに FOR FETCH ONLY 文節を指定した場合に使用されます。行の取り出しだけを行う場合には、非同期表キューが他よりも速い方法になります。
- *同期表*キュー。 これらの表キューが同期と呼ばれるのは、アプリケーションによって FETCH が出されるたびに行を 1 行読み取るためです。各データベース区画では、カーソルが、そのデータベース区画から次に読み取られる行に位置づけられます。

  同期表キューは、 SELECT ステートメントに FOR FETCH ONLY 文節が指定されていない場合に使用されます。区分データベース環境では、行の更新を行う場合には、データベース・マネージャーは同期表キューを使用します。
- マージ*表*キュー。 これらの表キューは、順序を保存します。
- *非*マージ*表*キュー。 これらの表キューは「正規」表キューとも呼ばれます。この表キューは、順序を保存しません。
- *listener 表*キュー。 これらの表キューは、相関副照会とともに使用されます。相関値が副照会に渡された後、このタイプの表キューを使用して、結果が親照会ブロックに戻されます。

最適化プログラムでの分類の影響

最適化プログラムは、アクセス・プランを選択する際に、データの分類によるパフォーマンスの影響を考慮します。分類は、取り出した行を要求された順序で並び替えることができる索引が存在しない場合に行われます。また、最適化プログラムが索引走査よりも分類の方が低コストであると判断した場合にも、分類が行われます。最適化プログラムは、データの分類を行う際には、以下のいずれかの処置を実行します。

- 照会の実行時に、分類の結果を「パイプ処理」する。 『パイプ分類と非パイプ分類』および 91ページの『照会最適化に影響する構成パラメーター』を参照してください。
- データベース・マネージャー内で分類を内部処理する。 190ページの『集約および分類の後入れ先出し操作』を参照してください。

パイプ分類と非パイプ分類

分類の完了時に、データの最終的な分類済みリストが 1 回の順次受け渡しで読み取り可能な場合には、結果は*パイプ処理* することができます。パイプ処理は、分類結果を受け

渡す他の (非パイプ) 手段を使用するよりも高速です。最適化プログラムは可能ならば、分類結果をパイプ処理することを選択します。

分類がパイプ処理されるかどうかには関係なく、分類時間は、分類する行の数、キー・サイズ、および行の幅を含め、いくつかの要因によって違ってきます。分類される行が、分類ヒープ内で使用可能なスペースより多くのスペースを占める場合は、複数の分類パスが実行され、各パスごとに行全体のうちの 1 つのサブセットが分類されることになります。各分類パスはバッファー・プール内の一時表に記憶されます。 (バッファー・プール管理の一部として、この一時表のページをディスクに書き込むこともできます。) すべての分類パスが完了したなら、それらの分類済みサブセットをマージして、分類済みの単一の行集合にする必要があります。分類をパイプ処理する場合、行をマージするときに、直接リレーショナル・データ・サービスに渡されます。

詳細については、262ページの『分類パフォーマンス問題の標識の探索』、または 91ページの『照会最適化に影響する構成パラメーター』の *sortheap* 構成パラメーターの説明を参照してください。

集約および分類の後入れ先出し操作

場合によっては、最適化プログラムは、リレーショナル・データ・サービス・コンポーネントのデータ管理サービス・コンポーネントに対して、分類操作または集約操作の後入れ先出しを選択することができます。これらの操作を後入れ先出しにすると、データ管理サービス・コンポーネントがデータを分類ルーチンまたは集約ルーチンに直接渡せるようになり、パフォーマンスが向上します。この後入れ先出しを行わない場合、データ管理サービスはまずこのデータをリレーショナル・データ・サービスに渡し、次いで分類または集約ルーチンとインターフェースを取ります。たとえば、次の照会にはこの最適化方法が適しています。

```
SELECT WORKDEPT, AVG(SALARY) AS AVG_DEPT_SALARY
     FROM EMPLOYEE
GROUP BY WORKDEPT
```

分類の集約

GROUP BY 操作で必要な順序を決めるのに分類が使用されるときは、最適化プログラムは、分類の実行中に GROUP BY の集約の一部または全部を実行することを選べます。これは、各グループにある行の数が多い場合は有利です。分類中に行われる何らかのグループ化により、分類をディスクにスピルさせる必要がなくなっているか少なくなっている場合はさらに有利です。

分類の集約が行われるときには、正しい結果が計算されるようにするために、必要な集約の段階が 3 つあります。 最初の集約の段階である 『部分集約』 は、分類ヒープがいっぱいになるまで集約値を計算します。部分集約は、非集約データが取り込まれて部分集合が作成される処理です。分類ヒープがいっぱいになると、残りのデータはディスクにスピルされ、現在いっぱいになっている分類ヒープで計算された部分集合のすべてが入るようになります。分類ヒープがリセットされた後、新しい集約が開始されます。

2 番目の集約の段階である 『中間集約』 は、すべてのスピルされた分類実行を取り込んで、さらにグループ化キーに対して集約を行います。グループ化キー列は区分化キー列のサブセットなので、この集約は終了しません。中間集約は、既存の部分集合を取り込んで新しい部分集合を作ります。この段階はオプションであり、区画内並列処理と区画間並列処理の両方に使用されます。最後のケースでは、グローバル・グループ化キーが使用可能になるときにグループ化は終了します。区画間並列処理では、グループ化キーが、複数の区画にわたってグループを分けている区分化キーのサブセットであって、集約を完了するために再区分化が必要な場合に、このことが生じます。集約を完了するために単一のエージェントに減らされる前に、各エージェントがスピルされた分類実行を終了するときに、似たようなケースが区画内並列処理で存在します。

最後の集約の段階である 『最終集約』 は、すべての部分集合を取り込んで集約を完了させます。最終集約は、部分集合を取り込んで最終集合を作ります。このステップは、GROUP BY オペレーターで常に生じます。分類が分割されないという保証はないので、分類が集約を完了させることはありません。完全な集約は、非集約データを取り込んで、最終集合を作ります。集約のこの方式は、すでに正しい順序になっているデータをグループ化するとき、また区分化で方式の使用が禁止されていないときに、よく使用されます。

区画内並列操作の最適化方式

最適化プログラムは、SQL ステートメントのコンパイル時に並列操作の程度が指定された場合には、照会をデータベース区画内で並列に実行するように、アクセス・プランの選択を行います。

実行時には、「サブエージェント」と呼ばれる複数のデータベース・エージェントが作成されて、照会を実行します。サブエージェントの数は、 SQL ステートメントのコンパイル時に決められた並列操作の程度以下になります。 SQL ステートメントの並列操作の程度の設定についての詳細は、 89ページの『アプリケーションの並列処理』を参照してください。エージェントおよびサブエージェントの詳細については、 269ページの『データベース・エージェント』を参照してください。

区分データベースの場合には、並列操作の程度は各区画に適用されます。たとえば、指定されたデータベース区画で実行される照会の一部分は、その SQL ステートメントのそのデータベース区画に指定された並列化の程度に基づいて、さらに並列化されます。

アクセス・プランを並列化するには、プランを、各サブエージェントによって実行される部分と調整エージェントによって実行される部分とに分割します。サブエージェントは、表キューを介して、データを調整エージェントか他のサブエージェントに渡します。区分データベースでは、サブエージェントは、表キューを介して、他のデータベース区画のサブエージェントとの間でデータの送受信を行うことができます。

このセクションでは、単一データベース区画内での並列操作方式について説明します。

並列走査の方式

リレーショナル走査および索引走査は、同じ表または索引上で並列して実行することができます。並列リレーショナル走査の場合、表は、ページ範囲または行範囲に分割されます。分割後、ページ範囲または行範囲がサブエージェントに割り当てられます。サブエージェントは割り当てられた範囲を走査し、その現行の範囲での作業が完了した時点で別の範囲が割り当てられます。

並列索引走査の場合には、索引は、索引キー値およびキー値あたりの索引項目数に基づいて、レコード範囲に分割されます。並列索引走査は、並列表走査と同様に、レコード範囲を割り当てられたサブエージェントを使用して行われます。サブエージェントには、現行の範囲での作業が完了した時点で新しい範囲が割り当てられます。

走査単位 (ページか行のいずれか) および走査の細分度は、最適化プログラムによって決められます。

並列走査は、サブエージェント間で均等になるように作業を分配します。並列走査の目標は、サブエージェント間の負荷を均衡させて、サブエージェントが同等に使用されるようにすることです。使用中のサブエージェントの数が使用可能なプロセッサーの数と等しく、ディスクが入出力要求で過度に作動しているということがない場合には、マシン・リソースは効率的に使用されていると言えます。

他のアクセス・プランの操作によっては、照会の実行時にデータの不均衡が生じることがあります。最適化プログラムは、データの均衡を維持できるように並列方式を選択します。

並列分類の方式

最適化プログラムは、以下のいずれかの並列分類方式を選択します。

ラウンドロビン分類

この分類は、「再分配分類」とも呼ばれます。これは効率的な共用メモリー分類で、すべてのサブエージェントに対して可能な限り均等にデータを再分配します。この分類は、ラウンドロビン・アルゴリズムを使用して、均等な分配を行います。 まず最初に、各サブエージェントごとに個々の分類を作成します。挿入フェーズでは、サブエージェントは、ラウンドロビン様式で、個々の分類のそれぞれに挿入を行います。こうすることで、より均等にデータを分配することができます。

区分分類

この分類は、分類が各サブエージェントごとに作成されるという点では、ラウンドロビン分類に似た働きをします。この分類では、サブエージェントはハッシュ関数を分類列に適用して、行をどの分類に挿入するかを判別します。たとえば、マージ結合の内部と外部が区分分類の場合には、サブエージェントはマージ結合を使用して、対応する区画を結合することができます。こうすることによって、マージ結合を並列に実行することができます。

複製分類

この分類は、すべてのサブエージェントがすべての分類出力を必要としている場合に使用されます。ある分類が作成されると、サブエージェントはその分類の挿入時に同期化されます。分類が完了すると、各サブエージェントが分類全体の読み取りを行います。この分類は、行数が少ない場合に、データ・ストリームのバランスをとり直すために使用することができます。

共用分類

この分類は、サブエージェントが分類結果上で並列走査をオープンする点以外は、複製分類と同じ働きをします。この分類は、ラウンドロビン分類と同様の方法で、サブエージェント間にデータを分配します。

並列一時表

サブエージェントが共同して同じ表に行を挿入することによって、一時表を生成することができます。この表は、**共用一時表**と呼ばれます。サブエージェントは、データ・ストリームが複製されるか区分化されるかに応じて、私用走査または並列走査のいずれかを共用一時表上でオープンします。

並列集約の方式

集約操作は、サブエージェントによって並列に実行することができます。集約操作では、データをグループ化列上に配列する必要があります。サブエージェントがグループ化列の値の集合に関する行をすべて確実に受け取ることができれば、集約を最後まで完全に実行できます。これは、以前の区分分類のためにグループ化列上のストリームがすでに区分化されている場合に生じます。

上記以外の場合は、サブエージェントは部分的に集約を実行し、別の方式を使用して集約を完了させます。その方式は以下のとおりです。

- 表キューをマージして、部分的に集約されたデータを調整エージェントに送る。調整プログラムにより集約が完全に行われます。
- 部分的に集約したデータを区分分類に挿入する。この分類はグループ化列上で区分化されます。こうすると、グループ化列の集合に関するすべての行が確実に 1 つの分類区分に入れられます。
- バランスを取るためにストリームを複製する必要がある場合は、部分的に集約したデータを複製分類に挿入できる。個々のサブエージェントは複製分類を使用して集約を完成させ、集約の結果と同じ内容のコピーを受け取ります。

並列結合の方式

結合操作は、サブエージェントによって並列に実行することができます。並列結合の方式は、データ・ストリームの特性によって決められます。

結合は、結合の内部または外部表において、データ・ストリームを区分化または複製 (あるいは、その両方) することによって、並列化することができます。たとえば、ネスト・ループ結合は、外部のストリームが並列走査のために区分化され、さらに内部のストリームが各サブエージェントによって独立して再評価された場合に、並列化することができます。マージ結合は、内部ストリームおよび外部ストリームが区分分類のために値によって区分化された場合に、並列化することができます。

自動要約表

要約表は照会応答時間を短縮するのに非常に役立ちます。いくつかの基本的な照会構造が予想できる多くの環境において、要約表を使用して以下のことが行えます。

- 1 つ以上の次元を越えてデータを集約する
- 表のグループを越えてデータを結合し、集約する
- 共通してアクセスされるデータのサブセット (つまり、『ホットな』水平区画または垂直区画) を識別する
- 区分データベース環境で表、または表の一部を再区分化する

要約表の情報は、 SQL コンパイラーに組み込まれています。 SQL コンパイラーでは、照会書き直し (151ページの『SQL コンパイラーによる照会書き直し』を参照) と最適化プログラム (160ページの『データ・アクセス概念および最適化』を参照) が組み合わされて照会が要約表と突き合わされ、照会用に基本表にアクセスする要約表を代用するかどうかが判別されます。要約表を使用して照会に応答するときは、 EXPLAIN 機能 (211ページの『第7章 SQL Explain 機能』を参照) を使用して、選択された要約表を判別することができます。要約表は多くの点で正規表のように働くので、表スペース定義を使用したデータ・アクセスの最適化、索引の作成、および RUNSTATS の発行に関する考慮事項は要約表にも当てはまります。

要約表がいかに役立つかを理解する助けとして、以下の表で多次元分析照会について示されています。この例では、照会がどれだけ要約表を活用するかが示されています。

この例では、 1 組の顧客と 1 組のクレジット・カード口座を含むウェアハウスのシナリオを想定します。ウェアハウスには、クレジット・カードが使用された一連のトランザクションが記録されています。各トランザクションには、一緒に購入された 1 組の品目が含まれます。 2 つの表 (トランザクション品目を含む表と、購入トランザクションを識別する表) が大きく、ともにスター型のハブであるため、この環境は、複数スター環境として分類されます。

トランザクションを説明する 3 つの階層次元があります。それは、製品、場所、そして時刻です。製品階層は、製品グループと製品ラインを表す 2 つの正規化された表に記録されます。場所階層には、市町村、都道府県、および国情報が含まれ、単一の非正規化された表で表されます。時刻階層には、日、月、および年情報が含まれ、単一データ・フィールドでエンコードされます。日付の次元は、組み込み機能を使用して、トランザクションの日付フィールドから抽出されます。このシナリオには、顧客の口座情報、および顧客情報を表す別の表もあります。

要約表は、次の各レベルについて、売上の合計を使用して作成されます。
- 製品階層
- 場所階層
- 年、月、日から構成される時刻階層

広い範囲にわたる照会でも、この保管された集約データから答えを取り出すことができます。以下の例では、製品グループおよび製品ラインの次元、市町村、都道府県、および国の次元、そして時間の次元にしたがって、売上の合計と数を計算しています。この例では、GROUP BY 文節にいくつか別の列が含まれています。

```
CREATE TABLE dba.PG_SALESSUM
  AS (
      SELECT l.id AS prodline, pg.id AS pgroup,
             loc.country, loc.state, loc.city,
             l.name AS linename, pg.name AS pgname,
             YEAR(pdate) AS year, MONTH(pdate) AS month,
             t.status,
             SUM(ti.amount) AS amount,
             COUNT(*) AS count
FROM   cube.transitem AS ti, cube.trans AS t,
       cube.loc AS loc, cube.pgroup AS pg,
       cube.prodline AS l
WHERE  ti.transid = t.id
   AND ti.pgid = pg.id
   AND pg.lineid = l.id
   AND t.locid = loc.id
         AND YEAR(pdate) > 1990
      GROUP BY l.id, pg.id, loc.country, loc.state, loc.city,
               year(pdate), month(pdate), t.status, l.name, pg.name
    )
DATA INITIALLY DEFERRED REFRESH DEFERRED;

REFRESH TABLE dba.SALESCUBE;
```

要約表は、通常、基本ファクト表と比べてかなり小さくなっています。 (例で示されているように) DEFERRED オプションを指定することにより、要約表が更新されるときを制御することができます。

このような事前計算済み合計を利用できる照会には、次のものがあります。
- 月と製品グループごとの売上
- 1991 以降の売上の合計
- 1995 または 1996 の売上
- 製品グループまたは製品ラインの売上の合計
- 1995、1996 の特定の製品グループまたは製品ラインの売上の合計
- 特定の国の売上の合計

これらの照会の正確な答えはこの要約表にはありませんが、答えの一部がすでに計算されているので、要約表を使用して答えを計算する方が、大きな基礎表を使用するよりも

かかる費用がはるかに安くなります。費用のかかる基礎データの結合、分類、および集約は、要約表では回避、または削減されます。

以下は、すでに計算されている要約表の中の結果を使用できるので、パフォーマンスが大幅に改善される照会のサンプルです。最初の例は、1995 と 1996 の売上の合計を戻します。

```
SET CURRENT REFRESH AGE=ANY

SELECT YEAR(pdate) AS year, SUM(ti.amount) AS amount
FROM   cube.transitem AS ti, cube.trans AS t,
       cube.loc AS loc, cube.pgroup AS pg,
       cube.prodline AS l
WHERE  ti.transid = t.id
   AND ti.pgid = pg.id
   AND pg.lineid = l.id
   AND t.locid = loc.id
   AND YEAR(pdate) IN (1995, 1996)
GROUP BY year(pdate);
```

2 番目の例は、1995 と 1996 の製品グループごとの売上の合計を戻します。

```
SET CURRENT REFRESH AGE=ANY

SELECT pg.id AS "PRODUCT GROUP",
       SUM(ti.amount) AS amount
FROM   cube.transitem AS ti, cube.trans AS t,
       cube.loc AS loc, cube.pgroup AS pg,
       cube.prodline AS l
WHERE  ti.transid = t.id
   AND ti.pgid = pg.id
   AND pg.lineid = l.id
   AND t.locid = loc.id
   AND YEAR(pdate) IN (1995, 1996)
GROUP BY pg.id;
```

データベースが大きいほど、このような照会の応答時間はさらに短縮できます。これは、要約表が基本表ほど速く大きくならないためです。要約表の 1 つの利点は、 DB2 ユニバーサル・データベースがその表を使用して、効果的に照会間の作業の重複をなくすことができる点です。 1 度計算して要約表を構築すれば、その内容を多数の照会に再利用することができます。

連合データベース照会コンパイラー・フェーズ

このセクションでは、連合データベース・システムにおける、追加の照会処理フェーズについて説明します。さらに、連合データベース照会のパフォーマンスを改善するための情報も載せられています。主なトピックは以下のとおりです。

- 197ページの『後入れ先出し分析』
- 204ページの『リモートでの SQL 生成とグローバル最適化』

後入れ先出し分析

後入れ先出し分析を実行すると、リモート・データ・ソースで特定の操作を実行できるかどうかが、 DB2 最適化プログラムに通知されます。操作は、関係演算子、システムまたはユーザー関数、または SQL 演算子 (GROUP BY、ORDER BY など) などの関数とすることができます。

後入れ先出しができない関数は、照会パフォーマンスに多大に影響することがあります。選択述部を、データ・ソースで評価するのではなく、ローカルに評価するときの影響を考慮する必要があります。この方法を使う場合、DB2 はリモート・データ・ソースから表全体を検索し、述部に対してローカルにフィルター操作しなければならないことがあります。ネットワークに制約があり、なおかつ表が大きい場合、照会パフォーマンスに影響する場合があります。

後入れ先出しされない演算子も、照会パフォーマンスに多大に影響することがあります。 たとえば、GROUP BY 演算子によってリモート・データ・ソースをローカルに集約すると、 DB2 はリモート・データ・ソースから表全体を検索しなければならない場合があります。

たとえば、ニックネーム N1 が、 DB2 (OS/390 版) データ・ソースに含まれるデータ・ソース表 EMPLOYEE を指しているとします。さらに、この表には 10,000 の行があり、列の 1 つには従業員の名前が、そして別の列には給与が入っているとします。ステートメントは次のようになります。

```
SELECT LASTNAME, COUNT(*)  FROM N1
   WHERE LASTNAME  > 'B' AND SALARY > 50000
   GROUP BY LASTNAME;
```

いくつかの可能性が考えられます。

- DB2 と DB2 (OS/390 版) での照合順序が同じ場合、照会述部は DB2 (OS/390 版) で後入れ先出しされる可能性があります。 通常は、表全体を DB2 にコピーして操作をローカルに実行するよりも、データ・ソースで結果をフィルターに掛けてグループ分けするほうが効率的です。連合システムでの後入れ先出し分析により、そのデータ・ソースで操作を実行できるかどうかが判別されます。この場合、述部および GROUP BY 操作はデータ・ソースで実行できます。
- 照合順序が同じでない場合、後入れ先出し分析により、述部全体をデータ・ソースで評価できないことが判別されます。しかし、最適化プログラムは、述部の SALARY > 50000 部分を後入れ先出しするよう決定する可能性があります。 範囲の比較は、依然として DB2 で実行する必要があります。
- 照合順序が同じであり、ローカル DB2 サーバーが非常に高速であることを最適化プログラムが知っていれば、最適化プログラムは、 GROUP BY 操作を DB2 でローカルに実行することが最善の (最も費用のかからない) 方法であると判断できます。述部はデータ・ソースで評価されます。これは、グローバル最適化によって結合された後入れ先出し分析の一例です。 DB2 は使用可能なパスを考慮しつつ、最も効率的なプランを選択します。

一般には、最適化プログラムによるデータ・ソースでの評価について、関数や演算子を考慮に入れられることを確認することが目標です。関数または SQL 演算子がリモート・データ・ソースで評価されるかどうかは、多くの要素に左右されます。キーとなる要素は、サーバー特性、ニックネーム特性、および照会特性の 3 つのグループに分けて述べられます。

後入れ先出しの機会に影響するサーバー特性

以下のセクションでは、後入れ先出しの機会に影響する可能性のあるデータ・ソース特有の要素を説明します。一般には、DB2 では豊富な SQL ダイアレクトを使用して照会を実行依頼するため、このような要素が存在しています。このダイアレクトは、 DB2 照会時にアクセスされるサーバーがサポートしている SQL ダイアレクトよりも、さらに多くの機能を提供することができます。 DB2 はデータ・サーバーでの機能の不足を補正することができますが、そのようにすると、その操作は DB2 で実行する必要があります。

SQL 機能: 各データ・ソースは、さまざまな SQL ダイアレクト、そして異なるレベルの機能をサポートしています。たとえば、GROUP BY リストを考慮してください。ほとんどのデータ・ソースは GROUP BY 演算子をサポートしていますが、 GROUP BY リストでの項目数に制限があるものもあれば、 GROUP BY リストで式が許可されるかどうかに関する制限があるものもあります。リモート・データ・ソースで制限がある場合、 DB2 は GROUP BY 操作をローカルに実行しなければならないことがあります。

SQL の制限: それぞれのデータ・ソースには、異なる SQL 制限が存在する可能性があります。たとえば一部のデータ・ソースでは、パラメーター・マーカーをリモート SQL ステートメントへの値にバインドする必要があります。したがって、パラメーター・マーカーの制限を調べ、各データ・ソースがそのようなバインド機構をサポートできることを確かめる必要があります。ある関数の値にバインドする適切な方法を DB2 が判別できない場合、この機能はローカルに評価する必要があります。

SQL の限界: DB2 では、リモート・データ・ソースよりも大きい整数を使用することができます。ただし、データ・ソースに送信されるステートメントに、限界を超過した値を組み込むことはできません。したがって、この定数を操作する関数や演算子は、ローカルに評価される必要があります。

サーバー特有の情報: このカテゴリーには、いくつかの要因が該当します。その 1 つの例は、NULL 値の分類 (最高値、最低値、または配列によって決定する) です。たとえば、NULL 値がデータ・ソースにおいて DB2 とは異なる方法で分類される場合、ヌル可能な式での ORDER BY 操作をリモートに評価することはできません。

照合順序: データ・ソースが使用するのと同じ照合順序を使うよう連合データベースを構成し、 *collating_sequence* サーバー・オプションを 'Y' に設定すると、最適化プログラムは後入れ先出しによる文字範囲比較述部を考慮できます。

連合サーバーからの照会で分類が必要な場合、分類が行われる箇所は、いくつかの要因により異なります。連合データベースの照合順序が、照会されたデータの格納されているデータ・ソースでの照合順序と同じであれば、分類はデータ・ソースで実行することができます。照合順序が同じであれば、最適化プログラムは、ローカル分類またはデータ・ソースでの分類のどちらが照会を完了するための最も効率的な方法であるかどうかを判断できます。同様に、照会で文字データの比較が必要な場合、この比較もデータ・ソースで実行できます。

数値比較は一般に、照合順序が異なっていたとしても、いずれかの位置で実行できます。ただし、連合データベースとデータ・ソースとの間でヌル文字の重みが違うと、異常な結果が発生する可能性があります。同じように、比較ステートメントの場合、大文字小文字を区別しないデータ・ソースにステートメントを実行依頼するときには注意が必要です。大文字小文字を区別しないデータ・ソースでは、文字 ″I″ と ″i″ に割り当てられている重みは同じです。デフォルトでは DB2 は大文字小文字を区別し、それぞれの文字に異なる重みを割り当てます。

連合データベースとデータ・ソースの照合順序が異なる場合、 DB2 はデータを連合データベースに送ることにより、分類と比較をローカルに実行できるようにします。これは、ユーザーが、連合サーバーに定義した照合順序で並べられた照会結果を要求しているためです。データをローカルに並べることにより、連合サーバーはこの要求を実現します。

ローカルの分類および比較のためにデータを検索すると、一般的にパフォーマンスは低下します。したがって、データ・ソースで使うのと同じ照合順序を使用するように、連合データベースを構成することを考慮してください。そのようにするなら、連合サーバーはデータ・ソースで分類と比較を行えるため、パフォーマンスは向上します。たとえば、DB2 UDB (OS/390 版) では、ORDER BY 文節で定義された分類は、 EBCDIC コード・ページに基づく照合順序で実行されます。連合サーバーを使用し、ORDER BY 文節で分類された DB2 (OS/390 版) データを検索する場合、 EBCDIC コード・ページに基づいて事前定義された照合順序を使用するように連合データベースを構成することをお勧めします。

連合データベースとデータ・ソースでの照合順序が異なり、データ・ソースの順序で並べられたデータが必要な場合は、照会をパススルー・モードで実行依頼するか、データ・ソース視点で照会を定義することができます。

照合順序およびその設定方法についての詳細は、 *管理の手引き: 計画* を参照してください。 *collating_sequence* サーバー・オプションについての詳細は、 107ページの表8 を参照してください。

サーバー・オプション: 一部のサーバー・オプションは、後入れ先出しの機会に影響します。特に、*collating_sequence*、 *varchar_no_trailing_blanks*、および *pushdown* の設定を検討してください。これらのオプションの設定については、 105ページの『連合データベース照会に影響を与えるサーバー・オプション』を参照してください。

DB2 タイプ・マッピングおよび関数マッピングの要因: DB2 が備えているデフォルトのローカル・データ・タイプ・マッピング (データ・タイプの表については、 *アプリケーション開発の手引き* を参照) は、各データ・ソースのデータ・タイプに十分なバッファー・スペースが確保されるように設計されています (データの消失を防ぐため)。ユーザーは、特定のアプリケーションに合うように、特定のデータ・ソースのタイプ・マッピングをカスタマイズすることもできます。たとえば、Oracle データ・ソースの DATE データ・タイプ (デフォルトでは、 DB2 TIMESTAMP データ・タイプにマップされる) にアクセスする場合、ローカル・データ・タイプを DB2 DATE データ・タイプに変更できます。

DB2 は、データ・ソースでサポートされていない関数を補正できます。関数が補正される場合には、以下の 3 つがあります。

- この関数が単にリモート・データ・ソースに存在しない。
- この関数は存在するが、オペランドの特性が関数の制約事項に違反している。この状況の一例として、IS NULL 関係演算子をあげることができます。ほとんどのデータ・ソースはこの演算子をサポートしていますが、 IS NULL 演算子の左辺には列名しか使用できないなど、制限が存在する場合もあります。
- 関数がリモートに評価されると、異なる結果を返すことがある。この状況の一例として、'>' (より大) 演算子をあげることができます。異なる照合順序のデータ・ソースの場合、「より大」演算子が DB2 によってローカルに評価されると、異なる結果が返される可能性があります。

後入れ先出しの機会に影響するニックネーム特性

以下のセクションでは、後入れ先出しの機会に影響することのあるニックネームに特有の要素を説明します。

ニックネーム列のローカル・データ・タイプ: 列のローカル・データ・タイプがデータ・ソースでの述部の評価を妨げていないことを確認します。前述したように、デフォルトのデータ・タイプ・マッピングはオーバーフローを避けるようになっています。しかし、長さの異なる 2 つの列の間での述部の結合は、 DB2 が長い方の列をバインドする方法によっては、短い列が存在するデータ・ソースでは行われません。このような状況が生じると、 DB2 最適化プログラムによって評価できる結合順序の数に影響することがあります。たとえば、 INTEGER または INT データ・タイプを使用して作成された Oracle データ・ソース列は、タイプ NUMBER(38) になります。 DB2 整数の範囲は 2**31 から (-2**31)-1 で、NUMBER(9) とほぼ等しいため、この Oracle データ・タイプのニックネーム列は、ローカル・データ・タイプ FLOAT となります。この場合、DB2 整数列と Oracle 整数列の結合は、 DB2 データ・ソース (短い方の結合列) では行われません。ただし、DB2 INTEGER データ・タイプがこの Oracle 整数列の領域を包含している場合は、 ALTER NICKNAME ステートメントを使用してローカル・データ・タイプを変更することによって、 DB2 データ・ソースで結合を実行できます。

列のオプション: ニックネームの列オプションを追加したり変更するには、 ALTER NICKNAME SQL ステートメントを使うことができます。

このようなオプションの 1 つに、″varchar_no_trailing_blanks″ があります。これは、後書きブランクを含まない列を識別するときに使用できます。コンパイラーの後入れ先出し分析ステップでは、列に対して実行するすべての操作を検査するときに、この情報を利用します。この指示に基づき、DB2 は異なってはいても等価な形式の述部を生成し、データ・ソースに送信されるリモート SQL ステートメントで使用できます。データ・ソースに対して異なる述部が評価される可能性はありますが、最終的な結果は同じになります。

別の列オプションは numeric_string です。このオプションは、その列の値が常に後書きブランクなしの数値であるかどうかを示します。

列オプションの値とデフォルトについては、 表15 を参照してください。

表 15. 列オプションとその設定値

オプション	有効な設定値		デフォルト設定
numeric_string	'Y'	はい - この列には数値データのストリングだけが含まれます。重要: この列に、後書きブランクを付けられた数値ストリングだけが含まれる場合、 'Y' を指定することはお勧めできません。	'N'
	'N'	いいえ - この列は数値データのストリングに限定されていません。	
	列の numeric_string を 'Y' に設定すると、列データの分類に干渉するブランクがこの列には含まれないことを、最適化プログラムに知らせることになります。このオプションは、データ・ソースの照合順序が DB2 の照合順序とは異なる場合に役立ちます。このオプションでマークされた列は、照合順序が異なるためにローカルな (データ・ソースの) 評価から除かれるということはありません。		

表 15. 列オプションとその設定値 (続き)

オプション	有効な設定値	デフォルト設定
varchar_no_trailing_blanks	このデータ・ソースが、ブランクが埋め込まれていない varchar 比較セマンティクスを使用するかどうかを指定する。後書きブランクを含んでいない可変長文字ストリングについて、一部の DBMS のブランク埋め込みなしの比較セマンティクスでは、DB2 の比較セマンティクスと同じ結果が戻されます。データ・ソースにあるすべての VARCHAR 表 / 視点の列に後書きブランクが含まれていないことが確かである場合は、データ・ソースについてこのサーバー・オプションを 'Y' に設定することを考慮してください。このオプションは、Oracle データ・ソースでしばしば使用されます。ニックネームを持つ可能性のあるすべてのオブジェクトを考慮に入れてください。 'Y' このデータ・ソースのブランク埋め込みなしの比較セマンティクスは、DB2 と同じです。 'N' このデータ・ソースのブランク埋め込みなしの比較セマンティクスは、DB2 と同じではありません。	'N'

後入れ先出しの機会に影響する照会特性

照会では、複数のデータ・ソースにあるニックネームを使用する SQL 演算子を参照できます。 DB2 が、セット演算子 (たとえば UNION) などの 1 つの演算子を使用して、参照された 2 つのデータ・ソースからの結果を結合しなければならない場合、この操作は DB2 で実行される必要があります。この演算子は、リモート・データ・ソースで直接に評価することはできません。

後入れ先出し分析による決定の分析と理解

SQL ステートメントを書き直すと、DB2 照会処理のために、さらに後入れ先出しの機会が提供されます。このセクションでは、照会が評価される位置を判別するためのツールを紹介し、照会分析と関連する一般的な質問 (および調査を提案する分野) をリストし、最後にデータ・ソースのアップグレードについて短く説明します。

照会が評価される位置の分析: DB2 には、照会が評価される箇所を示すユーティリティーが 2 つ付属しています。

- Visual Explain。このツールは、**db2cc** または **db2vexp** コマンドで開始します。照会アクセス・プランのグラフを表示するには、このツールを使用してください。各演算子の実行位置は、演算子の詳細表示に示されます。

  照会を完全に後入れ先出しする場合、 RQUERY 演算子の上に RETURN 演算子が示されているはずです。この RETURN 演算子は、標準の DB2 演算子です。
  RQUERY 演算子は、連合データベースの操作に固有のものです。 RQUERY は SQL

SELECT ステートメントをデータ・ソースに送り、照会結果を検索します。この SELECT ステートメントは、データ・ソースによってサポートされている SQL ダイアレクトを使用して生成されます。該当するデータ・ソースのための有効な照会を含めることができます。

- SQL Explain。このツールは、**db2expln** または **dynexpln** コマンドで開始します。アクセス・プラン戦略をテキストとして表示するには、このツールを使用してください。

照会がデータ・ソースまたは DB2 で評価される理由: このセクションでは、一般的なプラン分析についての疑問と、後入れ先出しの機会を増やすために調査する分野をリストしています。主な疑問には、以下のものがあります。

- この述部はなぜリモートで評価されないのか?

  述部が全く選択的なものであり、これを使用して行をフィルターに掛け、ネットワーク通信量を減らせる場合に、このような疑問が発生します。リモートでの述部評価は、同じデータ・ソースの 2 つの表間での結合をリモートに評価できるかどうかにも影響します。

  調べる必要のある分野は、以下のものがあります。

  – 副照会の述部。この述部には、別のデータ・ソースに関係する副照会が含まれていますか? この述部には、このデータ・ソースでサポートされていない SQL 演算子に関係する副照会が含まれていますか? すべてのデータ・ソースが、述部でのセット演算子をサポートしているわけではありません。

  – 述部関数。この述部には、このリモート・データ・ソースで評価できない関数が含まれていますか? 関係演算子は、関数として分類されます。

  – 述部のバインド要件。この述部をリモートに評価する場合には、特定の値をバインドする必要がありますか? その必要がある場合、このデータ・ソースでの SQL 制限に違反していませんか?

  – 最適化のグローバル度。最適化プログラムの側で、ローカル処理の方が費用効率が高いと判断している可能性があります。詳細については、204ページの『リモートでの SQL 生成とグローバル最適化』を参照してください。

- GROUP BY 演算子がリモートに評価されないのはなぜか?

  いくつかの点を調べることができます。

  – GROUP BY 演算子への入力はリモートに評価されますか? 評価されない場合、入力を調べてください。

  – そのデータ・ソースには、この演算子についての何らかの制約事項がありますか? たとえば、以下の例があります。

    - GROUP BY 項目の数が限定されている
    - 結合する GROUP BY 項目のバイト・カウントが限定されている
    - GROUP BY リストでは列だけが指定される

  – そのデータ・ソースはこの SQL 演算子をサポートしていますか?

- 最適化のグローバル度。最適化プログラムの側で、ローカル処理の方が費用効率が高いと判断している可能性があります。詳細については、『リモートでの SQL 生成とグローバル最適化』を参照してください。

- セット演算子がリモートに評価されないのはなぜか?

  いくつかの点を調べることができます。

  - それぞれのオペランドはどちらも、同じリモート・データ・ソースで完全に評価されていますか? 評価されていない場合で、完全に評価しなければならない場合は、それぞれのオペランドを調べてください。
  - そのデータ・ソースには、このセット演算子についての何らかの制約事項がありますか? たとえば、ラージ・オブジェクトまたは長フィールドは、この特定のセット演算子への入力として有効ですか?

- ORDER BY 演算がリモートに評価されないのはなぜか?

  以下の点を考慮してください。

  - ORDER BY 演算への入力はリモートに評価されますか? 評価されない場合、入力を調べてください。
  - ORDER BY 文節には文字式が含まれていますか? 含まれる場合、リモート・データ・ソースの照合順序は、 DB2 の照合順序と同じではありませんか?
  - そのデータ・ソースには、この演算子についての何らかの制約事項がありますか? たとえば、ORDER BY 項目の数が限定されていませんか? データ・ソースは、ORDER BY リストに対しての指定を列に制限していませんか?

データ・ソースのアップグレードとカスタマイズ: DB2 SQL コンパイラーには、データ・ソース SQL サポートについての情報が多数含まれていますが、データ・ソースはアップグレードまたはカスタマイズできるため、そのようなデータをたびたび調整する必要があります。その場合、ローカル・カタログ情報を変更することにより、機能の拡張を DB2 に通知してください。カタログを更新するときには、 DB2 DDL ステートメント (CREATE FUNCTION MAPPING や ALTER SERVER など) を使用します。詳細については、*SQL 解説書* を参照してください。

リモートでの SQL 生成とグローバル最適化

このフェーズは、照会を評価するためのグローバルで最適なアクセス戦略の作成に役立ちます。連合データベースの照会の場合、アクセス戦略には、元の照会を一連のリモート照会単位に分解してから結果を結合することが関係する場合があります。

後入れ先出し分析の出力を推奨値として使用することにより、最適化プログラムはそれぞれの操作が DB2 でローカルに評価されるのか、それともデータ・ソースでリモートに評価されるのかを決定します。この決定は、コスト・モデルの出力に基づくものです。コスト・モデルには、操作を評価するときのコストだけではなく、 DB2 とデータ・ソース間でデータまたはメッセージを伝送するときのコストも含まれています。

目標は最適化された照会を作成することです。ただし、グローバルな最適化からの出力には多くの要因が影響するため、照会のパフォーマンスにも影響があります。主な要因として、サーバー特性とニックネーム特性の 2 つを説明します。

サーバー特性 / グローバル最適化に影響するオプション

グローバル最適化に影響することのあるデータ・ソース・サーバーの要因には、以下のものがあります。

- CPU 速度の相対比率

  データ・ソースの CPU 速度が DB2 の CPU と比較してどの程度速いのか、あるいは遅いのかを示すには、 *cpu_ratio* サーバー・オプションを使用します。比率が低ければ、データ・ソースのワークステーション CPU は、 DB2 のワークステーション CPU よりも速いということです。比率が低い場合、DB2 最適化プログラムは、データ・ソースに対して後入れ先出しによる CPU 集約操作を実行しようとします。 この比率についての詳細は、 105ページの『連合データベース照会に影響を与えるサーバー・オプション』を参照してください。

- 入出力速度の相対比率

  データ・ソースのシステム入出力速度が DB2 のシステムと比較してどの程度速いのか、あるいは遅いのかを示すには、 *io_ratio* サーバー・オプションを使用します。比率が低ければ、データ・ソースのワークステーション入出力速度は、 DB2 のワークステーション入出力速度よりも速いということです。比率が低い場合、DB2 最適化プログラムは、データ・ソースに対して後入れ先出しによる入出力集約操作を実行します。 この比率についての詳細は、 105ページの『連合データベース照会に影響を与えるサーバー・オプション』を参照してください。

- DB2 とデータ・ソース間の通信速度

  ネットワーク容量を表示するには、 *comm_rate* サーバー・オプションを使います。比率が低い (DB2 とデータ・ソース間のネットワーク通信が遅いことを示す) 場合、DB2 最適化プログラムは、このデータ・ソースとの間でやりとりするメッセージ数を減らそうとします。比率を 0 に設定すると、最適化プログラムは、必要なネットワーク通信量が最小となる照会を作成します。この比率についての詳細は、 105ページの『連合データベース照会に影響を与えるサーバー・オプション』を参照してください。

- データ・ソースの照合順序

  データ・ソースの照合順序が、ローカル DB2 の照合順序と一致しているかどうかを示すには、 *collating_sequence* サーバー・オプションを使用します。このオプションを 'Y' に設定しないと、最適化プログラムは、このデータ・ソースから検索したデータが整列されていないと見なします。照合順序とパフォーマンスについての詳細は、198ページの『照合順序』を参照してください。

- リモート・プラン・ヒント

  プラン・ヒントがデータ・ソースでサポートされているかどうかを示すには、 *plan_hints* サーバー・オプションを使用します。プラン・ヒントはステートメントの

一部分であり、データ・ソース最適化プログラムについての追加情報を提供します。特定の照会タイプについてこの情報を利用すれば、照会パフォーマンスを改善することができます。プラン・ヒントは、データ・ソース最適化プログラムが索引を使用するかどうか、どの索引を使用するか、またはどの表結合順序を使うかを判別するのに役立ちます。

プラン・ヒントが使用可能であれば、データ・ソースに送信される照会には、追加情報が含まれます。たとえば、プラン・ヒントを使用して Oracle 最適化プログラムへ送信するステートメントは、次のようになります。

```
SELECT /*+ INDEX (table1, t1index)*/
   col1
   FROM table1
```

プラン・ヒントは、ストリング `/*+ INDEX (table1, t1index)*/` です。

- DB2 最適化プログラムの知識ベースでの情報

  DB2 には最適化プログラムの知識ベースがあり、そこには、固有のデータ・ソースについてのデータが含まれています。 DB2 最適化プログラムは、特定の DBMS で生成できないリモート・アクセス・プランを生成しません。つまり DB2 は、リモート・データ・ソースでの最適化プログラムが理解できない、あるいは受け入れられないプランの生成を避けます。

グローバル最適化に影響するニックネーム特性

以下のセクションでは、グローバル最適化に影響する、ニックネームに特有の要因を説明します。

索引についての考慮事項: DB2 は、データ・ソースにある索引についての情報を使用して、照会を最適化することができます。この理由のため、DB2 で使用可能な索引情報が最新のものであることが重要となります。ニックネームの作成時に、まずニックネームのための索引情報を獲得します。視点のニックネームについては、索引情報が収集されることはありません。

ニックネームに関する索引仕様の作成: ニックネームのために索引仕様を作成することができます。索引仕様では、 DB2 最適化プログラムが使用する索引定義 (実際の索引ではない) をカタログ内に作成します。索引仕様を作成するには CREATE INDEX SPECIFICATION ONLY ステートメントを使用します。ニックネームに対する索引仕様を作成する構文は、ローカル表で索引を作成する構文と似ています。詳細については、*管理の手引き: 計画* を参照してください。

以下の場合に、索引仕様の作成を考慮してください。

- DB2 が、ニックネームの作成時にデータ・ソースから索引情報を検索できない場合。
- 視点のニックネームのために索引が必要な場合。
- DB2 最適化プログラムで、ネストされたループ結合の内部表として特定のニックネームを使うようにする場合。索引が存在しない場合、ユーザーは結合列上に索引を作成できます。

視点のニックネームに対する CREATE INDEX ステートメントを発行する前に、必要事項を考慮してください。 1 つのケースとして、視点が索引付きの表での単なる SELECT である場合、ニックネームに関する索引のうち、表の索引と一致するもの (ローカルに) をデータ・ソースで作成すると、照会のパフォーマンスを著しく改善することができます。ただし、索引を、単なる SELECT ステートメントではない視点 (たとえば、 2 つの表を結合することにより作成する視点) でローカルに作成すると、照会のパフォーマンスが低下する可能性があります。たとえば、2 つの表の結合である視点で索引を作成する場合、最適化プログラムは、ネストされたループ結合の内部要素として、その視点を選択する場合があります。この結合は複数回評価されるため、照会のパフォーマンスは低下します。別の方法としては、データ・ソースの視点で参照される表ごとにニックネームを作成し、両方のニックネームを参照する部分視点を DB2 で作成することができます。

カタログ統計についての考慮事項: カタログ統計には、ニックネーム全体のサイズと、関連する列での値の範囲が示されます。これらの統計は、ニックネームを含む照会を処理するときのコストを最小にする方法を計算する場合に、最適化プログラムによって使用されます。ニックネーム統計は、表統計と同じカタログ視点に格納されます。統計のタイプとそれらをローカルに更新する方法については、 113ページの『第5章 システム・カタログ統計』と 135ページの『表統計とニックネーム統計の更新の規則』を参照してください。

DB2 は、データ・ソースに保持されている統計データを検索することはできますが、データ・ソースにある既存の統計データに対する更新を自動的に検出することはできません。さらに、DB2 には、オブジェクト定義を処理するための機構、あるいはデータ・ソースにあるオブジェクトに対する構造的な変更 (列の追加) を処理するための機構が備えられていません。オブジェクトの統計データまたは構造データに変更がある場合、以下の 2 つから処置を選択することができます。

- データ・ソースで RUNSTATS と同等の機能を実行する。その後、現在のニックネームを除去し、ニックネームを再作成します。この方法は、構造情報が変更された場合に使用してください。
- SYSSTAT.TABLES 視点の統計を手動で更新する。この方法では、実行する必要のあるステップは少なくなりますが、構造情報が変更された場合には機能しません。

グローバル最適化の決定の分析と理解

このセクションでは、照会の最適化を分析するためのツールを紹介し、照会の最適化と関連する一般的な質問 (および調査を提案する分野) を示します。

照会の最適化の分析: DB2 には、グローバル・アクセス・プランを示すユーティリティーが 2 つ付属しています。

- Visual Explain。このツールは、**db2cc** または **db2vexp** コマンドで開始します。照会アクセス・プランのグラフを表示するには、このツールを使用してください。各演算子の実行位置は、演算子の詳細表示に示されます。 RQUERY (選択操作) 演算子により、データ・ソースごとに生成されたリモート SQL ステートメントを見つけるこ

ともできます。各演算子の詳細を調べるなら、 DB2 最適化プログラムが各演算子に関する入出力として見積もる行数が分かります。さらに、各演算子を実行するときの、通信コストを含めた見積コストも確認できます。詳細については、573ページの『付録C. SQL EXPLAIN ツール』を参照してください。

- SQL Explain。このツールは、db2expln または dynexpln コマンドで開始します。アクセス・プラン戦略をテキストとして表示するには、このツールを使用してください。SQL Explain では、コスト情報は示されません。ただし、リモート Explain 機能によってサポートされているデータ・ソース用に、リモート最適化プログラムによって生成されるアクセス・プランを入手できます。詳細については、573ページの『付録C. SQL EXPLAIN ツール』を参照してください。

DB2 最適化の決定: このセクションでは、最適化についての疑問と、パフォーマンスを改善するために調べる主な分野をリストします。主な疑問には、以下のものがあります。

- 同じデータ・ソースの 2 つのニックネームの結合がリモートに評価されないのはなぜか?

  調べる必要のある分野は、以下のものがあります。

  – 結合操作。データ・ソースでは結合操作をサポートしていますか?
  – 結合述部。その結合述部はリモート・データ・ソースで評価されますか? 評価されない場合、その結合述部を調べてください。詳細については、203ページの『照会がデータ・ソースまたは DB2 で評価される理由』を参照してください。
  – (Visual Explain を使用した) 結合結果の行数。この結合により作成される行数は、2 つのニックネームを結合するときよりも多いですか? 数値は有効なものですか? 数値が無効である場合、ニックネーム統計を手動で更新することを考慮してください (SYSSTAT.TABLES)。

- GROUP BY 演算子がリモートに評価されていないのはなぜか?

  調べる必要のある分野は、以下のものがあります。

  – 演算子構文。その演算子が、リモート・データ・ソースで評価できることを確認してください。詳細については、203ページの『照会がデータ・ソースまたは DB2 で評価される理由』を参照してください。
  – 行数。 Visual Explain を使用して、 GROUP BY 演算子による入出力の見積行数を調べてください。この 2 つの数値は近いものですか? 近いものである場合、DB2 最適化プログラムは、この GROUP BY をローカルに評価する方が効率的であると見なします。さらに、これら 2 つの数値は有効なものですか? 数値が無効である場合、ニックネーム統計を手動で更新することを考慮してください (SYSSTAT.TABLES)。

- リモート・データ・ソースによってステートメントが完全に評価されないのはなぜか?

  DB2 最適化プログラムは、コスト・ベースの最適化を実行します。後入れ先出し分析で、各演算子はリモート・データ・ソースで評価できることが示されても、最適化プログラムは、グローバル最適化プランを生成するときには、コストによる見積もりを

利用します。そのプランに関与する要因は非常にたくさんあります。たとえば、リモート・データ・ソースが元の照会でそれぞれの操作を処理できても、リモート・データ・ソースの CPU 速度が DB2 の CPU 速度よりはるかに遅いため、 DB2 で操作を実行する方が有利であることが分かる場合があります。結果に満足できない場合、SYSCAT.SERVEROPTIONS のサーバー統計を調べてください。

- 最適化プログラムによって生成され、リモート・データ・ソースで完全に評価されるプランのパフォーマンスが、リモート・データ・ソースで直接に実行される元の照会よりもはるかに劣るのはなぜか?

  調べる必要のある分野は、以下のものがあります。

  – DB2 最適化プログラムによって生成されたリモート SQL ステートメント。これが元の照会と等しいことを確認します。述部の順序変更がないか調べます。適切な照会最適化プログラムであれば、照会での述部の順序に影響されることはありません。ただし、すべての DBMS 最適化プログラムが同じ動作をするわけではないので、リモート・データ・ソースの最適化プログラムによっては、入力述部の順序に基づいて異なるプランを生成してしまう可能性があります。これは、そのリモート最適化プログラムにおける固有の問題です。 DB2 への入力時に述部の順序を変更するか、そのリモート・データ・ソースのサービス団体に援助を依頼してください。

    また、述部が置き換えられていないか調べます。適切な照会最適化プログラムであれば、等価な述部の置き換えに影響されることはありません。ただし、すべての DBMS 最適化プログラムが同じ動作をするわけではないので、リモート・データ・ソースの最適化プログラムによっては、入力述部に基づいて異なるプランを生成してしまう可能性があります。たとえば、最適化プログラムによっては、述部の変換防止ステートメントを生成できないものもあります。

  – 戻された行の数。この数値は、Visual Explain から入手できます。照会によって多数の行が戻される場合、ネットワーク通信量がボトルネックになっている可能性があります。

  – その他の関数。リモート SQL ステートメントに、元の照会と比較して余分な関数が含まれていませんか? その余分な関数の中には、データ・タイプを変換するために生成されたものが含まれている可能性があります。それの関数が必要であることを確かめてください。

第7章 SQL Explain 機能

SQL Explain 機能は SQL コンパイラーの一部で、この機能を使用すると、静的または動的 SQL ステートメントのコンパイルを行う環境に関する情報を取得できます。取得した情報を使用して、SQL ステートメントの構造や、潜在的な実行のパフォーマンスを理解することができます。これには、次の情報が含まれます。

- 照会を処理する操作の順序
- コスト情報
- 述部および選択可能性の見積もり
- Explain 機能の実行時点の、 SQL ステートメントで参照されている全オブジェクトに関する統計

この情報は次のような場合に役に立ちます。

- 照会用に選択した実行プランを理解する
- アプリケーション・プログラムの設計を援助する
- アプリケーションを再バインドするべき時期を判別する
- データベース設計を援助する

この部分では、次の点について説明します。

- 212ページの『EXPLAIN ツールの選択』
- 214ページの『SQL Explain 機能の使用』
- 216ページの『Explain の初歩的な概念』
- 219ページの『Explain 情報の編成方法』
- 224ページの『Explain データの獲得』
- 227ページの『Explain 出力の使用に関する指針』
- 229ページの『Visual Explain』
- 230ページの『SQL アドバイス機能』

Explain 出力はリレーショナル表に保管され、任意で Visual Explain ツールを使用して表示できる形式で保管されます。関係する Explain 表に対して実行される照会を見つけるには、 Explain 表を使用することを考慮してください。 Explain 機能で使用される表およびそれらの表の作成方法については、 541ページの『付録B. Explain 表と定義』を参照してください。

EXPLAIN ツールの選択

DB2 では産業界で最も広範囲の Explain 機能を提供します。これには、Explain 済みの SQL ステートメントについて選択したアクセス・プランに関する詳細最適化プログラム情報が含まれています。 Explain 情報を獲得しアクセスするのに必要な柔軟性を提供するため、いくつかの方法が提供されています。

アクセス・プランを徹底的に分析するために使用できる詳細最適化プログラム情報は、実際のアクセス・プランとは別の Explain 表に保持されます。 Explain 表から情報を入手するには、次の 3 つの方法があります。

1. 自分自身の照会を作成する (541ページの『付録B. Explain 表と定義』の説明にある Explain 表に基づいて行う)
2. *db2exfmt* ツールを使用する
3. Visual Explain を使用する (Explain スナップショット情報を表示する)

Explain 表はサポートされるすべてのプラットフォームでアクセス可能であり、表には静的 SQL と動的 SQL の両方に関する情報が含まれています。 SQL ステートメントを使用すると、簡単な操作で出力したり、別の照会と比較したり、または同じ照会を時間外で比較したりできる Explain 表にアクセスすることができます。事前定義された書式で表示された Explain 表から情報を得る場合は、 *db2exfmt* ツールを使用することができます。このツールの詳細については、619ページの『付録D. db2exfmt - Explain 表フォーマット・ツール』を参照してください。代替方法として、表にアクセスするための独自のステートメントを作成することもできます。

注: このツール (および、*db2batch*、*dynexpln*、*db2vexp*、*db2_all* といったツール) は、*sqllib* ディレクトリーの *misc* サブディレクトリー内にあります。このツールがこのパスから移動されている場合には、上述したコマンド行入力は作動しない場合があります。

Visual Explain を使用すると、グラフィック・インターフェースを介して、アクセス・プランの分析や Explain 表からの最適化プログラム情報を分析することができます。静的 SQL と動的 SQL はどちらもこのツールを用いて分析することができます。 Visual Explain は一般にコントロール・センター内から呼び出されます。コントロール・センターは、コマンド行に *db2cc* と入力すると使用できます。また、Visual Explain は、コマンド行に *db2vexp* コマンドを入力して、 1 つの SQL ステートメントごとに直接呼び出すこともできます。一部のプラットフォームでは、Visual Explain は、 DB2 ユニバーサル・データベース・フォルダー内からフォルダーを使用して呼び出すことができます。 Visual Explain はサポートされるプラットフォームすべてで使用できるわけではありません。 Visual Explain がサポートされているかどうかを調べるには、 *概説およびインストール* を参照してください。 Visual Explain では、別のプラットフォームで収集または取得したスナップショットを見ることができます。たとえば、Windows NT クライアントでは、 DB2 (HP-UX サーバー版) で生成されたスナップショットをグラフ化することができます。 これが可能なのは、両方のプラットフォームがバージョン 5

レベル以降の場合に限られます。 Visual Explain からの出力は、簡単に操作して詳しい分析を行えるものではなく、他のアプリケーションでアクセスできる情報でもありません。 *db2vexp* コマンドの詳細については、コマンド行で *db2vexp -h* と入力するか、 コマンド*解説書* を参照してください。 Visual Explain の詳細については、*db2cc* を入力して、 コントロール・センターのオンライン・ヘルプを参照してください。

静的 SQL ステートメントのアクセス・プランに関する情報はパッケージの一部として生成され、システム・カタログに保管されます。 1 つまたは複数のパッケージで利用できるアクセス・プラン情報を見るには、コマンド行から *db2expln* ツールを使用できます。 *db2expln* は、選択したアクセス・プランを実際に具体化したものを示します。最適化プログラム情報については示しません。

dynexpln ツールは内部で *db2expln* を使用しますが、パラメーター・マーカーが入っていない動的 SQL ステートメントに対して素早い方法で Explain を実行します。 *dynexpln* 内部からの *db2expln* の使用は、入力 SQL ステートメントを疑似パッケージ内の静的ステートメントに変換することによって行われます。これが実行されても、情報は必ずしも完全に正確であるとは限りません。完全に正確な情報が必要な場合には、214ページの『SQL Explain 機能の使用』で説明されている Explain 機能を使用するようにしてください。

db2expln ツールは、生成された実際のアクセス・プランを調べることにより、実行時にどのような操作が行われるのかに関する比較的コンパクトで英語式の概要を提供します (コードの生成方法については、 150ページを参照してください)。 *db2expln* の使用と出力の解釈に関する付加的な情報については、 573ページの『付録C. SQL EXPLAIN ツール』を参照してください。

表16 では、 DB2 Explain 機能と共に使用できる他のツールとそれらの個々の特性について要約します。この表を使用して、使用中の環境とニーズに最も適したツールを選択してください。

表 16. Explain 機能ツール

希望する特性	Visual Explain	db2vexp	Explain 表	db2exfmt	db2expln	dynexpln
GUI インターフェース	可	可				
テキスト出力				可	可	可
「簡易」静的 SQL 分析					可	
サポートされる静的 SQL	可		可	可	可	
サポートされる動的 SQL	可	可	可	可		可*
サポートされる CLI アプリケーション	可		可	可		
DRDA アプリケーション・リクエスターで使用可能			可			

表 16. Explain 機能ツール (続き)

希望する特性	Visual Explain	db2vexp	Explain 表	db2exfmt	db2expln	dynexpln
詳細最適化プログラム情報	可	可	可	可		
複数ステートメントの分析に適合			可	可	可	可
アプリケーション内部からアクセス可能な情報			可			
注: * db2expln を間接的に使用します。制限がいくつかあります。						

SQL Explain 機能の使用

Explain 情報を獲得する手段として、次のものを使用します。

1. EXPLAIN および EXPLSNAP BIND/PREP オプション
2. CURRENT EXPLAIN MODE および CURRENT EXPLAIN SNAPSHOT 特殊レジスター
3. EXPLAIN SQL ステートメント
4. *db2vexp* ツール (Visual Explain を直接呼び出して情報を表示する)

Explain データを収集し使用する理由には、次の 4 つが挙げられます。

1. 照会を満たすためにデータベース・マネージャーが実行しなければならないステップ (アクセス・プラン) を理解する。 160ページの『データ・アクセス概念および最適化』は、 Explain 出力を理解する場合に参照する必要がある情報を提供しています。
2. パフォーマンス調整の優先事項を評価するのに役に立つ。照会のパフォーマンスを向上させるのに役立つ、多数のアクションがあります。これらの可能なアクションの大部分は、次の項目のサブトピックに記載されています。
   - 43ページの『第3章 アプリケーションについての考慮事項』
   - 91ページの『第4章 環境についての考慮事項』
   - 113ページの『第5章 システム・カタログ統計』

   これらの領域のいずれかで変更を行ってから、SQL Explain 機能を使用して、変更によって選択したアクセス・プランに与える影響があるならば、それを判別することができます。たとえば、97ページの『照会最適化に対する索引付けの影響』に記載されている推奨事項に基づいて索引を追加した場合、 Explain データを参考にして、実際には索引が期待したとおりに使用されているのかどうかを判別することができます。

Explain 出力は、選択したアクセス・プランとその相対コストを判別できるようにする情報を提供しますが、照会のパフォーマンスの向上を正確に測定する唯一の方法は、 317ページの『第12章 ベンチマーク・テスト』に記載されたベンチマーク・テスト技法を使用することです。

3. 照会パフォーマンスが変化した理由を理解するには、影響を分析するために、変更の前と後の Explain 情報が必要です。したがって、SQL ステートメントをデータベースにコンパイルする際に、次のことを行わなければなりません。
   - Explain 機能を使用して、変更前の計画情報を獲得し、変更後の Explain 表を保管する。または、db2exfmt Explain ツールからの出力を保管する。
   - この情報を表示するために Visual Explain にアクセスしない場合またはアクセスできない場合は、現行のカタログ統計を保管し印刷する。あるいは、そのどちらかを行う。 (141ページの『実動データベースのモデル化』に記載された db2look 生産性向上ツールは、このタスクの実行に使用できます。)
   - データ定義言語 (DDL) ステートメント (CREATE TABLE、CREATE VIEW、CREATE INDEX、CREATE TABLESPACE のステートメントを含む) を保管し印刷する。あるいは、そのどちらかを行う。

   上記の情報では、今後の分析の参照点として使用できるように、*前* ピクチャーを提供します。動的 SQL ステートメントの場合は、アプリケーションを最初に実行するときに、この情報を収集することもできます。静的 SQL ステートメントの場合は、バインド時にこの情報を収集することもできます。

   パフォーマンスの変更理由を分析する場合は、*前* データを、分析を開始しているときに照会と環境について収集した情報 (*後* データ) と比較することができます。

   簡単な例として、分析によって、索引がアクセス・パスの一部として使用されていないことが示されたとします。カタログ統計情報を使用すると、Visual Explain では、索引レベルの数 (NLEVELS 列) が、照会が最初にデータベースにバインドされたときよりもかなり大きくなっていることに気付きます。そこで、次のことを選択することになります。
   - 索引を再編成する。
   - 表と索引の新規の統計を収集する。
   - 照会を再バインドするときに Explain 情報を収集する。

   これらのアクションの後で、索引がアクセス・プランで再度使用されており、照会のパフォーマンスがもう問題にはなっていないことがわかるでしょう。

Explain の初歩的な概念

Explain 情報を使用して、最適化プログラムが 160ページの『データ・アクセス概念および最適化』に記載された選択に基づいて選択したアクセス・プランを分析することができます。たとえば、Explain 情報は、最適化プログラムが索引走査 (161ページの『索引走査の概念』を参照) を選択したことを示す場合があります。さらに、次の事柄を判別することもできます。

- 探索基準として使用した索引列の数 (171ページの『範囲区切り述部と索引検索引き数述部』に記載)
- 索引専用アクセスが使用されるかどうか (166ページの『索引のみのアクセス』に記載)
- ページの読み取りにリスト事前取り出しが使用されるかどうか (256ページの『リスト事前取り出しについて』に記載)

別の例として、Explain 情報は、2 つの表の結合方法を理解するのに役に立ちます。

- 結合方法
- 表の結合順序
- 分類の出現とタイプ

SELECT、SELECT INTO、UPDATE、INSERT、VALUES、VALUES INTO、および DELETE SQL ステートメントに対して Explain を使用できますが、 Explain を主に使用するのは、ステートメントの SELECT 部分のアクセス・パスを観察する場合です。

SQL 照会を実行するために、データベース・マネージャーは一般に次の事柄を行います。

- 1 つまたは複数のデータ・オブジェクト (表、索引、またはその両方) を使用する。
- 1 つまたは複数の操作 (たとえば、表走査、索引走査、および結合) を実行する。
- 結果セットを呼び出しアプリケーションに戻す。

次のような単純な SQL 照会の場合、

```
SELECT DEPTNO, DEPTNAME
  FROM DEPARTMENT
```

実行されたステップのグラフィック表示が Visual Explain によって次のように表示されます。

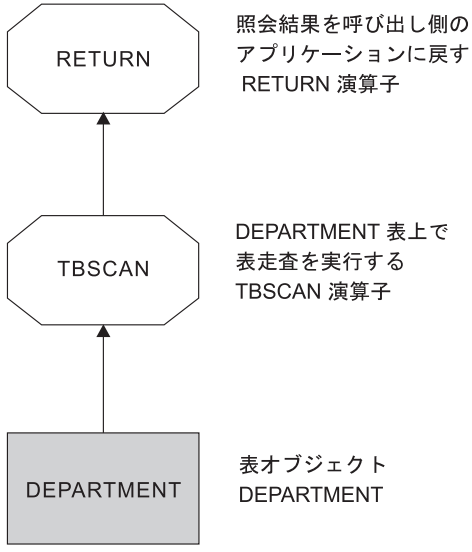


図 20. Explain 出力の図形表示

次のトピックでは、上記のオブジェクトおよび演算子について表示できる詳細のタイプを説明します。

- 『データ・オブジェクトの Explain 情報』
- 218ページの『データ演算子の Explain 情報』

データ・オブジェクトの Explain 情報

1 つのアクセス・プランは、 1 つまたは複数のデータ・オブジェクトを使用して SQL ステートメントを実行します。

オブジェクト統計: Explain 機能は、オブジェクトに関して次のような事柄を記録します。

- 作成時刻
- オブジェクトの統計が最後に収集された時刻 (113ページの『第5章 システム・カタログ統計』を参照)
- オブジェクト内のデータが順番に並べられたかどうかを示す (表または索引オブジェクトのみ)。
- オブジェクトの列数 (表または索引オブジェクトのみ)。
- オブジェクト内の行数の見積もり (表または索引オブジェクトのみ)
- オブジェクトがバッファー・プール内で占有するページ数
- 指定された表スペース (このオブジェクトが保管されている表スペース) にランダム入出力を 1 回行うための、合計見積オーバーヘッド (ミリ秒単位)

- 指定された表スペースから 4K ページを読み取るための、見積転送速度 (ミリ秒単位)
- 事前取り出しサイズおよびエクステント・サイズ (4K ページ単位)
- 索引を用いたデータ・クラスター化の程度
- このオブジェクトの索引が使用する葉ページの数、および木のレベル数
- このオブジェクトの索引内の個別全キー値の数
- 表内の合計オーバーフロー・レコード数

データ演算子の Explain 情報

単一のアクセス・プランでは、 SQL ステートメントを実行し、結果をユーザーに戻すために、データ上でいくつかの操作を実行します。 SQL コンパイラーは必要な操作を判別します。たとえば、表走査、索引走査、ネストされたループの結合、またはグループ化演算子などです。これらの演算子の大多数の詳細は、
160ページの『データ・アクセス概念および最適化』に示されています。

アクセス・プランで使用されるさまざまな演算子を示すことに加えて、 Explain 情報は各演算子およびアクセス・プランの累積効果にも使用可能です。

コスト情報の見積もり: 演算子については、以下に示した累積コストの見積もりを表示することができます。これらのコストは選択したアクセス・プランに関するもので、情報が獲得されている演算子までのコストが表示されます。

- 合計コスト (タイマーオン)
- ページ入出力の数
- CPU 命令の数
- 最初の行を取り出すためのコスト (タイマーオン)。必要な初期オーバーヘッドがあるならば、それも含む。
- コミュニケーション・コスト (フレーム単位)。

タイマーオン は、架空の相対計測単位です。タイマーオンは、最適化プログラムによって、内部値、たとえばデータベースの使用に応じて変わる統計などに基づいて決定されます。そのため、タイマーオンの見積コストが決定されるたびに、 SQL ステートメントのタイマーオン計測が同じになるという保証はありません。

演算子の特性: 以下に示す情報は Explain 機能によって記録されるもので、各演算子の特性を記述します。

- アクセスされた表のセット
- アクセスされた列のセット
- データが順番に並べられた列 (最適化プログラムが、この順番付けを後続の演算子が使用できると判別する場合)
- 適用された述部のセット
- 戻される行数の見積もり (カーディナリティー)

Explain 情報の編成方法

Explain インスタンスの概念について、すべての Explain 情報が編成されています。 Explain インスタンスは 1 つまたは複数の SQL ステートメントごとに、 1 回の Explain 機能の呼び出しを示します。 Explain インスタンスは次の事柄に関する Explain 情報を示します。

- **静的** SQL ステートメントでは、 1 つのパッケージに入っているすべての適格な SQL ステートメント
- 増分バインド SQL ステートメントでは、 1 つの特定の SQL ステートメント
- **動的** SQL ステートメントでは、1 つの特定の SQL ステートメント
- 各 EXPLAIN SQL ステートメント (動的か静的のどちらか)

1 つの Explain インスタンス内で獲得された Explain 情報には、 SQL コンパイル環境ならびにコンパイルされる SQL ステートメントを実行するために選ばれたアクセス・プランが入っています。 Explain 情報は次の 3 つのサブセットに編成されます。

Explain インスタンス情報 Explain インスタンスごとに獲得されたコンパイル環境情報

Explain スナップショット情報 Visual Explain が使用する情報。

Explain 表情報 Explain 表情報が要求されたときに収集された情報。

Explain インスタンス情報

Explain インスタンス情報は EXPLAIN_INSTANCE 表に保管されます。 Explain インスタンスの内部で Explain が実行された各 SQL ステートメントに関する特定の付加的な情報は、 EXPLAIN_STATEMENT 表に保管されます。

Explain インスタンスの識別: 以下の情報を用いると、それぞれの Explain インスタンスを固有に識別し、 SQL ステートメントの情報をこの機能の特定の呼び出しに関連づけることができます。

- Explain 情報を要求したユーザー
- Explain 要求が開始された時刻
- Explain が実行された SQL ステートメントが入っていたパッケージの名前
- Explain が実行された SQL ステートメントが入っていたパッケージのスキーマ
- スナップショットがその Explain 要求の一部であるかどうかを示す情報

環境設定: SQL コンパイラーが照会をどのように最適化したかに関係する環境情報が取得されます。環境情報には、以下のものが含まれます。

- 使用している DB2 のレベルの、バージョンおよびリリースの番号。

- 照会のコンパイルに使用される並列操作の程度。 CURRENT DEGREE 特殊レジスター、DEGREE バインド・オプション、SET RUNTIME DEGREE API、および *dft_degree* 構成パラメーターを使用すると、特定の照会のコンパイル時に使用される並列操作の程度を決めることができます。
- SQL ステートメントが動的と静的のどちらか。
- 照会のコンパイルに使用される照会最適化クラス。詳細については、68ページの『最適化クラスの調整』を参照してください。
- 照会のコンパイル時に指定されたカーソル・ブロック化のタイプ。カーソルの詳細については、*SQL 解説書* を参照してください。カーソルのブロック化の詳細については、79ページの『行のブロック化』を参照してください。
- 照会のコンパイル時に使用された分離レベル。詳細については、43ページの『並行性』を参照してください。
- 照会がコンパイルされたときのさまざまな構成パラメーターの値。照会の最適化に影響を与える可能性のある構成パラメーター (Explain スナップショットがとられたときに記録される以下のパラメーターを含む) の詳細については、 91ページの『照会最適化に影響する構成パラメーター』を参照してください。
  - 348ページの『バッファー・プール・サイズ (buffpage)』
  - 363ページの『分類ヒープ・サイズ (sortheap)』
  - 401ページの『活動アプリケーションの平均数 (avg_appls)』
  - 351ページの『データベース・ヒープ (dbheap)』
  - 357ページの『ロック・リスト用最大ストレージ (locklist)』
  - 389ページの『自動調整前のロック・リストの最大パーセント (maxlocks)』
  - 481ページの『CPU 速度 (cpuspeed)』
  - 480ページの『通信帯域幅 (comm_bandwidth)』

SQL ステートメントの識別: Explain インスタンスごとに、複数の SQL ステートメントに Explain が実行される場合があります。 Explain インスタンスを固有に識別する情報に加えて、次の情報は個々の SQL ステートメントをそれぞれ識別するのに役に立ちます。

- ステートメントのタイプ。SELECT、DELETE、INSERT、UPDATE、定位置 DELETE、定位置 UPDATE。
- SYSCAT.STATEMENTS カタログ視点に記録された、 SQL ステートメントを発行するパッケージのステートメントおよびセクション番号。

EXPLAIN_STATEMENT 表では、QUERYTAG フィールドと QUERYNO フィールドが ID を含み、 Explain 処理の一部として設定されます。

CLP または CLI セッション中に実行依頼された動的 Explain SQL ステートメントの場合、 EXPLAIN MODE または EXPLAIN SNAPSHOT が活動状態になっていると、

QUERYTAG が 『CLP』か 『CLI』 に設定されます。これが起こると、各ステートメントごとに 1 かそれ以上ずつ大きくなっている番号のデフォルトが QUERYNO になります。

その他の動的 Explain SQL ステートメント (CLP、CLI 以外から、または EXPLAIN SQL ステートメントを使用) の場合は、 QUERYTAG がブランクに設定され、QUERYNO が常に 『1』 になります。

コスト見積もり: Explain が実行されたステートメントごとに、選択されたアクセス・プランを実行するのに要する相対コストの見積もりが記録されます。このコストはタイマーオン という架空の相対的な計測単位を用いて指定します。経過時間の見積もりは、次の理由で提供**されません**。

- SQL 最適化プログラムは経過時間ではなく、ソースの消費を見積もる。
- 最適化プログラムは、経過時間に影響を及ぼす可能性のある因数をすべてモデル化するわけではないが、アクセス・プランの効果性に影響を及ぼさない因数を無視する。経過時間は実行時の因数の数に影響を**受けます**。これには、次のものが含まれます。システムの作業負荷、リソース競合の量、並列処理と入出力の量、行をユーザーに戻すためのコスト、およびクライアントとサーバーの間の通信時間。

ステートメント・テキスト: Explain が実行されたステートメントごとに、 SQL ステートメントのテキストが 2 つのバージョンで記録されます。 1 つ目のバージョンは SQL コンパイラーが受け取ったテキストです。 2 つ目のバージョンは、照会の内部コンパイラー表示から逆変換されたバージョンのステートメント・テキストです。この変換は他の SQL ステートメントに似ているように見えますが、必ずしも正しい SQL 構文に従っているわけでも、内部表示の実際の内容を全体として反映しているわけでも**ありません**。この変換は、単に、 SQL 最適化プログラムがアクセス・プランを選択する元となる SQL コンテキストを理解できるようにするために提供されています。ユーザー作成のステートメント・テキストを SQL ステートメントの内部表示と比較すると、最適化を向上させるために SQL コンパイラーが再書き込みされた方法を理解するのに役に立ちます。 (151ページの『SQL コンパイラーによる照会書き直し』を参照してください。) さらに、トリガーや制約などのステートメントに影響を及ぼす、環境内の他の要素も示します。この 『最適化された』 テキストが使用するキーワードの一部は、以下のようなものです。

$Cn	派生列の名前。n は整数値を表します。
$CONSTRAINT$	コンパイル中の元の SQL ステートメントに追加された制約の名前を示すタグ。 $WITH_CONTEXT$ 接頭部と組み合わせて表示されます。
$DERIVED.Tn	派生表の名前。 n は整数値を示します。
$INTERNAL_FUNC$	Explain が実行された照会について SQL コンパイラーが使用しても、汎用にはできない機能があることを示すタグ。

$INTERNAL_PRED$	Explain が実行された照会のコンパイル中に SQL コンパイラーが追加した述部があることを示すタグ。繰り返しますが、このような述部は汎用にはできません。内部述部は、トリガーおよび制約の結果として元の SQL ステートメントに追加された付加的な文脈を満たすために、コンパイラーが使用します。
RID	特定の行の行識別名 (RID) 列を識別するためのタグ。
$TRIGGER$	コンパイル中に元の SQL ステートメントに追加されたトリガーの名前を示すタグ。$WITH_CONTEXT$ 接頭部と組み合わせて表示されます。
$WITH_CONTEXT$(...)	元の SQL ステートメントに付加的なトリガーまたは制約が追加されると、この接頭部がテキストの最初に表示されます。この接頭部の後に、 SQL ステートメントのコンパイルおよび解決に影響を与えるトリガーまたは制約の名前のリストが表示されます。

Explain スナップショット情報

Explain スナップショットが要求されると、 SQL 最適化プログラムが選択したアクセス・プランを説明する付加的な Explain 情報が記録されます。この情報は、Visual Explain が求める書式で EXPLAIN_STATEMENT 表の SNAPSHOT 列に保管されます。この書式は他のアプリケーションでは使用できません。

Explain スナップショット情報の内容に関する付加的な情報は、 Visual Explain 自体と次の箇所から使用できます。

- 217ページの『データ・オブジェクトの Explain 情報』
- 218ページの『データ演算子の Explain 情報』

Explain 表情報

Explain 表情報が要求されると、 SQL 最適化プログラムが選択したアクセス・プランを説明する付加的な情報が記録されます。この情報は以下の Explain 表に保管されます。

- EXPLAIN_ARGUMENT。この表は、演算子があれば、そのおのおのに関する固有の特性を表します。
- EXPLAIN_INSTANCE。この表は、すべての Explain 情報の主制御表です。 Explain 表中のデータの各行は、この表内のある固有の 1 行に明示的にリンクされます。Explain 対象の SQL ステートメントのソースに関する基本情報および環境情報は、この表に保持されます。

- EXPLAIN_OBJECT。この表は、SQL ステートメントを満たすために生成されるアクセス・プランに必要なデータ・オブジェクトを識別します。
- EXPLAIN_OPERATOR。この表には、SQL コンパイラーが SQL ステートメントを満たすために必要とするすべての演算子が含まれます。
- EXPLAIN_PREDICATE。この表は、どの述部が特定の演算子に適用されるかを識別します。
- EXPLAIN_STATEMENT。この表には、さまざまなレベルの Explain 情報に関する SQL ステートメントのテキストが含まれます。この表には、ユーザーが入力した元の SQL ステートメントと、その SQL ステートメントを満たすアクセス・プランを選択するのに最適化プログラムで使用されるバージョンとが保管されます。
- EXPLAIN_STREAM。この表は、個々の演算子とデータ・オブジェクトの間の入出力データ・ストリームを表します。データ・オブジェクト自体は、EXPLAIN_OBJECT 表に示されています。データ・ストリームに関連する演算子は、EXPLAIN_OPERATOR 表にあります。
- ADVISE_WORKLOAD。この表により、データベースへの作業負荷を記述できます。作業負荷の各行は 1 つの SQL ステートメントであり、関係する頻度によって記述されます。この表は、db2advis ツールおよび「索引 (Index)」ウィザードによって、作業および情報を取り出したり格納したりするときに使用されます。
- ADVISE_INDEX。この表には、推奨索引についての情報が格納されています。この表のデータは、SQL コンパイラー、db2advis ユーティリティー、「索引 (Index)」ウィザード、またはユーザーによって入力されます。この表は、次の 2 つの目的で使用します。
  - 推奨索引を入手する。
  - 提案された索引についての入力に基づき索引を評価する。

上記の表すべてがデフォルトに作成されるわけではありません。それらの表は、 sqllib サブディレクトリーの misc サブディレクトリーにある EXPLAIN.DDL スクリプトを実行することにより、作成することができます。 Explain 表および Advise 表が要求されているデータベースに接続してください。それからコマンド db2 -tf EXPLAIN.DDL を発行すると表が作成されます。必要に応じて、「索引 (Index)」ウィザードで表を自動的に作成することもできます。

Visual Explain の長方形の オブジェクト・ノードは、 EXPLAIN_OBJECT 表の行に対応します。 Visual Explain の八角形の 『演算子』 ノードは、 EXPLAIN_OPERATOR 表の行に対応します。演算子間または演算子のオブジェクト間の各リンクは、EXPLAIN_STREAM 表の行に対応します。

Explain 表情報は Explain スナップショットについて記録された情報によく似ていますが、この情報は標準 SQL ステートメントを用いてアクセスできる通常のリレーショナル表に保管されます。

Visual Explain アクセス・プラン・グラフと同様に、 Explain 表は、アクセス・プラン内の演算子とデータ・オブジェクトの間の関連を反映するように設計されています。次の図は、これらの表の間の関連を示します。

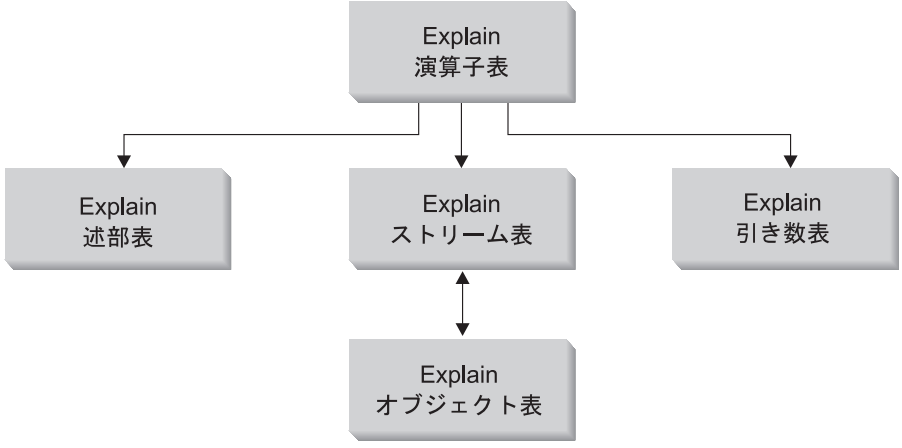


図 21. Explain 表の関連についての概要 (全部の表が示されているわけではありません)

複数のユーザーが共通に使用する Explain 表を持つことができます。 Explain 表は、1 人のユーザーに対して定義可能です。追加のユーザーごとに、定義された表を示す名前と同じ名前を使用して別名を定義することができます。共通の Explain 表を共有する各ユーザーには、それらの表に対する挿入許可が必要です。

Explain 表および表の作成方法の詳細については、 573ページの『付録C. SQL EXPLAIN ツール』を参照してください。 Explain 表情報の内容に関する付加的な情報は、次の箇所から使用できます。

- 217ページの『データ・オブジェクトの Explain 情報』
- 218ページの『データ演算子の Explain 情報』

`sqllib` ディレクトリーの `misc` サブディレクトリーで提供された `db2exfmt` ツールを使用して、 Explain 表の内容を読みやすい編成出力にフォーマットすることができます。

Explain データの獲得

SQL ステートメントの Explain データを獲得するには、 Explain 機能を呼び出す許可 ID と同じスキーマを使用して定義された、一連の Explain 表を持っていなければなりません。表の作成方法については、561ページの『Explain 表の表定義』を参照してください。

Explain 表情報の獲得

これらの表を定義しておくと、 SQL ステートメントがコンパイルされたときに Explain データが獲得され、 Explain データがすでに要求されています。

- **静的** SQL ステートメント、または**増分バインド** SQL ステートメントの場合、 EXPLAIN ALL か EXPLAIN YES オプションのどちらかが BIND または PREP コマンドで指定されると、 Explain 表情報が獲得されます。または、ソース・プログラムで静的 EXPLAIN SQL ステートメントが使用されます。

  注: 増分バインド SQL ステートメントは、実行時にコンパイルされると、実行時および非バインド実行時に Explain 表に置かれます。また、Explain 表への挿入の際に使用される Explain 表の修飾子および許可 ID は、パッケージ所有者のものであり、パッケージを実行する使用者のものではありません。

- **動的** SQL ステートメントの場合、次のいずれかの状況について、Explain 表情報が獲得されます。
  - EXPLAIN SQL ステートメント。 FOR SNAPSHOT 文節が使用されていないと、すべての Explain 情報が獲得され、 Explain 表に入れられます。

    EXPLAIN SQL ステートメントの例は次のようなものです。

    ```
    EXPLAIN PLAN FOR <any valid DELETE, INSERT, SELECT, SELECT INTO,
       UPDATE, VALUES, or VALUES INTO SQL statement>
    ```

  - CURRENT EXPLAIN MODE 特殊レジスターが YES に設定される。この設定により、SQL コンパイラーは Explain データを獲得し、 SQL ステートメントを実行して、照会の結果を戻します。
  - CURRENT EXPLAIN MODE 特殊レジスターが EXPLAIN に設定される。この設定により、SQL コンパイラーは Explain データを獲得しますが、 SQL ステートメントは実行しません。
  - CURRENT EXPLAIN MODE 特殊レジスターが RECOMMEND INDEXES に設定される。この設定により、SQL コンパイラーは、ADVISE_INDEX 表に置かれる Explain データおよび推奨索引を獲得します。ただし、SQL ステートメントは実行されません。
  - CURRENT EXPLAIN MODE 特殊レジスターが EVALUATE INDEXES に設定される。この設定により、 SQL コンパイラーは ADVISE_INDEX 表に置かれた索引を使用します。ユーザーは、評価する索引ごとに新しい行を挿入します。各索引に必要な情報は、索引名、表名、および評価される索引を構成する列名です。これらを入力したら、特殊レジスター CURRENT EXPLAIN MODE を EVALUATE INDEXES に設定します。その後、SQL コンパイラーは ADVISE_INDEX 表を走査します。その表のフィールド USE_INDEX は、『Y』 に設定されます (これらは、仮想索引と呼ばれます)。 EVALUATE INDEXES モードで実行するすべての動的ステートメントについては、それらの仮想索引が使用可能であるとして Explain が実行されます。仮想索引によってステートメントのパフォーマンスが改善される場合、 SQL コンパイラーは次に、その仮想索引を使用することを選択します。パフォーマンスが改善されないのであれば、その索引は無視されます。

EXPLAIN の結果を参照すれば、提案された索引が SQL コンパイラーによって使われたかどうかを確認できます。使用された索引は、アクセスを向上させるためにインプリメントされたと見なされます。

– EXPLAIN ALL オプションが BIND または PREP コマンドで設定されている。この設定により、CURRENT EXPLAIN MODE 特殊レジスターの設定値が NO であっても、 SQL コンパイラーは実行時に動的 SQL の Explain データを獲得します。さらに、SQL ステートメントも実行して、照会の結果を戻します。

注: Explain 情報が獲得されるのは、 SQL ステートメントがコンパイルされるときだけです。初期コンパイルに続いて、環境の変更にステートメントの再コンパイルが必要なときだけ、動的 SQL ステートメントが再コンパイルされます。同じ PREPARE ステートメントが同じ SQL ステートメントに対して連続して発行される場合、 SQL ステートメントにはコンパイルだけが行われ、最初に PREPARE ステートメントが発行されたときに、環境が変わっていないと想定して Explain データが獲得されます。

EXPLAIN SQL ステートメントの使用、または CURRENT EXPLAIN MODE レジスターの使用の詳細については、 *SQL 解説書* を参照してください。 BIND コマンドおよび PREP コマンドの詳細については、 *コマンド解説書* を参照してください。

Explain スナップショット情報の獲得

SQL ステートメントがコンパイルされ、Explain データが要求されていると、 Explain スナップショット・データが獲得されます。

- **静的** SQL ステートメント、または**増分バインド** SQL ステートメントの場合、EXPLSNAP ALL か EXPLSNAP YES 文節のどちらかが BIND または PREP コマンドで指定されると、 Explain スナップショットが獲得されます。または、FOR SNAPSHOT か WITH SNAPSHOT 文節を使用する静的 EXPLAIN SQL ステートメントがソース・プログラムで使用されます。

  注: 増分バインド SQL ステートメントは、実行時にコンパイルされると、実行時および非バインド実行時に Explain 表に置かれます。また、Explain 表への挿入の際に使用される Explain 表の修飾子および許可 ID は、パッケージ所有者のものであり、パッケージを実行する使用者のものではありません。

- **動的** SQL ステートメントの場合、次のいずれかの状態について、Explain スナップショットが獲得されます。

  – FOR SNAPSHOT または WITH SNAPSHOT 文節を使用する EXPLAIN SQL ステートメント。 FOR SNAPSHOT 文節には、 Explain スナップショットに関連する情報の他に獲得された Explain 表情報はありません。 WITH SNAPSHOT 文節には、 Explain スナップショットに関連する情報に加えて獲得された Explain 表情報があります。

    EXPLAIN SQL ステートメントを用いた Explain スナップショットの例は次のようなものです。

```
EXPLAIN PLAN FOR SNAPSHOT FOR <any valid DELETE, INSERT, SELECT,
  SELECT INTO, UPDATE, VALUES, or VALUES INTO SQL statement>
```

Explain スナップショットが 1 つだけとられ、獲得された情報は EXPLAIN_INSTANCE および EXPLAIN_STATEMENT 表に入れられます。

- CURRENT EXPLAIN SNAPSHOT 特殊レジスターが YES に設定される。 この設定により、SQL コンパイラーは Explain データのスナップショットをとり、 SQL ステートメントを実行して、照会の結果を戻します。
- CURRENT EXPLAIN SNAPSHOT 特殊レジスターが EXPLAIN に設定される。この設定により、SQL コンパイラーは Explain データのスナップショットをとりますが、 SQL ステートメントは実行しません。
- EXPLSNAP ALL オプションが BIND または PREP コマンドで設定されている。この設定により、CURRENT EXPLAIN SNAPSHOT 特殊レジスターの設定が NO であっても、 SQL コンパイラーは実行時に Explain データのスナップショットをとります。さらに、SQL ステートメントも実行して、照会の結果を戻します。

注: Explain 情報が獲得されるのは、 SQL ステートメントがコンパイルされるときだけです。初期コンパイルに続いて、環境の変更にステートメントの再コンパイルが必要なときだけ、動的 SQL ステートメントが再コンパイルされます。同じ PREPARE ステートメントが同じ SQL ステートメントに対して連続して発行される場合、 SQL ステートメントにはコンパイルだけが行われ、最初に PREPARE ステートメントが発行されたときに、環境が変わっていないと想定して Explain データが獲得されます。

EXPLAIN SQL ステートメントと FOR SNAPSHOT または WITH SNAPSHOT 文節の使用、あるいは CURRENT EXPLAIN SNAPSHOT 特殊レジスターの使用の詳細については、 *SQL 解説書* を参照してください。 BIND コマンドおよび PREP コマンドの詳細については、 *コマンド解説書* を参照してください。

Explain 出力の使用に関する指針

Explain データの分析が照会や環境の調整に役に立つ多数の方法があります。たとえば、次のとおりです。

- **索引が使用されていますか?**

  97ページの『照会最適化に対する索引付けの影響』に説明されているように、適切な索引を使用するとパフォーマンスの向上に非常に役立ちます。 Explain 出力を使用すると、一連の特定の照会に役に立つように作成した索引が使用されているかどうかを判別することができます。 Explain 出力では、次の領域での索引の使用法が探せます。

  - 結合述部
  - ローカル述部
  - GROUP BY 文節

– ORDER BY 文節
– 選択リスト

さらに、Explain 機能を使用して、既存の索引の代わりに別の索引を使用できるかどうか、あるいは、全く索引を使用できないのかを評価することができます。新規の索引を作成した後で、その索引の統計を収集し（RUNSTATS コマンドを使用)、照会を再コンパイルしてください。すでに Explain データを介して、索引走査の代わりに表走査が使用されていることに気付いているかもしれません。これは、表データのクラスター化を変更すると、起こる可能性があります。以前に使用していた索引のクラスター率が現在低下している場合は、次のようにすることができます。

– その索引に従ってデータをクラスター化するために、表を再編成する。
– RUNSTATS コマンドを使用して、表と索引のカタログ統計を更新する。
– 照会を再コンパイルする。
– 表の再編成によりアクセス・プランが向上したかどうかを判別するために、Explain 出力を再検査する。

- **アクセスのタイプはアプリケーションに適していますか?**

  Explain 出力を分析して、データに対するアクセスのタイプ (概して、実行しているアプリケーションのタイプには最適ではない) を探すことができます。たとえば、次のとおりです。

  – **オンライン・トランザクション処理 (OLTP) 照会**

    OLTP アプリケーションは、述部を区切る範囲を指定した索引走査を使用するのに最高の候補です。これは、そのアプリケーションが、キー列に対して等価述部を使用して修飾された数行しか戻さないようになっているためです。 OLTP 照会が表走査を使用している場合は、Explain データを分析して、索引走査が使用されなかった理由を判別することができます。

  – **ブラウズ専用照会**

    『ブラウズ』タイプの照会の探索基準は不明確で、多数の行を修飾しなければならなくなります。ユーザーが通常、出力データの数個の画面を見るだけならば、一部の結果が戻される前に、応答セット全体を計算する必要はないことを確かめるようにすることができます。この場合、ユーザーのゴールは最適化プログラムの基本操作方針、つまりデータの最初の数画面だけではなく、照会全体のリソースの消費を最小化することとは異なっています。

    たとえば、マージ走査結合と分類演算子の両方がアクセス・プランで使用されたことを Explain 出力が示す場合は、何行かがアプリケーションに戻される前に、応答セット全体が一時表で具体化されます。この場合、SELECT ステートメントの OPTIMIZE FOR 文節を用いてアクセス・プランを変更することができます。(OPTIMIZE FOR 文節の詳細については、 76ページの『OPTIMIZE FOR n ROWS 文節』を参照してください。) このように、最適化プログラムは一時表の応答セット全体を作成しないアクセス・プランを選択してから、最初の行をアプリケーションに戻そうとすることができます。

- **使用する結合方式のタイプは何ですか?**

  照会が 2 つの表を結合する場合は、使用されている結合処理のタイプを調べることができます。複数行が含まれる結合 (意思決定支援照会での結合など) は、通常、マージ結合を使用するよりも速く実行します。数行しか含まれない結合 (OLTP 照会など) は、一般に、ネストされたループ結合を使用するよりも速く実行します。しかし、どちらの場合にも、上記の一般的な結合の作業方法を変更することになる酌量すべき状況があります。たとえば、ローカル述部またはローカル索引の使用などがあります。 (これらの 2 つの結合方式が作動する方法については、 173ページの『ネストしたループ結合』および 174ページの『マージ結合』を参照してください。)

Visual Explain

Visual Explain を使用すると、他の方式、特により複雑な操作順序が含まれる方式と比較するときに、照会を詳しく調べることができます。 Visual Explain はサポートされるプラットフォームすべてで使用できるわけではありません。 Visual Explain がサポートされているかどうかを調べるには、 *概説およびインストール* を参照してください。

Visual Explain では、 Explain が実行された SQL ステートメントのアクセス・プランをグラフとして表示します。そのグラフの使用可能な情報を使用して、より良いパフォーマンスが得られるように SQL 照会をチューニングすることができます。さらに、Visual Explain を使用すると、動的に SQL ステートメントの Explain を行ったり、結果として出されるアクセス・プランのグラフを表示することができます。

最適化プログラムがアクセス・プランを選択するのに対し、 Visual Explain は、そのアクセス・プランの情報をアクセス・プラン・グラフとして表示します。そのグラフでは、表および索引、さらにそこで行われる操作がノードとして表され、データの流れがノード間のリンクによって表されます。

アクセス・プラン・グラフを表示するには、 Explain スナップショットを作成しておく必要があります。アクセス・プラン・グラフから、次の事柄に関する詳細を表示することができます。

- 表および索引 (さらに関連する列)
- 演算子 (表走査、分類、および結合など)
- 表スペースおよび関数

また、Visual Explain を使って次のことを行うこともできます。

- 最適化を行う際に使用した統計の表示。次に、それらの統計を現行のカタログ統計と比較すると、パッケージの再バインドがパフォーマンスを改善するかどうかを判別することができます。
- 表のアクセスに索引を使用したかどうかの判別。索引を使用しなかった場合、Visual Explain は、索引化を行うとどの列の役に立つかを判別するのに役に立ちます。

- さまざまな調整技法を実行した効果の表示。アクセス・プラン・グラフの照会前のバージョンと照会後のバージョンを比較することにより、行います。
- アクセス・プランでの各操作に関する情報の獲得。これにはコストの見積合計や検索される行数 (カーディナリティー) が含まれます。

Visual Explain についてのより詳しい情報が必要な場合は、コントロール・センターを介して使用できるオンライン情報を参照してください。コントロール・センターには、コマンド行で `db2cc` と入力するとアクセスすることができます。

SQL アドバイス機能

インデックス・アドバイザーは、データに合わせて索引を設計したり定義したりする必要を減らす管理ツールです。

インデックス・アドバイザーは、以下の点で優れています。

- 問題の照会に最適な索引の検索。
- 任意に適用されるリソース制限に基づく、一連の照会 (作業負荷) に最適な索引の検索。
- 索引を作成しないで行う、作業負荷での索引のテスト。

この SQL アドバイス機能に関連した、いくつかの概念を説明します。まず、*作業負荷* が存在します。作業負荷は、データベース・マネージャーが所定の期間に処理しなければならない一連の SQL ステートメントです。この SQL ステートメントには、SELECT、INSERT、UPDATE、および DELETE ステートメントがあります。たとえば、1 カ月の間、データベース・マネージャーは、1 000 の INSERT、 10 000 の UPDATE、10 000 の SELECT、および 1 000 の DELETE を処理しなければならないとします。作業負荷内の情報は、所定の期間における SQL ステートメントのタイプと頻度に関係するものです。アドバイス・エンジンは、この作業負荷情報をデータベース情報と共に使用して、索引を推奨します。アドバイス・エンジンの目的は、作業負荷の合計コストを最小化することです。

次に、*仮想索引* という概念があります。仮想索引とは、現在のデータベース・スキーマに存在しない索引です。これらの索引は、アドバイス機能によって提示された推奨索引であるか、アドバイス機能に評価させる索引のいずれかになります。さらにこれらの索引は、アドバイス機能が処理の一部であると見なし、推奨されない場合に廃棄される索引でもあります。仮想索引は、ADVISE_INDEX 表を使用して、アドバイス機能へ受け渡しされます。

アドバイス機能は、作業負荷とデータベースからの統計を使用し、推奨索引を生成します。

アドバイス機能は、2 つの Explain 表を使用します。

- ADVISE_WORKLOAD

この表では、考慮する作業負荷を記述します。表の各行は 1 つの SQL ステートメントであり、関係する頻度によって記述されます。作業負荷ごとに ID が存在し、それが WORKLOAD_NAME という表の 1 つのフィールドになります。同じ作業負荷の一部であるすべての SQL ステートメントには、同じ WORKLOAD_NAME が付けられていなければなりません。

「索引 (Index)」ウィザードおよび db2advis ツールは、作業負荷情報を取り出したり保管したりするのにこの表を使用します。

- ADVISE_INDEX

  この表には、推奨索引についての情報が格納されています。情報は、SQL コンパイラー、「索引 (Index)」ウィザード、db2advis ツール、またはユーザーによってこの表に入れられます。

  この表は、次の 2 つの目的で使用します。

  – アドバイス機能から推奨索引を入手する
  – 索引を評価する

注: この表を作成するには、 sqllib サブディレクトリーの misc サブディレクトリーにある EXPLAIN.DDL を実行します。 まだ作成していなければ、「索引 (Index)」ウィザードで表を作成することもできます。

インデックス・アドバイザーを使用するときの処理には、入力、アドバイザーの呼び出し、出力、および考慮する必要のあるいくつかの事例が関係します。

インデックス・アドバイザーの入力を作成するには、次の 3 つの方法があります。

- 作業負荷を取り込む。

  次のいずれかの方法を使用して、評価される SQL を作成します。

  – モニターを使って動的 SQL を入手する。
  – SYSSTMT カタログ視点を使って静的 SQL を入手する。
  – 値を ADVISE_INDEX 表にカット・アンド・ペーストして、ステートメントおよび頻度を追加する。
- 作業負荷の頻度を変更して、照会の重要性を増したり減らしたりする。
- データの制約を判別する (存在する場合)。

インデックス・アドバイザーを呼び出すには、次の 4 つの方法があります。

- コントロール・センターを使用する。

  これは、インデックス・アドバイザーを使用するときの、推奨の方法です。コントロール・センターからオブジェクト・ツリーを展開してゆき、**「索引 (Indexes)」**フォルダーを見つけます。 マウス・ボタン 2 で**「索引 (Indexes)」**フォルダーをクリックし、ポップアップ・メニューから**「作成 (Create)」->「索引 - ウィザードを使用 (Index using wizard)」**を選択します。 「索引 (Index)」ウィザードがオープンします。「索引 (Index)」ウィザードには広範囲のヘルプが備えられており、簡単に使うこ

とができます。ウィザードには、最近実行された SQL を探したり、最近使用されたパッケージを調べたりすることによって作業負荷を構成する機能、あるいは手動で SQL ステートメントを追加することによって作業負荷を構成する機能も備えられています。

- コマンド行プロセッサーを使用する。

  コマンド行で、db2advis と入力します。 db2advis は、次の 3 つの位置のいずれかから作業負荷を読み取ることにより始動します。

  – コマンド行
  – テキスト・ファイル内のステートメント
  – 提案された作業負荷 (SQL および頻度) で行が挿入された ADVISE_WORKLOAD 表

  ツールは、最善の索引を選ぶための内部最適化アルゴリズムと組み合わせて CURRENT EXPLAIN MODE レジスターを使用し、推奨索引を獲得します。出力は、必要に応じて、使用している端末の画面、ADVISE_INDEX 表、そして出力ファイルに出力されます。

  たとえば、単一照会 『select count(*) from sales where region = 'Quebec'』 について、ツールを使用して索引を推奨することができます。

```
$ db2advis -d sample ¥
 -s "select count(*) from sales where region = 'Quebec'" ¥
 -t 1
performing auto-bind

Bind is successful. Used bindfile: /home3/valentin/sqllib/bnd/db2advis.bnd

Calculating initial cost (without recommended indexes) [31.198040] timerons
Initial set of proposed indexes is ready.
Found maximum set of [1] recommended indexes
Cost of workload with all indexes included [2.177133] timerons
cost without index [0] is [31.198040] timerons.  Derived benefit is
[29.020907]
total disk space needed for initial set [1] MB
total disk space constrained to         [-1] MB
  1  indexes in current solution
 [31.198040] timerons  (without indexes)
 [2.177133] timerons  (with current solution)
 [%93.02] improvement

 Trying variations of the solution set.
Time elapsed.
LIST OF RECOMMENDED INDEXES
===========================
index[1], 1MB CREATE INDEX WIZ689 ON VALENTIN.SALES (REGION DESC)
===========================
Index Advisor tool is finished.
```

  作業負荷についても、 db2advis ツールを使用して索引を推奨できます。次のようにして、『sample.sql』 という入力ファイルを作成できます。

```
--#SET FREQUENCY 100
select count(*) from sales where region = ?;
--#SET FREQUENCY 3
select projno, sum(comm) tot_comm  from employee, emp_act
where employee.empno = emp_act.empno and
     employee.job='DESIGNER'
group by projno
order by tot_comm desc;
--#SET FREQUENCY 50
select * from sales where sales_date = ?;
```

その後、次のコマンドを実行します。

```
$ db2advis -d sample -i sample.sql -t 0
  found [3] SQL statements from the input file

Calculating initial cost (without recommmended indexes) [62.331280] timerons
Initial set of proposed indexes is ready.
Found maximum set of [2] recommended indexes
Cost of workload with all indexes included [29.795755] timerons
cost without index [0] is [58.816662] timerons.  Derived benefit is
[29.020907]
cost without index [1] is [33.310373] timerons.  Derived benefit is
[3.514618]
total disk space needed for initial set [2] MB
total disk space constrained to         [-1] MB
  2  indexes in current solution
 [62.331280] timerons  (without indexes)
 [29.795755] timerons  (with current solution)
 [%52.20] improvement

Trying variations of the solution set.
Time elapsed.
LIST OF RECOMMENDED INDEXES
===========================
index[1], 1MB CREATE INDEX WIZ119 ON VALENTIN.SALES (SALES_DATE DESC,
SALES_PERSON DESC)
index[2], 1MB CREATE INDEX WIZ63 ON VALENTIN.SALES (REGION DESC)
===========================
Index Advisor tool is finished.
```

- EXPLAIN モードおよび PREP コマンド・オプションに関係する自己指示方式を使う。

  たとえば、CURRENT EXPLAIN MODE 特殊レジスターは RECOMMEND INDEXES に設定されます。この設定により、SQL コンパイラーは、ADVISE_INDEX 表に置かれる Explain データおよび推奨索引を獲得します。ただし、SQL ステートメントは実行されません。

  または、CURRENT EXPLAIN MODE 特殊レジスターが EVALUATE INDEXES に設定されます。この設定により、 SQL コンパイラーは ADVISE_INDEX 表に置かれた索引を使用します。ユーザーは、評価する索引ごとに新しい行を挿入します。各索引に必要な情報は、索引名、表名、および評価される索引を構成する列名です。これ

らを入力したら、特殊レジスター CURRENT EXPLAIN MODE を EVALUATE INDEXES に設定します。その後、SQL コンパイラーは ADVISE_INDEX 表を走査して、 USE_INDEX=『Y』となっている索引を見つけます。 (これらは、仮想索引と呼ばれます)。 EVALUATE INDEXES モードで実行するすべての動的ステートメントについては、それらの仮想索引が使用可能であるとして Explain が実行されます。仮想索引によってステートメントのパフォーマンスが改善される場合、 SQL コンパイラーは次に、その仮想索引を使用することを選択します。パフォーマンスが改善されないのであれば、その索引は無視されます。 EXPLAIN の結果を参照すれば、提案された索引が SQL コンパイラーによって使われたかどうかを確認できます。使用された索引は、アクセスを向上させるためにインプリメントされたと見なされます。

- コール・レベル・インターフェース (CLI) を使用する。

  このインターフェースを使ってアプリケーションを作成している場合、アドバイザーも使用できます。

アドバイザーからの結果を使用する方法は、いくつかあります。

- インデックス・アドバイザーからの出力を解釈する。

  アドバイス機能によって推奨されている索引を表示するには、次の照会を使用することができます。

  ```
  SELECT CAST(CREATION_TEXT as CHAR(200))
     FROM ADVISE_INDEX
  ```

- インデックス・アドバイザーの推奨を適用する。
- 索引を除去する時期を判別する。

特定の照会についてより良い推奨を獲得するために、その照会そのものに対してアドバイス機能を使用することが提案されています。「索引 (Index)」ウィザードを使用し、該当する照会だけを含む作業負荷を作成することにより、 1 つの照会のために索引を推奨することができます。

サンプルの作業負荷は、イベント・モニターの出力から収集することができます。イベント・モニターでは、動的 SQL の実行を集めることができます。その後、このステートメントをアドバイス機能に送ります。

「索引 (Index)」ウィザードは、単純明快で使いやすいビジュアル・インターフェースであり、アドバイス機能にアクセスするための優れた方法です。

第3部 システムのチューニングと構成

第8章 操作上のパフォーマンス

実行時の SQL 照会のパフォーマンスを調整するための情報を、次のトピックに分けて説明します。

- DB2 でのメモリーの使用方法
- データベース・バッファー・プールの管理
- 複数のデータベース・バッファー・プールの管理
- バッファー・プールへのデータの事前取り出し
- 事前取り出しおよび並列入出力を行うための入出力サーバーの構成
- 分類
- カタログとユーザー表の再編成
- DMS 装置のパフォーマンスに関する考慮事項
- 初期化オーバーヘッドの管理
- データベース・エージェント
- データベース・システム・モニターの使用法
- メモリーの拡張

以下の章でも、パフォーマンスにどのように影響が出るかについて説明します。

- 43ページの『第3章 アプリケーションについての考慮事項』
- 91ページの『第4章 環境についての考慮事項』
- 113ページの『第5章 システム・カタログ統計』

物理データベースの設計に関する考慮事項については、 *管理の手引き: 計画* を参照することもできます。

DB2 でのメモリーの使用方法

システムのメモリー使用法は、DB2 で使用可能な多数の構成パラメーターを使って制御します。このパラメーターの中には、サーバーのメモリーを制御するパラメーター、クライアントのメモリーを制御するパラメーター、およびその両方を制御するパラメーターがあります。また、メモリーの割り振りまたは割り振り解除を実行するタイミングやシステム領域もそれぞれ異なります。

システム管理者は、システム全体でのメモリー使用量を平衡化することも考慮する必要があります。オペレーティング・システムで実行されるアプリケーションの種類によって、メモリーの使用法もそれぞれ異なる可能性があります。たとえば、データベース・マネージャーでは、データ・キャッシュのためには、オペレーティング・システムの機

能ではなく独自のバッファー・プールを使用しますが、アプリケーションの中にはファイル・システム・キャッシュを使用するものもあります。 詳細については、244ページの『メモリー使用法を制御するパラメーターの設定』を参照してください。

図22 は、データベース・マネージャーがさまざまなタイプのメモリーを使用する様子を示しています。メモリー使用を表すこの図は、エンタープライズ拡張エディション、または複数論理ノード環境には当てはまりません。エンタープライズ拡張エディション、または複数論理ノード環境には、 (ノードごとに 1 つの) データベース・マネージャー共用メモリー・セットがあります。

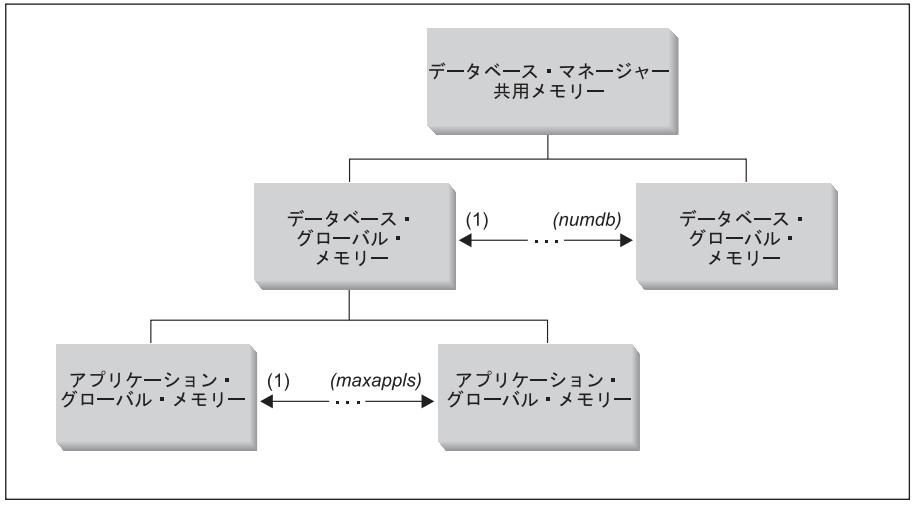


図 22. データベース・マネージャーによって使用されるメモリーのタイプ

メモリーは、以下に示す時点でデータベース・マネージャーの各インスタンスに対して割り振られます。

- データベース・マネージャーを開始したとき (db2start)、「データベース・マネージャー共用メモリー」というマークを付けられた領域が割り振られたとき、およびその領域が割り振られたままの状態でデータベース・マネージャーを停止したとき (db2stop)。 この領域には、すべてのデータベース接続でのアクティビティーを管理するうえで、データベース・マネージャーが必要としている情報があります。 最初のアプリケーションがデータベースに接続されると、グローバル・メモリー領域と私用メモリー領域の両方が割り振られます。
- データベースが最初に活動化または接続されると、『データベース・グローバル・メモリー』が割り当てられます。データベース・グローバル・メモリーは、データベースに接続する全アプリケーションによって使用されるもので、バッファー・プール、ロック・リスト、データベース・ヒープ、ユーティリティー・ヒープなどのメモリー領域が含まれます。

- アプリケーションがデータベースに接続すると、『アプリケーション・グローバル・メモリー』が割り当てられます (これは、区分データベース環境でのみ、または *intra_parallel* 構成パラメーターが使用可能な場合にのみ起こります)。このメモリーは、データを共用し、アクティビティーを調整するために、アプリケーションの代わりに作業するエージェントによって使用されます。
- (前の図では表されていません) エージェントが特定のアプリケーションを処理するために (接続要求、または並列環境では新規の SQL 要求の結果として) 割り当てられると、『エージェント私用メモリー』 がそのエージェントに割り当てられます。エージェント私用メモリー領域はエージェントに対して割り振られるもので、その特定のエージェント (分類ヒープやアプリケーション・ヒープなど) によってのみ使用されます。

  データベースがあるアプリケーションによってすでに使用中である場合は、その後から接続を行うアプリケーションには、そのアプリケーション用のエージェント私用メモリーとアプリケーション・グローバル共用メモリーしか割り振られません。

238ページの図22 は、構成パラメーターの設定値がメモリーに与える影響について示しています。特に、以下にリストするパラメーターを使用すると、特定の目的のために割り振られるメモリーの量を制限することができます。 (区分データベース環境では、このメモリーは各データベース区画上に必要です。)

- *numdb* は、複数のアプリケーションで並行して使用される活動データベースの最大数を定義します。データベースにはそれぞれグローバル・メモリー領域があるので、このパラメーターの値を大きくすると、割り振り可能なメモリー量も大きくなります。
- *maxappls* は、1 つのデータベースに同時に接続できるアプリケーションの最大数を定義します。このパラメーターは、そのデータベースの「エージェント私用メモリー」および「アプリケーション・グローバル・メモリー」に割り振り可能なメモリーの量に影響を与えます。 (このパラメーターを、データベースごとに個別に設定することができることに注意してください。)
- (前の図では表されていません) *maxagents* (および並列処理の場合には *max_coordagents*) は、インスタンス内のすべての活動データベースにおいて、同時に存在することができるデータベース・マネージャー・エージェントの数を制限します。このパラメーターは、*maxappls* と同じように、「エージェント私用メモリー」および「アプリケーション・グローバル・メモリー」に割り振られるメモリーの量を制限します。 (エージェントについては、269ページの『データベース・エージェント』を参照してください。)

