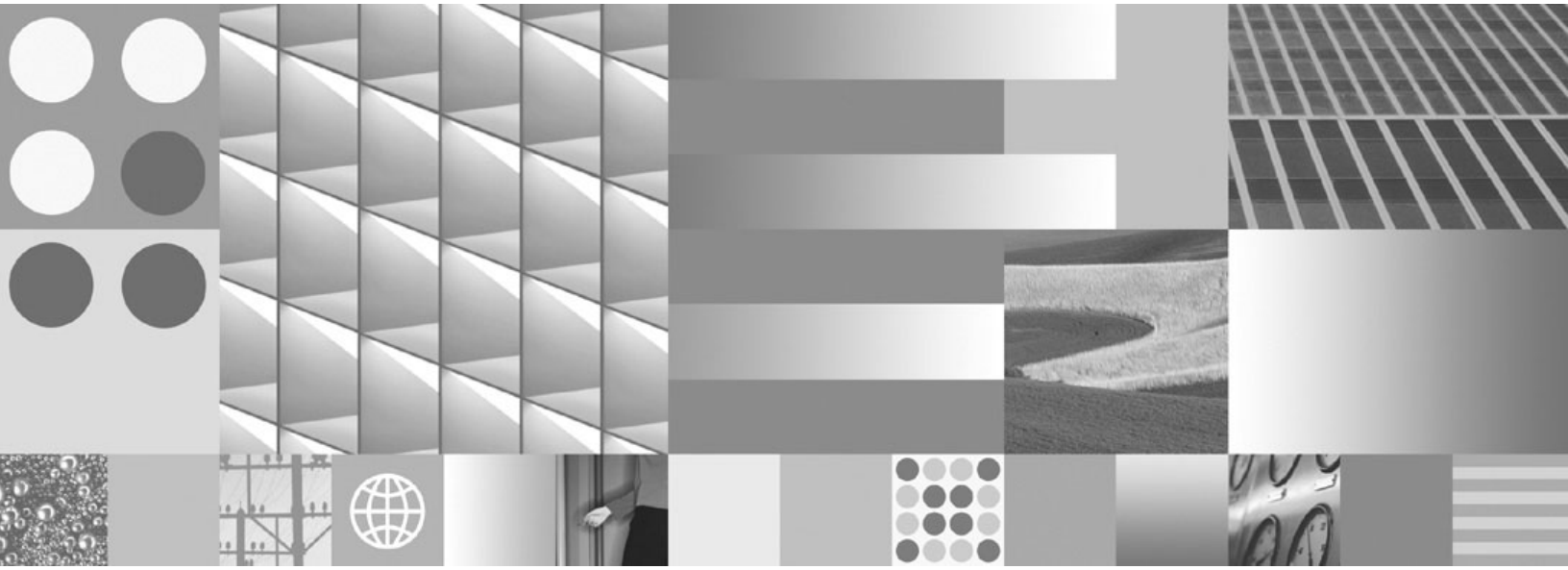
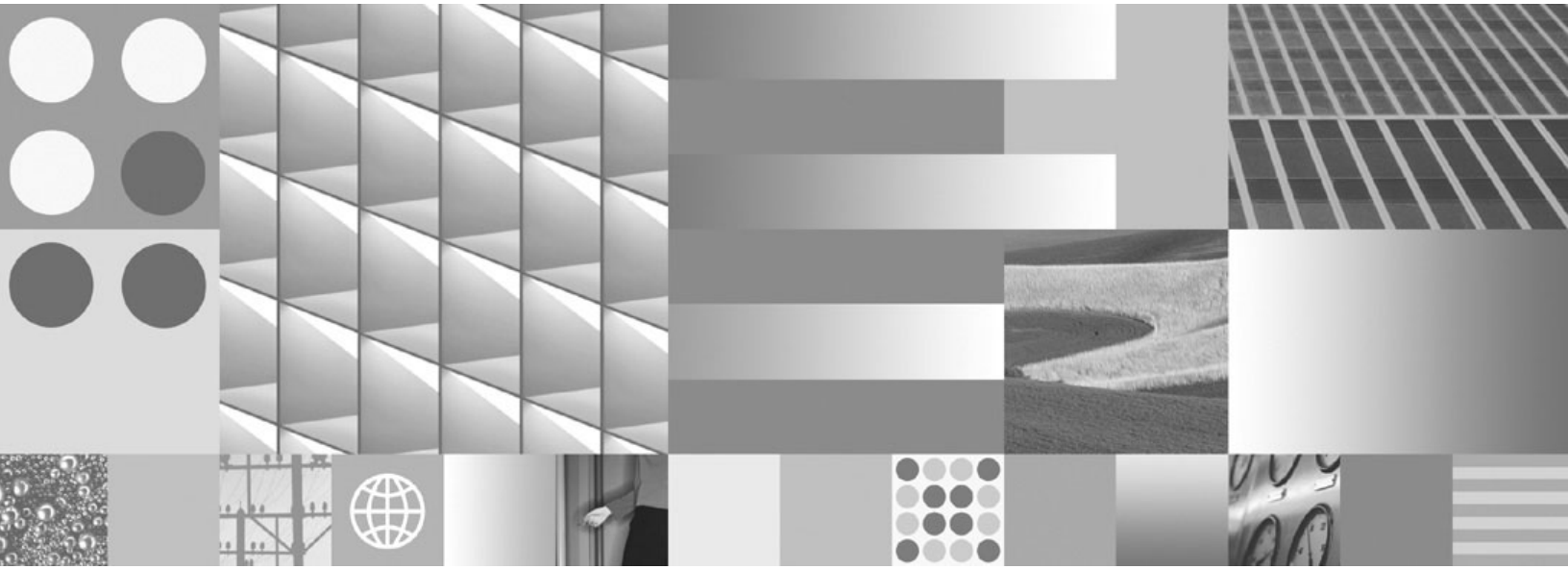
DB2 Version 9.5
for Linux, UNIX, and Windows



データベース・パフォーマンスのチューニング

SC88-4417-00
(英文原典：SC23-5867-00)

DB2 Version 9.5
for Linux, UNIX, and Windows



データベース・パフォーマンスのチューニング

SC88-4417-00
(英文原典：SC23-5867-00)

ご注意

本書および本書で紹介する製品をご使用になる前に、 477 ページの『付録 B. 特記事項』に記載されている情報をお読みください。

原　典： SC23-5867-00
DB2 Version 9.5 for Linux, UNIX, and Windows
Tuning Database Performance

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 2007.10

目次

第 1 部 パフォーマンスのエレメント

パフォーマンス とは、特定のワークロード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。

- システムで使用可能なリソース
- リソースの使用頻度と共用の程度

一般に、システムを調整することにより、その費用対効果の比率を向上させます。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、より大きいワークロードまたは要求がより多いワークロードを処理する

  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ワークロードを増やすことができます。
- 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
- ユーザーに対するサービスを低下させることなく、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察してください。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。

第 1 章 パフォーマンス・チューニングのガイドライン

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、ほとんどの場合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 および XQuery 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。

第 2 章 パフォーマンスの向上プロセスの開発

パフォーマンスの向上プロセスは、パフォーマンスのモニターおよびチューニングの局面への、反復的かつ長期的なアプローチです。モニターの結果に基づいて、ユーザーおよびユーザーのパフォーマンス・チームはデータベース・サーバーの構成を調整し、データベース・サーバーを使用するアプリケーションに変更を加えます。

パフォーマンスのモニターおよびチューニングは、データ、およびデータ・アクセスのパターンを使用するアプリケーションの種類に関するユーザーの知識に基づいて決定します。パフォーマンス要件は、アプリケーションの種類によって異なります。

以下のパフォーマンスの向上プロセスの概要を指針として検討してください。

パフォーマンスの向上プロセスを作成するには、以下を行います。

1. パフォーマンスの目標を定義します。
2. システムで主要な制約のパフォーマンス指針を設定します。
3. パフォーマンスのモニター計画を立て、それを実行します。
4. 継続的にモニター結果を分析し、チューニングを必要とするリソースを判別します。
5. 一度に 1 つの調整を行う。

   複数のリソースがチューニングを必要としていると判断する場合、あるいは調整するリソースでいくつかのチューニング・オプションを使用できる場合でも、変更は一度に 1 つのみにしてください。そうすることで、そのチューニング努力によって得られる結果は必ず望みどおりのものとなります。ある時点で、データベース・サーバーおよびアプリケーションのチューニングをしても、パフォーマンスが向上しなくなる場合があります。その場合は、ハードウェアをアップグレードする必要があります。

実際のパフォーマンス・チューニングでは、システム・リソース間のトレードオフが必要になります。たとえば、入出力パフォーマンスを向上させるために、バッファー・プール・サイズを増やすことができます。しかし、バッファー・プールを増やすにはより多くのメモリーを必要とするため、パフォーマンスの他の局面は低下する可能性があります。

第 3 章 ユーザーが提供できるパフォーマンス情報

システム・チューニングが必要なことを示唆する最初の兆候は、ユーザーからの問題報告かもしれません。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。通常は 2、3 の簡単な質問をするだけで、問題を突き止めるためにどこを調べればよいか、見当がつきます。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?
- 他のユーザーも同じような問題に直面していますか? 問題を訴えているユーザーは 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同じ問題に直面している場合、それら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?
- 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問題と思われますか?
- 問題の発生の仕方に何らかのパターンがありますか? たとえば、一日の特定の時間 (昼休み中など) に問題が発生しますか、それともある程度連続的に発生しますか?

第 4 章 パフォーマンス・チューニングの限界

チューニングによってシステム効率を改善することには、限界があります。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。

たとえば、システムにボトルネックが発生する場合には、チューニングによってパフォーマンスが改善されます。システム・パフォーマンスの限界に近づいているとき、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、どのようにシステムをチューニングして、このパフォーマンス低下を相殺すべきかを判別する必要があります。

しかし、チューニングを行ってもこれ以上は効果がないという限界点が存在します。それに達したときは、目標と期待値を環境における限界の範囲内に修正する必要があります。パフォーマンスを大幅に改善するには、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、またはこれらを組み合わせる必要があるかもしれません。

第 5 章 DB2 アーキテクチャーおよびプロセスの概要

DB2® アーキテクチャーおよびプロセスに関する一般情報により、特定のトピックに提供される詳細情報を理解しやすくなります。

次の図に、IBM® DB2 バージョン 9.1 のアーキテクチャーとプロセスの概要を示します。

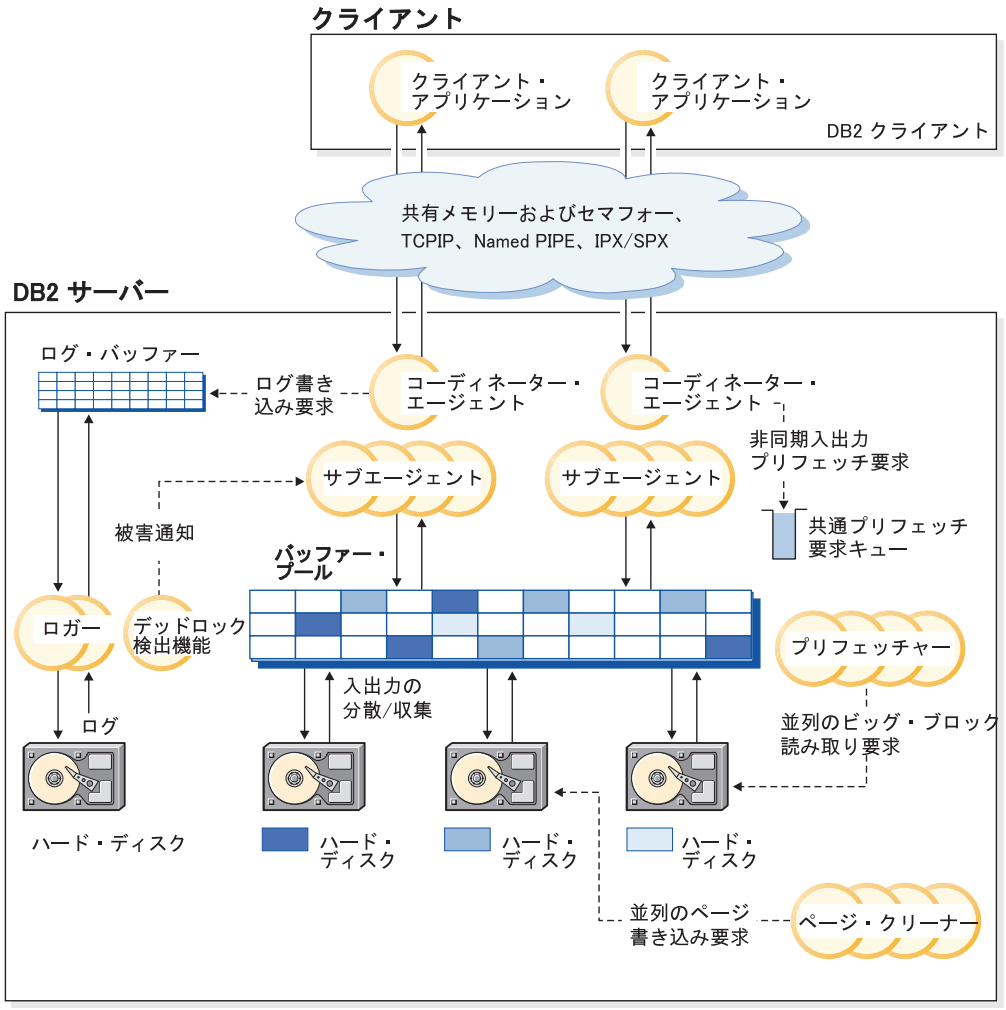


図 1. アーキテクチャーとプロセスの概要

クライアント・サイドでは、ローカルまたはリモート・アプリケーションのどちらかまたは両方が、 DB2 クライアント・ライブラリーとリンクしています。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)、または TCP/IP 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。このセクションのすべての図で、 EDU は、円または円グループ

として示されています。 EDU は、バージョン 9.5 のすべてのプラットフォーム上でスレッドとしてインプリメントされます。 DB2 エージェントは、最も一般的なタイプの EDU です。これらのエージェントは、アプリケーションに代わって SQL および XQuery 処理のほとんどを実行します。 プリフェッチャーおよびページ・クリーナーは他の共通 EDU です。

サブエージェントのセットは、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ことがあります。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、またはそのマシンがパーティション・データベースの一部である場合、複数のサブエージェントを割り当てることができます。たとえば、対称マルチプロセッシング (SMP) 環境では、複数の SMP サブエージェントが、多くのプロセッサーを活用することができます。

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 の作成および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、データベース・サーバー・メモリーのエリアであり、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、データベースのパフォーマンスのレベルを決定する主要なものとなります。アプリケーションが必要な分より多くのデータがバッファー・プールに存在する場合、ディスクで検索するよりも早くそのデータにアクセスすることができます。

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、データにアクセスする時間、およびアプリケーションに対するデータの準備状態を制御します。

- **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・プールに移動します。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スキャン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プリフェッチ・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。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。
- **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します。ページ・クリーナーはアプリケーション・エージェントから独立したバックグラウンド EDU です。これらは変更されたページを検出し、その変更されたページをディスクに書き込みます。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。

DB2 プロセス・モデル

DB2 プロセス・モデルの知識は、データベース・マネージャーおよび関連したコンポーネントの対話を理解するのに役立つため、問題の性質を判別するのに役立ちます。

すべての DB2 サーバーによって使用されるプロセス・モデルは、データベース・サーバーおよびクライアントと、ローカル・アプリケーション間に発生する通信を容易にします。また、データベース・アプリケーションが、データベース制御ブロックおよび重要なデータベース・ファイルから分離されていることも確認します。

DB2 サーバーは多くの様々なタスクを実行する必要があります。例えば、データベース・アプリケーション要求を処理したり、ログ・レコードがディスクに書き出されたことを確認したりします。それぞれのタスクは通常、個別のエンジン・ディスパッチ可能単位 (EDU) によって実行されます。前のリリースでは、ほとんどの EDU は、Linux® および UNIX® 環境では個別のプロセスを使用し、Windows ではメイン DB2 サーバー・プロセス内のオペレーティング・システム・スレッドを使用してインプリメントされていました。バージョン 9.5 からは、Linux および UNIX 環境の DB2 サーバーもスレッド化されており、そのために UNIX と Windows のどちらにおいても、EDU はオペレーティング・システム・スレッドを使用してインプリメントされています。

DB2 サーバーでマルチスレッド化アーキテクチャーを使用することには、多くの利点があります。新規スレッドでは、必要なメモリーおよびオペレーティング・システム・リソースがプロセスと比べて少なくなります。なぜなら、一部のオペレーティング・システム・リソースを同じプロセス内のすべてのスレッドの間で共有できるからです。また、一部のプラットフォームでは、スレッド用のコンテキスト切り替え時間がプロセスの場合と比べて低コストになるので、パフォーマンスを改善できます。しかし、最も重要な点として、すべてのプラットフォーム上でスレッド・モデルを使用すると、割り振る EDU を必要に応じて増やすことが簡単になり、複数の EDU によって共有する必要のあるメモリーを動的に割り振ることができるので (プロセス・ベース・モデルでは、ある EDU が別の EDU から割り振られたメモリーを見ることはできません)、DB2 サーバーを構成しやすくなります。これらの機能拡張について詳しくは、『メモリー構成の単純化』および『エージェントおよびプロセス・モデルの構成』のセクションを参照してください。

Linux および UNIX システムの前のリリースでは、ps システム・コマンドまたは db2_local_ps コマンドを使用して、アクティブな DB2 EDU をすべてリストすることができました。バージョン 9.5 からは、それらのコマンドは db2sysc プロセス内の EDU スレッドをリストしなくなります。その代わりとして、-edus オプションを付けて db2pd コマンドを指定して、アクティブな EDU スレッドをすべてリストできるようになりました。このコマンドは、UNIX と Windows システムのどちらでも有効です。

アクセスされるそれぞれのデータベースごとに、さまざまな EDU が開始され、プリフェッチ、通信、およびロギングなどのさまざまなデータベース・タスクを扱います。データベース・エージェントは、データベースのアプリケーション要求を処理するために作成された EDU の特殊なクラスです。

それぞれのクライアント・アプリケーション接続は、データベースを操作する単一の「**コーディネーター・エージェント**」を持ちます。コーディネーター・エージェントは、アプリケーションの代わりに作業を行い、専用メモリー、プロセス間通信 (IPC) または遠隔通信プロセスを必要に応じて使用して、他のエージェントと通信を行います。

DB2 アーキテクチャーは、アプリケーションが DB2 とは違うアドレス・スペースで実行されるように、**ファイアウォール**を提供します。ファイアウォールは、データベースおよびデータベース・マネージャーを、アプリケーション、ストアード・プロシージャー、およびユーザー定義関数 (UDF) から保護します。ファイアウォールは、アプリケーション・プログラミング・エラーが、内部バッファーまたはデータベース・マネージャーのファイルを上書きしないようにするため、データベース内のデータの保全性を保守します。ファイアウォールはまた、アプリケーション・エラーがデータベース・マネージャーを破壊しないため、信頼性も向上させます。

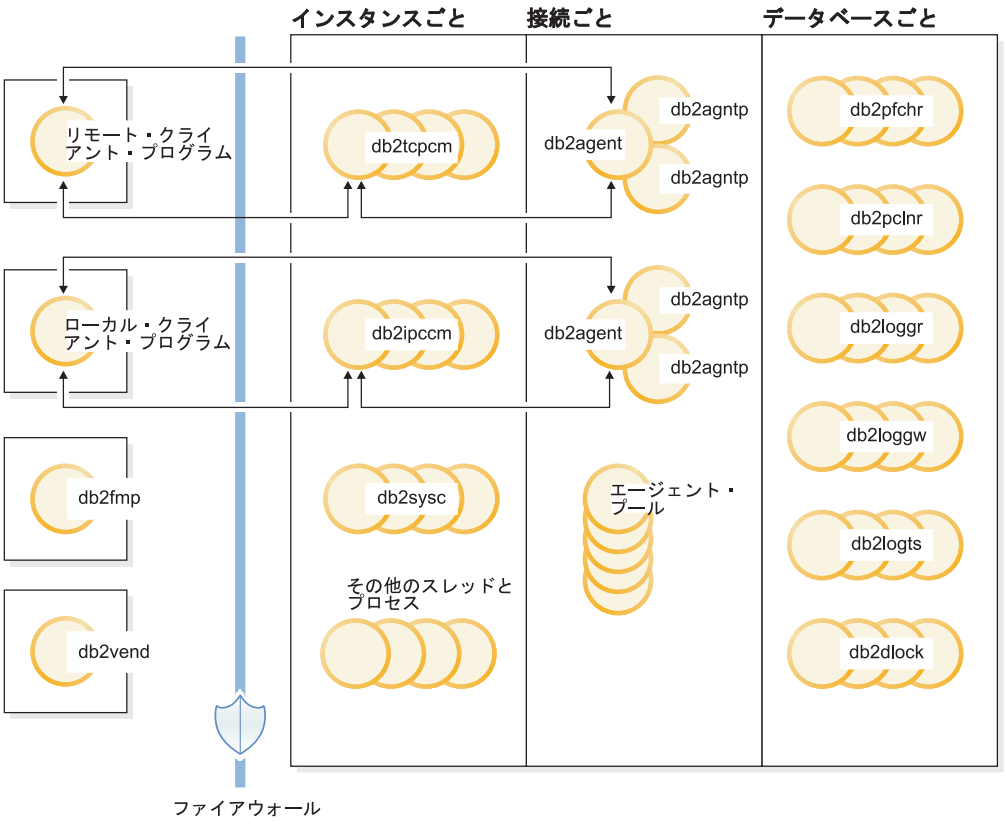


図 2. DB2 システムのプロセス・モデル

以下のリストは、図に表示されたオブジェクトの追加の詳細を提供します。

クライアント・プログラム

クライアント・プログラムは、データベース・サーバーのリモート側で、または同じマシンで実行します。これは、データベースとの最初の接触を、リスナーを通じて行います。コーディネーター・エージェント (db2agent) は、それから割り当てられます。

リスナー

クライアント・プログラムは、DB2 の開始時に開始された通信リスナーとの最初の接触を持ちます。それぞれの構成された通信プロトコルにはリスナーがあり、ローカル・クライアント・プログラムには、プロセス間通信 (IPC) リスナー (db2ipccm) があります。リスナーには以下が含まれます。

- **db2ipccm**、クライアント接続用
- **db2tcpcm**、TCP/IP 接続用
- **db2tcpdm**、TCP/IP ディスカバリー・ツール要求用

エージェント

ローカルまたはリモートのいずれかの、クライアント・アプリケーションからのすべての接続要求は、対応するコーディネーター・エージェント (**db2agent**) に割り振られます。コーディネーター・エージェントが作成されると、これはアプリケーションの代わりにすべてのデータベース要求を実行します。

データベース・パーティション・フィーチャー (DPF) が有効な環境または *intra_query* 並列処理が有効にされている環境では、コーディネーター・エージェントはデータベース要求をサブエージェント (それぞれ db2agntp および db2agnts) に分散します。これらのエージェントは、アプリケーションの要求を実行します。一度コーディネーター・エージェントが作成されると、これはデータベース上の要求を実行するサブエージェント (db2agntp) を調整することで、そのアプリケーションの代わりにすべてのデータベース要求をハンドルします。アプリケーションと関連しているものの、現在アイドル状態であるサブエージェントは、名前 **db2agnta** によって識別されます。

コーディネーター・エージェントは、以下の可能性があります。

- 別名でデータベースに接続されている。たとえば、″db2agent (DATA1)″ がデータベース別名 ″DATA1″ に接続されています。
- インスタンスにアタッチされている。たとえば、″db2agent (user1)″ がインスタンス ″user1″ にアタッチされています。

DB2 プロセス・モデルは他のタイプのエージェント (独立コーディネーター・エージェントまたはサブコーディネーター・エージェントなど) をインスタンス化して、特定の操作を実行します。例えば、独立コーディネーター・エージェント **db2agnti** を使用してイベント・モニターを実行したり、サブコーディネーター・エージェント **db2agnsc** を使用して、データベースの突然のシャットダウンの後に再始動の実行を並行して行います。

アイドルのエージェントはエージェント・プールにあります。これらのエージェントは、クライアント・プログラムの代わりとして操作するコーディネーター・エージェント、または既存のコーディネーター・エージェントの代わりとして操作するサブエージェントからの要求に対して使用可能です。適切なサイズのアイドル・エージェント・プールがあれば、重要なアプリケーション・ワークロードを持つ構成においてパフォーマンスを向上させることができます。なぜなら、アイドル・エージェントは必要に応じて直ちに使用でき (スレッドの作成およびメモリーと他のリソースの割り振りと開始を含む)、アプリケーション接続ごとに全く新規のエージェントを割り振る必要がないからです。バージョン 9.5 から、DB2 は希望に応じてア

イドル・エージェント・プールのサイズを自動的に管理することもできます。

db2fmp

fenced モード・プロセスです。これは、ファイアウォールの外で fenced ストアード・プロシージャーおよびユーザー定義関数を実行責任があります。db2fmp プロセスは常に分離されたプロセスですが、実行するルーチンのタイプによっては、マルチスレッドの場合があります。

db2vend

これは、インスタンスがログ・アーカイブ用のユーザー出口プログラムを実行するために、EDU の代わりにベンダー・コードを実行するプロセスです (UNIX のみ)。

データベース EDU

以下にリストには、各データベースによって使用される、いくつかの重要な EDU が含まれています。

- **db2pfchr**、バッファー・プール・プリフェッチャー用
- **db2pclnr**、バッファー・プール・ページ・クリーナー用
- **db2loggr**、トランザクション処理およびリカバリーをハンドルするログ・ファイルの取扱用
- **db2loggw**、ログ・ファイルへのログ・レコードの書き込み用
- **db2logts**、どのログ・ファイルでどの表スペースがログ・レコードを持つかをトラッキングする。この情報は、データベース・ディレクトリーの DB2TSCHG.HIS ファイルに記録されます。これは、表スペース・ロールフォワード・リカバリーの速度を上げるために使用されます。
- **db2dlock**、デッドロック検出用。マルチパーティション・データベース環境では、各パーティションの db2dlock EDU から集められた情報を調整するのに、**db2glock** と呼ばれる追加スレッドが使用されます。 db2glock はカタログ・パーティションでのみ実行可能です。
- **db2taskd**、バックグラウンド・データベース・タスクの分散用。タスクは **db2taskp** と呼ばれるスレッドによって実行されます。
- **db2hadrp**、HADR 1 次サーバー・スレッド
- **db2hadrs**、HADR スタンバイ・サーバー・スレッド
- **db2lfr**、個別のログ・ファイルを処理するログ・ファイル・リーダー用。
- **db2shred**、ログ・ページ内で個別のログ・レコードを処理する
- **db2redom**、再実行マスター用。リカバリー中に、再実行ログ・レコードを処理し、ログ・レコードを処理のために再実行作業に割り当てます。
- **db2redow**、再実行作業用。リカバリー中に、再実行マスターの要求で再実行ログ・レコードを処理します。
- **db2logmgr**、ログ・マネージャー用。リカバリー可能なデータベースのログ・ファイルを管理します。
- **db2wlmd**、ワークロード管理統計の自動収集用。
- イベント・モニター・スレッドは、以下のように識別されます。
  - **db2evm%1%2 (%3)** なお、**%1** の部分は以下のようになります。

- **g** - グローバル・ファイル・イベント・モニター
- **l** - ローカル・ファイル・イベント・モニター
- **t** - 表イベント・モニター
- **gp** - グローバル・パイプ・イベント・モニター
- **lp** - ローカル・パイプ・イベント・モニター

%2 の部分は以下のようになります。

- **i** - コーディネーター
- **p** - コーディネーターでない

また、**%3** はイベント・モニター名です。

- バックアップおよびリストア・スレッドは、以下のように識別されます。
  - **db2bm.%1.%2** は、バックアップおよびリストアのバッファー・マニピュレーター。**db2med.%1.%2** は、バックアップおよびリストアのメディア・コントローラー。ここで % の部分は以下のようになります。
    - **%1**- バックアップまたはリストアのセッションを制御するエージェントの EDU ID
    - **%2**- 特定のバックアップまたはリストアのセッションに属するスレッド (多数の場合もある) の間で、あいまいさをなくすのに使用される順次値

    例えば、db2bm.13579.2 は EDU ID 13579 の db2agent スレッドによって制御される 2 番目の db2bm スレッドです。

Database Server スレッドおよびプロセス

データベース・サーバーを実行するには、システム・コントローラー (UNIX の場合は **db2sysc**、**Windows の場合は db2syscs.exe**) が存在する必要があります。また、さまざまなタスクを実行するために、以下のスレッドおよびプロセスが開始される場合があります。

- **db2resync**、グローバル再同期リストをスキャンする再同期エージェント
- **db2wdog**、異常終了をハンドルする UNIX およびLinux オペレーティング・システムのウォッチドッグ
- **db2fcms**、高速コミュニケーション・マネージャーの送信側デーモン
- **db2fcmr**、高速コミュニケーション・マネージャーの受信側デーモン
- **db2pdbc**、リモート・ノードからの並列要求をハンドルする並列システム・コントローラー (マルチパーティション・データベース環境でのみ使用)
- **db2cart**、USEREXIT が使用可能に構成されたデータベースへのアクセス時のログ・ファイルのアーカイブ用
- **db2fmtlg**、LOGRETAIN が使用可能に、しかし USEREXIT が使用不可に構成されたデータベースへのアクセス時のログ・ファイルのフォーマット用
- **db2panic**、エージェント限度が特定のノードに達した後の緊急要求をハンドルするパニック・エージェント (パーティション・データベース環境でのみ使用)
- **db2srvlst**、DB2 for z/OS などのシステムのアドレスのリストを管理する
- **db2fmd**、障害モニター・デーモン
- **db2disp**、クライアント接続コンセントレーターのディスパッチャー

- **db2acd**、ヘルス・モニターおよび自動保守ユーティリティーをホストするオートノミック・コンピューティング・デーモン。このプロセスは以前は **db2hmon** と呼んでいました。
- **db2licc**、インストール済みの DB2 ライセンスを管理する
- **db2thcln**、EDU の終了時にリソースをリサイクルする (UNIX のみ)
- **db2aiothr**、データベース・パーティションの非同期入出力要求を管理する (UNIX のみ)
- **db2alarm**、要求されたタイマーの期限が切れたときに EDU に通知する (UNIX のみ)
- **db2sysc**、メイン・システム・コントローラー EDU は、重要な DB2 サーバー・イベントを処理する

デッドロック

デッドロックは、2 つのアプリケーションが他方が必要とするデータを互いにロックし、その結果、いずれのアプリケーションも実行を継続できないときに作成されます。 例えば、次の図では、アプリケーション A およびアプリケーション B の 2 つのアプリケーションが並行して実行します。アプリケーション A の最初のステップは表 1 の最初の行を更新することであり、2 番目のステップは表 2 の 2 番目の行を更新することです。アプリケーション B はまず表 2 の 2 番目の行を更新してから、次に表 1 の最初の行を更新します。ある時点、T1 で、アプリケーション A はその最初のステップを実行し、表 1 の最初の行を更新するためにロックします。同時に、アプリケーション B が表 2 の 2 番目の行を更新のためにロックします。 T2 で、アプリケーション A は次のステップを実行しようとし、表 2 の 2 番目の列を更新のためにロックすることを要求します。 しかし、同時に、アプリケーション B が表 1 の最初の行をロックして更新しようとします。アプリケーション A は表 2 の 2 番目の行の更新が完了するまでは表 1 の最初の行のロックを解放しようとせず、アプリケーション B は表 1 の最初の行をロックして更新できるようになるまでは表 2 の 2 番目の行のロックを解放しようとしないので、デッドロックが発生します。 アプリケーションは、 1 つ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。

デッドロックの概念

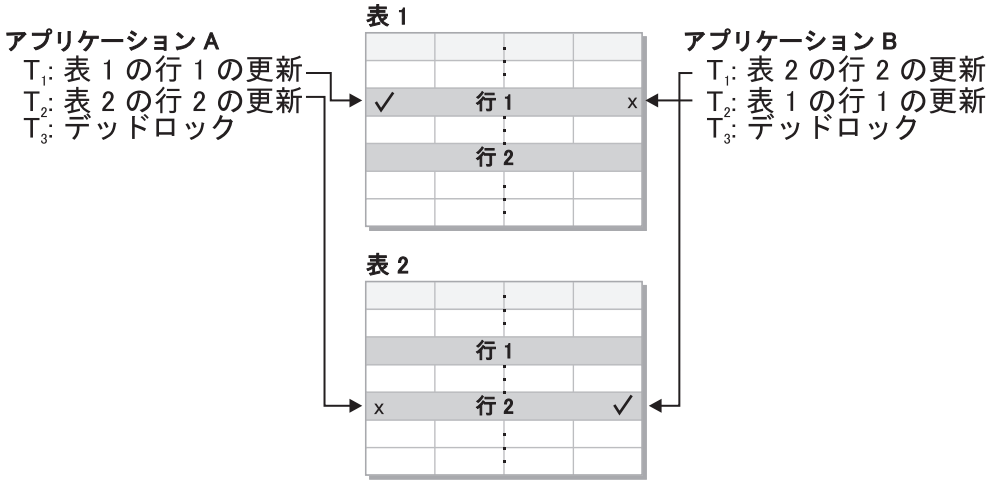


図 3. アプリケーション間のデッドロック

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ないので、デッドロックを解除し、アプリケーション処理を続行するためのデッドロック検出機能が必要です。この名前が示すとおり、デッドロック検出機能は、 *dlchktime* 構成パラメーターで指定された間隔で起動して、ロックに関して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。

デッドロックが見つかる場合、デッドロック検出機能は、1 つのデッドロック・プロセスを、ロールバックする*選択されたプロセス* として任意に選択します。選択されたプロセスはアクティブにされ、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から非コミット・トランザクション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、5 分というウェイクアップ・インターバルを選択すると、デッドロックはほぼ 5 分間存在することができます。これは、短いトランザクション処理の場合には長い時間のように思われます。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バランスを保つことが重要です。

注:

1. パーティション・データベース環境では、 *dlchktime* 構成パラメーター・インターバルは、カタログ・ノードにのみ適用されます。大量のデッドロックがパーティション・データベース環境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。
2. パーティション・データベースでは、それぞれのデータベース・パーティションは、システム・カタログ・ビューを含むデータベース・パーティションにロック・グラフ を送信します。グローバルなデッドロック検出は、このデータベース・パーティション上で行われます。

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。 たとえば、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などです。それらのプロセスが読み取り専用 SQL または XQuery 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てからその表の更新を行うと、プロセス A が X ロックを入手しようとしている行に対して、プロセス B が S ロックを持っているという状況になる場合があります。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、以下のいずれかを実行してください。

- 選択操作を実行するときには FOR UPDATE OF 節を使用する。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。行ブロックは使用できません。
- 照会を実行するときに、WITH RR USE AND KEEP UPDATE LOCKS 節または WITH RS USE AND KEEP UPDATE LOCKS 節を使用する。どちらかの節を使用すると、プロセス A がデータを読み取ろうとするときに U ロックがかけられ、行ブロッキングが可能になります。

データベースが作成されると同時に、詳細デッドロック・イベント・モニターが作成されます。他のモニターと同様に、このイベント・モニターにも関連したオーバーヘッドがあります。

このイベント・モニターが消費するディスク・スペースの量を制限するために、出力ファイルの最大数に達すると、イベント・モニターが非アクティブになり、メッセージが管理通知ログに書き込まれます。必要のない出力ファイルを除去すると、イベント・モニターは次のデータベースの活動化時に再度アクティブになります。

詳細デッドロック・イベント・モニターを必要としない場合は、以下のコマンドを使用してイベント・モニターをドロップできます。

```
DROP EVENT MONITOR db2detaildeadlock
```

アプリケーションがニックネームにアクセスするフェデレーテッド・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。

デッドロックに関する詳細をログに記録するには、データベース・マネージャーの構成パラメーター *diaglevel* を 4 に設定します。ログに記録された情報には、オブジェクト、ロック・モード、および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 および XQuery ステートメントまたは静的パッケージ名もログに記録されている可能性があります。動的 SQL および XQuery ステートメントは、*diaglevel* が 4 である場合にのみログに記録されます。

デフォルトのデッドロック・イベント・モニター

データベース作成時に、デフォルトでは DB2DETAILDEADLOCK というデッドロック・イベント・モニターが作成され、アクティブにされます。これはインスタンスがアクティブになるときに自動的に開始します。このモニターがアクティブの場合、診断情報がデッドロックの最初のオカレンスで収集され、複製を必要とせずに原因を調査することができます。

このイベント・モニターが消費するディスク・スペースの量を制限するために、出力ファイルの最大数に達すると、イベント・モニターが非アクティブになり、メッセージが管理通知ログに書き込まれます。必要のない出力ファイルを除去すると、イベント・モニターは次のデータベースの活動化時に再度アクティブになります。

コマンドは、以下のステートメントを使用して作成されます。

```
db2 create event monitor db2detaildeadlock for deadlocks with details write to file
'db2detaildeadlock' maxfiles 20 maxfilesize 512 buffersize 17 blocked append autostart
```

WITH DETAILS 節は、デッドロックの発生時に実行されたステートメントなどの情報と、ロック・リストを (十分なメモリーがデーモン・ヒープに存在していれば) 提供します。

出力ファイルは、データベース・ディレクトリー内のディレクトリー 'db2event' の下に作成されます。データベースを作成するときにロケーションを指定しない場合、データベース・ディレクトリーのロケーションは、データベース・マネージャー構成パラメーター *dftdbpath* を表示することで判別できます。例えば、AIX® 上のサンプル・データベース上で、イベント・モニター出力ファイルは NODE0000/SQL00001/db2event/db2detaildeadlock ディレクトリーにある場合があります。

イベント・モニターはファイルに、それぞれサイズが 2M (512 4K ページ) の、最大で 20 ファイルを書き込みます。 *maxfilesize* (2M) に達すると、出力ファイルはクローズされて、新規出力ファイルがオープンします。作成されたファイルの数が *maxfiles* (20) に達すると、モニターはシャットダウンされ、以下のようなメッセージが管理通知ログに記録されます。

```
2004-12-01-22.58.24.968000 Instance: DB2 Node: 000 PID: 1116(db2syscs.exe) TID: 2540
Appid: *LOCAL.DB2.041202080328 database monitor sqm__evmgr::log_ev_err Probe:2
Databse:XXX ADM2001W The Event Monitor "DB2DETAILDEADLOCK" was deactivated because the
MAXFILES and MAXFILES CREATE EVENT MONITOR parameters' limits have been reached.
```

必要のない出力ファイルを除去すると、モニターは次のデータベースの活動化時に再び開始されます。詳細デッドロック・イベント・モニターを必要としない場合は、以下のコマンドを使用してイベント・モニターをドロップできます。

```
DROP EVENT MONITOR db2detaildeadlock
```

デッドロックについて考慮している場合は、このモニターをドロップしないでください。

ディスク・ストレージの概要

ディスク・ストレージのパフォーマンス要因

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・ストレージに関連するいくつかの点について考慮します。

パフォーマンスに影響を及ぼすディスク・ストレージ管理の 4 つの要因は以下のとおりです。

- **ストレージの分割方法**

  限られた量のストレージを、索引とデータ間で、および表スペース間で分割する方法によって、さまざまな状況でそれぞれ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になります。

- **ディスク入出力の分散**

  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分散するかどうかは、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。

- **使用可能なストレージの不足**

  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。

第 2 部 表および索引

第 6 章 標準の表における表および索引の管理

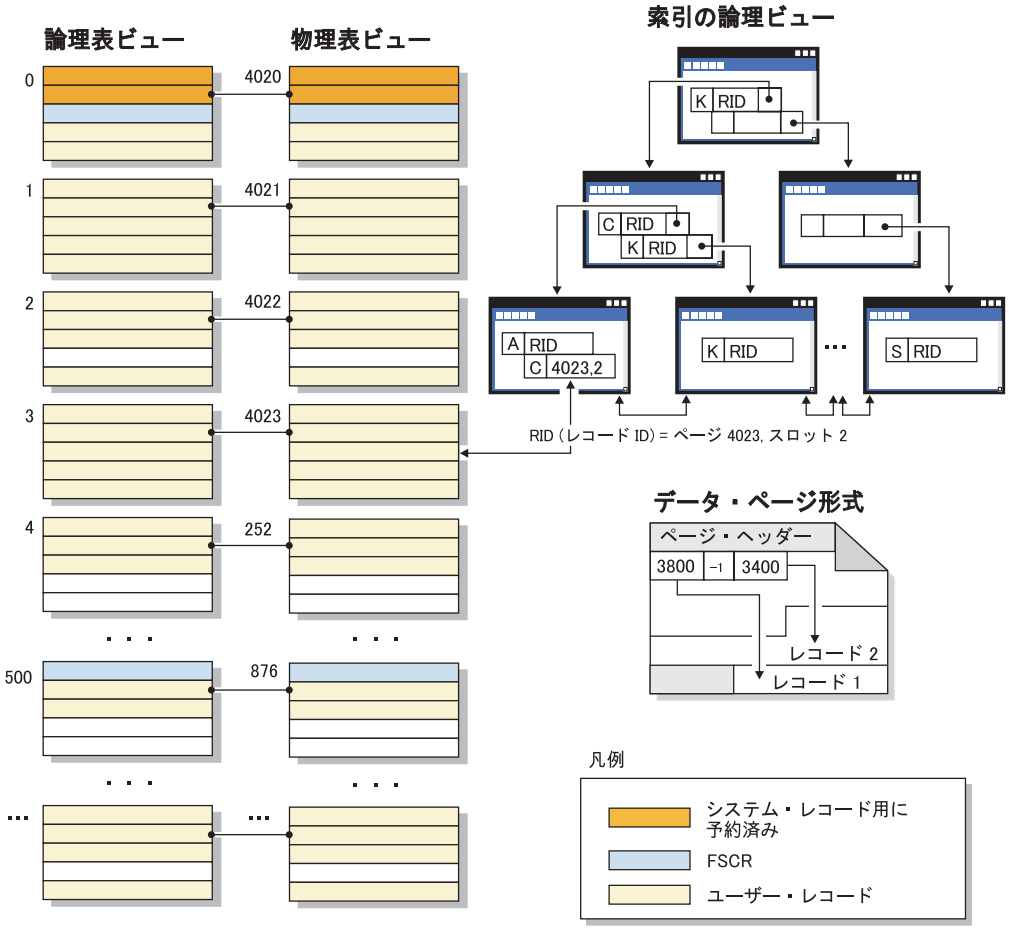


図 4. 標準の表の論理表、レコード、および索引構造

標準の表では、データはデータ・ページのリストとして論理的に編成されます。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 ページに収まるレコードの最大数が表 1 にあります。ほとんどのページにはユーザー・レコードのみ含まれています。ただし、いくつか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、標準の表の場合に、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) それぞれ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に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、**索引スキャン**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。

各データ・ページのフォーマットは同じです。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項目は、ページの異なるレコードに対応します。項目自体は、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) の項目は、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、ページ番号の後にスロット番号が付いたものです。タイプ 2 の索引レコードには、ridFlag という追加のフィールドも含まれます。 ridFlag は、このキーがマークされているか、削除されているかなど、索引中のキーの状況についての情報を保管します。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。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。

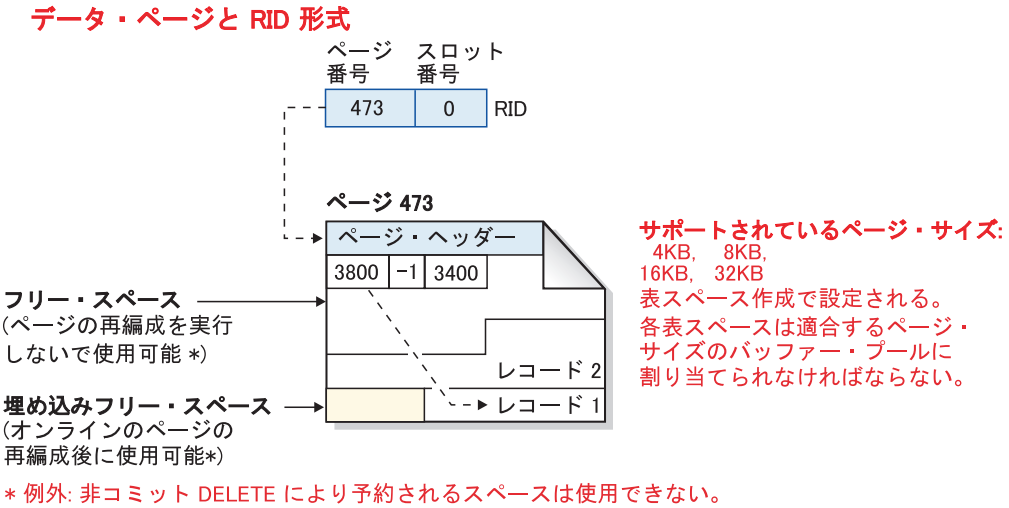


図 5. データ・ページとレコード ID (RID) 形式

表ページが再編成されるとき、レコードが物理的に削除された後で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ワークロードには大きいページ・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ワークロードには、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。

標準の表における索引の管理

DB2 索引は、書き込み先行ロギン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を使用します。

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スキャンできます。通常、索引ページは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ワークロードの場合に役立ちます。

バージョン 8.1 以降、DB2 では タイプ 2 索引が使用されています。以前のバージョンの DB2 から移行している場合、索引の再編成や、タイプ 1 の索引をタイプ 2 に変換する他の処置を実行しない限り、タイプ 1 とタイプ 2 の索引の両方が使用中です。索引タイプは、削除されたキーが索引ページから物理的に除去される方法を決定します。

- タイプ 1 の索引の場合、ページの最後の索引キーが除去されると、キー削除中に索引ページからキーが除去され、索引内のページが解放されます。
- タイプ 2 の索引の場合、索引キーは、表に X ロックがある場合のみ、キー削除中にページから除去されます。キーがすぐに除去できない場合、削除のマークが付けられ、後で物理的に除去されます。詳細については、タイプ 2 の索引について説明しているセクションを参照してください。

索引作成時に、MINPCTUSED 節をゼロより大きい値に設定して、オンライン索引デフラグを使用可能にしてある場合、索引リーフ・ページはオンラインでマージできます。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除去された後で、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、データベース・マネージャーは残りのキーを近隣のページのキーにマージしようとします。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。オンライン索引デフラグを使用することによってスペース再利用は改善されるかもしれませんが、MINPCTUSED 値が大きすぎると、マージの試行に要する時間が長くなり、マージが正常に実行される可能性は低くなります。この節の推奨値は、50 % 以下です。

注: オンライン・デフラグは索引ページからキーを除去する場合のみに発生するので、タイプ 2 の索引では、キーが単に削除のマークを付けられているだけで、ページから物理的に削除されていなければ行われません。

CREATE INDEX ステートメントの INCLUDE 節を使用すると、指定された列をキー列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。組み込み列更新の保守にかかるコストは、キー列の更新の場合よりも低いですが、索引に表示されない列の更新

の保守にかかるコストよりは高くなります。索引 B ツリーの順序付けは、キー列を使用することによってのみ実行できます。組み込み列では実行できません。

第 7 章 MDC 表のための表および索引管理

マルチディメンション・クラスタリング (MDC) 表の表および索引の編成は、標準の表編成と同じ論理構造に基づいています。標準の表と同じく、 MDC 表はデータの行を含むページに編成されて、列に分割され、各ページの行は行 ID (RID) によって識別されます。しかしそれに加えて、MDC 表のページはエクステント・サイズのブロックにグループ化されます。たとえば、エクステント・サイズが 4 の表を示す下の例で、 0 から 3 の番号が付けられた最初の 4 ページは表の最初のブロックとなります。 4 から 7 の番号が付けられた次のページのセットは、表の 2 番目のブロックとなります。

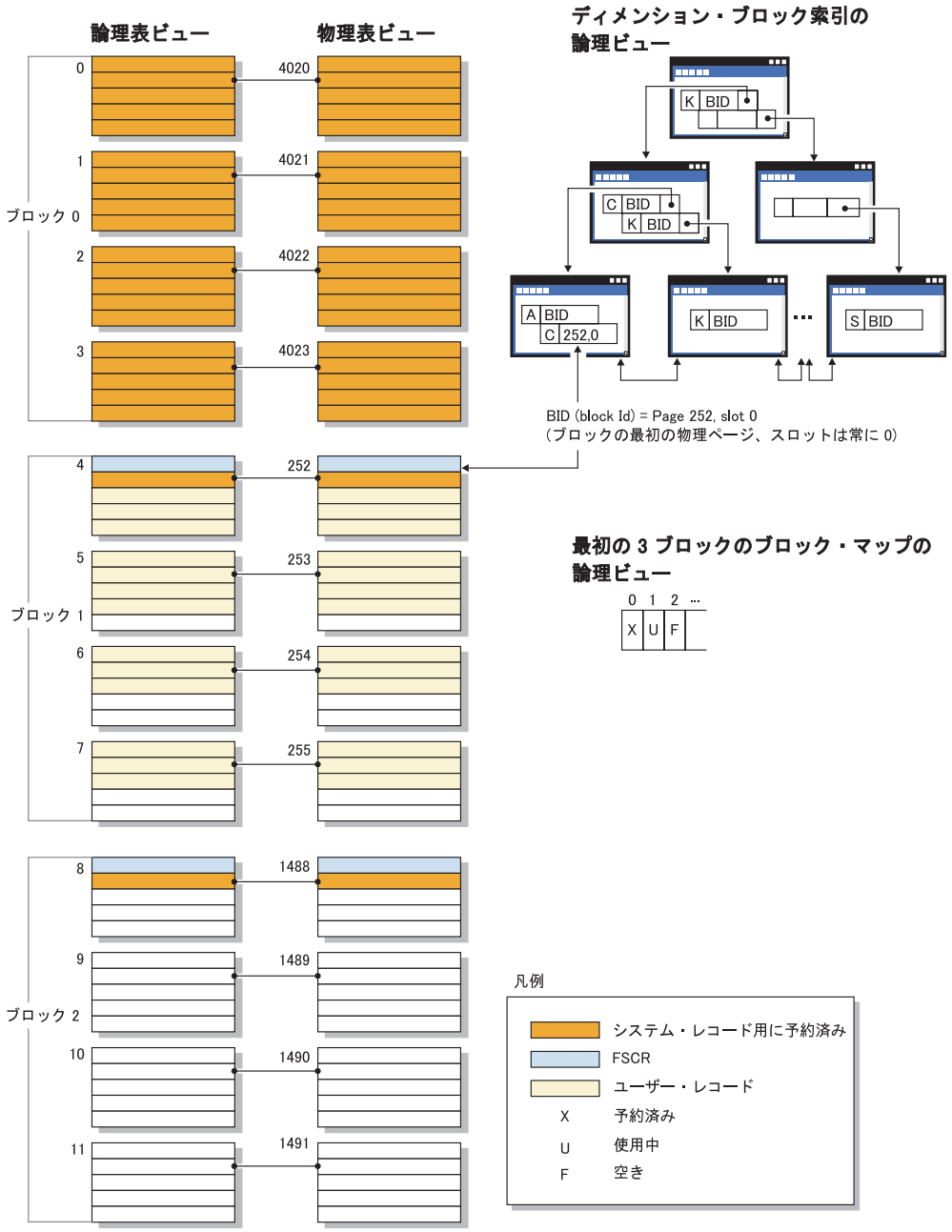


図 6. MDC 表の論理表、レコード、および索引構造

最初のブロックには、DB2 が表を管理するために使用する、フリー・スペース制御レコード (FSCR) を含む特殊な内部レコードが含まれます。続くブロックでは、最初のページに FSCR が含まれます。 FSCR はブロック内の各ページに存在する新規のレコード用にフリー・スペースをマップします。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。

名前が暗黙に示すように、MDC 表は複数のディメンションのデータをクラスター化します。各ディメンションは、CREATE TABLE ステートメントの ORGANIZE BY DIMENSIONS 節で指定した列または列のセットによって決まります。 MDC 表を作成するとき、以下の 2 種類の索引が自動的に作成されます。

- 単一のディメンションの各ブロックに対するポインターを含む、ディメンション・ブロック索引。
- すべてのディメンションのキー列を含む、複合ブロック索引。複合ブロック索引を使用して、挿入および更新の際のクラスタリングを保守することができます。

オプティマイザーは、特定の照会に最適のアクセス・プランを判別する際にディメンション・ブロック索引を利用するアクセス・プランを検討します。照会にディメンション値についての述部があるとき、オプティマイザーはディメンション・ブロック索引を使用して、これらの値を含むエクステントを識別し、そこからフェッチします。エクステントはディスク上の物理的に連続したページなので、これによりパフォーマンスが向上して入出力が最少になります。

さらに、データ・アクセス・プランの分析によって特定の RID 索引が照会のパフォーマンスを改善することが示された場合、その索引を作成することができます。

ディメンション・ブロック索引および複合ブロック索引に加えて、 MDC 表は各ブロックの可用性状況を示すビットマップを含むブロック・マップを保守します。以下の属性は、ビットマップ・リスト内にコード化されています。

- X (予約済み): 最初のブロックには表のシステム情報だけが含まれます。
- U (使用中): このブロックはディメンション・ブロック索引と共に使用され、それに関連付けられています。
- L (ロード済み): このブロックは現行のロード操作によってロードされました。
- C (チェック制約): このブロックはロード中の増分制約チェックを指定するためにロード操作によって設定されました。
- T (表のリフレッシュ): このブロックは AST 保守が必要であることを指定するためにロード操作によって設定されました。
- F (フリー): 他の属性が設定されていない場合、ブロックはフリーと見なされます。

各ブロックはブロック・マップ・ファイル内に項目があるので、表が拡大するとファイルも拡大します。この表は別個のオブジェクトとして保管されます。 SMS 表スペース内で、これは新規のファイル・タイプです DMS 表スペース内で、これはオブジェクト表に新規のオブジェクト記述子を持ちます。

第 8 章 MDC 表の非同期索引クリーンアップ

非同期索引クリーンアップ (AIC) を使用すると、マルチディメンション・クラスタリング (MDC) 表から条件を満たすデータのブロックを削除するのに効果的な方法であるロールアウト削除のパフォーマンスを向上させることができます。AIC は、索引項目を無効にする操作の後に行われる、索引の据え置きクリーンアップです。

標準のロールアウト削除の実行中、索引はその削除によって同期的にクリーンアップされます。レコード ID (RID) 索引が数多く含まれている表の場合、削除にかかる時間のかなりの部分は、削除している表の行を参照している索引キーを除去するために費やされます。削除がコミットされた後でそのような索引をクリーンアップするように指定すると、ロールアウトの速度を速めることが可能です。

MDC 表での AIC の利点を生かすには、*据え置き索引表クリーンアップ・ロールアウト* というメカニズムを明示的に有効にする必要があります。据え置きロールアウトを指定するには 2 つの方法があり、**DB2_MDC_ROLLOUT** レジストリー変数を DEFER に設定する方法と、SET CURRENT MDC ROLLOUT MODE ステートメントを発行する方法です。据え置き索引クリーンアップ・ロールアウト中は、トランザクションがコミットされるまでは RID 索引に対する更新は行われずに、ブロックにはロールアウトというマークが付けられます。行レベルの処理は必要ないため、削除中にブロック ID (BID) 索引はクリーンアップされたままの状態です。

データベースがシャットダウンされるなどしてロールアウト削除がコミットされるか、データベースの再始動後に表に最初にアクセスすると、ロールアウト AIC が起動されます。 AIC が進行中でも、クリーンアップ中の索引へのアクセスを含め、索引に対する照会は行えます。

1 つの MDC 表に対して 1 つの調整クリーナーが存在します。複数のロールアウトが関係する索引クリーンアップはその 1 つのクリーナーに統合されます。クリーナーはそれぞれの RID 索引用のクリーンアップ・エージェントを作成し、クリーンアップ・エージェントは同時に複数のそれらの RID 索引を更新します。またクリーナーは、ユーティリティー・スロットル機能とも統合されています。デフォルトでは、各クリーナーには 50 のユーティリティー影響優先度があります (許容される値は 1 から 100 までで、0 はスロットルなしを示します)。この優先度は、SET UTIL_IMPACT_PRIORITY コマンドまたは db2UtilityControl API を使用して変更することができます。

モニター

MDC 表でロールアウトされたブロックはクリーンアップが完了するまでは再利用できないので、据え置き索引クリーンアップ・ロールアウトの進行をモニターするのは有用です。 LIST UTILITIES モニター・コマンドを使用して、クリーンアップ中の各索引のユーティリティー・モニター項目を表示します。また、SYSPROC.ADMIN_GET_TAB_INFO_V95 表関数を使用すると、据え置き索引クリーンアップ・ロールアウトによって現在クリーンアップしている表のブロック数を照

会できます (BLOCKS_PENDING_CLEANUP)。データベース・レベルで MDC 表ブロックのペンディング・クリーンアップ数を照会する場合には、GET SNAPSHOT コマンドを使用します。

以下の LIST UTILITIES の出力例では、クリーンアップされた各索引のページ数の進行状況が示されています。クリーンアップされている表の RID 索引のうちの 1 つに関する各フェーズが、この出力ではリストされています。

```
db2 LIST UTILITIES SHOW DETAILS output.
ID                                = 2
Type                              = MDC ROLLOUT INDEX CLEANUP
Database Name                     = WSDB
Partition Number                  = 0
Description                       = TABLE.<schema_name>.<table_name>
Start Time                        = 06/12/2006 08:56:33.390158
State                             = Executing
Invocation Type                   = Automatic
Throttling:
   Priority                       = 50
Progress Monitoring:
  Estimated Percentage Complete = 83
   Phase Number                   = 1
      Description                 = <schema_name>.<index_name>
      Total Work                  = 13 pages
      Completed Work              = 13 pages
      Start Time                  = 06/12/2006 08:56:33.391566
   Phase Number                   = 2
      Description                 = <schema_name>.<index_name>
      Total Work                  = 13 pages
      Completed Work              = 13 pages
      Start Time                  = 06/12/2006 08:56:33.391577
   Phase Number                   = 3
      Description                 = <schema_name>.<index_name>
      Total Work                  = 9 pages
      Completed Work              = 3 pages
      Start Time                  = 06/12/2006 08:56:33.391587
```

第 9 章 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 つ以上のレベルがあり、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。

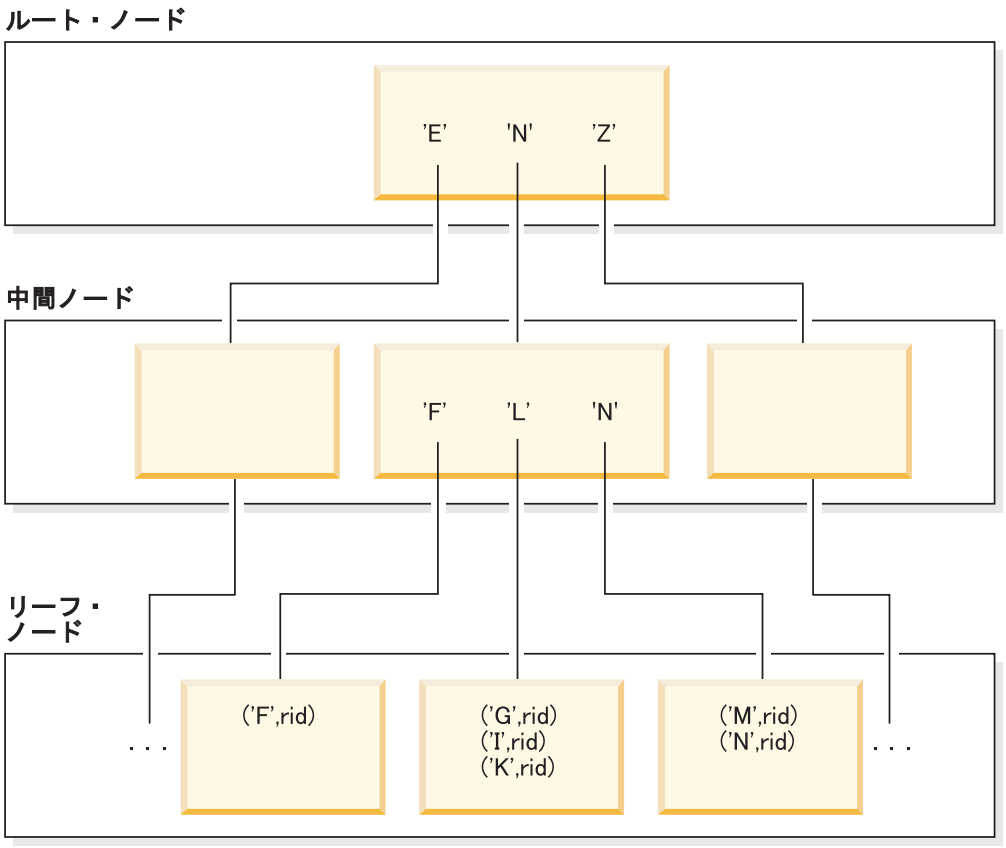


図 7. B+ ツリー構造

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、ここに索引キー値と、そのキー値のある表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。ルート・ノード・キーは特定の中間ノードを指しています。索引マネージャーは必要とする索引キーを持つリーフ・ノードが見つかるまで、中間ノードにこの手順を行います。

この図で検索するキーは「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真ん中のノードを指しています。その中間ノードで「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 の含まれる特定のリーフ・ノードを指します。その行 ID は基本表内の対応する行を識別します。リーフ・ノード・レベルには直前のリーフ・ノードへのポインターが入っている場合もあります。こうしたポインターによって、索引マネージャーはどちらの方向にでもリーフ・ノードをスキャンして、範囲内の 1 つの値を見つけた後、値の範囲を検索することができます。いずれの方向にもスキャンを実行する機能は、 ALLOW REVERSE SCANS 節を使用して索引が作成された場合にのみ使用可能です。

マルチディメンション・クラスタリング (MDC) 表では、表のために指定したクラスタリング・ディメンションごとにブロック索引が自動的に作成されます。もう 1 つの複合ブロック索引も作成されますが、その中には表のディメンションに関係した列ごとのキーの部分が含まれます。これらの索引には RID ではなく、ブロック ID (BID) へのポインターが含まれ、データ・アクセスを改善します。

DB2 バージョン 8.1 以降では、索引はタイプ 1 かタイプ 2 のいずれかです。*タイプ-1 索引* は古い索引のスタイルです。 DB2 の以前のバージョンで作成した索引はこの種類です。

タイプ 2 の索引はタイプ 1 の索引よりいくらか大きく、次のキーのロッキングを最小にする機能を提供します。タイプ 2 の索引のリーフ・ページで RID ごとに保存される 1 バイトの *ridFlag* バイトは、RID を論理的に除去されたとしてマークを付け、後で物理的に除去できるようにします。索引に含まれる可変長列ごとに、1 つの追加のバイトが列の値の実際の長さを保管します。削除されたとマークされていてもまだ索引ページから物理的に除去されていないキーがあるために、タイプ 2 の索引がタイプ 1 の索引より大きくなる場合もあります。 DELETE または UPDATE トランザクションのコミット後に削除済みとマークされたキーをクリーンアップすることができます。

第 3 部 プロセス

第 10 章 照会のパフォーマンスを向上させるためのロギング・オーバーヘッドの低減

すべてのデータベースが、データベースの変更のレコードが保管されるログ・ファイルを保守します。ロギング・ストラテジーは、以下の 2 つから選択できます。

- 循環ロギング。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。
- 保存ログ・レコード。ログ・レコードがログ・ファイルを満たすと、そのログ・ファイルが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。

ロギング・ストラテジーに関係なく、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。ログ・バッファーにあるデータは、ロガー・プロセスによってディスクに書き込まれます。次の状況では、照会処理は、ログ・データがディスクに書き込まれるのを待機する必要があります。

- COMMIT で。
- 対応するデータ・ページがディスクに書き込まれる前。これは DB2 が書き込み先行ロギングを使用するためです。書き込み先行ロギングの利点は、COMMIT ステートメントの実行によるトランザクションの完了時、変更されたデータおよび索引ページを必ずしもすべてディスクに書き込まなくてもよいということです。
- 何らかの変更がメタデータに加えられる前。ほとんどの結果は、DDL ステートメントの実行からのものです。
- ログ・バッファーがいっぱいである場合、ログ・バッファーにログ・レコードを書き込むとき。

DB2 は、処理の遅延を最小限に抑えるために、このような方法でログ・データのディスクへの書き込みを管理します。たくさんの短い並行トランザクションが発生する環境では、ほとんどの処理の遅延は、ログ・データがディスクに書き込まれるのを待機しなければならない COMMIT ステートメントが原因です。結果として、ロガー・プロセスは頻繁に少量のログ・データをディスクに書き込むので、ログ入出力のオーバーヘッドによってさらに遅延が生じます。このようなロギングの遅延に対してアプリケーション応答時間のバランスを取るには、 *mincommit* データベース構成パラメーターを 1 より大きい値に設定します。この設定により、いくつかのアプリケーションからの COMMIT の遅延はさらに長くなることもありますが、 1 つの操作で書き込まれるログ・データの量は多くなるかもしれません。

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。ログ保存が指定され、LOB 列が CREATE TABLE ステートメントで NOT LOGGED 節なしで定義されていないかぎり、

LOB 列の変更はログに記録されません。 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に記録されます。

第 11 章 挿入のパフォーマンスの向上

SQL ステートメントで INSERT を使って新しい情報を表に挿入する場合、INSERT アルゴリズムは最初にフリー・スペース制御レコード (FSCR) を検索して、十分なスペースのあるページを見つけます。ただし、FSCR にフリー・スペースが十分にあると示される場合でも、他のトランザクションの非コミット DELETE によって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。非コミットのフリー・スペースを確実に使用可能にするためには、トランザクションを頻繁に COMMIT する必要があります。

DB2MAXFSCRSEARCH レジストリー変数の設定値は、 INSERT の際に表内で検索される FSCR の数を決定します。このレジストリー変数のデフォルト値は 5 です。指定された数の FSCR の中にスペースが見つからない場合、挿入されるレコードは表の末尾に付加されます。 INSERT の速度を最適化するために、 2 つのエクステントがいっぱいになるまで、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 では、先回終了した位置から FSCR の検索が再開されます。

注: INSERT の速度を最適化するには、 DB2MAXFSCRSEARCH レジストリー変数を小さい値に設定してください (ただし、表がすぐに大きくなる可能性があります)。スペースの再使用を最適化するには、 DB2MAXFSCRSEARCH を大きい値に設定してください (ただし、 INSERT 速度が遅くなる可能性があります)。

表内のすべての FSCR がこのように検索された後、それ以上の検索は行われずに、挿入されるレコードが付加されます。 FSCR の検索は、(たとえば DELETE 後に) 表のどこかにスペースが作成されるまでは、再実行されません。

このほか、以下の 2 つの INSERT アルゴリズム・オプションがあります。

- 付加モード

  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 の検索と保守は行われません。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、単純に大きくなる表 (たとえばジャーナル) のパフォーマンスを改善します。
- クラスタリング索引を表に定義する

  この場合、データベース・マネージャーは、索引キー値が類似している他のレコードと同じページにレコードを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試行されます。それも成功しない場合は、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。この方式により、このキー値を使って行の新しいクラスター域が確立されます。

  表にクラスタリング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 を使用してください。 PCTFREE 節は、ロードや再

編成の後、表のデータ・ページに残しておくべきフリー・スペースのパーセントを指定します。これによって、クラスター索引操作の際に適切なページにフリー・スペースが検出される可能性が高くなります。

第 12 章 更新処理

エージェントがページを更新すると、データベース・マネージャーは次のプロトコルを使って、トランザクションが必要とする入出力を最小限にし、リカバリー可能性を保証します。

1. 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記述されます。このアクションの一部として、ログ・シーケンス番号 (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。
2. ページが変更されます。
3. 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。ページに加えられた変更が、ディスクに書き出されていないためです。
4. ログ・バッファーが更新されます。

   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、そのデータがディスクに強制書き込みされます。

パフォーマンスを向上させるため、これらの入出力は、適当なとき (システム負荷が落ち着いたときなど) 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。特に、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スワップアウトされる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スワップアウトされる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。

    先行ページ・クリーニングを使用可能である場合、この値は関係がなく、ページ・クリーニングを起動しません。
  - *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。

    先行ページ・クリーニングがデータベースに対して使用可能にされており、ページ・クリーナーの数がデータベースに対して適正に構成されていれば、この値を超えることはありません。
  - ヘイト・リスト上のクリーン・ページの数が極端に低くなっている。ページ・クリーナーは、先行ページ・クリーニング方式のこの条件に対してのみ反応します。

- ダーティー・ページが現在 LSNGAP 条件に関与している、または関与することが見込まれている。ページ・クリーナーは、先行ページ・クリーニング方式のこの条件に対してのみ反応します。

- NOT LOGGED INITIALLY 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 ステートメントが発行される場合。 COMMIT ステートメントが実行されると、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。

第 13 章 クライアント/サーバー処理モデル

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アタッチされています。

DB2 がクライアント接続を管理する方法は、接続コンセントレーターがオンかオフのどちらかによって異なります。接続コンセントレーターは、*max_connections* データベース・マネージャー構成パラメーターが *max_coordagents* 構成パラメーターより大きい値に設定されている場合、オンになっています。

- 接続コンセントレーターがオフの場合、それぞれのクライアント・アプリケーションには、コーディネーター・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。
- 接続コンセントレーターがオンになっている場合、各コーディネーター・エージェントは、たくさんのクライアント接続を一度に 1 つずつ管理することができ、他の作業エージェントがこの作業を実行するように調整することもあります。関連する一時的な接続がたくさんあるインターネット・アプリケーション、または比較的小さいトランザクションがたくさんある同様のインターネット・アプリケーションの場合、接続コンセントレーターは、より多くのクライアント・アプリケーションの接続を許可することにより、パフォーマンスを向上させます。また、各接続ごとのシステム・リソースの使用を削減します。

以下の図の DB2 サーバー内にある円は、エンジン・ディスパッチ可能単位 (EDU) を示します。これらはオペレーティング・システムのスレッドを使用してインプリメントされます。

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。

以下の図の A1 では、ローカル・クライアントはまず db2ipccm を介して通信を確立します。 A2 で、db2ipccm 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コーディネーター・エージェントになります。その後、コーディネーター・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、クライアント・アプリケーションとコーディネーターとの間の共用メモリー通信を確立します。 A4 で、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。

以下の図の B1 では、リモート・クライアントは、db2tcpcm EDU を使って通信を確立します。他の通信プロトコルが選択されると、適切な通信マネージャーが使用されます。 db2tcpcm EDU は、クライアント・アプリケーションと db2tcpcm の間に TCP/IP 通信を確立します。次に、B2 で db2agent を処理します。これは、アプリケーションのコーディネーター・エージェントになり、接続をこのエージェント

に渡します。 B4 でコーディネーター・エージェントはリモート・クライアント・アプリケーションと接触し、B5 でデータベースに接続します。

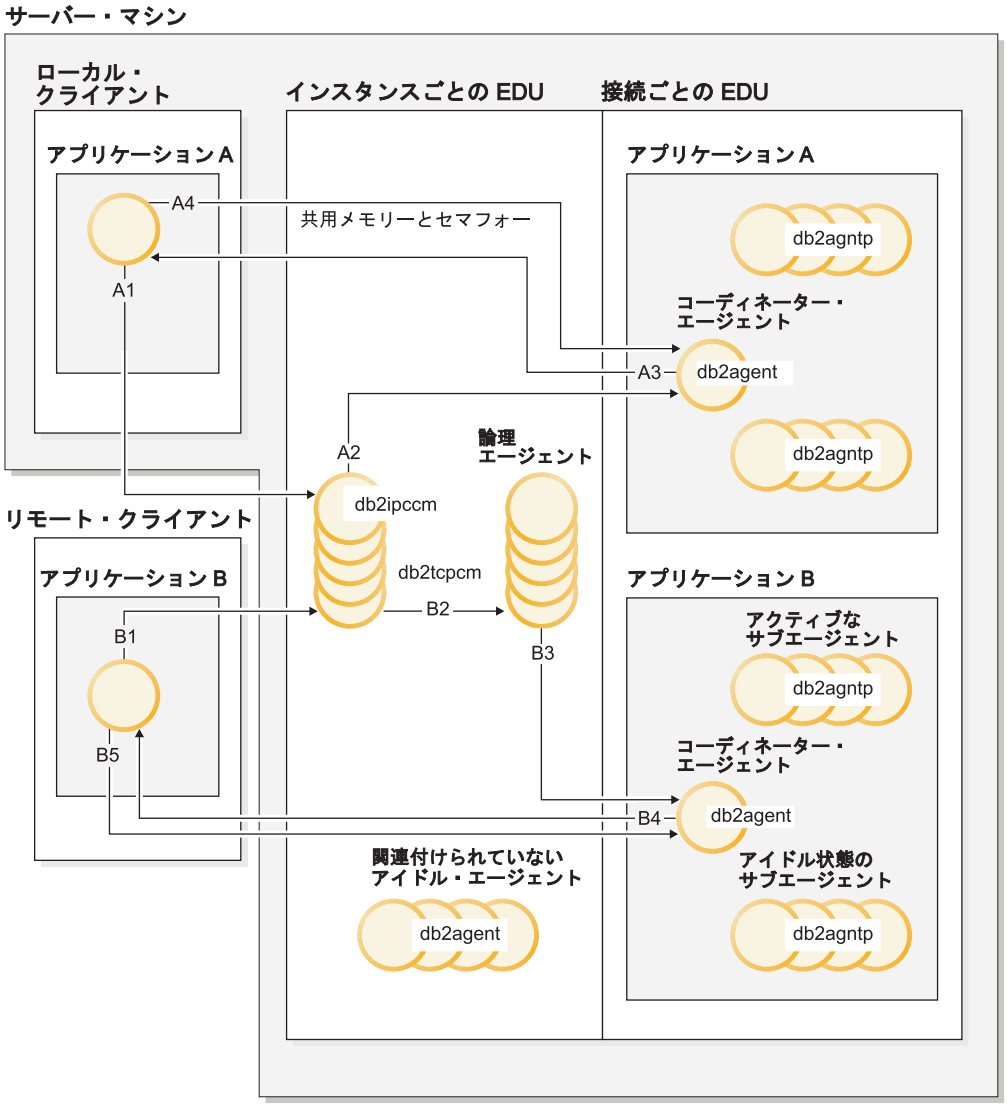


図 8. プロセス・モデルの概要

この図で他に注意することは、以下のとおりです。

- 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。
- 作業エージェントには、4 つのタイプがあります。アクティブなコーディネーター・エージェント、アクティブなサブエージェント、関連付けられたサブエージェント、およびアイドル・エージェントです。
- 各クライアント接続は、アクティブなコーディネーター・エージェントにリンクされます。
- パーティション・データベース環境、およびパーティション内並列処理環境では、コーディネーター・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・エージェントおよびプールされたエージェントが新しい作業が来るのを待機します。
- その他の EDU は、クライアント接続、ログ、2 フェーズ COMMIT、バックアップおよびリストア・タスク、およびその他のタスクを管理します。

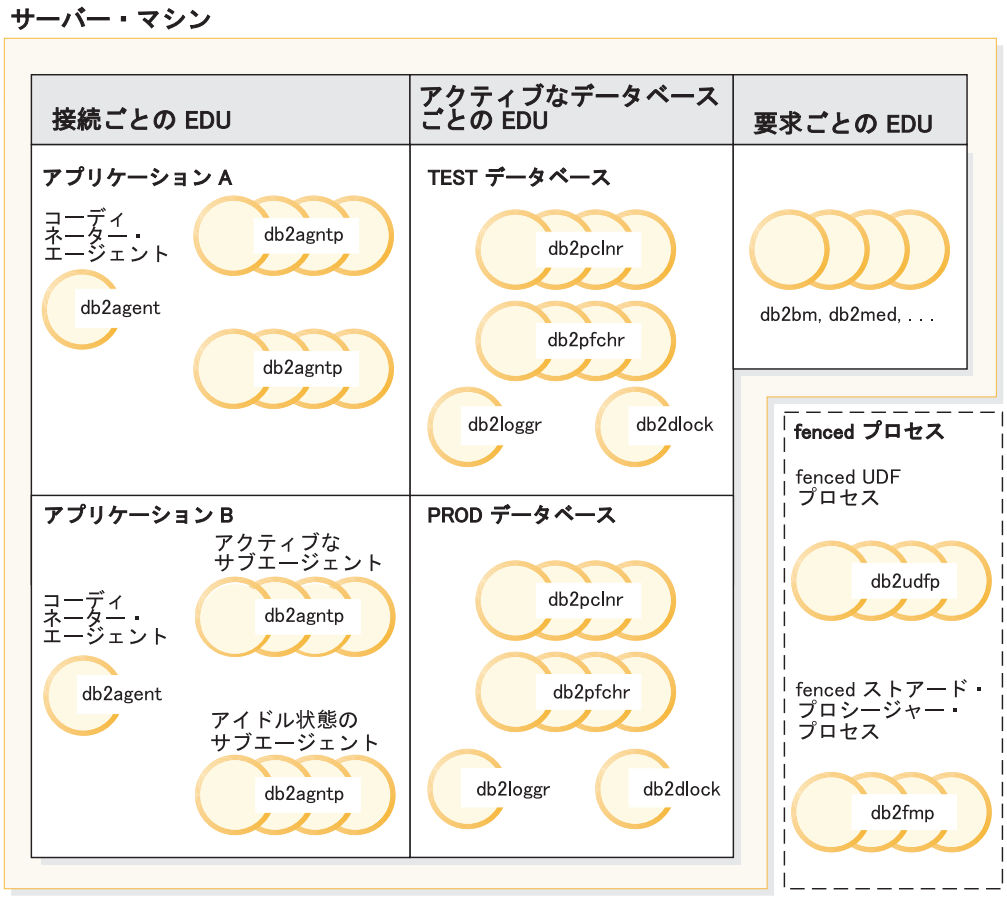


図 9. プロセス・モデル 2

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。アクティブなデータベースには、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。

図には示されていませんが、fenced ユーザー定義関数 (UDF) およびストアード・プロシージャーは、その作成と破棄に関連するコストを最小限に抑えるために管理されます。 *keepfenced* データベース・マネージャー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: unfenced UDF およびストアード・プロシージャーは、パフォーマンスを向上させるため、エージェントのアドレス・スペースで直接実行します。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。

複数データベース・パーティション処理モデルは、単一のデータベース・パーティション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データベース・パーティション処理モデルと、複数データベース・パーティション処理モデルとの類似点や相違点を示します。

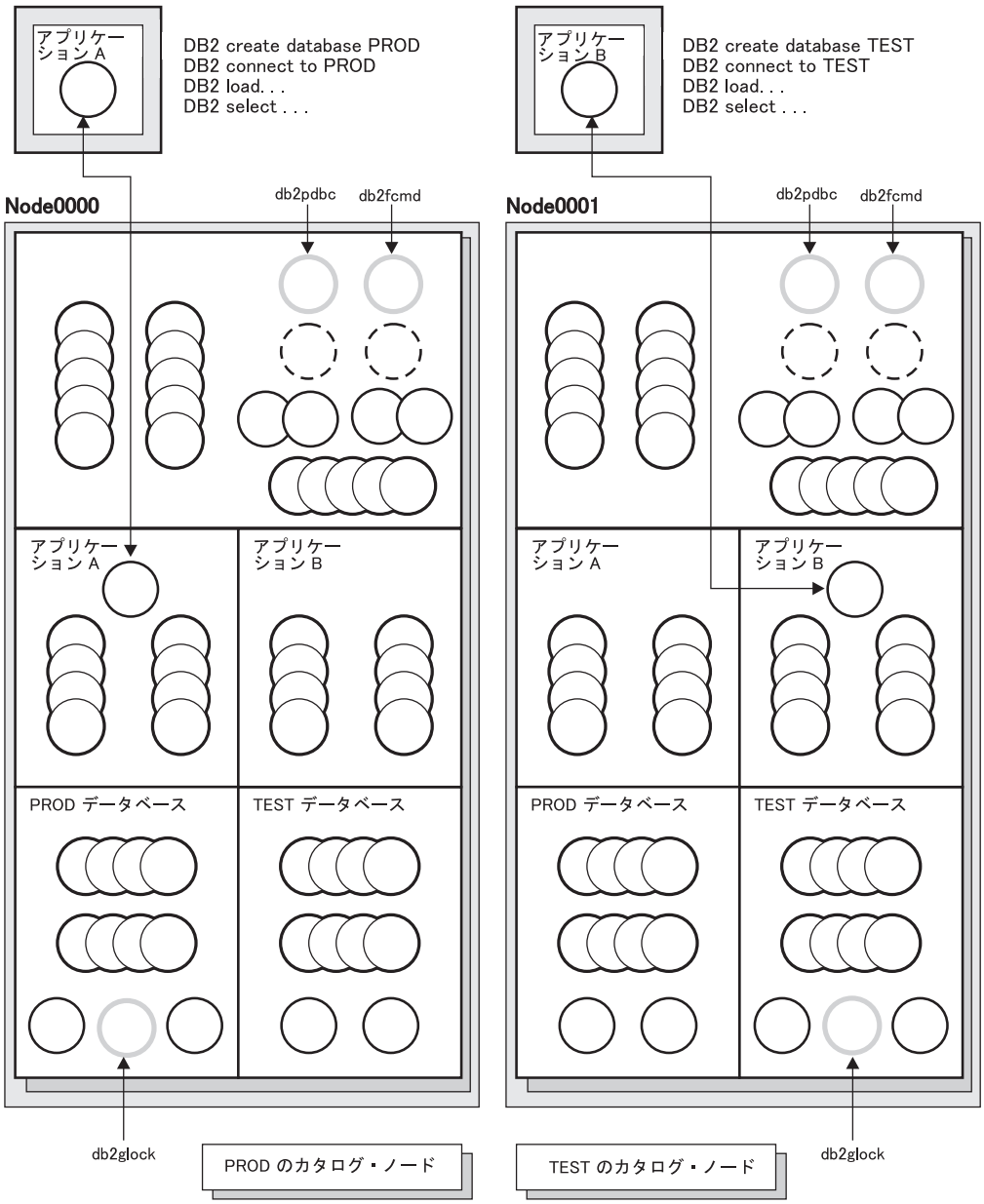


図 10. プロセス・モデルおよび複数のデータベース・パーティション

エンジン・ディスパッチ可能単位 (EDU) の大半は、単一データベース・パーティション処理モデルと、複数データベース・パーティション処理モデルで同じです。

複数データベース・パーティション (またはノード) 環境では、データベース・パーティションの 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、複数パーティション・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・パーティション間の要求を調整し、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、アクティブなデータベースがあるノード間のグローバル・デッドロックを制御します。

アプリケーションからの各 CONNECT は、接続を処理するためにコーディネーター・エージェントと関連する接続によって示されます。コーディネーター・エージェント は、アプリケーションと通信し、要求を受信し応答を送信するエージェントです。コーディネーター・エージェント自体で要求を満たすことも、複数のサブエージェントを調整して要求上で作業するようにすることもできます。コーディネーター・エージェントが存在するデータベース・パーティションは、そのアプリケーションのコーディネーター・ノード と呼ばれます。コーディネーター・ノードは、SET CLIENT CONNECT_NODE コマンドでも設定できます。

アプリケーションからのデータベース要求のパーツは、コーディネーター・プログラム・ノードにより他のデータベース・パーティションのサブエージェントに送られます。次いで、他のデータベース・パーティションからの結果すべてがコーディネーター・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・パーティションは、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・パーティションに保管されます。通常、すべてのユーザー表は一連のノード間で分散されます。

注: 複数のデータベース・パーティションを、同じマシンで稼働するように構成できます。これは、「複数論理パーティション」、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、データベース・パーティション間の通信は、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。

第 4 部 パフォーマンス・チューニングのためのクイック・スタート・ヒント

DB2 の新しいインスタンスを開始するとき、基本構成に関する以下の提案を検討してください。

- コントロール・センターの構成アドバイザーを使用して、システムに適切な開始デフォルトについての提案を入手します。 DB2 の出荷時のデフォルトは、固有のハードウェア環境に合わせて調整する必要があります。

  サイトで使用しているハードウェアについての情報を収集して、ウィザードの質問に応答できるようにします。提案された構成パラメーターの設定値を即時に適用するか、またはウィザードが応答に基づいてスクリプトを作成するようにして、そのスクリプトを後に実行することができます。

  このスクリプトは、後に参照するための最も一般的に調整されるパラメーターのリストも提供します。
- コントロール・センターおよび Client 構成アシスタントの他のウィザードを使用して、パフォーマンスに関連した管理タスクを行います。これらのタスクは通常、少しの時間と努力を費やすことによって多大のパフォーマンス向上を達成できるものです。