240ページの図23 はアプリケーションのサポートに使用されるメモリーの量を合計します。以下に示す構成パラメーターを使用して、 ″メモリー・セグメント″ (論理メモリーの部分) の数およびそれらのサイズに制限を加えることにより、図で説明したメモリーのサイズの制御を行うことができます。

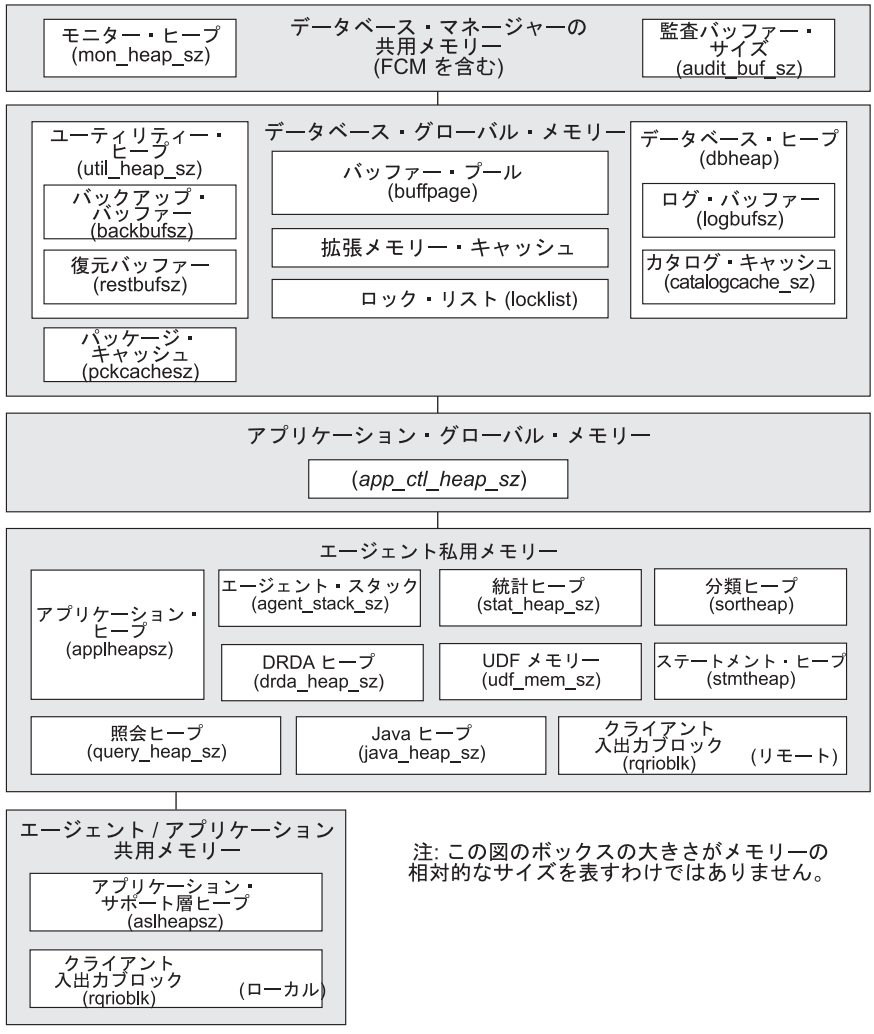


図 23. データベース・マネージャーによるメモリーの使用方法

データベース・マネージャー共用メモリー

メモリー・スペースは、データベース・マネージャーを実行するには必須のものです。このスペースは非常に大きくなる場合もあり、特に区画内並列環境および区画間並列環境の場合などはそれが顕著です。このスペースのサイズの予測および制御を行う場合には、以下のセクションを参照してください。

- 269ページの『データベース・エージェント』 。アプリケーションの代理として実行されているエージェントは、 *maxagents* の値が適切でない場合には特に、相当量のメモリー・スペースを必要とします。

- 245ページの『FCM 要件』 。区分データベース・システムの場合には、*fcm_num_buffers* の値が適切でない場合には特に、高速コミュニケーション・マネージャー (FCM) には相当量のメモリー・スペースが必要です。

  FCM のメモリー要件は、FCM バッファー・プールから割り振られるか、データベース・マネージャーの共用メモリーと FCM バッファー・プールの両方から割り振られます。どちらになるかは、区分データベース・システムが複数の論理ノードを使うかどうかで決まります。 FCM バッファー・プールについて詳しくは、以下の説明を参照してください。

FCM バッファー・プール

区分データベース・システムに複数の論理ノードが存在しない場合、データベース・マネージャーの共用メモリーと FCM バッファー・プールは、図24 で示すようになります。

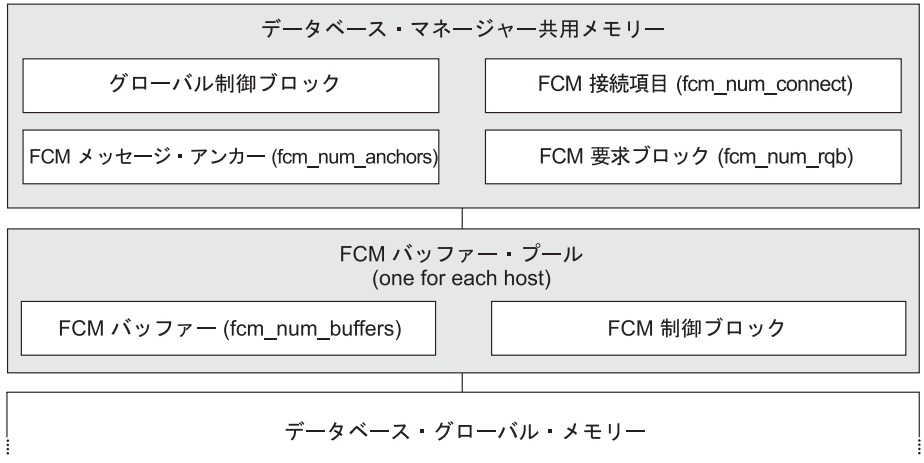


図 24. 複数の論理ノードが使用されない場合の FCM バッファー・プール

区分データベース・システムで複数の論理ノードが使用されている場合、データベース・マネージャーの共用メモリーと FCM バッファー・プールは 242ページの図25 で示すようになります。

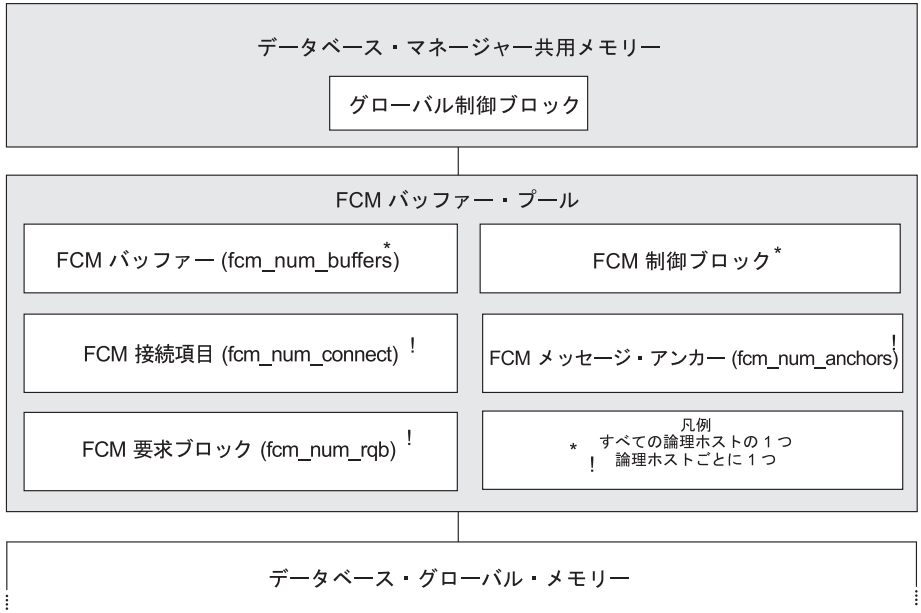


図 25. 複数の論理ノードが使用される場合の FCM バッファー・プール

データベース・グローバル・メモリー

データベース・グローバル・メモリーは、以下の構成パラメーターから影響を受けます。

- メモリー・セグメントの数は、*numdb* によって制限されます (481ページの『並行活動データベースの最大数 (numdb)』を参照)。
- メモリー・セグメントの最大サイズは、次のパラメーターの値によって決められます。
  - 348ページの『バッファー・プール・サイズ (buffpage)』 (バッファー・プール・サイズが -1 の場合)、またはバッファー・プールの作成または変更を行ったときに明示的に指定されたサイズ。
  - 357ページの『ロック・リスト用最大ストレージ (locklist)』
  - 351ページの『データベース・ヒープ (dbheap)』
  - 355ページの『ユーティリティー・ヒープ・サイズ (util_heap_sz)』
  - 398ページの『拡張記憶域メモリー・セグメント・サイズ (estore_seg_sz)』
  - 398ページの『拡張記憶域メモリー・セグメントの数 (num_estore_segs)』
  - 360ページの『パッケージ・キャッシュ・サイズ (pckcachesz)』

アプリケーション・グローバル・メモリー

アプリケーション・グローバル・メモリーは、以下の構成パラメーターから影響を受けます。 361ページの『アプリケーション制御ヒープ・サイズ

(app_ctl_heap_sz)』 並列システムの場合には、アプリケーション制御ヒープ用のスペースも必要になります。これは、1 つのデータベース区画において、同じアプリケーションの代理として処理を行うエージェントの間で共用されるものです。ヒープが割り振られるのは、アプリケーションからの要求を受け取った最初のエージェントが接続を要求するときです。エージェントは、調整エージェントかサブエージェントのいずれかになります (269ページの『データベース・エージェント』を参照してください)。

エージェント私用メモリー

- メモリー・セグメントの数は、次の値のうち小さい方の値によって制限されます。
  - すべての活動データベースに設定された *maxappls* の合計 (399ページの『活動アプリケーションの最大数 (maxappls)』を参照)。
  - *maxagents* の値 (405ページの『エージェントの最大数 (maxagents)』を参照)。
- メモリー・セグメントの最大サイズは、次のパラメーターの値によって決められます。
  - 367ページの『アプリケーション・ヒープ・サイズ (applheapsz)』
  - 363ページの『分類ヒープ・サイズ (sortheap)』
  - 366ページの『ステートメント・ヒープ・サイズ (stmtheap)』
  - 368ページの『統計ヒープ・サイズ (stat_heap_sz)』
  - 368ページの『照会ヒープ・サイズ (query_heap_sz)』
  - 370ページの『DRDA ヒープ・サイズ (drda_heap_sz)』
  - 370ページの『UDF 共用メモリー・セット・サイズ (udf_mem_sz)』
  - 372ページの『エージェント・スタック・サイズ (agent_stack_sz)』

エージェント / アプリケーション共用メモリー

- (ローカル・クライアントの場合には) エージェント / アプリケーション共用メモリー・セグメントの合計数は、次の値のうちの小さい方の値によって制限されます。
  - すべての活動データベースに設定された *maxappls* の合計 (399ページの『活動アプリケーションの最大数 (maxappls)』を参照)。
  - *maxagents* の値 (405ページの『エージェントの最大数 (maxagents)』を参照)、または (並列システムの場合には) *max_coordagents* (407ページの『調整エージェントの最大数 (max_coordagents)』を参照) の値。
- エージェント / アプリケーション共用メモリーは以下の構成パラメーターからも影響を受けます。
  - 376ページの『アプリケーション・サポート層ヒープ・サイズ (aslheapsz)』
  - 379ページの『クライアント入出力ブロック・サイズ (rqrioblk)』

メモリー使用法を制御するパラメーターの設定

搭載可能な最大量のメモリーがシステムに搭載されている場合でも、メモリーを割り振るパラメーターは、*絶対に* 許容最大値に設定しないでください。値を設定するときは、十分な注意が必要です。ほとんどのパラメーターでは、使い方によっては、データベース・マネージャーがマシン上で使用可能な全メモリーをごく簡単にまた瞬時に使い果たすことができるからです。 また、メモリー量が大きいとその管理のためにデータベース・マネージャーのほうで余計な作業が必要になり、オーバーヘッドが増大してしまいます。

UNIX ベースのオペレーティング・システムの中には、プロセスがスワップ・スペースにページアウトされるときではなく、プロセスがメモリーを割り振る時点でスワップ・スペースを割り振るものがあります。そのような場合には、共用メモリーの合計サイズと等量のページング・スペースをサポートする必要があります。

構成パラメーターの大部分では、メモリーは必要なときだけコミットされます。これらのパラメーターは特定のメモリー・ヒープの最大サイズを反映します。この規則の注意すべき例外として、パラメーターの値に基づいてメモリーが十分にコミットされる次のようなパラメーターがあります。

- 348ページの『バッファー・プール・サイズ (buffpage)』 (バッファー・プール・サイズが -1 の場合)、またはバッファー・プールの作成または変更を行ったときに明示的に指定されたサイズ。
- 364ページの『分類ヒープのしきい値 (sheapthres)』
- 357ページの『ロック・リスト用最大ストレージ (locklist)』
- 376ページの『アプリケーション・サポート層ヒープ・サイズ (aslheapsz)』
- 467ページの『FCM メッセージ・アンカーの個数 (fcm_num_anchors)』
- 468ページの『FCM バッファー数 (fcm_num_buffers)』
- 469ページの『FCM 接続項目数 (fcm_num_connect)』
- 470ページの『FCM 要求ブロック数 (fcm_num_rqb)』

この種のタイプのパラメーターの適切な値を決めるには、ベンチマーク・テストを行って決めるのが最も良い方法です。ベンチマーク・テストでは、通常の SQL ステートメントおよび最悪ケースの SQL ステートメントをサーバーに対して実行し、そこでパラメーターの値を修正しながら、パフォーマンスがこれ以上あがらないポイントを見つけます。パフォーマンスとパラメーター値の関係をグラフで表すと、曲線が停滞または降下を始めるポイントが、割り振りを増加してもアプリケーションには利益をもたらさずに、単にメモリーの無駄使いになってしまうポイントということになります。 (317ページの『第12章 ベンチマーク・テスト』を参照してください。)

パラメーターによっては、メモリー割り振りの上限が、現存のハードウェアおよびオペレーティング・システムのメモリー容量を超える場合があります。これらの限界値は、将来的な増加に備えて決められたものです。

パラメーターの有効範囲については、 331ページの『第13章 DB2 の構成』のパラメーター記述を参照してください。

FCM 要件

以下に示す高速コミュニケーション・マネージャー (FCM) の構成パラメーターの構成を行うときは、まずはデフォルト値から始めるようにしてください。

- 468ページの『FCM バッファー数 (fcm_num_buffers)』
- 470ページの『FCM 要求ブロック数 (fcm_num_rqb)』
- 469ページの『FCM 接続項目数 (fcm_num_connect)』
- 467ページの『FCM メッセージ・アンカーの個数 (fcm_num_anchors)』

これらのパラメーターをチューニングするには、データベース・システム・モニターを使用して、空きバッファー、空きメッセージ・アンカー、空き接続項目、および空き要求ブロックの最低水準点をモニターしてください。最低水準点が対応するフリー・データ項目の数の 10 % より少ない場合には、対応するパラメーターの値を増やす必要があります。データベース・システム・モニターについては、 275ページの『データベース・システム・モニターの使用法』を参照してください。

FCM 通信を使用可能にするための情報については、*管理の手引き: 計画* を参照してください。

データベース・バッファー・プールの管理

バッファー・プールとは、データベース・ページ (表の行項目や索引項目が含まれる) を一時的に読み込んだり変更したりするためのストレージのことです。バッファー・プールの目的は、データベース・システムのパフォーマンスを改善することです。ディスク上のデータよりメモリー上のデータの方が速くアクセスできます。したがって、データベース・マネージャーがディスクとの間で読み書きする頻度が少なくなるほど、パフォーマンスは向上します。

バッファー・プール (単数も複数も可) の構成は、データベースに接続されるアプリケーションのデータ操作タスクの大部分 (ラージ・オブジェクトやロング・フィールド・データを除く) がここで行われることを考えると、非常に重要なチューニング領域と言えます。

アプリケーションが表のある行に最初にアクセスしたときに、データベース・マネージャーは、その行を含むページをバッファー・プールに入れます。アプリケーションのいずれかが次にデータを要求した場合には、バッファー・プールが最初に検査されます。要求されたデータがバッファー・プールに保持されているページ上に見つかった場合は、データベース・マネージャーはディスク装置からそのデータをリトリーブする必要はありません。ディスク装置からデータをリトリーブする手間が省けると、パフォーマンスが速くなります。

バッファー・プールと関係あるストレージは、最初のアプリケーションがデータベースに接続したとき、またはデータベースが明示的に活動化されたときに、すべて割り振られます。バッファー・プールの恩恵に最初にあずかるのは、アプリケーションです。アプリケーションがすべて切断されると、バッファー・プールと関係あるストレージは割り振り解除されます。

ページは、データベースがシャットダウンされるまで、あるいは、あるページが占めるスペースが別のページによって要求されるまで、バッファー・プール内に留まります。別のページを入れるために選ばれるバッファー・プールのスペースは、以下に示す基準を使用して選択されます。

- ページへの最後の参照
- ページを最後に見たエージェントが、そのページを再び参照する可能性
- ページ上のデータ・タイプ
- ページがメモリーで変更されたがディスクには書き出されていないか否か。 (変更されたページは常に、上書きされる前にディスクに書き込まれる。)

注: 変更されたページは、ディスクに書き出された後でも、そのページが占めているスペースが他のページで必要とされない限り、バッファー・プールから取り除かれることはありません。上書きされるまでは、そのページのデータが必要ならば再度アクセスすることができます。

バッファー・プールの作成時、デフォルトではページ・サイズは 4 KB です。バッファー・プールを作成するとき、ページ・サイズは 4 KB、8 KB、16 KB または 32 KB のいずれかを選択して設定することができます。いずれかのページ・サイズを使ってバッファー・プールが作成される場合、関連付けられるのは同じページ・サイズで作成された表スペースだけです。作成してしまうと、バッファー・プールのページ・サイズを変更することはできません。希望のページ・サイズを持つ新規バッファー・プールを作成する必要があります。

Windows システムにおける大容量メモリーの使用

Windows 2000 を使用する場合、DB2 とオペレーティング・システムのサイズよりも小さいサイズの範囲で、64 GB までのバッファー・プール・サイズがサポートされます。(これは、DB2 がシステム上の基本製品であることを想定しています。) このサポートは、Microsoft Address Windowing Extensions (AWE) を通じて得ることができます。

AWE はバッファー・プールのサイズに関係なく使用することができますが、より大きなバッファー・プールで AWE を使用しなければならない場合には別の Windows 製品をお勧めします。 Windows 2000 Advanced Server は 8 GB までのメモリーをサポートします。 Windows 2000 Data Center Server は 64 GB までのメモリーをサポートします。

DB2 および Windows 2000 は、AWE バッファー・プールをサポートするように正しく構成されている必要があります。 AWE を利用するバッファー・プールはデータベースに存在していなければなりません。

3 GB のユーザー・スペースを割り振るには、/3GB Windows 2000 ブート・オプションを使用します。これにより、より大きなサイズの AWE ウィンドウを使用できます。AWE メモリー・インターフェース経由で 4 GB を超えるメモリーにアクセスできるようにするには、/PAE Windows 2000 ブート・オプションを使用します。 正しいブート・オプションが選択されていることを確認するには、「システム」を選んでから「起動/回復」を選択してください。ドロップダウン・リストから、使用可能なブート・オプションを確認することができます。使用したいブート・オプション (/3GB または /PAE) が選択されていれば、 AWE サポート・セットアップの次の作業に進むことができます。使用したいオプションが選択項目にない場合、システム・ドライブの boot.ini ファイルにそのオプションを追加してください。 boot.ini ファイルには、オペレーティング・システムの開始時に実行するアクションのリストが含まれています。既存パラメーターのリストの最後に /3GB か /PAE、またはその両方 (ブランクで区切る) を追加してください。この変更ファイルを保管しておくと、前述のように正しいブート・オプションを確認および選択することができます。

また、Windows 2000 では、DB2 をインストールしたユーザーに「メモリ内のページのロック」権利を関連付けるように変更する必要があります。「メモリ内のページのロック」権利を設定するには、 DB2 をインストールしたユーザーとして Windows 2000 にログオンし、 Windows 2000 の「スタート」メニューにある「管理ツール」フォルダーを選択して、「ローカル セキュリティ ポリシー」プログラムを選択します。「ローカル ポリシー」で、「ユーザー権利の割り当て」を選んで、「メモリ内のページのロック」権利を選択してください。

DB2 は DB2_AWE レジストリー変数の設定を必要とします。このレジストリー変数を正しく設定するには、AWE のサポートを許可したいバッファー・プールのバッファー・プール ID を知っておく必要があります。バッファー・プール ID は、SYSCAT.BUFFERPOOLS システム・カタログ・ビューの BUFFERPOOLID 列にあります。また、割り振る物理ページとアドレス・ウィンドウ・ページの数も認識していなければなりません。割り振る物理ページの数は、使用可能な物理ページの合計よりも小さい値にします。実際に選択する数は作業環境によって異なります。たとえば、DB2 とデータベース・アプリケーションだけがシステムで使用される環境の場合、 DB2_AWE 変数で使用する値として、物理ページの合計サイズよりも小さい、 0.5 GB から 1 GB までの範囲から選択することができます。その他の非データベース・アプリケーションがシステムを使用している環境の場合、より多くの物理ページを他のアプリケーションに対して許可するために合計から減算する値を増やす必要があります。 DB2_AWE レジストリー変数で使用される数は、AWE のサポートに使用され、また DB2 によって使用される物理ページの数です。 /3GB Windows 2000 ブート・オプション有効時のアドレス・ウィンドウ・ページの上限は、 1.5 GB または 2.5 GB です。

DB2 レジストリー変数 DB2_AWE の設定についての詳細は、本書の「付録 A DB2 レジストリーと環境変数」のレジストリー変数を参照してください。

バッファー・プール・ページの使用

バッファー・プール内のページには、以下のようなさまざまな属性があります。

- 使用中のページとは、現在読み取り中または更新中のページのことです。他のエージェントからは、それらのページの読み取りはできますが更新はできません。
- 「ダーティー」ページとは、データが変更されているがまだディスクに書き出されていないページのことです。ページがディスクに書き出されると、そのページは「クリーン」になったと見なされ、引き続きバッファー・プールに残ります。クリーン・ページが占めているスペースは新しいページによって使用可能であり、関連する拡張記憶キャッシュが定義されている場合はそこに移行することもできます。

バッファー・プール内の変更されたページのスペース占有率が *chngpgs_thresh* 構成パラメーターに指定した値を超えると、ページがバッファー・プールからディスクに書き出される場合があります。また、その場合、複数のページ・クリーナー・エージェントを組み込むように、データベースを構成する必要も生じます。これらのエージェントは、変更されたページをディスクに書き出して、データベース・エージェントがバッファー・プール内に使用可能なスペースを見つけることができるようにします。

ページ・クリーナー・エージェントは、もしこのエージェントがなかったらデータベース・エージェントが実行する必要がある入出力を実行します。 このエージェントを使用すると、データベース・エージェントがページをディスクに書き込む間トランザクションが待機させられることがなくなるので、アプリケーションの実行速度を速めることができます。 (ページ・クリーナー・エージェントは、データベース・エージェントと同時にジョブを実行することができるため、*非同期*ページ・クリーナー または*非同期*バッファー*書き込み機能* と呼ばれることがあります。)

ページ・クリーナー・エージェントの数を変更する場合は、 *num_iocleaners* 構成パラメーターを使用します (デフォルトを使うと、 1 つのページ・クリーナー・エージェントが作成されます)。詳細については、392ページの『非同期ページ・クリーナーの数 (num_iocleaners)』を参照してください。このパラメーターの値は、 1 からデータベース内の物理ディスクの数の間に設定します。この数が大きければ大きいほど、更新集中作業負荷を実行するときのパフォーマンスは良くなります。これは、非同期のデータ・ページ書き込みまたは索引ページ書き込みに関連したデータ・ページ書き込みまたは索引ページ書き込みが多い場合にも当てはまります。

さらに、ページをディスクに書き込んでおくと、データベース・マネージャーがデータベース・ログ・ファイルを使用しなくても、バッファー・プールの大部分をディスクから再作成することができるので、システムの破壊が起こったとしてもデータベースのリカバリーを迅速に行うことができるようになります。結果として、リカバリーが起きたときに読み取る必要のあるログのサイズが次の値を超えると、ページ・クリーニングが要求されます。

```
logfilsiz * softmax
```

説明:

- *logfilsiz* はログ・ファイルのサイズを表します (415ページの『ログ・ファイルのサイズ (logfilsiz)』を参照してください)。
- *softmax* は、データベース破壊が起きたらリカバリーする必要のあるログ・ファイルの比率 (パーセンテージ) を表します (423ページの『リカバリー範囲とソフト・チェックポイント間隔 (softmax)』を参照してください)。

  たとえば、softmax の値が 250 ならば、破壊が起きた場合にリカバリーする必要がある変更が 2.5 個のログ・ファイルに含まれていることになります。

データベース・システム・モニターを使用すると、リカバリー中のログ読み取り時間を最小化するためにページ・クリーニングが要求される回数をトラックするのに役に立ちます。詳細については、 *システム・モニター 手引きおよび解説書* にある *pool_lsn_gap_clns (生成されるバッファー・プール・ログ・スペース・クリーナー)* モニター・コンポーネントに関する説明を参照してください。

リカバリー中に読み取らなければならないログのサイズは、ログ中の次の位置の間の差になります。

- 最も新しく書き込まれたログ・レコード
- バッファー・プール内で一番古い変更を記述するログ・レコード

次の図は、ページ・クリーナー・エージェントとデータベース・エージェントとの間でのバッファー・プールの管理作業の分担状態と、データベース・エージェントがすべての入出力を行っている状態とを比較しています。

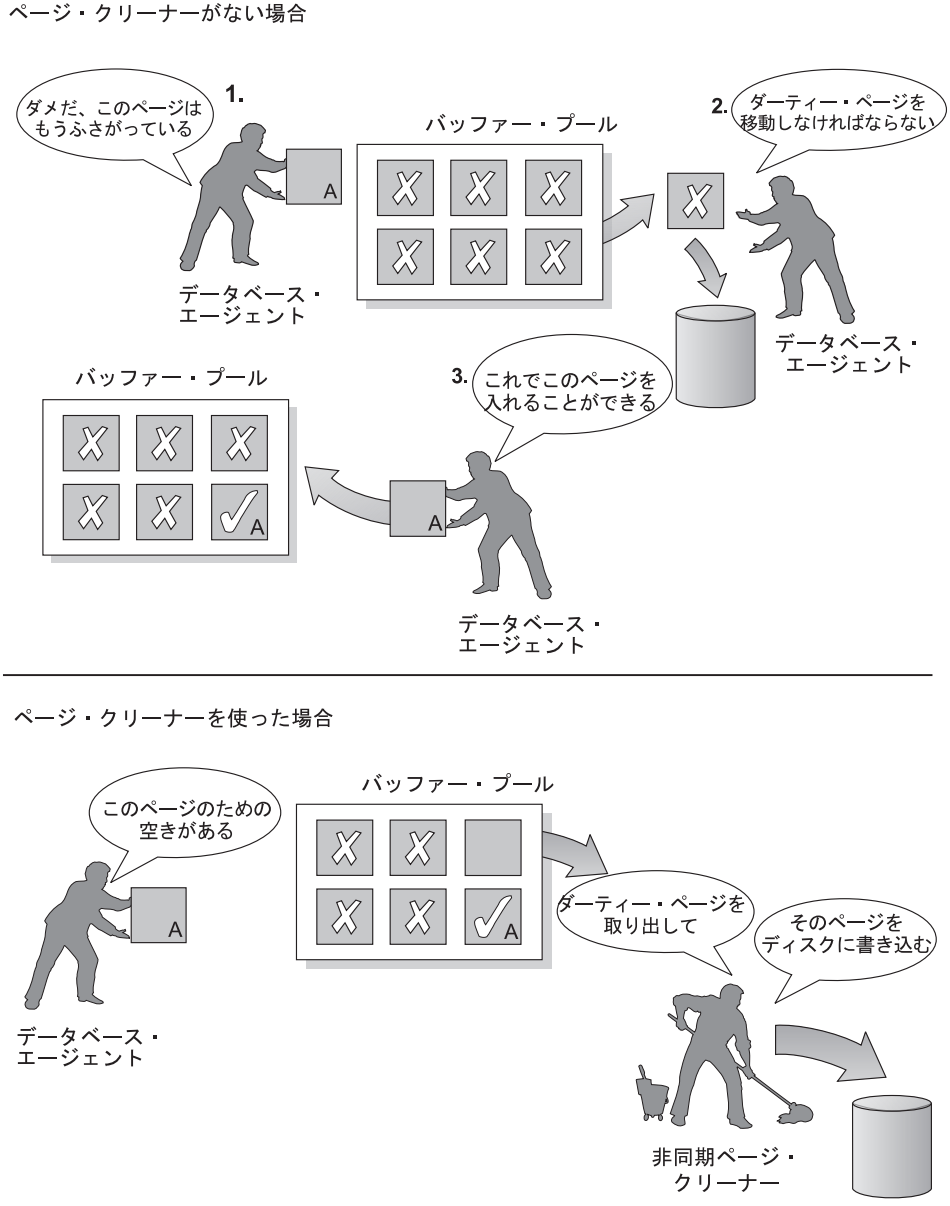


*図 26. 非同期*ページ・クリーナー. 「ダーティー」ページがディスクに書き出されます。

複数のデータベース・バッファー・プールの管理

各データベースにはそれぞれ、少なくとも 1 つはバッファー・プールが必要です。しかし、ユーザーのニーズによっては、単一のデータベースに対して、それぞれ別サイズで、複数のバッファー・プールを作成する必要が生じる場合があります。ステートメント CREATE、ALTER、および DROP BUFFERPOOL を使用すると、バッファー・プールの作成、変更、除去を行うことができます。どのデータをバッファー・プールにキャッシュするかについては、ステートメント CREATE TABLESPACE および ALTER TABLESPACE を使用して指定することができます。

buffpage 構成パラメーターは、 SYSCAT.BUFFERPOOLS カタログ視点に、バッファー・プールのサイズが -1 と指定されている場合に、任意のバッファー・プールのサイズを指定するものです。 (それ以外の場合には、このパラメーターは無視されます。) バッファー・プールのサイズは、 DDL ステートメントの ALTER BUFFERPOOL または CREATE BUFFERPOOL を用いて設定することができます。

新規または移行されたデータベースは、 IBMDEFAULTBP というデフォルト・バッファー・プールを持っています。また、そのサイズはプラットフォームによって決まります。データベースの作成または移行が終了したら、他のバッファー・プールをそのデータベース用に作成することができます。

データベース設計の作業時、ページ・サイズが 8 KB の表が最善であると決定していたかもしれません。その結果、バッファー・プールも 8 KB のページ・サイズで作成する必要があります (同じページ・サイズの 1 つ以上の表スペースも作成する)。

区分データベース環境の場合には、 1 つのデータベース用の各バッファー・プールのデフォルト定義は、全データベース区画に対して同一になっています (ただし、それ以外の定義が CREATE BUFFERPOOL ステートメントに指定されている場合、または特定のデータベース区画のバッファー・プールのサイズが ALTER BUFFERPOOL ステートメントを用いて変更された場合は除きます)。

4 KB のページ・サイズで表スペースを作成し、それを特定のバッファー・プールに割り当てていない場合、表スペースはデフォルトのバッファー・プールに割り当てられます。 4 KB (8 KB、16 KB、32 KB) より大きいページ・サイズで表スペースを作成する場合、同じページ・サイズを使用するバッファー・プールに割り当てる必要があります。このバッファー・プールが現在アクティブでない場合、 DB2 は同じページ・サイズを使用するアクティブなバッファー・プール (使用可能であれば) に表スペースの割り当てを試行します。この割り当てがなされる場合、それは一時的なものです。データベースが再度アクティブにされ、最初に指定されたバッファー・プールがアクティブである場合、 DB2 は表スペースをそのバッファー・プールに割り当てます。

ALTER TABLESPACE ステートメントを使って、異なるページ・サイズを使用するバッファー・プールを表スペースに追加することはできません。

バッファー・プールの作成または変更を行う場合は、データベースの開始時にすべてのバッファー・プールの割り振りができるように、すべてのバッファー・プールで必要なメモリーの合計分がデータベース・マネージャーに対して使用可能になっている必要があります。データベースの開始時にこのメモリーが使用可能になっていない場合、データベース・マネージャーはデフォルト・バッファー・プール (IBMDEFAULTBP) の開始と、別のページ・サイズで定義されているいずれかのバッファー・プールの開始を試みますが、そのサイズは最小サイズになります。 この最小バッファー・プールのサイズは、レジストリー変数 DB2_OVERRIDE_BPF で指定変更することができます。この (および他の) レジストリー変数と環境変数の詳細については、 501ページの『付録A. DB2 レジストリー変数と環境変数』を参照してください。バッファー・プールの開始が失敗するたびに警告メッセージが戻されます。データベースは、データベースの構成が変更されてデータベースが完全に再始動可能になるまでは、この操作状況で続行されます。

データベース・マネージャーを最小サイズの値で開始する理由は、データベースに接続できるようにするためです。これで、バッファー・プール・サイズを再編成するか、他の重要なタスクを実行して、正しいバッファー・プール・サイズでデータベースを再始動できます。そうした状況で、長時間に渡るデータベースの操作は行わないでください。

注: デフォルト・バッファー・プールに関するサイズおよび属性は変更することはできますが、除去することはできません。また、各バッファー・プールには、使用しているプラットフォームに基づいて最小サイズが決められています。

バッファー・プールに大きなメモリーを割り振ると、いくつかの利点があります。バッファー・プールのサイズを大きくした場合の利点について、以下に例をあげて説明します。

- 頻繁に要求されるデータ・ページをバッファー・プール内に保持しておけるので、迅速にアクセスができるようになります。入出力操作を減らすと入出力の競合も減るので、結果として、応答時間が短縮したり、入出力操作に必要なプロセッサー・リソースを減らすことになります。
- 同じ応答時間で、より高いトランザクション率を達成する可能性があります。
- 頻繁に使用するカタログ表などのディスク記憶装置や頻繁に参照するユーザー表および索引に関する、入出力の競合を回避することができます。また、一時表スペースを含むディスク記憶装置上の入出力の競合が減少すると、照会によって要求される分類の速度も速くなります。

1 つあるいは複数のバッファー・プールの選択

使用しているシステムに以下の条件のいずれかが当てはまる場合には、単一のバッファー・プールのみを使用すべきです。

- 合計バッファー・スペースが 10 000 ページ (4 KB 単位) より小さい。
- アプリケーションの知識があって目的に合ったチューニングができる担当者がいない。

- テスト・システム上で作業中である。

使用しているシステムが上記の 3 つの条件とは関係ない場合には、複数のバッファー・プールを使用した場合に改善される可能性のあるパフォーマンスについて以下で説明しますので、検討してみてください。

- 一時表スペースを別のバッファー・プールに移すことができるので、特に分類が多い照会など、一時記憶を必要とする照会のパフォーマンスを向上させることができます。
- たくさんの小さな更新トランザクション・アプリケーションから、繰り返し迅速にアクセスする必要があるデータを持っている場合には、そのデータを含む表スペースを別のバッファー・プールに移動することを検討してください。このバッファー・プールのサイズが適切ならば、ページが見つかる可能性が高くなるので、応答時間を短縮したりトランザクション・コストを下げるのに役立ちます。
- データを別のバッファー・プールに分離し、特定のアプリケーション、データ、索引を特別扱いにすることができます。たとえば、頻繁に更新される表や索引を入れるバッファー・プールを、頻繁に照会されるけれども更新は多くない表や索引のバッファー・プールとは別々にしたいという場合などに役立ちます。このようにすると、頻繁な更新 (表の最初のセット) が頻繁な照会 (表の 2 番目のセット) に与える影響を減らすことができます。
- めったに使用されないアプリケーションによってアクセスされるデータに対しては、小さなバッファー・プールを使用することができます (特に、アプリケーションが非常に大きな表に対して非常にランダムなアクセスを要求するような場合には、有効です)。このような場合には、1 回の照会分より長い時間、バッファー・プール・メモリー内にデータを保持しておく必要はありません。このデータには小さなバッファー・プールを用意して、それで余ったメモリーは他の用途 (たとえば、他のバッファー・プール) のために空けておくのが望ましい方法です。
- 活動やデータを種類によって別々のバッファー・プールに分類して入れると、パフォーマンスは向上はしたが相対的にコストがかかっていないという診断データが、統計および会計トレースから出される場合があります。

バッファー・プールへのデータの事前取り出し

事前に索引ページおよびデータ・ページをバッファー・プールに取り出しておくと、入出力を完了するのを待つ時間が減るので、パフォーマンスは向上します。ページの*事前取り出し* とは、いくつかのページを、後で使用するという想定でディスクからリトリーブしておくということです。事前取り出しには、2 つのカテゴリーがあります。

- *順次事前取り出し* とは、アプリケーションでページが必要になる前に、連続したページをバッファー・プールに読み込む機構のことです。 (254ページの『順次事前取り出しについて』を参照してください。)

- *リスト事前取り出し* またはリスト順次事前取り出しとは、必要なデータ・ページが連続していない場合でも、効果的にデータ・ページにアクセスする方法の 1 つです。 (256ページの『リスト事前取り出しについて』を参照してください。)

これらの 2 つのデータ・ページ読み取り方法は、通常の読み取りを補うものです。通常の読み取りは、1 ページまたは少数の連続ページがリトリーブされるときに使用されます。通常の読み取り時には、データの 1 ページが転送されます。

事前取り出しを使用可能にする方法についての詳細は、 256ページの『事前取り出しおよび並列入出力を行うための入出力サーバーの構成』を参照してください。

順次事前取り出しについて

1 回の入出力操作で複数の連続したページをバッファー・プールに読み込むと、アプリケーション実行時のオーバーヘッドは大幅に短縮されます。さらに、複数の入出力操作を並列して行うことによって複数のページ範囲を同時に読み込むと、アプリケーションが入出力操作の完了を待機する時間を短縮するのに役立ちます。

事前取り出しが開始されるのは、データベース・マネージャーが、順次入出力が適切であり、事前取り出しを行うとパフォーマンスが向上すると判断したときです。表走査や表分類などの場合、データベース・マネージャーは、順次事前取り出しにより入出力のパフォーマンスが向上することを容易に判断できます。 このような場合は、データベース・マネージャーは自動的に順次事前取り出しを開始します。 たとえば、次の例では表走査が必要になるので、順次事前取り出しが実行されます。

```
SELECT NAME FROM EMPLOYEE
```

データベース・マネージャーが事前取り出しで取り出すページ数は、各表スペースごとに CREATE TABLESPACE または ALTER TABLESPACE のいずれかのステートメントに PREFETCHSIZE 文節を指定して、定義することができます。 指定された値は、SYSCAT.TABLESPACES システム・カタログ表の PREFETCHSIZE 列に格納されます。

PREFETCHSIZE の値には、ユーザーの表スペースの EXTENTSIZE の値に表スペース・コンテナーの数を掛け合わせた値を、明示的に指定することをお勧めします。 (エクステント・サイズは、別のコンテナーを使用する前に、データベース・マネージャーがコンテナーに書き込むページ数のことです。 *管理の手引き: 計画* の『表スペースの設計と選択』を参照してください。) たとえば、エクステント・サイズが 16 ページで、表スペースがコンテナーを 2 つ持っていたとすると、事前取り出し量は 32 ページに設定することができます。

データベース・マネージャーは、バッファー・プールの使用をモニターしているので、データの事前取り出しを行ったために、別の作業単位で必要なページが、バッファー・

プールから除去されることはありません。 データベース・マネージャーは、問題を避けるために、事前に取り出すページ数を、ユーザーが表スペースに関して指定した数量以下に制限することがあります。

事前取り出しサイズの設定値によっては、特に大きな表の走査時に、パフォーマンスを大幅に向上させることができます。ユーザーの表スペース用に指定された PREFETCHSIZE を調整するために、データベース・システム・モニターおよび他のシステム・モニター・ツールを使用することができます。たとえば、次のような情報を得ることができます。

- オペレーティング・システムで使用可能なモニター・ツールを使用して、照会時に入出力待機時間が生じているかを調べることができます。
- データベース・システム・モニターによって提供された *pool_async_data_reads (バッファー・プール非同期データ読み取り)* データ要素を調べることによって、事前取り出しが行われているかを知ることができます。詳しくは、*システム・モニター 手引きおよび解説書* を参照してください。

照会時にデータの事前取り出しが行われているのにまだ入出力の待機時間が生じている場合は、 PREFETCHSIZE の値を増やしてみることができます。ただし、事前取り出し以外の原因により入出力待機が生じていることもあり、その場合は PREFETCHSIZE の値を増やしても照会のパフォーマンスは向上しません。

どのタイプの事前取り出しでも、事前取り出しサイズが表スペースのエクステント・サイズの倍数になっており、表スペースのエクステントが別個のコンテナーにあるときには、複数の入出力操作を並列に実行することができます。パフォーマンスを向上させるには、コンテナーが別個の物理装置を使用するように構成されていなければなりません。並列事前取り出しに関する詳細については、256ページの『事前取り出しおよび並列入出力を行うための入出力サーバーの構成』を参照してください。

順次検出について

順次事前取り出しがパフォーマンスの向上につながるかどうかが、瞬時には判断しにくい場合があります。このような場合、データベース・マネージャーが入出力をモニターし、ページが順次に読み取られている場合に事前取り出しを活動化することができます。 このように行う事前取り出しでは、データベース・マネージャーが事前取り出しを活動化すべきと判断したときは活動化し、非活動化すべきと判断したときに非活動化することができます。このタイプの順次事前取り出しのことを*順次検出* といい、索引ページとデータ・ページの両方に適用されます。データベース・マネージャーが順次検出を実行するかどうかは、 *seqdetect* 構成パラメーター (395ページの『順次検出フラグ (seqdetect)』を参照) を使って制御することができます。 順次検出がオンになっている場合には、たとえば以下のような SQL ステートメントがあったとき、そのステートメントに順次取り出しを用いると効果があると判別することができます。

```
SELECT NAME FROM EMPLOYEE
WHERE EMPNO BETWEEN 100 AND 3000
```

この例の場合、最適化プログラムは、EMPNO 列の索引を使って表走査を実行することがあります。この索引に関して表が高度にクラスター化されていると、データ・ページの読み取りはほぼ順次になるので、事前取り出しによってパフォーマンスが向上する可能性があります。この場合、データ・ページの事前取り出しが行われます。

この例では、索引ページの事前取り出しが行われます。大量の索引ページを検査する必要があるときに、データベース・マネージャーが索引ページの順次ページ読み取りが行われていることを見つけた場合には、索引ページの事前取り出しが行われます。

リスト事前取り出しについて

リスト事前取り出し またはリスト順次事前取り出しとは、必要なデータ・ページが連続していない場合でも、効果的にデータ・ページにアクセスする方法の 1 つです。リスト事前取り出しは、単一または複数の索引アクセスで用います。

事前取り出しおよび区画内並列操作

事前取り出しは、区画内並列操作 (索引または表の走査時に複数のサブエージェントを使用する操作) のパフォーマンスにとっては非常に重要です。これらの並列走査によりデータの使用速度が高くなり、高速の事前取り出しが必要になります。

事前取り出しが不十分だと、順次走査よりも並列走査のほうがコストが高くつきます。順次走査の実行時に事前取り出しが行われない場合には、エージェントが常に入出力を待機しなければならないので、照会の実行速度が遅くなります。並列走査の実行時に事前取り出しが行われない場合には、入出力を待機しているサブエージェントが 1 つあると、他のすべてのサブエージェントがそのサブエージェントを待機しなければならなくなります。

区画内並列操作の場合には事前取り出しの重要性が高いため、並列操作はより積極的に実行されます。順次検出機構は、隣接ページ間に大きなギャップがあってもそれを許容し、それらのページは順次であると見なされます。ないものとして扱うギャップの幅は、走査に関係するサブエージェントの数が多いほど大きくなります。

事前取り出しおよび並列入出力を行うための入出力サーバーの構成

事前取り出しを使用可能にするために、データベース・マネージャーは*入出力*サーバーと呼ばれる別の制御スレッドを開始して、ページの読み取りを実行します。 その結果、照会処理は 2 つの並列のアクティビティー、つまりデータ処理 (CPU) とデータ・ページ入出力に分けられます。入出力サーバーは、CPU の処理アクティビティーから事前取り出し要求が出るまで待機します。この事前取り出し要求には、予想される必須データのための入出力記述が含まれます。事前取り出しの目的に応じて、データベース・マネージャーが事前取り出し要求を生成する時点と方法が決定されます。 (詳細については、254ページの『順次事前取り出しについて』および 『リスト事前取り出しについて』を参照してください。)

次の図は、データの事前取り出しを行ってバッファー・プールに入れる際に、入出力サーバーがどのように使用されるかについて示しています。

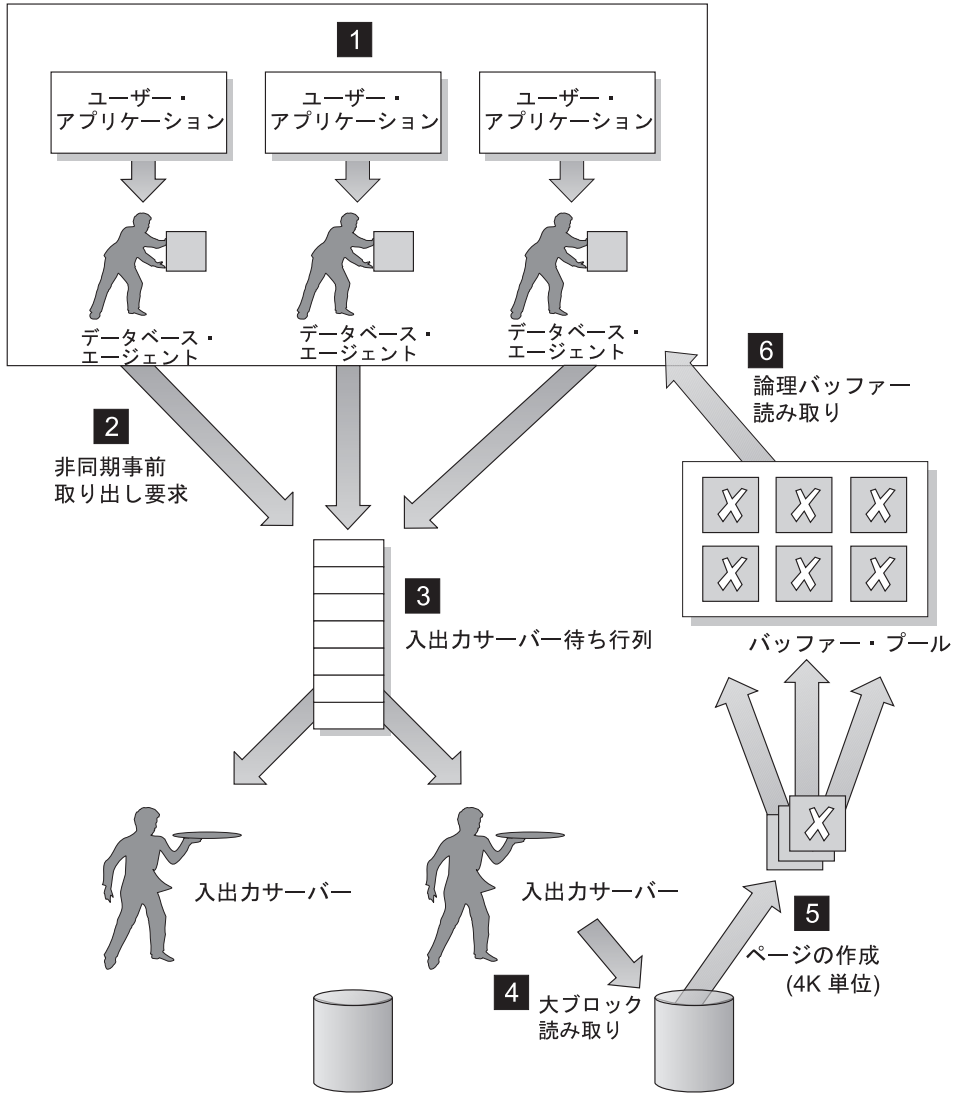


図 27. 入出力サーバーを使用したデータの事前取り出し

図27 で示されているステップについて説明します。

1 ユーザー・アプリケーションが、データベース・マネージャーによってユーザー・アプリケーションに割り当てられているデータベース・エージェントに SQL 要求を渡します。

2、 **3**

データベース・エージェントが、SQL 要求を満たすのに必要なデータを獲得するために、事前取り出しを使用しなければならないかどうかを判別し、事前取り出し要求を入出力サーバー・キューに書き込みます。

4、 **5**

利用可能な最初の入出力サーバーが、キューから事前取り出し要求を読み取り、表スペースのデータをバッファー・プールに読み込みます。キューに入っている事前取り出し要求の数と *num_ioservers* 構成パラメーターによって構成される入出力サーバーの数によっては、複数の入出力サーバーが表スペースから同時にデータを取り出すことがあります。

6 データベース・エージェントが、バッファー・プール内のデータ・ページに対して必須のアクションを実行し、 SQL 要求の結果をユーザー・アプリケーションに戻します。

num_ioservers 構成パラメーターを使用して、十分な数の入出力サーバーを構成すると、データの事前取り出しが適用される照会のパフォーマンスは大幅に向上します。予備の入出力サーバーを構成しても、余分な入出力サーバーは使用されず、そのメモリー・ページはページアウトされるので、パフォーマンスには影響ありません。それぞれの入出力サーバー・プロセスには番号が付けられ、利用可能なプロセスのうち番号が一番小さいプロセスが常にデータベース・マネージャーで使用されます。結果として、番号が大きいプロセスは一度も使用されないことがあります。

構成する入出力サーバーの数を決定する際は、次の点を考慮するようにしてください。

- データベースで実行できる並行アクティビティーの数量。つまり、指定された時間に入出力サーバーのキューに事前取り出し要求を書き込むことができるデータベース・エージェントの数。
- 入出力サーバーが並列で作業できる最高度。詳細については、『並列入出力を使用可能にする』を参照してください。

並列入出力の機会を最大にするには、 *num_ioservers* をデータベース内の物理ディスクの数以上に設定します。

並列入出力を使用可能にする

1 つの表スペースに対して複数のコンテナーが存在する状況では、データベース・マネージャーは、*並列入出力* を開始することができます。並列入出力とは、データベース・マネージャーが複数の入出力サーバーを使用して、単一照会の入出力要件を処理する機能のことです。各入出力サーバーにはそれぞれ別個のコンテナーごとに入出力作業負荷が割り当てられるため、複数のコンテナーを並列で読み取ることができます。入出力を並列で実行すると、入出力スループットが大幅に向上することになります。

別個の入出力サーバーはコンテナーごとに作業負荷を処理しますが、入出力を並列で実行できる入出力サーバーの実際の数は、要求されたデータが伝搬される物理装置の数に制限されます。これは、できるだけ多くの数の入出力サーバーが必要であるということも意味します。

並列入出力の開始および使用方法は、入出力を実行する理由によって決まります。

- **順次事前取り出し**

  順次事前取り出しの場合、事前取り出しサイズが表スペースのエクステント・サイズの倍数になっていると、並列入出力が開始されます。それぞれの事前取り出し要求は、エクステント境界に沿って、複数の小さい要求に分割されます。その後、これらの小さい要求は別々の入出力サーバーに割り当てられます。

- **リスト事前取り出し**

  リスト事前取り出しの場合、ページの各リストは、データ・ページが保管されるコンテナーに従って、小さいリストに分割されます。その後、これらの小さい要求は別々の入出力サーバーに割り当てられます。

- **データベースまたは表スペースのバックアップと復元**

  データのバックアップまたは復元の場合、並列入出力要求の数は、コンテナーの数と同じ最大値までに相当するエクステント・サイズに分割された、バックアップ・バッファー・サイズと等しくなります。

- **データベースまたは表スペースの復元**

  データの復元の場合、並列入出力要求が開始され、順次事前取り出しに使用されるのと同じ方法で分割されます。データをバッファー・プールに復元する代わりに、そのデータは復元バッファーからディスクに直接読み取られます。

- **ロード**

  データをロードする場合、LOAD コマンドの DISK_PARALLELISM オプションを使用すると、入出力並列操作のレベルを指定することができます。 (指定しなかった場合は、その表に関連するすべての表スペースの表スペース・コンテナーの累計値に基づくデフォルトが使用されます。)

並列入出力で最適なパフォーマンスを得るには、以下を確認してください。

- 十分な入出力サーバーがあること。入出力サーバーの数が、データベース内のすべての表スペースに使用されるコンテナーの数よりも、少し大きくなるように構成しなければなりません。
- エクステント・サイズおよび事前取り出しサイズが表スペースに適していること。バッファー・プールの使い過ぎを避けるため、事前取り出しサイズは大きすぎてはなりません。 (理想的なサイズは、エクステント・サイズに表スペース・コンテナーの数を掛け合わせた値です。) エクステント・サイズはかなり小さくする必要があり、適した値は 8 ～ 32 ページの範囲です。
- コンテナーが、別々の物理ドライブに常駐するように構成されていること。

- 並列操作の程度に一貫性を持たせるために、すべてのコンテナーが同じサイズであること。

  他のコンテナーより小さいコンテナーが 1 つまたは複数ある場合には、並列事前取り出しが最適化される可能性が少なくなります。たとえば、次のとおりです。

  – 他より小さいコンテナーが満杯になると、それから先のデータは残りの、他のコンテナーに保管されることになるので、コンテナー間の均衡がとれなくなります。コンテナーが不均衡になると、データの事前取り出しができるコンテナーの数が、コンテナーの合計数より少なくなるので、並列事前取り出しのパフォーマンスが低下します。

  – 他より小さいコンテナーが後で追加されたときにデータの均衡が再度図られた場合には、小さなコンテナーには他のコンテナーより少ないデータが入れられることになります。他のコンテナーに比べて少量のデータでは、並列事前取り出しは最適化されません。

  – ある 1 つのコンテナーが他より大きいときに、他のコンテナーがすべて満杯になったとすると、そのコンテナーが追加のデータを保管できる唯一のコンテナーになります。したがって、データベース・マネージャーは、この追加データにアクセスする際には、並列事前取り出しを使用することができなくなります。

- 区画内並列操作を使用する場合には、十分な入出力容量が用意されていること。区画内並列操作を SMP マシン上で使用すると、複数のプロセッサー上で照会を実行することができるので、照会の経過時間を短縮することができます。各プロセッサーを十分に活用するには、十分な入出力容量が必要です。通常、十分な入出力容量を提供するためには追加の物理ドライブが必要になります。

  事前取り出しは高速で実行し、入出力能力を効率良く使用する必要があります。事前取り出しが高速で行われるためには、事前取り出しサイズは大きくする必要があります。事前取り出しサイズは、エクステント・サイズに表スペース・コンテナーの数を掛け合わせた値にするようにしてください。理想的には、コンテナーは別々の物理ドライブに常駐するように構成する必要があります。

  必要となる物理ドライブの数は、ドライブと入出力バスの速度と容量、およびプロセッサーの速度によって変わります。

複数ページの同時割り振り

SMS 表スペースは、必要に応じて拡張されます。この拡張は、デフォルトには 1 ページずつ行われます。ただし、特定のワーク・ロード (たとえば、大量の挿入を行っている場合) では、 *db2empfa* ツールを使って、ページ・グループ単位またはエクステント・グループ単位で表スペースを拡張するよう DB2 に指示することにより、パフォーマンスを向上させることができます。 *db2empfa* ツールは、 *sqllib* ディレクトリーの *bin* サブディレクトリーにあります。このツールを実行すると、*multipage_alloc* データベース構成パラメーターが Yes に設定されます。 このツールの詳細については、コマンド*解説書* を参照してください。

使用可能メモリーを最大限活用する別の方法については、 278ページの『メモリーの拡張』で説明されています。

分類

分類は照会中に頻繁に必要になるので、分類ヒープ領域を正しく構成することは照会のパフォーマンス上きわめて重要になります。次のような場合に、分類が必要になります。

- 要求された配列を満たす索引がない (たとえば、 ORDER BY 文節を使用する SELECT ステートメント)
- 索引はあるが、索引を使用するよりも分類を行う方が効率が良い場合
- 索引の作成時 (*indexsort* 構成パラメーターが YES に設定されている場合)

さまざまなタイプの分類

分類には 2 つのステップが関係しています。

1. 分類フェーズ
2. 分類フェーズの結果を戻すフェーズ

この 2 つのステップで分類が処理される方法はいくつかのカテゴリーまたはタイプに分けられるので、以下ではそれを用いて分類の説明をします。分類フェーズを考慮する際、分類を『オーバーフロー』と『オーバーフローなし』に分けることができます。分類フェーズの結果の戻りを考慮する際、分類を『パイプ』と『非パイプ』に分けることができます。

オーバーフローとオーバーフローなし
: 分類される情報が分類ヒープ (分類の実行ごとに割り当てられるメモリーのブロック) に全く収まらない場合は、一時データベース表にオーバーフローします。オーバーフローしない分類の方が常に、オーバーフローする分類よりも効率良く実行されます。

パイプと非パイプ
: データの最終分類結果リストを保管する一時表を使わなくても、分類された情報を直接戻すことができる場合は、その分類は「パイプ分類」と呼ばれます。分類された情報を戻すのに一時表が必要な場合には、その分類は「非パイプ分類」と呼ばれます。パイプ分類は、非パイプ分類よりも常に効率よく実行されます。

分類に影響を与えるパラメーターの調整

分類のパフォーマンスに影響を与える事柄には、以下のものがあります。

- 次の構成パラメーターの設定値

  363ページの『分類ヒープ・サイズ (sortheap)』
  : 分類ごとに使用する記憶容量を指定する

364ページの『分類ヒープのしきい値 (sheapthres)』
すべての分類のインスタンス全体で使用可能な、分類用の合計メモリー量の制御を行う

- 大量の分類に関係するステートメント
- 不要な分類を避けるのに役立つ索引の脱落
- 分類を最小化しないアプリケーション論理
- 並列分類。分類のパフォーマンスを向上させるが、ステートメントが区画内並列操作を使用する場合にのみ行われる (258ページの『並列入出力を使用可能にする』を参照)。

分類パフォーマンス問題の標識の探索

分類に関して総合的な問題があるかどうかを知るには、分類に費やした合計 CPU 時間を、アプリケーション全体で費やした時間と比較してみてください。比較には、データベース・システム・モニターが役立ちます (275ページの『データベース・システム・モニターの使用法』を参照してください)。特に、「スナップショット・モニター」および「イベント・モニター」で構成され、コントロール・センターから使用可能なパフォーマンス・モニターは、デフォルト解釈では、*合計分類時間* と共に *入出力時間* および*ロック待機時間* などの時間を示します。

合計分類時間がその他の時間に占める比率が大きい場合には、やはりデフォルト解釈で示される以下の値を調べてみてください。

オーバーフロー分類のパーセンテージ
(スナップショット・モニターのパフォーマンス詳細画面上に示された) この変数は、オーバーフローした分類のパーセンテージを示します。オーバーフロー分類のパーセンテージが高いときは、ポストしきい値分類があった場合には、構成パラメーターの *sortheap* か *sheapthres* (または、その両方) を大きくしてください。 (ポストしきい値分類があったかどうかを判別するには、スナップショット・モニターを使用します。)

ポストしきい値分類
ポストしきい値分類の値が高い場合には、*sheapthres* を大きくするか、*sortheap* を小さくしてください (または、その両方を行ってください)。

一般に、インスタンス (*sheapthres*) で使用可能な分類メモリーの全体量は、過度のページングを引き起こさない限り、できるだけ大きくするようにしてください。分類全体を分類メモリーで行うことができます。ただし、その分類メモリーの増大に対応するために、オペレーティング・システムが過度のページ・スワッピングを実行するようになった場合には、大きな分類ヒープの利点がなくなる可能性があります。そのため、分類構成パラメーターを調整するときには、オペレーティング・システム・モニターを用いて、システム・ページングに変更があるかどうかを突き止めてください。

注: DB2 部分キーのバイナリー分類技法に、非整数のデータ・タイプ・キーを組み込むように改善がなされたので、長いキーを分類する時には追加のメモリーが必要です。長いキーが使用されていると思う場合、 *sortheap* 構成パラメーターを増やしてください。

また、パイプ分類では、アプリケーションがその分類に関連したカーソルをクローズするまで、分類ヒープが解放されないことに注意してください。そのため、パイプ分類はカーソルがクローズされるまでメモリーを消費することができます。

分類パフォーマンスの管理技法

データベース・システム・モニターとベンチマーク技法を使用すると、構成パラメーター *sortheap* および *sheapthres* を設定するのに役立ちます。データベース・マネージャーとそのデータベースごとに、次のことを実行してください。

- 代表的な作業負荷を設定し稼働する。
- 適用するデータベースごとに、ベンチマーク作業負荷期間中の次のパフォーマンス変数の平均値を収集する。
  - 使用中の合計分類ヒープ
  - 活動中の分類

  これらのパフォーマンス変数は、スナップショット・モニターのパフォーマンス詳細画面に示されます。
- データベースごとに、 *sortheap* を *使用中の合計分類ヒープ* の平均値に設定する。
- *sheapthres* は、次のように設定します。
  1. インスタンス中で最大の *sortheap* 値を持つ、データベースを判別します。
  2. このデータベースの分類ヒープの平均サイズを判別します。

     判別するのが困難である場合、最大の分類ヒープの 80% を使用します。
  3. 活動中の分類の平均数に、算出した分類ヒープの平均サイズを掛けた値に *sheapthres* を設定する。

     これは、初期設定としてお勧めします。次に、ベンチマーク技法を使用して、この値をより良いものにすることができます。

さらに、分類が重大なパフォーマンス問題となっている特定のアプリケーションおよびステートメントを識別することができます。

- イベント・モニターをアプリケーションおよびステートメントのレベルでセットアップして、合計分類時間が最も長いアプリケーションを識別する。
- そのようなアプリケーションのそれぞれにおいて、*合計分類時間* が最も長いステートメントを見つける。
- Visual Explain などのツールを用いて、そのようなステートメントを調整する。
- 適切な索引があるかどうかを確認する。 Visual Explain を用いて、指定されたステートメントのすべての分類操作を識別することができます。次に、ステートメントがアクセスする表ごとに、適切な索引があるかどうかを調べます。

注: Explain 表を探索して、どの照会で分類操作が行われているかを識別することができます。 (573ページの『付録C. SQL EXPLAIN ツール』を参照してください。)

カタログとユーザー表の再編成

索引を使用する SQL ステートメントのパフォーマンスは、大量の更新、削除、または挿入が行われた後に低下することがあります。普通、新しく挿入された行は、索引で定義されている論理順序と同じ順序には物理的に配置されません (クラスター索引を使用していなければ)。論理上の順次データは、実際には順序どおりにはなっていない物理的なデータ・ページに入っていることもあるので、データベース・マネージャーはデータにアクセスするのに追加の読み取り操作を実行する必要があります。

一般的に、表の再編成には、統計を実行するより長い時間がかかります。データの現在の統計を収集し、アプリケーションを再バインドするだけで、パフォーマンスは十分向上することがありますので、まずその方法を試してください。それでもパフォーマンスが向上しない場合は、表や索引のデータの配置が有効でない可能性があります。その場合は、再編成が役に立つかもしれません。このセクションの情報はユーザー独自の表の再編成だけでなく、同様に再編成が必要になったシステム・カタログ表にも適用されます。

タイプ付き表の場合、指定した表名は、階層のルート表の名前と同じでなければなりません。

REORGCHK コマンドは、表の物理的特性に関する情報を戻し、表を再編成した方が利点が多いかどうかを示します。このコマンドは、ローカルおよびリモート・ユーザーのどちらも使用できます。コマンド出力の解釈方法など詳細については、*コマンド解説書* を参照してください。

注: REORGCHK コマンドは、拡張索引、または宣言済み一時表のいかなるデータも表示しません。

REORG ユーティリティーは、任意指定で、指定された索引に従ってデータを物理順序で再配置します。 REORG には、索引を使用して表の行の順序を指定するというオプションがあり、このオプションを選択すると、その索引に従って表データがクラスター化され、 RUNSTATS ユーティリティーが収集する CLUSTERRATIO 統計または CLUSTERFACTOR 統計が改善されます。その結果、索引の順序に行を並べる必要がある SQL ステートメントの処理を効率的に行うことができるようになります。未使用で空のスペースを除去すると、 REORG は表をよりコンパクトに保管することができます (スペースを未使用のままにする ALTER TABLE を使用した時に PCTFREE を指定した場合でも可能)。

REORG または REORGCHK コマンドではニックネームを使用しないでください。

REORG ユーティリティーでは、影響を受ける表データと索引を操作する他のすべてのアプリケーションがオフラインでなければなりません。作業環境によっては、アプリケーションがデータを操作できない時間の長さを限定することができます。この環境では、オンラインの索引再編成ユーティリティーの使用を考慮することができます。

再編成のときに行われる索引の再作成で必要なログ・スペースは、以下の式を使用して計算されます。

```
2 * (10500 + ((number of index pages / extent size) * 110) +
(number of index pages * 45) +
(number of index pages / 16000) * 64 ))
```

計算のそれぞれの部分で、索引が再作成されるときにログに作成されるもの、および記録されるものと関連したオーバーヘッドの様々なタイプが決定されます。

表データの再編成をいつ行うかを決定する際に、次の要素を考慮することができます。

- 挿入、更新、および削除アクティビティーのボリューム
- クラスター率の高い索引を使用する照会のパフォーマンスに顕著な変化があるかどうか
- 統計 (RUNSTATS) を実行しても照会のパフォーマンスが向上しない
- REORGCHK コマンドが、表を再編成する必要があることを示している
- 表を再編成した場合のコスト (REORG ユーティリティーが再編成の完了まで表をロックすることによって生じる CPU 時間、経過時間、および並行性の低下を含む)

REORG ユーティリティーを実行するには、対象となる表に関する SYSADM、SYSMAINT、 SYSCTRL、 DBADM のいずれかの権限か、CONTROL 特権が必要です。

REORG ユーティリティーは一時表を使用しますが、この表は、表に列が追加された場合や表に LOB 列がある場合には、元の表よりもかなり大きくなる場合があります。これらの一時表を大きくすると、 REORG ユーティリティーによって作成された永続表も大きくなります。

注: REORG ユーティリティーを使用しても、宣言済み一時表を再編成することはできません。

REORG ユーティリティーを利用すると、一時 REORG 表を作成するための一時表スペースを指定することができます。一時表スペースを指定しないと、REORG ユーティリティーは、再編成される表が含まれているのと同じ表スペースに、一時 REORG 表を作成します。次の指針は、一時表スペースを使用するかどうかを判別するのに助けになります。

- 一時表スペースを指定する場合、一般的には、SMS 一時表スペースを指定することをお勧めします。 DMS 一時表スペースを使用すると、REORG を 1 つだけしか進行できないので、このタイプの表スペースはお勧めしません。

- 一般的には、一時 SMS 表スペースを指定することをお勧めします。複数の表の再編成に同じ表スペースを使用すると、処理は高速化されますが、ロギングの回数が増え、再編成する表のための十分なスペースが必要となります。一時表スペースを指定する場合、一般的には、SMS 一時表スペースを指定することをお勧めします。このタイプの表スペースでしか REORG を実行できないため、 DMS 一時表スペースはお勧めしません。

REORG ユーティリティーは、オープン・カーソルを暗黙的にクローズします。

8 KB のページを使用している表スペース内で表を再編成している、ということを忘れないでください。再編成中に使用される一時表スペースには、基本表スペースと同サイズのページがなければなりません。

REORG が正常に完了しなかった場合は、一時ファイル、表、または表スペースを**削除しないようにしてください**。これらのファイルや表は、障害が起こる前にどの程度の再編成が進行していたのかによって、 REORG ユーティリティーが行った変更をロールバックしたり、再編成を完了したりするためにデータベース・マネージャーが使用するものです。

区分データベースの場合、 REORG ユーティリティーは各区画ごとにデータの再編成を行いますので、ユーティリティーがある区画で失敗した場合には、その失敗した区画のみがロールバックされます。また、一時表を保管するディレクトリー・パスを指定した場合には、このパスは、各データベース区画でデータベース・マネージャーによって拡張されます。 したがって、他のデータベース区画と共用するパスを指定した場合には、一時ファイルは、そのパスの下の (ノード名で識別される) さまざまなサブディレクトリーに保管されます。

オンライン索引再編成

索引の葉ページにあるフリー・スペースの最大量としてユーザー定義可能な限界値を設定すれば、オンライン再編成が可能になります。葉ページから索引キーが削除された場合にこの限界値を超えていると、索引の隣接する葉ページがチェックされ、2 つの葉ページをマージできるかどうか判別されます。 2 つの隣接するページをマージするのに十分なスペースがあれば、データベースをオフラインにしないでマージが行われます。

この索引のオンライン再編成は、バージョン 6 以後のリリースで作成した索引でのみ行うことができます。このようなオンライン再編成の能力を既存の索引が必要とする場合には、索引の葉ページに必要な内部変更を行われるよう、その索引を一度消去してから再作成する必要があります。特定の索引についてオンライン索引再編成をオンにするには、索引の作成時に MINPCTUSED 値を指定します。 MINPCTUSED 値は、100 未満の値に設定する必要があります。この値は再編成の限界値、つまり、隣接する索引の葉ページのマージを試行する前に索引ページで使われるスペースのパーセンテージ (最小許容値) となります。 隣接する索引の葉ページをマージすることが目標なので、 MINPCTUSED の

値は 50 % より小さくすることをお勧めします。 MINPCTUSED の値をゼロ (デフォルト) にすると、オンライン再編成は使用できません。

オンライン索引再編成の終了後に解放された葉ページは、再使用することができます。ただし、解放されたページが使用できるのは、同じ表にある索引に対してだけです。表を完全に再編成すると、DMS 記憶モデルで作業している場合には、他のオブジェクト用にページが解放されます。 SMS 記憶モデルで作業している場合には、ディスク・スペースが解放されます。

索引の葉ページ以外のページは、オンライン索引再編成の終了後に解放されることはありません。しかし、表を完全に再編成すると、可能な限り索引を小さくすることができます。葉ページと葉ページ以外のページは、索引のレベルと同じ数だけ削減されます。

表を再編成する必要を削減する

表の再編成の必要を削減するには、表を作成した後に以下の事柄を行ってください。

- 表を変更して PCTFREE を追加する。
- 索引上の PCTFREE を使ってクラスター化索引を作成する。
- データを分類する。
- データをロードする。

既存の表に対して上記のいずれかを行った後、クラスター化索引を持つ表が現れます。クラスター化索引は、表上の PCTFREE に関連して、元の分類順序を保ちます。ページ上に十分なスペースができたので、正しいページに新しいデータを挿入することができます。これにより、クラスター化索引のクラスター化特性が保持されます。データがさらに挿入され、表のページがいっぱいになると、表の最後にレコードが追加され、表のクラスター特性は徐々に失われます。

クラスター化索引の作成後、REORG または分類と LOAD の実行をお勧めします。クラスター化索引はデータの特定の順序を保持しようとし、 RUNSTATS ユーティリティーが収集した CLUSTERRATIO または CLUSTERFACTOR 統計を改善します。

REORG 中に各ページに残されるフリー・スペースの量は、表の PCTFREE 値によって決まります。この値が設定されていなければ、REORG はデータの再編成時にページを埋めます。

DMS 装置のパフォーマンスに関する考慮事項

表スペース用にデータベース管理ストレージ (DMS) 装置コンテナーを使用している場合は、環境を効果的に管理できるように、次の点を理解している必要があります。

- **ファイル・システム・キャッシュ**

  ファイル・システム・キャッシュは次のように実行されます。

– DMS ファイル・コンテナー (およびすべての SMS コンテナー) の場合、オペレーティング・システムはファイル・システム・キャッシュ中のページをキャッシュすることがある
– DMS デバイス・コンテナーの場合、オペレーティング・システムはファイル・システム・キャッシュ中のページをキャッシュしない

注: DB2 が NOCACHE オプション付きでデータベース・ファイルをオープンするか否かを指定する。 DB2NTNOCACHE=ON の場合は、ファイル・システムのキャッシュは除去される。 DB2NTNOCACHE=OFF の場合、オペレーティング・システムは DB2 ファイルをキャッシュに保存する。これは、LONG FIELDS または LOBS を含んでいるファイルを除くすべてのデータに適用される。システム・キャッシュを除去すると、より多くのメモリーがデータベースに利用できるようになるため、バッファー・プールや分類ヒープの量を増やすことができる。

- **データのバッファリング**

  ディスクから読み取られる表データは通常、データベースのバッファー・プールで使用可能です (245ページの『データベース・バッファー・プールの管理』を参照してください)。アプリケーションが実際にそのページを使用してしまうよりも前に、データ・ページをバッファー・プールから解放できる場合もあります。 (これが起こるのは、バッファー・プール空間がその他のデータ・ページで必要とされる場合です。) システム管理ストレージ (SMS) またはデータベース管理ストレージ (DMS) ファイル・コンテナーを使用する表スペースについては、上記のファイル・システム・キャッシュの説明を参照してください。これにより、キャッシュに入っていなければ必要になった入出力を省くことができます。

  データベース管理ストレージ (DMS) を使用している表スペースは、ファイル・システムもキャッシュも**使用しません**。このような場合には、データベース・バッファー・プールのサイズを大きくし、ファイル・システム・キャッシュのサイズを小さくすることによって、装置コンテナーを使用する DMS 表スペースでは二重バッファリングは実行されないという事実を相殺することができます。

  装置コンテナーを使用する DMS 表スペースの入出力が同等の SMS 表スペースの入出力と比較して大きくなっていることを、システム・レベルのモニター・ツールの使用を介して検出した場合には、この差は、上記で説明した二重バッファリングが原因である可能性があります。

- **LOB データまたは LONG データの使用**

  アプリケーションが LOB データまたは LONG データのいずれかをリトリーブする場合には、データベース・マネージャーは、データのキャッシュにはバッファーを使用しません。アプリケーションがこれらのページの 1 つを要求するたびに、データベース・マネージャーはディスクからリトリーブしなければなりません。

  LOB または LONG データが SMS または DMS ファイル・コンテナーに格納される場合、ファイル・システム・キャッシュでバッファリングが行われ、結果としてパフォーマンスが向上することがあります。

システム・カタログにはいくつかの LOB 列が含まれているので、システム・カタログは SMS (または、DMS ファイル) 表スペースに入れておくことをお勧めします。

初期化オーバーヘッドの管理

ACTIVATE DATABASE コマンドは、選択されたデータベースを始動します。区分データベースでこのコマンドを使用すると、選択された区分データベースをすべての区画上で活動化しようと試みます。 このコマンドを使用すると、データベースの初期化または始動にアプリケーション時間がかかりません。

ACTIVATE DATABASE コマンドを使用して初期化したデータベースは、DEACTIVATE DATABASE コマンドを使用してシャットダウンする必要があります。最後にデータベースから切断されるアプリケーションは、データベースをシャットダウンしません。 ACTIVATE コマンドおよび DEACTIVATE コマンドの詳細については、*コマンド解説書* を参照してください。

データベースが開始されておらず、アプリケーションで CONNECT TO (または暗黙接続) が行われている場合、データベース・マネージャーが必要なデータベースを始動する間待機してからでなければ、アプリケーションはそのデータベースで作業することはできません。 これは、特定のデータベースに最初にアクセスするアプリケーションが負う始動コストです。区分データベースでは、この始動コストは各データベース区画が負います。いったんデータベースが開始されると、その他のアプリケーションは、データベースの始動に関連した時間コストをかけずに、そのデータベースに接続し使用することができます。

データベース・エージェント

DB2 サーバーは、データベース・マネージャーとクライアント / ローカル・アプリケーションとの間のコミュニケーションを支援しなければなりません。 UNIX ベースの環境では、*プロセス* を基本とするアーキテクチャーを使用します。たとえば、DB2 コミュニケーション listener はプロセスとして作成されます。 OS/2 および Windows NT などの Intel オペレーティング・システムでは、*スレッド* を基本とするアーキテクチャーを使用して、パフォーマンスの最大化を図ります。たとえば、DB2 コミュニケーション listener は、 DB2 サーバーのシステム・コントローラー・プロセス内にスレッドとして作成されます。アクセスされる各データベースごとに、さまざまなプロセス / スレッドが開始されて、各種のデータベース・タスク (たとえば、事前取り出し、コミュニケーション、ログ記録など) を処理します。

非常に重要なプロセス / スレッドの 1 つに、データベースに関するアプリケーションの操作を支援するデータベース・エージェントのプロセス / スレッドがあります。

論理エージェント は、データベース・マネージャーに接続されたアプリケーションを表します。論理エージェントには、アプリケーションに必要なすべての情報と制御ブロックがあります。論理エージェントの最大数は、データベース・マネージャー構成パラメ

ーター *max_logicagents* により制御されます。アプリケーションごとに 1 つの論理エージェントがあるので、このパラメーターは、インスタンスに接続できるアプリケーションの最大数を制御します。

作業エージェント は、アプリケーション要求を実行しますが、特定のアプリケーションに永続的に接続されるものではありません。作業エージェントには、アプリケーションが要求したデータベース・マネージャー内のアクションを完了するのに必要なすべての情報と制御ブロックがあります。

作業エージェントには、4 つのタイプがあります。*活動状態の調整エージェント* 、サブエージェント 、*非活動状態のエージェント* 、および*アイドル・エージェント* です。

アイドル・エージェントは、最も単純な形式の作業エージェントです。このエージェントは、論理エージェントに結合されておらず、アウトバウンド接続もありません。また、ローカル・データベース接続も、インスタンス接続もありません。

非活動状態のエージェントは作業エージェントの一形式であり、活動状態のトランザクションにはありません。このエージェントは論理エージェントに結合されておらず、アウトバウンド接続もありません。また、ローカル・データベース接続も、インスタンス接続もありません。非活動エージェントは自由に別の論理エージェントと結合され、その論理エージェントによって表されるアプリケーションの提供を開始します。

クライアント・アプリケーションの各プロセス / スレッドにはそれぞれ、データベース上で稼働する*活動状態の調整エージェント* が 1 つあります。調整エージェントが作成されると、そのエージェントが、アプリケーションに代わって、すべてのデータベース要求を実行し、さらに、プロセス間コミュニケーション (IPC) およびリモート・コミュニケーションのプロトコルを使用して、他のエージェントとコミュニケーションします。各エージェント・プロセスは自らの私用メモリーを使って操作を行いますが、データベース・マネージャーおよびデータベース・グローバル・リソース (バッファー・プールなど) は他のエージェントと共用します。トランザクションが完了すると、活動状態の調整エージェントは論理エージェントから切り離され、非活動状態のエージェントとなります。

区分データベース環境、および区画内並列処理が使用可能になっている環境では、調整エージェントはデータベース要求をサブエージェント に分配し、それらのサブエージェントがアプリケーションの要求を実行します。調整エージェントが作成されると、このエージェントは、データベースへの要求を実行するサブエージェントの調整を行うことによって、アプリケーションに代わってすべてのデータベース要求の処理を行います。

クライアントがデータベースから切断されるか、またはインスタンスから切り離されると、調整エージェントの状態は次のようになります。

- 活動状態のエージェント。他の論理エージェントが待機状態の場合は、作業エージェントが活動状態の調整エージェントになります。

- 他の論理エージェントが待機状態になく、プール・エージェントが最大数に達していない場合は、空き状態になり、アイドル中であることを示すマークが付けられる。
- 他の論理エージェントが待機状態になく、プール・エージェントが最大数に達した場合は、終了して、ストレージが解放される。

どのアプリケーションの代理の作業も実行せず、割り当てられるのを待っているエージェントは、アイドル・エージェントと見なされ、エージェント・プール に常駐します。これらのエージェントは、クライアント・プログラムの代理として作業を行う調整エージェントからの要求のために、あるいは、既存の調整エージェントの代理として作業を行うサブエージェントのために、使用することができます。使用可能なエージェントの数は、データベース・マネージャー構成パラメーター *maxagents* および *num_poolagents* によって変わります。

次の場合に、エージェント・プール (*num_poolagents*) のエージェントは、調整エージェントとして再使用されます。

- リモート TCP/IP ベースのアプリケーションの場合。
- UNIX ベースのオペレーティング・システム上のローカル・アプリケーションの場合。
- Windows NT および OS/2 オペレーティング・システム上のローカル / リモート・アプリケーションの場合。

それ以外の場合、リモート・アプリケーションは常に新規エージェントを作成します。

エージェントが必要なときにアイドル状態のエージェントがない場合には、新しいエージェントが動的に作成されます。ただし、新規のエージェントを作成すると、一定のオーバーヘッドが生じます。このように、クライアントのために活動化できるアイドル状態のエージェントがある方が CONNECT および ATTACH のパフォーマンスは向上します。

あるサブエージェントがアプリケーションの代理として作業を行うときには、そのサブエージェントはそのアプリケーションに*関連付けられている* と見なされます。割り当てられた作業が完了すると、サブエージェントはエージェント・プールに入れられますが、元のアプリケーションとの関連付けはそのまま残されます。そのアプリケーションが追加の作業を要求した場合、そのアプリケーションのために作業を行うエージェントを探すときには、データベース・マネージャーは、まずアイドル・プール内にそのアプリケーションと関連するエージェントがないかどうかを検査します。

接続済みアプリケーションの数と、処理可能なアプリケーション要求の数とを別々に制御する (前者は、 *max_logicagents* により定義される論理エージェントの数を使用して制御し、後者は、 *max_coordagents* により定義される活動調整エージェントの数を使用して制御する) 機能により、柔軟にデータベースで作業負荷を処理することが可能となります。接続済みアプリケーションの数と、処理可能なアプリケーション要求の数の間の 1 対 1 関係は、アプリケーションがデータベースを使用して作業する際の一般的な

状態です。ただし、作業環境によっては、接続済みアプリケーションの数と処理可能アプリケーション要求の数の間で多数対 1 の関係が必要です。

データベース・グローバル・リソースのオーバーヘッドは活動状態の調整エージェントと関連しているので、このエージェントの数が多ければ、使用可能なデータベース・グローバル・リソースの上限に達する可能性も多くなります。活動状態の調整エージェントよりも接続済みアプリケーションを多くすれば、使用可能なデータベース・グローバル・リソースの上限に達しないようにすることができます。 *max_logicagents* に *max_coordagents* の値より大きい値を設定すれば、データベースの作業に集中できます。

DB2 コネクトを XA トランザクション・サポート・コンセントレーターとして使用する方法の詳細と例については、 *DB2 コネクト 使用者の手引き* を参照してください。

リモート・システムに接続するのに DB2 コネクトを使用する必要がある環境で作業している場合には、*アウトバウンド接続プール* が使用されます。この接続プールにより、ホストへの (最初の接続の後の) 接続時間は減少します。ホストからの切断が要求されると、DB2 コネクトはインバウンド接続を除去しますが、プール内のホストへのアウトバウンド接続を保持します。ホストへの接続が新しく要求されると、 DB2 コネクトは既存のアウトバウンド接続 (使用可能な場合) をプールから再使用します。

注: 接続プールを使用する場合、 DB2 コネクトはインバウンド TCP/IP 接続およびアウトバウンド TCP/IP および SNA 接続だけを使用できます。 SNA を使って作業している場合は、接続がプールに入れられるよう、セキュリティー・タイプを NONE にする必要があります。

接続プールを使用する場合、活動状態のエージェントは切断後にアウトバウンド接続をクローズせず、リモート・ホストへの活動状態の接続を使ってエージェント・プールに入ります。このタイプのエージェントは、*非活動 DRDA エージェント* と呼ばれます。非活動 DRDA エージェントのプールは、アウトバウンド接続プールと同じものです。『DRDA』 は、『分散リレーショナル・データベース体系』 を表します。

4 つの異なる使用法および作業負荷要件に基づく、次の例を考慮してください。

1. 最初の例では、平均 40 人のユーザーが同時に DB2 コネクトを介してリモート・ホスト・データベースに接続します。時々、同時に接続する人の数は 50 に達しますが、 55 を超えることはありません。トランザクションの所要時間は短く、ユーザーは頻繁に接続と切断を行います。

   これらの条件により、同時に DB2 コネクトを介して接続を試行するユーザーの最大数が 55 であることが分かっているので、システム管理者は *max_coordagents* を 55 に構成する必要があります。 *num_poolagents* (エージェント・プールのサイズ) は、常に、接続中または接続試行中のユーザーの平均数なので、40 に設定する必要があります。 作業負荷がピークになる場合を除いて、このプール・サイズにより、新しい接続を確立せずにすべてのインバウンド・クライアントを満たすことのできる十分な数のリモート・データベース接続の存在が保証されます。

2. この 2 番目の例には約 1 000 のインバウンド・クライアントが存在し、作業負荷はかなり高くなっています。ユーザー接続は、最初の例と同じく短期間のものとなっています。システム管理者が、これ以上の同時接続を許可したくないとします。それで、システム管理者は *max_coordagents* と *num_poolagents* を両方とも 1 000 に設定します。 これにより、リモート・データベースに同時に接続するインバウンド・クライアントの最大数は 1 000 になります。すべてのクライアントが切断するとき、プールには 1 000 の接続されたエージェントが存在し、新しいインバウンド・クライアントの処理を待機しています。
3. 3 番目の例は、DB2 コネクトを介して 1 つのリモート・データベースに接続する単一のアプリケーションに関するものです。アプリケーションは、長い期間接続されたままになります。このシナリオでは、接続するクライアントが最大でも 1 人であることが分かっているので、エージェントと接続プールの最善の構成は *max_coordagents* を 11 に設定することです。この例では、リモート・ホストから接続と切断が頻繁に繰り返されることがないので、 *num_poolagents* をゼロに設定することができます。リモート・データベースにアクティブな接続を行うエージェントがプールに保持されないため、 *num_poolagents* をゼロに設定すると、事実上接続プールは使用できなくなります。新しいインバウンド・クライアントが接続するたびに、新しいエージェントが作成され、その処理を行う新しいリモート接続が確立されます。
4. 4 番目の例は、これまでの 3 つの作業負荷のシナリオすべてを応用したものです。この例では、システム管理者は、リモート・データベースへの同時アクセスをちょうど 100 に制限したいと思っています。それで、*max_coordagents* を 100 に設定し、接続パフォーマンスを最大にするため *num_poolagents* を 100 に設定します。ただし、DB2 コネクトがインストールされているシステムの作業負荷をモニターするため、後にローカル接続を行わなければならない可能性もあります。例外は、発生する同時モニターのスナップショットの数が常に 5 以下である場合です。この場合は、*max_coordagents* を 105 に設定します。この新しい構成値により、同時に接続するアプリケーションの最大数は以前の上限 100 を超え、偶発的なモニター・スナップショットまたはインスタンス接続 (あるいはその両方) に対応できるようになります。

区分データベース環境、および区画内並列処理が使用可能になっている環境の場合には、各区画 (つまり、各データベース・サーバーまたはノード) が独自のエージェント・プールを持っていて、そこからサブエージェントを引き出すことができます。このプールがあるので、必要になったり作業を終了したりするたびに、サブエージェントを作成したり破棄したりする必要がありません。サブエージェントはプール内に関連エージェントとして残されるので、それらが関連付けされたアプリケーションから新しい要求が出された場合には、データベース・マネージャーによってそれらのサブエージェントが使用されます。

データベース・エージェントの数に影響を与えるデータベース・マネージャー構成パラメーターは、以下のとおりです。

- 405ページの『エージェントの最大数 (maxagents)』。作業エージェントの数がこの値に達すると、これ以降に新しいエージェントを必要とする要求が出されても、エージェントの数がこの値を下回るようになるまでは、それらの要求は拒否されます。この値は、エージェントの合計数 (すべてのアプリケーション上で作業を行う調整エージェント、サブエージェント、非活動状態のエージェント、およびアイドル・エージェントを含む) に対して適用されます。
- 409ページの『エージェント・プール・サイズ (num_poolagents)』。エージェント・プール内の非活動状態のエージェント、アイドル・エージェント、および関連するサブエージェントの数は、この値を超えることはできません。
- 410ページの『プール内の初期エージェント数 (num_initagents)』。データベース・マネージャーの開始時に、この値に基づいて作業エージェントのプールが作成されます。これによって、初期の照会のパフォーマンスが速くなります。作業エージェントはすべてアイドル・エージェントとして開始します。
- 408ページの『論理エージェントの最大数 (max_logicagents)』。論理エージェントの最大数。アプリケーションごとに 1 つの論理エージェントがあるので、このパラメーターは、インスタンスに接続できるアプリケーションの最大数を制御します。
- 407ページの『調整エージェントの最大数 (max_coordagents)』。区分データベース環境および区画内並列処理が使用可能になっている環境の場合には、この値によって調整エージェントの数が制限されます。
- 406ページの『並行エージェントの最大数 (maxcagents)』 この値は、データベース・マネージャーが許可するトークン の数を制御します。クライアントがデータベースに接続しているときにデータベース・トランザクション (作業単位) が生じるたびに、調整エージェントはそのトランザクションを処理する許可 (処理トークンという) をデータベース・マネージャーから得る必要があります。 データベース・マネージャーは、処理トークンを持っているエージェントだけに、データベースに対する作業単位の実行を許可します。 トークンが利用不能な場合は、トークンが利用可能になるまでエージェントは待機します。トークンが利用可能になると、要求された作業単位が処理されます。

  このパラメーターは、ピーク時の使用要件がシステム・リソース (メモリー、 CPU、およびディスク) を超える環境で役立ちます。このような環境では、ピーク時のロードがページングなどの理由で極端な性能低下を引き起こす可能性があります。このパラメーターを使用して、負荷の制御や性能の低下を防ぐことができます。ただし、これは並行性または待ち時間、あるいはその両方に影響します。

区分データベース環境および区画内並列処理が使用可能になっている環境の場合、システム内のパフォーマンスとメモリー・コストへの影響は、以下に示すエージェント・プールのチューニング方法に強く関係しています。

- エージェント・プール・サイズに関するデータベース・マネージャー構成パラメーター (*num_poolagents*) は、 1 つの区画 (つまり、ノード) でアプリケーションとの関連付けを保持できるサブエージェントの数に影響します。プール・サイズが小さすぎる (ので、プールが満杯になった) 場合には、サブエージェントは、作業を行ったア

プリケーションと自分自身との関連付けを切り離し、終了します。この状況では、サブエージェントを作成してアプリケーションと再度関連付けをするということを常に行わなければならないため、パフォーマンスの低下を招きます。

さらに、*num_poolagents* の値が小さすぎた場合には、ある 1 つのアプリケーションが関連サブエージェントによってプールを満杯にしてしまう場合があります。他のアプリケーションが新しいサブエージェントを要求したときに、関連エージェント・プール内にサブエージェントがないとすると、そのアプリケーションは、他のアプリケーションのエージェント・プールからサブエージェントを「盗み」ます。この状況はかなりコストが高くつくので、パフォーマンス低下を招きます。

- 上記の状況は、任意の時点で多数のエージェントをアクティブにする場合のリソース・コストと比較して、どちらのコストが大きいかを検討する必要があります。

  たとえば、*num_poolagents* の値が大きすぎる場合には、関連するサブエージェントは、長い間未使用のままプール内に置かれる可能性があります。これらのサブエージェントもデータベース・マネージャー・リソースを使用するので、その分のリソースは他のタスクからは使用できなくなります。

データベース・マネージャーがプロセス (すなわちスレッド) として実行する非同期アクティビティーは、データベース・エージェント以外にもあります。たとえば、次のようなアクティビティーです。

- データベース入出力サーバー (または入出力事前取り出し) (253ページの『バッファー・プールへのデータの事前取り出し』を参照)
- データベース非同期ページ・クリーナー (245ページの『データベース・バッファー・プールの管理』を参照)
- データベース・ロガー
- データベース・デッドロック検出機能
- イベント・モニター
- 通信および IPC listener
- 表スペース・コンテナー再平衡機能

各種の DB2 プロセスの指定方法の詳細については、*問題判別の手引き* を参照してください。

データベース・システム・モニターの使用法

DB2 データベース・マネージャーは、 DB2 データベース・マネージャーの操作、パフォーマンス、および DB2 データベース・マネージャーを使用するアプリケーションに関するデータの保守を行います。こういったデータは、データベース・マネージャーの実行時に保守され、そこから重要なパフォーマンス情報やトラブルシューティング情報を得ることができます。たとえば、得ることができる情報には以下のようなものがあります。

- データベースに接続しているアプリケーションの数、それらの状況、および各アプリケーションが実行している SQL ステートメント。
- データベース・マネージャーおよびデータベースの構成がどのくらい適切であるかを示して、それらの構成をチューニングする際に役立つ情報。
- 指定されたデータベースでデッドロックが生じた場合、関係していたアプリケーションはどれか、および競合していたロックはどれか。
- あるアプリケーションまたはデータベースが保持しているロックのリスト。アプリケーションがあるロックを待機しているために先に進めないという場合には、そのロックに関する追加情報 (どのアプリケーションがロックを保持しているのかなど) も示される。

これらのデータの中には収集すると DB2 の操作にオーバーヘッドをかけるものがあるので、どの情報を収集するのかを制御する**モニター・スイッチ**が用意されています。 明示的にモニター・スイッチを設定するには、UPDATE MONITOR SWITCHES コマンドか sqlmon() API を使用します。 (使用には、SYSADM、SYSCTRL、SYSMAINT のいずれかの権限が必要です。)

データベース・マネージャーが保守するデータにアクセスするには、次の 2 種類の方法があります。

- **スナップショットをとる**

  スナップショットをとるには 3 つの方法があります。コマンド行から GET SNAPSHOT コマンドを使用する方法か、 OS/2 または Windows ベースのオペレーティング・システムのコントロール・センターのグラフィカル・インターフェースを使用する方法か、あるいは sqlmonss() API 呼び出しを使用して独自のアプリケーションを作成する方法です。

  コントロール・センター (DB2 フォルダーから、または db2cc コマンドを用いると使用することができる) は、決まった間隔でスナップショットをとることによって、モニター・データのサンプルを収集するパフォーマンス・モニター・ツールを提供します。このグラフィカル・インターフェースは、スナップショット・データをグラフまたはテキストのいずれかの方法で表示するもので、表示形式には詳細と要約の両方があります。また、データベース・モニターによって戻されるデータ要素を使用して、パフォーマンス変数を定義することもできます。

  コントロール・センターのスナップショット・モニター・ツールを使用すると、パフォーマンス変数にしきい値を指定することにより、例外条件を定義することもできます。つまり、しきい値に達したときにどの処置を実行するのかについて、事前に定義しておくことができるようになっています。事前定義できる処置には、ウィンドウまたは音声アラームによる通知、およびスクリプトまたはプログラムの実行 (あるいは、その両方) があります。

  コントロール・センターからスナップショットをとっている場合、該当オブジェクト、あるいはその子オブジェクトのいずれかに対するスナップショット・モニターを実行している間、データベース・オブジェクト (インスタンスまたはデータベースな

ど) を更新、変更、または削除する処置は実行できません。 (また、区分データベース・システムをモニターする場合、区分データベース・オブジェクトの視点を最新表示することはできません。) たとえば、データベース A のインスタンスを除去したいのであれば、このデータベースをモニターすることはできません。しかし、インスタンスだけをモニターしているのであれば、データベース A を変更することができます。

インスタンス (その子オブジェクトを含む) のモニターすべてを停止するには、インスタンスのポップアップ・メニューから**全モニターの停止**を選択します。この選択を行うと、パフォーマンス・モニターが保持するロックはすべて解除されるので、インスタンスのモニターは必ず停止してください。

- **イベント・モニターの使用法**

  イベント・モニターは、トランザクションの終了、ステートメントの終了、デッドロックの検出などの特定のイベントが起きた後に、システム・モニター情報を収集します。この情報は、ファイルまたは名前付きパイプに書き込むことができます。

  イベント・モニターを使用するには、以下のことを行ってください。

  1. コントロール・センターまたは SQL ステートメントの CREATE EVENT MONITOR を使用して、イベント・モニターの定義を作成します。このステートメントを使用すると、定義はデータベース・システム・カタログに保管されます。
  2. コントロール・センター、または次の SQL ステートメントを使用して、イベント・モニターを活動化します。

     ```
     SET EVENT MONITOR evname STATE 1
     ```

     名前付きパイプに書き込む場合には、イベント・モニターを活動化する前に、名前付きパイプからの読み取りを行うアプリケーションを開始してください。これを行うには、ユーザー独自のアプリケーションを作成するか、または **db2evmon** を使用することができます。イベント・モニターが活動化されてイベントのパイプへの書き込みが開始された後に **db2evmon** を使用すると、生成されるイベントを読み取って、それらを標準出力へ書き出します。
  3. トレースを読み取ります。ファイル・イベント・モニターを使用している場合には、以下のどちらかの方法で、モニターが作成する 2 進トレースを見ることができます。
     - **db2evmon** ツールを使用して、トレースを標準出力に形式設定する方法。
     - (Windows ベースのオペレーティング・システム、または OS/2 システムの) コントロール・センターの**「イベント解析プログラム (Event Analyzer)」**アイコンをクリックして、グラフィカル・インターフェースを使用し、トレースの表示、キーワードの探索、および不必要な情報を除いた表示を行う方法。

       注: モニターのデータベース・システムが、コントロール・センターと同じマシンで実行していない場合、トレースを表示する前にコントロール・センターと同じマシンに、イベント・モニター・ファイルをコピーする必要が

あります。別の方法としては、両方のマシンにアクセス可能なファイル共用システムにファイルを配置します。

システム・データベース・モニターとイベント・モニターの詳細については、 システム・モニター *手引きおよび解説書* を参照してください。

メモリーの拡張

使用しているマシンによっては、最大仮想アドレス可能メモリー量を超える実アドレス可能メモリーを持っている場合があります (たとえば、ほとんどのプラットフォームでは通常、仮想アドレス可能メモリーは 2 GB から 4 GB の間です)。仮想アドレス可能メモリーを超える分の実メモリーについては、*拡張記憶キャッシュ* として構成することができます。 拡張記憶キャッシュは、定義されたバッファー・プールのいずれからも使用することができるので、データベース・マネージャーのパフォーマンスを向上させることができます。拡張記憶キャッシュは、メモリー・セグメントとして定義されます。

実アドレス可能メモリーの一部を拡張記憶域キャッシュとして使用しようとする場合には、 jfs キャッシュまたはプロセス専用アドレス・スペースなどの他の目的ではこのメモリーをマシン上で使用できなくなることに注意してください。拡張記憶キャッシュに追加の実アドレス可能メモリーを割り当てると、システム・ページングが増加する可能性があります。

DB2 は、バッファー・プールを用いて、マシン内のアドレス可能メモリーを利用します (245ページの『データベース・バッファー・プールの管理』を参照してください)。拡張記憶キャッシュは、バッファー・プールにより、 2 次レベル・キャッシングとして使用されます (バッファー・プールを使用して行われるのが 1 次レベル・キャッシングです)。理想的には、バッファー・プールにはアクセス頻度が高いデータが保持され、拡張記憶キャッシュにはアクセスはされるがその頻度はあまり高くないデータが保持されます。

DB2_AWE レジストリー変数を使用して Windows 2000 Address Windowing Extensions (AWE) のバッファー・プールを割り振るときは、拡張記憶域キャッシュは使用できません。

以下に示すデータベース構成パラメーターは、拡張記憶で使用可能なメモリーの量とサイズに影響を与えます。

- *num_estore_segs* は、拡張記憶メモリー・セグメントの数を定義します。この構成パラメーターのデフォルトはゼロであり、これは拡張記憶キャッシュは持たないことを指定するものです。 (398ページの『拡張記憶域メモリー・セグメントの数 (num_estore_segs)』を参照してください。)

- *estore_seg_sz* は、各拡張メモリー・セグメントのサイズを定義します。このサイズは、拡張記憶キャッシュが使用されるプラットフォームによって制限されます。 (398 ページの『拡張記憶域メモリー・セグメント・サイズ (estore_seg_sz)』を参照してください。)

拡張記憶キャッシュはバッファー・プールの延長なので、常に、1 つまたは複数の特定のバッファー・プールに関連付けておく必要があります。したがって、キャッシュを作成した場合には、どのバッファー・プールがキャッシュを利用することができるかを宣言する必要があります。 CREATE BUFFERPOOL ステートメントと ALTER BUFFERPOOL ステートメントには、キャッシュの使用について制御する属性 NOT EXTENDED STORAGE と EXTENDED STORAGE があります。デフォルトを使用すると、IBMDEFAULTBP および新しく作成されるバッファー・プールは、いずれも拡張記憶を使用しません。

注: 異なるページ・サイズで定義されたバッファー・プールを使用している場合があります。そのバッファー・プールの一部またはすべては、拡張記憶域を使用するように定義できます。拡張記憶域サポートで使用されるページ・サイズは、定義される中で最大です。

データベース・マネージャーは、拡張記憶キャッシュに常駐しているデータを直接操作することはできません。ただし、ディスク装置からバッファー・プールに転送するよりもはるかに速い速度で、データを拡張記憶キャッシュからバッファー・プールに転送することが可能です。

拡張記憶キャッシュ内のページのデータが 1 行必要になった場合は、そのページ全体が対応するバッファー・プールに読み込まれます。

バッファー・プールおよび関連する拡張記憶キャッシュ (定義されている場合) は、データベースが活動化されたとき、あるいはデータベースに最初に接続されたときに割り振られます。

第9章 管理プログラムの使用法

管理プログラムを使用すると、データベースに対して実行されるアプリケーションの動作をモニターしたり、変更したりすることができます。

管理プログラムは、2 つの部分から構成されています。
- フロントエンド・ユーティリティー
- デーモン

管理プログラムを開始するには、管理プログラム・フロントエンド・ユーティリティーから開始コマンドを出して、管理プログラム・デーモンを開始します。デフォルトには、デーモンは区分データベースの各区画すべてで開始されますが、フロントエンド・ユーティリティーを使用して、単一のデーモンを特定の区画で開始し、その区画にあるデータベース区画に対する活動だけをモニターすることも可能です。また、デーモンは、単一区画のデータベース上の活動をモニターすることもできます。詳細については、282ページの『管理プログラムの開始と停止』を参照してください。

各管理プログラム・デーモンはそれぞれ、 1 つのデータベースに対して実行されるアプリケーションに関する統計を収集します。収集後、デーモンはそれらの統計を、その特定のデータベースに適用するものとして管理プログラム構成ファイルに指定した規則と照らして検査します。 (詳細については、285ページの『管理プログラム構成ファイルの作成』を参照してください。) 検査後、管理プログラムはそれらの規則に従って処置を行います。たとえば、規則によると、アプリケーションはリソースを過剰に使用している旨を示していたとします。この場合には、管理プログラムは、管理プログラム構成ファイルに指定した命令に従って、アプリケーションの優先順位を変更するか、そのアプリケーションのデータベース使用を強制的に止めさせます。

規則に対応する処置がアプリケーションの優先順位の変更であった場合は、管理プログラムは、管理プログラムがリソース違反を検出したデータベース区画に対するエージェントの優先順位を変更します。規則に対応する処置がアプリケーションの強制停止であった場合は、アプリケーションは、リソース違反を検出した管理プログラムがそのアプリケーションの調整プログラム・ノードや区分データベース環境で実行されている場合であっても、強制停止されます。

管理プログラムは、管理プログラムが行った処置のログもすべて記録します。ログ・ファイルを照会すると、管理プログラムが行った処置を検討することができます。詳細については、293ページの『管理プログラム・ログ・ファイル』と 294ページの『管理プログラム・ログ・ファイルの照会』を参照してください。

管理プログラムの開始と停止

db2gov 管理プログラム・フロントエンド・ユーティリティーを使用すると、 (全データベース区画または単一のデータベース区画のいずれかに対して) 管理プログラムの開始および停止を行うことができます。このユーティリティーを使用するには、SYSADM または SYSCTRL 権限が必須となります。

db2gov の構文は、次のとおりです。

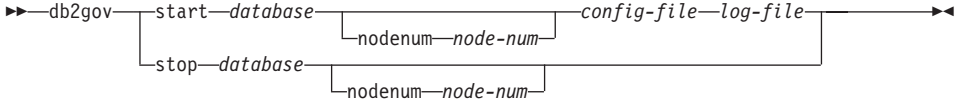


図 28. db2gov の構文

パラメーターは、以下のとおりです。

start **database**

管理プログラム・デーモンを開始して、指定されたデータベースのモニターを行います。 *database* には、データベース名かデータベース別名のいずれかを指定することができます。

指定するデータベース名は、管理プログラム構成ファイルに指定した名前と同じにする必要があります。管理プログラムは、それらの 2 つの名前を検査して、正しい構成ファイルが使用されているかどうかの確認を行います。フロントエンド・ユーティリティーがある別名を用いて開始されたときに、管理プログラム構成ファイルには異なる別名が含まれていたという場合には、管理プログラムはそれらの名前が同じデータベースの別名であるか否かを判別できないため、エラーが報告されます。

区分データベース環境を使用している場合は、全区画に対して管理プログラムを開始するときに、フロントエンド・ユーティリティーはまず最初に、構成ファイルにエラーが含まれていないかどうかを検査します。検査後、フロントエンド・ユーティリティーはノード構成ファイルを読み取り、コマンドを各データベース区画に送って、開始オプションを用いて各データベース区画に対して管理プログラム・フロントエンド・ユーティリティーを開始します (次に、管理プログラム・フロントエンド・ユーティリティーは各データベース区画でデーモンを開始します)。

注: 管理プログラムがデータベース・レベルでモニターを行うため、モニターされている各データベースごとに 1 つのデーモンが実行します。 (区分データベース環境では、各データベース区画ごとに 1 つのデーモンが実行します。) 管理プログラムが複数のデータベースに対して実行している場合、そのデータベース・サーバーで実行しているデーモンも複数です。

nodenum **node-num**
管理プログラム・デーモンを開始するデータベース区画を指定します。この番号は、ノード構成ファイルに指定した番号と同じものです。

管理プログラムを単一のデータベース区画に対して開始する場合には、フロントエンド・ユーティリティーはデーモンを作成して、管理プログラム構成ファイルの妥当性検査を行います。管理プログラム・デーモンは、その区画で別のデーモンがすでに実行されていないことを確認します。

config-file
データベースのモニター時に使用する構成ファイルを指定します。

構成ファイルのデフォルトの場所は、sqllib ディレクトリーとなります。指定されたファイルがそこにない場合、フロントエンドは指定された名前がファイルの正式名であるということを前提にします。

log-file 管理プログラムがログ・レコードを書き込むファイルの基底名を指定します。ログ・ファイルは、 sqllib ディレクトリーの log サブディレクトリーに保管されます。 (Windows NT の場合、 log サブディレクトリーはインスタンス・ディレクトリーの下にあります。) ログ・ファイル名の後には、管理プログラムが実行されているデータベース区画の番号が自動的に付加されます (たとえば、 mylog.0、mylog.1、mylog.2 など)。

stop **database**
指定されたデータベースをモニターしている管理プログラム・デーモンを停止します。

区分データベース環境を使用している場合には、フロントエンド・ユーティリティーは、ノード構成ファイルを読み取ることによって全データベース区画に対する管理プログラムを停止して、その後でコマンドを各データベース区画に送り、停止パラメーターを用いて管理プログラム・フロントエンド・ユーティリティーを呼び出します。その結果、各データベース区画でデーモンが停止します。

nodenum **node-num**
管理プログラム・デーモンを停止するデータベース区画を指定します。この番号は、ノード構成ファイルに指定した番号と同じものです。

フロントエンド・ユーティリティーが単一のデータベース区画の管理プログラム・デーモンを停止する場合は、ユーティリティーは、sqllib ディレクトリーの tmp サブディレクトリー内のファイルの作成、移動、あるいは削除を行うことによって、そのデータベース区画上のデーモンとコミュニケーションします。これらのファイルを削除または変更しようとしないでください。

管理プログラム・デーモン

(db2gov フロントエンド・ユーティリティーによって開始されたか、ウェイクアップされたかのいずれかの方法によって) 管理プログラム・デーモンが開始されると、デーモンはループで実行されます。デーモンが最初に行うタスクは、管理プログラム構成ファイルが変更されたか否か、あるいはファイルの読み取りがまだ行われていないか否かの検査です。いずれかの条件が真である場合には、デーモンはファイル内の規則を読み取ります。これによって、管理プログラム・デーモンの実行中にその動作を変更することができるようになります。

この後、管理プログラム・デーモンはスナップショット要求を出して、そのデータベース上で作動している各アプリケーションおよびエージェントごとに統計を取得します。

注: 一部のプラットフォームでは、 CPU 統計を DB2 モニターから取得することはできません。その場合には、会計規則や CPU 制限も使用できません。

統計取得後、管理プログラムは、統計を各アプリケーションごとに、管理プログラム構成ファイル内の規則に照らして検査します。規則がアプリケーションに適用される場合には、管理プログラムは、規則で指定された処置に従って、「アプリケーションを強制停止する」、「アプリケーションの優先順位を変更して間接的にそのデータベース区画上でそのアプリケーションのために処理を行っているエージェントとサブエージェントの両方の全エージェントの優先順位を変更する」、あるいは「アプリケーションのスケジュールを変更して間接的にそのアプリケーション上で処理を行っているエージェントの優先順位を変更する」のいずれかを行うことができます。管理プログラムは、管理プログラムが行う処置についてのレコードをすべて、ログ・ファイルに書き込みます。

注: 管理プログラムは、 *agentpri* データベース・マネージャー構成パラメーターがシステム・デフォルト以外である場合には、エージェントの優先順位を調整する代替手段として使用することはできません。 (この注は、Windows NT プラットフォームには適用されません。)

管理プログラムは、アプリケーションすべての検査を終えると、構成ファイル内に指定された時間間隔の間スリープします。その時間が経過すると、管理プログラムはウェイクアップして、実行ループを再度実行します。

管理プログラムがエラーまたはストップ信号を検出した場合、終結処理を行ってから終了します。終結処理によって、 (優先順位が設定してあるアプリケーションのリストを使用して) アプリケーション・エージェントの優先順位はすべてリセットされます。続いて、すでにアプリケーション上で処理を行っていないエージェントの優先順位もすべてリセットされます。こうすることにより、管理プログラムの終了後には、デフォルトでない優先順位で実行されているエージェントがないようにします。エラーが生じると、管理プログラムが異常終了した旨を示すメッセージが db2diag.log ファイルに書き込まれます。

注: 管理プログラム・デーモンはデータベース・アプリケーションではないので、データベースとの接続の維持は行いません。 (ただし、インスタンス接続機能は備わっています。) 管理プログラム・デーモンは、スナップショット要求を出すことができるので、データベース・マネージャーが終了した時点を検出することができます。

管理プログラム構成ファイルの作成

管理プログラムを開始するときには、データベースに対して実行されるアプリケーションを管理するのに使用する規則を含んだ構成ファイルの名前を指定します。管理プログラムは、これらの規則に基づいて処置を行います。

データベースの管理に関する要件が変更になった場合には、管理プログラムを停止しないで構成ファイルを編集することができます。各管理プログラム・デーモンは、構成ファイルが変更されたことを検出すると、ファイルを再度読み取ります。

各区画上の管理プログラム・デーモンは同一の構成ファイルを読み取りできなければならないため、構成ファイルは、すべてのデータベース区画に取り付けられるディレクトリーに作成する必要があります。

構成ファイルは、規則と注釈から構成されています。大部分の項目は、英大文字、英小文字、または英大文字小文字混合文字で指定することができます。例外は `applname` で、これは大文字小文字の区別をして指定する必要があります。

注釈は、中括弧 { } に入れて区切る必要があります。規則には、以下のものが含まれます。

- 規則が適用されるデータベース。
- アプリケーションのチェックを行うためにウェイクアップするまでに、管理プログラムがスリープする時間の長さ。
- アプリケーションの管理方法を指定する規則。これらの規則は、規則文節と呼ばれるより小さなコンポーネントから成ります。

ファイル内の各規則の後ろには、セミコロン (;) を置く必要があります。

以下の規則によって、モニターするデータベースと、デーモンが活動 (284ページの『管理プログラム・デーモン』で説明されています) のループを一回り処理した後ウェイクアップするまでの時間間隔が指定されます。これらの規則はそれぞれ、ファイルに 1 回しか指定できません。

dbname
モニター対象となるデータベースの名前または別名。

account *nnn*
各接続ごとの CPU 使用率統計を含む会計レコードが、指定された数の分ごとに書き出されます。

注: このオプションは、Windows NT では使用できません。

短い接続セッションが全体的にアカウント・インターバル内で発生する場合、ログ・レコードは作成されません。ログ・レコードが作成される場合、そこには前の接続に関するログ・レコード以来の CPU 使用量を反映する CPU 統計が含まれます。管理プログラムが停止してから再始動された場合、 CPU 使用量は 2 つのログ・レコードで反映される場合があります。これらはログ・レコードのアプリケーション ID を介して識別できます。管理プログラムのログ・ファイルについての詳細は、 293ページの『管理プログラム・ログ・ファイル』を参照してください。

interval

デーモンがウェイクアップする時間間隔 (秒単位)。時間間隔が指定されない場合には、120 秒の時間間隔が使用されます。

以下にあげる規則文節を組み合わせて、1 つの規則を作ります (すなわち、完全規則とはセミコロンで終わるものであり、各独立の文節とは異なります)。文節が指定するのは、規則が適用される時間帯、使用できるリソースの制限、および任意指定ですが、特定のユーザーまたはアプリケーション、規則に指定した制限を超えた場合に管理プログラムが行う処置です。文節は 1 つの規則には 1 回ずつしか指定できませんが、複数の規則に指定することができます。文節は、以下に説明する順序で指定する必要があります。以下の説明では、[ ] は任意指定の文節を示しています。

[desc] 規則に関するテキスト記述を指定します。記述は、単一引用符か二重引用符のいずれかで囲む必要があります。

[time] 規則が適用される時間帯を指定します。

時間帯は、`time hh:mm hh:mm` (たとえば、 `time 8:00 18:00`) という形式で指定する必要があります。この文節が指定されない場合は、規則は全日 (24 時間) 有効になります。

[authid]

アプリケーションを実行する許可 ID (authid) を 1 つまたは複数指定します。複数の authid を指定する場合は、たとえば `authid gene, michael, james` のように、コンマ (,) で区切る必要があります。この文節が規則になかった場合には、規則はすべての authid に適用されます。

[applname]

データベースへの接続を行う実行可能アプリケーション (または、オブジェクト・ファイル) の名前を指定します。

複数のアプリケーション名を指定する場合は、たとえば `applname db2bp, batch, geneprog` のように、コンマ (,) で区切る必要があります。この文節が規則になかった場合には、規則はすべてのアプリケーション名に適用されます。

注:

1. アプリケーション名は、大文字小文字を区別する必要があります。
2. データベース・マネージャーは、すべてのアプリケーション名を 20 文字で切り捨てます。管理したいアプリケーションがアプリケーション名の最初の 20 文字で固有に識別可能であることを確認しておく必要があります。確認を怠ると、望まないアプリケーションを管理対象としてしまう可能性があります。

   管理プログラム構成ファイルに指定されたアプリケーション名は 20 文字で切り捨てられて、構成ファイルの内部表記に一致させられます。

setlimit

管理プログラムが検査する制限を 1 つまたは複数指定します。制限値は、-1 か、さもなければ 0 より大きい値にしなければなりません (たとえば、 `cpu -1 locks 1000 rowssel 10000`)。制限 (cpu、 locks、 rowsread、 uowtime) は、最低でも 1 つは指定する必要があり、規則によって指定されていない制限は、その特定の規則によって制限されません。管理プログラムが検査できるのは、以下に示す制限です。

cpu *nnn*

アプリケーションが使用可能な CPU の秒数を指定します。 `-1` を指定すると、管理プログラムはアプリケーションの CPU 使用の制限は行いません。

注: このオプションは、Windows NT では使用できません。

locks *nnn*

アプリケーションが保持できるロック数を指定します。 `-1` を指定すると、管理プログラムはアプリケーションが保持するロック数の制限は行いません。

rowssel *nnn*

アプリケーションに戻される行数を指定します。この値は、調整プログラム・ノードでは非ゼロとなります。 `-1` を指定すると、管理プログラムは選択できる行数の制限は行いません。

uowtime *nnn*

作業単位 (UOW) が最初に活動状態になった時刻から経過可能な秒数を指定します。 `-1` を指定すると、経過時間は制限されません。

注: `sqlmon` (データベース・システム・モニター スイッチ) API を使用して作業単位スイッチを非活動化した場合には、管理プログラムが作業単位経過時間に基づいてアプリケーションを管理する機能に影響を与えます。管理プログラムはモニターを使って、システムについての情報を収集します。データベース・マネージャー構成ファイルでスイッチをオフにすると、インスタンス全体がオ

フになり、管理プログラムはそれ以上この情報を受け取りません。詳しくは、478ページの『データベース・システム・モニター・パラメーター』を参照してください。

idle *nnn*
接続において、指定された処置が行われる前に許されるアイドル状態の秒数を指定します。 -1 を指定すると、接続のアイドル時間は制限されません。

rowsread *nnn*
アプリケーションが選択できる行数を指定します。 -1 を指定すると、アプリケーションが選択できる行数は制限されません。

注: この制限値は、rowssel と同じものではありません。異なるのは、rowsread が結果セットを戻すために読み取りする必要のあった行数のカウントである点です。読み取りされた行数にはエンジンによるカタログ表の読み取りが含まれるので、索引使用時には行数が少なくなる可能性があります。

[action]
指定された制限の 1 つまたは複数を超えた場合に取る処置を指定します。次のように処置を指定することができます。

注: 制限を超えたときに action 文節が指定されていなかった場合には、管理プログラムは、アプリケーションのために処理を行っているエージェントの優先順位を 10 倍低くします。

priority *nnn*
アプリケーションのために処理を行っているエージェントの優先順位の変更を指定します。有効な値は -20 ～ +20 です。

このパラメーターを有効に使用するには、以下に注意してください。
- UNIX ベースのプラットフォーム上では、 *agentpri* データベース・マネージャー・パラメーターをデフォルト値に設定する必要があります。デフォルト値にしないと、このパラメーターが priority 文節を指定変更してしまいます。
- OS/2 上および Windows NT プラットフォームでは、 *agentpri* データベース・マネージャー・パラメーターと *priority* の処置とを一緒に使用することができます。

force アプリケーションにサービスを提供しているエージェントの強制停止を指定します。 (調整エージェントを終了する場合は、FORCE APPLICATION を出します。)

schedule [class]
スケジューリングによって、すべてのアプリケーションにおける公平

性を確保しながら同時に平均応答時間を最小化するという目的に向けて、アプリケーション上で処理を行っているエージェントの優先順位の調整が行われます。

管理プログラムは、DB2 内部照会コンパイラーからの照会コスト見積もりを使用して、アプリケーション上で処理を行うエージェントの優先順位を設定することによって、スケジューリングを強制的に行います。クラス・オプションが指定された場合は、規則によって選択されたアプリケーションはすべて、それらの間同士でのみスケジュールされます。このオプションが指定されなかった場合は、管理プログラムは 1 つまたは複数のクラスを使用しますが、スケジューリングは各クラス内で行います。

各クラス内では、アプリケーションの優先順位付けは以下に基づいて決められます。

- クラス内のアプリケーションによって保持されるロック数。 (ロッキングによって他の多くのアプリケーションを待機させているアプリケーションには、高い優先順位が与えられます。)
- アプリケーションの年令 (経過時間)。 (システム内に長時間存在しているアプリケーションには、高い優先順位が与えられます。)
- アプリケーションの残り実行時間の見積もり。 (終了が近いアプリケーションには、高い優先順位が与えられます。)

どのスケジュールの対象にもなっていないアプリケーションは、最高の権限で実行されます。

注: sqlmon (データベース・システム・モニター スイッチ) API を使用してステートメント・スイッチを非活動化した場合には、管理プログラムがステートメント経過時間に基づいてアプリケーションを管理する機能に影響を与えます。管理プログラムはモニターを使って、システムについての情報を収集します。データベース・マネージャー構成ファイルでスイッチをオフにすると、インスタンス全体がオフになり、管理プログラムはそれ以上この情報を受け取りません。

スケジュール処置には次のことが含まれます。

- それぞれ異なるグループのアプリケーションが、すべてのアプリケーションに平均に時間を分割することなく確実に時間を入手するようにします。

  たとえば、12 のアプリケーション (短いアプリケーション 3 つ、中程度 5 つ、長いアプリケーション 6 つ) が同時に実行している場合、これらは CPU を分割しているので、応答時間があまりないかもしれません。データベース管理者は、中程度の長さのアプリケ

ーションと、長いアプリケーションの 2 つのグループを設定できます。優先順位を使用して、管理プログラムはすべてのアプリケーションの実行を許可し、大部分を占める 3 つの中程度のアプリケーションと 3 つの長いアプリケーションを、同時に確実に実行します。これを行うために、管理プログラム構成ファイルには、中程度のアプリケーションに 1 つの規則、長いアプリケーションに別の規則が入っています。以下に、この点を例証する管理プログラム構成ファイルの一部を示します。

```
desc "Group together medium applications in 1 schedule class"
applname medq1, medq2, medq3, medq4, medq5
setlimit cpu -1
action schedule class;

desc "Group together long applications in 1 schedule class"
applname longq1, longq2, longq3, longq4, longq5, longq6
setlimit cpu -1
action schedule class;
```

- 複数のユーザー・グループのそれぞれ (たとえば、組織上の部門) が確実に等しい優先度を獲得できるようにします。

  あるグループが多数のアプリケーションを実行している場合でも、管理者は他のグループが自分のアプリケーション用に適度な応答時間を獲得できるようにすることができます。たとえば、3 つの部門 (金融、在庫、企画) が関係している場合、すべての金融ユーザーを 1 つのグループに、すべての在庫ユーザーを 2 つ目のグループに、すべての企画ユーザーを 3 つ目のグループに入れることができます。処理能力は 3 つの部門の間でより平均に、またはその逆に分割されます。以下に、この点を例証する管理プログラム構成ファイルの一部を示します。

  ```
  desc "Group together Finance department users"
  authid tom, dick, harry, mo, larry, curly
  setlimit cpu -1
  action schedule class;

  desc "Group together Inventory department users"
  authid pat, chris, jack, jill
  setlimit cpu -1
  action schedule class;

  desc "Group together Planning department users"
  authid tara, dianne, henrietta, maureen, linda, candy
  setlimit cpu -1
  action schedule class;
  ```

- 管理プログラムにすべてのアプリケーションをスケジュールさせます。

  クラス・オプションが処置に含まれていない場合、管理プログラムはスケジュール処理の下に落とされるアプリケーションの数に基づ

いた、独自のクラスを作成し、アプリケーションが実行している照会に DB2 照会コンパイラーのコスト見積もりに基づいてアプリケーションを別のクラスに入れます。管理者は、選択されるアプリケーションを限定しないことによって、すべてのアプリケーションをスケジュールするように選択できます。つまり、*applname* または *authid* 文節は提供されず、 *setlimit* 文節も制限を課しません。

注: 制限を超えたときに action 文節が指定されていなかった場合には、管理プログラムは、アプリケーションのために処理を行っているエージェントの優先順位を低くします。

複数の規則がアプリケーションに適用される場合、そのすべての規則が適用されます。設定されている規則と制限に応じて、最初に見つかった規則制限に関連付けられたアクションが最初に適用されるアクションとなります。例外なのは、規則の中の文節に -1 が指定された場合です。この場合には、後続する規則の文節の値は、それより前に*同じ*文節に指定された値のみを指定変更します。このとき、前に置かれた規則の他の文節は、有効なままになります。たとえば、ある規則は、経過時間が 1 時間を超えるか、選択された行数が 100 000 行を超えるかした場合 (すなわち、 `rowssel 100000 uowtime 3600`) には、そのアプリケーションの優先順位を低くするよう指示しているとします。また、後続する規則では、同じアプリケーションに無制限の経過時間を許している (すなわち、 `uowtime -1`) とします。この場合、アプリケーションが 1 時間を超えて実行されたとしても、優先順位は変更されません (`uowtime -1` が `uowtime 3600` を指定変更したからです) が、選択された行数が 100 000 行を超えたときには、優先順位は低くされます (`rowssel 100000` が有効なままだからです)。

292ページの図29 に、構成ファイルの例を示します。

```
{ Wake up once a second, the database name is ibmsampl
 do accounting every 30 minutes.  }
interval 1; dbname ibmsampl; account 30;

desc "CPU restrictions apply 24 hours a day to everyone"
setlimit cpu 600 rowssel 1000000 rowsread 5000000;

desc "Allow no UOW to run for more than an hour"
setlimit uowtime 3600 action force;

desc 'Slow down a subset of applications'
applname jointA, jointB, jointC, quryA
setlimit cpu 3 locks 1000 rowssel 500 rowsread 5000;

desc "Have governor prioritize these 6 long apps in 1 class"
applname longq1, longq2, longq3, longq4, longq5, longq6
setlimit cpu -1
action schedule class;

desc "Schedule all applications run by the planning dept"
authid planid1, planid2, planid3, planid4, planid5
setlimit cpu -1
action schedule;

desc "Schedule all CPU hogs in one class which will control consumption"
setlimit cpu 3600
action schedule class;

desc "Slow down the use of db2 CLP by the novice user"
authid novice
applname db2bp.exe
setlimit cpu 5 locks 100 rowssel 250;

desc "During day hours do not let anyone run for more than 10 seconds"
time 8:30 17:00 setlimit cpu 10 action force;

desc "Allow users doing performance tuning to run some of
     their applications during lunch hour"
time 12:00 13:00 authid ming, geoffrey, john, bill
applname tpcc1, tpcc2, tpcA, tpvG setlimit cpu 600 rowssel 120000 action force;

desc "Some people should not be limited -- database administrator
  and a few others.  As this is the last specification in the
  file, it will override what came before."
authid gene, hershel, janet setlimit cpu -1 locks -1 rowssel -1 uowtime -1;

desc "Increase the priority of an important application so it always
      completes quickly"
applname V1app setlimit cpu 1 locks 1 rowssel 1 action priority -20;
```

図 29. 管理プログラム構成ファイルの例

管理プログラム・ログ・ファイル

管理プログラム・デーモンが、アプリケーションの強制停止、管理プログラム構成ファイルの読み取り、アプリケーションの優先順位の変更、エラーや警告の検出、デーモンの開始または終了のいずれかを行うときには、レコードをログ・ファイルに書き込みます。各管理プログラム・デーモンごとに別々のログ・ファイルが使用されます。こうすることで、多くの管理プログラム・デーモンが同一のファイルに同時に書き込みを行うことによって起こるファイル・ロッキングのボトルネックを防ぐことができます。
db2govlg ユーティリティーを使用すると、複数のログ・ファイルを一緒にマージして、マージ結果を照会することができます。このユーティリティーは、294ページの『管理プログラム・ログ・ファイルの照会』で説明されています。

ログ・ファイルは、 sqllib ディレクトリーの log サブディレクトリーに保管されます。 (Windows NT の場合、 log サブディレクトリーはインスタンス・ディレクトリーの下にあります。) db2gov コマンドを出すときには、ログ・ファイルの基底名を指定します。管理対象とする各データベースの各ノード用のログ・ファイルは 1 つなので、ログ・ファイル名にはデータベース名を含めるようにしてください。区分データベース環境では、管理プログラムが実行されているデータベース区画のノード番号が自動的にログ・ファイル名の後ろに付加されることにより、各管理プログラムごとにファイル名が固有になるようになっています。

ログ・ファイルの各レコードの形式は、次のとおりです。

```
Date Time NodeNum RecType Message
```

Date と *Time* のフィールドは yyyy-mm-dd hh.mm.ss 形式なので、このフィールドで分類を行うことによって各データベース区画ごとのログ・ファイルをマージすることができます。

NodeNum フィールドは、管理プログラムが実行されているデータベース区画の番号を示します。

RecType フィールドには、ログに書き込まれるログ・レコードのタイプによって異なる値が入ります。フィールドに入れることができる値は、以下のとおりです。

- 管理プログラムが開始されたことを示す、START。
- アプリケーションが強制されたことを示す、FORCE。
- アプリケーションの優先順位が変更されたことを示す、PRIORITY。
- エラーを示す、ERROR。
- 警告を示す、WARNING。
- 管理プログラムが構成ファイルの読み取りを行ったことを示す、READCFG。
- 管理プログラムが停止されたことを示す、STOP。
- アプリケーションの会計統計を示す、ACCOUNT。

  フィールドは、以下のとおりです。

  – authid

- appl_id
- written_usr_cpu
- written_sys_cpu
- appl_con_time

• エージェントの優先順位に変更が生じたことを示す、SCHEDULE。

RecType フィールドには標準値が書き込まれるので、ログ・ファイルを照会してさまざまなタイプの処置を見ることができます。それに対し *Message* フィールドには、*RecType* フィールドの値によって変わるその他の非標準情報が入ります。たとえば、FORCE レコードまたは NICE レコードには *Message* フィールドのアプリケーション情報が示され、 ERROR レコードにはエラー・メッセージが入れられます。

ログ・ファイルの例を、次に示します。

```
1995-12-11 14.54.52    0 START      Database = TQTEST
1995-12-11 14.54.52    0 READCFG    Config = /u/db2instance/sqllib/tqtest.cfg
1995-12-11 14.54.53    0 ERROR      SQLMON Error: SQLCode = -1032
1995-12-11 14.54.54    0 ERROR      SQLMONSZ Error: SQLCode = -1032
```

管理プログラム・ログ・ファイルの照会

各管理プログラム・デーモンは、それぞれ固有のログ・ファイルに書き込みを行います。 db2govlg ユーティリティーを使用すると、ログ・ファイルの照会を行うことができます。単一区画あるいは全データベース区画のログ・ファイルを、日時で分類してリストすることができます。また *RecType* ログ・フィールドに基づいて、照会することも可能です。 db2govlg の構文は、次のとおりです。

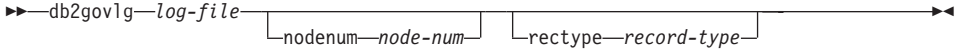


図 30. db2govlg の構文

パラメーターは、以下のとおりです。

log-file 照会したいログ・ファイル (複数も可) の基底名。

nodenum **node-num**
管理プログラムが実行されているデータベース区画のノード番号。

rectype **record-type**
照会したいレコードのタイプ。レコード・タイプは、次のとおりです。
- START
- READCFG
- STOP

- FORCE
- NICE
- ERROR
- WARNING
- ACCOUNT

このユーティリティーを使用するには、許可に関する制約事項はありません。したがって、すべてのユーザーが、管理プログラムがユーザーのアプリケーションに影響を与えていないかどうかを照会することができます。このユーティリティーへのアクセスに制約を加えたい場合には、 `db2govlg` ファイルのグループ許可を変更することができます。

管理プログラムの実行とデータベース・マネージャーのパフォーマンス

管理プログラムは、データベース・マネージャーのスナップショットを要求するので、データベース・マネージャーのパフォーマンスに影響を与えることがあります。管理プログラムが CPU を過剰に使用する場合には、管理プログラムのウェイクアップの時間間隔を大きくすると CPU の使用率を下げることができます。

第10章 プロセッサーの追加による構成のスケーリング

使用するシステムの構成特性が、現在のニーズや計画されたニーズに合っていないことがあります。そのような場合は、構成の容量やパフォーマンスを上げるための処置を検討する必要があります。たとえば、コンテナーを構成に追加すると、データを保管するための容量が増え、またユーティリティー使用時 (たとえば、データのロード時) のパフォーマンスも向上します。容量やパフォーマンスを上げるための別の方法として、対称マルチプロセッサー、または区分データベース環境のいずれかにおいて、メモリーの追加、およびプロセッサーの追加を行うこともできます。

この章では、構成においてプロセッサーの数を増やすことにより、パフォーマンスを向上させることに焦点を合わせています。

次のような場合には、システム構成のスケーリングについて考慮してください。

- 1 台のプロセッサーを単一区画構成で使用しており、しかもそれを最大限まで使用してしまっているため、構成を変更し、次のようにすることにした場合。
  - 新しい環境には、対称マルチプロセッサー (SMP) が最良であると決断した。このような選択を行うのは、おそらくは、複数のプロセッサーの場合に利用できる処理能力を使用したいためです。各プロセッサーがメモリーとストレージのシステム・リソースを共用します。すべてのプロセッサーが 1 つのシステムに収まっているため、システム間の通信回線などについての考慮事項が加わることがなく、また、新しいシステムをサポートするための管理スタッフを追加する必要もおそらくありません。システム間の調整も問題になりません。 DB2 ユニバーサル・データベースはこの環境をサポートします。
  - 新しい環境には、区分データベース環境が最良であると決断した。このような選択を行うのは、おそらくは、最初のプロセッサーとは物理的に独立している複数のプロセッサーで利用できる処理能力を使用したいためです。各プロセッサーはそれ自体のメモリーとストレージのシステム・リソースを持ち、これを他のプロセッサーと共用する必要はありません。先に述べた追加の考慮事項 (通信、スタッフ、タスク調整) が必要になる場合がありますが、複数のシステム間でデータとユーザー・アクセスを平均化する能力など、この選択には利点があります。 DB2 ユニバーサル・データベースはこの環境をサポートします。
- 現在すでに SMP 構成になっていて、 1 つまたは複数のプロセッサーを追加することを計画している場合。この場合には、この種の環境にかかわる考慮事項についてはすでによく理解されているはずです。 1 つまたは複数のプロセッサーを追加することによって、簡単に計算能力が環境に追加されますが、新しい考慮事項が加わることはありません。 DB2 ユニバーサル・データベースはこの環境をサポートします。
- 現在すでに区分データベース構成になっていて、そこに 1 つまたは複数のデータベース区画を追加することを計画している場合。この場合には、この種の環境にかかわ

る考慮事項についてはすでによく理解されているはずです。 1 つまたは複数のデータベース区画を追加することによって、簡単に計算能力が環境に追加されますが、より多数の区画に遷移を行うという点の他は、新しい考慮事項が加わることはありません。 DB2 ユニバーサル・データベースはこの環境をサポートします。

区分データベース構成におけるバリエーションの 1 つとして、データベース区画が SMP マシンである構成があります。 DB2 ユニバーサル・データベースはこの環境をサポートします。

環境を変更してシステムを拡大または縮小するときは、そのような変更が、データのロード、データベースのバックアップおよび復元などのデータベース手順に与える影響をよく知っている必要があります。

新しいデータベース区画を追加するときは、その処理が完了し、新しいサーバーが正常にシステムに組み込まれるまでは、新規区画を活用するデータベースの削除または作成を行うことはできません。

マシンへのプロセッサーの追加

既存のプロセッサーが常に使用状況にあるような場合は、マシンに 1 台または複数台の追加プロセッサーをインストールすることを検討してください。 DB2 データベース・マネージャーがこの新しいプロセッサーを利用できるようにするには、構成パラメーターを検討し、おそらくは更新する必要があります。 (Solaris のように、オペレーティング・システムによっては、プロセッサーを動的にオンラインまたはオフラインに構成変更することができるものがあります。) 使用するプロセッサー数を判別するために使用できるパラメーターおよび更新の必要がある可能性のあるパラメーターには、次のものがあります。

- 449ページの『デフォルトの程度 (dft_degree)』
- 473ページの『照会の最大並列処理度 (max_querydegree)』
- 474ページの『区画内並列処理機能の使用可能化 (intra_parallel)』

また、更新する必要のあるアプリケーションに関連するパラメーターについても考慮してください。詳細については、89ページの『アプリケーションの並列処理』を参照してください。

通信に TCP/IP が使用されている環境で作業している場合は、 DB2TCPCONNMGRS レジストリー変数の値を考慮する必要があります。この変数について詳しくは、 501ページの『付録A. DB2 レジストリー変数と環境変数』を参照してください。

停止しているシステムへのデータベース区画の追加

システムの稼働中、またはシステムの停止時に、区分データベース・システムにデータベース区画を追加することができます。この作業を行う方法を以下に説明します。新しいサーバーを追加するのは時間のかかる作業なので、これはデータベース・マネージャーがすでに実行しているときに行うこともできます。この手順については、300ページの『実行しているシステムへのデータベース区画の追加』に述べてあります。

ADD NODE コマンドを使用すると、システムにデータベース区画を追加することができます。このコマンドは以下のようにして呼び出すことができます。

- `db2start` のオプションとして
- 以下を使用して
  - コマンド行プロセッサー ADD NODE コマンド
  - `sqleaddn`
  - `sqlepstart`

このコマンドを呼び出すために使用する方法は、システムが停止している (`db2start` を使用) か、実行している (他のいずれでも使用) かによって決まります。

新規のデータベース区画を ADD NODE コマンドを使用してシステムに追加すると、すでにインスタンスに入っているすべてのデータベースはその新規データベース区画に作成されます。システム管理者は、作成されるデータベースで使用される一時表スペースのためのコンテナーを指定することもできます。このコンテナーは、以下のようにすることができます。

- 各データベースのカタログ・ノードに定義されるものと同じ (これはデフォルト)。
- 他のデータベース区画に定義されているものと同じ。
- まったく作成しない。 ALTER TABLESPACE ステートメントを使用して、各データベースに一時表スペース・コンテナーを追加してからでないと、データベースを使用できません。

新規区画上のデータベースは、 1 つか複数のノードグループを変更して新規のデータベース区画を含めるようにするまでは、データを入れるために使用できません。ノードグループの変更方法についての詳細は、 310ページの『データベース区画の追加と除去』を参照してください。

注: システムに定義されているデータベースがなく、 UNIX ベースのシステムで DB2 エンタープライズ拡張エディションを実行している場合、 `db2nodes.cfg` ファイルを編集して、新規データベース区画定義を追加します。以下の手順はどれも使用してはなりません。適用されるのは、データベースが存在する場合だけです。ノード構成ファイルの更新方法の詳細は、 *管理の手引き: 計画* の『ノード・グループの変更』を参照してください。

Windows NT についての考慮事項: Windows NT 上の DB2 エンタープライズ拡張エディションを使用している場合、インスタンスにデータベースがなければ、 DB2NCRT コマンドを使ってデータベース・システムを拡大または縮小する必要があります。このコマンドの詳細については、*コマンド解説書* を参照してください。他方、データベースがあれば、DB2START ADDNODE コマンドを使用します。このコマンドを使えば、システムのサイズ変更時に既存のデータベースごとにデータベース区画が確実に作成されます。 Windows NT 上で使用しなければならない DB2START コマンドとパラメーターの詳細については、 *コマンド解説書* を参照してください。 Windows NT では、ノード構成ファイル (db2nodes.cfg) を手作業で編集することは避けてください。そのようなことをすれば、ファイル内での整合性が失われるおそれがあります。

実行しているシステムへのデータベース区画の追加

システムが動いていて、アプリケーションがデータベースに接続されている間に、区分データベースのシステムに新しいデータベース区画を追加することができます。しかし、新規に追加されたサーバーがすべてのデータベースに使用できるようになるのは、データベース・マネージャーがシャットダウンされて再び始動された後です。

データベース区画を複数サーバー・システムに追加するためには、次のようにします。

1. システムにすでにあるサーバーにデータベース区画を作成する場合は、次のステップに進んでください。それ以外の場合は、次を行います。
   - UNIX プラットフォーム:
     a. 新しいサーバーをインストールする。この作業としては、実行可能ファイルをアクセス可能にし (共用ファイル・システム・マウントまたはローカル・コピーを使用)、オペレーティング・システム・ファイルをすでにあるプロセッサー上のオペレーティング・システム・ファイルと同期化させ、 sqllib ディレクトリーがファイル共用システムとしてアクセス可能であることを確認し、関連オペレーティング・システム・パラメーター (最大プロセス数など) が適切な値に設定されていることを確認する必要があります。
     b. すべてのデータベース区画において、 etc ディレクトリーの hosts ファイルのネーム・サーバーにホスト名を登録する。
   - Windows NT プラットフォーム:
     a. 新しいサーバーをインストールする。
     b. ADD NODE コマンドを新規サーバーで実行する。このコマンドも、システムにすでにある各データベースに対してローカルにデータベース区画を作成します。新規データベース区画のためのデータベース・パラメーターは、デフォルト値に設定されます。各データベース区画は、ユーザーがそこにデータを移すまでは空のままです。データベース構成パラメーター値を更新して、他のデータベース区画上の値と一致させる必要があります。
     c. ステップ 3 に進みます。

2. 任意のデータベース区画で DB2START コマンドを実行する。このとき、NODENUM、ADDNODE、HOSTNAME、PORT、および NETNAME パラメーターに対して新規区画値を指定する。 Windows NT プラットフォームでは、COMPUTER、USER、および PASSWORD パラメーターも指定する必要があります。 DB2START コマンドの詳細については、*コマンド解説書* を参照してください。

   また、データベースに作成する必要のある任意の一時表スペース・コンテナーのためのソースを任意指定で指定することもできます。表スペース情報を提供しないと、一時表スペース・コンテナー定義は各データベースのカタログ・ノードから検索されます。

   このコマンドが完了すると、新しいサーバーは停止します。 DB2STOP が実行されるまでは、ノード構成ファイルがこの新しいサーバー情報によって更新されることはありません。このため、 ADD NODE コマンド (ADDNODE パラメーターが指定されたときに呼ばれる) は確実に正しいデータベース区画で実行されます。このユーティリティーが終了すると、新しいサーバーは停止します。

3. DB2STOP コマンドを実行することによってデータベース・マネージャーを停止する。

   システム内のすべてのデータベース区画を停止すると、ノード構成ファイルが更新されて新規データベース区画が組み込まれます。

4. DB2START コマンドを実行してデータベース・マネージャーを開始する。

   これで、新規に追加されたデータベース区画が残りのシステムと共に開始されます。

   システム内のすべてのデータベース区画が実行中になると、データベースの作成または除去などのシステム全般にわたる活動を行うことができます。

   注: 新しい `db2nodes.cfg` ファイルにアクセスするには、すべてのデータベース区画サーバーに関して DB2START コマンドを 2 回発行しなければならない場合があります。

5. オプションとして、新規データベース区画のすべてのデータベースのバックアップをとる。
6. オプションとして、新規データベース区画にデータを再配分する。詳細は、309ページの『第11章 データベース区画間でのデータの再配分』を参照してください。

停止しているシステムへのデータベース区画の追加

区分データベース・システムの停止時に、新しいデータベース区画を追加することができます。新規に追加されたデータベース区画がすべてのデータベースに利用可能になるのは、データベース・マネージャーを再び始動したときです。オプションが 2 つあります。ノード構成ファイルは、データベース・マネージャーに更新させてもよいし、自分で手作業で更新することもできます。どちらの手順も、最初のステップは同じです。

注: Windows NT で作業している間は、手作業でノード構成ファイルを更新することはできません。手作業ではなく、(以下に示すように) データベース・マネージャーを使ってこのファイルを更新します。

複数サーバー・システムに新規データベース区画を追加するには、次のようにします。

1. DB2STOP を出して、すべてのデータベース区画を停止する。
2. システムにすでにあるプロセッサーにサーバーを作成する場合は、次のステップに進んでください。それ以外の場合は、次を行います。
   a. UNIX プラットフォーム:
      1) 新しいサーバーをインストールする。この作業としては、実行可能ファイルをアクセス可能にし (共用ファイル・システム・マウントまたはローカル・コピーを使用)、オペレーティング・システム・ファイルをすでにあるプロセッサー上のオペレーティング・システム・ファイルと同期化させ、 `sqllib` ディレクトリーがファイル共用システムとしてアクセス可能であることを確認し、関連オペレーティング・システム・パラメーター (最大プロセス数など) が適切な値に設定されていることを確認する必要があります。
      2) すべてのデータベース区画において、 `etc` ディレクトリーの `hosts` ファイルのネーム・サーバーにホスト名を登録する。
   b. Windows NT プラットフォーム:
      1) 新しいサーバーをインストールする。
      2) ADD NODE コマンドを新規サーバーで実行する。このコマンドも、システムにすでにある各データベースに対してローカルにデータベース区画を作成します。新規データベース区画のためのデータベース・パラメーターは、デフォルト値に設定されます。各データベース区画は、ユーザーがそこにデータを移すまでは空のままです。データベース構成パラメーター値を更新して、他のデータベース区画上の値と一致させる必要があります。
      3) DB2START コマンドを実行して、データベース・システムを開始する。ノード構成ファイル (`db2nodes.cfg`) は、すでに新規のサーバーのインストール時に更新されており、その新規のサーバーは組み込まれています。
      4) オプションとして、新規サーバーにデータを再配分する。これを行う方法についての詳細は、 309ページの『第11章 データベース区画間でのデータの再配分』を参照してください。
   c. `db2nodes.cfg` ファイルをデータベース・マネージャーに更新してもらいたい場合は、 303ページの『データベース・マネージャーによるノード構成ファイルの更新』の指示にしたがって続行する。

      注: Windows NT では、`db2nodes.cfg` を手作業で編集することは避けてください。そのようなことをすれば、ファイル内での整合性が失われるおそれがあります。手作業ではなく、データベース・マネージャーを使ってこのファイルを更新します。

自分で db2nodes.cfg ファイルを更新する場合は、 『ノード構成ファイルの手作業での更新』の指示にしたがって続行する。

データベース・マネージャーによるノード構成ファイルの更新

1 つまたは複数の新規データベース区画を区分データベース・システムに追加した後は、新規区画を使用可能にするために、 db2nodes.cfg ファイルを更新する必要があります。データベース・マネージャーにノード構成ファイルを更新させることにした場合、以下の情報で行うべき事柄の詳細が示されています。

注: ノード構成ファイルを手作業で更新することにした場合、このセクションの残りの情報を無視してください。

次のプロシージャーを続行します。

1. 新しいデータベース区画で DB2START コマンドを実行する。このとき、NODENUM、ADDNODE、HOSTNAME、PORT および NETNAME パラメーターを指定してください。 Windows NT プラットフォームでは、COMPUTER、USER、および PASSWORD パラメーターも指定する必要があります。 DB2START コマンドの詳細については、*コマンド解説書* を参照してください。これらのパラメーターに指定した値は、ノード構成ファイルの更新に使用されます。

   このコマンドが完了すると、新しいサーバーは停止します。 DB2STOP が実行されるまでは、ノード構成ファイルがこの新しいサーバー情報によって更新されることはありません。このため、 ADD NODE コマンド (ADDNODE パラメーターが指定されたときに呼ばれる) は確実に正しいデータベース区画で実行されます。このユーティリティーが終了すると、新しいデータベース区画は停止します。
2. DB2STOP コマンドを出す。

   DB2STOP コマンドを出すと、ノード構成ファイルが更新されて、新規のデータベース区画が組み込まれます。
3. DB2START コマンドを出して、データベース・システムを開始する。

   注: 新しいノード構成ファイルにアクセスするには、すべてのデータベース区画に関して DB2START コマンドを 2 回発行しなければならない場合があります。
4. オプションとして、新規データベース区画のすべてのデータベースのバックアップをとる。
5. オプションとして、新規サーバーにデータを再配分する。詳細は、309ページの『第11章 データベース区画間でのデータの再配分』を参照してください。

ノード構成ファイルの手作業での更新

1 つまたは複数の新規データベース区画を区分データベース・システムに追加した後は、新規区画を使用可能にするために、 db2nodes.cfg ファイルを更新する必要があります。ノード構成ファイルを手作業で更新することにした場合は、以下の情報の中に、

ノード構成ファイルを手動で更新するために行うべき事柄の詳細が示されています。(Windows NT 上での作業の際は、ノード構成ファイルを手作業で更新しないでください。)

注: データベース・マネージャーにノード構成ファイルを更新させることにした場合は、 303ページの『データベース・マネージャーによるノード構成ファイルの更新』に戻ってください。

次のプロシージャーを続行します。

1. db2nodes.cfg ファイルを編集し、新規データベース区画をそこに追加する。
2. 次のコマンドを出して、新しいノードを開始する。

   ```
   DB2START NODENUM nodenum
   ```

   新しいデータベース区画に割り当てる番号を *nodenum* の値として指定する。
3. この新規サーバーを論理データベース区画 (すなわち、ノード 0 ではない) にする場合は、 **db2set** コマンドを使用して、追加するデータベース区画の番号を指定し、DB2NODE レジストリー変数を更新する。
4. ADD NODE コマンドを新規データベース区画で実行する。

   このコマンドも、システムにすでにある各データベースに対してローカルにデータベース区画を作成します。新規データベース区画のためのデータベース・パラメーターは、デフォルト値に設定されます。各データベース区画は、ユーザーがそこにデータを移すまでは空のままです。データベース構成パラメーター値を更新して、他のデータベース区画上の値と一致させる必要があります。
5. ADD NODE コマンドが完了したら、DB2START コマンドを出して、システム内の他のデータベース区画を開始する。

   すべてのデータベース区画が正常に開始するまでは、データベースの作成または除去などのシステム全般にわたる活動を行おうとしてはなりません。
6. オプションとして、新規サーバーのすべての新規データベース区画のバックアップをとる。
7. オプションとして、新規データベース区画にデータを再配分する。詳細は、309ページの『第11章 データベース区画間でのデータの再配分』を参照してください。

システムからのデータベース区画の除去

DB2STOP コマンドに DROP NODENUM パラメーターを指定する、または sqlepstp API を使用することによって、データベース区画を除去することができます。除去を行う前に、まず、除去しようとしているデータベース区画がどのデータベースによっても使用されていないことを確認する必要があります。これを検査するには DROP NODE VERIFY コマンドを出します。

このデータベース区画を調整プログラムとしていたすべてのトランザクションがすべて正常にコミットされている、またはロールバックされていることを確認する必要があります。このためには、他のサーバーで破損リカバリーを行う必要があることがあります。

たとえば、(調整プログラム・ノードのような) 調整プログラム・データベース区画を除去すると、その調整プログラム・ノードが除去される前にトランザクションに関連する他のデータベース区画が破損した場合、破損したデータベース区画はどの未確定トランザクションの結果についても調整プログラム・ノードを照会することができなくなります。

区分データベース・システムからデータベース区画を除去するために、次のようにします。

1. このノードにある各データベースのデータを再配分する。これにより、除去しようとしているデータベース区画がどのデータベースによっても使用されていないという要件が満たされます。詳細は、309ページの『第11章 データベース区画間でのデータの再配分』を参照してください。
2. DROP NODE VERIFY コマンドまたは sqledrpn API を使用して、サーバーが使用中ではないことを確認する。

受け取ったメッセージに従って、ステップ 3 または 4 に進む。

3. メッセージ SQL6034W (Node not used in any database - ノードはほかのデータベースによって使用されていません) が出た場合は、以下を行う。
   a. DROP NODENUM パラメーターを指定した DB2STOP コマンドを出して、データベース区画を除去する。このコマンドが正常に終了すると、システムは停止します。
   b. 必要な場合は、DB2START コマンドによってデータベース・マネージャーを開始する。
4. SQL6035W (Node in use by database - ノードはほかのデータベースによって使用されています) が出た場合は、次のようにする。
   a. REDISTRIBUTE NODEGROUP コマンドを使用して、削除したいデータベース区画のデータを、メッセージ SQL6035W の指示に従って、データベース別名から他のデータベース区画へ再配分する。これが終了するまでは、データベース区画を除去することはできません。
   b. データベース区画に定義されているイベント・モニターがあれば、それを除去する。
   c. 2 に戻り、続行します。

分割したデータベースにノードを追加する際の問題

デフォルト・ページ・サイズ (4KB) と異なるページ・サイズで、複数のシステム一時表スペースを持つ分割したデータベースにノードを追加する際に、以下のようなエラー・メッセージや SQL コードが表示される場合があります: 『SQL6073N ノードの追加に失敗しました 』これはノードが生成される際、IBMDEFAULTBP バッファー・プールが 4 KB のページ・サイズで存在するために、発生します。

たとえば、現在の分割したデータベースにノードを追加するための **db2start** コマンドを使用することができます。

```
DB2START NODENUM 2 ADDNODE HOSTNAME newhost PORT 2
```

分割されたデータベースがデフォルトのページ・サイズでシステム一時表スペースを持つ場合以下のメッセージが返されます:

```
SQL6075W The Start Database Manager operation successfully added the node.
The node is not active until all nodes are stopped and started again.
```

ただし、分割されたデータベースがデフォルトのページ・サイズではないシステム一時表スペースを持つ場合は、以下のメッセージが返されます。

```
SQL6073N Add Node operation failed. SQLCODE = "<-902>"
```

同様の例では、db2nodes.cfg ファイルに新しいノードを記述し更新するとノード追加コマンドが使用できます。そのファイルを編集した後、デフォルトのページ・サイズでシステム一時表スペースを持っている、分割されたデータベースで ADD NODE コマンドを実行すると以下のメッセージが返されます:

```
DB20000I The ADD NODE command completed successfully.
```

ただし、分割されたデータベースがデフォルトのページ・サイズではないシステム一時表スペースを持つ場合は、以下のメッセージが返されます。

```
SQL6073N Add Node operation failed. SQLCODE = "<-902>"
```

上で概略を説明した問題を予防する 1 つの方法は以下を実行することです:

```
DB2SET DB2_HIDDENBP=16
```

このコマンドは **db2start** や ADD NODE コマンドを発行する前に実行します。このレジストリー変数は DB2 がデフォルトと異なるページ・サイズを使用している 16 ページ分のプールを隠しバッファーに割り振れるようにします。これが使用可能になることでノード追加操作が正常に完了します。

これらの問題を予防する別の方法としては、ADD NODE コマンドや **db2start** コマンドで WITHOUT TABLESPACES 文節を指定する方法があります。コマンド実行後、CREATE BUFFERPOOL ステートメントを使用して、バッファー・プールを作成し、ALTER TABLESPACE ステートメントを使用して、バッファー・プールにシステム一時表スペースを作成する必要があります。

デフォルト・ページ・サイズ (4 KB) と異なるページ・サイズで複数の表スペースを持つ既存のノード・グループにノードを追加する場合、以下のようなエラー・メッセージが出る場合があります : 『SQL0647N バッファー・プール "" は現在活動状態ではありません。』これは、デフォルトでないページ・サイズ・バッファー・プールを、表スペースとして活動状態になっていない、新規のノード上に作成した場合に発生します。

たとえば、ノード・グループにノードを追加するために ALTER NODEGROUP ステートメントを使用することができます:

```
DB2START
CONNECT TO mpp1
ALTER NODEGROUP ng1 ADD NODE (2)
```

デフォルトのページ・サイズの表スペースを持つノード・グループの場合は以下のようなメッセージが返されます:

```
SQL1759W Redistribute nodegroup is required to change data positioning for
objects in nodegroup "<ng1>" to include some added nodes or exclude
some drop nodes.
```

一方、ノード・グループが、デフォルトのページ・サイズでない表スペースを持つ場合以下のメッセージが返されます:

```
SQL0647N Bufferpool "" is currently not active.
```

この問題を予防する 1 つの方法としては、それぞれのページ・サイズのバッファー・プールを作成し、その後、ALTER NODEGROUP ステートメントを発行する前にデータベースに再接続する方法があります:

```
DB2START
CONNECT TO mpp1
CREATE BUFFERPOOL bp1 SIZE 1000 PAGESIZE 8192
CONNECT RESET
CONNECT TO mpp1
ALTER NODEGROUP ng1 ADD NODE (2)
```

この問題を防止する 2 つ目の方法は、以下のコマンドを実行する方法です:

```
DB2SET DB2_HIDDENBP=16
```

db2start コマンドや CONNECT、 ALTER NODEGROUP ステートメントなどを発行する前に実行します。

ALTER TABLESPACE ステートメントはノードに表スペースを追加するために使用する場合、他の問題が起こることもあります。たとえば、次のとおりです。

```
DB2START
CONNECT TO mpp1
ALTER NODEGROUP ng1 ADD NODE (2) WITHOUT TABLESPACES
ALTER TABLESPACE ts1 ADD ('ts1') ON NODE (2)
```

この一連のコマンドとステートメントは SQL0647N エラー・メッセージを生成します。 (想定メッセージ SQL1759W ではありません)。

この変更を正しく完了させるには、ALTER NODEGROUP... WITHOUT TABLESPACES ステートメントの後で、データベースに再接続する必要があります。

```
DB2START
CONNECT TO mpp1
ALTER NODEGROUP ng1 ADD NODE (2) WITHOUT TABLESPACES
CONNECT RESET
CONNECT TO mpp1
ALTER TABLESPACE ts1 ADD ('ts1') ON NODE (2)
```

この問題を防止する他の方法は、以下のコマンドを実行する方法です:

```
DB2SET DB2_HIDDENBP=16
```

db2start コマンドや CONNECT、ALTER NODEGROUP、 ALTER TABLESPACE ステートメントなどを発行する前に実行します。

第11章 データベース区画間でのデータの再配分

区分データベース環境で作業を行っている場合のみ、データの再配分を考慮する必要があります。単一区分データベース環境で作業を行っている場合には、ここで説明する情報を使用する必要はありません。

データ再配分ユーティリティーを使用すると、既存のノード・グループ内のデータベース区画間でデータを移動することができます。このユーティリティーを使用して行えることを、以下に示します。

- データベース区画間のデータ量と処理負荷のバランスを調整する。

  これは、通常処理において、すべてのデータにアクセスするようなデータベース表を持っている場合に便利です。

- データベース区画間のデータ配分にスキューを導入する。

  これは、通常処理において、一部のデータのみにアクセスするようなデータベース表を持っている場合に便利です。この場合には、アクセスの頻度が少ないデータはノード・グループ内の少数のデータベース区画に配分し、アクセスの頻度が多いデータは多数の区画に配分するといったように、表を再配分することができます。こうすることによって、実行頻度が最も多いアプリケーションにおけるアクセス・パフォーマンスとスループットが向上します。

データ再分配ユーティリティーは、REDISTRIBUTE NODEGROUP コマンドで呼び出します。このコマンドの構文について詳しくは、 *コマンド解説書* を参照してください。

表の併置を保存するために、この操作はノード・グループ内の表すべてに適用され、再配分は表レベルではなくノード・グループ・レベルで行われます。

必要なデータ配分を行うために、ユーティリティーは区分化マップを使用して、ノード・グループのデータベース区画間で表の行を移動します。指定したオプションによって、ユーティリティーはターゲット区分化マップを生成することもできるし、既存の区分化マップを入力として使用することもできます。

注:

1. データ再配分操作に必要だと思われるログ・スペース要件に基づいて、ログ・ファイルのサイズを指定する必要があります。またログが、データを再配分する各データベース区画ごとに行われる INSERT 操作および DELETE 操作を受け入れるのに十分な大きさであることも確認しなければなりません。
2. 複製された要約表を含むノード・グループでデータを再配分したい場合、まずこれらの表を除去し、ノード・グループを再配分してから表を再作成する必要があります。複製された要約表を含むノード・グループを再配分することはできません。

データの区分方法

デフォルトには、データ再配分ユーティリティーは、各ハッシュ区画に振り分ける行数を同じと想定するので、ハッシュ区画をノード・グループのデータベース区画すべてに均等に区分けします。各ハッシュ区画に同じ行数を振り分けるのではない場合は、*配分ファイル* を使用して、現行の配分方法を指定することができます。このファイルには、4 096 個のハッシュ区画のそれぞれに対する値が入っています。各値は、対応するハッシュ区画の*重み* として使用されます。データ再配分ユーティリティーは、ターゲット区分化マップを生成し、その中ではすべてのデータベース区画はほぼ同じ重みを有しています。したがって、配分ファイルを使用すると、データ配分がスキューされていた場合であっても、均等なデータ配分を行うことができます。

オートローダー・ユーティリティーを使用すると、 ANALYZE オプションを使用してデータ配分ファイルを作成することができます。このファイルは、データ再配分ユーティリティーへの入力として使用することが可能です。 オートローダー・ユーティリティーの詳細は、 *データ移動ユーティリティー手引きおよび解説書* を参照してください。

また代替的な方法としては、PARTITION および NODENUMBER という SQL 関数を使用して、ハッシュ区画またはデータベース区画間の現行のデータ配分を判別することができます。 (PARTITION 関数を使って、ハッシュ区分化での配分を判別します。) この情報を使用すると、配分ファイルとターゲット区分化マップの両方について知ることができます。

たとえば、不均等なデータ配分のために異常なほど多くの行数を持つデータベース区画 (もしあれば) を判別するには、以下のようにします。

```
SELECT PARTITION(column_name), COUNT(*) FROM table_name
   GROUP BY PARTITION(column_name)
   ORDER BY PARTITION(column_name) DESC
FETCH FIRST 100 ROWS ONLY
```

table_name が最も大きな表で、 column_name がその表の適切な列であることを確認してください。

データベース区画の追加と除去

ALTER NODEGROUP ステートメントを使用すると、ノード・グループにデータベース区画を追加したり、除去したりすることができます。データベース区画を追加するときには、その区画を事前にノード構成ファイルに定義しておく必要があります。

ALTER NODEGROUP ステートメントの使用後には、新しい区分化マップが作成されます。データ再配分ユーティリティーの使用時には、この新しい区分化マップをターゲット区分化マップにすることができます。 (ターゲット区分化マップを作成する他の方法としては、ユーザー自身で作成する方法があります。)

ALTER NODEGROUP ステートメントを WITHOUT TABLESPACES 文節で使用する場合、データの再配分前に表スペース・コンテナーを新しいデータベース区画 (または区画) に追加することが必要です。 ALTER NODEGROUP ステートメントの詳細については、 *SQL 解説書* を参照してください。

ターゲット区分化マップの指定

データ再配分ユーティリティーは区分化マップを使用して、データの再配分を行います。ユーティリティーが独自にターゲット区分化マップを作成することも、ユーザーの側でユーティリティーが使用するターゲット区分化マップを用意することも可能です。ユーザーが作成する場合には、項目 (1 つまたは複数) によって、データの再配分の結果として得られたノード・グループのタイプを示します。

- 単一区画ノード・グループの場合は、1 項目
- 複数区画ノード・グループの場合は、4 096 項目

ターゲット区分化マップに複数のデータベース区画を含める場合には、ノード・グループ内のすべての表に前もって区分化キーが定義されていなければなりません。

ターゲット区分化マップには、 SYSCAT.NODEGROUPDEF カタログ表に定義されたデータベース区画番号だけを入れることができます (ただし、 IN_USE 値が 'T' のものは除く)。 ('T' は、その区画がターゲット区分化マップにないことを意味します。) IN_USE 値が 'D' (除去を意味する) であるすべてのデータベース区画がターゲット区分化マップ内にないという場合には、再配分操作が正常に完了した時点で除去されています。

データベース区画間でのデータの再配分方法

データ再配分操作は、データベースの中の指定されたノード・グループ内の表のセットに対して行われます。 (アプリケーションは、操作の実行前に、カタログ・データベース区画でデータベースに接続する必要があります。) ユーティリティーはソース区分化マップとターゲット区分化マップの両方を使用して、どのハッシュ区画が新しい場所 (すなわち、新しいデータベース区画番号) に割り当てられているのかを識別します。新しい場所に割り当てられた区画に対応する行はすべて、ソース区分化マップに指定されたデータベース区画からターゲット区分化マップに指定されたデータベース区画に移動されます。

データ再配分ユーティリティーは、以下のことを行います。

1. ターゲット区分化マップの新しい区分化マップ ID を取得し、その ID を SYSCAT.PARTITIONMAP カタログ表に挿入する。
2. ノード・グループの SYSCAT.NODEGROUPS カタログ表内の REBALANCE_PMAP_ID 列を、新しい区分化マップ ID で更新する。
3. 任意の新しいデータベース区画を SYSCAT.NODEGROUPDEF カタログ表に追加する。

4. 除去したいデータベース区画があればその区画すべてについて、SYSCAT.NODEGROUPDEF カタログ表内の IN_USE 列を D に設定する。
5. カタログの更新を COMMIT する。
6. 新しいデータベース区画すべてに対して、データベース・ファイルを作成する。
7. ノード・グループ中のすべての表に対してそれぞれの表を基本とするデータを再配分する。これについては、『表のデータを再配分する方法』で説明されています。
8. データベース・ファイルを削除し、 SYSCAT.NODEGROUPDEF カタログ表の中から前に除去するものとしてマークを付けたデータベース区画の項目を削除する。
9. SYSCAT.NODEGROUPS カタログ表内のノード・グループ・レコードを更新して、PMAP_ID を値 REBALANCE_PMAP_ID に、REBALANCE_PMAP_ID を NULL に設定する。
10. SYSCAT.PARTITIONMAPS カタログ・ビューから古い区分化マップを削除する。
11. すべての変更を COMMIT する。

表のデータを再配分する方法

表のデータの再配分を行うときは、ユーティリティーは以下のことを行います。

1. SYSTABLES カタログ表にあるその表の行をロックする。
2. この表に関連するパッケージすべてを無効にする。表に関連付けられた区分化マップ ID は、表の再配分が行われるので変更されることになります。パッケージが無効にされたので、コンパイラーは表に関する新しい区分化情報を取得して、それに応じてパッケージを生成する必要があります。
3. 排他モードで表をロックする。
4. DELETE および INSERT を使って表のデータを再配分する。
5. 再配分操作が正常に終了したら、ユーティリティーは次のことを行う。
   a. 表に対して COMMIT を出す。
   b. 続いて、ノード・グループ内の次の表の処理を行う。

   表が完全に再配分される前に操作が失敗した場合は、ユーティリティーは次のことを行う。
   a. 表に加えた更新に対して ROLLBACK を出す。
   b. 再配分操作全体を終了して、エラーを戻す。

データの配分時にログ・スペース所要量の見積もりを行うことは重要です。ログは、データを再配分する各データベース区画ごとの INSERT 操作および DELETE 操作を受け入れるのに十分の大きさでなければなりません。ロギング所要量は、最多のデータを失うデータベース区画、または最多のデータを獲得するデータベース区画で最も大きくな

ります。より多くのデータベース区画に移動しようとしている場合は、現行データベース区画の、新しいデータベース区画数に対する比率が、 INSERT 操作および DELETE 操作の数を決定する助けになります。

たとえば、4 つのデータベース区画から 5 つのデータベース区画への移動の場合、元の 4 つのデータベース区画の約 20 % のデータが、新規データベース区画に移動します。これは、元の 4 つのデータベース区画が、各データベース区画のデータの合計に基づいて、それぞれ 20 % の DELETE 操作を行うことを意味します。新規データベース区画では、すべての INSERT 操作 (つまり、元の 4 つのデータベース区画すべてで行った DELETE 操作と同じ数に相当する操作) を行います。

上記の例は、均等なデータの配分が行われることを前提としています。区分化キーに多数の NULL 値が存在する場合のような、データの配分が均等に行われない場合もあります。このような場合、すべての行は、あるデータベース区画では以前の区分化体系により、また別のデータベース区画では新規の区分化体系により存在します。その結果、これら 2 つのデータベース区画で必要とされるログ・スペースの量が増え、おそらく、均等な配布を想定した場合の計算量を超えることになります。

実際の計算を行うときは、 (20 % などの) 変更のパーセンテージに、最大の表のサイズを乗算しなければなりません。これを行うのは、各表の再配分が、単一のトランザクションとして実行されるためです。

注: ただし、最大の表は均等に配分されているが、 (たとえば) 2 番目に大きな表には 1 つまたは複数の膨張したデータベース区画が存在する場合もあります。そのような場合には、最大の表ではなく、 2 番目に大きな表の使用を検討することをお勧めします。

データベース区画で挿入および削除されるデータの最大量を計算した後、その数を 2 倍して、アクティブ・ログのピーク・サイズを判別します。このサイズがアクティブ・ログ限界の 32 GB を超える場合、複数のステップでデータ再配分を行わなければなりません。 『makepmap』というユーティリティーを使用すれば、各ステップで 1 つずつ、一連のターゲット区分化マップを生成することができます。

再配分のエラーからのリカバリー

再配分操作の実行が開始されると、ファイルが `sqllib` ディレクトリーの `redist` サブディレクトリーに書き込まれます。この状況ファイルには、データベース区画上で行われた操作すべて、再配分された表の名前 (複数の場合もある)、および操作の完了状況がリストされます。ある表が再配分できなかった場合には、その表の名前と該当する SQLCODE がファイルにリストされます。入力パラメーターが正しくないために再配分操作が開始できない場合には、ファイルは作成されず、SQLCODE が戻されます。

このファイルの命名規則は、次に示すとおりです。

データベースname.ノード・グループname.timestamp (for UNIX platforms)
データベースname¥ノード・グループname¥date¥time (for non-UNIX platforms)

注: 非 UNIX プラットフォームの場合には、 nodegroupname の最初の 8 バイトのみが使用されます。

データの再配分操作が失敗した場合には、一部の表は再配分されたが、一部の表は再配分されなかったということが起こる可能性があります。これは、データの再配分は一度に 1 つの表に対してのみ行われるために起こります。リカバリーするには、次に示す 2 つの方法を取ることができます。

- CONTINUE オプションを使用して操作を続行して、残りの表の再配分を行う。
- ROLLBACK オプションを使用して再配分を取り消して、再配分された表を元の状態に設定し直す。ロールバック操作には、再配分操作でかかった時間とほぼ同じくらいの時間が必要になります。

いずれのオプションを使用する場合であっても、使用する前に、 SYSNODEGROUPS カタログ表の REBALANCE_PMID 列が非 NULL 値に設定されていることをチェックして、直前に行ったデータ再配分操作が確かに失敗していることを確認する必要があります。

このファイルを誤って削除してしまった場合であっても、 CONTINUE 操作を試行することができます。

データの再配分とその他の操作

ユーティリティーの実行中に、ノード・グループのオブジェクトに対して以下に示す操作を行うことができます。ただし、再配分中の表に対してはそれらの操作を実行することはできません。行うことができる操作は、以下のとおりです。

- 他の表に対する索引の作成。 CREATE INDEX ステートメントは、影響される表の区分化マップを使用します。
- 他の表の除去。 DROP TABLE ステートメントは、影響される表の区分化マップを使用します。
- 他の表に対する索引の除去。 DROP INDEX ステートメントは、影響される表の区分化マップを使用します。
- 他の表の照会。
- 他の表の更新。
- ノード・グループで定義された表スペースで新しい表を作成する。 CREATE TABLE ステートメントは、ターゲット区分化マップを使用します。
- ノード・グループでの表スペースの作成。

ユーティリティーの実行中に行うことができない操作は、以下のとおりです。

- そのノード・グループ上での別の再配分操作

- ノード・グループ内の任意の表に対する ALTER TABLE
- ノード・グループの除去
- ノード・グループの変更

データの再配分の終了後の作業

ノード・グループにまたがってのデータの再配分を完了したら、 RUNSTATS を実行して、再配分された可能性のある表に関連する統計を更新することを強くお勧めします。

RUNSTATS コマンドの詳細については、*コマンド解説書* を参照してください。

第12章 ベンチマーク・テスト

ベンチマークは、アプリケーション開発ライフ・サイクルの通常の部分のうちの 1 つです。これはアプリケーション開発者とデータベース管理者 (DBA) の両方が関係するチームの作業であり、パフォーマンスを調べてそれを向上させるためにアプリケーションに対して行うものです。アプリケーション・コードが可能な限り効率的に作成されているとすれば、アプリケーションの要件に合わせてデータベースとデータベース・マネージャーの構成パラメーター (場合によってはアプリケーション・パラメーターも) を調整することによって、さらにパフォーマンスの改善を実現できます。

ベンチマークには、いくつか異なる種類があります。 *1 秒当たりのトランザクション・ベンチマーク*は、特定の限定された実験条件の下でデータベース・マネージャーのスループット能力を調べるものです。アプリケーション ・ベンチマークは、同じようにスループット能力をテストしますが、インプリメント状態のアプリケーションの実行条件にもっと近い条件でテストします。構成パラメーターを調整する目的のベンチマークは、こうした 『現実環境』の条件に基づくものであり、アプリケーションが可能な限り効率的に実行されるまで、そのアプリケーションから取った SQL をパラメーター値を変えて繰り返し実行することが関係しています。

ここで説明するベンチマーク方式は、構成パラメーターを調整するために設計されたものです。しかし、同じ基本的な技法を、パフォーマンスに影響を与える次のような他の要因を調整するために使用することもできます。

- SQL ステートメント
- 索引
- 表スペース構成
- アプリケーション・コード
- ハードウェア構成

ベンチマークは、データベース・マネージャーが各種の条件下でどのように応答するかを調べるためのよい方法でもあります。デッドロック処理、ユーティリティー・パフォーマンス、データをデータベースに迅速にロードする異なる方式、ユーザーを追加する場合のトランザクション率、および新規リリースの製品を実動状態のときのアプリケーションへの影響をテストするよう、シナリオを作成することができます。

この部分では、次の点について説明します。

ベンチマーク・テストの方法論

このベンチマーク技法は、科学的方式に基づいています。同じ条件下で実行される同じテストが、比較可能な結果を出すような、繰り返しに適した環境が作成されます。

さらに、通常の環境でテスト・アプリケーションを実行することによってベンチマークを開始することもできます。パフォーマンス問題の範囲が狭められたなら、テストし観察する関数の有効範囲を制限して、特殊化したテスト・ケースを開発することができます。つまり、特殊化テスト・ケースで、価値のある情報を得るためにアプリケーション全体をエミュレートする必要はありません。単純な測定から開始し、根拠のある場合のみ複雑化させてください。

良いベンチマーク (または測定) の特性には、次のものがあります。

- 各テストは繰り返しが可能です。
- テストの各繰り返しは、同じシステム状態で開始します。
- 測定しているシステム以外のシステムに、活動状態の関数またはアプリケーションはありません (他の活動がある程度そのシステムに含まれているシナリオの場合は除きます)。

  注: 開始されるアプリケーションは、最小化された状態またはアイドル状態であっても、メモリーを使用します。そのために、ページングがベンチマークの結果をゆがめたり、反復可能性規則に違反したりする可能性が大きくなります。
- ベンチマークに使用されるハードウェアおよびソフトウェアは、インストール先の実動環境に適合しています。

どのようなベンチマークを使用するにしても、何度もシナリオを考案し、実行しなければなりません。さらに、それぞれの実行後にかぎとなる情報を取り入れることは、アプリケーションとデータベース両方のパフォーマンスを向上させるために必要な変更を見分ける上で非常に重要です。

ベンチマーク・テストの準備

パフォーマンス・ベンチマークを開始する前に、アプリケーションのデータベースの論理設計が完了していなければなりません。表、視点、および索引を設定し、データを入れておく必要があります。表を正規化し、アプリケーション・パッケージをバインドし、表に実データを入れてください。

データベースの最終的な物理設計を決めておいてください。データベース・マネージャーのオブジェクトを、それらの最終的なディスク位置に入れ、ログ・ファイルのサイズを設定し、作業ファイルとバックアップの位置を決め、さらにバックアップ手順をテストするようにします。さらに、パッケージを調べて、可能な場合には行ブロック化などのパフォーマンス・オプションを使用可能にします。

アプリケーションのプログラミングと単体テストの中で、ベンチマーク・プログラムを作成できるようなフェーズに達していなければなりません (320ページの『ベンチマーク・プログラムの作成』を参照)。アプリケーションの実際上の限界はベンチマーク・テスト時に分かりますが、ここで説明するベンチマークの目的はパフォーマンスを測定することであり、障害や異常終了を検出することではありません。

ベンチマーク・テスト・プログラムは、最終的な実動環境を可能な限り正確に表す状態で実行する必要があります。ベンチマークは、同じメモリー構成、同じディスク構成の同じモデル・サーバー上で実行するのが理想的です。最終的にアプリケーションに、たくさんのユーザーと大量のデータが関係してくる場合、このことは特に重要になります。ベンチマークで直接使用するオペレーティング・システム自体および通信機能またはファイル機能も、やはり調整済みでなければなりません。

実動サイズのデータベースでベンチマークを行うことも重要です。個々の SQL ステートメントが返すデータは、実動で実施されるのと同じ量で、同じ程度に分類されるものでなければなりません。この規則に従うなら、アプリケーションのメモリー要件が典型的なものになります。

ベンチマークの対象の SQL ステートメントのタイプは、*典型的* か*最悪のケース* のいずれかです。これらは以下で説明します。

典型的な SQL

典型的な SQL には、ベンチマークの対象のアプリケーションの一般的操作で実行されるステートメントが含まれます。選択するステートメントは、アプリケーションの性質によって異なります。たとえば、データ入力アプリケーションでは INSERT ステートメントをテストしますが、銀行業務トランザクションでは、FETCH、UPDATE、およびいくつかの INSERT をテストします。実行の頻度および選択されたステートメントによって処理されるデータの量は、標準的と見なされるものでなければなりません。量が多過ぎると、典型的な SQL ステートメントであっても、*最悪のケース* のカテゴリーに入るものと見なされます。

最悪のケースの SQL

このカテゴリーに該当するステートメントには、次のものがあります。

- 頻繁に実行されるステートメント。
- 大量のデータを処理するステートメント。
- 時間が重要なステートメント。

  たとえば、ある顧客から電話がかかったときに実行されるアプリケーションとそのステートメントとは、顧客を待たせている間に、顧客の情報の検索や更新を行わなければなりません。
- 結合される表の数が多いステートメント、またはアプリケーションの中でも最も複雑な SQL が含まれるステートメント。

  たとえば、口座の異なる種類すべての月ごとの活動を行うための組み合わされた顧客ステートメントを生成する銀行業務アプリケーション。共通表には

顧客の住所と口座番号のリストを入れ、他のいくつかの表は結合して、すべての必要な口座トランザクション情報を処理および統合するようにしなければなりません。 1 つの口座に必要な作業量に、同じ期間内に処理しなければならない何千もの口座の数を乗算し、潜在的な時間の節約を考慮すると、パフォーマンス要件が導き出されます。

- アクセス・パスが不適切なステートメント。たとえば、あまり実行されないステートメントや、関与する表に関して作成された索引によってサポートされないステートメント。
- 経過時間の長いステートメント。
- アプリケーション初期設定時にのみ実行されるが、リソース要件が不均衡なステートメント。

  たとえば、その日のうちに処理されなければならない会計作業のリストを生成するアプリケーション。このアプリケーションが開始されると、最初の主要 SQL ステートメントにより 7 とおりの結合が生成され、それらによりこのアプリケーションのユーザーが担当する全会計の大きなリストが作成されます。このステートメントは 1 日のうち数回実行されるだけですが、正しく調整されていないと実行に数分間もかかってしまいます。

ベンチマーク・プログラムの作成

ベンチマーク・プログラムを設計し実施するときに考慮すべき要因がいくつかあります。このプログラムの主な目的はユーザー・アプリケーションをシミュレートするということですから、プログラムの全体的な構造が異なることはあります。アプリケーション全体をベンチマークとして使用し、単に複数の SQL ステートメントを分析するタイミングを合わせる手段として用いることもできます。大きかったり複雑であったりするアプリケーションの場合は、重要なステートメントを含むブロックを組み込んでおくほうがより実際的であることもあります。

特定の SQL ステートメントのパフォーマンスをテストする場合、ベンチマーク・プログラムにそのステートメントだけを組み込み、あとは CONNECT、PREPARE、OPEN など他の必要なステートメントとタイミング機構を加えるという別の方法もあります。

考慮すべき別の要因は、使用するベンチマークのタイプです。 1 組の SQL ステートメントを、一定の時間間隔をおいて繰り返し実行するという方法があります。実行するステートメントの数と時間間隔の比率によって、アプリケーションのスループットが決まります。あるいは、SQL ステートメントを個別に実行するのに必要とされる時間を単純に判別するということもできます。

ベンチマーク・プログラムのタイプいかんにかかわらず、個別の SQL ステートメントまたはアプリケーション全体であれ、経過時間を算定するには効率的なタイミング・システムが必要です。個々の SQL ステートメントが別個に実行されるアプリケーションをシミュレートする場合、CONNECT、 PREPARE、および COMMIT ステートメント

のための時間を考慮することが重要です。しかし、多くの異なるステートメントを処理するプログラムの場合、おそらく単一の CONNECT または COMMIT しか必要ではないため、個々のステートメントの実行時間に焦点を絞ることが優先です。

パフォーマンス分析においては各照会ごとの経過時間が重要な要因ですが、他の潜在的な障害が必ずしも明示されるわけではありません。たとえば、CPU 使用率、ロック、およびバッファー・プール入出力に関する情報は、アプリケーションが CPU をフルに使用してはおらず、入出力が関係していることを示しているかもしれません。ベンチマーク・プログラムを行うことによって、必要に応じてより詳細な分析のためにこのようなデータを入手することができます。

必ずしもすべてのアプリケーションで、照会によって取り出した一連の行全体を特定の出力装置に送信する必要はありません。たとえば、応答セット全体を別のプログラムへの入力として使用する (つまり最初のアプリケーションから 1 行も出力として送信されない) ものがあります。画面出力のデータを形式制御すると通常は CPU の消費の度合いが高くなり、ユーザーの希望どおりには行かないこともあります。正確にシミュレーションするには、ベンチマーク・プログラムが特定のアプリケーションの行処理を反映していなければなりません。行が出力装置に送信した場合に形式制御が不十分だと、CPU の処理時間の大部分が消費され、 SQL ステートメント自体の実際のパフォーマンスが誤って表示されます。

db2batch ベンチマーク・ツール: ベンチマーク・ツール (db2batch) は、インスタンスの sqllib ディレクトリーの bin サブディレクトリー内に提供されています。このツールは、ベンチマーク・プログラムの作成について、上記の諸点を多分に考慮に入れたものです。このツールは、フラット・ファイルまたは標準入力のいずれかから SQL ステートメントを読み取り、それらステートメントを動的に記述、作成し、応答セットを返します。さらに、このツールには柔軟性があり、応答セットのサイズを調節したり、応答セットから出力装置に送られる行数を制御したりできるようになっています。

さらに、経過時間、CPU とバッファー・プールの使用率、およびデータベース・モニターから収集されるその他の統計を含め、提供されるパフォーマンス関連情報のレベルを指定することもできます。一連の SQL ステートメントの時間を設定している場合は、db2batch を指定すると、パフォーマンスの結果を要約し、算術方式と幾何学方式の両方を提供します。 db2batch の呼び出し構文、およびオプションについては、 *コマンド解説書* を参照してください。コマンド行で db2batch -h と入力して、使用可能な構文およびオプションを調べることもできます。

以下は、db2batch と入力ファイル db2batch.sql とをどのように使用するかを示す例です。

```
-- db2batch.sql
-- ------------
--#SET PERF_DETAIL 3 ROWS_OUT 5

-- This query lists employees, the name of their department
-- and the number of activities to which they are assigned for
-- employees who are assigned to more than one activity less than
-- full-time.
--#COMMENT Query 1
select lastname, firstnme,
       deptname, count(*) as num_act
from employee, department, emp_act
where employee.workdept = department.deptno and
      employee.empno = emp_act.empno and
      emp_act.emptime < 1
group by lastname, firstnme, deptname
having count(*) > 2;
--#SET PERF_DETAIL 1 ROWS_OUT 5
--#COMMENT Query 2
select lastname, firstnme,
       deptname, count(*) as num_act
from employee, department, emp_act
where employee.workdept = department.deptno and
      employee.empno = emp_act.empno and
      emp_act.emptime < 1
group by lastname, firstnme, deptname
having count(*) <= 2;
```

図 31. ベンチマーク入力ファイルのサンプル: db2batch.sql

次のベンチマーク・ツールの呼び出しを使用すると、

```
db2batch -d sample -f db2batch.sql
```

次の出力が生成されます。

```
--#SET PERF_DETAIL 3 ROWS_OUT 5
Query 1

Statement number: 1

select lastname, firstnme,
deptname, count(*) as num_act
from employee, department, emp_act
where employee.workdept = department.deptno and
employee.empno = emp_act.empno and
emp_act.emptime < 1
group by lastname, firstnme, deptname
having count(*) > 2
```

図 32. db2batch からの出力例 (第 1 部)

```
LASTNAME         FIRSTNME      DEPTNAME                        NUM_ACT
--------------------------------------------------------------------------
JEFFERSON        JAMES         ADMINISTRATION SYSTEMS                3
JOHNSON          SYBIL         ADMINISTRATION SYSTEMS                4
NICHOLLS         HEATHER       INFORMATION CENTER                    4
PEREZ            MARIA         ADMINISTRATION SYSTEMS                4
SMITH            DANIEL        ADMINISTRATION SYSTEMS                7
Number of rows retrieved is:        5
Number of rows sent to output is:   5
Elapsed Time is:           0.074      seconds
Locks held currently                        = 0
Lock escalations                            = 0
Total sorts                                 = 5
Total sort time (ms)                        = 0
Sort overflows                              = 0
Buffer pool data logical reads              = 13
Buffer pool data physical reads             = 5
Buffer pool data writes                     = 0
Buffer pool index logical reads             = 3
Buffer pool index physical reads            = 0
Buffer pool index writes                    = 0
Total buffer pool read time (ms)            = 23
Total buffer pool write time (ms)           = 0
Asynchronous pool data page reads           = 0
Asynchronous pool data page writes          = 0
Asynchronous pool index page reads          = 0
Asynchronous pool index page writes         = 0
Total elapsed asynchronous read time        = 0
Total elapsed asynchronous write time       = 0
Asynchronous read requests                  = 0
LSN Gap cleaner triggers                    = 0
Dirty page steal cleaner triggers           = 0
Dirty page threshold cleaner triggers       = 0
Direct reads                                = 8
Direct writes                               = 0
Direct read requests                        = 4
Direct write requests                       = 0
Direct read elapsed time (ms)               = 0
Direct write elapsed time (ms)              = 0
Rows selected                               = 5
Log pages read                              = 0
Log pages written                           = 0
Catalog cache lookups                       = 3
Catalog cache inserts                       = 3
Buffer pool data pages copied to ext storage    = 0
Buffer pool index pages copied to ext storage   = 0
Buffer pool data pages copied from ext storage  = 0
Buffer pool index pages copied from ext storage = 0
Total Agent CPU Time (seconds)              = 0.02
Post threshold sorts                        = 0
Piped sorts requested                       = 5
Piped sorts accepted                        = 5
```

図 33. db2batch からの出力例 (第 1 部)

```
--#SET PERF_DETAIL 1 ROWS_OUT 5
Query 2
Statement number: 2
select lastname, firstnme,
deptname, count(*) as num_act
from employee, department, emp_act
where employee.workdept = department.deptno and
employee.empno = emp_act.empno and
emp_act.emptime < 1
group by lastname, firstnme, deptname
having count(*) <= 2
LASTNAME         FIRSTNME      DEPTNAME                       NUM_ACT
------------------------------------------------------------------------
GEYER            JOHN          SUPPORT SERVICES                     2
GOUNOT           JASON         SOFTWARE SUPPORT                     2
HAAS             CHRISTINE     SPIFFY COMPUTER SERVICE DIV.         2
JONES            WILLIAM       MANUFACTURING SYSTEMS                2
KWAN             SALLY         INFORMATION CENTER                   2
Number of rows retrieved is:        8
Number of rows sent to output is:   5
Elapsed Time is:           0.037      seconds
Summary of Results
==================
                Elapsed             Agent CPU         Rows      Rows
Statement #     Time (s)            Time (s)          Fetched   Printed
1                     0.074               0.020         5         5
2                     0.037       Not Collected         8         5
Arith. mean     0.055
Geom.  mean     0.052
```

図 34. db2batch からの出力例 (第 2 部)

上記のサンプル出力には、データベース・システム・モニターによって返される特定のデータ要素が含まれています。これらのモニター要素の詳細については、システム・モニター *手引きおよび解説書* を参照してください。

次の例 (UNIX の場合) では、要約表だけが作成されます。

```
db2batch -d sample -f db2batch.sql -r /dev/null,
```

要約表だけを作成します。 `-r` オプションを使用すると、 `outfile1` は `/dev/null` で置き換えられ、 `outfile2` (要約表だけが含まれている) は空になります。そのため、 `db2batch` は次のような出力を画面に送信します。

```
Summary of Results
==================
                Elapsed             Agent CPU         Rows      Rows
Statement #     Time (s)            Time (s)          Fetched   Printed
1                    0.074                0.020          5         5
2                    0.037       Not Collected           8         5
Arith. mean     0.055
Geom.  mean     0.052
```

図 35. db2batch からの出力例 -- 要約表のみ

このベンチマーク・ツールにも CLI オプションがあります。このオプションで、キャッシュ・サイズを指定することができます。次の例では、キャッシュ・サイズが 30 ステートメントの CLI モードで db2batch が実行されます。

```
db2batch -d sample -f db2batch.sql -cli 30
```

| リモートで db2batch を実行することもできます。

```
-f <filename>
```

| または

```
-o <options>
```

| というベンチマーク・ツールのコマンド・パラメーターのいずれかを使用する場合、次のようになります。

- 制御オプション

  ```
  perf_detail
  ```

  および

  ```
  -p <perf_detail>
  ```

  (戻されるパフォーマンス情報のレベルを指定するために) 1 より大きい値が設定される場合、リモートでの実行中にはサポートされません。

- 制御オプション

  ```
  perf_detail
  ```

  および

  ```
  -p <perf_detail>
  ```

  (戻されるパフォーマンス情報のレベルを指定するために) 1 より大きい値が設定される場合、 Windows 3.x または DOS プラットフォーム版の DB2 ユニバーサル・データベースでは無効です。

| これら 2 項目以外の場合、

```
perf_detail
```

| および

| `-p <perf_detail>`

| 制御オプション値はサポートされ、全 DB2 ユニバーサル・データベース・プラットフォームで有効です。

ベンチマーク・テストの実行

| あるタイプのデータベース・ベンチマークでは、構成パラメーターを選択し、最大の効果を得るまで、そのパラメーターに異なるさまざまな値を使ってテストを実行します。単一テストでは、同じパラメーター値を使ってアプリケーションを何度か繰り返して (たとえば 20 または 30 回) 実行するようにし、平均時間を調べます。これにより、パラメーター変更の効果をよりよく知ることができます。

ベンチマークを実行するときは、最初の繰り返し (ウォームアップ実行) を、後続の繰り返し (通常実行) とは別個のケースであると見なすようにします。これが必要なのは、ウォームアップ実行の結果には、いくつかの起動活動 (バッファー・プールの初期設定など) が含まれるためです。このため、ウォームアップ実行は、通常実行より多少長くかかります。ウォームアップ実行から得られる情報は、現実的に有効である*可能性はあります* が、統計的には無効なものです。したがって、パラメーター値の特定の集合に関して平均時間や CPU を計算するときは、通常実行の結果を使用してください。

| ベンチマークのウォームアップ実行を作成するために「パフォーマンス構成 (Performance Configuration)」ウィザードを使用することができます。「パフォーマンス構成 (Performance Configuration)」ウィザードの一部として尋ねられる質問に応答することによって、ベンチマーク作業の通常実行において、環境の構成を調整するときに考慮すべき内容について考えることができます。「パフォーマンス構成 (Performance Configuration)」ウィザードを使用するためには、 `db2cc` を入力してコントロール・センターにアクセスし、そこから進みます。

個々の照会を使用してベンチマークを行っている場合は、直前の照会の影響が最小になっていることを確認する必要があります。バッファー・プールをフラッシュするとこの状態になります。つまり、(照会とは無関係な) 多数のページを読み取ってバッファー・プールに入れるということを行います。

パラメーター値の単一の集合での繰り返しを完了したら、単一のパラメーターを変更することができます。しかし、繰り返しから次の繰り返しまでの間に、以下に示すタスクを実行して、ベンチマーク環境がその元の状態に復元されるようにしてください。

- アプリケーション・データとデータベース・マネージャーの統計を、それらの元の状態に戻します。カタログ統計がテストにより更新された場合は、その統計については同じ値が繰り返し使用されるようにしてください。テストで使用されたデータがテスト中に更新される場合は、それが一貫したものでなければなりません。そのためには、次のようにします。

  - RESTORE ユーティリティーを使用して、データベース全体を復元します。データベースのバックアップ・コピーはその直前の状況になり、次回のテストのために用意が整った状態になります。
  - IMPORT または LOAD ユーティリティーを使って、エクスポートされたデータ・コピーを復元します。この方法では、影響を受けたデータだけを復元できます。REORG および RUNSTATS のユーティリティーは、このデータが入った表と索引に対して実行するようにしてください。
- アプリケーションをデータベースに再バインドすることによって、アプリケーションを元の状態に戻します。

以下は、OS/2 上でベンチマーク・テストを行うための追加の考慮事項です。

- シナリオの途中でページングが起きた場合は、 SWAPPER.DAT が元のサイズに戻っているかどうか確認してください。
- 必要に応じて、システムをリブートし、反復が可能になるようにします。

ベンチマーク・プログラムからの出力には、各テストの ID、プログラム実行の繰り返し回数、ステートメント番号、および実行時間が含まれるようにします。一連の測定の後ベンチマーク結果の要約は、次のように表示されます。

```
Test     Iter.   Stmt    Timing         SQL Statement
Numbr    Numbr   Numbr   (hh:mm:ss.ss)
 002      05      01     00:00:01.34    CONNECT TO SAMPLE
 002      05      10     00:02:08.15    OPEN cursor_01
 002      05      15     00:00:00.24    FETCH cursor_01
 002      05      15     00:00:00.23    FETCH cursor_01
 002      05      15     00:00:00.28    FETCH cursor_01
 002      05      15     00:00:00.21    FETCH cursor_01
 002      05      15     00:00:00.20    FETCH cursor_01
 002      05      15     00:00:00.22    FETCH cursor_01
 002      05      15     00:00:00.22    FETCH cursor_01
 002      05      20     00:00:00.84    CLOSE cursor_01
 002      05      99     00:00:00.03    CONNECT RESET
```

図 36. ベンチマークの結果の例

注: 上記レポートのデータは、例示目的でのみ示したものです。測定された結果を表すものでは**ありません**。

このレポートのデータを調べると、CONNECT (ステートメント 01) に 1.34 秒、OPEN CURSOR (ステートメント 10) に 2 分 8.15 秒かかり、 FETCHES (ステートメント 15) は 7 行を戻してその最大の遅延が .28 秒で、 CLOSE CURSOR (ステートメント 20) に .84 秒、および CONNECT RESET (ステートメント 99) に .03 秒かかっていることが分かります。

区切り付き ASCII 形式でデータを出力すると、後でそれを統計分析の目的でデータベース表やスプレッドシートにインポートするのに便利です。

ベンチマーク・レポートのサンプル出力は、次のようになります。

```
      PARAMETER        VALUES FOR EACH BENCHMARK TEST
      TEST NUMBER      001     002     003     004     005
      locklist         63      63      63      63      63
>>    buffpage         1000    1175    1250    1325    1400     <<
      maxappls         8       8       8       8       8
      applheapsz       48      48      48      48      48
      dbheap           128     128     128     128     128
      sortheap         256     256     256     256     256
      maxlocks         22      22      22      22      22
      stmtheap         1024    1024    1024    1024    1024
      SQL STMT         AVERAGE TIMINGS (seconds)
        01             01.34   01.34   01.35   01.35   01.36
        10             02.15   02.00   01.55   01.24   01.00
        15             00.22   00.22   00.22   00.22   00.22
        20             00.84   00.84   00.84   00.84   00.84
        99             00.03   00.03   00.03   00.03   00.03
```

図 37. ベンチマークのタイミング・レポートの例

注: 上記レポートのデータは、例示目的でのみ示したものです。測定された結果を表すものでは**ありません**。

この例のデータを調べると、 *buffpage* パラメーターの変更によって OPEN CURSOR の時間が 2.15 秒から 1.00 秒に連続的に低くなっていくことがわかります。 (前提として、バッファー・プールは 1 つだけで、そのサイズ (NPAGES) は -1 に設定されています。すなわち、バッファー・プールのサイズは、 *buffpage* パラメーターによって制御されます。)

まとめると、データベース・アプリケーションのベンチマークは、次のステップまたは繰り返しを行います。

ステップ 1

データベースおよびデータベース・マネージャーの調整パラメーターは、それらの**デフォルト値**のままにしておきます。ただし、次のものは例外です。

- テストの作業負荷と目標にとって重要なパラメーター。 (ベンチマーク・テストを実行しても、すべてのパラメーターを調整する時間はほとんどないので、一部のパラメーターには最適と推察される値を使用して開始し、そこから出発して調整を行うことができます。)
- アプリケーションの単体テストとシステム・テストのときに決められたログ・サイズ。 (詳細については、415ページの『ログ・ファイルのサイズ (logfilsiz)』を参照してください。)
- アプリケーションを実行可能にするために変更しなければならないパラメーター (つまり、ステートメント・ヒープのメモリーを超えたなどのイベントから負の SQL 戻りコードが戻されることがないようにするために必要な変更)。

この初期ケースに関して一連の繰り返しを実行し、平均時間、または CPU を計算します。

ステップ 2

テストする調整パラメーターを 1 つだけ選択し、その値を変更します。

ステップ 3

別の一連の繰り返しを実行し、その平均時間、または CPU を計算します。

ステップ 4

ベンチマーク・テストの結果にしたがって、次のいずれかを実行します。

- パフォーマンスが向上した場合は、同じパラメーターの値を変更して、ステップ 3 に戻ります。最適値が示されるまで、このパラメーターを変更し続けます。
- パフォーマンスが低下したか、または変わらなかった場合は、パラメーターを前の値に戻してステップ 2 に戻り、新しいパラメーターを選択します。すべてのパラメーターをテストするまで、この手順を繰り返します。

  注: パフォーマンス結果をグラフ化する予定であれば、曲線が上がり始めるかまたは下がり始める点を探すようにしてください。

ベンチマーク・テストのためのドライバー・プログラムを作成することもできます。そのドライバー・プログラムは、REXX などの言語を使用して作成するか、または UNIX ベースのプラットフォームの場合は、シェル・スクリプトを使用して作成できます。

このドライバー・プログラムはベンチマーク・プログラムを実行し、プログラムに適切なパラメーターを渡し、テストを複数回繰り返し、環境を一貫した状態に復元し、新しいパラメーター値を使って次のテストを設定し、さらにテスト結果を収集 / 統合します。これらのドライバー・プログラムは非常に柔軟性に富んでおり、一連のベンチマーク・テスト全体を実行し、結果を分析してから、さらに所定のテストの最終パラメーター値および最適パラメーター値のレポートを作成する、といったことも可能です。

第13章 DB2 の構成

構成パラメーターは、データベースまたはデータベース管理システムの操作特性に影響を及ぼす値です。

データベース・マネージャー構成パラメーターは、サーバーとクライアントに存在します。ただし、クライアントで設定できるのは、一部のデータベース・マネージャー構成パラメーターだけです。これらのパラメーターは、サーバー上で設定できるデータベース管理構成パラメーターのサブセットです。ご使用の DB2 ユニバーサル・データベース製品によっては、構成パラメーターに関して特定の論点があります。たとえば、DB2 エンタープライズ拡張エディションでは、 1 つのデータベース・マネージャー構成ファイルが、インスタンスにあるすべてのデータベース区画サーバーで共用されます。また、各データベース区画には、独自のデータベース構成ファイルがあります。

DB2 は、広範囲の調整パラメーターと構成パラメーターで設計されています。これらのパラメーターは、次の 2 つの一般的なカテゴリーに分けることができます。

- 333ページの『データベース・マネージャー・パラメーター』
- 340ページの『データベース・パラメーター』

個々のパラメーターに関する説明がありますが、その他に次のトピックでは、構成パラメーターから大きく影響を受けるパラメーターについて知ることができます。

- 332ページの『構成パラメーターの調整』
- 346ページの『機能別のパラメーター詳細』 (各関数域ごとに、構成パラメーターの独自のリストがあります。)
- 501ページの『付録A. DB2 レジストリー変数と環境変数』

  パフォーマンス関連の構成パラメーターに加えて、個々のプラットフォームのためのパフォーマンス関連の環境変数またはレジストリー変数があり、この使用についても考慮する必要があります。
- 237ページの『第8章 操作上のパフォーマンス』
- 317ページの『第12章 ベンチマーク・テスト』

まず、 335ページの表17 および 342ページの表19 のパラメーターの要約を一読し、次に、使用する作業環境に最大の利点をもたらすパラメーターの説明と調整に的を絞ってください。

構成パラメーターの調整

データベース・マネージャーは、パラメーターのデフォルト値に基づいてディスク・スペースおよびメモリーを割り振ります。この割り振りでは、ある程度の必要は満たすことができるかもしれませんが、デフォルト値を使用して最高のパフォーマンスを引き出すことはできない場合もあります。

デフォルト値は、比較的メモリーが小さいマシンに合わせて設定されており、データベース・サーバー専用に設定されています。したがって、次の環境では、デフォルト値を修正する必要があります。

- データベースが大きい
- 接続の数が多い
- 特定のアプリケーションでハイ・パフォーマンスが求められている
- 固有な照会またはトランザクション・ロードまたはタイプを使用する
- さまざまなマシン構成または使用法を使う

それぞれのトランザクション処理環境は、ある 1 つの面または複数の面において固有です。デフォルト構成を使用した場合、このような相違点によりデータベース・マネージャーのパフォーマンスが大きな影響を受けることがあります。このため、ユーザーの環境に合わせて構成を調整するよう強くお勧めします。

アプリケーションやユーザーのタイプが異なると、応答の所要時間や予想される結果も異なります。アプリケーションといっても、単純なデータ入力項目画面から、 1 つの作業単位で複数の表にアクセスする複合 SQL ステートメントで成る戦略アプリケーションまで、幅広いタイプがあります。たとえば、電話による顧客サービス・アプリケーションと、バッチの報告書作成アプリケーションとでは応答の所要時間はかなり違ってきます。

ユーザーのアプリケーションにベンチマークを付け、構成パラメーターの調整を行う場合に、関連する他のトピックも参照できます。

- 333ページの『データベース・マネージャー・パラメーター』
- 340ページの『データベース・パラメーター』
- 346ページの『機能別のパラメーター詳細』 (各関数ごとに、構成パラメーターのリストがあります。)
- 237ページの『第8章 操作上のパフォーマンス』
- 317ページの『第12章 ベンチマーク・テスト』
- *システム・モニター 手引きおよび解説書* の中のデータベース・システム・モニター要素の説明

データベース・マネージャー・パラメーター

データベース・マネージャーのパラメーターは、db2systm というファイルに保管されます。このファイルは、データベース・マネージャーのインスタンス作成時に作成されます。 UNIX ベースの環境では、このファイルはデータベース・マネージャーのインスタンス用サブディレクトリー sqllib にあります。その他のすべての環境では、このファイルは、デフォルト解釈として、 sqllib ディレクトリーのインスタンス用サブディレクトリーに入っています。 DB2INSTPROF 変数が設定されている場合は、このファイルは、 DB2INSTPROF 変数が指定するディレクトリーの instance サブディレクトリーに入っています。

区分データベース環境では、このファイルは、どのデータベース区画サーバーも同じファイルにアクセスすることができるように、ファイル共用システムに常駐します。データベース・マネージャーの構成は、すべてのデータベース区画サーバーで同じです。

パラメーターの多くは、データベース・マネージャーの単一のインスタンスに割り振られるシステム・リソースの量を制御したり、あるいは、環境上の考慮事項に基づいて、データベース・マネージャーのセットアップや異なる通信サブシステムを構成します。その他に、情報提供だけを目的とした、変更不能のパラメーターがあります。これらのすべてのパラメーターには、データベース・マネージャーのインスタンスに保管されているシングル・データベースとは関係なくグローバル適用度があります。

db2systm ファイルは直接編集できません。したがって、提供されている API か、または API を呼び出すツールを使って、変更や表示を行ってください。

警告: この製品が提供している以外の方法を使用してこのファイルを編集すると、システムが使用不能になることがあります。 DB2 で言及またはサポートされていない方法では、このファイルを変更しないよう**強くお勧めします**。

次の方法のいずれかを用いて、データベース・マネージャー構成パラメーターのリセット、更新、または表示を行うことができます。

- DB2 コントロール・センターを使用する。 DB2 コントロール・センターの「インスタンスの構成 (Configure Instance)」ノートブックを使用すると、クライアントかサーバーのいずれかで、データベース・マネージャー構成パラメーターを設定することができます。また、DB2 コントロール・センターは、「パフォーマンス構成 (Performance Configuration)」ウィザードも提供しており、サーバー上の構成パラメーターの値をこれで変更することができます。このウィザードは、データベースに実行されるトランザクションのタイプや作業負荷などに関する一連の質問に対する応答に基づいて、パラメーターに値を生成します。これらのインターフェースの使用法については、コントロール・センターで利用できるオンライン・ヘルプを参照してください。
- コマンド行プロセッサーを使用する。設定を変更するためのコマンドを、す早くかつ容易に入力することができます。次のコマンドの詳細については、*コマンド解説書* を参照してください。

- GET DATABASE MANAGER CONFIGURATION (または GET DBM CFG)
- UPDATE DATABASE MANAGER CONFIGURATION (または UPDATE DBM CFG)
- RESET DATABASE MANAGER CONFIGURATION (または RESET DBM CFG)

- アプリケーション・プログラミング・インターフェース (API) を使用する。 API は、アプリケーションから簡単に呼び出すことができます。詳しくは、*管理 API 解説書* を参照してください。
- クライアント構成アシスタントを使用する。クライアント構成アシスタントは、クライアント上の構成パラメーターの設定にのみ使用できます。

パラメーターを変更した後で、新しいパラメーター値を反映させるために、データベース・マネージャーを一度停止 (db2stop) してから再始動 (db2start) する必要があります。クライアントの場合、データベース・マネージャー構成パラメーターの変更は、クライアントがサーバーに次回接続したときに反映されます。新しいパラメーター値は実際にはすぐに反映されませんが、パラメーターの設定値を表示すると、必ず最新の更新内容が表示されます。

注: *dft_monswitches* パラメーターの値を更新する場合、データベース・マネージャーを再始動する必要はありません。このパラメーターは、値の変更時に自動的に更新されます。

データベース・マネージャー構成パラメーターの要約

次の表は、データベース・サーバーのデータベース・マネージャー構成ファイルに入っているパラメーターをリストしています。データベース・マネージャー構成パラメーターを変更する際は、各パラメーターの詳細情報も考慮に入れてください。デフォルトを含む個々の操作環境情報については、それぞれのパラメーターの説明に述べてあります。

次の表の『パフォーマンスへの影響』の欄は、各パラメーターのシステム性能に与える影響の重大度を示しています。この欄について完全に正確な情報を記述することはできないので、大まかな情報として扱ってください。

- **高** - パフォーマンスへの影響が大きいことを示します。これらのパラメーターの値は注意して検討してください。もちろん、デフォルトを受け入れるのが最善の場合もあります。
- **中** - パフォーマンスへの影響が多少あることを示します。これらのパラメーターをどの程度調整する必要があるかは、ユーザーの環境および必要によって異なります。
- **低** - パフォーマンスへの影響が低いことを示します。
- **なし** - パフォーマンスへの直接の影響がないことを示します。これらのパラメーターは、パフォーマンスの点で調整する必要はありませんが、通信サポートなど、システム構成の他の面では非常に重要になることがあります。

表 17. 構成可能なデータベース・マネージャー構成パラメーター

パラメーター	パフォーマンスへの影響	追加情報
agentpri	高	403ページの『エージェントの優先順位 (agentpri)』
agent_stack_sz	低	372ページの『エージェント・スタック・サイズ (agent_stack_sz)』
aslheapsz	高	376ページの『アプリケーション・サポート層ヒープ・サイズ (aslheapsz)』
audit_buf_sz	高	386ページの『監査バッファー・サイズ (audit_buf_sz)』
authentication	低	492ページの『認証タイプ (authentication)』
backbufsz	中	355ページの『デフォルトのバックアップ・バッファー・サイズ (backbufsz)』
catalog_noauth	なし	493ページの『権限なしで許可されるカタログ作成 (catalog_noauth)』
comm_bandwidth	中	480ページの『通信帯域幅 (comm_bandwidth)』
conn_elapse	中	467ページの『接続経過時間 (conn_elapse)』
cpuspeed	低 (注を参照)	481ページの『CPU 速度 (cpuspeed)』
datalinks	低	444ページの『データ・リンク・サポートの使用可能化 (datalinks)』
dft_account_str	なし	486ページの『デフォルトのチャージバック会計 (dft_account_str)』
dft_client_adpt	なし	462ページの『デフォルトのクライアント・アダプター番号 (dft_client_adpt)』
dft_client_comm	なし	461ページの『デフォルト・クライアント通信プロトコル (dft_client_comm)』
dft_monswitches • *dft_mon_bufpool* • *dft_mon_lock* • *dft_mon_sort* • *dft_mon_stmt* • *dft_mon_table* • *dft_mon_uow*	中	478ページの『デフォルト データベース・システム・モニター・スイッチ (dft_monswitches)』
dftdbpath	なし	494ページの『デフォルトのデータベース・パス (dftdbpath)』

表 17. 構成可能なデータベース・マネージャー構成パラメーター (続き)

パラメーター	パフォーマンスへの影響	追加情報
diaglevel	低	475ページの『診断エラーのキャプチャー・レベル (diaglevel)』
diagpath	なし	476ページの『診断データのディレクトリー・パス (diagpath)』
dir_cache	中	384ページの『ディレクトリー・キャッシュ・サポート (dir_cache)』
dir_obj_name	なし	460ページの『DCE ネームスペース内のオブジェクト名 (dir_obj_name)』
dir_path_name	なし	459ページの『DCE ネームスペース内のディレクトリー・パス名 (dir_path_name)』
dir_type	なし	458ページの『ディレクトリー・サービス・タイプ (dir_type)』
discover	中	463ページの『ディスカバリー・モード (discover)』
discover_comm	低	465ページの『通信プロトコルの探索ディスカバリー (discover_comm)』
discover_inst	低	466ページの『サーバー・インスタンスのディスカバリー (discover_inst)』
dos_rqrioblk	高	380ページの『DOS リクエスター入出力ブロック・サイズ (dos_rqrioblk)』
drda_heap_sz	低	370ページの『DRDA ヒープ・サイズ (drda_heap_sz)』
fcm_num_anchors	高	467ページの『FCM メッセージ・アンカーの個数 (fcm_num_anchors)』
fcm_num_buffers	高	468ページの『FCM バッファー数 (fcm_num_buffers)』
fcm_num_connect	高	469ページの『FCM 接続項目数 (fcm_num_connect)』
fcm_num_rqb	高	470ページの『FCM 要求ブロック数 (fcm_num_rqb)』
federated	中	487ページの『連合データベース・システムのサポート (federated)』
fileserver	なし	456ページの『IPX/SPX ファイル・サーバー名 (fileserver)』
indexrec	中	428ページの『索引再作成時刻 (indexrec)』

表 17. 構成可能なデータベース・マネージャー構成パラメーター (続き)

パラメーター	パフォーマンスへの影響	追加情報
initdari_jvm	中	413ページの『JVM による DARI プロセスの初期化 (initdari_jvm)』
intra_parallel	高	474ページの『区画内並列処理機能の使用可能化 (intra_parallel)』
ipx_socket	なし	457ページの『IPX/SPX ソケット番号 (ipx_socket)』
java_heap_sz	高	386ページの『Java インタープリター・ヒープの最大サイズ (java_heap_sz)』
jdk11_path	なし	487ページの『Java Development Kit 1.1 インストール・パス (jdk11_path)』
keepdari	中	411ページの『DARI プロセス保持標識 (keepdari)』
maxagents	中	405ページの『エージェントの最大数 (maxagents)』
maxcagents	中	406ページの『並行エージェントの最大数 (maxcagents)』
max_connretries	中	471ページの『ノード接続の再試行 (max_connretries)』
max_coordagents	中	407ページの『調整エージェントの最大数 (max_coordagents)』
maxdari	中	412ページの『DARI プロセスの最大数 (maxdari)』
max_logicagents	中	408ページの『論理エージェントの最大数 (max_logicagents)』
max_querydegree	高	473ページの『照会の最大並列処理度 (max_querydegree)』
max_time_diff	中	471ページの『ノード間の最大の時刻差 (max_time_diff)』
maxtotfilop	中	403ページの『オープンするファイルの最大合計数 (maxtotfilop)』
min_priv_mem	中	373ページの『コミットされる私用メモリーの最小値 (min_priv_mem)』
mon_heap_sz	低	382ページの『データベース・システム・モニター・ヒープ・サイズ (mon_heap_sz)』
nname	なし	453ページの『NetBIOS ワークステーション名 (nname)』

表 17. 構成可能なデータベース・マネージャー構成パラメーター (続き)

パラメーター	パフォーマンスへの影響	追加情報
notifylevel	低	477ページの『通知レベル (notifylevel)』
numdb	低	481ページの『並行活動データベースの最大数 (numdb)』
num_initagents	中	410ページの『プール内の初期エージェント数 (num_initagents)』
num_initdaris	中	414ページの『プール内の分離 DARI プロセスの初期数 (num_initdaris)』
num_poolagents	高	409ページの『エージェント・プール・サイズ (num_poolagents)』
objectname	なし	456ページの『IPX/SPX DB2 サーバー・オブジェクト名 (objectname)』
priv_mem_thresh	中	374ページの『私用メモリーしきい値 (priv_mem_thresh)』
query_heap_sz	中	368ページの『照会ヒープ・サイズ (query_heap_sz)』
restbufsz	中	356ページの『デフォルトの復元バッファー・サイズ (restbufsz)』
resync_interval	なし	434ページの『トランザクション再同期間隔 (resync_interval)』
route_obj_name	なし	460ページの『経路指定情報オブジェクト名 (route_obj_name)』
rqrioblk	高	379ページの『クライアント入出力ブロック・サイズ (rqrioblk)』
sheapthres	高	364ページの『分類ヒープのしきい値 (sheapthres)』
spm_log_file_sz	低	436ページの『同期点管理機能ログ・ファイル・サイズ (spm_log_file_sz)』
spm_log_path	中	435ページの『同期点管理機能ログ・ファイル・パス (spm_log_path)』
spm_max_resync	低	437ページの『同期点管理機能再同期エージェント限界 (spm_max_resync)』
spm_name	なし	436ページの『同期点管理機能名 (spm_name)』
ss_logon	なし	495ページの『DB2START/DB2STOP に必要な LOGON (ss_logon)』

表 17. 構成可能なデータベース・マネージャー構成パラメーター (続き)

パラメーター	パフォーマンスへの影響	追加情報
start_stop_time	低	472ページの『タイムアウトの開始と停止 (start_stop_time)』
svcename	なし	454ページの『TCP/IP サービス名 (svcename)』
sysadm_group	なし	488ページの『システム運用管理権限グループ名 (sysadm_group)』
sysctrl_group	なし	490ページの『システム制御権限グループ名 (sysctrl_group)』
sysmaint_group	なし	491ページの『システム保守権限グループ名 (sysmaint_group)』
tm_database	なし	433ページの『トランザクション・マネージャー・データベース名 (tm_database)』
tp_mon_name	なし	483ページの『トランザクション・プロセッサー・モニター名 (tp_mon_name)』
tpname	なし	455ページの『APPC トランザクション・プログラム名 (tpname)』
trust_allclnts	なし	496ページの『全クライアントの承認 (trust_allclnts)』
trust_clntauth	なし	497ページの『承認クライアントの認証 (trust_clntauth)』
udf_mem_sz	低	370ページの『UDF 共用メモリー・セット・サイズ (udf_mem_sz)』
注: *cpuspeed* パラメーターはパフォーマンスに大きく影響しますが、ごくまれな状況を除いて、パラメーターの説明に示されているデフォルト値を用いるようにしてください。		

表 18. 情報提供用のデータベース・マネージャー構成パラメーター

パラメーター	追加情報
nodetype	485ページの『マシン・ノード・タイプ (nodetype)』
release	439ページの『構成ファイルのリリース・レベル (release)』

データベース・パラメーター

個々のデータベース用のパラメーターは、SQLDBCON という構成ファイルに保管されます。このファイルは、データベースの他の制御ファイルとともに SQLnnnnn ディレクトリーに保管されています。 nnnnn はこのデータベースの作成時に割り当てられた番号です。 (このディレクトリーの場所については、 *管理の手引き: 計画* の『データベースの物理ディレクトリー』を参照してください。) 各データベースにはそれぞれ構成ファイルがあり、パラメーターの多くはそのデータベースに割り振られるリソースの量を指定します。ファイルには、データベースの状況を示すフラグと共に、記述情報も入っています。

SQLDBCON ファイルは直接編集できないので、提供されている API、または API を呼び出すツールを使って、変更や表示を行ってください。

警告: DB2 が提供している以外の方法を使用してこのファイルを編集すると、データベースが使用不能になることがあります。 DB2 で言及またはサポートされていない方法では、このファイルを変更しないよう**強くお勧めします**。

次の 3 つの方法のいずれかを用いて、データベース構成パラメーターのリセット、更新、または表示を行うことができます。

- コントロール・センターを使用する。 DB2 コントロール・センターは、「データベースの構成 (Configure Database)」ノートブックと 「パフォーマンス構成 (Performance Configuration)」ウィザードを提供しており、構成パラメーターの値をこれで変更することができます。このウィザードは、データベースに実行されるトランザクションのタイプや作業負荷などに関する一連の質問に対する応答に基づいて、パラメーターに値を生成します。これらのインターフェースの使用法については、コントロール・センターで利用できるオンライン・ヘルプを参照してください。

  区分データベース環境では、 SQLDBCON ファイルは各データベース区画ごとに存在しています。コントロール・センターの「データベースの構成 (Configure Database)」ノートブックは、コントロール・センターのツリー・ビューにあるデータベース・オブジェクトから立ち上げられた場合に、全区画上の値を変更します。データベース区画オブジェクトから立ち上げられた場合は、その区画の値だけを変更します。 (ただし、構成パラメーター値はすべての区画で同じにするようお勧めします。)

  注: 「パフォーマンス構成 (Performance Configuration)」ウィザードは、区分データベース環境では使用できません。

- コマンド行プロセッサーを使用する。設定を変更するためのコマンドを、す早くかつ容易に入力することができます。次のコマンドの詳細については、*コマンド解説書* を参照してください。
  - GET DATABASE CONFIGURATION (または GET DB CFG)
  - UPDATE DATABASE CONFIGURATION (または UPDATE DB CFG)
  - RESET DATABASE CONFIGURATION (または RESET DB CFG)

- アプリケーション・プログラミング・インターフェース (API) を使用する。 API は、ホスト言語プログラムから簡単に呼び出すことができます。詳しくは、*管理 API 解説書* を参照してください。

多くの変更可能なパラメーターを更新しても、アプリケーションがデータベースに接続している間は反映されません。まずすべてのアプリケーションをデータベースから切断する必要があります。 (データベースが活動化している場合は、いったん非活動化し、それから活動化し直す必要があります。) その後は、データベースに最初に接続した時点で、変更が反映されます。
newlogpath 、 *logfilsiz* および *logprimary* などのパラメーターの場合、スペースの割り振りに関係したオーバーヘッドが生じるため、設定を変更しても、それが反映されるまでにかなりの時間がかかることがあります。データベースにテスト接続すると、テスト接続時に設定が変更され、他のユーザーがオーバーヘッドの影響を受けないようにすることができます。ここで述べたオーバーヘッドに関心がある場合は、 ACTIVATE DATABASE コマンドの使用について考えてください。これについては、 *コマンド解説書* に解説してあります。

注: *mincommit* パラメーターの値を更新する場合、データベースを切断する必要はありません。このパラメーターは、値の変更時に自動的に更新されます。

データベース構成パラメーターをいくつか変更しただけで、 SQL 最適化プログラムが選択するアクセス・プランに影響が出ます。これらのデータベース・パラメーターについては、 91ページの『照会最適化に影響する構成パラメーター』に説明してあります。これらのパラメーターのいずれかを変更した場合には、 SQL ステートメントに最適なアクセス・プランが使用されるように、アプリケーションの再バインドを考慮してください。 BIND コマンドについて詳しくは、 *コマンド解説書* を参照してください。

新しいパラメーター値は実際にはすぐに反映されませんが、パラメーターの設定値を表示すると、必ず最新の更新内容が表示されます。

注: ヘルプまたはその他の DB2 の資料では、データベース構成パラメーター (たとえば、*userexit*) の大多数の許容値が 『Yes』 か 『No』、あるいは 『On』 か 『Off』 であると記述されています。混乱しないように、『Yes』 は 『On』 と同じであり、 『No』 は 『Off』 と同じものと見なします。

データベース構成パラメーターの要約

次の表は、データベース構成ファイルに入っているパラメーターをリストしています。データベース構成パラメーターを変更する際は、パラメーターの詳細情報も考慮に入れてください。

次の表の『パフォーマンスへの影響』の欄は、各パラメーターのシステム性能に与える影響の重大度を示しています。この欄について完全に正確な情報を記述することはできないので、大まかな情報として扱ってください。

- **高** - パフォーマンスへの影響が大きいことを示します。これらのパラメーターの値は注意して検討してください。もちろん、デフォルトを受け入れるのが最善の場合もあります。
- **中** - パフォーマンスへの影響が多少あることを示します。これらのパラメーターをどの程度調整する必要があるかは、ユーザーの環境および必要によって異なります。
- **低** - パフォーマンスへの影響が低いことを示します。
- **なし** - パフォーマンスへの直接の影響がないことを示します。これらのパラメーターは、パフォーマンスの点で調整する必要はありませんが、通信サポートなど、システム構成の他の面では非常に重要になることがあります。

表 19. 構成可能なデータベース構成パラメーター

パラメーター	パフォーマンスへの影響	追加情報
app_ctl_heap_sz	中	361ページの『アプリケーション制御ヒープ・サイズ (app_ctl_heap_sz)』
applheapsz	中	367ページの『アプリケーション・ヒープ・サイズ (applheapsz)』
autorestart	低	427ページの『自動再始動可能 (autorestart)』
avg_appls	高	401ページの『活動アプリケーションの平均数 (avg_appls)』
buffpage	高 (活動時)	348ページの『バッファー・プール・サイズ (buffpage)』
catalogcache_sz	中	352ページの『カタログ・キャッシュ・サイズ (catalogcache_sz)』
chngpgs_thresh	高	392ページの『ページ変更しきい値 (chngpgs_thresh)』
copyprotect	なし	441ページの『コピー保護可能 (copyprotect)』
dbheap	中	351ページの『データベース・ヒープ (dbheap)』
dft_degree	高	449ページの『デフォルトの程度 (dft_degree)』
dft_extent_sz	中	397ページの『表スペースのデフォルト・エクステント・サイズ (dft_extent_sz)』
dft_loadrec_ses	中	429ページの『ロード・リカバリー・セッションのデフォルト数 (dft_loadrec_ses)』
dft_prefetch_sz	中	396ページの『デフォルト事前取り出しサイズ (dft_prefetch_sz)』
dft_queryopt	中	449ページの『デフォルトの照会最適化クラス (dft_queryopt)』
dft_refresh_age	中	450ページの『デフォルトの最新表示の期間 (dft_refresh_age)』

表 19. 構成可能なデータベース構成パラメーター (続き)

パラメーター	パフォーマンスへの影響	追加情報
dft_sqlmathwarn	なし	447ページの『算術例外の続行 (dft_sqlmathwarn)』
dir_obj_name	なし	460ページの『DCE ネームスペース内のオブジェクト名 (dir_obj_name)』
discover_db	中	463ページの『データベースのディスカバリー (discover_db)』
dlchktime	中	388ページの『デッドロック検査の時間間隔 (dlchktime)』
dl_expint	なし	442ページの『データ・リンク・アクセス・トークン満了間隔 (dl_expint)』
dl_num_copies	なし	443ページの『データ・リンクのコピー数 (dl_num_copies)』
dl_time_drop	なし	443ページの『ドロップ後のデータ・リンク時間 (dl_time_drop)』
dl_token	低	443ページの『データ・リンク・トークン・アルゴリズム (dl_token)』
dl_upper	なし	444ページの『大文字のデータ・リンク・トークン (dl_upper)』
dyn_query_mgmt	低	438ページの『動的 SQL 照会管理 (dyn_query_mgmt)』
estore_seg_sz	中	398ページの『拡張記憶域メモリー・セグメント・サイズ (estore_seg_sz)』
indexrec	中	428ページの『索引再作成時刻 (indexrec)』
indexsort	低い (345 ページの注を参照)	395ページの『索引分類フラグ (indexsort)』
locklist	高 (自動調整に影響するとき)	357ページの『ロック・リスト用最大ストレージ (locklist)』
locktimeout	中	390ページの『ロック・タイムアウト (locktimeout)』
logbufsz	高	353ページの『ログ・バッファー・サイズ (logbufsz)』
logfilsiz	中	415ページの『ログ・ファイルのサイズ (logfilsiz)』
logprimary	中	417ページの『1 次ログ・ファイルの数 (logprimary)』
logretain	低	425ページの『ログの保存可能 (logretain)』