  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、「データベース作成」「表の作成」、「索引 (Index)」、および「複数サイト更新の構成」ウィザードがあります。ヘルス・センターは、モニター・ツールおよびチューニング・ツールのセットを提供します。
- コントロール・センターから設計アドバイザー・ツールを使用するか、または `db2advis` コマンドを使用して、どの索引、マテリアライズ照会表、マルチディメンション・クラスタリング (MDC) 表、およびデータベース・パーティションが照会パフォーマンスを向上させるかを判別します。
- ACTIVATE DATABASE コマンドを使用してデータベースを開始します。パーティション・データベースでは、このコマンドはすべてのデータベース・パーティション上のデータベースを活動化して、最初のアプリケーションが接続するときにデータベースを初期化するための起動時間を不要にします。

  注: ACTIVATE DATABASE コマンドを使用する場合、 DEACTIVATE DATABASE コマンドを使用してデータベースをシャットダウンしなければなりません。データベースから最後に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。
- データベース・マネージャーおよび各データベースに使用可能な各構成パラメーターをリストして短く説明しているサマリー表を参照してください。

  これらのサマリー表には、パラメーターを調整するとパフォーマンスが大きく、中程度に、または少なく向上、または低下するか、あるいは変化しないかを示す

列が含まれています。この表を使用して、パフォーマンスの改善を最大にするために調整できるパラメーターを見つけてください。

第 14 章 操作パフォーマンス

DB2 におけるメモリーの割り振り

DB2 において、さまざまな時点でメモリーの割り振りと割り振り解除が発生します。メモリーは、アプリケーションの接続時など、指定のイベントが発生する際に特定のメモリー領域に割り振ることができますし、構成パラメーターの設定の変更に基づいて割り振りを解除することもできます。

次の図は、データベース・マネージャーによって各種用途に割り振られるメモリーの様々な領域、およびユーザーがこのメモリーのサイズを制御することを可能にする構成パラメーターを示しています。複数の論理データベース・パーティションから成る Enterprise Server Edition 環境では、各データベース・パーティションに独自の Database Manager Shared Memory セットがあります。

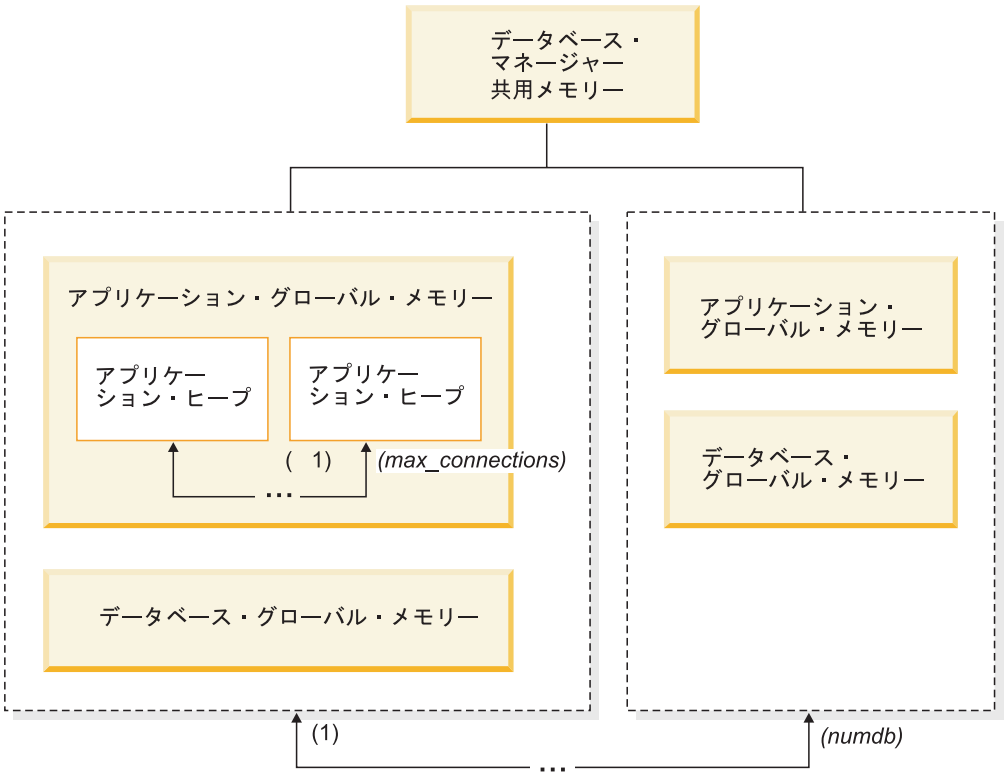


図 11. データベース・マネージャーによって使用されるメモリーのタイプ

メモリーは、次のイベントが発生したときに、データベース・マネージャーの各インスタンスに割り振られます。

- **データベース・マネージャーが開始されたとき (db2start):** データベース・マネージャーのグローバル共用メモリー (インスタンス共用メモリーともいう) が割り振られ、データベース・マネージャーが停止される (db2stop) まで割り振られたままになります。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

ーを管理するためにデータベース・マネージャーが使用する情報が含まれています。 DB2 は、データベース・マネージャーのグローバル共有メモリーのサイズを自動的に制御します。

- **データベースが最初に活動化または接続されるとき:** データベース・グローバル・メモリーが割り振られます。データベース・グローバル・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で使用できます。データベース・グローバル・メモリーのサイズは、 **database_memory** 構成パラメーターで指定されます。デフォルトでは、このパラメーターは automatic に設定されているので、DB2 はデータベースに割り振る初期メモリー量を計算し、データベースの必要に基づいて実行時にデータベースのメモリー・サイズを自動的に構成できます。 **database_memory** を設定して、最初の時点で必要とされるよりもメモリーのサイズを大きく指定しておけば、後から動的に追加のメモリーを分配することが可能になります。

  後続のメモリー領域は動的に調整できます。たとえば、ある領域に割り振られるメモリーを減らして別の領域のメモリーを増やすことができます。

  – バッファー・プール (ALTER BUFFERPOOL DDL ステートメントを使用)
  – データベース・ヒープ (ログ・バッファーを含む)
  – ユーティリティー・ヒープ
  – パッケージ・キャッシュ
  – カタログ・キャッシュ
  – ロック・リスト (このメモリー領域は動的に増やすことのみでき、減らすことはできません。)

  データベース・マネージャーのパーティション内並列処理構成パラメーター (intra_parallel) が有効な環境、接続コンセントレーターが有効な環境、またはデータベース・パーティション化フィーチャー (DPF) が有効な環境では、共有ソート・ヒープもデータベース・グローバル・メモリーの一部として割り振られます。さらに、**sheapthres** データベース・マネージャー構成パラメーターが 0 に設定されている場合 (デフォルト)、すべてのソートはデータベース・グローバル・メモリーを使用します。
- **アプリケーションがデータベースに接続するとき:** アプリケーション・ヒープが割り振られます。それぞれのアプリケーションには、固有のアプリケーション・ヒープがあります。必要に応じて、アプリケーションが **applheapsz** 構成パラメーターを使用して割り振ることができるメモリーの量を制限するか、または **appl_memory** 構成パラメーターを使用してアプリケーション・メモリー使用量全体を制限することができます。

  データベース・マネージャー構成パラメーター **max_connections** は、インスタンスにアタッチできる、またはインスタンス内に存在するデータベースに接続可能なアプリケーションの数の上限を設定します。データベースにアタッチする各アプリケーションは一部のメモリーの割り振りに関係するので、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。
- **エージェントの作成時:** エージェント専用メモリーは、並列環境における接続要求や新しい SQL 要求の結果としてエージェントが割り当てられるときにエージェントに割り振られます。エージェント専用メモリーはエージェントに対して割

り振られ、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 (専用ソート・ヒープなど) が含まれます。

図には、それぞれのタイプのメモリー領域に割り振られるメモリーの量を制限する、以下の構成パラメーターの設定も指定されています。パーティション・データベース環境では、このメモリーが各データベース・パーティションに割り振られることを覚えておいてください。

- **numdb**

  このパラメーターは、別のアプリケーションで使用するために並行してアクティブにできるデータベースの最大数を指定します。各データベースにはそれぞれのグローバル・メモリー領域があるため、このパラメーターの値を大きくすると、割り振られるメモリーの量も大きくなる可能性があります。

- **maxappls**

  このパラメーターは、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指定します。この値は、そのデータベースのエージェント専用メモリーとアプリケーション・グローバル・メモリーに割り振られるメモリーの量に影響します。このパラメーターは、すべてのデータベースで個別に設定できることを覚えておいてください。

  考慮する必要があるその他の 2 つのパラメーターは **max_coordagents** と **max_connections** です。このどちらもインスタンス・レベルで適用されます (DPF インスタンス上のノードごとに) 。

- **max_connections**

  このパラメーターは、DB2 サーバーに同時にアクセスできる接続またはインスタンスのアタッチの数を (DPF インスタンス上のノードごとに) 制限します。

- **max_coordagents**

  このパラメーターは、インスタンス内でアクティブであるすべてのデータベースの間で同時に存在できるデータベース・マネージャー・コーディネーター・エージェントの数を (DPF インスタンス上のノードごとに) 制限します。 **maxappls** と **max_connections** とを合わせて使用することにより、このパラメーターでは、エージェント専用メモリーとアプリケーション・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できます。

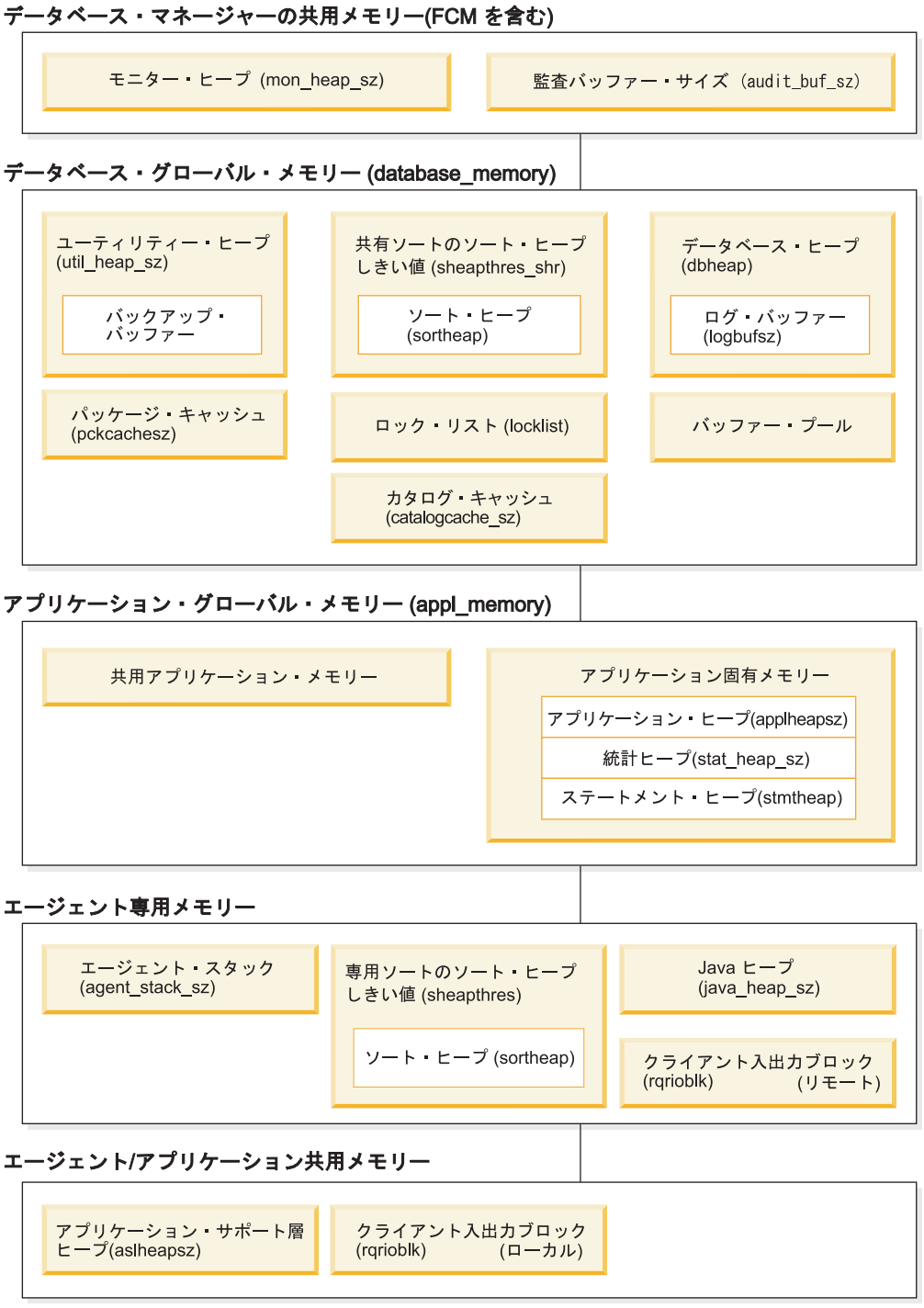
db2mtrk コマンドで起動されるメモリー・トラッカーによって、インスタンス内の現行のメモリーの割り振りを表示することができます。これには各メモリー・プールについての次のタイプの情報が含まれます。

- 現行のサイズ
- 最大サイズ (ハード・リミット)
- ラージ・サイズ (上限基準点)

データベース・マネージャー共用メモリー

データベース・マネージャー・メモリーは、複数の異なるメモリー領域に編成されています。次の図は、データベース・マネージャー・メモリーがどのように割り振られるかを示します。ここに示されている構成パラメーターは、様々なメモリー領

域のサイズを制御することを可能にします。



注: この図のボックスの大きさがメモリーの相対的なサイズを表すわけではありません。

図 12. データベース・マネージャーでのメモリーの使用方法

監査バッファー

このメモリー領域はデータベースの監査アクティビティーで使用されます。このバッファーのサイズは、 **audit_buf_sz** 構成パラメーターによって決まります。

モニター・ヒープ

このメモリー領域はデータベース・システムのモニター・データに使用されます。この領域のサイズは、**mon_heap_sz** 構成パラメーターによって決まります。

高速コミュニケーション・マネージャー (FCM) のバッファー・プール

パーティション・データベース・システムでは、 **fcm_num_buffers** の値が大きい場合は特に、高速コミュニケーション・マネージャー (FCM) に相当量のメモリー・スペースが必要です。 FCM メモリー所要量は FCM バッファー・プールから割り振られます。

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。

- **database_memory** パラメーター。データベース・グローバル・メモリーのサイズの下限を示します。
- 以下のパラメーターまたは因数は、データベース・グローバル・メモリー領域のサイズを制御します。
  - バッファー・プールのサイズ
  - ロック・リスト用最大ストレージ (**locklist**)
  - データベース・ヒープ (**dbheap**)
  - ユーティリティー・ヒープ・サイズ (**util_heap_sz**)
  - パッケージ・キャッシュ・サイズ (**pckcachesz**)
  - 共有ソート・ヒープ (**sheapthres_shr**)
  - カタログ・キャッシュ (**catalogcache_sz**)

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、**appl_memory** 構成パラメーターによって制御できます。以下の構成パラメーターを使用して、いずれか 1 つのアプリケーションが消費できるメモリーの量を制限できます。

- アプリケーション・ヒープ・サイズ (**applheapsz**)
- ステートメント・ヒープ・サイズ (**stmtheap**)
- 統計ヒープ・サイズ (**stat_heap_sz**)

エージェント専用メモリー

- 各エージェントには、それぞれの専用メモリー領域が必要です。 DB2 サーバーは必要な数のエージェントを作成し、特定の構成済みメモリー・リソースを収容します。コーディネーター・エージェントの最大数は、**max_coordagents** パラメーターを使用して制限できます。
- 各エージェントの専用メモリー領域の最大サイズは、次のパラメーターの値によって決められます。
  - 専用ソート・ヒープ・サイズ (**sheapthres** および **sortheap**)
  - 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、次のデータベース構成パラメーターのうちの小さい方の値によって制限されます。
  - すべてのアクティブなデータベースに設定された **maxappls** の合計。
  - **max_coordagents** の値。

  注: エンジン集中機能が有効にされている構成 (**max_connections** > **max_coordagents**) では、アプリケーションのメモリー消費は **max_connections** によって制限されます。
- エージェント/アプリケーション共用メモリーは以下のデータベース構成パラメーターからも影響を受けます。
  - アプリケーション・サポート層ヒープ・サイズ (**aslheapsz**) パラメーター
  - クライアント入出力ブロック・サイズ (**rqrioblk**) パラメーター

FCM バッファー・プールとメモリーの要件

パーティション・データベース・システムにおける、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・プールを以下に示します。

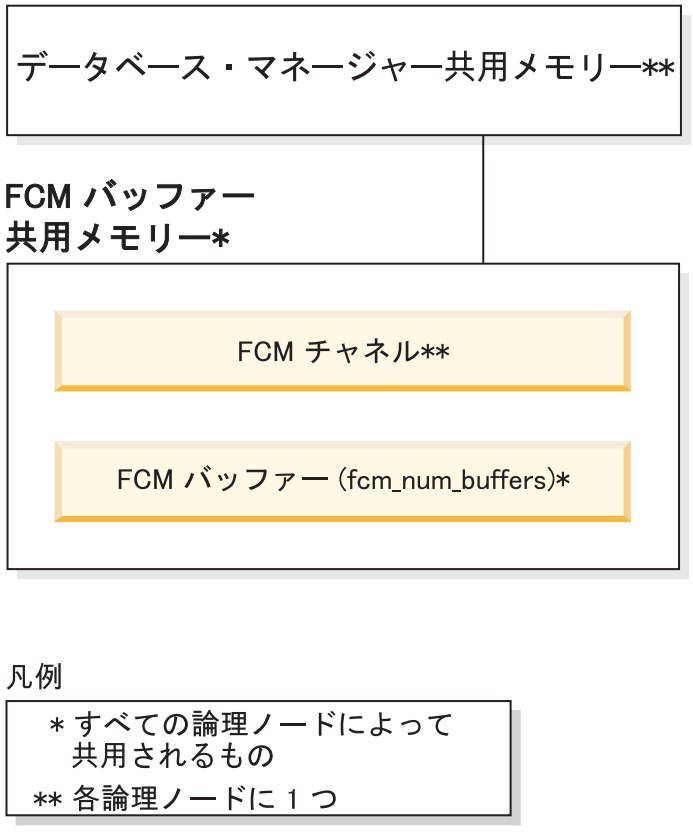


図 13. 複数の論理ノードが使用されている場合の FCM バッファー・プール

各データベース・パーティションの FCM バッファーの数は、*fcm_num_buffers* 構成パラメーターにより制御されます。デフォルトでは、このパラメーターは AUTOMATIC に設定されます。このパラメーターを手動で調整するには、buff_free

(現在空いているバッファー) および buff_free_bottom (空きバッファーの最小数) システム・モニター・エレメントからのデータを使用します。

各データベース・パーティションの FCM チャネルの数は、*fcm_num_channels* 構成パラメーターにより制御されます。デフォルトでは、このパラメーターは AUTOMATIC に設定されます。このパラメーターを手動で調整するには、ch_free (現在空いているチャネル) および ch_free_bottom (空いているチャネルの最小) システム・モニター・エレメントからのデータを使用します。

メモリー割り振りパラメーターのチューニング

メモリーを割り振るパラメーターを設定する際は、絶対に許容最大値を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。これが、メモリー割り振りパラメーターの設定についての第一の規則です。この規則は、メモリーの最大量を持つシステムにも当てはまります。メモリーに影響を与える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・マネージャーがコンピューター上で使用可能な全メモリーを簡単かつ瞬時に使い果たす可能性があるからです。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 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。このようなシステムの場合、共用メモリー・スペースの合計と同じ大きさのページング・スペースを確保するようにしてください。

ほとんどの構成パラメーターでは、メモリーは必要なときのみコミットされ、パラメーター設定によって特定のメモリー・ヒープの最大サイズが決まります。ただし、以下のような場合、パラメーターによって指定されたすべてのメモリーが割り振られます。

- ロック・リスト用最大ストレージ (*locklist*)
- アプリケーション・サポート層ヒープ・サイズ (*aslheapsz*)
- FCM バッファー数 (*fcm_num_buffers*)
- FCM チャネル数 (*fcm_num_channels*)
- バッファー・プール

注:

- ベンチマーク・テストは、メモリー・パラメーターの適切な値を設定するための最適な情報を提供します。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。
- 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、将来的な増加のためのものです。

- 有効なパラメーター範囲については、各パラメーターの詳細情報を参照してください。

セルフチューニング・メモリーの概要

セルフチューニング・メモリーは、メモリー構成パラメーターの値の設定とバッファー・プールのサイズ変更を自動的に実行することにより、メモリー構成のタスクを単純化します。メモリー・チューナーを有効にすると、使用可能メモリー・リソースが幾つかのメモリー・コンシューマー (ソート・メモリー、パッケージ・キャッシュ、ロック・リスト・メモリー、およびバッファー・プールを含む) に動的に分配されます。

以下の表に、セルフチューニング・メモリーに関するトピックをカテゴリー別にリストします。

表 1. セルフチューニング・メモリー情報の概要

カテゴリー	関連トピック
一般情報と制約事項	• 61 ページの『セルフチューニング・メモリー』 • 64 ページの『セルフチューニング・メモリーの操作に関する詳細および制限』 • self_tuning_mem-self_tuning_mem - セルフチューニング・メモリー構成パラメーター
有効/無効にする	• 61 ページの『セルフチューニング・メモリーを有効にする』 • 62 ページの『セルフチューニング・メモリーを無効にする』
モニター	• 63 ページの『セルフチューニングが有効になっているメモリー・コンシューマーの判別』
DPF の考慮事項	• 66 ページの『パーティション・データベース環境でのセルフチューニング・メモリー』 • 68 ページの『パーティション・データベース環境でのセルフチューニング・メモリーの使用』
自動的にチューニング可能な構成パラメーター	• *Configuration Parameter Reference* の『database_memory - データベース共用メモリー・サイズ』 • *Configuration Parameter Reference* の『locklist - ロック・リスト用最大ストレージ』 • *Configuration Parameter Reference* の『maxlocks - エスカレーション前のロック・リストの最大パーセント』 • *Configuration Parameter Reference* の『pckcachesz - パッケージ・キャッシュ・サイズ』 • *Configuration Parameter Reference* の『sheapthres_shr - 共用ソートのソート・ヒープのしきい値』 • *Configuration Parameter Reference* の『sortheap - ソート・ヒープ・サイズ』

セルフチューニング・メモリー

DB2 バージョン 9 からは、新しいメモリー・チューニング・フィーチャーにより、いくつかのメモリー構成パラメーターの値が自動的に設定されることによって、メモリー構成のタスクを単純化します。メモリー・チューナーを有効にすると、使用可能メモリー・リソースがバッファー・プール、パッケージ・キャッシュ、ロック・メモリー、およびソート・メモリーなどのメモリー・コンシューマーの間で動的に分配されます。

チューナーは、**database_memory** 構成パラメーターによって定義されたメモリー制限内で作動します。**database_memory** の値自体も自動的に調整することができます。 **database_memory** のセルフチューニングが有効である (AUTOMATIC に設定している) 場合、チューナーはデータベースの全体的なメモリー要件を判別し、現在のデータベース要件に応じてデータベース共用メモリーに割り振られるメモリーの量を増減します。例えば、現行データベース要件が高く、システムに十分の空きメモリーがある場合、さらに多くのメモリーがデータベース共用メモリーによって消費されます。データベース・メモリー所要量が下げられるか、またはシステムの空きメモリー量が過度に減ると、データベース共用メモリーの一部が解放されます。

database_memory パラメーターがセルフチューニング用に有効にしない場合 (AUTOMATIC に設定しない)、データベース全体はこのパラメーターに指定されたメモリー量を使用し、必要に応じてデータベースのメモリー・コンシューマー間でそれを分配します。データベースにより使用されるメモリーの量は、**database_memory** を数値に設定する、またはそれを COMPUTED に設定するという 2 とおりの方法で指定できます。 2 番目のケースでは、メモリーの合計量は、データベース起動時のデータベース・メモリー・ヒープの初期値の合計に基づいて計算されます。

データベース共有メモリーを **database_memory** 構成パラメーターを使用してチューニングすることに加え、他のメモリー・コンシューマーは以下のようにしてセルフチューニングに対応させることができます。

- バッファー・プールの場合は、ALTER BUFFERPOOL および CREATE BUFFERPOOL ステートメントを使用します。
- パッケージ・キャッシュの場合は、**pckcachesz** 構成パラメーターを使用します。
- ロッキング・メモリーの場合は、**locklist** および **maxlocks** 構成パラメーターを使用します。
- ソート・メモリーの場合は、**sheapthres_shr** および **sortheap** 構成パラメーターを使用します。

セルフチューニング・メモリーを有効にする

セルフチューニング・メモリーは、メモリー構成パラメーターの値の設定とバッファー・プールのサイズ変更を自動的に行うことにより、メモリー構成のタスクを単純化します。これが使用可能になると、メモリー・チューナーは使用可能メモリー・リソースを複数のメモリー・コンシューマー (たとえば、ソート、パッケージ・キャッシュ領域とロック・リスト域、およびバッファー・プール) の間で動的に分配します。

1. *self_tuning_mem* を ON に設定することにより、データベースのセルフチューニングを使用可能にします。 *self_tuning_mem* を ON に設定するには、UPDATE DATABASE CONFIGURATION コマンド、SQLFUPD API、またはコントロール・センターの「**データベース構成パラメーターの変更**」ウィンドウを使用します。
2. メモリー構成パラメーターによって制御されるメモリー領域のセルフチューニングを使用可能にするには、UPDATE DATABASE CONFIGURATION コマンド、SQLFUPD API、またはコントロール・センターの「**データベース構成パラメーターの変更**」ウィンドウを使用して、関連する構成パラメーターを AUTOMATIC に設定します。
3. バッファー・プールのセルフチューニングを使用可能にするには、バッファー・プール・サイズを AUTOMATIC に設定します。これを既存のバッファー・プールで行う場合は ALTER BUFFER POOL ステートメントを使用し、新規バッファー・プールで行う場合は CREATE BUFFER POOL ステートメントを使用します。 DPF 環境でバッファー・プールのサイズが AUTOMATIC に設定される場合は、sysibm.sysbufferpoolnodes には、そのバッファー・プールのいかなる項目も定義されているべきではありません。

注:

1. セルフチューニングは異なるメモリー領域にメモリーを再配分するため、セルフチューニングが行われるためには少なくとも 2 つのメモリー領域 (たとえばロック・メモリー領域とデータベース共用メモリー領域など) を使用可能にしなければなりません。 *sortheap* 構成パラメーターによって制御されるメモリーだけは例外です。 *sortheap* のみが AUTOMATIC に設定される場合、*sortheap* のセルフチューニングは使用可能になります。
2. *locklist* 構成パラメーターのセルフチューニングを使用可能にするには *maxlocks* のセルフチューニングも使用可能にしなければならないため、 *locklist* が AUTOMATIC に設定されると *maxlocks* も AUTOMATIC に設定されます。 *sheapthres* 構成パラメーターのセルフチューニングを使用可能にするには *sortheap* のセルフチューニングも使用可能にしなければならないため、 *sheapthres_shr* が AUTOMATIC に設定されると *sortheap* も AUTOMATIC に設定されます。
3. *sheapthres_shr* または *sortheap* の自動チューニングは、データベース・マネージャー構成パラメーター *sheapthres* が 0 に設定される場合にのみ可能になります。
4. セルフチューニング・メモリーは HADR 1 次サーバーでのみ実行されます。 HADR システムでセルフチューニング・メモリーを活動化すると、構成が正しく設定されていれば、2 次サーバーでは実行されず 1 次サーバーでのみ実行されます。 HADR データベースの役割を切り替えるコマンドが実行されると、セルフチューニング・メモリー操作も新しい 1 次サーバーで実行されるように切り替わります。

セルフチューニング・メモリーを無効にする

self_tuning_mem を OFF に設定することにより、データベース全体のセルフチューニングを使用不可にすることができます。 *self_tuning_mem* を OFF に設定すると、

AUTOMATIC に設定されているメモリー構成パラメーターおよびバッファー・プールは AUTOMATIC のままとなり、メモリー領域はその現行サイズのままとなります。

self_tuning_mem を OFF に設定するには、UPDATE DATABASE CONFIGURATION コマンド、SQLFUPD API、またはコントロール・センターの「**データベース構成パラメーターの変更**」ウィンドウを使用します。

セルフチューニングが使用可能になっているメモリー・コンシューマーが 1 つだけの場合にも、データベース全体のセルフチューニングを実質上非活動状態にすることができます。これは使用可能なメモリー領域が 1 つだけの場合にはメモリーを再配分できないためです。

たとえば、*sortheap* 構成パラメーターのセルフチューニングを使用不可にするには、以下を入力します。

```
UPDATE DATABASE CONFIGURATION USING SORTHEAP MANUAL
```

sortheap 構成パラメーターのセルフチューニングを使用不可にし、それと同時に *sortheap* の現行値を 2000 に変更するには、以下を入力します。

```
UPDATE DATABASE CONFIGURATION USING SORTHEAP 2000
```

メモリー構成パラメーターのセルフチューニングの中には、関連した別のメモリー構成パラメーターも共に使用可能になっていなければ、使用可能にならないものもあります。たとえば、*maxlocks* 構成パラメーターのセルフチューニングは、 *locklist* 構成パラメーターも共に使用可能な場合にのみ許可されます。同様に、*sheapthres_shr* 構成パラメーターのセルフチューニングは、*sortheap* 構成パラメーターのセルフチューニングも共に使用可能である場合にのみ、使用可能になります。ですから、*locklist* または *sortheap* パラメーターのセルフチューニングを使用不可にすると、それぞれ *maxlocks* または *sheapthres_shr* パラメーターのセルフチューニングも使用不可になります。

バッファー・プールのセルフチューニングは、バッファー・プールを特定のサイズに設定することにより使用不可にできます。たとえば、以下のステートメントは `bufferpool1` のセルフチューニングを使用不可にします。

```
ALTER BUFFERPOOL bufferpool1 SIZE 1000
```

セルフチューニングが有効になっているメモリー・コンシューマーの判別

構成パラメーターによって制御されるメモリー・コンシューマーのセルフチューニング設定を表示するには、以下の方法のいずれかを使用します。

- コマンド行から構成パラメーターのセルフチューニング設定を表示するには、SHOW DETAIL パラメーターを指定して GET DATABASE CONFIGURATION コマンドを使用します。

  セルフチューニングを使用可能にできるメモリー・コンシューマーは、出力で次のようにグループ化されます。

```
Description                     Parameter         Current Value       Delayed Value
-------------------------------------------------------------------------------------
Self tuning memory                  (SELF_TUNING_MEM) = ON (Active)        ON
```

```
Size of database shared memory (4KB)  (DATABASE_MEMORY) = AUTOMATIC(37200)   AUTOMATIC(37200)
Max storage for lock list (4KB)              (LOCKLIST) = AUTOMATIC(7456)    AUTOMATIC(7456)
Percent. of lock lists per application       (MAXLOCKS) = AUTOMATIC(98)      AUTOMATIC(98)
 Package cache size (4KB)                  (PCKCACHESZ) = AUTOMATIC(5600)    AUTOMATIC(5600)
Sort heap thres for shared sorts (4KB) (SHEAPTHRES_SHR) = AUTOMATIC(5000)    AUTOMATIC(5000)
Sort list heap (4KB)                         (SORTHEAP) = AUTOMATIC(256)     AUTOMATIC(256)
```

- db2CfgGet API を使用することによってチューニングが使用可能かどうかを判別することもできます。次の値が戻されます。

  ```
  SQLF_OFF                    0
  SQLF_ON_ACTIVE              2
  SQLF_ON_INACTIVE            3
  ```

  SQLF_ON_ACTIVE は、セルフチューニングが使用可能になり、アクティブになっている状況を記述し、SQLF_ON_INACTIVE はセルフチューニングは使用可能になるものの、現行ではアクティブになっていないことを示します。
- 構成の設定は、コントロール・センターの「**データベース構成**」ウィンドウからも表示できます。

バッファー・プールのセルフチューニング設定を表示するには、以下の方法のいずれかを使用します。

- セルフチューニングが使用可能になっているバッファー・プールのリストをコマンド行から検索するには、次のように入力します。

  ```
  db2 "select BPNAME, NPAGES from sysibm.sysbufferpools"
  ```

  バッファー・プールのセルフチューニングが使用可能になると、その特定のバッファー・プールについて、sysibm.sysbufferpools 表の NPAGES フィールドが -2 に設定されます。セルフチューニングが使用不可になると、NPAGES フィールドはバッファー・プールの現行サイズに設定されます。
- セルフチューニングが使用可能になったバッファー・プールの現行サイズを判別するには、次のようにスナップショット・モニターを使用し、バッファー・プールの現行サイズ (bp_cur_buffsz モニター・エレメントの値) を調べます。

  ```
  db2 get snapshot for bufferpools on db_name
  ```
- コントロール・センターを使ってバッファー・プールのセルフチューニング設定を表示するには、バッファー・プールを右クリックし、オブジェクトのオブジェクトの詳細ペインでバッファー・プールの属性を表示します。

メモリー・チューナーの反応は、メモリー・コンシューマーのサイズ変更に必要な時間によって制限されることに注意する必要があります。たとえば、バッファー・プールサイズを縮小するプロセスには長い時間がかかることがあるため、バッファー・プールメモリーをソート領域メモリー用に交換することによって得られるパフォーマンスのメリットをすぐには実現できない場合もあります。

セルフチューニング・メモリーの操作に関する詳細および制限

チューニング要件の判別

メモリー・コンシューマー間で公平かつ適切な比較を行えるようにするため、新しい共通の測定基準が開発されました。チューニングの対象となる各メモリー・コンシューマーが、メモリーの追加で予想される利益を計算して、それをセルフチューニング・メモリー・プロセスに報告します。セルフチューニング・メモリーはこれらの数値をメモリー・チューニングの基礎として使用して、必要性が最も少ないコ

ンシューマーからメモリーを取得し、最も利益を得るメモリー領域にそれを再配置します。

メモリー・チューニングの頻度

セルフチューニング・メモリーは、使用可能になっているとき、データベース・ワークロードの変動性を定期的に検査します。ワークロードが一定ではない場合 (つまり、実行される各照会が同様のメモリー特性を示さない場合)、メモリー・チューナーはメモリーの再配置の頻度を下げ (チューニング・サイクルの間隔が 10 分になるまで)、より安定した傾向予測ができるようにします。メモリー・プロファイルがより一定のワークロードでは、メモリー・チューナーはメモリーをチューニングする頻度を上げ (チューニング・サイクルの間隔が 30 秒になるまで)、より早く収束するするようにします。

セルフチューニング・メモリーの進行のトラッキング

現行のメモリー構成は、GET DATABASE CONFIGURATION コマンドを使用するか、またはスナップショットを使用して取得できます。セルフチューニングによる変更は、stmmlog ディレクトリーにあるメモリー・チューニングのログ・ファイルに記録されます。メモリー・チューニングのログ・ファイルには、各チューニング・インターバルの各メモリー・コンシューマーに関するリソース要求の要約が含まれています。これらのインターバルは、ログ項目内のタイム・スタンプに基づいて判別できます。

最適な構成に収束するまでの予想時間

この機能を使用可能のままにしておくと、メモリーの使用方法を最適化するための各パラメーターが早くチューニングされます。システムは初期構成から早ければ 1 時間でチューニングが完了します。ほとんどの場合、チューニングは最大 10 時間で完了します。この最長のケースが生じるのは、データベースに対して実行される照会が著しく異なるメモリー特性を示す場合です。

セルフチューニング・メモリーの制限

(*database_memory* の値が非常に小さく設定されている、または複数のデータベース、インスタンス、または他のアプリケーションがサーバー上で実行している、などの理由で) 使用可能なメモリー量が小さい場合、セルフチューニング・メモリーによるパフォーマンス上の利点は限定的なものとなります。

セルフチューニング・メモリーはデータベースのワークロードに基づいてチューニングを決定するので、メモリー特性が変動するワークロードでは、セルフチューニング・メモリーが効果的にチューニングする能力が制限されます。ワークロードのメモリー特性が常に変動する場合、セルフチューニング・メモリーはメモリーをチューニングする頻度を下げて、移り変わるターゲット条件に向けて繰り返しチューニングを行います。この場合、セルフチューニング・メモリーは完全な収束には至りませんが、その代わりに現行のワークロードに対してチューニングされたメモリー構成を保持しようとします。

パーティション・データベース環境でのセルフチューニング・メモリー

パーティション・データベース環境でセルフチューニング・メモリー機能を使用する場合、この機能がシステムを適切に調整するかどうかを決定するいくつかの要因があります。

パーティション・データベースでセルフチューニング・メモリーが使用可能にされると、単一のデータベース・パーティションがチューニング・パーティションとして指定され、メモリーのチューニングに関するすべての決定は、そのデータベース・パーティションのメモリーおよびワークロード特性に基づいて行われます。チューニングに関する決定がチューニング・パーティションで行われると、メモリーの調整が他のすべてのデータベース・パーティションに配布され、すべてのデータベース・パーティションが同様の構成を保守することが保証されます。

この単一チューニング・パーティション・モデルでは、メモリー要件が類似しているデータベース・パーティションでのみこの機能を使用する必要があります。以下に示すのは、パーティション・データベースでセルフチューニング・メモリーを使用可能にするかどうかを判別する際に使用するガイドラインです。

パーティション・データベースでセルフチューニングが推奨される場合

すべてのデータベース・パーティションでメモリー要件が類似していて、それらのパーティションが類似のハードウェア上で稼働している場合、変更を加えずにセルフチューニング・メモリーを使用可能にすることができます。これらのタイプの環境では、以下の特性が共通しています。

- すべてのデータベース・パーティションが同一のハードウェア上に存在し、複数の論理ノードが複数の物理ノードに均一に分散されている
- データが完璧かほぼ完璧に分散されている
- データベース・パーティション上で実行されるワークロードがデータベース・パーティション間で均一に分散されている。つまり、1 つ以上のヒープについてメモリー要件が高くなるデータベース・パーティションは存在しません。

そのような環境では、すべてのデータベース・パーティションを同等に構成することが望ましいと言えます。このとき、セルフチューニング・メモリーはシステムを適切に構成します。

パーティション・データベースで注意深いセルフチューニングが推奨される場合

環境内のほとんどのデータベース・パーティションでメモリー要件が類似していて、それらのパーティションが類似のハードウェア上で稼働している場合は、初期構成にいくらかの注意を払うかぎり、セルフチューニング・メモリーを使用することができます。これらのシステムには、データ用のデータベース・パーティションのセットと、ずっと少ない数のコーディネーター・パーティションのセット、およびカタログ・パーティションが存在する場合があります。このような環境では、コーディネーター・パーティションとカタログ・パーティションを、データを含むデータベース・パーティションとは異なった構成にすると便利です。

この環境では、いくらかの小さいセットアップを行うことにより、引き続きセルフチューニング・メモリー機能から益を得ることができます。データを含むデータベース・パーティションによってデータベース・パーティションの大部分が構成されるため、これらのデータベース・パーティションのすべてでセルフチューニングを使用可能にし、これらのデータベース・パーティションの 1 つをチューニング・パーティションとして指定する必要があります。加えて、カタログ・パーティションとコーディネーター・パーティションが異なる構成になっている可能性があるため、セルフチューニング・メモリーをこれらのパーティションで使用不可にする必要があります。カタログおよびコーディネーター・パーティションでセルフチューニングを使用不可にするには、これらのパーティションで *self_tuning_mem* データベース構成パラメーターを OFF に更新してください。

パーティション・データベースでセルフチューニングが推奨されない場合

各データベース・パーティションのメモリー要件が異なっている環境や、さまざまなデータベース・パーティションが大幅に異なるハードウェア上で稼働している場合では、セルフチューニング・メモリー機能を使用不可にすることをお勧めします。これを行うには、すべてのパーティションで *self_tuning_mem* データベース構成パラメーターを OFF にします。

異なるデータベース・パーティションのメモリー要件の比較

複数の異なるデータベース・パーティションのメモリー要件が十分に類似しているかどうかを判別する最良の方法は、スナップショット・モニターを参照することです。以下のスナップショット・エレメントがすべてのパーティションで類似していれば (差が 20% 以下)、それらのパーティションは類似しているとみなせます。

以下のデータを収集するには、コマンド `get snapshot for database on <dbname>` を発行します。

```
Total Shared Sort heap allocated           = 0
Shared Sort heap high water mark           = 0
Post threshold sorts (shared memory)       = 0
Sort overflows                             = 0

Package cache lookups                      = 13
Package cache inserts                      = 1
Package cache overflows                    = 0
Package cache high water mark (Bytes)      = 655360

Number of hash joins                       = 0
Number of hash loops                       = 0
Number of hash join overflows              = 0
Number of small hash join overflows        = 0
Post threshold hash joins (shared memory)  = 0

Locks held currently                       = 0
Lock waits                                 = 0
Time database waited on locks (ms)         = 0
Lock list memory in use (Bytes)            = 4968
Lock escalations                           = 0
Exclusive lock escalations                 = 0
```

以下のデータを収集するには、コマンド `get snapshot for bufferpools on <dbname>` を発行します。

```
Buffer pool data logical reads             = 0
Buffer pool data physical reads            = 0
Buffer pool index logical reads            = 0
Buffer pool index physical reads           = 0
Total buffer pool read time (milliseconds) = 0
Total buffer pool write time (milliseconds)= 0
```

パーティション・データベース環境でのセルフチューニング・メモリーの使用

パーティション・データベース環境でセルフチューニングが有効な場合、データベース・パーティションの 1 つがチューニング・パーティション となります。これは、メモリー構成をモニターして、構成変更があればそれを他のすべてのデータベース・パーティションに伝搬し、すべての関連するデータベース・パーティション間で構成の整合性を保持します。

チューニング・パーティションは、パーティション・グループのデータベース・パーティションの数および定義されたバッファー・プールの数などの、幾つかの特性に基づいて選択されます。

- チューニング・パーティションとして現在指定されているデータベース・パーティションを判別するには、以下の ADMIN_CMD を使用します。

  ```
  CALL SYSPROC.ADMIN_CMD( 'get stmm tuning dbpartitionnum' )
  ```

- チューニング・パーティションを変更するには、以下の ADMIN_CMD を使用します。

  ```
  CALL SYSPROC.ADMIN_CMD( 'update stmm tuning dbpartitionnum <db_partition_num>')
  ```

  このコマンドを発行すると、チューニング・パーティションは、非同期で更新されるか次回のデータベース始動時に更新されます。

- メモリー・チューナーが自動的にチューニング・パーティションを再選択するようにするには、<db_partition_num> 値に -1 を入力します。

DPF システムでのメモリー・チューナーの開始

メモリー・チューナーが DPF 環境において開始されるのは、データベースが明示的な ACTIVATE DATABASE コマンドによってアクティブ化された場合のみです。セルフチューニングでは、複数パーティション・システムでメモリーを正しく調整する前にすべてのパーティションがアクティブであることが必要だからです。

指定されたデータベース・パーティションでセルフチューニングを無効にする

- データベース・パーティションのサブセットでセルフチューニングを無効にするには、調整しないデータベース・パーティションに対して *self_tuning_mem* 構成パラメーターを OFF に設定します。
-

  特定のデータベース・パーティションの構成パラメーターによって制御された、メモリー・コンシューマーのサブセットに対してセルフチューニングを無効にするには、関連する構成パラメーターの値またはバッファー・プール・サイズの値を MANUAL またはそのデータベース・パーティションの特定の値に設定します。ただし、セルフチューニングの構成パラメーターの値は、すべての稼働パーティション間で整合させることをお勧めします。

- データベース・パーティションの特定のバッファー・プールで調整を無効にするには、ALTER BUFFER POOL コマンドを発行し、セルフチューニングを無効にするパーティションについて、サイズ値および PARTITIONNUM パラメーターの値を指定します。

  PARTITIONNUM 節を使用する特定データベース・パーティション上で、サイズを指定するALTER BUFFERPOOL ステートメントは、与えられたバッファー・プールに関する例外項目を SYSCAT.SYSBUFFERPOOLNODES カタログに作成します。あるいは例外項目が既に存在する場合は、これを更新します。このカタログにバッファー・プールの例外項目が存在する場合に、デフォルトのバッファー・プール・サイズが AUTOMATIC に設定されていると、そのバッファー・プールはセルフチューニングに関与しません。例外項目を除去して、バッファー・プールをセルフチューニング用に再び有効にするには、以下のようにします。

  1. ALTER BUFFERPOOL ステートメントを発行して、バッファー・プール・サイズを特定の値に設定することにより、このバッファー・プールのチューニングを使用不可にします。
  2. PARTITIONNUM 節を指定して別の ALTER BUFFERPOOL ステートメントを発行し、このデータベース・パーティション上のバッファー・プールのサイズをデフォルトのバッファー・プール・サイズに設定します。
  3. さらに別の ALTER BUFFERPOOL ステートメントを発行してサイズを AUTOMATIC に設定することにより、セルフチューニングを有効にします。

非均一環境でメモリーのセルフチューニングを有効にする

ご使用のデータをすべてのデータベース・パーティションに均一に分散し、各パーティションで実行されるワークロードが同様のメモリー要求量を持つのが理想的です。データ分散に偏りがあり、1 つ以上のデータベース・パーティションに他のデータベース・パーティションよりもかなり多い (または非常に少ない) データが含まれる場合、これらの変則的なデータベース・パーティションをセルフチューニング用に有効にするべきではありません。メモリー要求量がデータベース・パーティション間で偏っている場合にも同じことが当てはまります。これが生じる可能性があるのは、たとえば、リソースを多く使用するソートが 1 つのパーティションでのみ実行される場合、または一部のデータベース・パーティションが別のハードウェアと関連付けられており、他のデータベース・パーティションよりも使用可能メモリーが多い場合です。このタイプの環境にある一部のデータベース・パーティションでは、セルフチューニングを有効にすることができます。偏りがある環境でメモリーのセルフチューニングを活用するには、同様のデータおよびメモリー所要量のデータベース・パーティションのセットを識別し、セルフチューニング用にそれらを有効にします。残りのパーティションのメモリー構成は手動で構成してください。

バッファー・プール管理

バッファー・プールは、データベース・ページ用の作業メモリーおよびキャッシュを提供します。このバッファー・プールがあると、ディスク上ではなく、メモリー上のデータにアクセスできるため、データベース・システムのパフォーマンスが向上します。メモリーへのアクセスはディスクへのアクセスよりもはるかに速いため、データベース・マネージャーがディスクへの読み書きを行う必要が少なければ少ないほど、データベースのパフォーマンスは良くなります。ほとんどのページ・

データ操作はバッファー・プール内で行われるため、バッファー・プールの構成は、単独の最も重要なチューニングの分野となっています。

アプリケーションが表のある行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・プールで探します。ページがバッファー・プール内にない場合、データベース・マネージャーはディスクからページを読み取り、それをバッファー・プールに入れます。ページがバッファー・プールに入れられると、データベース・マネージャーはそのデータを使用して照会を処理できます。

バッファー・プールへのメモリーの割り振りは、データベースが活動化されたときに行われます。接続する最初のアプリケーションにより、データベースが暗黙的に活動化されることがあります。データベース・マネージャーが稼働しているときにバッファー・プールを作成、ドロップ、およびサイズ変更することも可能です。ALTER BUFFERPOOL ステートメントを使用して、バッファー・プールのサイズを増やすことができます。デフォルトでは、ALTER BUFFERPOOL ステートメントに IMMEDIATE キーワードを指定すると、割り振り可能なメモリーがある限り、コマンドの入力後即時にメモリーが割り振られます。割り振り可能なメモリーがない場合は、ステートメントは DEFERRED として実行し、データベースが再活動化されたときにメモリーが割り振られます。バッファー・プールのサイズを小さくする場合は、トランザクションがコミットしたときにメモリーの割り振りが解除されます。データベースが非活動化されると、バッファー・プール・メモリーは解放されます。最後のアプリケーションが終了すると、データベースが活動化された方法に従って、データベースが暗黙的に非活動化されます。

注: バッファー・プールのサイズを大きくする際に、*dbheap* データベース構成パラメーターのサイズを大きくしなくて済むようにするため、ほぼすべてのバッファー・プール・メモリーはデータベース共用メモリー・セットから出され、自動的にサイズ調整されています。

あらゆる状況において適切なバッファー・プールが使用できるようにするため、DB2 は、それぞれ 4K、8K、16K、および 32K のページ・サイズを持つ小さなシステム・バッファー・プールを作成します。各バッファー・プールのサイズは 16 ページです。これらのバッファー・プールはユーザーからは隠されています。システム・カタログやバッファー・プール・システム・ファイルには示されません。これらのバッファー・プールは、直接使用または変更することはできませんが、以下のような場合に DB2 によって使用されます。

- バッファー・プールが DEFERRED キーワードを使用して作成されたか、作成に使用できるメモリーの不足により必要なページ・サイズのバッファー・プールが活動化されないため、指定されたバッファー・プールが開始されない場合。

  管理通知ログにメッセージが書き込まれます。そして、必要に応じ、システム・バッファー・プールに表スペースが再マップされます。パフォーマンスは大幅に低下する可能性があります。
- データベース接続時に通常のバッファー・プールを得られない場合。

  この問題には、メモリー不足などの深刻な原因が関係していると思われます。システム・バッファー・プールがあるため、DB2 は完全な機能を提供しますが、パ

フォーマンスは大幅に低下するでしょう。この問題は、直ちに対処を必要とします。この問題が発生すると、警告が出され、管理通知ログにメッセージが書き込まれます。

バッファー・プールを作成するときには、別のページ・サイズを明示的に指定していない限り、データベースの作成時に指定されたページ・サイズが指定されます。ページは、表スペースのページ・サイズとバッファー・プールのページ・サイズが一致しないとバッファー・プールに読み込むことができないため、バッファー・プールのページ・サイズを指定する際には、使用している表スペースのページ・サイズを考慮してください。バッファー・プールのページ・サイズは、バッファー・プールを一度作成してしまうと後からは変更できません。ページ・サイズを変えるためには、異なるページ・サイズの新しいバッファー・プールを作成しなければなりません。

db2mtrk で起動したメモリー・トラッカーによって、バッファー・プールに割り振られたデータベース・メモリーの量を表示できます。次の db2mtrk 出力のサンプルは、1 つのユーザー作成のバッファー・プール (括弧内の番号 ″1″ で識別)、および 4 つのシステム・バッファー・プール (括弧内のページ・サイズで識別) を示します。

```
> db2mtrk -d
Tracking Memory on: 2005/10/24 at 12:47:26

Memory for database: XMLDB

   utilh     pckcacheh catcacheh bph (1)   bph (S32K)  bph (S16K)  bph (S8K)
   64.0K     576.0K    64.0K     4.2M      576.0K      320.0K      192.0K

   bph (S4K)  shsorth   lockh     dbh       other
   128.0K     0         640.0K    4.2M      192.0K
```

データ・ページのバッファー・プール管理

バッファー・プールのページの状況は*使用中* かそうではないか、および*ダーティー* か*クリーン* かで区別されます。

- 「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。ページが更新中の場合、更新者以外のだれもそれにアクセスできません。しかし、ページが更新中でない場合、多数の同時リーダーがいてもかまいません。
- 「ダーティー」のページというのは、変更されているがまだディスクに書き出されていないデータが含まれているページのことです。

ページは、データベースがシャットダウンされるまで、また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、あるいはページが (例えばオブジェクトのドロップの一部として) バッファー・プールから明示的にパージされるまで、バッファー・プール内に留まります。別のページをプールに入れるときに除去されるページは、次の基準で決定されます。

- そのページが最後に参照されてからの経過時間
- 再びページが参照される可能性
- ページ上のデータ・タイプ
- メモリー内で変更されたページがディスクに書き出されたかどうか (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ます。)

ディスクに書き込まれた変更されたページは、スペースが必要にならない限りバッファー・プールからは自動的に除去されません。

ページ・クリーナー・エージェント

調整状態の良いシステムでは、通常、変更されたページ、つまり「ダーティー」ページを、ページ・クリーナー・エージェントがディスクに書き込みます。ページ・クリーナー・エージェントは、バックグラウンド・プロセスとして入出力を実行し、実際のトランザクション作業をエージェントが実行できることでアプリケーションがより速く稼働できるようにします。ページ・クリーナー・エージェントは、他のエージェントの作業と連係しておらず、必要な場合にしか機能しないため、*非同期ページ・クリーナー* や*非同期バッファー書き込み機能* と呼ばれることもあります。

更新集中のワークロードでパフォーマンスを向上させるために、先行ページ・クリーニングを使用可能にすることもできます。特定の時点で書き出すダーティー・ページを選択する動作をページ・クリーナーが十分前もって行う点で、パフォーマンスを向上させることができます。これは、非同期のデータ・ページ書き込みや索引ページ書き込みの数に比べてデータ・ページ書き込みや索引ページ書き込みの数が多いことがスナップショットによって明らかである場合に、特に当てはまります。

詳しくは、 77 ページの『先行ページ・クリーニング』を参照してください。

バッファー・プールのデータ・ページ管理の図

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。

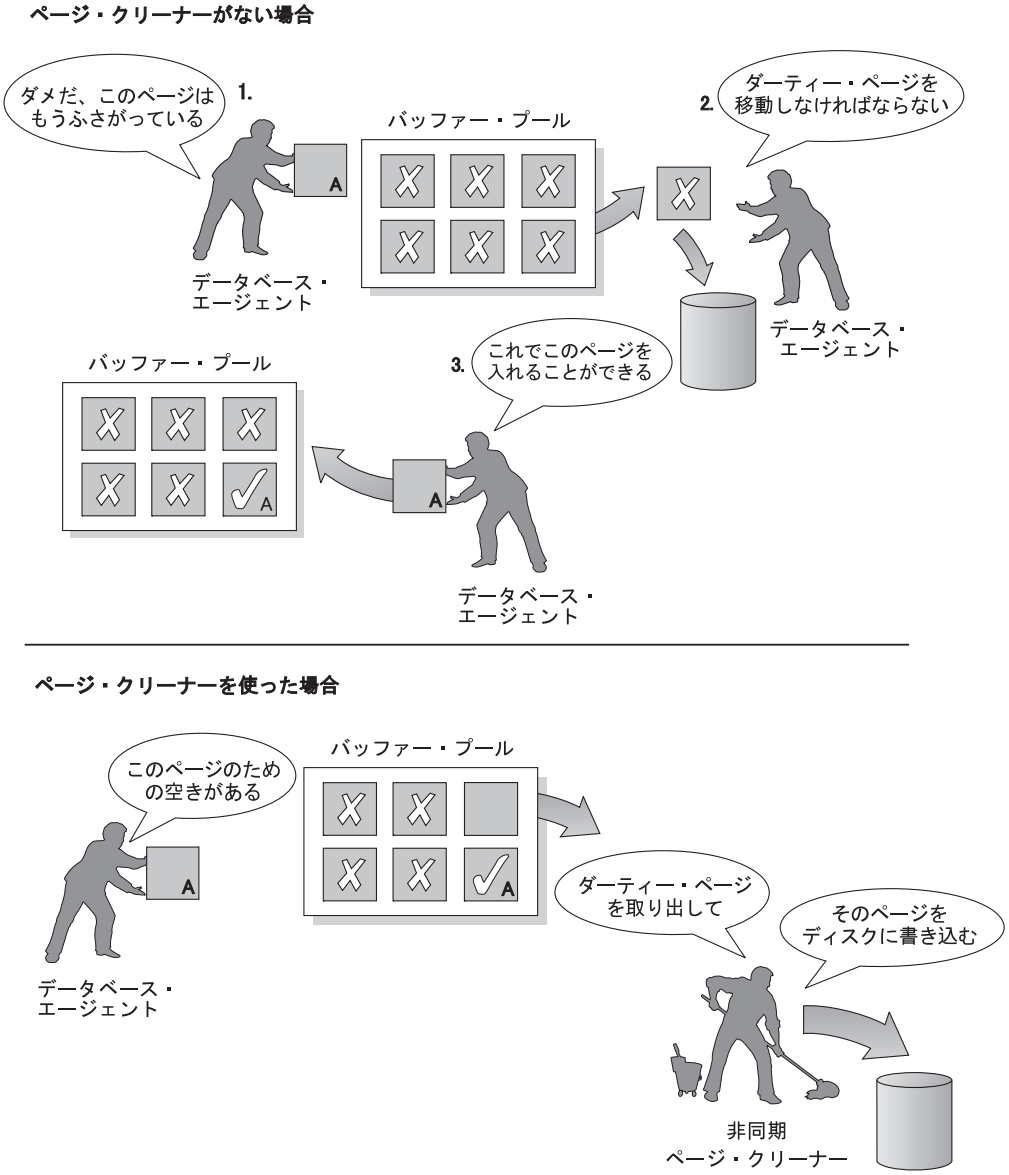


*図 14. 非同期*ページ・クリーナー： 「ダーティー」ページがディスクに書き出されます。

ページ・クリーニングと高速リカバリー

システムがクラッシュしたときのデータベースのリカバリーは、より多くのページがディスクに書き込まれている方が速やかに行えます。より多くのページがディスクに書き込まれていれば、データベース・マネージャーは、データベース・ログ・ファイルからトランザクションを再生しなくても、バッファー・プールのより大きな部分をディスクから再構築できるからです。

リカバリー時に読み取らなければならないログのサイズは、ログの中の以下のレコードの位置によって異なります。

- 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データ変更を記述するログ・レコード

リカバリー時に再生する必要のあるログのサイズが次の値を超えない場合に、ページ・クリーナーはダーティー・ページをディスクに書き込みます。

```
logfilsiz * softmax / 100 (4K ページ単位)
```

詳細は次のとおりです。

- *logfilsiz* はログ・ファイルのサイズを表します。
- *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。 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。

リカバリー時のログの読み取り時間をできるだけ短くするため、データベース・システム・モニターを使用して、ページ・クリーニングが実行された回数をトラッキングしてください。データベースに対して先行ページ・クリーニングを使用可能にしていない場合には、システム・モニター *pool_lsn_gap_clns (起動されるバッファー・プール・ログ・スペース・クリーナー)* モニター・エレメントからこの情報が提供されます。この代替ページ・クリーニングが有効になっている場合は、この条件が起きることはなく、 *pool_lsn_gap_clns* モニター・エレメントは常に 0 です。

log_held_by_dirty_pages モニター・エレメントは、ユーザーによって設定されたリカバリー基準に見合う十分な量のページをページ・クリーナーがクリーニングしていないかどうかを判別するために使用できます。 *log_held_by_dirty_pages* が常に *logfilsiz * softmax* よりかなり大きい場合には、さらにページ・クリーナーが必要であるか、または *softmax* を調整することが必要です。

複数のデータベース・バッファー・プールの管理

各データベースは最低 1 つのバッファー・プールを必要としますが、複数のページ・サイズの表スペースを持つ 1 つのデータベースに対して、サイズやページ・サイズの異なる複数のバッファー・プールを作成することもできます。それぞれのバッファー・プールの最小サイズは、プラットフォームに応じて異なります。ALTER BUFFERPOOL コマンドを使ってバッファー・プールをサイズ変更することができます。

新規データベースは、IBMDEFAULTBP というデフォルト・バッファー・プールを持っています。その全体のサイズはプラットフォームによって決まり、そのデフォルト・ページ・サイズはデータベースの作成時に指定されたページ・サイズによって決まります。デフォルトのページ・サイズは、情報データベースの構成パラメーター *pagesize* として保管されます。デフォルトのページ・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。デフォルトのバッファー・プールのサイズ変更と、その属性の変更は可能ですが、それをドロップすることはできません。

バッファー・プールのページ・サイズ

データベースを作成または移行した後で、追加のバッファー・プールを作成できます。デフォルトとして 8 KB ページ・サイズのデータベースを作成でき、デフォルト・バッファー・プールがデフォルト・ページ・サイズ (この場合、8 KB ページ・サイズ) で作成されます。その代わりに、バッファー・プールを 8 KB のページ・

サイズで作成し、同時に同じページ・サイズの 1 つ以上の表スペースを作成する必要があります。この方法では、データベースの作成時に 4 KB デフォルト・ページ・サイズを変更する必要はありません。異なるページ・サイズを使用するバッファー・プールに表スペースを割り当てることはできません。

注: 4 KB より大きいページ・サイズ (8 KB、16 KB、32 KB など) 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・サイズを使用する別のアクティブ・バッファー・プール (もしあれば)、または最初のクライアントがデータベースに接続した際に DB2 が作成するデフォルトのシステム・バッファー・プールに一時的に表スペースを割り当てようとします。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。

CREATE BUFFERPOOL ステートメントでバッファー・プールを作成するときに、特定のバッファー・プール・サイズを指定できます。サイズを指定しない場合は、DB2 によって AUTOMATIC に設定され、管理されます。後でバッファー・プール・サイズを変更するには、ALTER BUFFERPOOL ステートメントを使用します。

パーティション・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・パーティション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・パーティションのバッファー・プールのサイズが ALTER BUFFERPOOL ステートメントによって変更された場合は除きます。

大容量バッファー・プールの利点

大容量バッファー・プールには次の利点があります。

- 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、TEMPORARY 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ソートの速度も速くなります。

バッファー・プールが多数ある場合の利点

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) より少ない。
- 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す。

それ以外のすべての状況では、複数のバッファー・プールの使用を検討してください。これは、次の理由によります。

- TEMPORARY 表スペースを別のバッファー・プールに割り当てることができるので、特にソートが多い照会など、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の場合には、そのデータを含む表スペースを別のバッファー・プールに割り当て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・プールとは別々にするという場合などに役立ちます。このようにすると、表の最初のセット上の頻繁な更新が表の 2 番目のセット上の頻繁な照会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、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アプリケーションの場合は有効です。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・プールにソートして入れると、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。

開始時のバッファー・プール・メモリーの割り振り

CREATE BUFFERPOOL コマンドを使ってバッファー・プールを作成するか、ALTER BUFFERPOOL ステートメントを使ってバッファー・プールを変更する場合は、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・マネージャーがオンラインになっている間にバッファー・プールを作成または変更する場合、データベース・グローバル・メモリーの追加メモリーが使用できるようになっている必要があります。新規バッファー・プールを作成する際、または既存のバッファー・プールのサイズを増やす際に DEFERRED キーワードを指定し、その際必要とするメモリーが使用できない場合、データベース・マネージャーは、次にデータベースが活動化される際に変更を加えます。

このメモリーがデータベースの開始時に使用できないと、データベース・マネージャーは最小サイズの 16 ページのシステム・バッファー・プール (各ページ・サイズに 1 つ) のみで開始し、SQL1478W (SQLSTATE01626) 警告が戻されます。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、この操作状態で続行されます。パフォーマンスが最善ではないかもしれません。最小サイズの値でのみデータベース・マネージャーを開始するのは、データベース接続を可能にし、バッファー・プール・サイズを再構成して他の重要な

タスクを実行できるようにするためです。これらのタスクを実行してすぐ、データベースを再始動します。そうした状態で、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。

システム・バッファー・プールのみでデータベースを開始することを避けるには、DB2_OVERRIDE_BPF レジストリー変数を使用して、必要なメモリーが使用可能なメモリーに適合するように調整することができます。

先行ページ・クリーニング

バージョン 8.1.4 から、システムにページ・クリーニングを構成する代替方法が提供されました。この代替方法は、特定の時点で書き出すダーティー・ページを選択する動作をページ・クリーナーが十分前もって行う点で、デフォルトの動作とは異なります。このページ・クリーニングの新しい方法がデフォルトのページ・クリーニングと異なっているのは、主に次の 2 つの点です。

1. ページ・クリーナーは chngpgs_thresh 構成パラメーターを重視しない。

   この代替方法では、ページ・クリーナーが chngpgs_thresh 構成パラメーターの値に応じて反応することはなくなっています。代替方法によるページ・クリーニングでは、バッファー・プールを一定の比率でクリーンに保つのではなく、スワップアウトされたページの位置を書き出した直後にエージェントに通知するというメカニズムが採用されています。そのため、スワップアウトするページをエージェントがバッファー・プールから検索する必要はありません。スワップアウトされるページの数が許容値よりも低いと、ページ・クリーナーが起動してバッファー・プール全体を検索し、スワップアウト可能なページを書き出し、それらのページの位置をエージェントに通知します。

2. ページ・クリーナーはロガーの発行する LSN ギャップ・トリガーに反応しない。

   バッファー・プール内の最も古いページを更新したログ・レコードから現行のログ位置までのログ・スペースの量が softmax パラメーターによって許可されている量を超えている状態を、データベースが「LSN ギャップ」状態であるといいます。デフォルトのページ・クリーニング方法では、ロガーが LSN ギャップ状態を検出すると、ページ・クリーナーを起動して、 LSN ギャップ状態を作り出しているページをすべて書き出します。つまり、softmax パラメーターが許可するより古いページを書き出します。 LSN ギャップが起きていない間は、ページ・クリーナーは活動を休止しています。そして、LSN ギャップが起きると、ページ・クリーナーは活動化して大量のページを書き出し、それが終わるとまた活動を休止します。こうした処理のために I/O サブシステムが飽和してしまい、その影響がページの読み書きをしている他のエージェントに及ぶ可能性があります。さらに、LSN ギャップになる以前に、ページ・クリーナーのクリーニング速度が遅いために、DB2 のログ・スペースが不足してしまうこともあり得ます。

   ページ・クリーニングのこの代替方法では、長時間にわたって一定数の書き込みを継続することによって、動作を調節しています。そのために、現在 LSN ギャップ状態にあるページだけでなく、現在の活動レベルから判断してまもなく LSN ギャップ状態になりそうなページをも、十分前もってクリーニングします。

ページ・クリーニングのこの新しい方法を使用するには、DB2_USE_ALTERNATE_PAGE_CLEANING レジストリー変数を「ON」に設定します。

バッファー・プールへのデータのプリフェッチ

ページのプリフェッチ とは、いくつかのページがアプリケーションで必要になることを想定してディスクからリトリーブしておくということです。索引ページおよびデータ・ページをバッファー・プールにプリフェッチすると、入出力の待ち時間が減るので、パフォーマンスは向上します。加えて、並列入出力によってもプリフェッチの効率は拡張されます。

プリフェッチには、2 つのカテゴリーがあります。

- **順次プリフェッチ:** これは、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。
- **リスト・プリフェッチ:** これは、*リスト順次プリフェッチ* とも呼ばれます。連続していないデータ・ページを効率的にプリフェッチします。

これらの 2 つのデータ・ページ読み取り方法は、データベース・マネージャー・エージェントの読み取りを補うものです。データベース・マネージャー・エージェントの読み取りは、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のみ使用されますが、転送されるのは 1 ページのデータのみです。

プリフェッチおよびパーティション内並列処理

プリフェッチは、パーティション内並列処理 (索引または表のスキャン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重要です。これらの並列スキャンによりデータの使用速度が高くなり、高速のプリフェッチが必要になります。

プリフェッチが不十分だと、順次スキャンよりも並列スキャンのほうがコストが高くつきます。順次スキャンでプリフェッチ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。並列スキャンでプリフェッチが行われない場合には、 1 つのサブエージェントが入出力を待機するので、他のすべてのサブエージェントも待機しなければならなくなります。

パーティション内並列処理の場合にはプリフェッチの重要性が高いため、並列処理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。これらのギャップの幅は、スキャン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。

順次プリフェッチ

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、アプリケーションのオーバーヘッドは大幅に短縮されます。加えて、複数の並列入出力操作で複数のページ範囲をバッファー・プールに読み込むことによっても、入出力の待ち時間を短縮することができます。

プリフェッチが開始されるのは、データベース・マネージャーが、順次入出力が適切であり、プリフェッチ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。表スキャンや表ソートなどの場合、データベース・マネージャーは、順次プリフェッチ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プリフェッチを開始します。たとえば、以下の例ではおそらく表スキャンが必要になるので、順次プリフェッチが実行されます。

```
SELECT NAME FROM EMPLOYEE
```

表スペースの PREFETCHSIZE の含意

それぞれの表スペースごとにプリフェッチされるページ数を定義するには、CREATE TABLESPACE または ALTER TABLESPACE ステートメントのいずれかのステートメントの PREFETCHSIZE 節を使用します。指定した値は、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、表スペース・コンテナーの数、各コンテナーの物理ディスクの数 (RAID 装置を使用している場合)、および表スペースの EXTENTSIZE の値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。これは、データベース・マネージャーが別のコンテナーを使用する前にコンテナーに書き込むページ数です。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 つ持っている場合には、プリフェッチ量を 32 ページに設定することができます。各コンテナーに 5 つの物理ディスクがある場合は、プリフェッチ量を 160 ページに設定できます。

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、プリフェッチしたために、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、プリフェッチするページ数を、ユーザーが表スペースに対して指定するページ数未満に制限することができます。

プリフェッチ・サイズによっては、特に大きな表のスキャン時に、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・ツールを使用します。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・エレメントを調べることによって、プリフェッチが行われているかを知ることができます。

照会時にデータのプリフェッチ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 の値を増やすことができます。プリフェッチャー以外の原因により入出力待機が生じている場合、 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。

どのタイプのプリフェッチでも、プリフェッチ・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。より高いパフォーマンスを得るには、コンテナーが別個の物理装置を使用するように構成します。

順次検出

順次プリフェッチ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プリフェッチを活動化することができます。このように行うプリフェッチでは、データベース・マネージャーがプリフェッチ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。このタイプの順次プリフェッチのことを*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。 *seqdetect* 構成パラメーターを使用して、データベース・マネージャーが順次検出を実行する必要があるかどうか制御できます。

たとえば、順次検出がオンになっている場合には、以下のような SQL ステートメントは順次プリフェッチを用いると効果があります。

```
SELECT NAME FROM EMPLOYEE
WHERE EMPNO BETWEEN 100 AND 3000
```

この例の場合、オプティマイザーは、EMPNO 列の索引を使って表スキャンを開始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、プリフェッチによってパフォーマンスが向上する可能性があります。これによって、データ・ページ・プリフェッチが行われます。

この例では、索引ページのプリフェッチが行われます。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、索引ページのプリフェッチが行われます。

改善された順次プリフェッチ用のブロック・ベースのバッファー・プール

ディスクからページをプリフェッチすると、入出力オーバーヘッドのためにコストがかかります。入出力処理が並行して行われれば、スループットが非常に改善されます。多くのプラットフォームでは、ディスク上の連続する複数ページを読み取り、メモリーの不連続部分に格納するハイパフォーマンスのプリミティブが提供されています。このようなプリミティブを、通常、*散在データ読み取り* またはベクトル化入出力 といいます。一部のプラットフォームでは、これらのプリミティブのパフォーマンスが、大きなブロック・サイズによる入出力処理よりも劣ります。

デフォルトでは、バッファー・プールはページに基づいています。つまりディスク上の連続する複数ページは、メモリー内の不連続ページとしてプリフェッチされます。ディスク上の連続するページを読み取り、連続するページとしてバッファー・プール内に格納することができれば、順次プリフェッチが拡張されます。

このような目的で、ブロック・ベースのバッファー・プールを作成することができます。ブロック・ベースのバッファー・プールは、ページ・エリアとブロック・エリアから構成されます。ページ・エリアは、非順次プリフェッチ・ワークロード用に必要とされます。ブロック・エリアは複数のブロックで構成され、各ブロックには、指定された数の連続するページ (*ブロック・サイズ* という) が含まれます。

ブロック・ベースのバッファー・プールの最適な使用法は、指定するブロック・サイズによって異なります。ブロック・サイズは、順次プリフェッチを行う入出力サーバーがブロック・ベースの入出力を処理する際の細分度です。エクステントは、コンテナー間で表スペースがストライピングされるときの細分度です。ブロック・サイズが等しく定義された 1 つのバッファー・プールに、エクステント・サイズの異なる複数の表スペースをバインドできます。このため、エクステント・サイズとブロック・サイズを活用すれば、バッファー・プール・メモリーを効率的に使用することができます。以下のような状況では、バッファー・プール・メモリーがむだに消費される場合があります。

- (プリフェッチ要求サイズを決定する) エクステント・サイズが、バッファー・プールに指定された BLOCK_SIZE より小さい場合。
- プリフェッチ要求されたページのうち、いくつかのページがバッファー・プールのページ・エリアにすでに存在する場合。

入出力サーバーは各バッファー・プール・ブロックの一部のページがむだに消費されるのを許容しますが、ブロックのかなりの部分がむだに消費される場合、入出力サーバーはブロックに基づかないでバッファー・プールのページ・エリアにプリフェッチします。これは、最適なパフォーマンスではありません。

パフォーマンスを最適化するには、エクステント・サイズが等しいすべての表スペースを、表スペース・エクステント・サイズと同じブロック・サイズを持つバッファー・プールにバインドしてください。エクステント・サイズがブロック・サイズより大きい場合にも良好なパフォーマンスが得られますが、ブロック・サイズよりも小さい場合にはパフォーマンスが低下します。

ブロック・ベースのバッファー・プールを作成するには、 CREATE ステートメントおよび ALTER BUFFERPOOL ステートメントを使用します。

注: ブロック・ベースのバッファー・プールの用途は、順次プリフェッチです。アプリケーションが順次プリフェッチを使用しない場合、バッファー・プールのブロック・エリアはむだになります。

リスト・プリフェッチ

リスト・プリフェッチ または *リスト順次プリフェッチ* 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・プリフェッチは、単一または複数の索引アクセスで用います。

オプティマイザーが索引を使用して行にアクセスする場合、すべての行 ID (RID) 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、オプティマイザーは以前定義した索引 IX1 で索引スキャン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。

```
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'
```

この索引に従ってデータがクラスター化されていない場合、リスト・プリフェッチには、索引スキャンから獲得された RID のリストをソートするステップが含まれることになります。

プリフェッチと並列処理のための入出力サーバー構成

プリフェッチを使用可能にするため、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データ・ページを読み取ります。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プリフェッチ要求が出るまで待機します。このプリフェッチ要求には、照会を満たすために必要な入出力記述が含まれます。データベース・マネージャーがプリフェッチ要求をいつどのように実行するかは、使用できるプリフェッチ方式によって異なります。

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、データのプリフェッチ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。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。

入出力サーバーの数は、少なく評価するよりも多めに評価するほうが無難です。余分の入出力サーバーを指定した場合、これらの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ます。その結果、パフォーマンスには影響しません。各入出力サーバー・プロセスには、番号が付けられます。データベース・マネージャーは、常に最も低い番号のプロセスを使用するため、比較的高い番号のプロセスは、まったく使用されない場合があります。

必要な入出力サーバーの数を見積もる場合は、以下の点を考慮してください。

- 入出力サーバーのキューに同時にプリフェッチ要求を書き込めるデータベース・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。

num_ioservers データベース構成パラメーターの値を AUTOMATIC に設定して DB2 がシステム構成に基づくインテリジェント値を選出できるようにするのは、検討する価値があります。

非同期入出力の構成

一部のプラットフォームでは、ページ・クリーニングやプリフェッチといったアクティビティーのパフォーマンスを向上させるために、 DB2 が非同期入出力 (AIO) を使用します。AIO は、コンテナーに入っているデータが複数のディスクに分配される場合に最も有効です。パフォーマンスは、基礎のオペレーティング・システム AIO インフラストラクチャーをチューニングすることによっても改善されます。

たとえば、AIX® では、オペレーティング・システムで AIO のチューニングを行うこともできます。AIO が SMS ファイル・コンテナーと DMS ファイル・コンテナーの両方で稼働する場合は、 AIO サーバーというオペレーティング・システム・プロセスが入出力を管理します。このようなサーバーでは、まれに、AIO 要求の数が制限されることによって AIO の利点が制限される場合があります。 AIX 上の AIO サーバーの数を構成するには、 *smit* AIO *minservers* および *maxservers* パラメーターを使用します。

並列入出力を使用したプリフェッチの図: 次の図は、データのプリフェッチ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。

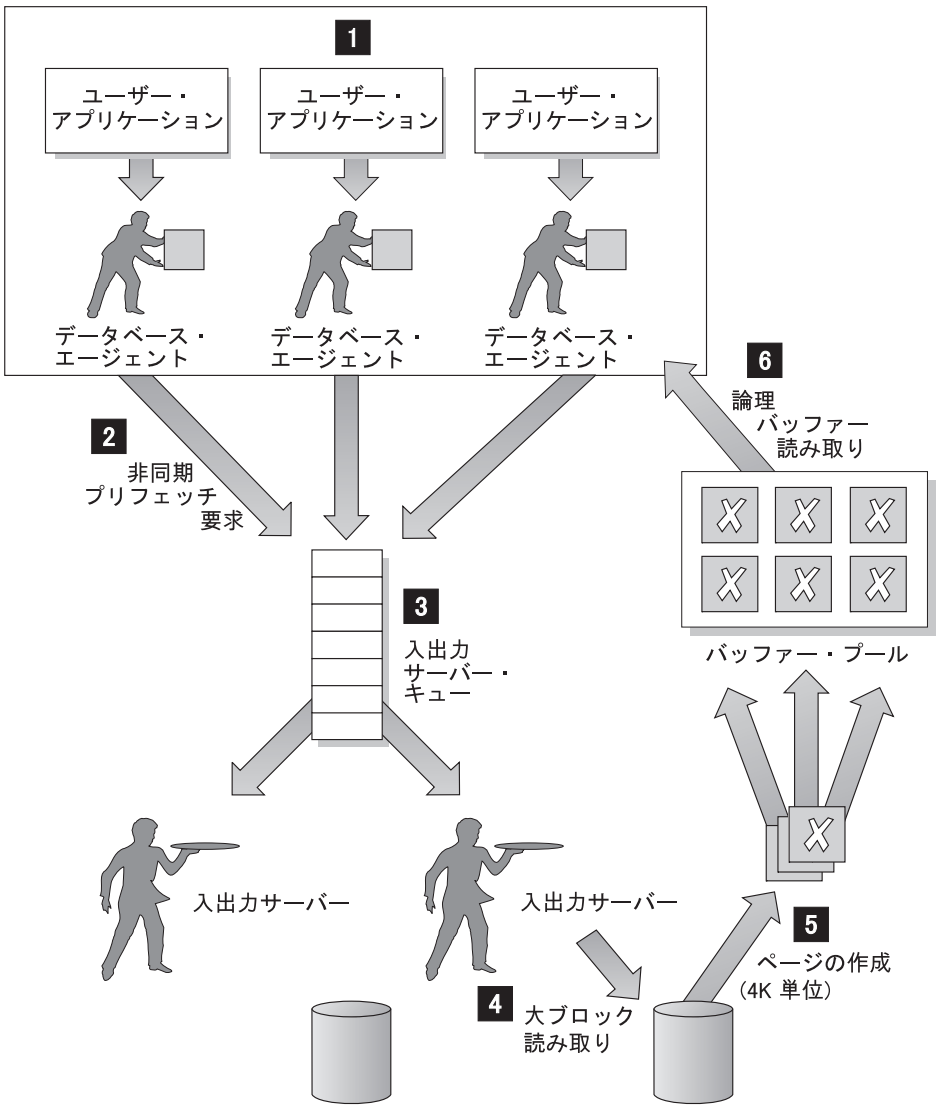


図 15. 入出力サーバーを使用したデータのプリフェッチ

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・エージェントに要求を渡します。

2、 **3**

データベース・エージェントが、要求を満たすのに必要なデータを獲得するために、プリフェッチを使用しなければならないかどうかを判別し、プリフェッチ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、キューからプリフェッチ要求を読み取り、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。表スペースから同時にデータを取り出せる入出力サーバーの数は、キューに入っているプリフェッチ要求の数と *num_ioservers* データベース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依存しています。

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走査実行し、その結果をユーザー・アプリケーションに戻します。

並列入出力の管理: 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場合、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。データベース・マネージャーはこの並列入出力において複数の入出力サーバーを使用して、 1 回の照会で入出力要件を処理します。各入出力サーバーは別個のコンテナーごとに入出力ワークロードを処理するため、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ワークロードを処理することができますが、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。そのため、物理装置と同じ数の入出力サーバーが必要になります。

並列入出力の開始は、以下に挙げる状況によりそれぞれ異なります。

- **順次プリフェッチ**

  順次プリフェッチの場合、プリフェッチ・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。それぞれのプリフェッチ要求は、エクステント境界に沿って、多く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・プリフェッチ**

  リスト・プリフェッチの場合、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、これらの小さいリスト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リストア**

  データのバックアップまたはリストアの場合、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リストア**

データのリストアの場合、並列入出力要求が開始され、順次プリフェッチで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・プールにリストアする代わりに、そのデータはリストア・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、入出力並列処理のレベルを指定することができます。このオプションを指定しない場合は、データベース・マネージャーは、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値を使用します。

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は、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少し多めに指定します。
- エクステント・サイズおよびプリフェッチ・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に、プリフェッチ・サイズを大きくし過ぎないでください。理想的なサイズは、エクステント・サイズ、各コンテナーの物理ディスクの数 (RAID 装置を使用している場合)、および表スペース・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 から 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存在すること。
- 並列処理の多重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、並列プリフェッチが最適化される可能性が少なくなります。次の例を考慮してください。

  – 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、データのプリフェッチ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、並列プリフェッチ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、並列プリフェッチ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、並列プリフェッチを使用することができなくなります。
- パーティション内並列処理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。 SMP マシンの場合、パーティション内並列処理を使用し、複数のプロセッサーで照会を実行することによって、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。

プリフェッチを高速で行い、入出力容量を効率的に使用するためには、プリフェッチ・サイズをできるだけ大きく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。

ソート・パフォーマンスのチューニング: 照会において、ソートまたはグループ化された結果が必要になることがよくあるので、ソートは頻繁に必要になります。また、ソート・ヒープ領域を正しく構成することも、照会で高いパフォーマンスを得るためにきわめて重要です。次のような場合に、ソートが必要になります。

- 要求された配列を満たす索引がない (たとえば、 ORDER BY 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ソートを行う方が効率が良い場合。
- 索引が作成される場合。
- 索引がドロップされることにより、索引ページ番号がソートされます。

ソートに影響を与えるエレメント

ソート・パフォーマンスに影響を与える次のような因子があります。

- 次のデータベース構成パラメーターの設定値:
  - ソート・ヒープ・サイズ (*sortheap*)。これは、ソートごとに使用するメモリー量を指定します。
  - ソート・ヒープしきい値 (*sheapthres*)、および共用ソートのソート・ヒープしきい値 (*sheapthres_shr*)。これは、すべてのソートのインスタンス全体で使用可能な、ソート用の合計メモリー量の制御を行います。
- 大量のソートを必要とするワークロードにおけるステートメントの数。
- 不要なソートを避けるのに役立つ索引の存在と不在
- ソートを最小化しないアプリケーション論理の使用
- 並列ソート。ソートのパフォーマンスを向上させるが、ステートメントがパーティション内並列処理を使用する場合にのみ行われます。
- ソートが、*オーバーフロー* と、*オーバーフローしない* のいずれであるか。ソートされるデータがソート・ヒープ (ソート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、データベースの一時表にオーバーフローします。オーバーフローしないソートの方が常に、オーバーフローするソートよりも効率良く実行されます。
- ソートの結果が、*パイプ* と*非パイプ* のいずれかであるか。データの最終ソート結果リストを保管する一時表を使わなくても、ソートされた情報を直接戻すことができる場合は、そのソートはパイプ・ソートになります。ソートされた情報を戻すのに一時表が必要な場合には、そのソートは非パイプ・ソートになります。パイプ・ソートは、非パイプ・ソートよりも常に効率よく実行されます。

  また、パイプ・ソートでは、アプリケーションがそのソートに関連したカーソルをクローズするまで、ソート・ヒープが解放されないことに注意してください。パイプ・ソートはカーソルがクローズされるまでメモリーを消費し続けることができます。

一般に、インスタンス (*sheapthres*) で使用可能なソート・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必要があります。ソートはソート・メモリー内で全体的に実行されますが、これは過度のページ・スワッピングを引き起こす可能性があります。このような場合、ラージ・ソート・ヒープの利点が生かされません。そのため、ソート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・ページングの変更を追跡してください。

ソート・パフォーマンスの管理技法

ソート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します。

1. 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、ソート合計時間が最も長いアプリケーションを識別します。
2. そのようなアプリケーションのそれぞれにおいて、*ソート合計時間* が最も長いステートメントを見つけます。

   Explain 表を検索して、ソート操作が行われている照会を識別することもできます。
3. これらのステートメントは設計アドバイザーへの入力として使用してください。これにより、ソートの必要を減らす索引が識別され、オプションで作成されます。

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してください。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・マネージャーごとに、またデータベースごとに、次のことを実行してください。

1. 代表的なワークロードを設定し稼働します。
2. 適用するデータベースごとに、ベンチマーク・ワークロード期間中の次のパフォーマンス変数の平均値を収集します。
   - 使用中のソート・ヒープ合計 (sort_heap_allocated モニター・エレメントの値)
   - アクティブ・ソートおよびアクティブ・ハッシュ結合 (active_sorts および active_hash_joins モニター・エレメントの値)。
3. データベースごとに、*sortheap* を *使用中のソート・ヒープ合計* の平均値に設定します。