表 19. 構成可能なデータベース構成パラメーター (続き)

パラメーター	パフォーマンスへの影響	追加情報
logsecond	中	419ページの『2 次ログ・ファイルの数 (logsecond)』
maxappls	中	399ページの『活動アプリケーションの最大数 (maxappls)』
maxfilop	中	402ページの『アプリケーションごとにオープンするデータベース・ファイルの最大数 (maxfilop)』
maxlocks	高 (自動調整に影響するとき)	389ページの『自動調整前のロック・リストの最大パーセント (maxlocks)』
mincommit	高	422ページの『グループ化するコミット数 (mincommit)』
min_dec_div_3	高	378ページの『10 進数除算の 3 の位取り (min_dec_div_3)』
newlogpath	低	419ページの『データベース・ログ・パスの変更 (newlogpath)』
num_db_backups	なし	430ページの『データベース・バックアップの数 (num_db_backups)』
num_estore_segs	中	398ページの『拡張記憶域メモリー・セグメントの数 (num_estore_segs)』
num_freqvalues	低	450ページの『頻度の高い値の保存数 (num_freqvalues)』
num_iocleaners	高	392ページの『非同期ページ・クリーナーの数 (num_iocleaners)』
num_ioservers	高	394ページの『入出力サーバー数 (num_ioservers)』
num_quantiles	低	451ページの『列変位数の数 (num_quantiles)』
pckcachesz	高	360ページの『パッケージ・キャッシュ・サイズ (pckcachesz)』
rec_his_retentn	なし	430ページの『リカバリー・ヒストリーの保存期間 (rec_his_retentn)』
seqdetect	高	395ページの『順次検出フラグ (seqdetect)』
softmax	中	423ページの『リカバリー範囲とソフト・チェックポイント間隔 (softmax)』
sortheap	高	363ページの『分類ヒープ・サイズ (sortheap)』
stat_heap_sz	低	368ページの『統計ヒープ・サイズ (stat_heap_sz)』
stmtheap	中	366ページの『ステートメント・ヒープ・サイズ (stmtheap)』

表 19. 構成可能なデータベース構成パラメーター (続き)

パラメーター	パフォーマンスへの影響	追加情報
trackmod	低	431ページの『変更されたページの追跡を可能にする (trackmod)』
tsm_mgmtclass	なし	431ページの『Tivoli Storage Manager 管理クラス (tsm_mgmtclass)』
tsm_nodename	なし	432ページの『Tivoli Storage Manager ノード名 (tsm_nodename)』
tsm_owner	なし	433ページの『Tivoli Storage Manager 所有者名 (tsm_owner)』
tsm_password	なし	432ページの『Tivoli Storage Manager パスワード (tsm_password)』
userexit	低	426ページの『ユーザー出口可能 (userexit)』
util_heap_sz	低	355ページの『ユーティリティー・ヒープ・サイズ (util_heap_sz)』
注: *indexsort* パラメーターをデフォルト以外の値に変更すると、索引作成のパフォーマンスにマイナスの影響が出ることがあります。このパラメーターには、必ずデフォルトを使用するようにしてください。		

表 20. 情報提供用のデータベース構成パラメーター

パラメーター	追加情報
backup_pending	445ページの『バックアップ保留標識 (backup_pending)』
codepage	441ページの『データベースのコード・ページ (codepage)』
codeset	440ページの『データベースのコード・セット (codeset)』
collate_info	441ページの『照合情報 (collate_info)』
country	440ページの『データベースの国別コード (country)』
database_consistent	445ページの『データベースの一貫性 (database_consistent)』
database_level	439ページの『データベース・リリース・レベル (database_level)』
log_retain_status	446ページの『ログ保存状況表示 (log_retain_status)』

表 20. 情報提供用のデータベース構成パラメーター (続き)

パラメーター	追加情報
loghead	421ページの『最初のアクティブ・ログ・ファイル (loghead)』
logpath	421ページの『ログ・ファイルの場所 (logpath)』
multipage_alloc	447ページの『複数ページ・ファイル割り振りの使用可能化 (multipage_alloc)』
numsegs	397ページの『SMS コンテナーのデフォルト数 (numsegs)』
release	439ページの『構成ファイルのリリース・レベル (release)』
restore_pending	446ページの『保留の復元 (restore_pending)』
rollfwd_pending	445ページの『ロールフォワード保留標識 (rollfwd_pending)』
territory	440ページの『データベースの領域 (territory)』
user_exit_status	446ページの『ユーザー出口状況表示 (user_exit_status)』

機能別のパラメーター詳細

次のセクションでは、さまざまな構成パラメーターについて理解し、調整するための、追加情報を詳細に記述します。次のように、個々のパラメーターについて機能または目的別に説明します。

- 347ページの『キャパシティー管理』
- 415ページの『ロギングおよびリカバリー』
- 438ページの『データベース管理』
- 453ページの『通信』
- 466ページの『区画データベース』
- 475ページの『インスタンス管理』

パラメーターごとに、次の情報について説明します。

構成タイプ

パラメーターの設定値がどの構成ファイルに入っているか。

- データベース・マネージャー (データベース・マネージャーのインスタンス、およびそのインスタンス中で定義されているすべてのデータベースが影響を受けます)

- データベース (特定のデータベースだけが影響を受けます)

パラメーター・タイプ

パラメーター値が変更可能かどうか。

- *構成可能*

  パラメーターは、ある範囲の値をとることができ、データベース管理者がアプリケーションについてもっている知識およびベンチマークの経験のいずれか、またはその両方に基づいて、パラメーターを調整することができる。

- *情報提供*

  これらのパラメーターを変更できるのはデータベース・マネージャーだけであり、データベースが作成された際の DB2 のリリース、または必要なバックアップが保留になっている指示などの情報を含むことができる。

キャパシティー管理

データベースとデータベース・マネージャーの両方のレベルで、システムのスループットを制御する構成パラメーターがいくつかあります。この種のパラメーターは、次のカテゴリーに分けることができます。

- 『データベース共用メモリー』
- 361ページの『アプリケーション共用メモリー』
- 363ページの『エージェント私用メモリー』
- 376ページの『エージェント / アプリケーション通信メモリー』
- 382ページの『データベース・マネージャー インスタンス・メモリー』
- 387ページの『ロック』
- 391ページの『入出力およびストレージ』
- 399ページの『エージェント』
- 411ページの『ストアード・プロシージャー (DARI)』

DB2 メモリー管理の紹介については、237ページの『DB2 でのメモリーの使用方法』を参照してください。

データベース共用メモリー

次のパラメーターは、システムで割り振られるデータベース・グローバル・メモリーを制御します。

- 348ページの『バッファー・プール・サイズ (buffpage)』
- 351ページの『データベース・ヒープ (dbheap)』

- 352ページの『カタログ・キャッシュ・サイズ (catalogcache_sz)』
- 353ページの『ログ・バッファー・サイズ (logbufsz)』
- 355ページの『ユーティリティー・ヒープ・サイズ (util_heap_sz)』
- 355ページの『デフォルトのバックアップ・バッファー・サイズ (backbufsz)』
- 356ページの『デフォルトの復元バッファー・サイズ (restbufsz)』
- 357ページの『ロック・リスト用最大ストレージ (locklist)』
- 360ページの『パッケージ・キャッシュ・サイズ (pckcachesz)』

データベース・グローバル・メモリーと、データベース・マネージャーで割り振られるそれ以外のメモリーとの関係については、 237ページの『DB2 でのメモリーの使用方法』を参照してください。

バッファー・プール・サイズ (buffpage)

構成タイプ
データベース

パラメーター・タイプ
構成可能

デフォルト[範囲]

UNIX 32 ビット・プラットフォーム
1 000 [2 〜 524 288]

UNIX 64 ビット・プラットフォーム
1 000 [2 〜 2 147 483 647]

OS/2 および Windows NT
250 [2 〜 524 288]

計測単位
ページ

割り振りタイミング
最初のアプリケーションがデータベースに接続したとき

解放タイミング
最後のアプリケーションがデータベースから切断したとき

関連パラメーター

- 392ページの『ページ変更しきい値 (chngpgs_thresh)』
- 351ページの『データベース・ヒープ (dbheap)』
- 392ページの『非同期ページ・クリーナーの数 (num_iocleaners)』

各データベースは、少なくとも 1 つのバッファー・プール (IBMDEFAULTBP、これはデータベースの作成時に作成される) を持ちますが、さらに多くのバッファー・プールを持つこともできます。すべてのバッファー・プールはグローバル・メモリーに常駐し、このデータベースを使用するすべてのアプリケーションが利用できます。メモリー

は、そのデータベースがあるマシンで割り振られます。バッファー・プールが十分に大きくて、メモリー内の必要なデータをすべて保持できる場合は、ディスク・アクティビティーはより少なくて済みます。逆に、バッファー・プールが十分大きくないと、データベース全体のパフォーマンスは著しく低下することがあり、アプリケーションで必要なデータを処理するためのディスク・アクティビティー (入出力) が増えるためにデータベース・マネージャーが入出力制約を受けることがあります。

buffpage パラメーターは、CREATE BUFFERPOOL または ALTER BUFFERPOOL ステートメントが NPAGES -1 を指定して実行されるときに、バッファー・プールのサイズを制御します。それ以外の場合は、*buffpage* パラメーターは無視され、バッファー・プールは NPAGES パラメーターの指定するページ数だけ作成されます。

バッファー・プールに *buffpage* パラメーターが活動中になっているかどうかを判別するには以下のようにしてください。

```
SELECT * from SYSCAT.BUFFERPOOLS.
```

NPAGES 値が -1 である各バッファー・プールは *buffpage* を使用します。

バッファー・プール・サイズと他のシステム・ユーザーのメモリー割り振りとの間には、トレードオフがあります。トランザクション率が高い複数ユーザーのサーバーではデータベース・サーバーのメモリー所要量が大きな意味を持つので、データベース・サーバーとファイル / 通信サーバーを別々のマシンに置くことがよくあります。

照会でニックネームにアクセスする場合、次のようなときには、バッファー・プール・サイズの増加を考慮に入れてください。

- 最適化プログラムが、ほとんどまたはすべての操作がローカル側で完了していると判断した場合。照会が処理されるとき、最適化プログラムは通常、可能な場合にデータ・ソースへの後入れ先出し操作を行います。たとえば、GROUP BY 演算子は通常、データ・ソースで評価されます。しかし、DB2 で表を実体化し、最小のコスト経路でローカル操作を実行することができます。これが当てはまるのは、 DB2 サーバー・ワークステーションがデータ・ソース・ワークステーションより強力な場合です。
- 分類操作がローカル側で完了した場合。ニックネームを含む照会は、DB2 の照合順序に従って分類されます。データ・ソースの照合順序が同じでない場合、すべての分類操作はローカル実行されます。

バッファー・プールは、最初のアプリケーションがデータベースに接続したとき、またはデータベースが明示的に活動化されたときに、すべて割り振られます。アプリケーションがデータベースにデータを要求すると、そのデータが含まれているページがディスクからバッファー・プールに転送されます。 (ディスク上の表では、データベース・データはページ単位で保管されていることに注意してください。) ページが変更され、次のことが起こった場合に限り、転送されたページがディスクに戻されます。

- すべてのアプリケーションがデータベースから切断された
- データベースが明示的に活動化された

- データベースが静止した (つまり、接続しているすべてのアプリケーションがコミットした)
- バッファー・プールに別のページを読み込むようそのスペースが要求された
- ページ・クリーナーが利用可能 (*num_iocleaners*) になっていて、データベース・マネージャーによって活動化された

推奨事項:

- *buffpage* 構成パラメーターを使用する代わりに、 CREATE BUFFERPOOL と ALTER BUFFERPOOL の SQL ステートメントを使用して、バッファー・プールとそのサイズを作成し変更することができます。
- 最適化プログラムは、このバッファー・プールのサイズを使ってアクセス・プランを立てます。このパラメーターを変更してから、アプリケーションの再バインドを考慮する必要があります (REBIND PACKAGE コマンドを使用)。
- すべてのバッファー・プールのサイズはパフォーマンスに大きな影響を与えるので、次の要素を考慮して、ページ・スワッピングが過度に起こらないようにしてください。
  - マシンにインストールされているメモリーの量
  - 同じマシンでデータベース・マネージャーと並行して実行している他のアプリケーションで必要なメモリー

  ページ・スワッピング は、アクセス中のページを保持できるだけの大きさのメモリーがないときに発生します。結果として、そのページは一時ディスク装置に書き込まれ (「スワップ」され)、他のページのためのスペースが確保されます。一時ディスク装置内のページが必要になると、そのページはメモリーに「逆スワップ」されます。
- 以下の場合は、マシン・メモリーの 75% までをデータベースに割り振ることができます。
  - 複数のユーザーがある
  - マシンはデータベース・サーバー専用になっている
  - 同一のデータおよび索引ページに、アクセスが何度も繰り返されている
  - マシンに 1 つのデータベースしかない
- 割り振られたバッファー・プール・ページごとに、データベース・ヒープ中のスペースの一部が内部制御構造用に使用されます。

  バッファー・プールの合計サイズが増える場合は、 *dbheap* も増やす必要があります。
- 統合環境で作業する際、データ・ソースの照合順序が DB2 の照合順序と一致する場合は、サーバー・オプション collating_sequence が一致を示すように設定されていることを確認します。つまり、collating_sequence オプションを 『Y』 に設定する必要があります。データ・ソースの照合順序と一致する特定の照合順序の連合データベースを作成できます。この方法を用いると、すべてのデータ・ソースが同じ照合順序を使用する場合、または、ほとんどあるいはすべての列関数が同じ照合順序を使用するデータ・ソースに向けられている場合は、パフォーマンスが高速になります。データ・ソースの照合順序が DB2 の照合順序と異なっている場合、 DB2 の照合順序に

依存しているほとんどの操作は、データ・ソースにおいてリモートに評価できません。このサーバー・オプションについて詳しくは、 *管理の手引き: インプリメンテーション* を参照してください。

データベース・システム・モニターを使用して、バッファー・プールのヒット率を計算し、バッファー・プールの調整を行うことができます。 *システム・モニター 手引きおよび解説書* を参照してください。

データベース・ヒープ (dbheap)

構成タイプ データベース

パラメーター・タイプ 構成可能

デフォルト[範囲]

UNIX 1200 [32 ～ 524 288]

OS/2 および Windows NTローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー 600 [32 ～ 524 288]

OS/2 および Windows NTローカル・クライアントをもつデータベース・サーバー 300 [32 ～ 524 288]

計測単位 ページ (4 KB)

割り振りタイミング データベースに最初に接続したとき

解放タイミング 最後のアプリケーションがデータベースから切断したとき

関連パラメーター

- 352ページの『カタログ・キャッシュ・サイズ (catalogcache_sz)』
- 353ページの『ログ・バッファー・サイズ (logbufsz)』

データベースごとに 1 つのデータベース・ヒープがあります。データベース・マネージャーは、データベースに接続するすべてのアプリケーションのためにそのデータベース・ヒープを使用します。これには、表、索引、表スペース、およびバッファー・プールの制御ブロック情報が入っています。また、ログ・バッファー (*logbufsz*)、およびカタログ・キャッシュ (*catalogcache_sz*) も入っています。したがって、ヒープのサイズは、ある時点でヒープに保管されている制御ブロックの数によって異なります。制御ブロック情報は、すべてのアプリケーションがデータベースから切断するまでヒープに保持されます。

最初に接続した時点で、データベース・マネージャーを開始するのに必要な最低限の量が割り振られます。必要に応じてデータ域は拡張していき、 *dbheap* で指定された最大量に達するまで拡張します。

推奨事項: データベース・ヒープのストレージが足りないため、ステートメントを処理できないことを示すエラーがアプリケーションに戻ったときは、この値を増す必要があります。

データベース・ヒープで使用されたメモリーの最大量をトラックするには、データベース・システム・モニターを使用することができます。詳しくは、*システム・モニター 手引きおよび解説書* の中の *db_heap_top (割り振られるデータベース・ヒープの最大値)* モニター要素の説明を参照してください。

このパラメーターを設定するときは、以下の点を考慮する必要があります。

- *logbufsz* の値。ログ・バッファーはデータベース・ヒープから割り振られるからです。
- *catalogcache_sz* の値。カタログ・キャッシュはデータベース・ヒープから割り振られるからです。

カタログ・キャッシュ・サイズ (catalogcache_sz)

構成タイプ データベース

パラメーター・タイプ 構成可能

デフォルト[範囲]

UNIX 64 [1 ～ 60 000]

OS/2 および Windows NTローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー 32 [1 ～ 60 000]

OS/2 および Windows NTローカル・クライアントをもつデータベース・サーバー 16 [1 ～ 60 000]

計測単位 ページ (4 KB)

関連パラメーター

- 351ページの『データベース・ヒープ (dbheap)』
- 353ページの『ログ・バッファー・サイズ (logbufsz)』
- 361ページの『アプリケーション制御ヒープ・サイズ (app_ctl_heap_sz)』

このパラメーターは、データベース・ヒープ (*dbheap*) からカタログ・キャッシュが使用できるスペースの最大量を示します。カタログ・キャッシュは、SQL ステートメントのコンパイルで表、視点、または別名を参照するときに使用する表記述子情報を保管するためのものです。

このキャッシュを使用すると、以前のステートメントで参照したのと同じ表、視点または別名を参照する場合、 SQL ステートメント (動的 SQL を含む) を結び付けるパフォーマンスが向上します。宣言済み一時表の記述子情報は、カタログ・キャッシュに保管されず、代わりにアプリケーション制御ヒープが使用されます。

ある表に対して DDL ステートメントを実行した場合、カタログ・キャッシュ内のその表の項目は除去されます。それ以外の場合は、他の表がそのスペースを必要とするまで表項目はキャッシュ内に保持されます。なお、その表を参照していた作業単位が完了するまで、表項目がキャッシュから除去されることはありません。

推奨事項: 最初はデフォルト値で開始し、次にそれをデータベース・システム・モニターを使って調整してください。

以下のモニター要素については、 システム・モニター *手引きおよび解説書* を参照してください。
- *cat_cache_lookups* (カタログ・キャッシュの検索)
- *cat_cache_inserts* (カタログ・キャッシュの挿入)
- *cat_cache_overflows* (カタログ・キャッシュのオーバーフロー)
- *cat_cache_heap_full* (カタログ・キャッシュ・ヒープがいっぱい)

これらのデータベース・システム・モニター要素を用いると、この構成パラメーターを調整する必要があるかどうかを判別することができます。このパラメーターを調整するときは、少しずつ (たとえば、一度に 2 ページ) 大きくしていってください。

一般的に、動的 SQL ステートメントを複数含んでいる作業単位の場合、または静的 SQL ステートメントを多数含んでいるパッケージをバインドしている場合は、より大きなキャッシュ・スペースが必要になります。

カタログ・キャッシュのサイズを設定するときは、ログ・ファイルのサイズ (*logbufsz*) についても考慮してください。 *catalogcache_sz* と *logbufsz* は、どちらもデータベース・ヒープ (*dbheap*) から割り振られるからです。

ログ・バッファー・サイズ (logbufsz)

構成タイプ データベース

パラメーター・タイプ 構成可能

デフォルト[範囲]

UNIX 32 ビット・プラットフォーム
8 [4 ～ 4 096]

UNIX 64 ビット・プラットフォーム
8 [4 ～ 65 535]

OS/2 および Windows NT
8 [4 ～ 4 096]

計測単位
ページ (4 KB)

関連パラメーター

- 352ページの『カタログ・キャッシュ・サイズ (catalogcache_sz)』
- 351ページの『データベース・ヒープ (dbheap)』
- 422ページの『グループ化するコミット数 (mincommit)』

このパラメーターは、ログ・レコードをディスクに書き込む前にバッファーとして使用するデータベース・ヒープ (*dbheap* パラメーターで定義する) の大きさを指定します。これらのログ・レコードは、以下のことが発生した場合にディスクに書き込まれます。

- *mincommit* 構成パラメーターの定義に従って、単一またはグループのトランザクションがコミットした
- ログ・バッファーが満杯
- 他のデータベース・マネージャー内部イベントが発生した

さらに、このパラメーターは *dbheap* パラメーターと同じかそれより小さくなっている必要があります。ログ・レコードのバッファリング機能によって、ディスクに書き込む頻度が少なくなり、一度に書き込まれるログ・レコードの数が多くなるので、ログ・ファイルのファイル入出力がさらに効率的になります。

推奨事項: 専用ログ・ディスクにおける読み取りアクティビティーがかなり活発である場合、またはディスク使用率が高い場合は、このバッファー域のサイズを増してください。専用ログ・ディスクの読み取りアクティビティーが非常に多い場合、またはディスク使用率が高い場合は、このバッファー域のサイズを大きくしてください。ログ・バッファー域は *dbheap* パラメーターで制御されているスペースを使用するので、このパラメーターの値を大きくする場合は、*dbheap* パラメーターの設定も検討してください。

特定のトランザクション (または作業単位) にどれだけのログ・バッファー域が使用されているかは、データベース・システム・モニターを使って判断することができます。

詳細については、 システム・モニター *手引きおよび解説書* の中の *log_space_used* (使用中の作業単位ログ・スペース) モニター要素の説明を参照してください。

ログ・バッファー・サイズを設定するときは、カタログ・キャッシュ・サイズ (*catalogcache_sz*) についても考慮してください。 *logbufsz_sz* と *catalogcache_sz* は、どちらもデータベース・ヒープ (*dbheap*) から割り振られるからです。

ユーティリティー・ヒープ・サイズ (util_heap_sz)

構成タイプ データベース

パラメーター・タイプ 構成可能

デフォルト[範囲] 5000 [16 〜 524 288]

計測単位 ページ (4 KB)

割り振りタイミング データベース・マネージャー・ユーティリティーで必要が生じたとき

解放タイミング ユーティリティーでメモリーが必要なくなったとき

関連パラメーター

- 『デフォルトのバックアップ・バッファー・サイズ (backbufsz)』
- 356ページの『デフォルトの復元バッファー・サイズ (restbufsz)』

このパラメーターは、BACKUP、RESTORE、 LOAD (ロード・リカバリーを含む) ユーティリティーが同時に使用できるメモリーの最大量を示します。

推奨事項: ユーティリティーにスペース不足が起きない限り、デフォルト値を使用してください。不足になったら、値を増します。システムのメモリーに制限がある場合は、このパラメーターの値を小さくし、データベース・ユーティリティーで使用されるメモリーを制限することができます。ただし、パラメーターの設定が低すぎると、ユーティリティーを並行して実行できないことがあります。並行ユーティリティーのために割り振りたいバッファーをすべて保持できるよう、十分な大きさを設定してください。

デフォルトのバックアップ・バッファー・サイズ (backbufsz)

構成タイプ データベース・マネージャー

適用範囲

- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー
- ローカル・クライアントをもつデータベース・サーバー
- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー
- ローカル・クライアントをもつサテライト・データベース・サーバー

パラメーター・タイプ 構成可能

デフォルト[範囲] 1024 [8 〜 16 384]

計測単位 ページ (4 KB)

割り振りタイミング バックアップ・ユーティリティーが呼び出されたとき

解放タイミング バックアップ・ユーティリティーが処理を完了したとき

関連パラメーター

- 『デフォルトの復元バッファー・サイズ (restbufsz)』
- 355ページの『ユーティリティー・ヒープ・サイズ (util_heap_sz)』

このパラメーターは、バックアップ・ユーティリティーを呼び出した時点でバッファー・サイズが明示指定されていない場合に、データベースのバックアップで使用するバッファーのサイズを指定します。バックアップ・ユーティリティーについての詳細は、*コマンド解説書* を参照してください。

データベースをバックアップすると、まずデータが内部バッファーにコピーされます。バッファーがいっぱいになると、バッファー内のデータはバックアップ・メディアに書き込まれます。

このバッファー・サイズを調整すると、バックアップ・ユーティリティーのパフォーマンスは向上します。並行しているデータベース操作のパフォーマンスへの影響は最小ですみます。

デフォルトの復元バッファー・サイズ (restbufsz)

構成タイプ データベース・マネージャー

適用範囲

- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー
- ローカル・クライアントをもつデータベース・サーバー
- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー
- ローカル・クライアントをもつサテライト・データベース・サーバー

パラメーター・タイプ 構成可能

デフォルト[範囲] 1024 [8 〜 16 384]

計測単位 ページ (4 KB)

割り振りタイミング 復元ユーティリティーが呼び出されたとき

解放タイミング 復元ユーティリティーが処理を完了したとき

関連パラメーター

- 355ページの『デフォルトのバックアップ・バッファー・サイズ (backbufsz)』
- 355ページの『ユーティリティー・ヒープ・サイズ (util_heap_sz)』

このパラメーターは、データベース復元ユーティリティーを呼び出した時点でバッファー・サイズが明示指定されていない場合に、データベースの復元で使用するバッファーのサイズを指定します。復元ユーティリティーについての詳細は、 *コマンド解説書* を参照してください。

データベースの復元では、まずデータがバックアップ・メディアから内部バッファーにコピーされます。バッファーがいっぱいになった時点でデータがターゲット・データベース・メディアに書き出されます。

このバッファー・サイズを調整すると、データベース復元ユーティリティーのパフォーマンスは向上します。並行しているデータベース操作のパフォーマンスへの影響は最小ですみます。

ロック・リスト用最大ストレージ (locklist)

構成タイプ データベース

パラメーター・タイプ 構成可能

デフォルト[範囲]

UNIX 100 [4 ～ 524 288]

OS/2 および NT ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー 50 [4 ～ 524 288]

OS/2 および NT ローカル・クライアントをもつデータベース・サーバー 25 [4 ～ 60 000]

計測単位 ページ (4 KB)

割り振りタイミング 最初のアプリケーションがデータベースに接続したとき

解放タイミング 最後のアプリケーションがデータベースから切断したとき

関連パラメーター

- 389ページの『自動調整前のロック・リストの最大パーセント (maxlocks)』
- 399ページの『活動アプリケーションの最大数 (maxappls)』

このパラメーターは、ロック・リストに割り振られるストレージの量を示します。データベースごとに 1 つのロック・リストがあります。その中には、データベースに並行して接続しているすべてのアプリケーションのロックが含まれています。ロッキングとは、複数のアプリケーションがデータベースのデータに並行アクセスするのを制御するために、データベース・マネージャーが使用する機構のことです。行と表の両方をロックすることができます。データベース・マネージャーは、内部で使用するロックを獲得することもできます。

ロックについて詳しくは、 51ページの『ロック』を参照してください。

ロック・リストでは 1 つのロックにつき 36 または 72 バイトが必要ですが、そのどちらであるかは、そのオブジェクトに関する他のロックが設定されているかどうかによって決まります。

- オブジェクトに他のロックが設定されていない場合は、72 バイトが必要です。
- オブジェクトに他のロックが設定されている場合は、36 バイトが必要です。

あるアプリケーションが使用しているロック・リストのパーセンテージが *maxlocks* に達すると、データベース・マネージャーはロック・エスカレーションを行い、そのアプリケーションが設定しているロックを行から表に代えます (後で説明します)。自動調整処理自体はそれほど時間がかかりませんが、表全体をロックすると (行の場合と比較) 並行性が低下するため、その表に以後アクセスする際のデータベース・パフォーマンスは全体として低下します。ロック・リスト・サイズの制御について、次のことを推奨します。

- COMMIT を頻繁に実行してロックを解放する。
- 多くの更新を実行するときは、更新を開始する前に表全体をロックする (SQL LOCK TABLE ステートメントを使用する)。ロックが 1 つだけ使用され、他のユーザーは更新を妨害できなくなりますが、データの並行性は低下します。

  また、ALTER TABLE ステートメントの LOCKSIZE パラメーターを使って、特定の表にロッキングがなされる方法を制御することもできます。詳細については、*SQL 解説書* を参照してください。

  反復可能読み取り分離レベルを使用すると、自動表ロックになります。分離レベルについての詳細は、 43ページの『第3章 アプリケーションについての考慮事項』を参照してください。
- 保留されている共用ロックの数を減らすため、可能であればカーソル固定分離レベルを使用する。アプリケーションの一貫性に関する要件に反しないようであれば、カーソル固定ではなく、非コミット読み取りを使用して、ロッキングの数をさらに少なくしてください。

ロック・リストがいっぱいになると、ロック・エスカレーションによって表ロックがさらに生成され、行ロックが少なくなるため、データベース内の共用オブジェクトの並行性は低下し、パフォーマンスは低下します。さらに、(表ロックの数が限定されているた

め) アプリケーション間のデッドロックが増え、トランザクションがロール・バックされることがあります。データベースのロック要求の最大数に達すると、アプリケーションは SQLCODE -912 を受け取ります。

推奨事項: ロック・エスカレーションがパフォーマンス問題を引き起こす場合は、このパラメーターまたは *maxlocks* パラメーターの値を大きくする必要があります。ロック・エスカレーションが実行されているかどうかは、データベース・システム・モニターを使って判別することができます。

詳細については、システム・モニター *手引きおよび解説書* の中の *lock_escals (ロック・エスカレーション)* モニター要素の説明を参照してください。

ロック・リストに必要なページ数は、次のステップで判別します。

1. ロック・リストのサイズの下限を計算します。

   ```
   (512 * 36 * maxappls) / 4096
   ```

   512 はアプリケーションごとの平均ロック数を推定した数字です。 36 は、ロックがすでに設定されているオブジェクトに、ロックを設定するのに必要なバイト数です。
2. ロック・リストのサイズの上限を計算します。

   ```
   (512 * 72 * maxappls) / 4096
   ```

   72 はオブジェクトに対する最初のロックで必要なバイト数を表しています。
3. データの並行性量を推定し、その推定値に基づいて、計算して求めた上限と下限の間に収まる *locklist* の初期値を選択します。
4. 後述されているように、データベース・システム・モニターを使ってこのパラメーターの値を調整します。

データベース・システム・モニターを使って、指定されたトランザクションで設定されるロックの最大数を判断することができます。

詳細は、システム・モニター *手引きおよび解説書* の中の *locks_held_top (保留するロックの最大数)* モニター要素の説明を参照してください。

この情報を用いて、アプリケーションごとの推定ロック数の妥当性検査や調整を行うことができます。妥当性検査を実行するには、複数のアプリケーションをサンプルとして使い、アプリケーション・レベルではなくトランザクション・レベルでモニター情報が提供されていることを確認する必要があります。

maxappls が増えた場合、または実行中のアプリケーションが頻繁にコミットを行わない場合は、 *locklist* を大きくすることができます。

このパラメーターを変更してから、アプリケーションの再バインドを考慮する必要があります (REBIND PACKAGE コマンドを使用)。

アプリケーションのパフォーマンスと影響する照会の最適化については、 41ページの『第2部 アプリケーション・パフォーマンスのチューニング』を参照してください。

パッケージ・キャッシュ・サイズ (pckcachesz)

構成タイプ データベース

パラメーター・タイプ 構成可能

デフォルト[範囲]

UNIX 32 ビット・プラットフォーム
-1 [-1, 32 ～ 128 000]

UNIX 64 ビット・プラットフォーム
-1 [-1, 32 ～ 524 288]

OS/2 および Windows NT
-1 [-1, 32 ～ 128 000]

計測単位 ページ (4 KB)

割り振りタイミング データベースが初期設定されるとき

解放タイミング データベースがシャットダウンされるとき

このパラメーターは、データベース・グローバル・メモリーから割り振られ、データベースに対する静的および動的 SQL ステートメントをキャッシュするために使用されます。区分データベース システムでは、各データベース区画にパッケージ・キャッシュが 1 つ設けられます。

パッケージをキャッシュしておくと、データベース・マネージャーは、パッケージを再ロードするときにシステム・カタログにアクセスする必要がなくなるので、または動的 SQL の場合はコンパイルの必要がなくなるので、内部オーバーヘッドが小さくなります。次のいずれかが生じるまで、セクションはパッケージ・キャッシュに保持されます。

- データベースがシャットダウンされた
- パッケージまたは動的 SQL ステートメントが無効になった
- キャッシュのスペースが足りなくなった

データベースに接続されているアプリケーションが何度も同じステートメントを使用するときは特に、そのような静的または動的 SQL ステートメントのセクションをキャッシュすると、パフォーマンスが向上します。このことは、特にトランザクション処理アプリケーションで重要です。

デフォルト (-1) を使用すると、ページ割り振りの計算に使用される値は、 *maxappls* 構成パラメーターに指定されている値の 8 倍になります。ただし、*maxappls* の 8 倍が 32 より小さい場合は、例外です。この場合は、デフォルト値の -1 を指定すると、*pckcachesz* は 32 に設定されます。

推奨事項: このパラメーターを調整するときは、パッケージ・キャッシュ用に予約される余分なメモリーを、他の目的、たとえば、バッファー・プールなどに割り振った方が効果的かどうかを検討してください。そのため、このパラメーターを調整する際には、ベンチマーク技法を使用してください。

最初は多くのセクションが使用されるが、その後反復されるセクションが少なくなる場合は、特に、このパラメーターの調整が重要になります。キャッシュが大き過ぎると、初期のセクションのコピーを保持するためにメモリーが無駄に使用されることになります。

以下のモニター要素については、 *システム・モニター 手引きおよび解説書* を参照してください。

- *pkg_cache_lookups* (パッケージ・キャッシュの検索)
- *pkg_cache_inserts* (パッケージ・キャッシュの挿入)
- *pkg_cache_size_top* (最大のパッケージ・キャッシュ・サイズ)
- *pkg_cache_num_overflows* (パッケージ・キャッシュのオーバーフローの回数)

これらのデータベース・システム・モニター要素を用いると、この構成パラメーターを調整する必要があるかどうかを判別することができます。

注: パッケージ・キャッシュは作業キャッシュなので、このパラメーターをゼロにすることはできません。このキャッシュには、現在実行されている SQL ステートメントの全セクションを保持するのに十分なメモリーを割り振る必要があります。現在必要なスペース以上のスペースが割り振られていると、セクションがキャッシュされます。これらのセクションは、必要になった時点でロードやコンパイルをせずに、そのまま実行することができます。

pckcachesz パラメーターによって指定された限界は、それほど厳格なものではありません。メモリーが依然としてデータベース共用セットで使用できる状態であれば、必要に応じてこの限界を超えることができます。 *pkg_cache_size_top* モニター要素を使用すると、パッケージ・キャッシュの最大サイズを判別することができます。また、 *pkg_cache_num_overflows* モニター要素を使用すると、 *pckcachesz* パラメーターで指定した限界を超えた回数を判別することができます。

アプリケーション共用メモリー

以下のパラメーターは、アプリケーションのために働くすべてのエージェント (調整およびサブエージェント) が使用する作業域を指定します。

- 『アプリケーション制御ヒープ・サイズ (app_ctl_heap_sz)』

アプリケーション制御ヒープ・サイズ (app_ctl_heap_sz)

構成タイプ データベース (Database)

パラメーター・タイプ 構成可能

デフォルト[範囲]

ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー
128 [1 〜 64 000]

ローカル・クライアントをもつデータベース・サーバー
64 [1 〜 64 000] (非 UNIX プラットフォームの場合)

128 [1 〜 64 000] (UNIX プラットフォームの場合)

ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー
256 [1 〜 64 000]

計測単位
ページ (4 KB)

割り振りタイミング
アプリケーションが開始するとき

解放タイミング
アプリケーションが終了するとき

関連パラメーター

- 352ページの『カタログ・キャッシュ・サイズ (catalogcache_sz)』
- 367ページの『アプリケーション・ヒープ・サイズ (applheapsz)』
- 474ページの『区画内並列処理機能の使用可能化 (intra_parallel)』

並列処理内を使用可能 (intra_parallel=ON) にした区分データベースおよび非区分データベースの場合、これは、アプリケーション制御ヒープに割り振られた共有メモリー域のサイズです。並列処理内を使用不可 (intra_parallel=OFF) にした非区分データベースの場合、これは、ヒープに割り振られる最大専用メモリーです。 1 つの区分に 1 つの接続が存在し、 1 つの接続に 1 つのアプリケーション制御ヒープが存在します。

アプリケーション制御ヒープは、主に、同じ要求のために機能するエージェント間で情報を共有するために必要となり、また、区分データベース環境の場合には、SQL ステートメントを表す実行可能なセクションを保管するために必要となります。並列処理が 1 以下の照会を実行しているとき、非区分データベースに対するこのヒープの使用量は最小となります。

このヒープは、宣言済み一時表の記述子情報を保管するためにも使用されます。明示的に除去されていないすべての宣言済み一時表の記述子情報は、このヒープのメモリーに保持され、宣言済み一時表を除去するまで除去することはできません。

推奨事項: 最初は、デフォルト値を使用して開始してください。複雑なアプリケーションを実行する場合、データベース区画が多数あるシステムの場合、または宣言済み一時表を使用する場合は、この値を大きく設定する必要があります。必要となるメモリーの量は、現在活動状態である宣言済み一時表が増えるとともに増加します。多数の列を持つ宣言済み一時表は、少しの列しか持たない表より表記述子のサイズが大きくなっています。それで、アプリケーションの宣言済み一時表に多数の列があると、アプリケーション制御ヒープの要求も大きくなります。

エージェント私用メモリー

次のパラメーターは、各データベース・エージェントで使用されるメモリーの量を制御します。

- 『分類ヒープ・サイズ (sortheap)』
- 364ページの『分類ヒープのしきい値 (sheapthres)』
- 366ページの『ステートメント・ヒープ・サイズ (stmtheap)』
- 367ページの『アプリケーション・ヒープ・サイズ (applheapsz)』
- 368ページの『統計ヒープ・サイズ (stat_heap_sz)』
- 368ページの『照会ヒープ・サイズ (query_heap_sz)』
- 370ページの『DRDA ヒープ・サイズ (drda_heap_sz)』
- 370ページの『UDF 共用メモリー・セット・サイズ (udf_mem_sz)』
- 372ページの『エージェント・スタック・サイズ (agent_stack_sz)』
- 373ページの『コミットされる私用メモリーの最小値 (min_priv_mem)』
- 374ページの『私用メモリーしきい値 (priv_mem_thresh)』
- 386ページの『Java インタープリター・ヒープの最大サイズ (java_heap_sz)』。UNIX ベースのプラットフォームでは、 *java_heap_sz* がエージェントごとに割り振られます。

私用エージェント・メモリーと、データベース・マネージャーで割り振られるそれ以外のメモリーとの関係については、 237ページの『DB2 でのメモリーの使用方法』を参照してください。

分類ヒープ・サイズ (sortheap)

構成タイプ	データベース
パラメーター・タイプ	構成可能
デフォルト[範囲]	256 [16 ～ 524 288]
計測単位	ページ (4 KB)
割り振りタイミング	分類を実行するために必要になったとき
解放タイミング	分類が完了したとき

関連パラメーター 『分類ヒープのしきい値 (sheapthres)』

このパラメーターは、私用分類に使用される私用メモリー・ページの最大数、または共用分類に使用される共用メモリー・ページの最大数を定義します。分類が私用分類の場合は、このパラメーターはエージェント私用分類に影響します。分類が共用分類の場合は、このパラメーターはデータベース共用メモリーに影響します。各分類には、データベース・マネージャーによって必要に応じて割り振られる個別の分類ヒープがあります。この分類ヒープは、データが分類される区域です。最適化プログラムが指示する場合には、そのプログラムが提供する情報を用いて、このパラメーターが指定するものより小さい分類ヒープが割り当てられます。

推奨事項:

ソート・ヒープの作動中、次のような判断が必要になります:

- 索引を適切に使うと、使用する分類ヒープの数を最小に抑えることができます。
- ハッシュ結合バッファーおよび動的ビットマップ (索引 ANDing およびスター型結合に使用) がソート・ヒープを使用します。これらの技法が使用される場合、このパラメーターのサイズを増やしておきます。
- 頻繁に大量データのソートの要求がある場合はこのパラメーターのサイズを増やしておきます。
- このパラメーターの値を大きくするときは、データベース・マネージャー構成ファイル内の *sheapthres* パラメーターも調整する必要があるかどうかを調べてください。
- この分類ヒープ・サイズは、最適化プログラムがアクセス・パスを判別するのに使用します。このパラメーターを変更してから、アプリケーションの再バインドを考慮する必要があります (REBIND PACKAGE コマンドを使用)。

分類ヒープのしきい値 (sheapthres)

構成タイプ データベース・マネージャー

適用範囲

- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー
- ローカル・クライアントをもつデータベース・サーバー
- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー
- ローカル・クライアントをもつサテライト・データベース・サーバー

パラメーター・タイプ 構成可能

デフォルト[範囲]

UNIX 32 ビット・プラットフォーム
20 000 [250 ～ 2 097 152]

UNIX 64 ビット・プラットフォーム

20 000

[250 ～ 2 147 483 647]

OS/2 および Windows NT

10 000 [250 ～ 2 097 152]

計測単位 ページ (4 KB)

関連パラメーター 363ページの『分類ヒープ・サイズ (sortheap)』

私用および共用分類は、2 つの異なるメモリー・ソースからメモリーを使用します。共用分類メモリー領域のサイズは、 *sheapthres* の値に基づくデータベースへの最初の接続時に静的に事前定義されます。私用分類メモリー領域は無制限です。

sheapthres パラメーターは、私用および共用分類に対して異なる方法で使用されます。

- 私用分類の場合このパラメーターは、指定された任意の時間に私用分類によって使用されたメモリーの合計量の、インスタンス幅の*緩和された* 制限です。インスタンスの私用分類メモリー使用量の合計がこの制限に達すると、追加の着信私用分類要求に割り当てられていたメモリーがかなり削減されます。
- 共用分類の場合このパラメーターは、指定された任意の時間に共用分類によって使用されたメモリーの合計量の、データベース幅の厳密な制限です。この制限に達すると、共用分類メモリー要求はそれ以上許可されません (共用分類メモリーの使用量の合計が、 *sheapthres* で指定された制限を下回るまで)。

分類ヒープを使用するこれらの操作の例には、分類、ハッシュ結合、動的ビットマップ (索引 ANDing およびスター型結合に使用) および表がメモリーにある操作が含まれます。

しきい値を明示的に定義すると、データベース・マネージャーが多数の分類のためにメモリーを過度に使用しないようにすることができます。

シングル・ノードからマルチ・ノード環境に移行した際、このパラメーターの値を増加させる理由はありません。一度、シングル・ノード (DB2 EE で) 環境のデータベースとデータベース・マネージャー構成パラメーターを調整すると、多くのケースでは、調整した値は、複数ノード (DB2 EE で) 環境でも問題なく作動します。

ヒープしきい値パラメーターのソートはデータベース・マネージャー構成パラメーターと同様に、全 DB2 インスタンスを対象として適用します。このパラメーターを異なったノードや区分で、異なった値に設定するには複数の DB2 インスタンスを作成します。これにより異なったノード・グループを超え、異なった DB2 データベースを管理する必要が出てきます。そのような調整は区分に分割されたデータベース環境の多くの利点という目的に反します。

推奨事項: このパラメーターを、ユーザーのデータベース・マネージャー・インスタンス内の最大の *sortheap* パラメーターの妥当な倍数に設定できれば、理想的です。このパ

ラメーターは、インスタンス内のデータベースについて定義されている *sortheap* の最大値より**少なくとも** 2 倍は大きくなければなりません。

私用分類を行う場合で、システムにメモリーの制約がない場合は、このパラメーターの理想値は、以下のステップを使用して計算することができます。
1. 各データベースの通常の分類ヒープ使用法を計算します。

```
    (typical number of concurrent agents running against the database)
  * (sortheap, as defined for that database)
```

2. 上記の結果を合計します。算出された分類ヒープの合計は、インスタンス内のすべてのデータベースに関する通常の状況に対応しています。

SMP 環境での分類の詳細については、 192ページの『並列分類の方式』を参照してください。

分類のパフォーマンスとメモリーの使用法とのバランスを取るためには、ベンチマーク技法を使ってこのパラメーターを調整してください。詳細については、 317ページの『第12章 ベンチマーク・テスト』を参照してください。

また、分類について詳しくは、 261ページの『分類』も参照してください。

データベース・システム・モニターを使用して、分類活動をトラックすることもできます。

詳細については、 *システム・モニター 手引きおよび解説書* にあるポストしきい値分類 (*post_threshold_sorts*) モニター要素の説明を参照してください。

ステートメント・ヒープ・サイズ (stmtheap)

構成タイプ	データベース
パラメーター・タイプ	構成可能
デフォルト[範囲]	2048 [128 〜 60 000]
計測単位	ページ (4 KB)
割り振りタイミング	各ステートメントをプリコンパイルまたは結び付けているとき
解放タイミング	各ステートメントのプリコンパイルまたは結び付けが完了したとき

ステートメント・ヒープは、SQL ステートメントのコンパイル時に、 SQL コンパイラーのワークスペースとして使用されます。このパラメーターは、このワークスペースのサイズを指定します。

この区域は永久的に割り振られるのではなく、SQL ステートメントを処理するたびに割り振られ、また後で解放されます。動的 SQL ステートメントの場合、この作業域はユ

ーザー・プログラムの実行時に使用されます。それに対して静的 SQL ステートメントの場合、この作業域はバインド処理時には使用されますが、プログラムの実行時は使用されません。

推奨事項: このパラメーターのデフォルト値は、多くの場合、受け入れることができます。非常に大きな SQL ステートメントがある場合に、データベース・マネージャーが、あるステートメントを最適化しようとしたときに (ステートメントが複雑すぎることを示す) エラーが戻された場合は、エラー状態が解決されるまでこのパラメーターの値を一定の量 (256 または 1024 など) ずつ増やしていってください。

アプリケーション・ヒープ・サイズ (applheapsz)

構成タイプ	データベース
パラメーター・タイプ	構成可能
デフォルト[範囲]	128 [16 ～ 60 000]
	64 [16 ～ 60 000] (区分データベース環境の場合)
計測単位	ページ (4 KB)
割り振りタイミング	エージェントがアプリケーション処理のために初期設定されるとき
解放タイミング	エージェントがアプリケーションのための仕事を完了したとき
関連パラメーター	361ページの『アプリケーション制御ヒープ・サイズ (app_ctl_heap_sz)』

このパラメーターは、特定のエージェントまたはサブエージェントのためにデータベース・マネージャーが使用できる私用メモリーのページ数を定義します。

ヒープは、エージェントまたはサブエージェントがアプリケーションのために初期設定されるときに割り振られます。割り振られる量は、要求を処理するためにエージェントまたはサブエージェントに与える必要のある最小値になります。エージェントまたはサブエージェントは、より大きな SQL ステートメントを処理するにはより大きなヒープ空間を必要とするため、データベース・マネージャー は、必要に応じて、このパラメーターが指定する最大値までのメモリーを割り振ります。

注: 区分データベース環境では、エージェントまたはサブエージェントのために、 SQL ステートメント実行セクションのコピーを格納するためにアプリケーション制御ヒープ (*app_ctl_heap_sz*) が使用されます。ただし、SMP のサブエージェントは、他の環境でのエージェントのように、 *applheapsz* を使用します。

推奨事項: アプリケーション・ヒープに十分なストレージがないことを示すエラーをアプリケーションが受け取った場合は、このパラメーターの値を増やしてください。

アプリケーション・ヒープ (*applheapsz*) は、エージェント私用メモリーから割り振られます。

統計ヒープ・サイズ (stat_heap_sz)

構成タイプ データベース

パラメーター・タイプ 構成可能

デフォルト[範囲] 4384 [1096 ～ 524 288]

計測単位 ページ (4 KB)

割り振りタイミング RUNSTATS ユーティリティーを開始したとき

解放タイミング RUNSTATS ユーティリティーが完了したとき

関連パラメーター

- 450ページの『頻度の高い値の保存数 (num_freqvalues)』
- 451ページの『列変位数の数 (num_quantiles)』

このパラメーターは、RUNSTATS コマンドを使って統計を収集するのに使用するヒープの**最大**サイズを示します。

推奨事項: 分散統計を収集しないとき、または比較的小さい表についてだけ分散統計を収集するときは、デフォルト値の使用が適切です。最小値を選択すると 1 つまたは 2 つの列から成る表しかヒープに合わないので、分散統計を収集する場合は、最小値の使用は推奨**されていません。**

統計を収集する列の数に応じて、このパラメーターを調整してください。比較的少数の列から成る小さな表の場合、分散統計を収集するためのメモリーは少なくて済みます。一方、多数の列を含む大きな表の場合は、メモリーがさらに必要になります。非常に大きな表の分散統計を収集するため大きな統計ヒープが必要な場合、システムの活動が少ない期間に統計を収集することができます。こうすると、他のユーザーのメモリー所要量を奪うことはありません。

照会ヒープ・サイズ (query_heap_sz)

構成タイプ データベース・マネージャー

適用範囲

- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー
- ローカル・クライアントをもつデータベース・サーバー
- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー

- ローカル・クライアントをもつサテライト・データベース・サーバー

パラメーター・タイプ	構成可能
デフォルト[範囲]	1000 [2 ～ 524 288]
計測単位	ページ (4 KB)
割り振りタイミング	アプリケーション (ローカルまたはリモート) がデータベースに接続したとき
解放タイミング	アプリケーションがデータベースから切断されたとき、またはアプリケーションがインスタンスから切り離されたとき
関連パラメーター	376ページの『アプリケーション・サポート層ヒープ・サイズ (aslheapsz)』

このパラメーターは、照会ヒープのために割り振るメモリー量の**最大値**を指定します。照会ヒープは、各照会をエージェントの私用メモリーに保管するために使用されます。各照会の情報には、入出力 SQLDA、ステートメント・テキスト、 SQLCA、パッケージ名、作成者、セクション番号、および一貫性トークンが含まれます。このパラメーターは、アプリケーションが不必要にエージェントの仮想メモリーを大量に消費することがないようにするためのものです。

照会ヒープは、ブロック・カーソル用に割り振られるメモリーに関しても使用されます。このメモリーは、カーソル制御ブロックと完全解決の出力 SQLDA から構成されます。

最初に割り振られる照会ヒープのサイズは、 *aslheapsz* パラメーターで指定されたアプリケーション・サポート層ヒープのサイズと同じです。照会ヒープ・サイズは、2 以上でかつ、 *aslheapsz* パラメーター以上である必要があります。指定された要求を処理する場合にこの照会ヒープのサイズが十分でないと、その要求に必要なサイズが再度割り振られます (*query_heap_sz* を超えることはありません)。この新しい照会ヒープが *aslheapsz* の 1.5 倍より大きくなると、照会が終了した時点で、照会ヒープには *aslheapsz* のサイズが再割り振りされます。

推奨事項: 多くの場合、デフォルト値で十分です。最低でも *query_heap_sz* には、*aslheapsz* の 5 倍以上の値を設定してください。そのように設定すると、*aslheapsz* より大きな照会が可能になり、同時に 3 つまたは 4 つのブロック・カーソルをオープンするためのメモリーを追加することができます。

LOB が非常に大きい場合は、その LOB を収容できるだけの大きさの照会ヒープを提供するために、このパラメーターの値を大きくする必要が生じる場合があります。

DRDA ヒープ・サイズ (drda_heap_sz)

構成タイプ データベース・マネージャー

適用範囲

- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー
- クライアント
- ローカル・クライアントをもつデータベース・サーバー
- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー
- ローカル・クライアントをもつサテライト・データベース・サーバー

パラメーター・タイプ 構成可能

デフォルト[範囲] 128 [16 ～ 60 000]

計測単位 ページ (4 KB)

割り振りタイミング

- DRDA アプリケーション・サーバー (AS) は、DRDA アプリケーション・リクエスター (AR) が DB2 データベースに接続するたびに DRDA ヒープを割り振る。
- DB2 コネクトは、DRDA AS に接続するたびに DRDA ヒープを割り振る。

解放タイミング DRDA AR がデータベースから切断したとき

このパラメーターは、 DB2 コネクトおよび DRDA アプリケーション・サーバー・サポート機能が使用するメモリーとして割り振られるページ数を指示します。このヒープから割り振られるメモリーの量は、次の項目により決まります。

- アプリケーションがオープンしているカーソルの数
- 入力ホスト変数の数
- 選択リストに含まれている項目の数
- 入出力データのサイズ
- バインドまたは準備されている SQL ステートメントの長さ

推奨事項: DRDA ヒープ・メモリーが不足であるというエラー・コードが出ない限り、デフォルト値を使用してください。

UDF 共用メモリー・セット・サイズ (udf_mem_sz)

構成タイプ データベース・マネージャー

適用範囲

- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー
- ローカル・クライアントをもつデータベース・サーバー
- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー
- ローカル・クライアントをもつサテライト・データベース・サーバー

パラメーター・タイプ 構成可能

デフォルト[範囲] 256 [128 ～ 60 000]

計測単位 ページ (4 KB)

割り振りタイミング UDF が開始するとき

解放タイミング UDF が完了したとき

このパラメーターは、分離されたユーザー定義関数 (UDF) と分離されていない UDF で共通です。分離された UDF の場合、データベース・プロセスと UDF とで共用するメモリーのデフォルト割り振りを指定します。単一区分データベース環境では、共用メモリー・セットはただ 1 つだけです。区分データベース環境では、各データベース区画サーバーに 1 つの共用メモリー・セットがあり、そのサーバーで実行されるすべてのアプリケーション・エージェントおよびサブエージェントが、同じ共用メモリー・セットを使用します。

分離されていない UDF の場合、パラメーターは私用メモリー・セットのサイズを指定します。単一区画データベース環境では、ヒープは私用メモリーから割り振られます。区分データベース環境では、ヒープは各データベース区画サーバーのアプリケーション・グローバル・メモリーから割り振られ、そのデータベース区画サーバーのアプリケーションのために実行されるエージェントおよびサブエージェントがこの同じ共用メモリー・セットを使用します。

分離されている UDF でも分離されていない UDF でも、このメモリーは UDF とデータベースの間でデータを受け渡しするために使用されます。

UDF がアプリケーションで使用されない場合は、メモリーは割り当てられません。同じアプリケーションで、分離されている UDF と分離されていない UDF の両方を実行する場合は、メモリー割り振りが 2 つ行われることになります。 1 つは分離された UDF 用、もう 1 つは分離されていない UDF 用です。

ユーザー定義関数については、 *アプリケーション開発の手引き* および *SQL 解説書* を参照してください。

推奨事項: デフォルト設定は、 LOB データを UDF に渡す場合を除いて他のどのような状況にも十分対応できるはずです。 LOB データを UDF に渡す状況では、メモリー

の割り振り量を増やす必要が生じることがあります。このパラメーターには、入力引き数のサイズおよび外部関数の結果より最低でも 2 ページは大きな値を設定してください。

注: UDF のメモリー要件は付加的なものであるため、このパラメーターの最適設定は、アプリケーション内で参照される UDF の数に影響されます。

エージェント・スタック・サイズ (agent_stack_sz)

構成タイプ データベース・マネージャー

適用範囲

- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー
- ローカル・クライアントをもつデータベース・サーバー
- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー
- ローカル・クライアントをもつサテライト・データベース・サーバー

パラメーター・タイプ 構成可能

デフォルト[範囲]

OS/2	64 [8 ～ 1000]
Windows NT	16 [8 ～ 1000]

計測単位 ページ (4 KB)

割り振りタイミング エージェントがアプリケーション処理のために初期設定されるとき

解放タイミング エージェントがアプリケーションのための仕事を完了したとき

エージェント・スタックは、各エージェント用に DB2 が割り振る仮想メモリーです。このメモリーは、 SQL ステートメントを処理するために必要になると、コミットされます。このパラメーターを使用すると、特定のアプリケーションの集まりに合わせてサーバーのメモリー使用状況を最適化することができます。単純な照会と比べて、照会が複雑になればなるほど使用するスタック・スペースは大きくなります。

このパラメーターは UNIX ベースのプラットフォームには適用されません。

推奨事項: ほとんどの場合、デフォルトのスタック・サイズを使用することができます。ユーザーの環境で非常に複雑な照会が多くなったときだけ、このパラメーターの値を大きくする必要が生じます。スタック・サイズの大きさが SQL ステートメントを処

理するのに十分ではないと、 db2diag.log ファイルにエラー項目がログに記録されて、SQL コードが出されます。 *agent_stack_sz* を大きくして、データベース・インスタンスを再始動する必要があります。

ユーザーの環境が次の条件を満たしていると、スタック・サイズを小さくして、他のクライアントが利用できるアドレス・スペースを広げることができます。
- 単純なアプリケーション (たとえば、簡単な OLTP) しか含まれておらず、複雑な照会が存在しない
- 比較的多数の並行クライアント (たとえば、100 以上) が必要である

エージェント・スタックのサイズと並行クライアントの数は、反比例しています。スタック・サイズを大きくすると、実行可能な並行クライアントの数は減ります。これは、アドレス・スペースが OS/2 および Windows NT プラットフォーム上で限定されているために発生します。たとえば、OS/2 では、400 MB のアドレス・スペースがあるという前提になります (この量は、 `config.sys` ファイルによって異なります)。 *agent_stack_sz* の値を 1 MB に設定すると、 400 以上のエージェントを入手することはできません。 (実際には、バッファー・プールなどアドレス・スペースの他の要件のために、入手できるエージェントはおそらくそれ以下になります。) つまり、*maxagents* をより大きな値に設定 (たとえば 5000 など) に設定すれば、この限界に到達することはありません。

コミットされる私用メモリーの最小値 (min_priv_mem)

構成タイプ
データベース・マネージャー

適用範囲
- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー
- ローカル・クライアントをもつデータベース・サーバー
- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー
- ローカル・クライアントをもつサテライト・データベース・サーバー

パラメーター・タイプ
構成可能

デフォルト[範囲]
32 [32 ～ 112 000]

計測単位
ページ (4 KB)

割り振りタイミング
データベース・マネージャーが開始したとき

解放タイミング
データベース・マネージャーが停止したとき

関連パラメーター
374ページの『私用メモリーしきい値 (priv_mem_thresh)』

このパラメーターは、データベース・マネージャー・インスタンスの開始 (db2start) 時に、データベース・サーバー・プロセスが私用仮想メモリーとして予約するページの数を指定します。追加の私用メモリーが必要であれば、サーバーはオペレーティング・システムからメモリーをさらに獲得しようと試みます。

このパラメーターは UNIX ベースのシステムには適用されません。

推奨事項: デフォルト値を使用してください。

データベース・サーバーにより多くのメモリーをコミットしたい場合だけ、この値を変更してください。このようにすると、割り振り時間の節約になります。ただし、この値を大きくすると DB2 以外のアプリケーションのパフォーマンスに影響することがあるので、この値を大きく設定し過ぎないように注意してください。

私用メモリーしきい値 (priv_mem_thresh)

構成タイプ データベース・マネージャー

適用範囲

- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー
- ローカル・クライアントをもつデータベース・サーバー
- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー
- ローカル・クライアントをもつサテライト・データベース・サーバー

パラメーター・タイプ 構成可能

デフォルト[範囲] 1296 [-1; 32 ～ 112 000]

32 [-1; 32 ～ 112 000] (ローカル・クライアントをもつサテライト・データベース・サーバー)

計測単位 ページ (4 KB)

関連パラメーター 373ページの『コミットされる私用メモリーの最小値 (min_priv_mem)』

このパラメーターは、未使用のエージェント私用メモリーのうち、新しく開始されるエージェントが使用できるよう、割り振られたままにしておく量を決定します。これは UNIX ベースのシステムには適用されません。

エージェントが終了すると、データベース・マネージャーは、そのエージェントが使用していたメモリーをすべて割り振り解除するのではなく、次の式で計算された*超過メモリー割り振り* だけを割り振り解除します。

```
割り振られた私用メモリー -
(使用された私用メモリー + priv_mem_thresh)
```

この式の結果が負の数になった場合は、何のアクションも実行されません。

次の表は、メモリーの割り振りと割り振り解除が行われる時点を示す例です。この例では、 *priv_mem_thresh* の任意の設定値として 100 を使います。

アクションの説明	割り振られるメモリー	使用されるメモリー
多数のエージェントが実行中。メモリーは割り振られている。	1000	1000
新しいエージェントが開始され、100 ページのメモリーを使用した。	1100	1100
200 ページのメモリーを使用していたエージェントが終了した。 (100 ページのメモリーが解放されましたが、将来の使用のために 100 ページが割り振られたままであることに注意してください。)	1000	900
50 ページのメモリーを使用していたエージェントが終了した。 (50 ページのメモリーが解放されましたが、余分な 100 ページが割り振られたままであることに注意してください。既存のエージェントで使用されている量と比較してください。)	950	850
新しいエージェントが開始され、 150 ページのメモリーが必要になった。 (150 ページのうち 100 ページはすでに割り振られているので、データベース・マネージャーはこのエージェント用に 50 ページを追加割り振りするだけで済みます。)	1000	1000

『-1』 に設定すると、このパラメーターでは *min_priv_mem* パラメーターの値が使用されます。

推奨事項: このパラメーターを設定するときは、同じマシン上にある他のプロセスのメモリー要件を考慮するのと同時に、クライアントの接続 / 切断パターンについても考える必要があります。

多くのクライアントが並行してデータベースに接続する時間が短いのに、しきい値を高めに設定すると、未使用のメモリーをコミット解除し、他のプロセスで使用可能にすることができなくなります。その結果、メモリー管理は低下し、その影響はメモリーを必要としている他のプロセスにまで及びます。

一定の数のクライアントが並行しており、その数が時々変化する場合、しきい値を高めに設定すると、クライアント・プロセスが使用できるメモリーを確保し、メモリーの割り振りおよび割り振り解除で生じるオーバーヘッドを減らすことができます。

エージェント / アプリケーション通信メモリー

次のパラメーターは、ユーザー・アプリケーションとエージェント・プロセスとの間でデータを渡すために割り振られるメモリーの量を制御します。

- 『アプリケーション・サポート層ヒープ・サイズ (aslheapsz)』
- 378ページの『10 進数除算の 3 の位取り (min_dec_div_3)』
- 379ページの『クライアント入出力ブロック・サイズ (rqrioblk)』
- 380ページの『DOS リクエスター入出力ブロック・サイズ (dos_rqrioblk)』

エージェント / アプリケーション共用メモリーと、データベース・マネージャーで割り振られるそれ以外のメモリーとの関係については、 237ページの『DB2 でのメモリーの使用方法』を参照してください。

アプリケーション・サポート層ヒープ・サイズ (aslheapsz)

構成タイプ
データベース・マネージャー

適用範囲
- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー
- ローカル・クライアントをもつデータベース・サーバー
- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー
- ローカル・クライアントをもつサテライト・データベース・サーバー

パラメーター・タイプ
構成可能

デフォルト[範囲]
15 [1 ～ 524 288]

計測単位
ページ (4 KB)

割り振りタイミング
データベース・マネージャーのエージェント・プロセスがローカル・アプリケーション用に開始するとき

解放タイミング
データベース・マネージャーのエージェント・プロセスが終了したとき

関連パラメーター
368ページの『照会ヒープ・サイズ (query_heap_sz)』

アプリケーション・サポート層ヒープは、ローカル・アプリケーションと関連エージェントとの間にある通信バッファーを表しています。このバッファーは、開始されるデータベース・マネージャー・エージェントごとに、共用メモリーとして割り振られます。

データベース・マネージャーへの要求または関連応答がバッファーに収まらない場合、それらの要求または応答は 2 つ以上の送受信の対に分けられます。このバッファーのサイズは、ほとんどの要求をシングル送受信の対で処理できるように設定してください。要求のサイズは、次のものを保持するのにどれくらいのストレージが必要かによって決まります。

- 入力 SQLDA
- SQLVAR にあるすべての関連データ
- 出力 SQLDA
- 他のフィールド (通常は 250 バイト以下)

このパラメーターは、通信バッファーの他にも、ブロック・カーソルがオープンしているときの入出力ブロック・サイズも指定することができます。ブロック・カーソル用のメモリーはアプリケーションの専用アドレス・スペースから割り振られます。したがって、各アプリケーション・プログラムに割り振る私用メモリーの量として、最適な値を指定する必要があります。データベース・クライアントが、アプリケーションの私用メモリーから、ブロック・カーソル用のスペースを割り振ることができない場合は、非ブロック・カーソルがオープンされます。

ローカル・アプリケーションから送信されたデータはデータベース・マネージャーに受信され、照会ヒープから割り振られた連続メモリーに入れられます。照会ヒープの初期サイズ (ローカル・クライアントとリモート・クライアントの両方) は、 *aslheapsz* パラメーターを使って決定します。照会ヒープの最大サイズは、 *query_heap_sz* パラメーターで定義されます。

推奨事項: アプリケーションの要求が小さい場合が多く、しかもアプリケーションがメモリー制約のあるシステムで実行される場合は、このパラメーターの値を小さくすることができます。照会は通常は非常に大きく、複数の送受信要求が必要になり、かつ使用するシステムのメモリーが制限されている場合には、このパラメーターの値を増やすこともできます。

次の式を使って、*aslheapsz* の最小ページ数を計算します。

```
aslheapsz >= ( sizeof(input SQLDA)
             + sizeof(each input SQLVAR)
             + sizeof(output SQLDA)
             + 250 ) / 4096
```

sizeof(x) は、指定の入力値または出力値のページ数を計算する x のサイズ (バイト) です。

このパラメーターがブロック・カーソルの数や潜在的なサイズに与える影響についても考慮する必要があります。大きな行ブロックの場合、転送する行の数またはサイズが大

きい (たとえば、データの量が 4 096 バイトより大きい) と、パフォーマンスは向上する可能性があります。しかし、レコード・ブロックを大きくすると、個々の接続の実効ページ・セットのサイズも大きくなってしまいます。

さらに、レコード・ブロックを大きくすると、アプリケーションが実際に要求する数より多くの FETCH 要求が出されます。 FETCH 要求の数を制御するには、ユーザーのアプリケーションで SELECT ステートメントの OPTIMIZE FOR 文節を使用します。 OPTIMIZE FOR 文節については、 76ページの『OPTIMIZE FOR n ROWS 文節』を参照してください。

10 進数除算の 3 の位取り (min_dec_div_3)

構成タイプ	データベース
パラメーター・タイプ	構成可能
デフォルト[範囲]	No [Yes, No]

min_dec_div_3 データベース構成パラメーターが、 SQL での 10 進数除算の位取りの計算に対する変更を使用可能にする手早い方法として追加されました。 *min_dec_div_3* には ″Yes″ か ″No″ を設定できます。 *min_dec_div_3* のデフォルト値は ″No″ です。

min_dec_div_3 データベース構成パラメーターは、除算を含む 10 進算術演算の位取りを変更します。値が ″No″ の場合は、位取りは 31-p+s-s’ として計算されます。詳しくは、*SQL 解説書* の第 3 章『SQL での 10 進数演算』を参照してください。 ″Yes″ に設定した場合、位取りは MAX(3, 31-p+s-s’) として計算されます。これにより、10 進数の除算の結果は常に、少なくとも 3 の位取りを持ちます。精度は常に 31 です。

このデータベース構成パラメーターを変更すると、既存のデータベースについてアプリケーションが変更される可能性があります。これは、10 進数除算の結果の位取りが、このデータベース構成パラメーターの変更によって影響を受けたときに起きます。下のリストは、アプリケーションに影響する可能性のあるいくつかのシナリオを示したものです。これらのシナリオは、既存のデータベースを持つデータベース・サーバーで *min_dec_div_3* を変更する前に考慮しておくべきです。

- 視点の列の 1 つの結果の位取りが変更された場合、ある設定で環境で定義された視点は、データベース構成パラメーターが変更された後で参照されると、 SQLCODE -344 でエラーになります。メッセージ SQL0344N は再帰的共通表式について述べていますが、オブジェクト名 (最初のトークン) が視点の場合は、このエラーを避けるために視点をドロップして再作成する必要があります。
- 静的パッケージは、パッケージが暗黙的または明示的に再バインドされるまで、振る舞いを変えません。たとえば、値を NO から YES に変更した後、再バインドが行われるまで、追加の位取りは組み込まれません。変更された静的パッケージはどれも、REBIND コマンドを使用して再バインドを強制することができます。
- 10 進数に関するチェック制約は、以前に受け入れられた値を制限する可能性があります。このような行は制約違反になりますが、チェック制約行が含まれる列のいずれ

かが更新されるか、 IMMEDIATE CHECKED オプション指定の SET INTEGRITY ステートメントが処理されるまで検出されません。このような制約の検査を強制するには、 ALTER TABLE ステートメントを実行してチェック制約を削除し、次に ALTER TABLE ステートメントを実行して再び制約を追加します。

注: DB2 バージョン 7 にはさらに以下の制限があります。

1. コマンド GET DB CFG FOR DBNAME は *min_dec_div_3* 設定を表示しません。現在の設定を判別する最も良い方法は、10 進数の除算結果の副次作用をモニターすることです。たとえば、次のステートメントを考えてみてください:

   ```
   VALUES (DEC(1,31,0)/DEC(1,31,5))
   ```

   このステートメントが sqlcode SQL0419N を返す場合、データベースは *min_dec_div_3* サポートを持っていないか、または ″No″ に設定されています。ステートメントが 1.000 を返す場合、 *min_dec_div_3* は ″Yes″ に設定されています。
2. *min_dec_div_3* は、次のコマンドを実行したときは構成キーワードのリストに示されません: ?UPDATE DB CFG

クライアント入出力ブロック・サイズ (rqrioblk)

構成タイプ データベース・マネージャー

適用範囲

- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー
- クライアント
- ローカル・クライアントをもつデータベース・サーバー
- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー
- ローカル・クライアントをもつサテライト・データベース・サーバー

パラメーター・タイプ 構成可能

デフォルト[範囲] 32 767 [4 096 ～ 65 535]

計測単位 バイト

割り振りタイミング

- リモート・クライアント・アプリケーションが、サーバー・データベースへの接続要求を発行したとき
- ブロック・カーソルがオープンされ、追加のブロックがクライアントでオープンされたとき

解放タイミング

- リモート・アプリケーションがデータベースから切断したとき
- ブロック・カーソルが閉じられたとき

関連パラメーター 『DOS リクエスター入出力ブロック・サイズ (dos_rqrioblk)』

このパラメーターは、リモート・アプリケーションとデータベース・サーバーのデータベース・エージェントとの間に置かれる通信バッファーのサイズを指定します。データベース・クライアントから、リモート・データベースへの接続要求が発行されると、クライアント上でこの通信バッファーが割り振られます。データベース・サーバーでは、接続が確立されてクライアント上の *rqrioblk* の値をサーバーが判別できるようになるまで、 32 767 バイトの通信バッファーが最初に割り振られます。サーバーがこの値を判別できるようになった後で、クライアントのバッファーが 32 767 バイトになっていないと、通信バッファーは再割り振りされます。

このパラメーターは、通信バッファーの他にも、ブロック・カーソルがオープンされているときのデータベース・クライアントの入出力ブロック・サイズも指定することができます。ブロック・カーソル用のメモリーはアプリケーションの専用アドレス・スペースから割り振られます。したがって、各アプリケーション・プログラムに割り振る私用メモリーの量として、最適な値を指定する必要があります。データベース・クライアントが、アプリケーションの私用メモリーから、ブロック・カーソル用のスペースを割り振ることができない場合は、非ブロック・カーソルがオープンされます。

推奨事項: 非ブロック・カーソルの場合は、 1 つの SQL ステートメントで転送したいデータ (ラージ・オブジェクト・データなど) が非常に大きくて、デフォルト値では不十分であるときなどが、このパラメーターの値を増す場合です。

このパラメーターがブロック・カーソルの数や潜在的なサイズに与える影響についても考慮する必要があります。大きな行ブロックの場合、転送する行の数またはサイズが大きい (たとえば、データの量が 4 096 バイトより大きい) と、パフォーマンスは向上する可能性があります。しかし、レコード・ブロックを大きくすると、個々の接続の実効ページ・セットのサイズも大きくなってしまいます。

さらに、レコード・ブロックを大きくすると、アプリケーションが実際に要求する数より多くの FETCH 要求が出されます。 FETCH 要求の数を制御するには、ユーザーのアプリケーションで SELECT ステートメントの OPTIMIZE FOR 文節を使用します。 OPTIMIZE FOR 文節の詳細については、 76ページの『OPTIMIZE FOR n ROWS 文節』を参照してください。

DOS リクエスター入出力ブロック・サイズ (dos_rqrioblk)

構成タイプ データベース・マネージャー

適用範囲

- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー
- クライアント
- ローカル・クライアントをもつデータベース・サーバー
- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー
- ローカル・クライアントをもつサテライト・データベース・サーバー

パラメーター・タイプ 構成可能

デフォルト[範囲] 4 096 [4 096 ～ 65 535]

計測単位 バイト

割り振りタイミング

- リモート DOS または Windows 3.1 クライアントがサーバー・データベースへの接続要求を出したとき
- ブロック・カーソルがオープンされ、追加のブロックがクライアントでオープンされたとき

解放タイミング

- リモート・アプリケーションがデータベースから切断したとき
- ブロック・カーソルがクローズされたとき

関連パラメーター 379ページの『クライアント入出力ブロック・サイズ (rqrioblk)』

このパラメーターは、 DOS/Windows 3.1 アプリケーションとデータベース・サーバーのデータベース・エージェントとの間にある通信バッファーのサイズを指定します。このパラメーターは *rqrioblk* パラメーターと似ていますが、 DOS/Windows 3.1 クライアントで使用されるブロックに、別の値を設定できる点が異なります。 DB2 構成ファイルでは、 *rqrioblk* パラメーター (32 ビット Windows、OS/2、および UNIX クライアント用) および *dos_rqrioblk* パラメーター (DOS および Windows 3.1 クライアント用) の両方を設定することができます。

このパラメーターは、通信バッファーの他にも、ブロック・カーソルがオープンされているときのデータベース・クライアントの入出力ブロック・サイズも指定することができます。ブロック・カーソル用のメモリーはアプリケーションの専用アドレス・スペースから割り振られます。したがって、各アプリケーション・プログラムに割り振る私用メモリーの量として、最適な値を指定する必要があります。データベース・クライアン

トが、アプリケーションの私用メモリーから、ブロック・カーソル用のスペースを割り振ることができない場合は、非ブロック・カーソルがオープンされます。

推奨事項: 非ブロック・カーソルの場合は、 1 つの SQL ステートメントで転送したいデータ (ラージ・オブジェクト・データなど) が非常に大きくて、デフォルト値では不十分であるときなどが、このパラメーターの値を増す場合です。

このパラメーターがブロック・カーソルの数や潜在的なサイズに与える影響についても考慮する必要があります。大きな行ブロックの場合、転送する行の数またはサイズが大きい (たとえば、データの量が 4 096 バイトより大きい) と、パフォーマンスは向上する可能性があります。しかし、レコード・ブロックを大きくすると、個々の接続の実効ページ・セットのサイズも大きくなってしまいます。

さらに、レコード・ブロックを大きくすると、アプリケーションが実際に要求する数より多くの FETCH 要求が出されます。 FETCH 要求の数を制御するには、ユーザーのアプリケーションで SELECT ステートメントの OPTIMIZE FOR 文節を使用します。 OPTIMIZE FOR 文節の詳細については、 76ページの『OPTIMIZE FOR n ROWS 文節』を参照してください。

データベース・マネージャー インスタンス・メモリー

次のパラメーターは、インスタンス・レベルで割り振られて使用されるメモリーを制御します。

- 『データベース・システム・モニター・ヒープ・サイズ (mon_heap_sz)』
- 384ページの『ディレクトリー・キャッシュ・サポート (dir_cache)』
- 386ページの『監査バッファー・サイズ (audit_buf_sz)』
- 386ページの『Java インタープリター・ヒープの最大サイズ (java_heap_sz)』

データベース・システム・モニター・ヒープ・サイズ (mon_heap_sz)

構成タイプ データベース・マネージャー

適用範囲

- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー
- ローカル・クライアントをもつデータベース・サーバー
- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー
- ローカル・クライアントをもつサテライト・データベース・サーバー

パラメーター・タイプ 構成可能

デフォルト[範囲]

UNIX 56 [0 ～ 60 000]

OS/2 および Windows NT ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー および ローカル・クライアントをもつサテライト・データベース・サーバー
32 [0 ～ 60 000]

OS/2 および Windows NT ローカル・クライアントをもつデータベース・サーバー
12 [0 ～ 60 000]

計測単位 ページ (4 KB)

割り振りタイミング *db2start* コマンドによりデータベース・マネージャーが開始されたとき

解放タイミング *db2stop* コマンドによりデータベース・マネージャーが停止したとき

関連パラメーター 478ページの『デフォルト データベース・システム・モニター・スイッチ (dft_monswitches)』

このパラメーターは、データベース・システム・モニターのデータ用に割り振るメモリー量をページ単位で判別します。スナップショットの獲得、モニター・スイッチの調整、モニターのリセット、またはイベント・モニターの活動化などの、データベース・モニター活動を実行すると、メモリーがモニター・ヒープから割り振られます。

ゼロに設定すると、データベース・マネージャーはデータベース・システム・モニター・データを収集しません。

推奨事項: モニター活動に必要なメモリーの量は、スイッチが設定されたモニター・アプリケーション (スナップショットを獲得するアプリケーションまたはイベント・モニター) の数とデータベース活動のレベルによって異なります。

モニター・ヒープで必要なページ数は、次の式を使って概算することができます。

```
( モニター・アプリケーションの数 + 1 ) *
( データベースの数 *
  (800 + ( アクセスされた表の数 * 20 )
   + ( ( 接続されたアプリケーションの数 + 1) *
       (200 + (表スペースの数 * 100) ) ) ) )
/ 4096
```

ヒープ中の使用可能メモリーが足りなくなると、次のいずれかの事態が生じます。

- このイベント・モニターが定義されているデータベースに、最初のアプリケーションが接続した時点で、 db2alert.log および db2diag.log ファイルにレベル 2 のエラー・メッセージが書き込まれます。

- SET EVENT MONITOR ステートメントにより動的に開始されたイベント・モニターが失敗した場合は、エラー・コードがアプリケーションに戻されます。
- モニター・コマンドまたは API サブルーチンが失敗した場合は、エラー・コードがアプリケーションに戻されます。

ディレクトリー・キャッシュ・サポート (dir_cache)

構成タイプ　データベース・マネージャー

適用範囲

- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー
- クライアント
- ローカル・クライアントをもつデータベース・サーバー
- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー
- ローカル・クライアントをもつサテライト・データベース・サーバー

パラメーター・タイプ　構成可能

デフォルト[範囲]　Yes [Yes; No]

割り振りタイミング

- アプリケーションが最初の接続を発行すると、私用キャッシュが割り振られます。
- データベース・マネージャー・インスタンスを開始（db2start）すると、共用キャッシュが割り振られます。

解放タイミング

- アプリケーション・プロセスを終了すると、私用キャッシュが解放されます。
- データベース・マネージャー・インスタンスを停止（db2stop）すると、共用キャッシュが解放されます。

dir_cache を Yes に設定すると、データベース、ノード、および DCS ディレクトリー・ファイルがメモリーにキャッシュされます。ディレクトリー・キャッシュを使用すると、ディレクトリー・ファイルの入出力が省略され、ディレクトリー情報を検索するためのディレクトリー検索が最小化されるので、接続コストは減少します。ディレクトリー・キャッシュには次の 2 つのタイプがあります。

- アプリケーションを実行しているマシンで、アプリケーション・プロセスごとに割り振られて使用される私用キャッシュ。

- 一部の内部データベース・マネージャー・プロセス用に割り振られて使用される、共用キャッシュ。

注: サポートされている Windows 環境では、私用キャッシュだけが使用されます。

私用キャッシュの場合、アプリケーションが最初の接続を発行すると、各ディレクトリー・ファイルが読み取られ、そのアプリケーションに関する情報が私用メモリーにキャッシュされます。その後の接続要求では、アプリケーション・プロセスはこのキャッシュを使用します。アプリケーション・プロセスが活動している間は、キャッシュは保守されます。私用キャッシュにデータベースがない場合は、ディレクトリー・ファイルで情報が検索されますが、キャッシュ自体は更新されません。アプリケーションがディレクトリー項目を修正すると、アプリケーションが次回の接続を発行した時点で、このアプリケーション用のキャッシュが最新表示されます。他のアプリケーションに属する私用キャッシュは最新表示されません。アプリケーションがディレクトリー項目を修正すると、アプリケーションが次回の接続を発行した時点で、このアプリケーション用のキャッシュが最新表示されます。 (コマンド行プロセッサー・セッションが使用するディレクトリー・キャッシュを最新表示するには、 `db2 terminate` コマンドを出してください。)

共用キャッシュの場合、データベース・マネージャー・インスタンスが開始（`db2start`）されると、各ディレクトリー・ファイルが読み取られ、情報が共用メモリーにキャッシュされます。このキャッシュは、データベース・マネージャー・プロセスの一部により使用され、インスタンスが停止するまで（`db2stop`）保守されます。アプリケーション・プロセスが終了すると、キャッシュも解放されます。インスタンスの実行中に、共用キャッシュが最新表示されることはありません。

推奨事項: ディレクトリー・ファイルが頻繁には変更されない場合、パフォーマンスが重要事項であれば、ディレクトリー・キャッシュを使用してください。

さらに、リモート・クライアントで、アプリケーションが複数の異なる接続要求を発行する場合には、ディレクトリーをキャッシュしておくと効果的です。この場合、キャッシュを使うと、シングル・アプリケーションがディレクトリー・ファイルを読み取る回数が減少します。

ディレクトリー・キャッシュをオンにすると、データベース・システム・モニター・スナップショットを獲得するときのパフォーマンスも向上します。スナップショットを呼び出すときは、データベースの別名ではなく、データベースの名前を明示的に参照する必要があります。

注: データベース・マネージャーを開始した後にデータベースのカタログ作成、カタログ解除、作成、または除去された場合、ディレクトリー・キャッシュがオンになっていると、スナップショットを呼び出したときにエラーが発生することがあります。

監査バッファー・サイズ (audit_buf_sz)

構成タイプ データベース・マネージャー

適用範囲

- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー
- ローカル・クライアントをもつデータベース・サーバー
- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー
- ローカル・クライアントをもつサテライト・データベース・サーバー

パラメーター・タイプ 構成可能

デフォルト[範囲] 0 [0 ～ 65 000]

計測単位 ページ (4 KB)

割り振りタイミング DB2 始動時

解放タイミング DB2 停止時

このパラメーターは、データベース監査時に使用されるバッファー・サイズを指定します。監査機能について詳しくは、 *管理の手引き: インプリメンテーション* にある『DB2 アクティビティーの監査』を参照してください。

このパラメーターのデフォルト値はゼロ (0) です。値がゼロ (0) の場合、監査バッファーは使用されません。値がゼロ (0) より大きければ、監査バッファー用のスペースが割り振られます。監査バッファーでは、監査機能によって生成される監査レコードが挿入されます。値に 4 KB ページを乗算すると、監査バッファーに割り振られるスペース量になります。監査バッファーを動的に割り振ることはできません。このパラメーターの新しい値を有効にするには、 DB2 を停止して再始動する必要があります。

このパラメーターをデフォルトからゼロ (0) よりも大きい一定の値に変更すれば、監査機能により、監査レコードを生成するステートメントの実行に対して非同期でディスクにレコードが書き込まれます。こうすれば、パラメーター値をゼロ (0) のままにしておく場合よりも DB2 のパフォーマンスが向上します。値がゼロ (0) であれば、監査レコードを生成するステートメントの実行と同期で (同時に)、監査機能がディスクにレコードを書き込むことになります。監査時の同期操作は、 DB2 で実行するアプリケーションのパフォーマンスを低下させます。

Java インタープリター・ヒープの最大サイズ (java_heap_sz)

構成タイプ データベース・マネージャー

適用範囲

- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー
- クライアント
- ローカル・クライアントをもつデータベース・サーバー
- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー
- ローカル・クライアントをもつサテライト・データベース・サーバー

パラメーター・タイプ 構成可能

デフォルト[範囲] 512 [0 ～ 4 096]

計測単位 ページ (4 KB)

割り振りタイミング Java アプリケーションが開始するとき

解放タイミング Java アプリケーションが完了したとき

関連パラメーター 487ページの『Java Development Kit 1.1 インストール・パス (jdk11_path)』

このパラメーターは、 Java インタープリターが使用するヒープの最大サイズを決定します。

各 DB2 プロセスに 1 つのヒープ (UNIX ベースのプラットフォームの各エージェントまたはサブエージェントに 1 つずつ、および他のプラットフォームのインスタンスに 1 つずつ)、さらに、分離された UDF と分離されたストアード・プロシージャー・プロセスごとに 1 つずつヒープがあります。どの場合でも、 Java UDF またはストアード・プロシージャーを実行するエージェントまたはプロセスだけがこのメモリーを割り振ります。区分データベース・システムでは、各区画で同じ値が使用されます。

ロック

次のパラメーターは、ユーザーの環境でのロック管理を制御します。

- 388ページの『デッドロック検査の時間間隔 (dlchktime)』
- 389ページの『自動調整前のロック・リストの最大パーセント (maxlocks)』
- 390ページの『ロック・タイムアウト (locktimeout)』

357ページの『ロック・リスト用最大ストレージ (locklist)』も参照してください。

51ページの『ロック』 は、データベース・マネージャーがロックを使ってデータの保全性を確保する方法を概説しています。

デッドロック検査の時間間隔 (dlchktime)

構成タイプ データベース

パラメーター・タイプ 構成可能

デフォルト[範囲] 10 000 (10 秒) [1 000 ～ 600 000]

計測単位 ミリ秒

関連パラメーター

- 357ページの『ロック・リスト用最大ストレージ (locklist)』
- 389ページの『自動調整前のロック・リストの最大パーセント (maxlocks)』

デッドロックは、同一データベースに接続した 2 つ以上のアプリケーションが、リソースが空くのを無限に待っているときに生じます。どのアプリケーションも、他のアプリケーションが必要としているリソースを保持しているため、この待ち状態は永久に解決されません。

デッドロック検査間隔は、データベースに接続しているすべてのアプリケーションの間でデッドロックが生じていないかどうかを、データベース・マネージャーが検査する頻度を定義します。

注:

1. 区分データベース環境では、このパラメーターはカタログ・ノードにだけ適用されます。
2. 区分データベース環境では、 2 番目の反復の後まではデッドロックのフラグが付けられません。

推奨事項: このパラメーターを大きくすると、デッドロック検査の頻度が低下するので、デッドロックが解決されるまでアプリケーション・プログラムが待たなければならない時間は長くなります。

このパラメーターを小さくすると、デッドロック検査の頻度が高まるので、デッドロックが解決されるまでアプリケーション・プログラムが待たなければならない時間は短くなりますが、データベース・マネージャーがデッドロックを検査するのにかかる時間は長くなります。デッドロックの間隔が極端に短いと、データベース・マネージャーはデッドロック検出を頻繁に実行することになるので、実行時のパフォーマンスは低下します。並行性を高めるためにこのパラメーターを低く設定する場合、不必要なロック・エスカレーションが行われないよう *maxlocks* と *locklist* を適切に設定する必要があります。適切に設定されていないと、ロック競合がさらに発生し、デッドロック状態がより多く発生することになります。

自動調整前のロック・リストの最大パーセント (maxlocks)

構成タイプ データベース

パラメーター・タイプ 構成可能

デフォルト[範囲]

UNIX 10 [1 ～ 100]

OS/2 および Windows NT 22 [1 ～ 100]

計測単位 パーセンテージ

関連パラメーター

- 357ページの『ロック・リスト用最大ストレージ (locklist)』
- 399ページの『活動アプリケーションの最大数 (maxappls)』

ロック・エスカレーションとは、行ロックの代わりに表ロックを設定し、リスト中のロック数を減らす処理です。このパラメーターは、あるアプリケーションにより設定されたロック・リストのパーセンテージを定義し、そのパーセンテージに達するとデータベース・マネージャーが自動調整を行うようにします。あるアプリケーションにより設定されたロックの数が、ロック・リストの合計サイズに関するこのパーセンテージに達すると、そのアプリケーションにより設定されたロックについて自動調整が行われます。ロック・リストのスペースが足りなくなったときも、ロック・エスカレーションが行われます。

データベース・マネージャーは、ロック・リストでアプリケーションのロックについて調べ、行ロックが一番多い表を見つけることにより、どのロックを自動調整するかを判断します。それらの行ロックの代わりにシングル表ロックを設定して、 *maxlocks* の値より低くなると、ロック・エスカレーションは停止します。まだ高いようであれば、ロック・リストのパーセンテージが *maxlocks* の値より低くなるまでロック・エスカレーションは継続されます。 *maxlocks* パラメーターに *maxappls* パラメーターを掛けた値は、 100 より小さくはなりません。

推奨: *maxlocks* を設定して、平均の 2 倍のロック数をアプリケーションに保持させるために、次の公式を使用することができます。

```
maxlocks = 2 * 100 / maxappls
```

2 は平均の 2 倍を表し、 100 は許される最大パーセンテージを表しています。並行で実行されるアプリケーションが少ない場合は、上記の公式の代わりに次の公式を使用できます。

```
maxlocks = 2 * 100 / (average number of applications running
concurrently)
```

maxlocks の設定時の考慮事項の 1 つに、これをロック・リストのサイズとともに使用するというポイントがあります (*locklist*)。ロック自動調整が行われる前にアプリケーションが保持するロック数の実際の限界は次のようになります。

```
maxlocks * locklist * 4 096 / (100 * 36)
```

4 096 はページ内のバイト数、 100 は *maxlocks* に許された最大パーセンテージ、 36 はロックあたりのバイト数です。アプリケーションの 1 つが 1 000 ロックを必要としていることが分かっていて、ロック自動調整を行いたい場合は、結果が 1 000 を超えるように、 *maxlocks* と *locklist* の値を選択してください。 (*maxlocks* に 10、 *locklist* に 100 を使用すると、この公式の結果は必要とされる 1 000 ロックを超えます。)

maxlocks の値が小さすぎる場合、他の並行アプリケーションのためのロック・スペースがまだ十分にあるときはロック自動調整が行われます。 *maxlocks* の値が大きすぎる場合、少数のアプリケーションがロック・スペースの大部分を占有し、他のアプリケーションはロック自動調整を行う必要があります。この場合、ロック自動調整が必要であるために、並行性が低くなります。

この構成パラメーターのトラックと調整を行うために、データベース・システム・モニターを使用することもできます。

ロック・タイムアウト (locktimeout)

構成タイプ データベース

パラメーター・タイプ 構成可能

デフォルト[範囲] -1 [-1; 0 〜 30 000]

計測単位 秒

関連パラメーター

- 357ページの『ロック・リスト用最大ストレージ (locklist)』
- 389ページの『自動調整前のロック・リストの最大パーセント (maxlocks)』

このパラメーターは、ロックを獲得する場合にアプリケーションが待機する時間を秒単位で指定します。これにより、アプリケーションのグローバル・デッドロックを防ぐことができます。

このパラメーターを 0 に設定すると、ロックは待機されません。この状況下で要求時に使用できるロックがなければ、アプリケーションはただちに -911 を受け取ります。

このパラメーターを -1 に設定すると、ロック・タイムアウト検出はオフになります。この状況下では、以下のいずれかの条件が発生するまで、ロックは (要求時に使用できるロックがない限り) 待機されます。

- ロックが認可される。

- デッドロックが発生する。

推奨事項: トランザクション処理 (OLTP) 環境では、初期値 30 秒から始めることができます。照会しか行わない環境では、もう少し高い値から始めることができます。どちらの場合も、ベンチマーク手法を使ってこのパラメーターを調整してください。

データ・リンク・マネージャーを使用して作業するときに、データ・リンク・マネージャー (dlfm) インスタンスの db2diag.log 内にロック・タイムアウトが存在する場合は、*locktimeout* の値を増やす必要があります。また、 *locklist* の値を増やすことも考慮する必要があります。

トランザクションの停止 (ユーザーがワークステーションから離れたことなどの理由で) などの異常事態が発生した場合に、待ち状態をす早く検出できるよう、この値を設定してください。作業負荷のピーク時はロックの待ち時間も長くなりますが、それが原因で、有効なロック要求がタイムアウトになることがないよう、この値を十分高く設定してください。

データベース・システム・モニターを使用すると、アプリケーション (接続) でロック・タイムアウトが発生した回数、または接続していたすべてのアプリケーションで生じたタイムアウト状態をデータベースが検出した回数をトラックすることができます。詳細は、*システム・モニター 手引きおよび解説書* の中の *locks_timeouts (ロック・タイムアウトの数)* モニター要素に関する説明を参照してください。

lock_timeout モニター要素の値が高い場合は、次の原因が考えられます。
- この構成パラメーターの値が低すぎる。
- アプリケーション (トランザクション) が拡張期間までロックを保持している。これらのアプリケーションについては、データベース・システム・モニターを使ってさらに調査することができます。
- 並行性の問題。ロック・エスカレーション (行レベルのロックから表レベルのロックへの) が原因であることもあります。詳細については、 389ページの『自動調整前のロック・リストの最大パーセント (maxlocks)』および 357ページの『ロック・リスト用最大ストレージ (locklist)』を参照してください。

このパラメーターの使用については、59ページの『ロックの待機とタイムアウト』を参照してください。

入出力およびストレージ

次のパラメーターは、データベースの操作に伴う入出力およびストレージのコストを制御します。

- 392ページの『ページ変更しきい値 (chngpgs_thresh)』
- 392ページの『非同期ページ・クリーナーの数 (num_iocleaners)』
- 394ページの『入出力サーバー数 (num_ioservers)』
- 395ページの『索引分類フラグ (indexsort)』

- 395ページの『順次検出フラグ (seqdetect)』
- 396ページの『デフォルト事前取り出しサイズ (dft_prefetch_sz)』
- 397ページの『SMS コンテナーのデフォルト数 (numsegs)』
- 397ページの『表スペースのデフォルト・エクステント・サイズ (dft_extent_sz)』
- 398ページの『拡張記憶域メモリー・セグメント・サイズ (estore_seg_sz)』
- 398ページの『拡張記憶域メモリー・セグメントの数 (num_estore_segs)』

ページ変更しきい値 (chngpgs_thresh)

構成タイプ データベース

パラメーター・タイプ 構成可能

デフォルト[範囲] 60 [5 ～ 99]

計測単位 パーセンテージ

関連パラメーター 『非同期ページ・クリーナーの数 (num_iocleaners)』

非同期ページ・クリーナーは、データベース・エージェントでバッファー・プールのスペースが必要になる前に、変更されたページをバッファー・プールからディスクに書き込みます。結果として、データベース・エージェントは、変更されたページが書き込まれるのを待って、バッファー・プール内のスペースを使用する必要がなくなります。これにより、データベース・アプリケーション全体のパフォーマンスは向上します。

このパラメーターは、非同期ページ・クリーナーが現在活動していない場合に、このクリーナーを開始するための変更ページのレベル (パーセンテージ) を指定することができます。ページ・クリーナーが開始されると、ディスクに書き込むページのリストが作成されます。ディスクへの書き込みが完了すると、ページ・クリーナーは再び非活動になり、次のトリガーにより開始されるまで待機します。

読み取り専用 (たとえば、照会) 環境では、これらのページ・クリーナーは使用されません。

推奨事項: 更新トランザクション作業負荷が多量にあるデータベースの場合、このパラメーター値をデフォルト値以下に設定すると、バッファー・プールにクリーン・ページを十分確保することができます。デフォルトよりパーセンテージを高くすると、データベースに非常に大きな表が少ししかない場合は、パフォーマンスが向上します。

非同期ページ・クリーナーの数 (num_iocleaners)

構成タイプ データベース

パラメーター・タイプ 構成可能

デフォルト[範囲] 1 [0 ～ 255]

計測単位 カウンター

関連パラメーター

- 348ページの『バッファー・プール・サイズ (buffpage)』
- 392ページの『ページ変更しきい値 (chngpgs_thresh)』

このパラメーターは、データベースあたりの非同期ページ・クリーナーの数を指定します。ページ・クリーナーは、データベース・エージェントでバッファー・プールのスペースが必要になる前に、変更されたページをバッファー・プールからディスクに書き込みます。結果として、データベース・エージェントは、変更されたページが書き込まれるのを待って、バッファー・プール内のスペースを使用する必要がなくなります。これにより、データベース・アプリケーション全体のパフォーマンスは向上します。

パラメーターをゼロ (0) に設定すると、ページ・クリーナーは開始されません。その結果、バッファー・プールからディスクへの書き込みは、すべてデータベース・エージェントが行うようになります。多数の物理ストレージに分かれて保管されているデータベースの場合、それらの装置のいずれかがアイドル状態にある可能性は高いため、このパラメーターの設定はパフォーマンスに重大な影響を及ぼします。ページ・クリーナーを構成しないと、ユーザーのアプリケーションでは、ログがいっぱいになる状態が定期的に発生することになる場合があります。

データベースに対して実行するアプリケーションが、主にデータを更新するトランザクションから構成されている場合、クリーナーの数を増やすとパフォーマンスは向上します。ページ・クリーナーの数を増やすと、ディスク上のデータベースの内容はいつでも最新のものになるので、電源異常などのソフト障害からリカバリーする時間も短くなります。

推奨事項: このパラメーターを設定するときは、以下の要素を考慮に入れてください。

- アプリケーションのタイプ
  - 更新は行わない照会専用のデータベースの場合、このパラメーターを 0 に設定します。照会作業ロードの結果、多数の TEMP 表が作成されている場合は、例外になります (これは、Explain ユーティリティーを使って判別できます)。
  - データベースに対してトランザクションを実行する場合、このパラメーターを、 1 からデータベースが使用している物理ストレージ装置の数までの間の任意の数に設定します。
- 作業負荷

  更新トランザクションの比率が高い環境では、構成するページ・クリーナーの数を増やす必要が生じることがあります。
- バッファー・プール・サイズ (*buffpage*)

  大きなバッファー・プールを持つ環境の場合も、構成するページ・クリーナーの数を増やす必要が生じることがあります。

データベース・システム・モニターを使用すると、イベント・モニターから与えられる、バッファー・プールからの書き出し活動に関する情報を用いて、この構成ファイルを調整することができます。

- 次の条件がどちらも真であれば、パラメーターの値を小さくすることができます。
  - *pool_data_writes* と *pool_async_data_writes* の値がだいたい同じである。
  - *pool_index_writes* と *pool_async_index_writes* の値がだいたい同じである。
- 次の条件のいずれかが真であれば、パラメーターの値を大きくする必要があります。
  - *pool_data_writes* の方が *pool_async_data_writes* よりずっと大きい。
  - *pool_index_writes* の方が *pool_async_index_writes* よりずっと大きい。

詳細については、 *システム・モニター 手引きおよび解説書* の中の以下のモニター要素の説明を参照してください。

- *pool_data_writes (バッファー・プール・データ書き込み)*
- *pool_index_writes (バッファー・プール索引書き込み)*
- *pool_async_data_writes (バッファー・プール非同期データ書き込み)*
- *pool_async_index_writes (バッファー・プール非同期索引書き込み)*

入出力サーバー数 (num_ioservers)

構成タイプ データベース

パラメーター・タイプ 構成可能

デフォルト[範囲] 3 [1 〜 255]

1 [1 〜 255] (ローカル・クライアントをもつサテライト・データベース・サーバー)

計測単位 カウンター

割り振りタイミング アプリケーションがデータベースに接続したとき

解放タイミング アプリケーションがデータベースから切断したとき

関連パラメーター

- 396ページの『デフォルト事前取り出しサイズ (dft_prefetch_sz)』
- 395ページの『順次検出フラグ (seqdetect)』

入出力サーバーは、バックアップや復元などのユーティリティーを使って、データベース・エージェントの代わりに事前取り出し入出力および非同期入出力を実行します。このパラメーターは、 1 つのデータベースに置く入出力サーバーの数を指定します。この数を超えた事前取り出しやユーティリティーの入出力が、データベースに実行されることはありません。入出力サーバーは、それが開始した入出力操作が進行中である間は待機します。事前取り出し以外の入出力は、データベース・エージェントで直接スケジュールされないので、 *num_ioservers* の制限は受けません。

推奨事項: システム内の入出力装置をすべて完全に活用するには、データベースがある物理装置の数より 1 つか 2 つ多い数を指定するのが一般的には最善の選択です。各入出力サーバーのオーバーヘッドは最小限に抑えられ、未使用の入出力サーバーはアイドル状態になるので、追加の入出力サーバーを構成するようお勧めします。

詳細については、253ページの『バッファー・プールへのデータの事前取り出し』 および 256ページの『事前取り出しおよび並列入出力を行うための入出力サーバーの構成』を参照してください。

索引分類フラグ (indexsort)

構成タイプ データベース

パラメーター・タイプ 構成可能

デフォルト[範囲] Yes [Yes; No]

このパラメーターは、索引の作成時に索引キーを分類するかどうかを示します。特にクラスター率またはクラスター係数が低い索引の場合、分類を最初に行うと索引作成のパフォーマンスは向上します。作成時に索引を分類すると、照会のパフォーマンスも向上することがあります。パフォーマンスの向上に伴うコストは、分類を行うのに必要なディスク・スペースが増えることです。最初の段階で分類を行わずに索引を作成した場合と比べると、必要なスペース量は 2 倍にも上ります。

推奨事項: ディスク・スペースが不足していない場合は、デフォルトの設定値 (Yes) を使用してください。この分類で必要になるディスク・スペースは、索引列に ORDER BY 文節を指定して表の索引列について SELECT を実行するのに必要なディスク・スペースとほぼ同じです。

対称マルチプロセッサー (SMP) 環境を持っているときにこのパラメーターに No を指定すると、 SMP 環境で使用できるマルチプロセスは、索引作成時には使えません。

順次検出フラグ (seqdetect)

構成タイプ データベース

パラメーター・タイプ 構成可能

デフォルト[範囲] Yes [Yes; No]

関連パラメーター 396ページの『デフォルト事前取り出しサイズ (dft_prefetch_sz)』

データベース・マネージャーは、入出力をモニターすることができます。順次ページを読み取り中は、入出力事前取り出しを活動化することもできます。このタイプの順次事前取り出しのことを、*順次検出* といいます。データベース・マネージャーが順次検出を実行するかどうかは、 *seqdetect* 構成パラメーターを使って制御することができます。

このパラメーターを No に設定すると、たとえば、表分類、表走査、またはリスト事前取り出しなどで事前取り出しが効果的であるとデータベース・マネージャーが判断したときだけ、事前取り出しが行われます。

推奨事項: ほとんどの場合は、このパラメーターにはデフォルト値を使用してください。順次検出をオフにするのは、重大な照会パフォーマンスの問題が他の調整方法では訂正できない場合だけにしてください。

デフォルト事前取り出しサイズ (dft_prefetch_sz)

構成タイプ データベース

パラメーター・タイプ 構成可能

デフォルト[範囲]

UNIX 32 [0 ～ 32 767]

OS/2 および Windows NT 16 [0 ～ 32 767]

計測単位 ページ

関連パラメーター

- 397ページの『表スペースのデフォルト・エクステント・サイズ (dft_extent_sz)』
- 394ページの『入出力サーバー数 (num_ioservers)』

表スペースを作成するときに、オプションとして PREFETCHSIZE n を指定することができます。 n は、事前取り出しを実行したときにデータベース・マネージャーが読み取るページ数を表しています。 CREATE TABLESPACE ステートメントで事前取り出しサイズを指定しないと、このパラメーターの値がデータベース・マネージャーにより使用されます。

詳細については、253ページの『バッファー・プールへのデータの事前取り出し』を参照してください。

推奨事項: システム・モニター・ツールを使用すると、システムが入出力を待っているときに CPU がアイドル状態になっているかどうかを判別することができます。使用している表スペースの事前取り出しサイズが定義されていない場合には、このパラメーターの値を大きくすると効果的なことがあります。

このパラメーターは、データベース全体についてデフォルトを設定するので、データベースに含まれるすべての表スペースにとって適切な設定にはならない場合があります。たとえば、値を 32 に設定すると、エクステント・サイズが 32 ページの表スペースに

は適切な設定でも、エクステント・サイズが 25 ページの表スペースには適切でない可能性があります。理想的なのは、表スペースごとに事前取り出しサイズを明示的に設定することです。

デフォルトのエクステント・サイズ (*dft_extent_sz*) で定義された表スペースの入出力を最小に抑えるには、このパラメーターを、 *dft_extent_sz* パラメーター値の因数または倍数 (整数) に設定してください。たとえば、*dft_extent_sz* パラメーターが 32 の場合、*dft_prefetch_sz* は 16 (32 の分数) か 64 (32 の倍数) に設定することができます。エクステント・サイズの倍数を事前取り出しサイズとして設定する場合、次の条件が満たされると、データベース・マネージャーは入出力をパラレルに実行することがあります。

- 事前取り出しを行うエクステントが別々の物理装置に置かれている
- 複数の入出力サーバーが構成されている (*num_ioservers*)

SMS コンテナーのデフォルト数 (numsegs)

構成タイプ データベース

パラメーター・タイプ 情報

計測単位 カウンター

このパラメーターは、デフォルトの表スペースに作成されるコンテナーの数を示します。ただし、SMS 表スペースにしか適用されません。このパラメーターは、データベースを作成したときの情報を示します。データベース作成 (CREATE DATABASE) コマンドで明示的に情報を指定したか、暗黙的に指定したかは関係ありません。 CREATE TABLESPACE ステートメントがこのパラメーターを使用することは**ありません**。

詳しくは、*管理の手引き: 計画* にある『データベースの物理ディレクトリー』を参照してください。

表スペースのデフォルト・エクステント・サイズ (dft_extent_sz)

構成タイプ データベース

パラメーター・タイプ 構成可能

デフォルト[範囲] 32 [2 ～ 256]

計測単位 ページ

関連パラメーター 396ページの『デフォルト事前取り出しサイズ (dft_prefetch_sz)』

表スペースを作成するときに、オプションとして `EXTENTSIZE n` を指定することができます。 `n` はエクステント・サイズを表しています。 CREATE TABLESPACE ステートメントでエクステント・サイズを指定しないと、データベース・マネージャーはこのパラメーターの値を使用します。

詳しくは、*管理の手引き: 計画* にある『表スペースの設計と選択』を参照してください。

推奨事項: 多くの場合、表スペースの作成時点で、エクステント・サイズを明示的に指定できます。このパラメーターの値を選択する前に、CREATE TABLESPACE ステートメントで、どのようにエクステント・サイズを明示的に選択するかをあらかじめ理解しておく必要があります。詳しくは、94ページの『照会最適化に対する表スペースの影響』を参照してください。

拡張記憶域メモリー・セグメント・サイズ (estore_seg_sz)

構成タイプ	データベース (Database)
パラメーター・タイプ	構成可能
デフォルト[範囲]	16 000 [0 ～ 1 048 575]
計測単位	ページ
関連パラメーター	『拡張記憶域メモリー・セグメントの数 (num_estore_segs)』

このパラメーターは、データベース内の各拡張メモリー・セグメントのページ数を指定します。このパラメーターは、マシンが仮想アドレス可能メモリーの最大量を超える実アドレス可能メモリーを持っている場合にのみ使用されます。

推奨事項: このパラメーターが効力を持つのは、拡張記憶域が使用可能で、*num_estore_segs* パラメーターで指示されるように使用されるときです。各拡張メモリー・セグメントで使用されるページ数を指定するときは、 *num_estore_segs* パラメーターの検討と修正を行って、拡張メモリー・セグメント数についても考慮する必要があります。拡張記憶域についての詳細は、 278ページの『メモリーの拡張』を参照してください。

拡張記憶域メモリー・セグメントの数 (num_estore_segs)

構成タイプ	データベース (Database)
パラメーター・タイプ	構成可能
デフォルト[範囲]	0 [0 ～ 214 7483 647]
関連パラメーター	『拡張記憶域メモリー・セグメント・サイズ (estore_seg_sz)』

このパラメーターは、データベースで使用できる拡張記憶域メモリー・セグメントの数を指定します。

デフォルトは、拡張記憶域メモリー・セグメントなしです。

推奨事項: このパラメーターは、使用するプラットフォーム環境に最大アドレス・スペースよりも大きいメモリーがあり、そのメモリーを使用したいときに、拡張記憶域メモリー・セグメントを使用できるようにするためにだけ使用してください。セグメント数を指定するときは、各セグメントのサイズについても考慮し、 *estore_seg_sz* パラメーターについて検討と修正を行う必要があります。

num_estore_segs と *estore_seg_sz* の構成パラメーターを共に設定したときは、CREATE/ALTER BUFFERPOOL ステートメントで、どのバッファー・プールが拡張メモリーを使用するのかを指定する必要があります。拡張記憶域についての詳細は、278ページの『メモリーの拡張』を参照してください。

エージェント

次のパラメーターは、並行して実行できるアプリケーションの数を制御し、最適なパフォーマンスが得られるようにします。

- 『活動アプリケーションの最大数 (maxappls)』
- 401ページの『活動アプリケーションの平均数 (avg_appls)』
- 402ページの『アプリケーションごとにオープンするデータベース・ファイルの最大数 (maxfilop)』
- 403ページの『オープンするファイルの最大合計数 (maxtotfilop)』
- 403ページの『エージェントの優先順位 (agentpri)』
- 405ページの『エージェントの最大数 (maxagents)』
- 406ページの『並行エージェントの最大数 (maxcagents)』
- 407ページの『調整エージェントの最大数 (max_coordagents)』
- 408ページの『論理エージェントの最大数 (max_logicagents)』
- 409ページの『エージェント・プール・サイズ (num_poolagents)』
- 410ページの『プール内の初期エージェント数 (num_initagents)』

活動アプリケーションの最大数 (maxappls)

構成タイプ データベース

パラメーター・タイプ 構成可能

デフォルト[範囲]

UNIX 40 [1 ～ 60 000]

OS/2 および Windows NT ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー 20 [1 ～ 60 000]

OS/2 および Windows NT ローカル・クライアントをもつデータベース・サーバー 10 [1 ～ 60 000]

計測単位 カウンター

関連パラメーター

- 405ページの『エージェントの最大数 (maxagents)』
- 407ページの『調整エージェントの最大数 (max_coordagents)』
- 389ページの『自動調整前のロック・リストの最大パーセント (maxlocks)』
- 357ページの『ロック・リスト用最大ストレージ (locklist)』
- 401ページの『活動アプリケーションの平均数 (avg_appls)』

このパラメーターは、データベースに並行接続 (ローカルおよびリモート) できるアプリケーションの最大数を指定します。データベースに接続するアプリケーションごとに私用メモリーがいくらか割り振られるので、並行アプリケーションの数を多くすると、使用されるメモリーも増えます。

このパラメーターには、接続されるアプリケーションの合計数に、 2 フェーズ・コミットまたはロールバックの完了を並行して処理する同じアプリケーションの数を加えた数値、またはそれよりも大きい数値を指定しなければなりません。次に、その合計数に、生じるかもしれない未確定トランザクションの予想数を加えます。未確定トランザクションの詳細については、 *管理の手引き: 計画* の『2 フェーズ・コミット時の問題をリカバリーする』を参照してください。

maxappls に達しているのに、アプリケーションがデータベースに接続しようとすると、エラーがアプリケーションに戻され、最大数のアプリケーションがデータベースに接続していることを示します。

データ・リンク・マネージャーを使用するアプリケーションが多くなる場合は、*maxappls* の値を増やす必要があります。以下の式を使用して、必要な値を計算してください。

```
<maxappls> = 5 * (ノードの数) + (データ・リンク・マネージャーを使用する
                 活動アプリケーションのピーク数)
```

データ・リンク・マネージャーでサポートされている最大値は 2 000 です。

区分データベース環境では、この値は、データベース区画で同時に活動状態にできるアプリケーションの最大数です。このパラメーターは、そのデータベース区画における活動状態アプリケーションの数を、データベース区画サーバーに制限します。これは、このサーバーがそのアプリケーションの調整プログラム・ノードであるか否かには関係ありません。区分データベース環境のカタログ・ノードには、他のタイプの環境の場合よりも大きい *maxappls* 値が必要です。これは、区分データベース環境では、各アプリケーションでカタログ・ノードへの接続が必要になるためです。

推奨事項: *maxlocks* パラメーターを小さくしたり *locklist* パラメーターを大きくしないでこのパラメーターの値を大きくすると、アプリケーションの限界ではなくデータベースのロック制限 (*locklist*) に達してしまうため、ロック・エスカレーション問題が起こることがあります。

アプリケーションの最大数は、ある程度 *maxagents* によっても制御されています。アプリケーションは、利用可能な接続 (*maxappls*) および利用可能なエージェント (*maxagents*) があるときに限り、データベースに接続できます。さらに、アプリケーションの最大数は、 *max_coordagents* 構成パラメーターによっても制御されます。すでに *max_coordagents* に到達している場合は、新しいアプリケーション (すなわち調整エージェント) を開始できないからです。

活動アプリケーションの平均数 (avg_appls)

構成タイプ データベース

パラメーター・タイプ 構成可能

デフォルト[範囲] 1 [1 ～ maxappls]

計測単位 カウンター

関連パラメーター

- 399ページの『活動アプリケーションの最大数 (maxappls)』

このパラメーターは、SQL 最適化プログラムが、選択されたアクセス・プランの実行時に利用可能になるバッファー・プール量を見積もるときに使用します。

推奨事項: 複数ユーザー環境で DB2 を実行する場合、特に複雑な照会や大きなバッファー・プールを使用する場合は、複数の照会ユーザーがシステムを使用しているので、バッファー・プールの使用可能度を控えめに見積もるように SQL 最適化プログラムに指示することができます。

このパラメーターを設定するときは、データベースを通常どおり使用する複合照会アプリケーションの数を見積もる必要があります。この見積もりには、簡単な OLTP アプリケーションは含めないでください。見積もりが容易でない場合は、次の数を乗算することができます。

- データベースに対して実行しているアプリケーションの平均数。データベース・システム・モニターを使って特定時刻のアプリケーション数を調べることができます。また、サンプリング手法を使って一定時間内の平均アプリケーション数を計算することができます。データベース・システム・モニターが提供する情報には、 OLTP アプリケーションと非 OLTP アプリケーションの両方が含まれます。
- 複合照会アプリケーションが占めるパーセンテージの推定値。

最適化プログラムに影響する他の構成パラメーターを調整するときは、このパラメーターも多少調整してください。そうすると、パスを選択するときの差を最小にすることができます。

このパラメーターを変更してから、アプリケーションの再バインドを考慮する必要があります (REBIND PACKAGE コマンドを使用)。

アプリケーションごとにオープンするデータベース・ファイルの最大数 (maxfilop)

構成タイプ データベース

パラメーター・タイプ 構成可能

デフォルト[範囲]

UNIX 64 [2 ～ 1950]

OS/2 および Windows NT 64 [2 ～ 32 768]

計測単位 カウンター

関連パラメーター

- 403ページの『オープンするファイルの最大合計数 (maxtotfilop)』
- 399ページの『活動アプリケーションの最大数 (maxappls)』

このパラメーターは、データベース・エージェントごとにオープンする、ファイル・ハンドルの最大数を指定します。あるファイルを開いたためにこの値を超えてしまった場合、このエージェントが使用しているファイルの一部が閉じられます。 *maxfilop* が小さすぎると、限度を超えないためにファイルを開閉するさいのオーバーヘッドが過度になり、パフォーマンスが低下することがあります。

SMS 表スペースと DMS 表スペース・ファイル・コンテナーは両方とも、データベース・マネージャーとオペレーティング・システムの間の対話でファイルとして扱われるため、ファイル・ハンドルが必要になります。 DMS ファイル表スペースで使用されるコンテナーの数と比較すると、 SMS 表スペースの方が一般的に多くのファイルを使用します。したがって、SMS 表スペースを使用している場合は、 DMS ファイル表スペースで必要な数より大きい値をこのパラメーターに指定する必要があります。

また、エージェントあたりのハンドル数を特定の数に制限して、データベース・マネージャーで使用するファイル・ハンドルの全合計がオペレーティング・システムの制限を超えないようにするために、このパラメーターを使うこともできます。実際の数は、並行して実行しているエージェントの数によって異なります。

オープンするファイルの最大合計数 (maxtotfilop)

構成タイプ データベース・マネージャー

適用範囲

- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー
- ローカル・クライアントをもつデータベース・サーバー
- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー
- ローカル・クライアントをもつサテライト・データベース・サーバー

パラメーター・タイプ 構成可能

デフォルト[範囲] 16 000 [100 ～ 32 768]

計測単位 カウンター

関連パラメーター 402ページの『アプリケーションごとにオープンするデータベース・ファイルの最大数 (maxfilop)』

このパラメーターは、 1 つのデータベース・マネージャー・インスタンスで実行中のエージェントおよび他のスレッドでオープンできるファイルの最大数を定義します。この値を超えた数のファイルをオープンしようとすると、アプリケーションにエラーが戻されます。

注: このパラメーターは UNIX ベースのプラットフォームには適用されません。

推奨事項: このパラメーターを設定するときは、データベース・マネージャー・インスタンスの各データベースで使用できるファイル・ハンドルの数を考慮に入れてください。このパラメーターの上限は、次の方法で見積もることができます。

1. インスタンスのデータベースごとにオープンできるファイル・ハンドルの最大数を計算します。次の式を使用します。

   ```
   maxappls * maxfilop
   ```

2. 上の式の結果を合計し、その数がパラメーターの最大数を超えないかどうか検査します。

新しいデータベースを作成する場合は、このパラメーターの値も再評価してください。

エージェントの優先順位 (agentpri)

構成タイプ データベース・マネージャー

適用範囲

- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー

- ローカル・クライアントをもつデータベース・サーバー
- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー
- ローカル・クライアントをもつサテライト・データベース・サーバー

パラメーター・タイプ 構成可能

デフォルト[範囲]

AIX -1 [41 ～ 125]

その他の UNIX
-1 [41 ～ 128]

Windows NT
-1 [0 - 6]

OS/2 -1 [200 ～ 231; 300 ～ 331; 400 ～ 431]

このパラメーターは、オペレーティング・システムのスケジューラーが、すべてのエージェント、およびその他のデータベース・マネージャー・インスタンス・プロセスやスレッドに指定した優先順位を制御します。区分データベース環境では、調整エージェントとサブエージェントの両方、並列システム・コントローラー、および FCM デーモンも含まれます。この優先順位により、DB2 プロセス、エージェントおよびスレッドに与えられる CPU 時間が、このマシンで実行される他のプロセスやスレッドとの関連で決まります。パラメーターを -1 に設定すると、それは特別なアクションが何も実行されないこと、および通常オペレーティング・システムがすべてのプロセスとスレッドをスケジュールする方法でデータベース・マネージャーがスケジュールされることを示します。このパラメーターに -1 以外の値を設定すると、データベース・マネージャーは、そのパラメーター値に設定された静的優先順位でプロセスとスレッドを作成します。このように、このパラメーターを使うと、ユーザーのマシンで実行するデータベース・マネージャー・プロセスおよびスレッドの優先順位を制御することができます。

このパラメーターを使用してデータベース・マネージャーのスループットを上げることができます。このパラメーターに設定する値は、データベース・マネージャーを実行しているオペレーティング・システムによって異なります。たとえば、UNIX ベースの環境では、数値が小さいと優先順位が高くなります。このパラメーターを 41 と 125 の間の値に設定すると、データベース・マネージャーは、 UNIX 静的優先順位としてこのパラメーターの値が設定されたエージェントを作成します。これは UNIX ベースの環境では重要です。低い数値のものはデータベース・マネージャーに対する高い優先順位が得られる一方、他のプロセス (アプリケーションおよびユーザーも含む) は十分な CPU 時間を獲得できないために遅れる可能性があるからです。したがって、マシンに期待される他の活動とのバランスを考えて、このパラメーターを設定する必要があります。

OS/2 環境では、より大きい数値を設定するとより高い優先順位が得られます。

推奨事項: 最初はデフォルト値を使用してください。この値を使用すると、他のユーザー / アプリケーションとデータベース・マネージャー・スループットとの間で応答時間をちょうどよい状態に調整することができます。

データベースのパフォーマンスを考慮するときは、ベンチマーク技法を使って、このパラメーターの最適な設定を判別することができます。特に CPU の使用率が非常に高い場合、データベース・マネージャー優先順位を高くすると他のユーザー処理のパフォーマンスが重大な影響を受けることがあるので、十分注意を払ってください。データベース・マネージャー・プロセスおよびスレッドの優先順位を高くすると、パフォーマンスは大きく向上します。

注: このパラメーターを UNIX ベースのプラットフォームで非デフォルトに設定すると、管理プログラムを使ってエージェントの優先順位を変更できません。

エージェントの最大数 (maxagents)

構成タイプ

データベース・マネージャー

適用範囲

- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー
- ローカル・クライアントをもつデータベース・サーバー
- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー
- ローカル・クライアントをもつサテライト・データベース・サーバー

パラメーター・タイプ

構成可能

デフォルト[範囲]

200 [1 〜 64 000]

400 [1 〜 64 000] (ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバーの場合)

10 [1 〜 64 000] (ローカル・クライアントをもつサテライト・データベース・サーバー の場合)

計測単位

カウンター

関連パラメーター

- 399ページの『活動アプリケーションの最大数 (maxappls)』
- 406ページの『並行エージェントの最大数 (maxcagents)』
- 407ページの『調整エージェントの最大数 (max_coordagents)』

- 412ページの『DARI プロセスの最大数 (maxdari)』
- 373ページの『コミットされる私用メモリーの最小値 (min_priv_mem)』
- 409ページの『エージェント・プール・サイズ (num_poolagents)』

このパラメーターは、随時アプリケーションの要求を受け付けることができる、利用可能なデータベース・マネージャー・エージェントの最大数を示します。エージェントは調整エージェントまたはサブエージェントのいずれも含まれます。調整エージェントの数を制限したい場合は、 *max_coordagents* パラメーターを使用してください。

メモリーは、エージェントが増えるたびに追加する必要があるので、このパラメーターは、メモリーの制約がある環境でデータベース・マネージャーの使用する合計メモリー量を制限するのに役立ちます。

推奨事項: *maxagents* の値は、並行アクセスできる個々のデータベースに設定された *maxappls* の値の合計数以上に設定してください。 *numdb* パラメーターより大きい場合は、 *maxappls* の最大値と *numdb* の積を用いるのが最も安全です。

エージェントを追加するたびに、データベース・マネージャーの開始時に割り振られたリソースのオーバーヘッドが多少生じます。

並行エージェントの最大数 (maxcagents)

構成タイプ データベース・マネージャー

適用範囲

- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー
- ローカル・クライアントをもつデータベース・サーバー
- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー
- ローカル・クライアントをもつサテライト・データベース・サーバー

パラメーター・タイプ 構成可能

デフォルト[範囲] -1 (*max_coordagents*) [-1; 1 ～ *max_coordagents*]

計測単位 カウンター

関連パラメーター

- 399ページの『活動アプリケーションの最大数 (maxappls)』

- 405ページの『エージェントの最大数 (maxagents)』
- 『調整エージェントの最大数 (max_coordagents)』

並行してデータベース・マネージャー・トランザクションを実行できるデータベース・マネージャー・エージェントの最大数を設定します。このパラメーターは、同時に実行されるアプリケーション活動が多いときのシステムの負荷を制御するのに使用します。たとえば、多数の接続を必要とするがそれらの接続を処理するためのメモリー量が限られているようなシステムがあります。同時に実行される活動が多い状態が続くとオペレーティング・システムで過度のページングが生じる可能性があるような、このような環境では、このパラメーターを調整することは効果的です。

このパラメーターにより、データベースに接続できるアプリケーションの数が制限されることはありません。データベース・マネージャーで並行処理できるデータベース・マネージャー・エージェントの数だけが制限されます。したがって、処理がピークのときのシステム・リソースの使用法を制限することができます。

値 -1 は、この制限が *max_coordagents* であることを示します。

推奨事項: ほとんどの場合は、このパラメーターのデフォルト値はそのまま使用できます。アプリケーションの並行性が原因で問題が生じている場合は、ベンチマーク・テストを使ってこのパラメーターを調整し、データベースのパフォーマンスを最適化することができます。

調整エージェントの最大数 (max_coordagents)

構成タイプ データベース・マネージャー

適用範囲

- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー
- ローカル・クライアントをもつデータベース・サーバー
- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー
- ローカル・クライアントをもつサテライト・データベース・サーバー

パラメーター・タイプ 構成可能

デフォルト[範囲] -1 (*maxagents* ～ *num_initagents*)

[-1, 0 ～ *maxagents*]

区分データベース環境および *intra_parallel* が Yes に設定されている環境では、デフォルトは

maxagents - num_initagents です。その他の環境では、デフォルトは *maxagents* です。このため、区分データベース環境以外では、システムが区画内並列処理用に構成されていなければ、 *max_coordagents* は常に *maxagents* に同じです。

区分データベース環境を持たず、したがって *intra_parallel* パラメーターを使用可能にしない場合は、 *max_coordagents* は *maxagents* に同じでなければなりません。

関連パラメーター

- 410ページの『プール内の初期エージェント数 (num_initagents)』
- 409ページの『エージェント・プール・サイズ (num_poolagents)』
- 405ページの『エージェントの最大数 (maxagents)』
- 474ページの『区画内並列処理機能の使用可能化 (intra_parallel)』

このパラメーターは、ある一時点に区分データベース環境または非区分データベース環境のサーバーに存在できる調整エージェントの最大数を決めます。

データベースに接続される、またはインスタンスに接続される各ローカルまたはリモートのアプリケーションごとに 1 つの調整エージェントが獲得されます。インスタンス接続を必要とする要求としては、 CREATE DATABASE、DROP DATABASE およびデータベース・システム・モニター・コマンドがあります。

論理エージェントの最大数 (max_logicagents)

構成タイプ データベース・マネージャー

パラメーター・タイプ 構成可能

デフォルト[範囲] -1 (*max_coordagents*) [-1; *max_coordagents* ～ 64 000]

このパラメーターは、インスタンスに接続できるアプリケーションの最大数を制御します。通常、アプリケーションごとに調整エージェントが割り当てられます。エージェントは、アプリケーションとデータベースの間の操作を手助けするものです。このパラメーターのデフォルト値を使用する場合、コンセントレーター機能は非活動状態になります。そのため、各エージェント・プロセスは自らの私用メモリーを使って操作を行いますが、データベース・マネージャーおよびデータベース・グローバル・リソース (バッファー・プールなど) は他のエージェントと共用します。このパラメーターに、デフォルトよりも大きい値を設定すると、コンセントレーター機能が活動化状態になります。

コンセントレーターの目的は、 DB2 コネクト・ゲートウェイが 10 000 個を超えるクライアント接続を処理できる程度にまで、クライアント・アプリケーションごとのサーバー・リソースを減らすことです。

DB2 コネクトを XA トランザクション・サポート・コンセントレーターとして使用する方法の詳細と例については、 *DB2 コネクト 使用者の手引き* を参照してください。

値 -1 は、この制限が *max_coordagents* であることを示します。

エージェント・プール・サイズ (num_poolagents)

構成タイプ

データベース・マネージャー

適用範囲

- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー
- ローカル・クライアントをもつデータベース・サーバー
- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー
- ローカル・クライアントをもつサテライト・データベース・サーバー

パラメーター・タイプ

構成可能

デフォルト[範囲]

-1 [-1, 0 ～ *maxagents*]

デフォルトを使用すると、非区分データベースとローカル・クライアントを持つサーバーのための値は、 *maxagents*/50 と *max_querydegree* のいずれか大きいほうになります。

デフォルトを使用すると、非区分データベースとローカルおよびリモートのクライアントを持つサーバーのための値は、 *maxagents*/50 x *max_querydegree* または *maxagents* - *max_coordagents* のいずれか大きいほうになります。

デフォルトを使用すると、データベース区画サーバーのための値は、 *maxagents*/10 x *max_querydegree* または *maxagents* - *max_coordagents* のいずれか大きいほうになります。

関連パラメーター

- 410ページの『プール内の初期エージェント数 (num_initagents)』
- 405ページの『エージェントの最大数 (maxagents)』

- 473ページの『照会の最大並列処理度 (max_querydegree)』
- 407ページの『調整エージェントの最大数 (max_coordagents)』

このパラメーターは、エージェント・プールをどこまで大きくしたいかの指針です (これは DB2 バージョン 2 で使用されていた *max_idleagents* パラメーターに置き換わるものです)。

エージェント・プールにはサブエージェントまたはアイドル・エージェントが入ります。アイドル・エージェントは、並列サブエージェントまたは調整エージェントとして使用することができます。このパラメーターの値で指示された数よりも多いエージェントが作成されると、それらのエージェントは、現行要求の実行の終了時にプールには戻されずに、終了します。

このパラメーターの値が 0 であると、エージェントは必要に応じて作成され、現行要求の実行が終了すると、エージェントも終了します。この値が *maxagents* の場合、プールが関連サブエージェントでいっぱいになると、新規の調整エージェントが作成できないため、サーバーは調整プログラム・ノードとしては使用できません。

推奨事項: 同時に接続するアプリケーションが少ない意思決定支援環境を実行する場合は、エージェント・プールにアイドル・エージェントが多数あるような状態にならないように *num_poolagents* を小さい値に設定してください。

多数のアプリケーションが並行して接続されているトランザクション処理環境を実行する場合は、エージェントの作成と終了を頻繁に行うためのコストがかかることがないように、 *num_poolagents* の値は大きくしてください。

プール内の初期エージェント数 (num_initagents)

構成タイプ データベース・マネージャー

適用範囲

- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー
- ローカル・クライアントをもつデータベース・サーバー
- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー
- ローカル・クライアントをもつサテライト・データベース・サーバー

パラメーター・タイプ 構成可能

デフォルト[範囲] 0 [0 ～ *num_poolagents*]

関連パラメーター

- 405ページの『エージェントの最大数 (maxagents)』
- 409ページの『エージェント・プール・サイズ (num_poolagents)』
- 407ページの『調整エージェントの最大数 (max_coordagents)』

このパラメーターは、 DB2START 時にエージェント・プールに作成されるアイドル・エージェントの初期数を決めます。

ストアード・プロシージャー (DARI)

次のパラメーターは、データベース・アプリケーション・リモート・インターフェース (DARI) アプリケーションを制御します。

- 『DARI プロセス保持標識 (keepdari)』
- 412ページの『DARI プロセスの最大数 (maxdari)』
- 413ページの『JVM による DARI プロセスの初期化 (initdari_jvm)』
- 414ページの『プール内の分離 DARI プロセスの初期数 (num_initdaris)』

注: 用語 DARI は、ストアード・プロシージャーのことです。

DARI プロセス保持標識 (keepdari)

構成タイプ データベース・マネージャー

適用範囲

- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー
- ローカル・クライアントをもつデータベース・サーバー
- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー
- ローカル・クライアントをもつサテライト・データベース・サーバー

パラメーター・タイプ 構成可能

デフォルト[範囲] Yes [Yes; No]

関連パラメーター 412ページの『DARI プロセスの最大数 (maxdari)』

このパラメーターは、 DARI 呼び出しが完了した後、 DARI プロセスを保持するか否かを指示します。 DARI プロセスは、ユーザー作成の DARI コードをデータベース・マネージャー・エージェント・プロセスと分離するために、別個のシステム・エンティティーとして作成されます。このパラメーターは、データベース・サーバーだけに適用されます。

keepdari を *no* に設定すると、 DARI 呼び出しのたびに新しい DARI プロセスが作成され、破棄されます。 *keepdari* を *yes* に設定すると、以後の DARI 呼び出しでも DARI プロセスが再利用されます。データベース・マネージャーが停止すると、未解決な DARI プロセスはすべて終了します。

このパラメーターを *yes* に設定すると、DARI プロセスを活動化するたびに、追加のシステム・リソースがデータベース・マネージャーのために消費されます。その数の最大値は、*maxdari* パラメーターの値です。このような状態は、後続の DARI 呼び出しを処理するのに利用可能な DARI プロセスが存在しない場合に生じます。 *maxdari* が 0 に設定されていると、このパラメーターは無視されます。

推奨事項: 非 DARI 要求の数に比べて DARI 要求の数が多い環境で、システム・リソースが制約されていない場合には、このパラメーターは *yes* に設定することができます。呼び出しの処理には既存の DARI プロセスが使用されるので、上記のように設定すると、 DARI プロセスの初期作成時のオーバーヘッドを避けることができます。結果として DARI のパフォーマンスは向上します。

たとえば、OLTP 貸借取引トランザクション・アプリケーションの場合、各トランザクションを実行するコードをストアード・プロシージャーに入れて、そのプロシージャーを DARI プロセスで実行することができます。このアプリケーションの場合、主な作業負荷は DARI プロセスで実行されます。このパラメーターを *no* に設定すると、トランザクションごとに新規の DARI プロセスが作成されるため、オーバーヘッドが生じ、結果としてパフォーマンスは著しく低下します。このパラメーターを *yes* に設定すると、各トランザクションは既存の DARI プロセスを使用しようとするので、オーバーヘッドは生じません。

DARI プロセスの最大数 (maxdari)

構成タイプ データベース・マネージャー

適用範囲

- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー
- ローカル・クライアントをもつデータベース・サーバー
- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー
- ローカル・クライアントをもつサテライト・データベース・サーバー

パラメーター・タイプ 構成可能

デフォルト[範囲] -1 (*max_coordagents*)
[-1; 0 ～ *max_coordagents*]

計測単位 カウンター

関連パラメーター

- 405ページの『エージェントの最大数 (maxagents)』
- 411ページの『DARI プロセス保持標識 (keepdari)』
- 414ページの『プール内の分離 DARI プロセスの初期数 (num_initdaris)』
- 407ページの『調整エージェントの最大数 (max_coordagents)』

このパラメーターは、データベース・サーバーに入れられる DARI プロセスの最大数を示します。この制限に達すると、新規の DARI 要求を呼び出すことができなくなることがあります。このパラメーターは、データベース・サーバーだけに適用されます。

1 つの調整エージェントには 1 つの DARI プロセスしか活動できません。したがって、DARI プロセスの最大数は調整エージェントの最大数 (*max_coordagents*) によって決まります。

推奨事項: データベース・マネージャーで DARI 機能を使用できる環境では、このパラメーターを使って、任意の一時点でデータベース・マネージャー内で DARI 呼び出しを処理できるように、適切な数の DARI プロセスを使用可能にしておくことができます。

このパラメーターを -1 に設定すると、 DARI プロセスの最大数は、*max_coordagents* パラメーターに設定された値と同じになります。

不適切な量のシステム・リソースが DARI プロセスに与えられており、データベース・マネージャーのパフォーマンスが影響を受けているため、使用中の環境でデフォルト値を使用するのが適切ではないと判断した場合、次の式を用いて、パラメーターを調整する際の開始点を決定することができます。

```
maxdari = 同時に DARI 呼び出しを行うことのできるアプリケーションの数
```

keepdari が *yes* に設定されていると、作成される個々の DARI プロセスは、DARI 呼び出しが処理されてエージェントに戻された後も引き続き存在し、システム・リソースを使用し続けます。

使用中の環境にさまざまな制限があるため、処理リソースを DARI に関連付けることができない場合は、このパラメーターをゼロ (0) に設定して DARI を使用不能にすることができます。

JVM による DARI プロセスの初期化 (initdari_jvm)

構成タイプ データベース・マネージャー

適用範囲

- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー
- ローカル・クライアントをもつデータベース・サーバー
- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー
- ローカル・クライアントをもつサテライト・データベース・サーバー

パラメーター・タイプ 構成可能

デフォルト[範囲] No [Yes; No]

関連パラメーター

- 412ページの『DARI プロセスの最大数 (maxdari)』
- 『プール内の分離 DARI プロセスの初期数 (num_initdaris)』
- 411ページの『DARI プロセス保持標識 (keepdari)』

このパラメーターは、それぞれの分離 DARI プロセスが始動時に Java 仮想マシン (JVM) をロードするかどうかを示します。このパラメーターにより、分離 Java ストアード・プロシージャーの最初の起動時間は減少します (特に、 *num_initdaris* パラメーターと一緒に使用した場合)。 Java 以外の分離ストアード・プロシージャーの場合、JVM は必要ないので、このパラメーターによって初期ロード時間が増加する可能性があります。

プール内の分離 DARI プロセスの初期数 (num_initdaris)

構成タイプ データベース・マネージャー

適用範囲

- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー
- ローカル・クライアントをもつデータベース・サーバー
- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー
- ローカル・クライアントをもつサテライト・データベース・サーバー

パラメーター・タイプ 構成可能

デフォルト[範囲] 0 [0 ～ *maxdari*]

関連パラメーター

- 412ページの『DARI プロセスの最大数 (maxdari)』
- 413ページの『JVM による DARI プロセスの初期化 (initdari_jvm)』
- 411ページの『DARI プロセス保持標識 (keepdari)』

このパラメーターは、 DB2START 時に DARI プールに作成される分離 DARI プロセスの初期数を示します。このパラメーターを設定すると、分離ストアード・プロシージャーの最初の起動時間が減少します。 *keepdari* が指定されていないと、このパラメーターは無視されます。

ロギングおよびリカバリー

重要なデータが失われないようにするには、環境のリカバリーが非常に大切になります。環境を管理し、十分なレベルまでデータまたはトランザクションをリカバリーするのに使用できるパラメーターがいくつかあります。この種のパラメーターは、次のカテゴリーに分けることができます。

- 『データベース・ログ・ファイル』
- 421ページの『データベース・ログ・アクティビティー』
- 427ページの『リカバリー』
- 433ページの『分散作業単位リカバリー』

データベース・ログ・ファイル

次のパラメーターは、データベースのロギングで使用するファイルの数、サイズおよび状況について記述しています。

- 『ログ・ファイルのサイズ (logfilsiz)』
- 417ページの『1 次ログ・ファイルの数 (logprimary)』
- 419ページの『2 次ログ・ファイルの数 (logsecond)』
- 419ページの『データベース・ログ・パスの変更 (newlogpath)』
- 421ページの『ログ・ファイルの場所 (logpath)』
- 421ページの『最初のアクティブ・ログ・ファイル (loghead)』

ログ・ファイルのサイズ (logfilsiz)

構成タイプ データベース

パラメーター・タイプ 構成可能

デフォルト[範囲]

UNIX 1000 [4 〜 65 535]

Windows NT 250 [4 ～ 65 535]

OS/2 250 [4 ～ 65 535]

計測単位 ページ (4 KB)

関連パラメーター

- 417ページの『1 次ログ・ファイルの数 (logprimary)』
- 419ページの『2 次ログ・ファイルの数 (logsecond)』
- 423ページの『リカバリー範囲とソフト・チェックポイント間隔 (softmax)』

このパラメーターは、個々の 1 次および 2 次ログ・ファイルのサイズを定義します。これらのログ・ファイル・サイズは、ログ・ファイルに書き込むことができるログ・レコードの数を制限します。ログ・ファイルがいっぱいになると、新しいログ・ファイルが必要になります。

1 次および 2 次ログ・ファイルの使用法、またログ・ファイルがいっぱいになったときのアクションは、実行しているロギングのタイプによって異なります。

- 循環ログ

  1 次ログ・ファイルに記録されていた変更をコミットすると、その 1 次ログ・ファイルを再利用することができます。ログ・ファイルのサイズが小さいのに、アプリケーションがデータベースに対する多数の変更内容をコミットせずに処理すると、 1 次ログ・ファイルはすぐにいっぱいになってしまいます。 1 次ログ・ファイルがすべていっぱいになると、データベース・マネージャーは新しいログ・レコードを保持するために 2 次ログ・ファイルを割り振ります。
- ログ保存のロギング

  1 次ログ・ファイルがいっぱいになると、そのログはアーカイブされ、新しい 1 次ログ・ファイルが割り振られます。

推奨事項: ログ・ファイルのサイズは、1 次ログ・ファイルの数を考慮に入れて決めてください。

- データベースに対する更新、削除、挿入トランザクションが大量なために、ログ・ファイルがすぐにいっぱいになってしまう場合は、 *logfilsiz* の値を大きくしてください。

  注: ログ・ファイルの合計サイズの限界は、32 GB です。つまり、ログ・ファイルの数 (*logprimary* + *logsecond*) に各ログ・ファイルのサイズ (バイト単位) (*logfilsiz* * 4096) を掛けた値が、 32 GB より小さくなければなりません。

  ログ・ファイルが小さすぎると、古いログ・ファイルのアーカイブ、新しいログ・ファイルの割り振り、および使用可能なログ・ファイルができるまでの待機などでオーバーヘッドが生じるため、システム・パフォーマンスに影響が及ぶことがあります。

- 1 次ログは、ここで設定したサイズで事前に割り振られるので、ディスク・スペースが足りない場合は *logfilsiz* の値を小さくしてください。

  ログ・ファイルが大きすぎると、メディアがログ・ファイル全体を保持できないことがあるので、アーカイブされたログ・ファイルやログ・ファイルのコピーを管理するときの柔軟性が低くなることがあります。

ログアーカイブを使用する場合は、最後のアプリケーションがデータベースから切断されると、現行のアクティブ・ログ・ファイルはクローズされ、切り捨てられます。データベースに次回接続するときには、次のログ・ファイルが使用されます。したがって、並行アプリケーションのロギング要件を知っていると、過度の量を割り振ってスペースを無駄にすることがないようログ・ファイル・サイズを決定することができます。

詳しくは、 *管理の手引き: インプリメンテーション* の『データベース・ロギングの構成パラメーター』の中のこのパラメーターの説明を参照してください。

1 次ログ・ファイルの数 (logprimary)

構成タイプ データベース

パラメーター・タイプ 構成可能

デフォルト[範囲] 3 [2 ～ 128]

計測単位 カウンター

割り振りタイミング

- データベースを作成したとき
- ログを別の場所に移動したとき (つまり、 *logpath* パラメーターを更新したとき)
- このパラメーター (*logprimary*) の値を大きくしてから、すべてのユーザーが切断した後に、データベースに次回接続したとき
- ログ・ファイルがアーカイブされ、新規のログ・ファイルが割り振られたとき (*logretain* または *userexit* パラメーターが使用可能になっていなければなりません)
- *logfilsiz* を変更した場合には、すべてのユーザーが切断した後にデータベースに次回接続する時点でアクティブ・ログ・ファイルがサイズ変更されます。

解放タイミング このパラメーターの値を小さくするまで解放されません。値を小さくすると、データベースに次回接続したときに不要なログ・ファイルが削除されます。

関連パラメーター

- 415ページの『ログ・ファイルのサイズ (logfilsiz)』
- 419ページの『2 次ログ・ファイルの数 (logsecond)』
- 425ページの『ログの保存可能 (logretain)』
- 426ページの『ユーザー出口可能 (userexit)』

1 次ログ・ファイルは、リカバリー・ログ・ファイルに割り振るストレージの固定量を確立します。このパラメーターを使用して、事前に割り振る 1 次ログ・ファイルの数を指定することができます。

循環ログの場合、 1 次ログは繰り返し順序どおりに使用されます。つまり、ログがいっぱいになると、次の順序の 1 次ログが利用可能であれば、それが使用されます。ログ・レコードを含む作業単位がすでにコミットまたはロールバックされていると、ログは利用可能と見なされます。次の順序の 1 次ログが利用可能でないと、 2 次ログが割り振られて使用されます。次の順序の 1 次ログが利用可能になるまで、または *logsecond* パラメーターで設定された限界に達するまで、追加の 2 次ログが割り振られます。データベース・マネージャーでこれらの 2 次ログ・ファイルが必要なくなると、動的に割り振り解除されます。

1 次および 2 次ログ・ファイルは、次の条件を満たしていなければなりません。
- (*logprimary* + *logsecond*) <= 128

推奨事項: このパラメーターの値を設定するときは、使用しているロギングのタイプ、ログ・ファイルのサイズ、および処理環境のタイプ (たとえば、トランザクションの長さやコミットの頻度) など、数多くの要素を考慮に入れなければなりません。

この値を大きくすると、データベースに最初に接続した時点で 1 次ログ・ファイルが事前割り振りされるので、ログのディスク所要量は増えます。

2 次ログの割り振りが頻繁に行われるようであれば、ログ・ファイルのサイズ (*logfilsiz*) または 1 次ログの数を増やすことによって、システムのパフォーマンスを改善することもできます。

頻繁にアクセスしないデータベースについては、ディスク装置を節約するため、パラメーターを 2 に設定してください。ロールフォワード・リカバリーが使用できるデータベースの場合には、新規のログを即時に割り振る際にオーバーヘッドが生じないよう、このパラメーターの値は 2 より大きく設定してください。

1 次ログ・ファイルのサイズを設定する際は、データベース・システム・モニターを利用することができます。

詳細については、 システム・モニター *手引きおよび解説書* の中の以下のモニター要素の説明を参照してください。
- *sec_log_used_top (使用している 2 次ログ・スペースの最大量)*
- *tot_log_used_top (使用しているログ・スペース合計の最大量)*

- *sec_logs_allocated (現在割り振られている 2 次ログ)*

これらのモニター値を少しの時間観察すると平均値が読み取れますが、この平均値は現在の要件をよく表しているので、調整する際に役立ちます。

2 次ログ・ファイルの数 (logsecond)

構成タイプ データベース

パラメーター・タイプ 構成可能

デフォルト[範囲] 2 [0 ～ 126]

計測単位 カウンター

割り振りタイミング *logprimary* が不足して必要が生じたとき (後述の詳細を参照)

解放タイミング 必要がなくなったとデータベース・マネージャーが判断したとき

関連パラメーター

- 415ページの『ログ・ファイルのサイズ (logfilsiz)』
- 417ページの『1 次ログ・ファイルの数 (logprimary)』
- 425ページの『ログの保存可能 (logretain)』
- 426ページの『ユーザー出口可能 (userexit)』

このパラメーターは、リカバリー・ログ・ファイル用に (必要に応じて) 作成されて使用される 2 次ログ・ファイルの数を指定します。 1 次ログ・ファイルがいっぱいになると、 *logfilsiz* で指定されたサイズの 2 次ログ・ファイルが必要に応じて一度に 1 つずつ割り振られます。このパラメーターは、その最大数を制御します。このパラメーターで指定した数より多くの 2 次ログ・ファイルが必要になると、エラー・コードがアプリケーションに戻され、データベースはシャットダウンされます。

2 次ログの使用法については、417ページの『1 次ログ・ファイルの数 (logprimary)』を参照してください。

推奨方法: 2 次ログ・ファイルは、定期的に大量のログ・スペースが必要になるデータベースに使用してください。たとえば、月に 1 回だけ実行するアプリケーションの場合、 1 次ログ・ファイルのログ・スペースより大きなスペースが必要になる可能性があります。 2 次ログ・ファイルは永続的なファイル・スペースを必要としないので、このような状況では利点が多いといえます。

データベース・ログ・パスの変更 (newlogpath)

構成タイプ データベース

パラメーター・タイプ 構成可能

デフォルト[範囲] Null [任意の有効なパスまたは装置]

関連パラメーター

- 421ページの『ログ・ファイルの場所 (logpath)』
- 445ページの『データベースの一貫性 (database_consistent)』

このパラメーターでは、最大 242 バイトのストリングを指定して、ログ・ファイルを保管する場所を変更できます。このストリングはパス名を指す場合もあれば、ロー・デバイスを指す場合もあります。ストリングがパス名を指す場合は、相対パス名ではなく、完全修飾パス名を指定してください。

注: 区分データベース環境では、パスには自動的にノード番号が付けられます。ノード番号は、複数の論理ノード構成において、パスを固有にしておくために付けられます。

装置を指定するには、オペレーティング・システムが装置として識別するストリングを指定します。たとえば、次のとおりです。

- Windows NT では、 `¥¥.¥d:` または `¥¥.¥PhysicalDisk5`

  注: 装置にログを書き込むには、 Windows NT バージョン 4.0 (サービス・パック 3 以降付き) をインストールしておく必要があります。

- UNIX ベースのプラットフォームでは、 `/dev/rdblog8`

注: 装置の指定は、AIX、Windows 2000、Windows NT、 Solaris、HP-UX、NUMA-Q、および Linux のプラットフォームでのみ可能です。

次の条件が両方とも満たされないと、新しい設定は *logpath* の値に反映されません。

- データベースが一貫していることを *database_consistent* パラメーターが示している
- すべてのユーザーがデータベースから切断している

初めてデータベースに新規接続した時点で、データベース・マネージャーはログを *logpath* が指定した新しい場所に移動します。

以前のログ・パスにログ・ファイルが存在し、そのようなログ・ファイルがアーカイブされていない場合があります。そのようなログ・ファイルは手動でアーカイブしなければなりません。また、そのデータベースで複製を実行している場合、複製では、ログ・パスの変更以前のログ・ファイルも引き続き必要です。ユーザー出口可能 (*userexit*) データベース構成パラメーターを Yes に設定してデータベースを構成し、すべてのログ・ファイルが DB2 によって自動的に、あるいは手動でアーカイブされている場合、 DB2 がログ・ファイルの検索を行い、複製プロセスを完了することができます。それ以外の場合は、手動でそれらのファイルを以前のログ・パスから新しいログ・パスへコピーできます。

推奨事項: 理想的には、ログ・ファイルは入出力が多く**ない**物理ディスクに置きます。たとえば、オペレーティング・システムや高ボリューム・データベースと同じディスクにログを置かないでください。このようにすると、入出力に関連したオーバーヘッドは最小に抑えられ、ロギング・アクティビティーは向上します。

データベース・システム・モニターを使って、データベースのロギングに関連した入出力の数をトラックすることができます。

詳細については、 *システム・モニター 手引きおよび解説書* にある以下のモニター要素の説明を参照してください。
- *log_reads* (読み取られたログ・ページ数)
- *log_writes* (書き込まれたログ・ページ数)

これらのデータ要素は、データベース・ロギングに関連する入出力活動量を戻します。オペレーティング・システム・モニター・ツールを使用して、他のディスクの入出力活動を収集し、この 2 つの入出力活動を比較することができます。

ログ・ファイルの場所 (logpath)

構成タイプ データベース

パラメーター・タイプ 情報

関連パラメーター 419ページの『データベース・ログ・パスの変更 (newlogpath)』

このパラメーターは、ロギングで使用するカレント・パスを示します。このパラメーターは、 *newlogpath* パラメーターに対する変更が有効になった後でデータベース・マネージャーにより設定されるため、ユーザーが直接変更することはできません。

データベースを作成すると、そのデータベースが含まれているディレクトリーのサブディレクトリーに、リカバリー・ログ・ファイルが作成されます。デフォルトでは、そのサブディレクトリーには SQLOGDIR という名前が付けられ、データベースのディレクトリーに置かれます。

最初のアクティブ・ログ・ファイル (loghead)

構成タイプ データベース

パラメーター・タイプ 情報

このパラメーターには、現在活動状態になっているログ・ファイルの名前が入ります。

データベース・ログ・アクティビティー

次のパラメーターは、データベース・ロギングのタイプおよびパフォーマンスを制御します。

- 422ページの『グループ化するコミット数 (mincommit)』

- 423ページの『リカバリー範囲とソフト・チェックポイント間隔 (softmax)』
- 425ページの『ログの保存可能 (logretain)』
- 426ページの『ユーザー出口可能 (userexit)』

グループ化するコミット数 (mincommit)

構成タイプ	データベース
パラメーター・タイプ	構成可能
デフォルト[範囲]	1 [1 〜 25]
計測単位	カウンター

このパラメーターを使うと、最低限の回数のコミットが実行されるまで、ログ・レコードのディスク書き込みを遅らせることができるようになります。この遅延によりログ・レコード書き込みに関連するデータベース・マネージャーのオーバーヘッドが少なくなります。そのため、データベースに対して複数のアプリケーションを実行しており、それらのアプリケーションによって非常に短い時間フレームに多数のコミットが要求される場合のパフォーマンスを向上させることができます。

このようにコミットがグループ化されることは、このパラメーターの値が 1 より大きく、データベースに接続したアプリケーションの数がこのパラメーターの値と同じかそれよりも大きい場合だけ行われます。コミットがグループ化されている間、アプリケーションから出されるコミット要求は、1 秒が経過するまで、またはコミット要求の数がこのパラメーターの値に達するまで保留されます。

このパラメーターに指定した値はただちに反映されます。データベースからすべてのアプリケーションが切断されるまで待つ必要はありません。

推奨事項: 複数の読み取り / 書き込みアプリケーションが通常並行してデータベース・コミットを行うように要求する場合は、この値をデフォルト値より大きくしてください。このようにすることにより、データベース・コミットが同時に行われる頻度が低くなるので、ロギング・ファイルの入出力がより効率的になり、同時にデータベース・コミットが要求されるたびに、より多くのログ・レコードを書き込むことができるようになります。

1 秒あたりのトランザクション数をサンプルとして調べ、 1 秒あたりのトランザクション数のピーク (またはそれに近い数) に対応できるようこのパラメーターを調整することもできます。ピーク時のアクティビティーに対応できると、トランザクション集中時の、ログ・レコード書き込みに関連したオーバーヘッドを最小に抑えることができます。

mincommit の値を大きくした場合、いっぱいになったログ・バッファーがトランザクション集中時に書き込まれるのを避けるために、 *logbufsz* パラメーターも大きくする必要があるかもしれません。この場合、*logbufsz* が次の計算値と等しくなるようにしてください。

```
mincommit * (トランザクションで使用される平均ログ・スペース)
```

データベース・システム・モニターを使用して、次の方法でパラメーターを調整することができます。

- 1 秒あたりのトランザクション数のピークを計算する

  サンプルとして通常の 1 日をモニターして、トランザクション集中時に相当する時間を調べることができます。次のモニター要素を追加すると、トランザクションの合計を計算することができます。
  - *commit_sql_stmts (試行されたコミット・ステートメント)*
  - *rollback_sql_stmts (試行されたロールバック・ステートメント)*

  この情報および利用可能なタイム・スタンプを使って、1 秒あたりのトランザクション数を計算することができます。
- トランザクションごとのログ・スペースを計算する

  次のモニター要素を使って、一定時間および一定数のトランザクションに関してサンプルを調べると、使用されるログ・スペースの平均を計算することができます。
  - *log_space_used (使用中の作業単位ログ・スペース)*

  データベース・システム・モニターの詳細については、 システム・モニター *手引きおよび解説書* を参照してください。

リカバリー範囲とソフト・チェックポイント間隔 (softmax)

構成タイプ データベース

パラメーター・タイプ 構成可能

デフォルト[範囲] 100 [1 ～ 100 * *logprimary*]

計測単位 1 次ログ・ファイルのサイズのパーセンテージ

関連パラメーター

- 415ページの『ログ・ファイルのサイズ (logfilsiz)』
- 417ページの『1 次ログ・ファイルの数 (logprimary)』

このパラメーターは、次のために使用されます。

- 破損 (電源障害など) の後でリカバリーの必要があるログ数を決めるため。たとえば、デフォルト値を使用すると、データベース・マネージャーはリカバリーする必要

のあるログ数を 1 に保とうとします。このパラメーターの値として 300 を指定すると、データベース・マネージャーは、リカバリーする必要のあるログ数を 3 に保とうとします。

破損リカバリーに必要なログ数を有効にするために、データベース・マネージャーはこのパラメーターを使用して、ページ・クリーナーを活動させ、指定されたリカバリー・ウィンドウよりも古いページがすでにディスクに書き込まれていることを確認します。

- ソフト・チェックポイントの頻度を決定するため。

電源障害などの原因でデータベース障害が発生した場合、次のような変更がデータベースに加えられていることがあります。

- コミットされていないが、バッファー・プールのデータは更新した
- コミットされたが、バッファー・プールからディスクに書き込まれていない
- コミットされ、バッファー・プールからディスクに書き込まれた

データベースを再始動すると、データベースの破損をリカバリーしてデータベースの一貫性を保つ (つまり、コミットされたすべてのトランザクションがデータベースに適用され、コミットされていないトランザクションが適用されない状態) ためにログ・ファイルが使用されます。

ログ・ファイルのどのレコードをデータベースに適用する必要があるかを判別するために、データベース・マネージャーはログ制御ファイルを使用します。このログ制御ファイルは定期的にディスクに書き出されます。このイベントの頻度に基づいて、データベース・マネージャーは、コミットされたトランザクションのログ・レコードを適用するか、またはすでにバッファー・プールからディスクに書き出されている変更を記述しているログ・レコードを適用することができます。これらのログ・レコードはデータベースには影響しませんが、これらのログ・レコードを適用しようとすると、データベースの再始動処理で多少のオーバーヘッドが生じます。

ログ・ファイルがいっぱいになったとき、およびソフト・チェックポイントをとるときには、ログ制御ファイルは必ずディスクに書き込まれます。この構成パラメーターを使って、追加のソフト・チェックポイントのトリガーとすることができます。

ソフト・チェックポイントの時点は、「現在の状態」と「記録済み状態」の差によって決まります。これは *logfilsiz* のパーセンテージとして示されます。「記録済み状態」は、ディスクのログ制御ファイル中で一番古い有効なレコードにより判別され、「現在の状態」はメモリー上のログ制御情報によって判別されます。 (一番古い有効ログ・レコードは、リカバリー処理で最初に読み取られるログ・レコードです。) 次の式で計算した値がこのパラメーターの値より大きいか等しい場合に、ソフト・チェックポイントが使用されます。

```
( (space between recorded and current states) / logfilsiz ) * 100
```

推奨事項: 受け入れ可能なリカバリー・ウィンドウが 1 ログ・ファイルより大きいか小さいかに基づいて、このパラメーターの値を大きくしたり小さくしたりすることができ

ます。このパラメーターの値を小さくすると、データベース・マネージャーはページ・クリーナーをより頻繁に使用するとともに、ソフト・チェックポイントもより頻繁に行うことになります。これによって、処理する必要のあるログ・レコードの数、および、破損リカバリーで処理される冗長ログ・レコードの数を少なくすることができます。

しかし、ページ・クリーナーの使用が多くなり、ソフト・チェックポイントの頻度が高くなると、データベースのロギングに関連したオーバーヘッドが増加するため、データベース・マネージャーのパフォーマンスに影響が出ることに注意してください。さらに、次の状態では、ソフト・チェックポイントの頻度が高くなってもデータベースの再始動にかかる時間は短くなりません。

- トランザクションは非常に長いが、コミット点はほとんどない。
- バッファー・プールが非常に大きいため、コミット・トランザクションを含むページがディスクに書き込まれる頻度が低い。 (この状態を避けるためには、非同期ページ・クリーナーが有効です。 392ページの『非同期ページ・クリーナーの数 (num_iocleaners)』を参照してください。)

どちらの状態も、メモリーに保持されているログ制御情報は頻繁に変更されないので、未変更のログ制御情報をディスクに書き込んでもあまり効果はありません。

ログの保存可能 (logretain)

構成タイプ データベース

パラメーター・タイプ 構成可能

デフォルト[範囲] No [Recovery; Capture; No]

関連パラメーター

- 426ページの『ユーザー出口可能 (userexit)』
- 446ページの『ログ保存状況表示 (log_retain_status)』
- 445ページの『バックアップ保留標識 (backup_pending)』

値は以下のとおりです。

- No - ログは保存されません。
- Recovery - ログは保存され、順方向リカバリーに使用できます。また、データ複製を使用していれば、収集プログラムは、ログにレコードされている更新を変更表に書き込むことができます。
- Capture - 収集プログラムが更新を変更表に書き込む目的でのみログが保存されます。データ複製収集プログラムが使用した後、これらのログを枝取りしないでおけば、フォワード・リカバリーの際に使用することができます。

logretain が Recovery に設定されているか、または *userexit* が Yes に設定されていると、アクティブ・ログ・ファイルはロールフォワード・リカバリーで使用できるようアーカイブされ、オンライン・アーカイブ・ログ・ファイルになります。このアクティビティーは、ログ保存ロギングといいます。

logretain を Recovery に設定した後、または *userexit* を Yes に設定した後 (または両方の後) には、データベースの全バックアップを実行する必要があります。この状態は、*backup_pending* フラグ・パラメーターで示されます。

logretain が No に設定されていて *userexit* が No に設定されている場合、ロールフォワード・リカバリーはデータベースで使用できません。

logretain が Capture に設定されている際、データ複製収集プログラムがそのログ・ファイルを使い切ると、 PRUNE LOGFILE コマンドを呼び出して、ログ・ファイルを削除します。データベースでロールフォワード・リカバリーを実行する場合には、 *logretain* を Capture に設定しないでください。

logretain が No に設定されていて *userexit* が No に設定されていると、ログは保存されません。この場合、データベース・マネージャーは *logpath* ディレクトリー内のログ・ファイルをすべて (オンライン・アーカイブ・ログ・ファイルも含めて) 削除し、新規のアクティブ・ログ・ファイルを割り振り、循環ログに戻ります。

ユーザー出口可能 (userexit)

構成タイプ データベース

パラメーター・タイプ 構成可能

デフォルト[範囲] No [Yes; No]

関連パラメーター

- 425ページの『ログの保存可能 (logretain)』
- 446ページの『ユーザー出口状況表示 (user_exit_status)』
- 445ページの『バックアップ保留標識 (backup_pending)』

このパラメーターを使用可能にすると、*logretain* パラメーターの設定値に関係なくログ保存ロギングが実行されます。このパラメーターは、ログ・ファイルのアーカイブおよび検索を行う際にユーザー出口プログラムを使用する必要があることも示します。ログ・ファイルは、データベース・マネージャーがログ・ファイルを閉じたときにアーカイブされます。また、ROLLFORWARD ユーティリティーがデータベースを復元する際にログ・ファイルが必要になったときに検索されます。

logretain または *userexit* のいずれか (または両方) のパラメーターが使用可能な場合には、データベースの全バックアップを実行する必要があります。この状態は、*backup_pending* フラグ・パラメーターで示されます。

どちらのパラメーターも選択を解除すると、ログは保存されなくなるので、データベースのロールフォワード・リカバリーは利用できなくなります。この場合、データベース・マネージャーは *logpath* ディレクトリー内のログ・ファイルをすべて (オンライン・アーカイブ・ログ・ファイルも含めて) 削除し、新規のアクティブ・ログ・ファイルを割り振り、循環ログに戻ります。

ユーザー出口プログラムの詳細については、 *管理の手引き: インプリメンテーション* の『データベース・リカバリー用のユーザー出口』を参照してください。

リカバリー

次のパラメーターは、データベース・リカバリーのさまざまな要素を制御します。

- 『自動再始動可能 (autorestart)』
- 428ページの『索引再作成時刻 (indexrec)』
- 429ページの『ロード・リカバリー・セッションのデフォルト数 (dft_loadrec_ses)』
- 430ページの『データベース・バックアップの数 (num_db_backups)』
- 430ページの『リカバリー・ヒストリーの保存期間 (rec_his_retentn)』
- 431ページの『変更されたページの追跡を可能にする (trackmod)』

433ページの『分散作業単位リカバリー』も参照してください。

次のパラメーターは、Tivoli Storage Manager (TSM) で作業する際に使用されます。

- 431ページの『Tivoli Storage Manager 管理クラス (tsm_mgmtclass)』
- 432ページの『Tivoli Storage Manager パスワード (tsm_password)』
- 432ページの『Tivoli Storage Manager ノード名 (tsm_nodename)』
- 433ページの『Tivoli Storage Manager 所有者名 (tsm_owner)』

自動再始動可能 (autorestart)

構成タイプ データベース

パラメーター・タイプ 構成可能

デフォルト[範囲] On [On; Off]

このパラメーターを On にすると、アプリケーションがデータベースに接続したときに、データベース・マネージャーが必要に応じて再始動データベース・ユーティリティーを自動的に呼び出します。データベース再始動ユーティリティーは、*破損リカバリー* と呼ばれる操作を実行します。アプリケーションがデータベースに接続しているときに、データベースが異常終了すると、この操作が実行されます。データベース異常終了の原因としては、電源障害やシステム・ソフトウェア障害などが考えられます。障害が生じた時点で、データベースのバッファー・プールには含まれていたがディスクには書き込まれていなかったコミット済みトランザクションがあれば、そのトランザクション

が適用されます。また、ディスクに書き込まれたコミットされていないトランザクションがあると、そのトランザクションは取り消されます。

autorestart が使用可能でない場合に、破損リカバリーを必要としている (再始動する必要がある) データベースに対してアプリケーションが接続を試みると、 SQL1015N エラーが戻されます。この場合、アプリケーションからデータベース再始動ユーティリティーを呼び出すか、またはリカバリー・ツールから再始動操作を選択してデータベースを再始動することができます。

索引再作成時刻 (indexrec)

構成タイプ データベースおよびデータベース・マネージャー

適用範囲

- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー
- ローカル・クライアントをもつデータベース・サーバー
- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー
- ローカル・クライアントをもつサテライト・データベース・サーバー

パラメーター・タイプ 構成可能

デフォルト[範囲]

UNIX データベース・マネージャー
restart [restart; access]

OS/2 および Windows NT データベース・マネージャー
access [restart; access]

データベース
システム設定の使用 [system; restart; access]

関連パラメーター 427ページの『自動再始動可能 (autorestart)』

このパラメーターは、無効な索引をデータベース・マネージャーが再作成する時点を示します。次の 3 つの設定値があります。

SYSTEM *システム設定値を使用*。データベース・マネージャー構成ファイルで指定された時点で無効な索引を再作成します。 (注: この設定は、データベース構成だけで有効です。)

ACCESS *索引アクセス時*。索引に最初にアクセスしたときに無効な索引が再作成されます。

RESTART *データベースの再始動時*。 RESTART DATABASE コマンドが明示的または暗黙的に発行されたときに無効な索引が再作成されます。

autorestart パラメーターが使用可能になっていると、 RESTART DATABASE コマンドは暗黙的に発行されます。

これらの値と等価な数値と API 定数については、 *管理 API 解説書* を参照してください。

致命的なディスク問題が発生したときに、索引が無効になることがあります。データ自体に問題があった場合は、そのデータは失われる可能性があります。一方、索引に問題があった場合は、索引を再作成すればリカバリーします。ユーザーがデータベースに接続しているときに索引を再作成すると、次の 2 つの問題が生じることがあります。

- 索引ファイルを再作成すると、応答時間が予期せず低下することがあります。表にアクセスし、括弧の特定の索引を使用しているユーザーは、索引が再作成されるまで待機しなければなりません。
- 特に、索引を再作成する原因となったユーザー・トランザクションで COMMIT または ROLLBACK が一度も実行されなかった場合は、再作成した後で予期しないロックが起こることがあります。

推奨事項: ユーザーが多いサーバーで、かつ再始動時について心配する必要がない状況では、このオプションの最適な選択は、破損後にデータベースをオンラインに復帰させる処理の一部として、 DATABASE RESTART 時に索引を再作成することです。

このパラメーターを「ACCESS」に設定すると、索引が再作成されている間は、データベース・マネージャーのパフォーマンスが低下します。この特定の索引または表にアクセスしているユーザーは、索引が再作成されるまで待機しなければなりません。

このパフォーマンスを「RESTART」に設定すると、索引を再作成する際のデータベースの再始動にかかる時間は長くなりますが、データベースがオンラインに復帰されると、通常の処理には何の影響もありません。

ロード・リカバリー・セッションのデフォルト数 (dft_loadrec_ses)

構成タイプ	データベース
パラメーター・タイプ	構成可能
デフォルト[範囲]	1 [1 ～ 30 000]
計測単位	カウンター

このパラメーターは、表ロードのリカバリーで使用するセッションのデフォルト数を指定します。ロード・コピーを検索するのに最適な入出力セッションの数を設定してください。ロード・コピーの検索は、復元操作と似ています。環境変数 DB2LOADREC で指定されたコピー位置ファイルの項目を使って、このパラメーターを指定変更することができます。

ロード検索で使用されるデフォルトのバッファー数は、このパラメーターの値より 2 つ多い数です。コピー位置ファイル中のバッファー数を指定変更することができます。

このパラメーターは、ロールフォワード・リカバリーが使用可能な場合に限り適用されます。

ロード・リカバリーについて詳しくは、*データ移動ユーティリティー手引きおよび解説書* を参照してください。

データベース・バックアップの数 (num_db_backups)

構成タイプ データベース (Database)

パラメーター・タイプ 構成可能

デフォルト[範囲] 12 [1 ～ 32 768]

関連パラメーター 『リカバリー・ヒストリーの保存期間 (rec_his_retentn)』

このパラメーターは、データベース用に保持するデータベース・バックアップの数を指定します。バックアップが指定した数に達すると、以前のバックアップにはリカバリー・ヒストリー・ファイル内で有効期限切れというマークが付けられます。有効期限が切れたデータベースのバックアップに関連する表スペース・バックアップとロード・コピー・バックアップのリカバリー・ヒストリー・ファイル項目にも、有効期限切れというマークが付けられます。バックアップに有効期限切れというマークが付けられると、物理バックアップを格納場所 (たとえば、ディスク、テープ、ADSM) から消去することができます。次のデータベース・バックアップでは、有効期限が切れた項目がリカバリー・ヒストリー・ファイルから除去されます。

データベース・バックアップにヒストリー・ファイルで有効期限切れというマークが付けられると、 DB2 データ・リンク・マネージャーを介してリンクされている対応するファイルのバックアップすべてが、アーカイブ・サーバーから消去されます。

rec_his_retentn 構成パラメーターは、 *num_db_backups* の値と互換性のある値に設定されます。たとえば、*num_db_backup* が大きな値に設定される場合、 *rec_his_retentn* の値は、その数のバックアップを十分サポートできる大きさにする必要があります。

リカバリー・ヒストリーの保存期間 (rec_his_retentn)

構成タイプ データベース

パラメーター・タイプ 構成可能

デフォルト[範囲] 366 [-1; 0 ～ 30 000]

計測単位 日数

関連パラメーター 『データベース・バックアップの数 (num_db_backups)』

このパラメーターは、バックアップのヒストリー情報を保存しておく日数を指定します。バックアップ、復元、およびロードの情報を得るのにリカバリー・ヒストリー・ファイルが必要ない場合は、このパラメーターに小さい数字を設定することができます。

このパラメーターを -1 に設定すると、リカバリー・ヒストリー・ファイルは、使用可能なコマンドか API を明示的に発行したときだけ削除されます。 -1 でない場合は、完全データベース・バックアップを実行するたびにリカバリー・ヒストリー・ファイルは削除されます。

このパラメーターの値は *num_db_backups* パラメーターの値を指定変更しますが、*rec_his_retentn* と *num_db_backups* はともに機能する必要があります。 *num_db_backups* が大きな値に設定される場合、 *rec_his_retentn* の値は、その数のバックアップを十分サポートできる大きさにする必要があります。

PRUNE ユーティリティーで FORCE オプションを使用しない限り、保存期間の長さに関係なく、最新の完全データベース・バックアップとその復元の集まりは必ず保持されます。このユーティリティーについての詳細は、 *コマンド解説書* を参照してください。

変更されたページの追跡を可能にする (trackmod)

構成タイプ データベース

パラメーター・タイプ 構成可能

デフォルト[範囲] No [Yes, No]

このパラメーターが ″Yes″ に設定されると、データベース・マネージャーはデータベースの変更をトラックします。こうして、バックアップ・ユーティリティーが、増分バックアップにより調査して、バックアップ・イメージに潜在的に組み込まれなければならないデータベース・ページのサブセットを検出できるようにします。このパラメーターを ″Yes″ に設定した後は、増分バックアップを行うためのベースラインを得るために全データベース・バックアップを行う必要があります。また、このパラメーターが使用可能で、表スペースが作成される場合、その表スペースを含むバックアップをとる必要があります。これは、データベース・バックアップか、表スペース・バックアップのいずれかです。このバックアップに続いて、増分バックアップにこの表スペースを含めることが許可されます。

Tivoli Storage Manager 管理クラス (tsm_mgmtclass)

構成タイプ データベース

パラメーター・タイプ 構成可能

デフォルト[範囲] Null [任意のストリング]

Tivoli Storage Manager 管理クラスは、バックアップがとられているオブジェクトについて、 TSM サーバーによるバックアップ・バージョン管理方法を示します。

デフォルトは、「TSM 管理クラスなし」です。

管理クラスは Tivoli Storage Manager の管理者によって割り当てられます。割り当てた後は、このパラメーターに管理クラス名を設定する必要があります。 TSM バックアップの実行時点で、データベース・マネージャーはこのパラメーターを使用して、管理クラスを TSM に渡します。

Tivoli Storage Manager に関する詳細については、 *データ回復と高可用性の手引きと解説書* の『Tivoli Storage Manager』を参照してください。

Tivoli Storage Manager パスワード (tsm_password)

構成タイプ データベース

パラメーター・タイプ 構成可能

デフォルト[範囲] Null [任意のストリング]

このパラメーターは、 Tivoli Storage Manager (TSM) 製品に関連したパスワードのデフォルト設定を一時変更するために使用します。別のノードから TSM にバックアップされたデータベースを復元するには、パスワードが必要です。

注: DB2 によるバックアップで、 *tsm_nodename* を指定変更する (たとえば、 BACKUP DATABASE コマンドによって) 場合は、 *tsm_password* も設定する必要があります。

デフォルトを使用できるのは、バックアップを行ったものと同じノードにある TSM からデータベースを復元するために限られます。 *tsm_nodename* は、 DB2 によるバックアップ時に指定変更されることがあります。

Tivoli Storage Manager に関する詳細については、 *データ回復と高可用性の手引きと解説書* の『Tivoli Storage Manager』を参照してください。

Tivoli Storage Manager ノード名 (tsm_nodename)

構成タイプ データベース

パラメーター・タイプ 構成可能

デフォルト[範囲] Null [任意のストリング]

このパラメーターは、 Tivoli Storage Manager (TSM) 製品に関連したノード名のデフォルト設定を一時変更するために使用します。別のノードから TSM にバックアップされたデータベースを復元するには、ノード名が必要です。

デフォルトを使用できるのは、バックアップを行ったものと同じノードにある TSM からデータベースを復元するために限られます。 *tsm_nodename* は、 DB2 によるバックアップ時に指定変更される (たとえば、 BACKUP DATABASE コマンドで) ことがあります。

Tivoli Storage Manager に関する詳細については、 *データ回復と高可用性の手引きと解説書* の『Tivoli Storage Manager』を参照してください。

Tivoli Storage Manager 所有者名 (tsm_owner)

構成タイプ データベース

パラメーター・タイプ 構成可能

デフォルト[範囲] Null [任意のストリング]

このパラメーターは、 Tivoli Storage Manager (TSM) 製品に関連した所有者のデフォルト設定を一時変更するために使用します。別のノードから ADSM にバックアップされたデータベースを復元するには、所有者名が必要です。 *tsm_owner* は、 DB2 によるバックアップ時に指定変更される (たとえば、 BACKUP DATABASE コマンドで) ことがあります。

注: 所有者名には大文字小文字の区別があります。

デフォルトを使用できるのは、バックアップを行ったものと同じノードにある TSM からデータベースを復元するために限られます。

Tivoli Storage Manager に関する詳細については、 *データ回復と高可用性の手引きと解説書* の『Tivoli Storage Manager』を参照してください。

分散作業単位リカバリー

以下のパラメーターは、分散作業単位 (DUOW) トランザクションのリカバリーに影響します。

- 『トランザクション・マネージャー・データベース名 (tm_database)』
- 434ページの『トランザクション再同期間隔 (resync_interval)』
- 435ページの『同期点管理機能ログ・ファイル・パス (spm_log_path)』
- 436ページの『同期点管理機能名 (spm_name)』
- 436ページの『同期点管理機能ログ・ファイル・サイズ (spm_log_file_sz)』
- 437ページの『同期点管理機能再同期エージェント限界 (spm_max_resync)』

トランザクション・マネージャー・データベース名 (tm_database)

構成タイプ データベース・マネージャー

適用範囲

- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー
- クライアント
- ローカル・クライアントをもつデータベース・サーバー
- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー
- ローカル・クライアントをもつサテライト・データベース・サーバー

パラメーター・タイプ 構成可能

デフォルト[範囲] 1ST_CONN [任意の有効なデータベース名]

このパラメーターは、 DB2 インスタンスごとにトランザクション・マネージャー (TM) データベースの名前を識別します。 TM データベースには、以下のデータベースを使用することができます。

- DB2 ユニバーサル・データベースのローカル・データベース
- ホストまたは AS/400 システム上にない、 DB2 ユニバーサル・データベースのリモート・データベース
- DB2 (OS/390 版) バージョン 5 のデータベース (TCP/IP によってアクセスし、同期点管理機能を使用しない場合)

TM データベースとは、ログ機能および調整プログラムとして使用されるデータベースで、未確定トランザクションのリカバリーを実行するときに使用されます。

このパラメーターに **1ST_CONN** を設定すると、ユーザーは最初に TM データベースに接続するようになります。

分散作業単位に関する詳細については、 *管理の手引き: 計画* の『分散データベース』を参照してください。

推奨事項: 管理と操作を単純化するために、多数のインスタンスに対して少数のデータベースを作成し、それらのデータベースを TM データベース専用にすることができます。

トランザクション再同期間隔 (resync_interval)

構成タイプ データベース・マネージャー

適用範囲

- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー
- ローカル・クライアントをもつデータベース・サーバー

- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー
- ローカル・クライアントをもつサテライト・データベース・サーバー

パラメーター・タイプ 構成可能

デフォルト[範囲] 180 [1 ～ 60 000]

計測単位 秒

このパラメーターは、トランザクション・マネージャー (TM)、リソース・マネージャー (RM)、または同期点管理機能 (SPM) で未解決の未確定トランザクションが見つかった場合に、 TM、RM、または SPM がリカバリーを試行する時間の間隔を秒単位で指定します。このパラメーターは、分散作業単位 (DUOW) 環境でトランザクションを実行している場合に適用されます。

分散作業単位に関する詳細については、 *管理の手引き: 計画* の『分散データベース』を参照してください。

推奨事項: ユーザーの環境で、未確定トランザクションがデータベースに対する他のトランザクションを妨害しないようであれば、このパラメーターの値を大きくすることができます。 DB2 コネクト・ゲートウェイを用いて DRDA2 アプリケーション・サーバーにアクセスしている場合は、ローカル・データ・アクセスへの妨害がなくても、アプリケーション・サーバーに未確定トランザクションがあっても影響がないかどうかを考慮する必要があります。未確定トランザクションがない場合は、パフォーマンスの影響はごくわずかです。

同期点管理機能ログ・ファイル・パス (spm_log_path)

構成タイプ データベース・マネージャー

適用範囲

- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー
- ローカル・クライアントをもつデータベース・サーバー
- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー
- ローカル・クライアントをもつサテライト・データベース・サーバー

パラメーター・タイプ 構成可能

デフォルト sqllib/spmlog [任意の有効なパスまたは装置]

このパラメーターは、同期点管理機能 (SPM) ログが作成されるディレクトリーを指定します。デフォルトでは、ログは `sqllib/spmlog` ディレクトリーに書き込まれます。そ

の結果、高ボリューム・トランザクション環境では、入出力ボトルネックが起きる可能性があります。このパラメーターを使って、現在の sqllib/spmlog ディレクトリーよりも高速のディスクに SPM ログ・ファイルを置いてください。こうすることにより、SPM エージェント間の並行性が向上します。

同期点管理機能の詳細については、 *インストールおよび構成 補足* を参照してください。

未確定 DRDA トランザクションのリカバリーの詳細については、 *管理の手引き: 計画* の『ホスト上の未確定トランザクションのリカバリー』を参照してください。

同期点管理機能名 (spm_name)

構成タイプ データベース・マネージャー

適用範囲

- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー
- ローカル・クライアントをもつデータベース・サーバー
- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー
- ローカル・クライアントをもつサテライト・データベース・サーバー

パラメーター・タイプ 構成可能

デフォルト TCP/IP ホスト名から作成される

このパラメーターは、データベース・マネージャーに対して、同期点管理機能 (SPM) のインスタンスの名前を識別します。

同期点管理機能の詳細については、 *インストールおよび構成 補足* を参照してください。

未確定 DRDA トランザクションのリカバリーの詳細については、 *管理の手引き: 計画* の『ホスト上の未確定トランザクションのリカバリー』を参照してください。

同期点管理機能ログ・ファイル・サイズ (spm_log_file_sz)

構成タイプ データベース・マネージャー

適用範囲

- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー
- ローカル・クライアントをもつデータベース・サーバー

- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー
- ローカル・クライアントをもつサテライト・データベース・サーバー

パラメーター・タイプ 構成可能

デフォルト[範囲] 256 [4 ～ 1 000]

計測単位 ページ

このパラメーターは、 4 KB ページの同期点管理機能 (SPM) のログ・ファイル・サイズを識別します。ログ・ファイルは sqllib の下の spmlog サブディレクトリーに入っており、 SPM が初めて開始したときに作成されます。

同期点管理機能の詳細については、 *インストールおよび構成 補足* を参照してください。

未確定 DRDA トランザクションのリカバリーの詳細については、 *管理の手引き: 計画* の『ホスト上の未確定トランザクションのリカバリー』を参照してください。

推奨事項: 同期点管理機能ログ・ファイルのサイズは、パフォーマンスを維持するのに十分な大きさでなければなりませんが、スペースを浪費しないような大きさである必要があります。必要なサイズは、保護会話を用いるトランザクションの数と、 COMMIT または ROLLBACK が出される頻度によって決まります。

SPM ログ・ファイルのサイズを変更するには、次のようにします。

1. LIST DRDA INDOUBT TRANSACTIONS コマンドを用いて、未確定トランザクションがないことを判別する。
2. 未確定トランザクションがなければ、データベース・マネージャーを停止する。
3. データベース・マネージャー構成を新しい SPM ログ・ファイル・サイズで更新する。
4. $HOME/sqllib ディレクトリーに進み、 rm -fr spmlog を出して現行の SPM ログを削除する。 (注意: これで、 AIX コマンドが表示されます。他のシステムでは別の除去または削除コマンドが必要です。)
5. データベース・マネージャーを開始する。データベース・マネージャーの始動中に、指定されたサイズの新しい SPM ログが作成されます。

同期点管理機能再同期エージェント限界 (spm_max_resync)

構成タイプ データベース・マネージャー

適用範囲

- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー
- ローカル・クライアントをもつデータベース・サーバー

- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー
- ローカル・クライアントをもつサテライト・データベース・サーバー

パラメーター・タイプ 構成可能

デフォルト[範囲] 20 [10 ～ 256]

このパラメーターは、再同期操作を同時に実行できるエージェントの数を識別します。

未確定 DRDA トランザクションのリカバリーの詳細については、 *管理の手引き: 計画* の『ホスト上の未確定トランザクションのリカバリー』を参照してください。

同期点管理機能の詳細については、*インストールおよび構成 補足* を参照してください。

データベース管理

データベースに関する情報を記述したりデータベースの管理を制御するのに使用できるパラメーターがいくつかあります。次のグループに分けることができます。

- 『Query Enabler』
- 439ページの『属性』
- 442ページの『DB2 データ・リンク・マネージャー』
- 444ページの『状況』
- 447ページの『コンパイラーの設定』

Query Enabler

次のパラメーターは、 Query Enabler の制御を記述しています。

- 『動的 SQL 照会管理 (dyn_query_mgmt)』

動的 SQL 照会管理 (dyn_query_mgmt)

構成タイプ データベース

パラメーター・タイプ 構成可能

デフォルト[範囲] 0 (DISABLE) [1(ENABLE)、0 (DISABLE)]

このパラメーターは、 DB2 クエリー・パトローラーがインストールされているときに使用されます。データベース構成パラメーター *dyn_query_mgmt* が 『ENABLE』 に設定されており、動的照会のコストがユーザーまたはグループの trap_threshold (DB2 クエリー・パトローラーのユーザー・プロファイル表で指定されている) を上回る場合、DB2 クエリー・パトローラーはこの照会を採用します。 trap_threshold はコスト・ベー

スのトリガーで、 DB2 クエリー・パトローラーでの照会の採用をユーザーが設定するのに使用します。動的照会が採用されると、実行時パラメーターを指定するダイアログが表示されます。

dyn_query_mgmt が 『DISABLE』 に設定されている場合、照会は採用されません。

属性

次のパラメーターは、データベースの一般的な情報を記述します。

- 『構成ファイルのリリース・レベル (release)』
- 『データベース・リリース・レベル (database_level)』
- 440ページの『データベースの領域 (territory)』
- 440ページの『データベースの国別コード (country)』
- 440ページの『データベースのコード・セット (codeset)』
- 441ページの『データベースのコード・ページ (codepage)』
- 441ページの『照合情報 (collate_info)』
- 441ページの『コピー保護可能 (copyprotect)』

copyprotect 以外のパラメーターは、情報提供専用です。

構成ファイルのリリース・レベル (release)

構成タイプ データベース・マネージャー、データベース

適用範囲

- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー
- ローカル・クライアントをもつデータベース・サーバー
- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー
- ローカル・クライアントをもつサテライト・データベース・サーバー

パラメーター・タイプ 情報

関連パラメーター 『データベース・リリース・レベル (database_level)』

このパラメーターは、構成ファイルのリリース・レベルを指定します。

データベース・リリース・レベル (database_level)

構成タイプ データベース

パラメーター・タイプ 情報

関連パラメーター 439ページの『構成ファイルのリリース・レベル (release)』

このパラメーターは、データベースを使用できるデータベース・マネージャーのリリース・レベルを示します。移行が完了していないか、または失敗した場合は、移行されていないデータベースのリリース・レベルが示されるので、 *release* パラメーター (データベース構成ファイルのリリース・レベル) とは違っていることがあります。それ以外の場合、*database_level* の値は *release* パラメーターの値と同じになります。

データベースの領域 (territory)

構成タイプ データベース

パラメーター・タイプ 情報

関連パラメーター 『データベースの国別コード (country)』

このパラメーターは、データベースを作成するときに使用した領域を示します。データベース・マネージャーは、この領域を用いて、 *country* パラメーターの値を判別します。データベース・マネージャーが領域を使用する方法の詳細については、 *管理の手引き: 計画* の付録『各国語サポート』を参照してください。

データベースの国別コード (country)

構成タイプ データベース

パラメーター・タイプ 情報

関連パラメーター 『データベースの領域 (territory)』

このパラメーターは、データベースを作成するときに使用した国別コードを示します。*country* パラメーターは、 *territory* パラメーターから派生しています。データベース・マネージャーが国別コードを使用する方法の詳細については、 *管理の手引き: 計画* の付録『各国語サポート』を参照してください。

データベースのコード・セット (codeset)

構成タイプ データベース

パラメーター・タイプ 情報

関連パラメーター 441ページの『データベースのコード・ページ (codepage)』

このパラメーターは、データベースを作成するときに使用したコード・セットを示します。データベース・マネージャーは、コード・セットを用いて *codepage* パラメーターの値を判別します。データベース・マネージャーがコード・セットを使用する方法の詳細については、 *管理の手引き: 計画* の付録『各国語サポート』を参照してください。

データベースのコード・ページ (codepage)

構成タイプ データベース

パラメーター・タイプ 情報

関連パラメーター 440ページの『データベースのコード・セット (codeset)』

このパラメーターは、データベースを作成するときに使用したコード・ページを示します。 *codepage* パラメーターは、 *codeset* パラメーターから派生しています。データベース・マネージャーがコード・ページを使用する方法の詳細については、 *管理の手引き : 計画* の付録『各国語サポート』を参照してください。

照合情報 (collate_info)

このパラメーターは、 GET DATABASE CONFIGURATION API を使用する場合に限り表示できます。コマンド行プロセッサーまたはコントロール・センターからは、表示**できません**。

構成タイプ データベース

パラメーター・タイプ 情報

このパラメーターは、データベースの照合情報を 260 バイトで提供します。最初の 256 バイトは、データベースの照合順序を指定します。「n」番目のバイトには、データベースのコード・ページ中で基本 10 進表記が「n」になっているコード・ポイントの、分類重みがあります。

最後の 4 バイトには、照合順序のタイプに関する内部情報があります。この内部情報は、データベースのプラットフォームにとって適当な整数として扱うことができます。次の 3 つの値があります。

- **0** - 固有でない重みが順序列に含まれている。
- **1** - 固有な重みが順序列に含まれている。
- **2** - 順序が一致している。つまり、バイトごとのストリングが対応している。

この内部タイプ情報を使用する場合は、さまざまなプラットフォームでデータベースの情報を検索するときに生じる可能性のあるバイト逆転について考慮する必要があります。

照合順序は、データベースを作成するときに指定することができます。

コピー保護可能 (copyprotect)

構成タイプ データベース

パラメーター・タイプ 構成可能

デフォルト[範囲] No [Yes; No]

このパラメーターは、コピー保護属性を使用可能にします。デフォルトでは使用不能になっています。バージョン 2 より前のバージョンのデータベース・マネージャーでは、デフォルト設定でコピー保護属性が使用可能です。

このパラメーターは UNIX ベースの環境には適用されません。

copyprotect パラメーターを設定しても、データベース・バックアップ・ユーティリティーおよびデータベース復元ユーティリティーは影響されません。コピー保護のデータベースを、バックアップを実行して別のワークステーションに復元し、カタログおよびアクセスを行うことができます。

警告: データベース・マネージャーまたはオペレーティング・システムを再インストールする際には、その前にすべてのデータベースからコピー保護を除去してください。コピー保護を除去しないと、データベースにアクセスしようとしたときにエラーが生じます。再インストールが終わってから、コピー保護を使用可能にすることができます。

DB2 データ・リンク・マネージャー

次のパラメーターは、 DB2 データ・リンク・マネージャーと関連があります。

- 『データ・リンク・アクセス・トークン満了間隔 (dl_expint)』
- 443ページの『データ・リンクのコピー数 (dl_num_copies)』
- 443ページの『ドロップ後のデータ・リンク時間 (dl_time_drop)』
- 443ページの『データ・リンク・トークン・アルゴリズム (dl_token)』
- 444ページの『大文字のデータ・リンク・トークン (dl_upper)』
- 444ページの『データ・リンク・サポートの使用可能化 (datalinks)』

データ・リンク・アクセス・トークン満了間隔 (dl_expint)

構成タイプ	データベース (Database)
パラメーター・タイプ	構成可能
デフォルト[範囲]	60 [-1, 1 〜 31 536 000]
計測単位	秒

このパラメーターは、生成されるファイル・アクセス制御トークンが有効な時間間隔 (秒単位) を指定します。トークンが有効な秒数は、トークンの生成時から開始されます。データ・リンク・ファイル・システム・フィルターは、この満了時刻が入ったトークンの妥当性を検査します。

ファイル・アクセス制御トークンの詳細については、 *DB2 データ・リンク・マネージャー 概説およびインストール* を参照してください。

このパラメーターのデフォルト値は、60 秒です。このパラメーターが ″-1″ に設定されている場合、アクセス制御トークンは有効期限切れとなります。これを回避する方法

は、このパラメーターをその最大値 31536000 (秒) に設定することです。これは 1 年の有効期限に相当し、すべてのアプリケーションに適合するはずです。

このパラメーターは、 『READ PERMISSION DB』 を指定する DATALINK 列に適用されます。

データ・リンクのコピー数 (dl_num_copies)

構成タイプ データベース (Database)

パラメーター・タイプ 構成可能

デフォルト[範囲] 0 [0 ～ 15]

このパラメーターは、データベースにファイルがリンクされるときにアーカイブ・サーバー (TSM サーバーなど) で作成されるファイルの追加コピー数を指定します。

このパラメーターのデフォルト値はゼロ (0) です。

このパラメーターは、『Recovery=Yes』 を指定する DATALINK 列に適用されます。

ドロップ後のデータ・リンク時間 (dl_time_drop)

構成タイプ データベース (Database)

パラメーター・タイプ 構成可能

デフォルト[範囲] 1 [0 ～ 365]

計測単位 日数

このパラメーターは、 DROP DATABASE の発行後にアーカイブ・サーバー (TSM サーバーなど) でファイルが保存される時間間隔 (日数) を指定します。

このパラメーターのデフォルト値は 1 日です。値がゼロ (0) であれば、 DROP コマンドの発行時にファイルがただちにアーカイブ・サーバーから削除されます。
(DATALINK 列で ON UNLINK DELETE パラメーターが指定されない限り、実際のファイルは削除されません。)

このパラメーターは、 『Recovery=Yes』 を指定する DATALINK 列に適用されます。

データ・リンク・トークン・アルゴリズム (dl_token)

構成タイプ データベース (Database)

パラメーター・タイプ 構成可能

デフォルト[範囲] MAC0 [MAC0; MAC1]

このパラメーターは、 DATALINK ファイル・アクセス制御トークンの生成で使われるアルゴリズムを指定します。値 MAC1 (メッセージ確認コード) は値 MAC0 より安全なメッセージ確認コードを生成しますが、パフォーマンスのオーバーヘッドも大きくなります。

ファイル・アクセス制御トークンの詳細については、 *DB2 データ・リンク・マネージャー 概説およびインストール* を参照してください。

このパラメーターは、 『READ PERMISSION DB』 を指定する DATALINK 列に適用されます。

大文字のデータ・リンク・トークン (dl_upper)

構成タイプ データベース (Database)

パラメーター・タイプ 構成可能

デフォルト[範囲] NO [YES; NO]

このパラメーターは、ファイル・アクセス制御トークンで大文字を使用するかどうかを指定します。値 『YES』 は、アクセス制御トークンの文字がすべて大文字であることを指定します。値 『NO』 は、そのトークンに大文字と小文字の両方を含められることを指定します。

ファイル・アクセス制御トークンの詳細については、 *DB2 データ・リンク・マネージャー 概説およびインストール* を参照してください。

このパラメーターは、 『READ PERMISSION DB』 を指定する DATALINK 列に適用されます。

データ・リンク・サポートの使用可能化 (datalinks)

構成タイプ データベース・マネージャー

パラメーター・タイプ 構成可能

デフォルト[範囲] NO [YES; NO]

このパラメーターは、データ・リンク・サポートを使用可能にするかどうかを指定します。値 『YES』 は、ネイティブのファイル・システム (たとえば、 AIX 上の JFS) に格納されているファイルをリンクするデータ・リンク・マネージャーで、データ・リンク・サポートが使用可能であることを指定します。値 『NO』 は、データ・リンク・サポートが使用不能であることを指定します。

状況

次のパラメーターは、データベースの状態を記述しています。

- 445ページの『バックアップ保留標識 (backup_pending)』

- 『データベースの一貫性 (database_consistent)』
- 『ロールフォワード保留標識 (rollfwd_pending)』
- 446ページの『ログ保存状況表示 (log_retain_status)』
- 446ページの『ユーザー出口状況表示 (user_exit_status)』
- 446ページの『保留の復元 (restore_pending)』
- 447ページの『複数ページ・ファイル割り振りの使用可能化 (multipage_alloc)』

バックアップ保留標識 (backup_pending)

構成タイプ データベース

パラメーター・タイプ 情報

このパラメーターをオンに設定すると、データベースにアクセスする前に、データベースの全バックアップを実行しなければならないことが示されます。このパラメーターは、データベース構成が変更された場合にのみオンとなり、データベースはリカバリー不能状態からリカバリー可能状態に移行します (つまり、最初は *logretain* および *userexit* パラメーターは両方とも NO に設定されましたが、その後、一方または両方のパラメーターが YES に設定され、データベース構成の更新が受け入れられたということです)。

データベースの一貫性 (database_consistent)

構成タイプ データベース

パラメーター・タイプ 情報

データベースが一貫性を保っているかどうかを示します。

YES は、すべてのトランザクションがコミットまたはロール・バックされている、つまり、一貫性があることを示しています。データベースが一貫性を保っているときにシステムが「破損」した場合、データベースを使用可能にするために特別の処置を行う必要はありません。

NO は、保留中のトランザクション、または保留中の他のタスクがデータベースにあること、つまり、データは現時点では一貫性がないことを示します。データベースの一貫性がないときにシステムが「破損」した場合、データベースを使用可能にするためには、 RESTART DATABASE コマンドを使ってデータベースを再始動する必要があります。 RESTART DATABASE コマンドの詳細については、 *コマンド解説書* を参照してください。

ロールフォワード保留標識 (rollfwd_pending)

構成タイプ データベース

パラメーター・タイプ 情報

このパラメーターは、次のいずれかの状態を示します。

- **DATABASE** - このデータベースでロールフォワード・リカバリー手順が必要なことを示します。
- **TABLESPACE** - いくつかの表スペースをロールフォワードする必要があることを示します。
- **NO** - データベースが使用可能で、ロールフォワード・リカバリーの必要がないことを示します。

データベースまたは表スペースは、 (ROLLFORWARD DATABASE を使用して) リカバリーしない限りアクセスできません。 ROLLFORWARD DATABASE について詳しくは、*コマンド解説書* を参照してください。

ログ保存状況表示 (log_retain_status)

構成タイプ データベース

パラメーター・タイプ 情報

関連パラメーター 425ページの『ログの保存可能 (logretain)』

このパラメーターを設定すると、ログ・ファイルが、ロールフォワード・リカバリー用に保存されていることが示されます。

logretain パラメーターの設定値が Recovery と等しい場合、このパラメーターが設定されます。

ユーザー出口状況表示 (user_exit_status)

構成タイプ データベース

パラメーター・タイプ 情報

関連パラメーター 426ページの『ユーザー出口可能 (userexit)』

このパラメーターを Yes に設定すると、データベース・マネージャーでロールフォワード・リカバリーが使用できることと、データベース・マネージャーが呼び出したログ・ファイルを保存および検索するときにユーザー出口プログラムが使用されることが示されます。

保留の復元 (restore_pending)

構成タイプ データベース

パラメーター・タイプ 情報

このパラメーターは、 RESTORE PENDING 状況がデータベースにあるかどうかを示します。

複数ページ・ファイル割り振りの使用可能化 (multipage_alloc)

構成タイプ データベース

パラメーター・タイプ 情報

複数ページ・ファイル割り振りは、挿入パフォーマンスを上げるために使用されます。これは SMS 表スペースにだけ適用されます。これが使用可能になっていると、すべての SMS 表スペースがこれに影響されます。すなわち、個別の SMS 表スペースを選択することはできません。

このパラメーターのデフォルトは No です。すなわち、複数ページ・ファイル割り振りは使用可能になりません。

データベースの作成後、このパラメーターを Yes に設定し、複数ページ・ファイル割り振りが可能になっていることを示すことができます。これは、 *db2empfa* ツールを使用して行われます。ひとたび Yes に設定された後は、このパラメーターを No に変更することはできません。

このツールについて詳しくは、 *コマンド解説書* を参照してください。

コンパイラーの設定

次のパラメーターは、コンパイラーに影響する情報を提供します。

- 『算術例外の続行 (dft_sqlmathwarn)』
- 449ページの『デフォルトの程度 (dft_degree)』
- 449ページの『デフォルトの照会最適化クラス (dft_queryopt)』
- 450ページの『デフォルトの最新表示の期間 (dft_refresh_age)』
- 450ページの『頻度の高い値の保存数 (num_freqvalues)』
- 451ページの『列変位数の数 (num_quantiles)』

算術例外の続行 (dft_sqlmathwarn)

構成タイプ データベース

パラメーター・タイプ 構成可能

デフォルト[範囲] No [No、Yes]

このパラメーターは、SQL ステートメントのコンパイル時に、算術計算エラーと検索変換エラーをエラーまたは警告として処理することを決めるデフォルト値を設定します。静的 SQL ステートメントの場合は、このパラメーターの値はバインド時のパッケージに関連付けられます。動的 SQL DML ステートメントでは、このパラメーターの値は、ステートメントを準備するときに使用されます。

警告: データベースの *dft_sqlmathwarn* 値を変更すると、算術式を含む検査制約、トリガー、および視点の動作も変わることがあります。このため、変更がデータベースのデータ保全性に影響することがあります。データベースに対する *dft_sqlmathwarn* の設定変更は、新しい算術例外の処理動作が検査制約、トリガー、および視点にどのように影響するかを注意深く検討した後でのみ行ってください。変更後に、引き続き変更するときも同様に注意深い検討が必要です。

例として、次のような除算を含む検査制約について考えてみます。

```
A/B > 0
```

dft_sqlmathwarn が 『No』 の場合は、 B=0 を INSERT すると、このゼロによる除算は算術計算エラーとして処理されます。この挿入操作は DB2 がその制約を検査できないため失敗します。 *dft_sqlmathwarn* を 『Yes』 に変更すると、ゼロによる除算は算術計算の警告として処理され、結果は NULL になります。この結果の NULL によって、『>』 述部は UNKNOWN と評価され、挿入操作は成功します。 *dft_sqlmathwarn* を 『No』 に戻すと、同一行への挿入試行は失敗します。これは、ゼロによる除算エラーによって DB2 が制約を評価できないためです。 *dft_sqlmathwarn* が 『Yes』 だったときに B=0 を挿入された行は、表の中に保持され、選択することができます。制約を評価させた行の更新は失敗となりますが、一方、制約の再評価を必要としない行の更新は成功します。

dft_sqlmathwarn を 『No』 から 『Yes』 に変更する前に、算術式の NULL を明示的に処理するように制約を書き直すことを考えてください。たとえば、次のとおりです。

```
( A/B > 0 ) AND ( CASE
                    WHEN A IS NULL THEN 1
                    WHEN B IS NULL THEN 1
                    WHEN A/B IS NULL THEN 0
                    ELSE 1
                    END
                  = 1 )
```

これは、A と B がともにヌル可能であれば使用することができます。また、A または B のいずれかがヌル可能でない場合は、対応する IS NULL WHEN 文節を取り除くことができます。

dft_sqlmathwarn を 『Yes』 から 『No』 に変更するときは、その前に、矛盾するおそれのあるデータをまず検査してください。そのために、たとえば、次のような述部を使用することができます。

```
WHERE A IS NOT NULL AND B IS NOT NULL AND A/B IS NULL
```

矛盾する行が分離されているときは、 *dft_sqlmathwarn* を変更する前にその矛盾を訂正するような適切なアクションをとる必要があります。また、変更した後も、算術式の制約を手作業で検査し直すことができます。そのためには、まず、影響される表を検査保留状態 (SET CONSTRAINTS ステートメントに OFF 文節を付ける) に置き、次にその

表の検査を要求します (SET CONSTRAINTS ステートメントに IMMEDIATE CHECKED 文節を付ける)。矛盾するデータは算術計算エラーとして指示され、そのため制約は評価されません。

推奨事項: 算術例外を含んでいるような照会の処理が特に必要でないかぎり、デフォルトの設定値は NO に設定してください。そのような照会を含むときは YES に指定します。別のデータベース・マネージャーでは発生する算術例外に関係なく結果が得られるような SQL ステートメントを処理する場合は、このようなことが起こり得ます。

デフォルトの程度 (dft_degree)

構成タイプ データベース

パラメーター・タイプ 構成可能

デフォルト[範囲] 1 [-1、1 ～ 32 767]

関連パラメーター 473ページの『照会の最大並列処理度 (max_querydegree)』

このパラメーターは、 CURRENT DEGREE 特殊レジスターおよび DEGREE バインド・オプションのデフォルト値を指定します。

デフォルト値は 1 です。

値 1 は、区画内並列処理を行わないことを意味します。値 -1 は、プロセッサー数と照会のタイプに基づいて、区画内並列処理の程度を最適化プログラムが決定することを意味します。

SQL ステートメントの区画内並列処理の程度は、 CURRENT DEGREE 特殊レジスターまたは DEGREE バインド・オプションを使用してステートメントのコンパイル時に指定されます。活動アプリケーションに対する実行時区画内並列処理の最大度は SET RUNTIME DEGREE コマンドによって指定されます。照会の最大並列処理度 (*max_querydegree*) 構成パラメーターは、すべての SQL 照会に対する、照会の区画内並列処理の最大度を指定します。

実際の実行時程度は、次の中の最も小さいものになります。
- *max_querydegree* 構成パラメーター
- アプリケーション実行時の程度
- SQL ステートメントのコンパイル時の程度

デフォルトの照会最適化クラス (dft_queryopt)

構成タイプ データベース

パラメーター・タイプ 構成可能

デフォルト[範囲] 5 [0 ～ 9]

計測単位 照会最適化クラス (下記参照)

照会最適化クラスは、SQL 照会のコンパイル時に、最適化プログラムに異なる最適化の程度を使用するように指示するために使用されます。このパラメーターは、 SET CURRENT QUERY OPTIMIZATION ステートメントも QUERYOPT バインド・コマンドも使用しないときに使用されるデフォルトの照会最適化クラスを設定することによって、柔軟性を高めるものです。

現在定義されている照会最適化クラスは、次のとおりです。

0 - 最低の照会最適化。
1 - およそ DB2 バージョン 1 程度の最適化。
2 - 簡単な最適化。
3 - 適度の照会最適化。
5 - アクセス・プランを選択するときに使われる努力を制限する、有効な発見的手法の照会最適化。これがデフォルトです。
7 - 有効な照会最適化。
9 - 最大の照会最適化。

推奨事項: 適切な照会最適化クラスを選択するための情報および指針は、 68ページの『最適化クラスの調整』に説明してあります。

プログラムがどのようにしてデータベース構成パラメーターを検索し、修正できるかについてさらに詳しくは、*管理 API 解説書* を参照してください。

デフォルトの最新表示の期間 (dft_refresh_age)

構成タイプ データベース

パラメーター・タイプ 構成可能

デフォルト[範囲] 0 [0、99999999999999 (ANY)]

このパラメーターには、 CURRENT REFRESH AGE 特殊レジスターが指定されていない場合の、 REFRESH AGE のデフォルト値を指定します。このパラメーターは、タイム・スタンプ期間の値を、データ・タイプ DECIMAL(20,6) で指定します。この時刻期間は、特定の REFRESH DEFERRED 要約表に対する REFRESH TABLE ステートメントを処理してから、どれくらいの期間、その要約表を照会の最適化に使用できるかを示しています。 CURRENT REFRESH AGE の値が 99999999999999 (ANY) であり、QUERY OPTIMIZATION クラスが 5 以上の場合、 REFRESH DEFERRED 要約表は、動的 SQL 照会の処理を最適化するものと見なされます。

頻度の高い値の保存数 (num_freqvalues)

構成タイプ データベース

パラメーター・タイプ 構成可能

デフォルト[範囲] 10 [0 ～ 32 767]

計測単位 カウンター

関連パラメーター

- 『列変位数の数 (num_quantiles)』
- 368ページの『統計ヒープ・サイズ (stat_heap_sz)』

このパラメーターは、 RUNSTATS コマンドに WITH DISTRIBUTION オプションを指定して実行したときに収集される「頻度が高い値」の数を指定します。このパラメーターの値を大きくすると、統計を収集するときの統計ヒープの量 (*stat_heap_sz*) も増えます。

最適化プログラムは、「頻度が高い値」の統計を使って、ある列でのデータ値の分散を調べます。値を高く設定すると、SQL 最適化プログラムが利用できる情報量は増えますが、追加のカタログ・スペースが必要になります。 0 を指定すると、ユーザーが分散統計を要求した場合でも、値の頻度に関する統計は保存されません。

このパラメーターを更新すると、最適化プログラムは、一様に分散していないデータについて、ある種の述部 (=、<、>、IS NULL、 IS NOT NULL) の選択性をより正確に見積もることができます。選択性をより正確に計算できれば、より効率のよいアクセス・プランを選択することができます。

このパラメーターの値を変更した場合は、次の作業を行ってください。
- すべてのユーザーがデータベースから切断した後で、データベースに再接続し、RUNSTATS コマンドを実行します。
- 静的 SQL を含むパッケージがあれば、それを再バインドします。

詳細については、121ページの『分配統計の収集と使用』を参照してください。

推奨事項: このパラメーターを更新するときは、一般的に選択述部を使用している (最も重要な表の中の) 最も重要な列が、どの程度一様になっていないかを判別する必要があります。 SQL SELECT ステートメントを使用すると、列に含まれる各値の出現箇所の数が順序付けられてランキングになって示されます。均一に配分されている列、固有の列、長い列、または LOB 列については、考慮しないでください。このパラメーターの場合、実際に適切な値の範囲は 10 ～ 100 です。

高頻度値の統計を収集する処理では、 CPU とメモリー・リソース (*stat_heap_sz*) がかなり必要となるので注意してください。

列変位数の数 (num_quantiles)

構成タイプ データベース

パラメーター・タイプ 構成可能

デフォルト[範囲] 20 [0 ～ 32 767]

計測単位 カウンター

関連パラメーター

- 450ページの『頻度の高い値の保存数 (num_freqvalues)』
- 368ページの『統計ヒープ・サイズ (stat_heap_sz)』

このパラメーターは、RUNSTATS コマンドに WITH DISTRIBUTION オプションを指定して実行したときに収集される、変位数の数を制御します。このパラメーターの値を大きくすると、統計を収集するときの統計ヒープの量 (*stat_heap_sz*) も増えます。

最適化プログラムは、「変位数」の統計を用いて、ある列でのデータ値の分散を調べます。値を高く設定すると、SQL 最適化プログラムが利用できる情報量は増えますが、追加のカタログ・スペースが必要になります。 0 または 1 に指定すると、分散統計を収集するようユーザーが要求しても、変位数の統計は保存されません。

このパラメーターを更新すると、最適化プログラムは、一様に分散していないデータについて、範囲述部の選択性をより正確に見積もることができます。最適化プログラムの決定において、この情報は、索引走査か表走査のどちらを選択するかに強く影響します。 (頻度の高い値の範囲にアクセスするには表走査、頻度の低い値の範囲には索引走査を使用するのが効果的です。)

このパラメーターの値を変更した場合は、次の作業を行ってください。
- すべてのユーザーがデータベースから切断した後で、データベースに再接続し、RUNSTATS コマンドを実行します。
- 静的 SQL を含むパッケージがあれば、それを再バインドします。

詳しくは、121ページの『分配統計の収集と使用』を参照してください。

推奨事項: このパラメーターのデフォルト値を使用した場合、一方向範囲述部 (>、>=、<、または <=) の見積時のエラーは最大で約 2.5 %、 BETWEEN 述部の見積時のエラーは最大で約 5% です。以下のようにすると、変位値の数を簡単に概算することができます。
- 範囲照会の行数を見積もるときに許容される最大エラーを決めます (これを P とします)。
- 変位値の数は、ほとんどの述部が BETWEEN 述部のときは 100/P、ほとんどの述部が他のタイプの範囲述部 (<、<=、 > または >=) のときは 50/P と概算できます。

たとえば、変位値を 25 にすると、BETWEEN 述部の場合の見積エラーは最大で 4%、″>″ 述部の場合の見積エラーは最大で 2% ということになります。このパラメーターの、実際に適切な値の範囲は 10 ～ 50 です。

通信

次のパラメーター・グループは、クライアント / サーバー環境での DB2 の使用状況について記述しています。

- 『通信プロトコルの設定』
- 458ページの『分散サービス』
- 463ページの『DB2 ディスカバリー機能』

通信プロトコルの設定

次のパラメーターは、データベース・クライアントおよびデータベース・サーバーを構成する場合に使用します。

- 『NetBIOS ワークステーション名 (nname)』
- 454ページの『TCP/IP サービス名 (svcename)』
- 455ページの『APPC トランザクション・プログラム名 (tpname)』
- 456ページの『IPX/SPX ファイル・サーバー名 (fileserver)』
- 456ページの『IPX/SPX DB2 サーバー・オブジェクト名 (objectname)』
- 457ページの『IPX/SPX ソケット番号 (ipx_socket)』

NetBIOS ワークステーション名 (nname)

構成タイプ

データベース・マネージャー

適用範囲

- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー
- クライアント
- ローカル・クライアントをもつデータベース・サーバー
- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー

パラメーター・タイプ

構成可能

デフォルト

Null

このパラメーターでは、 NetBIOS LAN 環境のワークステーション上にあるデータベース・インスタンスに固有の名前を割り当てることができます。この *nname* が、ワークステーションの NetBIOS に実際に登録される NetBIOS 名の基になります。

NetBIOS プロトコルは NetBIOS 名を使った接続を確立するので、 *nname* パラメーターをクライアントとサーバー・ノードの両方に設定する必要があります。

クライアント・アプリケーションでは、アクセスするデータベースが存在しているサーバーの *nname* が認識されていなければなりません。サーバーの *nname* は、たとえば CATALOG NETBIOS NODE コマンドなどで、 『server-nname』 パラメーターとしてクライアントのノード・ディレクトリーでカタログする必要があります。

サーバー・ノードの *nname* が新しい名前に変わった場合には、そのサーバーのデータベースにアクセスしているすべてのクライアントが、この新しい名前をカタログしなければなりません。

TCP/IP サービス名 (svcename)

構成タイプ データベース・マネージャー

適用範囲

- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー
- ローカル・クライアントをもつデータベース・サーバー
- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー
- ローカル・クライアントをもつサテライト・データベース・サーバー

パラメーター・タイプ 構成可能

デフォルト Null

このパラメーターは、リモート・クライアント・ノードからの通信を待つ場合にデータベース・サーバーが使用する TCP/IP ポートの名前を指定します。この名前は、データベース・マネージャーが予約している 2 つの連続したポートのうち最初のポート名でなければなりません。 2 番目のポートは、下位レベルのクライアントからの割り込み要求を処理するために使用します。

TCP/IP を使ってデータベース・クライアントから送られてきた接続要求を受け入れるには、そのサーバーに指定されたポートでデータベース・サーバーが待機していなければなりません。データベース・サーバーのシステム管理者は、1 つのポート (番号 *n*) を予約し、関連した TCP/IP サービス名をサーバー上のサービス・ファイルに定義する必要があります。データベース・サーバーが下位レベルのクライアントからの要求をサポートする場合は、サーバー上のサービス・ファイルに 2 番目 (番号 *n+1*。割り込み要求用) のポートを定義する必要があります。

データベース・サーバー・ポート (番号 *n*) および TCP/IP サービス名を、データベース・クライアント上のサービス・ファイルに定義しなければなりません。下位レベルのクライアントの場合、割り込みポート (番号 *n+1*) もクライアント上のサービス・ファイルに定義する必要があります。

サービス・ファイルの位置は、ユーザーの操作環境によって異なります。たとえば、次のとおりです。
- UNIX の場合 − /etc/services
- OS/2 の場合 − ¥tcpip¥etc¥services
- OS/2 Warp の場合 − ¥mptn¥etc¥services

データベース・サーバーを開始したときにどのポートで接続要求を待機するかをデータベース・サーバーが判断できるよう、 *svcename* パラメーターにはメインの接続ポートと関連したサービス名を設定してください。下位レベルのクライアントをサポートしているか、または使用している場合には、割り込みポートのサービス名は構成ファイルに保管されません。割り込みポート番号は、メインの接続ポート番号から導き出すことができます (*割り込みポート番号 = メイン接続ポート + 1*)。

データベース・サーバーに合った TCP/IP の設定について詳しくは、 *インストールおよび構成 補足* を参照してください。

APPC トランザクション・プログラム名 (tpname)

構成タイプ データベース・マネージャー

適用範囲
- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー
- ローカル・クライアントをもつデータベース・サーバー
- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー

パラメーター・タイプ 構成可能

デフォルト Null

このパラメーターは、APPC 通信プロトコルを使用している場合に、データベース・サーバーに割り振り要求を発行するのにデータベース・クライアントが使用する、リモート・トランザクション・プログラムの名前を定義します。このパラメーターは、データベース・サーバーの構成ファイルで設定しなければなりません。

このパラメーターは、 SNA トランザクション・プログラム定義で構成されたトランザクション・プログラム名と同じである必要があります。ご使用の DB2 製品に合った APPC の設定については、 *インストールおよび構成 補足* を参照してください。

推奨事項: この名前に使用できる文字は、以下のものだけです。
- アルファベット (A 〜 Z、または a 〜 z)
- 数値 (0 〜 9)
- ドル記号 ($)、番号記号 (#)、アットマーク (@)、およびピリオド (.)