   注: DB2 部分キーのバイナリー・ソート技法に、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、長いキーをソートする時には追加のメモリーが必要です。ソートで長いキーが使用される場合、 *sortheap* 構成パラメーターを増やす必要があるかもしれません。
4. *sheapthres* を設定します。適切なサイズを見積もるには、次のようにします。
   a. 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。
   b. このデータベースのソート・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、最大のソート・ヒープの 80% を使用します。

c. アクティブなソートの平均数に、算出したソート・ヒープの平均サイズを掛けた値に *sheapthres* を設定します。

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。

セルフチューニング・メモリー機能を使って、ソートに必要なメモリー・リソースを、自動的および動的に、割り振りおよび割り振り解除することもできます。この機能を使用するには、以下のことを行ってください。

- *self_tuning_mem* 構成パラメーターを「ON」に設定して、データベースのセルフチューニング・メモリーを使用可能にします。
- *sortheap* および *sheapthres_shr* 構成パラメーターを「AUTOMATIC」に設定します。
- *sheapthres* 構成パラメーターを「0」に設定します。

表および索引の編成の保守

時間と共に、レコードがより多くのデータ・ページに分散されるため、表のデータはフラグメント化し、表と索引のサイズが増加します。これによって、照会実行中に読み取る必要のあるページ数が増加します。表と索引の再編成によってデータはコンパクトになり、無駄なスペースが再利用されて、データ・アクセスが改善されます。

索引または表の再編成を実行するためのステップは、次のとおりです。

1. 再編成しなければならない表や索引があるか判断します。
2. 再編成の方式を選択します。
3. 対象となるオブジェクトの再編成を実行します。
4. 再編成の進行をモニターします。
5. オンライン表再編成の場合、必要に応じて再編成プロセスを一時停止できます。これは後で再開できます。
6. 再編成の結果を評価し、操作が成功したか失敗したかを判断します。オフライン表再編成および索引再編成の場合、操作は同期になり、再編成の結果はコマンドが戻るときに明らかになります。オンライン表再編成は非同期で処理されるため、結果を評価するには、履歴ファイルを参照する必要があります。
7. オンライン表再編成を実行した場合、それをリカバリーするか選択できます。98 ページの『失敗したオンライン表再編成のリカバリー』を参照。
8. 再編成されたオブジェクトについての統計を収集します。
9. 再編成されたオブジェクトにアクセスするアプリケーションを再バインドします。

表の再編成

表データに多くの変更を加えた後、論理上順次データは、順序どおりになっていない物理データ・ページ上にある場合があるので、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。多数の行を削除した場合にも、追加の読み取り操作が必要になります。このような場合、表

を再編成することにより、索引と一致させ、スペースを再利用することを検討することができます。データベース表だけでなく、システム・カタログ表も再編成できます。

注: 通常、表の再編成は統計の実行より時間がかかるため、 RUNSTATS を実行してデータの現在の統計をリフレッシュし、アプリケーションを再バインドすることもできます。統計のリフレッシュでパフォーマンスが向上しない場合、再編成が役に立ちます。 REORG TABLE ユーティリティーのオプションと動作に関する詳細は、「コマンド・リファレンス」を参照してください。

表の再編成が必要であることを示す、次の要因を検討してください。

- 照会によってアクセスされる表における大量の挿入、更新、および削除アクティビティー
- 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おける顕著な変化
- RUNSTATS を実行して統計情報をリフレッシュしても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
- 照会パフォーマンスの低下を向上させた場合のコストと、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キング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) の間のトレードオフ。

表の再編成の必要性を少なくする

表の再編成の必要性を少なくするには、表を作成した後に以下のタスクを行ってください。

- 表を変更して PCTFREE を追加します。
- 索引上の PCTFREE を使ってクラスタリング索引を作成します。
- データをソートします。
- データをロードします。

これらのタスクを実行した後、表およびそのクラスタリング索引と、表上の PCTFREE の設定値は、元のソート順序を保存するのに役立ちます。表ページに十分なスペースがある場合、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、索引のクラスタリング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、上のクラスター特性は徐々に失われます。

クラスタリング索引を作成した後に REORG TABLE またはソートを実行し、LOAD を実行すると、その索引はデータの特定の順序を保持しようするので、RUNSTATS ユーティリティーによって収集された CLUSTERRATIO または CLUSTERFACTOR 統計は改善されます。

注: マルチディメンション・クラスタリング (MDC) 表を作成すると、表を再編成する必要性は少なくなります。 MDC 表のクラスタリングは、 CREATE TABLE ステートメントの ORGANIZE BY DIMENSIONS 節の引数として指定された列に保守

されます。ただし、未使用ブロックが多すぎる、またはブロックをコンパクトにする必要があると判断される場合は、 REORGCHK は MDC 表の再編成を勧める場合があります。

表を再編成する方式の選択

表再編成には 2 つの異なる方式 (*従来の* オフライン REORG と、*インプレース* のオンライン REORG) があります。

REORG コマンドの INPLACE オプションは、オンライン再編成を指定します。これを指定しない場合、オフライン REORG が実行されます。

それぞれの再編成方式には、利点と欠点があります。それらについては以下に要約しています。再編成方式を選択する際、どちらの方式がユーザーの優先順位に適合する利点を備えているか考慮してください。例えば、再編成を素早く完了することよりも、失敗したときにリカバリーできることのほうが重要な場合は、オンライン再編成の方式が最も適した方式と考えられます。

オフライン再編成の利点

- 表再編成が最も高速で行える (特に LOB/LONG データの再編成が不要な場合)。
- 完了した時点で、表および索引が完全にクラスター化される。
- 表が再編成されると索引が再作成される。索引の再作成のために、別の手順を実行する必要がありません。
- TEMPORARY 表スペースにシャドー・コピーを作成できる。これによって、ターゲット表や索引を含む表スペースに必要なスペース量が削減されます。
- データを再クラスター化するのに、クラスタリング索引以外の索引を指定および使用することができる。オンライン再編成では、存在する場合、既存のクラスタリング索引を使用しなければなりません。

オフライン再編成の欠点

- 表へのアクセスが制限される。アプリケーションには、表への読み取りアクセスのみが、再編成の SORT フェーズおよび BUILD フェーズ中にのみ許可されます。
- シャドー・コピー方式が使用されるので、大きなスペースが必要になる。
- オンライン REORG に比べて REORG 処理をよく制御できない。オフライン再編成は、一時停止して再始動することはできません。

オンライン再編成の利点

- アプリケーションは、REORG 中、切り捨てフェーズ中以外は表へのフル・アクセスが許可される。
- REORG 処理をよりよく制御できる。処理はバックグラウンドで非同期に実行し、一時停止、再開、および停止することができます。例えば、表に対して多数の更新または削除が実行されている場合、REORG 処理を一時停止できます。
- 障害が発生した場合、処理をリカバリーできる。
- 表が増分的に処理されるので、作業用ストレージが少なくてすむ。
- 再編成の効果が REORG 進行中にすぐに分かる。

オンライン再編成の欠点

- REORG 中に表にアクセスするトランザクションのタイプによっては、結果としてデータのクラスター化や索引のクラスター化が完全には実行されない。
- 再編成の始めに再編成されたページにさらなる更新が行われ、それによって処理の後のほうで再編成された表よりも細かくフラグメント化される可能性がある。
- オフライン REORG に比べて実行速度が遅い。通常のクラスタリング REORG の場合 (単なるスペースの再利用でない場合)、オンライン REORG の実行には 10 倍から 20 倍の時間がかかります。(複数のアプリケーションが同時に実行している表、または定義された索引の数が多い表の場合は、非常に長い時間かかる可能性があります。)
- オンライン REORG はリカバリー可能な処理だが、その代わりにロギング要件が増える。膨大な量のログ・スペースが必要になることもあります (最大で表の数倍のサイズまで)。これは REORG 中に移動された行の数、表に定義された索引の数、および索引のサイズによって決まります。
- 索引は保守されますが、再作成はされません。そのため後から索引の再編成が必要になることがあります。

表 2. オンライン再編成とオフライン再編成の比較

特性	オフライン再編成	オンライン再編成
パフォーマンス	高速	低速
完了時のデータのクラスター係数	良好	完全にはクラスター化されない
並行性 (表へのアクセス)	NO ACCESS から READ ONLY の範囲	READ ONLY からフル・アクセスの範囲
データ・ストレージ・スペース所要量	大きい	大きくない
ロギング・ストレージ・スペース所要量	大きくない	大きくなることがある
ユーザー制御 (処理を一時停止/再開できるかどうか)	あまりよく制御できない	よく制御できる
リカバリー可能性	部分的なリカバリーができない。成功か失敗のみ。	リカバリー可能
索引の再作成	行われる	行われない
すべてのタイプの表のサポート	はい	いいえ
クラスタリング索引以外の索引の指定	はい	いいえ
TEMPORARY 表スペースの使用	はい	いいえ

表 3. オンライン再編成およびオフライン再編成に関してサポートされる表タイプ

表タイプ	オフライン再編成のサポート	オンライン再編成のサポート
マルチディメンション・クラスタリング表 (MDC)	あり[1]	いいえ
範囲クラスター表 (RCT)	なし[2]	いいえ

表 3. オンライン再編成およびオフライン再編成に関してサポートされる表タイプ (続き)

表タイプ	オフライン再編成のサポート	オンライン再編成のサポート
付加モードの表	いいえ	なし[3]
LONG/LOB データのある表	あり[4]	いいえ
タイプ 1 索引のある表	あり[5]	いいえ
システム・カタログ表 SYSIBM.SYSTABLES、 SYSIBM.SYSSEQUENCES、 SYSIBM.SYSDBAUTH、 SYSIBM.SYSROUTINEAUTH	はい	いいえ

1. クラスター化は MDC ブロック索引経由で自動的に維持されるため、MDC 表の再編成にはスペース再利用のみが関係します。ブロック索引が使用されるので、索引は指定できません。
2. RCT の範囲域は常にクラスター化されたままになります。
3. 付加モードをオフにした後、オンライン再編成を実行できます。
4. LONG/LOB データの再編成には、膨大な時間がかかることがあります。LONG/LOB データの再編成によって照会パフォーマンスがあがることはありません。スペース再利用のためにのみ行う必要があります。
5. 再編成後に、すべての索引はタイプ 2 索引として再作成されます。

これらの表再編成方式の実行方法について、詳しくは REORG TABLE 構文の説明を参照してください。

表再編成の進行状況のモニター

表再編成の現在の進行状況は、データベース・アクティビティーの履歴ファイルに書き込まれます。履歴ファイルには、それぞれの再編成イベントに関するレコードが含まれます。このファイルを表示するには、再編成対象の表が格納されているデータベースに対して `db2 list history` コマンドを実行してください。

このほか、表スナップショットを使用して、表再編成の進行状況をモニターすることもできます。表再編成モニター・データは、データベース・モニター表スイッチの設定値にかかわらず記録されます。

エラーが発生した場合、SQLCA ダンプが履歴ファイルに書き込まれます。インプレース表再編成の場合、状況は「PAUSED (一時停止)」と記録されます。

オフライン表再編成

オフライン表再編成では、シャドー・コピー方式を使用して、再編成される表の完全なコピーを作成します。

従来のオフライン表再編成には、4 つのフェーズがあります。

1. *SORT*
   索引が REORG TABLE コマンドで指定された場合、またはクラスタリング索引が表で定義されている場合、表の行は最初にその索引に従ってソートされます。INDEXSCAN オプションが指定された場合、索引スキャンが表をソートするのに

使用されます。それ以外の場合、表スキャン・ソートが使用されます。このフェーズはクラスタリング REORG にのみ適用されます。スペース再利用再編成は BUILD フェーズから始まります。

2. *BUILD*
   このフェーズでは、再編成された表全体のコピーが、再編成される表が常駐する表スペース内か、REORG コマンドで指定された TEMPORARY 表スペース内のいずれかに作成されます。
3. *REPLACE*
   このフェーズでは、TEMPORARY 表スペースからコピーを戻すことによって、または、再編成されている表の表スペースにある新しく作成されたオブジェクトをポイントすることによって、元の表オブジェクトが置換されます。
4. *RECREATE ALL INDEXES*
   表に定義されたすべての索引が再作成されます。

スナップショット・モニターまたはスナップショット管理ビューを使用して、REORG TABLE の進行をモニターして、再編成が現在どのフェーズにいるのか判断できます。

オンライン REORG よりも オフライン表 REORG の方が、ロック状態による制限が厳しくなります。コピーの作成中に、表への読み取りアクセスは可能です。しかし、元の表を再編成したコピーに置き換える際、および索引を再作成する際は、表の排他的アクセスが必要となります。

REORG 処理の間は終始 IX 表スペース・ロックが必要です。BUILD フェーズでは、U ロックが必要で、これが表に対して保持されます。U ロックでは、ロック所有者は表のデータを更新できます。しかし、他のアプリケーションはデータを更新できません (読み取りアクセスのみ許可されます)。REPLACE フェーズが開始すると、U ロックは Z ロックにアップグレードされます。このフェーズ中は、他のアプリケーションはデータにアクセスできません。このロックは REORG が完了するまで保持されます。

オフライン再編成処理によって、いくつかのファイルが作成されます。これらのファイルはデータベース・ディレクトリーに保管され、先頭に表スペース ID とオブジェクト ID が付いています。例えば、0030002.ROR は表スペース ID が 3、表 ID が 2 の表の再編成の状態ファイルです。

以下は、SMS 表のオフライン REORG の際に作成される一時ファイルです。

- .DTR データ・シャドー・コピー・ファイル
- .LFR LONG フィールド・ファイル
- .LAR LONG フィールド割り振りファイル
- .RLB LOB データ・ファイル
- .RBA LOB 割り振りファイル
- .BMR ブロック・オブジェクト・ファイル (MDC 表用)

以下は、索引再編成の際に作成される一時ファイルです。

- .IN1 シャドー・コピー・ファイル

以下は、SORT フェーズの際に、システム一時表スペースに作成される一時ファイルです:

- .TDA データ・ファイル
- .TIX 索引ファイル
- .TLF LONG フィールド・ファイル
- .TLA LONG フィールド割り振りファイル
- .TLB LOB ファイル
- .TBA LOB 割り振りファイル
- .TBM ブロック・オブジェクト・ファイル

再編成処理に関連するファイルは、システムから手動で除去しないでください。

オフラインで表を再編成する:

オフラインでの表の再編成は、表をデフラグする最も速い方法です。再編成によって表のスペース所要量は削減され、データ・アクセスおよび照会パフォーマンスが改善されます。

表を再編成するには、SYSADM 権限、SYSCTRL 権限、SYSMAINT 権限、または DBADM 権限か、表に対する CONTROL 特権が必要です。表を再編成するにはデータベース接続が必要です。

再編成を必要とする表を見つけたら、それらの表に (またオプションで、それらの表で定義された索引に) REORG ユーティリティーを実行できます。

1. CLP を使用して表を再編成するには、REORG TABLE コマンドを発行します。

   ```
   db2 reorg table test.employee
   ```

   TEMPORARY 表スペース mytemp を使用して表を再編成するには、次のように入力します。

   ```
   db2 reorg table test.employee use mytemp
   ```

   表を再編成し、索引 myindex に従って行を再配列する場合、次のように入力します。

   ```
   db2 reorg table test.employee index myindex
   ```

2. SQL 呼び出しステートメントを使用して表を再編成するには、次のように ADMIN_CMD プロシージャーを使用して REORG TABLE コマンドを発行します。

   ```
   call sysproc.admin_cmd ('reorg table employee index myindex')
   ```

3. DB2 管理 API を使用して表を再編成する場合、db2REORG API を使用します。

表の再編成後、表の統計を収集して、オプティマイザーが照会アクセス・プランを評価するために最も正確なデータを保有するようにする必要があります。

オフライン表再編成のリカバリー:

オフライン表再編成は、REPLACE フェーズが始まるまでは、「全か無か」の処理です。つまり、SORT フェーズまたは BUILD フェーズ中にシステムが破損した場

合、REORG TABLE 操作はロールバックされ、クラッシュ・リカバリーで再実行されません。代わりに、リカバリー後に REORG TABLE コマンドを再実行しなければなりません。

REPLACE フェーズ開始後にシステムが破損した場合は、REORG TABLE 操作を完了しなければなりません。これは、すべての REORG TABLE 作業が行われ、元の表が使用できない状態になっている可能性があるためです。クラッシュ・リカバリー中に再編成されたオブジェクトの一時ファイルが必要になりますが、ソートに使用した TEMPORARY 表スペースは必要ありません。リカバリーでは REPLACE フェーズが始めから再開され、リカバリーにはコピー・オブジェクト内のすべてのデータが必要となります。この場合、SMS 表スペースと DMS 表スペースの間で相違点があります。再編成が同じ表スペース内で行われた場合、SMS オブジェクトは、1 つのオブジェクトから別のオブジェクトにコピーする必要がありますが、DMS では、新しく再編成されたオブジェクトをポイントするだけで、元の表がドロップされます。索引は再作成されませんが、クラッシュ・リカバリー中に無効とマークされ、データベースは標準の規則に従って、索引をいつ再作成するか (データベースを再開するときか、索引に最初にアクセスするときか) を決定します。

索引 REBUILD フェーズで破損が生じた場合は、新しいオブジェクトがすでに存在するので、何も再実行されません。(上で説明されているように、索引は再作成されませんが、クラッシュ・リカバリー中に無効とマークされ、データベースは標準の規則にのっとって、索引をいつ再作成するか、データベースを再開するときか索引に最初にアクセスするときかを決定します。)

ロールフォワード・リカバリーについては、古いバージョンの表がディスク上にある場合、REORG TABLE が再実行されます。ロールフォワードでは、BUILD フェーズ中にログに記録された RID を使用して BUILD フェーズと REPLACE フェーズが繰り返され、再編成された表を作成した操作順序が再適用されます。この場合も、前述のように索引は無効とマークされます。これは索引スキャンまたはスキャン・ソートがまったく実行されていないことを意味し、TEMPORARY 表スペースについては、もともと TEMPORARY 表スペースが使用されていた場合に、再編成されたオブジェクトのコピー用の TEMPORARY 表スペースのみが必要となります。並列リカバリーのため、ロールフォワード中に複数の REORG TABLE 操作が同時に再実行されることがあります。この場合、ディスク・スペース使用量は実行時よりも大きくなります。

オフライン表再編成のパフォーマンスの改善:

オフライン表再編成のパフォーマンスは、データベース環境の特性によって大きく左右されます。

NO ACCESS モードで REORG TABLE を実行しても、ALLOW READ ACCESS モードで REORG TABLE を実行しても、実質的にパフォーマンスは変わりません。唯一の違いは、READ ACCESS モードでは、DB2 は表のロックを置換する前にアップグレードするので、ユーティリティーは、既存のスキャンが完了してそのロックを解除するまで待たなければならない場合があることです。どちらの場合でも、REORG TABLE 操作の索引再作成フェーズ中は、表は使用不可になります。

パフォーマンス向上のためのヒント

表スペースに十分なスペースが存在する場合は、TEMPORARY 表スペースは使わずに、元の表と再編成された表のコピーに同じ表スペースを使います。これによって、TEMPORARY 表スペースから表をコピーするのに要する時間を節約できます。

- REORG TABLE 操作中に保守する必要のある索引を減らすため、表を再編成する前に不要な索引をドロップすることを考慮します。
- 再編成された表が常駐する表スペースのプリフェッチ・サイズが正しく設定されていることを確認します。
- INTRA_PARALLEL を使用可能にして、索引再作成が並列処理で行われるようにします。
- データベース構成パラメーター *sortheap* および *sheapthres* を調整して、ソートのためのスペースを制御します。それぞれのプロセッサーは専用ソートを実行するため、*sheapthres* の値は少なくとも*sortheap* x *使用するプロセッサー数* でなければなりません。
- ページ・クリーナーの数を調整して、ダーティー索引ページができるだけ早くバッファー・プールからクリーンアップされるようにします。

オンライン表再編成

オンライン表再編成、すなわちインプレース表再編成では、表に対するフル・アクセスを許可しながら表を再編成できます。オンライン REORG TABLE では、中断なしでデータにアクセスできる一方、オンライン REORG TABLE のパフォーマンスはオフライン REORG TABLE よりも遅くなります。

オンライン表再編成中、表全体は一度に再編成されません。むしろ、表の一部が順番に再編成されます。データは TEMPORARY 表スペースにコピーされません。行が既存の表オブジェクト内で移動してクラスター化が再確立され、フリー・スペースが再利用され、オーバーフロー行が除去されます。

オンライン REORG TABLE には、次の 4 つの基本フェーズがあります。

1. N ページを選択する
   このフェーズでは、DB2 が N ページを選択します (N は REORG TABLE 処理を行う連続するページのエクステント・サイズ。最小は 32)。
2. 範囲を空ける
   N ページを選択した状態で、オンライン REORG TABLE はこの範囲内のすべての行を、表内の空きページに移動します。移動した各行は、行の新しいロケーションの RID を含む RP (REORG TABLE ポインター) レコードを残します。行はそのデータを含む RO (REORG TABLE オーバーフロー) レコードとして表の空きページに挿入されます。
   REORG TABLE が一連の行の移動を完了すると、その表に発生する既存のすべてのデータ・アクセス (例えば、現在実行中のアプリケーションによるアクセス) が完了するのを待機します。これら既存のアクセス (古いスキャナーと呼ばれる) は表データにアクセスする際、古い RID を使用します。この待機期間中に開始したすべてのアクセス (新しいスキャナーと呼ばれる) は、新しい RID を使ってデータにアクセスします。すべての古いスキャナーが完了すると、REORG TABLE は RP レコードを削除し、 RO レコードを標準レコードに変換することによって、移動した行をクリーンアップします。

3. 範囲を埋める
   すべての行が空くと、行が再編成された形式 (使用されている索引に従ってソートされ、定義されている PCTFREE 制限に従った形式) で再び書き込まれます。範囲内のすべてのページが埋まると、次の連続した N ページが表で選択され、再び処理が始まります。
4. 表を切り捨てる
   デフォルトでは、表のすべてのページが再編成されると、スペースの再利用のために表が切り捨てられます。NOTRUNCATE オプションが指定されている場合は、再編成された表は切り捨てられません。

オンライン表 REORG 中に作成されるファイル

オンライン表再編成中に、データベース・パーティションごとに .OLR 状態ファイルが作成されます。このファイルは、 xxxxyyy.OLR という名前のバイナリー・ファイルです (xxxx はプール ID、yyyy はオブジェクト ID (16 進数形式) をそれぞれ表します)。このファイルには、オンライン再編成を一時停止の状態から再開するのに必要な情報が含まれています。

状態ファイルには、以下の情報が含まれています。

- REORG のタイプ。
- 再編成されている表のライフ LSN
- 次に空になる範囲
- 再編成がデータをクラスター化しているか、それとも単にスペースを再利用しているだけか
- データをクラスター化するのに使用される索引の ID

.OLR ファイルに関してはチェックサムが維持されます。ファイルが破損してチェックサム・エラーが発生した場合や表 LSN がライフ LSN と一致しなかった場合は、新しい REORG を開始しなければならなくなり、新しい状態ファイルが作成されます。

.OLR 状態ファイルが削除された場合は、REORG TABLE 処理を再開できず、SQL2219 エラーが戻されます。新しい REORG TABLE 処理を開始しなければなりません。

再編成処理に関連するファイルは、システムから手動で除去しないでください。

オンラインで表を再編成する:

オンラインまたはインプレース表再編成では、表の再編成中に表にアクセスすることができます。

表を再編成するには、SYSADM 権限、SYSCTRL 権限、SYSMAINT 権限、または DBADM 権限か、表に対する CONTROL 特権が必要です。表を再編成するにはデータベース接続が必要です。

オンライン表再編成は、CLP コマンドを使用するか、SQL 呼び出しステートメントを使用するか、あるいは DB2 API を通して実行できます。

1. CLP を使用してオンラインで表を再編成するには、次のように INPLACE オプションを使用して REORG TABLE コマンドを発行します。

   ```
   db2 reorg table test.employee inplace
   ```

2. SQL 呼び出しステートメントを使用してオンラインで表を再編成するには、次のように ADMIN_CMD プロシージャーを使用して REORG TABLE コマンドを発行します。

   ```
   call sysproc.admin_cmd ('reorg table employee inplace')
   ```

3. DB2 管理 API を使用して表を再編成する場合、db2REORG API を使用します。

表の再編成後、表の統計を収集して、オプティマイザーが照会アクセス・プランを評価するために最も正確なデータを保有するようにする必要があります。

失敗したオンライン表再編成のリカバリー:

オンライン表再編成の失敗は、ディスク・フルやロギング・エラーといった処理エラーによるものが多数です。オンライン表再編成が失敗した場合、SQLCA メッセージが履歴ファイルに書き込まれます。

パーティション・データベース環境の 1 つ以上のデータベース・パーティションにエラーが発生した場合、戻される SQL コードはエラーを報告した最初のノードからのものになります。

ランタイム中に失敗が発生した場合、オンライン表再編成は一時停止されてロールバックされます。システムがダウンした場合、再始動の際、クラッシュ・リカバリーが始まり、再編成は一時停止されてロールバックされます。その後、REORG TABLE コマンドで RESUME オプションを指定して、再編成を再開できます。オンライン表再編成処理は完全にログに記録されるため、再編成は確実にリカバリーできます。

例えば、ある状況においては、オンライン表再編成は *num_log_span* 制限を超えることがあります。この場合、DB2 は REORG TABLE を強制して一時停止の状態にします。スナップショット出力では、REORG TABLE ユーティリティーの状態は「PAUSED」として示されます。

オンライン表再編成の一時停止は、割り込みによるものです。つまり、ユーザーによって (REORG TABLE コマンドの一時停止オプションや強制終了アプリケーション・コマンドを使用して) 一時停止されることもあれば、状況によっては (例えば、システムが破損した場合) DB2 によって一時停止されることもあります。

オンライン表再編成の一時停止と再開:

進行中のオンライン表再編成は、一時停止して、再開することができます。

オンライン表再編成を一時停止または再開するには、次のいずれかが必要です。

- sysadm
- sysctrl
- sysmaint
- dbadm

- 表に対する CONTROL 特権

1. オンライン表再編成を一時停止するには、 次のように PAUSE オプションを使用して REORG TABLE コマンドを発行します。

   ```
   db2 reorg table homer.employee inplace pause
   ```

2. 一時停止したオンライン表再編成を再開するには、 次のように RESUME オプションを指定して REORG TABLE コマンドを発行します。

   ```
   db2 reorg table homer.employee inplace resume
   ```

   注:

   - オンライン表再編成が一時停止されると、その表の新規の再編成は開始できません。新規の再編成処理を開始する前に、一時停止した再編成を再開するか、一時停止した再編成を停止しなければなりません。
   - RESUME 要求が発行されると、後の START 要求または中間の RESUME 要求でどんな TRUNCATE オプションが指定されようとも、再編成処理は元の REORG コマンドで指定された TRUNCATE オプションを採用します。しかし切り捨てフェーズで再編成が行われ、ユーザーが NOTRUNCATE を指定した RESUME 要求を発行した場合、表は切り捨てられずに再編成が完了します。
   - リストア操作およびロールフォワード操作後に、再編成を再開することはできません。

オンライン表再編成のロックおよび並行性に関する考慮事項:

オンライン表再編成のもっとも重要な面の一つは、ロックの制御方法です。これはアプリケーション並行性に関して重要なことであるためです。

オンライン REORG TABLE 中の特定の時点で、操作により以下のロックが保持されます。

- 表スペースへの書き込みアクセスを確実にするために、REORG TABLE 操作の影響を受ける表スペースに対する IX ロックが取得されます。
- 表ロックが取得され、REORG TABLE のすべての操作の間保持されます。ロックのレベルは、再編成中にその表に許可されるアクセス・モードによって決まります。
  ALLOW WRITE ACCESS が指定されている場合、表に対する IS ロックが取得されます。
  ALLOW READ ACCESS が指定されている場合、表に対する S ロックが取得されます。
- 切り捨てフェーズ中、再編成が切り捨て範囲の外に行を移動するときは、古いスキャナー (REORG TABLE 操作中に存在し、レコードの古い RIDにアクセスするデータ・アクセス) によって新しい行が挿入される可能性があるため、表に対する S ロックが要求されます。 DB2 はこのロックが取得されるまで待ってから、切り捨てを開始します。切り捨てフェーズの最後に REORG TABLE が表の物理的切り捨てを行うとき、S ロックは特殊な Z ロックにアップグレードされます。これは、既存のアプリケーションによって保持される表ロック (UR スキャナーからの IN ロックを含む) がすべてなくなるまで、REORG TABLE 操作は完了しないことを意味します。

- 表ロックのタイプによっては、行ロックも取得されることがあります。
  表に対する S ロックが保持されている場合、個々の行のレベルの S ロックは不要なので、ロックはそれ以上必要ありません。
  表に対する IS ロックが保持されている場合は、行が移動される前に行レベルの S ロックが取得され、移動完了後に解放されます。
- オンライン REORG TABLE や、オンライン・バックアップといった他のオンライン DB2 ユーティリティーのオブジェクトへのアクセスを制御するために、内部ロックが必要となる場合があります。

ロックは REORG TABLE および同時ユーザー・アプリケーションの両方のパフォーマンスに影響を与えます。オンライン表再編成を実行する際は、ロック・アクティビティーについて熟知するために、ロック・スナップショット・データを調べることを強くお勧めします。

表再編成のモニター

GET SNAPSHOT コマンド、SNAPTAB_REORG 管理ビュー、または SNAP_GET_TAB_REORG 表関数を使用して、表再編成操作の状況に関する情報を得ることができます。

データベースに接続されていなければならず、また次の権限が必要です。

- SYSMON 権限
- SNAPTAB_REORG 管理ビューに対する SELECT 特権か CONTROL 特権、または SNAP_GET_TAB_REORG 表関数に対する EXECUTE 特権。
- SQL を使用して再編成操作に関する情報にアクセスするには、SNAPTAB_REORG 管理ビューを使用します。 例えば、次の SELECT ステートメントは、現在接続されているデータベースのすべてのデータベース・パーティションに対する表再編成操作に関する詳細を戻します。

  ```
  SELECT SUBSTR(TABNAME, 1, 15) AS TAB_NAME, SUBSTR(TABSCHEMA, 1, 15)
     AS TAB_SCHEMA, REORG_PHASE, SUBSTR(REORG_TYPE, 1, 20) AS REORG_TYPE,
     REORG_STATUS, REORG_COMPLETION, DBPARTITIONNUM
     FROM SYSIBMADM.SNAPTAB_REORG ORDER BY DBPARTITIONNUM
  ```

  再編成されている表がまったくない場合、0 行が戻されます。
- スナップショット・モニターを使用して再編成操作に関する情報にアクセスするには、GET SNAPSHOT FOR TABLES ON *database* コマンドを発行して、表再編成モニター・エレメント (先頭に reorg_ が付いている) の値を調べます。

オフライン REORG TABLE は同期なので、オフライン表再編成で発生したすべてのエラーは、ユーティリティーの呼び出し元 (アプリケーションまたはコマンド行のどちらか) に戻されます。

オンライン表再編成のエラー処理は、オフライン表再編成のエラー処理と若干異なります。オンライン表再編成は非同期のため、CLP に SQL メッセージは書き込まれません。オンライン表再編成中に戻された SQL エラーを表示するには、LIST HISTORY REORG コマンドを発行します。

オンライン表再編成は db2Reorg 処理としてバックグラウンドで実行します。これは、REORG TABLE 処理から呼び出し元アプリケーションに正常な結果が戻されて

も、再編成が正常に完了したことにはならないことを意味します呼び出し元アプリケーションがそのデータベース接続を終了しても db2Reorg 処理は続行します。

索引の再編成

表は、削除や挿入によって更新されるので、索引パフォーマンスは次のような場合に低下します。

- リーフ・ページのフラグメント化。

  リーフ・ページがフラグメント化されると、表ページを取り出すためにさらに多くのリーフ・ページを読み取らなければならないので、入出力のコストは増えます。
- 物理的な索引ページ順序がそのページ上のキーの順序と一致しなくなった (これを、*不良クラスター* 索引という)。

  リーフ・ページが適切にクラスター化されないと、順次プリフェッチの効率が低下し、入出力の待ち時間が長くなります。
- 索引が、最も効率的なレベル数を超えて増大している。

  この場合、索引を再編成する必要があります。

索引を作成するときに MINPCTUSED パラメーターを設定する場合、キーが削除され、フリー・スペースが、指定されたパーセント未満になっていると、データベース・サーバーは自動的に索引リーフ・ページをマージします。このプロセスを、*オンライン索引デフラグ* といいます。ただし、索引クラスタリングをリストアし、スペースを解放し、そしてリーフ・レベルを下げるには、以下に挙げるいずれかの方法を取ってください。

- 索引をドロップした後再作成します。
- REORG INDEXES コマンドを使用してオンラインで索引を再編成します。

  この方法は、実稼働環境で選択できる場合があります。なぜなら、この方法では、索引の再ビルド中に表の読み取り/書き込みが行えるからです。
- 表の再編成およびオフラインでの索引の再編成の両方を可能にするオプションを指定して、 REORG TABLE コマンドを使用します。

オンライン索引再編成

ALLOW WRITE ACCESS オプションを指定して REORG INDEXES コマンドを使用すると、表への読み取り/書き込みアクセスが許可されている間、指定された表の索引すべてが再ビルドされます。再編成の進行中に、基礎表に加えられ、索引に影響を与える変更は、DB2 ログに記録されます。さらに、使用可能なメモリー・スペースがある場合は、その変更内容は内部メモリー・バッファー・スペースにも記録されます。再編成によって、ログに記録された変更は処理され、索引の再ビルド時に現行の書き込みアクティビティーにキャッチアップします。内部メモリー・バッファー・スペースは、ユーティリティー・ヒープからオンデマンドで割り振られる指定のメモリー域であり、作成中または再編成中の索引に加えられる変更を保管します。メモリー・バッファー・スペースの使用によって、索引の再編成は、まずメモリーから変更を直接読み取ることによって処理し、次いで必要であれば、もっと後にログ全体を読み取ることによって実行できます。割り振られたメモリーは、再

編成操作が完了すると解放されます。再編成の完了後には、再ビルドされた索引は完全にクラスター化されない場合があります。索引に対して PCTFREE が指定される場合、再編成時にそのスペースの比率が各ページに保存されます。

パーティション表では、オンラインでの索引再編成と索引の個別クリーンアップがサポートされています。索引を個別に再編成するには、索引名を次のように指定します。 `REORG INDEX index_name for TABLE table_name`

空間インデックスまたは多次元クラスタリング (MDC) 表では、ALLOW WRITE モードでのオンラインの索引再編成はサポートされていません。

注: REORG INDEXES/INDEX コマンドの CLEANUP ONLY オプションによって索引を完全に再編成することはできません。 CLEANUP ONLY ALL オプションは、削除済みとしてマークされ、コミット済みとして認識されるキーを除去します。また、このオプションは、ページ内のすべてのキーが削除済みとしてマークされ、コミット済みとして認識されるページの解放も行います。ページが解放されると、隣接するリーフ・ページがマージされます (ただし、これを行って、マージされたページに少なくとも PCTFREE フリー・スペースが残る場合)。 PCTFREE は、索引の作成時に定義されたフリー・スペースのパーセンテージです。 CLEANUP ONLY PAGES オプションは、ページ内のすべてのキーが削除済みとしてマークされ、コミット済みとして認識されるページだけを削除します。

パーティション表で CLEANUP ONLY オプションを使って索引を再編成するときは、すべてのアクセス・レベルがサポートされます。 CLEANUP ONLY オプションが指定されない場合は、デフォルトのアクセス・レベル ALLOW NO ACCESS だけがサポートされます。

索引の再編成には以下の要件があります。

- 索引および表に関する SYSADM、SYSMAINT、SYSCTRL、または DBADM 権限か、CONTROL 特権
- 索引の保管先表スペースのフリー・スペースの容量が、索引の現行サイズと等しい

  CREATE TABLE ステートメントを発行する際は、再編成対象の索引を LARGE 表スペースに置くようにしてください。
- 追加ログ・スペース

  索引の再編成では、そのアクティビティーをログに記録します。その結果、特にシステムがビジーであり、他の並行アクティビティーがログに記録される場合、再編成は失敗することがあります。

  注: ALLOW NO ACCESS オプションを指定した REORG INDEXES ALL が失敗する場合、索引は無効とマークされ、操作は未完了となります。ただし、ALLOW READ ACCESS オプションまたは ALLOW WRITE ACCESS を指定した REORG が失敗する場合、元の索引オブジェクトはリストアされます。

表および索引を再編成するタイミングの決定

表データに多くの変更を加えた後、論理的には連続しているデータが、順序どおりになっていない物理データ・ページ上に置かれる場合があります (特に、多くの更

新操作がオーバーフロー・レコードを作成した場合)。このようにデータが編成された場合、データベース・マネージャーは順次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。多数の行を削除した場合にも、追加の読み取り操作が必要になります。

表再編成は無駄なスペースを除去しながらデータをデフラグし、オーバーフロー・レコードを取り込むために行を再配列し、データ・アクセスおよび最終的には照会パフォーマンスを改善します。特定の索引に従ってデータを再配列するよう指定して、最小限のデータ読み取りで照会がデータにアクセスできるようにすることも可能です。

表データに対して多くの変更を加えると、索引が更新されることになり、索引のパフォーマンスが低下する可能性があります。索引リーフ・ページがフラグメント化されるかまたは適切にクラスター化されず、最適のパフォーマンスを得るために索引は必要なレベルより高いレベルを開発するおそれがあります。これらの問題はすべて余分な入出力を発生させ、これによってパフォーマンスが低下します。

以下の要因のいずれかが見られた場合、表または索引の再編成が必要であると判断できるかもしれません。

- 表が最後に再編成されて以来、大量の挿入、更新、および削除アクティビティーが表に対して実行された
- 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おける顕著な変化
- RUNSTATS を実行して統計情報をリフレッシュしてもパフォーマンスが向上しない
- REORGCHK コマンドが、表または索引の再編成の必要性を示している (注: 場合によっては、REORGCHK が常に表の再編成を推奨し、再編成を実行した後でさえそれを推奨することがあります)。例えば、32 kb のページ・サイズを使用していて、平均レコード長が 15 バイト、各ページの最大レコード数が 253 である場合、各ページには使用不可能なバイト数が 32700- (15 x 253)=28905 あることになります。この場合、ページの約 88% がフリー・スペースということになります。ユーザーは REORGCHK の推奨を分析して、再編成を実行するためのコストと益のバランスを取る必要があります
- db.tb_reorg_req (再編成の必要性) ヘルス・インディケーターが ATTENTION 状態である。このヘルス・インディケーターの収集の詳細には、再編成によって効果を得られる可能性のある表と索引のリストが表示されます。

REORGCHK コマンドはデータ編成に関する統計情報を戻すので、特定の表または索引で再編成が必要かどうかを判別するのに役立ちます。しかし、カタログ統計表に対する特定の照会を、決まったインターバルまたは特定の回数実行することによって提供されるパフォーマンス履歴を活用すれば、パフォーマンスを大幅に向上させる可能性のある傾向を見つけることができます。

表または索引の再編成が必要かどうかを判別するには、カタログ統計表を照会して、以下の統計をモニターします。

1. 行のオーバーフロー

   SYSSTAT.TABLES ビューの OVERFLOW 列を照会して、オーバーフロー値をモニターします。この列内の値は、元のページに収まらない行の数を示していま

す。表の中の可変長列がデータ・ページ上の割り当てられた場所に収まらなくなるほど、その可変長列が原因でレコード長が拡張すると、行データはオーバーフローします。列が表定義に追加され、後で行の更新によってその列がマテリアライズされる場合、長さの変更も生じます。この場合、行の元の位置にはポインターが保持され、実際の値はポインターが示す別の場所に保管されます。これはパフォーマンスを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ためです。この 2 つのステップから成るプロセスにより、処理時間が長くなり、必要な入出力の回数も多くなる可能性があります。

表データを再編成すると行のオーバーフローを除去できるので、オーバーフロー行の数が増えるにつれて、表データの再編成が効果的である可能性が高くなります。

2. 統計のフェッチ

   SYSCAT.INDEXES および SYSSTAT.INDEXES カタログ統計表にある、以下の 3 つの列を照会して、索引の順序で表にアクセスする場合のプリフェッチの効果性について判別します。これらの統計では、基本表に対するプリフェッチャーの平均パフォーマンスが示されています。

   - AVERAGE_SEQUENCE_FETCH_PAGES 列には、表に順序どおりにアクセスできるページの平均数が保管されています。順序どおりにアクセスできるページは、プリフェッチが可能です。この値が小さいということは、プリフェッチャーが本来の効果性を発揮できないということです。これは、表スペースの PREFETCHSIZE 設定に指定されたページ数全体を読み取ることができないためです。数が大きい場合、プリフェッチャーの実行は効果的です。クラスター索引および表の場合、この値は NPAGES の値 (行が入っているページの数) に近づくはずです。
   - AVERAGE_RANDOM_FETCH_PAGES 列は、索引を使用して表の行をフェッチするときの、順次ページ・アクセスの間のランダム表ページの平均数を保管します。ほとんどのページが順次の場合、プリフェッチャーは数の少ないランダムのページは無視して、構成されたプリフェッチ・サイズまでプリフェッチを継続します。表のまとまりがなくなるにつれ、ランダム・フェッチ・ページ数は増加します。通常このようにまとまりがなくなるのは、表の最後かオーバーフロー・ページのいずれかで、適切でない順序で挿入が行われることにより発生します。これによって行われるフェッチでは、索引を使用して何らかの範囲の値にアクセスする際に、照会パフォーマンスが低下します。
   - AVERAGE_SEQUENCE_FETCH_GAP 列は、索引を使用してフェッチする際の、表ページ・シーケンス間の平均ギャップを保管します。各ギャップは索引リーフ・ページのスキャンにより検出され、一連の表ページの間でランダムにフェッチしなければならない表ページの平均数を表します。これらは、多くのページがランダムにアクセスされて、プリフェッチャーに割り込みが行われたときに発生します。数が大きい場合、表はまとまりがなく、索引に対するクラスター化の多重度が低いことを示しています。

3. 削除対象としてマークが付けられているものの、除去されていない RID を含む、索引リーフ・ページの数

   通常、タイプ 2 では、 RID に削除対象としてマークが付けられた時点で、その RID が物理的に削除されることはありません。これは、有用なスペースがそれら

の論理的には削除済みの RID で占められる可能性があるということです。すべての RID が削除済みとしてマークされているリーフ・ページ数を検索するには、 SYSCAT.INDEXES および SYSSTAT.INDEXES 統計表の NUM_EMPTY_LEAFS 列を照会します。一部の RID に、削除対象としてマークが付けられているリーフ・ページに関しては、論理的には削除されている RID の合計数が NUMRIDS_DELETED 列に保管されます。

この情報を使用して、 CLEANUP ALL オプションを指定して REORG INDEXES を実行することにより、再利用できるスペースの量の見積もりを行ってください。 RID に削除対象のマークが付けられている、ページ内のスペースだけを再利用するには、 CLEANUP ONLY PAGES オプションを指定して REORG INDEXES を実行します。

4. 索引のクラスター率およびクラスター係数の統計

   クラスター率統計は、 SYSTCAT.INDEXES カタログ表の CLUSTERRATIO 列に保管されます。この値 (0 から 100 まで) は、索引のデータ・クラスタリングの程度を表わします。 DETAILED 索引統計を収集する場合は、代わりにより細かいクラスター率統計 (0 から 1) が CLUSTERFACTOR 列に保管され、CLUSTERRATIO の値は -1 になります。これらのクラスタリング統計のいずれか 1 つだけを、 SYSCAT.INDEXES カタログ表に記録できます。CLUSTERFACTOR 値と CLUSTERRATIO 値を比較するには、CLUSTERFACTOR に 100 を乗算してパーセンテージを得ます。

   注: 一般に、高度なクラスタリングが可能なのは、1 つの表の中で 1 つの索引だけです。

   索引専用アクセスではない索引スキャンは、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、この種のスキャンでは各データ・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。

   特定の索引で表データのクラスタリングが最初に行われ、クラスタリングの統計情報から、この索引に関するデータのクラスタリングが不十分であることがわかった場合、表を再編成して、データのクラスタリングを再び行うこともできます。

5. リーフ・ページの数

   索引が使用するリーフ・ページの数を調べるには、SYSCAT.INDEXES 表の NLEAF 列を照会します。その数から、索引の完全なスキャンに必要な索引ページ入出力の回数が分かります。

   理想的なのは、索引の占めるスペースが可能な限り最小で、索引スキャンに必要な I/O が少なくなることです。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、索引のサイズが大きくなることがあります。 表の再編成中に索引を再作成すると、各索引を最小のスペースで作成することができます。

   注: デフォルトでは、索引の作成を行う場合、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。フリー・スペースの量を増やすには、索引の作成

時に PCTFREE パラメーターを指定します。索引を再編成するたびに PCTFREE 値が使用されます。フリー・スペースが 10% を超えている場合、追加のスペースによって追加の索引の挿入に対応するので、索引の再編成の頻度が減ります。

6. 空のデータ・ページの数

   表の空ページの数を計算するには、 SYSCAT.TABLES の FPAGES および NPAGES 列を照会し、 FPAGES の数から NPAGES の数を引きます。 FPAGES 列には、使用中のページの合計数が保管されています。 NPAGES 列には、行を含んでいるページの数が保管されています。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、空ページをレクラメーションして、表に使用されるスペースの量を圧縮できます。表スキャンで空ページがバッファー・プール中に読み取られるので、未使用ページの再利用により、表スキャンのパフォーマンスが向上します。

   表内のページの総数 (FPAGES) が NPARTITIONS *1 エクステント・サイズ以下の場合、表の再編成は推奨しません。 NPARTITIONS は、これがパーティション表である場合はデータ・パーティション数、そうでない場合は 1 です。パーティション・データベース環境では、表のデータベース・パーティション・グループ内のデータベース・パーティションの数を計算に含めた後、条件は、FPAGES <= 表のデータベース・パーティション・グループ内のデータベース・パーティションの数 * NPARTITIONS * 1 エクステント・サイズに変更されます。

再編成をする前に、照会パフォーマンスの低下を向上させた場合のコストと、表または索引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キング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) の間のトレードオフを考慮してください。

表および索引の再編成のコスト

表または索引に対して再編成を実行することによって、ある程度のオーバーヘッドが発生するのは避けられないため、オブジェクトの再編成を行うかどうかを決める時点でこれらを考慮する必要があります。

表および索引の再編成のコストには、以下のものが含まれます。

- ユーティリティーを実行する際の CPU 使用量
- REORG ユーティリティーの実行中の並行性の低下。REORG のロック要件のため、並行性が低下します。
- 追加のストレージ要件。(オフライン表再編成を行うためには、表のシャドー・コピーを保持するためのストレージ・スペースが別途必要になります。オンラインまたはインプレース表再編成では、追加のロギング・スペースが必要になります。オンライン索引再編成を行うためには、索引および追加のログ・スペースのシャドー・コピーを保持するためのストレージ・スペースが別途必要になります。オフライン索引再編成はログ・スペースをあまり使用しないので、シャドー・コピーを必要としません。

場合によっては、再編成された表は元の表よりも大きくなり、それに応じてスペース所要量も増加します。以下の状況では、再編成後に表が大きくなる可能性があります。

- 行の順序を決定するために索引が使用されているクラスタリング REORG TABLE では、表レコードが可変長の場合 (例えば Varchar を使用している場合)、あるページに含まれる行数が元の表よりも少なくなることがあり、そのためにより多くのスペースを使用することになる可能性があります。
- 表が作成された後、再編成前に追加された列が表にある場合、追加の列が再編成後に初めていくつかの行において認識される可能性があります。
- 各ページに残されたフリー・スペースの量 (PCTFREE 属性の値によって示される) が、最後の再編成時よりも増加した場合。
- 表に LOB がある場合、LOB が以前より多くのスペースを使用する可能性があります。

オフライン表再編成のスペース所要量

オフライン再編成はシャドー・コピー方式をとるので、表のもう 1 つのコピーを収容するのに十分な追加のストレージが必要です。シャドー・コピーは元の表が常駐する表スペースか、ユーザーが指定した TEMPORARY 表スペースのいずれかに作成されます。

表スキャン・ソートが使用される場合は、ソート処理に追加の TEMPORARY 表スペース・ストレージが必要になることがあります。必要な追加スペースが再編成される表のサイズと同じ大きさになる場合もあります。クラスタリング索引が SMS タイプまたはユニーク DMS タイプの場合、この索引の再作成ではソートは必要ありません。むしろ、この索引は新しく再編成されたデータをスキャンすることによって再作成されます。再作成を必要とするほかの索引にはソートが必要であり、最大で再編成される表のサイズの TEMPORARY 表スペースが必要となる可能性があります。

オフライン表 REORG では、制御ログ・レコードはほとんど生成されないので、消費されるログ・スペース量はそれほど大きくありません。REORG で索引が使用されない場合、ログ・レコードは表データ・ログ・レコードのみとなります。索引が指定された場合、またはクラスタリング索引が表に存在する場合、行の RID は、新しいバージョンの表に配置された順にログに記録されます。RID の各ログ・レコードは最大 8000 の RID (各 RID は 4 バイトを消費) を保持します。これがオフライン表再編成中にログ・スペースが使い尽くされてしまう要因になることがあります。RID はデータベースがリカバリー可能 (LOGRETAIN=ON) の場合にのみログに記録されることに注意してください。

オンライン表再編成のログ・スペース所要量

オンライン表 REORG に必要なログ・スペースは、オフライン表 REORG に必要な量より大きいのが普通です。スペース所要量は、再編成される行の数、索引の数、索引キーのサイズ、および表の最初の編成状態がどの程度悪かったかによって決まります。表のログ・スペース使用率についての標準のベンチマークを設定することをお勧めします。

表の各行は、オンライン表再編成中に 2 回移動される可能性があります。索引が 1 つある場合、各行は新しいロケーションを反映するよう索引キーを更新し、ひとたび古いロケーションへのすべてのアクセスが完了すると、キーが再び更新されて古い RID の参照が除去されます。行が戻されたとき、これらの索引キーへの更新が再び実行されます。これらすべてのアクティビティーは、オンライン表再編成を完全にリカバリーできるようにするためにログに記録されるので、少なくとも 2 つのデータ・ログ・レコード (各インスタンスに行データが含まれる) と 4 つの索引ログ・レコード (各インスタンスにキー・データが含まれる) が存在することになります。特にクラスタリング索引は索引ページを埋め尽くして、索引の分割やマージを引き起こす傾向がありますが、これについてもログに記録される必要があります。

オンライン表再編成は頻繁に内部コミットを発行するため、通常はそれほど多くのアクティブ・ログを保持しません。オンライン REORG が多数のアクティブ・ログを保持するとすれば、それは切り捨てフェーズ中です。切り捨てフェーズでは S 表ロックを取得するためです。表再編成はロックを取得できない場合、ログを保持したまま待機するので、他のトランザクションが素早くそのログを埋め尽くしてしまう可能性があります。

表および索引の再編成の必要性を少なくする

さまざまな対策を講じることによって、表や索引を再編成する必要が生じる頻度を減らし、不必要な再編成操作の実行にかかるコストをなくすことができます。

表の再編成の必要性を少なくする

- 複数パーティション表を使用します。通常、表が小さければ小さいほど、再編成を行う必要が生じる可能性は小さくなります。
- マルチディメンション・クラスタリング (MDC) 表を作成します。このクラスタリングは CREATE TABLE ステートメントの ORGANIZE BY DIMENSION 節で指定された列で自動的に保守されます。
- 表の APPEND モードをオンにします。これらの新しい行の索引キー値が、たとえば常に高いキー値である場合は、表のクラスタリング属性はそのキー値を表の最後に置こうとします。この場合、表を付加モードにすることは、クラスタリング索引よりもよい方法です。
- 表を作成した後
  - 表を変更して PCTFREE を追加します。
  - 索引に対する PCTFREE を指定してクラスタリング索引を作成します。
  - 表にロードする前にデータをソートします。

  表の PCTFREE が正しく設定されたクラスタリング索引は、元のソート順序を維持するのに役立ちます。表ページに十分なスペースがある場合、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、索引のクラスタリング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、上のクラスター特性は徐々に失われます。

  クラスタリング索引を作成した後に REORG TABLE またはソートを実行し、LOAD を実行すると、その索引はデータの特定の順序を保持しようするので、RUNSTATS ユーティリティーによって収集された CLUSTERRATIO または CLUSTERFACTOR 統計は改善されます。

索引の再編成の必要性を少なくする

- 索引ページに対する PCTFREE または LEVEL2 PCTFREE を指定してクラスター索引を作成する。範囲は 0 から 99% で、デフォルト値は 10% です。
- MINPCTUSED を使って索引を作成します。範囲は 0 から 99% まで指定可能ですが、推奨値は 50% です。しかし、代わりに REORG INDEXES コマンドの CLEANUP ONLY ALL オプションを使用して、リーフ・ページをマージすることも考慮してください。

自動再編成

表データに多くの変更を加えた後、表および索引をフラグメント化できるようになります。論理的には連続しているデータは、連続していない物理ページ上に格納される可能性があります。その場合、データベース・マネージャーは、データにアクセスするために余分の読み取り操作を実行することが必要になります。

情報の中でも RUNSTATS によって収集される統計情報には、表の中のデータ分布が示されます。特に、それらの統計情報を分析すると、再編成が必要になるタイミングや必要な再編成の種類が示されていることがあります。自動再編成は、REORGCHK のさまざまな公式を使用することによって、表および索引の再編成が必要かどうかを判定します。再編成が必要かどうかを調べるため、統計情報の更新の原因となった表および索引が定期的に評価されます。再編成が必要なら、索引の再編成または表の古典的な再編成が内部でスケジュールに入れられます。そのためには、再編成する表に対する書き込みアクセス権なしでアプリケーションが機能しなければなりません。

自動再編成フィーチャーを有効または無効にするには、AUTO_REORG、AUTO_TBL_MAINT、および AUTO_MAINT のデータベース構成パラメーターを使用します。

パーティション・データベース環境の場合、自動再編成を実行するかどうかの決定や自動再編成の開始は、カタログ・パーティション上でなされます。データベース構成パラメーターを有効にすることが必要なのは、カタログ・パーティションにおいてだけです。再編成は、ターゲット表の存在しているすべてのデータベース・パーティション上で実行されます。

表および索引を再編成するタイミングと方法がわからない場合は、自動再編成をデータベース保守プランの一部に含めることができます。

自動再編成を考慮した表および索引は、コントロール・センターまたはヘルス・センターから自動保守ウィザードを使用することによって構成できます。

表および索引の自動再編成を有効にする

よく編成された表および索引データを得ることは、効率的なデータ・アクセスと最適のワークロード・パフォーマンスにとって重要です。表データに多くの変更を加えた後、論理上順次データは、順序どおりになっていない物理データ・ページ上にある場合があるので、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。多数の行を削除した場合にも、追加の読み取り操作が必要になります。データの再編成の時期と方法を気にせずにすむ

よう、表の自動再編成を使用して、DB2 でオフラインの表と索引の再編成の管理を有効にします。DB2 を有効にして、データベース表だけでなく、システム・カタログ表も再編成することができます。

この機能は、グラフィカル・ユーザー・インターフェース・ツールか、コマンド行インターフェースのいずれかを使用してオンにすることができます。

- グラフィカル・ユーザー・インターフェース・ツールを使って、データベースを自動再編成するようセットアップするには、次のようにします。
  1. コントロール・センターでデータベース・オブジェクトを右クリックするか、ヘルス・センターで自動再編成の対象として構成するデータベース・インスタンスを右クリックすることにより、「自動保守の構成」ウィザードを開きます。ポップアップ・ウィンドウで「**自動保守の構成**」を選択してください。
  2. このウィザードで、自動再編成を使用可能にし、REORG ユーティリティーを実行するメンテナンス時間帯を指定することができます。
- コマンド行インターフェースを使って、データベースを自動再編成するようにセットアップするには、次の構成パラメーターをそれぞれ「ON」に設定します。
  - AUTO_MAINT
  - AUTO_TBL_MAINT
  - AUTO_REORG

パフォーマンスを向上させるためのリレーショナル索引の使用

索引を使うと、表データへアクセスするときのパフォーマンスを向上させることができます。リレーショナル索引はリレーショナル・データの作業時に使われます。XML データのアクセスには、XML データに対する索引が使われます。

オプティマイザーはリレーショナル表データにアクセスするためにリレーショナル索引を使用するかどうかを決定しますが (次に挙げる場合を除き)、ユーザーはどの索引がパフォーマンスを向上させるかを判別し、それらの索引を作成しなければなりません。例外は、マルチディメンション・クラスタリング (MDC) 表の作成時に指定する各ディメンションに対して自動的に作成される、ディメンション・ブロック索引および複合ブロック索引です。

また、次のような場合には RUNSTATS ユーティリティーを実行して、リレーショナル索引に関する新規の統計を収集する必要もあります。

- リレーショナル索引を作成した後
- プリフェッチ・サイズを変更した後

また、RUNSTATS ユーティリティーは、定期的なインターバルで実行し、統計を現行のものに保持する必要があります。索引についての最新の統計がないと、オプティマイザーは照会に対する最適のデータ・アクセス・プランを判別することができません。

注: 特定のパッケージでリレーショナル索引を使用するかどうかを決めるには、Explain 機能を使用します。リレーショナル索引を計画するには、コントロール・センターから設計アドバイザーを使用するか `db2advis` ツールを使用して、 1 つ以上の SQL ステートメントによって使用される可能性のあるリレーショナル索引についてのアドバイスを入手してください。

リレーショナル索引を使用しない場合と比べての索引の利点

表に索引が存在しない場合、 SQL 照会において参照される各表ごとに、表スキャンを実行しなければなりません。表スキャンでは、各表行が順次アクセスされる必要があるため、表が大きいほど表スキャンにかかる時間も長くなります。表スキャンは、表の中のほとんどの行を必要とする複雑な照会に効果的です。他方、索引スキャンは、表の行の一部だけを戻す照会で、表の行に効果的にアクセスすることができます。

オプティマイザーは、索引列が SELECT ステートメント内で参照され、表スキャンより索引スキャンのほうが速いと見積もった場合、リレーショナル索引スキャンを選択します。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、索引全体をスキャン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。

出力の配列要件が索引の列と一致する場合、列の順序で索引をスキャンすることにより、ソートをしなくても正しい順序で行を検索できます。

各リレーショナル索引項目は、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。 CREATE INDEX ステートメントの ALLOW REVERSE SCANS パラメーターを指定する場合、その値は昇順と降順の両方で検索することができます。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。リレーショナル索引を使用して行を配列順に取得することにより、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ソートする必要がなくなります。

検索キーの値および行ポインターに加えて、リレーショナル索引には組み込み列を含めることができます。これらの列は、索引付けされた行内の索引付けされていない列です。そのような列があると、オプティマイザーは表そのものにアクセスしなくても必要な情報を索引のみから入手することができます。

注: 照会されている表にリレーショナル索引が存在していても、順序付けられた結果セットが得られるとは限りません。 ORDER BY 節を使用することによってのみ、結果セットの順序付けが確実になります。

索引はアクセス時間を大幅に短縮することができますが、同時に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。

- 各索引には、ストレージおよびディスク・スペースが必要になります。正確な量は、表のサイズとリレーショナル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、索引キーの値を変更する 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リレーショナル索引が再作成されたり、既存のリレーショナル索引に索引が追加されます。

リレーショナル索引が作成された時に使用された索引 PCTFREE をオーバーライドするには、 LOAD コマンドに indexfreespace MODIFIED BY パラメーターを指定します。

- 各リレーショナル索引は、オプティマイザーが考慮する SQL 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、その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してください。

リレーショナル索引の計画のヒント

作成するリレーショナル索引は、リレーショナル・データおよびそれにアクセスする照会に基づいたものにする必要があります。

コントロール・センターから設計アドバイザーを使用するか、 db2advis ツールを使用して、特定の照会またはワークロードを定義する照会の集合に最適な索引を調べてください。このツールは、INCLUDE 列などのパフォーマンス向上させるフィーチャーのあるリレーショナル索引、継承されたユニーク索引、および ALLOW REVERSE SCANS 索引を推奨します。

さまざまな目的のリレーショナル索引を作成する方法を判別するのに、以下の指針が役立つでしょう。

- あるソートを回避するには、可能であれば CREATE UNIQUE INDEX ソートを使用して、主キーおよびユニーク・キーを定義します。
- データ検索を向上させるには、 INCLUDE 列をユニーク索引に追加します。次のような列の場合に、追加するとよいでしょう。
  - 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検索効率が向上する場合
  - 索引スキャン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
- 小さな表に効率的にアクセスするには、リレーショナル索引を使用して、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・ビューの 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。
- 効率的に検索するには、キーの配列を昇順にするか降順にするかを、最も頻繁に使用される順序に基づいて決定します。 CREATE INDEX ステートメントの ALLOW REVERSE SCANS パラメーターを指定すれば逆方向にも値を検索できますが、指定された索引の順序のスキャンのほうが、逆方向のスキャンよりも少し良くなります。
- 索引の保守コストおよびスペースを少なくするには、以下のようにします。
  - 列上の他の索引キーの部分キーとなるようなリレーショナル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。

– 根拠もなくすべての列にリレーショナル索引を作成しない。必要のない索引はスペースを使用するだけでなく、準備時間を増やす原因ともなります。このことは、複合照会、ダイナミック・プログラミング結合列挙による最適化クラスを使用する場合、特に重要です。

  表に対して定義するリレーショナル索引の数に関する、以下の一般規則を使用してください。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 つの索引だけを作成してください。
  - 読み取り専用照会環境では、 5 を超える索引を作成することも可能です。
  - 混合照会および OLTP 環境では、 2 から 5 の索引を作成できます。

- 親表での削除および更新操作のパフォーマンスを向上させるには、外部キーについてリレーショナル索引を作成します。
- IMMEDIATE MQT と INCREMENTAL MQT が関係する DELETE と UPDATE 操作のパフォーマンスを向上させるには、 MQT の暗黙のユニーク・キー (MQT の定義の GROUP BY 節内の列) について固有のリレーショナル索引を作成します。
- ソート操作を速くするには、リレーショナル・データのソート時に頻繁に使用される列についてリレーショナル索引を作成します。
- 複数列リレーショナル索引で結合のパフォーマンスを向上させるには、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列か、明確な値が最も多い列を最初のキーとして使用します。
- 新しく挿入された行がその索引に応じてクラスタリングされ続けるようにし、またページ分割を避けるには、クラスタリング索引を定義します。クラスタリング索引は、表を再編成する必要性を大幅に少なくします。

  表を定義するときには PCTFREE キーワードを使用して、ページに適切に挿入ができるようにどれだけのフリー・スペースをページに残すかを指定してください。 LOAD コマンドの pagefreespace MODIFIED BY 節を指定することもできます。
- オンライン索引デフラグを使用可能にするには、リレーショナル索引の作成時に MINPCTUSED オプションを使用します。 MINPCTUSED は、索引リーフ・ページ上で使用される最小スペースの限界値を指定し、オンライン索引デフラグを使用可能にします。これにより、索引ページから物理的にキーを除去する削除の場合に、パフォーマンスを低下させる再編成の必要性を低くできる可能性があります。

以下の場合に、リレーショナル索引の作成を考慮してください。

- 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 節の中で使用される列には、リレーショナル索引を作成してください。

  次のような WHERE 節の場合、

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

  一般に、WORKDEPT 列が多くの重複値を含んでいる場合を除き、 WORKDEPT に対する索引があれば便利です。

- 照会によって要求される順序で行を配列するには、 1 つ以上の列にリレーショナル索引を作成してください。 ORDER BY 節においてだけでなく、DISTINCT 節や GROUP BY 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 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 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列の名前を指定する複合キーを持つリレーショナル索引を作成します。この方法で索引を作成すると、リレーショナル・データを索引のみから検索できるようになり、表にアクセスするよりも効率的です。

  たとえば、次の SQL ステートメントを考察してみましょう。

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

  EMPLOYEE 表の WORKDEPT 列と LASTNAME 列にリレーショナル索引が定義されている場合、表全体をスキャンするよりも索引をスキャンする方が、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、この列はリレーショナル索引の最初の列でなければなりません。

- INCLUDE 列についてリレーショナル索引を作成すると、表での索引の使用が改善されます。前の例を使用し、次のようにして固有のリレーショナル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 が索引のリーフ・ページでのみ保管されるということを意味します。

リレーショナル索引のパフォーマンスのヒント

リレーショナル索引の使用および管理について、以下の提案を考慮してください。

- **大規模なユーティリティー・ヒープを指定する**

  他のユーザーまたはアプリケーションからの基礎表への書き込みアクセスは、CREATE INDEX および REORG INDEXES の両方でサポートされています。リレーショナル索引用の基礎表に多くの更新アクティビティーが作成または再編成されると予想される場合には、大規模なユーティリティー・ヒープを構成することを検討してください。大規模なユーティリティー・ヒープは、キャッチアップ・フェーズでの索引作成または索引再編成を高速化します。作成中または再編成中の索引に対するすべての書き込みアクティビティーは、 DB2 ログおよび内部メモリー・バッファー・スペースに記録されます。内部メモリー・バッファー・スペースは、ユーティリティー・ヒープからオンデマンドで割り振られる指定のメモリー域であり、作成中または再編成中の索引に加えられる変更を保管し

ます。このメモリーの使用によって、キャッチアップ・フェーズを高速化できます。割り振られたメモリーは、作成または再編成操作が完了すると解放されます。作成中または再編成中の索引に加えられる変更のすべてまたは大半を入れられる十分なユーティリティー・ヒープがあるようにすれば、キャッチアップ・フェーズでのパフォーマンスにはかなりの向上がみられる可能性があります。

- **SMP マシンで実行している場合は、sheapthres 構成パラメーターを大きくする**

  各サブエージェントは、**sortheap** 構成パラメーターで指定されたメモリー量を取得します。そのようにして、ソート・オーバーフローを回避するために表をスキャンします。ソート・オーバーフローの数をモニターし、それに応じて **sheapthres** を大きくする必要があります。

- **リレーショナル索引には別々の表スペースを指定する**

  索引は、表データ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、索引アクセス時の読取/書込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。また、より高速の物理装置に索引表スペースを作成することもできます。さらに、索引表スペースを別々のバッファー・プールに割り当てることもできます。これにより、索引ページは表データ・ページと競合することがないため、バッファーに長く保持される可能性があります。

  索引を別々の表スペースに置かない場合、データと索引ページの両方が同じエクステント・サイズとプリフェッチ数量を使用します。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対して異なる値を選択することができます。索引は一般に表より小さく、分散しているコンテナーも少ないため、8 または 16 ページといったエクステント・サイズになるのが通常です。照会オプティマイザーは、アクセス・プランを選択するときに、表スペースに対する装置の速度を考慮します。

- **クラスタリングの多重度を確認する**

  SQL ステートメントが 順序付けを必要としているとき (たとえば、ORDER BY、GROUP BY、および DISTINCT SQL 節を含む場合など) に、たとえ索引がその順序付けを満たしているとしても、以下の場合にはオプティマイザーが索引を選択しない可能性があります。

  – 索引クラスタリングの程度が低い場合。この情報については、SYSCAT.INDEXES の CLUSTERRATIO 列および CLUSTERFACTOR 列を調べてください。
  – 表が小さいので、表のスキャンや応答セットのソート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

  クラスタリング索引を作成した後、 REORG TABLE をクラシック・モードで実行し、完全に編成された索引を作成するようにします。表を再クラスター化するには、代わりにソートおよび LOAD を実行することもできます。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。クラスタリング索引を作成した後で、追加の索引を作成してください。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、クラスタリング索引はデータの特定順序を維持しようと試みます。

クラスタリング率を維持するためには、ロードまたは再編成を行う前に、表の変更時において適切な PCTFREE を指定してください。 PCTFREE によって指定される各ページのフリー・スペースによって、挿入のためのスペースができ、これらの挿入を適切にクラスター化することができます。表に PCTFREE を指定しない場合、再編成によってすべての余分のスペースが除去されます。

注: クラスタリングは、レンジ・クラスター表を使用しない限り、現状では更新の間維持されません。つまり、クラスタリング索引のキー値が変更されるようにしてレコードを更新する場合、レコードはクラスタリング順序を維持するために新規ページに必ずしも移動されるとは限りません。クラスタリングを維持するために、UPDATE の代わりに、 DELETE とそれから INSERT を使用してください。

- **表および索引の統計を最新のものにしておく**

  新規リレーショナル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オプティマイザーが判断することが可能になります。

- **オンライン索引デフラグを使用可能にする**

  オンライン索引デフラグは、リレーショナル索引に対して、MINPCTUSED 節がゼロより大きく設定されている場合に使用可能です。オンライン索引デフラグを使用すると、あるページのフリー・スペースが指定されたレベルと同じかそれより下回った場合にリーフ・ページをマージすることによって、索引を圧縮することができます。その間も索引は使用可能なままです。

- **必要に応じてリレーショナル索引を再編成する**

  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。表に対する更新によって索引ページのプリフェッチの効率が悪くなる可能性があるからです。

  索引を再編成するには、索引をドロップするか、再作成するか、または REORG ユーティリティーを使用してください。

  頻繁に再編成を行う必要を少なくするためには、リレーショナル索引の作成時に適切な PCTFREE を指定して、各索引のリーフ・ページに作成時と同じパーセンテージのフリー・スペースを残しておくようにします。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。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。そのために、索引ページのプリフェッチの効率が低下します。

  注: リレーショナル索引の作成時に指定する PCTFREE は、索引が再編成されるときにも保持されます。

  リレーショナル索引をドロップして再作成する、またはリレーショナル索引を再編成すると、ほぼ隣接し連続した一連の新しいページも作成され、索引ページのプリフェッチが向上します。時間とリソースの面ではコストがかかりますが、REORG TABLE ユーティリティーでも確実にデータ・ページをクラスタリングすることができます。このようにしてクラスタリングを行うと、大量のデータ・ページにアクセスする索引スキャン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) 環境では、「クラシック」REORG TABLE モード (シャドー表を使用して表の再編成を高速で行う)で、複数のプロセッサーを使用して索引の再作成ができます。

- **リレーショナル索引使用に関する EXPLAIN 情報を分析する**

  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、それぞれのリレーショナル索引が最低でも 1 回は使用されているかどうかを検査す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、その索引のドロップを検討する必要があります。

  また、EXPLAIN 情報を使用すると、大きな表の表スキャンがネスト・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。そのように処理されている場合は、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見なされているかのいずれかです。

- **サイズが大きく異なる表には Volatile 表を使用する**

  VOLATILE 表は、実行時のサイズが空から非常に大きなものまで変化する可能性がある表です。カーディナリティーが大きく異なるこの種類の表に対しては、オプティマイザーは索引スキャンの代わりに表スキャンを行うアクセス・プランを生成する可能性があります。

  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、オプティマイザーは Volatile 表に対して索引スキャンを使用できるようになります。オプティマイザーは、以下の状況での統計に関係なく、表スキャンの代わりに索引スキャンを使用します。

  – 参照される列がすべて索引内にある場合。
  – 索引が索引スキャンで述部を適用できる場合。

  表が型付き表である場合、型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。

索引のクリーンアップおよび保守

索引の作成後、索引をコンパクトに編成しなければ、パフォーマンスが低下してしまいます。以下の提案を参考にして、索引を可能な限り小さく、効率的にしてください。

- オンライン索引デフラグを使用可能にする

  MINPCTUSED 節を使って索引を作成します。必要に応じて、既存の索引をドロップして再作成してください。

- COMMIT を頻繁に実行します。 COMMIT の頻繁な実行が不可能な場合は、明示的に、またはロック・エスカレーションによって表に対する X ロックを取得します。

  削除済みとマークされた索引キーは、 COMMIT の後に表から物理的に除去されます。表に対する X ロックを使用すると、以下に説明するように、キーが削除済みとマークされた時点で物理的に除去されます。

- REORGCHK を使用すると、索引や表 (またはその両方) をいつ再編成すべきか、また CLEANUP ONLY オプションを使っていつ REORG INDEXES を実行すべきかを判別するのに役立ちます。

  再編成中に索引への読み取りおよび書き込みアクセスを可能にするには、ALLOW WRITE ACCESS オプションを使って REORG INDEXES を実行します。

注: DB2 バージョン 8.1 以降では、すべての新しい索引はタイプ 2 索引として作成されます。唯一の例外は、すでにタイプ 1 索引が存在する表に索引を追加する場合です。この場合、新しい索引もまたタイプ 1 索引になります。表にどのタイプの索引が存在するかを判別するには、 INSPECT コマンドを実行してください。タイプ 1 索引をタイプ 2 索引に変換するには、 REORG INDEXES コマンドを実行します。

タイプ 2 索引には、主に次のような利点があります。

- 長さが 255 バイトを超える列に対して、索引を作成することができます。
- 次キー・ロッキングの使用が最小限に抑えられて、並行性が改善されます。キーは索引ページから物理的に除去されるのではなく削除済みとマークされるので、ほとんどの次キー・ロッキングは除去されます。キー・ロッキングについての詳細は、ロッキングがパフォーマンスに与える影響に関するトピックを参照してください。

以下の状況では、削除済みとマークされた索引キーがクリーンアップされます。

- その後の挿入、更新、または削除アクティビティー中

  キーの挿入中、クリーンアップすればページ分割の必要がなくなり、索引サイズの増加を防げるような場合には、削除済みとマークされてコミット済みと認識されたキーがクリーンアップされます。

  キーの削除中、ページ上のすべてのキーが削除済みとマークされた場合には、すべてのキーが削除済みとマークされ、それらの削除がすべてコミット済みになった別の索引ページを見つけるよう試行されます。そのようなページが見つかると、索引ツリーから削除されます。

  キーを削除するとき、表に対する X ロックが存在すれば、キーは単に削除済みとマークされる代わりに、物理的に削除されます。この物理的削除の際、削除済みとマークされてコミット済みと認識されたキーが同じページ上に存在すれば、それらもすべて除去されます。
- CLEANUP オプションを使って REORG INDEXES コマンドを実行するとき

  CLEANUP ONLY PAGES オプションを指定すると、すべてのキーが削除済みとマークされてコミット済みと認識されている索引ページが検索されて、解放されます。

  CLEANUP ONLY ALL オプションを指定すると、すべてのキーが削除済みとマークされてコミット済みと認識されている索引ページが解放されるだけでなく、未削除の RID を含むページにおいて、削除済みとマークされてコミット済みと認識されている RID もまた解放されます。

さらにこのオプションを指定すると、隣接するリーフ・ページをマージすれば最低でも PCTFREE のフリー・スペースがマージ後のページに確保されるような場合、リーフ・ページをマージするよう試行されます。 PCTFREE 値は、索引の作成時に定義されるフリー・スペースのパーセントです。デフォルトの PCTFREE は 10 % です。 2 つのページをマージできる場合、いずれかのページが解放されます。

パーティション化された表の場合、デタッチされたデータ・パーティションが存在している状態で正確な索引の統計を生成するため、非同期索引のクリーンアップが完了した後に RUNSTATS を実行する必要があります。表にデタッチされたデータ・パーティションが存在しているか判別するには、SYSDATAPARTITIONS 表の状況フィールドで「I」 (索引クリーンアップ) または「D」 (依存 MQT からデタッチ) の値を確認します。

- 索引の再作成時

  索引を再作成するユーティリティーには、以下のものがあります。

  – REORG INDEXES (いずれの CLEANUP オプションも使用しない場合)
  – REORG TABLE (INPLACE オプションを使用しない場合)
  – IMPORT (REPLACE オプションを使用する場合)
  – LOAD (INDEXING MODE REBUILD オプションを使用する場合)

パーティション表での索引の動作について

パーティション表の索引は通常の表の索引と似ていて、各索引には表のすべてのデータ・パーティション内の行に対するポインターが含まれています。しかし重要な 1 つの相違点は、パーティション表の各索引は独立したオブジェクトであるということです。パーティション化されたデータベース環境では、表と同じ方法で索引はデータベース・パーティションに分散されます。パーティション表の索引は他の索引に対して独立して動作できるので、パーティション表に索引を作成する際には、どの表スペースを使用するかを特に考慮する必要があります。

パーティション表上の索引は、表のデータ・パーティションが複数の表スペースにまたがっている場合であっても、単一の表スペースに作成されます。DMS 表スペースと SMS 表スペースは両方とも、表とは別の場所にある索引の使用をサポートしています。指定されたすべての表スペースは、同一のデータベース・パーティション・グループになければなりません。各索引は、LARGE 表スペースを含めて独自の表スペースに配置できます。各索引表スペースは、データ・パーティションとして DMS か SMS のどちらかの同一のストレージ・メカニズムを使用しなければなりません。 LARGE 表スペース内の索引は、最大 2^29 ページまで含めることが可能です。

パーティション表上の索引に関する別の利点には、以下のものがあります。

- 索引のドロップおよびオンラインの索引の作成に関するパフォーマンスが向上します。
- 表の各索引における表スペース特性に異なる値を使用できます (たとえば、スペースの使用効率をより良いものにするには、各索引でページ・サイズを異ならせるのが適切な場合があります)。
- 入出力競合を削減して、表の索引データに効率的な同時アクセスができます。

- 個々の索引をドロップすると、索引再編成を行わなくてもシステムがスペースをすぐに使用できます。
- 索引再編成を実行することを選択した場合、個々の索引を再編成することができます。

図 16 は、1 つの表スペースに内在する、パーティション表上のパーティション化されていない索引を示しています。

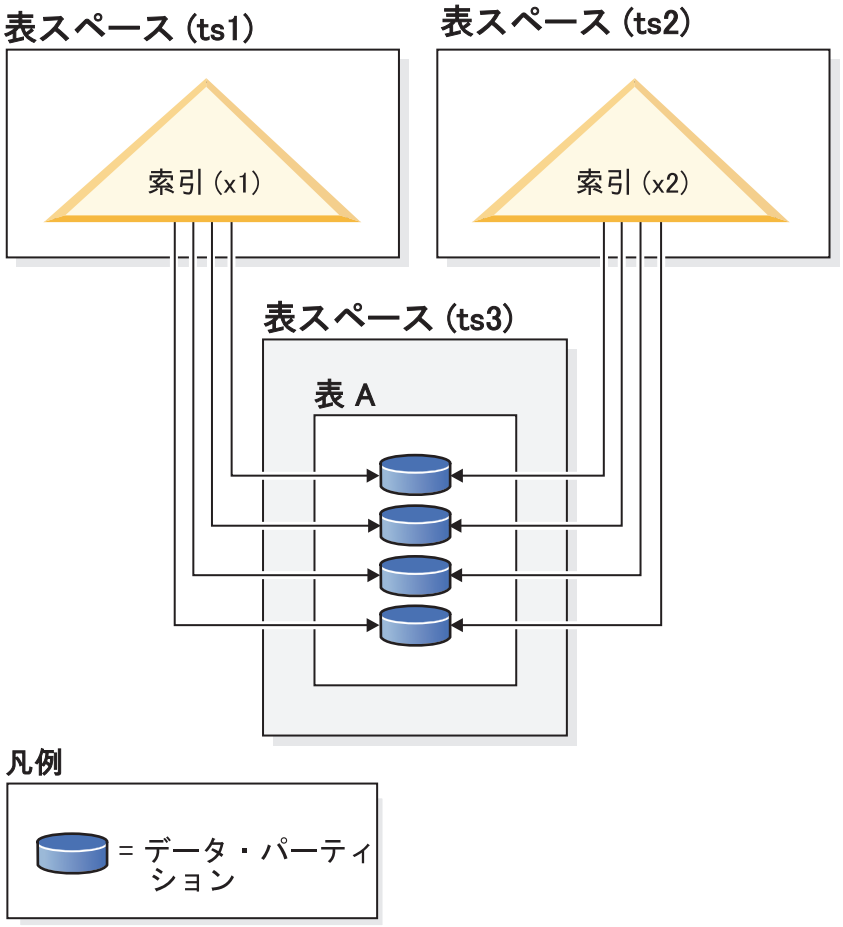


図 16. パーティション表上の索引の動作

121 ページの図 17 は、複数のデータベース・パーティションにも分散されている、パーティション表上の索引の動作を示しています。

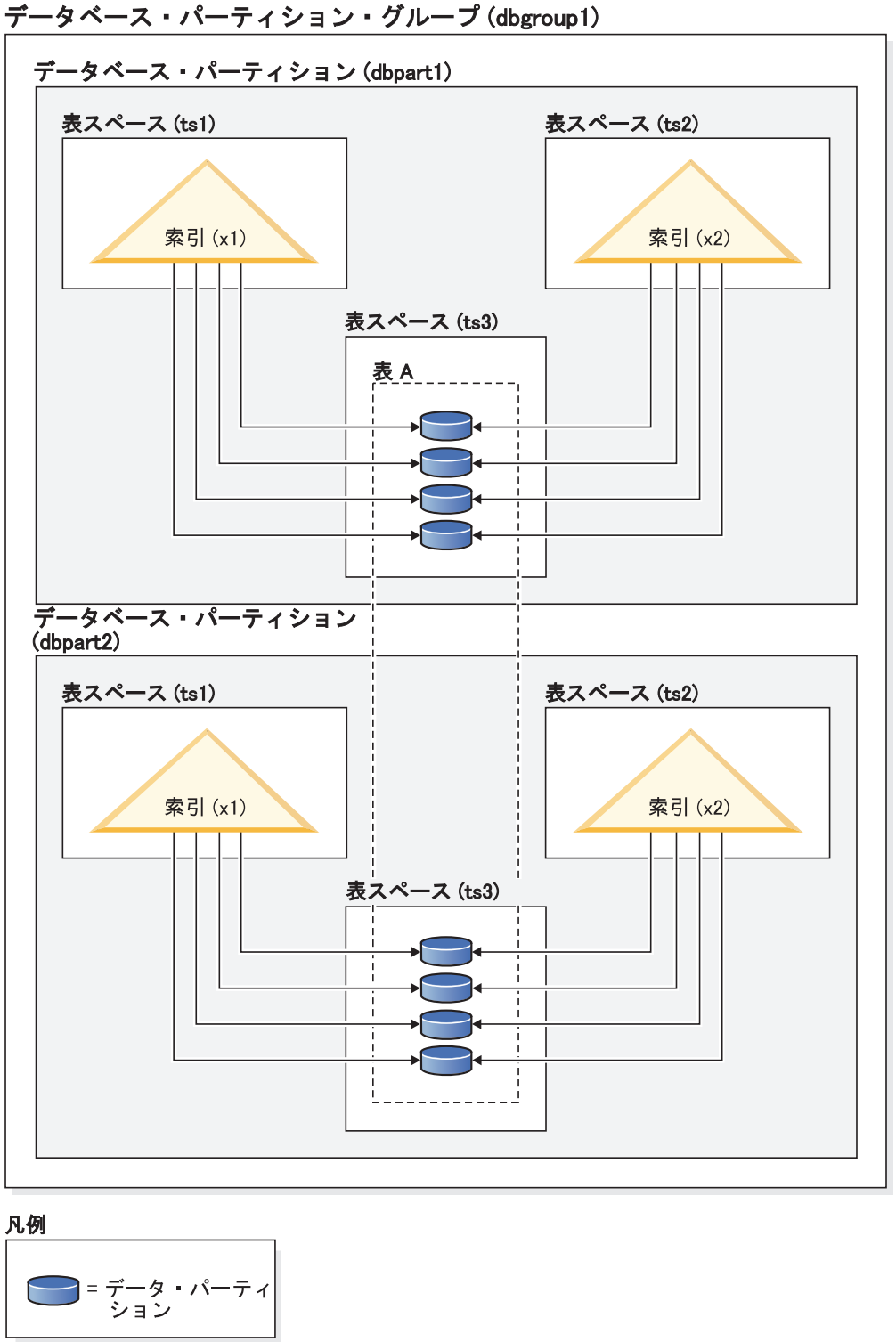


図 17. 分散とパーティション化の両方が行われている表上の索引の動作。

CREATE INDEX ...IN <tbspace1> ステートメントでパーティション表に索引表スペースを指定でき、これは CREATE TABLE .. INDEX IN <tbspace2> ステートメントで指定した索引表スペースとは異なるものにすることも可能です。

パーティション表に限り、CREATE INDEX ステートメントで IN 節を使用して索引位置をオーバーライドでき、それにより索引の表スペース位置を指定することができます。この方法により、必要に応じて、別の表スペース内に、1 つのパーティション表上の複数の異なる索引を配置することができます。パーティション化されていない索引を配置する場所を指定しないでパーティション表を作成し、特定の表スペースを指定しない CREATE INDEX ステートメントを使用して索引を作成する場合、最初のアタッチされたデータ・パーティションまたは表示可能なデータ・パーティションの表スペースに索引が作成されます。次の 3 つの考え得る各ケースをケース 1 から順番に評価して、索引が作成される場所を判別します。この評価は、いずれかのケースに一致すると停止します。

```
ケース 1:
When an index table space is specified in CREATE INDEX ... IN
<tbspace1> statement the table space specified in <tbspace1>
is used for this index.
ケース 2:
When an index table space is specified in the CREATE TABLE .. INDEX IN
<tbspace2> statement the table space specified in <tbspace2>
is used for this index.
ケース 3:
When no table space is specified, use the table space used by the first
attached or visible data partition.
```