IPX/SPX ファイル・サーバー名 (fileserver)

構成タイプ データベース・マネージャー

適用範囲

- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー
- ローカル・クライアントをもつデータベース・サーバー
- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー

パラメーター・タイプ 構成可能

デフォルト Null

関連パラメーター

- 『IPX/SPX DB2 サーバー・オブジェクト名 (objectname)』
- 457ページの『IPX/SPX ソケット番号 (ipx_socket)』

このパラメーターは、データベース・マネージャーのインターネットワーク・アドレスが登録されている NetWare ファイル・サーバーの名前を指定します。データベース・マネージャーのインターネットワーク・アドレスは、 NetWare ファイル・サーバーのバインダリーに保管されています。登録済みのファイル・サーバー名を変更した場合は、そのサーバー・インスタンスにアクセスするすべてのクライアントで次の処置を行う必要があります。

- サーバー・ノードを UNCATALOG する
- 新しいファイル・サーバー名を指定して、サーバー・ノードを CATALOG する

詳細については、*インストールおよび構成 補足* を参照してください。

IPX/SPX DB2 サーバー・オブジェクト名 (objectname)

構成タイプ データベース・マネージャー

適用範囲

- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー
- ローカル・クライアントをもつデータベース・サーバー
- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー

パラメーター・タイプ 構成可能

デフォルト Null

関連パラメーター

- 456ページの『IPX/SPX ファイル・サーバー名 (fileserver)』
- 『IPX/SPX ソケット番号 (ipx_socket)』

このパラメーターは、 IPX/SPX ネットワークでのデータベース・マネージャー・インスタンスの名前を付けます。 NetWare ファイル・サーバーに登録されている各サーバー・インスタンスに、固有の名前を付ける必要があります。データベース・サーバーでこの名前を変更した場合、そのサーバーにアクセスするすべてのクライアントは、サーバー・ノードを一度カタログ解除してから、新しいオブジェクト名を指定して再カタログする必要があります。

ご使用の DB2 製品に合った IPX/SPX の設定については、 *インストールおよび構成補足* を参照してください。

IPX/SPX ソケット番号 (ipx_socket)

構成タイプ データベース・マネージャー

適用範囲

- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー
- ローカル・クライアントをもつデータベース・サーバー
- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー

パラメーター・タイプ 構成可能

デフォルト[範囲] 879E [879E ～ 87A2] 競合を避けるために、Novell には DB2 用のソケット番号が 5 つ (879E から 87A2) 登録されています。

関連パラメーター

- 456ページの『IPX/SPX ファイル・サーバー名 (fileserver)』
- 456ページの『IPX/SPX DB2 サーバー・オブジェクト名 (objectname)』

このパラメーターは、「定式」ソケット番号を指定し、 DB2 サーバーのインターネットワーク・アドレスで、接続の端点を指定します。特定の機械で実行しているすべての Novell IPX/SPX アプリケーションの間で、各 DB2 サーバー・インスタンスのソケット

番号は固有でなければなりません。固有であれば、DB2 サーバーは、このソケット番号を使って着信 IPX/SPX 接続を listen することができます。

ご使用の DB2 製品に合った IPX/SPX の設定については、 *インストールおよび構成補足* を参照してください。

分散サービス

次のパラメーターは、 DCE ディレクトリー・サービスを使用できるようデータベース・クライアントとデータベース・サーバーを構成します。

- 『ディレクトリー・サービス・タイプ (dir_type)』
- 459ページの『DCE ネームスペース内のディレクトリー・パス名 (dir_path_name)』
- 460ページの『DCE ネームスペース内のオブジェクト名 (dir_obj_name)』
- 460ページの『経路指定情報オブジェクト名 (route_obj_name)』
- 461ページの『デフォルト・クライアント通信プロトコル (dft_client_comm)』
- 462ページの『デフォルトのクライアント・アダプター番号 (dft_client_adpt)』

DB2 での DCE ディレクトリーの使用法については、 *管理の手引き: インプリメンテーション* にある『分散コンピューティング環境 (DCE) ディレクトリー・サービスの使用』を参照してください。

ディレクトリー・サービス・タイプ (dir_type)

構成タイプ

データベース・マネージャー

適用範囲

- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー
- UNIX および OS/2 クライアント
- UNIX および OS/2 ローカル・クライアントをもつデータベース・サーバー
- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー

パラメーター・タイプ

構成可能

デフォルト[範囲]

NONE [NONE; DCE]

関連パラメーター

- 460ページの『DCE ネームスペース内のオブジェクト名 (dir_obj_name)』
- 459ページの『DCE ネームスペース内のディレクトリー・パス名 (dir_path_name)』
- 460ページの『経路指定情報オブジェクト名 (route_obj_name)』

- 461ページの『デフォルト・クライアント通信プロトコル (dft_client_comm)』
- 462ページの『デフォルトのクライアント・アダプター番号 (dft_client_adpt)』

このパラメーターは、DCE ディレクトリー・サービスが使用されているかどうかを示します。

このパラメーターを NONE に設定すると、 CONNECT または ATTACH 要求のターゲットとしてローカルのディレクトリー・ファイルだけが検索されます。しかし、 *dir_path_name* および *dir_obj_name* パラメーターを使用して、データベース・インスタンスとデータベースの名前を DCE ネームスペースに記録することはできます。

このパラメーターを DCE に設定すると、このデータベース・マネージャー・インスタンス内で実行しているアプリケーションが CONNECT または ATTACH 要求のターゲットを検出できなかった場合に、DCE ディレクトリーが検索されます。

これらの値と等価な数値と API 定数については、 *管理 API 解説書* を参照してください。

DCE ネームスペース内のディレクトリー・パス名 (dir_path_name)

構成タイプ データベース・マネージャー

適用範囲

- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー
- UNIX および OS/2 クライアント
- UNIX および OS/2 ローカル・クライアントをもつデータベース・サーバー
- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー

パラメーター・タイプ 構成可能

デフォルト /.:/subsys/database/

関連パラメーター

- 460ページの『DCE ネームスペース内のオブジェクト名 (dir_obj_name)』
- 458ページの『ディレクトリー・サービス・タイプ (dir_type)』
- 460ページの『経路指定情報オブジェクト名 (route_obj_name)』

この値と *dir_obj_name* パラメーターの値を連結すると、グルーバル・ネームスペースにおけるデータベース・マネージャー・インスタンスの固有名ができます。

このインスタンス内で実行しているすべてのクライアント・アプリケーションは、CONNECT または ATTACH 要求のデフォルト・パス名としてその固有名を使用します。ただし、DB2DIRPATHNAME 環境変数によって指定変更された場合は使用しません。

推奨事項: DCE 管理担当者から与えられた名前を使用してください。

DCE ネームスペース内のオブジェクト名 (dir_obj_name)

構成タイプ データベース・マネージャー、データベース

適用範囲

- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー
- UNIX および OS/2 クライアント
- UNIX および OS/2 ローカル・クライアントをもつデータベース・サーバー
- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー

パラメーター・タイプ 構成可能

デフォルト Null

関連パラメーター

- 458ページの『ディレクトリー・サービス・タイプ (dir_type)』
- 459ページの『DCE ネームスペース内のディレクトリー・パス名 (dir_path_name)』

ディレクトリー内の、ユーザーのデータベース・マネージャー・インスタンス (またはユーザーのデータベース) を表しているオブジェクト名。この値と *dir_path_name* の値を連結すると、 *dir_type* パラメーターで指定したディレクトリー・サービスが管理するネームスペースにある、データベース・マネージャー・インスタンスまたはデータベースを固有に識別するグローバル名になります。

このパラメーターは、 *dir_path_name* パラメーターが指定されている場合だけ有効です。

構成パラメーター、*dir_path_name* と *dir_obj_name* の全長は、 255 文字より少なくなければなりません。

詳しくは、*管理の手引き: インプリメンテーション* の『分散コンピューティング環境 (DCE) ディレクトリー・サービスの使用』を参照してください。

経路指定情報オブジェクト名 (route_obj_name)

構成タイプ データベース・マネージャー

適用範囲

- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー
- クライアント
- ローカル・クライアントをもつデータベース・サーバー
- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー

パラメーター・タイプ 構成可能

デフォルト Null

関連パラメーター

- 459ページの『DCE ネームスペース内のディレクトリー・パス名 (dir_path_name)』
- 458ページの『ディレクトリー・サービス・タイプ (dir_type)』

このパラメーターは、デフォルトの経路指定情報のオブジェクト項目を指定します。この項目は、 DRDA サーバーへのアクセスを試行しているすべてのクライアント・アプリケーションで使用されます。これは、OS/2 および UNIX ベースの環境にだけ適用されます。

このパラメーターの値が */.:/* または */.../* で始まっている場合は、その値はそのまま使用されます。それ以外の場合は、その値は *dir_path_name* パラメーター (または DB2DIRPATHNAME 環境変数) の値に付け加えられ、経路指定情報オブジェクトの名前が形成されます。

環境変数 DB2ROUTE を使用して、デフォルトを指定変更することができます。

このパラメーターは、 *dir_type* パラメーターが DCE に設定されている場合だけ有効です。

詳しくは、*管理の手引き: インプリメンテーション* の『分散コンピューティング環境 (DCE) ディレクトリー・サービスの使用』を参照してください。

デフォルト・クライアント通信プロトコル (dft_client_comm)

構成タイプ データベース・マネージャー

適用範囲

- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー
- クライアント
- ローカル・クライアントをもつデータベース・サーバー

- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー

パラメーター・タイプ 構成可能

デフォルト[範囲] Null [Null; TCPIP; APPC; IPXSPX (OS/2 のみ); NETBIOS (OS/2 のみ)]

関連パラメーター 458ページの『ディレクトリー・サービス・タイプ (dir_type)』

このパラメーターは、このインスタンス中のクライアント・アプリケーションがリモート接続用に使用する通信プロトコルを示します。この内容は、 1 つまたは複数のトークンから成る文字ストリングです。複数のトークンを指定する場合は、コンマを使ってそれらを区切ってください。作業環境のことを考慮すると、トークンの順序は重要なものです。

このパラメーターは、DCE と一緒にだけ使用でき、 OS/2 および UNIX ベースの環境にだけ適用されます。

DB2CLIENTCOMM 環境変数を設定すると、このパラメーターの値を一時的に指定変更することができます。

このパラメーターの値が Null で、環境変数が設定されていないと、サーバーの・グローバル・ディレクトリー・オブジェクトで指定されている最初のプロトコルが使用されます。

dir_type が NONE に設定されていると、このパラメーターは無視されます。

推奨事項: 最も頻繁に使用するプロトコルを最初に指定してください。

デフォルトのクライアント・アダプター番号 (dft_client_adpt)

構成タイプ データベース・マネージャー

適用範囲

- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー
- クライアント
- ローカル・クライアントをもつデータベース・サーバー

パラメーター・タイプ 構成可能

デフォルト[範囲] 0 [0 ～ 15]

関連パラメーター

- 461ページの『デフォルト・クライアント通信プロトコル (dft_client_comm)』

- 458ページの『ディレクトリー・サービス・タイプ (dir_type)』。 (dir_type が DCE に設定されている場合)

このパラメーターは、サーバーの nname が DCE セル・ディレクトリー・サービス (CDS) から抽出されている NETBIOS プロトコルのデフォルトのクライアント・アダプター番号を定義します。このパラメーターは、OS/2 環境にだけ適用されます。

このパラメーターは DCE と共にしか使用できません。

DB2CLIENTADPT 環境変数を設定すると、このパラメーターの値を一時的に指定変更することができます。この環境変数に非数字または範囲外の番号が入っていると、アダプター番号は 0 になります。

DB2 ディスカバリー機能

DB2 ディスカバリー機能を設定するために、以下のパラメーターを使用できます。

- 『データベースのディスカバリー (discover_db)』
- 『ディスカバリー・モード (discover)』
- 465ページの『通信プロトコルの探索ディスカバリー (discover_comm)』
- 466ページの『サーバー・インスタンスのディスカバリー (discover_inst)』

データベースのディスカバリー (discover_db)

構成タイプ データベース (Database)

パラメーター・タイプ 構成可能

デフォルト[範囲] Enable[Disable、Enable]

このパラメーターは、ディスカバリー要求がサーバーで受信されたときに、データベースの情報がクライアントに戻されないようにするために使用されます。

このパラメーターのデフォルトは、このデータベースに対してディスカバリーを使用可能にします。

このパラメーター値を「使用不能 (Disable)」に変更することにより、機密データの入ったデータベースをディスカバリー処理から隠すことができます。この処置は、データベースに対する他のセキュリティー管理に加えて行うことができます。

これらの値と等価な数値と API 定数については、 *管理 API 解説書* を参照してください。

ディスカバリー・モード (discover)

構成タイプ データベース・マネージャー

適用範囲	• ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー • クライアント • ローカル・クライアントをもつデータベース・サーバー • ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー • ローカル・クライアントをもつサテライト・データベース・サーバー
パラメーター・タイプ	構成可能
デフォルト[範囲]	SEARCH [DISABLE, KNOWN, Search]
関連パラメーター	465ページの『通信プロトコルの探索ディスカバリー (discover_comm)』

管理サーバーでは、この構成パラメーターは、 DB2ADMIN の開始時に開始されるディスカバリー・モードのタイプを判別します。

- DB2ADMIN の開始時に *discover* = SEARCH になっている場合、 *discover_comm* で指定されているプロトコルごとに、探索ディスカバリー接続マネージャーが開始されます。さらに、DB2COMM レジストリー変数で指定されているプロトコルごとに、接続マネージャーが開始されます。これにより、クライアントからの探索ディスカバリー要求を管理サーバーが処理することができます。 SEARCH では、 KNOWN ディスカバリーで提供される機能のスーパーセットが提供されます。 *discover* = SEARCH の場合、管理サーバーは、クライアントからの探索と既知の両方のディスカバリー要求を処理することができます。
- DB2ADMIN の開始時に *discover* = KNOWN になっている場合、開始されるのは、DB2COMM レジストリー変数で指定されている接続マネージャーだけです。これらの接続マネージャーは、KNOWN ディスカバリー要求を処理できます。
- DB2ADMIN の開始時に *discover* = DISABLE になっている場合、管理サーバーはいかなるタイプのディスカバリー要求も処理しません。

サーバー・インスタンスでは、 *discover* = DISABLE になっている場合、このサーバー・インスタンスの情報がクライアントから隠されます。任意のクライアントからこのシステムに対して既知ディスカバリー要求が出される場合、管理サーバーはこのインスタンスに関する情報の収集を行いません。

クライアントでは、以下の処理が行われます。

- *discover* = SEARCH の場合、クライアントは、探索ディスカバリー要求を出して、ネットワーク上の DB2 サーバー・システムを見つけることができます。探索ディスカ

バリーでは、既知ディスカバリーで提供される機能のスーパーセットが提供されます。 *discover* = SEARCH の場合、クライアントから探索と既知の両方のディスカバリー要求を出すことができます。

- *discover* = KNOWN の場合、クライアントから出せるのは既知ディスカバリー要求だけです。特定システムにある管理サーバーの接続情報を指定すれば、その DB2 システム上のすべてのインスタンスおよびデータベースの情報がそのクライアントに戻されます。
- *discover* = DISABLE の場合、そのクライアントではディスカバリーを行うことができません。

デフォルトのディスカバリー・モードは SEARCH です。

これらの値と等価な数値と API 定数については、 *管理 API 解説書* を参照してください。

DB2 ディスカバリー機能についての詳細は、使用するプラットフォームに該当する*概説およびインストール* を参照してください。

通信プロトコルの探索ディスカバリー (discover_comm)

構成タイプ データベース・マネージャー

適用範囲

- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー
- クライアント
- ローカル・クライアントをもつデータベース・サーバー
- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー
- ローカル・クライアントをもつサテライト・データベース・サーバー

パラメーター・タイプ 構成可能

デフォルト[範囲] None [NETBIOS および TCPIP の任意の組み合わせ]

関連パラメーター 463ページの『ディスカバリー・モード (discover)』

管理サーバーでは、このパラメーターは、 DB2ADMIN の開始時に開始される探索ディスカバリー・マネージャーを定義します。これらのマネージャーは、クライアントからの探索ディスカバリー要求を扱います。

注: *discover_comm* で定義されているプロトコルは、 DB2COMM レジストリー変数でも指定しなければなりません。

クライアントでは、このパラメーターはクライアントが探索ディスカバリー要求を出すために使用するプロトコルを定義します。

複数のプロトコルを、間をコンマで区切って指定することができます。また、パラメーターをブランクにしておくこともできます。

このパラメーターのデフォルトは「NONE」です。これは、探索ディスカバリーの通信プロトコルがないことを意味します。

サーバー・インスタンスのディスカバリー (discover_inst)

構成タイプ データベース・マネージャー

適用範囲

- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー
- クライアント
- ローカル・クライアントをもつデータベース・サーバー
- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー

パラメーター・タイプ 構成可能

デフォルト[範囲] ENABLE [ENABLE, DISABLE]

| このパラメーターは、このインスタンスが DB2 ディスカバリー機能によって検出できるかどうかを指定します。デフォルト enable は、インスタンスが検出可能であること、 disable はインスタンスの検出が禁止されることを指定します。

これらの値と等価な数値と API 定数については、 *管理 API 解説書* を参照してください。

| DB2 ディスカバリー機能についての詳細は、使用するプラットフォームに該当する*概説およびインストール* を参照してください。

区画データベース

次のパラメーター・グループは、並列操作および区分データベース環境についての情報を記述しています。

- 467ページの『通信』
- 473ページの『並列処理』

通信

次のパラメーターは、区分データベース環境での通信について記述しています。

- 『接続経過時間 (conn_elapse)』
- 『FCM メッセージ・アンカーの個数 (fcm_num_anchors)』
- 468ページの『FCM バッファー数 (fcm_num_buffers)』
- 469ページの『FCM 接続項目数 (fcm_num_connect)』
- 470ページの『FCM 要求ブロック数 (fcm_num_rqb)』
- 471ページの『ノード接続の再試行 (max_connretries)』
- 471ページの『ノード間の最大の時刻差 (max_time_diff)』
- 472ページの『タイムアウトの開始と停止 (start_stop_time)』

接続経過時間 (conn_elapse)

構成タイプ データベース・マネージャー

適用範囲 ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー

パラメーター・タイプ 構成可能

デフォルト[範囲] 10 [0 ～ 100]

計測単位 秒

関連パラメーター 471ページの『ノード接続の再試行 (max_connretries)』

このパラメーターは、 TCP/IP 接続を 2 つのデータベース区画サーバー間で設定するための秒数を指定します。このパラメーターで指定された時間内に接続が完了すると、通信が設定されます。失敗すると、通信を設定するために再び試行されます。*max_connretries* パラメーターで指定された回数だけ接続が試行され、そのすべてがタイムアウトになると、エラーになります。

FCM メッセージ・アンカーの個数 (fcm_num_anchors)

構成タイプ データベース・マネージャー

適用範囲

- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー
- ローカル・クライアントをもつデータベース・サーバー
- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー

- ローカル・クライアントをもつサテライト・データベース・サーバー

パラメーター・タイプ 構成可能

デフォルト[範囲] -1 [-1、128 ～ *fcm_num_rqb*]

非区分データベース・システムでは、 *intra_parallel* パラメーターが活動状態でないと、このパラメーターを使用できません。

関連パラメーター

- 470ページの『FCM 要求ブロック数 (fcm_num_rqb)』
- 474ページの『区画内並列処理機能の使用可能化 (intra_parallel)』

このパラメーターは、 FCM メッセージ・アンカー の数を指定します。エージェントは、メッセージ・アンカーを使用してエージェント間でメッセージを送信します。デフォルト (-1) は、 *fcm_num_rqb* に指定された値の 75 % を示します。

FCM バッファー数 (fcm_num_buffers)

構成タイプ データベース・マネージャー

適用範囲

- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー
- ローカル・クライアントをもつデータベース・サーバー
- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー
- ローカル・クライアントをもつサテライト・データベース・サーバー

パラメーター・タイプ 構成可能

デフォルト[範囲] 512、1 024、または 4 096 [128 ～ *fcm_num_rqb*]

- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー: デフォルトは 1 024
- ローカル・クライアントをもつデータベース・サーバー: デフォルトは 512
- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー: デフォルトは 4 096

単一区画データベース・システムでは、*intra_parallel* パラメーターが活動状態でないと、このパラメーターを使用できません。

このパラメーターは、データベース・サーバー間およびデータベース・サーバー内で内部通信 (メッセージ) に使用される 4 KB バッファーの数を指定します。

FCM について詳しくは、 *管理の手引き: インプリメンテーション* の『FCM 通信の使用可能化』を参照してください。

1 つのプロセッサー上に複数の論理ノードがある場合は、このパラメーターの値を大きくする必要があるかもしれません。また、システムを使用するユーザー数、システム上のデータベース区画サーバーの数、またはアプリケーションの複雑さのためにメッセージ・バッファーが不足する場合も、このパラメーターの値を増やす必要があります。

AIX 以外のシステムで複数の論理ノードを使用する場合は、 *fcm_num_buffers* 個のバッファーの 1 つのプールが同一マシン上のすべての複数論理ノードで共用されます。一方、AIX の場合は、次のようになります。

- データベース・マネージャーが使用する汎用メモリーが十分にある場合は、 FCM バッファー・ヒープはそこから割り振られる。この場合は、各データベース区画サーバーが *fcm_num_buffers* 個のバッファーを持ちます。複数のデータベース区画サーバーで FCM バッファーの 1 つのプールを共用することはありません (これは DB2 バージョン 5 から新たに追加された機能です)。
- データベース・マネージャーが使用する汎用メモリーが十分にない場合は、 FCM バッファー・ヒープは別のメモリー域 (AIX 共用メモリー・セット) から割り振られる。これを同一マシンのすべての複数論理ノードが共用します。 *fcm_num_buffers* の 1 つのプールが同一マシンのすべての複数論理ノードで共用されます。これは、非 AIX システムの場合と同じであり、また、 AIX で DB2 パラレル・エディションのバージョン 1.2 を使用する場合とも同じです。

既存のパラレル・エディションを AIX で使用している場合の推奨事項: 複数の論理ノードを使用する場合、パラレル・エディションのバージョン 1.2 で使用していた *fcm_num_buffers* の値をそのまま使用すると、マシンあたりの使用ストレージがかなり大きくなる可能性があります。たとえば、4 つの複数論理ノード構成の場合、以前の FCM バッファーの 4 倍を使うことになります。

使用する値を再検討してください。複数論理ノードを持つマシン (複数のマシンの場合もある) に合計いくつの FCM バッファーが割り振られることになるかを考えてください。その上で、*fcm_num_buffers* の値を変更することができます。

FCM 接続項目数 (fcm_num_connect)

構成タイプ データベース・マネージャー

適用範囲

- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー
- ローカル・クライアントをもつデータベース・サーバー
- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー
- ローカル・クライアントをもつサテライト・データベース・サーバー

パラメーター・タイプ 構成可能

デフォルト[範囲] -1 [-1、128 〜 *fcm_num_rqb*]

非区分データベース・システムでは、 *intra_parallel* パラメーターが活動状態でないと、このパラメーターを使用できません。

関連パラメーター 『FCM 要求ブロック数 (fcm_num_rqb)』

このパラメーターは、 FCM *接続項目* 数を指定します。エージェントは、接続項目を使用してお互いにデータを渡します。デフォルト (-1) は、 *fcm_num_rqb* に指定された値の 75 % を示します。

FCM 要求ブロック数 (fcm_num_rqb)

構成タイプ データベース・マネージャー

適用範囲

- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー
- ローカル・クライアントをもつデータベース・サーバー
- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー
- ローカル・クライアントをもつサテライト・データベース・サーバー

パラメーター・タイプ 構成可能

デフォルト[範囲]

UNIX 32 ビット・プラットフォーム
256、512、2 048 [128 〜 120 000]

UNIX 64 ビット・プラットフォーム
256、512、2 048 [128 〜 524 288]

OS/2 および Windows NT

10 000 [250 ～ 2 097 152]

- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー: デフォルトは 512
- ローカル・クライアントをもつデータベース・サーバー: デフォルトは 256
- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー: デフォルトは 2 048

非区分データベース・システムでは、 *intra_parallel* パラメーターが活動状態でないと、このパラメーターを使用できません。

このパラメーターは、 FCM *要求ブロック* 数を指定します。要求ブロックは、 FCM デーモンとエージェントの間、またはエージェント間で情報を渡すためのメディアです。

システムを使用するユーザー数、システム上のデータベース区画サーバーの数、および実行される照会の複雑さにしたがって、要求ブロックの必要量が異なります。最初はデフォルトで開始し、データベース・システム・モニターの結果を使用して、このパラメーターのチューニングを行ってください。

ノード接続の再試行 (max_connretries)

構成タイプ データベース・マネージャー

適用範囲 ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー

パラメーター・タイプ 構成可能

デフォルト[範囲] 5 [0 ～ 100]

関連パラメーター 467ページの『接続経過時間 (conn_elapse)』

2 つのデータベース区画サーバー間で通信を設定しようとして失敗した (たとえば、*conn_elapse* パラメーターで指定されている値になった) 場合に、 *max_connretries* は、データベース区画サーバーに対して行うことのできる接続再試行の数を指定します。このパラメーターに指定された値を超えると、エラーが戻されます。

ノード間の最大の時刻差 (max_time_diff)

構成タイプ データベース・マネージャー

適用範囲	ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー
パラメーター・タイプ	構成可能
デフォルト[範囲]	60 [1 ～ 1 440]
計測単位	分

データベース区画サーバーは、それぞれシステム・クロックを持っています。このパラメーターは、ノード構成ファイルにリストされているデータベース区画サーバー間で許される時刻差の最大値を分の単位で指定します。

2 つ以上のデータベース区画サーバーが 1 つのトランザクションに関連付けられていて、それらのクロックがこのパラメーターで指定されている時間内で同期していないと、そのトランザクションは拒否され、警告またはエラー・メッセージが db2diag.log ファイルに記録されます。 (トランザクションは、データ修正が関係している場合だけ拒否されます。)

DB2 ユニバーサル・データベース エンタープライズ拡張エディションではコーディネート*世界時* (UTC) が使用されるので、このパラメーターを設定するときに異なる時間帯のことを考慮する必要はありません。コーディネート世界時はグリニッジ標準時と同じです。

タイムアウトの開始と停止 (start_stop_time)

構成タイプ	データベース・マネージャー
適用範囲	ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー
パラメーター・タイプ	構成可能
デフォルト[範囲]	10 [1 ～ 1 440]
計測単位	分

このパラメーターは、区分データベース環境にだけ適用されます。これは、すべてのデータベース区画サーバーが DB2START または DB2STOP コマンドに応答しなければならない時間を分単位で指定します。また、これは ADDNODE 操作時のタイムアウト値としても使用されます。

指定された時間内に DB2START コマンドに応答しないデータベース区画サーバーは、そのインスタンスのために home ディレクトリーの sqllib サブディレクトリーの中の log サブディレクトリーにある db2start エラー・ログにメッセージを送ります。これらのノードを再始動するには、その前にそれらのノードに DB2STOP を出す必要があります。

指定された時間内に DB2STOP コマンドに応答しないデータベース区画サーバーは、そのインスタンスのために home ディレクトリーの sqllib サブディレクトリーの中の log サブディレクトリーにある db2stop エラー・ログにメッセージを送ります。 DB2STOP は、応答しないデータベース区画サーバーのそれぞれに出すことも、またはすべてのデータベース区画サーバーに出すこともできます。 (それらの内、すでに停止しているサーバーは、すでに停止していることを示すメッセージを戻します。)

並列処理

次のパラメーターは、並列処理について記述しています。

- 『照会の最大並列処理度 (max_querydegree)』
- 474ページの『区画内並列処理機能の使用可能化 (intra_parallel)』

照会の最大並列処理度 (max_querydegree)

構成タイプ データベース・マネージャー

適用範囲

- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー
- ローカル・クライアントをもつデータベース・サーバー
- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー
- ローカル・クライアントをもつサテライト・データベース・サーバー

パラメーター・タイプ 構成可能

デフォルト[範囲] -1 (ANY) [ANY、 1 〜 32 767] (ANY は、システムによって決定されることを意味します)

関連パラメーター

- 449ページの『デフォルトの程度 (dft_degree)』
- 474ページの『区画内並列処理機能の使用可能化 (intra_parallel)』

このパラメーターは、データベース・マネージャーのこのインスタンスで実行する任意の SQL ステートメントに使用される、区画内並列処理の最大度を指定します。 SQL ステートメントは、その実行時には、 1 つの区画内でこの数を超える並列操作を使用することはありません。データベース区画で区画内並列処理を使用可能にするには、*intra_parallel* パラメーターを 『YES』 に設定する必要があります。

この構成パラメーターのデフォルト値は -1 です。この値は、システムが、最適化プログラムが決定した並列処理度を使用することを意味します。これ以外の値の場合は、ユーザー指定の値が使用されます。

注: SQL ステートメントのための並列処理度は、 CURRENT DEGREE 特殊レジスターまたは DEGREE バインド・オプションを使用してステートメントのコンパイル時に指定することができます。

活動アプリケーションに対する並列処理の照会最大度は SET RUNTIME DEGREE コマンドで変更可能です。実際の実行時程度は、次の中の最も小さいものです。

- *max_querydegree* 構成パラメーター
- アプリケーション実行時の程度
- SQL ステートメントのコンパイル時の程度

照会の実際の並列処理度決定の例外は、索引の作成時に起こります。この場合、*intra_parallel* が 『YES』 で、表が十分に大きくて複数プロセッサーが使用できるときは、索引の作成では、オンライン・プロセッサーの数に 1 を加えた数 (最大 6 まで) が使用されます。上述のその他のパラメーター、バインド・オプション、または特殊レジスターからの影響はありません。

区画内並列処理機能の使用可能化 (intra_parallel)

構成タイプ
データベース・マネージャー

適用範囲

- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー
- ローカル・クライアントをもつデータベース・サーバー
- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー
- ローカル・クライアントをもつサテライト・データベース・サーバー

パラメーター・タイプ
構成可能

デフォルト[範囲]
NO (0) [SYSTEM (-1)、NO (0)、YES (1)]

値 -1 は、このパラメーターの値を、データベース・マネージャーが実行しているハードウェアに基づいて 『YES』 または 『NO』 に設定します。

関連パラメーター
473ページの『照会の最大並列処理度 (max_querydegree)』

このパラメーターは、データベース・マネージャーが区画内並列処理機能を使用できるか否かを指定します。

このパラメーターが ″YES″ であるときに並列処理機能によるパフォーマンスの向上を利用できる操作の例としては、データベース照会と索引の作成があります。

注: このパラメーターの値を変更すると、パッケージがデータベースに再バインドされることがあります。この場合、再バインド中にパフォーマンスの低下が起きる可能性があります。

インスタンス管理

データベース・マネージャーのインスタンスを管理するためのパラメーターがいくつかあります。この種のパラメーターは、次のカテゴリーに分けることができます。

- 『診断』
- 478ページの『データベース・システム・モニター・パラメーター』
- 480ページの『システム管理』
- 488ページの『インスタンス管理』

診断

次のパラメーターは、データベース・マネージャーで利用できる診断情報を制御します。

- 『診断エラーのキャプチャー・レベル (diaglevel)』
- 476ページの『診断データのディレクトリー・パス (diagpath)』
- 477ページの『通知レベル (notifylevel)』

診断エラーのキャプチャー・レベル (diaglevel)

構成タイプ データベース・マネージャー

適用範囲

- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー
- クライアント
- ローカル・クライアントをもつデータベース・サーバー
- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー
- ローカル・クライアントをもつサテライト・データベース・サーバー

パラメーター・タイプ 構成可能

デフォルト[範囲] 3 [0 ～ 4]

関連パラメーター 476ページの『診断データのディレクトリー・パス (diagpath)』

このパラメーターは、 db2diag.log ファイルに記録される診断エラーのタイプを決定します。有効な値は次のとおりです。

0 - 診断データをキャプチャーしない

1 - 重大エラーのみ

2 - すべてのエラー

3 - すべてのエラーおよび警告

4 - すべてのエラー、警告、および通知メッセージ

diaglevel パラメーターの値に基づいて作成されるエラー・ファイル、イベント・ログ・ファイル (Windows NT のみ)、アラート・ログ・ファイル、および任意のダンプ・ファイルが入るディレクトリーは、 *diagpath* 構成パラメーターを使用して指定されます。

推奨事項: 問題を解決しやすくするために、このパラメーターの値を大きくし、追加の問題判別データを収集することができます。

診断データのディレクトリー・パス (diagpath)

構成タイプ データベース・マネージャー

適用範囲

- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー
- クライアント
- ローカル・クライアントをもつデータベース・サーバー
- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー
- ローカル・クライアントをもつサテライト・データベース・サーバー

パラメーター・タイプ 構成可能

デフォルト[範囲] Null [任意のパス名]

関連パラメーター 475ページの『診断エラーのキャプチャー・レベル (diaglevel)』

このパラメーターは、 DB2 診断情報の完全修飾パスを指定します。このディレクトリーには、使用するプラットフォームにしたがって、ダンプ・ファイル、トラップ・ファイル、エラー・ログ・ファイル、およびアラート・ログ・ファイルを入れることができます。

Null を指定すると、診断情報は次のいずれかのディレクトリーまたはフォルダーにあるファイルに書き込まれます。

- OS/2 およびサポートされている Windows 環境の場合:
  - DB2INSTPROF 環境変数またはキーワードが設定**されていない**場合、情報は x:¥SQLLIB¥DB2INSTANCE に書き込まれます。 x:¥SQLLIB は DB2PATH レジストリー変数または環境変数で指定されているドライブ参照およびディレクトリーを、DB2INSTANCE はインスタンス所有者の名前を表しています。

    注: このディレクトリーは SQLLIB という名前である必要はありません。
  - DB2INSTPROF 環境変数またはキーワードが設定されている場合、情報は x:¥DB2INSTPROF¥DB2INSTANCE に書き込まれます。 DB2INSTPROF はインスタンス・プロファイル・ディレクトリー名を、 DB2INSTANCE はインスタンス所有者の名前を表しています。
- UNIX ベース環境の場合、 INSTHOME/sqllib/db2dump に書き込まれます。 INSTHOME はインスタンス所有者のホーム・ディレクトリーを表しています。
- Macintosh 環境の場合、DB2 フォルダーに書き込まれます。

推奨事項: 複数インスタンスの diagpath には、デフォルトを使用するか、またはその場所をまとめてください。

区分データベース環境では、指定するパスはファイル共用システムにある必要があります。

通知レベル (notifylevel)

構成タイプ データベース・マネージャー

適用範囲

- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー (Windows NT)
- クライアント (Windows NT)
- ローカル・クライアントをもつデータベース・サーバー (Windows NT)
- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー (Windows NT)
- ローカル・クライアントをもつサテライト・データベース・サーバー (Windows 95、Windows 98、および Windows NT)

パラメーター・タイプ 構成可能

デフォルト[範囲] 2 [0 ～ 4]

このパラメーターは、ファイルに書き込まれる管理通知エラー・メッセージのタイプを指定します。サテライト・ノード・タイプのサーバーの場合、エラーは *instance*.nfy という通知ファイルに書き込まれます。他のすべてのノード・タイプの場合、このパラ

メーターが使用できるのは Windows NT プラットフォーム上だけです。その場合、エラーは Windows NT イベント・ログに書き込まれます。エラーは、DB2、収集および変更適用プログラム、および独自のアプリケーションによって書き込むことができます。

このパラメーターの有効な値は、次のとおりです。

0 -- 診断データをキャプチャーしない

1 -- 重大エラーのみ

2 -- すべてのエラー

3 -- すべてのエラーおよび警告

4 -- すべてのエラー、警告、および通知メッセージ

ユーザー・アプリケーションで通知ファイルまたは Windows NT イベント・ログに書き込めるようにするには、 db2AdminMsgWrite API を呼び出す必要があります。この API についての詳細は、*管理 API 解説書* を参照してください。

推奨事項: 問題を解決しやすくするために、このパラメーターの値を大きくし、追加の問題判別データを収集することができます。

データベース・システム・モニター・パラメーター

次のパラメーターを使用すると、データベース・システム・モニターの各種の面を制御することができます。

- 『デフォルト データベース・システム・モニター・スイッチ (dft_monswitches)』

デフォルト データベース・システム・モニター・スイッチ (dft_monswitches)

構成タイプ データベース・マネージャー

適用範囲

- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー
- ローカル・クライアントをもつデータベース・サーバー
- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー
- ローカル・クライアントをもつサテライト・データベース・サーバー

パラメーター・タイプ 構成可能

デフォルト すべてのスイッチをオフにする

このパラメーターのビットはそれぞれスイッチを表しており、これらのビットを使って多数のスイッチを設定することができます。データベース・マネージャー構成を更新する際に使うインターフェースによっては、このパラメーターを直接更新することができます。また、次のパラメーターを設定して、これらのスイッチを個別に更新することもできます。

dft_mon_uow スナップショット・モニターの作業単位 (UOW) スイッチのデフォルト値

dft_mon_stmt スナップショット・モニターのステートメント・スイッチのデフォルト値

dft_mon_table スナップショット・モニターの表スイッチのデフォルト値

dft_mon_bufpool スナップショット・モニターのバッファー・プール・スイッチのデフォルト値

dft_mon_lock スナップショット・モニターのロック・スイッチのデフォルト値

dft_mon_sort スナップショット・モニターの分類スイッチのデフォルト値

これらデータベース・システム・モニター・スイッチのいずれかを変更すると、その変更はすぐに反映されます。つまり、データベース・マネージャーを停止して再始動する必要はありません。

注: 既存のモニター・アプリケーションがスイッチの新しいデフォルト値を自動的に使用することはありません。新しい値 (複数可) を使用するには、アプリケーションを終了し、インスタンスに再び追加しなければなりません。

スナップショット・モニターの詳細、およびそれがモニター・スイッチを使用する方法については、システム・モニター *手引きおよび解説書* を参照してください。

推奨事項: いずれかのスイッチをオンにすると、そのスイッチに関連したモニター・データを収集するようデータベース・マネージャーに指示が出されます。追加のモニター・データを集めると、データベース・マネージャーのオーバーヘッドが増すので、システム・パフォーマンスはその影響を受けます。

どのモニター・アプリケーションも、モニター要求を最初に出したとき (たとえば、スイッチの設定、イベント・モニターの活動化、スナップショットの獲得) に、これらのデフォルトのスイッチ設定値を継承します。データベース・マネージャーの開始時からデータを収集したい場合に限り、構成ファイルでスイッチをオンにしてください。 (スイッチをオンにしなくても、各モニター・アプリケーションは対応するスイッチを独自に設定することができます。また、スイッチが設定されたときからのデータが収集されます。)

システム管理

次のパラメーターは、システム管理と関連があります。

- 『通信帯域幅 (comm_bandwidth)』
- 481ページの『CPU 速度 (cpuspeed)』
- 481ページの『並行活動データベースの最大数 (numdb)』
- 483ページの『トランザクション・プロセッサー・モニター名 (tp_mon_name)』
- 485ページの『マシン・ノード・タイプ (nodetype)』
- 486ページの『デフォルトのチャージバック会計 (dft_account_str)』
- 487ページの『Java Development Kit 1.1 インストール・パス (jdk11_path)』
- 487ページの『連合データベース・システムのサポート (federated)』

通信帯域幅 (comm_bandwidth)

構成タイプ
データベース・マネージャー

適用範囲
ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー

パラメーター・タイプ
構成可能

デフォルト[範囲]
-1 [.1 〜 100 000]

値 -1 は、このパラメーターをデフォルトにリセットします。デフォルト値は、高速スイッチを使用するか否かに基づいて算出されます。

計測単位
メガバイト / 秒

通信帯域幅として計算された値 (1 秒あたりのメガバイト) は、区分データベース・システムのデータベース区画サーバー間である種の操作を行うコストを見積もるために SQL 最適化プログラムが使用します。最適化プログラムは、クライアントとサーバーの間の通信コストのモデルを作成するわけではないので、このパラメーターはデータベース区画サーバー間の名目上の帯域幅 (これがある場合には) だけを反映します。

この値を明示的に設定して、テスト・システムに実稼働環境のモデルを作成したり、ハードウェア・アップグレードの影響を評価することができます。

推奨事項: 異なる環境のモデルを作成したい場合だけ、このパラメーターを調整してください。

アクセス・パスを判別するために、最適化プログラムは通信帯域幅を使用します。このパラメーターを変更してから、アプリケーションの再バインドを考慮する必要があります (REBIND PACKAGE コマンドを使用)。

CPU 速度 (cpuspeed)

構成タイプ データベース・マネージャー

適用範囲

- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー
- ローカル・クライアントをもつデータベース・サーバー
- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー
- ローカル・クライアントをもつサテライト・データベース・サーバー

パラメーター・タイプ 構成可能

デフォルト[範囲] -1 [1^{-10} ～ 1] 値を -1 に設定すると、測定プログラムの実行に基づいてパラメーター値がリセットされます。

計測単位 秒

SQL 最適化プログラムは、CPU 速度 (1 つの命令あたりのミリ秒) を使って、特定の操作を実行した場合のコストを見積もります。このパラメーターの値は、データベース・マネージャーをインストールするときに、 CPU 速度を測定するために設計されたプログラムの出力を基にして自動的に設定されます。このプログラムは、次のいずれかが原因でベンチマーク結果が利用できない場合に実行されます。

- プラットフォームが `db2spec.dat` ファイルをサポートしていない
- `db2spec.dat` ファイルが**ない**
- IBM RISC システム /6000 モデル 530H のデータがファイルに含まれていない
- 現在使用しているマシンのデータがファイルに含まれていない

この値を明示的に設定して、テスト・システムに実稼働環境のモデルを作成したり、ハードウェア・アップグレードの影響を評価することができます。 -1 に設定すると、*cpuspeed* が再計算されます。

推奨事項: 異なる環境のモデルを作成したい場合だけ、このパラメーターを調整してください。

アクセス・パスを判別するために、最適化プログラムは CPU の速度を使用します。このパラメーターを変更してから、アプリケーションの再バインドを考慮する必要があります (REBIND PACKAGE コマンドを使用)。

並行活動データベースの最大数 (numdb)

構成タイプ データベース・マネージャー

適用範囲

- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー
- ローカル・クライアントをもつデータベース・サーバー
- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー
- ローカル・クライアントをもつサテライト・データベース・サーバー

パラメーター・タイプ 構成可能

デフォルト[範囲]

UNIX 8 [1 〜 256]

OS/2 および Windows NT ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー 8 [1 〜 256]

OS/2 および Windows NT ローカル・クライアントをもつデータベース・サーバー および ローカル・クライアントをもつサテライト・データベース・サーバー 3 [1 〜 256]

計測単位 カウンター

このパラメーターは、並行して活動できる (つまり、複数のアプリケーションが接続できる) ローカル・データベースの数を指定します。区分データベース環境では、このパラメーターは、データベース区画サーバーの活動状態のデータベース区画数を制限します。これは、このサーバーがそのアプリケーションの調整プログラム・ノードであるか否かには関係ありません。

各データベースはストレージを使用し、活動データベースは新規の共用メモリー・セグメントを使用するので、マシン上の個別データベースの数を制限すると、システム・リソースの使用を減らすことができます。ただし、意味もなくデータベースの数を減らすことが目的ではありません。つまり、互いの関係を無視してすべてのデータを 1 つのデータベースにまとめれば使用するディスク・スペースは減少しますが、得策とはいえません。通常は、機能の点で関係のある情報だけを同一データベースに保持するのが得策です。

推奨事項: 一般的には、この値をすでにデータベース・マネージャーに定義されているデータベースの実際の数に設定し、短い期間 (たとえば、6 か月から 1 年) の間隔で、データベース数の増加に合わせて適切に大きくしていくのが最善の方法です。値を過度に大きくしないでください。ただし、このパラメーターを頻繁に更新しないで新しいデータベースを追加できるように設定してください。

numdb パラメーターを変更すると、割り振られるメモリーの合計量は影響を受けます。そのため、このパラメーターを頻繁に更新するのは勧められていません。このパラメーターを更新する場合は、データベースまたはデータベースに接続するアプリケーション用のメモリーを割り振ることができる、次の他の構成パラメーターについても考慮してください。

- 348ページの『バッファー・プール・サイズ (buffpage)』
- 357ページの『ロック・リスト用最大ストレージ (locklist)』
- 367ページの『アプリケーション・ヒープ・サイズ (applheapsz)』
- 361ページの『アプリケーション制御ヒープ・サイズ (app_ctl_heap_sz)』
- 363ページの『分類ヒープ・サイズ (sortheap)』
- 366ページの『ステートメント・ヒープ・サイズ (stmtheap)』
- 376ページの『アプリケーション・サポート層ヒープ・サイズ (aslheapsz)』
- 351ページの『データベース・ヒープ (dbheap)』
- 382ページの『データベース・システム・モニター・ヒープ・サイズ (mon_heap_sz)』
- 368ページの『統計ヒープ・サイズ (stat_heap_sz)』

トランザクション・プロセッサー・モニター名 (tp_mon_name)

構成タイプ

データベース・マネージャー

適用範囲

- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー
- クライアント
- ローカル・クライアントをもつデータベース・サーバー
- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー
- ローカル・クライアントをもつサテライト・データベース・サーバー

パラメーター・タイプ

構成可能

デフォルト

デフォルトはなし

有効値

- CICS
- MQ
- ENCINA
- CB
- SF
- TUXEDO
- TOPEND

- ブランクまたはその他の値 (UNIX、OS/2、および Windows NT の場合。 Solaris または SINIX では、他の値を使用することはできません。)

このパラメーターは、使用するトランザクション処理 (TP) モニター製品の名前を識別します。

- アプリケーションが WebSphere エンタープライズ版 CICS 環境で稼働する場合、このパラメーターは 『CICS』 に設定する必要があります。
- アプリケーションが WebSphere エンタープライズ版 Encina 環境で稼働する場合、このパラメーターは 『ENCINA』 に設定する必要があります。
- アプリケーションが WebSphere エンタープライズ版 Component Broker 環境で稼働する場合、このパラメーターは 『CB』 に設定する必要があります。
- アプリケーションが IBM MQSeries 環境で稼働する場合、このパラメーターは 『MQ』 に設定する必要があります。
- アプリケーションが BEA Tuxedo 環境で稼働する場合、このパラメーターは 『TUXEDO』 に設定する必要があります。
- アプリケーションが IBM San Francisco 環境で稼働する場合、このパラメーターは 『SF』 に設定する必要があります。

IBM WebSphere EJB および **Microsoft Transaction Server** ユーザーは、このパラメーターの値を構成する必要はありません。

上記の製品のいずれも使用しない場合、このパラメーターは構成せずに、ブランクのままにしておかなければなりません。

OS/2 および Windows NT 環境での DB2 ユニバーサル・データベースの以前のバージョンでは、このパラメーターには、 XA Transaction Manager の関数 *ax_reg* および *ax_unreg* が含まれている DLL のパスと名前が入れられていました。この形式は引き続きサポートされています。このパラメーターの値が、前述の TP モニター名のどれとも一致しない場合、その値は *ax_reg* および *ax_unreg* 関数が含まれているライブラリー名と見なされます。これは、UNIX、OS/2、および Windows NT 環境で当てはまります。

TXSeries CICS および Encina ユーザーの場合: OS/2 および Windows NT でのこの製品の以前のバージョンでは、このパラメーターを 『libEncServer:C』 または 『libEncServer:E』 として構成しなければなりませんでした。これは引き続きサポートされていますが、もはや必要ではありません。パラメーターを 『CICS』 または 『ENCINA』 として構成するだけで十分です。

MQSeries ユーザーの場合: OS/2 および Windows NT でのこの製品の以前のバージョンでは、このパラメーターを 『mqmax』 として構成しなければなりませんでした。これは引き続きサポートされていますが、もはや必要ではありません。パラメーターを 『MQ』 として構成するだけで十分です。

Component Broker ユーザーの場合: OS/2 および Windows NT でのこの製品の以前のバージョンでは、このパラメーターを 『somtrx1i』 として構成しなければなりませんでした。これは引き続きサポートされていますが、もはや必要ではありません。パラメーターを 『CB』 として構成するだけで十分です。

San Francisco ユーザーの場合: OS/2 および Windows NT でのこの製品の以前のバージョンでは、このパラメーターを 『ibmsfDB2』 として構成しなければなりませんでした。これは引き続きサポートされていますが、もはや必要ではありません。パラメーターを 『SF』 として構成するだけで十分です。

このパラメーターに指定できるストリングの最大長は、19 文字です。

この情報は、DB2 ユニバーサル・データベースの XA OPEN ストリングで構成することもできます。複数のトランザクション処理モニターが 1 つの DB2 インスタンスを使用している場合は、この機能を使用する必要があります。 XA OPEN ストリングの使用に関する詳細については、 *管理の手引き: 計画* を参照してください。

マシン・ノード・タイプ (nodetype)

構成タイプ データベース・マネージャー

適用範囲

- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー
- クライアント
- ローカル・クライアントをもつデータベース・サーバー
- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー
- ローカル・クライアントをもつサテライト・データベース・サーバー

パラメーター・タイプ 情報

このパラメーターには、マシンにインストールした DB2 製品に関する情報およびその結果のデータベース・マネージャー構成タイプに関する情報があります。このパラメーターで戻される可能性のある値、およびそのノード・タイプに関連した製品は次のとおりです。

- **ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー** - DB2 サーバー製品。ローカルおよびリモートのデータベース・クライアントをサポートし、他のリモート・データベース・サーバーにアクセスできます。
- **クライアント** - データベース・クライアント。リモート・データベース・サーバーにアクセスする機能を持っています。

- **ローカル・クライアントをもつデータベース・サーバー** - DB2 リレーショナル・データベース管理システム。ローカルのデータベース・クライアントをサポートし、他のリモート・データベース・サーバーにアクセスする機能を持っています。
- **ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー** - DB2 サーバー製品。ローカルおよびリモートのデータベース・クライアントをサポートし、他のデータベース・サーバーにアクセスでき、かつ、区画並列処理が行えます。
- **ローカル・クライアントをもつサテライト・データベース・サーバー** - DB2 リレーショナル・データベース管理システム。ローカルのデータベース・クライアントをサポートし、他のリモート・データベース・サーバーにアクセスする機能を持っています。

これらの値と等価な数値と API 定数については、 *管理 API 解説書* を参照してください。

デフォルトのチャージバック会計 (dft_account_str)

構成タイプ データベース・マネージャー

適用範囲

- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー
- クライアント
- ローカル・クライアントをもつデータベース・サーバー
- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー
- ローカル・クライアントをもつサテライト・データベース・サーバー

パラメーター・タイプ 構成可能

デフォルト[範囲] Null [任意の有効なストリング]

アプリケーションが接続要求を出すたびに、 DB2 コネクトで生成した接頭部とユーザー提供の接尾部から成る会計 IDが、アプリケーション・リクエスターから DRDA アプリケーション・サーバーに送信されます。この会計情報には、システム管理者が、各ユーザー・アクセスとリソースの使用法を関連付けるための機構が含まれています。

注: このパラメーターは、DB2 コネクトだけに適用されます。

接尾部は、`sqlesact()` API を呼び出したアプリケーション・プログラムか、または DB2ACCOUNT 環境変数を設定したユーザーにより提供されます。接尾部が API によっても環境変数によっても提供されない場合は、 DB2 コネクトはこのパラメーターの値をデフォルトの接尾部の値として使用します。このパラメーターは、会計ストリング

を DB2 コネクトに転送する機能のない下位レベルのデータベース・クライアント (バージョン 2 以前のバージョン) の場合に特に効果的です。

推奨事項: 会計ストリングは以下の文字を使用して設定してください。
- 英字 (A ～ Z)
- 数値 (0 ～ 9)
- 下線 (_)

Java Development Kit 1.1 インストール・パス (jdk11_path)

構成タイプ データベース・マネージャー

適用範囲
- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー
- クライアント
- ローカル・クライアントをもつデータベース・サーバー
- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー
- ローカル・クライアントをもつサテライト・データベース・サーバー

パラメーター・タイプ 構成可能

デフォルト[範囲] Null [有効なパス]

関連パラメーター
- 386ページの『Java インタープリター・ヒープの最大サイズ (java_heap_sz)』

このパラメーターは、 Java Development Kit 1.1 をインストールするためのディレクトリーを指定します。 Java インタープリターが使用する CLASSPATH およびその他の環境変数は、このパラメーターの値から算出されます。

このパラメーターにはデフォルトがないので、 Java Development Kit をインストールするときは、このパラメーターに値を指定する必要があります。

連合データベース・システムのサポート (federated)

構成タイプ データベース・マネージャー

適用範囲
- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー

- ローカル・クライアントをもつデータベース・サーバー
- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー

パラメーター・タイプ 構成可能

デフォルト[範囲] No [Yes; No]

このパラメーターは、データ・ソース (DB2 ファミリーおよび Oracle など) によって管理されるデータの分散要求を、アプリケーションが実行依頼できるかどうかを指定します。

インスタンス管理

次のパラメーターは、データベース・マネージャー・インスタンスのセキュリティーおよび管理と関連があります。

- 『システム運用管理権限グループ名 (sysadm_group)』
- 490ページの『システム制御権限グループ名 (sysctrl_group)』
- 491ページの『システム保守権限グループ名 (sysmaint_group)』
- 492ページの『認証タイプ (authentication)』
- 493ページの『権限なしで許可されるカタログ作成 (catalog_noauth)』
- 494ページの『デフォルトのデータベース・パス (dftdbpath)』
- 495ページの『DB2START/DB2STOP に必要な LOGON (ss_logon)』
- 496ページの『全クライアントの承認 (trust_allclnts)』
- 497ページの『承認クライアントの認証 (trust_clntauth)』

システム運用管理権限グループ名 (sysadm_group)

構成タイプ データベース・マネージャー

適用範囲

- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー
- クライアント
- ローカル・クライアントをもつデータベース・サーバー
- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー
- ローカル・クライアントをもつサテライト・データベース・サーバー

パラメーター・タイプ 構成可能

デフォルト Null

関連パラメーター

- 490ページの『システム制御権限グループ名 (sysctrl_group)』
- 491ページの『システム保守権限グループ名 (sysmaint_group)』

システム運用管理 (SYSADM) 権限とは、データベース・マネージャーの権限では最高レベルの権限で、すべてのデータベース・オブジェクトを制御します。このパラメーターは、データベース・マネージャー・インスタンスのための SYSADM 権限を持つグループ名を定義します。

SYSADM 権限は、特定の操作環境で使用されているセキュリティー機能によって判別されます。

- Windows 95 または Windows 98 オペレーティング・システムでは、 SYSADM グループは NULL でなければならない。

  システム・セキュリティーを使用する場合は、 Windows 95 または Windows 98 クライアントではこのパラメーターは「NULL」である必要があります。これは、Windows 95 または Windows 98 オペレーティング・システムではグループ情報を保管しないため、ユーザーが指定された SYSADM グループのメンバーであるかどうかを判別する方法がないからです。グループ名が指定されている場合、どのユーザーもそのグループのメンバーになることはできません。
- Windows NT および Windows 2000 オペレーティング・システムでは、このパラメーターは 8 文字以下の名前を持つ任意のローカル・グループに設定することができ、 Windows NT および Windows 2000 セキュリティー・データベースに定義される。このパラメーターに「NULL」を指定する場合は、その管理グループのすべてのメンバーが SYSADM 権限を持ちます。
- UNIX ベース・システムでは、このパラメーターの値として「NULL」を指定したときは、 SYSADM グループはインスタンス所有者の 1 次グループでなければならない。

  値が「NULL」でないときは、 SYSADM グループは任意の有効な UNIX グループ名とすることができます。
- OS/2 では、このパラメーターに指定された値が「NULL」の場合は、ユーザー・プロファイル管理 (UPM) に管理者として定義されたユーザーは SYSADM 権限を持つ。

  このパラメーターにグループ名を指定した場合は、そのグループに属するユーザーだけが SYSADM 権限を持ちます。指定するグループは、どのユーザー・プロファイル管理 (UPM) グループであってもかまいません。

DCE セキュリティーを使用し、 *sysadm_group* を「NULL」とする場合は、デフォルトの DCE グループ名 DB2ADMIN が使用されます。権限 ID マッピングが DB2ADMIN である有効な DCE プリンシパルがすでに存在している必要があります。異なるグループ名を指定することもできます。

パラメーターをデフォルト (NULL) に復元するには、 UPDATE DBM CFG USING SYSADM_GROUP NULL を使用してください。このときのキーワード 『NULL』 は大文字で指定する必要があります。 DB2 コントロール・センターの「インスタンスの構成 (Configure Instance)」ノートブックを使用することもできます。

システム制御権限グループ名 (sysctrl_group)

構成タイプ データベース・マネージャー

適用範囲

- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー
- クライアント
- ローカル・クライアントをもつデータベース・サーバー
- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー
- ローカル・クライアントをもつサテライト・データベース・サーバー

パラメーター・タイプ 構成可能

デフォルト Null

関連パラメーター

- 488ページの『システム運用管理権限グループ名 (sysadm_group)』
- 491ページの『システム保守権限グループ名 (sysmaint_group)』

このパラメーターは、システム制御 (SYSCTRL) 権限を持つグループ名を定義します。 SYSCTRL には、システム・リソースに影響する操作を行える特権がありますが、データに直接アクセスすることはできません。

警告: システム・セキュリティーが使用される (すなわち、認証が CLIENT、SERVER、DCS、またはほかの有効な認証である) 場合、 Windows 95 および Windows 98 クライアントでは、このパラメーターは Null にする必要があります。これは、 Windows 95 および Windows 98 オペレーティング・システムはグループ情報を保管しないため、ユーザーが、指定された SYSCTRL グループのメンバーであるかどうかを判別する方法がないからです。グループ名が指定されている場合、どのユーザーもそのグループのメンバーになることはできません。これは DCE 認証が使用される場合は異なります。この場合は、グループ名を指定することができます。

パラメーターをデフォルト (NULL) に復元するには、 UPDATE DBM CFG USING SYSCTRL_GROUP NULL を使用してください。このときのキーワード 『NULL』 は

大文字で指定する必要があります。 DB2 コントロール・センターの「インスタンスの構成 (Configure Instance)」ノートブックを使用することもできます。

システム保守権限グループ名 (sysmaint_group)

構成タイプ データベース・マネージャー

適用範囲

- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー
- クライアント
- ローカル・クライアントをもつデータベース・サーバー
- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー
- ローカル・クライアントをもつサテライト・データベース・サーバー

パラメーター・タイプ 構成可能

デフォルト Null

関連パラメーター

- 488ページの『システム運用管理権限グループ名 (sysadm_group)』
- 490ページの『システム制御権限グループ名 (sysctrl_group)』

このパラメーターは、システム保守 (SYSMAINT) 権限を持つグループ名を定義します。 SYSMAINT には、あるインスタンスに関連したすべてのデータベースについて、データに直接アクセスしないで保守操作を実行する特権があります。

警告: システム・セキュリティーが使用される (すなわち、認証が CLIENT、SERVER、DCS、またはほかの有効な認証である) 場合、 Windows 95 および Windows 98 クライアントでは、このパラメーターは Null にする必要があります。これは、 Windows 95 および Windows 98 オペレーティング・システムはグループ情報を保管しないため、ユーザーが、指定された SYSMAINT グループのメンバーであるかどうかを判別する方法がないからです。グループ名が指定されている場合、どのユーザーもそのグループのメンバーになることはできません。これは DCE 認証が使用される場合は異なります。この場合は、グループ名を指定することができます。

パラメーターをデフォルト (NULL) に復元するには、 UPDATE DBM CFG USING SYSMAINT_GROUP NULL を使用してください。このときのキーワード 『NULL』 は大文字で指定する必要があります。 DB2 コントロール・センターの「インスタンスの構成 (Configure Instance)」ノートブックを使用することもできます。

認証タイプ (authentication)

構成タイプ データベース・マネージャー

適用範囲

- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー
- クライアント
- ローカル・クライアントをもつデータベース・サーバー
- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー
- ローカル・クライアントをもつサテライト・データベース・サーバー

パラメーター・タイプ 構成可能

デフォルト[範囲] SERVER [CLIENT; SERVER; SERVER_ENCRYPT; DCS; DCS_ENCRYPT; DCE; DCE_SERVER_ENCRYPT; KERBEROS; KRB_SERVER_ENCRYPT]

このパラメーターは、ユーザーの認証を行う場所と方法を決定します。

認証が SERVER であれば、ユーザー ID とパスワードがクライアントからサーバーに送られ、サーバーで認証が行えます。 SERVER_ENCRYPT 値の動作は、ネットワーク上で送信されたパスワードが暗号化されることを除き、 SERVER 値の動作と同じです。

CLIENT に設定すると、すべての認証はクライアントで行われるので、サーバーで認証を行う必要はなくなります。

値 DCS を設定すると、ホストまたは AS/400 システムで認証が行われます。 DCS_ENCRYPT 値の動作は、ネットワーク上で送信されたパスワードが暗号化されることを除き、 DCS 値の動作と同じです。 APPC およびクライアントのパスワードを DB2 サーバーに公開しない通信製品を使用する場合は、 DCS を使用すれば次のものを獲得することができます。

- SERVER タイプの認証 (DRDA 以外のクライアントの場合)
- CLIENT タイプの認証 (DRDA クライアントの場合)

DCE を設定した場合は、認証が DCE サーバーで DCE セキュリティー・サービスを使用して行われることを意味します。 DCE_SERVER_ENCRYPT 値の動作は、ネットワーク上で送信されたパスワードが暗号化されることを除き、 DCE 値の動作と同じです。

DCE_SERVER_ENCRYPT 値はサーバーでのみ使用できます。この値は、サーバーが DCE 認証または SERVER_ENCRYPT 認証のいずれかを受け入れ可能であることを示します。

値 KERBEROS を設定すると、認証用の Kerberos セキュリティー・プロトコルを使用して、認証が Kerberos サーバーで実行されます。 Kerberos セキュリティー・システムをサポートしているサーバーおよびクライアントで、認証タイプ KRB_SERVER_ENCRYPT を使用すると、有効なシステム認証タイプは KERBEROS になります。 Kerberos セキュリティー・システムをサポートしていないクライアントの場合、有効なシステム認証タイプは SERVER_ENCRYPT と同じになります。

注: Kerberos 認証タイプがサポートされるのは、 Windows 2000 を実行しているサーバーだけです。

パスワード暗号化をサポートしている認証は、 SERVER_ENCRYPT、DCS_ENCRYPT、DCE_SERVER_ENCRYPT、および KRB_SERVER_ENCRYPT です。これらの値の機能は、それぞれ、SERVER、DCS、DCE、および KERBEROS と認証位置の点で同じです。ただし、ソースでカタログされた認証タイプによって指定されるように、送信されるパスワードがソースで暗号化され、ターゲットでの暗号化解除を必要とする点が異なります。暗号化された値と暗号化されていない値で認証位置が一致するものを使用すれば、認証が行われる場所に影響を与えずに、クライアントとゲートウェイの間、またはゲートウェイとサーバーの間で、様々な暗号化の組み合わせを選択することができます。

これらの値と等価な数値と API 定数については、 *管理 API 解説書* を参照してください。

DCE または DCS を使用する場合とその理由、および連合データベースに関連した認証の問題について詳しくは、 *管理の手引き: インプリメンテーション* にある『データベース・アクセスの制御』の章を参照してください。

推奨事項: 通常は、デフォルト (SERVER) が適切です。 Kerberos、DB2 コネクト、または DCE で処理される着信要求があるときは、 *管理の手引き: インプリメンテーション* にある『データベース・アクセスの制御』の章を参照してください。

権限なしで許可されるカタログ作成 (catalog_noauth)

構成タイプ データベース・マネージャー

適用範囲

- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー
- クライアント
- ローカル・クライアントをもつデータベース・サーバー

- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー
- ローカル・クライアントをもつサテライト・データベース・サーバー

パラメーター・タイプ 構成可能

デフォルト[範囲]

ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー; ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー

NO [NO (0) ～ YES (1)]

クライアント; ローカル・クライアントをもつデータベース・サーバー; ローカル・クライアントをもつサテライト・データベース・サーバー

YES [NO (0) ～ YES (1)]

このパラメーターは、ユーザーが SYSADM 権限なしでデータベースおよびノード、または DCS および ODBC ディレクトリーのカタログを作成したりアンカタログ化したりできるかどうかを指定します。このパラメーターのデフォルト値 (0) は、SYSADM 権限が必須であることを示します。このパラメーターが 1 (yes) に設定されると、SYSADM 権限は必要ありません。

デフォルトのデータベース・パス (dftdbpath)

構成タイプ データベース・マネージャー

適用範囲

- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー
- ローカル・クライアントをもつデータベース・サーバー
- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー
- ローカル・クライアントをもつサテライト・データベース・サーバー

パラメーター・タイプ 構成可能

デフォルト[範囲]

UNIX インスタンス所有者のホーム・ディレクトリー [任意の既存パス]

OS/2 および Windows NT

DB2 がインストールされているドライブ [任意の既存のパス]

このパラメーターは、データベース・マネージャーでデータベースを作成するときのデフォルト・ファイル・パスを示します。データベースの作成時にパスが指定されていないと、データベースは、*dftdbpath* パラメーターで指定されたパスに作成されます。

区分データベース環境では、データベースを作成するパスが NFS マウントのパスではないこと (UNIX ベースのプラットフォームの場合) またはネットワーク・ドライブであること (Windows NT 環境の場合) を確認する必要があります。指定されるパスは、物理的に、それぞれのデータベース区画サーバーに存在する必要があります。混乱を避けるために、各データベース区画サーバーにローカルにマウントされているパスを指定するのが最良です。このパスの名前の最大長は 205 文字です。システムは、このパスの終わりにノード名を付加します。

データベースのサイズが大きくなっていく場合、および多数のユーザーが (現在の環境と目的に応じて) データベースを作成できる状況では、すべてのデータベースを指定位置で作成して保管することが効果的な場合があります。また、データベースの保全性やバックアップとリカバリーの容易さを考慮すると、データベースを他のアプリケーションやデータから分離しておくのはよいことです。

UNIX ベースの環境では、 *dftdbpath* 名の長さは 215 文字以下でなければならず、有効な絶対パス名を指定しなければなりません。 OS/2 および Windows NT の場合は、*dftdbpath* はドライブ文字とすることができ、任意指定でコロンを続けることができます。

推奨事項: できれば、高頻度使用のデータベースは、オペレーティング・システム・ファイルやデータベース・ログなどの頻繁にアクセスされるデータとは別のディスクに入れてください。

DB2START/DB2STOP に必要な LOGON (ss_logon)

構成タイプ データベース・マネージャー

適用範囲

- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー
- ローカル・クライアントをもつデータベース・サーバー

パラメーター・タイプ 構成可能

デフォルト[範囲] YES [NO (0)、YES (1)]

このパラメーターは、OS/2 環境にだけ適用されます。このパラメーターのデフォルトを使用する場合は、 DB2START または DB2STOP を出す前に LOGON ユーザー ID とパラメーターが必要です。

全クライアントの承認 (trust_allclnts)

構成タイプ データベース・マネージャー

適用範囲

- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー
- ローカル・クライアントをもつデータベース・サーバー
- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー

パラメーター・タイプ 構成可能

デフォルト[範囲] YES [NO、YES、DRDAONLY]

関連パラメーター

- 492ページの『認証タイプ (authentication)』
- 497ページの『承認クライアントの認証 (trust_clntauth)』

このパラメーターは、 *authentication* パラメーターが CLIENT に設定されているときだけ活動状態になります。

このパラメーターと *trust_clntauth* は、データベース環境に対するユーザーの妥当性を検査する場所を決定するために使用されます。

このパラメーターにデフォルトの 『YES』 を使用すると、すべてのクライアントが承認クライアントとして扱われます。すなわち、サーバーは、クライアントでセキュリティー・レベルを使用でき、クライアントでユーザーの妥当性検査を行えると見なします。

このパラメーターは、 *authentication* パラメーターが CLIENT に設定されているときだけ 『NO』 にすることができます。このパラメーターを 『NO』 に設定すると、承認されていないクライアントは、サーバーに接続するときに、ユーザー ID とパスワードの組み合わせを提供しなければならなくなります。承認されていないクライアントは、ユーザーを認証するためのセキュリティー・サブシステムを持っていないオペレーティング・システム・プラットフォームです。

このパラメーターを 『DRDAONLY』 に設定すると、 DB2 (MVS 版)、DB2 (OS/390 版)、DB2 (VM および VSE 版)、および DB2 (OS/400 版) から、 DRDA クライアントを除くすべてのクライアントが保護されます。上記のクライアントだけを、クライアント側の認証を行うよう承認することができます。他のすべてのクライアントには、サーバーによって認証されているユーザー ID とパスワードが必要です。

trust_allclnts を 『DRDAONLY』 に設定すると、クライアントの認証場所を判別するのに *trust_clntauth* パラメーターが使用されます。 *trust_clntauth* を 『CLIENT』 に設定すると、認証はクライアントで行われます。 *trust_clntauth* を 『SERVER』 に設定すると、認証は、クライアント (パスワードが指定されなかった場合) およびサーバー (パスワードが指定された場合) で行われます。

承認クライアントについての詳細は、 *管理の手引き: インプリメンテーション* の『サーバーに対する認証方式の選択』を参照してください。

承認クライアントの認証 (trust_clntauth)

構成タイプ データベース・マネージャー

適用範囲

- ローカル・クライアントおよびリモート・クライアントをもつデータベース・サーバー
- ローカル・クライアントをもつデータベース・サーバー
- ローカル・クライアントおよびリモート・クライアントをもつ区分データベース・サーバー

パラメーター・タイプ 構成可能

デフォルト[範囲] CLIENT [CLIENT、SERVER]

関連パラメーター

- 492ページの『認証タイプ (authentication)』
- 496ページの『全クライアントの承認 (trust_allclnts)』

このパラメーターは、承認クライアントがサーバーで認証されるのか、または、クライアントが接続のためにユーザー ID とパスワードの組み合わせを提供したときにクライアントで認証されるのかを指定します。このパラメーター (および *trust_allclnts*) は、*authentication* パラメーターが CLIENT に設定されている場合にだけ活動状態になります。ユーザー ID とパラメーターを提供しないと、クライアントがユーザーを検査したと見なされ、それ以降の妥当性検査はサーバーでは行われません。

このパラメーターを CLIENT (デフォルト) に設定すると、承認クライアントはユーザー ID とパスワードの組み合わせを提供しなくても接続することができ、オペレーティング・システムがすでにこのユーザーを認証しているものとされます。 SERVER に設定すると、ユーザー ID とパスワードがサーバーで検査されます。

CLIENT を指定する数値は 0 です。SERVER を指定する数値は 1 です。

承認クライアントについての詳細は、 *管理の手引き: インプリメンテーション* の『サーバーに対する認証方式の選択』を参照してください。

第4部 付録

付録A. DB2 レジストリー変数と環境変数

以下に、始動と実行を行うときに知っている必要のある DB2 レジストリー変数と環境変数をリストします。それぞれに簡単な説明を付けてありますが、使用する環境には適用されないものもあります。

以下のようにして、サポートされているすべてのレジストリー変数のリストを表示できます。

```
db2set -lr
```

以下のようにして、現行またはデフォルト・インスタンスの変数の値を変更できます。

```
db2set registry_variable_name=new_value
```

環境変数を更新するには、set コマンドを使用してから、システムをリブートしなければなりません。

変更されたレジストリー変数の値は、 DB2START コマンドの発行前に設定されていなければなりません。レジストリー変数の変更と使用についての詳細は、*管理の手引き: インプリメンテーション* を参照してください。

バイナリー・レジストリー変数の説明で使用される値には、同等の値を持つものがあります。 YES、1、および ON はすべて同等です。同じように、NO、0、および OFF もすべて同等です。それぞれの値は同じ意味として使用できます。

表 21. 一般レジストリー変数

変数名 説明	オペレーティング・システム	値
DB2ACCOUNT	全	デフォルト = null
リモート・ホストに送信される会計ストリング。詳細は、*DB2 コネクト 使用者の手引き* を参照。		
DB2BIDI	全	デフォルト = NO 値: YES または NO
この変数は双方向サポートを可能にする。 DB2CODEPAGE 変数は、使用するコード・ページを宣言する場合に使用する。双方向サポートについての追加詳細は、 *管理の手引き: 計画* の付録『各国語サポート』を参照。		
DB2_BLOCK_ON_LOG_DISK_FULL	ALL	デフォルト = No 値: Yes または No

表 21. 一般レジストリー変数 (続き)

変数名	オペレーティング・システム	値
説明		
この DB2 レジストリー変数の設定によって、活動状態のログ・パスに DB2 が新しいログ・ファイルを作成できないときにディスク・フル・エラーが生成されるのを防ぐことができる。 代わりに、DB2 は成功するまで 5 分ごとにログ・ファイルを作成しようとする。 DB2 は作成を試みるたびにメッセージを db2diag.log ファイルに書き込む。ログ・ディスクがいっぱいの状態であるためアプリケーションがハングしていることを確認する唯一の方法は、 db2diag.log ファイルをモニターすることである。 ログ・ファイルが正常に作成されるまで、表データを更新しようとするユーザー・アプリケーションはトランザクションをコミットできない。読み取り専用の照会が直接影響を受けることはないが、更新要求によりロックされているデータへのアクセス、または、更新アプリケーションによりバッファー・プールに固定されているデータ・ページへのアクセスが照会で必要になった場合には、読み取り専用照会もハングしているように見える。		
DB2CODEPAGE	全	デフォルト: オペレーティング・システムの指定どおりに言語 ID から得られる。
データベース・クライアント・アプリケーションのために DB2 に提示されるデータのコード・ページを指定する。設定するように DB2 文書に明確に記述されているか、または DB2 サービスで求められない限り、 DB2CODEPAGE は設定しないこと。 DB2CODEPAGE にオペレーティング・システムでサポートされていない値を設定すると、その結果は予測できなくなる。 通常、DB2 は自動的にオペレーティング・システムからコード・ページ情報を得るので、 DB2CODEPAGE を設定する必要はない。		
DB2COUNTRY	全	デフォルト: オペレーティング・システムの指定どおりに言語 ID から得られる。
クライアント・アプリケーションの国別コード、領域コード、およびテリトリー・コードを指定する。これは、日付と時刻の形式に影響する。		
DB2DBDFT	全	デフォルト = null
暗黙接続に使用するデータベースのデータベース別名を指定する。アプリケーションがデータベースに接続していないが SQL ステートメントが発行されている場合、デフォルト・データベースで DB2DBDFT 環境変数が定義されていれば、暗黙接続が行われる。		
DB2DBMSADDR	Windows 32 ビット・オペレーティング・システム	デフォルト = 0x20000000 (Windows NT の場合) 0x90000000 (Windows 95 の場合) 値: 0x20000000 ～ 0xB0000000 (増分は 0x10000)

表 21. 一般レジストリー変数 (続き)

変数名	オペレーティング・システム	値
説明		
デフォルトのデータベース・マネージャー共用メモリーのアドレスを 16 進形式で指定する。共用メモリーのアドレス衝突のために *db2start* が失敗する場合、このレジストリー変数は強制的にデータベース・マネージャー・インスタンスに変更され、別のアドレスのその共用メモリーに割り当てられる。		
DB2_DISABLE_FLUSH_LOG	全	デフォルト = OFF 値: ON または OFF
オンライン・バックアップの完了時にアクティブ・ログ・ファイルがクローズされないようにするかどうか指定する。 オンライン・バックアップが完了した時点で、最後のアクティブ・ログ・ファイルは切り捨てられ、クローズされて、アーカイブが可能になる。これにより、オンライン・バックアップは、リカバリーに使用可能なアーカイブ・ログの完全セットとなる。 ログ順序番号 (LSN) スペースの部分が無駄に使用されることを懸念している場合は、最後のアクティブ・ログがクローズされないようにすることができる。アクティブ・ログ・ファイルが切り捨てられるたびに、 LSN は切り捨てられたスペースと比例した量だけ増加する。毎日膨大な数のオンライン・バックアップが実行される場合、最後のアクティブ・ログ・ファイルがクローズされないようにすることができる。 オンライン・バックアップが完了したすぐあとで、ログの完全なメッセージを受け取っていることに気付く場合、最後のアクティブ・ログ・ファイルがクローズされないようにすることができる。ログ・ファイルが切り捨てられると、予約済みアクティブ・ログ・スペースが、切り捨てられたログのサイズと比例した量だけ増加する。切り捨てられたログ・ファイルが再利用されると、アクティブ・ログ・スペースは解放される。再利用は、ログ・ファイルが非活動状態になったすぐあとに行われる。この処理は、ログの完全なメッセージを受け取る間の短い時間に行われる。		
DB2DISCOVERYTIME	OS/2 および Windows 32 ビット・オペレーティング・システム	デフォルト = 40 秒。 最低 = 20 秒
SEARCH ディスカバリーが DB2 システムを探索する時間を指定する。		
DB2INCLUDE	全	デフォルト = 現行ディレクトリー
DB2 PREP 処理において、 SQL INCLUDE テキスト・ファイル・ステートメントの処理時に使用されるパスを指定する。これによって、INCLUDE ファイルが検出されるディレクトリーのリストが提供される。いろいろなプリコンパイル済み言語において DB2INCLUDE が使用される方法については、 *アプリケーション開発の手引き* を参照。		

表 21. 一般レジストリー変数 (続き)

変数名 説明	オペレーティング・システム	値
DB2INSTDEF	OS/2 および Windows 32 ビット・オペレーティング・システム	デフォルト = DB2
DB2INSTANCE が定義されていない場合に使用される値を設定する。		
DB2INSTOWNER	Windows NT	デフォルト = null
インスタンスの初回作成時に DB2 プロファイル登録で作成されるレジストリー変数。この値は、インスタンスを所有するマシンの名前に合わせて設定される。		
DB2_LIC_STAT_SIZE	全	デフォルト = null 範囲: 0 ～ 32 767
このレジストリー変数は、システムのライセンス統計が入っているファイルの最大サイズ (MB 単位) を判別する。値がゼロの場合、ライセンス統計の収集はオフになる。認識または定義されていない場合、この変数はデフォルト (無制限) に設定される。統計は、ライセンス・センターを使って表示する。		
DB2NBDISCOVERRCVBUFS	全	デフォルト = 16 バッファー 最小値 = 16 バッファー
この変数は NetBIOS 検索ディスカバリーで使用される。変数は、クライアントが同時に受信可能なディスカバリー応答数を指定する。クライアントがこの値で指定した数よりも多くの応答を同時に受け取ると、超過した応答は NetBIOS 層によって廃棄される。デフォルトは、16 の NetBIOS 受信バッファー。デフォルト値よりも小さい値を選択すると、デフォルトが使用される。		
DB2OPTIONS	全 (Windows 3.1 および Macintosh を除く)	デフォルト = null
コマンド行プロセッサー・オプションを設定する。		
DB2SLOGON	Windows 3.x	デフォルト = null 値: YES または NO
DB2 (Windows 3.x 版) での保護ログオンを使用可能にする。 DB2SLOGON=YES ならば、 DB2 はユーザー ID とパスワードをファイルに書かず、代わりにそれらを保守するためにメモリーのセグメントを使用する。 DB2SLOGON が使用可能になっていると、ユーザーは Windows 3.x を開始するたびにログオンする必要がある。		
DB2TIMEOUT	Windows 3.x および Macintosh	デフォルト = (設定しない)

表 21. 一般レジストリー変数 (続き)

変数名	オペレーティング・システム	値
説明		
Windows 3.x および Macintosh のクライアントの場合、長い SQL 照会時のタイムアウト時間を制御するために使用される。このタイムアウト時間が経過すると、ダイアログ・ボックスがポップアップして、その照会を中断するか続行するか尋ねる。この変数のための最小値は 30 秒である。 DB2TIMEOUT の値を 1 から 30 の間の数に設定すると、デフォルトの最小値が使用される。 DB2TIMEOUT の値を 0 または負数に設定すると、タイムアウト機能は使用不能になる。この機能は、デフォルトでは使用不能になっている。		
DB2TRACENAME	Windows 3.x および Macintosh	デフォルト = DB2WIN.TRC (Windows 3.x の場合)、 DB2MAC.TRC (Macintosh 場合)
Windows 3.x および Macintosh では、トレース情報が保管されているファイルの名前を指定する。各システムのデフォルトは、現行のインスタンス・ディレクトリー (たとえば ¥sqllib¥db2) に保管されている。トレース・ファイルを指名するときは、全パス名を指定することを強くお勧めする。		
DB2TRACEON	Windows 3.x および Macintosh	デフォルト = NO 値: YES または NO
Windows 3.x および Macintosh では、問題が発生したときに IBM に情報を提供するためにトレースをオンにしておく。 (自分で解決できない問題に遭遇することがないのであれば、トレースをオンにしないようにお勧めする。) クライアントでトレース機能を使用する場合の詳細は、 *問題判別の手引き* を参照。		
DB2TRCFLUSH	Windows 3.x および Macintosh	デフォルト = NO 値: YES または NO
Windows 3.x および Macintosh の場合、 DB2TRACEFLUSH を DB2TRACEON=YES と一緒に使用することができる。 DB2TRACEFLUSH=YES を指定すると、各トレース・レコードが即座にトレース・ファイルに書き込まれる。この指定により、DB2 システムはかなりスローダウンするので、デフォルトの設定は DB2TRACEFLUSH=NO である。これを設定しておくと、アプリケーションによってシステムがハングし、システムをリブートする必要が起きたときに役に立つ。このキーワードの設定によって、トレース・ファイルとトレース項目がリブート時に失われないことが保証される。		
DB2TRCSYSERR	Windows 3.x および Macintosh	デフォルト = 1 値: 1 ～ 32 767
システム・エラーがいくつになるとクライアントはトレースをオフにするか、その数を指定する。デフォルト値のシステム・エラー・トレース数は 1 で、この後、トレースはオフになる。		
DB2YIELD	Windows 3.x	デフォルト = NO 値: YES または NO

表 21. 一般レジストリー変数 (続き)

変数名	オペレーティング・システム	値
説明		
リモート・サーバーと通信している Windows 3.x クライアントの動作を指定する。 NO に設定すると、クライアントは CPU を他の Windows 3.x アプリケーションに譲らず、そのクライアント・アプリケーションがリモート・サーバーと通信している間は Windows 環境は一時停止となる。他のタスクを再開するには、その通信操作が完了するのを待たなければならない。 YES に設定すると、システム機能は通常どおりである。アプリケーションは、 DB2YIELD=YES で実行するように推奨する。システムが破損される場合は、 DB2YIELD=NO に設定する必要がある。アプリケーション開発を行う場合は、通信操作が完了するのを待機している間に、アプリケーションが Windows メッセージを受け入れ、処理できるようにアプリケーションを作成する必要がある。		

表 22. システム環境変数

変数名	オペレーティング・システム	値
説明		
DB2CONNECT_IN_APP_PROCESS	全	デフォルト = YES 値: YES または NO
この変数を NO に設定すると、 DB2 コネクト エンタープライズ・エディション・マシン上のローカル DB2 コネクト・クライアントはエージェント内で強制的に実行される。エージェント内で実行する利点としては、ローカル・クライアントをモニターできることと、ローカル・クライアントが SYSPLEX サポートを使用できることがある。		
DB2DOMAINLIST	Windows NT サーバーのみ	デフォルト = null 値: コンマ (『,』) 区切りの Windows NT ドメイン・ネームのリスト
1 つ以上の Windows NT ドメインを定義する。このドメインに属しているユーザーの接続要求のみが受け入れられる。 このレジストリー変数は、 DB2 ユニバーサル・データベース・バージョン 7.1 (またはそれ以降) が稼働する DB2 サーバーおよびクライアントの純粋な Windows NT ドメイン環境下でのみ使用する。		
DB2ENVLIST	UNIX	デフォルト: null

表 22. システム環境変数 (続き)

変数名	オペレーティング・システム	値
説明		
ストアード・プロシージャーまたはユーザー定義関数の特定の変数名をリストする。デフォルトでは、**db2start** コマンドは、接頭部が **DB2** または **db2** になっているユーザー環境変数を除いて、すべてのユーザー環境変数をフィルターに掛けて除去する。特定のレジストリー変数をストアード・プロシージャーまたはユーザー定義関数のどちらかに渡さなければならない場合、 DB2ENVLIST レジストリー変数にその変数名をリストできる。 その場合、各変数を 1 つまたは複数のスペースで区切る。 DB2 は、それ自体の PATH と LIBPATH を構成する。したがって、PATH または LIBPATH が DB2ENVLIST に指定されている場合、変数名の実際の値は、DB2 構成値の最後に付加される。		
DB2INSTANCE	全	デフォルト = DB2INSTDEF (OS/2 および Windows 32 ビット・オペレーティング・システムの場合)
デフォルト解釈により活動状態になるインスタンスを指定するために使用される環境変数。 UNIX では、ユーザーは DB2INSTANCE に値を指定する必要がある。		
DB2INSTPROF	OS/2、Windows 3.x、および Windows 32 ビット・オペレーティング・システム	デフォルト: null
OS/2、Windows 3.x および Windows 32 ビットのオペレーティング・システムにおいて、インスタンス・ディレクトリーが DB2PATH 以外の場所にある場合のその位置を示すために使用される環境変数。		
DB2LIBPATH	UNIX	デフォルト: null
DB2LIBPATH レジストリー変数に LIBPATH の値を指定する。 ユーザー ID が変更された場合、LIBPATH の値は親プロセスと子プロセスとの間で継承することはできない。 **db2start** 実行可能はルートに置かれるので、 DB2 はエンド・ユーザーの LIBPATH 設定を継承することができない。DB2ENVLIST レジストリー変数に変数名、LIBPATH をリストする場合、 LIBPATH のその値を DB2LIBPATH レジストリー変数にも指定しなければならない。 **db2start** 実行可能は、次に DB2LIBPATH の値を読み取り、その値を DB2 構成 LIBPATH の最後に付加する。		
DB2NODE	全	デフォルト: null 値: 1 ～ 999
接続する DB2 エンタープライズ拡張エディション・データベース区画サーバーのターゲット論理ノードを指定する。この変数を指定しない場合、ターゲット論理ノードはデフォルトとして、マシン上のポート 0 に定義された論理ノードに設定される。		

表 22. システム環境変数 (続き)

変数名 説明	オペレーティング・システム	値
DB2_PARALLEL_IO	全	デフォルト: null 値: * (すべての表スペース) または定義された複数の表スペースのコンマ区切りリスト
データベース内に複数のコンテナーがある場合、表スペース・コンテナーからの読み取り、または表スペース・コンテナーへの書き込み中に、 DB2 は並列入出力を使用する場合がある。 1 つのコンテナーに対して強制的に並列入出力を行う場合は、このレジストリー変数を使用する。変更を有効にするには、このレジストリー変数を設定してから、 DB2STOP を発行し、DB2START を入力する。		
DB2PATH	OS/2、Windows 3.x、および Windows 32 ビット・オペレーティング・システム	デフォルト: (オペレーティング・システムによって異なる)
OS/2、Windows 3.x および Windows 32 ビットのオペレーティング・システムにおいて、この製品がインストールされるディレクトリーを指定するために使用される環境変数。		
DB2_STRIPED_CONTAINERS	全	デフォルト: null 値: ON、null
表スペース・コンテナーとして RAID 装置を使用する場合、エクステント・サイズを RAID ストライプ・サイズと同じかまたはその倍数にして、表スペースを作成するように推奨されている。ただし、コンテナー・タグは 1 ページなので、エクステントで RAID ストライプが並べられることはない。そのため、入出力要求中に、最適な数よりも多くの物理ディスクにアクセスしなければならなくなる場合がある。 DMS 表スペース・コンテナーを使用する場合、タグにそれ自身の (完全) エクステントを割り当てることにより、この問題を避けることができる。これにより、前述の問題は避けられるが、コンテナー内で余分なエクステントのオーバーヘッドが必要となる。 変更を有効にするには、このレジストリー変数を設定してから、 DB2STOP を発行し、DB2START を入力する。		

表 23. 通信変数

変数名 説明	オペレーティング・システム	値
DB2CHECKCLIENTINTERVAL	AIX、ただしサーバーのみ	デフォルト = 0 値: ゼロより大きい数値。
APPC クライアント接続の状況を検査するのに使用する。クライアント終了を照会完了まで待つのではなく早期に検出できるようにする。ゼロに設定すると、検査は行われない。ゼロより大きい数値に設定した場合、その値は DB2 の内部作業単位を表す。次の検査頻度を示す値が基準として用意されている。低い頻度の場合 300、中程度の頻度の場合 100、高い頻度の場合 50 を使用する。クライアント状況の検査の頻度を多くし、しかもデータベース要求を実行していると、照会が完了するまでの時間が長くなる。 DB2 の作業負荷が大きい (つまり内部要求が多い) 場合に、 DB2CHECKCLIENTINTERVAL を小さい値に設定すると、作業負荷が少ない状態よりパフォーマンス上の影響が大きくなり、大半の時間は DB2 の待ち時間になる。		
DB2COMM	全、ただしサーバーのみ	デフォルト = null 値: APPC、 IPXSPX、 NETBIOS、 NPIPE、 TCPIP の任意の組み合わせ
データベース・マネージャーを開始したときに開始されるコミュニケーション・マネージャーを指定する。これを指定しないと、サーバーではどの DB2 コミュニケーション・マネージャーも開始されない。		
DB2_FORCE_NLS_CACHE	AIX、 HP_UX、 Solaris	デフォルト = 偽 値: TRUE または FALSE
マルチスレッド・アプリケーションにおいてロック競合が起きないようにするために使用する。レジストリー変数が 『真』 の場合、スレッドが初めてコード・ページと国別コードの情報にアクセスする際にそれらの情報が保管される。その時点以降、この情報を要求する他のスレッドにこのキャッシュ情報が使用される。したがってロック競合は除かれ、特定の状態下でパフォーマンスが向上することになる。アプリケーションにより接続間のロケール設定が変更される場合、この設定値は使用できない。アプリケーションでロケール設定を変更するのは 『スレッド・セーフ』 ではないので普通はこの変更は行われない。したがってこのような状態は考慮する必要がないと思われる。		
DB2NBADAPTERS	OS/2 および Windows NT	デフォルト = 0 範囲: 0 〜 15。 複数の値を指定するときは、コンマで区切る
DB2 NetBIOS 通信に使用するローカル・アダプターを指定するために使用される。各ローカル・アダプターをその論理アダプター番号で指定する。		

表 23. 通信変数 (続き)

変数名 説明	オペレーティング・システム	値
DB2NBCHECKUPTIME	OS/2 および Windows NT、ただしサーバーのみ	デフォルト = 1 分 値: 1 〜 720
NetBIOS プロトコル・チェックアップ手順の各呼び出しの間の時間間隔を指定する。チェックアップ時間は分の単位で指定する。 より小さい値を指定すると、 NetBIOS プロトコル・チェックアップはより頻繁に実行され、予期しない時点でエージェント / セッションが終了すると、残されているメモリーやその他のシステム・リソースは解放される。		
DB2NBINTRLISTENS	OS/2 および Windows NT、ただしサーバーのみ	デフォルト = 1 値: 1 〜 10 複数の値を指定するときは、コンマで区切る
リモート・クライアントの割り込みに備えて非同期に出される NetBIOS 送信 listen コマンド (NCB) の数を指定する。この融通性は「割り込みが活動化されている」環境のために準備されており、このおかげでサーバーが他のリモート割り込みを扱っているときに、リモート・クライアントからの割り込み呼び出しが接続を設定できる。 DB2NBINTRLISTENS をより小さい値に設定すると、 NetBIOS セッションと NCB がサーバーで節約される。しかし、クライアント割り込みが多い環境では、 DB2NBINTRLISTENS をより大きな値に設定し、クライアントの割り込みに応答できるようにする必要がある。 **注:** この値は、それが指定されている位置に意味がある。すなわち、値は、DB2NBADAPTERS のそれぞれの対応位置の値に関連する。		
DB2NBRECVBUFFSIZE	OS/2 および Windows NT、ただしサーバーのみ	デフォルト = 4096 バイト 範囲: 4096 〜 65536
DB2 NetBIOS プロトコル受信バッファーのサイズを指定する。これらのバッファーは、NetBIOS 受信 NCB に割り当てられる。より小さい値にするとサーバー・メモリーが節約され、一方、クライアント・データの転送が大きいときはより大きい値が必要になる。		
DB2NBBRECVNCBS	OS/2 および Windows NT、ただしサーバーのみ	デフォルト = 10 範囲: 1 〜 99

表 23. 通信変数 (続き)

変数名	オペレーティング・システム	値
説明		
操作中にサーバーが発行し、保守する NetBIOS ″receive_any″ コマンド (NCB) 数を指定する。この値は、サーバーが接続されているリモート・クライアント数に応じて調整できる。より小さい値にすると、サーバー・リソースが節約される。 **注:** DB2NBBRECVNCBS によって、使用中の各アダプターそれぞれに固有の受信 NCB 値を指定することができる。すなわち、値が指定されている位置が意味を持ち、各値は、DB2NBADAPTERS のそれぞれの対応位置の値に関連する。		
DB2NBRESOURCES	OS/2 および Windows NT、ただしサーバーのみ	デフォルト = null
複数コンテキスト環境で DB2 使用のために割り振る NetBIOS リソース数を指定する。この変数は複数コンテキスト・クライアント操作の場合に限られる。		
DB2NBSENDNCBS	OS/2 および Windows NT、ただしサーバーのみ	デフォルト = 6 範囲: 1 〜 720
サーバーが使用するために予約する送信 NetBIOS コマンド (NCB) 数を指定する。この値は、サーバーが接続されているリモート・クライアント数に応じて調整できる。 DB2NBSENDNCBS をより小さい値に設定すると、サーバー・リソースが節約される。 ただし、他のすべての送信コマンドが使用中になっているときにサーバーがリモート・クライアントへの送信を待機しないようにするには、より大きい値を設定する必要がある。		
DB2NBSESSIONS	OS/2 および Windows NT、ただしサーバーのみ	デフォルト = null 範囲: 5 〜 254
DB2 が DB2 使用のために予約しておくように要求するセッション数を指定する。 DB2NBSESSIONS の値は、 DB2NBADAPTERS で指定される各アダプターに固有のセッションを要求するように設定できる。 **注:** この値は、それが指定されている位置に意味がある。すなわち、値は、DB2NBADAPTERS のそれぞれの対応位置の値に関連する。		
DB2NBXTRANCBS	OS/2 および Windows NT、ただしサーバーのみ	デフォルト = アダプター当たり 5 範囲: 5 〜 254
db2start コマンドが出されたときにサーバーが予約する必要のある「余分の」NetBIOS コマンド (NCB) 数を指定する。 DB2NBXTRANCBS の値は、 DB2NBADAPTERS で指定される各アダプターに固有のセッションを要求するように設定できる。		
DB2NETREQ	Windows 3.x	デフォルト = 3 範囲: 0〜25

表 23. 通信変数 (続き)

変数名	オペレーティング・システム	値
説明		
Windows 3.x クライアントで並行して実行できる NetBIOS 要求数を指定する。この値を大きく指定すればするほど、 1MB レベルより下のメモリーが多く使用される。 NetBIOS サービスを使用する並行要求数がここで設定された値になると、それ以降の NetBIOS サービスに対する着信要求はキューに保持され、現行要求が完了してから活動状態になる。 DB2NETREQ に 0 (ゼロ) を入力すると、 Windows データベース・クライアントは NetBIOS 待機オプションを使用して同期モードで NetBIOS 呼び出しを出す。 このモードでは、データベース・クライアントは、現行の NetBIOS 要求を活動状態にすることができるだけで、この現行要求が完了するまでは他の要求を処理しない。これは、他のアプリケーション・プログラムに影響する場合がある。値 0 は、古いリリースとの互換性のためにだけサポートされている。 0 は使用しないように強く推奨する。		
DB2RETRY	OS/2 および Windows NT	デフォルト = 0 範囲: 0〜20 000
DB2 で APPC listener の再始動が試行される回数。サーバー / ゲートウェイで SNA サブシステムがダウンした場合、このプロファイル変数と DB2RETRYTIME を一緒に使用すると、 APPC listener を自動的に再始動できる。その際他のプロトコルを使用しているクライアント通信は不通にならない。 このシナリオの場合、 DB2 を停止してから再始動して APPC クライアント通信を再び開始する必要はなくなる。		
DB2RETRYTIME	OS/2 および Windows NT	デフォルト = 1 分 範囲: 0〜7 200 分
DB2 で APPC listener を開始するための連続再試行が行われる間隔を示す分数。増分は 1 分。サーバー / ゲートウェイで SNA サブシステムがダウンした場合、このプロファイル変数と DB2RETRY を一緒に使用すると、 APPC listener を自動的に再始動できる。その際他のプロトコルを使用しているクライアント通信は不通にならない。 このシナリオの場合、 DB2 を停止してから再始動して APPC クライアント通信を再び開始する必要はなくなる。		
DB2SERVICETPINSTANCE	OS/2、 Windows NT、 AIX、および Sun Solaris	デフォルト = null

表 23. 通信変数 (続き)

変数名	オペレーティング・システム	値
説明		
以下が原因で生じる問題を解決するために使用される。 • 同じマシンで複数のインスタンスを実行している。 • 同じマシンでバージョン 6 またはバージョン 7 のインスタンスを実行しており、同じ TP 名を登録しようとしている。 **db2start** コマンドが呼び出されると、指定されたインスタンスが以下の TP 名の APPC listener を開始する。 • DB2DRDA • x'07'6DB		
DB2SOSNDBUF	Windows 95 および Windows NT	デフォルト = 32767
Windows 95 および Windows NT オペレーティング・システムでの TCP/IP 送信バッファーの値を指定する。		
DB2SYSPLEX_SERVER	OS/2、 Windows NT、および UNIX	デフォルト = null
SYSPLEX 探査を DB2 (OS/390 版) に接続した際に使用できるかどうかを指定する。このレジストリー変数を設定しなかったり非ゼロ値に設定したりすると、探査は使用可能。このレジストリー変数をゼロ (0) に設定すると、探査は使用禁止。ゼロに設定すると、DCS データベース・カタログ項目の指定内容にかかわらず、ゲートウェイで SYSPLEX 探査を使用できない。詳しくは、 *コマンド解説書* および **CATALOG DCS DATABASE** コマンドを参照。		
DB2TCPCONNMGRS	全	デフォルト = 1 (並列マシンの場合); 最大で 8 つの接続マネージャーに切り上げられたプロセッサー数の平方根 (対称マルチプロセッサー・マシンの場合)。 値: 1 〜 8

表 23. 通信変数 (続き)

変数名	オペレーティング・システム	値 説明
このレジストリー変数が設定されていない場合、デフォルトの接続マネージャーが使用される。このレジストリー変数が設定されていれば、ここで割り当てた値がデフォルト値を上書きする。指定された数 (最大 8) の TCP/IP 接続マネージャーが作成される。 1 より小さい値が指定された場合、DB2TCPCONNMGRS には値 1 が設定され、値が範囲外であることを示す警告がログに記録される。 8 より大きい値が指定された場合、 DB2TCPCONNMGRS には値 8 が設定され、値が範囲外であることを示す警告がログに記録される。 1 から 8 の値を指定した場合、その値がそのまま使用される。複数の接続マネージャーが作成されれば、複数のクライアント接続を同時に受け取る場合の接続スループットが向上するはずである。ユーザーが SMP マシン上で実行している場合、または DB2TCPCONNMGRS レジストリー変数を変更した場合、追加の TCP/IP 接続マネージャー・プロセス (UNIX の場合) またはスレッド (OS/2 および Windows オペレーティング・システムの場合) が行われる場合がある。追加のプロセスまたはスレッドでは、追加のストレージを必要とする。 **注:** 接続マネージャーの番号を 1 に設定すると、多くのユーザーを持つ、または頻繁に接続、切断が繰り返される、あるいはその両方のシステムのリモート接続上でパフォーマンスの低下が生じる。		
DB2_VI_ENABLE	Windows NT	デフォルト = OFF 値: ON または OFF
仮想インターフェース (VI) 体系通信プロトコルを使用するかどうかを指定する。このレジストリー変数が 『ON』 の場合、FCM では VI を使用してノード間通信を行う。このレジストリー変数が 『OFF』 の場合、FCM では TCP/IP を使用してノード間通信を行う。 **注:** このレジストリー変数は、インスタンス中のすべてのデータベース区画で同じでなければならない。		
DB2_VI_VIPL	Windows NT	デフォルト = `vipl.dll`
DB2 で使用される仮想インターフェース・プロバイダー・ライブラリー (VIPL) の名前を指定する。このライブラリーを正常にロードするには、このレジストリー変数で使用されるライブラリー名は PATH ユーザー環境変数にもなければならない。 現在サポートされているすべての実装では、同じライブラリー名を使用する。		
DB2_VI_DEVICE	Windows NT	デフォルト = null 値: `nic0 / VINIC`
装置の記号名、もしくはネットワーク・インターフェース・カード (NIC) に関連した仮想インターフェース・プロバイダー・インスタンスを指定する。独立ハードウェア・ベンダー (IHV) はそれぞれ独自の NIC を作成している。 Windows NT マシンで使用できる NIC は 1 つだけである。同一マシン上の複数の論理ノードは同一の NIC を共用する。装置記号名 『VINIC』 は、Synfinity インター・コネクトで使用する場合、大文字かつ単一である必要がある。他の現在サポートされている、すべてのインプリメンテーションは、装置記号名として 『nic0』 を使用する。		

表 24. DCE ディレクトリー変数

変数名 説明	オペレーティング・システム	値
DB2DIRPATHNAME	OS/2、UNIX、および Windows 32 ビット・オペレーティング・システム	デフォルト = null
データベース・マネージャー構成ファイル内の DIR_PATH_NAME パラメーターの一時的な指定変更を指定する。ディレクトリー・サーバーが使用されていて、 CONNECT ステートメントまたは ATTACH コマンドのターゲットが明示的にはカタログされていない場合は、ターゲットは DB2DIRPATHNAME (指定されていれば) と連結され、完全修飾の DCE 名が形成される。 **注:** DB2DIRPATHNAME 変数はインスタンスのグローバル名には影響しない。これは、常にデータベース・マネージャー構成パラメーター DIR_PATH_NAME および DIR_OBJ_NAME によって識別される。		
DB2CLIENTADPT	OS/2 および Windows 32 ビット・オペレーティング・システム	デフォルト = null 範囲: 0〜15
OS/2 および Windows 32 ビット・オペレーティング・システムの NETBIOS プロトコルのクライアント・アダプター番号を指定する。この DB2CLIENTADPT 値は、データベース・マネージャー構成ファイルの DFT_CLIENT_ADPT パラメーター値を指定変更する。		
DB2CLIENTCOMM	OS/2、UNIX、および Windows 32 ビット・オペレーティング・システム	デフォルト = null
データベース・マネージャー構成ファイル内の DFT_CLIENT_COMM パラメーター値の一時的な指定変更を指定する。 DFT_CLIENT_COMM と DB2CLIENTCOMM を両方とも指定しないと、オブジェクト内で最初に検出されたプロトコルが使用される。 いずれか一方または両方を指定すると、最初に合致したプロトコルだけが使用される。どちらの場合も、最初の接続が失敗すると再試行は行われない。		
DB2ROUTE	OS/2、UNIX、および Windows 32 ビット・オペレーティング・システム	デフォルト = null
クライアントが異なるデータベース・プロトコルでデータベースに接続したときにクライアントが使用する経路指定情報オブジェクトの名前を指定する。この DB2ROUTE 値は、データベース・マネージャー構成ファイルの ROUTE_OBJ_NAME パラメーター値を指定変更する。		

表 25. コマンド行変数

変数名	オペレーティング・システム	値
説明		
DB2BQTIME	全	デフォルト = 1 秒 最大値: 1 秒
コマンド行プロセッサー・フロントエンドが、バックエンド・プロセスが活動状態で、フロントエンドへの接続を設定しているかどうかを検査する前にスリープする時間を指定する。		
DB2BQTRY	全	デフォルト = 60 再試行 最小値: 0 再試行
コマンド行プロセッサー・フロントエンド・プロセスが、バックエンド・プロセスがすでに活動状態であるかどうかを判別しようとする回数を指定する。これは、DB2BQTIME と一緒に働く。		
DB2IQTIME	全	デフォルト = 5 秒 最小値: 1 秒
コマンド行プロセッサー・バックエンド・プロセスが、フロントエンド・プロセスがコマンドを渡すのを入力キューで待機する時間を指定する。		
DB2RQTIME	全	デフォルト = 5 秒 最小値: 1 秒
コマンド行プロセッサー・バックエンド・プロセスが、フロントエンド・プロセスからの要求を待機する時間を指定する。		

表 26. MPP 構成変数

変数名	オペレーティング・システム	値
DB2ATLD_PORTS	DB2 UDB EEE (AIX、Solaris、Windows NT の場合)	デフォルト = 6000:6063 値: num1:num2。両方の値とも 1 ～ 65535、かつ num1<=num2。
オートローダー・ユーティリティーの内部 TCP/IP 通信で使用されるポート番号の範囲を指定する。設定する場合、オートローダーは内部デフォルト・ポート範囲の 6000:6063 を使用する。オートローダーのデフォルトのポート範囲を使う別のアプリケーションがある場合、この変数は代替ポート範囲を選択するために使用される場合がある。		
DB2ATLD_PWFILE	DB2 UDB EEE (AIX、Solaris、Windows NT の場合)	デフォルト = null 値: ファイル・パス式

表 26. MPP 構成変数 (続き)

変数名	オペレーティング・システム	値
オートローダー認証の際に使用されるパスワードを含むファイルへのパスを指定する。設定しないと、オートローダーは、その構成ファイルからパスワードを取り出すか、対話式で入力を促すプロンプトを出す。この変数を使用すると、パスワード・セキュリティーの問題が扱われ、認証情報からのオートローダー構成情報の分離を許可する。		
DB2CHGPWD_EEE	DB2 UDB EEE (AIX および Windows NT の場合)	デフォルト = null 値: YES または NO
AIX または Windows NT EEE システムでのパスワード変更を他のユーザーに許可するかどうかを指定する。すべての区画またはノードのパスワードは、 Windows NT ドメイン・コントローラ (Windows NT の場合) または NIS (AIX の場合) を使って集中保守する必要がある。集中保守しないと、すべての区画またはノード間でパスワードが一致しなくなるおそれがある。その結果、ユーザーが変更を加えるために接続するデータベース区画でのみパスワードが変更される可能性がある。このグローバル・レジストリー変数の変更は、ルート・ディレクトリーおよび DAS インスタンスで行わなければならない。		
DB2_FORCE_FCM_BP	AIX	デフォルト = No 値: Yes または No
このレジストリー変数は、 DB2 UDB EEE for AIX で複数の論理区画を使用している場合に適用できる。 DB2START を発行すると、 DB2 により FCM バッファーがデータベース・グローバル・メモリーから、またはこのメモリーの余地が足りない場合は独立した共用メモリー・セグメントから割り当てられる。このセグメントは、同一の物理マシン上の (このインスタンス用の) すべての FCM デーモンで使用される。どちらが選択されるかは、作成される FCM バッファーの数にかなり左右される (つまり、FCM_NUM_BUFFERS データベース・マネージャー構成パラメーターによって決まる)。このレジストリー変数を Yes に設定すると、 FCM バッファーは常に独立したメモリー・セグメント内に作成される。FCM バッファーが独立したメモリー・セグメント内に作成される場合、同一ノード上の別々の論理区画の FCM デーモン間で共用メモリーを介して通信が行われる。グローバル・メモリーに作成される場合、同一ノード上の FCM デーモン間の通信は UNIX ソケットを介して行われる。このような方法で共用メモリーを介して通信が行われる場合、高速という利点がある。不利な点としては、他の目的、特にデータベース・バッファー・プールとして使用できる共用メモリー・セグメントが少なくなることが挙げられる。この場合、データベース・バッファー・プールの最大サイズは小さくなる。		
DB2_NUM_FAILOVER_NODES	全	デフォルト: 2 値: 0 ～ 論理ノードの数

表 26. MPP 構成変数 (続き)

変数名	オペレーティング・システム	値
高可用性環境でフェールオーバー・ノードとして使用できるノードの数を指定する。高可用性環境であれば、ノードに障害が発生したときに、そのノードを別のホストで 2 番目の論理ノードと再始動できる。この変数で使用する数から、フェールオーバー・ノードの FCM リソース用に予約されるメモリーの量が判別される。 たとえば、 1 と 2 という 2 つの論理ノードがあるホスト A と、 3 と 4 という 2 つの論理ノードがあるホスト B を想定する。 DB2_NUM_FAILOVER_NODES は 2 に設定されているとする。 DB2START 中に、ホスト A とホスト B は FCM が必要とするだけのメモリーを予約するため、最大 4 つの論理ノードを管理することができる。そして、一方のホストで障害が発生した場合、もう一方のホストで、障害が発生したホストの論理ノードを再始動することができる。		
DB2PORTRANGE	Windows NT	値: nnnn:nnnn
この値は、FCM によって使用される TCP/IP ポート範囲に設定されるので、別のマシン上に作成される追加の区分も同じポート範囲になる。		
DB2_UPDATE_PART_KEY	ALL	デフォルト = Yes 値: Yes または No
フィックスパック 3 以降の場合、デフォルト値は Yes である。このレジストリー変数は、区分化キーの更新が許されるかどうかを指定する。		

表 27. SQL コンパイラー変数

変数名 説明	オペレーティング・システム	値
DB2_ANTIJOIN	全	デフォルト=NO (EEE 環境) デフォルト=YES (EEE 環境以外) 値: YES または NO
DB2 ユニバーサル・データベース EEE 環境の場合に Yes が指定されていると、最適化プログラムは、機会があるたびに、 DB2 がより効率的に処理できる非結合に 『NOT EXISTS』 副照会をトランスフォームする。非 EEE 環境 で No が指定されていると、最適化プログラムは 『NOT EXISTS』 副照会を非結合にほとんどトランスフォームしない。		
DB2_CORRELATED_PREDICATES	全	デフォルト = Yes 値: Yes または No

表 27. SQL コンパイラー変数 (続き)

変数名 説明	オペレーティング・システム	値
この変数のデフォルトは Yes である。結合内の相関列上に固有索引があり、このレジストリー変数が Yes の場合、この最適化プログラムは、結合条件の相関を検出し、補正しようとする。このレジストリー変数が Yes の場合、最適化プログラムは相関しているケースを検出するため固有索引統計の KEYCARD 情報を使用し、結合された、関連する述部の選び方を動的に調整する。こうすることで、結合するサイズやコスト面で、より正確な見積もりを立てることができる。単純な等価述部の相関で調整が行われる。たとえば、C1 および C2 に索引が存在する場合の `WHERE C1=5 AND C2=10` など。索引は固有である必要はないが、等価述部列は索引内のすべての列を網羅していなければならない。		
DB2_HASH_JOIN アクセス・プランのコンパイル時に可能な結合方法としてハッシュ結合を指定する。	全	デフォルト = NO 値: YES または NO
DB2_LIKE_VARCHAR	全	デフォルト=Y、N 値: Y、N、S、または 0 から 6.2 までの浮動小数点定数

表 27. SQL コンパイラー変数 (続き)

変数名	オペレーティング・システム	値
説明		

サブエレメント統計の収集と使用を制御する。ブランクで区切られた一連のサブフィールドまたはサブエレメントのフォームに構造がある場合、データ内容に関する統計が列にある。
 	
このレジストリー変数は、次の形式の述部を最適化プログラムが処理する方法に影響する。
`COLUMN LIKE '%xxxxxx%'`
 	
xxxxxx は文字のストリングである。
 	
このレジストリー変数の使用方法を示す構文は次の通りである。
`db2set DB2_LIKE_VARCHAR=[Y|N|S|num1] [,Y|N|S|num2]`
 	
説明
• コンマの前にある用語、または述部の右にある唯一の用語は、正の値のサブエレメント統計を持たない列に対してのみ、以下の意味になる。
– S - % 文字で囲まれたストリングの長さに基づいて、列を形成するために連結する一連のエレメントの各エレメントの長さを見積もる。
– Y - デフォルト。アルゴリズム・パラメーターのデフォルト値 1.9 を使用する。アルゴリズム・パラメーターで可変長サブエレメント・アルゴリズムを使用する。
– N - 固定長サブエレメント・アルゴリズムを使用する。
– num1 - 可変長サブエレメント・アルゴリズムにより、アルゴリズム・パラメーターとして num1 の値を使用する。
• コンマの後の用語は次のような意味がある。
– N - デフォルト。サブエレメント統計を収集や使用もしない。
– Y - サブエレメント統計を収集する。正の値のサブエレメント統計を持つ列の場合に、アルゴリズム・パラメーターのデフォルト値 1.9 と一緒に収集した統計を使用する可変長サブエレメント・アルゴリズムを使用する。
– num2 - サブエレメント統計を収集する。正の値のサブエレメント統計を持つ列の場合に、アルゴリズム・パラメーターとして num2 の値と一緒に収集した統計を使用する可変長サブエレメント・アルゴリズムを使用する。		

DB2_SELECTIVITY	ALL	デフォルト = No 値: Yes または No

表 27. SQL コンパイラー変数 (続き)

変数名	オペレーティング・システム	値
説明		
このレジストリー変数は、SELECTIVITY 文節が使用できるかどうかを制御する。 SELECTIVITY 文節について詳しくは、*SQL 解説書*、言語エレメント、検索条件を参照。 このレジストリー変数が Yes に設定された場合、述部が、少なくとも 1 つの式がホスト変数を含む基本述部であるときは、 SELECTIVITY 文節を設定可能である。		
DB2_NEW_CORR_SQ_FF	全	デフォルト = OFF 値: ON または OFF

『ON』 に設定すると、 SQL 最適化プログラムが特定の副照会述部について計算した選択可能性値に影響する。このパラメーターを使用すると、副照会の SELECT リストで MIN または MAX 総計関数を使用する等価副照会述部の選択可能性値の正確度を高めることができる。たとえば:
`SELECT * FROM T WHERE`
`T.COL = (SELECT MIN(T.COL)`
`FROM T WHERE ...)`		

DB2_PRED_FACTORIZE	全	デフォルト = NO 値: YES または NO

表 27. SQL コンパイラー変数 (続き)

変数名	オペレーティング・システム	値
説明		

最適化プログラムが、論理和から追加の述部を取り出す機会を探索するかどうかを指定する。状況によっては、追加の述部は中間の推定カーディナリティー、または結果セットを変更できる。以下の照会がある。
`SELECT n1.empno,`
`n1.lastname`
`FROM employee n1,`
`employee n2`
`WHERE`
`((n1.lastname='SMITH'`
`AND n2.lastname='JONES')`
`OR (n1.lastname='JONES'`
`AND n2.lastname='SMITH'))`
最適化プログラムは、以下の追加述部を生成できる。
`SELECT n1.empno,`
`n1.lastname`
`FROM employee n1,`
`employee n2`
`WHERE n1.lastname IN`
`('SMITH', 'JONES')`
`AND n2.lastname IN`
`('SMITH', 'JONES')`
`AND`
`((n1.lastname='SMITH'`
`AND n2.lastname='JONES')`
`OR (n1.lastname='JONES'`
`AND n2.lastname='SMITH'))`		

表 28. パフォーマンス変数

変数名	オペレーティング・システム	値
説明		
DB2_AVOID_PREFETCH	全	デフォルト = OFF 値: ON または OFF
破損リカバリーにおいて、事前取り出しを使用するか否かを指定する。 DB2_AVOID_PREFETCH=0N の場合は、事前取り出しは使用されない。		

表 28. パフォーマンス変数 (続き)

変数名	オペレーティング・システム	値
説明		
DB2_AWE	Windows 2000	デフォルト = null 値: <entry>[,<entry>,...] (<entry>=<バッファー・プール ID>、<物理ページの数>、<アドレス・ウィンドウの数>)
Windows 2000 の DB2 UDB では、バッファー・プールに 64 GB までのメモリーを割り振ることができる。 Windows 2000 は、Address Windowing Extensions (AWE) バッファー・プールをサポートするように正しく構成されている必要がある。これには、「メモリ内のページのロック」権利をユーザーに関連付け、物理ページおよびアドレス・ウィンドウ・ページを割り振り、このレジストリー変数を設定する作業が含まれる。この変数を設定するには、 AWE サポートに使用するバッファー・プールのバッファー・プール ID を知っておく必要がある。バッファー・プールの ID は、 SYSCAT.BUFFERPOOLS システム・カタログ・ビューの BUFFERPOOLID 列にある。 **注:** AWE サポートが使用可能になっている場合には、拡張ストレージをデータベースのバッファー・プールに対して使用することはできない。また、このレジストリー変数を使用して参照されるバッファー・プールは、すでに SYSCAT.SYSBUFFERPOOLS に存在していなければならない。		
DB2_BINSORT	全	デフォルト = YES 値: YES または NO
分類の CPU 時間と経過時間が減少する新しい分類アルゴリズムを使用可能にする。この新アルゴリズムにより、DB2 UDB の非常に効率的な整数分類技法が、あらゆる分類データ・タイプ (BIGINT、CHAR、VARCHAR、FLOAT、DECIMAL、およびそれらを組み合わせたデータ・タイプ) に拡張される。この新アルゴリズムを使用可能にするには、次のコマンドを使用する。 `db2set DB2_BINSORT = yes`		
DB2BPVARS	Windows NT	デフォルト = パス

表 28. パフォーマンス変数 (続き)

変数名	オペレーティング・システム	値
説明		

バッファー・プールのチューニング時にパラメーターが含まれるファイルへのパスを指定する。現在サポートされているパラメーターは、 NT_SCATTER_DMSFILE, NT_SCATTER_DMSDEVICE および NT_SCATTER_SMS である。
 	
上記の個々のパラメーターのデフォルトはゼロ (OFF) で、可能な値はゼロ (OFF) および 1 (ON) である。各パラメーターは、それぞれのコンテナー・タイプの分散読み取りをオンにする場合に使用する。使用可能 (ON) にすることができるのは、登録で DB2NTNOCACHE が ON に設定されている場合だけである。 DB2NTNOCACHE の設定が OFF (または未設定) の場合は、 db2diag.log に警告メッセージが書き込まれ、分散読み取りは使用不可のままになる。これらのパラメーターは、それぞれのコンテナー・タイプに対する順次事前取り出しが大量に行われ、 ON に設定された DB2NTNOCACHE の使用をすでに決定したシステムで推奨されている。
注: DB2NTNOCACHE を ON に設定するとき、 Windows NT ファイル・キャッシングをオフにする。
 	
以下にファイル・パスの設定方法の例を示す。
`db2set DB2BPVARS =`
`f:¥BPVARSFILE`
 	
ファイルの内容は、ここに示したパラメーターを以下の形式で指定する。
`parameter=value`		

DB2CHKPTR	全	デフォルト = OFF 値: ON または OFF
入力のポインター検査が必要であるか否かを指定する。		
DB2_ENABLE_BUFPD	全	デフォルト = OFF 値: ON または OFF
照会パフォーマンスを向上させるために DB2 で中間バッファリングを使用するかどうか指定する。バッファリングによって、あらゆる環境で照会パフォーマンスが向上するわけではない。それぞれの照会パフォーマンスが向上するかどうか判別するために、テストを行う必要がある。		
DB2_EXTENDED_OPTIMIZATION	全	デフォルト = OFF 値: ON または OFF
照会パフォーマンスを向上させるために、照会最適化プログラムが最適化拡張を使用するかどうかを指定する。拡張によって、あらゆる環境で照会パフォーマンスが向上するわけではない。それぞれの照会パフォーマンスが向上するかどうか判別するために、テストを行う必要がある。		

表 28. パフォーマンス変数 (続き)

変数名	オペレーティング・システム	値
説明		
DB2MAXFSCRSEARCH	全	デフォルト= 5 値: -1、1 ～ 33 554
表にレコードが追加された場合、検索のために、フリー・スペース制御レコードの数を指定する。デフォルトは、フリー・スペース制御レコードを 5 つ検索する。この値の変更は、スペース再利用で挿入速度の平衡を取れるようにする。スペース再利用の最適化のため大きな値を使用する。挿入速度の最適化のため小さな値を使用する。値を -1 に設定するとデータベース・マネージャーはすべてのフリー・スペース制御レコードを強制的に検索する。		
DB2MEMDISCLAIM	AIX	デフォルト = YES 値: YES または NO
AIX では、DB2 処理が使用するメモリーに、関連するページング・スペースを存在させることができる。このページング・スペースは、関連するメモリーが解放された後も予約されたままになる場合がある。これは、AIX システムの (調整可能な) 仮想メモリー管理割り振りポリシーによって異なる。DB2MEMDISCLAIM レジストリー変数は、解放されたメモリーと予約ページング・スペースとの関連付けを AIX で解除する要求を DB2 エージェントが明示的に出すかどうかを制御する。 DB2MEMDISCLAIM を YES に設定すると、ページング・スペースの所要量が少なくなり、ページングのディスク活動も減る。 DB2MEMDISCLAIM を NO に設定すると、ページング・スペースの所要量は多くなり、ページングのディスク活動も増える。ページング・スペースが多い場合や、ページングが行われないほど実メモリーが十分にある場合などは、 NO を設定してもパフォーマンスはわずかしか向上しない。		
DB2MEMMAXFREE	全	デフォルト= 8 388 608 バイト 値: 0 ～ 2^{32}-1 バイト
DB2 の処理により保存された未使用メモリーの最大量を指定する (バイト単位)。		
DB2_MMAP_READ	AIX	省略値 = ON 値: ON または OFF
DB2_MMAP_WRITE と一緒に使用して、入出力の代替方法として DB2 が mmap を使用できるようにする。複数プロセスが同一ファイルの異なるセクションに書き出す場合は、ほとんどの環境において、オペレーティング・システムのロックを防ぐために mmap を使用するのがよい。しかし、 (バッファー・プールとは無関係に) JFS ファイル・システムからメモリーへの DB2 データの AIX によるキャッシングを可能にして、デフォルトが OFF のパラレル・エディション V1.2 から移行した場合もある。 DB2 UDB と同等のパフォーマンスにしたい場合、バッファー・プールのサイズを増やすか、DB2_MMAP_READ と DB2_MMAP_WRITE を OFF に変更することができる。		