索引が作成される場所は、CREATE TABLE ステートメントの実行時に何が行われるかによって異なります。パーティション化されていない表の場合、INDEX IN 節を指定しないと、データベースがユーザーに代わって指定を行い、それはユーザーのデータ表スペースと同じになります。パーティション表の場合には、ブランクのままにするとブランクとして残り、ケース 3 が当てはまります。

例 1: この例では、パーティション表 sales (a int、b int、c int) が存在することを想定し、表スペース 'ts1' にユニーク索引 'a_idx' を作成します。

```
CREATE UNIQUE INDEX a_idx ON sales ( a ) IN ts1
```

例 2: この例では、パーティション表 sales (a int、b int、c int) が存在することを想定し、表スペース 'ts2' に索引 'b_idx' を作成します。

```
CREATE INDEX b_idx ONsales ( b ) IN ts2
```

非同期索引クリーンアップ

非同期索引クリーンアップ (AIC) は、索引項目を無効にする操作の後に行われる、索引の据え置きクリーンアップです。索引のタイプに応じて、項目は行 ID (RID) またはブロック ID (BID) 別にできます。どちらの場合であっても、バックグラウンドで非同期的に操作される索引クリーナーによってこうした項目は除去されます。

AIC は、データ・パーティションがパーティション表からデタッチされるのを促進します。パーティション表に 1 つ以上の非パーティション索引が含まれていると、AIC が開始されます。この場合、AIC はデタッチされたデータ・パーティションおよび疑似削除された項目を参照するすべての非パーティション索引項目を除去します。すべての索引が消去された後に、デタッチされたデータ・パーティションに関連する ID がシステム・カタログから除去されます。

注: パーティション表に定義済みの従属マテリアライズ照会表 (MQT) がある場合、SET INTEGRITY 操作が実行されるまで AIC は開始されません。

AIC の進行中、通常の表アクセスが維持されます。索引にアクセスする照会は、まだ消去されていない無効な項目を無視します。

多くの場合、パーティション表に関連付けられた各非パーティション索引に対して 1 つのクリーナーが開始されます。 AIC タスクを適切なデータベース・パーティションに分散し、データベース・エージェントを割り当てる責任を担っているのは、内部タスク分散デーモンです。

分散デーモンとクリーナー・エージェントは内部システム・アプリケーションです。これらは、LIST APPLICATION 出力にそれぞれ **db2taskd** および **db2aic** というアプリケーション名で表示されます。偶然の中断を防ぐよう、システム・アプリケーションを強制終了することはできません。データベースがアクティブな限り、分散デーモンはオンラインのままです。クリーニングが完了するまでは、クリーナーもアクティブなままです。クリーニングの進行中にデータベースを非アクティブにすると、データベースの再活動時に AIC が再開します。

パフォーマンス

AIC がパフォーマンスに与える影響はごくわずかです。

疑似削除された項目がコミット済みかどうかを判別するには、瞬時の行ロック・テストが必要です。しかし、ロックは決して獲得されないので、並行性には影響ありません。

各クリーナーは最小の表スペース・ロック (IX) および表ロック (IS) を獲得します。それらのロックは、他のアプリケーションがロックを待機中であるとクリーナーが判別した際に解放されます。このことが生じると、クリーナーは一時的に 5 分間処理を中断します。

クリーナーは、ユーティリティー・スロットル機能とも統合されています。デフォルトでは、各クリーナーには 50 のユーティリティー影響優先度があります。この優先度は、SET UTIL_IMPACT_PRIORITY コマンドまたは db2UtilityControl API を使用して変更することができます。

モニター

AIC は、LIST UTILITIES コマンドでモニターできます。それぞれの索引クリーナーは、モニターに別個のユーティリティーとして表示されます。

以下の例は、コマンド行プロセッサー (CLP) インターフェースを使用して、現行のデータベース・パーティションにある WSDB データベース内の AIC 活動を例示しています。

```
$ db2 list utilities show detail

ID                                       = 2
タイプ                                   = 非同期索引クリーンアップ
データベース名                           = WSDB
パーティション番号                       = 0
説明                                     = 表: USER1.SALES、索引: USER1.I2
開始時刻                                 = 12/15/2005 11:15:01.967939
```

```
状態                            = 実行中
呼び出しタイプ                  = 自動
Throttling:
      優先順位                    = 50
Progress Monitoring:
            合計作業                  = 5 ページ
            完了作業                  = 0 ページ
            開始時刻                  = 12/15/2005 11:15:01.979033

ID                              = 1
タイプ                          = 非同期索引クリーンアップ
データベース名                  = WSDB
パーティション番号              = 0
説明                            = 表: USER1.SALES、索引: USER1.I1
開始時刻                        = 12/15/2005 11:15:01.978554
状態                            = 実行中
呼び出しタイプ                  = 自動
Throttling:
      優先順位                    = 50
Progress Monitoring:
            合計作業                  = 5 ページ
            完了作業                  = 0 ページ
            開始時刻                  = 12/15/2005 11:15:01.980524
```

この場合、USERS1.SALES 表で作動する 2 つのクリーナーがあります。1 つのクリーナーは索引 I1 を処理し、もう 1 つは索引 I2 を処理します。「進捗モニター」セクションには、クリーニングが必要な索引ページの見積もり合計数と、クリーンな索引ページの現行数が示されます。

State フィールドは、クリーナーの現行状態を示します。通常はこの状態は「実行中」です。使用できるデータベース・エージェントにクリーナーが割り当てられるのを待機している場合、またはロック競合のためにクリーナーが一時的に中断している場合には、クリーナーは「待機中」状態になる場合があります。

注: それぞれのデータベース・パーティションは、各データベース・パーティション上のタスクに限り ID を割り当てるので、異なるデータベース・パーティションの異なるタスクであっても、ユーティリティー ID が同じになる場合があります。

オンライン索引のデフラグ

索引のリーフ・ページにある使用済みスペースの最小量としてユーザー定義可能な限界値により、オンライン索引デフラグが使用可能になります。リーフ・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、索引の隣接するリーフ・ページがチェックされ、2 つのリーフ・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、バックグラウンドで即時にマージが行われます。

索引のオンラインでのデフラグは、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。既存の索引が、オンラインでのマージ機能を必要とする場合、その索引をいったんドロップしてから MINPCTUSED 節を使って再作成することが必要です。 MINPCTUSED 値は、100 未満の値に設定します。隣接する索引のリーフ・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、オンラインでのデフラグは使用できません。

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、CREATE INDEX ステートメントで MINPCTUSED 節を指定した場合は例外です。 MINPCTUSED 節は、索引リーフ・ページのスペースのパーセントを指定します。索引キーの削除時に、ページ上でいっぱいになったスペースのパーセントが、指定された値以下である場合には、データベース・マネージャーは残りのキーを隣接するページのキーにマージしようとします。隣接ページにスペース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。

索引のノンリーフ・ページは、オンライン索引デフラグ中にマージされません。しかし、空のノンリーフ・ページは削除され、同じ表上の他の索引による再利用が可能になるようにされます。これらのノンリーフ・ページを DMS ストレージ・モデルの他のオブジェクト用に解放したり、 SMS ストレージ・モデルのディスク・スペースを解放したりするには、表または索引の完全な再編成を実行することが必要です。表および索引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。索引のノンリーフ・ページは、オンライン索引デフラグ中にマージされませんが、空になったときに削除され、再利用のために解放されます。索引中のレベルの数と、リーフ・ページおよびノンリーフ・ページの数が削減されることもあります。

タイプ 2 の索引の場合、キーは表に X ロックがある場合のみ、キー削除中にページから除去されます。そのような操作中は、オンラインでの索引デフラグが効果的です。しかし、キー削除中に表に X ロックがない場合、キーは削除のマークを付けられますが、物理的には索引ページから除去されません。その結果、デフラグは行われません。

キーに削除のマークが付けられたのに物理的には索引ページに残っているタイプ 2 の索引でデフラグを実行するには、CLEANUP ONLY ALL オプションを指定した REORG INDEXES コマンドを実行します。 CLEANUP ONLY ALL オプションは、MINPCTUSED の値にかかわらず索引のデフラグを実行します。 CLEANUP ONLY ALL を指定して REORG INDEXES を実行すると、そのようなマージによってマージされたページに少なくとも PCTFREE フリー・スペースを残す場合、 2 つの近隣のリーフ・ページがマージされます。 PCTFREE は、索引作成時に指定され、デフォルトは 10 % です。

パーティション表でのクラスタリング索引の動作について

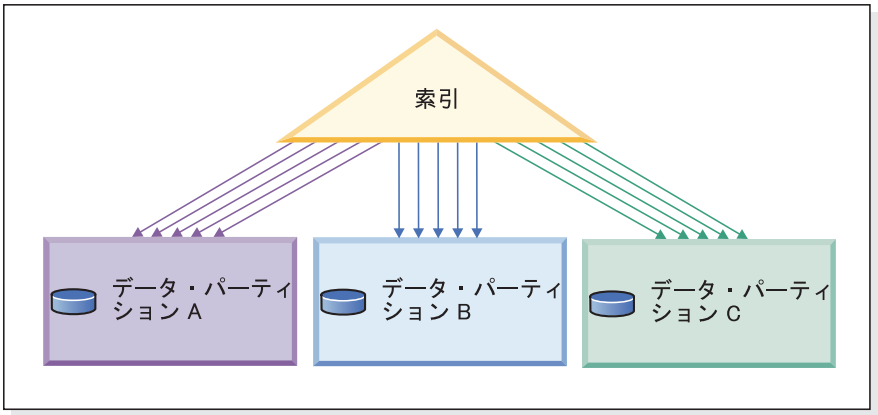
クラスタリング索引には、REGULAR 表に対するのと同じ利点がパーティション表に関してもあります。しかし、表パーティション・キー定義に関してクラスタリング索引を選択する際には注意が必要です。

任意のクラスタリング・キーを使用して、パーティション表にクラスタリング索引を作成できます。データベース・サーバーは、クラスタリング索引を使用して、各データ・パーティション内でデータをローカルにクラスター化しようとします。クラスター化された挿入の際、索引でルックアップが行われ、適切な行 ID (RID) を検索します。この RID は、表内のスペースを検索してレコードを挿入する際の開始点として使用されます。効率が良く、十分に保守されたクラスタリングを実行するには、索引列と表パーティション・キー列間に相関がなければなりません。そのような相関を確保する 1 つの方法は、以下の例に示されているように、表パーティション・キー列ごとに索引列を先頭に付けることです。

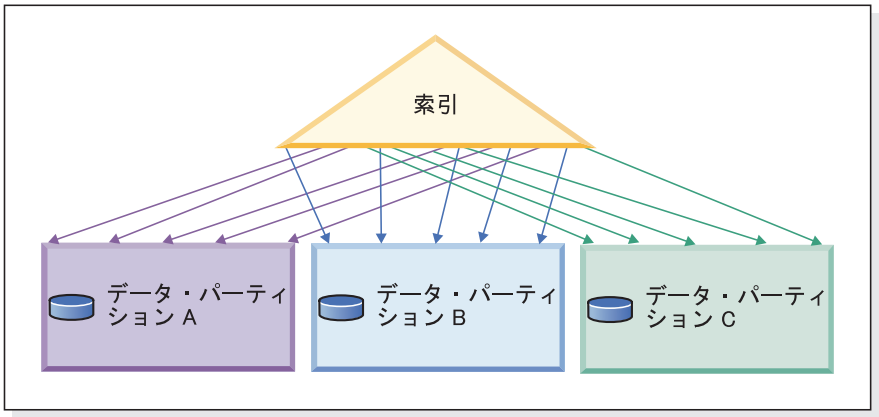
```
PARTITION BY RANGE (Month, Region)
CREATE INDEX ...(Month, Region, Department) CLUSTER
```

データベース・サーバーがこの相関を強制することはありませんが、適切なクラスタリングを行うために索引中のすべてのキーがパーティション ID ごとに互いにグループ化されていることが期待されています。たとえば、表を四半期でパーティション化し、クラスタリング索引を日付に定義すると、その四半期と日付間には関連があるため、効率のよい最適なデータのクラスタリングを行うことが可能です。データ・パーティションのすべてのキーが索引内で相互にグループ化されているからです。

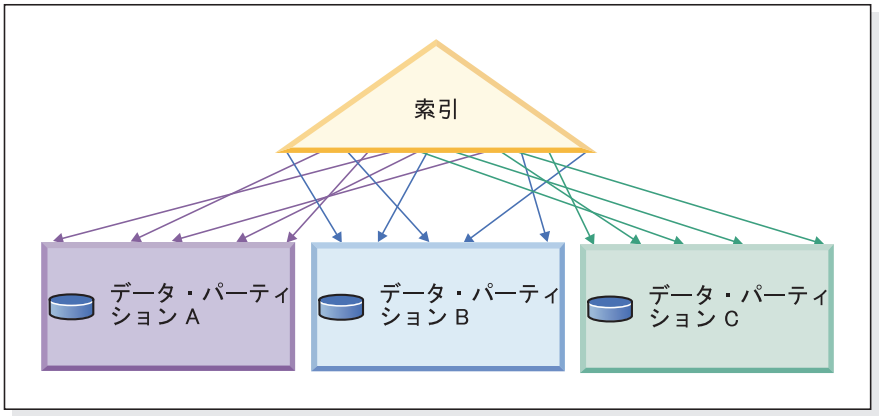
接頭部としてパーティション・キーを使用したクラスタリング (相関)



クラスタリングがパーティション・キーに一致しない (ローカルにクラスター化)



クラスタリングなし



凡例

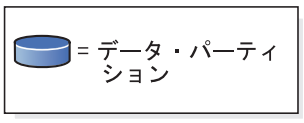


図 18. パーティション表でのクラスター化された索引の見込まれる効果。最初の図では、データはグローバルとローカルの両方でクラスター化されます。

127 ページの図 18 に示されているように、各例の索引およびデータのレイアウトが指定されていると、クラスタリングが表パーティション・キーと相関関係にある場合に最適なスキャン・パフォーマンスが得られます。クラスタリングが表パーティション・キーと相関していないと、索引がローカルにクラスター化されることはおそらくありません。表パーティション列と索引列間に相関があることが期待されているため、ローカルでの最適なクラスター化シナリオが生じることは極めてまれです。

クラスタリングの利点には、以下のものがあります。

- 各データ・パーティション内で、行はキーの順序になります。
- クラスタリング索引は、キーの順序で表内をトラバースするため、スキャンのパフォーマンスが向上します。なぜなら、スキャンは最初のページの最初の行を取り出し、その後そのページのすべての行を取り出すまで同じページの各行を取り出してから、次に移動するからです。つまり、どの時点でも表の 1 ページのみがバッファー・プール内になければならないという意味です。対照的に、表がクラスター化されていない場合、異なるページから各行が取り出される確率が高くなります。バッファー・プールに表全体を保持できる余地がある場合を除いて、各ページが何度も取り出されることになり、スキャンの速度がとても遅くなります。

  パーティション表の場合、スキャンの際に 1 度限り各ページが取り出されるという理想的な状況は、表パーティション・キーがクラスタリング・キーの前に付いている場合に限り生じ得ます (127 ページの図 18 の最初の図を参照してください)。しかし、前述のようにクラスタリング・キーが表パーティション・キーと相関しておらず、データがローカルにクラスター化される場合、バッファー・プールに各データ・パーティションの 1 ページを保持できるスペースが十分にあれば、クラスター化された索引の利点を十分に生かすことが依然として可能です。指定のデータ・パーティションの取り出されるそれぞれの行が、同じデータ・パーティションの以前に取り出された行の近くにあるからです (127 ページの図 18 の 2 番目の図を参照してください)。既に言及したように、クラスタリング・キーが表パーティション・キーと相関していない場合にはクラスタリングを十分に維持することができませんが、ご使用の表で高水準の挿入、更新、および削除活動を期待しない場合にはこの方法が役に立ちます。

  バッファー・プールにすべてのデータ・パーティションのページを保持するための十分なスペースがない場合でも、クラスター化索引を定義すると、幾らかの利点を引き続き活用できます。

データベース・エージェント

アプリケーションがアクセスするそれぞれのデータベースごとに、各種プロセスまたはスレッドは、様々なアプリケーション・タスクの実行を開始します。これらのタスクには、ロギング、通信、プリフェッチなどが含まれます。

データベース・エージェントとは、アプリケーション要求にサービスを提供するために使用される、データベース・マネージャー内のスレッドです。バージョン 9.5 では、エージェントはすべてのプラットフォーム上でスレッドとして実行されます。

アプリケーション接続の最大数は、 *max_connections* データベース・マネージャー構成パラメーターにより制御されます。各アプリケーション接続の作業は、1 つの作業エージェントによって調整されます。

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アタッチされるものではありません。コーディネーター・エージェントは、アプリケーションが切断するまでそのアプリケーションにアタッチしたままなので、アプリケーションとの最も長い関連を示します。この規則の唯一の例外は、エンジン・コンセントレーターが使用可能にされている場合です。この場合、コーディネーター・エージェントはその関連をトランザクション境界で終了することがあります (トランザクション COMMIT、ROLLBACK)。

以下の 4 種類の作業エージェントがあります。

- アイドル・エージェント
- アクティブ・コーディネーター・エージェント
- サブエージェント

アイドル・エージェント

これは、最も単純な形式のエージェントです。このエージェントにはアウトバウンド接続がなく、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。

アクティブ・コーディネーター・エージェント

クライアント・アプリケーションのデータベース接続にはそれぞれ、データベース上の作業を調整するアクティブ・エージェントが 1 つあります。コーディネーター・エージェントが作成された後、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、プロセス間通信 (IPC) および遠隔通信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・プロセスは自らの専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、アクティブ・コーディネーター・エージェントは非アクティブ・エージェントになります。

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デタッチされると、そのコーディネーター・エージェント状態は次のようになります。

- アクティブ・エージェント。他の接続が待機状態の場合は、作業エージェントがアクティブ・コーディネーター・エージェントになります。
- 他の接続が待機状態になく、プール・エージェントが自動的に管理されているか、または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ます。
- 他の接続が待機状態になく、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ます。

サブエージェント

コーディネーター・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。コーディネーター・エージェントが作成された後、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。 DB2 バージョン 9.5 では、サブエージェントは非パーティション環境および照会内並列処理が有効にされていない環境にも存在することがあります。

どのアプリケーションの作業も実行せず、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・プログラムの作業を行うコーディネーター・エージェントからの要求のために、あるいは、既存のコーディネーター・エージェントの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 *num_poolagents* によって変わります。

エージェントが必要なときにアイドル・エージェントがない場合には、新規エージェントが動的に作成されます。新規エージェントを作成するには一定のオーバーヘッドが必要なので、アイドル・エージェントをクライアントに対してアクティブにできる場合、 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作業を行うとき、そのサブエージェントはアプリケーションに*関連付けられます*。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、データベース・マネージャーは新規エージェントを作成する前に、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検査します。

データベース・エージェント管理

ほとんどのアプリケーションは、接続済みアプリケーションの数と、データベースによる処理が可能なアプリケーション要求の数の間の 1 対 1 の関係を確立します。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。

これらの要因を別々に制御する機能は、以下の 2 つのデータベース・マネージャー構成パラメーターによって提供されます。

- *max_connections* パラメーター。これは、接続済みアプリケーションの数を指定します。
- *max_coordagents* パラメーター。これは、同時に処理可能なアプリケーション要求の数を指定します。

接続コンセントレーターは、 *max_connections* 値が *max_coordagents* 値より大きい場合に使用可能になります。

各アクティブ・コーディネーター・エージェントはグローバル・リソースのオーバーヘッドを必要とするので、このエージェントの数が多ければ、使用できるデータベース・グローバル・リソースの上限に達する可能性も高くなります。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のを避けるには、*max_connections* に、 *max_coordagents* より高い値を設定することができます。

max_connections および *max_coordagents* の値を設定すると、AUTOMATIC も指定できます。 AUTOMATIC を使用することが有利になる 2 つの特定のシナリオがあります。

- 必要とされるすべての接続をシステムが処理できることははっきりしているが、使用されるグローバル・リソースの量を (コーディネーター・エージェントの数を制限することにより) 制限する場合は、*max_connections* パラメーターのみに AUTOMATIC を指定します。 *max_connections* が、*max_coordagents* より大きい値が指定された AUTOMATIC として設定されている場合、これは (十分なシステム・リソースがある限り) 接続がいくつ許可されていても接続コンセントレーターが有効であるが、コーディネーター・エージェントの最大数は制限されたままであることを意味します。これは並行して実行するアプリケーションの数を制限することで、メモリーとディスクの両方の制約を制御するために使用できます。
- 接続およびコーディネーター・エージェントの最大数を制限するために人工的な制限をシステムに課したくないが、システムが (接続されたアプリケーションと処理されるアプリケーション要求との間で) 多対 1 の関係を必要としている、またはそれに利点があることが分かっている場合、接続コンセントレーターを有効にして、両方のパラメーターを AUTOMATIC に設定する必要があります。両方のパラメーターが AUTOMATIC に設定されている場合、データベース・マネージャーは指定されたこの値を、コーディネーター・エージェント対接続の理想的な数を表す比率として使用します。

例

次のセットアップを考慮してください。

- *max_connections* パラメーターが AUTOMATIC (値 300) に設定されている。
- *max_coordagents* パラメーターが AUTOMATIC (値 100) に設定されている。

max_connections 対 *max_coordagents* の比率は、300 対 100 です。データベース・マネージャーは接続が行われる時には新しいコーディネーター・エージェントを優先するので、集中は必要な場合にのみ適用されます。上記の設定は以下のように変換されます。

- 接続 1 から 100 では、新しいコーディネーター・エージェントが作成される。
- 接続 101 から 300 では、新しいコーディネーター・エージェントが作成されない。それらは既に作成済みの 100 エージェントを共有する。
- 接続 301 から 400 では、新しいコーディネーター・エージェントが作成される。
- 接続 401 から 600 では、新しいコーディネーター・エージェントが作成されない。それらは既に作成済みの 200 エージェントを共有する。
- 以降、同様に続きます。

注: この例では、接続済みアプリケーションが、各ステップで新規コーディネーター・エージェントを確実に作成できるほど十分に実行されていることを前提としています。一定の期間後に、接続済みアプリケーションが機能しなくなっている場合、コーディネーター・エージェントは非アクティブになり、終了します。

接続の数は減っているが、残りの接続により実行中の作業の量が多い場合は、コーディネーター・エージェントの数が直ちには減らない場合があります。

max_connections および *max_coordagents* パラメーターは、エージェント・プール、またはエージェント終了に直接影響することはありません。通常のエージェント終了規則が引き続き適用されますが、これはコーディネーター・エージェントへの接続の数が、指定した比率を正確に表していない可能性があることを意味します。エージェントは、終了する前に再利用のためにエージェント・プールに戻される場合があります。

より細分性の高い制御が必要な場合は、単純化された比率を推奨します。例えば、上記の 300 対 100 の比率は、3 対 1 の比率に減らすことができます。*max_connections* を 3 (AUTOMATIC) に設定し、*max_coordagents* を 1 (AUTOMATIC) に設定する場合、3 の接続ごとに 1 のコーディネーター・エージェントを作成できます。

クライアント接続用の接続コンセントレーターの改善

比較的一過性の接続が多いインターネット・アプリケーション、およびそれに類するアプリケーションに対して、接続コンセントレーターのパフォーマンスが改善され、より多くのクライアント接続を効率的に処理できるようになりました。さらに、それぞれの接続のメモリー使用量も削減され、コンテキスト切り替えの数が減りました。

注: 接続コンセントレーターは、 *max_connections* 値が *max_coordagents* 値より大きい場合に使用可能になります。

多数のユーザーが同時接続する必要のある環境では、システム・リソースをより効率的に使用するために、接続コンセントレーターを使用可能にすることができます。この機能は、これまで DB2 Connect™ 接続プールでのみ利用できた機能を取り入れたものです。接続プールと接続コンセントレーターは、どちらも「DB2 Connect ユーザーズ・ガイド」で説明されています。最初の接続の後、接続コンセントレーターはホストへの接続時間を削減します。ホストからの切断が要求されると、インバウンド接続はドロップされますが、ホストへのアウトバウンド接続はプール内に保持されます。ホストへの新しい接続が要求されると、データベース・マネージャーは既存のアウトバウンド接続をプールから再使用することを試みます。

注: アプリケーションで接続プールまたは接続コンセントレーターを使用する場合、パフォーマンス最適化のために、キャッシュされるデータ・ブロック・サイズを制御するパラメーターを調整してください。詳細については、「DB2 Connect ユーザーズ・ガイド」を参照してください。

接続プールを使用する場合、 DB2 Connect はインバウンド TCP/IP 接続、アウトバウンド TCP/IP 接続だけを使用できます。

使用例

例 1: ある ESE 環境に 1 つのデータベース・パーティションがあり、平均して 1,000 ユーザーがデータベースに接続するとします。時には、接続されているユーザーの数がさらに多くなることもあるため、1000 という制限を設定するべきではありません。並行トランザクションの数は 200 に達しますが、250 を超えることは決してありません。トランザクションは短期間です。

このワークロードを処理するために、管理者は以下のようなデータベース・マネージャー構成パラメーターを設定できます。

- *max_coordagents* を 250 に設定して、並行トランザクションの最大数をサポートします。
- *max_connections* は AUTOMATIC (値 1000) として設定し、これにより接続をいくつでもサポートします (この例では、250 より大きい値であれば、接続コンセントレーターをオンにするには十分です)。
- *num_poolagents* はデフォルトとして設定します。これによって、新規作成によるオーバーヘッドを避けながら、データベース・エージェントが着信するクライアント要求にサービスを提供できるようにします。

例 2: 一見したところ接続に制限数がない、例 1 のものと似たシステムでは、接続の数に基づいて、番号コーディネーター・エージェントも増やすことができます。この例では、平均して 1000 ユーザーが接続し、約 250 の同時トランザクションが実行されると想定していますが、ユーザーのピーク数は時々予測不能です。時には 2000 ユーザーが接続する可能性があり、そのうちの平均しておよそ 500 が作業を実行すると予期されています。通常は 1000 ユーザーだけが接続し、それには通常 250 のコーディネーター・エージェントで十分であるので、500 のコーディネーター・エージェントはほとんどの場合に許可されません。

このワークロードを処理するために、データベース・マネージャー構成を以下のように更新できます。

```
db2 update dbm cfg using MAX_COORDAGENTS 250 AUTOMATIC
db2 update dbm cfg using MAX_CONNECTIONS 1000 AUTOMATIC
```

これは、接続の数が 1000 を超えて増える場合、接続の合計数により判別された最大数で、追加のコーディネーター・エージェントが必要に応じて作成されるということです。ワークロードの増加に応じてどちらのパラメーターにも値を指定し、どちらも AUTOMATIC と設定すると、データベース・マネージャーは、コーディネーター・エージェント対接続の概算比率を維持します。

例 3:

接続コンセントレーターを有効にはしないが、接続ユーザーの数を制限する (例えば同時に 250 の接続ユーザーだけを許可する) システムでは、データベース・マネージャー構成パラメーターを以下のように設定します。

- *max_connections* を 250 に設定します。
- *max_coordagents* を 250 に設定します。

例 4:

接続コンセントレーターを有効にせず、接続するユーザーの数も制限しないシステムでは、データベース・マネージャー構成パラメーターを以下のように設定します。

```
db2 update dbm cfg using MAX_COORDAGENTS AUTOMATIC
db2 update dbm cfg using MAX_CONNECTIONS AUTOMATIC
```

パーティション・データベースにおけるエージェント

パーティション・データベース環境、およびパーティション内並列処理が使用可能になっている環境の場合には、各データベース・パーティション (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、または新規の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。

パーティション・データベース環境およびパーティション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 つのデータベース・パーティション (ノードとも呼ばれる) でアプリケーションとの関連付けを保持できるエージェントとサブエージェントの両方の数に影響します。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ている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。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、パフォーマンスが低下します。

  デフォルトでは *num_poolagents* は AUTOMATIC (値 100) に設定されます。このパラメーターが AUTOMATIC に設定されると、データベース・マネージャーはプールするアイドル・エージェントの数を自動的に管理します。

  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。そして、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・プールからアクティブでないサブエージェントをリサイクルします。この動作により、リソースは完全に使用されます。
- エージェントを少ししか持たないことと、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・コストと比較してください。

  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、他のタスクでは使用可能になっていないデータベース・マネージャー・リソースが使用される可能性があります。

  注: 接続コンセントレーターが有効な場合、*num_poolagents* によって指定されたエージェント数は、同時にプール内でアイドル状態のままであるエージェントの正確な数を必ずしも反映するわけではありません。エージェント数は、さらに多くのワークロード・アクティビティーのオカレンスを処理するために、一時的に超過する可能性があります。

その他の非同期プロセスおよび非同期スレッド

データベース・マネージャーが独自のプロセスまたは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があります。

- データベース入出力サーバーまたは入出力プリフェッチャー
- データベース非同期ページ・クリーナー
- 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
- 通信および IPC listener
- 表スペース・コンテナー再平衡機能

データベース・システム・モニター情報

DB2 データベース・マネージャーは、データベース・マネージャーの操作、パフォーマンス、およびこれ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 および XQuery 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) も示されます。

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、 SYSMON のいずれかの権限が必要です。)

スナップショットをとるか、イベント・モニターを使用するかのいずれかの方法により、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ことができます。

スナップショットをとる

以下の 3 つのうちのいずれかの方法によってスナップショットをとることができます。

- コマンド行から GET SNAPSHOT コマンドを使用します。
- db2GetSnapshot() API 呼び出しを使用して独自のアプリケーションを作成します。

- スナップショット表関数を使用して、データベース・システムの特定の領域に関するモニター・データを戻してください。

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まれます。

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・モニターがアクティブ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 バイナリー・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。

デッドロック・イベント・モニターを使用する場合のパフォーマンスの影響

デッドロック・イベント・モニターがアクティブで、HISTORY オプションが有効になっている場合、DB2 データベース・システムの一般的なパフォーマンスは以下のように影響を受けます。

- ステートメント履歴にリストされるキャッシュされた動的 SQL および XQuery ステートメントのパッケージ・キャッシュで使用されるメモリーは、特定のステートメントの履歴が必要でなくなる (つまり、現在の作業単位が終了する) まで解放されません。 キャッシュで使用するスペースが増加し、解放されないため、パッケージのキャッシュのサイズが増加します。

- ステートメント情報をステートメント履歴リストにコピーすることにより、システム・パフォーマンスに小さな影響があります。
- データベース・パーティションにある各アクティブ・アプリケーションのステートメント履歴リストを保持するために、各データベース・パーティションで DB2 システム・モニター・ヒープの使用が増加します。増加する量は、各アプリケーションにより作業単位で実行されるステートメントの数によって異なります。モニター・ヒープに関して推奨される計算は以下のとおりです。

```
If an event monitor is of type DEADLOCK
   and the WITH DETAILS HISTORY option is running,
add X*100 bytes times the maximum number of concurrent applications
   you expect to be running,
where X is the expected maximum number of statements in your
   application's unit of work.
If the event monitor is of type DEADLOCK
   and the WITH DETAILS HISTORY VALUES option is running,
also add X*Y bytes times the maximum number of concurrent applications
   you expect to be running,
where Y is the sum of the expected maximum size of parameter values being
   bound into your SQL and XQuery statements.
```

デッドロック・イベント・モニターがアクティブで、VALUES オプションが有効になっている場合、DB2 データベース・システムの一般的なパフォーマンスは以下のように影響を受けます (HISTORY オプションに関してすでに挙げたものに加えて)。

- ステートメント情報をステートメント履歴リストにコピーすることにより、システム・パフォーマンスに非常に小さな影響があります。
- データベース・パーティションにある各アクティブ・アプリケーションのステートメント履歴リストを保持するために、各データベース・パーティションで DB2 システム・モニター・ヒープの使用が増加します。増加する量は、各アプリケーションによって作業単位で実行されるステートメントの数に加え、ステートメントごとに使用されるデータ値の数によって異なります。
- データベース・マネージャーは、変数のサイズと数に応じて、大量の追加のデータ値を保守するため、パフォーマンスに影響を与える場合があります。

デッドロック・イベント・モニターに HISTORY と VALUES の両方のオプションが指定される場合、DB2 システム・モニター・ヒープがメモリーに与える影響は相当なものになります。この影響を減らすには、必要な場合のみこれらのオプションを使用するようにしてください。影響を減らす別の方法としては、イベント・モニターを有効にする前に、すべてのデータベース・パーティションで構成されている DB2 システム・モニターのヒープ・サイズを増やすことができます。

実際にデッドロックが発生し、デッドロック・イベント・モニターがアクティブである場合、イベント・モニター・レコードの生成のためにシステム・パフォーマンスが影響を受けます。影響の度合いとその期間は、関係するステートメントの履歴リストのステートメントとデータ値の数だけでなく、デッドロックに関係したアプリケーションおよびデータベース・パーティションの数によっても決まります。

効率的な SELECT ステートメント

SQL は柔軟な高水準言語なので、複数の異なる SELECT ステートメントを作成して、同じデータを検索することができます。しかし、ステートメントの形式が異なり、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。

SELECT ステートメントについては、以下の指針を考慮してください。

- 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。
- 応答セットを必要な行のみに制限する述部を使用します。
- 必要な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、 OPTIMIZE FOR を指定します。この節は、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。
- 検出する行数が少ない場合には、 OPTIMIZE FOR *k* ROWS 節だけを指定します。 FETCH FIRST n ROWS ONLY 節は不要です。ただし、*n* の値が大きく、先頭の *k* 行を素早く取り出し、後続の *k* 行については遅延を可能にする場合には、両方の節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。以下の例は、両方の節を表しています。

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

- 行ブロッキングを利用するには、 FOR READ ONLY または FOR FETCH ONLY 節を指定してパフォーマンスを改善します。検出された行には排他ロックがかけられないので、これに伴って並行性も向上します。追加の照会の再書き込みも行えます。 BLOCKING ALL BIND オプションと一緒に FOR READ ONLY または FOR FETCH ONLY 節を指定するなら、フェデレーテッド・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も同様に向上します。
- 位置指定の更新を使用して更新するカーソルに対しては、 FOR UPDATE OF 節を指定して、データベース・マネージャー が初めにより適切なロッキング・レベルを選択して、発生する可能性のあるデッドロックを回避できるようにします。FOR UPDATE カーソルは、行ブロッキングの利点は生かせません。
- 検索済み更新を使用して更新するカーソルに対しては、 FOR READ ONLY および USE AND KEEP UPDATE LOCKS 節を使用して影響を受ける行を強制的に U ロックすると、デッドロックを回避し、行ブロッキングを引き続き使用できます。
- 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、ランタイム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、VARCHAR 型ではなく、CHAR 型
  – 浮動小数点数や 10 進数ではなく、整数
  – 文字列型ではなく、日時
  – 文字列型ではなく、数値

- ソート操作が発生する可能性を小さくするには、 DISTINCT または ORDER BY などの節や操作を、必要でなければ省略します。
- 表に行があるかどうか調べるときには、単一の行を選択します。カーソルをオープンして 1 つの行を取り出すか、単一行 (SELECT INTO) 選択を実行してください。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。

  表が非常に小さいものであると分かっているのでない限り、以下のステートメントを使用して非ゼロ値を検査することはしないでください。

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

  大規模な表の場合、すべての行のカウントを行うとパフォーマンスに影響します。
- 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ます。
- 複数の述部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ます。大きい IN リストをホスト変数と共に使用すると、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。索引は、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。
- データベース・マネージャーは一部の結合手法を使用できないので、可能なら、結合述部で式または OR 節を使用しないようにします。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、パーティション・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パーティション化されるようにしてください。

第 15 章 ガバナー・ユーティリティー

ガバナーは、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動作をモニターし、ガバナー構成ファイルで指定した規則によって、特定の動作を変更することができます。

ガバナー・インスタンスは、フロントエンド・ユーティリティーおよび 1 つ以上のデーモンで構成されています。開始するガバナーの各インスタンスは、データベース・マネージャーのインスタンスに特定のものです。デフォルトでは、ガバナーを開始すると、パーティション・データベースの各データベース・パーティションでガバナー・デーモンが開始します。ただし、モニターする単一のデータベース・パーティションでデーモンが開始するように指定することもできます。

注: ガバナーがアクティブになると、そのスナップショット要求によって、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可能性があります。パフォーマンスを向上させるには、ガバナー・ウェイクアップ・インターバルを大きくすることにより CPU の使用を削減してください。

それぞれのガバナー・デーモンは、データベースに対して実行しているアプリケーションについての情報を収集します。そしてその情報を、このデータベースについてガバナー構成ファイルで指定した規則と比較して検査します。

ガバナーは、構成ファイルの規則によって指定されたとおりにアプリケーション・トランザクションを管理します。たとえば、規則を適用すると、アプリケーションは特定のリソースを過剰に使用していることが示されたとします。規則は、アプリケーションの優先順位を変更する、またはそのアプリケーションをデータベースから強制切断するなどの取るべき処置を指定します。

規則に関連した処置がアプリケーションの優先順位を変更する場合、ガバナーはリソース違反が起こったデータベース・パーティションに対するエージェントの優先順位を変更します。パーティション・データベースでは、アプリケーションがデータベースから強制切断される場合、違反を検出したデーモンがそのアプリケーションのコーディネーター・ノードで実行されていても、その処置が行われます。

ガバナーは、ガバナーが行った処置のログをすべて記録します。処置について検討するには、ログ・ファイルを照会してください。

ガバナーの開始と停止

ガバナー・ユーティリティーは、データベースに接続しているアプリケーションをモニターし、そのデータベースに対するガバナー構成ファイルで指定した規則に従って、それらのアプリケーションの動作を変更します。

ガバナーを開始する前に、構成ファイルを作成する必要があります。

ガバナーを開始または停止するには、 *sysadm* または *sysctrl* 権限を持っていなければなりません。

ガバナーを開始または停止するには、次のようにします。

1. ガバナーを開始するには、DB2 コマンド行で *db2gov* コマンドを実行します。以下の必要パラメーターを入力してください。
   - START *database_name*

     指定するデータベース名は、指定する構成ファイルでのデータベース名と同じものにしなければなりません。名前が異なると、エラーが戻されます。ガバナーが複数のデータベースに対して実行している場合、それぞれのデータベースごとにデーモンが開始されることに注意してください。
   - *config_file_name*

     そのデータベースに対するガバナーの構成ファイル名。ファイルがデフォルトのロケーション (sqllib ディレクトリー) にない場合、ファイル名と共にパスも含める必要があります。
   - *log_file_name*

     このガバナーに対するログ・ファイルの基本名。パーティション・データベースでは、ガバナーのこのインスタンスに対してデーモンが実行されるデータベース・パーティションごとに、データベース・パーティション番号が付加されます。

   パーティション・データベースの単一のデータベース・パーティションでガバナーを開始するには、 *nodenum* オプションを追加してください。

   たとえば、*sales* というデータベースに対するガバナーを、パーティション・データベースのノード 3 のみで、 *salescfg* という構成ファイルおよび *saleslog* というログ・ファイルを使用して開始するには、以下のコマンドを入力してください。

   ```
   db2gov START sales nodenum 3 salescfg saleslog
   ```

   sales データベースのすべてのデータベース・パーティションでガバナーを開始するには、以下のコマンドを入力してください。

   ```
   db2gov START sales  salescfg saleslog
   ```
2. ガバナーを停止するには、 STOP オプションを指定して *db2gov* コマンドを入力します。

   たとえば、 *sales* データベースのすべてのデータベース・パーティションでガバナーを停止するには、以下のコマンドを入力してください。

   ```
   db2gov STOP sales
   ```

   データベース・パーティション 3 でのみガバナーを停止するには、以下のコマンドを入力してください。

   ```
   db2gov START sales nodenum 3
   ```

ガバナー・デーモン

ガバナー・デーモンは、db2gov ユーティリティーによって実行するか、またはウェイクアップするかのいずれかで開始されます。ガバナー・デーモンが開始されると、以下のタスク・ループを実行します。

1. ガバナー構成ファイルが変更されたか、またはファイルがまだ読み取られていないかどうかを検査します。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、ガバナー・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。
2. データベース上で作動しているアプリケーションおよびエージェントごとに、リソース使用統計についてのスナップショット情報を要求します。

   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。
3. 統計を各アプリケーションごとに、ガバナー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、ガバナーは指定された処置を実行します。

   注: ガバナーは、累積された情報と構成ファイルで定義された値とを比較します。これは、アプリケーションが既にブリーチしている可能性がある新規の値で構成ファイルが更新されている場合、そのブリーチに関係しているガバナーが、次のガバナーのインターバルに即時に適用されることを意味します。
4. これは実行するすべての処置について、ガバナー・ログ・ファイルにレコードを書き込みます。

注: ガバナーは、 *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ために使用することはできません。

ガバナーは、タスクを終えると、構成ファイル内で指定されたインターバルの間はスリープします。そのインターバルが経過すると、ガバナーはウェイクアップして、タスク・ループを再度実行します。

ガバナーがエラーまたはストップ信号を検出した場合、クリーンアップ処理を行ってから終了します。クリーンアップ処理では、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、すべてのアプリケーション・エージェントの優先順位がリセットされます。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、ガバナーの終了後には、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。エラーが起きた場合、ガバナーは、異常終了したことを示すメッセージを管理通知ログに書き込みます。

注: ガバナー・デーモンはデータベース・アプリケーションではなく、そのためデータベースへの接続は維持しませんが、インスタンスの接続はあります。スナップショット要求を出すことができるので、ガバナー・デーモンは、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。

ガバナーの構成

ガバナーを構成するには、ガバナーのインスタンスがモニターするデータベース、および照会の管理方法を決定する、構成ファイルを作成します。

構成ファイルは、規則のセットで構成されています。最初の 3 つの規則は、モニターするデータベース、ログ・レコードを書き込むインターバル、およびモニターの

ためにウェイクアップするインターバルを指定します。残りの規則は、データベース・サーバーのモニター方法、および特定の状況でどのような処置を取るかを指定します。

ガバナー構成ファイルを作成するには、次のようにします。

1. すべてのデータベース・パーティションに取り付けられていて使用可能なディレクトリーに、記述名を持った ASCII ファイルを作成します。例えば、**sales** データベースをモニターするガバナー・インスタンスの構成ファイルに、 *govcfgsales* といった名前を付けることができます。
2. 任意のテキスト・エディターでそのファイルを開き、構成情報および処置の条件を入力します。

   各規則の終わりには、セミコロン (;) を置きます。以下の構成情報が推奨されています。

   - 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: ガバナー・インスタンスが CPU 使用統計をログ・ファイルに書き込む、分単位の間隔。
   - interval: ガバナー・デーモンがウェイクアップしてアクティビティーをモニターする、秒単位のインターバル。インターバルを指定しない場合、デフォルト値の 120 秒が使用されます。

   たとえば、構成ファイル内の最初の 3 つの規則は次のようになります。

   ```
   { Wake up once a second, the database name is sales,
    do accounting every 30 minutes.  }
   interval 1; dbname sales; account 30;
   ```

   モニターする状態、およびその規則が真と評価されたときに取る処置を指定する規則を追加します。たとえば、次のようにして、作業単位 (UOW) が実行する時間を 1 時間に制限し、その後はデータベースから強制切断するという規則を追加することができます。

   ```
   setlimit uowtime 3600 action force;
   ```

3. ファイルを保管します。

ガバナー構成ファイル

ガバナー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規則を含んだ構成ファイルを指定します。ガバナーはそれぞれの規則を評価し、規則が真と評価されたときに、指定されたとおりの処置を行います。

規則要件が変更になった場合には、ガバナー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。各ガバナー・デーモンは、構成ファイルが変更されたことを検出し、ファイルを再度読み取ります。

構成ファイルは、各データベース・パーティション上のガバナー・デーモンが同一の構成ファイルを読み取れるように、すべてのデータベース・パーティション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、モニターするデータベース、ログ・レコードを書き込むインターバル、およびガバナー・デーモンのスリープ・インターバルを識別する 3 つの必須規則で構成されています。これらのパラメーターに続いて、構成ファイルには、オ

プションのアプリケーション・モニターに関する規則および処置が含まれています。次のコメントは、すべての規則に適用されます。

- 注釈は、中括弧 { } に入れて区切ります。
- 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。例外はアプリケーション名で (applname 規則の引数として指定されます)、これは大文字小文字の区別をします。
- 各規則は、セミコロン (;) で終わります。

必須規則

以下の規則によって、モニターするデータベース、およびデーモンが各活動のループ後にウェイクアップするインターバルを指定します。これらの規則はそれぞれ、ファイルに 1 回のみ指定されます。

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® 環境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。ガバナー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。

interval
デーモンがウェイクアップする時間間隔 (秒単位)。インターバルを指定しない場合、デフォルト値の 120 秒が使用されます。

処置を管理する規則

必須規則に続けて、アプリケーションの管理方法を指定する規則を追加することができます。これらの規則は、規則節と呼ばれるより小さなコンポーネントから成ります。それらを使用する場合、節は次のように、規則ステートメントに特定の順序で入力しなければなりません。

1. **desc** (オプション): 規則に関する注釈。引用符で囲みます。
2. **time** (オプション): 規則が評価される時間帯。
3. **authid** (オプション): アプリケーションがステートメントを実行する、 1 つ以上の許可 ID。
4. **applname** (オプション): データベースに接続する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。この名前には、大文字と小文字の区別があります。アプリケーション名にスペースが含まれる場合には、その名前を二重引用符で囲む必要があります。
5. **setlimit**: ガバナーが検査する制限。たとえば、これらには CPU 時間、戻される行の数、またはアイドル時間などがあります。

6. **action** (オプション): 制限に達した場合に実行する処置。処置が指定されていない場合、制限に達すると、ガバナーはアプリケーションに対して作動しているエージェントの優先順位を 10 低くします。アプリケーションに対する処置には、エージェントの優先順位を下げる、データベースから強制切断する、または運用についてのスケジューリング・オプションを設定することが含まれます。

規則節を組み合わせて、1 つの規則を作ります。次の例に示すように、各節は規則ごとに 1 回のみ使用し、規則の終わりにセミコロンを置きます。

```
desc "Allow no UOW to run for more than an hour"
setlimit uowtime 3600 action force;

desc "Slow down the use of db2 CLP by the novice user"
authid novice
applname db2bp.exe
setlimit cpu 5 locks 100 rowssel 250;
```

複数の規則がアプリケーションに適用される場合、そのすべてが適用されます。通常、最初に検出された規則制限に関連付けられた処置が最初に適用される処置となります。例外は、規則内の節に -1 を指定した場合です。この場合には、後続する規則の節の値は、それより前に*同じ* 節に指定された値のみをオーバーライドします。このとき、前に置かれた規則の他の節は、有効なままになります。例えば、ある規則が rowssel 100000 uowtime 3600 節を使用して、経過時間が 1 時間を超えた場合、または選択された行が 100 000 行を超えた場合には、そのアプリケーションの優先順位を低くするように指定しているとします。また、後続する規則では uowtime -1 節を使用して、同じアプリケーションに無制限の経過時間を許可するように指定しているとします。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、その優先順位は変更されません。つまり、 uowtime -1 が uowtime 3600 をオーバーライドするということです。ただし、アプリケーションが 100 000 行を超える行を選択した場合は、rowssel 100000 が有効なままであるために、優先順位が下げられます。

規則適用の順序

ガバナーは、構成ファイル内の規則をファイルの最初から最後まで処理します。ただし、後の規則の setlimit 節が前の規則よりも緩やかな場合は、より制限的な規則が適用されます。たとえば、以下の構成ファイルでは、後の規則にかかわらず、admin は 5000 行に制限されます。最初の規則の方がより制限的であるためです。

```
desc "Force anyone selecting 5000 or more rows"
setlimit rowssel 5000 action force;

desc "Allow user admin to select more rows"
authid admin
setlimit rowssel 10000 action force;
```

緩やかな規則で、ファイル内の先の部分に出てくるより制限的な規則をオーバーライドするには、-1 オプションを指定して、新しい規則を適用する前に、前の規則をクリアします。たとえば、以下の構成ファイルでは、最初の規則がすべてのユーザーを 5000 行に制限します。 2 番目の規則は admin に対するこの規則をクリアし、 3 番目の規則は admin の制限を 10000 行にリセットします。

```
desc "Force anyone selecting 5000 or more rows"
setlimit rowssel 5000 action force;

desc "Clear the rowssel limit for admin"
```

```
authid admin
setlimit rowssel -1;

desc "Now set the higher rowssel limit for admin"
authid admin
setlimit rowssel 10000 action force;
```

ガバナーの規則エレメント

ガバナー構成ファイル内の各規則は、規則の適用に関する条件、および規則が真と評価される場合に取られる処置を指定する節から構成されています。節は、以下に説明する順序で指定する必要があります。節の記述において、[ ] はオプションの節であることを示します。

オプションの先頭エレメント

[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。管理する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。

   ガバナー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。

制限節

setlimit

ガバナーが検査する制限を 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。ガバナーが検査できるのは、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、ガバナーはアプリケーションの CPU 使用の制限は行いません。

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、ガバナー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、コーディネーター・ノードでは非ゼロとなります。 -1 を指定すると、ガバナーは選択できる行数の制限は行いません。 *nnn* に指定できる最大値は、4 294 967 298 です。

uowtime *nnn*
作業単位 (UOW) が最初にアクティブ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モニター・スイッチまたはタイム・スタンプ・モニター・スイッチを非活動化した場合には、ガバナー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。ガバナー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、ガバナーはそれ以上この情報を受け取りません。

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。

注: バックアップおよびリストアなどの一部のデータベース・ユーティリティーでは、データベースへの接続を確立し、次いでガバナーからは可視でない作業を EDU により実行します。これらのデータベース接続はアイドルと表示され、アイドル時間制限を超過する場合があります。ガバナーがこれらのユーティリティーに対してアクションを取らないようにするために、呼び出す許可 ID から、それらに -1 を指定できます。例えば、ガバナーが許可 ID の DB2SYS により実行しているユーティリティーに対してアクションを取らないようにするには、「authid DB2SYS setlimit idle -1」と指定します。

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。 *nnn* に指定できる最大値は、4 294 967 298 です。

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ます。また、索引使用時には行数が少なくなる可能性があります。

処置節

[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 節が指定されていなかった場合には、ガバナーは、アプリケーションの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。

nice *nnn*

アプリケーションの処理を行っているエージェントの相対的な優先順位の変更を指定します。有効な値は -20 から +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優先順位の値をオーバーライドしてしまいます。
- Windows 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 *優先順位* の処置とを一緒に使用することができます。

ガバナーを使用して、デフォルトのユーザー・サービス・スーパークラス、SYSDEFAULTUSERCLASS で実行するアプリケーションの優先順位を制御できます。ガバナーを使用して、このサービス・スーパークラスで実行するアプリケーションの優先順位を低くする場合、エージェントは自分自身をそのアウトバウンド相関関係子から切断し (両者が関連付けられている場合)、ガバナーによって指定されたエージェント優先順位に従ってその相対優先順位を設定します。ガバナーを使用して、ユーザー定義サービスのスーパークラスおよびサブクラスでエージェント優先順位を変更することはできません。その代わりに、スーパークラスまたはサブクラスのエージェント優先順位の設定を使用して、それらのサービス・クラスで実行するアプリケーションを制御する必要があります。一方で、ガバナーを使用してサービス・クラスでの接続を強制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。 (コーディネーター・エージェントを終了するために FORCE APPLICATION を実行します。)

注: 複数パーティション・データベース環境では、強制停止処置は、アプリケーションの調整データベース・パーティション上でガバナー・デーモンが実行している場合のみ実行されます。したがって、データベース・パーティション A 上でガバナー・デーモンが実行しているときに、調整データベース・パーティションがデータベース・パーティション B であるアプリケーションの限度を超えた場合には、強制停止処置はスキップされます。

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。

ガバナーは、次の 3 つの基準に基づいて、スケジューリングの優先度が高いアプリケーションを選択します。

- 最も多くロックを保持しているアプリケーション

  これは、ロック待機を削減しようとするために選択されます。
- 経過時間の最も長いアプリケーション
- 見積もられた残り実行時間が最も短いアプリケーション

  これは、できるだけ多くの短期間のステートメントを、インターバルの間に完了させようとするために選択されます。

各規準で上位の 3 つのアプリケーションには、他のアプリケーションよりも高い優先順位が与えられます。つまり、各基準のグループで 1 位のアプリケーションには最も高い優先順位が、その次のアプリケーションには 2 番目に高い優先順位が、そして 3 位のアプリケーションには 3 番目に高い優先順位が与えられます。単一のアプリケーションが複数の基準において 3 位以内となった場合、そのアプリケーションには最も高い順位となった基準において該当する優先順位が与えられ、他の基準においては、次に高い優先順位が次に順位の高かったアプリケーションに与えられます。たとえば、アプリケーション A は最も多くロックを保持しているが、見積もりの残り実行時間は 3 番目に短いとします。この場合、このアプリケーションには 1 つめの基準において最も高い優先順位が与えられ、見積もりの残り実行時間が 4 番目に短いアプリケーションに、その基準において 3 番目に高い優先順位が与えられます。

このガバナー規則によって選択されたアプリケーションは、 3 つのクラスに分けられます。それぞれのクラスごとに、ガバナーは上にリストした基準に基づいて、各クラスからの上位 3 つである、 9 個のアプリケーションを選択します。クラス・オプションを指定した場合、この規則によって選択されたすべてのアプリケーションが単一のクラスと見なされ、 9 個のアプリケーションが選択されて上記のように他より高い優先順位を与えられます。

複数のガバナー規則で同じアプリケーションが選択された場合、最後に選択された際の規則によって管理されます。

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、ガバナー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。ガバナー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、ガバナー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、14 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) を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、それぞれの応答時間は満足のいくものでは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、ガバナーはすべての短い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、ガバナー構成ファイルには、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、この点を例証するガバナー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 つ目のグループに、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、この点を例証するガバナー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;
```

- ガバナー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、ガバナーは、スケジュール処理に該当するアクティブ・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ついての 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々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* 節は提供されず、 *setlimit* 節も制限を課しません。

注: 制限を超えたときに action 節が指定されていなかった場合には、ガバナーは、アプリケーションの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。

ガバナー構成ファイルの例

以下の例は、アクションについていくつかの規則を設定するガバナー構成ファイルを示しています。

```
 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;
```

ガバナー・ログ・ファイル

ガバナー・デーモンは処置を実行する度に、ログ・ファイルにレコードを書き込みます。処置には以下のものが含まれます。

- アプリケーションの強制停止
- ガバナー構成ファイルの読み取り
- アプリケーション優先順位の変更
- エラーまたは警告の検出
- 開始または終了

ガバナー・デーモンには、それぞれに別個のログ・ファイルがあります。ログ・ファイルが別になっていることで、多くのガバナー・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可能性のある、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。ログ・ファイルを 1 つにマージして照会するには、 db2govlg ユーティリティーを使用します。

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。ただし、 Windows オペレーティング・システムの場合、 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 コマンドを使用してガバナーを開始するときには、ログ・ファイルの基底名を指定します。管理対象となる各データベース・パーティション用のログ・ファイルを区別するため、ログ・ファイル名には必ずデータベース名を含めてください。パーティション・データベース環境においては、各ガバナーごとにファイル名が固有になるように、ガバナー・デーモンが実行されているパーティションのデータベース・パーティション番号が、自動的にログ・ファイル名の後に付加されます。

ログ・ファイル・レコード・フォーマット

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
```

注: *Date* および *Time* フィールドの形式は、 yyyy-mm-dd hh.mm.ss です。このフィールドでソートを行うことによって各データベース・パーティションごとのログ・ファイルをマージすることができます。

NodeNum フィールドは、ガバナーが実行されているデータベース・パーティションの番号を示します。

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。

- `START`: ガバナーが開始された
- `STOP`: ガバナーが停止された
- `FORCE`: アプリケーションが強制された
- `NICE`: アプリケーションの優先順位が変更された
- `ERROR`: エラーが起きた
- `WARNING`: 警告が起きた
- `READCFG`: ガバナーが構成ファイルの読み取りを行った
- `ACCOUNT`: アプリケーションの会計統計
- `SCHEDGRP`: エージェントの優先順位に変更が生じた

これらの値の一部について、以下で詳細に説明します。

START

START レコードは、ガバナーが始動するときに書き込まれます。それは、以下のようなフォーマットになります。

```
Database = <database_name>
```

STOP STOP レコードは、ガバナーが停止するときに書き込まれます。それは、以下のようなフォーマットになります。

```
Database = <database_name>
```

FORCE

FORCE レコードは、ガバナー構成ファイル内の規則の要求に従ってアプリケーションを強制することをガバナーが判別したときに書き出されます。FORCE レコードは、以下のようなフォーマットになります。

```
<appl_name> <auth_id> <appl_id> <coord_partition> <cfg_line>
<restriction_exceeded>
```

詳細は次のとおりです。

<coord_partition>
アプリケーションの調整データベース・パーティションの番号を指定します。

<cfg_line>
アプリケーションを強制する規則が位置する、ガバナー構成ファイル内の行番号を指定します。

<restriction_exceeded>
規則超過の詳細を提供します。以下の値が有効です。

- `CPU`: アプリケーション USR cpu と SYS cpu の合計時間 (秒単位)

- Locks: アプリケーションが保持したロックの合計数
- Rowssel: アプリケーションが選択した行の合計数
- Rowsread: アプリケーションが読み取った行の合計数
- Idle: アプリケーションがアイドルだった合計時間
- ET (経過時間): アプリケーションの現行作業単位が開始した (作業単位設定限度を超えた) ときからの経過時間

NICE NICE レコードは、ガバナー構成ファイル内で指定された優先順位アクションによって、アプリケーションの優先順位が変更されるときに書き込まれます。 NICE レコードは、以下のようなフォーマットになります。

```
<appl_name> <auth_id> <appl_id> <nice value> (<cfg_line>)
<restriction_exceeded>
```

詳細は次のとおりです。

<nice value>
アプリケーションのエージェント・プロセス用の優先度の値の増分 (または減分) を指定します。

<cfg_line>
アプリケーションの優先順位を変更する規則が位置する、ガバナー構成ファイル内の行番号を指定します。

<restriction_exceeded>
規則超過の詳細を提供します。以下の値が有効です。

- CPU: アプリケーション USR cpu と SYS cpu の合計時間 (秒単位)
- Locks: アプリケーションが保持したロックの合計数
- Rowssel: アプリケーションが選択した行の合計数
- Rowsread: アプリケーションが読み取った行の合計数
- Idle: アプリケーションがアイドルだった合計時間
- ET (経過時間): アプリケーションの現行作業単位が開始した (作業単位設定限度を超えた) ときからの経過時間

ERROR
ERROR レコードは、ガバナーがシャットダウンするときに書き込まれます。

WARNING
WARNING レコードは、以下の状況でガバナー・ログに書き込まれます。

- アプリケーションを強制するために sqlefrce API が呼び出されたが、正の SQLCODE が戻された。
- スナップショット呼び出しが 1611 以外の正の SQLCODE を戻した (SQL1611 データが戻されませんでした。)。
- スナップショットが -1224 (SQL1224N データベース・マネージャーが新しい要求を受け付けることができないか、進行中のすべての要求処理を終了したか、エラーまたは強制割り込みがあったために特定の要求を終了しました。) または -1032 (SQL1032N start database manager コマンドが発

行されていません。) 以外の負の SQLCODE を戻した。これらの戻りコードは、以前にアクティブだったインスタンスがダウンしたときに生じます。

- UNIX 環境で、シグナル・ハンドラーをインストールする試みが行われたが失敗した。

ACCOUNT

ACCOUNT レコードは、以下の状況でガバナー・ログに書き込まれます。

- このアプリケーションの最後の ACCOUNT レコードが書き込まれたときから、このアプリケーションの agent_usr_cpu または agent_sys_cpu の値が変更された。
- アプリケーションがもはやアクティブでないことが分かった。

ACCOUNT レコードは、以下のようなフォーマットになります。

<auth_id> <appl_id> <applname> <connect time> <agent_usr_cpu delta> <agent_sys_cpu delta>

SCHEDGRP

SCHEDGRP レコードは、以下の場合に書き込まれます。

- アプリケーションがスケジューリング・グループに追加される場合
- アプリケーションがあるスケジューリング・グループから別のスケジューリング・グループへ移動する場合

SCHEDGRP レコードは、以下のようなフォーマットになります。

```
<appl_name> <auth_id> <appl_id> <cfg_line> <restriction_exceeded>
```

詳細は次のとおりです。

<cfg_line>
アプリケーションをスケジュールする規則が位置する、ガバナー構成ファイル内の行番号を指定します。

<restriction_exceeded>
規則超過の詳細を提供します。以下の値が有効です。

- CPU: アプリケーション USR cpu と SYS cpu の合計時間 (秒単位)
- Locks: アプリケーションが保持したロックの合計数
- Rowssel: アプリケーションが選択した行の合計数
- Rowsread: アプリケーションが読み取った行の合計数
- Idle: アプリケーションがアイドルだった合計時間
- ET (経過時間): アプリケーションの現行作業単位が開始した (作業単位設定限度を超えた) ときからの経過時間

ここ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
```

ガバナー・ログ・ファイルの照会

各ガバナー・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。単一データベース・パーティションあるいは全データベース・パーティションのログ・ファイルを、日時でソート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 19. db2govlg の構文

パラメーターは、以下のとおりです。

log-file 照会するログ・ファイル (複数も可) の基底名。

nodenum *node-num*
ガバナーが実行されているデータベース・パーティションのノード番号。

rectype *record-type*
照会するレコードのタイプ。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、ガバナー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必要があります。

第 16 章 ベンチマーク・テスト

ベンチマーク・テストは、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、現行の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場合、データベースおよびデータベース・マネージャーの構成パラメーターをチューニングすることにより、さらにパフォーマンスの改善を実現できます。また、アプリケーション・パラメーターもチューニングして、アプリケーションの要件をさらに満たすこともできます。

特定の種類の情報を得るには、さまざまなタイプのベンチマーク・テストを実行します。

- *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、同じスループット能力をより実動状態に近い条件下でテストするものです。

構成パラメーターをチューニングするベンチマークは、これらの「現実環境」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必要です。

ここで説明されるベンチマーク方式は構成パラメーターのチューニングを目的としています。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。

ベンチマーク・テスト方式

ベンチマーク・テストは、反復可能環境に基づいています。これにより、適切に比較できる結果を、同じ条件で実行した同じテストから得ることができます。

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。パフォーマンス上の問題を追及していく際に、テ

ストしている機能の有効範囲を制限する特殊化したテスト・ケースを作成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、必要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。

- 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。
- 他のアクティブ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、他にアクティブな関数やアプリケーションがシステムにありません。

  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スキューしたり、反復可能度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稼働環境に適合しています。

ベンチマークには、シナリオを作成し、その後そのシナリオで数回アプリケーションを実行し、実行ごとにキー情報を把握してください。実行ごとにキー情報を把握することは、アプリケーションとデータベースの両方のパフォーマンスを向上させる可能性のある変更を判別するために特に重要です。

ベンチマークの準備

パフォーマンスのベンチマークを開始する前に、アプリケーションがそれに対して実行するデータベースの論理設計を完了しておいてください。表、ビュー、および索引をセットアップし、データを入れておきます。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、表に現実的なデータを入れておきます。

データベースの最終的な物理設計も決めておいてください。データベース・マネージャー・オブジェクトを最終ディスク・ロケーションに置き、作業ファイルとバックアップのロケーションを判別してログ・ファイルをサイズ変更し、バックアップ手順をテストします。さらに、パッケージをチェックして、可能な場合には行ブロッキング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。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、最終的な実稼働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。

ベンチマーク・テストは実動サイズ・データベースで実行するようにしてください。個々の SQL ステートメントが戻すデータは、実動での場合と同じ量で、同じ程度にソートされるものでなければなりません。この規則により、アプリケーションが典型的なメモリー所要量をテストすることになります。

ベンチマークの対象の SQL ステートメントは、*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、平均と見なしてください。量が多過ぎる場合、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します。

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・テストの作成

ベンチマーク・プログラムを設計してインプリメントする場合は、さまざまな要因を考慮に入れてください。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は異なります。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のみ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こともできます。

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、単に 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判別するということもできます。

すべてのベンチマーク・テストにおいて、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追跡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 しか必要ではなく、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、ロッキング、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 をフルに使用するのではなく入出力制約であ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しません。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、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ようにすることもできます。画面出力のデータを形式制御すると通常は CPU の消費の多重度が高くなり、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され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関する指針の多くを使用します。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。また、このツールを使用すると、応答セットのサイズや、その応答セットから出力装置に送信される行数をコントロールすることもできます。

さらに、経過時間、CPU とバッファー・プールの使用率、ロッキング、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ができます。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。構文およびオプションには、コマンド行で db2batch -h と入力します。

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、そのパラメーターに異なるさまざまな値を使用して、最大の効果を得るまで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用してアプリケーションを何度か反復して (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がより明確になります。

ベンチマークを実行するときは、最初の反復 (ウォームアップ実行) を、後続の反復 (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。ウォームアップ実行にはバッファー・プールの初期化などのいくつかの開始アクティビティーが含まれるため、通常実行よりいくらか時間が長くかかります。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。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、通常実行の結果のみ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構成アドバイザーの使用を考慮することができます。構成アドバイザーが尋ねる質問により、ベンチマーク・アクティビティーにおいて通常実行の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。構成アドバイザーは、コントロール・センターからも開始できますし、適切なオプションを使用した db2 autoconfigure コマンドを実行しても開始できます。

ベンチマークが個々の照会を使用する場合、バッファー・プールをフラッシュして、直前の照会の潜在的な影響が最小限になるようにしてください。バッファー・プールをフラッシュするには、照会とは関係のない多数のページを読み取り、それでバッファー・プールを満たします。

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リストアされるようにしてください。

- . 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。

- テストで使用されるデータがテストによって更新される場合、それらは一貫性のあるものにし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リストアします。データベースのバックアップ・コピーはその直前の状態を含み、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・コピーをリストアします。この方法では、影響を受けたデータだけをリストアできます。 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元の状態に戻すには、それをデータベースに再バインドします。

以下に、データベース・アプリケーションのベンチマークを実行するために行う、ステップまたは反復のサマリーを示します。

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。

- テストのワークロード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。
- 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、曲線が停滞す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 Linux および 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リストアし、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。

ベンチマーク・テストの分析例

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サマリーは、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 20. 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**。

これを分析すると、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) が 7 行を戻してその最大の遅延が .28 秒であり、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 秒かかったことが分かります。

使用するプログラムが区切り付き ASCII フォーマットでデータを出力できる場合、後でそれをデータベース表やスプレッドシートにインポートし、さらに統計分析を行うことができます。

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
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 21. 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**。

第 17 章 設計アドバイザー

DB2 設計アドバイザーは、ワークロード・パフォーマンスを大幅に向上させるのに役立つツールです。複雑なワークロードの場合、索引、MQT、クラスタリング・ディメンション、またはデータベース・パーティションを選択することは、非常に困難なことがあります。設計アドバイザーは、ワークロードのパフォーマンスを改善するのに必要なすべてのオブジェクトを識別します。設計アドバイザーは、ワークロードの SQL ステートメントのセットを考慮に入れ、以下の推奨を生成します。

- 新規索引
- 新規マテリアライズ照会表 (MQT)
- クラスター索引の追加の判別
- マルチディメンション・クラスタリング (MDC) 表への変換
- 表の再配分
- 指定されたワークロードが使用していない索引および MQT の削除 (GUI ツールから)

こうした推奨のすべて、またはその一部を、設計アドバイザーにただちにインプリメントさせることができます。あるいは後でインプリメントするようスケジュールすることもできます。

設計アドバイザーの GUI またはコマンド行ツールのいずれかを使用することにより、設計アドバイザーで以下のタスクを簡単に行えます。

新規データベースの計画またはセットアップ

データベースの設計の際に、以下の目的で設計アドバイザーを使用します。

- パーティション・データベース環境のテスト環境、および索引、MQT、MDC 表のテスト環境に、代替の設計を生成する。
- パーティション・データベース環境では、次の目的で設計アドバイザーを使用できる。
  - データベースにデータをロードする前に、データベース・パーティション化ストラテジーを決定する。
  - 単一パーティション DB2 データベースから複数パーティション DB2 データベースへの移行を支援する。
  - 別のデータベース製品から複数パーティション DB2 データベースへの移行を支援する。
- 手動で生成した 索引、MQT、MDC 表、またはデータベース・パーティション化ストラテジーを評価する。

ワークロード・パフォーマンスのチューニング

データベースをセットアップした後、次の目的で設計アドバイザーを使用できます。

- 特定のステートメントまたはワークロードのパフォーマンスを改善する。
- サンプル・ワークロードのパフォーマンスを基準に、一般的なデータベース・パフォーマンスを向上させる。

- アクティビティー・モニターによって識別される最も頻繁に実行される照会のパフォーマンスを向上させる。
- 新しいキー照会のパフォーマンスを最適化する方法を判別する。
- 共用メモリー・ユーティリティーに関するヘルス・センターからの推奨、またはソートを集中的に行うワークロードのソート・ヒープの問題に関するヘルス・センターからの推奨にこたえる。
- ワークロードで使用していないオブジェクトを検索する。

設計アドバイザーの出力

設計アドバイザー GUI を使用する場合、設計アドバイザーからの推奨を表示、保管、またはインプリメントできます。コマンド行ツールから設計アドバイザーを実行する場合、出力はデフォルトでは標準出力に印刷され、 ADVISE_TABLE および ADVISE_INDEX 表に保管されます。

- ADVISE_TABLE には、MQT、MDC 表、およびデータベース・パーティション化ストラテジーの推奨に関して、USE_TABLE='Y' が入っています。

  MQT 推奨は ADVISE_MQT 表にもあります。MDC 推奨は ADVISE_TABLE 表にもあります。データベース・パーティション化ストラテジー推奨は ADVISE_PARTITION 表にもあります。これらの表の RUN_ID 値は、設計アドバイザーの毎回の実行ごとに ADVISE_INSTANCE 表に記録される START_TIME 値に対応しています。
- ADVISE_INDEX 表には、索引の推奨に関して USE_INDEX='Y' または 'R' が入ります。

また ADVISE_INSTANCE 表は、設計アドバイザーが実行される度に、 1 行が更新されます。

- START_TIME フィールドおよび END_TIME フィールドは、それぞれユーティリティーの開始時間と停止時間を示します。
- ユーティリティーが正常に終了すると、STATUS フィールドに 'COMPLETED' が入ります。
- MODE フィールドは、-m オプションが使用されたかどうかを示します。
- COMPRESSION フィールドは、圧縮タイプが使用されたかどうかを示します。

-o オプションを使用すると、設計アドバイザーの推奨をファイルに保管できます。保管される設計アドバイザー出力は、以下のエレメントで構成されています。

- 新規索引、MQT、データベース・パーティション化ストラテジー、および MDC 表のための CREATE STATEMENTS。
- MQT の REFRESH ステートメント。
- 新規オブジェクトの RUNSTATS コマンド。
- 既存の MQT と索引があって、これをワークロードの実行に使用する場合は、推奨されるスクリプトにそれらの MQT と索引が示されます。

  注: ADVISE_MQT 表の COLSTATS 列には、MQT の列統計が含まれます。統計は、XML 構造です。以下のとおりです。

```
<?xml version=¥"1.0¥" encoding=¥"USASCII¥"?>
<colstats>
        <column>
         <name>COLNAME1</name>
         <colcard>1000</colcard>
         <high2key>999</high2key>
         <low2key>2</low2key>
    </column>
    ....

    <column>
         <name>COLNAME100</name>
         <colcard>55000</colcard>
         <high2key>49999</high2key>
         <low2key>100</low2key>
    </column>
</colstats>
```

索引が定義されている基本表も追加されます。索引および MQT のランキングは利点値を使用して行えます。また、利点 - 0.5*オーバーヘッドを使用する MQT と、利点 - オーバーヘッドを使用する索引のランク付けを行うこともできます。索引および MQT のリストの後に、ワークロード内のステートメントのリストが続きます。これには、SQL テキスト、ステートメントのステートメント番号、推奨値から推定されるパフォーマンスの改善 (利点)、ならびにステートメントによって使用された表、索引、および MQT のリストが含まれます。

注: SQL テキストでの元のスペーシングはこの出力で保持されていますが、SQL テキストは読みやすくするための 80 文字のコメント行に分割されています。MDC およびパーティション化については、この XML 出力で明示的に示されていません。この出力の例は以下のとおりです。

```
--<?xml version="1.0"?>
--<design-advisor>
--<mqt>
--<identifier>
--<name>MQT612152202220000</name>
--<schema>ZILIO2 </schema>
--</identifier>
--<statementlist>3</statementlist>
--<benefit>1013562.481682</benefit>
--<overhead>1468328.200000</overhead>
--<diskspace>0.004906</diskspace>
--</mqt>
.....
--<index>
--<identifier>
--<name>IDX612152221400000</name>
--<schema>ZILIO2 </schema>
--</identifier>
--<table><identifier>
--<name>PART</name>
--<schema>TPCD </schema>
--</identifier></table>
--<statementlist>22</statementlist>
--<benefit>820160.000000</benefit>
--<overhead>0.000000</overhead>
--<diskspace>9.063500</diskspace>
--</index>
.....
--<statement>
--<statementnum>11</statementnum>
--<statementtext>
--
```