表 28. パフォーマンス変数 (続き)

変数名	オペレーティング・システム	値
説明		
DB2_MMAP_WRITE	AIX	省略値 = ON 値: ON または OFF
DB2_MMAP_READ と一緒に使用して、入出力の代替方法として DB2 が mmap を使用できるようにする。複数プロセスが同一ファイルの異なるセクションに書き出す場合は、ほとんどの環境において、オペレーティング・システムのロックを防ぐために mmap を使用するのがよい。 しかし、 (バッファー・プールとは無関係に) JFS ファイル・システムからメモリーへの DB2 データの AIX キャッシングをできるようにして、デフォルトが OFF のパラレル・エディション V1.2 から移行した場合もある。 DB2 UDB と同等のパフォーマンスにしたい場合、バッファー・プールのサイズを増やすか、DB2_MMAP_READ と DB2_MMAP_WRITE を OFF に変更することができる。		
DB2_NO_PKG_LOCK	全	デフォルト = OFF 値: ON または OFF
グローバル SQL キャッシュが、キャッシュされたパッケージ項目を保護するためにパッケージ・ロックを使用しないで操作できるようにする。 (パッケージ・ロックは内部システム・ロック。) パフォーマンスを向上させるためには (ロックの獲得と解放には時間がかかるので)、現時点では 『no package lock』モードでの動作を選択することができる。このモードでは、特定のデータベース操作ができない。パッケージを無効にする操作、パッケージを操作不能にする操作、パッケージを直接変更する操作が、これに含まれる。		
DB2NTMEMSIZE	Windows NT	デフォルト = (メモリー・セグメントにより異なる)
Windows NT では、プロセス間で確実にアドレスが一致するために DLL 初期設定時にすべての共用メモリー・セグメントを予約する必要がある。そこで、必要に応じて Windows NT のデフォルトを指定変更するために新しいプロファイル登録値 DB2NTMEMSIZE が用意されている。 ほとんどの状態では、デフォルト値で十分なはずである。メモリー・セグメント、デフォルトのサイズ、および指定変更オプションは以下のとおり。 1) データベース・カーネル: デフォルトのサイズ 16777216 (16 MB); 指定変更オプション DBMS:<number of bytes> 2) 並列 FCM バッファー: デフォルトのサイズ 22020096 (21 MB); 指定変更オプション FCM:<number of bytes> 3) データベース Admin GUI: デフォルトのサイズ 33554432 (32 MB); 指定変更オプション DBAT:<number of bytes> 4) 分離ストアード・プロシージャー: デフォルトのサイズ 16777216 (16 MB); 指定変更オプション APLD:<number of bytes> 指定変更オプションをセミコロン (;) で区切って、複数のセグメントを指定変更できる。たとえば、データベース・カーネルを 256K に制限し、 FCM バッファーを 64 MB に制限するには、以下のようにする。 `db2set DB2NTMEMSIZE=DBMS:256000;FCM:64000000`		

表 28. パフォーマンス変数 (続き)

変数名	オペレーティング・システム	値
説明		
DB2NTNOCACHE	Windows NT	デフォルト = OFF 値: ON または OFF
DB2 が NOCACHE オプション付きでデータベース・ファイルをオープンするか否かを指定する。DB2NTNOCACHE=ON の場合は、ファイル・システムのキャッシュは除去される。DB2NTNOCACHE=OFF の場合、オペレーティング・システムは DB2 ファイルをキャッシュに保存する。これは、long fields または LOBS を含んでいるファイルを除くすべてのデータに適用される。 システム・キャッシュを除去すると、より多くのメモリーがデータベースに利用できるようになるため、バッファー・プールや分類ヒープの量を増すことができる。		
DB2NTPRICLASS	Windows NT	デフォルト = null 値: R、H、(任意の他の値)
DB2 インスタンスの優先度クラスを設定する (プログラム DB2SYSCS.EXE)。次の 3 つの優先度クラスがある。 • NORMAL_PRIORITY_CLASS (デフォルトの優先度クラス) • REALTIME_PRIORITY_CLASS (『R』 を使って設定) • HIGH_PRIORITY_CLASS (『H』 を使って設定) この変数は、個々のスレッド優先順位 (DB2PRIORITIES を使って設定) と一緒に使われ、システム中の別のスレッドに関する DB2 スレッドの絶対優先順位を決定する。 **注:** この変数を使用する際には、注意しなければならない。誤用すると、システム・パフォーマンス全体に悪い影響を及ぼす可能性がある。 詳細は、Win32 資料の *SetPriorityClass()* API を参照。		
DB2NTWORKSET	Windows NT	デフォルト = 1,1
DB2 に利用できる最小および最大の実効ページ・セットを変更するために使用される。デフォルトとして、ページングが行われていない場合は、プロセスの実効ページ・セットは必要なだけ大きくすることができる。ただし、ページングが発生しているときは、プロセスが持つことができる最大の実効ページ・セットは約 1 MB である。 DB2NTWORKSET を使えば、このデフォルトの動作を指定変更できる。 DB2 に対する DB2NTWORKSET の指定は、 DB2NTWORKSET=min,max の構文を使用する。ここで、min と max はメガバイト単位で表される。		
DB2_OVERRIDE_BPF	全	デフォルト = 設定しない 値: 正数のページ数

表 28. パフォーマンス変数 (続き)

変数名	オペレーティング・システム	値
説明		
データベース活動化時または初回接続時に作成されるバッファー・プールのサイズをページ単位で指定する。これが役立つのは、メモリー制約の結果としてデータベース活動化時または初回接続時に障害が発生する場合である。データベース・マネージャーによって最小のバッファー・プール (16 ページ) も起動しない場合、ユーザーはこの環境変数を使ってさらに小さいページ数を指定した後に再試行することができる。メモリー制約は、実メモリーの不足 (めったに起こらない) のために起こることもあれば、データベース・マネージャーが間違った構成の大きいバッファー・プールを割り当てようとしたために起こることもある。設定時に、この値は現行バッファー・プールを指定変更する。		
DB2_PINNED_BP	AIX、HP-UX	デフォルト = NO 値: YES または NO

この変数は、一部の AIX オペレーティング・システムで、データベースに関連するデータベース・グローバル・メモリー (バッファー・プールを含む) をメイン・メモリーに保持するために使用される。データベース・グローバル・メモリーをシステム・メイン・メモリーに保持することにより、データベース・パフォーマンスがより一貫性のあるものになる。
 	
たとえば、バッファー・プールがシステム・メイン・メモリーからスワップアウトされた場合、データベース・パフォーマンスは低下する。バッファー・プールをシステム・メモリーに保持することによってディスク入出力が減ると、データベース・パフォーマンスは改善される。別のアプリケーションがより多くのメイン・メモリーを要求した場合、システム・メイン・メモリー所要量に応じて、データベース・グローバル・メモリーをメイン・メモリーからスワップアウトできる。
 	
64 ビット環境で HP-UX を使用するときは、このレジストリー環境を変更する他に、 DB2 インスタンス・グループに MLOCK 特権を与えなければならない。これを行うには、root のアクセス権限を持つユーザーに対して、次の作業を依頼する必要がある。
1. DB2 インスタンス・グループを `/etc/privgroup` ファイルに追加する。たとえば、DB2 インスタンス・グループが `db2iadm1` グループに属している場合、次の行を `/etc/privgroup` ファイルに追加する。
`db2iadm1 MLOCK`
2. 次のコマンドを発行する。
`setprivgrp -f /etc/privgroup`		

DB2PRIORITIES	全	値の設定はプラットフォームにより異なる
DB2 プロセスとスレッドの優先順位を制御する。		
DB2_RR_TO_RS	全	デフォルト = NO 値: YES または NO

表 28. パフォーマンス変数 (続き)

変数名	オペレーティング・システム	値
説明		
次のキーのロック は、すべての INSERT および DELETE ステートメントの次のキー、および SELECT ステートメントの結果セットよりも大きい次のキー値を自動的にロックすることにより、反復可能読み取り (RR) 分離レベルを保証する。索引のキー部分を変更する UPDATE ステートメントの場合、元の索引キーは削除され、新しいキー値が挿入される。キーの挿入とキーの削除の両方で、次のキーのロックが行われる。次のキーロックは、ANSI および SQL92 規格を保証するために必要であり、これは DB2 のデフォルトである。 アプリケーションが停止したりハングしたりする場合、アプリケーションのスナップショット情報を調べる。問題が次のキーのロックに関するものと思われる場合、 2 つの条件に基づいて、DB2_RR_TO_RS レジストリー変数をオンにする。つまり、どのアプリケーションも反復可能読み取り (RR) の振る舞いに依存せず、走査の際にコミットされずに削除されたものをスキップするようにされている場合に、DB2_RR_TO_RS をオンにする。スキップの振る舞いは、 RR、読み取り固定 (RS)、およびカーソル固定 (CS) の分離レベルに影響を与える。 (非コミット読み取り (UR) 分離レベルの場合、行のロックはない。) DB2_RR_TO_RS がオンの場合、索引キーの挿入時および削除時には次のキーのロックは行われないため、 RR の振る舞いはユーザー表での走査に関して保証されない。カタログ表は、このオプションの影響を受けない。 振る舞いにおける別の変更として、 DB2_RR_TO_RS がオンの場合、走査の際に、削除されたもののコミットされていない行をスキップする。走査が行われるはずの行であったとしても、それはスキップされる。		
DB2_SORT_AFTER_TQ	全	デフォルト = NO 値: YES または NO
受信終了時にデータを分類することが必要で、受信ノード数が送信ノード数と等しい場合、最適化プログラムが区分データベースの直接表キューを処理する方法を指定する。 DB2_SORT_AFTER_TQ=NO の場合、最適化プログラムは送信終了時には行の分類を、受信終了時には行のマージを行う傾向がある。 DB2_SORT_AFTER_TQ=YES の場合、最適化プログラムは分類をしないで行を送信し、すべての行を受信した後の受信終了時にもマージを行わない傾向がある。		
DB2_STPROC_LOOKUP_FIRST	全	デフォルト = OFF 値: ON または OFF

表 28. パフォーマンス変数 (続き)

変数名	オペレーティング・システム	値
説明		
以前の DB2_DARI_LOOKUP_ALL。この変数は、*sqllib* サブディレクトリーの *function* サブディレクトリー、および *sqllib* サブディレクトリーの *function* サブディレクトリーの *unfenced* サブディレクトリーを検索する前に、 UDB サーバーがすべての DARI およびストアード・プロシージャーのカタログ検索を実行するかどうかを指定する。 **注:** 上記のディレクトリーにある PARAMETER TYPE DB2DARI のストアード・プロシージャーの場合、この値を 『ON』 に設定するとパフォーマンスが低下する。これは、カタログ検索が、関数ディレクトリーを検索する前に EEE 構成内の別のノードで実行されるからである。 ストアード・プロシージャーを呼び出す際の DB2 のデフォルトの振る舞いは、 *sqllib* サブディレクトリーの *function* サブディレクトリー、および *sqllib* サブディレクトリーにある *function* サブディレクトリーの *unfenced* サブディレクトリーから、ストアード・プロシージャーと同じ名前を持つ共用ライブラリーを検索することである。これは、システム・カタログ内のストアード・プロシージャーの共用ライブラリー名を検索する前に行われる。共用ライブラリーと同じ名前を持つのは PARAMETER TYPE DB2DARI のストアード・プロシージャーだけであるため、 DB2DARI ストアード・プロシージャーだけが DB2 のデフォルトの振る舞いから益を受ける。異なる PARAMETER TYPE を使用してカタログを作成したストアード・プロシージャーを使用する場合、 DB2 が上記のディレクトリーを検索するために費やす時間のため、それらのストアード・プロシージャーのパフォーマンスは低下する。 PARAMETER TYPE DB2DARI としてカタログを作成していないストアード・プロシージャーのパフォーマンスを向上させるには、 DB2_STPROC_LOOKUP_FIRST レジストリー変数を ON に設定する。このレジストリー変数は、DB2 が上記のディレクトリーを検索する前に、システム・カタログ内のストアード・プロシージャーの共用ライブラリー名を検索するよう強制する。		

表 29. データ・リンク変数

変数名	オペレーティング・システム	値
説明		
DLFM_BACKUP_DIR_NAME	AIX、Windows NT	デフォルト: null 値: TSM または任意の有効なパス
使用するバックアップ装置を指定する。このレジストリー変数の設定を TSM から実行時のパスに変更しても、アーカイブ・ファイルは移動されない。新しいバックアップだけが新しい位置に置かれる。それまでにアーカイブされたファイルは移動させられない。		
DLFM_BACKUP_LOCAL_MP	AIX、Windows NT	デフォルト: null 値: DFS システム内のローカル・マウント・ポイントへの任意の有効なパス

表 29. データ・リンク変数 (続き)

変数名	オペレーティング・システム	値
説明		
DFS システム内のマウント・ポイントへの完全修飾パスを指定する。パスを指定した場合、DLFM_BACKUP_DIR_NAME で指定されているパスの代わりに、そのパスが使用される。		
DLFM_BACKUP_TARGET	AIX、Windows NT	デフォルト: null 値: LOCAL、TSM、XBSA
使用するバックアップのタイプを指定する。		
DLFM_BACKUP_TARGET_LIBRARY	AIX、Windows NT	デフォルト: null 値: DLL または共用ライブラリー名への任意の有効なパス
DLL または共用ライブラリーへの完全修飾パスを指定する。このライブラリーは、 *libdfmxbsa.a* ライブラリーを使用してロードされる。		
DLFM_ENABLE_STPROC	AIX、Windows NT	デフォルト: NO 値: YES または NO
ストアード・プロシージャーを使ってファイル・グループをリンクするかどうかを指定する。		
DLFM_FS_ENVIRONMENT	AIX、Windows NT	デフォルト: NATIVE 値: NATIVE または DFS
データ・リンク・サーバーが稼働する環境を指定する。 NATIVE を指定すると、データ・リンク・サーバーは単一マシンの状態で稼働する。この状態では、サーバーはそのマシン自体のファイルを管理することができる。 DFS を指定すると、データ・リンク・サーバーは分散ファイル・システム (DFS) 環境で稼働する。この環境では、サーバーはファイル・システム全体のファイルを管理することができる。DFS ファイル・セットとネイティブのファイル・システムを組み合わせて使用することはできない。		
DLFM_GC_MODE	AIX、Windows NT	デフォルト: PASSIVE 値: SLEEP、 PASSIVE、または ACTIVE
データ・リンク・サーバーでのガーベッジ・ファイル・コレクションの制御を指定する。 SLEEP に設定すると、ガーベッジ・コレクションは行われない。 PASSIVE に設定すると、他のトランザクションが実行されていない場合にのみガーベッジ・コレクションが実行される。 ACTIVE に設定すると、他のトランザクションが実行されている場合でもガーベッジ・コレクションが実行される。		

表 29. データ・リンク変数 (続き)

変数名 説明	オペレーティング・システム	値
DLFM_INSTALL_PATH	AIX、 Windows NT	デフォルト AIX の場合: /usr/lpp/ db2_06_00 /adm NT の場合: DB2PATH /bin 範囲: 任意の有効なパス
データ・リンク実行可能ファイルがインストールされているパスを指定する。		
DLFM_LOG_LEVEL	AIX、 Windows NT	デフォルト: LOG_INFO 値: LOG_CRIT、 LOG_DEBUG、 LOG_ERR、 LOG_INFO、 LOG_NOTICE、 LOG_WARNING
記録する診断情報のレベルを指定する。		
DLFM_PORT	Windows 3.n を除くすべて	デフォルト: 50100 値: 任意の有効なポート番号
DB2 データ・リンク・マネージャーを実行するデータ・リンク・サーバーとの通信に使用するポート番号を指定する。この環境変数は、表に 『DATALINKS』 列が含まれている場合にだけ使用する。		
DLFM_TSM_MGMTCLASS	AIX、Windows NT、Solaris	デフォルト: デフォルト TSM 管理クラス 値: 有効 TSM 管理クラス
リンクしたファイルをアーカイブ、検索するために使用する TSM 管理クラスを指定する。この変数の値セットがない場合は、デフォルト TSM 管理クラスが使用される。		

表 30. その他の変数

変数名 説明	オペレーティング・システム	値
DB2ADMINSERVER	OS/2、 Windows 95、 Windows NT および UNIX	デフォルト = null
DB2 管理サーバーとしてセットアップする DB2 インスタンスを指定する。		
DB2CLIINIPATH	全	デフォルト = null

表 30. その他の変数 (続き)

変数名	オペレーティング・システム	値
説明		
DB2 CLI/ODBC 構成ファイル (db2cli.ini) のデフォルト・パスを指定変更し、クライアントの異なる位置を指定するために使用される。ここで指定される値は、クライアント・システム上の有効なパスでなければならない。		
DB2DEFPREP	全	デフォルト = NO 値: ALL、YES または NO
DEFERRED_PREPARE プリコンパイル・オプションが利用可能になる前にプリコンパイルされたアプリケーションのために、このオプションの実行時の動作をシミュレートする。たとえば、DB2 V2.1.1 またはそれ以前のアプリケーションを DB2 V2.1.2 以降の環境で実行するときは、 DB2DEFPREP を使用して、望ましい「据え置き準備」動作を指示することができる。		
DB2_DJ_COMM	全	デフォルト = null 値には、libdrda.a、libsqlnet.a、libnet8.a、libdrda.dll、libsqlnet.dll、 libnet8.dll などが含まれる。
データベース・マネージャーの始動時にロードされるラッパー・ライブラリーを指定する。この変数を指定すると、頻繁に使用されるラッパーをロードするときの実行時コストが減少する。他のオペレーティング・システムには別の値がサポートされている (Windows NT オペレーティング・システムには .dll 拡張子、 AIX オペレーティング・システムには .a 拡張子)。ライブラリー名は、プロトコルおよびオペレーティング・システムによって異なる。この変数は、データベース・マネージャー・パラメーター *federated* を YES に設定しない限り使用できない。		
DB2DMNBCKCTLR	Windows NT	デフォルト = null 値: ? またはドメイン・ネーム
DB2 サーバーがバックアップ・ドメイン・コントローラーになっている場合、そのドメイン・ネームが分かれば、 DB2DMNBCKCTLR=DOMAIN_NAME と設定する。 DOMAIN_NAME は大文字でなければならない。ローカル・マシンがバックアップ・ドメイン・コントローラーになっているドメインを DB2 が判別するには、 DB2DMNBCKCTLR=? と設定する。 DB2DMNBCKCTLR プロファイル変数を設定しなかったりブランクに設定したりすると、 DB2 は 1 次ドメイン・コントローラーで認証を実行する。 **注:** デフォルトでは、DB2 はバックアップ・ドメイン・コントローラーを使用しない。バックアップ・ドメイン・コントローラーは 1 次ドメイン・コントローラーと同期しないことがあり、機密漏れが生じることがあるからである。 1 次ドメイン・コントローラーのセキュリティー・データベースが更新されたが、その変更内容がバックアップ・ドメイン・コントローラーに伝搬していない場合に、同期しなくなることがある。この事態は、ネットワーク待ち時間が生じた場合やコンピューターのブラウザー・サービスが作動可能でない場合に起こることがある。		

表 30. その他の変数 (続き)

変数名 説明	オペレーティング・システム	値
DB2_ENABLE_LDAP	全	デフォルト = NO 値: YES または NO
Lightweight Directory Access Protocol (LDAP) を使用するかどうかを指定する。 LDAP は、ディレクトリー・サービスへのアクセス方式の 1 つである。		
DB2_FALLBACK	Windows NT	デフォルト = OFF 値: ON または OFF
この変数を使用することによって、フォールバック処理中に強制的にすべてのデータベース接続を切断できる。これは、Microsoft Cluster Server (MSCS) がある Windows NT 環境で、フェールオーバー・サポートと一緒に使用される。 DB2_FALLBACK が未設定、または OFF に設定されていて、フォールバックの間データベースが接続されている場合、 DB2 リソースをオフラインにすることはできない。 つまり、フォールバック処理は失敗する。		
DB2_FORCE_TRUNCATION	全	デフォルト = NO 値: YES または NO
再始動リカバリーの際に使われる。 『NO』 に設定されている場合、不正なページがあまりにも早く再始動リカバリーを停止したことを判別すると、再始動リカバリーを一時停止する (つまり、すべてのアクティブなログは読み込まれていない)。これは普通、ログの 1 つの不正なページが原因で起こる。この変数を 『YES』 に設定することによって、再始動リカバリーに、まるでログの終わりに達したかのように処理を継続する必要がある旨の信号を送ることができる。変数を 『YES』 に設定した後、データベースが再びアクティブになると、再始動リカバリーの際に読み込まれなかったログが上書きされる。デフォルトは 『NO』 で、不正なページが検出されない場合は継続**しない**ことを意味する。この変数は、IBM サービス技術員の指示のもとでのみ使用すること。		
DB2_GRP_LOOKUP	Windows NT	デフォルト = null 値: LOCAL、DOMAIN
この変数を使うと、DB2 は、ユーザー・アカウントを妥当性検査する場所と、グループ・メンバー検索を実行する場所を判別できる。変数を LOCAL に設定すると、DB2 で常に DB2 サーバー上のグループを列挙し、ユーザー・アカウントを妥当性検査することができる。変数を DOMAIN に設定すると、ユーザー・アカウントが属する Windows NT ドメイン上のグループを DB2 で常に列挙し、ユーザー・アカウントを妥当性検査することができる。		
DB2_INDEX_2BYTEVARLEN	全	デフォルト = NO 値: YES または NO

表 30. その他の変数 (続き)

変数名	オペレーティング・システム	値
説明		
このレジストリー変数を使用すると、 255 バイトよりも長い列を索引キーの一部として指定することができる。このレジストリー変数を YES にする前にすでに作成された索引には、引き続き 255 キー制限がある。このレジストリー変数を Yes にした後に作成された索引は、レジストリー変数を再び No にしたとしても、2 バイトの索引として扱われる。 このレジストリー変数を変更することにより、 CREATE TABLE、CREATE INDEX、および ALTER TABLE を含むいくつかの SQL ステートメントが影響を受ける。これらのステートメントについての詳細は、 *SQL 解説書* に記述された変更を参照。		
DB2LDAP_BASEDN	全	デフォルト = null 値: 任意の有効な基底ドメイン・ネーム。
LDAP ディレクトリーの基底ドメイン・ネームを指定する。		
DB2LDAPCACHE	全	デフォルト = YES 値: YES または NO

LDAP キャッシュを使用可能にするかどうかを指定する。このキャッシュは、ローカル・マシン上のデータベース、ノード、および DCS ディレクトリーのカタログを作成するのに使用する。
 	
確実にキャッシュ内の項目を最新のものにするには、以下を行う。
`REFRESH LDAP DB DIR`
`REFRESH LDAP NODE DIR`
 	
これらのコマンドは、データベース・ディレクトリーおよびノード・ディレクトリーの項目を更新し、正しくない項目は除去する。		

DB2LDAP_CLIENT_PROVIDER	Windows 95/98/NT/2000 のみ	デフォルト = null (使用可能であれば Microsoft が使用される。そうでなければ IBM が使用される。) 値: IBM または Microsoft
Windows 環境で稼働している場合、 DB2 は LDAP ディレクトリーへアクセスするために、 Microsoft LDAP クライアントか IBM LDAP クライアントのいずれかの使用をサポートしている。このレジストリー変数は、 DB2 が使用する LDAP クライアントを明示的に選択するのに使用する。 **注:** このレジストリー変数の現行値を表示するには、以下の `db2set` コマンドを使用する。 `db2set DB2LDAP_CLIENT_PROVIDER`		

表 30. その他の変数 (続き)

変数名 説明	オペレーティング・システム	値
DB2LDAPHOST LDAP ディレクトリーの位置のホスト名を指定する。	全	デフォルト = null 値: 任意の有効なホスト名
DB2LDAP_SEARCH_SCOPE Lightweight Directory Access Protocol (LDAP) の区画またはドメインで検出された情報の検索範囲を指定する。 『LOCAL』 を指定すると、LDAP ディレクトリー内の探索は使用不可になる。 『DOMAIN』 を指定すると、現行ディレクトリー区画の LDAP 内だけを探索する。 『GLOBAL』 を指定すると、オブジェクトが見つかるまで全ディレクトリー区画内の LDAP を探索する。	全	デフォルト = DOMAIN 値: LOCAL、DOMAIN、GLOBAL
DB2LOADREC ロールフォワード時にロード・コピーの位置を指定変更するために使用される。ユーザーがロード・コピーの物理的な位置を変更している場合には、ロールフォワードを出す前に DB2LOADREC を設定しておく必要がある。	全	デフォルト = null
DB2LOCK_TO_RB ロック・タイムアウトの場合にトランザクション全体をロールバックするか、または現行のステートメントだけをロールバックするかを指定する。 DB2LOCK_TO_RB が STATEMENT に設定されていると、ロック・タイムアウトによってロールバックされるのは、現行のステートメントだけになる。 その他の設定では、トランザクション全体がロールバックされる。	全	デフォルト = null 値: ステートメント
DB2_NEWLOGPATH2 このパラメーターを使用すると、重複ロギングを行うために別のパスを使用するかどうかを指定できる。使用されるパスは、 *logpath* データベース構成パラメーターの現行値に 『2』 を付加することで生成される。	UNIX	デフォルト = 0 値: 0 または 1
DB2NOEXITLIST	全	デフォルト = OFF 値: ON または OFF

表 30. その他の変数 (続き)

変数名	オペレーティング・システム	値
説明		
定義される場合、この値は、 DB2 がアプリケーションに出口リスト・ハンドラーをインストールしないこと、かつ COMMIT を実行しないことを示す。普通、DB2 はアプリケーションに処理出口リスト・ハンドラーをインストールし、アプリケーションが正常に終了するとその出口リスト・ハンドラーが COMMIT 操作を実行する。 アプリケーションが DB2 ライブラリーを動的にロードし、そのライブラリーをアンロードしてからアプリケーションが終了する場合、出口リスト・ハンドラーのルーチンは、既にアプリケーションからアンロードされているので、そのハンドラーの呼び出しは失敗する。この方法でアプリケーションが操作する場合、 DB2NOEXITLIST 変数を設定し、アプリケーションが必ず必須の COMMIT すべてを明示的に呼び出すことを確認する。		
DB2REMOTEPREG	Windows 95 および Windows NT	デフォルト = null 値: Windows 95 または Windows NT の任意の有効なマシン名
DB2 インスタンス・プロファイルおよび DB2 インスタンスの Win32 登録リストが入っているリモート・マシン名を指定する。 DB2REMOTEPREG の値は、 DB2 のインストール後にただ一度だけ設定し、変更してはならない。この変数の使用には十分な注意が必要である。		
DB2ROUTINE_DEBUG	AIX および Windows NT	デフォルト = OFF 値: ON、OFF
Java ストアード・プロシージャー用のデバッグ機能を使用可能にするかどうかを指定する。 Java ストアード・プロシージャーをデバッグしない場合は、デフォルト OFF を使用する。デバッグを使用可能にすると、パフォーマンス上の影響がある。 Java ストアード・プロシージャーのデバッグについての詳細は、 *アプリケーション開発の手引き* を参照。		
DB2SORCVBUF	Windows 95 および Windows NT	デフォルト = 32767
Windows 95 および Windows NT オペレーティング・システムでの TCP/IP 受信バッファーの値を指定する。		
DB2SORT	全、ただしサーバーのみ	デフォルト = null
ロード・ユーティリティーが実行時にロードするライブラリーの位置を指定する。このライブラリーには、索引付きデータの分類に使用される関数の入り口点が入っている。表索引の生成時に LOAD ユーティリティーとともに製作者提供の分類用製品を利用するときは、 DB2SORT を使用する。提供されるパスは、データベース・サーバーとの関係で表される必要がある。		

表 30. その他の変数 (続き)

変数名 説明	オペレーティング・システム	値
DB2SYSTEM	Windows NT、Windows 95、OS/2 および UNIX	デフォルト = null
DB2 サーバー・システムを識別するためにユーザーおよびデータベース管理者が使用する名前を指定する。できれば、この名前は使用するネットワーク内で固有のものであることが望ましい。 この名前は、コントロール・センターのオブジェクト・ツリーのシステム・レベルに表示されるので、これにより、管理者がコントロール・センターから管理できるサーバー・システムを識別するのに役立つ。 クライアント構成援助機能の「ネットワーク探索」機能を使用すると、 DB2 ディスカバリーは、この名前を戻し、その結果のオブジェクト・ツリーのシステム・レベルにその名前を表示する。この名前によって、ユーザーがアクセスしたいデータベースの入っているシステムを知ることができる。 DB2SYSTEM の値は、インストール時に次のように設定される。 • Windows NT または Windows 95 では、セットアップ・プログラムが Windows システムに指定されているコンピューター名と等しい名前を設定する。 • OS/2 では、ユーザーはインストール・プロセスで DB2SYSTEM 名を入力するように促される。 • UNIX システムでは、 UNIX システムの TCP/IP ホスト名に等しい名前に設定される。		
DB2UPMPR	OS/2	省略値 = ON 値: ON または OFF
OS/2 において、ユーザーが誤ったユーザー ID またはパスワードを入力したときに、 UPM ログオン画面を表示するか否かを指定する。		
DB2_VENDOR_INI	AIX、HP-UX、Sun Solaris、および Windows NT	デフォルト = null 値: 任意の有効なパスおよびファイル。
すべてのベンダー特定の環境設定を含むファイルを示す。データベース・マネージャーが開始した時に、値が取り込まれる。		
DB2_XBSA_LIBRARY	AIX、HP-UX、Sun Solaris、および Windows NT	デフォルト = null 値: 任意の有効なパスおよびファイル。

表 30. その他の変数 (続き)

変数名	オペレーティング・システム	値
説明		

ベンダーの提供する XBSA ライブラリーを示す。 AIX で、共有オブジェクトが shr.o という名前でない場合は、設定にそのオブジェクトを組み込む必要がある。 HP-UX、Solaris、Windows NT では、共有オブジェクト名は必要ない。たとえば、Legato's NetWorker Business Suite Module for DB2 を使用するには、レジストリー変数を次のように設定する。
`db2set DB2_XSBA_LIBRARY="/usr/lib/libxdb2.a(bsashr10.o)"`
XBSA インターフェースは、BACKUP DATABASE または RESTORE DATABASE コマンドから呼び出すことができる。たとえば:
`db2 backup db sample use XBSA`
`db2 restore db sample use XBSA`		

付録B. Explain 表と定義

Explain 表は、Explain 機能起動時のアクセス・プランを捕らえたものです。このセクションでは、次の Explain 表および定義が説明されています。

Explain 表は、Explain 機能を呼び出す前に作成しておく必要があります。 'sqllib' ディレクトリーの 'misc' サブディレクトリーにある EXPLAIN.DDL ファイルに備えられている、コマンド行プロセッサー入力スクリプトの例を使用してください。 Explain 表が要求されているデータベースに接続します。それからコマンド `db2 -tf EXPLAIN.DDL` を発行すると表が作成されます。詳細については、561ページの『Explain 表の表定義』を参照してください。

Explain 機能によって Explain 表を移植しても、トリガーを活動化されたり、参照制約や検査制約を活動化されることはありません。たとえば、挿入トリガーが EXPLAIN_INSTANCE 表に定義されており、かつ適切なステートメントが Explain されているとしても、トリガーは活動化されません。

Explain 機能の詳細については、211ページの『第7章 SQL Explain 機能』を参照してください。

Explain 表の凡例:

見出し	説明
列名	列の名前
データ・タイプ	列のデータ・タイプ
ヌル可能?	○: ヌルは許される ×: ヌルは許されない

キー? PK: この列は基本キーの一部
FK: この列は外部キーの一部
説明 列の説明

EXPLAIN_ARGUMENT 表

EXPLAIN_ARGUMENT 表は、個々の演算子に固有の特性がある場合、それを示します。

表 31. EXPLAIN_ARGUMENT 表

列名	データ・タイプ	ヌル可能?	キー?	説明
EXPLAIN_REQUESTER	VARCHAR(128)	×	FK	この Explain 要求を開始した許可 ID。
EXPLAIN_TIME	TIMESTAMP	×	FK	Explain 要求の開始時刻。
SOURCE_NAME	VARCHAR(128)	×	FK	動的ステートメントに Explain 要求を出したときに実行していたパッケージの名前、または静的 SQL に Explain 要求を出したときのソース・ファイルの名前。
SOURCE_SCHEMA	VARCHAR(128)	×	FK	Explain 要求のソースのスキーマ、または修飾子。
EXPLAIN_LEVEL	CHAR(1)	×	FK	この行に関連する Explain 情報のレベル。
STMTNO	INTEGER	×	FK	パッケージ内のステートメントのうち、この Explain 情報に関連するもののステートメント番号。
SECTNO	INTEGER	×	FK	この Explain 情報に関連するパッケージのセクション番号。
OPERATOR_ID	INTEGER	×	×	この照会内の演算子の固有の ID。
ARGUMENT_TYPE	CHAR(8)	×	×	この演算子の引き数のタイプ。
ARGUMENT_VALUE	VARCHAR(1024)	○	×	この演算子の引き数の値。値が LONG_ARGUMENT_VALUE にある場合は NULL。
LONG_ARGUMENT_VALUE	CLOB(1M)	○	×	この演算子の引き数の値。この場合、テキストは ARGUMENT_VALUE に収まりません。値が ARGUMENT_VALUE にある場合は NULL。

表 32. ARGUMENT_TYPE および ARGUMENT_VALUE 列値

ARGUMENT_TYPE 値	可能な ARGUMENT_VALUE 値	説明
AGGMODE	COMPLETE PARTIAL INTERMEDIATE FINAL	部分集約標識。
BITFLTR	TRUE FALSE	パフォーマンスを向上するために、ハッシュ結合はビット・フィルターを使用する。
CSETEMP	TRUE FALSE	共通副次式上の一時表フラグ
DIRECT	TRUE	直接取り出し標識
DUPLWARN	TRUE FALSE	重複警告標識

表 32. ARGUMENT_TYPE および ARGUMENT_VALUE 列値 (続き)

ARGUMENT_TYPE 値	可能な ARGUMENT_VALUE 値	説明
EARLYOUT	TRUE FALSE	初期アウト標識
ENVVAR	このタイプの行にはそれぞれ次のものが含まれます。 • 環境変数名 • 環境変数値	最適化プログラムに影響する環境変数
FETCHMAX	IGNORE INTEGER	FETCH 演算子の MAXPAGES 引き数の値を指定変更する。
GROUPBYC	TRUE FALSE	Group By 列が与えられているかどうか。
GROUPBYN	INTEGER	比較列の数
GROUPBYR	このタイプの行にはそれぞれ次のものが含まれます。 • group by 文節にある列の序数値 (その後にコロンとスペースがつづく) • 列の名前	Group By 要件
INNERCOL	このタイプの行にはそれぞれ次のものが含まれます。 • 配列された列の序数値 (その後にコロンとスペースがつづく) • 列の名前 • 順序値 **(A)** 昇順 **(D)** 降順	内部順序列
ISCANMAX	IGNORE INTEGER	ISCAN 演算子の MAXPAGES 引き数の値を指定変更する。
JN_INPUT	INNER OUTER	演算子が内部または外部のどちらの結合を送る演算子であるかを示す。
LISTENER	TRUE FALSE	Listener 表キューの標識。
MAXPAGES	ALL NONE INTEGER	事前取り出しに予期される最大ページ。
MAXRIDS	NONE INTEGER	リスト事前取り出し要求ごとに含まれる最大行 ID。
NUMROWS	INTEGER	分類されることになっている行数。
ONEFETCH	TRUE FALSE	1 つの取り出し標識。

表 32. ARGUMENT_TYPE および ARGUMENT_VALUE 列値 (続き)

ARGUMENT_TYPE 値	可能な ARGUMENT_VALUE 値	説明
OUTERCOL	このタイプの行にはそれぞれ次のものが含まれます。 • 配列された列の序数値 (その後にコロンとスペースがつづく) • 列の名前 • 順序値 **(A)** 昇順 **(D)** 降順	外部順序列
OUTERJN	LEFT RIGHT	外部結合の標識。
PARTCOLS	列の名前	演算子の区分化列。
PREFETCH	LIST NONE SEQUENTIAL	適格な事前取り出しのタイプ。
RMTQTEXT	照会テキスト	リモート照会テキスト
ROWLOCK	EXCLUSIVE NONE REUSE SHARE SHORT (INSTANT) SHARE UPDATE	行ロック意図。
ROWWIDTH	INTEGER	分類される行の幅。
SCANDIR	FORWARD REVERSE	走査方向。
SCANGRAN	INTEGER	区画内並列性、区画内並列性の走査の細分性。SCANUNIT の単位で表される。
SCANTYPE	LOCAL PARALLEL	区画内並列性、索引または表の走査。
SCANUNIT	ROW PAGE	区画内並列性、走査の細分性の単位。
SERVER	リモート・サーバー	リモート・サーバー
SHARED	TRUE	区画内並列性、共用 TEMP 標識。
SLOWMAT	TRUE FALSE	具体化遅延フラグ。
SNGLPROD	TRUE FALSE	単一エージェントによって生成される区画内並列性分類、または TEMP。

表 32. ARGUMENT_TYPE および ARGUMENT_VALUE 列値 (続き)

ARGUMENT_TYPE 値	可能な ARGUMENT_VALUE 値	説明
SORTKEY	このタイプの行にはそれぞれ次のものが含まれます。 • キー内の列の順序値 (後に、コロンとスペースが続く) • 列の名前 • 順序値 **(A)** 昇順 **(D)** 降順	分類キー列
SORTTYPE	PARTITIONED SHARED ROUND ROBIN REPLICATED	区画内並列性、分類タイプ。
TABLOCK	EXCLUSIVE INTENT EXCLUSIVE INTENT NONE INTENT SHARE REUSE SHARE SHARE INTENT EXCLUSIVE SUPER EXCLUSIVE UPDATE	表ロック意図。
TQDEGREE	INTEGER	区画内並列性、表キューにアクセスするサブエージェントの数。
TQMERGE	TRUE FALSE	マージする (分類済み) 表キューの標識。
TQREAD	READ AHEAD STEPPING SUBQUERY STEPPING	表キューの読み取り特性。
TQSEND	BROADCAST DIRECTED SCATTER SUBQUERY DIRECTED	表キューの送信特性。
TQTYPE	LOCAL	区画内並列性、表キュー。
TRUNCSRT	TRUE	切り捨て分類 (限定数の行が作成される)。
UNIQUE	TRUE FALSE	固有性標識
UNIQKEY	このタイプの行にはそれぞれ次のものが含まれます。 • キー内の列の順序値 (後に、コロンとスペースが続く) • 列の名前	固有のキー列
VOLATILE	TRUE	揮発性表

EXPLAIN_INSTANCE 表

EXPLAIN_INSTANCE 表は、すべての Explain 情報用の主制御表です。 Explain 表中のデータの各行は、この表内のある固有の 1 行に明示的にリンクされます。
EXPLAIN_INSTANCE 表は、Explain 対象の SQL ステートメントのソースに関する基本情報、および Explain 機能の環境に関する情報を提供します。

この表の定義については、563ページの『EXPLAIN_INSTANCE 表の定義』を参照してください。

表 33. EXPLAIN_INSTANCE 表

列名	データ・タイプ	ヌル可能?	キー?	説明
EXPLAIN_REQUESTER	VARCHAR(128)	×	PK	この Explain 要求を開始した許可 ID。
EXPLAIN_TIME	TIMESTAMP	×	PK	Explain 要求の開始時刻。
SOURCE_NAME	VARCHAR(128)	×	PK	動的ステートメントに Explain 要求を出したときに実行していたパッケージの名前、または静的 SQL に Explain 要求を出したときのソース・ファイルの名前。
SOURCE_SCHEMA	VARCHAR(128)	×	PK	Explain 要求のソースのスキーマ、または修飾子。
EXPLAIN_OPTION	CHAR(1)	×	×	この要求に関して要求された Explain 情報の内容を示します。 可能な値は以下のとおりです。 **P** PLAN SELECTION
SNAPSHOT_TAKEN	CHAR(1)	×	×	この要求に関して Explain スナップショットが取られたかどうかを示します。 可能な値は以下のとおりです。 **Y** はい。1 つまたは複数の Explain スナップショットが取られ、 EXPLAIN_STATEMENT 表に保管されました。正規の Explain 情報もキャプチャーされました。 **N** Explain スナップショットは取られていません。正規の Explain 情報がキャプチャーされました。 **O** Explain スナップショットだけが取られました。正規の Explain 情報もキャプチャーされました。
DB2_VERSION	CHAR(7)	×	×	この Explain 要求を処理した DB2 ユニバーサル・データベースの製品リリース番号。形式は vv.rr.m です。 **vv** バージョン番号 **rr** リリース番号 **m** 保守リリース番号
SQL_TYPE	CHAR(1)	×	×	Explain インスタンスが静的と動的のどちらの SQL に関するものであったかどうかを示します。 可能な値は以下のとおりです。 **S** 静的 SQL **D** 動的 SQL

表 33. EXPLAIN_INSTANCE 表 (続き)

列名	データ・タイプ	ヌル可能?	キー?	説明
QUERYOPT	INTEGER	×	×	Explain の呼び出しの時点で SQL コンパイラーにより使用される照会最適化クラスを示します。値は、Explain 中の SQL ステートメントについて SQL コンパイラーが実行するのは、どのレベルの照会最適化であるかを示します。
BLOCK	CHAR(1)	×	×	SQL ステートメントのコンパイル時に使用されるのは、どのタイプのカーソル・ブロック化であるかを示します。詳細は、SYSCAT.PACKAGES 内の BLOCK 列を参照してください。 可能な値は以下のとおりです。 **N** ブロック化なし **U** ブロック確定カーソル **B** 全カーソルのブロック
ISOLATION	CHAR(2)	×	×	SQL ステートメントのコンパイル時に使用されるのは、どのタイプの分離機能であるかを示します。詳細は、SYSCAT.PACKAGES 内の ISOLATION 列を参照してください。 可能な値は以下のとおりです。 **RR** 反復可能読み取り **RS** 読み取り固定 **CS** カーソル固定 **UR** 非コミット読み取り
BUFFPAGE	INTEGER	×	×	Explain の呼び出しの時点で設定された BUFFPAGE データベース構成の値が含まれています。
AVG_APPLS	INTEGER	×	×	Explain 呼び出し時に、 AVG_APPLS データベース構成パラメーターの値が入れられます。
SORTHEAP	INTEGER	×	×	Explain の呼び出しの時点で設定された SORTHEAP データベース構成の値が含まれています。
LOCKLIST	INTEGER	×	×	Explain の呼び出しの時点で設定された LOCKLIST データベース構成の値が含まれています。
MAXLOCKS	SMALLINT	×	×	Explain の呼び出しの時点で設定された MAXLOCKS データベース構成の値が含まれています。
LOCKS_AVAIL	INTEGER	×	×	最適化プログラムによって各ユーザーごとに使用可能と見なされるロックの数が含まれています。(LOCKLIST および MAXLOCKS から派生したもの)。
CPU_SPEED	DOUBLE	×	×	Explain の呼び出しの時点で設定された CPUSPEED データベース・マネージャー構成設定の値が含まれています。
REMARKS	VARCHAR(254)	○	×	ユーザー提供の注釈。
DBHEAP	INTEGER	×	×	Explain 呼び出し時に DBHEAP データベース構成設定値が入れられます。
COMM_SPEED	DOUBLE	×	×	Explain 呼び出し時に COMM_BANDWIDTH データベース構成設定値が入れられます。

表 33. EXPLAIN_INSTANCE 表 (続き)

列名	データ・タイプ	ヌル可能?	キー?	説明
PARALLELISM	CHAR(2)	×	×	可能な値は以下のとおりです。 • N = 並列性なし • P = 区画内並列性 • IP = 区画間並列性 • BP = 区画内並列性と区画間並列性
DATAJOINER	CHAR(1)	×	×	可能な値は以下のとおりです。 • N = 非連合システム・プラン • Y = 連合システム・プラン

EXPLAIN_OBJECT 表

EXPLAIN_OBJECT 表は、 SQL ステートメントを満たすために生成されるアクセス・プランが必要とするデータ・オブジェクトを指定します。

表 34. EXPLAIN_OBJECT 表

列名	データ・タイプ	ヌル可能?	キー?	説明
EXPLAIN_REQUESTER	VARCHAR(128)	×	FK	この Explain 要求を開始した許可 ID。
EXPLAIN_TIME	TIMESTAMP	×	FK	Explain 要求の開始時刻。
SOURCE_NAME	VARCHAR(128)	×	FK	動的ステートメントに Explain 要求を出したときに実行していたパッケージの名前、または静的 SQL に Explain 要求を出したときのソース・ファイルの名前。
SOURCE_SCHEMA	VARCHAR(128)	×	FK	Explain 要求のソースのスキーマ、または修飾子。
EXPLAIN_LEVEL	CHAR(1)	×	FK	この行に関連する Explain 情報のレベル。
STMTNO	INTEGER	×	FK	パッケージ内のステートメントのうち、この Explain 情報に関連するもののステートメント番号。
SECTNO	INTEGER	×	FK	この Explain 情報に関連するパッケージのセクション番号。
OBJECT_SCHEMA	VARCHAR(128)	×	×	このオブジェクトが属しているスキーマ。
OBJECT_NAME	VARCHAR(128)	×	×	オブジェクトの名前。
OBJECT_TYPE	CHAR(2)	×	×	オブジェクトのタイプに関する記述ラベル。
CREATE_TIME	TIMESTAMP	○	×	オブジェクトの作成時刻。表関数の場合はヌル。
STATISTICS_TIME	TIMESTAMP	○	×	このオブジェクトに関する統計の最後の更新時刻。このオブジェクトに関する統計が存在しない場合は、ヌルです。
COLUMN_COUNT	SMALLINT	×	×	このオブジェクトの列数。
ROW_COUNT	INTEGER	×	×	このオブジェクト内の見積行数。
WIDTH	INTEGER	×	×	オブジェクトの平均幅 (バイト単位)。索引の場合は -1 に設定します。
PAGES	INTEGER	×	×	オブジェクトがバッファー・プールで占有するページ数の見積もり。表関数の場合は -1 に設定します。

表 34. EXPLAIN_OBJECT 表 (続き)

列名	データ・タイプ	ヌル可能?	キー?	説明
DISTINCT	CHAR(1)	×	×	オブジェクト内の行が固有 (つまり、重複なし) かどうかを示します。 可能な値は以下のとおりです。 **Y** ○ **N** ×
TABLESPACE_NAME	VARCHAR(128)	○	×	このオブジェクトが保管される表スペースの名前。表スペースが関係しない場合は、ヌルです。
OVERHEAD	DOUBLE	×	×	指定された表スペースにランダム入出力を 1 回行うためのオーバーヘッドの見積合計 (ミリ秒単位)。コントローラー・オーバーヘッド、ディスク・シーク、および待ち時間が含まれます。表スペースが関係しない場合は -1 に設定します。
TRANSFER_RATE	DOUBLE	×	×	指定の表スペースからデータ・ページを読み取るための時間の見積もり (ミリ秒単位)。表スペースが関係しない場合は -1 に設定します。
PREFETCHSIZE	INTEGER	×	×	事前取り出しの実行時に読み取るデータ・ページの数。表関数の場合は -1 に設定します。
EXTENTSIZE	INTEGER	×	×	データ・ページを単位とするエクステント・サイズ。この数のページが、次のコンテナーにスイッチする前に、表スペース内の 1 つのコンテナーに書き込まれます。表関数の場合は -1 に設定します。
CLUSTER	DOUBLE	×	×	索引とのデータ・クラスター化の程度。 >= 1 の場合は CLUSTERRATIO。 0 以上かつ 1 より小さい場合、これは CLUSTERFACTOR。表、表関数、またはこの統計が利用不能の場合は -1 に設定します。
NLEAF	INTEGER	×	×	この索引オブジェクトの値が占める葉ページの数。表、表関数、またはこの統計が利用不能の場合は -1 に設定します。
NLEVELS	INTEGER	×	×	この索引オブジェクトのツリーでの索引レベルの番号。表、表関数、またはこの統計が利用不能の場合は -1 に設定します。
FULLKEYCARD	BIGINT	×	×	この索引オブジェクトに含まれる個別フル・キー値の数。表、表関数、またはこの統計が利用不能の場合は -1 に設定します。
OVERFLOW	INTEGER	×	×	表の中にあるオーバーフロー・レコードの合計数。索引、表関数、または統計が利用不能の場合は -1 に設定します。
FIRSTKEYCARD	BIGINT	×	×	第 1 キーの値の数。表、表関数、またはこの統計が利用不能の場合は -1 に設定します。
FIRST2KEYCARD	BIGINT	×	×	索引の最初の {2,3,4} 列を使用する第 1 キーの値の数。表、表関数、またはこの統計が利用不能の場合は -1 に設定します。
FIRST3KEYCARD	BIGINT	×	×	
FIRST4KEYCARD	BIGINT	×	×	

表 34. EXPLAIN_OBJECT 表 (続き)

列名	データ・タイプ	ヌル可能?	キー?	説明
SEQUENTIAL_PAGES	INTEGER	×	×	索引キー順にディスク上には位置されており、間に大きなギャップが少ない、またはない葉ページの数。表、表関数、またはこの統計が利用不能の場合は -1 に設定します。
DENSITY	INTEGER	×	×	索引によって占有されるページの範囲内のページ数に対する、 SEQUENTIAL_PAGES の比率。パーセンテージで表されます (0 ～ 100 の整数)。表、表関数、またはこの統計が利用不能の場合は -1 に設定します。

表 35. 可能な OBJECT_TYPE 値

値	説明
IX	索引
TA	表
TF	表関数

EXPLAIN_OPERATOR 表

EXPLAIN_OPERATOR 表は、 SQL コンパイラーが SQL ステートメントを満たすために必要とするすべての演算子を格納します。

表 36. EXPLAIN_OPERATOR 表

列名	データ・タイプ	ヌル可能?	キー?	説明
EXPLAIN_REQUESTER	VARCHAR(128)	×	FK	この Explain 要求を開始した許可 ID。
EXPLAIN_TIME	TIMESTAMP	×	FK	Explain 要求の開始時刻。
SOURCE_NAME	VARCHAR(128)	×	FK	動的ステートメントに Explain 要求を出したときに実行していたパッケージの名前、または静的 SQL に Explain 要求を出したときのソース・ファイルの名前。
SOURCE_SCHEMA	VARCHAR(128)	×	FK	Explain 要求のソースのスキーマ、または修飾子。
EXPLAIN_LEVEL	CHAR(1)	×	FK	この行に関連する Explain 情報のレベル。
STMTNO	INTEGER	×	FK	パッケージ内のステートメントのうち、この Explain 情報に関連するもののステートメント番号。
SECTNO	INTEGER	×	FK	この Explain 情報に関連するパッケージのセクション番号。
OPERATOR_ID	INTEGER	×	×	この照会内の演算子の固有の ID。
OPERATOR_TYPE	CHAR(6)	×	×	演算子のタイプの記述ラベル。
TOTAL_COST	DOUBLE	×	×	この演算子に至るまで (この演算子を含む) の、選択したアクセス・プランの実行にかかる合計コスト (タイマーオン単位) の累積の見積もり。
IO_COST	DOUBLE	×	×	この演算子に至るまで (この演算子を含む) の、選択したアクセス・プランの実行にかかる入出力コスト (データ・ページの入出力単位) の累積の見積もり。

表 36. EXPLAIN_OPERATOR 表 (続き)

列名	データ・タイプ	ヌル可能?	キー?	説明
CPU_COST	DOUBLE	×	×	選択したアクセス・プランを実行するとき、またこの演算子を含めるときにかかる CPU コスト（命令数）の累積の見積もり。
FIRST_ROW_COST	DOUBLE	×	×	アクセス・プランへの 1 行目を取り出すとき、またこの演算子を含めるときにかかる累積合計（タイマーオン数）の見積もり。この値には、必須の初期オーバーヘッドが含まれます。
RE_TOTAL_COST	DOUBLE	×	×	この演算子に至るまで (この演算子を含む) の、選択したアクセス・プランの次の行の取り出しにかかるコスト (タイマーオン単位) の累積の見積もり。
RE_IO_COST	DOUBLE	×	×	この演算子に至るまで (この演算子を含む) の、選択したアクセス・プランの次の行の取り出しにかかる入出力コスト (データ・ページの入出力単位) の累積の見積もり。
RE_CPU_COST	DOUBLE	×	×	この演算子に至るまで (この演算子を含む) の、選択したアクセス・プランの次の行の取り出しにかかる CPU コスト (タイマーオン単位) の累積の見積もり。
COMM_COST	DOUBLE	×	×	この演算子に至るまで (この演算子を含む) の、選択したアクセス・プランの実行にかかる通信コスト (TCP/IP フレーム単位) の累積の見積もり。
FIRST_COMM_COST	DOUBLE	×	×	この演算子に至るまで (この演算子を含む) の、選択したアクセス・プランの最初の行の取り出しにかかる通信コスト (TCP/IP フレーム単位) の累積の見積もり。この値には、必須の初期オーバーヘッドが含まれます。
BUFFERS	DOUBLE	×	×	この演算子とその入力に必要なバッファーの見積もり。
REMOTE_TOTAL_COST	DOUBLE	×	×	リモート・データベース操作の実行にかかる合計コスト (タイマーオン単位) の累積の見積もり。
REMOTE_COMM_COST	DOUBLE	×	×	この演算子に至るまで (この演算子を含む) の、選択したリモート・アクセス・プランの実行にかかる通信コストの累積の見積もり。

表 37. OPERATOR_TYPE 値

値	説明
DELETE	削除
FETCH	取り出し
FILTER	行フィルター
GENROW	生成行
GRPBY	グループ化
HSJOIN	ハッシュ結合
INSERT	挿入
IXAND	動的ビットマップ索引 AND
IXSCAN	索引走査
MSJOIN	マージ走査結合
NLJOIN	ネストしたループ結合
RETURN	結果

表 37. OPERATOR_TYPE 値 (続き)

値	説明
RIDSCN	行 ID (RID) 走査
RQUERY	リモート照会
SORT	分類
TBSCAN	表走査
TEMP	一時表構造
TQ	表キュー
UNION	合併
UNIQUE	重複除去
UPDATE	更新

EXPLAIN_PREDICATE 表

EXPLAIN_PREDICATE は、特定の演算子によって適用される述部を指定します。

表 38. EXPLAIN_PREDICATE 表

列名	データ・タイプ	ヌル可能?	キー?	説明
EXPLAIN_REQUESTER	VARCHAR(128)	×	FK	この Explain 要求を開始した許可 ID。
EXPLAIN_TIME	TIMESTAMP	×	FK	Explain 要求の開始時刻。
SOURCE_NAME	VARCHAR(128)	×	FK	動的ステートメントに Explain 要求を出したときに実行していたパッケージの名前、または静的 SQL に Explain 要求を出したときのソース・ファイルの名前。
SOURCE_SCHEMA	VARCHAR(128)	×	FK	Explain 要求のソースのスキーマ、または修飾子。
EXPLAIN_LEVEL	CHAR(1)	×	FK	この行に関連する Explain 情報のレベル。
STMTNO	INTEGER	×	FK	パッケージ内のステートメントのうち、この Explain 情報に関連するもののステートメント番号。
SECTNO	INTEGER	×	FK	この Explain 情報に関連するパッケージのセクション番号。
OPERATOR_ID	INTEGER	×	×	この照会内の演算子の固有の ID。
PREDICATE_ID	INTEGER	×	×	特定の演算子のための述部の固有の ID。
HOW_APPLIED	CHAR(5)	×	×	特定の演算子によって述部がどのように使用されているか。

表 38. EXPLAIN_PREDICATE 表 (続き)

列名	データ・タイプ	ヌル可能?	キー?	説明
WHEN_EVALUATED	CHAR(3)	×	×	この述部で使用される副照会をいつ評価するかの指示。 可能な値は以下のとおりです。 **ブランク** この述部には副照会がありません。 **EAA** この述部で使用される副照会は適用時に評価されます (EAA)。つまり、指定の演算子によって行が処理されるごとに、述部が適用されるので副照会が再評価されます。 **EAO** この述部で使用される副照会はオープン時に評価されます (EAO)。つまり、指定の演算子に対して副照会は 1 回だけ再評価され、結果はそれぞれの行へ述部を適用する際に再利用されます。 **MUL** この述部の副照会のタイプは複数あります。
RELOP_TYPE	CHAR(2)	×	×	この述部で使用される関係演算子のタイプ。
SUBQUERY	CHAR(1)	×	×	この述部に対して、副照会からのデータ・ストリームが要求されているか。複数の副照会ストリームが必要な場合があります。 可能な値は以下のとおりです。 **N** 副照会ストリームは不要 **Y** 1 つ以上の副照会ストリームが必要
FILTER_FACTOR	DOUBLE	×	×	この述部が修飾する小数部の行数の見積もり。
PREDICATE_TEXT	CLOB(1M)	○	×	SQL ステートメントの内部表示から再作成された述部のテキスト。 利用不能の場合はヌル。

表 39. 可能な HOW_APPLIED 値

値	説明
JOIN	以前は表を結合した。
RESID	残余述部として評価された。
SARG	索引またはデータ・ページの検索引き数述部として評価された。
START	開始条件として使用された。
STOP	停止条件として使用された。

表 40. 可能な RELOP_TYPE 値

値	説明
ブランク	適用しない
EQ	等しい
GE	それ以上または等しい

表 40. 可能な RELOP_TYPE 値 (続き)

値	説明
GT	それ以上
IN	リストされている
LE	それ以下または等しい
LK	類似
LT	それ以下
NE	等しくない
NL	ヌル
NN	ヌル以外

EXPLAIN_STATEMENT 表

EXPLAIN_STATEMENT 表には、さまざまなレベルの Explain 情報に関する SQL ステートメントのテキストが含まれます。この表には、ユーザーが入力した元の SQL ステートメントと、その SQL ステートメントを満たすアクセス・プランを選択するのに (最適化プログラムで) 使用されるバージョンとが保管されます。後のバージョンは、書き直されているか、 SQL コンパイラーで判別された追加の述部によって拡張されているか、またはその両方であるので、元のバージョンとはあまり類似していないことがあります。

この表の定義については、567ページの『EXPLAIN_STATEMENT 表の定義』を参照してください。

表 41. EXPLAIN_STATEMENT 表

列名	データ・タイプ	ヌル可能?	キー?	説明
EXPLAIN_REQUESTER	VARCHAR(128)	×	PK、FK	この Explain 要求を開始した許可 ID。
EXPLAIN_TIME	TIMESTAMP	×	PK、FK	Explain 要求の開始時刻。
SOURCE_NAME	VARCHAR(128)	×	PK、FK	動的ステートメントに Explain 要求を出したときに実行していたパッケージの名前、または静的 SQL に Explain 要求を出したときのソース・ファイルの名前。
SOURCE_SCHEMA	VARCHAR(128)	×	PK、FK	Explain 要求のソースのスキーマ、または修飾子。
EXPLAIN_LEVEL	CHAR(1)	×	PK	この行に関連する Explain 情報のレベル。 有効な値は次のとおりです。 **O** 元のテキスト (ユーザー入力) **P** PLAN SELECTION
STMTNO	INTEGER	×	PK	パッケージ内のステートメントのうち、この Explain 情報に関連するもののステートメント番号。動的 Explain SQL ステートメントの場合は 1。静的 SQL ステートメントの場合、この値は SYSCAT.STATEMENTS カタログ視点で使用されているものと同じ。

表 41. EXPLAIN_STATEMENT 表 (続き)

列名	データ・タイプ	ヌル可能?	キー?	説明
SECTNO	INTEGER	×	PK	パッケージ内のセクションのうち、この SQL ステートメントを含むもののセクション番号。動的 Explain SQL ステートメントの場合、これは、実行時にこのステートメントのセクションを保持するのに使用されるセクション番号。静的 SQL ステートメントの場合、この値は SYSCAT.STATEMENTS カタログ視点で使用されているものと同じ。
QUERYNO	INTEGER	×	×	Explain 対象の SQL ステートメントの数値 ID。 CLP または CLI を介して発行された動的 SQL ステートメントの場合 (EXPLAIN SQL ステートメントを除く)、デフォルト値は 1 ずつ順番に大きくなる値です。そうでない場合、デフォルト値は静的 SQL ステートメントでは STMTNO の値で、動的 SQL ステートメントでは 1 です。
QUERYTAG	CHAR(20)	×	×	Explain 対象の個々の SQL ステートメントの ID タグ。 CLP を介して発行された動的 SQL ステートメントの場合 (EXPLAIN SQL ステートメントは除く)、デフォルト値は 'CLP' です。 CLI を介して発行された動的 SQL ステートメントの場合 (EXPLAIN SQL ステートメントは除く)、デフォルト値は 'CLI' です。そうでない場合は、デフォルト値にはブランクを使用します。
STATEMENT_TYPE	CHAR(2)	×	×	Explain 対象の照会のタイプに関する記述ラベル 可能な値は以下のとおりです。 **S** 選択 **D** 削除 **DC** カーソルの現在位置の削除 **I** 挿入 **U** 更新 **UC** カーソルの現在位置の更新
UPDATABLE	CHAR(1)	×	×	このステートメントが更新可能であると見なされるかどうかを示します。これは特に、潜在的に更新可能であると見なされることのある SELECT ステートメントに関係しています。 可能な値は以下のとおりです。 ' ' 適用不能 (ブランク) **N** × **Y** ○
DELETABLE	CHAR(1)	×	×	このステートメントが更新可能であると見なされるかどうかを示します。これは特に、潜在的に削除可能であると見なされることのある SELECT ステートメントに関係しています。 可能な値は以下のとおりです。 ' ' 適用不能 (ブランク) **N** × **Y** ○

表 41. EXPLAIN_STATEMENT 表 (続き)

列名	データ・タイプ	ヌル可能?	キー?	説明
TOTAL_COST	DOUBLE	×	×	このステートメントについて選択されたアクセス・プランの実行のための合計コストの見積もり (タイマーオン単位)。 EXPLAIN_LEVEL が *O* (オリジナル・テキスト) の場合は、この時点で選択されているアクセス・プランがないため、ゼロに設定されます。
STATEMENT_TEXT	CLOB(1M)	×	×	Explain 対象の SQL ステートメントのテキスト、またはその一部。 Explain のプラン選択レベル用に示されたテキストは、内部表示から再構成されており、実際は SQL に類似しています。つまり、再構成されたステートメントは、正しい SQL 構文に従うことは保証されていません。
SNAPSHOT	BLOB(10M)	○	×	示されている Explain_Level での、この SQL ステートメントの内部表示のスナップショット。 この列は、DB2 Visual Explain 製品での使用を意図しています。 EXPLAIN_LEVEL が *0* (オリジナル・テキスト) の場合は、ステートメントのこの特定バージョンが取り込まれた時点でアクセス・プランが選択されていないため、この列はヌルに設定されます。
QUERY_DEGREE	INTEGER	×	×	Explain の呼び出し時の区画内並列性の度数。元のステートメントの場合、ここには区画内並列性の指定された度数が入ります。 PLAN SELECTION の場合、ここには使用のプランに応じて生成された区画内並列性の度数が入ります。

EXPLAIN_STREAM 表

EXPLAIN_STREAM 表は、個々の演算子とデータ・オブジェクトの間の、入出力データ・ストリームを表します。データ・オブジェクト自体は、EXPLAIN_OBJECT 表に示されています。データ・ストリームに関連する演算子は、EXPLAIN_OPERATOR 表にあります。

表 42. EXPLAIN_STREAM 表

列名	データ・タイプ	ヌル可能?	キー?	説明
EXPLAIN_REQUESTER	VARCHAR(128)	×	FK	この Explain 要求を開始した許可 ID。
EXPLAIN_TIME	TIMESTAMP	×	FK	Explain 要求の開始時刻。
SOURCE_NAME	VARCHAR(128)	×	FK	動的ステートメントに Explain 要求を出したときに実行していたパッケージの名前、または静的 SQL に Explain 要求を出したときのソース・ファイルの名前。
SOURCE_SCHEMA	VARCHAR(128)	×	FK	Explain 要求のソースのスキーマ、または修飾子。
EXPLAIN_LEVEL	CHAR(1)	×	FK	この行に関連する Explain 情報のレベル。
STMTNO	INTEGER	×	FK	パッケージ内のステートメントのうち、この Explain 情報に関連するもののステートメント番号。

表 42. EXPLAIN_STREAM 表 (続き)

列名	データ・タイプ	ヌル可能?	キー?	説明
SECTNO	INTEGER	×	FK	この Explain 情報に関連するパッケージのセクション番号。
STREAM_ID	INTEGER	×	×	指定演算子内のこのデータ・ストリームの固有 ID。
SOURCE_TYPE	CHAR(1)	×	×	このデータ・ストリームのソースを示します。 **O** 演算子 **D** データ・オブジェクト
SOURCE_ID	SMALLINT	×	×	このデータ・ストリームのソースである照会内の、演算子に対する固有 ID。 SOURCE_TYPE が「D」の場合は -1 に設定されます。
TARGET_TYPE	CHAR(1)	×	×	このデータ・ストリームのターゲットを示します。 **O** 演算子 **D** データ・オブジェクト
TARGET_ID	SMALLINT	×	×	このデータ・ストリームのターゲットである照会内の、演算子に対する固有 ID。 TARGET_TYPE が「D」の場合は -1 に設定されます。
OBJECT_SCHEMA	VARCHAR(128)	○	×	影響を受けるデータ・オブジェクトが属するスキーマ。 SOURCE_TYPE および TARGET_TYPE が共に 'O' である場合、ヌルに設定します。
OBJECT_NAME	VARCHAR(128)	○	×	データ・ストリームのサブジェクトであるオブジェクトの名前。 SOURCE_TYPE および TARGET_TYPE が共に 'O' である場合、ヌルに設定します。
STREAM_COUNT	DOUBLE	×	×	データ・ストリームの見積カーディナリティー。
COLUMN_COUNT	SMALLINT	×	×	データ・ストリーム内の列数。
PREDICATE_ID	INTEGER	×	×	このストリームが述部の副照会の一部の場合は、述部 ID がこの値に反映されます。そうでない場合は、列は -1 に設定されます。
COLUMN_NAMES	CLOB(1M)	○	×	この列には、このストリームに関連した列の名前や順序情報が含まれています。 名前は次の形式です。 `NAME1(A)+NAME2(D)+NAME3+NAME4` ここで、*(A)* は昇順の列、 *(D)* は降順の列を示し、順序情報がないものは、列が順序づけられていないか、順序が関係ないかのいずれかを示します。
PMID	SMALLINT	×	×	区分化マップの ID。

表 42. EXPLAIN_STREAM 表 (続き)

列名	データ・タイプ	ヌル可能?	キー?	説明
SINGLE_NODE	CHAR(5)	○	×	このデータ・ストリームが単一または複数の区分にあるかどうかを示します。 **MULT** 複数の区分にある **COOR** 調整プログラム・ノードにある **HASH** ハッシュを使用して指定される **RID** 行 ID を使用して指定される **FUNC** 関数を使用して指定される (PARTITION() または NODENUMBER()) **CORR** 相関値を使用して指定される **Numberic** 事前判別された単一ノードに指定される
PARTITION_COLUMNS	CLOB(64K)	○	×	このデータ・ストリームが区分化される列のリスト。

ADVISE_INDEX 表

ADVISE_INDEX 表は、推奨索引を示しています。

表 43. ADVISE_INDEX 表

列名	データ・タイプ	ヌル可能?	キー?	説明
EXPLAIN_REQUESTER	VARCHAR(128)	×	×	この Explain 要求を開始した許可 ID。
EXPLAIN_TIME	TIMESTAMP	×	×	Explain 要求の開始時刻。
SOURCE_NAME	VARCHAR(128)	×	×	動的ステートメントに Explain 要求を出したときに実行していたパッケージの名前、または静的 SQL に Explain 要求を出したときのソース・ファイルの名前。
SOURCE_SCHEMA	VARCHAR(128)	×	×	Explain 要求のソースのスキーマ、または修飾子。
EXPLAIN_LEVEL	CHAR(1)	×	×	この行に関連する Explain 情報のレベル。
STMTNO	INTEGER	×	×	パッケージ内のステートメントのうち、この Explain 情報に関連するもののステートメント番号。
SECTNO	INTEGER	×	×	この Explain 情報に関連するパッケージのセクション番号。
QUERYNO	INTEGER	×	×	Explain 対象の SQL ステートメントの数値 ID。 CLP または CLI を介して発行された動的 SQL ステートメントの場合 (EXPLAIN SQL ステートメントを除く)、デフォルト値は 1 ずつ順番に大きくなる値です。そうでない場合、デフォルト値は静的 SQL ステートメントでは STMTNO の値で、動的 SQL ステートメントでは 1 です。

表 43. ADVISE_INDEX 表 (続き)

列名	データ・タイプ	ヌル可能?	キー?	説明
QUERYTAG	CHAR(20)	×	×	Explain 対象の個々の SQL ステートメントの ID タグ。CLP を介して発行された動的 SQL ステートメントの場合 (EXPLAIN SQL ステートメントは除く)、デフォルト値は 'CLP' です。 CLI を介して発行された動的 SQL ステートメントの場合 (EXPLAIN SQL ステートメントは除く)、デフォルト値は 'CLI' です。そうでない場合は、デフォルト値にはブランクを使用します。
NAME	VARCHAR(128)	×	×	索引の名前。
CREATOR	VARCHAR(128)	×	×	索引名の修飾子。
TBNAME	VARCHAR(128)	×	×	索引が定義されている表またはニックネームの名前。
TBCREATOR	VARCHAR(128)	×	×	表名の修飾子。
COLNAMES	CLOB(64K)	×	×	列名のリスト。
UNIQUERULE	CHAR(1)	×	×	固有値に関する規則。 D = 重複が許可される P = 1 次索引 U = 固有な項目だけが許可される
COLCOUNT	SMALLINT	×	×	キーの中の列数または (もしあれば) 組み込み列数。
IID	SMALLINT	×	×	索引の内部 ID。
NLEAF	INTEGER	×	×	葉ページの数。統計が収集されていない場合は -1。
NLEVELS	SMALLINT	×	×	索引レベルの数。統計が収集されていない場合は -1。
FULLKEYCARD	BIGINT	×	×	個別の全キー値の数。統計が収集されていない場合は -1。
FIRSTKEYCARD	BIGINT	×	×	第 1 キーの値の数。統計が収集されていない場合は -1。
CLUSTERRATIO	SMALLINT	×	×	索引によるデータ・クラスター化の程度。統計が収集されていない場合、または詳細な索引統計が収集されている場合は -1 (それらの場合は CLUSTERFACTOR の方が使用されます)。
CLUSTERFACTOR	DOUBLE	×	×	より高い計算精度のクラスター化。また、詳細索引統計を収集していない場合、あるいはニックネームに索引が定義されていない場合は -1 です。
USERDEFINED	SMALLINT	×	×	ユーザー定義。
SYSTEM_REQUIRED	SMALLINT	×	×	この索引が基本キー制約または固有キー制約に必要な場合、またはタイプ付き表のオブジェクト ID (OID) 列の場合は 1。 この索引が基本キー制約または固有キー制約に必要であり、しかもタイプ付き表のオブジェクト ID (OID) 列である場合は 2。 それ以外の場合は 0。
CREATE_TIME	TIMESTAMP	×	×	索引の作成時刻。
STATS_TIME	TIMESTAMP	○	×	この索引について記録されている統計値が最後に変更された時刻。統計が利用できない場合は、ヌル。

表 43. ADVISE_INDEX 表 (続き)

列名	データ・タイプ	ヌル可能?	キー?	説明
PAGE_FETCH_PAIRS	VARCHAR(254)	×	×	文字形式で表された、整数の対のリスト。個々の対は、仮バッファーを使用してこの索引による表走査を行うのに必要なページ・フェッチ回数を表すもの。 (データが利用できない場合は、長さ 0 のストリング。)
REMARKS	VARCHAR(254)	○	×	ユーザー提供の注釈、またはヌル。
DEFINER	VARCHAR(128)	×	×	索引を作成したユーザー。
CONVERTED	CHAR(1)	×	×	将来の利用のために予約済み。
SEQUENTIAL_PAGES	INTEGER	×	×	索引キー順にディスク上には位置されており、間に大きなギャップが少ない、またはない葉ページの数。 (統計が入手できない場合は -1。)
DENSITY	INTEGER	×	×	索引によって占有されるページの範囲内のページ数に対する、 SEQUENTIAL_PAGES の比率。 % で表されます (0 ～ 100 の整数、統計が入手できない場合は -1)。
FIRST2KEYCARD	BIGINT	×	×	索引の最初の 2 列を使用する個別キーの数 (統計がない場合、または適用されない場合は -1)。
FIRST3KEYCARD	BIGINT	×	×	索引の最初の 3 列を使用する個別キーの数 (統計がない場合、または適用されない場合は -1)。
FIRST4KEYCARD	BIGINT	×	×	索引の最初の 4 列を使用する個別キーの数 (統計がない場合、または適用されない場合は -1)。
PCTFREE	SMALLINT	×	×	索引を最初に作成する際に予約する索引のそれぞれのページのパーセンテージ。このスペースは、索引の作成後に行う挿入のために使用可能です。
UNIQUE_COLCOUNT	SMALLINT	×	×	固有キーに必要な列の数。常に <=COLCOUNT。列を含む場合のみ、< COLCOUNT。索引に固有キーがない場合は -1 (重複を許す)。
MINPCTUSED	SMALLINT	×	×	ゼロ以外の場合、オンライン索引再編成が使用可能になり、その値は、ページのマージ前に使用する最小スペースのしきい値となります。
REVERSE_SCANS	CHAR(1)	×	×	Y = 索引は逆方向走査をサポートする N = 索引は逆方向走査をサポートしない
USE_INDEX	CHAR(1)	○	×	Y = 推奨または評価された索引 N = 推奨されない索引
CREATION_TEXT	CLOB(1M)	×	×	索引を作成するのに使用する SQL ステートメント。
PACKED_DESC	BLOB(20M)	○	×	表の内部記述。

ADVISE_WORKLOAD 表

ADVISE_WORKLOAD 表は、作業負荷を構成するステートメントを示しています。作業負荷について詳しくは、 *管理の手引き: パフォーマンス* を参照してください。

表 44. ADVISE_WORKLOAD 表

列名	データ・タイプ	ヌル可能?	キー?	説明
WORKLOAD_NAME	CHAR(128)	×	×	このステートメントが属している SQL ステートメント (作業負荷) の集合の名前。
STATEMENT_NO	INTEGER	×	×	作業負荷内のステートメントのうち、この Explain 情報に関連するもののステートメント番号。
STATEMENT_TEXT	CLOB(1M)	×	×	SQL ステートメントの内容。
STATEMENT_TAG	VARCHAR(256)	×	×	Explain 対象の個々の SQL ステートメントの ID タグ。
FREQUENCY	INTEGER	×	×	作業負荷内でこのステートメントが使用される回数。
IMPORTANCE	DOUBLE	×	×	ステートメントの重要性。
COST_BEFORE	DOUBLE	○	×	推奨索引が作成されない場合の照会のコスト (タイマーオン)。
COST_AFTER	DOUBLE	○	×	推奨索引が作成される場合の照会のコスト (タイマーオン)。

Explain 表の表定義

Explain 表は、Explain 機能を呼び出す前に作成しておく必要があります。次の定義は、必要な Explain 表を作成する方法を指定するものです。

- 562ページの『EXPLAIN_ARGUMENT 表の定義』
- 563ページの『EXPLAIN_INSTANCE 表の定義』
- 564ページの『EXPLAIN_OBJECT 表の定義』
- 565ページの『EXPLAIN_OPERATOR 表の定義』
- 566ページの『EXPLAIN_PREDICATE 表の定義』
- 567ページの『EXPLAIN_STATEMENT 表の定義』
- 568ページの『EXPLAIN_STREAM 表の定義』
- 569ページの『ADVISE_INDEX 表の定義』
- 571ページの『ADVISE_WORKLOAD 表の定義』

あるいは、 'sqllib' ディレクトリーの 'misc' サブディレクトリーにある EXPLAIN.DDL ファイルに備えられている、コマンド行プロセッサーの入力スクリプトのサンプルを使用してください。 Explain 表が要求されているデータベースに接続します。それからコマンド `db2 -tf EXPLAIN.DDL` を発行すると表が作成されます。

EXPLAIN_ARGUMENT 表の定義

```
CREATE TABLE EXPLAIN_ARGUMENT ( EXPLAIN_REQUESTER   VARCHAR(128)  NOT NULL,
                                EXPLAIN_TIME        TIMESTAMP     NOT NULL,
                                SOURCE_NAME         VARCHAR(128)  NOT NULL,
                                SOURCE_SCHEMA       VARCHAR(128)  NOT NULL,
                                EXPLAIN_LEVEL       CHAR(1)       NOT NULL,
                                STMTNO              INTEGER       NOT NULL,
                                SECTNO              INTEGER       NOT NULL,
                                OPERATOR_ID         INTEGER       NOT NULL,
                                ARGUMENT_TYPE       CHAR(8)       NOT NULL,
                                ARGUMENT_VALUE      VARCHAR(1024) NOT NULL,
                                LONG_ARGUMENT_VALUE CLOB(1M)      NOT LOGGED,
                                   FOREIGN KEY (EXPLAIN_REQUESTER,
                                                EXPLAIN_TIME,
                                                SOURCE_NAME,
                                                SOURCE_SCHEMA,
                                                EXPLAIN_LEVEL,
                                                STMTNO,
                                                SECTNO)
                                   REFERENCES EXPLAIN_STATEMENT
                                   ON DELETE CASCADE )
```

EXPLAIN_INSTANCE 表の定義

```
CREATE TABLE EXPLAIN_INSTANCE ( EXPLAIN_REQUESTER VARCHAR(128) NOT NULL,
                                EXPLAIN_TIME      TIMESTAMP    NOT NULL,
                                SOURCE_NAME       VARCHAR(128) NOT NULL,
                                SOURCE_SCHEMA     VARCHAR(128) NOT NULL,
                                EXPLAIN_OPTION    CHAR(1)      NOT NULL,
                                SNAPSHOT_TAKEN    CHAR(1)      NOT NULL,
                                DB2_VERSION       CHAR(7)      NOT NULL,
                                SQL_TYPE          CHAR(1)      NOT NULL,
                                QUERYOPT          INTEGER      NOT NULL,
                                BLOCK             CHAR(1)      NOT NULL,
                                ISOLATION         CHAR(2)      NOT NULL,
                                BUFFPAGE          INTEGER      NOT NULL,
                                AVG_APPLS         INTEGER      NOT NULL,
                                SORTHEAP          INTEGER      NOT NULL,
                                LOCKLIST          INTEGER      NOT NULL,
                                MAXLOCKS          SMALLINT     NOT NULL,
                                LOCKS_AVAIL       INTEGER      NOT NULL,
                                CPU_SPEED         DOUBLE       NOT NULL,
                                REMARKS           VARCHAR(254),
                                DBHEAP            INTEGER      NOT NULL,
                                COMM_SPEED        DOUBLE       NOT NULL,
                                PARALLELISM       CHAR(2)      NOT NULL,
                                DATAJOINER        CHAR(1)      NOT NULL,
                                    PRIMARY KEY (EXPLAIN_REQUESTER,
                                                 EXPLAIN_TIME,
                                                 SOURCE_NAME,
                                                 SOURCE_SCHEMA))
```

EXPLAIN_OBJECT 表の定義

```
CREATE TABLE EXPLAIN_OBJECT ( EXPLAIN_REQUESTER VARCHAR(128) NOT NULL,
                              EXPLAIN_TIME      TIMESTAMP    NOT NULL,
                              SOURCE_NAME       VARCHAR(128) NOT NULL,
                              SOURCE_SCHEMA     VARCHAR(128) NOT NULL,
                              EXPLAIN_LEVEL     CHAR(1)      NOT NULL,
                              STMTNO            INTEGER      NOT NULL,
                              SECTNO            INTEGER      NOT NULL,
                              OBJECT_SCHEMA     VARCHAR(128) NOT NULL,
                              OBJECT_NAME       VARCHAR(128) NOT NULL,
                              OBJECT_TYPE       CHAR(2)      NOT NULL,
                              CREATE_TIME       TIMESTAMP,
                              STATISTICS_TIME   TIMESTAMP,
                              COLUMN_COUNT      SMALLINT     NOT NULL,
                              ROW_COUNT         INTEGER      NOT NULL,
                              WIDTH             INTEGER      NOT NULL,
                              PAGES             INTEGER      NOT NULL,
                              DISTINCT          CHAR(1)      NOT NULL,
                              TABLESPACE_NAME   VARCHAR(128),
                              OVERHEAD          DOUBLE       NOT NULL,
                              TRANSFER_RATE     DOUBLE       NOT NULL,
                              PREFETCHSIZE      INTEGER      NOT NULL,
                              EXTENTSIZE        INTEGER      NOT NULL,
                              CLUSTER           DOUBLE       NOT NULL,
                              NLEAF             INTEGER      NOT NULL,
                              NLEVELS           INTEGER      NOT NULL,
                              FULLKEYCARD       BIGINT       NOT NULL,
                              OVERFLOW          INTEGER      NOT NULL,
                              FIRSTKEYCARD      BIGINT       NOT NULL,
                              FIRST2KEYCARD     BIGINT       NOT NULL,
                              FIRST3KEYCARD     BIGINT       NOT NULL,
                              FIRST4KEYCARD     BIGINT       NOT NULL,
                              SEQUENTIAL_PAGES  INTEGER      NOT NULL,
                              DENSITY           INTEGER      NOT NULL,
                                  FOREIGN KEY (EXPLAIN_REQUESTER,
                                               EXPLAIN_TIME,
                                               SOURCE_NAME,
                                               SOURCE_SCHEMA,
                                               EXPLAIN_LEVEL,
                                               STMTNO,
                                               SECTNO)
                                  REFERENCES EXPLAIN_STATEMENT
                                  ON DELETE CASCADE )
```

EXPLAIN_OPERATOR 表の定義

```
CREATE TABLE EXPLAIN_OPERATOR ( EXPLAIN_REQUESTER VARCHAR(128) NOT NULL,
                                EXPLAIN_TIME      TIMESTAMP    NOT NULL,
                                SOURCE_NAME       VARCHAR(128) NOT NULL,
                                SOURCE_SCHEMA     VARCHAR(128) NOT NULL,
                                EXPLAIN_LEVEL     CHAR(1)      NOT NULL,
                                STMTNO            INTEGER      NOT NULL,
                                SECTNO            INTEGER      NOT NULL,
                                OPERATOR_ID       INTEGER      NOT NULL,
                                OPERATOR_TYPE     CHAR(6)      NOT NULL,
                                TOTAL_COST        DOUBLE       NOT NULL,
                                IO_COST           DOUBLE       NOT NULL,
                                CPU_COST          DOUBLE       NOT NULL,
                                FIRST_ROW_COST    DOUBLE       NOT NULL,
                                RE_TOTAL_COST     DOUBLE       NOT NULL,
                                RE_IO_COST        DOUBLE       NOT NULL,
                                RE_CPU_COST       DOUBLE       NOT NULL,
                                COMM_COST         DOUBLE       NOT NULL,
                                FIRST_COMM_COST   DOUBLE       NOT NULL,
                                REMOTE_TOTAL_COST DOUBLE       NOT NULL,
                                REMOTE_COMM_COST  DOUBLE       NOT NULL,
                                  FOREIGN KEY (EXPLAIN_REQUESTER,
                                               EXPLAIN_TIME,
                                               SOURCE_NAME,
                                               SOURCE_SCHEMA,
                                               EXPLAIN_LEVEL,
                                               STMTNO,
                                               SECTNO)
                                  REFERENCES EXPLAIN_STATEMENT
                                  ON DELETE CASCADE )
```

EXPLAIN_PREDICATE 表の定義

```
CREATE TABLE EXPLAIN_PREDICATE ( EXPLAIN_REQUESTER VARCHAR(128) NOT NULL,
                                 EXPLAIN_TIME      TIMESTAMP    NOT NULL,
                                 SOURCE_NAME       VARCHAR(128) NOT NULL,
                                 SOURCE_SCHEMA     VARCHAR(128) NOT NULL,
                                 EXPLAIN_LEVEL     CHAR(1)      NOT NULL,
                                 STMTNO            INTEGER      NOT NULL,
                                 SECTNO            INTEGER      NOT NULL,
                                 OPERATOR_ID       INTEGER      NOT NULL,
                                 PREDICATE_ID      INTEGER      NOT NULL,
                                 HOW_APPLIED       CHAR(5)      NOT NULL,
                                 WHEN_EVALUATED    CHAR(3)      NOT NULL,
                                 RELOP_TYPE        CHAR(2)      NOT NULL,
                                 SUBQUERY          CHAR(1)      NOT NULL,
                                 FILTER_FACTOR     DOUBLE       NOT NULL,
                                 PREDICATE_TEXT    CLOB(1M)     NOT LOGGED,
                                      FOREIGN KEY (EXPLAIN_REQUESTER,
                                                   EXPLAIN_TIME,
                                                   SOURCE_NAME,
                                                   SOURCE_SCHEMA,
                                                   EXPLAIN_LEVEL,
                                                   STMTNO,
                                                   SECTNO)
                                      REFERENCES EXPLAIN_STATEMENT
                                      ON DELETE CASCADE )
```

EXPLAIN_STATEMENT 表の定義

```
CREATE TABLE EXPLAIN_STATEMENT ( EXPLAIN_REQUESTER VARCHAR(128) NOT NULL,
                                 EXPLAIN_TIME      TIMESTAMP    NOT NULL,
                                 SOURCE_NAME       VARCHAR(128) NOT NULL,
                                 SOURCE_SCHEMA     VARCHAR(128) NOT NULL,
                                 EXPLAIN_LEVEL     CHAR(1)      NOT NULL,
                                 STMTNO            INTEGER      NOT NULL,
                                 SECTNO            INTEGER      NOT NULL,
                                 QUERYNO           INTEGER      NOT NULL,
                                 QUERYTAG          CHAR(20)     NOT NULL,
                                 STATEMENT_TYPE    CHAR(2)      NOT NULL,
                                 UPDATABLE         CHAR(1)      NOT NULL,
                                 DELETABLE         CHAR(1)      NOT NULL
                                 TOTAL_COST        DOUBLE       NOT NULL,
                                 STATEMENT_TEXT    CLOB(1M)     NOT NULL
                                                                NOT LOGGED,
                                 SNAPSHOT          BLOB(10M)    NOT LOGGED,
                                 QUERY_DEGREE      INTEGER      NOT NULL,
                                     PRIMARY KEY (EXPLAIN_REQUESTER,
                                                  EXPLAIN_TIME,
                                                  SOURCE_NAME,
                                                  SOURCE_SCHEMA,
                                                  EXPLAIN_LEVEL,
                                                  STMTNO,
                                                  SECTNO),
                                     FOREIGN KEY (EXPLAIN_REQUESTER,
                                                  EXPLAIN_TIME,
                                                  SOURCE_NAME,
                                                  SOURCE_SCHEMA)
                                     REFERENCES EXPLAIN_INSTANCE
                                     ON DELETE CASCADE )
```

EXPLAIN_STREAM 表の定義

```
CREATE TABLE EXPLAIN_STREAM ( EXPLAIN_REQUESTER VARCHAR(128) NOT NULL,
                              EXPLAIN_TIME      TIMESTAMP    NOT NULL,
                              SOURCE_NAME       VARCHAR(128) NOT NULL,
                              SOURCE_SCHEMA     VARCHAR(128) NOT NULL,
                              EXPLAIN_LEVEL     CHAR(1)      NOT NULL,
                              STMTNO            INTEGER      NOT NULL,
                              SECTNO            INTEGER      NOT NULL,
                              STREAM_ID         INTEGER      NOT NULL,
                              SOURCE_TYPE       CHAR(1)      NOT NULL,
                              SOURCE_ID         SMALLINT     NOT NULL,
                              TARGET_TYPE       CHAR(1)      NOT NULL,
                              TARGET_ID         SMALLINT     NOT NULL,
                              OBJECT_SCHEMA     VARCHAR(128),
                              OBJECT_NAME       VARCHAR(128),
                              STREAM_COUNT      DOUBLE       NOT NULL,
                              COLUMN_COUNT      SMALLINT     NOT NULL,
                              PREDICATE_ID      INTEGER      NOT NULL,
                              COLUMN_NAMES      CLOB(1M)     NOT LOGGED,
                              PMID              SMALLINT     NOT NULL,
                              SINGLE_NODE       CHAR(5),
                              PARTITION_COLUMNS CLOB(64K)    NOT LOGGED,
                                  FOREIGN KEY (EXPLAIN_REQUESTER,
                                               EXPLAIN_TIME,
                                               SOURCE_NAME,
                                               SOURCE_SCHEMA,
                                               EXPLAIN_LEVEL,
                                               STMTNO,
                                               SECTNO)
                                  REFERENCES EXPLAIN_STATEMENT
                                  ON DELETE CASCADE )
```

ADVISE_INDEX 表の定義

```
CREATE TABLE ADVISE_INDEX (EXPLAIN_REQUESTER VARCHAR(128) NOT NULL
                             WITH DEFAULT '',
                           EXPLAIN_TIME      TIMESTAMP    NOT NULL
                              WITH DEFAULT CURRENT TIMESTAMP,
                           SOURCE_NAME       VARCHAR(128) NOT NULL
                             WITH DEFAULT '',
                           SOURCE_SCHEMA     VARCHAR(128) NOT NULL
                             WITH DEFAULT '',
                           EXPLAIN_LEVEL     CHAR(1)      NOT NULL
                             WITH DEFAULT '',
                           STMTNO            INTEGER      NOT NULL
                              WITH DEFAULT 0,
                           SECTNO            INTEGER      NOT NULL
                              WITH DEFAULT 0,
                           QUERYNO           INTEGER      NOT NULL
                              WITH DEFAULT 0,
                           QUERYTAG          CHAR(20)     NOT NULL
                             WITH DEFAULT '',
                           NAME              VARCHAR(128) NOT NULL,
                           CREATOR           VARCHAR(128) NOT NULL
                             WITH DEFAULT '',
                           TBNAME            VARCHAR(128) NOT NULL,
                           TBCREATOR         VARCHAR(128) NOT NULL
                             WITH DEFAULT '',
                           COLNAMES          CLOB(64K)    NOT NULL,
                           UNIQUERULE        CHAR(1)      NOT NULL
                             WITH DEFAULT '',
                           COLCOUNT          SMALLINT     NOT NULL
                              WITH DEFAULT 0,
                           IID               SMALLINT     NOT NULL
                              WITH DEFAULT 0,
                           NLEAF             INTEGER      NOT NULL
                              WITH DEFAULT 0,
                           NLEVELS           SMALLINT     NOT NULL
                              WITH DEFAULT 0,
                           FIRSTKEYCARD      BIGINT       NOT NULL
                              WITH DEFAULT 0,
                           FULLKEYCARD       BIGINT       NOT NULL
                              WITH DEFAULT 0,
                           CLUSTERRATIO      SMALLINT     NOT NULL
                              WITH DEFAULT 0,
                           CLUSTERFACTOR     DOUBLE       NOT NULL
                              WITH DEFAULT 0,
                           USERDEFINED       SMALLINT     NOT NULL
                              WITH DEFAULT 0,
                           SYSTEM_REQUIRED   SMALLINT     NOT NULL
                              WITH DEFAULT 0,
                           CREATE_TIME       TIMESTAMP    NOT NULL
                              WITH DEFAULT CURRENT TIMESTAMP,
                           STATS_TIME        TIMESTAMP
                              WITH DEFAULT CURRENT TIMESTAMP,
                           PAGE_FETCH_PAIRS  VARCHAR(254) NOT NULL
                             WITH DEFAULT '',
```

```
REMARKS           VARCHAR(254)
                    WITH DEFAULT '',
DEFINER           VARCHAR(128) NOT NULL
                    WITH DEFAULT '',
CONVERTED          CHAR(1) NOT NULL
                    WITH DEFAULT '',
SEQUENTIAL_PAGES  INTEGER      NOT NULL
                     WITH DEFAULT 0,
DENSITY           INTEGER      NOT NULL
                     WITH DEFAULT 0,
FIRST2KEYCARD     BIGINT       NOT NULL
                     WITH DEFAULT 0,
FIRST3KEYCARD     BIGINT       NOT NULL
                     WITH DEFAULT 0,
FIRST4KEYCARD     BIGINT       NOT NULL
                     WITH DEFAULT 0,
PCTFREE           SMALLINT     NOT NULL
                     WITH DEFAULT -1,
UNIQUE_COLCOUNT   SMALLINT     NOT NULL
                     WITH DEFAULT -1,
MINPCTUSED        SMALLINT     NOT NULL
                     WITH DEFAULT 0,
REVERSE_SCANS     CHAR(1)      NOT NULL
                     WITH DEFAULT 'N',
USE_INDEX         CHAR(1),
CREATION_TEXT     CLOB(1M)     NOT NULL
                     NOT LOGGED WITH DEFAULT '',
PACKED_DESC       BLOB(1M)     NOT LOGGED)
```

ADVISE_WORKLOAD 表の定義

```
CREATE TABLE ADVISE_WORKLOAD (WORKLOAD_NAME  CHAR(128)    NOT NULL
                                WITH DEFAULT 'WK0',
                              STATEMENT_NO   INTEGER      NOT NULL
                                WITH DEFAULT 1,
                              STATEMENT_TEXT CLOB(1M)     NOT NULL NOT LOGGED,
                              STATEMENT_TAG  VARCHAR(256) NOT NULL
                                WITH DEFAULT '',
                              FREQUENCY      INTEGER      NOT NULL
                                WITH DEFAULT 1,
                              IMPORTANCE     DOUBLE       NOT NULL
                                WITH DEFAULT 1,
                              COST_BEFORE    DOUBLE,
                              COST_AFTER     DOUBLE)
```

付録C. SQL EXPLAIN ツール

db2expln ツールは、システム・カタログ表に保管されているパッケージにある静的 SQL ステートメント用に選択されたアクセス・プランを記述します。これを使用して、バインド時に Explain データがキャプチャーされなかったパッケージ用に選択されたアクセス・プランに関する早見 Explain を入手することができます。

dynexpln ツールは、動的ステートメント用に選択されたアクセス・プランを記述します。このツールは、ステートメント用に静的パッケージを作成し、 **db2expln** ツールを使用してステートメントを記述します。

特定の SQL ステートメント用に選択されたアクセス・プランについて理解するために、これらの EXPLAIN ツールを使用することができます。あるいは、特定の SQL ステートメント用に選択されたアクセス・プランについて理解するために、 Visual Explain と組み合わせて統合化 Explain 機能 (211ページの『第7章 SQL Explain 機能』) を使用することも可能です。 Explain 機能を用いることにより、動的および静的 SQL ステートメントの両方を Explain することができます。 EXPLAIN ツールとの相違点の 1 つは、 Visual Explain では、Explain 情報がグラフィック形式で表示されることです。それ以外は、2 つの方式で提示される明細のレベルは同じです。

`db2expln` および `dynexpln` の出力を十分に活用するためには、以下の事項について理解しておく必要があります。

- サポートされる各種 SQL ステートメントと、それらのステートメントに関連した用語 (SELECT ステートメント内の述部など)。
- パッケージの目的 (アクセス・プラン)。 (この情報については、160ページの『データ・アクセス概念および最適化』を参照してください。)
- システム・カタログ表の目的および内容。 (この情報については、*SQL 解説書* を参照してください。)
- 41ページの『第2部 アプリケーション・パフォーマンスのチューニング』に記述されている他の概念。

以下の部分で、`db2expln` および `dynexpln` について説明します。

- db2expln および dynexpln の実行
- db2expln の構文およびパラメーター
- db2expln の使用上の注意
- dynexpln 構文およびパラメーター
- dynexpln の使用上の注意
- db2expln および dynexpln 出力の説明

- db2expln および dynexpln 出力の例

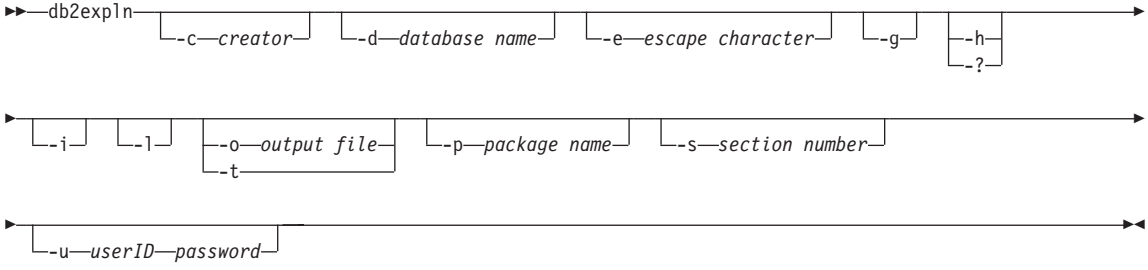
db2expln および dynexpln の実行

EXPLAIN ツール (db2expln および dynexpln) は、ご使用のインスタンスの sqllib ディレクトリーの misc サブディレクトリーの中にあります。 db2expln および dynexpln が現行ディレクトリーにない場合は、該当の PATH 環境変数に示されているディレクトリーにあるはずです。

db2expln プログラムは、データベースが最初にアクセスされるときに、 db2expln.bnd ファイルを用いて、データベースに接続されバインドされます。 db2expln.bnd ファイルは、 sqllib ディレクトリーの bnd サブディレクトリーの中にあります。

db2expln を実行するには、システム・カタログ視点に対する SELECT 特権と、db2expln のパッケージの EXECUTE 権限を持っている必要があります。 dynexpln を実行するには、データベースへの BINDADD 権限を所有し、データベースに接続する際に使用するスキーマが存在しているか、もしくはデータベースへの EXPLICIT_SCHEMA 権限を所有している必要があります。また、SQL ステートメントを説明するために必要な特権も必要です。 (SYSADM 権限または DBADM 権限を持っている場合は、自動的にこれらの権限レベルをすべて持つことになります。)

db2expln の構文およびパラメーター



ここで、

-c creator

パッケージの作成者のユーザー ID。

このオプションを指定しないと、指定するようプロンプトで指示されます。

作成者の名前は、LIKE 述部で使用されるパターン照合文字、パーセント記号 (%) および下線 (_) を使用して指定することができます。

-d database name

Explain されるパッケージが入っているデータベースの名前。

このオプションを指定しないと、指定するようプロンプトで指示されます。

-e escape character
パターン照合文字ではなく、エスケープ文字として解釈される文字を指定するために使用されます。

たとえば、パッケージ TESTID.CALC% を Explain するための db2expln コマンドは、 db2expln -c TESTID -p CALC% です。しかし、このコマンドは、CALC で始まるものであればその他のプランも Explain することになります。TESTID.CALC% パッケージだけを Explain したい場合は、エスケープ文字を使用しなければなりません。このコマンドを db2expln -c TESTID -e ! -p CALC!% に変更することによって、 ! 文字はエスケープ文字として使用し、 !% は % 文字として解釈することを指定します。

-g 最適化プログラムのプランをグラフ表示します。個々のセクションが検査され、オリジナルの最適化プログラムのプランのグラフ (Visual Explain で生成されたもの) が構成されます。生成されるグラフは、オリジナルのプランと一致しない場合もあることに注意してください。

-h or -?
入力パラメーターに関するヘルプ情報を入手します。このオプションを指定すると、他のすべてのオプションが指定変更されます。

-i Explain されるプランの中のオペレーター ID を表示します。オペレーター ID を使用すると、 db2expln からの出力を、Explain 機能からの出力と突き合わせることができます。

-l このオプションを指定した場合は、パッケージ名を小文字のみの形式か、または大文字と小文字を混合させた形式で指定することができます。この **-l** オプションを指定しないと、パッケージ名は大文字に変換されます。

-o output file
db2expln が結果を書き込むファイルの名前。

ファイル名を指定しないで **-o** を指定すると、ファイル名を指定するようプロンプトで指示されます。デフォルトのファイル名は、db2expln.out です。

-t 出力は端末に送信されます。

-o または **-t** を指定しないと、ファイル名を指定するようプロンプトで指示され、デフォルト (端末に出力を表示する) が示されます。

-p package name
Explain されるパッケージの名前。

このオプションを指定しないと、指定するようプロンプトで指示されます。

パッケージ名は、LIKE 述部で使用されるパターン照合文字、パーセント記号 (%) および下線 (_) を使用して指定することができます。

-s section number

パッケージ内にある Explain するセクションの番号。番号ゼロ (0) は、パッケージ内のすべてのセクションを Explain したい場合に指定できます。パッケージ作成者（**-c**）またはパッケージ名 (**-p**) 引き数が、複数のパッケージ (したがって複数のセクション) が Explain されることを示している場合は、そのセクション値 (指定されている場合) はゼロ (0) で上書きされます。

このオプションを指定しないと、指定するようプロンプトで指示されます。

セクション番号は、システム・カタログ SYSCAT.STATEMENTS を照会して調べます (システム・カタログ表の説明は、 *SQL 解説書* を参照してください)。

-u userID password

データベースに接続するときは、与えられたユーザー ID とパスワードを使用してください。

ユーザー ID とパスワードはどちらも、DB2 命名規則に従った有効なもので、さらにデータベースが認識できるものでなければなりません。

上記のオプション・フラグの中には、ご使用のオペレーティング・システムにとって特別な意味をもつものもあり、その結果、db2expln コマンド行で正しく解釈されないこともあります。しかし、これらの文字の前にエスケープ文字を付けて入力することも可能です。詳細については、ご使用のオペレーティング・システムのユーザーズ・マニュアルを参照してください。

db2expln によって作成されるヘルプおよび初期状況メッセージは、標準出力に書き出されます。 EXPLAIN ツールにより作成されるすべてのプロンプトおよびその他の状況メッセージは、標準エラーに書き出されます。 Explain テキストは、選択した出力オプションに基づいて、標準出力またはファイルに書き出されます。

-p および **-c** オプションでは、 LIKE パターンでパッケージと作成者に関するストリング定数を指定して、 Explain を 1 回呼び出すだけで複数のプランを Explain することができます。つまり、下線 (_) を使用して単一文字を表し、パーセント記号 (%) を用いてゼロ個以上の文字があることを表すことができます。

たとえば、SAMPLE という名前のデータベース内のすべてのパッケージに関するすべてのセクションの Explain を、ファイル **my.exp** に書き込まれるようにするには、次のように入力します。

```
db2expln -d SAMPLE -p % -c % -s 0 -o my.exp
```

db2expln の使用上の注意

db2expln で表示される共通のメッセージを以下に示します。

- `No packages found for database <database>, package pattern: <creator>.<package>.` (データベース database には、パッケージが見つかりません。パッケージ・パターン creator.package。)

  データベース内に指定されたパターンに一致するパッケージが見つからなかった場合に、このメッセージが表示されます。

- `Bind messages can be found in db2expln.msg.` (バインド・メッセージは db2expln.msg にあります。)

  `db2expln.bnd` のバインドがうまく行われなかった場合に、このメッセージが出力に表示されます。検出された問題に関するさらに詳しい情報は、現行ディレクトリー内のファイル `db2expln.msg` にあります。

- `Section number overridden to 0 for potential multiple packages.` (複数のパッケージが存在する可能性があるため、セクション番号は 0 に指定変更されました。)

  `db2expln` が複数のパッケージを検出する可能性がある場合に、このメッセージが出力に表示されます。いずれかのパターン照合文字がパッケージまたは作成者入力引き数に使用されている場合に、この処置が取られます。

- `No static sections qualify from package.` (パッケージに適した静的セクションがありません。)

  指定されたパッケージに動的 SQL ステートメントだけが含まれている (すなわち、静的セクションがないことを意味する) 場合に、このメッセージが出力に表示されます。

- `Database <database>, package <creator>.<package> is not valid. Rebind and then rerun db2expln.` (データベース database、パッケージ creator.package が有効ではありません。再バインドしてから、db2expln を再実行してください。)

  指定されたパッケージが現在は有効ではない場合に、このメッセージが出力に表示されます。指示通りに、プランに対して BIND または REBIND コマンドを再発行し、データベース内に有効なパッケージを作成し直して、さらに `db2expln` を再実行してください。

- `Section not processed: Produced by unsupported release.` (セクションは処理されません。サポートされていないリリースによって作成されたものです。)

  現在処理中のセクションが、実行可能なこの `db2expln` が提供されたリリースの DB2 以外で作成されたものである場合に、このメッセージが出力に表示されます。その場合は、該当のセクションを作成した DB2 のリリースから `db2expln` のコピーを使用してください。

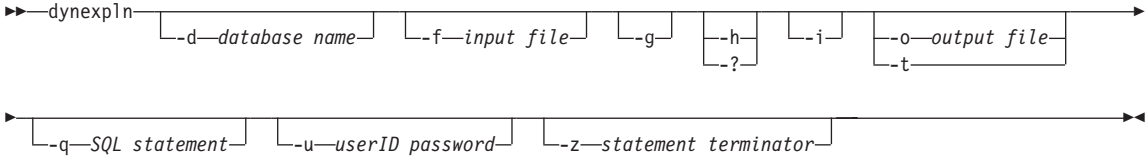
除外される SQL ステートメント: 以下のステートメントについては、Explain されません。

- BEGIN/END DECLARE SECTION
- BEGIN/END COMPOUND
- INCLUDE

- WHENEVER
- COMMIT および ROLLBACK
- CONNECT
- OPEN カーソル
- FETCH
- CLOSE カーソル
- PREPARE
- EXECUTE
- EXECUTE IMMEDIATE
- DESCRIBE
- 動的 DECLARE CURSOR
- SQL 制御ステートメント

複合 SQL ステートメント内の個々のサブステートメントには、独自のセクションが存在していることがあり、そのセクションを **db2expln** で Explain することができます。

dynexpln 構文およびパラメーター



ここで、

-d database name
Explain されるパッケージが入っているデータベースの名前。

このオプションを指定しないと、指定するようプロンプトで指示されます。

-f input file
Explain される SQL ステートメントが入っているファイルの名前。

ステートメント終止符 (**-z**) オプションを使用していないかぎり、ファイルの各行には、1 つの SQL ステートメントしか入力してはなりません。 SQL コメントをファイルに入力することができます。 SQL コメントは、 -- で始め、行末に向かって進めます。

-g 最適化プログラムのプランをグラフ表示します。個々のセクションが検査され、オリジナルの最適化プログラムのプランのグラフ (Visual Explain で生成さ

れたもの) が構成されます。生成されるグラフは、オリジナルのプランと一致しない場合もあることに注意してください。

-h or -?
入力パラメーターに関するヘルプ情報を入手します。このオプションを指定すると、他のすべてのオプションが指定変更されます。

-i Explain されるプランの中のオペレーター ID を表示します。オペレーター ID を使用すると、 db2expln からの出力を、Explain 機能からの出力と突き合わせることができます。

-o output file
db2expln が結果を書き込むファイルの名前。

-t 出力は端末に送信されます。

出力 (**-o**) および **-t** オプションの両方が指定されている場合は、出力は端末に送信されます。

出力ファイル **-o** または **-t** オプションを指定しないと、ファイル名を指定するようプロンプトで指示され、デフォルト (端末で出力を表示する) が示されます。

-q SQL statement
Explain する SQL ステートメント。

このオプションを指定せず、しかも入力ファイル (**-f**) オプション・パラメーターを指定していない場合は、 Explain する SQL ステートメントを入力するようプロンプトで指示されます。

このオプションと入力ファイル (**-f**) オプション・パラメーターの両方を指定している場合は、 dynexpln は、まず SQL ステートメント (**-s**) オプションで提供されたステートメントを記述し、次に、入力ファイル (**-f**) の中のステートメントを記述します。

-u userID password
データベースに接続するときは、与えられたユーザー ID とパスワードを使用してください。

ユーザー ID とパスワードはどちらも、DB2 命名規則に従った有効なもので、さらにデータベースが認識できるものでなければなりません。

-z statement terminator
SQL ステートメントの終わりに達したことを示すために用いられる文字。

デフォルトは、ステートメント終止符なしです。このオプションが使用されていないと、ファイルの各行は、別個の SQL ステートメントであると見なされます。このオプションを使用すると、dynexpln は、指定された終了文字を使用して、ステートメントを区切ります。

上記のオプション・フラグの中には、ご使用のオペレーティング・システムにとって特別な意味をもつものもあり、その結果、dynexpln コマンド行で正しく解釈されないこともあります。しかし、これらの文字の前にエスケープ文字を付けて入力することも可能です。詳細については、ご使用のオペレーティング・システムのユーザーズ・マニュアルを参照してください。

ステートメント終止符 (**-z**) オプションを使用する場合には、 SQL ステートメント (**-s**) オプションを用いて複数のステートメントを入力することができます。この場合は、終了文字でステートメントを区切る必要があります。

dynexpln によって作成されるヘルプおよび初期状況メッセージは、標準出力に書き出されます。 EXPLAIN ツールにより作成されるすべてのプロンプトおよびその他の状況メッセージは、標準エラーに書き出されます。 Explain テキストは、選択した出力オプションに基づいて、標準出力またはファイルに書き出されます。

たとえば、SAMPLE という名前のデータベースに接続して、ファイル TRYIT にあるすべてのステートメントを Explain し、結果をファイル my.exp に書き出すときは、次のように入力します。

```
dynexpln -d SAMPLE -f TRYIT -o my.exp
```

dynexpln の使用上の注意

動的ステートメントを Explain するために、dynexpln は、そのステートメントに対して静的アプリケーションを作成してから、 db2expln を起動します。静的ステートメントを作成するときは、dynexpln は、それらのステートメントを使って小さい C プログラムを生成してから、 DB2 事前コンパイラーを呼び出してパッケージを作成します。(生成される C プログラムは完全なものではないので、コンパイルすることはできません。事前コンパイラーがパッケージを組み立てるのに必要な程度の情報が入っているだけです。)

dynexpln で表示される共通のメッセージを以下に示します。

- db2expln からのすべてのエラー・メッセージ。

  dynexpln は db2expln を起動するので、 db2expln のエラー・メッセージのほとんどを見ることができます。

- Error connecting to the database. (データベースに接続中にエラー。)

  データベースへの接続中にエラーが起こった場合に、このメッセージが出力に表示されます。接続が完了できなかった理由を示す CLI エラー・メッセージも表示されます。エラーの原因を訂正して、再び dynexpln を実行します。

- The file "<filename>" must be removed before dynexpln will run. (dynexpln を実行する前に、ファイル filename を除去する必要があります。)

dynexpln が実行される時点で該当のファイルが存在していると、このメッセージが表示されます。該当のファイルを除去するか、 DYNEXPLN_PACKAGE 環境変数の値を変更して、作成する予定のファイルの名前を変更し、再び dynexpln を実行します。

- The package "<creator>.<package>" must be dropped before dynexpln will run. (dynexpln を実行する前に、パッケージ creator.package を除去しておく必要があります。)

  dynexpln が実行される時点で該当のパッケージが存在していると、このメッセージが表示されます。該当のパッケージを除去して実行するか、または **DYNEXPLN_PACKAGE** 環境変数の値を変更して、作成する予定のパッケージの名前を変更して、再び dynexpln を実行します。

- Error writing file "<filename>". (ファイル filename に書き込み中にエラー。)

  該当のファイルに書き出すことができないと、このメッセージが表示されます。 dynexpln が現行のディレクトリーにファイルを書き出して、再び実行できるようにしてください。

- Error reading input file "<filename>". (入力ファイル filename 読み取り中にエラー。)

  -f オプションで指定されたファイルを読むことができないと、このメッセージが表示されます。該当のファイルが存在し、 dynexpln がそれを読めるようにしてください。 dynexpln をもう一度実行します。

環境変数: dynexpln と組み合わせて使用できるような 2 つの異なる環境変数があります。

- **DYNEXPLN_OPTIONS** は、自分用のステートメントのパッケージを組み立てるときに使用する SQL プリコンパイラー・オプションです。 CLP を介して PREP コマンドを発行するときに用いるのと同じ構文変数を使用します。

  たとえば、次のようにします。 DYNEXPLN_OPTIONS="OPTLEVEL 5 BLOCKING ALL"

- **DYNEXPLN_PACKAGE** は、データベースの中に作成されるパッケージの名前です。記述されるステートメントは、このパッケージに置かれます。この変数が定義されていないと、パッケージには **DYNEXPLN** のデフォルト値が与えられます。 (この環境変数の名前の最初の 8 文字だけが使用されます。)

  この名前は、 dynexpln が使用する中間ファイルの名前を作成するときにも使用します。

db2expln および dynexpln 出力の説明

出力では、各パッケージの Explain 情報が次の 2 つの部分に分かれます。

- バインド日付や関係するバインド・オプションなどのパッケージ情報。
- セクション番号 (前) および Explain される SQL ステートメント (後) などの、セクション情報。セクション情報の下には、表示された SQL ステートメント用に選択されたアクセス・プランの Explain 出力があります。

アクセス・プラン (つまりセクション) のステップは、データベース・マネージャーが実行する順序で示されます。それぞれの主なステップは、左寄せの見出しで示され、そのステップに関する情報はその下に字下げして示されます。アクセス・プランの Explain 出力には、出力の左方マージンに字下げの棒線があります。これらの棒線は操作の「有効範囲」を示します。すなわち、同じ操作の中でもより下位の字下げレベルにある (さらに右側にずれている) 操作は、すぐ上位にある字下げレベルに戻る前に処理されます。

選択されたアクセス・プランが元の SQL ステートメント (出力に表示されたステートメント) の増補されたものに基づいていることを思い出してください。たとえば、元のステートメントがいくつものトリガーや制約を活動化するものである場合があります。また、元の SQL ステートメントが、SQL コンパイラーの照会再書き込みコンポーネントによって、同等ではあるものの、より効率的な形式に書き直されることもあります。最適化プログラムがこのステートメントを満足させるのに最も効率的なプランを決める際には、これらのすべての要素が情報として最適化プログラムに提供されます。したがって、Explain 出力に示されるアクセス・プランが、元の SQL ステートメントについて予期していたアクセス・プランとは本質的に異なってしまうという場合もあります。統合化 Explain 機能 (211ページの『第7章 SQL Explain 機能』参照) は、最適化に使用される実際の SQL ステートメントを、照会の内部表現を逆変換して作成し、 SQL のようなステートメントの形式で示します。

db2expln または dynexpln からの出力を、 Explain 機能の出力と比較するときに、オペレーター ID オプション (**-i**) は非常に便利です。 db2expln または dynexpln が、Explain 機能から新しいオペレーターの処理を開始するたびに、オペレーター ID 番号が、Explain されるプランの左側に印刷されます。オペレーター ID は、異なる表示のアクセス・プランの中で、ステップを突き合わせるために使用することができます。Explain 機能の出力の中のオペレーターと、 db2expln および dynexpln によって示される操作の間が、つねに 1 対 1 で対応しているわけではないことに注意してください。

以下の項目では、 db2expln および dynexpln によって作成される Explain テキストについて説明します。

- 表アクセス
- 一時表
- 結合
- データ・ストリーム
- 挿入、更新、および削除
- 行 ID (RID) の作成
- 集約
- 並列処理
- 連合ステートメント処理

- その他のステートメント

表アクセス

このステートメントは、アクセスする表の名前とタイプを指定します。使用できる形式には、次の 2 つがあります。

1. 正規表には、以下の 3 つのタイプがあります。
   - アクセス表名:

     ```
     Access Table Name = schema.name  ID = ts,n
     ```

     説明:

     - *schema.name* は、アクセスする表の完全修飾名です。
     - *ID* は、 SYSCAT.TABLES カタログ内での表に対応する TABLESPACEID と TABLEID です。
   - アクセス階層表名:

     ```
     Access Hierarchy Table Name = schema.name  ID = ts,n
     ```

     説明:

     - *schema.name* は、アクセスする表の完全修飾名です。
     - *ID* は、 SYSCAT.TABLES カタログ内での表に対応する TABLESPACEID と TABLEID です。
   - アクセス要約表名:

     ```
     Access Summary Table Name = schema.name  ID = ts,n
     ```

     説明:

     - *schema.name* は、アクセスする表の完全修飾名です。
     - *ID* は、 SYSCAT.TABLES カタログ内での表に対応する TABLESPACEID と TABLEID です。
2. 一時表には、以下の 2 つのタイプがあります。
   - アクセス一時表 ID:

     ```
     Access Temp Table  ID = tn
     ```

     説明:

     - *ID* は、db2expln から割り当てられた対応の ID です。
   - アクセス宣言済みグローバル一時表 ID:

     ```
     Access Global Temp Table  ID = ts,tn
     ```

     説明:

     - *ID* は、 SYSCAT.TABLES カタログ内での表に対応する TABLESPACEID (ts)、および db2expln から割り当てられた、対応する ID (tn) です。

表アクセス・ステートメントに続いて、アクセスを詳しく記述するためにさらにステートメントが提供されています。これらのステートメントは、表アクセス・ステートメントの下に字下げされます。可能なステートメントは、以下のとおりです。

- 列の数
- 並列走査
- 走査方向
- 行アクセス方式
- ロック意図
- 述部
- その他の表ステートメント

列の数

次のステートメントは、表の各行から使用する列の数を示します。

```
#Columns = n
```

並列走査

次のステートメントは、データベース・マネージャーがサブエージェントをいくつか使用して、表から並列に読み取りをすることを示します。

```
Parallel Scan
```

このテキストが示されない場合は、表からの読み取りは 1 つのエージェント (またはサブエージェント) によってのみ行われます。

走査方向

次のステートメントは、データベース・マネージャーが列を逆順に読み取ることを示します。

```
Scan Direction = Reverse
```

このテキストが示されていない場合は、走査方向は順方向で、これがデフォルトです。

行アクセス方式

次のステートメントのうちの 1 つが表示されている場合、表内の修飾行がアクセスされる方法を示します。

- `Relation Scan` ステートメントは、修飾行を検索するために表を順次走査することを示します。
  - 次のステートメントは、データの事前取り出しが行われないことを示しています。

    ```
    Relation Scan
    |  Prefetch: None
    ```

  - 次のステートメントは、事前取り出しされるページ数を最適化プログラムが事前に判断したことを示します。

```
Relation Scan
|  Prefetch: n Pages
```

– 次のステートメントは、データを事前取り出しすることを示します。

```
Relation Scan
|  Prefetch: Eligible
```

– 次のステートメントは、修飾行が索引によって識別およびアクセスされることを示します。

```
Index Scan:  Name = schema.name  ID = xx
|  Index Columns:
```

説明:

- *schema.name* は、走査する索引の完全修飾名です。
- *ID* は、 SYSCAT.INDEXES カタログ表の中の対応する ID 列です。

その後には、索引の列ごとに 1 つの行が続きます。索引の各列は、以下のいずれかの形式でリストされます。

```
n: column_name (Ascending)
n: column_name (Descending)
n: column_name (Include Column)
```

次のステートメントは、索引走査のタイプを明確にするために提供されています。

- 索引の範囲区切り述部は、以下のようにして示されます。

```
#Key Columns = n
|  Start Key: xxxxx
|  Stop Key: xxxxx
```

ここで、xxxxx は以下のいずれかに該当します。

- 索引の始まり
- 索引の終わり
- 包括値: または 排他値:

  索引走査には、包括キー値が含まれます。走査に排他キー値は含まれません。キーの値は、キーの各部を構成する以下の行のいずれかによって指定されます。

```
n: 'string'
n: nnn
n: yyyy-mm-dd
n: hh:mm:ss
n: yyyy-mm-dd hh:mm:ss.uuuuuu
n: NULL
n: ?
```

  リテラル・ストリングが示される場合は、最初の 20 文字だけが表示されます。ストリングの長さが 20 文字を超える場合、ストリングの終わりには ...