```
-- select
-- c_name, c_custkey, o_orderkey, o_orderdate, o_totalprice,
-- sum(l_quantity) from tpcd.customer, tpcd.orders,
-- tpcd.lineitem where o_orderkey in( select
-- l_orderkey from tpcd.lineitem group by l_orderkey
-- having sum(l_quantity) > 300 ) and c_custkey
-- = o_custkey and o_orderkey = l_orderkey group by
-- c_name, c_custkey, o_orderkey, o_orderdate, o_totalprice
-- order by o_totalprice desc, o_orderdate fetch first
-- 100 rows only
--</statementtext>
--<objects>
--<identifier>
--<name>MQT612152202490000</name>
--<schema>ZILIO2 </schema>
--</identifier>
--<identifier>
--<name>ORDERS</name>
--<schema>TPCD </schema>
--</identifier>
--<identifier>
--<name>CUSTOMER</name>
--<schema>TPCD </schema>
--</identifier>
--<identifier>
--<name>IDX612152235020000</name>
--<schema>ZILIO2 </schema>
--</identifier>
--<identifier>
--<name>IDX612152235030000</name>
--<schema>ZILIO2 </schema>
--</identifier>
--<identifier>
--<name>IDX612152211360000</name>
--<schema>ZILIO2 </schema>
--</identifier>
--</objects>
--<benefit>2091459.000000</benefit>
--<frequency>1</frequency>
--</statement>
```

XML 構造には複数の列を含めることができます。各列には、列カーディナリティー (つまり、列内の値の数) が示され、オプションで high2 および low2 キーが示されます。

MQT、データベース・パーティション、または MDC 推奨が提供されている場合、関連する ALTER TABLE ストアード・プロシージャー呼び出しコマンドが ADVISE_TABLE の ALTER_COMMAND 列に置かれます。

注: ALTER TABLE ストアード・プロシージャー呼び出しコマンドは、ALTOBJ ストアード・プロシージャーに関する表の制限のために成功しない場合があります。

若干の変更を加えたら、この出力ファイルを CLP スクリプトとして実行して、推奨オブジェクトを作成できます。実行できる変更には、次のものが含まれます。

- すべての RUNSTATS コマンド・ステートメントをまとめて、新規または変更オブジェクトに対する単一の RUNSTATS 呼び出しにする。
- システム生成 ID ではなく、もっと使用しやすいオブジェクト名を指定する。
- すぐにインプリメントしないオブジェクトの DDL を、除去またはコメント化する。

設計アドバイザーの使用

設計アドバイザーの実行は、コマンド行またはコントロール・センターの設計アドバイザー GUI から行えます。

- **コントロール・センターを使う方法:**
  1. 『設計アドバイザーのワークロードの定義』.
  2. コントロール・センターをオープンします。
  3. 該当するデータベースまたはデータベース内の特定の索引を右クリックし、ポップアップ・メニューから設計アドバイザーを選択することにより、コントロール・センターから設計アドバイザーを実行します。
  4. 設計アドバイザーが推奨の生成を終了した後、それらの推奨をレポートに保存するか、または設計アドバイザーにより推奨の一部またはすべてをインプリメントすることができます。
- **コマンド行を使う方法:**
  1. 『設計アドバイザーのワークロードの定義』.
  2. ワークロードに対して db2advis コマンドを実行します。

     注: データベース上の統計が最新のものではない場合、生成される推奨の信頼性が低下します。
  3. db2advis からの出力を解釈し、必要な変更を行います。
  4. 選択された設計アドバイザーの推奨を実際に設定します。

設計アドバイザーのワークロードの定義

ワークロード とは、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 ステートメントのことです。たとえば、1 カ月の間にデータベース・マネージャーが INSERT を 1000 行、 UPDATE を 10000 行、SELECT を 10000 行、および DELETE を 1000 行処理する必要があるとします。設計アドバイザーは指定されたワークロードを分析し、ワークロード・ステートメントのタイプ、特定のステートメントが使用される頻度、およびデータベースの特性などの要素を考慮して、ワークロードの実行に要するトータル・コストを最小化するための推奨値を生成します。

「設計アドバイザー GUI ワークロード (Design Advisor GUI workload)」ページから、新規のワークロード・ファイルを作成するか、または既存のワークロード・ファイルを変更できます。以下の複数のソースからファイルにステートメントをインポートできます。

- 区切り文字で区切られているテキスト・ファイル
- イベント・モニター表
- Query Patroller 履歴データ表 (コマンド行から -qp オプションを使用する)
- EXPLAINED_STATEMENT 表で EXPLAIN されているステートメント
- DB2 スナップショットでキャプチャーした最新の SQL ステートメント
- ワークロード・マネージャー・アクティビティー表

- ワークロード・マネージャー・イベント・モニター表 (コマンド行から -wlm オプションを使用する)

SQL ステートメントをインポートした後で、ステートメントの追加・変更・修正・削除、およびステートメントの頻度の変更を行えます。

コマンド行から、以下を使用して設計アドバイザーを実行します。

- インラインでコマンドを入力した、単一の SQL ステートメント
- DB2 スナップショットでキャプチャーした動的 SQL ステートメントのセット
- ワークロード・ファイルに組み込んだ SQL ステートメントのセット
- 動的 SQL ステートメントから設計アドバイザーを実行するには、以下のようにします。
  1. 次のコマンドを実行して、データベース・モニターをリセットします。

     ```
     db2 reset monitor for database database-name
     ```

  2. データベースに対する動的 SQL ステートメントを実行するまで、しばらく待機します。
  3. db2advis コマンドに -g オプションを指定して発行します。後で参照できるように、動的 SQL ステートメントを ADVISE_WORKLOAD 表に保管する場合は、 -p オプションも使用します。
- ワークロード・ファイルに含まれている SQL ステートメントから設計アドバイザーを実行するには、以下のようにします。
  1. 手動でワークロード・ファイルを作成するか (各 SQL ステートメントをセミコロンで区切る)、または前述のソース (複数可) から SQL ステートメントをインポートします。
  2. ワークロード内のステートメントの頻度を設定します。ワークロード・ファイル内の各ステートメントには、デフォルトで頻度 1 が割り当てられています。 SQL ステートメントの頻度は、そのステートメントがワークロード内に出現する回数を表しますが、この回数は他のステートメントが出現する回数との相対値で示されています。たとえば、ある SELECT ステートメントがワークロード内に 100 回出現し、別の SELECT ステートメントが 10 回出現するとします。これら 2 つのステートメントの相対頻度を表すために、最初の SELECT ステートメントに頻度 10 を割り当て、 2 番目の SELECT ステートメントの頻度を 1 にします。ワークロード内の特定のステートメントの頻度または重みを手動で変更することも可能で、そのステートメントの次の行に - - # SET FREQUENCY *n* を挿入します。 *n* はステートメントに割り当てる頻度値です。
  3. db2advis コマンドに -i オプションとワークロード・ファイルの名前を指定して実行します。
- ADVISE_WORKLOAD 表を含むワークロードから設計アドバイザーを実行するには、 db2advis に -w オプションとワークロード名を指定して実行します。

設計アドバイザーを使用した、単一パーティション・データベースから複数パーティション・データベースへのマイグレーション

単一パーティション・データベースから複数パーティション・データベースへのマイグレーションの際に、設計アドバイザーを役立てることができます。設計アドバイザーは、データの分散についての勧告だけでなく、新規の索引、マテリアライズ照会表 (MQT)、およびマルチディメンション・クラスタリング (MDC) 表についての勧告も提供します。

1. db2licm コマンドの使用による DB2 製品またはフィーチャー・ライセンス・キーの登録
2. 複数パーティション・データベースのパーティション・グループに少なくとも 1 つの表スペースを作成します。

   注: 設計アドバイザーは既存の表スペースへのデータの再分散のみを推奨する可能性があるため、設計アドバイザーを実行する前に、設計アドバイザーの考慮対象にする表スペースがパーティション化されたデータベース内にあるようにしてください。
3. データベース・パーティション化機能を設計アドバイザー GUI で選択するか、または db2advis コマンドにパーティション化オプションを指定して、設計アドバイザーを実行します。
4. コントロール・センターで設計アドバイザーを使用している場合は、データベース・パーティション化の推奨値を自動的にインプリメントできます。db2advis コマンドを使用している場合は、 db2advis 出力ファイルを若干変更してから、設計アドバイザーによって生成された DDL ステートメントを実行します。

設計アドバイザーの制限と制約事項

1. **索引に関する推奨事項の制限**
   - マテリアライズ照会表 (MQT) 上で索引の使用が推奨されているのは、REFRESH TABLE のパフォーマンスではなくワークロードのパフォーマンスを改善するためです。また、更新、挿入、または削除がワークロードに含まれていない場合、 MQT を変更する (たとえば、更新する) パフォーマンスには、 IMMEDIATE MQT に関するものは含まれません。
   - マルチディメンション・クラスタリングを選択する場合には、クラスタリング RID 索引のみをお勧めします。アドバイザーは、表の MDC 構造を作成するのではなく、オプションとして RID クラスタリング索引をオプションとして組み込みます。
2. **MQT に関する推奨事項の制限**
   - 設計アドバイザーはインクリメンタル MQT を推奨しません。インクリメンタル MQT を作成する場合は、 REFRESH IMMEDIATE MQT を取り、選択したステージング表でインクリメンタルに変換します。
   - MQT 用に推奨される索引は、ワークロードのパフォーマンスを改善するためのものであり、 MQT リフレッシュのパフォーマンスを改善するためのものではありません。
   - 指定したワークロードに更新、挿入、または削除が組み込まれていない場合、推奨されている REFRESH IMMEDIATE MQT の更新がパフォーマンスに及

ぼす影響は考慮されません。暗黙指定されたユニーク・キー上で作成されたユニーク索引を REFRESH IMMEDIATE MQT に含めることをお勧めします。暗黙指定されたユニーク・キーは、 MQT 照会定義の GROUP BY 節にある列で構成されます。

3. **MDC に関する推奨事項の制限**
   - 既存の表には代表的なデータのセットを取り込む必要があります。そうでない場合、設計アドバイザーは表の MDC を考慮しません。最小でも 24 から 36 MB のデータを推奨します。 12 エクステントより小さい表は必ず除外されます。
   - サンプリング・オプション -r がコマンドに指定されていない場合、または MQT サンプリングが GUI ツールで選択されていない場合は、新規 MQT に関する MDC の推奨事項は考慮されません。
   - 設計アドバイザーは、型付き表または一時表に対しては MDC に関する推奨を行いません。
   - 設計アドバイザーは、フェデレーテッド表に対しては MDC に関する推奨を行いません。
   - 設計アドバイザーの実行中に使用するサンプリング・データ用にストレージ・スペースが必要です (大規模な表の表データの約 1 %)。このスペースが存在しない場合は、サンプリング済みの表は基本列に対してのみ、無相関である前提事項に基づいて検査されます。このケースでは、警告メッセージが生成されます。
   - 収集された統計のない表は、スキップして考慮対象から外されます。
   - 設計アドバイザーは複数列のディメンションに対しては推奨を行いません。
   - MDC 選択でサンプリングが実行されるようにするため、既存の表にはデータが含まれていなければなりません。
4. **データベース・パーティション化に関する推奨事項の制限**

   設計アドバイザーは、DB2 Enterprise Server Edition でのデータベース・パーティション化のみを推奨します。db2advis コマンドにデータベース・パーティション化オプションを指定すると、エラーが戻されます。設計アドバイザー GUI では、単一パーティション・データベース環境でデータベース・パーティション化機能を選択することはできません。
5. **追加の制限**

   シミュレーション・カタログ表は、設計アドバイザーの実行中に作成されます。設計アドバイザーの実行が完了すると、この表はドロップされます。設計アドバイザーの実行が不完全であると、一部のシミュレーション・カタログ表がドロップされない場合があります。このとき、コマンド行からユーティリティーを再始動することによって、設計アドバイザーを使用してシミュレーション・カタログ表をドロップできます。シミュレーション・カタログ表を除去するには、 -f オプションと -n オプションの両方を指定します (-n オプションには、不完全な実行で使用されたのと同じユーザー名を指定します)。 -f オプションを指定しない場合は、設計アドバイザーは表の除去に必要な DROP ステートメントを生成します。

   注: バージョン 9.5 の時点で、-f オプションはデフォルトです。これは、MQT を選択して db2advis を実行する場合、データベース・マネージャーはスキーマ

名と同じユーザー ID を使用して、自動的にすべてのローカル・シミュレーション・カタログ表をドロップすることを意味します。

これらシミュレーション済みのカタログ表を保管するために別個の表スペースを作成し、 DROP TABLE RECOVERY を ″OFF″ に設定してください。こうすることにより、クリーンアップは容易になり、設計アドバイザーの実行はより高速になります。シミュレーション・カタログ表の個別の表スペースは、カタログ・ノードでのみ定義しなければなりません。

第 5 部 データベース・アプリケーション・パフォーマンスのチューニング

第 18 章 アプリケーションに関する考慮事項

並行性の問題

多数のユーザーがリレーショナル・データベース内のデータにアクセスしたり、変更したりするため、データベース・マネージャーは、ユーザーがこれらの変更を行い、かつデータ保全性が保持されていることを確認できるようにし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します。
  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。

     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 (「幻像読み取り行」) が結果表の一部として戻されます。

注: 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。

フェデレーテッド・データベース・システム内での並行性の制御

フェデレーテッド・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 ステートメントをサブミットできます。データ・ソース (DBMS およびデータから成る) を参照するには、 DB2 はニックネーム を使用します。 ニックネームは、その他のデータベース・マネージャー内でのオブジェクトの別名です。フェデレーテッド・システムでは、 DB2 は、要求されたデータのホストとなるデータベース・マネージャーの並行性制御プロトコルに依存しています。

DB2 フェデレーテッド・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、表およびビューについての情報が移動する場合に、ニックネームによるその情報への参照を、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よってデータにアクセスすると、 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。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。

分離レベルとパフォーマンス

分離レベル は、データがアクセスされている間、データが他のプロセス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 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。 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

注: 一部のホスト・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します。その他のデータベースでは、この分離レベルは非コミット読み取り分離レベルに似た動作をします。

これ以降では、それぞれの分離レベルの詳細について、パフォーマンスへの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幻像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。適正にロックをかけるのは、ある特定の行が照会で参照されると仮定し (その行は行のリストに追加される)、その照会が再び実行された場合に、別のアプリケーションがその行を挿入または更新できないようにするためです。これにより、幻像読み取り行が発生しなくなります。例えば、10 000 個の行をスキャンしてそれらに述部を適用する場合、たとえ 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、戻されるデータすべてをアプリケーションが確認するまでは、一時表や行ブロッキングが使用されている場合であっても、それらのデータはすべて未変更のままになります。

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。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、オプティマイザーは索引のスキャン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。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。

参照制約を評価する際、ユーザーが設定した分離レベルに関係なく DB2 によって、子表のスキャン時に使用される分離レベルが反復可能読み取り (RR) へと内部的にアップグレードされることがまれにあります。これが起こるとさらに多くのロックがコミットまで保持されることになるため、デッドロックやロックのタイムアウトが発生する可能性が高くなります。これを回避するために、外部キーの列のみを含む索引を作成することをお勧めします。これにより、RI スキャンでこの索引が使用されることになります。

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**不可能**になります。

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*幻像読み取り* 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 個の行をスキャンする例を考えると、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。

注: 読み取り固定分離レベルを使う場合、一時表や行ブロッキング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、オプティマイザーは、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。

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 行をスキャンする例を考えてみると、カーソル固定を使う場合は、現行カーソル位置の行だけにロックがかかります。ロックは、カーソルが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する場合に、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トランザク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ドロップ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ドロップされている表、ビュー、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 を使用してプログラムを実行するとき、アプリケーションは分離レベル CS を使用できます。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 にエスカレート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 にエスカレーションされることを意味します。このエスカレーションを防ぐには、以下の 2 つの選択肢があります。

- 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します。 SELECT ステートメントを変更して、FOR READ ONLY 節を組み込みます。
- アプリケーション・プログラム内でカーソルを未確定のままにし、 BLOCKING ALL および STATICREADONLY YES オプションを使ってプログラムをプリコンパイルまたはバインドします。これにより、プログラム実行時に未確定カーソルはすべて読み取り専用として扱われます。

10 000 行をスキャンする反復可能読み取りの例の場合と同様に、非コミット読み取りを使用すると、行ロックは獲得されません。

非コミット読み取りを使用すると、反復不能読み取り動作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、読み取り専用表での照会に対して、または 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。

分離レベルのまとめ

以下の表には、さまざまな分離レベルの望ましくない影響が要約されています。

表 4. 分離レベルのまとめ

分離レベル	コミットしていない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

以下の表には、アプリケーションの初期分離レベルを決定するのに役立つ簡単な発見的手法が示されています。この表はあくまで参考用です。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に従えば、他の分離レベルの方が適切である場合もあります。

表 5. 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
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、データがアクセスされている間、データが他のプロセスからどのようにロックされ分離されるかを決めるものなので、並行性の要件とデータ保全性の要件のバランスを取る分離レベルを選択する必要があります。指定する分離レベルは、作業単位の持続期間中で有効です。

注: 分離レベルは ステートメント・レベルで XQuery ステートメントに指定できません。

分離レベルはいくつかの異なった方法で指定することができます。どの分離レベルが SQL または XQuery ステートメントのコンパイルに使用されるかを判別するために、以下のヒューリスティックが使用されています。

静的 SQL

- ステートメントに分離節が指定されている場合には、その節の値が使用されます。
- ステートメントで指定されている分離節がない場合には、使用される分離レベルは、パッケージがデータベースにバインドされた際にそのパッケージ用に指定されたレベルです。

動的 SQL

- ステートメントに分離節が指定されている場合には、その節の値が使用されます。
- ステートメントで指定されている分離節がなく、 SET CURRENT ISOLATION ステートメントが現行セッション中に発行されている場合には、 CURRENT ISOLATION 特殊レジスターの値が使用されます。
- ステートメントで指定されている分離節がなく、SET CURRENT ISOLATION ステートメントが現行セッション中に発行されていない場合には、使用される分離レベルは、パッケージがデータベースにバインドされた際にそのパッケージ用に指定されたレベルです。

静的または動的 XQuery ステートメントの場合:

- XQuery 式が評価される際に、環境の分離レベルによって分離レベルが決まります。

注: 商業用に作成された数多くのアプリケーションでは、分離レベルを選択する方式を提供しています。詳細については、アプリケーション資料を参照してください。

分離レベルを指定するには、以下のようにします。

- **プリコンパイル時またはバインド時:**

  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。 PREP または BIND API を使用して、分離レベルを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。

  注: 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。

- **REXX™ をサポートするデータベース・サーバーの場合:**

  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。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。

  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。

  REXX アプリケーションが使用している分離レベルを確認するには、 SQLISL REXX 変数の値を調べてください。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。

- **ステートメント・レベルの場合:**

  WITH 節を使用します。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。

  以下の SQL ステートメントの分離レベルを指定することができます。

  – SELECT

- SELECT INTO
- Searched (検索条件付き) DELETE
- INSERT
- Searched (検索条件付き) UPDATE
- DECLARE CURSOR

以下の条件が、ステートメントに指定された分離レベルに適用されます。

- WITH 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。その他の場合には、ステートメントは UR から CS に自動的に変更されます。

- **実行時の CLI または実行時の ODBC からの場合:**

  CHANGE ISOLATION LEVEL コマンドを使用します。DB2 コール・レベル・インターフェース (DB2 コール・レベル・インターフェース) の場合は、 DB2 コール・レベル・インターフェース 構成の一部として分離レベルを変更できます。実行時に、*SQLSetConnectAttr* 関数を SQL_ATTR_TXN_ISOLATION 属性と共に使用し、 *ConnectionHandle* が参照する現行接続のトランザクション分離レベルを設定してください。 db2cli.ini ファイル内で TXNISOLATION キーワードを使用することもできます。

- **実行時に JDBC または SQLJ で作業する場合:**

  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。

  SQLJ では、SQLJ プロファイル・カスタマイザー (db2sqljcustomize コマンド) を使用してパッケージを作成します。 このパッケージに指定できるオプションには、その分離レベルが含まれます。

- **現行セッション内での動的 SQL 用**

  SET CURRENT ISOLATION ステートメントを使用して、セッション内で発行される動的 SQL のために、分離レベルを設定します。このステートメントを発行すると、CURRENT ISOLATION 特殊レジスターは、現行セッション内で発行されるすべての動的 SQL の分離レベルを指定する値に設定されます。いったん設定されると、CURRENT ISOLATION 特殊レジスターは、どのパッケージがステートメントを発行したかに関係なく、そのセッション内でコンパイルされる後続のすべての動的 SQL ステートメントにその分離レベルを提供します。この分離レベルは、セッションが終了するまで、または SET CURRENT ISOLATION ステートメントが RESET オプションと共に発行されるまで、適用されます。

ロックおよび並行性の制御

並行性制御を提供し、制御されていないデータへのアクセスを回避するために、データベース・マネージャーは、バッファー・プール、表、データ・パーティション、表ブロック、または表行をロックします。*ロック* は、データベース・マネージャー・リソースを *lock owner* というアプリケーションに関連付け、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。

データベース・マネージャーは、以下に基づいて行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。分離レベルは以下のいずれかです。
  - 非コミット読み取り (UR)
  - カーソル固定 (CS)
  - 読み取り固定 (RS)
  - 反復可能読み取り (RR)

  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。パフォーマンスの影響を最小化するには、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。
- オプティマイザーにより選択されるアクセス・プラン。表スキャン、索引スキャン、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 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)、TABLE (表ロックの場合)、または BLOCKINSERT (MDC 表のみのブロック・ロックの場合) を選択できます。 MDC 表で BLOCKINSERT 節が使用される場合、ブロック・レベルのロックが行われる INSERT 操作を除いて、行レベルのロックが実行されます。トランザクションが非結合セルに大量の挿入を実行する場合、MDC 表には ALTER TABLE ... LOCKSIZE BLOCKINSERT ステートメントを使用してください。読み取り専用の表には、ALTER TABLE ... LOCKSIZE TABLE ステートメント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。パーティション表では、表ロックがまず獲得され、次にアクセスされたデータの命令に従ってデータ・パーティション・ロックが獲得されます。
- 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、 `locklist` か `maxlocks` のいずれか、あるいはその両方の値を増やしてください。また、同時に保持されるロックの数を減らすには、頻繁にトランザクションが COMMIT を実行して、保持されたロックを解放するようにします。

たいていのロックは表や行に対するものですが、バッファー・プールの作成、変更、またはドロップ時には、バッファー・プールのロックが設定されます。このロックで使用されるモードは EXCLUSIVE (X) です。システムがモニターするデータの収集時にこのタイプのロックを検出することがあります。このスナップショットを表示すると、使用されているロック名がバッファー・プールそのものの ID であることが分かります。

通常、以下の場合以外は、行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、述部なしのスキャンが必要な場合。
- 表の LOCKSIZE 属性が「TABLE」である場合。
- ロック・リストが満杯で、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。

これが MDC 表である場合、行レベルのロックの代わりにブロック・レベルのロックが使用される場合があります。それには以下が含まれます。

- 表の LOCKSIZE 属性が 『BLOCKINSERT』 である場合。
- 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・プランに述部が関係する場合。
- 探索型の更新または削除操作にディメンション列のみが関係する場合。

ロック・エスカレーションは並行性を低下させます。 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。

行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。

- UR スキャン。行データが変更されていない限り行ロックは保持されません。
- CS スキャン。カーソルが行に位置している場合にのみ行ロックが保持されます。
- RS スキャン。トランザクション中、限定した行ロックのみ保持されます。
- RR スキャン。トランザクションの間、すべての行ロックが保持されます。

ロック属性

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。明示的にロックできるオブジェクトの唯一のタイプは表です。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。 マルチディメンション・クラスタリング (MDC) 表の場合には、ブロック・ロックをかけることもできます。パーティション表の場合は、データ・パーティション・ロックをかけることができます。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。照会が実行される分離レベルは、ロック期間に影響を与えます。

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。様々なレベルでのロックに関する詳細については、ロック・モード参照表を参照してください。

表 6. ロック・モードのサマリー

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、ブロック、表、データ・パーティション	ロックの所有者は、非コミット・データを含め、オブジェクト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。 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 (意図的共有)	表スペース、ブロック、表、データ・パーティション	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。 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 (次キー共有)	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。 NS ロック・モードは、次キー・ロッキングには使用されません。このモードは、これらのスキャンにおける次キー・ロッキングの影響を最小化するために CS および RS スキャンの際に S モードの代わりに使用されます。
S (共有)	行、ブロック、表、データ・パーティション	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。
IX (意図的排他)	表スペース、ブロック、表、データ・パーティション	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、データを読み取ったり更新したりできます。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 (意図的排他共有)	表、ブロック、データ・パーティション	ロック所有者は、データを読み取ったり更新したりできます。 同時に実行される他のアプリケーションは、その表を読み取ることができます。
U (更新)	行、ブロック、表、データ・パーティション	ロック所有者は、データを更新できます。他の作業単位はロックされたオブジェクト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。
NW (次キーの弱い排他)	行	行が索引に挿入されるとき、NW ロックが次の行に獲得されます。タイプ 2 索引の場合、次の行が現在 RR スキャンによってロックされている場合にのみ、これが発生します。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。このロック・モードは、W および 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。
X (排他)	行、ブロック、表、バッファー・プール、データ・パーティション	ロック所有者は、ロックされたオブジェクトのデータを読み取ったり更新したりできます。ロックされたオブジェクト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 (弱い排他)	行	このロックは、タイプ 2 索引が定義されていない表に行を挿入するときに、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。重複値が見つかるときに重複値がコミットされたかどうかを判別するために、ユニーク索引に挿入中にもこのロックが使用されます。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。

表 6. ロック・モードのサマリー (続き)

ロック・モード	適用できるオブジェクト・タイプ	説明
Z (超排他)	表スペース、表、データ・パーティション	このロックが表上で獲得されるのは、表が変更またはドロップされるとき、表の索引が作成またはドロップされるとき、あるいは表の一部のタイプ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。

ロックの細分性

1 つのアプリケーションがデータベース・オブジェクト上のロックを保持している場合、別のアプリケーションはそのオブジェクトにアクセスできません。そのため、行レベルのロック (ロックされているためにアクセスできないデータの量が最小限に抑えられる) は、ブロック・レベル、データ・パーティション・レベル、または表レベルのロックと比較して最大の並行性が得られます。ただし、ロックには、ストレージや処理時間が必要なので、単一の表ロックはロック・オーバーヘッドを最小化します。

ALTER TABLE ステートメントの LOCKSIZE 節は、行、データ・パーティション、ブロック、または表レベルでロックの有効範囲 (細分性) を指定します。デフォルトでは、行ロックが使用されます。 S (共用) および X (排他) ロックのみがこれらの定義済みの表ロックによって要求されます。 ALTER TABLE ステートメントの LOCKSIZE ROW 節を使用しても、通常のロック・エスカレーションに支障はありません。

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 を使用して永続的な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。その他のユーザーは、表に S ロックを獲得することもできます。
- 表は、通常、読み取り専用アプリケーションによってアクセスされますが、時折、簡単な保守のために単一のユーザーによってアクセスされます。したがって、そのユーザーには X ロックが必要です。保守プログラムが実行されている間、読取専用アプリケーションはロックアウトされますが、その他の環境では、読み取り専用アプリケーションは、最小のロッキング・オーバーヘッドで並行して表にアクセスすることができます。

MDC 表の場合、INSERT 操作の間のみブロック・レベルのロックを使用するために、LOCKSIZE 節に BLOCKINSERT を指定できます。これが指定されると、他のすべての操作には行レベルのロックが実行されますが、INSERT 操作には最低限のロックしか実行されません。つまり、行の挿入時にはブロック・レベルのロックが使用されますが、索引の更新時に RR スキャンが検出される場合には次のキーのロックに行レベルのロックが使用されます。 BLOCKINSERT ロックは次のような場合に役立ちます。

- 異なるセルに大量の挿入を実行する複数のトランザクションがある。

- 複数のトランザクションによって同じセルに対して並行挿入が発生しない場合、または個々のブロックへ挿入することを想定していない各トランザクションによってセルごとに十分なデータが挿入されて並行挿入が発生する場合。

ALTER TABLE ステートメントは、グローバルにロックを指定するので、その表に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が影響を受けます。個々のアプリケーションは、その代わりにアプリケーション・レベルで表ロックを指定するために LOCK TABLE ステートメントを使用することができます。

ロックの待機とタイムアウト

ロックのタイムアウト検出は、ロックが異常な状況で解放されるまで無限にアプリケーションが待機しなくて済むようにするデータベース・マネージャーの機能です。たとえば、あるトランザクションが別のユーザーのアプリケーション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ロックを解放するということをせずにワークステーションから席をはずしてしまっているかもしれません。こうしたケースでアプリケーション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します。

このパラメーターを設定すると、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。 ロック要求がペンディングにされている時間が **locktimeout** 値より長くなると、要求している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。例えば、APPL1 が、既に APPL2 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、APPL1 は SQLCODE -911 (SQLSTATE 40001) と理由コード 68 を返します。 **locktimeout** のデフォルト値は -1 です。これにより、ロックのタイムアウト検出はオフになります。

注: アプリケーションは、SET CURRENT LOCK TIMEOUT を使用して、表、行、データ・パーティションおよび MDC ブロック・ロックに対するデータベース・レベルの **locktimeout** 設定をオーバーライドできます。

ロック・タイムアウトに関するレポート・ファイルを生成するには、**DB2_CAPTURE_LOCKTIMEOUT** レジストリー変数を ON に設定します。このロック・タイムアウト・レポートに含まれる情報には、ロックのタイムアウトの原因となったロック競合に関係した主なアプリケーション、およびロック自体に関する詳細 (ロック名、ロック・タイプ、行 ID、表スペース ID、および表 ID など) に関する情報が含まれます。

ロック要求タイムアウトに関する詳しい情報のログを db2diag.log ファイルに記録するには、データベース・マネージャー構成パラメーター **diaglevel** を 4 に設定します。ログに記録された情報には、オブジェクト、ロック・モード、および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 または XQuery ステートメントまたは静的パッケージ名もログに記録されている可能性があります。動的 SQL または XQuery ステートメントは、**diaglevel** が 4 である場合にのみログに記録されます。

ロック待機およびロック・タイムアウトに関する情報を、ロック待機情報システム・モニター・エレメント、または **db.apps_waiting_locks** ヘルス・インディケーターから取得することができます。

ロック・タイムアウトのレポート

ロック・タイムアウトのレポート作成機能は、ロック・タイムアウト・イベントに関する情報をキャプチャーします。これには、ロックのタイムアウトの原因となったロック競合に関係した主なアプリケーションに関する情報が含まれます。この機能は、**DB2_CAPTURE_LOCKTIMEOUT** レジストリー変数によって制御されます。

情報のキャプチャーは、ロック要求側 (ロック・タイムアウト・エラーを受け取ったアプリケーション) および現在のロック所有者の両方に対して行われ、レポートはデータベース構成パラメーターが決定するディレクトリーに保管されます。

ロックのタイムアウトが生じ、ロック・タイムアウトのレポート作成機能がアクティブである場合には、以下の情報がキャプチャーされ、ロック・タイムアウトのレポート・ファイルに置かれます。

競合しているロック

- ロックの名前とタイプ
- 行 ID、表スペース ID、および表 ID を含むロックの詳細。SYSCAT.TABLES システム・カタログ・ビューを照会して表名を検索するには、この情報を使用します。

ロック要求側

- アプリケーションの ID 情報 (アプリケーション名およびコーディネーター・パーティションなど)
- ロック・タイムアウトの値
- 要求ロック・モード
- 要求 SQL コンテキスト (該当する場合)
- パッケージ ID
- セクション・エントリー番号
- SQL 情報 (ステートメントが動的か静的か、ステートメントのタイプなど)
- 有効なロック分離
- 関係するステートメント・テキスト (使用可能な場合。詳しくは、194 ページの『機能の制約』を参照してください。)
- アプリケーションの状況
- 現行の操作
- ロック・エスカレーション

ロック所有者または代表者

複数のロック所有者がいる場合があります。例えば、ロックが複数のアプリケーションによって共有モードに保持されている場合、最初に検出されるロック所有者の情報のみが報告されます。その最初のロック所有者が、他の複数のロック所有者の代表者です。

- アプリケーションの ID 情報 (アプリケーション名およびコーディネーター・パーティションなど)
- 保持されているロック・モード
- このパーティションで現在アクティブである SQL ステートメントのリスト
  1. パッケージ ID
  2. セクション・エントリー番号
  3. SQL 情報 (ステートメントが動的か静的か、ステートメントのタイプなど)
  4. 有効なロック分離
  5. 関係するステートメント・テキスト (キャプチャー可能な場合)
- このデータベース・パーティションでの現行の作業単位からの非アクティブの SQL ステートメントのリスト (ステートメント履歴のあるデッドロック・イベント・モニターがアクティブである場合のみ)
  1. パッケージ ID
  2. セクション・エントリー番号
  3. SQL 情報 (ステートメントが動的か静的か、ステートメントのタイプなど)
  4. 有効なロック分離
  5. 関係するステートメント・テキスト (キャプチャー可能な場合)

オペレーティング・システムや、他の関連する環境に関する情報など、さらに情報を収集するには、カスタマイズ済みの db2cos スクリプトを使用してください。

機能の使用法

DB2_CAPTURE_LOCKTIMEOUT レジストリー変数がロック・タイムアウトのレポート作成機能を制御します。

このレジストリー変数が ON に設定されている場合、この機能は DB2 インスタンス内で発生する各ロック・タイムアウトに関する基本的な情報をキャプチャーし、ロック・タイムアウト・レポートが作成されます。このレジストリー変数が空に設定された場合、この機能は使用不可になります。

ロック・タイムアウトのレポート作成を有効にするには、以下のコマンドを発行します。

```
db2set DB2_CAPTURE_LOCKTIMEOUT=ON
```

ロック・タイムアウトのレポート作成を無効にするには、以下のコマンドを発行します。

```
db2set DB2_CAPTURE_LOCKTIMEOUT=
```

場合によっては、ロックのタイムアウト発生前にロック所有者の作業単位内で実行されたすべてのステートメントをキャプチャーする必要があるかもしれません。そうするのが有用なのは、例えば、ロックのタイムアウトがその前に実行された SQL ステートメントが原因で発生した場合です。現行のロック所有者のロック・タイムアウト情報に、作業単位の履歴を含めるには、ステートメント履歴の節を使用して

デッドロック・イベント・モニターをアクティブにします。例えば、以下のいずれかのステートメントを使用してください。

```
create event monitor testit for deadlocks with details history write to file path global
create event monitor testit for deadlocks with details history write to table
```

この CREATE EVENT MONITOR ステートメントには、追加のオプションがあり、例えば、データが書き込まれる表スペースおよび表の名前を指定することができます。詳しくは、CREATE EVENT MONITOR ステートメントの説明を参照してください。

ステートメント履歴機能のあるイベント・モニターは全アプリケーションに影響し、DB2 データベース・マネージャーによるモニター・ヒープ使用量が増加します。これは、ロック競合の原因が、関係するアプリケーションによって現在実行されているステートメントではない場合、または、レポート作成の目的が、単にロックのタイムアウトに関係するアプリケーションを判別することにとどまらない場合にのみ使用してください。

例えば、アプリケーション APPL1 は以下のステートメントを実行します。

```
INSERT INTO T1 VALUES (1)
SELECT * FROM T5
```

次に、APPL2 が以下のステートメントを実行します。

```
SELECT * FROM T1
```

この例では、APPL2 によって実行される SELECT ステートメントが、その前に APPL1 によって実行された INSERT ステートメントが取得した行ロックを待機しています。この場合、APPL2 によって現在実行されている SELECT ステートメントを見るだけでは、ロック競合の実際の原因を判別する助けにはなりません。ロック所有者のステートメント履歴によって、その前に INSERT ステートメントがあり、それがロック競合の原因であることわかります。

ロックを保持しているアプリケーションが、ロックのタイムアウト時に SQL ステートメントを実行していない場合でも、ステートメント履歴機能のあるイベント・モニターを作成していると役立ちます。履歴機能のあるデッドロック・イベント・モニターがアクティブである場合、作業単位内に含まれる、アプリケーションの以前の SQL ステートメントがレポートに書き込まれます。

機能の制約

ロック・タイムアウトのレポートは、ロック競合やロック・タイムアウトといったイベントに関する必要な詳細を常にキャプチャーするわけではありません。場合によっては、以下にリストするように、ロック・タイムアウト・レポート作成機能ですべての情報を使用できるわけではありません。

- すべてのケースで SQL ステートメント・テキストが使用可能とは限らない。例えば、ロックのタイムアウトに静的 SQL ステートメントが関係していた場合など。
- DB2 ユーティリティーおよび内部機能は、SQL を実行せずにロックを取得できる場合がある。例えば、オンライン・バックアップは、表スペースや表を処理する際には、オンライン・バックアップの間ずっとこれらのオブジェクトに対するロックを保持します。表のバックアップ中に、ロックを待機していたアプリケー

ションがタイムアウトになる場合もあります。バックアップ・コマンドが CLP から実行された場合、アプリケーション名は「db2bp」(CLP) として報告され、ロック所有者にアクティブな SQL ステートメントはありません。DB2 ユーティリティーまたは内部機能も、アプリケーションによるロックの解放を待機中にタイムアウトになることがあります。この際、要求側にアクティブな SQL ステートメントはありません。

- リモート・パーティションでアプリケーションのために機能しているサブエージェントは、アプリケーションの SQL ステートメントや他の情報に関して完全な情報を持っていない場合がある。特に、ロック所有者がリモート・サブエージェントである場合、このリモート・サブエージェントには、アプリケーションの完全なステートメント履歴はありません。
- 通常の DB2 操作中に実行された内部カタログ表の処理は SQL を使用しない。機能は単に関係する表を判別するに過ぎず、競合の原因となっている、または競合を検出している DB2 のコンポーネントは判別しません。
- 一部のロックは、特定の SQL ステートメントによって取得されることがない。これらは表ではないオブジェクト (パッケージなど) を表すか、または他の内部 DB2 処理によって使用されます。

ロック・タイムアウトのレポート・ファイル

ロック・タイムアウト・レポート作成機能がアクティブで、ロックのタイムアウトが生じた場合、ロック・タイムアウト・エラーを受信するエージェントはレポート・ファイルを生成します。このファイルは、ロックのタイムアウトが検出されたデータベース・パーティションの診断データ・ディレクトリー・パスに置かれます。

診断データ・ディレクトリー・パスは、データベース・マネージャー構成パラメーター **diagpath** によって定義されます。このパラメーターが NULL の場合は、レポート・ファイルの場所の **diagpath** パラメーター記述を参照してください。

パーティション・データベース環境では、1 つのアプリケーションが 1 つ以上のデータベース・パーティションで同時に作業を実行し、ロックを取得している場合があります。ロック・タイムアウトのレポート作成機能は、ロックのタイムアウト問題が生じたデータベース・パーティションでレポートを生成するため、それぞれのタイムアウト問題が生じた特定のデータベース・パーティションを判別するのに役立ちます。レポートには、そのパーティションでのロック・タイムアウト問題に固有のコンテキストに関連した情報が含まれます。

レポートは、以下の形式の名前で、ファイルに保管されます。
db2locktimeout.*par*.*AGENTID*.*yyyy-mm-dd-hh-mm-ss*

- ここで、*par* はデータベース・パーティション番号です。非パーティション・データベース環境では、*par* は `0` に設定されます。
- *AGENTID* はエージェント ID です。
- *yyyy-mm-dd-hh-mm-ss* は、年、月、日、時間、分、および秒から成るタイム・スタンプです。

ロック・タイムアウトのレポート・ファイル名の一例は、以下のとおりです。/home/juntang/sqllib/db2dump/db2locktimeout.000.4944050.2006-08-11-11-09-43

注: ロック・タイムアウトのレポート・ファイルが必要なくなった時点で、ディレクトリーから削除してください。レポート・ファイルは他の診断ログと同じ場所にあり、そのディレクトリーがいっぱいになると、DB2 システムがシャットダウンする恐れがあるためです。ロック・タイムアウトのレポート・ファイルを長期間にわたって保管する必要がある場合は、別のディレクトリーまたはフォルダーにファイルを移動してください。

ロックの変換

すでに保持しているロック・モードの変更は、*変換* と呼ばれます。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。照会によって間接的に 1 プロセスの中で同じデータ・オブジェクトに、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。

一部のロック・モードは表にのみ適用され、その他のロック・モードは行またはブロックにのみ適用されます。行またはブロックの場合、通常、X が必要なときに S または U (更新) ロックを保持している場合に、変換が行われます。

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大きい場合は、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。

照会が行を更新するとき、二重変換が発生する場合もあります。索引アクセスによって行が読み取られ、S としてロックされる場合、行を含む表には、目的ロックが含まれています。ただし、ロック・タイプが IX ではなく IS である場合、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。

ロック変換は、通常、照会が実行されるときに暗黙的に行われます。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。

システム・モニター・エレメントの *lock_current_mode* および *lock_mode* は、データベースで発生しているロック変換に関する情報を提供することができます。

ロックに関連したパフォーマンスの問題の回避

並行性およびデータ保全性のためにロックを調整するときは、以下の指針を考慮してください。

- 多数のユーザーによるデータの並行アクセスを促進する頻繁な COMMIT ステートメントを含む小さな作業単位を作成します。

  アプリケーションが論理的に一貫しているとき、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。

- COMMIT ステートメントを発行する前に CURSOR WITH HOLD をクローズします。

  状態によっては、結果セットが閉じてトランザクションがコミットされた後にも、ロックが残ります。COMMIT ステートメントを発行する前に CURSOR WITH HOLD を閉じることによって、ロックが確実に解除されます。
- INSERT ステートメントを別個の作業単位として実行します。

  状態によっては、結果セットが閉じてトランザクションがコミットされた後にも、ロックが残ります。INSERT ステートメントを別個の作業単位として実行することによって、ロックが確実に解除されます。
- 適切な分離レベルを指定します。

  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 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 または XQuery ステートメント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。
- LOCK TABLE ステートメントを正しく使用します。

  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キング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。

  LOCK TABLE IN SHARE MODE

  時間の点で一致した データにアクセスする、つまり特定の時点での表の現行データにアクセスする場合。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
表の大部分を更新する場合。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。

- アプリケーションで ALTER TABLE ステートメントを使用します。

  LOCKSIZE パラメーターを指定した ALTER TABLE ステートメントは、 LOCK TABLE ステートメントの代わりになります。 LOCKSIZE パラメーターでは、次の表アクセスの行ロックまたは表ロックのロックの細分性を指定できます。 MDC 表の場合、BLOCKINSERT 節のロックの細分性も指定できます。

  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を制限することによって、照会のパフォーマンスが向上する場合があります。ただし、ロックはすべて表全体にかけられるので、並行性は低下します。 MDC 表の場合、BLOCKINSERT 節を選択すると、ブロック・レベルでロックし、挿入に関して行ロックを回避することによって INSERT 操作のパフォーマンスが向上する場合があります。行レベルのロックは、他のすべての操作について実行され、キーの挿入では反復可能読み取り (RR) スキャナーを保護するために実行されます。 BLOCKINSERT オプションは、個々のトランザクションによるセルへの大規模な挿入に役立ちます。 LOCKSIZE を選択しても、通常のロック・エスカレーションは妨げられません。
- カーソルをクローズして、カーソルが保持しているロックを解放します。

  CLOSE CURSOR ステートメント (その中に WITH RELEASE 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、そのカーソルのために保持された読み取りロックをすべて解放しようとします。表読み取りロックには、表ロック IS、S、および U があります。行読み取りロックには、行ロック S、NS、および U があります。ブロック読み取りロックには、ブロック・ロック IS、S、および U があります。

  WITH RELEASE 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 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 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。

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。

- パーティション・データベース環境でロッキングに影響を与える構成パラメーターを変更するときは、すべてのデータベース・パーティションに変更を加えたかどうか確認してください。

ロック・エスカレーション問題の修正

データベース・マネージャーは、行またはブロック・レベルから表レベルにロックを自動的にエスカレーションすることができます。パーティション化された表では、データベース・マネージャーは、行またはブロック・レベルからデータ・パーティションレベルにロックを自動的にエスカレーションすることができます。*maxlocks* データベース構成パラメーターは、ロック・エスカレーションが起動されるときを指定します。ロック・エスカレーションを起動するロックを獲得する表には影響を与えません。ラージ・オブジェクト (LOB) および LONG VARCHAR 記述子がロックされている表から始めて、まず、大部分のロックを持つ表のロックがエスカレーションされ、その後、次に高いロック数を持つ表というように、保持しているロック数が *maxlocks* で指定された値の約半分になるまでエスカレーションされます。

適切に設計されているデータベースでは、ロック・エスカレーションはめったに起こりません。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 (*lock_escalation* モニター・エレメントまたは *db.lock_escal_rate* ヘルス・インディケーターで示される) まで下がると、問題を分析し、問題の解決方法を判別する必要があります。

ロック・エスカレーション情報が記録されていることを確認します。データベース・マネージャーの構成パラメーター *notifylevel* を 3 (デフォルト) または 4 に設定します。 2 という *notifylevel* では、エラー SQLCODE のみが報告されます。 3 または 4 という *notifylevel* では、ロック・エスカレーションが失敗するとき、エラー SQLCODE およびエスカレーションが失敗した表に関する情報が記録されます。現行の照会ステートメントは、それが現在実行している動的照会ステートメントであり、*notifylevel* が 4 に設定されている場合にのみログに記録されます。

受諾不能なロック・エスカレーションの原因を診断するには、以下の一般的なステップに従い、予防手段を適用してください。

1. ロックがエスカレーションされているすべての表の管理通知ログ内を分析します。 このログ・ファイルには、以下の情報が含まれています。
   - 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。
   - エスカレーション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。
   - エスカレーションの一部として獲得される新しい表レベルのロック。 通常、「S」(共用ロック) または「X」(排他ロック) が獲得されます。
   - 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。

2. 管理通知ログ内の情報を使用して、エスカレーション問題の解決方法を判別します。 以下の可能性を考慮します。
   - 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、グローバルに許可されるロックの数を増やす。パーティション・データベースでは、すべてのデータベース・パーティションでこの変更を行ってください。

     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。
   - エスカレーションを引き起こしたプロセスを調整します。これらのプロセスの場合、明示的に LOCK TABLE ステートメントを発行することができます。
   - 分離の程度を変更します。しかしこの場合、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して、特定の時間に存在するロックの数を減らす。
   - LONG VARCHAR または様々な種類のラージ・オブジェクト (LOB) データが必要なトランザクションに対する頻繁な COMMIT ステートメントを考慮します。結果セットがマテリアライズされるまで、この種類のデータはディスクから検索されませんが、まずデータが参照されるときに記述子がロックされます。その結果、一般的なデータを含む行のロックよりもさらに多くのロックが保持されている場合があります。

ロック据え置きによる非コミット・データの評価

並行性を向上させる目的で、DB2 では、レコードが照会の述部を満たしたことがわかるようになるまで、場合によっては CS または RS 分離スキャンの行ロックを据え置くことができます。 デフォルトでは、表または索引スキャン中に行ロックが実行されると、 DB2 はスキャンされる各行をロックしてから、行が照会の条件を満たしているかどうかを判別します。スキャンの並行性を向上させるために、行が照会の条件を満たしていると判別されるまで行ロックを据え置くことができます。

この機能を利用するには、DB2_EVALUNCOMMITTED レジストリー変数を使用可能にしてください。

この変数を使用可能にすると、非コミット・データに対して述部評価を行えるようになります。これは、非コミット更新を含む行が照会を満たしていなくても、更新されたトランザクションが完了してから述部評価を行うようにすれば、行が照会を満たしている場合があるということを意味します。さらに、削除された非コミット行は、表スキャン中にスキップされます。 DB2_SKIPDELETED レジストリー変数を使用可能にすると、 DB2 は削除されたキーをタイプ 2 索引スキャンでスキップします。

これらのレジストリー変数の設定は、動的 SQL または XQuery ステートメントの場合はコンパイル時に適用され、静的 SQL または XQuery ステートメントの場合はバインド時に適用されます。つまり、レジストリー変数を実行時に使用可能にしても、 DB2_EVALUNCOMMITTED をバインド時に使用可能にしない限り、ロック回避ストラテジーは採用されません。レジストリー変数を実行時ではなくバインド時に使用可能にすると、ロック回避ストラテジーは有効になります。静的 SQL ま

たは XQuery ステートメントの場合は、パッケージを再バインドすると、バインド時のレジストリー変数の設定が適用されます。静的 SQL または XQuery ステートメントを暗黙的に再バインドすると、DB2_EVALUNCOMMITTED の現在の設定が使用されます。

各種アクセス・プランの非コミットの評価の適用度

表 7. RID 索引単独アクセス

述部	非コミットの評価
なし	いいえ
検索引数述部	はい

表 8. データ単独アクセス (リレーショナルまたは据え置き RID リスト)

述部	非コミットの評価
なし	いいえ
検索引数述部	はい

表 9. RID 索引 + データ・アクセス

述部		非コミットの評価	
索引	**データ**	**索引アクセス**	**データ・アクセス**
なし	なし	いいえ	いいえ
なし	検索引数述部	いいえ	いいえ
検索引数述部	なし	はい	いいえ
検索引数述部	検索引数述部	はい	いいえ

表 10. ブロック索引 + データ・アクセス

述部		非コミットの評価	
索引	**データ**	**索引アクセス**	**データ・アクセス**
なし	なし	いいえ	いいえ
なし	検索引数述部	いいえ	はい
検索引数述部	なし	はい	いいえ
検索引数述部	検索引数述部	はい	はい

例

以下の例では、デフォルトのロック動作と新規の非コミットの評価動作を比較します。

以下の表は、SAMPLE データベースの ORG 表です。

```
DEPTNUMB DEPTNAME       MANAGER DIVISION   LOCATION
-------- -------------- ------- ---------- -------------
      10 Head Office        160 Corporate  New York
      15 New England         50 Eastern    Boston
      20 Mid Atlantic        10 Eastern    Washington
      38 South Atlantic      30 Eastern    Atlanta
      42 Great Lakes        100 Midwest    Chicago
```

```
51 Plains            140 Midwest    Dallas
66 Pacific           270 Western    San Francisco
84 Mountain          290 Western    Denver
```

この表に対して、以下のトランザクションがデフォルトのカーソル固定 (CS) 分離レベルで実行されます。

表 11. ORG 表に対する CS 分離レベルでのトランザクション

セッション 1	セッション 2
SAMPLE への接続	SAMPLE への接続
+c update org set deptnumb=5 where manager=160	
	select * from org where deptnumb >= 10

セッション 1 の非コミット UPDATE は、表の最初の行に対する排他的レコード・ロックを保持し、セッション 1 で更新中の行が現在セッション 2 の照会を満たしていない場合でも、セッション 2 の SELECT 照会が戻らないようにしています。これは、照会でアクセスされる行は、その行にカーソルが置かれている間はロックされなければならないと CS 分離レベルによって命令されているためです。セッション 2 は、セッション 1 がロックを解除するまで最初の行をロックできません。

表のスキャン中は、非コミットの評価機能を使用してセッション 2 でのロック待機を回避できます。この機能は、最初に述部を評価してから、実際の述部の評価を行うために行をロックします。このように、セッション 2 の照会では表の最初の行をロックしようとしないので、アプリケーション並行性が向上します。これは、セッション 2 の述部は、セッション 1 の deptnumb=5 という非コミット値に関して評価されるという意味でもあります。セッション 2 の照会では、セッション 1 の更新のロールバックがセッション 2 の照会を満たしているにもかかわらず、結果セットの最初の行が省略されます。

操作の順序を逆にすると、非コミットの評価で並行性がさらに向上する可能性があります。デフォルトのロッキング動作では、セッション 2 で最初に行ロックが獲得され、セッション 2 の照会でロックされる行をセッション 1 の UPDATE が変更しない場合でも、セッション 1 の検索済み UPDATE が実行されないようにします。セッション 1 の検索済み UPDATE が最初に行を検査して、条件を満たしている場合にのみ行をロックしようとする場合は、セッション 1 の照会は非ブロッキングになります。

制約事項

この新機能には、以下の外部制約事項が適用されます。

- レジストリー変数 DB2_EVALUNCOMMITTED を使用可能にする必要があります。
- 分離レベルは CS または RS でなければなりません。
- 行ロックが発生します。
- 検索引数述部評価述部が存在します。
- 非コミットの評価はカタログ表でのスキャンに適用できません。

- MDC 表の場合は索引スキャンのためにブロック・ロッキングを据え置くことができますが、表スキャンの場合はブロック・ロッキングを据え置くことはできません。
- 据え置きロッキングは、インプレース表 REORG を実行している表に対しては発生しません。
- 索引がタイプ 1 の索引スキャンでは、据え置きロッキングは発生しません。
- Iscan-Fetch プランの場合は、行ロッキングはデータ・アクセスのために据え置かれるのではなく、表の中の行に移動する前に索引アクセス中に行がロックされます。
- 削除された行は表スキャン中に無条件でスキップされますが、削除されたタイプ 2 索引キーはレジストリー変数 DB2_SKIPDELETED が使用可能になっている場合にのみスキップされます。

非コミット追加を無視するオプション

レジストリー変数 DB2_SKIPINSERTED は、カーソル固定 (CS) や読み取り固定 (RS) の分離レベルを使用して、カーソルで非コミット追加を無視可能にするかどうかを制御します。DB2 データベース・システムは、次の方法で非コミット追加を処理できます。

- DB2 データベース・システムは、INSERT トランザクションが完了 (コミットやロールバック) し、それに応じてデータを処理するまで待機します。 これはデフォルト・オプション (OFF) です。

  次の例では、デフォルト・オプション OFF が設定済みのときのインスタンスを示します。

  – 2 つのアプリケーションが表を使用してアプリケーション間でデータを受け渡す (1 番目のアプリケーションが表にデータを挿入し 2 番目のアプリケーションが読み取る) と想定します。 表に示されている順序で 2 番目のアプリケーションによってデータが処理され、読み取る次の行が 1 番目のアプリケーションによって挿入されている場合、2 番目のアプリケーションは、挿入がコミットされるまで待機する必要があります。 このような場合、DB2_SKIPINSERTED 用のデフォルト値を使用する必要があります。
  – データを削除し、データの新規イメージを挿入することによってアプリケーションがデータを変更すると想定します。 そのように UPDATE ステートメントを使用しない場合、DB2_SKIPINSERTED 用のデフォルト値を使用する必要があります。
- DB2 データベース・システムは、非コミット追加を無視でき、多くの場合並行性を向上できます。 この処理を実行するためには、レジストリー変数を ON に指定する必要があります。

  一般に、ON はほとんどのアプリケーションで並行性を向上させ、有効です。レジストリー変数が有効な場合、非コミットの行の挿入はまだ挿入されていないものとして処理されます。

ロック・タイプの互換性

ロックの互換性は、あるアプリケーションがあるオブジェクトのロックを現行で保持しているときに、別のアプリケーションが同じオブジェクトのロックを要求する

場合に問題になります。 2 つのロック・モードに互換性があれば、オブジェクトに対する 2 番目のロックの要求は認可されます。

要求されたロックのロック・モードがすでに保持されているロックと互換性がないなら、ロック要求は認可されません。その場合、要求は最初のアプリケーションがロックを解放し、さらに他の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。

以下の表では、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ときに、ロック要求が認可される状態に関する情報を表示します。**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。要求側がロック待機中に、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 12. 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**	**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
IN	はい	はい	はい	はい	はい	はい	はい	はい	はい	no	はい	はい
IS	はい	はい	はい	はい	はい	はい	はい	はい	no	no	no	no
NS	はい	はい	はい	はい	はい	no	no	はい	no	no	はい	no
S	はい	はい	はい	はい	はい	no	no	はい	no	no	no	no
IX	はい	はい	はい	no	no	はい	no	no	no	no	no	no
SIX	はい	はい	はい	no	no	no	no	no	no	no	no	no
U	はい	はい	はい	はい	はい	no	no	no	no	no	no	no
X	はい	はい	no	no	no	no	no	no	no	no	no	no
Z	はい	no	no	no	no	no	no	no	no	no	no	no
NW	はい	はい	no	はい	no	no	no	no	no	no	no	はい
W	はい	はい	no	no	no	no	no	no	no	no	はい	no

注:

I 意図

N なし

NS 次キー共有

S 共有

X 排他

U 更新

Z 超排他

NW 次キーの弱い排他

W 弱い排他

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**

標準の表のロック・モードおよびアクセス・パス

このトピックには、異なるデータ・アクセス・プランに関する標準の表のロッキング方式に関する参照情報が含まれています。

以下の表では、異なるアクセス・プランのそれぞれのレベルの標準の表で獲得されるロックのタイプをリストしています。それぞれの項目は、表ロックおよび行ロックの 2 つの部分から成り立っています。ダッシュは、特定のレベルのロッキングは行われないことを示します。

注:

1. マルチディメンション・クラスタリング (MDC) 環境では、さらにロック・レベル BLOCK が使用されます。
2. ロック・モードは、SELECT ステートメントのロック要求節を使用して明示的に変更できます。

表 13. 述部なしの表スキャンのロック・モード

分離レベル	読み取り専用および未確定のスキャン	カーソル操作		検索型更新または削除	
		スキャン	現在の場所	スキャン	更新または削除
RR	S/-	U/-	SIX/X	X/-	X/-
RS	IS/NS	IX/U	IX/X	IX/X	IX/X
CS	IS/NS	IX/U	IX/X	IX/X	IX/X
UR	IN/-	IX/U	IX/X	IX/X	IX/X

表 14. 述部ありの表スキャンのロック・モード

分離レベル	読み取り専用および未確定のスキャン	カーソル操作		検索型更新または削除	
		スキャン	現在の場所	スキャン	更新または削除
RR	S/-	U/-	SIX/X	U/-	SIX/X
RS	IS/NS	IX/U	IX/X	IX/U	IX/X
CS	IS/NS	IX/U	IX/X	IX/U	IX/X
UR	IN/-	IX/U	IX/X	IX/U	IX/X

注: タイプ 1 索引の IN ロックを持つ UR 分離レベルの場合、または索引の組み込み列に述部がある場合、分離レベルは、CS および IS 表ロックや NS 行ロックへのロックにアップグレードされます。

表 15. 述部なしの RID 索引スキャンのロック・モード

分離レベル	読み取り専用および未確定のスキャン	カーソル操作		検索型更新または削除	
		スキャン	現在の場所	スキャン	更新または削除
RR	S/-	IX/S	IX/X	X/-	X/-
RS	IS/NS	IX/U	IX/X	IX/X	IX/X

表 15. 述部なしの RID 索引スキャンのロック・モード (続き)

分離レベル	読み取り専用および未確定のスキャン	カーソル操作		検索型更新または削除	
		スキャン	現在の場所	スキャン	更新または削除
CS	IS/NS	IX/U	IX/X	IX/X	IX/X
UR	IN/-	IX/U	IX/X	IX/X	IX/X

表 16. 単一修飾行での RID 索引スキャンのロック・モード

分離レベル	読み取り専用および未確定のスキャン	カーソル操作		検索型更新または削除	
		スキャン	現在の場所	スキャン	更新または削除
RR	IS/S	IX/U	IX/X	IX/X	IX/X
RS	IS/NS	IX/U	IX/X	IX/X	IX/X
CS	IS/NS	IX/U	IX/X	IX/X	IX/X
UR	IN/-	IX/U	IX/X	IX/X	IX/X

表 17. 開始述部と停止述部のみでの RID 索引スキャンのロック・モード

分離レベル	読み取り専用および未確定のスキャン	カーソル操作		検索型更新または削除	
		スキャン	現在の場所	スキャン	更新または削除
RR	IS/S	IX/S	IX/X	IX/X	IX/X
RS	IS/NS	IX/U	IX/X	IX/X	IX/X
CS	IS/NS	IX/U	IX/X	IX/X	IX/X
UR	IN/-	IX/U	IX/X	IX/X	IX/X

表 18. 索引と他の述部 (sargs、resids) のみでの RID 索引スキャンのロック・モード

分離レベル	読み取り専用および未確定のスキャン	カーソル操作		検索型更新または削除	
		スキャン	現在の場所	スキャン	更新または削除
RR	IS/S	IX/S	IX/X	IX/S	IX/X
RS	IS/NS	IX/U	IX/X	IX/U	IX/X
CS	IS/NS	IX/U	IX/X	IX/U	IX/X
UR	IN/-	IX/U	IX/X	IX/U	IX/X

次の表には、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、さらに条件を付ける
- 効果的なプリフェッチが行えるようにソートする

表 19. 据え置きデータ・ページ・アクセスに使用される索引スキャンのロック・モード: 述部なしでの RID 索引スキャン

分離レベル	読み取り専用および未確定のスキャン	カーソル操作		検索型更新または削除	
		スキャン	現在の場所	スキャン	更新または削除
RR	IS/S	IX/S		X/-	
RS	IN/-	IN/-		IN/-	
CS	IN/-	IN/-		IN/-	
UR	IN/-	IN/-		IN/-	

表 20. 据え置きデータ・ページ・アクセスに使用される索引スキャンのロック・モード: 述部なしでの RID 索引スキャン後

分離レベル	読み取り専用および未確定のスキャン	カーソル操作		検索型更新または削除	
		スキャン	現在の場所	スキャン	更新または削除
RR	IN/-	IX/S	IX/X	X/-	X/-
RS	IS/NS	IX/U	IX/X	IX/X	IX/X
CS	IS/NS	IX/U	IX/X	IX/X	IX/X
UR	IN/-	IX/U	IX/X	IX/X	IX/X

表 21. 据え置きデータ・ページ・アクセスに使用される索引スキャンのロック・モード: 述部 (sargs、resids) での RID 索引スキャン

分離レベル	読み取り専用および未確定のスキャン	カーソル操作		検索型更新または削除	
		スキャン	現在の場所	スキャン	更新または削除
RR	IS/S	IX/S		IX/S	
RS	IN/-	IN/-		IN/-	
CS	IN/-	IN/-		IN/-	
UR	IN/-	IN/-		IN/-	

表 22. 据え置きデータ・ページ・アクセスに使用される索引スキャンのロック・モード: 開始述部と停止述部のみの RID 索引スキャン

分離レベル	読み取り専用および未確定のスキャン	カーソル操作		検索型更新または削除	
		スキャン	現在の場所	スキャン	更新または削除
RR	IS/S	IX/S		IX/X	
RS	IN/-	IN/-		IN/-	
CS	IN/-	IN/-		IN/-	
UR	IN/-	IN/-		IN/-	

表 23. 据え置きデータ・ページ・アクセスに使用される索引スキャンのロック・モード (開始述部と停止述部のみの RID 索引スキャン後)

分離レベル	読み取り専用および未確定のスキャン	カーソル操作		検索型更新または削除	
		スキャン	現在の場所	スキャン	更新または削除
RR	IN/-	IX/S	IX/X	IX/X	IX/X
RS	IS/NS	IX/U	IX/X	IX/U	IX/X
CS	IS/NS	IX/U	IX/X	IX/U	IX/X
UR	IS/-	IX/U	IX/X	IX/U	IX/X

表 24. 据え置きデータ・ページ・アクセスに使用される索引スキャンのロック・モード (述部ありの RID 索引スキャン後)

分離レベル	読み取り専用および未確定のスキャン	カーソル操作		検索型更新または削除	
		スキャン	現在の場所	スキャン	更新または削除
RR	IN/-	IX/S	IX/X	IX/S	IX/X
RS	IS/NS	IX/U	IX/X	IX/U	IX/X
CS	IS/NS	IX/U	IX/X	IX/U	IX/X
UR	IN/-	IX/U	IX/X	IX/U	IX/X

MDC 表の表および RID 索引スキャンのロック・モード

マルチディメンション・クラスタリング (MDC) 環境では、さらにロック・レベル BLOCK が使用されます。以下の表には、種々のアクセス・プランの各レベルで取得されるロックのタイプをリストしています。各項目は、表ロック、ブロック・ロック、行ロックの 3 つの部分から成ります。ダッシュは、特定のレベルのロッキングは使用されないことを示します。

注: ロック・モードは、SELECT ステートメントのロック要求節を使用して明示的に変更できます。

表 25. 述部なしの表スキャンのロック・モード

分離レベル	読み取り専用および未確定のスキャン	カーソル操作		検索型更新または削除	
		スキャン	現在の場所	スキャンまたは削除	更新
RR	S/-/-	U/-/-	SIX/IX/X	X/-/-	X/-/-
RS	IS/IS/NS	IX/IX/U	IX/IX/U	IX/X/-	IX/I/-
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/X/-	IX/X/-
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/X/-	IX/X/-

表 26. ディメンション列上の述部のみでの表スキャンのロック・モード

分離レベル	読み取り専用および未確定のスキャン	カーソル操作		検索型更新または削除	
		スキャン	現在の場所	スキャンまたは削除	更新
RR	S/-/-	U/-/-	SIX/IX/X	U/-/-	SIX/X/-

表 26. ディメンション列上の述部のみでの表スキャンのロック・モード (続き)

分離レベル	読み取り専用および未確定のスキャン	カーソル操作		検索型更新または削除	
		スキャン	現在の場所	スキャンまたは削除	更新
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/U/-	X/X/-
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/U/-	X/X/-
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/U/-	X/X/-

表 27. 索引と他の述部 (sargs、resids) での 表索引スキャンのロック・モード

分離レベル	読み取り専用および未確定のスキャン	カーソル操作		検索型更新または削除	
		スキャン	現在の場所	スキャンまたは削除	更新
RR	S/-/-	U/-/-	SIX/IX/X	U/-/-	SIX/IX/X
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X

以下の 2 つの表には、 MDC 表上の RID 索引のロック・モードが表されています。

表 28. 述部なしの RID 索引スキャンのロック・モード

分離レベル	読み取り専用および未確定のスキャン	カーソル操作		検索型更新または削除	
		スキャン	現在の場所	スキャンまたは削除	更新
RR	S/-/-	IX/IX/S	IX/IX/X	X/-/-	X/-/-
RS	IS/IS/NS	IX/IX/U	IX/IX/X	X/X/X	X/X/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	X/X/X	X/X/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	X/X/X	X/X/X

表 29. 単一修飾行での RID 索引スキャンのロック・モード

分離レベル	読み取り専用および未確定のスキャン	カーソル操作		検索型更新または削除	
		スキャン	現在の場所	スキャンまたは削除	更新
RR	IS/IS/S	IX/IX/U	IX/IX/X	X/X/X	X/X/X
RS	IS/IS/NS	IX/IX/U	IX/IX/X	X/X/X	X/X/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	X/X/X	X/X/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	X/X/X	X/X/X

表 30. 開始述部と停止述部のみでの RID 索引スキャンのロック・モード

分離レベル	読み取り専用および未確定のスキャン	カーソル操作		検索型更新または削除	
		スキャン	現在の場所	スキャンまたは削除	更新
RR	IS/IS/S	IX/IX/S	IX/IX/X	IX/IX/X	IX/IX/X
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/X	IX/IX/X

表 30. 開始述部と停止述部のみでの RID 索引スキャンのロック・モード (続き)

分離レベル	読み取り専用および未確定のスキャン	カーソル操作		検索型更新または削除	
		スキャン	現在の場所	スキャンまたは削除	更新
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/X	IX/IX/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/IX/X	IX/IX/X

表 31. 索引述部のみでの RID 索引スキャンのロック・モード

分離レベル	読み取り専用および未確定のスキャン	カーソル操作		検索型更新または削除	
		スキャン	現在の場所	スキャンまたは削除	更新
RR	IS/S/S	IX/IX/S	IX/IX/X	IX/IX/S	IX/IX/X
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X

表 32. 他の述部 (sargs、resids) での RID 索引スキャンのロック・モード

分離レベル	読み取り専用および未確定のスキャン	カーソル操作		検索型更新または削除	
		スキャン	現在の場所	スキャンまたは削除	更新
RR	IS/S/S	IX/IX/S	IX/IX/X	IX/IX/S	IX/IX/X
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X

注: 据え置きデータ・ページ・アクセスに使用される RID 索引スキャンのロック・モードを示す以下の表において、 IN ロックを含む UR 分離レベルでは、タイプ 1 索引の場合や索引の組み込み列上に述部が存在している場合、分離レベルは CS にアップグレードされ、ロックは IS 表ロック、IS ブロック・ロック、および NS 行ロックにアップグレードされます。

表 33. 据え置きデータ・ページ・アクセスに使用される RID 索引スキャンのロック・モード: 述部なしでの RID 索引スキャン

分離レベル	読み取り専用および未確定のスキャン	カーソル操作		検索型更新または削除	
		スキャン	現在の場所	スキャンまたは削除	更新
RR	IS/S/S	IX/IX/S		X/-/-	
RS	IN/IN/-	IN/IN/-		IN/IN/-	
CS	IN/IN/-	IN/IN/-		IN/IN/-	
UR	IN/IN/-	IN/IN/-		IN/IN/-	

表 34. 据え置きデータ・ページ・アクセスに使用される RID 索引スキャンのロック・モード: 据え置きデータ・ページ・アクセス (述部なしでの RID 索引スキャン後)

分離レベル	読み取り専用および未確定のスキャン	カーソル操作		検索型更新または削除	
		スキャン	現在の場所	スキャンまたは削除	更新
RR	IN/IN/-	IX/IX/S	IX/IX/X	X/-/-	X/-/-
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/X	IX/IX/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/X	IX/IX/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/IX/X	IX/IX/X

表 35. 据え置きデータ・ページ・アクセスに使用される RID 索引スキャンのロック・モード: 述部 (sargs、resids) での RID 索引スキャン

分離レベル	読み取り専用および未確定のスキャン	カーソル操作		検索型更新または削除	
		スキャン	現在の場所	スキャンまたは削除	更新
RR	IS/S/-	IX/IX/S		IX/IX/S	
RS	IN/IN/-	IN/IN/-		IN/IN/-	
CS	IN/IN/-	IN/IN/-		IN/IN/-	
UR	IN/IN/-	IN/IN/-		IN/IN/-	

表 36. 据え置きデータ・ページ・アクセスに使用される RID 索引スキャンのロック・モード: 据え置きデータ・ページ・アクセス (述部 (sargs、resids) での RID 索引スキャン後)

分離レベル	読み取り専用および未確定のスキャン	カーソル操作		検索型更新または削除	
		スキャン	現在の場所	スキャンまたは削除	更新
RR	IN/IN/-	IX/IX/S	IX/IX/X	IX/IX/S	IX/IX/X
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X

表 37. 据え置きデータ・ページ・アクセスに使用される RID 索引スキャンのロック・モード: 開始述部と停止述部のみの RID 索引スキャン

分離レベル	読み取り専用および未確定のスキャン	カーソル操作		検索型更新または削除	
		スキャン	現在の場所	スキャンまたは削除	更新
RR	IS/IS/S	IX/IX/S		IX/IX/X	
RS	IN/IN/-	IN/IN/-		IN/IN/-	
CS	IN/IN/-	IN/IN/-		IN/IN/-	
UR	IN/IN/-	IN/IN/-		IN/IN/-	

表 38. 据え置きデータ・ページ・アクセスに使用される RID 索引スキャンのロック・モード: 据え置きデータ・ページ・アクセス (開始述部と停止述部のみの RID 索引スキャン後)

分離レベル	読み取り専用および未確定のスキャン	カーソル操作		検索型更新または削除	
		スキャン	現在の場所	スキャンまたは削除	更新
RR	IN/IN/-	IX/IX/S	IX/IX/X	IX/IX/X	IX/IX/X
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
UR	IS/-/-	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X

MDC 表のブロック索引スキャンのロック

以下の表には、種々のアクセス・プランの各レベルで取得されるロックのタイプをリストしています。各項目は、表ロック、ブロック・ロック、行ロックの 3 つの部分から成ります。 ダッシュは、特定のレベルのロッキングは行われないことを示します。

注: ロック・モードは、SELECT ステートメントのロック要求節を使用して明示的に変更できます。

表 39. 述部なしでの索引スキャンのロック・モード

分離レベル	読み取り専用および未確定のスキャン	カーソル操作		検索型更新または削除	
		スキャン	現在の場所	スキャンまたは削除	更新
RR	S/--/--	IX/IX/S	IX/IX/X	X/--/--	X/--/--
RS	IS/IS/NS	IX/IX/U	IX/IX/X	X/X/--	X/X/--
CS	IS/IS/NS	IX/IX/U	IX/IX/X	X/X/--	X/X/--
UR	IN/IN/-	IX/IX/U	IX/IX/X	X/X/--	X/X/--

表 40. ディメンション述部のみでの索引スキャンのロック・モード

分離レベル	読み取り専用および未確定のスキャン	カーソル操作		検索型更新または削除	
		スキャン	現在の場所	スキャンまたは削除	更新
RR	IS/-/-	IX/IX/S	IX/IX/X	X/-/-	X/-/-
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/X/-	IX/X/-
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/X/-	IX/X/-
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/X/-	IX/X/-

表 41. 開始述部と停止述部のみでの索引スキャンのロック・モード

分離レベル	読み取り専用および未確定のスキャン	カーソル操作		検索型更新または削除	
		スキャン	現在の場所	スキャンまたは削除	更新
RR	IS/S/-	IX/IX/S	IX/IX/S	IX/IX/S	IX/IX/S
RS	IX/IX/S	IX/IX/U	IX/IX/X	IX/IX/-	IX/IX/-
CS	IX/IX/S	IX/IX/U	IX/IX/X	IX/IX/-	IX/IX/-

表 41. 開始述部と停止述部のみでの索引スキャンのロック・モード (続き)

分離レベル	読み取り専用および未確定のスキャン	カーソル操作		検索型更新または削除	
		スキャン	現在の場所	スキャンまたは削除	更新
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/IX/-	IX/IX/-

表 42. 述部での索引スキャンのロック・モード

分離レベル	読み取り専用および未確定のスキャン	カーソル操作		検索型更新または削除	
		スキャン	現在の場所	スキャンまたは削除	更新
RR	IS/S/-	IX/IX/S	IX/IX/X	IX/IX/S	IX/IX/X
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X

以下の表には、据え置きデータ・ページ・アクセスに使用されるブロック索引スキャンのロック・モードをリストしています。

表 43. 据え置きデータ・ページ・アクセスに使用されるブロック索引スキャンのロック・モード: 述部なしでのブロック索引スキャン

分離レベル	読み取り専用および未確定のスキャン	カーソル操作		検索型更新または削除	
		スキャン	現在の場所	スキャンまたは削除	更新
RR	IS/S/--	IX/IX/S		X/--/--	
RS	IN/IN/--	IN/IN/--		IN/IN/--	
CS	IN/IN/--	IN/IN/--		IN/IN/--	
UR	IN/IN/--	IN/IN/--		IN/IN/--	

表 44. 据え置きデータ・ページ・アクセスに使用されるブロック索引スキャンのロック・モード: 据え置きデータ・ページ・アクセス (述部なしでのブロック索引スキャン後)

分離レベル	読み取り専用および未確定のスキャン	カーソル操作		検索型更新または削除	
		スキャン	現在の場所	スキャンまたは削除	更新
RR	IN/IN/--	IX/IX/S	IX/IX/X	X/--/--	X/--/--
RS	IS/IS/NS	IX/IX/U	IX/IX/X	X/X/--	X/X/--
CS	IS/IS/NS	IX/IX/U	IX/IX/X	X/X/--	X/X/--
UR	IN/IN/--	IX/IX/U	IX/IX/X	X/X/--	X/X/--

表 45. 据え置きデータ・ページ・アクセスに使用されるブロック索引スキャンのロック・モード: ディメンション述部のみでのブロック索引スキャン

分離レベル	読み取り専用および未確定のスキャン	カーソル操作		検索型更新または削除	
		スキャン	現在の場所	スキャンまたは削除	更新
RR	IS/S/--	IX/IX/--		IX/S/--	

表 45. 据え置きデータ・ページ・アクセスに使用されるブロック索引スキャンのロック・モード: ディメンション述部のみでのブロック索引スキャン (続き)

分離レベル	読み取り専用および未確定のスキャン	カーソル操作		検索型更新または削除	
		スキャン	現在の場所	スキャンまたは削除	更新
RS	IS/IS/NS	IX/--/--		IX/--/--	
CS	IS/IS/NS	IX/--/--		IX/--/--	
UR	IN/IN/--	IX/--/--		IX/--/--	

表 46. 据え置きデータ・ページ・アクセスに使用されるブロック索引スキャンのロック・モード: 据え置きデータ・ページ・アクセス (ディメンション述部のみでのブロック索引スキャン後)

分離レベル	読み取り専用および未確定のスキャン	カーソル操作		検索型更新または削除	
		スキャン	現在の場所	スキャンまたは削除	更新
RR	IN/IN/--	IX/IX/S	IX/IX/X	IX/S/--	IX/X/--
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/U/--	IX/X/--
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/U/--	IX/X/--
UR	IN/IN/--	IX/IX/U	IX/IX/X	IX/U/--	IX/X/--

表 47. 据え置きデータ・ページ・アクセスに使用されるブロック索引スキャンのロック・モード: 開始述部と停止述部のみのブロック索引スキャン

分離レベル	読み取り専用および未確定のスキャン	カーソル操作		検索型更新または削除	
		スキャン	現在の場所	スキャンまたは削除	更新
RR	IS/S/--	IX/IX/--		IX/X/--	
RS	IN/IN/--	IN/IN/--		IN/IN/--	
CS	IN/IN/--	IN/IN/--		IN/IN/--	
UR	IN/IN/--	IN/IN/--		IN/IN/--	

表 48. 据え置きデータ・ページ・アクセスに使用されるブロック索引スキャンのロック・モード: 据え置きデータ・ページ・アクセス (開始述部と停止述部のみのブロック索引スキャン後)

分離レベル	読み取り専用および未確定のスキャン	カーソル操作		検索型更新または削除	
		スキャン	現在の場所	スキャンまたは削除	更新
RR	IN/IN/--	IX/IX/X		IX/X/--	
RS	IS/IS/NS	IN/IN/--		IN/IN/--	
CS	IS/IS/NS	IN/IN/--		IN/IN/--	
UR	IS/--/--	IN/IN/--		IN/IN/--	

表 49. 据え置きデータ・ページ・アクセスに使用されるブロック索引スキャンのロック・モード: 他の述部 (sargs、resids) でのブロック索引スキャン

分離レベル	読み取り専用および未確定のスキャン	カーソル操作		検索型更新または削除	
		スキャン	現在の場所	スキャンまたは削除	更新
RR	IS/S/--	IX/IX/--		IX/IX/--	
RS	IN/IN/--	IN/IN/--		IN/IN/--	
CS	IN/IN/--	IN/IN/--		IN/IN/--	
UR	IN/IN/--	IN/IN/--		IN/IN/--	

表 50. 据え置きデータ・ページ・アクセスに使用されるブロック索引スキャンのロック・モード: 据え置きデータ・ページ・アクセス (他の述部 (sargs、resids) でのブロック索引スキャン後)

分離レベル	読み取り専用および未確定のスキャン	カーソル操作		検索型更新または削除	
		スキャン	現在の場所	スキャンまたは削除	更新
RR	IN/IN/--	IX/IX/S	IX/IX/X	IX/IX/S	IX/IX/X
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
UR	IN/IN/--	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X

パーティション表での動作をロックする

表全体のロックに加えて、パーティション表の各データ・パーティションごとのロックがあります。これにより、非パーティション表と比べて、より良い細分性および並行性の増大が可能になります。新規のデータ・パーティション・ロックは、db2pd コマンド、イベント・モニター、管理ビュー、および表関数の出力によって識別されます。

表にアクセスするとき、ロック動作は表ロックを最初に取得し、次にアクセスされたデータの命令に従ってデータ・パーティション・ロックを取得します。アクセス方式および分離レベルは、結果セットに含まれていないデータ・パーティションのロックを必要とする場合があります。これらのデータ・パーティション・ロックが取得されると、表ロックと同じだけ長く保持されます。たとえば、カーソル固定 (CS) の索引のスキャンは、以前にアクセスされたデータ・パーティションでロックを保持し、後続のキーでそのデータ・パーティションが参照される場合に、データ・パーティション・ロックを再取得するコストを削減する可能性があります。さらにデータ・パーティション・ロックは、表スペースへのアクセスを確保するコストを担います。非パーティション表の場合、表スペースのアクセスは表ロックによって処理されます。したがって、パーティション表の表レベルに排他ロックまたは共有ロックがある場合でも、データ・パーティション・ロックは発生します。

より良い細分性によって、1 つのトランザクションは、他のトランザクションが他のデータ・パーティションにアクセスしている間に、指定されたデータ・パーティションへ排他的にアクセスでき、行ロックを避けることができます。これは、大量更新用に選択されるプランの結果として、またはデータ・パーティション・レベルへのロックのエスカレーションによって生じることがあります。データ・パーティ

ションが共有または排他的にロックされている場合であっても、多数のアクセス方式の表ロックは、通常意図的ロックです。これにより、並行性が増大します。しかし、非意図的ロックがデータ・パーティション・レベルで必要とされ、すべてのデータ・パーティションがアクセスされる可能性があることをプランが示す場合には、並行トランザクションからデータ・パーティション・ロック間のデッドロックが発生しないようにするために、表レベルで非意図が選択されることがあります。

SQL LOCK TABLE ステートメントのロック

パーティション表の場合、LOCK TABLE ステートメントに対して取得される唯一のロックは表レベルであり、データ・パーティション・ロックは取得されません。これは、行、ブロック、またはデータ・パーティション・レベルでデッドロックを避けるだけでなく、後続の DML ステートメントで表に対して行ロックが起きないようにします。 LOCK TABLE IN EXCLUSIVE MODE の使用は、索引を更新するときに排他的アクセスを保証するのに使用されますが、大きな更新の間にタイプ 2 索引の増大を制限するのに役立ちます。

ALTER TABLE ステートメントの LOCKSIZE TABLE パラメーターの影響

ALTER TABLE ステートメントには、LOCKSIZE TABLE の設定についてのオプションがありますが、これは表が意図的ロックなしで共有または排他的にロックされるようにします。パーティション表の場合、このロック計画は、表ロック、およびアクセスされるあらゆるデータ・パーティションのデータ・パーティション・ロックの両方に適用されます。

行ロックおよびブロック・ロックのエスカレーション

パーティション表の場合、行ロックおよびブロック・ロックのエスカレーションの単位は、データ・パーティション・レベルになります。これは、データ・パーティションが SHARE、EXCLUSIVE、SUPER EXCLUSIVE にエスカレートされる場合にも、エスカレートされない他のデータ・パーティションは影響を受けず、表が他のトランザクションによりさらにアクセスできることを意味します。トランザクションは、指定のデータ・パーティションのエスカレーション後に、他のデータ・パーティションの行ロックを継続できます。エスカレーションの通知ログ・メッセージには、パーティション表の名前に加えて、エスカレートされたデータ・パーティションが含まれます。したがって、ロック・エスカレーションによって索引への排他的アクセスを確保することはできません。ステートメントが排他表レベル・ロックを使用するか、明示的 LOCK TABLE IN EXCLUSIVE MODE ステートメントが発行されるか、または表が LOCKSIZE TABLE 属性を使用するかのいずれかが必要です。アクセス方式の全体の表ロックはオプティマイザーによって選択され、データ・パーティションの除去に依存します。データ・パーティションの除去が発生しない場合、表への大きな更新により、排他表ロックが選択される可能性があります。

ロック情報の解釈

以下の SNAPLOCK 管理ビューの例は、パーティション表から戻されたロック情報を解釈する際の参考にしてください。

例 1:

この SNAPLOCK 管理ビューは、オフラインでの索引再編成中にキャプチャーされたものです。

```
SELECT SUBSTR(TABNAME, 1, 15) TABNAME, TAB_FILE_ID, SUBSTR(TBSP_NAME, 1, 15) TBSP_NAME, DATA_PARTITION_ID, LOCK_OBJECT_TYPE,
LOCK_MODE, LOCK_ESCALATION FROM SYSIBMADM.SNAPLOCK where TABNAME like 'TP1' and LOCK_OBJECT_TYPE like 'TABLE_%'
ORDER BY TABNAME, DATA_PARTITION_ID, LOCK_OBJECT_TYPE, TAB_FILE_ID, LOCK_MODE

TABNAME         TAB_FILE_ID          TBSP_NAME       DATA_PARTITION_ID LOCK_OBJECT_TYPE   LOCK_MODE  LOCK_ESCALATION
--------------- -------------------- --------------- ----------------- ------------------ ---------- ---------------
TP1                           32768 -                              -1 TABLE_LOCK         Z                       0
TP1                               4 USERSPACE1                     0 TABLE_PART_LOCK    Z                       0
TP1                               5 USERSPACE1                     1 TABLE_PART_LOCK    Z                       0
TP1                               6 USERSPACE1                     2 TABLE_PART_LOCK    Z                       0
TP1                               7 USERSPACE1                     3 TABLE_PART_LOCK    Z                       0
TP1                               8 USERSPACE1                     4 TABLE_PART_LOCK    Z                       0
TP1                               9 USERSPACE1                     5 TABLE_PART_LOCK    Z                       0
TP1                              10 USERSPACE1                     6 TABLE_PART_LOCK    Z                       0
TP1                              11 USERSPACE1                     7 TABLE_PART_LOCK    Z                       0
TP1                              12 USERSPACE1                     8 TABLE_PART_LOCK    Z                       0
TP1                              13 USERSPACE1                     9 TABLE_PART_LOCK    Z                       0
TP1                              14 USERSPACE1                    10 TABLE_PART_LOCK    Z                       0
TP1                              15 USERSPACE1                    11 TABLE_PART_LOCK    Z                       0
TP1                               4 USERSPACE1                     - TABLE_LOCK         Z                       0
TP1                               5 USERSPACE1                     - TABLE_LOCK         Z                       0
TP1                               6 USERSPACE1                     - TABLE_LOCK         Z                       0
TP1                               7 USERSPACE1                     - TABLE_LOCK         Z                       0
TP1                               8 USERSPACE1                     - TABLE_LOCK         Z                       0
TP1                               9 USERSPACE1                     - TABLE_LOCK         Z                       0
TP1                              10 USERSPACE1                     - TABLE_LOCK         Z                       0
TP1                              11 USERSPACE1                     - TABLE_LOCK         Z                       0
TP1                              12 USERSPACE1                     - TABLE_LOCK         Z                       0
TP1                              13 USERSPACE1                     - TABLE_LOCK         Z                       0
TP1                              14 USERSPACE1                     - TABLE_LOCK         Z                       0
TP1                              15 USERSPACE1                     - TABLE_LOCK         Z                       0
TP1                              16 USERSPACE1                     - TABLE_LOCK         Z                       0

  26 record(s) selected.
```

この例では、パーティション表 TP1 に対するアクセスと並行性の制御に、タイプ TABLE_LOCK のロック・オブジェクトと DATA_PARTITION_ID -1 を使用しています。 タイプ TABLE_PART_LOCK のロック・オブジェクトは、各データ・パーティションのほとんどのアクセスおよび並行性の制御に使用されます。

この出力では、他にもタイプ TABLE_LOCK のロック・オブジェクトがありますが (TAB_FILE_ID 4 から 16)、これらのロック・オブジェクトには DATA_PARTITION_ID の値がありません。この種のロック (オブジェクトがデータ・パーティションまたはパーティション表の索引に対応する TAB_FILE_ID と TBSP_NAME を持つ) は、オンライン・バックアップ・ユーティリティーとの並行性を制御するのに使用される場合があります。

ロッキングに影響を与える要因

以下の要素は、データベース・マネージャーのロックのモードおよび細分性に影響を与えます。

- アプリケーションが実行する処理のタイプ
- データ・アクセス方式
- 索引がタイプ 2 またはタイプ 1 であるかどうか
- 様々な構成パラメーター

アプリケーション・プロセスのロックとタイプ

ロックの属性を決めるためにアプリケーション処理を次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、明示的な FOR READ ONLY 節を含む SELECT ステートメント、あるいは、明示されていないものの、 PREP コマンドまたは BIND コマンドで指定された BLOCKING オプションの値に基づいて照会コンパイラーが読み取り専用と見なす SELECT ステートメントがすべて含まれます。 この処理タイプは、共用ロック (S、NS、または IS) のみを必要とします。

- 変更を意図

  このタイプには、FOR UPDATE 節、USE AND KEEP UPDATE LOCKS 節、USE AND KEEP EXCLUSIVE LOCKS 節を使用する SELECT ステートメント、または (暗黙に変更が意図された) 未確定ステートメントとして照会コンパイラーに解釈される SELECT ステートメントがすべて含まれます。このタイプは、共用および更新ロック (行では S、U、X、ブロックでは IX、U、X、S、表では IX、U、X) を使用します。

- 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。このタイプには排他ロック (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ターゲット表にデータを挿入、更新、または削除するステートメントは、2 種類の処理を行います。副選択ステートメントで戻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定されます。ターゲット表のロックは、変更処理の規則によって決定されます。

ロックとデータ・アクセス方式

アクセス・プラン とは、特定の表からデータを取得するためにオプティマイザーが選択する方式です。アクセス・プランは、ロック・モードに大きな影響を与える可能性があります。たとえば、索引スキャンを使ってある特定の行を見つける場合、オプティマイザーは表に関する行レベル・ロッキング (IS) をおそらく選択するでしょう。たとえば、従業員番号 EMPNO に関する索引が EMPLOYEE 表に含まれる場合、 1 人の従業員についての情報を選び出すために、以下の SELECT 節を含むステートメントを使用し、索引を介してアクセスできます。

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

索引を使用しない場合には、選択された行を検出するために表全体を順次にスキャン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 (性別) 列に関する索引が存在しない場合、表のスキャンを使用し、以下のような SELECT 節を含むステートメントによってすべての男性従業員を選び出すことができます。

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

注: カーソル制御される処理の場合、アプリケーションが更新または削除対象の行を見つけるために、基礎となるカーソルのロック・モードが使われます。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。

範囲クラスター表のロックインの作業は、標準キーまたは次キー・ロッキングの作業とは若干異なります。範囲クラスター表内で行範囲にアクセスする場合、範囲指定した行の一部が空であっても、範囲内のすべての行がロックされます。標準キーまたは次キー・ロッキングの場合、既存のレコードが指定されている行だけがロックされます。

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詳細は、参照表に示されています。

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、それによりロッキングのシナリオがさらに複雑になることを示唆しています。ロック獲得のタイミングおよびロックの持続性は、分離レベルに依存します。反復可能読み取り分離レベルは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が保持され、 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。

索引タイプと次キー・ロッキング

トランザクションによってタイプ 1 索引の内容が変更されるため、ある程度の次キー・ロッキングが発生します。タイプ 2 索引の場合、次キー・ロッキングの発生は最低限に抑えられます。

- タイプ 2 索引の次キー・ロッキング

  次キー・ロッキングは、索引にキーが挿入されるときに発生します。

  キーが索引に挿入されるとき、索引内で新しいキーの次のキーに対応する行が RR 索引スキャンによって現在ロックされている場合にのみ、その行がロックされます。次キーロックで使用されるロック・モードは NW です。次キーロックは、キー挿入が実際に実行される前に解放されます。キー挿入は、表に行が挿入されるときに発生します。

  さらに、行の更新によってその行の索引キー値が変更された場合にも、元のキー値が削除済みとマークされ、新しいキー値が索引に挿入されるため、キー挿入が発生します。索引の組み込み列だけに影響を与える更新の場合、キーをインプレースで更新することができ、次キー・ロッキングは発生しません。

RR スキャンの際、スキャン範囲の終わりの次に来るキーに対応する行は、 S モードでロックされます。スキャン範囲の後にキーが存在しない場合には、索引の末尾をロックするために、表末ロックが獲得されます。スキャン範囲の後に存在するキーが削除済みとマークされている場合には、削除済みとマークされていないキーが見つかるまで、対応する行のロックが保持されます (その後、見つかったキーに対応する行がロックされます)。または、索引の末尾がロックされるまで、行のロックが保持されます。

- タイプ 1 索引の次キー・ロッキング

  次キーロックは、索引での挿入や削除の際、および索引スキャンの際に発生します。表の行が更新、削除、または挿入されるとき、その行に関して X ロックが獲得されます。挿入の場合は、W ロックにダウングレードされる場合もあります。

  表索引においてキーが削除または挿入されるとき、索引内の削除または挿入対象のキーの次に来るキーに対応する行がロックされます。キー値に影響を与える更新の場合、元のキー値がまず削除された後、新しい値が挿入されるため、2 つの次キーのロックが獲得されます。これらのロックの存続期間は、次のように決定されます。

  – 索引キーの削除の場合、次キーのロック・モードは X で、ロックはコミット時まで保持されます。
  – 索引キーの挿入の場合、次キーのロック・モードは NW です。このロックは、ロックの競合がある場合にのみ獲得されます。その場合、キーが実際に索引に挿入される前に、ロックが解放されます。
  – RR スキャンの場合、索引スキャン範囲の終わりの次に来るキーに対応する表行は、 S モードでロックされ、コミット時まで保持されます。
  – CS/RS スキャンの場合、索引スキャン範囲の終わりの次に来るキーに対応する行は、 NS モードでロックされます (ロックの競合が存在する場合)。このロックは、スキャン範囲の末尾が検証されたときに解放されます。

  タイプ 1 索引で、キー挿入中またはキー削除中に、次キー・ロッキングの結果としてデッドロックが発生する可能性もあります。以下の例は、2 つのトランザクションによってデッドロックが発生する例を示しています。タイプ 2 索引の場合、このようなデッドロックは発生しません。

  以下の例では、 6 つの行に関する値 1 5 6 7 8 12 を持つ索引について考えます。

  1. トランザクション 1 はキー値 8 の行を削除します。値 8 の行が X モードでロックされます。対応するキーが索引から削除されるとき、値 12 の行が X モードでロックされます。
  2. トランザクション 2 はキー値 5 の行を削除します。値 5 の行が X モードでロックされます。対応するキーが索引から削除されるとき、値 6 の行が X モードでロックされます。
  3. トランザクション 1 が、キー値 4 の行を挿入します。この行は W モードでロックされます。新しいキーを索引に挿入しようとすると、値 6 の行が NW モードですでにロックされています。このロックは、トランザクション 2 がこの行に関して獲得している X ロックを待機しようとします。

4. トランザクション 2 が、キー値 9 の行を挿入します。この行は W モードでロックされます。新しいキーを索引に挿入しようとすると、キー値 12 の行が NW モードですでにロックされています。このロックは、トランザクション 1 がこの行に関して獲得している X ロックを待機しようとします。

タイプ 1 索引を使用する場合、このようにしてデッドロックが発生し、いずれかのトランザクションがロールバックされます。

ロック待機モードの方針を指定する

個々のセッションで、ロック待機モードの方針を指定できるようになりました。この方針は、セッションが即時に取得できないロックを必要とするときに使用します。方針は、セッションが次の処理をするかどうか示します。

- ロックを取得できないときに SQLCODE と SQLSTATE を戻す
- ロックを無限に待機する
- ロックを指定時間、待機する
- ロックを待機するときデータベース構成パラメーター *locktimeout* の値を使用する

ロック待機モードの方針は、新規の SET CURRENT LOCK TIMEOUT ステートメントで指定されます。これは特殊レジスター CURRENT LOCK TIMEOUT の値を変更します。 特殊レジスター CURRENT LOCK TIMEOUT では、ロックを取得できないことを示すエラーを戻す前にロックを待機する秒数を指定します。

伝統的なロッキング方法では、アプリケーションが互いにブロックする可能性があります。 これは、あるアプリケーションが別のアプリケーションによるロックの解放を待機しているときに発生します。 このようなブロッキングの影響を処理する方針では、通常ブロックの最大許容期間を指定するためのメカニズムを提供します。これは、ロックを取得しない場合でもアプリケーションが待機する時間です。 以前は、データベース構成パラメーター *locktimeout* の値を変更することによってデータベース・レベルでのみこの値を指定できました。

パラメーター *locktimeout* の値をすべてのロックに適用しますが、この新規関数によって影響受けるロック・タイプには、行、表、索引キー、マルチディメンション・クラスタリング (MDC) ブロック・ロックがあります。

アプリケーションのチューニング

SELECT ステートメントの制限のガイドライン

オプティマイザーは、アプリケーションが必ず SELECT ステートメントで指定されているすべての行を検索することを想定します。 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ということがよくあります。

このようなアプリケーションのパフォーマンスを改善するには、以下の方法で SELECT ステートメントを変更します。

- FOR UPDATE 節を使用して、その後に置く UPDATE ステートメントで更新できる列を指定します。
- 戻される列を読み取り専用にするには、 FOR READ/FETCH ONLY 節を使用します。
- 全結果セットの中の最初の *n* 行の検索を優先させるには、 OPTIMIZE FOR *n* ROWS 節を使用します。
- 指定された数の行だけを検索するには、 FETCH FIRST *n* ROWS ONLY 節を使用します。
- 一度に 1 つずつ行を検索するには、 DECLARE CURSOR WITH HOLD ステートメントを使用します。

注: FOR UPDATE、FETCH FIRST *n* ROWS ONLY 節、または OPTIMIZE FOR *n* ROWS 節を使用する場合、あるいはカーソルを SCROLLing として宣言する場合は、行ブロックに影響します。

以下のセクションでは、各方式のパフォーマンス上の利点について説明します。

FOR UPDATE 節

FOR UPDATE 節は、その後に置かれる UPDATE ステートメントで更新できる列だけを組み込むことによって、結果セットを制限します。 FOR UPDATE 節を列名なしで指定する場合は、表またはビュー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、表またはビューの 1 つの列を識別している必要があります。

以下の場合は、FOR UPDATE 節は使用できません。
- SELECT ステートメントに関連したカーソルを削除できない場合。
- 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 節に含まれている場合。

CLI アプリケーションでは、 DB2 CLI 接続属性 SQL_ATTR_ACCESS_MODE を同じ目的に使用してください。

FOR READ または FETCH ONLY 節

FOR READ ONLY 節または FOR FETCH ONLY 節は、戻される結果を読み取り専用にします。読み取り専用として定義されているビューでは SELECT の結果表も読み取り専用なので、この節は許可されていますが、何の効果もありません。

データベース・マネージャーが排他ロックの代わりにデータのブロックを検索できる場合、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。指定した UPDATE または DELETE ステートメントで使用されている照会には、 FOR READ ONLY 節は使用しないでください。

CLI アプリケーションでは、 DB2 CLI 接続属性 SQL_ATTR_ACCESS_MODE を同様の目的に使用することができます。

OPTIMIZE FOR n ROWS 節

OPTIMIZE FOR 節は、結果のサブセット 1 つだけを検索するのが目的なのか、または最初の数行だけの検索を優先的に行うのが目的なのかを宣言します。そうすると、オプティマイザーは、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るようになります。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は、OPTIMIZE FOR 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 節はサーバーがデータベースから修飾行を検索する方法と、修飾行をクライアントに戻す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。オプティマイザーは、多数のランダム同期入出力が行われないように、すべての修飾行の行 ID のソートを必要とするリスト・プリフェッチ・アクセス方式を使用することを選択します。このソートのため、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。この遅延を防止するため、以下のようにステートメントに OPTIMIZE FOR 節を追加してください。

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

この場合、最高給を得ている 20 人の従業員だけが検索されるので、オプティマイザーは、おそらく、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 節が指定されている場合、オプティマイザーは、大量データ操作を行わない、あるいはソートなどの行のフローを妨げるアクセス・プランを使用しようとします。 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。この節の使用には、以下の効果があります。

- 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ます。 NESTED LOOP 結合は、オーバーヘッド・コストが少なく、通常、少数の行を検索するのにより効果的なので、最もよく選択されます。
- ORDER BY にはソートが必要ないので、 ORDER BY 節に一致する索引が選ばれやすい。
- リスト・プリフェッチはソート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。
- 少数の行だけが必要であると理解しているので、順次プリフェッチが選ばれることは少ない。
- 結合照会では、外部表の索引が ORDER BY 節に必要な ORDER BY を指定している場合は、 ORDER BY 節にある列で成る表が外部表として選ばれやすい。

OPTIMIZE FOR 節はすべての最適化レベルに適用されますが、 3 より下のクラスは貪欲型結合列挙方法を使用するので、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。貪欲型結合列挙方法を使用すると、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 節では、修飾行をすべて検索します。すべての修飾行を検索する場合、合計経過時間は、オプティマイザー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。

パッケージ・アプリケーションがコール・レベル・インターフェース (DB2 CLI または ODBC) を使用している場合は、 db2cli.ini 構成ファイルにある OPTIMIZEFORNROWS キーワードを使用して、 DB2 CLI に OPTIMIZE FOR 節を各照会ステートメントの終了に自動的に付加させることができます。

データがニックネームから選択されるとき、結果はデータ・ソース・サポートによって異なります。ニックネームによって参照されるデータ・ソースが OPTIMIZE FOR 節をサポートしており、 DB2 オプティマイザーが照会全体をデータ・ソースにプッシュダウンする場合、この節はデータ・ソースに送られたリモート SQL 内で生成されます。データ・ソースでこの節がサポートされていない場合、またはオプティマイザーが、ローカルな実行が最もコストの低いプランであると判別した場合、 OPTIMIZE FOR 節はローカルに適用されます。この場合、DB2 オプティマイザーは、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、プランの生成にオプティマイザーが利用できるオプションははっきり限定されず、 OPTIMIZE FOR 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 節と OPTIMIZE FOR 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・サイズに影響します。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。

FETCH FIRST n ROWS ONLY 節

FETCH FIRST *n* ROWS ONLY 節は、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。 制限しない場合に結果セットに含まれる行数にかかわりなく、 *n* 行だけが検索されます。

FETCH FIRST 節と OPTIMIZE FOR 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・サイズに影響します。最適化という目的のため、この 2 つの値は互いに独立しています。

DECLARE CURSOR WITH HOLD ステートメント

WITH HOLD 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、オープン・カーソルは開いたままの状態になり、オープン WITH HOLD カーソルの現行のカーソル位置を保護しているロック以外のロックはすべて解放されます。

トランザクションがロールバックされると、オープン・カーソルはすべてクローズされ、すべてのロックが解放されて LOB ロケーターが解放されます。

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。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 節が指定されているものと自動的に想定させてください。

オーバーヘッド削減のための行ブロッキングの指定

行のブロッキングは、すべてのステートメントおよびデータ・タイプ (LOB データ・タイプを含む) でサポートされます。行のブロッキングは、単一の操作で複数の行からなるブロック を取り出し、カーソルに対するデータベース・マネージャーのオーバーヘッドを削減します。

注: 指定する行のブロックは、メモリー内のページ数になります。これは、ディスク上のエクステントに物理的にマップされるマルチディメンション (MDC) 表ブロックではありません。

行ブロッキングは、BIND または PREP コマンドの以下の引数によって指定されます。

UNAMBIG
FOR READ ONLY 節で指定されたカーソルの場合、ブロッキングが発生します。

FOR READ ONLY または FOR UPDATE 節で宣言されていない、*未確定* ではなく*読み取り専用* であるカーソルはブロック化されます。*未確定* カーソルはブロック化されません。

ALL FOR READ ONLY 節で指定され、FOR UPDATE 節で指定されていないカーソルの場合、ブロッキングが発生します。

NO どのカーソルの場合もブロッキングは行われない。

読み取り専用カーソルおよび未確定カーソルの定義について詳しくは、『DECLARE CURSOR ステートメント』を参照してください。

注: SELECT ステートメントの中で、FETCH FIRST *n* ROWS ONLY 節、または OPTIMIZE FOR *n* ROWS 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。

- 上記の公式で計算される値
- FETCH FIRST 節の *n* の値
- OPTIMIZE FOR 節の *n* の値

2 つのデータベース・マネージャー構成パラメーターは、正しく設定する必要があります。値は両方とも、メモリーのページ数として設定されます。これらのパラメーターの値は、ブロック・サイズの計算に使用できるよう、記録して置いてください。

- データベース・マネージャー構成パラメーター *aslheapsz* は、ローカル・アプリケーションのアプリケーション・サポート層ヒープ・サイズを示します。

- データベース・マネージャー構成パラメーター *rqrioblk* は、リモート・アプリケーションとデータベース・サーバー上のデータベース・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。

LOB データ・タイプに対して行データのブロッキングを有効にする前に、システム・リソースに対する影響について理解しておくことは重要です。 LOB 列が戻されたときに、サーバーでは LOB 値への参照を各データ・ブロックに保管するために多くの共有メモリーが消費されます。こうした参照の数は、データベース構成パラメーター *rqrioblk* の値に従って変化します。

ヒープに割り当てられるメモリーの量を増やすには、以下のようにして *database_memory* データベース構成パラメーターを変更します。

- パラメーターを AUTOMATIC に設定すると、データベースのメモリーを自動的に管理するようにデータベース・マネージャーに指示が出されます。
- 現在、パラメーターがユーザー定義の数値に設定されている場合は、256 ページずつ値を増やします。

LOB 値を参照する既存の組み込み SQL アプリケーションのパフォーマンスを改善するには、BIND コマンドを使用して、BLOCKING ALL 節または BLOCKING UNAMBIGUOUS 節のいずれかを指定してブロッキングを要求することにより、アプリケーションを再バインドすることができます。組み込みアプリケーションは、行のブロックがサーバーから取り出されてから、サーバーから一度に 1 行ずつ LOB 値を取り出します。 UDF が大量の LOB を結果として戻した場合、DB2 はサーバーで大量のメモリーが消費されている LOB データの単一行の取り出しに戻る可能性があります。

行ブロッキングを指定するには、次のようにします。

1. *aslheapsz* および *rqrioblk* 構成パラメーターの値を使用して、各ブロックに対して戻される行数を見積もる。どちらの公式でも、 *orl* は出力行の長さ (バイト単位) です。
   - ローカル・アプリケーションには以下の公式を使用します。

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

     1 ページあたりのバイト数は、4 096 です。
   - リモート・アプリケーションには以下の公式を使用します。

     ```
     Rows per block = rqrioblk / orl
     ```
2. 行ブロッキングを使用可能にするため、 PREP または BIND コマンドの BLOCKING オプションに適切な引数を指定します。

   BLOCKING オプションを指定しない場合、デフォルトの行ブロッキング・タイプは UNAMBIG です。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キングは ALL です。

照会チューニングのガイドライン

照会チューニングの指針に従って、アプリケーション・プログラムの SQL および XQuery ステートメントを細かく調整してください。この指針は、使用するシステム・リソースと、大規模な表および複雑な照会から値を戻すために必要な時間を最小限にとどめることを目的としています。

注: オプティマイザーが使用する最適化クラスにより、照会コンパイラーが SQL および XQuery コードをより効率的な形式に再書き込みできるために、細かなチューニングは不要であることもあります。

オプティマイザーによるアクセス・プランの選択は、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ご注意ください。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・テストを行えば、どのような調整によってアクセス・プランが改善できるかを知ることができます。

REOPT BIND オプションを使用した照会最適化

ホスト変数、特殊レジスター、グローバル変数、またはパラメーター・マーカーを持つ静的および動的 SQL および XQuery ステートメントの照会最適化 (または再最適化) を有効にするには、REOPT BIND オプションを使ってパッケージをバインドしてください。このオプションを使用すると、パッケージに属し、かつホスト変数、パラメーター・マーカー、グローバル変数、または特殊レジスターを含んだ SQL または XQuery ステートメントのアクセス・パスが、コンパイラーの選ぶデフォルトの推定値ではなく、これらの変数の実際の値を使って最適化されます。照会の実行時に値が与えられてから、最適化は実行されます。

REOPT を指定してバインディングすることによりパフォーマンスを向上させる

パラメーター・マーカー、ホスト変数、グローバル変数、および特殊レジスターなどの入力変数に使用される値が、デフォルトのフィルター係数の推定値の予測範囲外の場合、SQL または XQuery 照会の実行中のパフォーマンスが不十分になることがあります。デフォルトのフィルター係数は、実データ値が不明なシナリオで使用されるものであり、実行時 (実データ値を使用する) に実際に戻される行数の推定値です。

REOPT BIND オプションは、DB2 がホスト変数、パラメーター・マーカー、グローバル変数、および特殊レジスターの値を使用して実行時にアクセス・パスを最適化するようにするかどうかを指定します。 REOPT 値は、BIND、PREP、または REBIND コマンドの以下の引数によって指定されます。

REOPT NONE
ホスト変数、パラメーター・マーカー、グローバル変数、または特殊レジスターを含んだ所定の SQL または XQuery ステートメントのアクセス・パスを、これらの変数の実際の値を使って最適化しません。その代わり、それらの変数のデフォルトの見積もり値が使用されます。このプランをキャッシュに入れて、それ以降使用します。これがデフォルトの動作です。

REOPT ONCE
所定の SQL または XQuery ステートメントのアクセス・パスを、初めて照会を実行するときに、ホスト変数、パラメーター・マーカー、グローバル変数、または特殊レジスターの実際の値を使って最適化します。このプランをキャッシュに入れて、それ以降使用します。

REOPT ALWAYS
所定の SQL または XQuery ステートメントのアクセス・パスを、毎回の実

行時に、ホスト変数、パラメーター・マーカー、グローバル変数、または特殊レジスターの値を使って、いつもコンパイルおよび再最適化します。

SQL および XQuery 照会でのデータ・サンプリング

データベースは非常に大きくなり、これらのデータベースに対する照会も非常に複雑になっているので、 1 つの照会に関連したすべてのデータを検索することは実際的でない (あるいは、不必要な) 場合があります。ユーザーは全体的な傾向やパターンだけに関心があるかもしれません。この場合、一定の誤差の範囲内でおおまかな応答を出すだけで十分です。このような照会の速度を上げる 1 つの方法は、データベースの無作為標本 (ランダム・サンプル) に対して照会を実行することです。DB2 では、SQL および XQuery 照会によって効率的にデータ・サンプリングを実行できます。これによって、高度の正確さを維持しながら、非常に大きな照会のパフォーマンスを何十倍も改善することができます。

サンプリングの最も一般的な適用分野は、AVG、SUM、 COUNT などの集約照会です。この場合、データのサンプルからある程度正確な集約結果が得られます。監査のために表の実際の行からランダム・サブセットを取得する際にも、またデータ・マイニングおよび分析の操作の速度を上げるためにも、サンプリングを使用できます。

DB2 では、行レベルおよびブロック・レベルの 2 つのサンプリング方式が提供されています。

行レベルのベルヌーイ・サンプリング

行レベルのベルヌーイ・サンプリングは、表の行の P パーセントのサンプルを取ります。その際、各行を P/100 の確率でサンプルに組み込み、 1-P/100 の確率で除外する検索引数述部を使用します。

行レベルのベルヌーイ・サンプリングによって、データ・クラスタリングにかかわらず、常に有効な無作為標本が得られます。ただし、索引を使用できない場合、この種のサンプリングのパフォーマンスは低くなる可能性があります。すべての行を検索して、サンプリング述部を適用する必要があるためです。索引が存在しない場合、サンプリングしない照会の実行に比べて、 I/O はまったく節約されません。索引が使用される場合には、索引リーフ・ページ内の RIDS に対してサンプリング述部が適用されるため、この種のサンプリングのパフォーマンスは改善します。これには、通常、選択された RID ごとに 1 つの I/O、および索引リーフ・ページごとに 1 つの I/O が必要です。

システム・ページ・レベルのサンプリング

システム・ページ・レベルのサンプリングは、行ではなくページのサンプルを取るという点を除いて、行レベルのサンプリングと同じです。あるページがサンプルに組み込まれるかどうかは、P/100 の確率で決定されます。ページが組み込まれる場合、そのページに含まれるすべての行が組み込まれます。

サンプルに組み込まれるページごとに 1 つの I/O だけが必要であるため、システム・ページ・レベルのサンプリングは非常に高いパフォーマンスを実現します。サンプリングしない場合に比べて、ページ・レベルのサンプリングはパフォーマンスを数十倍も改善します。ただし、集約見積もりの正確さは、行レベルのサンプリン

グに比べてページ・レベルのサンプリングの方が悪いという傾向があります。ブロックあたりの行数が多い場合や、ページ内で高いレベルのクラスター化が行われている列を照会が参照している場合には、その格差が最も顕著です。

特定のタスクに最適のサンプリング方式は、ユーザーの時間制約および求められている正確さの多重度によって決まります。

サンプリング方式の指定

表からのデータの無作為標本に対して照会を実行するには、 SQL ステートメントの表参照節で TABLESAMPLE 節を使用できます。サンプリング方式を指定するには、キーワード BERNOULLI または SYSTEM を使用します。

キーワード BERNOULLI は、行レベルのベルヌーイ・サンプリングが実行されることを指定します。

キーワード SYSTEM は、システム・ページ・レベルのサンプリングが実行されることを指定します。ただし、行レベルのベルヌーイ・サンプリングの方が効率的だとオプティマイザーが判断した場合には、そちらが実行されます。

アプリケーションの並列処理

DB2 は、主に対称マルチプロセッサー (SMP) マシン上で並列環境をサポートしますが、単一プロセッサー・マシン上でも限られた範囲でサポートします。SMP マシンでは、データベースに複数のプロセッサーがアクセスでき、複雑な SQL 要求の実行を複数のプロセッサー間で分け合って、並列に実行することができます。

アプリケーションのコンパイル時にインプリメントする並列処理の多重度を指定するには、 CURRENT DEGREE 特殊レジスターまたは DEGREE BIND オプションを使用します。*程度* とは、並行して実行する照会のパーツの数を指します。プロセッサーの数と並列処理の多重度として選択された値との間には、厳密な関係はありません。マシン上のプロセッサーよりも多い数または少ない数を指定することもできます。単一プロセッサー・マシンの場合も、 1 より高い程度を設定して何らかの方法でパフォーマンスを改善できます。しかし、並列処理の多重度が高くなれば、システム・メモリーと CPU のオーバーヘッドが増えるという点にご注意ください。

照会の並列実行の使用時にパフォーマンスを最適化するには、いくつかの構成パラメーターを変更する必要があります。特に並列処理の多重度の高い環境では、共用メモリーとプリフェッチの量を制御する構成パラメーターを検討して変更する必要があります。

以下の 3 つの構成パラメーターは、パーティション内並列処理を制御および管理します。

- *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処理の多重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 BIND オプションをオーバーライドします。

- *dft_degree* データベース構成パラメーターは、 CURRENT DEGREE 特殊レジスターおよび DEGREE BIND オプションのデフォルト値を設定します。

照会で DEGREE = ANY を指定してコンパイルすると、データベース・マネージャーによってパーティション内並列処理の多重度が選ばれます。この程度は、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、係数の値およびシステム上のアクティビティーの量によっては、実際に実行時に使用される多重度の値がプロセッサーの数よりも少ない場合があります。システムが酷使されている場合は、照会の実行のまえに並列処理が低められる可能性があります。なぜなら、パーティション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・リソースを酷使し、それが他のデータベース・ユーザーのパフォーマンスに悪影響を及ぼすからです。

SQL オプティマイザーによって選択された並列処理の多重度を表示するには、 SQL Explain 機能を使用してアクセス・プランを表示します。実行時に実際に使用されている並列処理の多重度に関する情報を表示するには、データベース・システムのモニターを使用します。

非 SMP 環境での並列処理

SMP マシンがなくても、並列処理の多重度を指定できます。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 度以上を宣言しておいた方が有利です。この場合、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、次の照会の処理を開始できます。しかし、2 度以上を宣言しても、単一プロセッサー・マシン上の入出力並列処理は制御されません。 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、入出力並列処理を制御することができます。キーワード ANY は、 *dft_degree* 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ためにも使用できます。オプティマイザーは、ANY キーワードによって、パーティション内並列処理の多重度を判別することができます。

第 19 章 環境に関する考慮事項

表スペースが照会の最適化に与える影響

表スペースの特性のうちのあるものは、照会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、照会オプティマイザーは、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。オプティマイザー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。

    以下の公式を使って、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)
    ```

    詳細は次のとおりです。

    - 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。

    例えば、ディスクの 1 分当たりの回転数が 7 200 であるとします。回転待ち時間の公式を使用すると、次のようになります。

    ```
    (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、以下の公式を使って、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 / 1024000 * page_size
```

詳細は次のとおりです。

- 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 (例えば、4 KB 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を考慮する必要があります。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。順次スキャン中に、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、順次スキャンは 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。上記の公式の spec_rate をこの計算結果で置き換えます。例えば、4 つのコンテナー表スペースの中の表をスキャン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 つの値によって提供される入出力コスト情報は、いくつかの方法でオプティマイザー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。

- プリフェッチ

  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、オプティマイザーは、ディスクからデータおよび索引ページをプリフェッチすることによって、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プリフェッチを行うと、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。

  オプティマイザーは SYSCAT.TABLESPACES の PREFETCHSIZE 列および EXTENTSIZE 列の情報を使用して、表スペースに対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、および/または ALTER TABLESPACE ステートメントを使用するときに設定できます。デフォルトのプリフェッチ・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。このパラメーターのサイズ変更に関する推奨を検討して、データの移動を改善するために必要な変更を行ってください。

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、アプリケーションを再バインドすること、および、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ことを考慮してください。

フェデレーテッド・データベースに影響を与えるサーバー・オプション

フェデレーテッド・システムは、DB2 DBMS (フェデレーテッド・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フェデレーテッド・データベースに識別されます。このステートメント発行の際に、サーバー・オプションを含めることもできます。このサーバー・オプションにより DB2 と指定したデータ・ソースに関係するフェデレーテッド・システムのさまざまな動作を細かく調整、制御することができます。後でサーバー・オプションを変更するには、 ALTER SERVER ステートメントを使用します。

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、分散結合のインストール・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。

指定するサーバー・オプションの値は、照会のプッシュダウン分析、グローバル最適化、およびフェデレーテッド・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、 *cpu_ratio* オプションのように、サーバー・オプションの値としてパフォーマンス統計を指定することができます。これは、データ・ソースおよびフェデレーテッド・サーバーにおける CPU の相対速度を指定します。また、ソースとフェデレーテッド・サーバーのデータ入出力装置の相対速度を表す値を *io_ratio* として設定できます。 CREATE SERVER ステートメントを実行すると、このデータはカタログ・ビュー SYSCAT.SERVEROPTIONS に追加され、オプティマイザーはデータ・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。統計情報が変更された場合 (たとえばデータ・ソース CPU がアップグレードされた場合) などには ALTER SERVER ステートメントを使用して、 SYSCAT.SERVEROPTIONS の設定に変更を反映してください。次回以降、オプティマイザーはデータ・ソースへのアクセス・プランを選択する際に、その新規の情報を使用します。

第 20 章 カタログ統計

照会コンパイラーが照会プランを最適化する際には、データベースの表、索引、および統計ビューのサイズに関する統計情報が、コンパイラーの判断にかなり影響します。さらに、表、索引、および統計ビューの特定の列が行の選択や表の結合に使用される場合、オプティマイザーはそれらの列でのデータ分布情報も使用します。この情報は、照会ごとの代替アクセス・プランのコストを見積もるために使用されます。

表サイズおよびデータ分散情報に加えて、索引のクラスター比率、索引内のリーフ・ページ数、元のページからオーバーフローする表の行数、および表内の入力されているページ数と空のページ数についての統計情報も収集できます。この情報は、表および索引を再編成する時期を決定するために使用します。

パーティション・データベース環境で表の統計を収集する場合、統計はユーティリティーが実行されるデータベース・パーティション上にある表の部分についてのみ収集されるか、表を含むデータベース・パーティション・グループ内の最初のデータベース・パーティションについて収集されます。統計ビューの統計を収集する場合、統計は全データベース・パーティションについて収集されます。

表、統計ビュー、または表およびそれに関連する索引で RUNSTATS ユーティリティーを実行する場合、以下の種類の統計情報がシステム・カタログ表に保管されます。

表および索引の場合

- 使用中のページの数
- 行を含んでいるページの数
- オーバーフローしている行の数
- 表内の行の数 (カーディナリティー)
- MDC 表の場合、データを含んでいるブロックの数
- パーティション表の場合、単一のデータ・パーティション内でのデータ・クラスタリングの程度

表または統計ビュー内の各列、および索引キーの中の最初の列の場合

- 列のカーディナリティー
- 列の平均の長さ
- 列内で 2 番目に高い値
- 列内で 2 番目に低い値
- 列内の NULL の数

各 XML 列ごとに、以下の統計情報が収集されます。 XML 列の各行は、XML 文書を保管します。指定されたパスまたはパスと値のペアのノード・カウントは、パスまたはパスと値のペアによって到達可能なノードの数のことです。指定されたパスまたはパスと値のペアの文書数は、指定されたパスまたはパスと値のペアを含む文書の数のことです。

- NULL の XML 文書の数
- 非 NULL の XML 文書の数
- 異なるパスの数
- 異なるパスごとのノード・カウントの合計
- 異なるパスごとの文書数の合計
- 最大のノード・カウントを持つ (パス、ノード・カウント) の *k* ペア
- 最大の文書数を持つ (パス、文書数) の *k* ペア
- 最大のノード・カウントを持つ (パス、値、ノード・カウント) の *k* トリプル
- 最大の文書数を持つ (パス、値、文書数) の *k* トリプル
- テキストまたは属性値へ導く異なるパスごとに、以下のとおりです。
  - このパスが取る可能性のある別個の値の数
  - 最大値
  - 最小値
  - テキストまたは属性ノードの数
  - テキストまたは属性ノードを含む文書の数

指定した列のグループの場合
- 列グループのタイム・スタンプに基づいた名前
- 列グループのカーディナリティー

索引のみの場合
- 索引項目の数 (索引のカーディナリティー)
- リーフ・ページの数
- 索引レベルの数
- この索引への表データのクラスタリングの程度。
- データ・パーティションに関しての索引キーのクラスタリングの程度。
- 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、索引キーの順序のディスク上のリーフ・ページの数の比率
- 索引の最初の列の固有値の数
- 索引の最初、2 番目、3 番目、4 番目の列の固有値の数
- 索引のすべての列の固有値の数
- 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リーフ・ページの数
- すべての RID が削除対象としてマークされているページの数
- 一部の RID が削除対象としてマークされているページで、削除対象としてマークされている RID の数

索引に関する詳細な統計を要求した場合、索引に対する表のクラスタリングの程度、およびさまざまなバッファー・サイズに関するページ・フェッチの見積もりに関する、より優れた情報も保管します。

表および索引に関する以下の種類の統計も収集できます。
- データ分散統計

オプティマイザーはデータ分散統計を使用して、データが均等に分布していない表および統計ビュー、および列に膨大な数の重複値がある表および統計ビューに関して効果的なアクセス・プランを見積もります。

- 詳細索引統計

  オプティマイザーは詳細な索引統計を使用して、索引を介した表へのアクセスの効果性を判別します。

- サブエレメント統計

  オプティマイザーは、 LIKE 述部でのサブエレメント統計、特にストリング内に組み込まれたパターンを検索するもの (`LIKE %disk%` など) を使用します。

以下の場合、分散統計は収集されません。

- 構成パラメーター *num_freqvalues* および *num_quantiles* をゼロ (0) に設定している場合。
- データの分散が分かっている場合。たとえば、各データ値が固有である場合など。
- 列が、統計が収集されないデータ・タイプである場合。該当するデータ・タイプは、LONG、ラージ・オブジェクト (LOB)、または構造化列です。
- 副表の行タイプの場合、表レベルの統計 NPAGES、FPAGES、および OVERFLOW は収集されません。
- 変位値分散が要求されたが、列の中に非 NULL 値が 1 つしかない場合。
- 拡張索引または宣言済み一時表の場合。

注: 宣言済み一時表に対して RUNSTATS を実行することはできますが、宣言済み一時表にはカタログ項目がないため、結果の統計はシステム・カタログに保管されません。ただし、宣言済み一時表のカタログ情報を表すメモリー構造に保管されます。したがって、ときには、これらの表に対して RUNSTATS を実行することが役立つ場合もあります。

自動統計収集

DB2 オプティマイザーは、あらゆる照会に関し、最も効果的なアクセス・プランを判別するためにカタログ統計を使用します。ある表または索引に関する統計が古い、あるいは不完全であるなら、オプティマイザーが最適でないプランを選択したり、照会の実行がスローダウンしたりする可能性があります。しかし、特定のワークロードのために収集する統計を決めて、それらの統計を最新に保つことには、時間がかかります。

DB2 の自動表保守フィーチャーの一部である自動統計収集を使えば、データベース統計の更新が必要かどうかを DB2 データベース・マネージャーに判断させることができます。自動統計収集は、リアルタイム統計フィーチャーを使用してステートメント・コンパイル時に行うこともできるし、あるいは RUNSTATS ユーティリティーをバックグラウンドで実行することによって行うこともできます。バックグラウンド統計収集は、リアルタイム統計収集が使用不可な場合にも有効にすることができます。リアルタイム統計収集を有効にするには、バックグラウンド統計収集を使用可能にする必要があります。DB2 バージョン 9 以降では、新しいデータベースの作成時に、バックグラウンド自動統計収集はデフォルトで有効になります。

DB2 バージョン 9.5 以降では、新しいデータベースの作成時に、バックグラウンド自動統計収集とリアルタイム統計の両方がデフォルトで有効になります。

バックグラウンド統計収集およびリアルタイム統計収集について

自動統計は、同期的にも非同期的にも (RUNSTATS ユーティリティーを実行することにより) 収集できます。*非同期*収集は、バックグラウンドで実行されます。リアルタイム統計フィーチャーが使用可能になると、ステートメント・コンパイル時に統計を*同期的*に収集することもできます。リアルタイム統計が使用可能である場合は、データおよび索引マネージャーが維持するメタデータを使用して統計を作成することも可能です。照会オプティマイザーは、照会のニーズおよび表更新アクティビティーの量に基づいて統計を収集する方法を決定します。表更新アクティビティーは、更新、削除、および挿入の数によって測定されます。

リアルタイム統計は、SQL ステートメントが最適化される前に、そのニーズによって決定されます。こちらの方がよりタイムリーな統計収集となり、より正確な統計となります。正確な統計を収集できると、照会実行プランをより良いものにできますし、パフォーマンスを向上させることが可能です。リアルタイム統計が使用可能でないと、2 時間間隔で非同期統計収集が行われます。一部のアプリケーションでは、正確な統計を出すためには、この間隔では不十分な場合があります。

リアルタイム統計が使用可能な場合にも、非同期統計収集の検査は引き続き 2 時間間隔で行われます。また、以下の場合にもリアルタイム統計の非同期収集要求が開始します。

- 表アクティビティーが同期収集を必要とするほど高くはないものの、非同期収集を十分必要とする高さである場合。
- 表が大きいために、同期統計収集でサンプリングを使用した場合。
- 同期統計が作成されたものである場合。
- 同期統計収集の収集時間が超過したために失敗した場合。

非同期要求は多くて 2 つ同時に処理できますが、それは対象となる表が異なる場合に限ります。一方の要求はリアルタイム統計によって開始され、もう一方の要求は非同期統計収集の検査によって開始されます。

自動統計収集によるパフォーマンスへの影響は、以下のいくつかの方法によって最低限に抑えられています。

- 非同期統計収集は、スロットル調整した RUNSTATS ユーティリティーを使用して実行されます。スロットル調整することにより、現在のデータベースのアクティビティーに基づいて、RUNSTATS ユーティリティーの消費するリソースの量が制御されます。データベースのアクティビティーが増加すると RUNSTATS ユーティリティーの実行速度が低下し、リソースの要求が小さくなります。
- 同期統計収集は、照会 1 つにつき 5 秒に制限されています。この値は、RTS 最適化ガイドラインによって制御できます。同期収集が時間制限を超えると、非同期収集要求が送信されます。
- 同期統計収集では、統計はシステム・カタログ内に格納されません。代わりに、統計は統計キャッシュに格納され、後ほど非同期操作によってシステム・カタログ内に格納されます。これにより、システム・カタログの更新に関係したオーバ

ーヘッドを回避できますし、ロック競合の可能性も回避できます。統計キャッシュ内の統計は、後続の SQL コンパイル要求で使用できます。

- 1 つの表で行われる同期統計収集は、1 つだけです。同期統計収集を必要とする他のエージェントは、可能な場合には統計を作成し、引き続きステートメント・コンパイルを行います。この動作は DPF 環境でも強制されます。この環境では、異なるデータベース・パーティション上にあるエージェントが同期統計を必要とする場合があります。
- デフォルトでは、同期および非同期操作で収集される統計は、分散情報を含む基本表統計やサンプリングを使用した詳細な索引統計です (RUNSTATS コマンドを、WITH DISTRIBUTION オプションおよび SAMPLED DETAILED INDEXES ALL オプションを指定して発行します)。この収集する統計のタイプは、統計プロファイルを使用可能にすることによってカスタマイズできます。統計プロファイルでは、直前のデータベース・アクティビティーに関する情報を使用してデータベースのワークロードに必要な統計を決定します。また、特定の表に関する独自の統計プロファイルを作成して、その表に対して収集される統計のタイプをカスタマイズすることもできます。
- 統計が欠落している表またはアクティビティーのレベルが高い表 (更新、削除、および挿入の数で測る) だけを、統計収集の対象として考慮します。ただし表が統計収集の基準を満たしている場合であっても、照会を最適化するために必要とされない限りは同期統計は収集されません。場合によっては、照会オプティマイザーは統計なしでアクセス・プランを選択することもできます。
- 非同期統計収集の検査に関しては、大きい表 (構成ページが 4000 を超える表) の場合は、表に対する大量のアクティビティーによって確かに統計が変化したかどうかを判断するためにサンプリングします。変化が確実な場合にだけ、このような大きな表の統計は収集されます。
- 非同期統計収集の場合、メンテナンス・ポリシー定義で指定される最適なメンテナンス時間帯に実行されるように、RUNSTATS ユーティリティーは自動でスケジューリングされます。このポリシーはまた自動統計収集を有効にする表のセットも指定するので、不必要なリソース消費をさらにおさえます。
- 同期統計収集および作成は、メンテナンス・ポリシー定義で指定されたメンテナンス時間帯には従いません。なぜなら、同期要求は直ちに生じますし、収集時間が制限されているからです。同期統計収集および作成は、自動統計収集の有効範囲内の表集合を指定するポリシーに従います。
- 自動統計収集の実行中も、影響を受ける表に対し、あたかも RUNSTATS コマンドがその表で実行されていないかのように、引き続き通常のデータベース・アクティビティー (更新、挿入、削除) が可能です。
- 非同期統計収集の場合、SYSPROC.NNSTAT ストアード・プロシージャーは、ニックネーム統計を自動的にリフレッシュするカタログ・ベースの収集メソッドを使用して実行されます。リアルタイム統計 (同期、または作成されたもの) は、ニックネームに関しては収集されません。

リアルタイム同期統計収集の実行対象となるのは、通常表、マテリアライズ照会表 (MQT)、および宣言済みのグローバル一時表 (DGTT) です。

DGTT に関しては、非同期統計は収集されません。つまり、リアルタイム統計プロセスは DGTT に対する非同期要求を開始しません。

自動統計収集 (同期または非同期) は、以下の対象に対しては行われません。

- 統計ビュー
- VOLATILE というマークが付けられている表 (SYSCAT.TABLES に VOLATILE フィールドが設定されている表)
- SYSSTAT カタログ・ビューに対して UPDATE ステートメントを直接発行して、統計が手動で更新された表

  統計が手動で変更された場合、DB2 はユーザーが表の統計を保守していると見なします。そのため、DB2 はその表の統計を保守しません。手動で更新された統計のある表の統計を DB2 に保守させるには、その表に対して手動で RUNSTATS を発行してください。統計を手動で更新した後にマイグレーションした表は、統計が DB2 によって自動的に保守されることになります。

データベース・パーティション・フィーチャー (DPF) 環境では、統計は単一のデータベース・パーティションで収集されてから、外挿されます。1 つの表が複数のデータベース・パーティションに存在し、その表に統計がない場合、DB2 データベース・マネージャーは、必ずデータベース・パーティション・グループの最初のデータベース・パーティションで統計 (同期および非同期の両方) を収集します。表に既存の統計があると、それらの統計が最後に収集されたデータベース・パーティション上で統計は収集されます。これにより、必ず同じデータベース・パーティション上で統計が収集されるので、統計の整合性が保たれます。

データベースがアクティブにされてから最低 5 分間は、リアルタイム統計収集アクティビティーは行われません。

リアルタイム統計が有効になったなら、定義済みのメンテナンス時間帯をスケジュールしなければなりません。デフォルトでは、メンテナンス時間帯は定義されておらず、すべての時間帯が含まれるようになっています。メンテナンス時間帯が定義されていないと、同期統計収集だけが行われます。その場合、収集される統計はメモリー内に限定され、小さな表の場合を除いて、通常はサンプリングを使用して収集されます。

リアルタイム統計プロセスは、静的および動的 SQL の両方で生じます。

IMPORT コマンドを使用して切り捨てられた表は、失効した統計を持っているものとして自動的に認識されます。

自動統計収集 (同期と非同期の両方) は、統計収集の対象の表を参照する、キャッシュに入れられた動的ステートメントを無効にします。このようにすることで、キャッシュに入れられた動的ステートメントを最新の統計で再最適化できます。

リアルタイム統計および Explain プロセス

Explain 機能を使用して Explain しただけの照会 (実行していないもの) に、リアルタイム処理はありません。以下の表では、CURRENT EXPLAIN MODE レジスターの種々の値の動作を要約しています。

表 51. CURRENT EXPLAIN MODE レジスターのリアルタイム統計収集の動作

CURRENT EXPLAIN MODE	リアルタイム統計収集を考慮するかどうか
YES	はい

表 51. CURRENT EXPLAIN MODE レジスターのリアルタイム統計収集の動作 (続き)

CURRENT EXPLAIN MODE	リアルタイム統計収集を考慮するかどうか
EXPLAIN	いいえ
NO	はい
REOPT	はい
RECOMMEND INDEXES	いいえ
EVALUATE INDEXES	いいえ

自動統計収集および統計キャッシュ

同期的に収集された統計をすべての照会で利用できるようにするために、DB2 バージョン 9.5 では統計キャッシュが導入されました。このキャッシュは、カタログ・キャッシュの一部です。データベース・パーティション・フィーチャー (DPF) 環境では、このキャッシュはカタログ・データベース・パーティションにのみ置かれます。カタログ・キャッシュは同じ SYSTABLES オブジェクトの複数の項目を格納できます。これにより、すべてのデータベース・パーティションのカタログ・キャッシュ・サイズが増加します。リアルタイム統計が有効な場合は、**CATALOGCACHE_SZ** データベース構成パラメーター値を大きくすることを考慮してください。

DB2 バージョン 9 以降では、新しいデータベースの初期データベース構成を決定するのに構成アドバイザーを使用します。構成アドバイザーは、**AUTO_STMT_STATS** 構成パラメーターを ON に設定することを常に推奨します。

自動統計収集および統計プロファイル

同期統計および非同期統計は統計プロファイル (事実上、表に対するもの) に基づいて収集されますが、以下の例外があります。

- 同期統計収集のオーバーヘッドを最小化するため、DB2 データベース管理システムはサンプリングを使用して統計を収集する場合があります。その場合のサンプリング率とサンプリング方法は、統計プロファイルで指定されているものとは異なる可能性があります。
- 同期統計収集によって、統計が作成される場合があります。統計プロファイルで指定されている統計をすべて作成することは不可能な場合があります。例えば、COLCARD、HIGH2KEY、および LOW2KEY などの列統計を作成することは、列が索引と関連付けられていない場合には不可能です。

統計プロファイルで指定されているすべての統計を同期統計収集で収集できない場合、非同期収集要求がサブミットされます。

リアルタイム統計収集は統計収集のオーバーヘッドを最小化するように設計されていますが、まずテスト環境で試行してパフォーマンスに悪影響を与えないことを確かめてください。一部の OLTP シナリオ、特に照会を実行する時間の長さに上限がある場合には、悪影響を与える可能性があります。

統計の自動収集を使用可能にする

正確で完全なデータベースの統計を得ることは、効率的なデータ・アクセスと最適のワークロード・パフォーマンスにとって重要です。関係のあるデータベース統計を更新および保守するには、自動表保守機能の自動統計収集フィーチャーを使用してください。単一データベース・パーティションがシングル・プロセッサーで作動する環境 (シリアル環境) では、この機能の拡張として、照会データを収集し、統計プロファイルを生成することができます。この機能により、DB2 はワークロードが必要とする十分な統計のセットを自動的に収集することができるようになります。このオプションは MPP 環境、特定のフェデレーテッド環境では使用できず、パーティション内並列処理が有効な環境でも使用できません。

自動統計収集を有効にするには、以下のようにします。

1. 「自動保守の構成」ウィザードまたはコマンド行を使用してデータベース・インスタンスを構成します。
   - 「自動保守の構成」ウィザードを使用するには、次のようにします。
     a. コントロール・センターからデータベース・オブジェクトを右クリックするか、ヘルス・センターからデータベース・インスタンスを右クリックすることにより、ウィザードを開きます。
     b. ポップアップ・ウィンドウで「**自動保守の構成**」を選択してください。このウィザードで、自動統計収集を有効にし、自動的に統計を収集する表を指定し、RUNSTATS ユーティリティーを実行するための保守ウィンドウを指定することができます。
   - コマンド行を使用するには、次の構成パラメーターをそれぞれ ON に設定します。
     – **AUTO_MAINT**
     – **AUTO_TBL_MAINT**
     – **AUTO_RUNSTATS**
2. オプション: 自動統計プロファイルの生成を有効にするには、次の 2 つの構成パラメーターを ON に設定します。
   - **AUTO_STATS_PROF**
   - **AUTO_PROF_UPD**
3. オプション: リアルタイム統計収集を有効にするには、**AUTO_STMT_STATS** 構成パラメーターを ON に設定します。 この構成パラメーターが ON に設定された場合、照会の最適化のために必要とされる度に、表統計がステートメントのコンパイル時に自動的にコンパイルされます。

自動統計収集とプロファイルによって使用されるストレージ

自動統計収集および再編成の機能は、作業データをデータベース中の表に格納します。それらの表は、SYSTOOLSPACE 表スペースの中に作成されます。
SYSTOOLSPACE 表スペースは、データベースがアクティブになった時点でデフォルト・オプションにより自動作成されます。それらの表のストレージ要件は、データベース中の表の数に比例し、 1 つの表に対して約 1 KB として計算する必要があります。それがデータベースの重大なサイズである場合は、表スペースを自分でドロップしてから再作成し、ストレージを適切に割り振るとよいでしょう。表スペ

ースの中の自動保守およびヘルス・モニター表は、自動的に再作成されます。表スペースがドロップされると、それらの表にキャプチャーされた履歴は失われます。

自動統計収集アクティビティーのロギング

データベースで生じた統計収集アクティビティーについて知るため、DB2 バージョン 9.5 では統計ログが導入されました。この統計ログは、自動および手動による統計収集を含め、データベースに関するすべての統計アクティビティーを記録します。

統計ログのデフォルト名は、db2optstats.*number*.log です。これは、$DIAGPATH/events ディレクトリーにあります。統計ログは回転ログです。統計ログの動作は、**DB2_OPTSTATS_LOG** レジストリー変数によって制御します。

統計ログの内容を直接表示することもできますし、SYSPROC.PD_GET_DIAG_HIST 表関数を使用して照会することもできます。

SYSPROC.PD_GET_DIAG_HIST 表関数は、タイム・スタンプ、DB2 インスタンス名、データベース名、プロセス ID、プロセス名、およびスレッド ID などのログ・イベントに関する標準情報が含まれる複数の列を戻します。またログには、別のログ機能で使用するための汎用列も含まれます。以下の表で、統計ログにおけるこうした汎用列の使用法について説明します。

表 52. 統計ログ・ファイルでの汎用列

列名	データ・タイプ	説明
OBJTYPE	VARCHAR (64)	イベントが適用されるオブジェクトのタイプ。統計ロギングの場合、これは収集される統計のタイプです。また、開始時あるいは停止時の統計収集バックグラウンド・プロセスを指すこともあります。さらに、自動統計収集によって実行されるアクティビティー (サンプリング・テスト、初期サンプリング、表評価アクティビティーなど) のこともあります。 統計収集アクションに可能な値は、以下のとおりです。 **TABLE STATS** 表統計が収集されます。 **INDEX STATS** 索引統計が収集されます。 **TABLE AND INDEX STATS** 表統計と索引統計の両方が収集されます。 自動統計操作に可能な値は、以下のとおりです。 **EVALUATION** 自動統計バックグラウンド収集プロセスで、評価フェーズが開始されました。評価フェーズでは、統計を必要とするかどうかを判別するために表を検査し、必要な場合には統計が収集されます。 **INITIAL SAMPLING** 表の統計を、サンプリングを使用して収集しています。このサンプル統計は、システム・カタログに格納されます。これにより、自動統計収集は、統計のない表の統計を迅速に収集できます。その後の統計収集では、非サンプル統計が収集されます。初期サンプリングは、自動統計収集の評価フェーズで実行されます。 **SAMPLING TEST** 表の統計を、サンプリングを使用して収集しています。このサンプル統計は、システム・カタログに格納されません。このサンプル統計は現在のカタログ統計と比較されて、この表に対して完全統計を収集すべきかどうか、またいつ収集すべきかが判別されます。このサンプリング・テストは、自動統計収集の評価フェーズで実行されます。 **STATS DAEMON** 統計デーモンは、リアルタイム統計プロセスによってサブミットされた要求を処理するために使用されるバックグラウンド・プロセスです。このオブジェクト・タイプは、バックグラウンド・プロセスの開始および停止時にログに記録されます。

表 52. 統計ログ・ファイルでの汎用列 (続き)

列名	データ・タイプ	説明
OBJNAME	VARCHAR(255)	該当する場合には、イベントが適用されるオブジェクトの名前。統計ログの場合、表または索引の名前が入ります。OBJTYPE が STATS DAEMON または EVALUATION の場合、OBJNAME はデータベース名で、OBJNAME_QUALIFIER は NULL になります。
OBJNAME_QUALIFIER	VARCHAR(255)	統計ログの場合、表または索引のスキーマが入ります。
EVENTTYPE	VARCHAR(24)	イベント・タイプは、このイベントに関連付けられているアクションまたは verb です。統計ロギングに可能な値は、以下のとおりです。 **COLLECT** このアクションは、統計収集操作に関連してログに記録されます。 **START** このアクションは、リアルタイム統計バックグラウンド・プロセス (OBJTYPE = STATS DAEMON) または自動統計収集の評価フェーズ (OBJTYPE = EVALUATION) が開始されるとログに記録されます。 **STOP** このアクションは、リアルタイム統計バックグラウンド・プロセス (OBJTYPE = STATS DAEMON) または自動統計収集の評価フェーズ (OBJTYPE = EVALUATION) が停止されるとログに記録されます。 **ACCESS** 統計収集のために表に対するアクセスが試行されました。このイベント・タイプは、オブジェクトが使用できないために失敗したアクセスをログに記録するために使用されます。
FIRST_EVENTQUALIFIERTYPE	VARCHAR (64)	最初のイベント修飾子のタイプです。イベント修飾子は、イベントによって影響を受けた対象について記述するために使用します。統計ログの場合、最初のイベント修飾子はイベントが発生した際のタイム・スタンプです。 最初のイベント修飾子タイプの値は、AT です。
FIRST_EVENTQUALIFIER	CLOB(16k)	イベントの最初の修飾子です。統計ロギングの場合、最初のイベント修飾子は統計イベントが発生した際のタイム・スタンプです。統計イベントのタイム・スタンプは、TIMESTAMP 列によって表されるログ・レコードのタイム・スタンプとは異なる場合があります。
SECOND_EVENTQUALIFIERTYPE	VARCHAR (64)	2 番目のイベント修飾子のタイプです。統計ロギングの場合、可能な値は BY または NULL です。その他のイベント・タイプではこのフィールドは使用されません。

表 52. 統計ログ・ファイルでの汎用列 (続き)

列名	データ・タイプ	説明
SECOND_EVENTQUALIFIER	CLOB(16k)	イベントの 2 番目の修飾子です。 統計ロギングの場合、COLLECT イベント・タイプで統計が収集された方法を示します。可能な値は以下のとおりです。 **User** DB2 ユーザーが、RUNSTATS、LOAD、CREATE INDEX、または REDISTRIBUTE コマンドを使用して統計収集を実行しました。 **Synchronous** 統計収集は、DB2 によって SQL ステートメント・コンパイル時に実行されました。統計は、システム・カタログ内ではなく統計キャッシュに格納されます。 **Synchronous sampled** 統計収集は、DB2 によって SQL ステートメント・コンパイル時にサンプリングを使用して実行されました。統計は、システム・カタログ内ではなく統計キャッシュに格納されます。 **Fabricate** 統計は、SQL ステートメント・コンパイル時に、データおよび索引マネージャーが維持している情報を使用して作成されました。統計は、システム・カタログ内ではなく統計キャッシュに格納されます。 **Fabricate partial** 一部の統計に限り、SQL ステートメント・コンパイル時に、データ・マネージャーおよび索引マネージャーが保守している情報を使用して作成されました。特に、特定の列の HIGH2KEY と LOW2KEY だけは作成されたものです。統計は、システム・カタログ内ではなく統計キャッシュに格納されます。 **Asynchronous** 統計は DB2 によってバックグラウンド・プロセスで収集され、システム・カタログに格納されました。 その他のイベント・タイプではこのフィールドは使用されません。
THIRD_EVENTQUALIFIERTYPE	VARCHAR (64)	3 番目のイベント修飾子のタイプです。統計ロギングの場合、可能な値は DUE TO または NULL です。

表 52. 統計ログ・ファイルでの汎用列 (続き)

列名	データ・タイプ	説明
THIRD_EVENTQUALIFIER	CLOB(16k)	イベントの 3 番目の修飾子です。 統計ロギングの場合、統計アクティビティーを完了できなかった理由を示します。 可能な値は以下のとおりです。 **Timeout** 同期統計収集の予定時間を超過しました。 **Error** エラーが原因で統計アクティビティーが失敗しました。 **RUNSTATS error** RUNSTATS エラーが原因で同期統計収集が失敗しました。 一部のエラーでは、統計は収集できないものの SQL ステートメント・コンパイルは正常に完了するという場合があります。例えば、統計を収集するにはメモリーが不十分な場合、SQL ステートメント・コンパイルは続行されます。 **Object unavailable** データベース・オブジェクトにアクセスできなかったため、その統計を収集できませんでした。考えられる理由には、以下のものが含まれます。 • オブジェクトが超排他 (Z) モードでロックされている • オブジェクトが存在する表スペースを使用できない • 表索引を再作成する必要がある **Conflict** 別のアプリケーションが既に同期統計を収集していたため、同期統計収集が実行されませんでした。 エラーの詳細については、FULLREC 列または db2diag.log を確認してください。
EVENTSTATE	VARCHAR(255)	イベントの結果のオブジェクトまたはアクションの状態。 統計ロギングの場合、統計操作の状態を示します。 可能な値は以下のとおりです。 • start • success • failure

例

以下の例に挙げる照会では、PD_GET_DIAG_HIST を呼び出し、最大で、現在のタイム・スタンプからさかのぼって過去 1 年分のイベントの統計ログ・レコードを返します。

```
SELECT pid,
tid,
substr(eventtype, 1,10),
substr(objtype,1,30) as objtype,
substr(objname_qualifier,1,20) as objschema,
substr(objname,1,10) as objname,
substr(first_eventqualifier,1,26) as event1,
substr(second_eventqualifiertype,1,2) as event2_type,
substr(second_eventqualifier,1,20) event2,
substr(third_eventqualifiertype,1,6) event3_type,
substr(third_eventqualifier,1,15) event3,
substr(eventstate,1,20) as eventstate
FROMTABLE( SYSPROC.PD_GET_DIAG_HIST
       ( 'optstats', 'EX', 'NONE',
          CURRENT_TIMESTAMP - 1 year, CAST( NULL AS TIMESTAMP ))) as sl
order by timestamp(varchar(substr(first_eventqualifier,1,26),26))
;
```

結果は、FIRST_EVENTQUALIFIER 列に格納されているタイム・スタンプ順に並べられます。この列は、統計イベントの時間を表します。

```
PID   TID                 EVENTTYPE  OBJTYPE                    OBJSCHEMA          OBJNAME    EVENT1                     EVENT2_TYPE EVENT2              EVENT3_TYPE EVENT3          EVENTSTATE
----- ------------------- ---------- -------------------------- ------------------ ---------- -------------------------- ----------- ------------------- ----------- --------------- --------------------
28399          1082145120 START      STATS DAEMON               -                  PROD_DB    2007-07-09-18.37.40.398905 -           -                   -           -               success
28389        183182027104 COLLECT    TABLE AND INDEX STATS      DB2USER            DISTRICT   2007-07-09-18.37.43.261222 BY          Synchronous         -           -               start
28389        183182027104 COLLECT    TABLE AND INDEX STATS      DB2USER            DISTRICT   2007-07-09-18.37.43.407447 BY          Synchronous         -           -               success
28399          1082145120 COLLECT    TABLE AND INDEX STATS      DB2USER            CUSTOMER   2007-07-09-18.37.43.471614 BY          Asynchronous        -           -               start
28399          1082145120 COLLECT    TABLE AND INDEX STATS      DB2USER            CUSTOMER   2007-07-09-18.37.43.524496 BY          Asynchronous        -           -               success
28399          1082145120 STOP       STATS DAEMON               -                  PROD_DB    2007-07-09-18.37.43.526212 -           -                   -           -               success
28389        183278496096 COLLECT    TABLE STATS                DB2USER            ORDER_LINE 2007-07-09-18.37.48.676524 BY          Synchronous sampled -           -               start
28389        183278496096 COLLECT    TABLE STATS                DB2USER            ORDER_LINE 2007-07-09-18.37.53.677546 BY          Synchronous sampled DUE TO      Timeout         failure
28389          1772561034 START      EVALUATION                 -                  PROD_DB    2007-07-10-12.36.11.092739 -           -                   -           -               success
28389          8231991291 COLLECT    TABLE AND INDEX STATS      DB2USER            DISTRICT   2007-07-10-12.36.30.737603 BY          Asynchronous        -           -               start
28389          8231991291 COLLECT    TABLE AND INDEX STATS      DB2USER            DISTRICT   2007-07-10-12.36.34.029756 BY          Asynchronous        -           -               success
28389          1772561034 STOP       EVALUATION                 -                  PROD_DB    2007-07-10-12.36.39.685188 -           -                   -           -               success
```

大きい統計ログの照会パフォーマンスの向上

統計ログ・ファイルが大きい場合、ログ・レコードを表にコピーし、索引を作成し、それから統計を収集することにより、照会のパフォーマンスを向上させることができます。

手順

1. ログ・レコードおよび任意の列を保持する表を作成するには、以下のようにします。

```
create table db2user.stats_log
(pid       bigint,
 tid       bigint,
 timestamp timestamp,
 dbname    varchar(128),
 retcode   integer,
 eventtype varchar(24),
 objtype   varchar(30),
 objschema varchar(20),
 objname   varchar(30),
 event1_type varchar(20),
 event1      timestamp,
 event2_type varchar(20),
 event2      varchar(40),
 event3_type varchar(20),
 event3      varchar(40),
 eventstate  varchar(20)
);
```

2. 照会のカーソルを SYSPROC.PD_GET_DIAG_HIST に対して宣言します。

```
declare c1 cursor for
SELECT pid,
tid,
timestamp,
dbname,
retcode,
eventtype,
substr(objtype,1,30) as objtype,
substr(objname_qualifier,1,20) as objschema,
substr(objname,1,30) as objname,
substr(first_eventqualifiertype,1,20),
substr(first_eventqualifier,1,26),
substr(second_eventqualifiertype,1,20),
substr(second_eventqualifier,1,40),
substr(third_eventqualifiertype,1,20),
substr(third_eventqualifier,1,40),
substr(eventstate,1,20)
FROMTABLE( SYSPROC.PD_GET_DIAG_HIST
       ( 'optstats', 'EX', 'NONE',
          CURRENT_TIMESTAMP - 1 year, CAST( NULL AS TIMESTAMP ))) as sl
;
```

3. LOAD コマンド、およびカーソルからロード機能を使用して、統計ログ・レコードを表にロードします。

```
load from c1 of cursor replace into db2user.stats_log;
```

4. 表に、索引を作成し、統計を収集します。

```
create index sl_ix1 on db2user.stats_log(eventtype, event1);
create index sl_ix2 on db2user.stats_log(objtype, event1);
create index sl_ix3 on db2user.stats_log(objname);

runstats on table db2user.stats_log with distribution and sampled detailed indexes all;
```

統計の収集および更新のガイドライン

RUNSTATS コマンドは、表、索引、および統計ビューに関する統計を収集し、アクセス・プランの選択のための正確な情報を、オプティマイザーに提供します。

以下のような状態のときに、 RUNSTATS ユーティリティーを使用して統計を収集してください。

- 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。
- 表に新しい索引を作成する場合。最後に RUNSTATS を実行してからその表を変更していない場合、新しい索引に関してのみ RUNSTATS を実行する必要があります。
- 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。
- データの変更、削除、および挿入によって、表とその索引が大幅に変更される場合。 (この場合「大量」とは、表データと索引データの 10 % から 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。
- 現在の表と前の表を比較する場合。決まったインターバルで統計を更新すると、パフォーマンス上の問題を容易に発見できます。
- プリフェッチ数量が変更される場合。
- REDISTRIBUTE DATABASE PARTITION GROUP ユーティリティーを使用している場合。

注: DB2 の以前のバージョンでは、このコマンドは DATABASE PARTITION GROUP キーワードではなく NODEGROUP キーワードを使用していました。

- XML 列での統計を RUNSTATS ユーティリティーを使用して収集してください。 RUNSTATS が XML 列のみの統計を収集するのに使用されるとき、LOAD または RUNSTATS ユーティリティーの以前の実行によって収集された非 XML 列の既存の統計は保存されます。いくつかの XML 列の統計が以前に収集されている場合、XML 列の以前に収集された統計は、現行のコマンドでその XML 列の統計が収集されない場合ドロップされ、現行のコマンドでその XML 列の統計が収集される場合には置換されます。

RUNSTATS のパフォーマンスを向上させ、統計の保管に使用されるディスク・スペースを節約するために、データ分散統計を収集しなければならない列だけを指定するように意識してください。

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。新しい統計の場合、照会オプティマイザーによって別のアクセス・プランを選択される場合があります。

一度にすべての統計を収集する時間がない場合には、一度に少数の表、索引、または統計ビューに対して RUNSTATS を実行して統計を更新するようにし、これをそれぞれのオブジェクトのセットに対して順番に行ってください。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、照会の最適化中に警告メッセージ (SQL0437W、理由コード 6) が出されます。たとえば、これは RUNSTATS を実行して表分散統計を収集して、ある表の活動の後に RUNSTATS を再度実行してその表に関する索引統計を収集すると発生することがあります。照会の最適化中に、表で行われた活動の結果として不整合が検出され、削除された場合、警告メッセージが出されます。このような場合、 RUNSTATS を再度実行し、分散統計を更新しなければなりません。

確実に索引統計を表と同期化させるために、 RUNSTATS を実行して、表と索引の両方の統計を同時に収集してください。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、すべてのノードでそれらの 2 つの統計のセットが同期していないことになります。

実稼働中のシステムで RUNSTATS を呼び出すと、稼働中処理のパフォーマンスに悪影響を与える可能性があります。RUNSTATS ユーティリティーはスロットル・オプションをサポートするようになりました。これは、ハイレベルのデータベース・アクティビティー中の RUNSTATS 実行のパフォーマンス影響を制限するのに使用できます。

パーティション・データベース環境で表の統計を収集する際に、RUNSTATS はこれを実行したデータベース・パーティションにある表の統計しか収集しません。このデータベース・パーティションの RUNSTATS 結果は、他のデータベース・パーティションに外挿されます。 RUNSTATS を実行したデータベース・パーティションに特定の表の一部分が含まれていない場合、その要求は、その表の一部を含むデータベース・パーティション・グループの中の先頭のデータベース・パーティションに送られます。

統計ビューの統計を収集する際には、ビューに参照される基本表を含む全データベース・パーティションについて統計が収集されます。

RUNSTATS の効率と収集された統計の有用性を改善するには、以下のヒントを考慮してください。

- 表を結合するために使用される列、または WHERE、GROUP BY、および照会の類似の節で使用される列に関する統計のみを収集します。索引に含まれていれば、これらの列は RUNSTATS コマンドで ONLY ON KEY COLUMNS 節を使用して指定できます。
- 特定の表および表内の特定の列について *num_freqvalues* および *num_quantiles* の値をカスタマイズします。
- 詳細な索引統計のために実行されるバックグラウンド計算量を減らすために、SAMPLE DETAILED 節を指定して DETAILED 索引統計を収集します。SAMPLE DETAILED 節を指定すると、統計を収集するのに必要な時間が短縮され、ほとんどの場合に十分な精度が得られます。
- データが入っている表の索引を作成する場合は、索引の作成時に統計が作成されるように、COLLECT STATISTICS 節を追加します。
- 表の行が大量に追加または削除された場合、または統計を収集する列のデータが更新された場合は、 RUNSTATS を再実行して統計を更新してください。
- RUNSTATS は単一データベース・パーティションの統計しか収集しないので、常にデータがすべてのデータベース・パーティションに分散していない場合には、統計はあまり正確になりません。データ分散がひずんでいると疑われる場合は、RUNSTATS を実行する前に REDISTRIBUTE DATABASE PARTITION GROUP コマンドを使用して、データベース・パーティションにデータを再分散させることもできます。

カタログ統計の収集

表、索引、および統計ビューのカタログ統計を収集することにより、オプティマイザーが照会に最も適したアクセス・プランを選択するために使用する情報を提供できます。

表と索引には、次のいずれかが必要です。

- sysadm
- sysctrl
- sysmaint
- dbadm
- 表に対する CONTROL 特権
- LOAD 権限

統計ビューには、次のいずれかが必要です。

- sysadm
- sysctrl
- sysmaint
- dbadm
- 表に対する CONTROL 特権

さらに、統計ビューの行へのアクセス権を持っていなければなりません。特に、統計ビューの定義で参照されているそれぞれの表、ビュー、またはニックネームには、次のいずれかの特権が必要です。

- SYSADM または DBADM
- CONTROL
- SELECT

カタログ統計を収集するには、以下を行います。

1. 統計情報を収集する表、索引、または統計ビューを含んでいるデータベースに接続します。
2. DB2 コマンド行から、適切なオプションを指定して RUNSTATS コマンドを実行します。これらのオプションを使用することにより、表、索引、または統計ビューに対して実行される照会に関して収集される統計を調整できます。
3. RUNSTATS が完了したら、 COMMIT ステートメントを発行してロックを解放します。
4. 統計情報を再生成した表、索引、または統計ビューにアクセスするパッケージを再バインドします。

グラフィカル・ユーザー・インターフェースを使用して、オプションの指定および統計の収集を行うには、コントロール・センターを使用します。

注:

1. RUNSTATS ユーティリティーではニックネームの使用がサポートされていないので、フェデレーテッド・データベース照会の場合は別の方法で統計を更新します。照会でフェデレーテッド・データベースにアクセスする場合は、すべてのデータベース内の表に対して RUNSTATS を実行し、次いでリモート・データベースにアクセスするニックネームをドロップして再作成することにより、新しい統計をオプティマイザーで使用できるようにします。
2. パーティション・データベース環境で表の統計を収集する際に、RUNSTATS はこれを実行したデータベース・パーティションにある表の統計しか収集しません。このデータベース・パーティションの RUNSTATS 結果は、他のデータベース・パーティションに外挿されます。 RUNSTATS を実行したデータベース・パーティションに特定の表の一部分が含まれていない場合、その要求は、その表の一部を含むデータベース・パーティション・グループの中の先頭のデータベース・パーティションに送られます。

   統計ビューの統計を収集する際には、ビューに参照される基本表を含む全データベース・パーティションについて統計が収集されます。

特定の列に関する分散統計の収集

RUNSTATS とそれ以降の照会計画分析の両方を効率的に行うために、 WHERE、GROUP BY、および同様の節で照会が使用する列だけで、分散統計を収集することもできます。列の結合グループで、カーディナリティー統計を収集することもできます。グループ内の列を参照する照会の選択を確立するときに、オプティマイザーはそのような情報を使用して、列の相関を検出します。

以下のステップでは、データベースが **sales** であり、それに索引 **custidx1** および **custidx2** を含む、表 **customers** が入っているということを想定しています。

表および索引を含むデータベースに接続し、以下の許可レベルのいずれかを持っていなければなりません。

- sysadm
- sysctrl
- sysmaint
- dbadm
- 表に対する CONTROL 特権

注: RUNSTATS は、これを実行したデータベース・パーティションにある表の統計しか収集しません。このデータベース・パーティションの RUNSTATS 結果は、他のデータベース・パーティションに外挿されます。 RUNSTATS を実行したデータベース・パーティションに表の一部が含まれていない場合、その要求は、表のその部分を保持するデータベース・パーティション・グループの中の先頭のデータベース・パーティションに送られます。

特定の列に関する統計を収集するには、以下を行います。

1. **sales** データベースに接続します。
2. DB2 コマンド行で以下のコマンドのいずれかを実行します。どれを実行するかは、要件によって異なります。
   - 列 **zip** および **ytdtotal** に関する分散統計を収集するには、以下を行います。

     ```
     RUNSTATS ON TABLE sales.customers
          WITH DISTRIBUTION ON COLUMNS (zip, ytdtotal)
     ```

   - 同じ列に関する分散統計を収集するものの、分散のデフォルトを調整するには、以下を行います。

     ```
     RUNSTATS ON TABLE sales.customers
          WITH DISTRIBUTION ON
          COLUMNS (zip, ytdtotal NUM_FREQVALUES 50 NUM_QUANTILES 75)
     ```

   - **custidx1** および **custidx2** で索引付けされている列の分散統計を収集するには、以下を行います。

     ```
     RUNSTATS ON TABLE sales.customer
              ON KEY COLUMNS
     ```

   - 特定の列 **zip** と **ytdtotal**、および **region** と **territory** を含む列グループだけに関する列統計を収集するには、以下を行います。

     ```
     RUNSTATS ON TABLE sales.customers
               ON COLUMNS (zip, (region, territory), ytdtotal)
     ```

   - 非 XML 列の統計が、STATISTICS オプションを指定した LOAD コマンドを使用して収集されているとします。非 XML 統計を XML 列 **miscinfo** の統計で補うには、以下のようにします。

     ```
     RUNSTATS ON TABLE sales.customers
        ON COLUMNS (miscinfo)
     ```

   - 表の非 XML 列のみについての列統計を収集するには、以下のようにします (EXCLUDING XML COLUMNS オプションは、XML 列を指定できる他のすべての節に優先します)。

```
RUNSTATS ON TABLE sales.customers
   EXCLUDING XML COLUMNS
```

コントロール・センターを使用して、分散統計を収集することもできます。

索引統計の収集

索引統計を収集することにより、オプティマイザーで、照会を解決するために索引を使用すべきかどうかを評価できます。

以下のステップでは、データベースが **sales** であり、それに索引 **custidx1** および **custidx2** を含む、表 **customers** が入っているということを想定しています。

表および索引を含むデータベースに接続し、以下の許可レベルのいずれかを持っていなければなりません。

- sysadm
- sysctrl
- sysmaint
- dbadm
- 表に対する CONTROL 特権

SAMPLED DETAILED オプションを指定して RUNSTATS を実行するには、統計ヒープが 2MB 必要です。この追加メモリー要件用に、追加の 488 4K ページを *stat_heap_sz* データベース構成パラメーターの設定に割り振ってください。ヒープが小さすぎた場合、 RUNSTATS は統計の収集を試行する前にエラーを戻します。

索引の詳細な統計を収集するには、以下を行います。

1. **sales** データベースに接続します。
2. DB2 コマンド行で以下のコマンドのいずれかを実行します。どれを実行するかは、要件によって異なります。
   - **custidx1** と **custidx2** の両方に関する詳細な統計を作成するには、以下を行います。

     ```
     RUNSTATS ON TABLE sales.customers AND DETAILED INDEXES ALL
     ```

   - 両方の索引に関する詳細な統計を作成するものの、索引入力ごとに詳細な統計を実行するのではなく、サンプリングを使用するには、以下を行います。

     ```
     RUNSTATS ON TABLE sales.customers AND SAMPLED DETAILED INDEXES ALL
     ```

   - 索引および表の統計に整合性を持たせるために、表の分散統計のほかに詳細なサンプリング済み統計を作成するには、以下を行います。

     ```
     RUNSTATS ON TABLE sales.customers
              WITH DISTRIBUTION ON KEY COLUMNS
              AND SAMPLED DETAILED INDEXES ALL
     ```

コントロール・センターを使用して、索引および表統計を収集することもできます。

表データのサンプルでの統計の収集

表統計は、特定の照会に最適なアクセス・プランを選択するときにオプティマイザーによって使用されるので、統計が特定時点の表の状態を正確に反映するものであ

ることは重要です。表に対する活動が増えれば増えるほど、統計収集の頻度も増やす必要があります。データベースのサイズが増えるにつれて、効率的に統計を収集することが一層重要になります。統計を収集する表データからランダムにサンプリングすると、 RUNSTATS のパフォーマンスは向上します。 I/O 性能や CPU 制約が限られたシステムの場合、こうしたパフォーマンス上の利点は計り知れません。サンプルが小さいほど、RUNSTATS は速く終了します。

バージョン 8.2 以降、RUNSTATS コマンドには TABLESAMPLE オプションが用意されており、表のデータのサンプルから統計を収集できるようになっています。サンプリングはデータの一部だけを使用するので、この機能を使用すると統計収集の効率がよくなります。同時に、複数のサンプリング方式によって、正確性が保証されます。

サンプルを収集する方法は 2 とおりから指定できます。ベルヌーイ方式は、行レベルでデータをサンプリングします。データ・ページの完全表スキャンを実行中に、各行を順番に検討し、数値パラメーターで指定した確率 P に基づいて行を選択します。こうして選択した行からのみ、統計が収集されます。同様にして、SYSTEM 方式はページ・レベルでデータをサンプリングします。このとき、各ページを確率 P に基づいて選択し、確率 1-P/100 で除外します。

サンプルに組み込まれるページごとに 1 つの I/O だけが必要であるため、ページ・レベルのサンプリングは非常に高いパフォーマンスを実現します。行レベルのサンプリングを使用した場合、各表ページを完全表スキャンに取り込むため、I/O コストは減りません。しかしながら、I/O の量が減ることはなくても、統計情報の収集処理は CPU での処理を主としたものなので、行レベルのサンプリングでもパフォーマンスはかなり向上します。

データ値が高クラスター化されている状況では、ページ・レベル・サンプリングよりも行レベル・サンプリングのほうが良いサンプルを提供できます。ページ・サンプリングと比べると、行レベルのサンプルでは各データ・ページから P パーセントの行をサンプリングするので、全データの統計的傾向をより反映することができます。ページ・レベルのサンプリングでは、P パーセントのページに含まれる行すべてがサンプリング対象になります。行が表の中でランダムに分散している場合は、行サンプル統計とページ・サンプル統計の正確さはほぼ同じです。

RUNSTATS コマンドに REPEATABLE オプションを指定せずに何回も実行すると、各サンプルがランダムに生成されます。 REPEATABLE 節を使用すると、TABLESAMPLE オプションを指定した RUNSTATS コマンドを最後に実行したときと同じサンプルが生成されます。変化のすくないデータをもつ表に対して一貫性のある統計の生成が必要な場合に、この機能は便利です。

統計プロファイルを使用した統計の収集

RUNSTATS ユーティリティーには、統計プロファイルを登録して使用するオプションがあります。このプロファイルは、特定の表にどの統計を収集するかを指定したオプションの集まりです (たとえば、表統計、索引統計、または分散統計)。

この機能によって、RUNSTATS コマンド発行時に指定するオプションの保管が可能になり、コマンド・オプションをタイプし直さなくても繰り返し同じ統計を表に収集できるので、統計の収集が簡単になります。

統計の収集中であってもなくても、統計プロファイルを登録または更新できます。たとえば、プロファイルの登録と統計の収集を同時に行うには、 RUNSTATS コマンドに SET PROFILE オプションを指定して実行します。統計の収集は行わずに、プロファイルの登録だけを行う場合は、 RUNSTATS コマンドに SET PROFILE ONLY オプションを指定して実行します。

すでに登録してある統計プロファイルを使用して統計を収集するには、 RUNSTATS コマンドに表の名前と USE PROFILE オプションだけを指定して実行します。

統計プロファイル内で特定の表について現在指定されているオプションを表示するには、次の SELECT ステートメントを使用してカタログ表を照会します。 tablename はプロファイルの指定を確認する表の名前です。

```
SELECT STATISTICS_PROFILE FROM SYSIBM.SYSTABLES WHERE NAME = tablename
```

自動統計プロファイル作成

DB2 自動統計プロファイル作成機能を使用することで、統計プロファイルを自動的に生成することもできます。この機能を使用可能にすると、データベース活動に関する情報が収集され、照会フィードバック・ウェアハウスに保管されます。このデータに基づいて、統計プロファイルが生成されます。この機能を使用可能にすると、特定のワークロードとどの統計が関係し合っているのかが分からないために生じる問題が軽減され、最小限の統計情報を収集することで最適なデータベース・ワークロードのパフォーマンスを実現できます。

この機能は自動統計収集機能とともに使用できます。自動統計収集機能は、自動的に生成された統計プロファイル内に含まれる情報に基づいて統計保守を自動的にスケジュールに入れます。

この機能を使用可能にするには、該当する構成パラメーターを設定して、自動表保守がすでに使用可能になっていることが必要です。 AUTO_STATS_PROF 構成パラメーターは照会フィードバック・データの収集を活動化し、 AUTO_PROF_UPD 構成パラメーターは統計プロファイルの生成を活動化し、自動統計収集で使用できるようにします。

注: 自動統計プロファイル生成は DB2 シリアル・モードでのみ活動化でき、フェデレーテッド環境、複数パーティション MPP 環境、およびパーティション内並列処理が有効な環境での照会には使用できません。

統計プロファイル生成が最適なのは、多数の述部を適用し大きく複雑な照会を実行する環境、述部列のデータどうしの相関関係が密な環境、複数の表を結合およびグループ化する環境です。一方、トランザクション処理を主に行う環境には、適していません。

この機能は、いくつかの異なる方法で使用できます。

- テスト環境。テスト・システムでは、AUTO_STATS_PROF と AUTO_PROF_UPD を ON に設定します。テスト・システムでは、ランタイム・モニターのパフォーマンス・オーバーヘッドを大目に見ることができます。テスト・システムで実際のデータや照会を使用する場合には、この環境によって RUNSTATS の統計パラメーターに適正な相関や設定を調べることができ、その結果を統計プロファイル

に保管しておきます。このプロファイルを実動システムに転送することも可能で、その場合照会にモニター・オーバーヘッドを取らずに済みます。

- 実稼働環境で、特定の照会に関するパフォーマンスの問題を扱う。特定の照会セットにパフォーマンス上の問題が検出され、その原因が統計または相関の不良である可能性がある場合に、 AUTO_STATS_PROF をオンにして一定時間ターゲット・ワークロードを実行します。自動統計プロファイルは照会のフィードバックを分析して、 SYSTOOLS.OPT_FEEDBACK_RANKING* 表に勧告を作成します。これらの勧告をよく読み、それに基づいて手動で統計プロファイルを改良できます。この勧告に基づいて DB2 が統計プロファイルを自動で更新するようにするには、 AUTO_STATS_PROF をオンにするときに AUTO_PROF_UPD もオンにします。

注: 照会のモニター、および照会フィードバック・データのフィードバック・ウェアハウスへの保管に関連して、いくらかのパフォーマンス・オーバーヘッドがあります。

照会フィードバック・ウェアハウスの作成

フィードバック・ウェアハウスは SYSTOOLS スキーマの 5 つの表からなっており、これらの表には照会実行中に検出された述部に関する情報、および統計収集に関する推奨事項が保管されます。 5 つの表とは、OPT_FEEDBACK_QUERY、OPT_FEEDBACK_PREDICATE、 OPT_FEEDBACK_PREDICATE_COLUMN、OPT_FEEDBACK_RANKING、および OPT_FEEDBACK_RANKING_COLUMN です。

自動統計プロファイル作成を使用するには、SYSINSTALLOBJECTS ストアード・プロシージャーを使用してまず照会フィードバック・ウェアハウスを作成する必要があります。このストアード・プロシージャーは、オブジェクトを作成およびドロップする SYSTOOLS スキーマの共通ストアード・プロシージャーです。

次のようにして、SYSINSTALLOBJECTS ストアード・プロシージャーを呼び出します。

```
call SYSINSTALLOBJECTS ( toolname, action, tablespacename, schemaname)
```

詳細は次のとおりです。

toolname
: オブジェクトを作成またはドロップするツールの名前を指定します。この場合は 'ASP' または 'AUTO STATS PROFILING' です。

action
: 行う処置を指定します。'C' は作成、'D' はドロップ。

tablespacename
: フィードバック・ウェアハウス表を作成する表スペースの名前。この入力パラメーターはオプションです。このパラメーターを指定しない場合、デフォルトのユーザー・スペースが使用されます。

schemaname
: オブジェクトを作成またはドロップするときに使用するスキーマの名前。このパラメーターはここでは使用しません。

たとえば、フィードバック・ウェアハウスを表スペース ″A″ に作成するには、次のように入力します。 `call SYSINSTALLOBJECTS ('ASP', 'C', 'A', '')`

カタログ統計の表

以下の表では、カタログ統計を含むシステム・カタログ表と、特定の統計を収集する RUNSTATS オプションに関する情報が提供されています。

表 53. 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	はい	はい
NPAGES	行が含まれているページの数	はい	はい
OVERFLOW	オーバーフローしている行の数	はい	いいえ
CARD	表内の行の数 (カーディナリティー)	はい	Yes (注 1)
ACTIVE_BLOCKS	MDC 表の場合、占有されているブロックの合計数	はい	いいえ

注:

1. 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 54. 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	はい	Yes (注 1)
AVGCOLLEN	列の平均長	はい	Yes (注 1)
HIGH2KEY	列内で 2 番目に高い値	はい	Yes (注 1)
LOW2KEY	列内で 2 番目に低い値	はい	Yes (注 1)
NUMNULLS	列内の NULL の数	はい	Yes (注 1)
SUB_COUNT	サブエレメントの平均数	はい	No (注 2)
SUB_DELIM_LENGTH	各サブエレメントを分ける各区切り文字の平均長	はい	No (注 2)

注:

1. 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。
2. これらの統計では、ブランクで区切られている一連のサブフィールドまたはサブエレメントを含む列の、データに関する情報が提供されています。 SUB_COUNT および SUB_DELIM_LENGTH 統計が収集されるのは、タイプ CHAR、VARCHAR、GRAPHIC、および VARGRAPHIC の、 1 バイト文字セットのストリング列の場合だけです。

表 55. 複数列統計 (SYSCAT.COLGROUPS および SYSSTAT.COLGROUPS)

統計	説明	RUNSTATS オプション	
		表	索引
COLGROUPCARD	列グループのカーディナリティー	はい	いいえ

注: 以下の 2 つの表でリストされている複数列分布統計は、 RUNSTATS では収集されません。手動で更新することはできません。

表 56. 複数列分布統計 (SYSCAT.COLGROUPDIST および SYSSTAT.COLGROUPDIST)

統計	説明	RUNSTATS オプション	
		表	索引
TYPE	F = 頻度 Q = 変位値	はい	いいえ
ORDINAL	グループ内の列の序数	はい	いいえ
SEQNO	*n* 番目の TYPE 値を表す、シーケンス番号 *n*。	はい	いいえ
COLVALUE	文字リテラルまたは NULL 値としてのデータ値	はい	いいえ

表 57. 複数列分布統計 2 (SYSCAT.COLGROUPDISTCOUNTS および SYSSTAT.COLGROUPDISTCOUNTS)

統計	説明	RUNSTATS オプション	
		表	索引
TYPE	F = 頻度 Q = 変位値	はい	いいえ
SEQNO	*n* 番目の TYPE 値を表す、シーケンス番号 *n*。	はい	いいえ
VALCOUNT	TYPE = F の場合、VALCOUNT は、この SEQNO によって識別される列グループでの、COLVALUE のオカレンスの数です。 TYPE = Q の場合、VALCOUNT は、この SEQNO の列グループで、値が COLVALUE 以下である行の数です。	はい	いいえ
DISTCOUNT	TYPE = Q の場合、この列には、この SEQNO の列グループでの、 COLVALUE 以下の固有値の数が含まれています。利用不能の場合は NULL です。	はい	いいえ

表 58. 索引統計 (SYSCAT.INDEXES と SYSSTAT.INDEXES)

統計	説明	RUNSTATS オプション	
		表	索引
NLEAF	索引リーフ・ページの数	いいえ	はい
NLEVELS	索引レベルの数	いいえ	はい
CLUSTERRATIO	表データのクラスタリングの程度	いいえ	Yes (注 2)
CLUSTERFACTOR	クラスタリングの詳細の程度	いいえ	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 3)	いいえ	はい
FIRSTKEYCARD	索引の最初の列の固有値の数	いいえ	はい
FIRST2KEYCARD	索引の最初の 2 つの列の固有値の数	いいえ	はい
FIRST3KEYCARD	索引の最初の 3 つの列の固有値の数	いいえ	はい
FIRST4KEYCARD	索引の最初の 4 つの列の固有値の数	いいえ	はい
FULLKEYCARD	索引のすべての列の固有値の数 (ただし、すべての RID が削除対象としてマークされているタイプ 2 索引のキー値はすべて除く)	いいえ	はい
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	いいえ	詳細説明 (注 1、2)
AVGPARTITION_CLUSTERRATIO	単一のデータ・パーティション内でのデータ・クラスタリングの程度	いいえ	Yes (注 2)
AVGPARTITION_CLUSTERFACTOR	単一のデータ・パーティション内での、クラスタリングの度合いのより精密な測定	いいえ	詳細説明 (注 1、2)
AVGPARTITION_PAGE_FETCH_PAIRS	単一のデータ・パーティションを基にして生成された異なるバッファー・サイズのページ取り出し見積もり	いいえ	詳細説明 (注 1、2)
DATAPARTITION_CLUSTERFACTOR	索引スキャン中のデータ・パーティション参照の数	No (注 6)	Yes (注 6)
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リーフ・ページの数	いいえ	はい
AVERAGE_SEQUENCE_PAGES	順次にアクセスできる索引ページの平均数これは、プリフェッチャーが順次として検出できる索引ページの数です。	いいえ	はい
AVERAGE_RANDOM_PAGES	順次ページ・アクセスの間のランダム索引ページの平均数	いいえ	はい

表 58. 索引統計 (SYSCAT.INDEXES と SYSSTAT.INDEXES) (続き)

統計	説明	RUNSTATS オプション	
		表	索引
AVERAGE_SEQUENCE_GAP	シーケンス間のギャップ	いいえ	はい
AVERAGE_SEQUENCE_FETCH_PAGES	順次にアクセスできる表ページの平均数これは、索引を使用して表の行をフェッチするときに、プリフェッチャーが順次として検出できる表ページの数です。	いいえ	Yes (注 4)
AVERAGE_RANDOM_FETCH_PAGES	索引を使用して表の行をフェッチするときの、順次ページ・アクセスの間のランダム表ページの平均数	いいえ	Yes (注 4)
AVERAGE_SEQUENCE_FETCH_GAP	索引を使用して表の行をフェッチするときの、シーケンス間のギャップ	いいえ	Yes (注 4)
NUMRIDS	索引内のレコード ID (RID) の数 (タイプ 2 索引の削除対象の RID を含む)。	いいえ	はい
NUMRIDS_DELETED	索引内の削除対象としてマークされている RID の合計数 (すべてのレコード ID が削除対象としてマークされている、リーフ・ページの RID は除く)	いいえ	はい
NUM_EMPTY_LEAFS	すべてのレコード ID が削除対象としてマークされている、リーフ・ページの合計数	いいえ	はい
INDCARD	索引項目の数 (索引のカーディナリティー)	いいえ	はい

注:

1. 詳細索引統計は、RUNSTATS コマンドに DETAILED 節を指定します。
2. 表のサイズが相当大きいものでない限り、 DETAILED 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 節を指定しないと、CLUSTERRATIO 統計だけが収集されます。
3. 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、索引ページがプリフェッチされたときに、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、オプティマイザーが見積もるのに使用されます。
4. これらの統計は、この表が DMS 表スペースにある場合は計算できません。
5. プリフェッチ統計は、統計収集が LOAD または CREATE INDEX コマンドの呼び出し時に指定されている場合でも、それらのコマンドの実行中には収集されません。プリフェッチ統計は、順次検出フラグ構成パラメーター (seqdetect) がオフになっている場合も収集されません。
6. 表の RUNSTATS オプションが ″No″ なら、表統計の収集時に統計が収集されないことを意味し、索引の RUNSTATS オプションが ″Yes″ なら、 INDEXES オプションを指定した RUNSTATS コマンドが使用されるときに、統計が収集されることを意味します。