が示されます。キーによっては、セクションが実行されるまで判別できないものがあります。その場合は、値として ? が示されます。

- 索引のみのアクセス

  必要なすべての列が索引キーから入手できる場合、このステートメントが表示され、表データにはアクセスしません。

- 次のステートメントは、索引ページの事前取り出しが行われないことを示しています。

  ```
  Index Prefetch: None
  ```

- 次のステートメントは、索引ページを事前取り出しすることを示します。

  ```
  Index Prefetch: Eligible
  ```

- 次のステートメントは、データ・ページの事前取り出しが行われないことを示しています。

  ```
  Data Prefetch: None
  ```

- 次のステートメントは、データ・ページを事前取り出しすることを示します。

  ```
  Data Prefetch: Eligible
  ```

- 索引管理プログラムに渡されて索引項目を修飾するのに役立つ述部があれば、述部の数を示すために次のステートメントを使用します。

  ```
  Sargable Index Predicate(s)
  |   #Predicates = n
  ```

– Fetch Direct ステートメントは、アクセス・プランですでに作成されている行 ID (RID) を使用して、修飾行にアクセスしていることを示します。

ロック意図

表アクセスのたびに、表および行レベルで獲得されるロックのタイプを、次のステートメントを用いて表示することができます。

```
Lock Intents
|  Table: xxxx
|  Row  : xxxx
```

表ロックに可能な値は、以下のとおりです。

- 排他
- 意図排他
- 意図なし
- 意図共用
- 共用
- 共用意図排他
- 超排他
- 更新

行ロックに可能な値は、以下のとおりです。

- 排他
- 次キー排他 (db2expln 出力に表示されません)
- なし
- 共用
- 次キー共用
- 更新
- 次キー弱排他
- 弱排他

これらのロック・タイプの解説は、52ページの『ロックの属性』にあります。

述部

アクセス・プランに用いられる述部に関する情報を提供するステートメントは、2 つあります。

1. 次のステートメントは、一度データが戻されたときに、述部の数が評価されることを示します。

   ```
   Residual Predicate(s)
   |   #Predicates = n
   ```

2. 次のステートメントは、データ・アクセス中に、述部の数が評価されることを示します。述部のカウントには、集約や分類などの後入れ先出し操作は含まれていません。

   ```
   Sargable Predicate(s)
   |   #Predicates = n
   ```

このステートメントに表示された述部の数は、 SQL ステートメント中の述部の数を反映していないことがあります。なぜなら、述部は次のような場合があるからです。

- 同じ照会内で複数回適用される。
- 照会最適化処理時に暗黙述部が追加され、変形と拡張が行われる。
- 照会最適化処理時に変形と圧縮が行われ、述部が少なくなる。

その他の表ステートメント

- 次のステートメントは、1 行だけにアクセスできることを示します。

  ```
  Single Record
  ```

- 次のステートメントは、この表アクセスに使用されている分離レベルが、パッケージとは異なる分離レベルを使用しているときに表示されます。

  ```
  Isolation Level: xxxx
  ```

  さまざまな理由により、異なる分離レベルを使用することも可能です。その理由としては、以下のことが挙げられます。

– パッケージが反復可能読み取りにバインドされており、参照保全制約に影響を与える。参照保全制約を調べるために親表にアクセスすると、この表で不要なロックを保持しないように、カーソル固定の分離レベルに低下します。
– 非コミット読み取りにバインドされているパッケージが DELETE または UPDATE ステートメントを出している。実際に削除するために表アクセスをすると、カーソル固定にアップグレードします。

- 次のステートメントは、使用可能なメモリーが十分ある場合に、一時表から読み取られた行の一部またはすべてがバッファー・プール以外にキャッシュされることを示します。

  ```
  Keep Rows In Private Memory
  ```

- 表に揮発性のカーディナリティー属性セットがある場合、そのことが以下のように示されます。

  ```
  Volatile Cardinality
  ```

一時表

一時表は、アクセス・プランを実行中にデータを一時的な作業表に保管するために使用されます。この表は、アクセス・プラン実行中にのみ存在します。通常は、アクセス・プランの初期段階で副照会の評価が必要になったとき、または中間結果が利用可能なメモリーに納まらないときに、一時ファイルが使用されます。

一時表を作成することが必要になると、 2 つのステートメントのうちのどちらかが表示されることがあります。これらのステートメントは、一時表を作成し、その表に行が挿入されるように指示します。 ID は、一時表を参照するときの便宜上 db2expln によって割り当てられる ID です。この ID は、接頭部として文字 't' が付き、その表が一時表であることを示します。

- 次のステートメントは、一般的な一時表が作成されることを示します。

  ```
  Insert Into Temp Table  ID = tn
  ```

- 次のステートメントは、通常の一時表が複数のサブエージェントによって並列に作成されることを示します。

  ```
  Insert Into Shared Temp Table  ID = tn
  ```

- 次のステートメントは、分類済みの一時表が作成されることを示します。

  ```
  Insert Into Sorted Temp Table  ID = tn
  ```

- 次のステートメントは、分類済みの一時表が複数のサブエージェントによって並列に作成されることを示します。

  ```
  Insert Into Sorted Shared Temp Table  ID = tn
  ```

- 次のステートメントは、宣言済みグローバル一時表が作成されることを示します。

  ```
  Insert Into Global Temp Table  ID = ts,tn
  ```

- 次のステートメントは、宣言済みグローバル一時表が複数のサブエージェントによって並列に作成されることを示します。

```
Insert Into Shared Global Temp Table  ID = ts,tn
```

- 次のステートメントは、分類された宣言済みグローバル一時表が作成されることを示します。

```
Insert Into Sorted Global Temp Table  ID = ts,tn
```

- 次のステートメントは、分類された宣言済みグローバル一時表が複数のサブエージェントによって並列に作成されることを示します。

```
Insert Into Sorted Shared Global Temp Table  ID = ts,tn
```

上記のステートメントの後には、いずれも次の 1 行を付加することができます。

```
#Columns = n
```

これは、一時表に挿入している各行を何列にするかを示します。

分類済みの一時表

次のような操作から、分類済みの一時表を作成することができます。

- ORDER BY
- DISTINCT
- GROUP BY
- Merge Join
- '= ANY' 副照会
- '<> ALL' 副照会
- INTERSECT または EXCEPT
- UNION (ALL キーワードの指定なし)

分類済みの一時表を作成するための元のステートメントの後に、いくつかの追加ステートメントを付加することもできます。

- 次のステートメントは、分類に使用するキー列の数を示します。

```
#Sort Key Columns = n
```

  分類キー中の列ごとに、次の行のうち 1 つが表示されます。

```
Key n: column_name (Ascending)
Key n: column_name (Descending)
Key n: (Ascending)
Key n: (Descending)
```

- 次のステートメントは、実行時に最適な分類ヒープを割り当てることができるように、行数および行サイズの見積もりを提供します。

```
Sortheap Allocation Parameters:
|  #Rows     = n
|  Row Width = n
```

- 分類結果の先頭行だけが必要な場合は、次のように表示されます。

```
Sort Limited To Estimated Row Count
```

- 対称マルチプロセッサー (SMP) 環境での分類の場合は、実行される分類のタイプは次のようなステートメントのいずれかによって指示されます。

  ```
  Use Partitioned Sort
  Use Shared Sort
  Use Replicated Sort
  Use Round-Robin Sort
  ```

  各種の分類技法についての説明は、192ページの『並列分類の方式』を参照してください。

- 次のステートメントは、分類の結果を分類ヒープに残すかどうかを指定します。

  ```
  Piped
  ```

  および

  ```
  Not Piped
  ```

  パイプ分類が指示されている場合は、データベース・マネージャーは、分類の出力をメモリーに保管して、分類結果を別の一時表には入れません。 (パイプ分類と非パイプ分類の説明については、 189ページの『最適化プログラムでの分類の影響』を参照してください。)

- 次のステートメントは、分類中に重複値を除去することを示しています。

  ```
  Duplicate Elimination
  ```

- 分類の中で集約が行われている場合は、下記のステートメントのいずれかによって指示されます。

  ```
  Partial Aggregation
  Intermediate Aggregation
  Buffered Partial Aggregation
  Buffered Intermediate Aggregation
  ```

一時表の完了

一時表を作成するための後入れ先出し操作を含む表アクセス (つまり、表アクセスの有効範囲内で生じる一時表の作成) の後には、「完了 (completion)」ステートメントがあります。このステートメントは、一時表がそれ以後の一時表アクセスで行を提供できるようにしておくことによって、ファイルの終わりを処理します。次の行のうち 1 つが表示されます。

```
Temp Table Completion  ID = tn
Shared Temp Table Completion  ID = tn
Sorted Temp Table Completion  ID = tn
Sorted Shared Temp Table Completion  ID = tn
```

表関数

表関数とは、データを表の形式でステートメントに戻すユーザー定義関数 (UDF) のことです。表関数についての詳細は、*SQL 解説書* を参照してください。表関数は次のステートメントによって示されます。

```
Access User Defined Table Function
|   Name = schema.funcname
|   Language = xxxx
|   Fenced   Deterministic   NULL Call   Disallow Parallel
```

表関数を書き込む際に使用する言語 (C、OLE、または Java) と、表関数の属性を指定します。

結合

結合には 3 つのタイプがあります (これらの結合の説明については、 172ページの『結合の概念』を参照してください)。

- ハッシュ結合
- マージ結合
- ネスト・ループ結合

結合セクションが実行される時になると、以下のステートメントのうち 1 つが表示されます。

```
Hash Join
```

または

```
Merge Join
```

または

```
Nested Loop Join
```

左方外部結合を行うこともできます。左方外部結合は、下記のステートメントのいずれかで指示されます。

```
Left Outer Hash Join
```

または

```
Left Outer Merge Join
```

または

```
Left Outer Nested Loop Join
```

マージ結合とネスト・ループ結合の場合、結合の外部表は、出力に表示されている直前のアクセス・ステートメント内で参照される表です。結合の内部表は、結合ステートメントの有効範囲内にあるアクセス・ステートメントで参照される表です。ハッシュ結合

の場合、逆に結合の有効範囲内のアクセス・ステートメントによって参照される表が外部表になり、結合の前に表示されるアクセス・ステートメントによって参照される表が内部表になります。

ハッシュ結合とマージ結合の場合、次のような追加のステートメントが表示されます。

- ある種の環境では、結合は、内部表の中の任意の行が外部表の現在行と一致しているかだけを判別する必要があります。これは、次のステートメントで指示されます。

  ```
  Early Out: Single Match Per Outer Row
  ```

- 結合が完了したあとで、述部を適用することもできます。適用される述部の数は、次のように指示されます。

  ```
  Residual Predicate(s)
  |  #Predicates = n
  ```

ハッシュ結合の場合、次のような追加のステートメントが表示されます。

- ハッシュ表は内部表から作成されます。作成されたハッシュ表が、内部表のアクセスに関する述部に後入れ先出しされた場合、そのことが内部表のアクセスに関するステートメント中に次のように指示されます。

  ```
  Process Hash Table For Join
  ```

- 外部表にアクセスする際には、プローブ表が作成されて結合のパフォーマンスが向上します。プローブ表が作成された場合、そのことは外部表のアクセスに関する次のステートメントで指示されます。

  ```
  Process Probe Table For Hash Join
  ```

- ハッシュ表を作成するのに必要なバイト数の見積もりは次のように表されます。

  ```
  Estimated Build Size: n
  ```

- プローブ表に必要なバイト数の見積もりは次のように表されます。

  ```
  Estimated Probe Size: n
  ```

ネストしたループ結合の場合、次の追加ステートメントが結合ステートメントの直後に表示されます。

```
Piped Inner
```

このステートメントは、結合の内部表が別の一連の操作の結果であることを示します。これを、*複合内部* ともいいます。

結合が 3 つ以上の表に関係する場合、Explain ステップは上から下に読みます。たとえば、Explain 出力に次のような流れがあるとします。

```
Access ..... W
Join
|  Access ..... X
```

```
Join
|  Access ..... Y
Join
|  Access ..... Z
```

この場合、実行ステップは次のようになります。

1. W から修飾行を取り出す。
2. W からの行を、X からの行 (次の行) と結合し、結果 P1 (部分結合の結果番号 1) を呼び出す。
3. P1 を Y からの行 (次の行) と結合して、 P2 を作成する。
4. P2 を Z からの行 (次の行) と結合し、 1 つの完全結果行を入手する。
5. さらに行が Z にあれば、ステップ 4 に戻る。
6. さらに行が Y にあれば、ステップ 3 に戻る。
7. さらに行が X にあれば、ステップ 2 に戻る。
8. さらに行が W にあれば、ステップ 1 に戻る。

データ・ストリーム

アクセス・プラン内では、ある一連の操作から別の一連の操作へとデータの作成と流れを制御することがしばしば必要になります。データ・ストリームという概念を用いると、あるアクセス・プラン内での一群の操作を 1 つの単位として制御することが可能になります。データ・ストリームの先頭は、次のステートメントで示します。

```
Data Stream n
```

ここで、n は、参照を容易にするために db2expln によって割り当てられた固有の ID です。データ・ストリームの末尾は、次のステートメントで示します。

```
End of Data Stream n
```

この 2 つのステートメントの間のすべての操作が、同一のデータ・ストリームの一部と見なされます。

データ・ストリームにはいくつもの特性があり、最初のデータ・ストリーム・ステートメントの後には 1 つまたは複数のステートメントに続けて、それらの特性を記述することができます。

- データ・ストリームの操作がアクセス・プランで以前に生成された値によって決まる場合、そのデータ・ストリームには、以下のマークが付けられます。

  ```
  Correlated
  ```

- 分類済みの一時表と同様、以下のステートメントは、データ・ストリームの結果をメモリーに保持するかどうかを示します。

  ```
  Piped
  ```

  および

```
Not Piped
```

一時表の場合にそうであったように、パイプ・データ・ストリームも、実行時にメモリーが十分になければディスクに書き込むことができます。アクセス・プランでは、いずれの場合にも対応できるようになっています。

- 次のステートメントは、このデータ・ストリームから要求されているのが 1 つのレコードだけであることを示します。

```
Single Record
```

データ・ストリームがアクセスされると、次のステートメントが出力に表示されます。

```
Access Data Stream n
```

挿入、更新、および削除

これらの SQL ステートメントの Explain テキストは、解説するまでもありません。これらの SQL 操作に使用可能なステートメント・テキストは、次のとおりです。

- `Insert: Table Name = schema.name ID = ts,n`
- `Update: Table Name = schema.name ID = ts,n`
- `Delete: Table Name = schema.name ID = ts,n`
- `Insert: Hierarchy Table Name = schema.name ID = ts,n`
- `Update: Hierarchy Table Name = schema.name ID = ts,n`
- `Delete: Hierarchy Table Name = schema.name ID = ts,n`
- `Insert: Summary Table Name = schema.name ID = ts,n`
- `Update: Summary Table Name = schema.name ID = ts,n`
- `Delete: Summary Table Name = schema.name ID = ts,n`
- `Insert: Global Temporary Table ID = ts, tn`
- `Update: Global Temporary Table ID = ts, tn`
- `Delete: Global Temporary Table ID = ts, tn`

行 ID (RID) の作成

一部のアクセス・プランでは、実際の表アクセスが実行される前に、修飾行 ID (RID) を分類しておき、重複を削除しておくか (index ORing の場合)、またはアクセスされているすべての索引に出てくる RID を識別するための手法を使用すると (index ANDing の場合)、効率がよくなります。 Explain ステートメントによって示される RID 作成には、主に 3 つの使用法があります。

- 次のステートメントは、『Index ORing』 を使用して修飾 RID のリストを作成します。

```
Index ORing RID Preparation
```

Index ORing は、複数の索引アクセスを作成し、その結果を結合して、アクセスされるいずれの索引にも出てくる別個の RID を組み込む技法を指します。 OR キーワードにより述部が接続される場合や、IN 述部がある場合に、最適化プログラムは索引の OR を考慮します。索引アクセスは、同一の索引または別々の索引に作成できます。

- RID 作成のもう 1 つの使用法は、以下に示すように、リストの事前取り出し中に使用される入力データを作成することです。

  ```
  List Prefetch RID Preparation
  ```

- *Index ANDing* とは、複数の索引アクセスを作成し、その結果を結合して、アクセスされるすべての索引に出てくる RID を組み込む技法を指します。 Index ANDing 処理は、次のステートメントで開始されます。

  ```
  Index ANDing
  ```

  最適化プログラムが結果のセットのサイズを算定した場合は、下記のステートメントによって算定結果が示されます。

  ```
  Optimizer Estimate of Set Size: n
  ```

  Index ANDing フィルター操作は、RID を処理し、ビット・フィルター操作を使用して、アクセスされるすべての索引に出てくる RID を判別します。下記のステートメントは、index ANDing 用に RID が処理されていることを示します。

  ```
  Index ANDing Bitmap Build
  Index ANDing Bitmap Probe
  Index ANDing Bitmap Build and Probe
  ```

  最適化プログラムがビットマップ用に結果のセットのサイズを算定した場合は、次のステートメントによって算定結果が示されます。

  ```
  Optimizer Estimate of Set Size: n
  ```

どのタイプの RID 作成の場合でも、リスト事前取り出しを実行できる場合は、次のステートメントを用いてそれが表示されます。

```
Prefetch: Enabled
```

集約

集約は、SQL ステートメント述部によって指定される基準があれば、その基準に合致する行で実行されます。何らかの集約関数が実行されると、以下のステートメントのいずれかが表示されます。

```
Aggregation
Predicate Aggregation
Partial Aggregation
Partial Predicate Aggregation
```

```
Intermediate Aggregation
Intermediate Predicate Aggregation
Final Aggregation
Final Predicate Aggregation
```

述部集約とは、データに実際にアクセスするときに、集約操作が後入れ先出し式に述部として処理されることを表します。

上述の集約ステートメントのいずれの場合もその下に、実行される総計関数のタイプの指示があります。

- `Group By`
- `Column Function(s)`
- `Single Record`

特定の列関数は、元の SQL ステートメントから引き出すことができます。単一のレコードは、MIN または MAX 演算の条件を満たす索引から取り出されます。

述部集約を使用すると、集約が示される表アクセス・ステートメントに続いて、集約「完了」になります。これは、各グループの完了またはファイルの終わりの際に必要とされる処理をすべて実行します。次の行のうちのいずれかが表示されます。

```
Aggregation Completion
Partial Aggregation Completion
Intermediate Aggregation Completion
Final Aggregation Completion
```

並列処理

SQL ステートメントを並列で実行する場合 (区画内並列処理または区画間並列処理のいずれかを使用して) は、特別な操作が必要になります。並列プランの操作について、以下で説明します。

- 区画内並列プランを実行するときは、サブエージェントをいくつか使用してプランの部分が同時に実行されます。サブエージェントの作成は、次のステートメントによって指示されます。

  ```
  Process Using n Subagents
  ```

- 区画間並列プランを実行しているときは、セクションはいくつかのサブセクションに分けられます。各サブセクションは、1 つまたはいくつかのノードに送られて、実行されます。重要なサブセクションは、*調整プログラム・サブセクション* です。調整プログラム・サブセクションは、あらゆるプランの中で最初のサブセクションです。これは、最初に制御を得るもので、他のサブセクションを分配したり、呼び出し側のアプリケーションに結果を戻したりする責任があります。

  サブセクションの分配は、次のステートメントによって指示されます。

  ```
  Distribute Subsection #n
  ```

サブセクションを受け取るノードは、以下の 8 つの方法のいずれかで判別することができます。

- 以下のステートメントは、列の値に基づいて、ノードグループ内にあるノードにサブセクションが送られることを示します。

```
Directed by Hash
|   #Columns = n
|  Partition Map ID = n, Nodegroup = ngname, #Nodes = n
```

- 以下のステートメントは、事前に決められたノードにサブセクションが送られることを示します。 (これは、ステートメントが NODENUMBER() 関数を使用しているときに、よく見られます。)

```
Directed by Node Number
```

- 以下のステートメントは、指定ノードグループで事前に決められた区分番号に対応するノードにサブセクションが送られることを示します。 (これは、ステートメントが PARTITION () 関数を使用しているときに、よく見られます。)

```
Directed by Partition Number
|  Partition Map ID = n, Nodegroup = ngname, #Nodes = n
```

- 以下のステートメントは、アプリケーションのカーソル用の現在行を提示したノードに、サブセクションが送られることを示します。

```
Directed by Position
```

- 以下のステートメントは、ステートメントがコンパイルされたときに判別された、ある 1 つのノードだけがサブセクションを受け取ることを示します。

```
Directed to Single Node
|  Node Number = n
```

- 以下のステートメントは、調整プログラム・ノードに対してサブセクションが実行されることを示します。

```
Directed to Coordinator Node
```

- 以下のステートメントは、リストされたすべてのノードにサブセクションが送られることを示します。

```
Broadcast to Node List
|  Nodes = n1, n2, n3, ...
```

- 以下のステートメントは、ステートメントが実行されるときに判別された、ある 1 つのノードだけがサブセクションを受け取ることを示します。

```
Directed to Any Node
```

- 区分データベース環境内のサブセクション同士の間、または対称マルチプロセッサー (SMP) 環境にあるサブエージェント同士の間でデータを移動するために、表キューが使用されます。表キューは、次のように記述されます。
  - 下記のステートメントは、データが表キューに挿入されることを示します。

```
Insert Into Synchronous Table Queue  ID = qn
Insert Into Asynchronous Table Queue  ID = qn
Insert Into Synchronous Local Table Queue  ID = qn
Insert Into Asynchronous Local Table Queue  ID = qn
```

– データベース区画表キューの場合は、表キューに挿入された行の宛先は、以下のいずれかで記述されます。

```
Broadcast to Coordinator Node
```

すべての行が調整プログラム・ノードに送られます。

```
Broadcast to All Nodes of Subsection n
```

指定のサブセクションが実行されているすべてのデータベース区画に、すべての行が送られます。

```
Hash to Specific Node
```

行にある値に基づいて、各行がデータベース区画に送られます。

```
Send to Specific Node
```

ステートメントが実行されている間に、各行が決定されたデータベース区画に送られます。

```
Send to Random Node
```

各行がランダム・データベース区画に送られます。

– ある種の状況では、データベース区画表キューは、一部の行を一時的に一時表にオーバーフローさせる必要があります。このような可能性は、次のステートメントによって識別します。

```
Rows Can Overflow to Temporary Table
```

– 表アクセスの際に後入れ先出し操作により行を表キューに挿入した場合、即時送信できなかった行について示した「完了」ステートメントがその後に表示されます。次の行のうちのいずれかが表示されます。

```
Insert Into Synchronous Table Queue Completion  ID = qn
Insert Into Asynchronous Table Queue Completion  ID = qn
Insert Into Synchronous Local Table Queue Completion  ID = qn
Insert Into Asynchronous Local Table Queue Completion  ID = qn
```

– 下記のステートメントは、データが表キューから検索されることを示します。

```
Access Table Queue  ID = qn
Access Local Table Queue  ID = qn
```

これらのメッセージの後には常に、検索される列の数の表示が付いています。

```
#Columns = n
```

– 表キューが受信端で行を分類する場合は、表キュー・アクセスには、下記のメッセージのいずれかが出されます。

```
Output Sorted
Output Sorted and Unique
```

これらのメッセージの後には、分類操作に使用されるキーの数の表示が付いています。

```
#Key Columns = n
```

分類キー中の列ごとに、次のうち 1 つが表示されます。

```
Key n: (Ascending)
Key n: (Descending)
```

– 表キューの受信端によって述部が行に適用される場合は、次のメッセージが表示されます。

```
Residual Predicate(s)
|   #Predicates = n
```

- 区分データベース環境にある一部のサブセクションは、次のステートメントを用いて、サブセクションの先頭まで、明示的にループバックします。

```
Jump Back to Start of Subsection
```

連合ステートメント処理

連合データベースで SQL ステートメントを実行する場合、ほかのデータ・ソースに対してステートメントの部分を実行できなければなりません。

アクセスされるデータ・ソースは以下のように指示されます。

```
Distributed Subquery #n
|   #Columns = n
```

分散副照会から戻されるデータに述部に適用することができます。適用される述部の数は、次のように指示されます。

```
Residual Predicate(s)
|   #Predicates = n
```

各分散副照会の詳細は、個別に指定されます。分散副照会のオプションは、以下のように記述されます。

- 副照会のデータ・ソースは、以下のいずれかで示されます。

```
Server: server_name (type, version)
Server: server_name (type)
Server: server_name
```

- 副照会の SQL ステートメントは以下のように表示されます。

```
Subquery SQL Statement:
statement
```

- 副照会で参照されるニックネームは、以下のようにリストされます。

```
Nickname Referenced:
Schema.nickname Base = baseschema.basetable
```

- 副照会を実行する前に連合サーバーからデータ・ソースに値が渡される場合、値の数は以下のように示されます。

  ```
  #Input Columns: n
  ```

- 副照会を実行した後でデータ・ソースから連合サーバーに値が渡される場合、値の数は以下のように示されます。

  ```
  #Output Columns: n
  ```

その他のステートメント

- データ定義言語ステートメントのセクションは、出力に次のように示されます。

  ```
  DDL Statement
  ```

  DDL ステートメントには、そのほかに Explain 出力はありません。

- 更新可能な特殊レジスター (**CURRENT EXPLAIN SNAPSHOT** など) 用の SET ステートメントのセクションは、出力に次のように示されます。

  ```
  SET Statement
  ```

  SET ステートメントには、そのほかに Explain 出力はありません。

- SQL ステートメントに DISTINCT が含まれる場合、次のテキストが Explain 出力に表示されます。

  ```
  Distinct Filter  #Columns = n
  ```

  ここで、n は、入手している個別行に含まれる列の数です。個別行の値を検索するには、重複値をスキップできるように行を順序付ける必要があります。データベース・マネージャーが明示的に重複を除去する必要がなければ、このステートメントは表示されません。以下のような場合があります。

  – 固有索引が存在しており、索引キー内のすべての列が DISTINCT 操作の一部になっている。
  – 分類中に重複を除去することができる。

- 以下のステートメントは、次の操作が特定のレコード ID に依存している場合に表示されます。

  ```
  Positioned Operation
  ```

  このステートメントは、 WHERE CURRENT OF 構文を使用する SQL ステートメントに使われます。

- 次のステートメントは、結果には適用しなければならないが、別の操作の一部として適用することはできない述部がある場合に表示されます。

  ```
  Residual Predicate Application
  |   #Predicates = n
  ```

- 次のステートメントは、SQL ステートメントに UNION 演算子がある場合に表示されます。

  ```
  UNION
  ```

- 次のステートメントは、後続の操作に使用される行の値を作成することだけを目的とした操作がアクセス・プラン内にある場合に表示されます。

  ```
  Table Constructor
  |  n-Row(s)
  ```

  表構成プログラムを使用して、1 つの集合として存在している値を一連の行に変形し、後続の操作に渡すことができます。表構成プログラムを次の行に入力するよう要求されると、次のステートメントが表示されます。

  ```
  Access Table Constructor
  ```

- 次のステートメントは、特定の条件の下でのみ処理される操作があるときに表示されます。

  ```
  Conditional Evaluation
  |  Condition #n:
  |  |  #Predicates = n
  |  Action #n:
  ```

  条件付き評価は、SQL CASE ステートメントなどの活動や、参照保全制約やトリガーなどの内部機構を実行するときに使用します。処置に何も示されていない場合は、条件が真であるときにのみデータ操作命令が処理されます。

- ALL、ANY、または EXISTS 副照会がアクセス・プラン内で処理中である場合には、以下のステートメントのうちのいずれかが表示されます。
  - `ANY/ALL Subquery`
  - `EXISTS Subquery`
  - `EXISTS SINGLE Subquery`
- 特定の UPDATE 操作と DELETE 操作の前に、表中の特定の行の位置を確立する必要があります。このことは、次のステートメントで示します。

  ```
  Establish Row Position
  ```

- 次のステートメントは、アプリケーションに戻っている行がある場合に表示されます。

  ```
  Return Data to Application
  |   #Columns = n
  ```

  操作が表アクセスに後入れ先出しされる場合、完了フェーズが必要になります。このフェーズは、次のように表示されます。

  ```
  Return Data Completion
  ```

db2expln および dynexpln 出力の例

db2expln および dynexpln からの出力のレイアウトと形式をご理解いただくために、 5 つの例を示します。これらの例は、DB2 で提供される SAMPLE データベースに対して実行されたものです。それぞれの例について、簡単な説明が添えられています。 1 つの例と次の例との重要な相違点は、**太字**で示してあります。

例 1: 非並列プラン

この例は、全従業員の名前、職種、部門名とその場所、および現在携わっているプロジェクト名のリストを要求するだけのものです。このアクセス・プランの特徴は、指定したそれぞれの表から関係するデータを結合するのにマージ結合を使用するという点です。索引を使用することができないので、アクセス・プランは各表の関係走査を行います。各表は結合される前に分類される必要があります。

```
******************** PACKAGE ****************************************

Package Name = DOOLE.DYNEXPLN
Prep Date = 2000/01/03
Prep Time = 15:47:58

Bind Timestamp = 2000-01-03-15.47.58.607455

Isolation Level          = Cursor Stability
Blocking                 = Block Unambiguous Cursors
Query Optimization Class = 5

Partition Parallel       = No
Intra-Partition Parallel = No

Function Path            = "SYSIBM", "SYSFUN", "DOOLE"

-------------------- SECTION ---------------------------------------
Section = 1


SQL Statement:

  SELECT x.lastname, x.job, y.deptname, y.location, z.projname
  FROM employee AS x, department AS y, project AS z
  WHERE x.workdept = y.deptno AND x.workdept = z.deptno AND y.deptno
          = z.deptno


Estimated Cost        = 126
Estimated Cardinality = 153

Access Table Name = DOOLE.DEPARTMENT  ID = 2,4
|  #Columns = 3
|  Relation Scan
|  |  Prefetch: Eligible
|  Lock Intents
|  |  Table: Intent Share
```

```
|  |  Row  : Next Key Share
|  Insert Into Sorted Temp Table  ID = t1
|  |  #Columns = 3
|  |  #Sort Key Columns = 1
|  |  |  Key 1: DEPTNO (Ascending)
|  |  Sortheap Allocation Parameters:
|  |  |  #Rows     = 40
|  |  |  Row Width = 48
|  |  Piped
Sorted Temp Table Completion  ID = t1
Access Temp Table  ID = t1
|  #Columns = 3
|  Relation Scan
|  |  Prefetch: Eligible
Merge Join
|  Access Table Name = DOOLE.PROJECT  ID = 2,7
|  |  #Columns = 2
|  |  Relation Scan
|  |  |  Prefetch: Eligible
|  |  Lock Intents
|  |  |  Table: Intent Share
|  |  |  Row  : Next Key Share
|  |  Insert Into Sorted Temp Table  ID = t2
|  |  |  #Columns = 2
|  |  |  #Sort Key Columns = 1
|  |  |  |  Key 1: DEPTNO (Ascending)
|  |  |  Sortheap Allocation Parameters:
|  |  |  |  #Rows     = 38
|  |  |  |  Row Width = 28
|  |  |  Piped
|  Sorted Temp Table Completion  ID = t2
|  Access Temp Table  ID = t2
|  |  #Columns = 2
|  |  Relation Scan
|  |  |  Prefetch: Eligible
Merge Join
|  Access Table Name = DOOLE.EMPLOYEE  ID = 2,5
|  |  #Columns = 3
|  |  Relation Scan
|  |  |  Prefetch: Eligible
|  |  Lock Intents
|  |  |  Table: Intent Share
|  |  |  Row  : Next Key Share
|  |  Insert Into Sorted Temp Table  ID = t3
|  |  |  #Columns = 3
|  |  |  #Sort Key Columns = 1
|  |  |  |  Key 1: WORKDEPT (Ascending)
|  |  |  Sortheap Allocation Parameters:
|  |  |  |  #Rows     = 63
|  |  |  |  Row Width = 32
|  |  |  Piped
|  Sorted Temp Table Completion  ID = t3
|  Access Temp Table  ID = t3
|  |  #Columns = 3
|  |  Relation Scan
```

```
|  |  |  Prefetch: Eligible
Return Data to Application
|  #Columns = 5

End of section


Optimizer Plan:

                RETURN
               (   1)
                  |
                MSJOIN
                (   2)
               /      \
         MSJOIN          TBSCAN
          (   3)         (  12)
         /      \          |
    TBSCAN      TBSCAN     SORT
     (   4)     (   8)     (  13)
       |          |          |
      SORT       SORT      TBSCAN
     (   5)     (   9)     (  14)
       |          |          |
     TBSCAN     TBSCAN   Table:
     (   6)     (  10)    DOOLE
       |          |      EMPLOYEE
  Table:      Table:
  DOOLE       DOOLE
           DEPARTMENT  PROJECT
```

アクセス・プランの最初の部分では、DEPARTMENT および PROJECT 表にアクセスし、マージ結合を使ってそれらの表を結合します。この結合の結果は EMPLOYEE 表に結合されます。結果行はアプリケーションに戻されます。

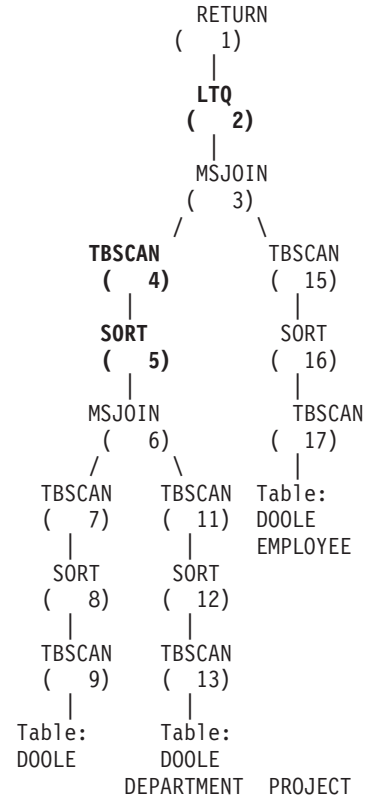
例 2: 区画内並列処理による単一区画データベースのプラン

| この例は、602ページの『例 1: 非並列プラン』と同じ SQL ステートメントを示していますが、この照会は、4-way の SMP マシン用にコンパイルされたものです。

```
******************** PACKAGE ****************************************

Package Name = DOOLE.DYNEXPLN
Prep Date = 2000/01/03
Prep Time = 15:48:51

Bind Timestamp = 2000-01-03-15.48.51.402403

Isolation Level          = Cursor Stability
Blocking                 = Block Unambiguous Cursors
Query Optimization Class = 5

Partition Parallel       = No
Intra-Partition Parallel = Yes (Bind Degree = 4)
```

```
Function Path            = "SYSIBM", "SYSFUN", "DOOLE"

---------- SECTION ----------------------------------------
Section = 1


SQL Statement:

  SELECT x.lastname, x.job, y.deptname, y.location, z.projname
  FROM employee AS x, department AS y, project AS z
  WHERE x.workdept = y.deptno AND x.workdept = z.deptno AND y.deptno
          = z.deptno

Intra-Partition Parallelism Degree = 4

Estimated Cost        = 142
Estimated Cardinality = 153

Process Using 4 Subagents
|  Access Table Name = DOOLE.DEPARTMENT  ID = 2,4
|  |  #Columns = 3
|  |  Parallel Scan
|  |  Relation Scan
|  |  |  Prefetch: Eligible
|  |  Lock Intents
|  |  |  Table: Intent Share
|  |  |  Row  : Next Key Share
|  |  Insert Into Sorted Shared Temp Table  ID = t1
|  |  |  #Columns = 3
|  |  |  #Sort Key Columns = 1
|  |  |  |  Key 1: DEPTNO (Ascending)
|  |  |  Use Round-Robin Sort
|  |  |  Sortheap Allocation Parameters:
|  |  |  |  #Rows     = 40
|  |  |  |  Row Width = 48
|  |  |  Piped
|  Sorted Shared Temp Table Completion  ID = t1
|  Access Temp Table  ID = t1
|  |  #Columns = 3
|  |  Relation Scan
|  |  |  Prefetch: Eligible
|  Merge Join
|  |  Access Table Name = DOOLE.PROJECT  ID = 2,7
|  |  |  #Columns = 2
|  |  |  Parallel Scan
|  |  |  Relation Scan
|  |  |  |  Prefetch: Eligible
|  |  |  Lock Intents
|  |  |  |  Table: Intent Share
|  |  |  |  Row  : Next Key Share
|  |  |  Insert Into Sorted Shared Temp Table  ID = t2
|  |  |  |  #Columns = 2
|  |  |  |  #Sort Key Columns = 1
|  |  |  |  |  Key 1: DEPTNO (Ascending)
```

```
|  |  |  |  Use Replicated Sort
|  |  |  |  Sortheap Allocation Parameters:
|  |  |  |  |  #Rows     = 38
|  |  |  |  |  Row Width = 28
|  |  |  |  Piped
|  |  Sorted Shared Temp Table Completion  ID = t2
|  |  Access Temp Table  ID = t2
|  |  |  #Columns = 2
|  |  |  Relation Scan
|  |  |  |  Prefetch: Eligible
|  Insert Into Sorted Shared Temp Table  ID = t3
|  |  #Columns = 5
|  |  #Sort Key Columns = 1
|  |  |  Key 1: (Ascending)
|  |  Use Partitioned Sort
|  |  Sortheap Allocation Parameters:
|  |  |  #Rows     = 61
|  |  |  Row Width = 72
|  |  Piped
|  Access Temp Table  ID = t3
|  |  #Columns = 5
|  |  Relation Scan
|  |  |  Prefetch: Eligible
|  Merge Join
|  |  Access Table Name = DOOLE.EMPLOYEE  ID = 2,5
|  |  |  #Columns = 3
|  |  |  Parallel Scan
|  |  |  Relation Scan
|  |  |  |  Prefetch: Eligible
|  |  |  Lock Intents
|  |  |  |  Table: Intent Share
|  |  |  |  Row  : Next Key Share
|  |  |  Insert Into Sorted Shared Temp Table  ID = t4
|  |  |  |  #Columns = 3
|  |  |  |  #Sort Key Columns = 1
|  |  |  |  |  Key 1: WORKDEPT (Ascending)
|  |  |  |  Use Partitioned Sort
|  |  |  |  Sortheap Allocation Parameters:
|  |  |  |  |  #Rows     = 63
|  |  |  |  |  Row Width = 32
|  |  |  |  Piped
|  |  Sorted Shared Temp Table Completion  ID = t4
|  |  Access Temp Table  ID = t4
|  |  |  #Columns = 3
|  |  |  Relation Scan
|  |  |  |  Prefetch: Eligible
|  Insert Into Asynchronous Local Table Queue  ID = q1
Access Local Table Queue  ID = q1  #Columns = 5
Return Data to Application
|  #Columns = 5

End of section


Optimizer Plan:
```



このプランは、最初の例のプランとほとんど同じです。主な相違は、プランが最初に開始されるときに 4 つのサブエージェントを作成すること、および、アプリケーションに戻す前におのおののサブエージェントの作業の結果を収集するために、プランの終了時に表キューを作成することです。

興味深い別の点として、EMPLOYEE と結合する前には余分の分類が必要になります。この分類が必要になるのは、 DEPARTMENT と PROJECT のマージ結合を処理するサブエージェントにより、異なる順序で行が結合される可能性があるからです。

例 3: 区画間並列処理による複数区画データベースのプラン

この例は、602ページの『例 1: 非並列プラン』と同じ SQL ステートメントを示していますが、この照会は、3 つのデータベース区画からなる区分データベースでコンパイルされたものです。

```
****************** PACKAGE ************************************

Package Name = DOOLE.DYNEXPLN
Prep Date = 2000/01/03
Prep Time = 15:21:29
```

```
Bind Timestamp = 2000-01-03-15.21.29.990983

Isolation Level          = Cursor Stability
Blocking                 = Block Unambiguous Cursors
Query Optimization Class = 5

Partition Parallel       = Yes
Intra-Partition Parallel = No

Function Path            = "SYSIBM", "SYSFUN", "DOOLE"

-------------------- SECTION -------------------------------------
Section = 1


SQL Statement:

  SELECT x.lastname, x.job, y.deptname, y.location, z.projname
  FROM employee AS x, department AS y, project AS z
  WHERE x.workdept = y.deptno AND x.workdept = z.deptno AND y.deptno
          = z.deptno


Estimated Cost        = 118
Estimated Cardinality = 263

Coordinator Subsection:
   Distribute Subsection #2
   |  Broadcast to Node List
   |  |  Nodes = 13, 82, 193
   Distribute Subsection #3
   |  Broadcast to Node List
   |  |  Nodes = 13, 82, 193
   Distribute Subsection #1
   |  Broadcast to Node List
   |  |  Nodes = 13, 82, 193
   Access Table Queue  ID = q1  #Columns = 5
   Return Data to Application
   |  #Columns = 5

Subsection #1:
   Access Table Queue  ID = q2  #Columns = 3
   |  Output Sorted
   |  |  #Key Columns = 1
   |  |  |  Key 1: (Ascending)
   Merge Join
   |  Access Table Name = DOOLE.DEPARTMENT  ID = 2,4
   |  |  #Columns = 3
   |  |  Relation Scan
   |  |  |  Prefetch: Eligible
   |  |  Lock Intents
   |  |  |  Table: Intent Share
   |  |  |  Row  : Next Key Share
   |  |  Insert Into Sorted Temp Table  ID = t1
   |  |  |  #Columns = 3
```

```
    |  |  |  #Sort Key Columns = 1
    |  |  |  |  Key 1: DEPTNO (Ascending)
    |  |  |  Sortheap Allocation Parameters:
    |  |  |  |  #Rows     = 40
    |  |  |  |  Row Width = 48
    |  |  |  Piped
    |  Sorted Temp Table Completion  ID = t1
    |  Access Temp Table  ID = t1
    |  |  #Columns = 3
    |  |  Relation Scan
    |  |  |  Prefetch: Eligible
    Merge Join
    |  Access Table Queue  ID = q3  #Columns = 2
    |  |  Output Sorted
    |  |  |  #Key Columns = 1
    |  |  |  |  Key 1: (Ascending)
    Insert Into Asynchronous Table Queue  ID = q1
    |  Broadcast to Coordinator Node
    |  Rows Can Overflow to Temporary Table

Subsection #2:
    Access Table Name = DOOLE.EMPLOYEE  ID = 2,5
    |  #Columns = 3
    |  Relation Scan
    |  |  Prefetch: Eligible
    |  Lock Intents
    |  |  Table: Intent Share
    |  |  Row  : Next Key Share
    |  Insert Into Sorted Temp Table  ID = t2
    |  |  #Columns = 3
    |  |  #Sort Key Columns = 1
    |  |  |  Key 1: WORKDEPT (Ascending)
    |  |  Sortheap Allocation Parameters:
    |  |  |  #Rows     = 27
    |  |  |  Row Width = 32
    |  |  Piped
    Sorted Temp Table Completion  ID = t2
    Access Temp Table  ID = t2
    |  #Columns = 3
    |  Relation Scan
    |  |  Prefetch: Eligible
    |  Insert Into Asynchronous Table Queue  ID = q2
    |  |  Hash to Specific Node
    |  |  Rows Can Overflow to Temporary Tables
    Insert Into Asynchronous Table Queue Completion  ID = q2

Subsection #3:
    Access Table Name = DOOLE.PROJECT  ID = 2,7
    |  #Columns = 2
    |  Relation Scan
    |  |  Prefetch: Eligible
    |  Lock Intents
    |  |  Table: Intent Share
    |  |  Row  : Next Key Share
    |  Insert Into Sorted Temp Table  ID = t3
```

```
    |  |  #Columns = 2
    |  |  #Sort Key Columns = 1
    |  |  |  Key 1: DEPTNO (Ascending)
    |  |  Sortheap Allocation Parameters:
    |  |  |  #Rows     = 38
    |  |  |  Row Width = 28
    |  |  Piped
    Sorted Temp Table Completion  ID = t3
    Access Temp Table  ID = t3
    |  #Columns = 2
    |  Relation Scan
    |  |  Prefetch: Eligible
    |  Insert Into Asynchronous Table Queue  ID = q3
    |  |  Hash to Specific Node
    |  |  Rows Can Overflow to Temporary Tables
    Insert Into Asynchronous Table Queue Completion  ID = q3

End of section


Optimizer Plan:

                RETURN
              (   1)
                |
                BTQ
              (   2)
                |
                MSJOIN
              (   3)
              /      \
       MSJOIN          MDTQ
       (   4)         (  14)
       /    \           |
    MDTQ    TBSCAN    TBSCAN
   (   5)   (  10)    (  15)
     |        |         |
   TBSCAN    SORT      SORT
   (   6)   (  11)    (  16)
     |        |         |
    SORT     TBSCAN    TBSCAN
   (   7)   (  12)    (  17)
     |        |         |
   TBSCAN   Table:     Table:
   (   8)   DOOLE      DOOLE
     |      DEPARTMENT  PROJECT
  Table:
  DOOLE
  EMPLOYEE
```

このプランは、最初の例のプランと全く同じ内容ですが、セクションが 4 つのサブセクションに分けられています。サブセクションは、次のようなタスクを行います。

- **調整プログラム・サブセクション**。このサブセクションは、他のサブセクションを調整するものです。このプランでは、他のサブセクションを分散させ、アプリケーションに戻される結果を集めるために表キューを使用します。
- **サブセクション #1**。このサブセクションは表キュー q2 を走査し、マージ結合を使って DEPARTMENT 表と結合します。 2 番目のマージ結合は、表キュー q3 のデータへの追加を行います。結合された行は次に、表キュー q1 を使って調整プログラム・サブセクションに送られます。
- **サブセクション #2**。このサブセクションは EMPLOYEE 表を走査して分類し、結果を使用して特定のノードへハッシュします。これらの結果は、サブセクション #1 が読み取ります。
- **サブセクション #3**。このサブセクションは PROJECT 表を走査して分類し、結果を使用して特定のノードへハッシュします。これらの結果は、サブセクション #1 が読み取ります。

例 4: 区画間並列処理と区画内並列処理による複数区画データベースのプラン

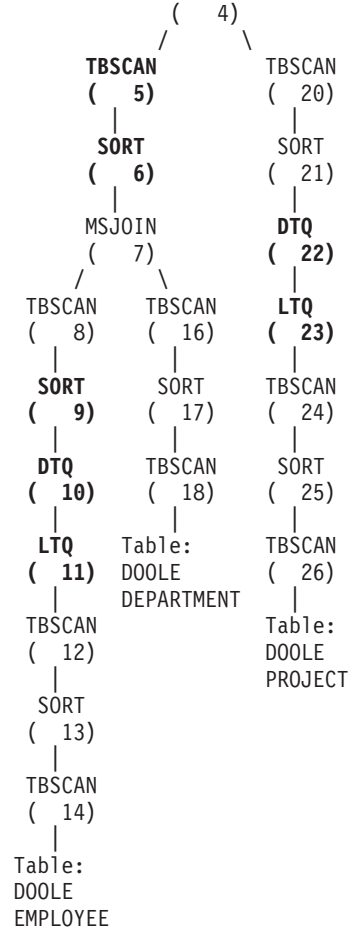
この例は、602ページの『例 1: 非並列プラン』と同じ SQL ステートメントを示していますが、この照会は、 3 つのデータベース区画 (4-way SMP マシン上にある) から成る区分データベースでコンパイルされたものです。

```
******************** PACKAGE ****************************************

Package Name = DOOLE.DYNEXPLN
Prep Date = 2000/01/03
Prep Time = 15:22:14

Bind Timestamp = 2000-01-03-15.22.14.659970

Isolation Level          = Cursor Stability
Blocking                 = Block Unambiguous Cursors
Query Optimization Class = 5

Partition Parallel       = Yes
Intra-Partition Parallel = Yes (Bind Degree = 4)

Function Path            = "SYSIBM", "SYSFUN", "DOOLE"

-------------------- SECTION ---------------------------------------
Section = 1


SQL Statement:

  SELECT x.lastname, x.job, y.deptname, y.location, z.projname
  FROM employee AS x, department AS y, project AS z
  WHERE x.workdept = y.deptno AND x.workdept = z.deptno AND y.deptno
          = z.deptno

Intra-Partition Parallelism Degree = 4
```

```
Estimated Cost        = 140
Estimated Cardinality = 263

Coordinator Subsection:
   Distribute Subsection #2
   |  Broadcast to Node List
   |  |  Nodes = 13, 82, 193
   Distribute Subsection #3
   |  Broadcast to Node List
   |  |  Nodes = 13, 82, 193
   Distribute Subsection #1
   |  Broadcast to Node List
   |  |  Nodes = 13, 82, 193
   Access Table Queue  ID = q1  #Columns = 5
   Return Data to Application
   |  #Columns = 5

Subsection #1:
   Process Using 4 Subagents
   |  Access Table Queue  ID = q3  #Columns = 3
   |  Insert Into Sorted Shared Temp Table  ID = t1
   |  |  #Columns = 3
   |  |  #Sort Key Columns = 1
   |  |  |  Key 1: (Ascending)
   |  |  Use Partitioned Sort
   |  |  Sortheap Allocation Parameters:
   |  |  |  #Rows     = 27
   |  |  |  Row Width = 32
   |  |  Piped
   |  Access Temp Table  ID = t1
   |  |  #Columns = 3
   |  |  Relation Scan
   |  |  |  Prefetch: Eligible
   |  Merge Join
   |  |  Access Table Name = DOOLE.DEPARTMENT  ID = 2,4
   |  |  |  #Columns = 3
   |  |  |  Parallel Scan
   |  |  |  Relation Scan
   |  |  |  |  Prefetch: Eligible
   |  |  |  Lock Intents
   |  |  |  |  Table: Intent Share
   |  |  |  |  Row  : Next Key Share
   |  |  |  Insert Into Sorted Shared Temp Table  ID = t2
   |  |  |  |  #Columns = 3
   |  |  |  |  #Sort Key Columns = 1
   |  |  |  |  |  Key 1: DEPTNO (Ascending)
   |  |  |  |  Use Partitioned Sort
   |  |  |  |  Sortheap Allocation Parameters:
   |  |  |  |  |  #Rows     = 40
   |  |  |  |  |  Row Width = 48
   |  |  |  |  Piped
   |  |  Sorted Shared Temp Table Completion  ID = t2
   |  |  Access Temp Table  ID = t2
   |  |  |  #Columns = 3
   |  |  |  Relation Scan
```

```
    |  |  |  |  Prefetch: Eligible
    |  Insert Into Sorted Shared Temp Table  ID = t3
    |  |  #Columns = 6
    |  |  #Sort Key Columns = 1
    |  |  |  Key 1: (Ascending)
    |  |  Use Partitioned Sort
    |  |  Sortheap Allocation Parameters:
    |  |  |  #Rows     = 44
    |  |  |  Row Width = 76
    |  |  Piped
    |  Access Temp Table  ID = t3
    |  |  #Columns = 6
    |  |  Relation Scan
    |  |  |  Prefetch: Eligible
    |  Merge Join
    |  |  Access Table Queue  ID = q5  #Columns = 2
    |  |  Insert Into Sorted Shared Temp Table  ID = t4
    |  |  |  #Columns = 2
    |  |  |  #Sort Key Columns = 1
    |  |  |  |  Key 1: (Ascending)
    |  |  |  Use Partitioned Sort
    |  |  |  Sortheap Allocation Parameters:
    |  |  |  |  #Rows     = 38
    |  |  |  |  Row Width = 28
    |  |  |  Piped
    |  |  Access Temp Table  ID = t4
    |  |  |  #Columns = 2
    |  |  |  Relation Scan
    |  |  |  |  Prefetch: Eligible
    |  Insert Into Asynchronous Local Table Queue  ID = q2
    Access Local Table Queue  ID = q2  #Columns = 5
    Insert Into Asynchronous Table Queue  ID = q1
    |  Broadcast to Coordinator Node
    |  Rows Can Overflow to Temporary Table

Subsection #2:
    Process Using 4 Subagents
    |  Access Table Name = DOOLE.EMPLOYEE  ID = 2,5
    |  |  #Columns = 3
    |  |  Parallel Scan
    |  |  Relation Scan
    |  |  |  Prefetch: Eligible
    |  |  Lock Intents
    |  |  |  Table: Intent Share
    |  |  |  Row  : Next Key Share
    |  |  Insert Into Sorted Shared Temp Table  ID = t5
    |  |  |  #Columns = 3
    |  |  |  #Sort Key Columns = 1
    |  |  |  |  Key 1: WORKDEPT (Ascending)
    |  |  |  Use Round-Robin Sort
    |  |  |  Sortheap Allocation Parameters:
    |  |  |  |  #Rows     = 27
    |  |  |  |  Row Width = 32
    |  |  |  Piped
    |  Sorted Shared Temp Table Completion  ID = t5
```

```
    |  Access Temp Table  ID = t5
    |  |  #Columns = 3
    |  |  Relation Scan
    |  |  |  Prefetch: Eligible
    |  Insert Into Asynchronous Local Table Queue  ID = q4
   Access Local Table Queue  ID = q4  #Columns = 3
   Insert Into Asynchronous Table Queue  ID = q3
    |  Hash to Specific Node
    |  Rows Can Overflow to Temporary Tables

Subsection #3:
   Process Using 4 Subagents
    |  Access Table Name = DOOLE.PROJECT  ID = 2,7
    |  |  #Columns = 2
    |  |  Parallel Scan
    |  |  Relation Scan
    |  |  |  Prefetch: Eligible
    |  |  Lock Intents
    |  |  |  Table: Intent Share
    |  |  |  Row  : Next Key Share
    |  |  Insert Into Sorted Shared Temp Table  ID = t6
    |  |  |  #Columns = 2
    |  |  |  #Sort Key Columns = 1
    |  |  |  |  Key 1: DEPTNO (Ascending)
    |  |  |  Use Round-Robin Sort
    |  |  |  Sortheap Allocation Parameters:
    |  |  |  |  #Rows     = 38
    |  |  |  |  Row Width = 28
    |  |  |  Piped
    |  Sorted Shared Temp Table Completion  ID = t6
    |  Access Temp Table  ID = t6
    |  |  #Columns = 2
    |  |  Relation Scan
    |  |  |  Prefetch: Eligible
    |  Insert Into Asynchronous Local Table Queue  ID = q6
   Access Local Table Queue  ID = q6  #Columns = 2
   Insert Into Asynchronous Table Queue  ID = q5
    |  Hash to Specific Node
    |  Rows Can Overflow to Temporary Tables

End of section


Optimizer Plan:

                RETURN
               (   1)
                 |
                BTQ
                 (   2)
                 |
                LTQ
               (   3)
                 |
                MSJOIN
```



このプランは、607ページの『例 3: 区画間並列処理による複数区画データベースのプラン』 にあるプランと似ていますが、複数のサブエージェントが各サブセクションを実行する点が異なります。また、各サブセクションの最後に、ローカル表キューが、すべてのサブエージェントの結果を収集してから、修飾行が 2 番目の表キューに挿入され、特定のノードでハッシュされる点も異なります。

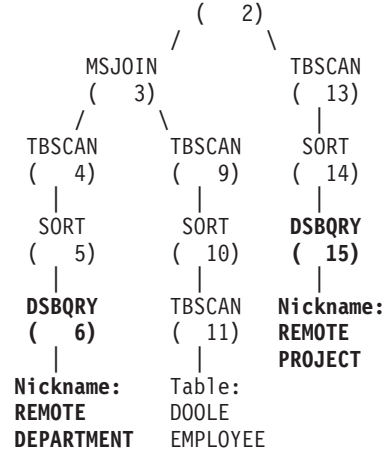
例 5: 連合データベース・プラン

この例は、602ページの『例 1: 非並列プラン』と同じ SQL ステートメントを示していますが、この照会は、連合データベースでコンパイルされたものです。この例では、表 DEPARTMENT および PROJECT はデータ・ソースに、表 EMPLOYEE は連合サーバーにあります。

```
****************** PACKAGE ************************************

Package Name = DOOLE.DYNEXPLN
Prep Date = 2000/01/03
```

```
Prep Time = 16:29:01

Bind Timestamp = 2000-01-03-16.29.01.479230

Isolation Level          = Cursor Stability
Blocking                 = Block Unambiguous Cursors
Query Optimization Class = 5

Partition Parallel       = No
Intra-Partition Parallel = No

Function Path            = "SYSIBM", "SYSFUN", "DOOLE"

-------------------- SECTION --------------------------------------
Section = 1


SQL Statement:

  SELECT x.lastname, x.job, y.deptname, y.location, z.projname
  FROM employee AS x, department AS y, project AS z
  WHERE x.workdept = y.deptno AND x.workdept = z.deptno AND y.deptno
          = z.deptno

Estimated Cost        = 1954
Estimated Cardinality = 100800

Distribute Subquery #2
   |  #Columns = 3
   Insert Into Sorted Shared Temp Table  ID = t1
   |  #Columns = 3
   |  #Sort Key Columns = 1
   |  |  Key 1: Remote Query #2, Output Column 1 (Ascending)
   |  Sortheap Allocation Parameters:
   |  |  #Rows     = 1000
   |  |  Row Width = 56
   |  Piped
   Access Temp Table  ID = t1
   |  #Columns = 3
   |  Relation Scan
   |  |  Prefetch: Eligible
   Merge Join
   |  Access Table Name = DOOLE.DEPARTMENT  ID = 2,5
   |  #Columns = 3
   |  Relation Scan
   |  |  Prefetch: Eligible
   |  |  Lock Intents
   |  |  |  Table: Intent Share
   |  |  |  Row  : Next Key Share
   |  |  Insert Into Sorted Temp Table  ID = t2
   |  |  |  #Columns = 3
   |  |  |  #Sort Key Columns = 1
   |  |  |  |  Key 1: WORKDEPT (Ascending)
   |  |  |  Sortheap Allocation Parameters:
   |  |  |  |  #Rows     = 63
```

```
    |  |  |  |  Row Width = 32
    |  |  |  Piped
    |  Sorted Temp Table Completion  ID = t2
    |  Access Temp Table  ID = t2
    |  |  #Columns = 3
    |  |  Relation Scan
    |  |  |  Prefetch: Eligible
   Merge Join
    |  Distribute Subquery #1
    |  |  #Columns = 2
    |  Insert Into Sorted Temp Table  ID = t3
    |  |  #Columns = 2
    |  |  |  Key 1: Remote Query #1, Output Column 1 (Ascending)
    |  |  Sortheap Allocation Parameters:
    |  |  |  #Rows     = 1000
    |  |  |  Row Width = 36
    |  |  Piped
    |  Access Temp Table  ID = t3
    |  |  #Columns = 2
    |  |  Relation Scan
    |  |  |  Prefetch: Eligible
   Return Data to Application
    |  #Columns = 5

   Distributed Subquery #1:
   Server: REMOTE_SAMPLE  (DB2/CS 7.1)
   Subquery SQL Statement:

        SELECT A0."DEPTNO", A0."PROJNAME"
        FROM "DOOLE"."PROJECT" A0

   Nicknames Referenced:
      REMOTE.PROJECT  ID = 7  Base = DOOLE.PROJECT
   #Output Columns = 2

   Distributed Subquery #2:
   Server: REMOTE_SAMPLE  (DB2/CS 7.1)
   Subquery SQL Statement:

        SELECT A0."DEPTNO", A0."DEPTNAME", A0."LOCATION"
        FROM "DOOLE"."DEPARTMENT" A0

   Nicknames Referenced:
      REMOTE.DEPARTMENT  ID = 4  Base = DOOLE.DEPARTMENT
   #Output Columns = 3

End of section


Optimizer Plan:

               RETURN
              (   1)
                |
               MSJOIN
```

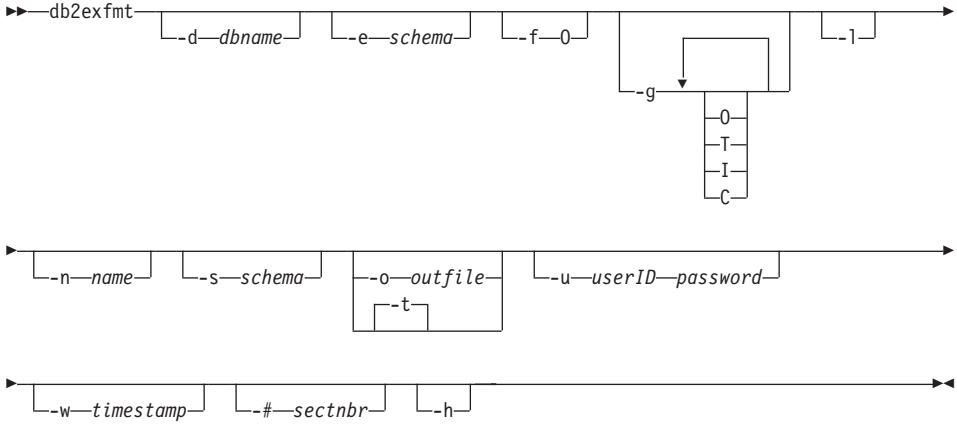


このプランは、最初の例のプランと全く同じ内容ですが、 2 つの表のデータはデータ・ソースから取られます。この 2 つの表には、分散副照会を使用してアクセスします。この例では、単純にそれらの表のすべての行を選択しています。データが連合サーバーに戻されると、そのデータはローカル表から取られたデータと結合させられます。

付録D. db2exfmt - Explain 表フォーマット・ツール

db2exfmt ツールを使用して、Explain 表の内容を形式設定できます。このツールは、インスタンス sqllib ディレクトリーの misc サブディレクトリーにあります。

このツールを使用するには、形式設定する Explain 表に対する読み取りアクセスが必要です。



-d dbname
パッケージを含むデータベースの名前。

-e schema
Explain 表のスキーマ。

-f 形式設定フラグ。このリリースでは、O (オペレーターの要約) だけがサポートされています。

-g グラフのプラン。 -g だけを指定すると、グラフの後に、すべての表に関する形式化された情報が生成されます。他にも指定する場合は、以下の有効値の組み合わせを指定できます。

- **O** グラフだけを生成します。表の内容は形式設定されません。
- **T** グラフ中で、各オペレーターの下に合計コストが示されます。
- **I** グラフ中で、各オペレーターの下に入出力コストが示されます。
- **C** グラフ中で、各オペレーターの予期出力のカーディナリティー (タプルの数) が示されます。

-l パッケージ名の処理時に大文字小文字を考慮します。

-n name
Explain 要求のソースの名前 (SOURCE_NAME)。

-s schema
Explain 要求のソースのスキーマつまり修飾子 (SOURCE_SCHEMA)。

-o outfile
出力ファイル名。

-t 出力の宛先を端末にします。

-u user ID password
データベースに接続するときは、与えられたユーザー ID とパスワードを使用してください。

ユーザー ID とパスワードはどちらも、命名規則に従った有効なもので、さらにデータベースが認識できるものでなければなりません。

-w timestamp
タイム・スタンプを Explain します。最新の Explain 要求を取得するには、-1 を指定します。

-# sectnbr
ソース中のセクション番号。すべてのセクションを要求するには、ゼロを指定します。

-h ヘルプ情報を表示します。このオプションを指定すると、他のオプションはすべて無視され、ヘルプ情報が表示されます。

-h オプションと -l オプションの場合を除いて、パラメーター値を指定していなかったり指定が誤っていたりするとそのことを指摘されます。

Explain 表のスキーマを指定しないと、環境変数 **USER** の値がデフォルトとして使用されます。この変数がない場合は、 Explain 表スキーマを指定するよう指示されます。

ソース名、ソースのスキーマ、および Explain タイム・スタンプは LIKE 述部の形式で指定できます。その際、パターン照合文字としてパーセント記号 (%) と下線 (_) を使用して、 1 回の呼び出しで複数のソースを選択できます。最新の Explain ステートメントの場合、 Explain 時を -1 と指定できます。

ファイル名を指定せずに -o を指定し、しかも -t を指定しないと、ファイル名 (デフォルトの名前は db2exfmt.out) を指定するよう指示されます。 -o と -t を両方とも指定しないと、ファイル名 (デフォルト・オプションは端末出力) を指定するよう指示されます。 -o と -t を両方とも指定すると、出力は端末に送信されます。

付録E. DB2 ライブラリーの使用法

DB2 ユニバーサル・データベース ライブラリーは、オンライン・ヘルプ、ブック (PDF および HTML)、および HTML 形式のサンプル・プログラムから成っています。このセクションでは、ユーザーに提供される情報について紹介し、その入手方法を示します。

オンライン製品情報をご利用になるには、インフォメーション・センターを使用することができます。詳細については、637ページの『インフォメーション・センターを使用した情報へのアクセス』を参照してください。ここではタスク情報、DB2 ブック、トラブルシューティング情報、サンプル・プログラム、および Web の DB2 情報を見ることができます。

DB2 PDF ファイルおよびハードコピー版資料

DB2 情報

以下に示す表では、DB2 ブックを 4 つのカテゴリーに分類しています。

DB2 の手引きおよび解説書
これらの資料は、すべてのプラットフォームに共通の DB2 情報を含んでいます。

DB2 のインストールおよび構成の情報
これらの資料は、特定のプラットフォーム上の DB2 ごとに用意されています。たとえば、 OS/2、Windows、および UNIX ベースのプラットフォームで稼働するそれぞれの DB2 用に、別個の*概説*および*インストール* 資料が用意されています。

プラットフォーム共通のサンプル・プログラム (HTML 形式)
これらのサンプルは、アプリケーション開発クライアントとともにインストールされるサンプル・プログラムの HTML 版です。これらのサンプルは参考用であり、実際のプログラムに代わるものではありません。

リリース情報
これらのファイルには、DB2 ブックには含められなかった最新の情報が記載されています。

インストール情報、リリース情報、およびチュートリアルは、製品 CD-ROM から HTML 形式で参照することができます。ほとんどの資料は、製品 CD-ROM から HTML 形式で表示できますし、 DB2 の資料 CD-ROM から Adobe Acrobat (PDF) 形

式で表示し印刷することができます。 IBM にハードコピー版の資料を注文したい場合は、 633ページの『印刷資料の注文方法』を参照してください。注文可能な資料については、以下の表をご覧ください。

OS/2 および Windows プラットフォームの場合、 HTML ファイルは `sqllib¥doc¥html` ディレクトリーにインストールできます。 DB2 情報はいくつかの言語で提供されています。しかし、すべての言語に翻訳されているわけではありません。ある言語で情報が提供されていない場合は、英語版の情報が提供されます。

UNIX プラットフォームの場合、言語ごとに異なる複数の HTML ファイルを `doc/%L/html` ディレクトリーにインストールできます。ここで、*%L* は地域を表しています。詳細については、適切な*概説およびインストールの手引き* を参照してください。

DB2 ブックを入手して情報を利用するには、次のようなさまざまな方法があります。

- 636ページの『オンライン情報の表示』
- 640ページの『オンライン情報の検索』
- 633ページの『印刷資料の注文方法』
- 633ページの『PDF 資料の印刷』

表 45. DB2 情報

資料名	記述	資料番号 PDF ファイル名	HTML ディレクトリー
	DB2 の手引きおよび解説書情報		
管理の手引き	*管理の手引き: 計画* は、データベース概念について概説し、設計 (たとえば、論理および物理データベース設計) に関する情報を提供し、高い可用性について解説しています。	SC88-8513 db2d1x70	db2d0
	管理の手引き: インプリメンテーション は、設計、データベースへのアクセス、監査、バックアップ、およびリカバリーなどのインプリメンテーションについて説明しています。	SC88-8511 db2d2x70	
	管理の手引き: パフォーマンス は、データベース環境について解説し、さらにアプリケーションのパフォーマンスの評価と調整の方法について説明しています。	SC88-8512 db2d3x70	
管理 API 解説書	データベースの管理に使用できる DB2 アプリケーション・プログラミング・インターフェース (API) およびデータ構造について説明します。また、この資料は、アプリケーションから API を呼び出す方法も示します。	SC88-8514 db2b0x70	db2b0
アプリケーション構築の手引き	環境設定に関する情報を提供し、Windows、OS/2、および UNIX ベースのプラットフォームでの DB2 アプリケーションのコンパイル、リンク、実行の各ステップについて説明します。	SC88-8515 db2axx70	db2ax
APPC, CPI-C, and SNA Sense Codes	DB2 ユニバーサル・データベース製品をご使用中に発生する可能性のあるセンス・コード APPC、CPI-C、および SNA についての一般情報を提供します。 HTML 形式でのみご利用いただけます。	資料番号なし db2apx70	db2ap

表 45. DB2 情報 (続き)

資料名	記述	資料番号 PDF ファイル名	HTML ディレクトリー
アプリケーション開発の手引き	DB2 データベースにアクセスするアプリケーションを、組み込み SQL または Java (JDBC および SQLJ) を使用して開発する方法について説明します。さらに、ストアード・プロシージャーの作成方法、ユーザー定義関数の作成方法、ユーザー定義タイプの作成方法、トリガーの使用方法、区画化されている環境または統合されているシステムでのアプリケーションの開発方法などについて解説されています。	SC88-8516 db2a0x70	db2a0
コール・レベル・インターフェースの手引きおよび解説書	DB2 データベースにアクセスするアプリケーションを、 DB2 コール・レベル・インターフェース (Microsoft ODBC 仕様互換の呼び出し可能 SQL) を使用して開発する方法について説明します。	SC88-8517 db2l0x70	db2l0
コマンド解説書	コマンド行プロセッサーの使用法について説明し、データベースの管理に使用できる DB2 コマンドについて解説しています。	SC88-8518 db2n0x70	db2n0
コネクティビティー 補足	DB2 (AS/400 版)、DB2 (OS/390 版)、DB2 (MVS 版)、または DB2 (VM 版) を DRDA アプリケーション・リクエスターとして DB2 ユニバーサル・データベース とともに使用するためのセットアップ情報および参照情報を提供します。また、この資料は DRDA アプリケーション・サーバーを DB2 コネクト アプリケーション・リクエスターとともに使用する方法の詳細を示します。 HTML と PDF でのみ利用可能	資料番号なし db2h1x70	db2h1
データ移動ユーティリティー手引きおよび解説書	データの移動を行う DB2 ユーティリティー (インポート、エクスポート、ロード、AutoLoader、および DPROP など) の使用法について説明しています。	SC88-8522 db2dmx70	db2dm

表 45. DB2 情報 (続き)

資料名	記述	資料番号 PDF ファイル名	HTML ディレクトリー
データウェアハウスセンター 管理の手引き	データウェアハウスセンターを使用してデータウェアハウスを構築および保守する方法を説明します。	SC88-8545 db2ddx70	db2dd
データウェアハウスセンター アプリケーション統合の手引き	プログラマーがアプリケーションをデータウェアハウスセンターおよび情報カタログ・マネージャーと統合するのに役立つ情報を提供します。	SC88-8546 db2adx70	db2ad
DB2 コネクト 使用者の手引き	DB2 コネクト製品の概念、プログラミング、および一般的な使用方法に関する情報を提供します。	SC88-8521 db2c0x70	db2c0
DB2 クエリー・パトローラー 管理の手引き	DB2 クエリー・パトローラー・システムの運用の概説を行い、運用および管理に関する詳細情報、および管理用グラフィカル・ユーザー・インターフェース・ユーティリティーについてのタスク情報を提供します。	SC88-8525 db2dwx70	db2dw
DB2 クエリー・パトローラー 使用者の手引き	DB2 クエリー・パトローラーのツールや関数の使用方法を説明します。	SD88-7277 db2wwx70	db2ww
用語集	DB2 およびそのコンポーネントで使用される用語の定義を示します。 HTML 形式と SQL 解説書 で利用可能	資料番号なし db2t0x70	db2t0
イメージ、オーディオ、およびビデオ・エクステンダー 管理およびプログラミングの手引き	DB2 エクステンダーの一般情報について提供し、画像、音声、およびビデオ (IAV) エクステンダーの管理と構成について、および IAV エクステンダーを使用したプログラミングについて説明しています。さらに、参照情報、診断情報 (メッセージ解説)、およびサンプルも収録されています。	SC88-8609 dmbu7x70	dmbu7
情報カタログ・マネージャー 管理の手引き	情報カタログを管理するためのガイドです。	SC88-8547 db2dix70	db2di
情報カタログ・マネージャー プログラミングの手引きおよび解説書	情報カタログ・マネージャー用の体系化されたインターフェースの定義を示します。	SC88-8549 db2bix70	db2bi

表 45. DB2 情報 (続き)

資料名	記述	資料番号 PDF ファイル名	HTML ディレクトリー
情報カタログ・マネージャー 使用者の手引き	情報カタログ・マネージャー・ユーザー・インターフェースの使用に関する情報を提供します。	SC88-8548 db2aix70	db2ai
インストールおよび構成補足	プラットフォーム固有の DB2 クライアントの計画、インストール、およびセットアップのガイドです。この補足資料には、バインド、クライアント / サーバー通信の設定、 DB2 GUI ツール、DRDA AS、分散インストール、分散要求の構成、および異種データ・ソースへのアクセスについても説明されています。	GC88-8524 db2iyx70	db2iy
メッセージ解説書	DB2、情報カタログ・マネージャー、およびデータウェアハウスセンターから出されるメッセージとコードをリストし、取るべき処置を解説しています。	第 1 巻 GC88-8543 db2m1x70 第 2 巻 GC88-8544 db2m2x70	db2m0
OLAP Integration Server Administration Guide	OLAP Integration Server の Administration Manager 構成要素の使用方法を説明します。	SC27-0782 db2dpx70	n/a
OLAP Integration Server Metaoutline User's Guide	標準の OLAP Metaoutline インターフェースを使用して (Metaoutline Assistant を使用するのではなく) OLAP metaoutline を作成しデータを取り込む方法を説明しています。	SC27-0784 db2upx70	n/a
OLAP Integration Server Model User's Guide	(Model Assistant ではなく) 標準的な OLAP Model Interface を使用して OLAP モデルを作成する方法を説明します。	SC27-0783 db2lpx70	n/a
OLAP のセットアップおよびユーザーズ・ガイド	OLAP スターター・キットの構成およびセットアップに関する情報を提供します。	SC88-8652 db2ipx70	db2ip
Hyperion Essbase スプレッドシート アドインユーザーズ・ガイド for Excel	Excel 作表計算プログラムを使用して OLAP データを分析する方法を説明します。	SC88-8724 db2epx70	db2ep

表 45. DB2 情報 (続き)

資料名	記述	資料番号 PDF ファイル名	HTML ディレクトリー
Hyperion Essbase スプレッドシート アドイン ユーザーズ・ガイド for Lotus 1-2-3	ロータス 1-2-3 作表計算プログラムを使用して OLAP データを分析する方法を説明します。	SC88-8723 db2tpx70	db2tp
レプリケーションの手引きおよび解説書	DB2 に付属の IBM レプリケーション・ツールの計画、構成、管理、および使用方法に関する情報を提供します。	SC88-8550 db2e0x70	db2e0
地理情報エクステンダー使用者の手引きおよび解説書	地理情報エクステンダーのインストール、構成、管理、プログラミング、およびトラブルシューティングに関する情報を提供します。また、地理情報データの概念についての重要事項を示し、地理情報エクステンダー固有の参照情報 (メッセージおよび SQL) を提供します。	SC88-8624 db2sbx70	db2sb
SQL 概説	SQL の概念を紹介し、構造体とタスクの例を多数提供しています。	SC88-8539 db2y0x70	db2y0
SQL 解説書	SQL の構文、セマンティクス、および言語規則について説明します。また、この資料には、各リリース間の互換性、製品の制限事項、およびカタログ・ビューも含まれます。	第 1 巻 SC88-8540 db2s1x70 第 2 巻 SC88-8657 db2s2x70	db2s0
システム・モニター 手引きおよび解説書	データベースおよびデータベース・マネージャーに関連したさまざまな情報を収集する方法を示します。この資料は、この情報を利用して、データベース活動の把握、パフォーマンス向上、および問題原因の判別を行う方法を説明しています。	SC88-8523 db2f0x70	db2f0

表 45. DB2 情報 (続き)

資料名	記述	資料番号 PDF ファイル名	HTML ディレクトリー
テキスト・エクステンダー *管理およびプログラミング*	DB2 エクステンダーの一般情報、テキスト・エクステンダーの管理および構成情報、およびテキスト・エクステンダーを使用したプログラミングの方法について解説します。この資料には、参照情報、診断情報 (メッセージ解説)、およびサンプルが含まれています。	SC88-8610 desu9x70	desu9
問題判別の手引き	エラーの原因の判別、問題からのリカバリー、および DB2 カスタマー・サービスの支援の下での診断ツールの使用法を記載しています。	GD88-7271 db2p0x70	db2p0
新機能	DB2 ユニバーサル・データベース バージョン 7 の新しい機能および拡張機能について説明します。	SC88-8541 db2q0x70	db2q0
DB2 のインストールおよび構成の情報			
DB2 コネクト エンタープライズ・エディション (OS/2 および Windows 版) 概説およびインストール	OS/2 および Windows 32 ビット オペレーティング・システム版の DB2 コネクト エンタープライズ・エディションで、計画、移行、インストール、および構成を行う場合の情報を提供します。また、この資料はサポートされている多数のクライアントのインストールおよびセットアップについても説明します。	GC88-8520 db2c6x70	db2c6
DB2 コネクト エンタープライズ・エディション (UNIX 版) 概説およびインストール	UNIX ベースのプラットフォームでの DB2 コネクト エンタープライズ・エディションの計画、移行、インストール、構成、およびタスクに関する情報を提供します。また、この資料はサポートされている多数のクライアントのインストールおよびセットアップについても説明します。	GC88-8519 db2cyx70	db2cy

表 45. DB2 情報 (続き)

資料名	記述	資料番号 PDF ファイル名	HTML ディレクトリー
DB2 コネクト パーソナル・エディション 概説およびインストール	OS/2 および Windows 32 ビット オペレーティング・システムの DB2 コネクト パーソナル・エディションで、計画、移行、インストール、および構成を行う場合のタスク情報を提供します。また、この資料はサポートされているすべてのクライアントのインストールおよびセットアップについても説明します。	GC88-8533 db2c1x70	db2c1
DB2 コネクト パーソナル・エディション (Linux 版) 概説およびインストール	サポートされる Linux 配布プログラムの DB2 コネクト パーソナル・エディションで、計画、インストール、移行、および構成を行う場合の情報を提供します。	GC88-8528 db2c4x70	db2c4
DB2 データ・リンク・マネージャー 概説およびインストール	AIX および Windows 32 ビット オペレーティング システムの DB2 データ・リンク・マネージャーで、計画、インストール、構成を行う場合の情報を提供します。	GC88-8532 db2z6x70	db2z6
DB2 エンタープライズ拡張エディション (UNIX 版) 概説およびインストール	UNIX ベースのプラットフォームでの DB2 エンタープライズ拡張エディションの計画、インストール、および構成に関する情報を提供します。また、この資料はサポートされている多数のクライアントのインストールおよびセットアップについても説明します。	GC88-8530 db2v3x70	db2v3
DB2 エンタープライズ拡張エディション (Windows 版) 概説およびインストール	Windows 32 ビット オペレーティング・システムの DB2 エンタープライズ拡張エディションで、計画、インストール、および構成を行う場合の情報を提供します。また、この資料はサポートされている多数のクライアントのインストールおよびセットアップについても説明します。	GC88-8529 db2v6x70	db2v6

表 45. DB2 情報 (続き)

資料名	記述	資料番号 PDF ファイル名	HTML ディレクトリー
DB2 ユニバーサル・データベース (OS/2 版) 概説およびインストール	OS/2 オペレーティング・システムでの DB2 ユニバーサル・データベースの計画、インストール、移行、および構成に関する情報を提供します。また、この資料はサポートされている多数のクライアントのインストールおよびセットアップについても説明します。	GC88-8534 db2i2x70	db2i2
DB2 ユニバーサル・データベース (UNIX 版) 概説およびインストール	UNIX ベースのプラットフォームでの DB2 ユニバーサル・データベースの計画、インストール、移行、および構成に関する情報を提供します。 また、この資料はサポートされている多数のクライアントのインストールおよびセットアップについても説明します。	GC88-8536 db2ixx70	db2ix
DB2 ユニバーサル・データベース (Windows 版) 概説およびインストール	Windows 32 ビット オペレーティング・システムの DB2 ユニバーサル・データベースで、計画、インストール、移行、および構成を行う場合の情報を提供します。また、この資料はサポートされている多数のクライアントのインストールおよびセットアップについても説明します。	GC88-8537 db2i6x70	db2i6
DB2 パーソナル・エディション 概説およびインストール	OS/2 および Windows 32 ビット オペレーティング・システム版の DB2 ユニバーサル・データベース パーソナル・エディションで、計画、インストール、移行、および構成を行う場合の情報を提供します。	GC88-8535 db2i1x70	db2i1
DB2 パーソナル・エディション (Linux 版) 概説およびインストール	サポートされる Linux 配布プログラムの DB2 ユニバーサル・データベース パーソナル・エディションで、計画、インストール、移行、および構成を行う場合の情報を提供します。	GC88-8538 db2i4x70	db2i4
DB2 クエリー・パトローラー インストールの手引き	DB2 クエリー・パトローラーのインストール情報を提供します。	GC88-8526 db2iwx70	db2iw

表 45. DB2 情報 (続き)

資料名	記述	資料番号 PDF ファイル名	HTML ディレクトリー
ウェアハウス・マネージャー インストールの手引き	ウェアハウス・エージェント、ウェアハウス・トランスフォーマー、および情報カタログ・マネージャーのインストール情報を提供します。	GC88-8572 db2idx70	db2id
プラットフォーム共通のサンプル・プログラム (HTML 形式)			
サンプル・プログラム (HTML)	DB2 のサポートするすべてのプラットフォームでのプログラム言語用に、サンプル・プログラム (HTML 形式) を提供します。これらのサンプル・プログラムは、参照用としてのみ提供されています。サンプルは、すべてのプログラミング言語で利用できるわけではありません。 HTML サンプルが利用できるのは、 DB2 アプリケーション開発クライアントがインストールされている場合だけです。 プログラムの詳細については、アプリケーション構築の手引き を参照してください。	資料番号なし	db2hs
リリース情報			
DB2 コネクト リリース情報	DB2 コネクトの資料には含められなかった最新の情報が収録されています。	注 #2 を参照してください。	db2cr
DB2 インストール情報	DB2 ブックには含められなかったインストールに関する最新の情報が収録されています。	製品 CD-ROM からのみ利用できます。	
DB2 リリース情報	DB2 ブックには含められなかった DB2 製品とその機能に関する最新の情報が収録されています。	注 #2 を参照してください。	db2ir

注:

1. ファイル名の 6 桁目の文字 *x* は、その資料の言語を表します。たとえば、ファイル名 `db2d0e70` は、*管理の手引き* の英語版であることを示し、ファイル名 `db2d0f70` は同じ資料のフランス語版を示します。資料の言語を表すためにファイル名の 6 桁目で使用されている文字は以下のとおりです。

言語	識別子
ブラジル・ポルトガル語	b
ブルガリア語	u
チェコ語	x
デンマーク語	d
オランダ語	q
英語	e
フィンランド語	y
フランス語	f
ドイツ語	g
ギリシャ語	a
ハンガリー語	h
イタリア語	i
日本語	j
韓国語	k
ノルウェー語	n
ポーランド語	p
ポルトガル語	v
ロシア語	r
簡体字中国語	c
スロベニア語	l
スペイン語	z
スウェーデン語	s
繁体字中国語	t
トルコ語	m

2. DB2 ブックには含められなかった最新の情報が、「リリース情報」で HTML 形式および ASCII ファイルとして利用できます。 HTML 版は、インフォメーション・センターおよび製品 CD-ROM からご利用になれます。 ASCII ファイルの参照方法:
   - UNIX ベースのプラットフォームでは、ファイル `Release.Notes` を参照してください。このファイルは `DB2DIR/Readme/%L` ディレクトリーにあります。ここで `%L` は地域名を、 `DB2DIR` は以下のものを表します。
     - `/usr/lpp/db2_07_01` (AIX の場合)
     - `/opt/IBMdb2/V7.1` (HP-UX、DYNIX/ptx、Solaris、および Silicon Graphics IRIX の場合)
     - `/usr/IBMdb2/V7.1` (Linux の場合)
   - これ以外のプラットフォームでは、ファイル `RELEASE.TXT` を参照してください。このファイルは、製品がインストールされているディレクトリーにあります。OS/2 プラットフォームでは、 **IBM DB2** フォルダーをダブルクリックし、**Release Notes** アイコンをダブルクリックすることもできます。

PDF 資料の印刷

資料のハードコピー版が必要な場合、 DB2 の資料 CD-ROM にある PDF ファイルを印刷することができます。 Adobe Acrobat Reader を使用すれば、資料全体または特定のページを印刷することができます。ライブラリー内の各資料のファイルについては、623ページの表45 を参照してください。

Adobe Acrobat Reader の最新版は、Adobe の Web サイト http://www.adobe.co.jp/ から入手できます。

PDF ファイルは、DB2 の資料 CD-ROM に収録されており、ファイル拡張子 PDF が付いています。 PDF ファイルにアクセスするには以下のようにします。

1. DB2 の資料 CD-ROM を挿入します。 UNIX ベースのプラットフォームの場合は、DB2 資料 CD-ROM をマウントします。マウントの手順については、*概説およびインストール* を参照してください。
2. Acrobat Reader を起動します。
3. 以下に示すいずれかの位置から必要な PDF ファイルを開きます。
   - OS/2 および Windows プラットフォームでは:

     x:¥doc¥*language* ディレクトリー。ここで、*x* は CD-ROM ドライブを、*language* は 2 桁の言語を表す国コード (たとえば、EN は英語) を示します。
   - UNIX ベースのプラットフォームでは:

     CD-ROM の */cdrom*/doc/*%L* ディレクトリー。ここで、*/cdrom* は CD-ROM のマウント・ポイントを、 *%L* は地域名を表します。

さらに、PDF ファイルを CD-ROM からローカル・ドライブまたはネットワーク・ドライブにコピーし、そこから参照することもできます。

印刷資料の注文方法

DB2 オンライン文書

オンライン・ヘルプへのアクセス

すべての DB2 構成要素で、オンライン・ヘルプを利用できます。以下の表に、さまざまな種類のヘルプを示します。

ヘルプの種類	内容	利用方法
コマンド・ヘルプ	コマンド行プロセッサーのコマンド構文について説明します。	コマンド行プロセッサーの対話モードから、次のように入力します。 `? command` ここで *command* はキーワードまたはコマンド全体を表します。 たとえば、`? catalog` と入力すると、すべての CATALOG コマンドに関するヘルプが表示され、`? catalog database` と入力すると、 CATALOG DATABASE コマンドのヘルプが表示されます。
クライアント構成アシスタントのヘルプ コマンド・センターのヘルプ コントロール・センターのヘルプ データウェアハウスセンターのヘルプ イベント・アナライザーのヘルプ 情報カタログ・マネージャーのヘルプ サテライト管理センターのヘルプ スクリプト・センターのヘルプ	そのウィンドウまたはノートブックで実行できるタスクについて説明します。このヘルプは、知っておく必要のある概説および前提条件に関する情報を含みます。また、ウィンドウやノートブックの制御の使用方法を示します。	ウィンドウまたはノートブックから、**「ヘルプ (Help)」**プッシュボタンをクリックするか、または **F1** キーを押します。

ヘルプの種類	内容	利用方法

メッセージ・ヘルプ	メッセージの原因、および取るべき処置を説明します。	コマンド行プロセッサーの対話モードから、次のように入力します。
		`? XXXnnnnn`
		ここで、*XXXnnnnn* は有効なメッセージ識別子を表します。
		たとえば、`? SQL30081` と入力すると、メッセージ SQL30081 に関するヘルプを表示します。
		一度に 1 画面分のメッセージ・ヘルプを表示させるには、次のように入力します。
		`? XXXnnnnn | more`
		メッセージ・ヘルプをファイルに保管するには、次のように入力します。
		`? XXXnnnnn > filename.ext`
		ここで、*filename.ext* はメッセージ・ヘルプを保管するファイルを表します。

SQL ヘルプ	SQL ステートメントの構文について説明します。	コマンド行プロセッサーの対話モードから、次のように入力します。 `help statement` ここで、*statement* は SQL ステートメントを表します。 たとえば、`help SELECT` と入力すると、 SELECT ステートメントのヘルプが表示されます。 **注:** UNIX ベースのプラットフォームでは、SQL ヘルプを利用できません。
SQLSTATE ヘルプ	SQL 状態およびクラス・コードについて説明します。	コマンド行プロセッサーの対話モードから、次のように入力します。 `? sqlstate or ? class code` ここで、*sqlstate* は有効な 5 桁の SQL 状態を、*class code* は SQL 状態の最初の 2 桁を表します。 たとえば、`? 08003` によって SQL 状態 08003 のヘルプが表示され、 `? 08` によってクラス・コード 08 のヘルプが表示されます。

オンライン情報の表示

この製品に付属のブックは、ハイパーテキスト・マークアップ言語 (HTML) ソフトコピー形式です。ソフトコピー形式では情報を検索または表示したり、ハイパーテキスト・リンクを利用して関連情報に移動したりすることができます。また、1 つの端末を超えてライブラリーを容易に共用することができます。

オンライン・ブックやサンプル・プログラムは、 HTML バージョン 3.2 仕様に準拠するすべてのブラウザーを使って表示できます。

オンライン・ブックまたはサンプル・プログラムは、次のようにして表示します。

- DB2 管理ツールを実行している場合、インフォメーション・センターを使用します。
- ブラウザーで、**「ファイル (File)」—>「ページを開く (Open Page)」** をクリックします。次のようなページを開いて、DB2 情報に関する説明とリンクを表示してください。
  - UNIX ベースのプラットフォームでは、以下のページを開きます。

    INSTHOME/sqllib/doc/%*L*/html/index.htm

    ここで *%L* はロケール名です。
  - その他のプラットフォームでは、以下のページを開きます。

    sqllib¥doc¥html¥index.htm

    パスは DB2 がインストールされているドライブです。

    インフォメーション・センターをインストールしていない場合、 **DB2 Information** アイコンをダブルクリックしてページを開くことができます。このアイコンは、ご使用のシステムに応じて、製品のメイン・フォルダー内または Windows 「スタート」メニューにあります。

Netscape ブラウザーのインストール

システムに Web ブラウザーがインストールされていない場合、製品の箱の中にある Netscape CD-ROM から Netscape をインストールすることができます。インストールに関する詳細な説明については、以下を参照してください。

1. Netscape CD-ROM を挿入します。
2. UNIX ベースのプラットフォームでは、CD-ROM をマウントします。マウントの手順については、*概説およびインストール* を参照してください。
3. インストールの手順については、 CDNAV*nn*.txt ファイルを参照します。ここで、*nn* は 2 桁の言語識別子を表します。ファイルは CD-ROM のルート・ディレクトリーにあります。

インフォメーション・センターを使用した情報へのアクセス

インフォメーション・センターを使用すると、DB2 製品情報にすばやくアクセスすることができます。インフォメーション・センターは、DB2 管理ツールを使用できるすべてのプラットフォームで利用できます。

インフォメーション・センターは「インフォメーション・センター (Information Center)」アイコンをダブルクリックすることによってオープンできます。このアイコンのある場所はシステムによって異なります。メイン・プロダクト・フォルダーか Windows の**「スタート」**メニューのどちらかです。

Windows プラットフォームの DB2 では、ツールバーおよび**ヘルプ**・メニューを使用して、インフォメーション・センターにアクセスすることもできます。

インフォメーション・センターは 6 種類の情報を提供します。適切なタブをクリックすると、種類ごとに提供されているトピックが表示されます。

タスク (Tasks) DB2 を使用して実行できる主要なタスク。

参照 (Reference)
DB2 参照情報 (キーワード、コマンド、API など)。

ブック (Books) DB2 ブック。

トラブルシューティング (Troubleshooting)
エラー・メッセージのカテゴリーと、メッセージに対する回復処置。

サンプル・プログラム (Sample Programs)
DB2 アプリケーション開発クライアントに付属のサンプル・プログラム。 DB2 アプリケーション開発クライアントをインストールしていない場合、このタブは表示されません。

Web WWW 上にある DB2 情報。この情報にアクセスするには、ご使用のシステムから Web への接続が必要です。

リストから項目を 1 つ選択すると、インフォメーション・センターはビューアーを立ち上げて情報を表示します。選択した情報の種類に応じて、ビューアーはシステム・ヘルプ・ビューアー、エディター、または Web ブラウザーです。

インフォメーション・センターには検索機能が備わっており、リストを参照せずに特定のトピックを探すことができます。

テキストの全検索を行うには、インフォメーション・センター内のハイパーテキスト・リンク「**DB2 オンライン情報の検索 (Search DB2 Online Information)**」検索フォームに従います。

通常、HTML 検索サーバーは自動的に始動します。 HTML 情報の検索がうまくいかない場合は、以下の方法の 1 つを使用して、検索サーバーを始動しなければならない場合もあります。

Windows では

「スタート」をクリックし、**「プログラム」—>「IBM DB2」—>「Information」—>「Start HTML Search Server」**を選択します。

OS/2 では

「DB2 (OS/2 版)」フォルダーをダブルクリックして、**「Start HTML Search Server」**アイコンをダブルクリックします。

HTML 情報の検索でこの他の問題が発生した場合は、リリース情報を参照してください。

注: 検索機能は、Linux、DYNIX/ptx、および Silicon Graphics IRIX 環境では利用できません。

DB2 ウィザードの使用

ウィザードを使用すると、各タスクをステップごとに進めることによって、さまざまな管理タスクを遂行することができます。ウィザードは、コントロール・センターおよびクライアント構成アシスタントを通して使用できます。以下の表では、ウィザードとその目的をリストしています。

注: データベース作成、索引作成、マルチサイト更新の構成、およびパフォーマンス構成ウィザードは、区分データベース環境で使用できます。

ウィザード	内容	利用方法
データベース追加 (Add Database)	クライアント・ワークステーション上にデータベースのカタログを作成します。	クライアント構成アシスタントから、「**追加 (Add)**」をクリックします。
データベース・バックアップ *(Back up Database)*	バックアップ計画を決定、作成、およびスケジュールします。	「コントロール・センター (Control Center)」からバックアップするデータベースを右クリックし、**「バックアップ (Backup)」—>「ウィザードを使用するデータベース (Database Using Wizard)」**を選択します。
マルチサイト更新の構成 (Configure Multisite Update)	マルチサイト更新、分散トランザクション、または 2 フェーズ・コミットを構成します。	「コントロール・センター (Control Center)」から、**「データベース (Databases)」**フォルダーを右クリックして、**「マルチサイト更新 (Multisite Update)」**を選択します。

ウィザード	内容	利用方法
データベース作成 (Create Database)	データベースを作成し、いくつかの基本的な構成タスクを実行します。	「コントロール・センター (Control Center)」から、**「データベース (Databases)」**フォルダーを右クリックして、**「作成 (Create)」—> 「ウィザードを使用するデータベース (Database Using Wizard)」**を選択します。
表作成 (Create Table)	基本的なデータ・タイプを選択して、表の基本キーを作成します。	「コントロール・センター (Control Center)」から、**「表 (Tables)」**アイコンを右クリックして、**「作成 (Create)」—>「ウィザードを使用する表 (Table Using Wizard)」**を選択します。
表スペース作成 (Create Table Space)	新しい表スペースを作成します。	「コントロール・センター (Control Center)」から、**「表スペース (Table Spaces)」**アイコンを右クリックして、**「作成 (Create)」—> 「ウィザードを使用する表スペース (Table Space Using Wizard)」**を選択します。
索引作成 (Create Index)	すべての照会について、作成すべき索引および除去すべき索引を提案します。	「コントロール・センター (Control Center)」から、**「索引 (Index)」**アイコンを右クリックして、**「作成 (Create)」—>「ウィザードを使用する索引 (Index Using Wizard)」**を選択します。
パフォーマンス構成 (Performance Configuration)	ビジネス要件に適合するように構成パラメーターを更新して、データベースのパフォーマンスを調整します。	「コントロール・センター (Control Center)」から、調整したいデータベースを右クリックして、**「ウィザードを使用するパフォーマンスの構成 (Configure Performance Using Wizard)」**を選択します。 区分データベース環境では、「Database Partitions」視点から、調整したい最初のデータベース区画を右クリックして、**「ウィザードを使用するパフォーマンスの構成 (Configure Performance Using Wizard)」**を選択します。

ウィザード	内容	利用方法
データベース復元 *(Restore Database)*	障害の後、データベースを回復します。どのバックアップを使用し、どのログを再生するかの判別を支援します。	「コントロール・センター (Control Center)」から復元するデータベースを右クリックし、**「復元 (Restore)」—>「ウィザードを使用するデータベース (Database Using Wizard)」**を選択します。

文書サーバーのセットアップ

デフォルトでは、DB2 情報はローカル・システムにインストールされます。つまり、DB2 情報にアクセスする必要のある各担当者が同じファイルをインストールする必要があります。 DB2 情報を 1 か所に格納するには、次のようにします。

1. ¥sqllib¥doc¥html のすべてのファイルとサブディレクトリーを、ローカル・システムから Web サーバーにコピーします。各ブックには独自のサブディレクトリーがあり、そのブックを構成する必要な HTML および GIF ファイルが入っています。ディレクトリー構造は常に同じ状態に保つ必要があります。
2. Web サーバーを構成して、ファイルを新しい場所で検索するようにします。さらに詳しい情報については、 *インストールおよび構成 補足* の NetQuestion 付録を参照してください。
3. インフォメーション・センターの Java バージョンをご使用の場合は、すべての HTML ファイルのベース URL を指定できます。この URL はブックのリストに使用してください。
4. 資料ファイルが表示されるようになったなら、よく使うトピックにはブックマークを付けておいてください。ブックマークを付けるページは、たとえば以下のものがあります。
   - ブックのリスト
   - 頻繁に使用されるブックの目次
   - 頻繁に参照する情報 (たとえば、ALTER TABLE トピックなど)
   - 検索フォーム

中央のマシンから DB2 ユニバーサル・データベース オンライン文書ファイルを提供する方法については、 *インストールおよび構成 補足* の NetQuestion 付録を参照してください。

オンライン情報の検索

HTML ファイルの情報を検索するには、以下の方法のどれか 1 つを使用してください。

- 最上部にある**「検索 (Search)」**をクリックします。検索フォームを使用して特定のトピックを見つけます。この機能は、 Linux、DYNIX/ptx、または Silicon Graphics IRIX 環境ではご利用になれません。
- 最上部にある**「索引 (Index)」**をクリックします。索引を使用して、ブック内の特定のトピックを見つけます。
- HTML 資料またはヘルプの目次あるいは索引を表示してから、 Web ブラウザーの検索機能を利用してブック内の特定のトピックを見つけます。
- Web ブラウザーのブックマーク機能を使用して、特定のトピックにすばやく戻ります。
- インフォメーション・センターの検索機能を使用して、特定のトピックを検索します。詳しくは、637ページの『インフォメーション・センターを使用した情報へのアクセス』を参照してください。

付録F. 特記事項

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、プログラムまたはサービスの操作性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。実施権の許諾については、下記の宛先に書面にてご照会ください。

〒106-0032 東京都港区六本木 3 丁目 2-31
AP 事業所
IBM World Trade Asia Corporation
Intellectual Property Law & 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。

本書は定期的に見直され、必要な変更 (たとえば、技術的に不適切な表現や誤植など) 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 Canada Ltd.
Office of the Lab Director
1150 Eglinton Avenue East
Toronto, Ontario
M3C 1H7
CANADA

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 以外の製品に関する情報は、その製品の供給者、出版物、もしくはその他の公に利用可能なソースから入手したものです。IBM は、それらの製品のテストは行っておりません。したがって、他社製品に関する実行性、互換性、またはその他の要求については確証できません。 IBM 以外の製品の性能に関する質問は、それらの製品の供給者にお願いします。

IBM の将来の方向または意向に関する記述については、予告なしに変更または撤回される場合があり、単に目標を示しているものです。

本書には、日常の業務処理で用いられるデータや報告書の例が含まれています。より具体性を与えるために、それらの例には、個人、企業、ブランド、あるいは製品などの名前が含まれている場合があります。これらの名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。

それぞれの複製物、サンプル・プログラムのすべての部分、またはすべての派生した創作物には、次のように、著作権表示を入れていただく必要があります。

© (お客様の会社名) (西暦年). このコードの一部は、IBM Corp. のサンプル・プログラムから取られています。 © Copyright IBM Corp. _年を入れる_. All Rights Reserved.

商標

アスタリスク (*) 付きの以下の用語は、IBM Corporation の商標です。

ACF/VTAM
AISPO
AIX
AIX/6000
AIXwindows
AnyNet
APPN
AS/400
BookManager
CICS
C Set++
C/370
DATABASE 2
DataHub
DataJoiner
DataPropagator
DataRefresher
DB2
DB2 Connect
DB2 Extenders
DB2 OLAP Server
DB2 Universal Database
Distributed Relational Database Architecture
DRDA
eNetwork
Extended Services
FFST
First Failure Support Technology
IBM
IMS
IMS/ESA
LAN DistanceMVS
MVS/ESA
MVS/XA
Net.Data
OS/2
OS/390
OS/400
PowerPC
QBIC
QMF
RACF
RISC System/6000
RS/6000
S/370
SP
SQL/DS
SQL/400
System/370
System/390
SystemView
VisualAge
VM/ESA
VSE/ESA
VTAM
WebExplorer
WIN-OS/2

以下は、それぞれ各社の商標または登録商標です。

Tivoli および、NetView は、Tivoli Systems, Inc. の商標です。

Microsoft、Windows、Windows NT および Windows ロゴは、Microsoft Corporation の米国およびその他の国における商標です。

Java およびすべての Java 関連の商標およびロゴは、Sun Microsystems, Inc. の米国およびその他の国における商標または登録商標です。

UNIX は、The Open Group がライセンスしている米国およびその他の国における登録商標です。

他の会社名、製品名およびサービス名などはそれぞれ各社の商標または登録商標です。

索引

日本語, 数字, 英字, 特殊文字の順に配列されています。なお, 濁音と半濁音は清音と同等に扱われています。

［ア行］

[カ行]

[サ行]

[タ行]

[ナ行]

[ハ行]

[マ行]

[ヤ行]

[ラ行]

[数字]

A

B

C

D

E

F

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z

IBM と連絡をとる

技術上の問題がある場合は、時間をとって*問題判別の手引き*に定義されている処置を検討し、それらの提案を実行した後で、DB2 顧客サービスに連絡をとってください。この資料には、DB2 顧客サービスがお客さまを支援するために必要とする情報が説明されています。

製品情報

以下の情報は英語で提供されます。内容は英語版製品に関する情報です。

http://www.ibm.com/software/data/

DB2 World Wide Web ページには、ニュース、製品説明、研修スケジュールなどの DB2 に関する最新情報が提供されています。ただし、提供されている情報は英語です。

http://www.ibm.com/software/data/db2/library/

「DB2 Product and Service Technical Library」では、よくされる質問 (FAQ)、修正内容、資料、および最新の DB2 技術情報などの情報へのアクセスが提供されています。

注: この情報のご提供は英語のみとなりますのでご注意ください。

http://www.elink.ibmlink.ibm.com/pbl/pbl/

「International Publications」注文用 Web サイトでは、マニュアルの注文方法についての情報を提供しています。ただし、提供されている情報は英語です。

http://www.ibm.com/education/certify/

IBM の「Professional Certification Program」Web サイトでは、 DB2 を含むさまざまな IBM 製品の認証テストの情報を提供しています。ただし、提供されている情報は英語です。

ftp.software.ibm.com

匿名でログオンしてください。ディレクトリー `/ps/products/db2` には、 DB2 および多数の他製品に関連したデモ、修正プログラム、情報、およびツールがあります。ただし、提供されている情報は英語です。

comp.databases.ibm-db2, bit.listserv.db2-l

これらのインターネット・ニュースグループは、ユーザーが DB2 製品に関する自分の経験について話し合うために利用できます。ただし、提供されている情報は英語です。

Compuserve: GO IBMDB2

このコマンドを入力すると、IBM DB2 Family forum にアクセスできます。すべての DB2 製品が、このフォーラムでサポートされています。ただし、提供されている情報は英語です。

米国以外の国で IBM に連絡する方法については、 *IBM Software Support Handbook* の Appendix A を参照してください。この資料にアクセスするには、 Web ページ: http://www.ibm.com/support/ にアクセスし、ページの最下部にある「IBM Software Support Handbook」リンク・ボタンを選択します。

注: 国によっては、 IBM が承認している販売業者が、 IBM サポート・センターの代わりにそれら販売業者のサポート・センターに連絡する場合があります。

Printed in Japan

SC88-8512-01

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