表 59. 列分散統計 (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 統計以下の固有値の数	DISTRIBUTION (注 2)	いいえ
TYPE	行の統計が頻出値統計または変位値統計かの標識	分散	いいえ
SEQNO	表の行を固有に識別するのに役立つシーケンス番号の頻度のランク	分散	いいえ
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	分散	いいえ
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	分散	いいえ

注:

1. 列分散統計は、RUNSTATS コマンドに WITH DISTRIBUTION 節を指定します。列の値が十分に不均一でないかぎり、分散統計は**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。

分散統計

以下の 2 つの種類のデータ分散統計を収集できます。

- 頻度統計

  これらの統計は、重複の数が最も多い列およびデータ値、その次に重複値が多い列およびデータ値というように、 *num_freqvalues* 構成パラメーターで指定したレベルまで、それらの情報を提供します。頻度統計の収集を使用不可にするには、*num_freqvalues* を 0 に設定します。

  num_freqvalues を各表または統計ビュー、および特定の列の RUNSTATS オプションとして設定することもできます。

- 変位値統計

  これらの統計は、データ値がほかの値との関連でどのように分布しているかに関する情報を提供します。これらの K 変位値と呼ばれる統計の値 V とは、 K 個以上の値がその値 V 以下であることを示すものです。 K 変位値は、値の昇順をソートすることで計算できます。 K 変位値は、その範囲の最後から K 番目の位置の値です。

  列データ値がグループ化されるセクションの数を指定するには、 *num_quantiles* データベース構成パラメーターを、 2 から 32,767 の間の値に設定します。デフォルト値である 20 では、オプティマイザー見積エラーは、等価、より小、またはより大の述部で最大プラス・マイナス 2.5%、 BETWEEN 述部で最大エラーでプラス・マイナス 5% です。変位値統計の収集を使用不可にするには、num_quantiles を 0 または 1 に設定します。

num_quantiles を各表または統計ビュー、または特定の列に設定することもできます。

注: より大きな *num_freqvalues* および *num_quantiles* 値を指定すると、 RUNSTATS の実行時に、 *stat_heap_sz* データベース構成パラメーターで指定されているメモリーでは不足することがあります。また多くの CPU リソースも必要になることがあります。

分散統計の収集時に所定の表または統計ビューに関して分散統計を作成および更新すべきかどうかを判断するには、次の 2 つの要因を考慮してください。

- アプリケーションが静的または動的 SQL および XQuery ステートメントを使用するか。

  分散統計は、ホスト変数を使用しない動的照会および静的照会で最も便利です。ホスト変数を含む照会を使用する場合、オプティマイザーは分散統計を限定された方法で使用することになります。
- 列内のデータの分布が均一か。

  表のうちの少なくとも 1 つの列に、かなり 「不均一」なデータ分散があり、その列が下記の節のような等号述部または範囲述部に頻繁に現れる場合、分散統計を作成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;
```

データ分散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。

- 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、 1 つ以上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。データが範囲 (5、10) の中にクラスター化されている、以下の例を考慮してください。

C1
0.0
5.1
6.3
7.1
8.2
8.4
8.5
9.1
93.6
100.0

  変位値統計は、オプティマイザーがこのようなデータ分散を扱う際に役立ちます。

  列データが均一に分布していないかどうかを判別するには、以下の例のような照会を実行してください。

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

- 重複データ値が頻繁に発生する場合。データが以下の頻度で分布している列について考慮してください。

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

  オプティマイザーが重複値を処理するのに役立つように、変位値と頻出値の両方の統計を作成してください。

索引統計のみを収集する場合

次の状況では、索引データにのみ基づく統計を収集するかもしれません。

- 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。

指定する統計制度のレベル

分散統計を収集する精度を決定するには、データベース構成パラメーター *num_quantiles* および *num_freqvalues* を指定します。表または列の統計を収集するときに、これらのパラメーターを RUNSTATS オプションとして指定することもできます。大きな値を設定するほど、分散統計の作成および更新時に RUNSTATS が使用する精度が高くなります。ただし精度を高くすると、 RUNSTATS の実行とカタログ表で必要なストレージの両方で、より多くのリソースを使用しなければなりません。

ほとんどのデータベースでは、 *num_freqvalues* データベース構成パラメーターに 10 から 100 を指定します。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、頻出値統計を作成するのが理想です。データベース・マネージャーはこ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。変位値統計のみを収集する必要がある場合、 *num_freqvalues* をゼロに設定します。

変位値の数を設定する場合、 *num_quantiles* データベース構成パラメーターの設定として 20 から 50 を指定します。変位値の数を決めるときの経験則は、次のとおりです。

- 範囲照会の行数の見積もりで許容できる最大エラー P をパーセント単位で求めます。
- 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。

例えば、変位値の数が 25 であれば、最大見積エラーは、BETWEEN 述部の場合は 4%、「>」述部の場合は、2% という結果になる必要があります。一般に、変位値は 10 以上を指定します。 50 を超える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。頻出値統計のみを必要とする場合、 *num_quantiles* をゼロに設定してください。このパラメーターを「1」に設定すると、値の範囲全体が 1 つの変位値内に収まるため、変位値統計は収集されません。

分散統計のオプティマイザーの使用

オプティマイザーは分散統計を使用することにより、照会を満たす、可能なさまざまなアクセス・プランのコストをより正確に見積もることができます。

WITH DISTRIBUTION 節を指定して RUNSTATS を実行しなかった場合、カタログ統計表には、表または統計ビューのサイズ、表または統計ビュー内の最大値と最小値、その索引のいずれかに対する表のクラスタリングの多重度、索引付けされた列内の固有値の数に関する情報のみが含まれます。

最大値と最小値の間の値の分散に関して追加情報がなければ、オプティマイザーはデータ値が均等に分布していると想定します。データ値がそれぞれ大きく異なる場合、範囲内のいくつかの部分でクラスター化されている場合、または多くの重複値を含んでいる場合、オプティマイザーは最適なアクセス・プランよりも小さく選択します。

次の例を考慮してください。

コストが最低のアクセス・プランを選択するために、等号述部または範囲述部を満たす列を含む行の数を、オプティマイザーが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、オプティマイザーが最適のアクセス・プランを選択する可能性が大きくなります。たとえば、次のような照会を考えてみます。

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

C1 と C2 の両方に索引が 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。通常、照会を実行するために主なコストは行を検索するコストであるため、最適なプランとは、必要な検索が最も少ないプランです。このような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必要があります。

分散統計が使用可能でなく、 RUNSTATS が表または統計ビューに対して実行されている場合、オプティマイザー使用できる情報は、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)、固有値の数 (COLCARD)、および行数 (CARD) だけです。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散が、十分に均一である場合だけです。使用できる分散統計がなく、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が行われている場合は、見積もりはけた違いのものになり、オプティマイザー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。

分散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。

分散統計の使用の拡張例

オプティマイザーが分散統計を使用する仕方を理解するには、フォーム C1 = KEY の等号述部を含む照会についてまず考慮してください。

頻出値の統計の例

頻出値が使用可能であれば、以下のように、オプティマイザーはそれらの統計を使用して、適切なアクセス・プランのコストを選択できます。

- KEY が、N 最大頻出値のいずれかである場合、オプティマイザーは、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、オプティマイザーは、 (COLCARD - N) 個の非頻出値が均一に分散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
```

CARD は表内の行の数、COLCARD は列のカーディナリティー、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 (C1) 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

この列に関して、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な場合、表内の行の数は 50、列のカーディナリティーは 5 です。述部 C1 = 3 の場合、ちょうど 40 個の行がそれを満たしています。データが均等に分布している

とオプティマイザーが想定している場合、述部を満たす行の数は 50/5 = 10 として見積もられ、 -75% のエラーになります。オプティマイザーが頻出値統計を使用できる場合、行の数は 40 と見積もられ、エラーなしになります。

2 つの行が述部 C1 = 1 を満たす、別の例について考慮します。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。以下の公式を使って、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、オプティマイザーは、たとえば以下のように前述の公式 (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
```

変位値統計の例

以下の変位値統計の説明では、「K 変位値」という用語が使用されます。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 V 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ソートすることで計算できます。 K 変位値は、そのソートされた列の K 番目の行にあるデータ値です。

変位値統計を使用できる場合、以下の例で示されているように、オプティマイザーが範囲述部を満たす行の数をより正確に見積もることができます。以下の値を含む列 (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

次のように、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。データ分散が均一であると仮定し、前述の公式 (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。

変位値統計を使用できる場合、オプティマイザーは、変位値の 1 つから最も大きい値である 8.5 を見つけ、それに対応する K の値 7 を使用して行の数を見積もることにより、同じ述部 (C <= 8.5) を満たす行の数を見積もります。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。オプティマイザーにおいてデータ分散が均一であると想定して公式 (1) が使用される場合、述部を満たす行の数は 1 として見積もられ、エラーは -87.5% になります。

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。ただし、オプティマイザーは変位値を使用して、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、オプティマイザーは線形補間を使用します。

```
       10 - 8.5
r_2 *= ---------- x (値を持つ行の数 > 8.5 かつ <= 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。

前述の例で変位値により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 から 10 でクラスター化されているのに、標準の見積公式ではそのデータ値が 0 から 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。データ値の分散状況からは、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。データ分散が均一であると仮定し、公式 (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。 公式 (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
```

線形補間を使用すると、r_2 は次のように見積もられます。

```
  30 - 20
  ------- x (値を持つ行の数 > 20 かつ <= 40)
  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。

詳細索引統計

DETAILED 節を指定して索引に RUNSTATS を実行する場合、必要なデータ・ページ・フェッチの数を、さまざまなバッファー・プール・サイズに基づいてオプティマイザーが見積もることのできる、統計情報を収集します。この追加情報は、索引を介して表にアクセスするコストを、オプティマイザーがより正確に見積もる助けになります。

注: 詳細な索引統計を収集する場合、 RUNSTATS はより多くのメモリーを必要とし、 CPU の処理にかかる時間も長くなります。統計的に有効な数の項目のみに関して計算された情報で、 SAMPLED DETAILED オプションは、統計ヒープとして 2MB が必要です。この追加メモリー要件用に、追加の 488 4K ページを *stat_heap_sz* データベース構成パラメーターの設定に割り振ってください。ヒープが小さすぎた場合、 RUNSTATS は統計の収集を試行する前にエラーを戻します。

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 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。

DETAILED 統計は、完全な索引スキャンがさまざまなバッファー・サイズの下で行われるとした場合に、表のデータ・ページにアクセスするのに必要な物理入出力の数に関する、簡潔な情報を提供します。 RUNSTATS は索引のページをスキャンして、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 つだけの場合、索引によって参照される新しいページごとに、ページ不在になります。さらに悪い状況として、各行が別のページを参照する場合、入出力の数が、多くても索引付けされた表内の行と同じになります。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、すべての表ページの読み取りが一度だけ行われます。結果として、物理入出力の数はバッファー・サイズの単調で非増加の関数になります。

また統計情報では、索引順序に対する表の行のクラスタリングの程度に関する、より優れた見積もりも提供されます。索引との関連でクラスター化が行われている表の行が少ないほど、索引を介して表の行にアクセスしなければならない入出力が多数あります。オプティマイザーは索引を介した表へのアクセスのコストを見積もる際に、バッファー・サイズとクラスタリングの程度の両方を考慮します。

索引に含まれていない参照列を照会する場合には、 DETAILED 索引統計を収集しなければなりません。また、DETAILED 索引統計は以下のような状況で使用する必要があります。

- 表にクラスタリングの程度が異なる複数の非クラスタリング索引が存在する場合
- クラスタリング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

予備知識なしでは、またはさまざまなバッファー・サイズでの索引スキャンを強制的に行って、その結果の物理入出力をモニターするのでなければ、これらの条件を

評価するのは困難です。これらの状況が生じ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、検査して、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。

サブエレメント統計

ブランクで区切られているサブフィールドまたはサブエレメントを含む列が表に含まれており、照会が WHERE 節でそれらの列を参照している場合、アクセス・プランを最適なものにするために、サブエレメント統計を収集しなければなりません。

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 fruit fly 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%'
```

列のサブエレメント構造に関して、オプティマイザー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。

以下の統計は、 LIKE STATISTICS 節を指定した RUNSTATS を実行した場合に収集されます。

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。

DB2_LIKE_VARCHAR レジストリー変数は、次の形式の述部をオプティマイザーが処理する方法に影響します。

```
COLUMN LIKE '%xxxxxx'
```

xxxxxx には任意の文字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)。「wildcard LIKE predicates」として述べられます。 すべての述部に対して、オプティマイザー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 述部の場合、オプティマイザーは、一致する COLUMN が連結されている一連のエレメントの構造を含んでいると推定し、前後の % 文字を含まないストリングの長さに基づいて、各エレメントの長さを見積もります。

サブエレメント統計の値を調べるには、 SYSIBM.SYSCOLUMNS を照会します。例:

```
select substr(NAME,1,16), SUB_COUNT, SUB_DELIM_LENGTH
   from sysibm.syscolumns where tbname = 'DOCUMENTS'
```

注: LIKE STATISTICS 節を使用する場合、 RUNSTATS は時間がかかることがあります。例えば、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、オーバーヘッドのパーセンテージは少なくなります。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。

ユーザーが更新できるカタログ統計

ユーザー定義関数の統計

ユーザー定義関数 (UDF) に関する統計情報を作成する場合、SYSSTAT.FUNCTIONS カタログ・ビューを編集します。 UDF 統計が使用可能であれば、オプティマイザーは各種アクセス・プランのコストを見積もる際にそれらを使用できます。 RUNSTATS ユーティリティーでは、 UDF の統計は収集されません。使用できる統計がないなら、統計の列の値は -1 になり、オプティマイザーは単純な UDF を前提とするデフォルト値を使用します。

パフォーマンスを向上させるための見積もりを提供できる、統計列についての情報を、次の表にまとめます。

表 60. 関数統計 (SYSCAT.ROUTINES 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。
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: 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: 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します。
- PERCENT_ARGBYTES: 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。
- UDF は計算しか実行しないため、INITIAL_INSTS、 IOS_PER_INVOC、IOS_PER_ARGBYTE、および INITIAL_IOS は 0 に設定します。

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 つの引数を入力とし、第 2 引数の中で、第 1 引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 引数の長さが第 2 引数と比べると十分に小さく、第 2 引数の平均 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、最初の引数は検出されず、2 番目の引数全体が探索されることになります。
- 最初の引数は 2 番目の引数内のどの場所にも現れる可能性があるため、最初の引数が検出されるとき、平均して 2 番目の引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、スクラッチパッド域の設定コストを記録するなど、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。

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。

モデル化および what-if の計画のカタログ統計

表および索引の実際の状態を反映しないものの、計画を目的としてデータベースに行うことのできるさまざまな変更を調べられるように、システム・カタログ内の統計情報を変更できます。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します。
- 「what-if」照会パフォーマンス分析を実行します。

実動システムでの手動による統計の更新は行わないでください。そのようにすると、動的 SQL または XQuery ステートメントを含む実動照会で最適なアクセス・プランを、オプティマイザーが選択できません。

要件

表と索引およびそれらのコンポーネントの統計を変更するには、データベースに対する明示的な DBADM 権限が必要です。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。 DBADM はすべてのユーザーの統計行を見ることができ、 SYSSTAT スキーマで定義されているビューに対して SQL UPDATE ステートメントを実行して、それらの統計列の値を更新できます。

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。 DBADM 権限がない場合、各データベース・オブジェクトに対する以下の権限があれば、それぞれのオブジェクトの統計を変更できます。

- 表に対する明示的な CONTROL 特権。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。
- フェデレーテッド・データベース・システム内のニックネームに対する明示的な CONTROL 特権。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 オプティマイザーが生成するグローバル・アクセス戦略だけです。
- ユーザー定義関数 (UDF) の所有権

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'
```

カタログ統計を手動で更新するときは慎重に行ってください。軽率な変更により、以降の照会のパフォーマンスに対して重大な影響が及ぶ可能性があります。テストまたはモデル化のために使用する非実動データベースでも、以下の方法のいずれかを使用して、それらの表に適用された更新をリフレッシュし、統計を整合状態にすることができます。

- 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、最新表示します。
- 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、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、オプティマイザーが特定の統計値か値の組み合わせを無効と判断する場合があります。デフォルト値が使用されて、警告が出されます。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。

実動データベースのモデル化の統計

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 または XQuery 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、ビュー、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。データベースを再作成して統計を設定するスクリプトで、 `-e` オプションと `-m` オプションを一緒に使用できます。

`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。オプティマイザー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
```

コントロール・センターにも、この *db2look* ユーティリティー (「DDL の生成」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 または z/OS 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 または z/OS オブジェクトを抽出して、 DB2 Database for Linux, UNIX, and Windows 内にそのオブジェクトを再作成する場合に非常に役立ちます。

DB2 Database for Linux, UNIX, and Windows の統計と OS/390 の統計の間にはいくつか違いがあります。 db2look ユーティリティーは、適切な場合に DB2 for OS/390 and z/OS から DB2 Database for Linux, UNIX and Windows への適切な変換を実行し、 DB2 for OS/390 で対応するものが存在しない DB2 Database for Linux, UNIX and Windows 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 for OS/390 and z/OS の統計を DB2 Database for Linux, UNIX and Windows 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 Database for Linux, UNIX and Windows の統計列を、「S390_x」は DB2 for OS/390 and z/OS 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 for OS/390 and z/OS には、PCTPAGES という別の統計があり、表の行を含んでいるアクティブな表スペース・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 または z/OS 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 for OS/390 and z/OS の 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 Database for Linux, UNIX, and Windows 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 for OS/390 の SYSIBM.SYSCOLUMNS には列 SEQNO がありません。これは DB2 で必要なため、db2look により自動的に生成されます。

手動でのカタログ統計更新の一般規則

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用のビューに保管するということです。さらに、各種統計相互間のリレーションシップ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 (列内の固有値の数は、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 に減らすと仮定します。このとき最初に SYSSTAT.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 SYSSTAT.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 に増やす場合にも、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 であるか、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 から 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。

初めて表が作成されるとき、システム・カタログ統計は表に統計がないことを示す -1 に設定されます。統計が収集されるまで、 DB2 では SQL または XQuery ステートメントのコンパイルおよび最適化にデフォルト値を使用します。 新しい値がデフォルト値と不整合の場合、表または索引の統計の更新が失敗することがあります。このため、表の作成後、表または索引の統計の更新を試みる前に、RUNSTATS を実行することをお勧めします。

手動での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。

- SYSSTAT.COLUMNS の HIGH2KEY および LOW2KEY を手動で更新する場合、生成される値の性質は次のようになります。
  - HIGH2KEY、LOW2KEY の値は、対応するユーザー列のデータ・タイプの有効値でなければなりません。
  - 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 (引用符は含みません。これを含めるとストリング長は最大 68 になります)。つまり、HIGH2KEY、LOW2KEY 値の判別の際には、対応するユーザー列の値の最初の 33 文字だけが考慮されます。
  - HIGH2KEY/LOW2KEY 値は、UPDATE ステートメントの SET 節でコスト計算の操作なしに使用できるような方法で格納されます。これは、文字ストリングの場合、単一引用符がストリングの先頭と最後に追加され、ストリング中の既存のすべての引用符に対しては余分の引用符が追加されるということです。HIGH2KEY、LOW2KEY のユーザー列値および対応する値の例を以下の表に示します。

表 61. HIGH2KEY および LOW2KEY 値のデータ・タイプ

ユーザー列のデータ・タイプ	ユーザー・データ	対応する HIGH2KEY、LOW2KEY 値
INTEGER	-12	-12
CHAR	abc	'abc'
CHAR	ab'c	'ab''c'

  - 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。
- 列のカーディナリティー (SYSSTAT.COLUMNS 内の COLCARD 統計) は、対応する表または統計ビュー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。
- 列の NULL の数 (SYSSTAT.COLUMNS 内の NUMNULLS 統計) は、対応する表または統計ビュー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。
- 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。

手動での分散統計の更新の規則

手動で分散統計を更新するのは、実動データベースをモデル化するときか、人為的に構成されたデータベースに対して what-if テストを実行する場合だけにしてください。実動データベースに対する分散統計の更新は行わないでください。

カタログ内のすべての統計が整合しているか確認してください。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、その列の固有値の数 (この数は、カタログ・ビュー 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 で分散統計が使用できる場合に、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するとします。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、照会の実行時にオプティマイザー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。

手動での表およびニックネーム統計の更新の規則

SYSSTAT.TABLES で更新できる統計値は、
CARD、FPAGES、NPAGES、OVERFLOW だけです。MDC 表の場合は
ACTIVE_BLOCKS も含まれます。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 値以上で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「取り出し (Fetch)」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「取り出し (Fetch)」値よりも大きくなければなりません。

フェデレーテッド・データベース・システムで作業している場合、リモート・ビューに対するニックネームについての統計を手操作で提供または更新するときには、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・ビュー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 オプティマイザー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・ビューが SELECT リスト上で列関数が適用されていない単一の基本表上に定義されている場合が含まれます。複合ビューでは、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 オプティマイザーがビュー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・ビューを作成することを考慮してください。

手動での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、 32 ビット・オペレーティング・システムの場合は MIN(NPAGES, 524287)、 64 ビット・オペレー

ティング・システムの場合は MIN(NPAGES, 2147483647) 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'
```

説明

```
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. リレーショナル索引の場合、以下の規則は、FIRSTKEYCARD、FIRST2KEYCARD、FIRST3KEYCARD、FIRST4KEYCARD、FULLKEYCARD、および INDCARD 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 <= INDCARD == CARD
     ```
4. XML データの索引の場合、以下の規則は、FIRSTKEYCARD、FIRST2KEYCARD、FIRST3KEYCARD、FIRST4KEYCARD、FULLKEYCARD、および INDCARD に適用されます。
   - これらの関連は以下のようになる必要があります。

```
FIRSTKEYCARD <= FIRST2KEYCARD <= FIRST3KEYCARD <= FIRST4KEYCARD
   <= FULLKEYCARD <= INDCARD
```

5. 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。

第 21 章 ルーチン

ストアード・プロシージャーのガイドライン

ストアード・プロシージャーは、リモート・データベースに対する 1 つの呼び出しを許可して、多くの状態が繰り返されるデータベース・アプリケーション環境でプログラム式のプロシージャーを実行します。例えば、固定データ集合を受信したり、データベースに対して同じ複数の要求の集合を実行したり、固定データ集合を戻したりする際には、データベースへのアクセスが複数に及ぶことがあります。

リモート・データベースに関する 1 つの SQL ステートメントを処理する際には、1 つの要求と 1 つの受信という 2 つの伝送を送信する必要があります。 1 つのアプリケーションには多数の SQL ステートメントが含まれているので、作業を完了するには多数の伝送が必要です。

しかし、IBM データ・サーバー・クライアントが多数の SQL ステートメントをカプセル化したストアード・プロシージャーを使用する場合は、プロセス全体で必要な伝送は 2 つのみです。

普通ストアード・プロシージャーは、データベース・エージェントから分離されているプロセスで実行されます。この分離により、ルーターを使用して通信するには、ストアード・プロシージャーとエージェント・プロセスが必要になります。しかし、エージェント・プロセスで実行される特殊な種類のストアード・プロシージャーを使用するとパフォーマンスが向上する可能性がありますが、データやデータベースが破壊されるという重大なリスクを負うことになります。

これらのリスクがあるストアード・プロシージャーは、*not fenced* として作成されます。not fenced ストアード・プロシージャーの場合、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からストアード・プロシージャーが分離されません。ストアード・プロシージャーの操作によりデータベース制御構造が不慮または故意に損傷されないようにすることを DBA が保証するには、*not fenced* オプションを省略します。

データベースが損傷するリスクがあるので、*not fenced* ストアード・プロシージャーを使用するのは、パフォーマンスの恩恵を最大限受ける必要がある場合に**限定**してください。さらに、not fenced ストアード・プロシージャーとして実行できるようにする前に、プロシージャーが適切にコード化されており、徹底的にテストされていることを絶対に確認してください。 not fenced ストアード・プロシージャーの実行中に致命的エラーが生じると、データベース・マネージャーは、そのエラーがアプリケーションまたはデータベース・マネージャー・コードのどちらで起きたかを判別し、該当するリカバリーを実行します。

not fenced ストアード・プロシージャーによりデータベース・マネージャーがリカバリーできないほど破壊されることがあり、その結果としてデータが脱落したり、データベースが破壊される可能性さえあります。 not fenced トラステッド・ストアード・プロシージャーを実行する際には、最大限の注意を払ってください。ほとんどすべての場合に、アプリケーションのパフォーマンス分析を適切に行うと、結果

として not fenced ストアード・プロシージャーを使用しなくてもパフォーマンスは良好になります。例えば、トリガーによりパフォーマンスが向上する可能性があります。

SQL プロシージャーのパフォーマンスの改善

DB2 による SQL PL とインライン SQL PL のコンパイルの概要

SQL プロシージャーのパフォーマンスを改善する方法を説明する前に、 CREATE PROCEDURE ステートメントの実行時に DB2 が SQL プロシージャーをコンパイルする方法について取り上げる必要があります。

SQL プロシージャーの作成時に、 DB2 は、プロシージャー本体の中にある SQL 照会とプロシージャー・ロジックを分離します。 SQL 照会については、パフォーマンスの最大化のために、パッケージ内のセクションに静的にコンパイルします。静的にコンパイルした照会のセクションの主な中身は、 DB2 オプティマイザーがその照会のために選択したアクセス・プランです。パッケージとは、そのようなセクションの集合です。パッケージとセクションの詳細については、「DB2 SQL 解説書」を参照してください。一方、プロシージャー・ロジックは、ダイナミック・リンク・ライブラリーにコンパイルします。

プロシージャーの実行時に、プロシージャー・ロジックから SQL ステートメントに制御が移るたびに、 DLL と DB2 エンジンとの間で「コンテキストの切り替え」が発生します。 DB2 バージョン 8.1 以降、 SQL プロシージャーは「unfenced モード」で実行されます。つまり、 DB2 エンジンと同じアドレッシング・スペースで実行されるということです。したがって、ここで言う「コンテキストの切り替え」とは、オペレーティング・システム・レベルで発生する完全な「コンテキストの切り替え」ではなく、むしろ DB2 内の層の切り替えです。頻繁に呼び出されるプロシージャー (OLTP アプリケーション内のプロシージャーなど) や、多数の行を処理するプロシージャー (データ・クレンジングを実行するプロシージャーなど) でコンテキストの切り替えの数を減らせば、パフォーマンスにかなりの影響を与えることができます。

SQL PL を含んだ SQL プロシージャーは、個々の SQL 照会をパッケージ内の各セクションに静的にコンパイルすることによってインプリメントするのに対し、インライン SQL PL 関数は、その名が示すとおり、関数の本体を、関数を使用する照会の中にインライン化することによってインプリメントします。 SQL 関数内の各照会は、あたかも関数本体が 1 つの照会であるかのように一緒にコンパイルされます。このコンパイルは、その関数を使用するステートメントのコンパイルが行われるたびに発生します。ただし、SQL プロシージャーの場合とは異なり、 SQL 関数内のプロシージャー・ステートメントは、データ・フロー・ステートメントとは別の層で実行されるわけではありません。したがって、プロシージャー・ステートメントとデータ・フロー・ステートメントの間で制御が移るたびに、コンテキストの切り替えが発生するわけではないということです。

ロジック内に副作用がなければ SQL 関数を使用する

このように、プロシージャー内の SQL PL と関数内のインライン SQL PL とではコンパイルの方法が違うので、プロシージャー・コードが SQL データを照会するだけでデータを変更しない限り、つまり、データベース内外のデータに関する副作

用がない限り、プロシージャー・コードは、プロシージャー内よりも関数内にあったほうが実行速度が上がると考えられます。

ただし、このようなメリットを生かせるのは、実行する必要のあるすべてのステートメントが SQL 関数内でサポートされている場合に限られます。 SQL 関数には、データベースを変更する SQL ステートメントを組み込めません。また、関数のインライン SQL PL として使用できるのは、 SQL PL のサブセットにすぎません。例えば、 CALL ステートメントの実行、カーソルの宣言、 SQL 関数による結果セットの生成などは実行できません。

以下に示すのは、パフォーマンスを最大化する目的で SQL 関数に変換するのに適している SQL PL を含んだ SQL プロシージャーの一例です。

```
CREATE PROCEDURE GetPrice (IN Vendor CHAR&(20&),
                           IN Pid INT, OUT price DECIMAL(10,3))
LANGUAGE SQL
BEGIN
  IF Vendor eq; ssq;Vendor 1ssq;
    THEN SET price eq; (SELECT ProdPrice
                         FROM V1Table
                         WHERE Id = Pid);
  ELSE IF Vendor eq; ssq;Vendor 2ssq;
    THEN SET price eq; (SELECT Price FROM V2Table
                         WHERE Pid eq; GetPrice.Pid);
  END IF;
END
```

これを SQL 関数として記述すると、以下のようになります。

```
CREATE FUNCTION GetPrice (Vendor CHAR(20), Pid INT)
RETURNS  DECIMAL(10,3)
LANGUAGE SQL
BEGIN
  DECLARE price DECIMAL(10,3);
  IF Vendor = 'Vendor 1'
    THEN SET price = (SELECT ProdPrice
                       FROM V1Table
                       WHERE Id = Pid);
  ELSE IF Vendor = 'Vendor 2'
    THEN SET price = (SELECT Price FROM V2Table
                WHERE Pid = GetPrice.Pid);
  END IF;
  RETURN price;
END
```

関数の呼び出しは、プロシージャーの呼び出しとは異なることも覚えておく必要があります。関数を呼び出すには、 VALUES ステートメントを使用するか、SELECT ステートメントや SET ステートメントなどの中で式が有効な場所に関数を記述して呼び出します。以下はいずれも、この新しい関数を呼び出す方法として有効です。

```
VALUES (GetPrice('IBM', 324))

SELECT VName FROM Vendors WHERE GetPrice(Vname, Pid) < 10

SET price = GetPrice(Vname, Pid)
```

SQL PL プロシージャー内で 1 つのステートメントを使用すれば十分な場合に複数のステートメントを使用しない

基本的に SQL は簡潔に記述するほうが良いのですが、実際には簡潔でない SQL を記述してしまうこともよくあります。例えば、次のような SQL ステートメントがあるとしましょう。

```
INSERT INTO tab_comp VALUES (item1, price1, qty1);
INSERT INTO tab_comp VALUES (item2, price2, qty2);
INSERT INTO tab_comp VALUES (item3, price3, qty3);
```

これは、以下の 1 つのステートメントとして記述できます。

```
INSERT INTO tab_comp VALUES (item1, price1, qty1),
                            (item2, price2, qty2),
                            (item3, price3, qty3);
```

この複数行の挿入ステートメントの実行にかかる時間は、元の 3 つのステートメントの実行にかかる時間のほぼ 3 分の 1 です。これだけを取り出したコードであれば、パフォーマンスの改善はごくわずかでしょうが、ループやトリガー本体などの中でこのコード断片を繰り返し実行する場合は、かなりの改善が期待できます。

同じように、以下のような一連の SET ステートメントがあるとしましょう。

```
SET A = expr1;
SET B = expr2;
SET C = expr3;
```

これは、以下の 1 つの VALUES ステートメントとして記述できます。

```
VALUES expr1, expr2, expr3 INTO A, B, C;
```

この書き換えでは、元の一連のステートメントのセマンティクスをそのまま保持しています。ただし、元のいずれか 2 つのステートメントの間に依存関係が存在する場合は別です。この点を示す以下の例について考えてみましょう。

```
SET A = monthly_avg * 12;
SET B = (A / 2) * correction_factor;
```

この 2 つのステートメントを以下のように書き換えるとしましょう。

```
VALUES (monthly_avg * 12, (A / 2) * correction_factor) INTO A, B;
```

この場合は、元のセマンティクスがそのまま保持されていません。 INTO キーワードの前の両方の式は「並列的に」評価されるからです。つまり、*B* に代入される値は *A* に代入される値に基づくというのが、元のステートメントで意図されているセマンティクスですが、書き換え後のコードにはそれが反映されていないということです。

複数の SQL ステートメントを 1 つの SQL 式にまとめる

SQL 言語には、他のプログラム言語と同じように、 2 種類の条件構造体が用意されています。つまり、プロシージャー型の構造体 (IF ステートメント、CASE ステートメント) と関数型の構造体 (CASE 式) です。 1 つの計算処理を表すためにどちらのタイプの構造体でも使用できる状況では、ほとんどの場合、どちらを使用するかは好みの問題です。ただし、 CASE 式によって記述したロジックは、 CASE ステートメントや IF ステートメントによって記述したロジックよりもコンパクトであり、効率的でもあります。

以下の SQL PL コード断片について考えてみましょう。

```
IF (Price <= MaxPrice) THEN
  INSERT INTO tab_comp(Id, Val) VALUES(Oid, Price);
ELSE
  INSERT INTO tab_comp(Id, Val) VALUES(Oid, MaxPrice);
END IF;
```

この IF 節の条件は、 tab_comp.Val 列に挿入する値を決定するという目的のためだけに使用しています。プロシージャー層とデータ・フロー層の間のコンテキストの切り替えを避けるために、この同じロジックを CASE 式付きの 1 つの INSERT で記述すれば、以下のようになります。

```
INSERT INTO tab_comp(Id, Val)
       VALUES(Oid,
            CASE
               WHEN (Price <= MaxPrice) THEN Price
               ELSE MaxPrice
            END);
```

CASE 式は、スカラー値が有効な場所であればどんなコンテキストでも使用できるというのは注目に値します。特に便利なのは、代入の右辺で使用できるということです。例:

```
IF (Name IS NOT NULL) THEN
  SET ProdName = Name;
ELSEIF (NameStr IS NOT NULL) THEN
  SET ProdName = NameStr;
ELSE
  SET ProdName = DefaultName;
END IF;
```

これは、以下のように記述できます。

```
SET ProdName = (CASE
                  WHEN (Name IS NOT NULL) THEN Name
                  WHEN (NameStr IS NOT NULL) THEN NameStr
                  ELSE  DefaultName
                END);
```

実際に、この例の場合はさらに優れた解決策があります。

```
SET ProdName = COALESCE(Name, NameStr, DefaultName);
```

SQL を分析し、その書き換えを検討するために時間を取ることのメリットを過小評価しないでください。パフォーマンス上のメリットは、プロシージャーの分析と書き換えにかけた時間の何倍もの価値があるはずです。

SQL の一括設定のセマンティクスを活用する

ループ、代入、カーソルなどのプロシージャー型の構造体を使用すれば、 SQL DML ステートメントだけでは記述できない計算処理を記述できます。その一方で、プロシージャー・ステートメントが手元にあると、実際には SQL DML ステートメントだけで計算処理を記述できる場合でも、プロシージャー・ステートメントに頼ってしまう危険があります。すでに見たとおり、プロシージャーによる計算処理は、 DML ステートメントによって記述した等価の計算処理よりもパフォーマンスが桁違いに落ちることがあります。以下のコード断片について考えてみましょう。

```
DECLARE cur1 CURSOR FOR SELECT col1, col2 FROM tab_comp;
OPEN cur1;
FETCH cur1 INTO v1, v2;
```

```
WHILE SQLCODE <> 100 DO
  IF (v1 > 20) THEN
    INSERT INTO tab_sel VALUES (20, v2);
  ELSE
    INSERT INTO tab_sel VALUES (v1, v2);
  END IF;
  FETCH cur1 INTO v1, v2;
END WHILE;
```

まずループ本体は、『複数の SQL ステートメントを 1 つの SQL 式にまとめる』の項で取り上げた書き換えを適用することによって改善できます。

```
DECLARE cur1 CURSOR FOR SELECT col1, col2 FROM tab_comp;
OPEN cur1;
FETCH cur1 INTO v1, v2;
WHILE SQLCODE <> 100 DO
  INSERT INTO tab_sel VALUES (CASE
                                WHEN v1 > 20 THEN 20
                                ELSE v1
                              END, v2);
  FETCH cur1 INTO v1, v2;
END WHILE;
```

しかし、よく見ると、このコード・ブロック全体は、サブ SELECT 付きの 1 つの INSERT として記述できます。

```
INSERT INTO tab_sel (SELECT (CASE
                               WHEN col1 > 20 THEN 20
                               ELSE col1
                             END),
                             col2
                     FROM tab_comp);
```

元のコードでは、 SELECT ステートメントの各行で、プロシージャー層とデータ・フロー層の間のコンテキストの切り替えが発生します。一方、書き換えた後のコードでは、コンテキストの切り替えがまったく発生しないので、オプティマイザーは計算処理全体をグローバルに最適化できます。

ただし、以下のように各 INSERT ステートメントの対象になっている表がそれぞれ異なる場合、これほど劇的な単純化は不可能です。

```
DECLARE cur1 CURSOR FOR SELECT col1, col2 FROM tab_comp;
OPEN cur1;
FETCH cur1 INTO v1, v2;
WHILE SQLCODE <> 100 DO
  IF (v1 > 20) THEN
    INSERT INTO tab_default VALUES (20, v2);
  ELSE
    INSERT INTO tab_sel VALUES (v1, v2);
  END IF;
  FETCH cur1 INTO v1, v2;
END WHILE;
```

それでも、以下のようにすれば、SQL の一括設定の機能を活用できます。

```
INSERT INTO tab_sel (SELECT col1, col2
                     FROM tab_comp
                     WHERE col1 <= 20);
INSERT INTO tab_default (SELECT col1, col2
                         FROM tab_comp
                         WHERE col1 > 20);
```

このようにカーソル・ループを除去するには時間がかかりますが、既存のプロシージャー・ロジックのパフォーマンスを改善できることを考えれば、そのための価値は十分にあると言えます。

DB2 オプティマイザーに常に最新の情報を提供する

プロシージャーの作成時に、個々の SQL 照会は、パッケージ内の各セクションにコンパイルされます。 DB2 オプティマイザーが照会の実行プランを選択するための基礎になるのは、特に表の統計 (表のサイズや、列内のデータ値の相対度数など) と、照会のコンパイルの時点で使用できる索引です。表にかなりの変更があった場合は、その表に関する統計を DB2 で収集するべきです。また、統計を更新した場合や、新しい索引を作成した場合は、その表を使用する SQL プロシージャーに関連するパッケージを再バインドして、最新の統計と索引に基づくプランを DB2 で作成するようにしてください。

表の統計を更新するには、RUNSTATS コマンドを使用します。 SQL プロシージャーに関連するパッケージを再バインドするには、 DB2 バージョン 8.1 に用意されている REBIND_ROUTINE_PACKAGE 組み込みプロシージャーを使用します。例えば、プロシージャー MYSCHEMA.MYPROC のパッケージを再バインドするには、以下のコマンドを使用できます。

```
CALL SYSPROC.REBIND_ROUTINE_PACKAGE('P', 'MYSCHEMA.MYPROC', 'ANY')
```

「P」は、このパッケージがプロシージャーに対応していることを示し、 'ANY' は、関数とタイプの解決時に SQL パス内のすべての関数とタイプを対象にすることを示します。詳細については、「コマンド解説書」の『REBIND コマンド』の項目を参照してください。

配列の使用

配列を使用して、アプリケーションとストアード・プロシージャーの間でデータの集合を効果的に受け渡しし、リレーショナル表を使用せずに SQL プロシージャー内でデータの一時的な集合を格納し取り扱うことができます。SQL プロシージャー内で使用可能な配列の演算子を使って、データの保管と取り出しを効率的に行うことができます。アプリケーションが適度なサイズの配列を作成するなら、巨大な配列 (数メガバイト規模) を作成するよりも、はるかに良いパフォーマンスを得ることができます。これは配列全体がメイン・メモリーに格納されるためです。詳しい追加情報は、「*関連リンク*」セクションをご覧ください。

第 22 章 照会アクセス・プラン

SQL および XQuery コンパイラーの処理

SQL および XQuery コンパイラーは、いくつかのステップを実行して、実行可能なアクセス・プランを作成します。このステップを以下の図に示し、図の下のセクションで説明します。一部のステップは、フェデレーテッド・データベースの照会にのみ実行されることにご注意ください。

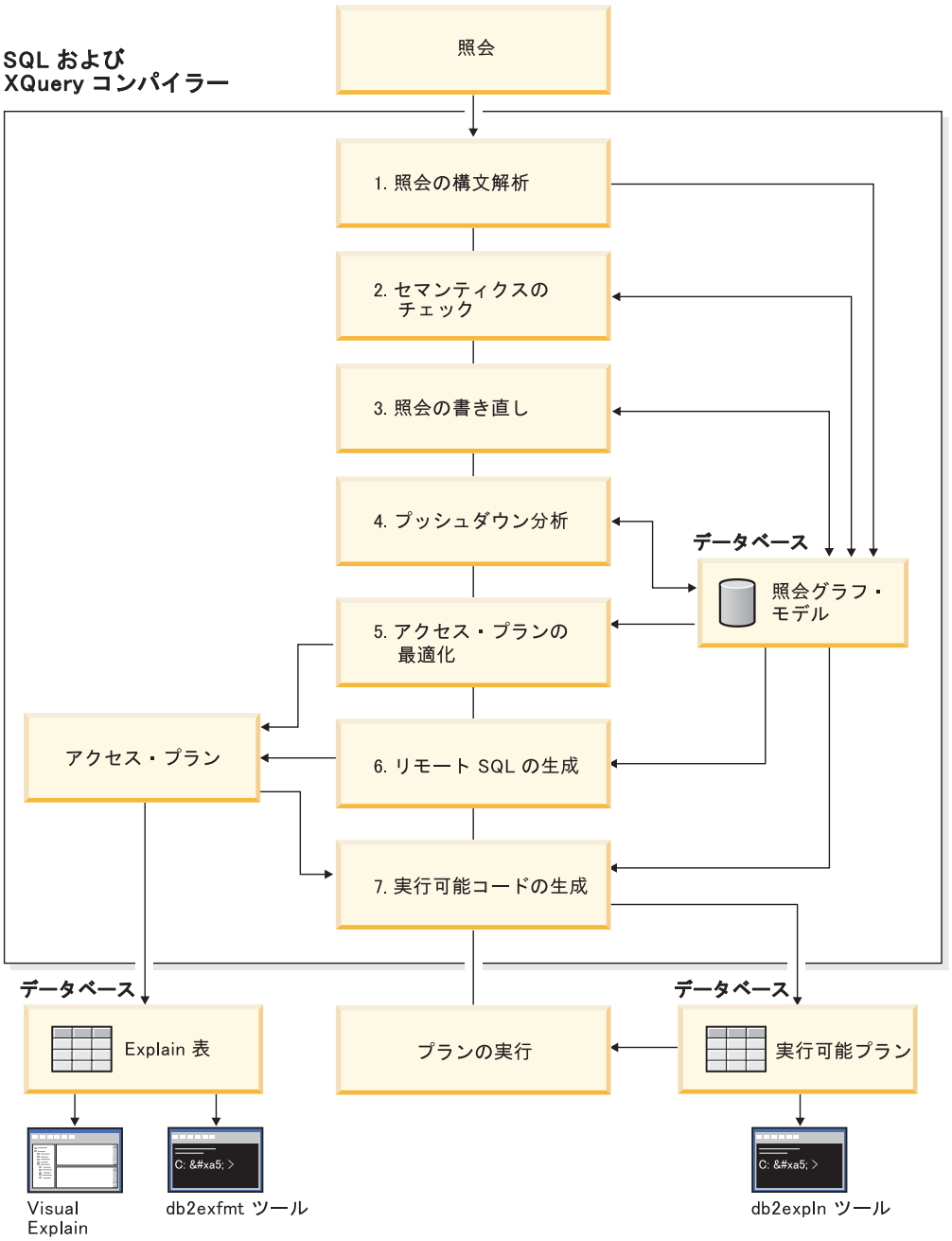


図 22. SQL および XQuery コンパイラーによって実行されるステップ

照会グラフ・モデル

照会グラフ・モデル とは、以下で説明するステップで処理される照会を表示するメモリー内の内部的データベースです。

1. **照会の構文解析**

   SQL および XQuery コンパイラーは、照会を分析して構文の妥当性検査を行います。構文エラーが検出されると、照会コンパイラーは処理を停止し、照会をサブミットしたアプリケーションに対して該当するエラーを戻します。構文解析が完了すると、照会の内部表示が作成され、照会グラフ・モデルに保管されます。

2. **セマンティクスのチェック**

   コンパイラーは、ステートメントのパーツ間に不整合がないことを確認します。セマンティクス検査の 1 つの簡単な例として、コンパイラーは、 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・タイプであるか検査します。

   コンパイラーはまた、機能上のセマンティクスを照会グラフ・モデルに追加します。それには、参照制約、表チェック制約、トリガー、およびビュー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、および分散キーを含む、照会のセマンティクスすべてが含まれます。

3. **照会の書き直し**

   コンパイラーは、照会グラフ・モデルに保管されているグローバルなセマンティクスを使用して、照会をもっと最適化しやすい形に変形し、その結果を照会グラフ・モデルに保管します。

   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、*一般述部プッシュダウン* といいます。パーティション・データベース環境では、以下の照会操作では、計算量が多くなります。

   - 集約
   - 行の再配分
   - 副照会の外側にある表の列への参照を含む副照会である相関副照会

   パーティション・データベース環境での一部の照会では、照会の書き直しの一環として非相関化が行われます。

4. **プッシュダウン分析 (フェデレーテッド・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 (プッシュダウン) できるかどうかを、オプティマイザーに通知することです。このタイプのプッシュダウン・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、一般の述部のプッシュダウン操作を拡張するものとなります。

   フェデレーテッド・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。

5. **アクセス・プランの最適化**

   コンパイラーのうちのオプティマイザーの部分は、照会グラフ・モデルを入力データとして使用して、照会に答えるための多数の代替実行プランを生成します。各代替プランの実行コストを見積もるため、オプティマイザーは表、索引、列、および関数の統計を使用します。そして、見積実行コストの最も低いプランを選択します。オプティマイザーは、照会グラフ・モデルを使用して照会のセマンティクスを分析したり、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。

オプティマイザーの部分では、別のタイプのプッシュダウン操作である、集約およびソート 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。

さらにオプティマイザーでは、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。環境にパーティション・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) 環境で照会内並列処理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、オプティマイザー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。

コンパイラーのこのステップでの出力は、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となります。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。

6. **リモート SQL 生成 (フェデレーテッド・データベース)**

   オプティマイザー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。リモート SQL 生成のステップでは、データ・ソースごとに実行される操作のために、データ・ソース固有の SQL ダイアレクトに基づく有効な SQL ステートメントが生成されます。

7. **実行可能コードの生成**

   最終ステップでは、コンパイラーはアクセス・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。

   ホスト変数、特殊レジスター、またはパラメーター・マーカーを持つ静的および動的 SQL および XQuery ステートメントの (再) 最適化の照会を使用可能にするには、パッケージを REOPT BIND オプションを使ってバインドします。これを使うと、パッケージに属し、かつホスト変数、パラメーター・マーカー、または特殊レジスターを含んだ SQL または XQuery ステートメントのアクセス・パスが、コンパイラーの選ぶデフォルトの推定値ではなく、これらの変数の値を使って最適化されます。照会の実行時に値が与えられてから、この最適化は実行されます。

   静的 SQL および XQuery ステートメントのアクセス・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。この情報は、*db2expln* ツールによって使用されます。

注: たびたび変更されるテーブルでは、RUNSTATS を適切なインターバルで実行してください。オプティマイザーが最も効率的なアクセス・プランを作成するには、テーブルとそのデータに関する最新の統計情報が必要です。アプリケーションを再バインドして、更新済みの統計を利用してください。 RUNSTATS を実行しなかっ

た (またはオプティマイザーが、 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、オプティマイザーは、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。占有されたブロックの合計数が ACTIVE_BLOCKS 列に保管されます。

照会書き直しのメソッドとその例

照会書き直し段階では、照会コンパイラーは SQL および XQuery ステートメントをより最適化しやすい形式に変換し、その結果として可能なアクセス・パスを改善することができます。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。

SQL または XQuery ステートメントに適用される照会書き直しの規則の数を変更するには、最適化クラスを変更します。照会書き直しの結果をいくつか表示させるには、 Explain 機能または Visual Explain を使用します。

照会の書き直しは、以下の 3 つの基本的な方法のいずれかで行えます。

- **操作のマージ**

  操作、特に SELECT 操作ができるだけ少ない照会を作成するため、 SQL および XQuery 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ます。以下の例に、マージ可能な操作をいくつか示します。

  – 例 - ビューのマージ

    ビューを使用する SELECT ステートメントでは、表の結合順序が制限されたり、余分な表結合が生成されたりすることがあります。照会書き直し時にビューをマージすれば、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

    SELECT ステートメントに副照会が含まれている場合、表の配列処理の選択が制限される可能性があります。

  – 例 - 余分な結合の除去

    照会書き直し時には、余分な結合を除去して SELECT ステートメントを単純化することができます。

  – 例 - 共用集約

    照会がさまざまな関数を使用している場合は、書き直しによって、実行する必要のある計算の数を減らすことができます。

- **操作の移動**

  照会を最小限の操作数と述部数で構成するため、コンパイラーは照会を書き直して照会操作を移動します。以下の例に、移動可能な操作をいくつか示します。

  – 例 - DISTINCT の除去

    照会書き直し時には、オプティマイザーで DISTINCT 操作の位置を移動して、その操作のコストを少なくすることができます。以降の例では、DISTINCT 操作は完全に除去されているケースを挙げています。

– 例 - 一般的な述部プッシュダウン

照会書き直し時に、オプティマイザーは述部を適用する順序を変更して、できるだけ早い機会により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

パーティション・データベース環境にいる場合、データベース・パーティション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・パーティション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) を減らすことは、照会の書き直し時の目標です。

- **述部の変換**

SQL および XQuery 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、変換可能な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、オプティマイザー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

照会書き直し時には、より効率的なアクセス・プランのために、OR 述部を IN 述部に変換することができます。また、IN 述部を OR 述部に変換すれば一層効率的なアクセス・プランが作成されるのであれば、 SQL および XQuery コンパイラーでそのような変換をすることもできます。

コンパイラー書き直しの例: ビューのマージ

EMPLOYEE 表に次の 2 つのビュー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ビューで、他方は $35,000 を超える収入がある従業員を表示するビューです。

```
    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 を超える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 つのビューをマージすると、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 つのビューからの 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、オプティマイザー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 つのビューの使う基本表が同じである場合、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 および XQuery 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。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 および XQuery 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。まずビューを作成します。

```
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、基本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、上記のビューへのアクセス権だけです。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。

- ビュー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、ツールなどが自動生成したクエリーは最適でないこともあります。

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、SQL および XQuery コンパイラーが、以下のような複数の関数を使用する照会を処理するとします。

```
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。

コンパイラー書き直しの例: DISTINCT の除去

EMPNO 列を EMPLOYEE 表の主キーとして定義した場合、次の照会は、

```
SELECT DISTINCT EMPNO, FIRSTNME, LASTNAME
  FROM EMPLOYEE
```

DISTINCT 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、主キーが選択されているため、 SQL および XQuery 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場合は、オプティマイザーはソートなどの必要な処理を含む計画を作成して、列を明確にする必要があります。

例 - 一般的な述部プッシュダウン

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ビューがあるとします。

```
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' がビュー D11_EMPLOYEE にプッシュされます。これにより、その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'
```

述部のプッシュダウンは、ビューに限定されません。述部がプッシュダウン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

パーティション・データベース環境では、 SQL および XQuery 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 上にパーティション化されていないので、各プロジェクトが各データベース・パーティション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 および XQuery 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一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・パーティションすべてにブロードキャストすることができます。

コンパイラー書き直しの例: 暗黙の述部

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、オプティマイザー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、オプティマイザーは結合する行の数を減らします。

例 - OR から IN への変換

OR 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 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 節に変換して、索引 OR 処理を実行できるようにすることがあります。

述部の分類とアクセス・プラン

ユーザー・アプリケーションはデータベースから一連の行を要求するのに、結果セットとして戻す特定の行に対する修飾子を指定した照会ステートメントを使います。通常この修飾子は照会の WHERE 節に指定します。こうした修飾子を*述部* と言います。述部は評価プロセスでいつどのように使用されるかによって 4 種類に分類できます。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 SARGable 述部
3. データ SARGable 述部
4. Residual 述部

注: *SARGable (SARGable 述部)* とは、各行を Fetch することにより、検索条件に該当するかどうかを確認できる条件節のことです。

次の表は述部カテゴリーの要約です。続くセクションでそれぞれのカテゴリーをもっと詳しく説明します。

表 62. 述部タイプの特性に関するサマリー

特性	**述部タイプ**			
	範囲区切り	**索引 SARGable**	**データ SARGable**	**Residual**
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

表 62. 述部タイプの特性に関するサマリー (続き)

特性	述部タイプ			
	範囲区切り	索引 SARGable	データ SARGable	Residual
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

範囲区切り述部および索引 SARGable 述部

範囲区切り述部は索引スキャンの範囲を限定します。それは索引探索の開始および停止キー値を提供します。索引 SARGable 述部は、探索の範囲を限定できませんが、述部が必要とする列は索引キーの一部であるため、索引を利用して評価することができます。たとえば、次の索引を考えてみてください。

```
INDEX IX1:  NAME    ASC,
            DEPT    ASC,
            MGR     DESC,
            SALARY  DESC,
            YEARS   ASC
```

次の WHERE 節を含む照会についても考えてください。

```
WHERE  NAME  = :hv1
 AND   DEPT  = :hv2
 AND   YEARS > :hv5
```

最初の 2 つの述部 (NAME = :hv1, DEPT = :hv2) は索引 SARGable 述部であり、YEARS > :hv5 は範囲区切り述部です。

オプティマイザーはこれらの述部を評価するにあたり、基本表を読むのではなく、索引データを使用します。これらの*索引 SARGable* 述部によって、表から読み取る必要のある行が少なくなりますが、アクセスする索引ページの数は変わりません。

XMLEXISTS および XMLTABLE 式に生じる XML データ上の述部は、XSCAN データ演算子スキャンでもサポートされます。これらの述部のうち、索引範囲スキャンにサポートされるものもあります。

データ SARGable 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 *SARGable* 述部と呼ばれます。通常このタイプの述部は表の個々の行にアクセスすることを必要とします。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索引から獲得できなかったものを取り出します。

たとえば、PROJECT 表に定義されている次の単一の索引を考えてみましょう。

```
INDEX IX0:  PROJNO ASC
```

この場合、次の照会では、`DEPTNO = 'D11'` 述部はデータ SARGable と見なされます。

```
SELECT PROJNO, PROJNAME, RESPEMP
  FROM PROJECT
 WHERE DEPTNO = 'D11'
 ORDER BY PROJNO
```

Residual 述部

Residual 述部は表へのアクセス以上の入出力コストを必要とします。以下の特性を持っている場合があります。

- 相関副照会を使用する
- ANY、ALL、SOME、または IN 節を含む定量化された副照会を使用する
- 表とは別のファイルに保存されている LONG VARCHAR や LOB のデータを読み取る

こうした述部は、リレーショナル・データ・サービスによって評価されます。

索引のみに適用された述部が、データ・ページがアクセスされる際に再度適用しなければいけなくなる場合があります。たとえば、索引 OR 結合または索引 ANDing 結合を使用するアクセス・プランは、データ・ページにアクセスする際には、常に Residual 述部を再適用します。

フェデレーテッド・データベースの照会コンパイラー・フェーズ

フェデレーテッド・データベースのプッシュダウン分析

フェデレーテッド・データベースでの照会において、オプティマイザーは、プッシュダウン分析を実行することによって、リモート・データ・ソースで操作を実行できるかどうかを判別します。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) の関数とすることができます。

注: 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。その場合は、ローカル・カタログ情報を変更することによって、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。

関数がリモート・データ・ソースにプッシュダウンできない場合、その関数は、照会のパフォーマンスに大きな影響を与える可能性があります。選択述部をデータ・ソースで評価したときではなく、ローカルに評価したときの影響を考慮してください。このような評価では、DB2 にリモート・データ・ソースから表全体を検索させ、それを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場合があります。またネットワークに制約があったり、表のサイズが大きかったりする場合も、それによってパフォーマンスが影響を受ける可能性があります。

プッシュダウンされない演算子も、照会のパフォーマンスに大きな影響を与えることがあります。たとえば、GROUP BY 演算子でリモート・データ・ソースをローカルに集約するなら、 DB2 にリモート・データ・ソースから表全体を検索させる必要も生じるかもしれません。

例として、ニックネーム N1 が、 DB2 for OS/390® and z/OS のデータ・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 つには従業員の姓が、そして別の列には給与が入っています。次のようなステートメントを考えてみましょう。

```
SELECT LASTNAME, COUNT(*)  FROM N1
   WHERE LASTNAME  > 'B' AND SALARY > 50000
   GROUP BY LASTNAME;
```

照合シーケンスが DB2 と DB2 for OS/390 and z/OS で同じかどうかによって、いくつかの可能性が考えられます。

- 照合シーケンスが同じ場合、照会の述部が DB2 for OS/390 and z/OS にプッシュダウンできる可能性は高くなります。通常は、DB2 に表全体をコピーしてからローカルに操作を実行するよりも、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方が効率的です。このような照会では、述部および GROUP BY 操作をデータ・ソースで実行できます。
- 照合シーケンスが異なる場合は、述部の全体をデータ・ソースで評価することはできません。ただし、オプティマイザーは、述部の SALARY > 50000 部分をプッシュダウンする方法をとる場合があります。範囲の比較は、依然として DB2 で実行する必要があります。
- 照合シーケンスが同じで、ローカル DB2 サーバーが非常に高速であることをオプティマイザーが認識している場合は、 GROUP BY 操作を DB2 でローカルに実行するのが最善の (最もコストのかからない) 方法であると判断される可能性があります。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プッシュダウン分析の一例です。

一般に、目的は、オプティマイザーに確実にデータ・ソース上で関数や演算子を評価させることにあります。関数や 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。この評価が行われる要因は、次の 3 つのグループに分類できます。

- サーバー特性
- ニックネーム特性
- 照会特性

プッシュダウンの使用に影響を与えるサーバー特性

特定のデータ・ソースに固有の要因により、プッシュダウンが行われるかどうかに影響が生じる場合があります。一般に、このような要因が存在するのは、DB2 で豊富な SQL ダイアレクトがサポートされているためです。このダイアレクトは、照会によって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持つ場合があります。 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、その中には、GROUP BY リストの項目数が制限されているものもあれば、 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 制約: それぞれのデータ・ソースには、異なる SQL 制約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。ある関数の値を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 制限: DB2 では、リモート・データ・ソースよりも大きい整数を使用できる場合がありますが、リモート制限を超える値をデータ・ソースへの送信ステートメントに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。

サーバーの特性: いくつかの要因は、このカテゴリーに分類されます。 1 つの例としては、NULL 値が最高値や最低値としてソートされているか、配列に依存するかという要素があります。 NULL 値がデータ・ソースにおいて DB2 とは異なるソートを受ける場合は、 NULL 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。

照合シーケンス: ローカルでソートや比較を行うためにデータを取り出すと、通常はパフォーマンスが低下します。したがって、データ・ソースで使うのと同じ照合シーケンスを使用するように、フェデレーテッド・データベースを構成することを考慮してください。データ・ソースが使用するのと同じ照合シーケンスを使うようにフェデレーテッド・データベースを構成し、 *collating_sequence* サーバー・オプションを「Y」に設定しているなら、オプティマイザーは、結果としてパフォーマンスの改善が得られる場合に多くの照会操作をプッシュダウンすることを考慮に入れることができます。

照合シーケンスが同一である場合は、以下の操作をプッシュダウンできるかもしれません。

- 文字データや数値データの比較
- 文字範囲比較述部
- ソート

ただし、フェデレーテッド・データベースとデータ・ソースとの間で NULL 文字の並び順序が違うと、異常な結果が発生する可能性があります。大文字小文字を区別しないデータ・ソースに比較ステートメントをサブミットすると、ステートメントが予期せぬ結果を戻すことがあります。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並び順序は同じです。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並び順序を割り当てます。

パフォーマンスを向上させるため、フェデレーテッド・サーバーでは、データ・ソースでソートや比較を行うことが可能です。例えば、DB2 for OS/390 and z/OS では、ORDER BY 節で定義されたソートは、 EBCDIC コード・ページに基づく照合シーケンスで実行されます。フェデレーテッド・サーバーを使用し、ORDER BY 節

でソートされた DB2 for OS/390 and z/OS データを検索する場合は、 EBCDIC コード・ページに基づいて事前定義された照合シーケンスを使用するようにフェデレーテッド・データベースを構成してください。

フェデレーテッド・データベースとデータ・ソースの照合シーケンスが異なる場合、DB2 はデータをフェデレーテッド・データベースに取り出します。これは、ユーザーが、フェデレーテッド・サーバーに定義されている照合シーケンスで並べられた照会結果を要求しているためで、フェデレーテッド・サーバーはデータをローカルに並べることによってこの要求に応答するからです。データ・ソースの照合シーケンスにある順序でデータを示させる必要がある場合は、照会をパススルー・モードでサブミットするか、データ・ソース・ビューでその照会を定義してください。

サーバー・オプション: いくつかのサーバー・オプションは、プッシュダウンが行われるかどうかに影響を与える場合があります。特に、*collating_sequence*、*varchar_no_trailing_blanks*、および *pushdown* の設定を検討してください。

DB2 タイプ・マッピングおよび関数マッピングの要因: DB2 のデフォルトのローカル・データ・タイプ・マッピングは、データの損失を防ぐため、データ・ソースの各データ・タイプに十分なバッファー・スペースを割り振るよう設計されています。特定のアプリケーションに合わせて、特定のデータ・ソースのタイプ・マッピングをユーザーの手でカスタマイズすることも可能です。たとえば、Oracle データ・ソース列の 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は、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。

次の 3 つのケースでは、データ・ソースでサポートされていない関数を DB2 で補うことができます。

- 関数がリモート・データ・ソースに存在しない。
- 関数は存在するが、オペランドの特性が関数の制限に違反している。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。
- 関数は、リモート側で評価されると違う値を戻すことがあります。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。照合シーケンスの異なるデータ・ソースでは、「より大」演算子が DB2 によってローカルに評価されると、異なった結果が返される可能性があります。

プッシュダウンの使用に影響を与えるニックネーム特性

次のニックネーム固有の要因により、プッシュダウンが行われるかどうかに影響が生じる場合があります。

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。オーバーフローが生じるのを潜在的に防ぐため、デフォルトのデータ・タイプ・マッピングを使用してください。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 のオプティマイザーが結合シーケン

スで評価を行える機会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したりするには、SQL ステートメント ALTER NICKNAME を使用します。

末尾ブランクが含まれていない列の識別には、 *varchar_no_trailing_blanks* オプションを使用します。コンパイラーのプッシュダウン分析ステップでは、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。

この列の値が常に末尾ブランクなしの数値であるかどうかを識別するには、*numeric_string* オプションを使用してください。

次の表は、これらのオプションについて説明しています。

表 63. 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、数値ストリングと末尾ブランクしか含まれない場合は、Y は指定しないでください。	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、列データのソートに干渉するブランクがこの列には含まれないことを、オプティマイザーに知らせることになります。このオプションは、データ・ソースの照合シーケンスが DB2 の照合シーケンス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、照合シーケンス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 63. 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します。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/ビュー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。ビューを含め、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

プッシュダウンの使用に影響を与える照会特性

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。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組み合わせる場合は、この操作を DB2 で実行する必要があります。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。

フェデレーテッド照会を評価する場合の分析のガイドライン

DB2 には、照会が評価される場面を示す 2 つのユーティリティーがあります。

- Visual Explain。このツールは、**db2cc** 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。

  照会をプッシュダウンする場合は、RETURN 演算子が示されるはずです。この RETURN 演算子は、標準の DB2 演算子です。ニックネームからデータを選択する SELECT ステートメントの場合は、SHIP 演算子も示されます。 SHIP 演算子は、フェデレーテッド・データベース操作に固有のものです。この演算子はデータ・フローのサーバー・プロパティーを変更し、リモート演算子とローカル演算子を区別します。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。

  INSERT、DELETE、または UPDATE 照会が完全にリモート・データベースにプッシュダウンできる場合は、アクセス・プランに SHIP ステートメントが示されることはないかもしれません。リモート側で実行されるすべての INSERT、UPDATE、および DELETE ステートメントは、RETURN 演算子で示されます。ただし、照会の全体をプッシュダウンできない場合は、 SHIP 演算子に、どの操作がリモート側で実行されるかが示されます。

- 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・ソースや DB2 で評価される理由

プッシュダウンの機会を増やす方法を考慮するときには、次のかぎとなる問題を考慮してください。

- 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、副照会の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。オプティマイザーの側で、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。

- 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。オプティマイザーの側で、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。
  – GROUP BY 演算子の節には文字式が含まれているか? 含まれている場合は、大文字小文字の区別がリモート・データ・ソースと DB2 で一致しているかどうかを調べてください。

- 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 節には文字式が含まれていますか? 含まれている場合は、リモート・データ・ソースと DB2 の照合シーケンスや大文字小文字の区別が同じかどうかを確認してください。
  – 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、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?

フェデレーテッド・データベースにおけるリモート SQL 生成とグローバル最適化

リレーショナル・ニックネームを使用するフェデレーテッド・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。このようにリモート SQL を生成することは、照会のためのグローバルで最適なアクセス戦略を作成するのに役立ちます。

オプティマイザーは、プッシュダウン分析の出力を使用して、各操作が DB2 でローカルに評価されるのか、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。これは、コスト・モデルの出力での決定に基づくものです。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・ソースの間でデータや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生成することですが、グローバルな最適化の出力は、大きく分けて次の 2 つの要素の影響を受けます。この影響は、照会のパフォーマンスにも及びます。

- サーバー特性
- ニックネーム特性

グローバルな最適化に影響を与えるサーバー特性とオプション

グローバルな最適化に影響を与える、データ・ソース・サーバーの要素には、次のようなものがあります。

- 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指定するには、 *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・ソースのコンピューターの CPU は、DB2 のコン

ピューターの CPU よりも速いということです。比率が低い場合、DB2 オプティマイザーは、データ・ソースに対してプッシュダウンによる CPU 集中の操作を実行しようとします。

- 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 オプティマイザーは、データ・ソースに対してプッシュダウンによる入出力集中の操作を考慮します。

- 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 オプティマイザーは、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、オプティマイザーは、必要なネットワーク通信量が最小になるアクセス・プランを作成します。

- データ・ソースの照合シーケンス

  データ・ソースの照合シーケンスが、ローカル DB2 の照合シーケンスと一致しているかどうかを指定するには、 *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、オプティマイザーは、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。

- リモート・プラン・ヒント

  プラン・ヒントがデータ・ソースで生成または使用されるかどうかを指定するには、 *plan_hints* サーバー・オプションを使用します。デフォルトでは、DB2 はプラン・ヒントを一切データ・ソースに送信しません。

  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・オプティマイザーについての追加情報を提供します。一部の照会では、この情報がパフォーマンスの向上に役立つ場合があります。プラン・ヒントは、データ・ソース・オプティマイザー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 オプティマイザー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 オプティマイザー・ナレッジ・ベースでの情報

  DB2 にはオプティマイザー・ナレッジ・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 オプティマイザーは、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・ソースでのオプティマイザーが理解できない、あるいは受け入れられないプランの生成を避けます。

グローバルな最適化に影響を与えるニックネーム特性

次のニックネーム固有の要因が、グローバルな最適化に影響を与える場合があります。

索引の考慮事項: 照会を最適化するには、データ・ソースにある索引に関する情報を DB2 で使用することができます。この理由から、DB2 で使用できる索引情報は、最新のものであることが重要です。ニックネームの索引情報は、ニックネームが作成されるときに最初に取得されます。ビュー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。

ニックネームでの索引の仕様の作成: ニックネームのための索引の仕様を作成できます。索引の仕様は、DB2 オプティマイザーが使用するカタログに索引定義 (実際の索引ではない) を構築します。索引の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。ニックネームの索引の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。

索引の仕様の作成は、次のような場合に考慮してください。

- 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取り出せない場合。
- ビューのニックネームのために索引が必要な場合。
- DB2 オプティマイザーで、 NESTED LOOP 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。

ビューのニックネームに対して CREATE INDEX ステートメントを発行するときは、まず、その必要があるかどうかを考慮してください。ビューが索引付きの表での単なる SELECT ステートメントである場合は、データ・ソースにある表の索引と一致するニックネームのローカル索引を作成すると、照会のパフォーマンスを大幅に改善できます。しかし、2 つの表を結合させて作成したビューのような、単なる SELECT ステートメントではないビューでローカル索引を作成すると、照会のパフォーマンスは低下する場合があります。たとえば、2 つの表を結合させたビューで索引を作成した場合、オプティマイザーは、そのビューを、NESTED LOOP 結合の内部エレメントとして選択する可能性があります。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・ソースのビュー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ローカル・ビューを DB2 で作成することができます。

カタログ統計の考慮事項: システム・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。オプティマイザーは、これらの統計を、ニックネームが含まれている照会の処理において最もコストが低いパスを計算するのに使用します。ニックネーム統計は、表統計と同じカタログ・ビューに格納されます。

DB2 では、データ・ソースに保管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して行われた更新を自動的に検出することはできません。さらに DB2 では、オブジェクト定義に加えられた変更

や、列の追加のような、データ・ソースのオブジェクトに対する構造上の変更は処理できません。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は、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します。次いで、現在のニックネームをドロップして、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 ビューの統計を手動で更新します。この方法は、実行するステップは少ないですが、構造情報が変更された場合には機能しません。

フェデレーテッド・データベース照会のグローバル分析

グローバル・アクセス・プランは、提供されている以下の 2 つのユーティリティーで示されます。

- Visual Explain。コントロール・センターから開始するか、コントロール・センターを始動させる *db2cc* コマンドを実行します。 Visual Explain は、照会アクセス・プランのグラフを表示するのに使用します。各演算子の実行位置は、演算子の詳細表示に示されます。照会のタイプによって、 SHIP または RETURN 演算子で各データ・ソースのために生成されたリモート SQL ステートメントを見出すこともできます。各演算子の詳細を調べるなら、 DB2 オプティマイザー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もりコストも確認できます。
- SQL Explain。このツールは、*db2expln* または *dynexpln* コマンドで開始します。 SQL Explain は、アクセス・プラン戦略をテキストとして表示するのに使用します。 SQL Explain では、コスト情報は示されませんが、リモート Explain 機能でサポートされているデータ・ソース用に、リモート・オプティマイザーで生成されるアクセス・プランを入手できます。

DB2 最適化の決定

パフォーマンスの向上のため、以下の最適化に関する問題とかぎとなる分野について考慮します。

- 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。
  - (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。

– 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 オプティマイザーは、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 オプティマイザーは、コスト・ベースの最適化を実行します。プッシュダウン分析で、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、オプティマイザーは、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。

- オプティマイザーによって生成され、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 オプティマイザー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。適切な照会オプティマイザーであれば、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 オプティマイザーが同じ動作をするわけではないので、リモート・データ・ソースのオプティマイザーによっては、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・オプティマイザー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。適切な照会オプティマイザーであれば、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 オプティマイザーが同じ動作をするわけではないので、リモート・データ・ソースのオプティマイザーによっては、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、オプティマイザーによっては、述部の推移的閉包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。

データ・アクセス方式

SQL または XQuery ステートメントをコンパイルするとき、照会オプティマイザーは照会を実現するさまざまな方法の実行コストを見積もります。見積もりに基づいて、オプティマイザーは最良のアクセス・プランを選択します。アクセス・プランとは、 SQL または XQuery 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 および XQuery 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 および XQuery ステートメントのアクセス・プランは、アプリケーション実行時に作成されます。

表のデータにアクセスするには 2 つの方法があります。

- 表全体を順番にスキャンする
- 最初に表の索引にアクセスすることにより、表の特定の行に位置付ける

照会が要求する結果を作り出すために、通常 WHERE 節で記述される述部の条件にしたがって行を選択します。アクセスされた表の選択された行は結合されて結果セットを作り出します。この結果セットをさらにソートしたりグループ化して処理する場合もあります。

索引スキャンによるデータ・アクセス

索引スキャン 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の理由で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スキャンすることによって) 条件限定の行の集合の範囲を狭めるため。索引の*スキャン範囲* (スキャンの開始点と停止点) 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。
- 出力を並べ替えるため。
- 要求した列データを直接取得するため。要求されたデータがすべて索引内にある場合、索引の対象である表にはアクセスする必要がありません。これは、*索引のみのアクセス* と呼ばれます。

索引が ALLOW REVERSE SCANS オプションで作成されていれば、スキャンは定義した方向とは逆方向に実行することもできます。

注: 適当な索引が作成されていない場合、または索引スキャンの方がコストがかかる場合、オプティマイザーは表スキャンを選択します。索引スキャンのコストが高くなり得るのは、表が小さい場合、索引クラスター率が低い場合、照会が表のほとんどの行を必要とする場合です。アクセス・プランが表スキャンと索引スキャンのどちらを使用するかを知るには、 Explain 機能を使用してください。

範囲を区切るための索引スキャン

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、オプティマイザーは索引の各列を最初の列から順に評価し、 WHERE 節の等式と他の述部を満足させるかどうかを調べます。*述部* は、WHERE 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 つのエレメントです。以下の場合に、述部は索引スキャンの範囲を区切るのに使用できます。

- 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ます。
- 「IS NULL」または「IS NOT NULL」であるかどうかがテストされます。
- 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。
- 狭義および広義の不等比較をテストします。

以下の例で索引が範囲を制限するのに使用される場合を説明します。

- 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
```

この場合、以下の述部を使用して索引 IX1 のスキャンの範囲を制限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 WHERE 節で、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。この例ではホスト変数を使用していますが、パラメーター・マーカー、式、または定数などの変数でも同じ効果があります。

- ALLOW REVERSE SCANS パラメーターを使用して作成された単一索引を考えてみましょう。こうした索引は、索引の作成時に定義した方向へのスキャンと、反対方向 (逆方向) へのスキャンをサポートします。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 の降順の (DESCending) 値に基づいて形成されます。逆方向スキャンを許可することにより、列に対する索引が降順でスキャンするように定義されているとしても、昇順でスキャンすることができます。実際に索引を両方向に使用する時には、それはユーザーによって制御されるのではなく、アクセス・プランを作成および考慮する時のオプティマイザーにより制御されます。

以下の WHERE 節では、NAME および DEPT の述部だけが索引スキャン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。

不等比較をテストする索引スキャン

ある種の不等比較述部は索引スキャンの範囲を区切ることができます。不等比較述部には、以下の 2 つのタイプがあります。

- 狭義の不等比較述部

  範囲を区切る述部として使用できる狭義の不等比較演算子は、より大きい (>) とより小さい (<) です。

  索引スキャンの範囲を区切るのに考慮される狭義の不等比較述部は 1 つの列に対するものだけです。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
  ```

- 広義の不等比較述部

  以下は、範囲を区切る述部として使用できる広義の不等比較演算子です。

  – >= と <=

  – BETWEEN

  – LIKE

  索引スキャンの範囲を区切る場合、広義不等比較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スキャン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 のすべての索引をスキャンしますが、 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 のスキャンを停止します。

データ並び替えのための索引スキャン

照会がソートされた順序での出力を要求している場合、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。並び替えまたはソートは、 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。例外は、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。

次の照会を考えてみましょう。

```
WHERE NAME = 'JONES'
  AND DEPT = 'D93'
ORDER BY MGR
```

この照会では、NAME も DEPT も必ず同じ値となり、結果としてその列についてはソート済みになるため、行を並び替えるのに索引を使用できます。つまり前述の WHERE 節と ORDER BY 節は次のものと同じです。

```
WHERE NAME = 'JONES'
  AND DEPT = 'D93'
ORDER BY NAME, DEPT, MGR
```

このほかにもユニーク索引を使用することによって、ソート対象条件を省略することもできます。次の索引定義と ORDER BY 節を考えてみましょう。

```
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 列での順序付けは不要です。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。

索引アクセスのタイプ

照会が表から必要としているデータがすべて表の索引から取得できると、オプティマイザーが判断できる場合があります。他方、表にアクセスするのに複数の索引を使用する場合もあります。レンジ・クラスター表の場合は、データ・レコードの位置を計算する「仮想」索引を介してデータにアクセスできます。

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'
```

しかしながら、しばしば必要な列が索引内にないことがあります。それらの列のデータを得るには、表の行を読み取らなければなりません。オプティマイザーが索引のみのアクセスを選べるようにするには、組み込み列付きでユニーク索引を作成します。たとえば、次の索引定義を考えてみてください。

```
CREATE UNIQUE INDEX IX1 ON EMPLOYEE
   (NAME ASC)
    INCLUDE (DEPT, MGR, SALARY, YEARS)
```

この索引は NAME 列をユニーク索引とし、しかも DEPT、 MGR、SALARY、および YEARS 列のデータも保管して維持します。これにより、次の照会は索引のみのアクセスで済んでしまいます。

```
SELECT NAME, DEPT, MGR, SALARY
   FROM EMPLOYEE
   WHERE NAME='SMITH'
```

しかし、索引に INCLUDE 列を追加することを考慮する場合、余分の記憶域スペースと保守コストに見合うメリットがあるかどうかを考えてください。こうした索引を読み取るだけで済む照会がめったに実行されないのなら、コストに見合うメリットがあるとは言えないかもしれません。

複数の索引アクセス

WHERE 節の述部を実行するために、オプティマイザーが同じ表の複数の索引をスキャンすることを選択する場合があります。たとえば、以下の 2 つの索引定義を考えてみてください。

```
INDEX IX2:  DEPT    ASC
INDEX IX3:  JOB     ASC,
            YEARS   ASC
```

次の述部は上記の 2 つの索引を使用することにより実行できます。

```
WHERE DEPT = :hv1
  OR (JOB   = :hv2
  AND YEARS >= :hv3)
```

索引 IX2 をスキャンすることによって、 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 をスキャンすることによって、 JOB = :hv2 AND YEARS >= :hv3 述部を満たす RID のリストが作成されます。表にアクセスする前に、これらの 2 つの RID リストは組み合わせられて重複は除かれます。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 節で指定されている述部でも使用される場合があります。

```
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 ANDing 結合* の目的は、共通 RID を検索することです。索引 ANDing 結合は、同じ表内の対応する列に複数の索引を作成し、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スキャンを行うと、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 をスキャンすると、 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 のスキャンおよび 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。

複数索引のスキャンを使った動的ビットマップのパフォーマンスを実現するためには、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、並行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、オプティマイザー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: オプティマイザーは、単一の表にアクセスするのに、索引 ANDing 操作と索引 OR 操作を組み合わせることはしません。

レンジ・クラスター表での索引アクセス

標準的な表とは異なり、従来の B ツリー索引のような行にキー値をマップする物理索引は、レンジ・クラスター表には不要です。その代わりに、列ドメインが持つ順次性を利用し、表内の行の位置に従い機能的にマッピングをします。このマッピングの単純な例では、範囲の最初のキー値は表の最初の行となり、範囲の 2 番目の値は表の 2 番目の行となります。

オプティマイザーは表のレンジ・クラスター・プロパティーを使用し、完全にクラスター化された索引に基づいてアクセス・プランを生成します。その際のコストはレンジ・クラスター関数の計算だけです。レンジ・クラスター表には元のキー値配列が保持されるので、表内の行のクラスター化は保証されます。

索引アクセスとクラスター率

アクセス・プランを選択するとき、オプティマイザーはディスクから必要なページをバッファー・プールに取り出すの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。

索引スキャンの場合、オプティマイザーは、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 表の以下の列の情報を使用します。

- CLUSTERRATIO 情報は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。数が大きいほど、行は索引キーの順序に並んでいます。表の行がほとんど索引キーの順序に並んでいれば、いくつもの行を 1 つのデータ・ページがバッファーにある内にそのページから読み取ることができます。この列の値が -1 の場合、オプティマイザーは、使用可能なら PAGE_FETCH_PAIRS および CLUSTERFACTOR 情報を使用します。
- PAGE_FETCH_PAIRS には CLUSTERFACTOR 情報と共に、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。これらの列のデータは、索引に対して RUNSTATS を DETAILED 節付きで実行した場合だけ集められます。

索引のクラスタリング統計が使用不可である場合、オプティマイザーは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リング率が低いことを想定しています。

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と 2 つの索引のキー列間に事実上の相関がある場合を除いて、 100% クラスタリングできるのは 1 つの索引だけです。

表を再編成するときに、行をクラスター化し挿入処理中この特性を保持するのに使用する索引を指定することができます。更新および挿入を行うと索引に対する表のクラスター化が低下するため、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 DELETE 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少なくするには、ALTER TABLE で PCTFREE パラメーターを変更します。こうすると、追加の挿入データがあっても既存のデータとのクラスター化が維持されます。

結合

結合とは、情報の何らかの共通の領域に基づいて複数の表からの情報を組み合わせるプロセスのことです。 1 つの表の行は別の表の行と、対応する行が結合基準に合致する場合に、組にされます。

たとえば、次の 2 つの表を考えてみてください。

Table1

PROJ	PROJ_ID
A	1
B	2
C	3
D	4

Table2

PROJ_ID	名前
1	Sam
3	Joe
4	Mary
1	Sue
2	Mike

表の ID 列が同じ値である場合に 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
```

この照会で次の結果行のセットが生成されます。

PROJ	PROJ_ID	名前
A	1	Sam
A	1	Sue
B	2	Mike
C	3	Joe
D	4	Mary

結合述部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各種コストに応じて、オプティマイザーは以下の結合方式どれかを選択します。

- ネスト・ループ結合
- マージ結合
- ハッシュ結合

2 つの表を結合する場合、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 回だけスキャンされます。内部表が複数回スキャンされるかどうかは、結合の種類および存在する索引によります。照会によって 3 つ以上の表が結合されるとしても、オプティマイザーは 1 回に 2 つの表だけを結合します。必要なら中間結果を保持するために一時表が作成されます。

INNER または LEFT OUTER JOIN のような明示的な結合演算子を指定して、結合で表をどう使用するかを決定することができます。しかしながら、この方法で照会を変更する前に、オプティマイザーにどう表を結合するか決定させてみるべきです。それから、照会のパフォーマンスを分析して結合演算子を追加するかどうか決めてください。

結合方式

照会が表の結合を要求する場合、オプティマイザーは以下の 3 つの基本結合方式の 1 つを選択することができます。

- ネスト・ループ結合
- マージ結合
- ハッシュ結合

これらの方式を以下のセクションで説明します。

ネスト・ループ結合

ネスト・ループ結合は、以下の 2 つの方法のいずれかで実行されます。

- 外部表のアクセスされる各行ごとに、内部表をスキャンする

  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を考えてみてください。

外部表 T1: 列 A	内部表 T2: 列 A
2	3
3	2
3	2
	3
	1

ネスト・ループ結合を実行するのに、データベース・マネージャーは以下のステップを実行します。

1. T1 から最初の行を読みます。 A の値は「2」です。
2. 一致するもの (「2」) が見つかるまで T2 をスキャンしてから、その 2 つの行を結合します。
3. 次に一致するもの (「2」) が見つかるまで T2 をスキャンしてから、その 2 つの行を結合します。
4. 表の終わりまで T2 をスキャンします。
5. T1 に戻って、次の行 (「3」) を読みます。
6. T2 を最初の行から始めて一致するもの (「3」) が見つかるまでスキャンし、その 2 つの行を結合します。
7. 次に一致するもの (「3」) が見つかるまで T2 をスキャンしてから、その 2 つの行を結合します。
8. 表の終わりまで T2 をスキャンします。
9. T1 に戻って、次の行 (「3」) を読みます。
10. 前と同じように T2 をスキャンし、一致する行 (「3」) をすべて結合します。

- 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

  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。次の例を考慮してください。

  ```
  OUTER.C1 + OUTER.C2 <= INNER.C1
  OUTER.C4 < INNER.C3
  ```

  この方法を使用すると、結合述部の選択可能性などいくつかの要因に依存するものの、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せる場合があります。

ネスト・ループ結合を評価するとき、オプティマイザーは、結合を実行する前に外部表をソート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る場合があります。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し、なおかつ外部表がソートされている場合、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、オプティマイザーは、結合後にソートを行うとコストが高くなると予期した場合に、結合前にソート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、結合後のソートが必要になる場合があります。

マージ結合

スキャン・マージ結合 またはソート・マージ結合 ということもあるマージ結合には、 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、またはソートによって、結合する列での入力の並べ替えが必要になります。結合列が LONG フィールド列やラージ・オブジェクト (LOB) 列である場合には、マージ結合は使用できません。

マージ結合では結合される表は同時にスキャンされます。マージ結合の外部表は 1 回だけスキャンされます。外部表に繰り返される値がなければ、内部表も 1 回だけスキャンされます。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スキャン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
2
3
3

内部表 T2: 列 A
1
2
2
3
3

マージ結合を実行するのに、データベース・マネージャーは以下のステップを実行します。

1. T1 から最初の行を読みます。 A の値は「2」です。
2. 一致するものが見つかるまで T2 をスキャンしてから、その 2 つの行を結合します。
3. 列が一致する間は T2 のスキャンを続け、行を結合します。
4. T2 内で「3」を読んだら、T1 に戻って次の行を読みます。
5. 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。
6. 列が一致する間は T2 のスキャンを続け、行を結合します。
7. T2 の終わりに達します。
8. 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 のスキャンが開始されます。データベース・マネージャーはこの位置を覚えておきます。

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。タイプが DECFLOAT の列の場合は、精度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。

まず指定された INNER 表をスキャンし、 *sortheap* データベース構成パラメーターで指定されるソート・ヒープから引き出されるメモリー・バッファーに行をコピーします。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値に基づくセクションに分割されています。 INNER 表のサイズが使用可能なソート・ヒープのスペースより大きい場合は、選択したセクションから一時表にバッファーが書き込まれます。

内部表が処理されたら、2 番目の表、つまり OUTER 表がスキャンされ、まず結合述部の列で計算されたハッシュ値を比較することにより、 INNER 表からの行と突き合わされます。 OUTER 行の列のハッシュ値が INNER 行の列のハッシュ値と一致したら、実際の結合述部列の値が比較されます。

一時表に書き込まれていない表の部分に対応した OUTER 表の行は、メモリー内の INNER 表の行と直ちに突き合わされます。対応する INNER 表の部分が一時表に書き込まれている場合は、 OUTER 行も一時表に書き込まれます。最後に、一致している表の部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値が突き合わされ、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十分に得るには、 *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るかもしれません。

ハッシュ結合のパフォーマンス上の効果を十分に得るには、 *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るかもしれません。

ハッシュ・ループとディスクへのオーバーフローを避けることができれば、ハッシュ結合のパフォーマンスが最高になります。ハッシュ結合のパフォーマンスを調整するには、*sheapthres* に使用可能なメモリーの最大量を見積もり、それから **sortheap** パラメーターを調整してください。可能な限りハッシュ・ループとディスク・オーバーフローを避けられるところまで設定値を大きくしてください。ただし *sheapthres* パラメーターで指定した制限に達しないようにします。

sortheap 値を増やすことも複数ソートのある照会のパフォーマンスを改善します。

最適の結合を選択する方法

照会のために最良の結合方法を選択するのに、オプティマイザーはさまざまな方式を使用します。こうした方式の中に、照会の最適化クラスによって決定される以下の検索方法があります。

- 最長一致の結合列挙
  - スペースおよび時間の観点から有効です。
  - 単一方向列挙です。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、最長一致の結合列

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、明示的に指定したものであれ述部の推移的閉包で暗黙的に生成されたものであれ、多数の結合述部がある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、スペースおよび時間の所要量も幾何級数的に増えます。
  - 最適なアクセス・プランのための効率的かつ完全な探索です。
  - DB2 for OS/390 and z/OS が使用する方法と似ています。

結合列挙アルゴリズムは、オプティマイザーが探索するプランの組み合わせの数の重要な決定要素です。

スター・スキーマ結合

照会で参照される表はほとんど常に結合述部によって接続されています。結合述部を使わないで 2 つの表が結合すると、 2 つの表のデカルト積が形成されます。デカルト積では、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。そのようなプランは効率よく実行するとは考えられないため、オプティマイザー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。

例外は、最適化クラスが 9 に設定されているか、特別な種類のスター・スキーマである場合だけです。スター・スキーマ には、ファクト表と呼ばれる中心的な表とディメンション表と呼ばれるほかのいくつかの表があります。照会に関係なく、ディメンション表にはすべてファクト表にアタッチする単一の結合だけがあります。それぞれのディメンション表には、ファクト表の特定の列についての情報を展開する追加の値が入っています。一般的な照会はディメンション表の値を参照する複数のローカル述部で成っており、ディメンション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、複数の小さいディメンション表のデカルト積を計算してから、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 は、少なくとも 2 つのディメンション表を持つスター・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したり、ディメンション表のデカルト積の計算を含む潜在的なプランを入れる検索スペースを大きくすることができます。デカルト積を計算するプランが見積もりで最低コストである場合は、そのプランがオプティマイザーによって選択されます。

上記で説明したスター・スキーマ結合方式は、主キーの索引を結合に使用すると想定しています。それとは別に、外部キー索引に関係するシナリオもあります。ファクト表の外部キー列が単一列索引で、全ディメンション表をまたがって比較的高い選択可能性がある場合には、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. 各ディメンション表を次の方法で処理します。
   - ディメンション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. ビットマップごとに直前のビットマップに対して AND 述部を使用します。

3. 最後のビットマップを処理した後に、残す RID を判別します。
4. それらの RID をソートする (オプション)。
5. 基本表の行を取り出す。
6. SELECT 節で必要とされるディメンション表の列にアクセスして、ファクト表とそれらの各ディメンション表とを再結合します。
7. Residual 述部を再適用します。

この技法は複数列索引を必要としません。この技法を選択するために、ファクト表とディメンション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、実際にファクト表とディメンション表の間のリレーションシップはそうした関係である必要があります。

スター型結合技法が作成し使用する動的ビットマップはソート・ヒープ・メモリーを必要としますが、そのサイズはソート・ヒープ・サイズ (*sortheap*) データベース構成パラメーターで指定します。

初期外部結合

オプティマイザーは、表の 1 つからの各行が、他の表からの最大でも 1 行と結合する必要しかないことを検出する場合、初期外部結合を選択できます。

初期外部結合は、表の 1 つのキー列に結合述部がある場合に可能です。例えば、従業員とその直接の上司の名前を戻す以下の照会を考慮してください。

```
SELECT EMPLOYEE.NAME AS EMPLOYEE_NAME, MANAGER.NAME AS MANAGER_NAME
FROM EMPLOYEE AS EMPLOYEE, EMPLOYEE AS MANAGER
WHERE EMPLOYEE.MANAGER_ID=MANAGER.ID
```

ID 列が EMPLOYEE 表内のキーであり、すべての従業員に少なくとも 1 人の上司がいると想定すると、上記の結合により MANAGER 表内の後続の一致行の検索を回避できます。

初期外部結合は、照会内に DISTINCT がある場合も可能です。例えば、$30000 を超える価格のモデルを持つ自動車メーカーの名前を戻す以下の照会を考慮してください。

```
SELECT DISTINCT MAKE.NAME
FROM MAKE, MODEL
WHERE MAKE.MAKE_ID=MODEL.MAKE_ID AND MODEL.PRICE>30000
```

自動車メーカーごとに、その製造モデルのいずれかの価格が $30000 を超えるかどうかを判別する必要があるだけです。自動車メーカーと、価格が $30000 を超えるすべての製造モデルとの結合は、照会結果の正確さのためには不要です。

初期外部結合も、結合が GROUP BY に MIN または MAX 集約関数をフィードする場合は可能です。例えば、以下の照会について考えてみましょう。これは、2000 年より前に特定の株の終値がその始値よりも 10% 以上高くなった銘柄記号と、そのような日付のうち最も後のものとを戻します。

```
SELECT DAILYSTOCKDATA.SYMBOL, MAX(DAILYSTOCKDATA.DATE) AS DATE
FROM SP500, DAILYSTOCKDATA
WHERE SP500.SYMBOL=DAILYSTOCKDATA.SYMBOL AND DAILYSTOCKDATA.DATE<'01/01/2000' AND
      DAILYSTOCKDATA.CLOSE / DAILYSTOCKDATA.OPEN >= 1.1
GROUP BY DAILYSTOCKDATA.SYMBOL
```

「修飾セット」を、日付と株価の要件を満たす DAILYSTOCKDATA 表からの行のセットとして定義し、SP500 表からの特定の銘柄記号と結合させます。 SP500 表からの各銘柄記号行について、DAILYSTOCKDATA 表からの修飾セットが DATE に基づいて降順で配列されている場合、最初の行がその特定の記号の最新の日付であるので、必要なのは各記号の修飾セットからの最初の行を戻すことだけです。修飾セット内の他の行は、照会の正確さには必要ありません。

複合表

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が、通常この表は別の内部表との別の結合の外部表になります。これを「複合外部」結合と言います。ある場合、特に最長一致の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後の結合の内部表にすると役に立ちます。ある結合の内部表が 2 つ以上の表を結合した結果で成っていると、そのプランは「複合内部」結合です。たとえば次の照会を考えてみましょう。

```
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 (T3xT4)、それから、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つ場合があります。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、オプティマイザーは結合の内部表となる表の最大数に異なる制約を課します。複合内部結合は最適化クラス 5、7、および 9 で使用できます。

パーティション・データベース環境の複製されたマテリアライズ照会表

複製マテリアライズ照会表は、データベースが表データの事前計算された値を管理できるようにすることによって、パーティション・データベース環境で頻繁に実行される結合のパフォーマンスを改善します。

照会および複製マテリアライズ表の例を考えてみてください。次のような前提事項があります。

- SALES 表は、複数パーティション表スペース REGIONTABLESPACE にあり、REGION 列で分割されています。
- EMPLOYEE 表および DEPARTMENT 表が単一パーティション・データベース・パーティション・グループにあります。

EMPLOYEE 表の情報に基づき、複製マテリアライズ照会表を作成します。

```
CREATE TABLE R_EMPLOYEE
  AS (
      SELECT EMPNO, FIRSTNME, MIDINIT, LASTNAME, WORKDEPT
      FROM   EMPLOYEE
     )
DATA INITIALLY DEFERRED REFRESH IMMEDIATE
IN REGIONTABLESPACE
REPLICATED;
```

複製マテリアライズ照会表の内容を更新するには、次のステートメントを実行します。

```
REFRESH TABLE R_EMPLOYEE;
```

注: 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。

次の例は、従業員ごとの売上、部署の合計、総合計を計算します。

```
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・パーティションにしか存在しない 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 表が保管されている各データベース・パーティションで複製される 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・パーティション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。

コロケーテッド結合における複製マテリアライズ照会表

複製マテリアライズ照会表は結合のコロケーションでも助けになります。たとえば、スター・スキーマに 20 のノードにまたがる大規模なファクト表がある場合、ファクト表とディメンション表の結合はこれらの表が連結されていると最も効率的です。同一のデータベース・パーティション・グループにすべての表があれば、多い場合でも 1 つのディメンション表がコロケーテッド結合のために正しくパーティション化されます。他のディメンション表はすべてコロケーテッド結合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分散キーと対応していないためです。

C1 で分割された FACT (C1、C2、C3、...) という表、C1 で分割された DIM1 (C1、dim1a、dim1b、...) という表、C2 で分割された DIM2 (C2、dim2a、dim2b、...) という表などがある場合を考えてみてください。

この場合、述部 DIM1.C1 = FACT.C1 は連結できるので、 FACT と DIM1 の間の結合は完全であることが分かります。これらの表は両方とも C1 列で分割されています。

しかし述部 WHERE DIM2.C2 = FACT.C2 による DIM2 の結合は連結できません。これは FACT が C1 列で分割されており、C2 列ではないからです。この場合、DIM2 をファクト表のデータベース・パーティション・グループに複製して、データベース・パーティションごとにローカルに結合するようにできます。

注: この複製マテリアライズ照会表についての説明はデータベース内複製と関連しています。データベース間複製はサブスクリプション、コントロール表、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。

複製マテリアライズ照会表を作成する場合、ソース表はデータベース・パーティション・グループの単一ノード表でもマルチノード表でも構いません。ほとんどの場

合、複製される表は小さく、単一ノード・データベース・パーティション・グループ内に配置することができます。表からの列のサブセットだけを指定することにより、または述部を使用して行数を指定することにより、さらには両方の方式を使用することにより、複製されるデータを制限することができます。複製マテリアライズ照会表を機能させるにはデータ・キャプチャー・オプションは必要ありません。

ソース表のコピーをすべてのデータベース・パーティションに作成するため、マルチノード・データベース・パーティション・グループに複製マテリアライズ照会表を作成することもできます。すべてのデータベース・パーティションに対しソース表をブロードキャストするよりも、大規模なファクト表とディメンション表間の結合は、この環境内でローカルで行う方が良いです。

複製された表の索引は、自動的に作成されません。ソース表のものとは異なる索引を作成することができます。しかし、ソース表になかった制約違反を防ぐため、ユニーク索引の作成や複製された表に制約を加えることはできません。制約はソース表に同じ制約があっても許可されません。

複製された表は照会内で直接参照できますが、特定のパーティション上の表データを見るために複製された表で NODENUMBER() 述部を使用することはできません。

EXPLAIN 機能を使用して、複製マテリアライズ照会表が照会のためのアクセス・プランで使用されたかどうかを調べてください。オプティマイザーが選択するアクセス・プランが複製マテリアライズ照会表を使用するかどうかは、結合される必要のある情報に依存します。オプティマイザーがオリジナル・ソース表をデータベース・パーティション・グループの他のデータベース・パーティションにブロードキャストするほうがコストがかからないと判断した場合、オプティマイザーが複製マテリアライズ照会表を使用しない場合もあります。

パーティション・データベースでの結合ストラテジー

いくつかの面でパーティション・データベース環境の結合のための戦略はパーティションのないデータベース環境と違います。パフォーマンスを改善するために、標準の結合方式に加えて別の技法を適用することができます。

パーティション・データベース環境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、表コロケーションがあります。表コロケーションは、パーティション・データベース環境において、ある表のデータを、同じ分散キーに基づいて、同じデータベース・パーティション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。コロケーションが行われると、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・パーティションに移動せずに処理されます。結合の応答セットのみがコーディネーター・ノードに移動されます。

表キュー

パーティション・データベース環境での結合技法の説明は以下の用語を使用します。

- 表キュー

  データベース・パーティション間 (または、単一パーティション・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。
- 指示表キュー

行が受信データベース・パーティションの 1 つにハッシュされる表キュー。

- ブロードキャスト表キュー

  行がすべての受信データベース・パーティションに送信されるが、ハッシュは行われない表キュー。

表キューは、以下の環境で使用されます。

- パーティション間並列処理の使用時に、あるデータベース・パーティションの表データを別のデータベース・パーティションに渡す
- パーティション内並列処理の使用時に、1 つのデータベース・パーティション内で表データを渡す
- 単一パーティション・データベースの使用時に、1 つのデータベース・パーティション内で表データを渡す

各表キューは単一方向にデータを渡します。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・パーティション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 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・パーティションでは、カーソルが、そのデータベース・パーティション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 節が指定されていない場合に使用されます。パーティション・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。

パーティション・データベース環境での結合方式

以下の図はパーティション・データベース環境における結合方式を示します。

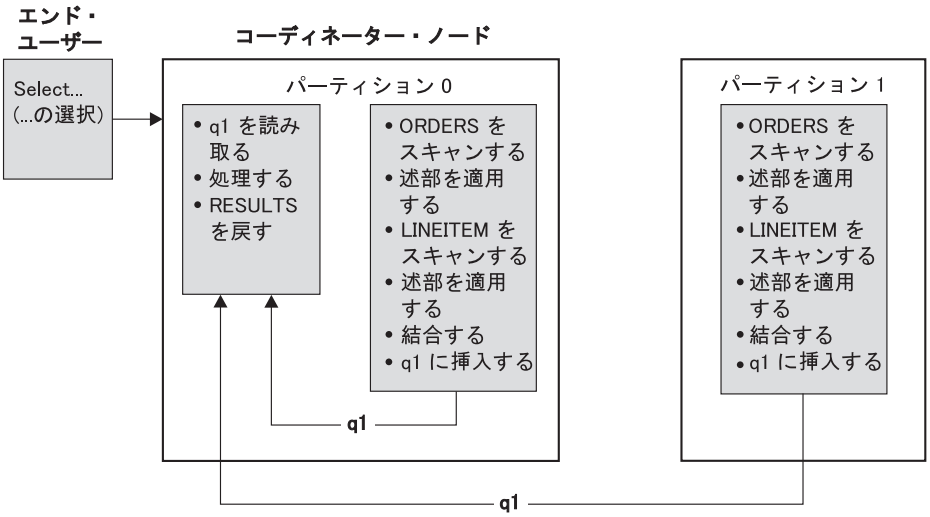
注: 図中の q1、q2、q3 は、例に出てくる表キューと対応しています。図に示されている表は、これらのシナリオの目的に合わせて、 2 つのデータベース・パーティションにまたがって分割されています。矢印は、表キューが送られる方向を示します。なお、コーディネーター・ノードはデータベース・パーティション 0 です。

コロケーテッド結合

コロケーテッド結合はデータがあるデータベース・パーティションでローカルに発生します。結合の完成後、そのデータベース・パーティションはデータを他のデータベース・パーティションに送信します。オプティマイザーがコロケーテッド結合を処理するためには、結合される表は連結され、対応する分散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。

次の図に例を示します。

注: 複製マテリアライズ照会表はコロケーテッド結合の可能性を高めます。



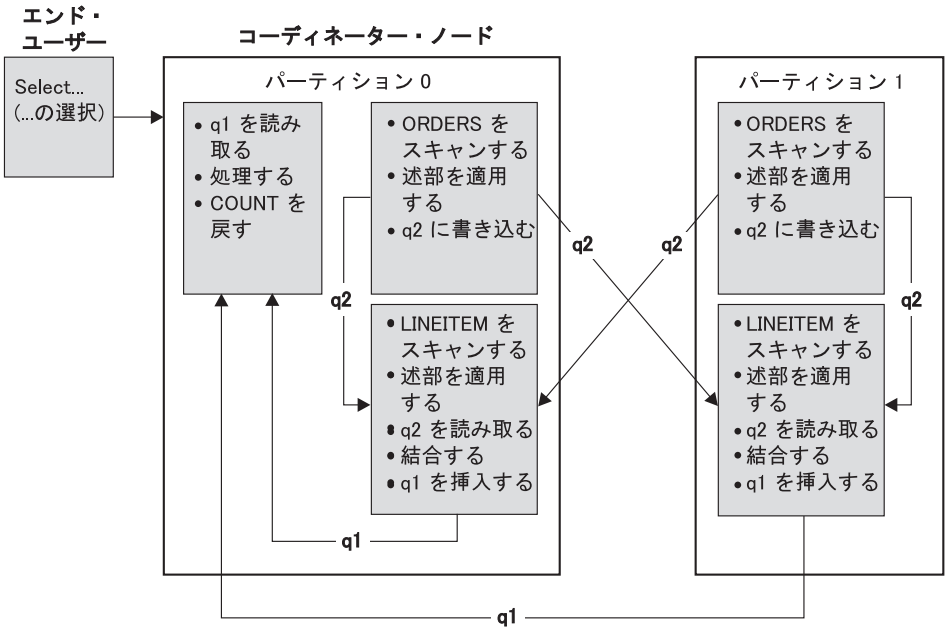
LINEITEM 表と ORDERS 表は ORDERKEY 列上でパーティション化される。
結合は、各データベース・パーティションでローカルに行われる。
この例では、結合述部は次のように想定されている。
ORDERS.ORDERKEY = LINEITEM.ORDERKEY.

図 23. コロケーテッド結合の例

外部表のブロードキャスト結合

外部表のブロードキャスト結合は、結合される表の間に等価結合述部がない場合に使用できる並列結合方式です。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。たとえば、外部表のブロードキャスト結合は、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分割されていない場合に使用されます。両方の表をパーティションに分割するよりも、小さな表を大きな表にブロードキャストするほうがコストがかからな

い可能性があります。次の図に例を示します。

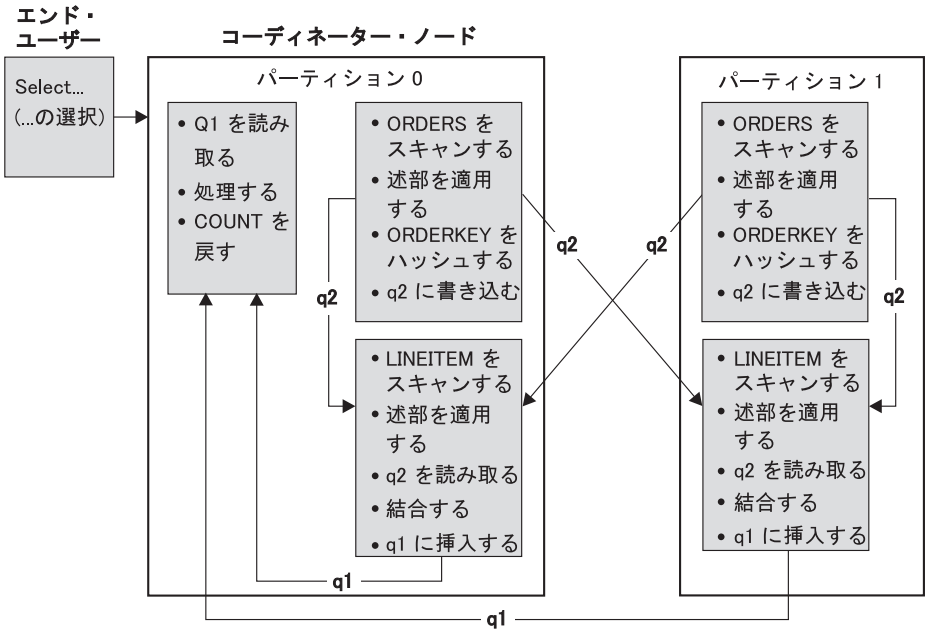


ORDERS 表は、LINEITEM 表を持つデータベース・パーティションすべてに送られる。表キュー q2 は、内部表のデータベース・パーティションすべてにブロードキャストされる。

図 24. 外部表のブロードキャスト結合の例

外部表の指示結合

外部表の指示結合方式では、外部表の各行を内部表の分割属性に基づいて内部表の一部分に送ります。結合は、このデータベース・パーティション上で行われます。次の図に例を示します。

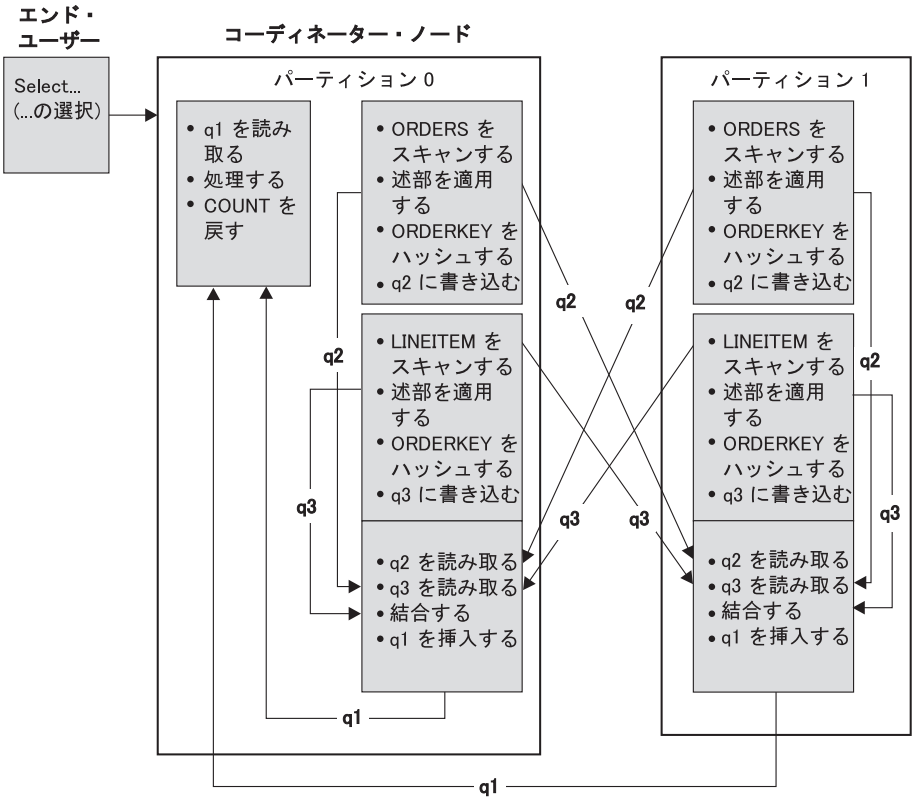


LINEITEM 表は ORDERKEY 列上でパーティション化される。
ORDERS 表は別の列でパーティション化される。
ORDERS 表がハッシュされて、適切な LINEITEM 表のデータベース・パーティションに送られる。
この例では、結合述部は次のように想定されている。
ORDERS.ORDERKEY = LINEITEM.ORDERKEY.

図 25. 外部表の指示結合の例

内部表および外部表の指示結合

内部表および外部表の指示結合方式では、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・パーティションのセットに送られます。結合は、これらのデータベース・パーティション上で行われます。次の図に例を示します。例は次の図で示します。

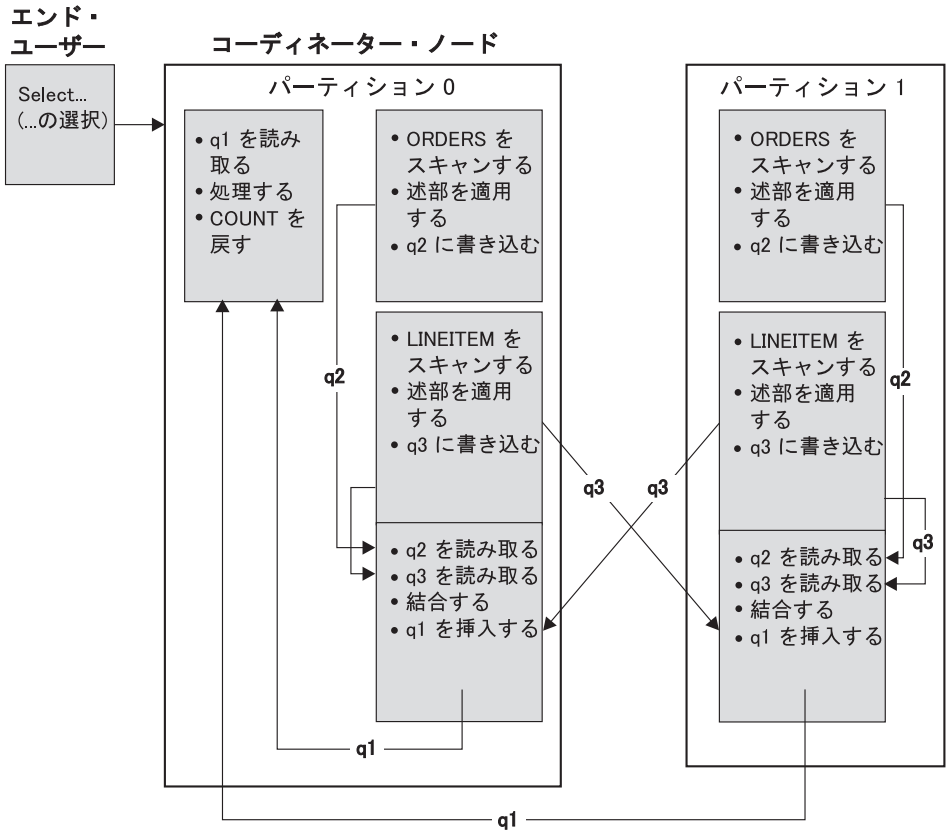


いずれの表も、ORDERKEY 列上ではパーティション化されない。
どちらの表もハッシュされ、新しいデータベースに送られて、そのパーティションで結合される。
両方の表キュー q2 と q3 が送られる。
この例では、結合述部は次のように想定されている。
ORDERS.ORDERKEY = LINEITEM.ORDERKEY

図 26. 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

内部表のブロードキャスト結合方式では、内部表が外部結合表のすべてのデータベース・パーティションに対してブロードキャストされます。次の図に例を示します。

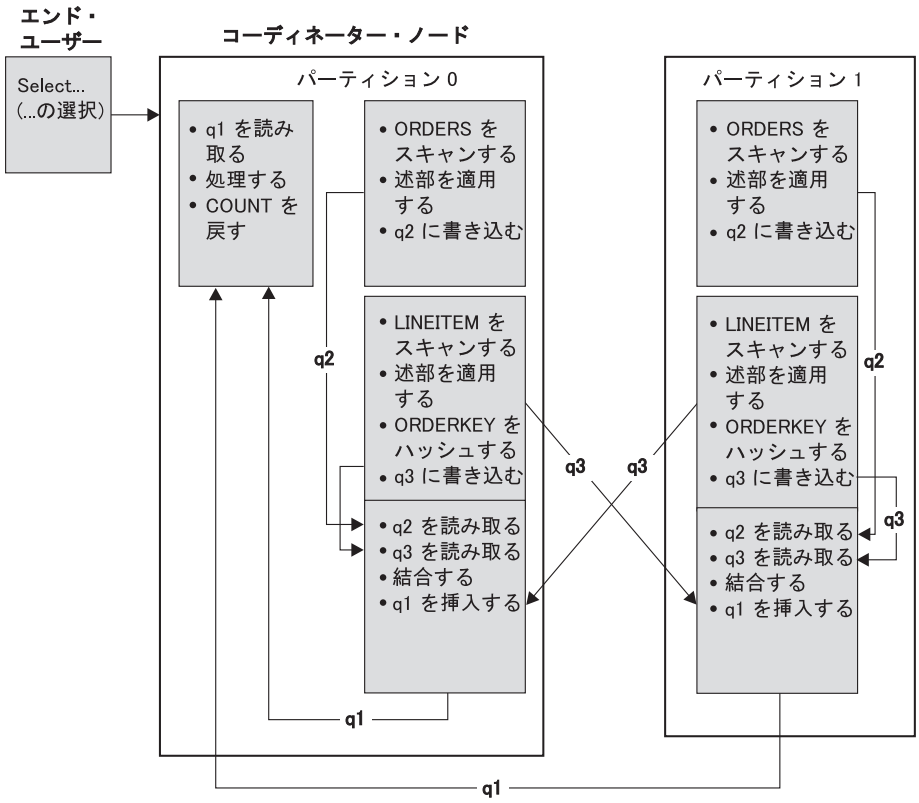


ORDERS 表は、LINEITEM 表を持つデータベース・パーティションすべてに送られる。表キュー q3 は、外部表のデータベース・パーティションすべてにブロードキャストされる。

図 27. 内部表のブロードキャスト結合の例

内部表の指示結合

内部表の指示結合方式では、内部表の各行を、外部表の分割属性に基づいて外部結合表のデータベース・パーティションの 1 つに送ります。結合は、このデータベース・パーティション上で行われます。次の図に例を示します。



ORDERS 表は ORDERKEY 列上でパーティション化される。
LINEITEM 表は別の列でパーティション化される。
LINEITEM 表がハッシュされて、適切な ORDERS 表のデータベース・パーティションに送られる。
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 28. 内部表の指示結合の例

ソートとグループ化の影響

オプティマイザーは、アクセス・プランを選択する際に、データのソートによるパフォーマンスの影響を考慮します。ソートは、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。ソートはオプティマイザーが索引スキャンよりソートの方がコストがかからないと判断した場合にも行われる場合があります。オプティマイザーは以下のいずれかの方法でデータをソートします。

- 照会の実行時に、ソートの結果をパイプ処理します。
- データベース・マネージャー内でソートを内部処理します。

パイプ・ソートと非パイプ・ソート

データの最終的なソートされたリストが 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。パイピングはソート結果を渡すのに非パイプの方法よりも高速に行えます。オプティマイザーは可能ならば、ソート結果をパイプ処理することを選択します。

ソートがパイプ処理されるかどうかには関係なく、ソート時間は、ソートする行の数、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。

ソートされる行が、ソート・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、複数のソート・パスが実行され、各パスごとに行全体のうちの 1 つのサブセットがソートされることになります。各ソート・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・プールに十分のスペースがない場合、この一時表のページはディスクに書き込みます。すべてのソート・パスが完了したなら、それらのソート済みサブセットをマージして、ソート済みの単一の行集合にする必要があります。ソート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。

グループ化およびソート・プッシュダウン演算子

場合によっては、オプティマイザーは、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に対して、ソート操作または集約操作のプッシュダウンを選択することができます。これらの操作をプッシュダウンにすると、データ管理サービス・コンポーネントがデータをソート・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。このプッシュダウンを行わない場合、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、次いでソート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
```

ソートのグループ化操作

GROUP BY 操作で必要な順序をソートが生成する場合に、オプティマイザーは、ソートの実行中に GROUP BY の集約の一部または全部を実行することができます。これは、各グループにある行の数が多い場合は有利です。ソート中に行われる何らかのグループ化により、ソート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。

ソート中の集約には、正しい結果を戻すためには、集約の以下の最大 3 つのステージが必要です。

1. 最初の集約の段階である部分集約は、ソート・ヒープがいっぱいになるまで集約値を計算します。部分集約において、集約されていないデータが取り込まれて部分的な集約が作成されます。ソート・ヒープがいっぱいになったら、現在のソート・ヒープで計算された部分集約のすべてを含め、残りのデータをディスクに書き出します。ソート・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. 2 番目の集約の段階である中間集約は、書き出されたソート実行のすべてを取り込んで、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分散キー列のサブセットなので、集約はまだ完了しません。中間集約は既存の部分集約を使用して、新しい部分集約を作り出します。このステージは常に行われるわけではありません。これはパーティション内並列処理とパーティション間並列処理の両方で使用されます。パーティション内並列処理では、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。パーティション間並列処理では、グループ化キーが、複数のデータベース・パーティションにわたってグループを分けている分散キーのサブセットであって、集約を完了するために再分散が必要な場合に、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、各エージェントが書き出されたソート実行のマージを終了するときに、似たようなケースがパーティション内並列処理で存在します。

3. 最後の集約の段階である最終集約は、すべての部分集約を取り込んで最終集約を作り出します。このステップは、GROUP BY 演算子で常に生じます。ソートが分割されないという保証はないので、ソート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、最終集約を作ります。分散がその使用を禁止していない場合、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のに通常使用されます。

最適化ストラテジー

パーティション内並列処理の最適化ストラテジー

SQL ステートメントのコンパイル時に並列処理の多重度が指定された場合、オプティマイザーは、シングル・データベース・パーティション内で並列して照会を実行するアクセス・プランを選択します。

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指定された並列処理の多重度以下になります。

オプティマイザーは、アクセス・プランを並列化するため、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コーディネーター・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、データをコーディネーター・エージェントか他のサブエージェントに渡します。パーティション・データベース環境では、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・パーティション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。

パーティション内の並列スキャン方式

リレーショナル・スキャンおよび索引スキャンは、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・スキャンの場合、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スキャンし、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。

並列索引スキャンの場合には、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。並列索引スキャンは、並列表スキャンと同様に、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。

オプティマイザーは、スキャンの単位 (ページまたは行) と細分度を決定します。

並列スキャンは、サブエージェント間で均等になるように作業を分散します。並列スキャンの目標は、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・プラン方式によっては、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。オプティマイザーは、サブエージェント間でデータのバランスを維持できるように並列方式を選択します。

パーティション内の並列ソート方式

オプティマイザーは、以下のいずれかの並列ソート方式を選択します。

- **ラウンドロビン・ソート**

  このソートは、*再配分ソート* とも呼ばれます。このソート方式では、共用メモリーを効率的に使用し、すべてのサブエージェントに対して可能な限り均一にデータを再配分します。このソートは、ラウンドロビン・アルゴリズムを使用して、均等な分散を行います。まず最初に、各サブエージェントごとに個々のソートを作成します。挿入フェーズでは、サブエージェントが、ラウンドロビン様式で個々のソートにそれぞれデータを挿入していくことによって、より均等なデータの分散を行います。

- **パーティション・ソート**

  このソートは、ソート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・ソートに似た働きをします。このソートでは、サブエージェントはハッシュ関数をソート列に適用して、行をどのソートに挿入するかを判別します。たとえば、マージ結合の内部表と外部表がパーティション・ソートの場合、サブエージェントは、マージ結合を使用することによって、対応する表の部分を結合し並列で実行できます。

- **複製ソート**

  このソートは、各サブエージェントがすべてのソート出力を必要とする場合に使用されます。あるソートが作成されると、サブエージェントは、そのソートに行が挿入されるときに同期化されます。ソートが完了すると、各サブエージェントがソート全体の読み取りを行います。このソートは、行数が少ないとき、データ・ストリームのバランスをとり直すのに使用できます。

- **共有ソート**

  このソートは、複製ソートと同様の働きをしますが、共有ソートの場合は、ソートされた結果に対してサブエージェントが並列スキャンをオープンし、ラウンドロビン・ソートと同様の方法でサブエージェント間にデータが分配されます。

パーティション内並列一時表

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、一時表を生成できます。この表は、**共有一時表**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分割されるかに応じて、専用スキャンまたは並列スキャンのいずれかを共有一時表上でオープンします。

パーティション内の並列集約方式

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、以前のパーティション・ソートのためにグループ化列上のストリームがすでに分割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コーディネーター・エージェントに送る。コーディネーターにより集約が完全に行われます。
- 部分的に集約データをパーティション・ソートに挿入します。このソートは、グループ化列上で分割されるため、グループ化列の集合に関するすべての行が確実に 1 つのソート・パーティションに入れられます。
- 処理のバランスをとるためにストリームの複製が必要な場合は、部分的に集約データを複製ソートに挿入できます。個々のサブエージェントは複製ソート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。

パーティション内の並列結合方式

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、結合の内部表または外部表でデータ・ストリームをパーティションに分割または複製 (あるいは、その両方) することによって、並列化できます。たとえば、ネスト・ループ結合は、外部のストリームが並列スキャンのためにパーティション化され、さらに内部のストリームが各サブエージェントで別々に再評価されると並列化できます。マージ結合は、内部ストリームと外部ストリームがパーティション・ソートのために値でパーティション化されると並列化できます。

MDC 表の最適化ストラテジー

マルチディメンション・クラスタリング (MDC) 表を作成した場合、オプティマイザーは追加の最適化ストラテジーを適用できるので、多くの照会のパフォーマンスが向上する可能性があります。これらのストラテジーは主にブロック索引の効率が改善されたことに基づいていますが、複数ディメンションでのクラスタリングによる利点もデータ検索の高速化を可能にしています。

注: MDC 表の最適化ストラテジーは、パーティション内並列処理およびパーティション内並列処理によるパフォーマンス上の利点もインプリメントすることができます。

MDC 表による以下の特定の利点を検討してください。

- ディメンション・ブロック索引の参照数により、表の必要な部分を識別して必要なブロックだけを高速にスキャンすることができます。
- ブロック索引は RID 索引よりも小さいので、参照数はより高速になります。
- 索引の AND 操作および OR 操作をブロック・レベルで実行して、 RID と結合することができます。
- データはエクステント上でクラスター化されることが保証されているので、検索がより高速になります。

- ロールアウトを使用できるときに行の削除が高速になります。

SALES という名前で、ディメンションが **region** および **month** 列に定義されている MDC 表についての次の簡単な例を検討してください。

```
SELECT * FROM SALES
    WHERE MONTH='March' AND REGION='SE'
```

この照会では、オプティマイザーはディメンション・ブロック索引のルックアップを実行して、 month が March で region が SE のブロックを検索することができます。その後、その結果となる表のブロックだけを高速にスキャンして、結果セットを取り出すことができます。

ロールアウト削除

ロールアウトを使用する削除が許可される条件が満たされれば、MDC 表から行を削除するための、さらに効果的な方法が使用されます。条件は以下のとおりです。

- DELETE ステートメントが検索されたが、位置を特定できない (つまり、『WHERE CURRENT OF』 節を使用しない)。
- WHERE 節がない (すべての行が削除される) または WHERE 節の条件だけがディメンションにある。
- 表が DATA CAPTURE CHANGES 節で定義されていない。
- 表が参照整合性リレーションシップで親でない。
- 表に削除トリガーが定義されていない。
- 表が即時に更新される MQT で使用されない。
- カスケード削除操作がロールアウトできる (その外部キーがその表のディメンション列のサブセットである場合)。
- DELETE ステートメントは、(CREATE TRIGGER ステートメント上の OLD TABLE AS 節により指定される) トリガー SQL 操作の前に、影響を受ける一連の行を識別する一時表に対して実行される SELECT ステートメントには入れることができません。

ロールアウト削除の場合、削除レコードはログに記録されません。その代わり、レコードの入ったページはページのパーツを再フォーマットすることによって空にされます。再フォーマットされたパーツに対する変更はログに記録されますが、レコード自体はログに記録されません。

デフォルトの動作である*即時クリーンアップ・ロールアウト* は、削除時に RID 索引をクリーンアップするためのものです。このモードは、**DB2_MDC_ROLLOUT** レジストリー変数を IMMEDIATE に設定するか、または `IMMEDIATE` を SET CURRENT MDC ROLLOUT MODE ステートメントで指定することでも指定できます。標準の削除と比較して、索引の更新のロギングに変更がない場合、パフォーマンスの向上は RID 索引の数によって異なります。 RID 索引が少ないと、合計時間およびログ・スペースの割合が改善されます。

ログに保存されるスペース量の見積もりは、この公式で出すことができます。ここで、`N` は削除されるレコードの数であり、`S` は削除されるレコードの合計サイズ (NULL 標識および varchar の長さなどのオーバーヘッドを含む) であり、`P` は削除されるレコードの入ったブロックのページ数です。

```
S + 38*N - 50*P
```

この数値は、実際のログ・データを縮約したものです。アクティブ・ログ保存でのスペース所要量は、ロールバックに予約されているスペースの保存のため 2 倍になります。

代替方法として、*据え置きクリーンアップ・ロールアウト* を使用して、トランザクション・コミット後に RID 索引を更新できます。このモードは、**DB2_MDC_ROLL_OUT** レジストリー変数を DEFER に設定するか、または DEFERRED を SET CURRENT MDC ROLLOUT MODE ステートメントで指定することでも指定できます。据え置きロールアウトで、RID 索引は削除のコミット後に、バックグラウンドで非同期にクリーンアップされます。このロールアウトのメソッドは、非常に大規模な削除の場合、または表に多数の RID 索引が存在する場合は結果としてかなり高速な削除となります。即時索引クリーンアップでは索引内の各行は 1 つずつクリーンアップされるのに対し、据え置き索引クリーンアップ中に索引は並列でクリーンアップされるので、クリーンアップ操作全体の速度は向上します。さらに、非同期索引クリーンアップでは、索引キーではなく索引ページにより索引更新をログ記録するので、DELETE ステートメントのトランザクション・ログスペース所要量は大幅に削減されます。

注: 据え置きクリーンアップ・ロールアウトには、データベース・ヒープから取られる追加のメモリー・リソースが必要です。 DB2 が必要とするメモリー構造を割り振れない場合、据え置きクリーンアップ・ロールアウトは失敗し、メッセージが管理者ログに書き込まれます。

据え置きクリーンアップ・ロールアウトを使用する状況

削除のパフォーマンスが最も重要な要因であり、RID 索引が表に定義されている場合は、据え置きクリーンアップ・ロールアウトを使用します。索引クリーンアップの前に、ロールアウトされたブロックの索引ベースのスキャンは、ロールアウトされたデータの量に応じてパフォーマンスがやや低下します。即時索引クリーンアップと据え置き索引クリーンアップとを決定する場合に考慮すべき、以下のような他の問題があります。

- 削除のサイズ: 非常に大規模な削除に対しては、据え置きクリーンアップ・ロールアウトを選択します。ディメンション削除ステートメントが数多くの小さな MDC 表に対して発行される場合は、非同期に索引オブジェクトをクリーンアップするためのオーバーヘッドが、削除中の時間の節約という利点を上回ってしまう可能性があります。
- 索引の数およびタイプ: 表に行レベルの処理を必要とする数多くの RID 索引が含まれている場合は、据え置きクリーンアップ・ロールアウトを使用します。
- ブロック可用性: 削除ステートメントのコミット直後に、その削除ステートメントにより解放されるブロック・スペースを使用できるようにするには、即時クリーンアップ・ロールアウトを使用します。
- ログ・スペース: ログ・スペースが制限されている場合、大規模削除の据え置きクリーンアップ・ロールアウトを使用します。
- メモリー制約: 据え置きクリーンアップ・ロールアウトは、据え置きクリーンアップが保留中のすべての表の追加のデータベース・ヒープを消費します。

削除でロールアウト動作を無効にするには、**DB2_MDC_ROLLOUT** レジストリー変数を OFF に設定するか、または NONE を SET CURRENT MDC ROLLOUT MODE ステートメントで指定できます。

パーティション表の最適化ストラテジー

照会述部に基づき、照会に応答するためにアクセスする必要があるのは表のデータ・パーティションのサブセットのみであると判断するデータベース・サーバーの機能のことです。データ・パーティションの除去は、パーティション表に対して意思決定支援照会を実行する際に特に役立ちます。

パーティション表は、データ・パーティションまたは範囲と呼ばれる複数のストレージ・オブジェクトに表データを分割するというデータ編成スキームを使用します。分割は、表の 1 つ以上の表パーティション・キー列の値に従って行われます。指定された表のデータは、CREATE TABLE ステートメントの PARTITION BY 節で提供された仕様に基づいて、複数のストレージ・オブジェクトにパーティション化されます。このストレージ・オブジェクトは異なる表スペース、同じ表スペース内、またはその両方に配置することができます。

以下の例は、データ・パーティションの除去のパフォーマンス上の利点について示しています。次のステートメントを発行した場合について考慮してみます。

```
CREATE TABLE custlist(subsdate DATE, Province CHAR(2), AccountID INT)
PARTITION BY RANGE(subsdate)
(STARTING FROM '1/1/1990' IN ts1,
STARTING FROM '1/1/1991' IN ts1,
STARTING FROM '1/1/1992' IN ts1,
STARTING FROM '1/1/1993' IN ts2,
STARTING FROM '1/1/1994' IN ts2,
STARTING FROM '1/1/1995' IN ts2,
STARTING FROM '1/1/1996' IN ts3,
STARTING FROM '1/1/1997' IN ts3,
STARTING FROM '1/1/1998' IN ts3,
STARTING FROM '1/1/1999' IN ts4,
STARTING FROM '1/1/2000' IN ts4,
STARTING FROM '1/1/2001' ENDING '12/31/2001' IN ts4);
```

2000 年の顧客情報に関心があるとします。以下の照会を発行する場合について考えます。

```
SELECT * FROM custlist WHERE subsdate BETWEEN '1/1/2000' AND '12/31/2000';
```

345 ページの図 29 に示されているように、データベース・サーバーはこの照会を解決するために、表スペース 4 (ts4) の 1 つのデータ・パーティションのみにアクセスする必要があると判断します。

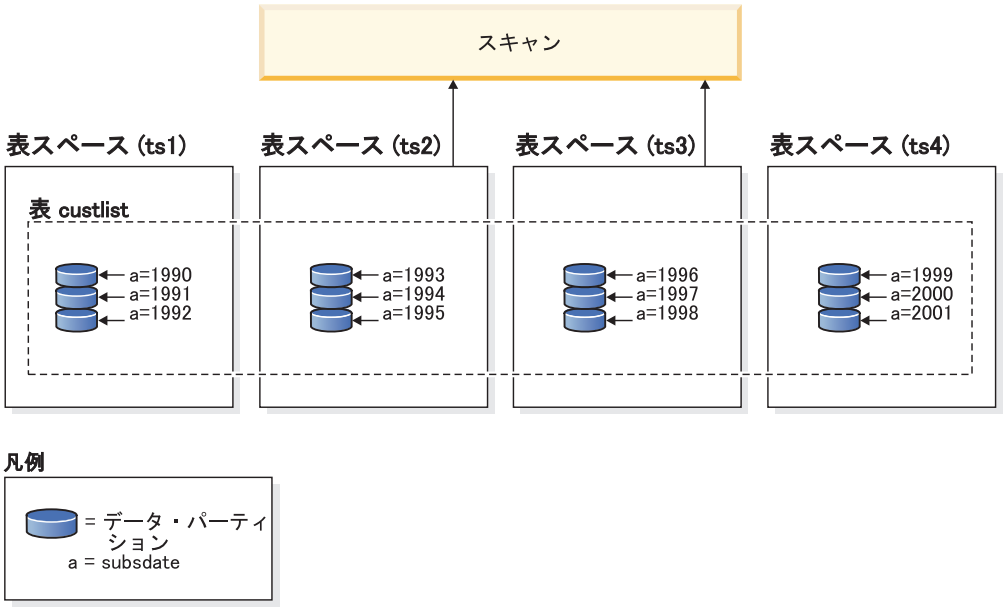


図 29. パーティション表におけるデータ・パーティションの除去のパフォーマンス上の利点

図 30 に示されているデータ・パーティションの除去の別の例は、2 つの索引が関係する索引スキャンで以下のスキームに基づいています。

```
CREATE TABLE multi (sale_date date, region char(2))
PARTITION BY (sale_date)
(STARTING '01/01/2005' ENDING '12/31/2005' EVERY 1 MONTH);
CREATE INDEX sx ON multi(sale_date);
CREATE INDEX rx ON multi(region);
```

以下の照会を発行する場合について考えます。

```
SELECT * FROM multi WHERE
sale_date BETWEEN '6/1/2005' AND '7/31/2005' AND REGION = 'NW';
```

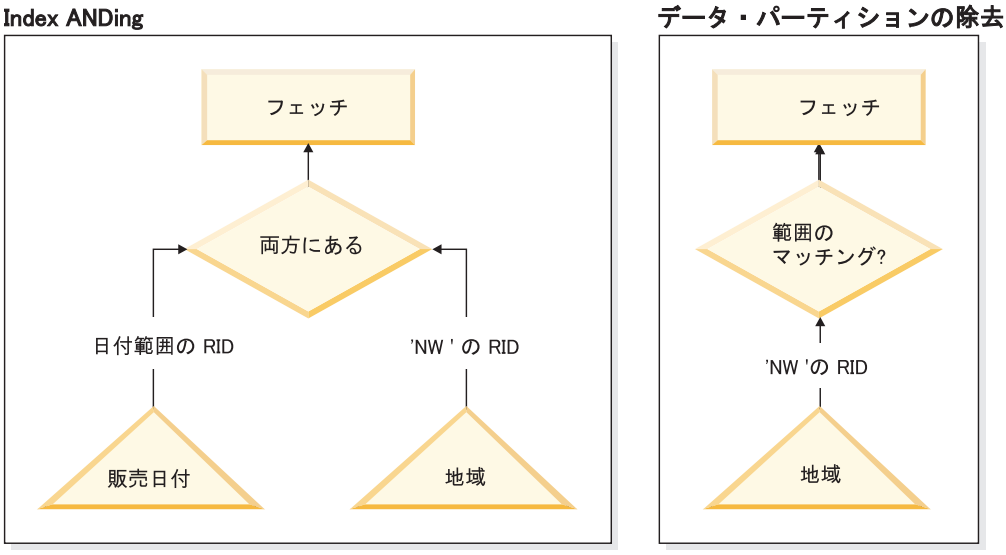


図 30. 表のパーティション化と索引 AND 演算の両方のオプティマイザーの決定パス

表のパーティション化を使用しないで考えられる 1 つのプランは、索引 AND 演算です。索引 AND 演算は、以下のタスクを実行します。

- 各索引から関連するすべての索引項目を読み取る
- 行 ID (RID) の両方のセットを保管する
- RID を突き合わせて、両方の索引のどちらで生じたかを判別する
- RID を使用して行を取り出す

345 ページの図 30 で示されているように、表のパーティション化を使用すると、region と sale_date の両方での一致を検出するために索引が読み取られ、それにより一致する行を素早く取得することができます。

DB2 Explain

DB2 Explain を使用して、DB2 オプティマイザーが選択したパーティションの除去を判別することもできます。**DP Elim Predicates** 情報には、以下の照会を解決するためにどのデータ・パーティションがスキャンされるかが示されます。

```
SELECT * FROM custlist WHERE subsdate
BETWEEN '12/31/1999' AND '1/1/2001'

Arguments:
---------
DPESTFLG: (Number of data partitions accessed are Estimated)
    FALSE
DPLSTPRT: (List of data partitions accessed)
    9-11
DPNUMPRT: (Number of data partitions accessed)
   3

DP Elim Predicates:
------------------
Range 1)
  Stop  Predicate: (Q1.A <= '01/01/2001')
 Start Predicate: ('12/31/1999' <= Q1.A)



Objects Used in Access Plan:
 ---------------------------

  Schema: MRSRINI
  Name:    CUSTLIST
  Type:    Data Partitioned Table
   Time of creation:      2005-11-30-14.21.33.857039
   Last statistics update:    2005-11-30-14.21.34.339392
    Number of columns:      3
    Number of rows:      100000
   Width of rows:      19
   Number of buffer pool pages:    1200
    Number of data partitions:    12
    Distinct row values:      No
    Tablespace name:      <VARIOUS>
```

複数列のサポート

データ・パーティションの除去は、表パーティション・キーとして複数列が使用されている場合に向いています。

たとえば、以下のステートメントを発行した場合について考えています。

```
CREATE TABLE sales (year INT, month INT)
 PARTITION BY RANGE(year, month)
 (STARTING FROM (2001, 1) ENDING AT(2001,3) IN ts1,
 ENDING AT(2001,6) IN ts2,
 ENDING AT(2001,9) IN ts3,
 ENDING AT(2001,12) IN ts4,
 ENDING AT(2002,3) IN ts5,
 ENDING AT(2002,6) IN ts6,
 ENDING AT(2002,9) IN ts7,
 ENDING AT(2002,12) IN ts8)
```

次に、以下の照会を発行します。

```
SELECT * FROM sales WHERE year = 2001 AND month < 8
```

照会オプティマイザーは、この照会を解決するために ts1、ts2 および ts3 内のデータ・パーティションのみにアクセスする必要があると推論します。

注: 表パーティション・キーが複数列から構成されている場合、複合キーの先頭列に述部がある場合に限ってデータ・パーティションの除去が可能です。表パーティション・キーとして使用される非先頭列は独立していないからです。

複数範囲のサポート

複数範囲 (が OR で結ばれたもの) を持つデータ・パーティションでデータ・パーティションの除去を行うことが可能です。前述の例で作成された表を使用して、以下の照会を実行します。

```
SELECT * FROM sales
WHERE (year = 2001 AND month <= 3) OR (year = 2002 and month >= 10)
```

データベース・サーバーは、2001 年の最初の四半期と 2002 年の最後の四半期のデータのみにアクセスします。

生成列

生成列を表パーティション・キーとして使用できます。

たとえば、以下のステートメントを発行できます。

```
CREATE TABLE sales(a INT, b INT GENERATED ALWAYS AS (a / 5))
 IN ts1,ts2,ts3,ts4,ts5,ts6,ts7,ts8,ts9,ts10
 PARTITION BY RANGE(b)
 (STARTING FROM (0) ENDING AT(1000) EVERY (50))
```

この場合、生成列の述部がデータ・パーティションの除去に使用されます。加えて、列を生成するために使用される式が単調の場合、データベース・サーバーはソース列上の述部を生成列上の述部に変換し、生成列でデータ・パーティションの除去を可能にします。

たとえば、以下の照会があるとします。

```
SELECT * FROM sales WHERE a > 35
```

データベース・サーバーは、a (a > 35) から b (b > 7) に追加の述部を生成し、データ・パーティションの除去が可能になります。

結合述部

結合述部が表アクセス・レベルに下がると、結合述部をデータ・パーティションの除去に使用することもできます。結合述部は、ネストされたループ結合 (NLJN) の内部の表アクセス・レベルにのみ下がります。

次の表を考慮してください。

```
CREATE TABLE T1(A INT, B INT)
 PARTITION BY RANGE(A, B)
 (STARTING FROM (1, 1)
 ENDING (1,10) IN ts1, ENDING (1,20) IN ts2,
 ENDING (2,10) IN ts3, ENDING (2,20) IN ts4,
  ENDING (3,10) IN ts5, ENDING (3,20) IN ts6,
 ENDING (4,10) IN ts7, ENDING (4,20) IN ts8)

CREATE TABLE T2 (A INT, B INT)
```

使用する述部は以下のとおりです。

```
P1: T1.A = T2.A
P2: T1.B > 15
```

この例では、結合の外部値が不明なため、コンパイル時にアクセスされるデータ・パーティションを正確に判別することはできません。この場合、ホスト変数またはパラメーター・マーカーが使用される場合と同様に、必要な値が結合される実行時にデータ・パーティションの除去が生じます。

実行時に T1 が NLJN の内部にある場合、述部に基づいて、T2.A のすべての外部値に対してデータ・パーティションの除去が動的に生じます。実行時に、述部 T1.A = 3 および T1.B > 15 が外部値 T2.A = 3 に対して適用され、表スペース ts6 と ts7 のデータ・パーティションにアクセスする資格を与えます。

表 T1 と T2 の列 A に以下の値がある場合を考えてみてください。

外部表 T2: 列 A	内部表 T1: 列 A	内部表 T1: 列 B	内部表 T1: データ・パーティション・ロケーション
2	3	20	ts6
3	2	10	ts3
3	2	18	ts4
	3	15	ts6
	1	40	ts3

(内部表に対する表スキャンを前提として) ネスト・ループ結合を実行するために、データベース・マネージャーは以下のステップを実行します。

1. T2 から最初の行を読み取ります。A の値は 2 です。
2. T2.A の値 (2) を結合述部 T1.A = T2.A で列 T2.A にバインドします。述部は T1.A = 2 となります。
3. 述部 T1.A = 2 および T1.B > 15 を使用して、データ・パーティションの除去を適用します。これにより、表スペース ts4 と ts5 のデータ・パーティションが資格を得ます。

4. T1.A = 2 および T1.B > 15 の適用後に行が検出されるまで、表 T1 の表スペース ts4 と ts5 のデータ・パーティションをスキャンします。資格にかなう最初の検出行は、T1 の行 3 です。
5. 一致した行を結合します。
6. 次の一致 (T1.A = 2 および T1.B > 15) が検出されるまで、表 T1 の表スペース ts4 と ts5 のデータ・パーティションをスキャンします。それ以上、行は見つかりません。
7. T2 のすべての行がなくなるまで、T2 の次の行 (A の値を 3 に置換する) に対してステップ 1 から 6 までを繰り返します。

マテリアライズ照会表

マテリアライズ照会表 (MQT) は、複雑な照会、とりわけ以下の操作が必要な照会の応答時間を改善するのに高い効果を発揮します。

- 1 ディメンション以上の集約データ
- 表のグループを対象とする結合および集約データ
- 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」水平データベース・パーティションまたは垂直データベース・パーティション) からのデータの照会
- パーティション・データベース環境での表からの再パーティション・データ、または表の一部

MQT の情報は、SQL および XQuery コンパイラーに組み込まれています。コンパイラーでは、照会書き直しのフェーズとオプティマイザーで照会と MQT の突き合わせを行って、基本表にアクセスする照会の MQT を代用するかどうかを決定します。MQT を使用する場合は、EXPLAIN 機能を使用して、選択した MQT についての情報を得られます。

MQT は多くの点でレギュラー表のように働くため、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 の発行に関する指針は、MQT にも当てはまります。

MQT がいかに役立つかを理解する助けとして、以下の表はマルチディメンション分析照会について示しています。この例では、照会がどれだけ MQT を活用するかが示されています。

この例では、一連の顧客と一連のクレジット・カード口座が含まれているデータベース・ウェアハウスを想定しています。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、一緒に購入された一連の品目が含まれます。 2 つの大きな表 (1 つはトランザクション品目を含む表、もう 1 つは購入トランザクションを示す表) が含まれるため、このスキーマはマルチスター型として分類されます。

トランザクションの記述には、製品、場所、そして時刻という 3 つの階層ディメンションがあります。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保管されます。場所階層には、市町村、都道府県、および国または地域の情報が含まれ、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、単一の日付フィールドでエンコードされます。日付のディメ

ンションは、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。このスキーマの他の表は、顧客の口座情報や顧客情報が表されます。

MQT は、以下の階層の各レベルごとに、売上の合計と数を使用して作成されます。

- 製品
- 場所
- 時刻 (年月日)

多くの照会では、この保管された集約データで用が足ります。次の例は、製品グループと製品ラインのディメンション、市町村、都道府県、国のディメンション、および時間のディメンションにしたがって売上の合計と数を計算する MQT の作成方法を示しています。この例では、GROUP BY 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;
```

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
- 1990 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 MQT 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、MQT を使用して答えを計算する方が、大きな基本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。 MQT では、コストのかかる基本データの結合、ソート、および集計を減らすことができます。

次のサンプル照会は、MQT 例にある、すでに計算された結果を使用することによってパフォーマンスを大幅に改善できる例です。

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;
```

MQT が大きくなるペースは基本表が大きくなるペースよりも遅いため、基本表が大きくなれば大きくなるほど、応答時間の向上は大きなものになります。 MQT では、MQT が作成されるときと更新されるときに一度計算を行い、その内容を多くの照会で再利用することによって、照会間での作業の重複を効果的に取り除くことができます。

Explain 機能

SQL または XQuery コンパイラーは、アクセス・プランに関する情報と、静的または動的 SQL および XQuery ステートメントの環境に関する情報をキャプチャーします。キャプチャーされた情報は、ステートメントおよびデータベース・マネージャー構成を調整して、パフォーマンスを向上させるために個々の SQL または XQuery ステートメントをどのように実行したらよいかを理解するのに役立ちます。

以下の理由で、 Explain データを収集して使用します。

- 照会を満たすために、データベース・マネージャーがどのように表および索引にアクセスするかを理解する
- パフォーマンス・チューニング・アクションを評価する

  データベース・マネージャー、SQL または XQuery ステートメント、およびデータベースのいくつかの面を変更する場合、 Explain データを調べて、行ったアクションによってパフォーマンスがどのように変更されたかを調査しなければなりません。

キャプチャーされた情報には、次のものが含まれます。

- 照会を処理する操作の順序
- コスト情報

- 述部および述部ごとの選択可能性の見積もり
- Explain 情報かキャプチャーされた時点の、 SQL または XQuery 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
- SQL または XQuery ステートメントを再最適化するのに使用する、ホスト変数、パラメーター・マーカー、または特殊レジスターの値。

Explain 情報をキャプチャーするには、その前に、リレーショナル表 (オプティマイザーが Explain 情報を保管する) を作成し、キャプチャーする Explain 情報の種類を判別する特殊レジスターを設定しなければなりません。

Explain 情報を表示するために、コマンド行ツールまたは Visual Explain のいずれかを使用できます。使用するツールによって、収集される Explain データを判別する特殊レジスターをどのように設定するかが決まります。たとえば、 Visual Explain のみを使用する必要がある場合、スナップショット情報だけをキャプチャーする必要があります。 Explain 表に対して、コマンド行ユーティリティーのいずれかを使用するか、またはカスタム SQL または XQuery ステートメントを使用して、詳細な分析を実行する必要がある場合、すべての Explain 情報をキャプチャーしなければなりません。

Explain 情報の使用のガイドライン

Explain 情報は、以下の 2 つの主な理由で使用します。

- アプリケーションのパフォーマンスが変化した理由を理解する
- パフォーマンス・チューニングの努力を評価する

パフォーマンス変化の分析

照会パフォーマンスが変化した理由を理解するには、前と後の Explain 情報が必要です。これは、以下のステップを実行すると取得できます。

- 変更前の照会の Explain 情報をキャプチャーし、結果の Explain 表を保管します。または、`db2exfmt` Explain ツールからの出力を保管します。
- 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または印刷します。 `db2look` 生産性向上ツールを使用して、このタスクの実行に役立てることもできます。
- 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) を保管または印刷します。

このようにして収集した情報では、将来の分析での参照点が提供されます。動的 SQL または XQuery 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ができます。静的 SQL および XQuery 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化を分析するために、収集した情報を、分析を開始したときの照会および環境に関して収集する情報と比較します。

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、以下のアクションのいずれかを実行するように、選択することになります。

- 索引を再編成します。
- 表と索引の新規の統計を収集します。
- 照会を再バインドするときに Explain 情報を収集します。

アクションのいずれかを実行したあとで、アクセス・プランを再度調べます。索引が再度使用されている場合、照会のパフォーマンスはもう問題ではなくなっています。索引がまだ使用されていない場合、またはパフォーマンスにまだ問題がある場合、 2 番目のアクションを実行してその結果を調べます。問題が解決されるまで、これらのステップを繰り返します。

パフォーマンス・チューニングの努力の評価

構成パラメーターの調整、コンテナーの追加、新しいカタログ統計の収集などの、多数のアクションを行うことにより、照会パフォーマンスの向上に寄与することができます。

これらの領域のいずれかで変更を行ってから、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、索引の指針に基づいて索引またはマテリアライズ照会表 (MQT) を追加した場合、 Explain データを参考にして、実際に索引またはマテリアライズ照会表が期待したとおりに使用されているかどうかを判別できます。

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、ベンチマーク・テスト技法を使用することです。

Explain 情報のキャプチャーのガイドライン

Explain データがキャプチャーされるのは、SQL または XQuery ステートメントがコンパイルされるときに Explain データを要求する場合です。 Explain データを要求するときに、キャプチャーした情報を使用する方法を考慮してください。

注:

1. 追加バインド SQL または XQuery ステートメントが実行時にコンパイルされる場合、データはバインド時ではなく実行時に Explain 表に置かれます。これらのステートメントの場合、挿入される 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。
2. Explain 情報がキャプチャーされるのは、 SQL または XQuery ステートメントがコンパイルされるときだけです。初期コンパイルの後、動的照会ステートメントは、環境の変化に伴って再コンパイルが必要な場合、あるいは Explain 機能がアクティブな場合、再コンパイルされます。同じ PREPARE ステートメントを同じ照会ステートメントに対して発行すると、このステートメントを準備または実行するたびに、ステートメントがコンパイルされ、 explain データがキャプチャーされます。
3. BIND オプション REOPT ONCE/ALWAYS を使用してパッケージをバインドすると、ホスト変数、パラメーター・マーカー、グローバル変数、または特殊レジ

スターを含む SQL または XQuery ステートメントはコンパイルされ、これらの変数が分かっている場合には実際の値を使用してアクセス・パスが作成され、コンパイル時でこうした値の分からない場合にはデフォルトの推定値を使用してアクセス・パスが作成されます。

4. FOR REOPT ONCE 節を使用すると、指定された SQL または XQuery ステートメントを、パッケージ・キャッシュ内の同じステートメントと突き合わせようとします。すでに再最適化されてキャッシュに入れられた照会ステートメントの値が、指定された照会ステートメントの再最適化に使用されます。ユーザーに必要なアクセス権があるなら、Explain 表には、新たに生成された、再最適化されたアクセス・プランと、この再最適化に使用された値が入ることになります。
5. マルチ・パーティション・システムでステートメントを Explain する場合は、REOPT ONCE を使用して、最初にコンパイルおよび再最適化されたのと同じデータベース・パーティション上で Explain する必要があります。そうしないと、エラーが戻ります。

Explain 表内の情報のキャプチャー

- **静的または追加バインド SQL および XQuery ステートメントの場合**:

  BIND または PREP コマンドに EXPLAIN ALL または EXPLAIN YES オプションのどちらかを指定するか、またはソース・プログラムに静的 EXPLAIN ステートメントを含めます。

- **動的 SQL および XQuery ステートメント**:

  次のいずれかの状況について、Explain 表情報がキャプチャーされます。

  – CURRENT EXPLAIN MODE 特殊レジスターが以下のように設定されます。
    - YES: SQL および XQuery コンパイラーは、Explain データをキャプチャーし、照会ステートメントを実行します。
    - EXPLAIN: SQL および XQuery コンパイラーは Explain データをキャプチャーしますが、照会ステートメントは実行しません。
    - RECOMMEND INDEXES: SQL および XQuery コンパイラーは Explain データをキャプチャーし、推奨索引が ADVISE_INDEX 表に入れられますが、照会ステートメントは実行されません。
    - EVALUATE INDEXES: SQL および XQuery コンパイラーは、評価のために ADVISE_INDEX 表に置かれた索引を使用します。 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、照会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。提案された索引が役立つかどうかを調べるには、 EXPLAIN 結果を検討してください。
    - REOPT: 照会コンパイラーは、実行時のステートメント再最適化中にホスト変数、特殊レジスター、グローバル変数、またはパラメーター・マーカーの実際の値が使用できる場合、静的または動的 SQL または XQuery ステートメントのために Explain データをキャプチャーします。

- EXPLAIN ALL オプションが BIND または PREP コマンドで設定されています。 CURRENT EXPLAIN MODE 特殊レジスターが NO に設定されていても、

照会コンパイラーは実行時に Explain データを動的 SQL および XQuery 用にキャプチャーします。SQL または XQuery ステートメントも照会を実行してその結果を戻します。

Explain スナップショット情報のキャプチャー

Explain スナップショットが要求されると、 Explain スナップショット情報は、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 自体から使用できます。この情報には、データ・オブジェクトおよびデータ演算子に関する情報が含まれています。

SQL または XQuery 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、以下のように、Explain スナップショット・データがキャプチャーされます。

- **静的または追加バインド SQL および XQuery ステートメントの場合**:

  EXPLSNAP ALL か EXPLSNAP YES 節のどちらかが BIND または PREP コマンドに指定されたり、または FOR SNAPSHOT か WITH SNAPSHOT 節を使用する静的 EXPLAIN ステートメントがソース・プログラムに含まれている場合に、 Explain スナップショットがキャプチャーされます。
- **動的 SQL および XQuery ステートメント**:

  次のいずれかの場合に、Explain スナップショットがキャプチャーされます。
  - FOR SNAPSHOT または WITH SNAPSHOT 節を使用する EXPLAIN ステートメントを発行します。 FOR SNAPSHOT 節では、Explain スナップショット情報のみがキャプチャーされます。 WITH SNAPSHOT 節では、スナップショット情報に加えて、すべての Explain 情報がキャプチャーされます。
  - CURRENT EXPLAIN SNAPSHOT 特殊レジスターが以下のように設定されます。
    - YES: 照会コンパイラーは、スナップショット Explain データをキャプチャーし、 SQL または XQuery ステートメントを実行します。
    - EXPLAIN: 照会コンパイラーは、スナップショット Explain データをキャプチャーしますが、 SQL または XQuery ステートメントは実行しません。
  - BIND または PREP コマンドに EXPLSNAP ALL オプションを指定します。CURRENT EXPLAIN SNAPSHOT 特殊レジスターの設定値が NO であっても、照会コンパイラーは実行時にスナップショット Explain データをキャプチャーします。また SQL または XQuery ステートメントを実行します。

Explain 情報の分析のガイドライン

EXPLAIN 情報は、主に照会ステートメントのアクセス・パスの分析のために使用します。 Explain データの分析が照会や環境の調整に役立つ多数の方法があります。以下の種類の分析を考慮してください。

- **索引の使用**

適切な索引は、パフォーマンスをかなり向上させることができます。 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 節
- ORDER BY 節
- WHERE XMLEXISTS 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。 新規索引を作成した後、 RUNSTATS コマンドを使用してその索引の統計を収集したり、照会を再コンパイルします。すでに Explain データを介して、索引スキャンの代わりに表スキャン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リングを変更すると、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、その索引に応じてそのデータをクラスター化する表を再編成し、 RUNSTATS コマンドを使用して索引と表の両方の統計を収集してから、照会を再コンパイルすることができます。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、再コンパイルされた照会の Explain 出力を再検査します。

- **アクセス・タイプ**

  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。例:

  - **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スキャン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 照会が表スキャンを使用している場合は、Explain データを分析して、索引スキャンが使用されなかった理由を判別することができます。

  - **表示専用照会**

    「ブラウズ」タイプの照会の検索基準は不明確で、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オプティマイザーの基本操作方針、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・スキャン結合とソート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マテリアライズされます。この場合、SELECT ステートメントの OPTIMIZE FOR 節を用いてアクセス・プランを変更することができます。このオプションを指定する場合、オプティマイザーは

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。

- **結合方式**

  照会が 2 つの表を結合する場合は、使用されている結合のタイプを調べます。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、ハッシュ結合やマージ結合を使用するとより速く実行します。数行しか含まれない結合 (OLTP 照会など) は、一般に、NESTED LOOP 結合を使用するとより速く実行します。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。

アクセス・プランを使用して、REFRESH TABLE および SET INTEGRITY ステートメントからパフォーマンス上の問題を自己診断する

EXPLAIN for REFRESH TABLE および SET INTEGRITY ステートメントにより、これらのステートメントに関するパフォーマンス上の問題を自己診断するために使用できるアクセス・プランを生成できます。これは、マテリアライズ照会表 (MQT) を適切に保守するのに役立ちます。

REFRESH TABLE または SET INTEGRITY ステートメントのアクセス・プランを取得するには、以下のいずれかの方法を使用します。

- EXPLAIN ステートメントで EXPLAIN PLAN FOR REFRESH TABLE または EXPLAIN PLAN FOR SET INTEGRITY オプションを使用する
- REFRESH TABLE または SET INTEGRITY ステートメントを発行する前に CURRENT EXPLAIN MODE 特殊レジスターを EXPLAIN に設定し、発行した後に CURRENT EXPLAIN MODE 特殊レジスターを NO に設定する

制約事項:

- REFRESH TABLE および SET INTEGRITY ステートメントは再最適化に適格でないため、REOPT Explain モード (または Explain スナップショット) はこれらの 2 つのステートメントに適していません。
- EXPLAIN ステートメントの WITH REOPT ONCE 節 (指定された EXPLAIN 可能ステートメントを再最適化することを示す) も REFRESH TABLE および SET INTEGRITY に適していません。

シナリオ

このシナリオでは、EXPLAIN および REFRESH TABLE ステートメントからアクセス・プランを生成し、それを使用してパフォーマンス上の問題の原因を自己診断する方法を示します。

ステップ 1

表を作成し、そこにデータを入力します。例:

```
CREATE TABLE T
    (i1 INT NOT NULL,
     i2 INT NOT NULL,
     PRIMARY KEY (i1));
```

```
INSERT INTO T VALUES (1,1), (2,1), (3,2), (4,2);
CREATE TABLE MQT AS (SELECT i2, COUNT(*) AS CNT FROM T GROUP BY i2)
DATA INITIALLY DEFERREDREFRESH DEFERRED;
```

ステップ 2

次のように、EXPLAIN および REFRESH TABLE ステートメントを発行します。

```
EXPLAIN PLAN FOR REFRESH TABLE MQT;
```

注: このステップの代わりに、次のようにして、SET CURRENT EXPLAIN MODE 特殊レジスターで EXPLAIN モードを設定することもできます。

```
SET CURRENT EXPLAIN MODE EXPLAIN;
REFRESH TABLE MQT;
SET CURRENT EXPLAIN MODE NO;
```

ステップ 3

db2exfmt コマンドを使用して、Explain 表の内容をフォーマットし、アクセス・プランを取得します。このツールは、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にあります。

```
db2exfmt –d <dbname> -o refresh.exp -1
```

ステップ 4

アクセス・プランを分析して、パフォーマンス上の問題の原因を判別します。例えば、上記のステートメントからプランを分析することにより、T が LARGE 表である場合、TABLE SCAN が使用されると非常にコストがかかります。索引を作成すると、照会のパフォーマンスが向上します。

Explain ツール

DB2 で提供されている包括的な Explain 機能により、 SQL または XQuery ステートメントでオプティマイザーが選択するアクセス・プランについての、詳細な情報が提供されます。Explain データが保管される表は、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、静的と動的の両方の SQL および XQuery ステートメントに関する情報が含まれています。いくつかのツールおよび方法が提供されているため、 Explain 情報のキャプチャー、表示、および分析を柔軟に行うことができます。

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オプティマイザー情報は、実際のアクセス・プランとは別の Explain 表に保管されます。 Explain 表から情報を入手するための以下の方法のうち、 1 つ以上を使用してください。

- Visual Explain を使用して、Explain スナップショット情報を表示します。

  コントロール・センターから Visual Explain を呼び出して、照会アクセス・プランのグラフィカル表示を参照します。静的と動的の両方の SQL および XQuery ステートメントを分析できます。

  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 クライアントでは、 DB2 for HP-UX サーバー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。
- db2exfmt ツールを使用して、事前に形式設定されている出力の Explain 情報を表示します。
- db2expln および dynexpln ツールを使用します。

静的 SQL または XQuery ステートメントの 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 ツールを使用します。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。オプティマイザー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 または XQuery 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 または XQuery 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、 Explain 機能を使用してください。

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。

- Explain 表に対する独自の照会を作成します。

  独自の照会を作成することにより、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をかけて比較したりすることができます。

注: このコマンド行の Explain ツールや、ほかのツール (db2batch、 dynexpln、db2_all など) 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。ツールをこのパスから移動させると、コマンド行の方法はうまくいかなくなります。

次の表では、 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 64. Explain 機能ツール

希望する特性	**Visual Explain**	**Explain 表**	**db2exfmt**	**db2expln**	**dynexpln**
GUI インターフェース	はい				
テキスト出力			はい	はい	はい
「簡易」静的 SQL および XQuery 分析				はい	
サポートされる静的 SQL および XQuery	はい	はい	はい	はい	
サポートされる動的 SQL および XQuery	はい	はい	はい	はい	はい*
サポートされる CLI アプリケーション	はい	はい	はい		
DRDA® アプリケーション・リクエスターで使用可能		はい			
詳細オプティマイザー情報	はい	はい	はい		
複数ステートメントの分析に適合		はい	はい	はい	はい
アプリケーション内部からアクセス可能な情報		はい			

表 64. Explain 機能ツール (続き)

希望する特性	Visual Explain	Explain 表	db2exfmt	db2expln	dynexpln
注:					
* db2expln を間接的に使用します。制限がいくつかあります。					

事実上 Explain 時点のものであるカタログ統計の表示

EXPLAIN 機能を使用すると、実質的にはステートメントの EXPLAIN 時の統計がキャプチャーされます。こうした統計はシステム・カタログに格納されているものとは異なる可能性があります。特に、リアルタイム統計収集が有効になっている場合には、そう言えます。Explain 表にデータが追加されているものの Explain スナップショットが作成されていない場合には、EXPLAIN_OBJECT 表に記録される統計は一部に限定されます。

EXPLAIN されているステートメントに関連したすべてのカタログ統計をキャプチャーするには、Explain 表にデータが追加されると同時に Explain スナップショットを作成します。その後、SYSPROC.EXPLAIN_FORMAT_STATS 関数を使用してスナップショットのカタログ統計をフォーマット設定します。

db2exfmt ツールを使用して EXPLAIN 情報をフォーマット設定する場合、スナップショットが収集されているのであれば自動的に SYSPROC.EXPLAIN_FORMAT_STATS 関数が使用されてカタログ統計が表示されます。Visual Explain はスナップショット内にあるすべての統計を自動的に表示します。

SQL および XQuery Explain ツール

db2expln ツールは、 SQL および XQuery 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、 Explain データがキャプチャーされなかったとき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。静的 SQL および XQuery ステートメントでは、 db2expln を使用してシステム・カタログ表に保管されたパッケージを調べます。動的 SQL および XQuery ステートメントでは、 db2expln を使用して照会キャッシュ内のセクションを調べます。

dynexpln ツールを使用して、動的ステートメント用に選択されたアクセス・プランを記述することもできます。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。しかし、動的 SQL および XQuery ステートメントは db2expln を使用して調べることができるので、このユーティリティーが残っているのは以前のバージョンとの互換性のためだけです。

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 bin 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接続され、db2expln.bnd、db2exsrv.bnd、および db2exdyn.bnd ファイルを用いてデータベースにバインドされます。

db2expln を実行するには、システム・カタログ・ビューに対する SELECT 特権と、 db2expln、db2exsrv、および db2exdyn のパッケージの EXECUTE 特権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 SQL スキーマが存在しているか、もしくはデータベースへの IMPLICIT_SCHEMA 権限を所有している必要があります。 db2expln または dynexpln を使用して動的 SQL および XQuery ステートメントを Explain するには、 Explain されている照会ステートメント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。)

dynexpln

dynexpln ツールは、以前のバージョンとの互換性のために今でも使用可能です。しかし、db2expln の **dynamic-options** を使用して、 dynexpln のすべての機能を実行できます。

db2expln の dynamic-options を使用するとき、ステートメントは真の動的 SQL または XQuery ステートメントとして準備され、生成されるプランは照会キャッシュから Explain されます。 explain の出力方法は、ステートメントを静的 SQL または XQuery ステートメントとして準備する dynexpln よりも正確なアクセス・プランを提供します。これにより、パラメーター・マーカーなど動的 SQL および XQuery ステートメントだけで使用できる機能も使用可能になります。

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 プリ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。プリ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>" をドロップ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ドロップして実行するか、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 および XQuery 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 つの部分に分かれて表示されます。

- バインド日付や関係する BIND 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 または XQuery 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 または XQuery ステートメント用に選択されたアクセス・プランの Explain 出力が表示されます。

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。字下げの棒線は、アクセス・プランの Explain 出力の左マージンに表示されます。これらの棒線は、操作の有効範囲を示すものともなります。ずっと右側にある、同じ操作の中でもより下位の字下げレベルの操作は、すぐ上位にある字下げレベルに戻る前に処理されます。

出力に表示されている、選択されたアクセス・プランが元の SQL または XQuery 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。さらに、照会コンパイラーの照会書き直しコンポーネントは、 SQL または XQueryステートメントを同等ではあるもののより効率的な形式に書き直すことがあります。オプティマイザー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、オプティマイザーはこれらのすべての要素を含む情報を使用します。したがって、Explain 出力に示されるアクセス・プランが、元の SQL または XQuery 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。Explain 表、SET CURRENT EXPLAIN モード、および Visual Explain を含む Explain 機能は、最適化に使用される実際の SQL または XQuery ステートメントを、照会の内部表記を逆変換して作成し、 SQL または XQuery 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、演算子 ID オプション (**-opids**) は非常に便利です。 db2expln または dynexpln が、 Explain 機能から新しい演算子の処理を開始するたびに、演算子 ID 番号が、Explain されるプランの左側に印刷されます。演算子 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 機能の出力の中の演算子と、 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。

表アクセス情報: 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. レギュラー表には、以下の 3 つのタイプがあります。
   - アクセスした表名:

     ```
     Access Table Name = schema.name  ID = ts,n
     ```

     詳細は次のとおりです。
     - *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。
   - アクセスした階層表名:

     ```
     Access Hierarchy Table Name = schema.name  ID = ts,n
     ```

詳細は次のとおりです。

- *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。

• アクセスしたマテリアライズ照会表名:

```
Access Materialized Query Table Name = schema.name  ID = ts,n
```

詳細は次のとおりです。

- *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。

• アクセスした一時表 ID:

```
Access Temp Table  ID = tn
```

詳細は次のとおりです。

- *ID* は、db2expln から割り当てられた対応の ID です。

• アクセスした宣言済みグローバル一時表 ID:

```
Access Global Temp Table  ID = ts,tn
```

詳細は次のとおりです。

- *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
- ブロック・アクセス
- 並列スキャン
- スキャンの方向
- 行アクセス方式
- ロックの意図
- 述部
- その他のステートメント

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
```

ブロック・アクセス

以下のステートメントは、表に 1 つ以上のディメンション・ブロック索引が定義されていることを示しています。

```
Clustered by Dimension for Block Index Access
```

このテキストが表示されていない場合は、表は DIMENSION 節を指定されないで作成されました。

並列スキャン

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。

スキャンの方向

次のステートメントは、データベース・マネージャーが行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、スキャン方向は順方向で、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スキャンすることを示します。
  - 次のステートメントは、データのプリフェッチ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、プリフェッチされるページ数をオプティマイザー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、データをプリフェッチ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 type
    | Index Columns:
    ```

    詳細は次のとおりです。

    - *schema.name* は、スキャン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。
    - 索引のタイプは、以下の 1 つです。

```
Regular Index (Not Clustered)
Regular Index (Clustered)
Dimension Block Index
Composite Dimension Block Index
XML データに対する索引
```

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、索引スキャン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。

- Start of Index
- End of Index
- Inclusive Value: or Exclusive Value:

  索引スキャンには、inclusive キー値が含まれます。スキャンに exclusive 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。

- Index-Only Access

  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、索引ページのプリフェッチ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、索引ページをプリフェッチ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・ページのプリフェッチ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・ページをプリフェッチすることを示します。

  ```
  Data Prefetch: Eligible
  ```

- 索引ガバナー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
  ```

– 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場合、それは次のステートメントによって示されます。

```
Fetch Direct Using Row IDs
```

表に 1 つ以上のブロック索引が定義されている場合、ブロックまたは行 ID を使用して行にアクセスすることができます。このことは、次に示します。

```
Fetch Direct Using Block or Row IOs
```

ロックの意図

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。

- Exclusive
- Intent Exclusive
- Intent None
- Intent Share
- Share
- Share Intent Exclusive
- Super Exclusive
- Update

行ロックに可能な値は、以下のとおりです。

- Exclusive
- Next Key Exclusive (db2expln 出力に表示されません)
- なし
- Share
- Next Key Share
- Update
- Next Key Weak Exclusive
- Weak Exclusive

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、ブロック化索引から検索されたデータのブロックごとに、述部の数が評価されることを示します。

   ```
   Block Predicates(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、集約やソートなどのプッシュダウン操作は含まれていません。

   ```
   Sargable Predicate(s)
   |  #Predicates = n
   ```

3. 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
   ```

このステートメントに表示された述部の数は、照会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ます。
- 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ます。
- 照会最適化処理時に変形と圧縮が行われ、述部が少なくなります。

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、ステートメント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ダウングレードします。
  – 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ます。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 volatile 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
  ```

一時表の情報: 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、一時表が使用されます。

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、ソート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、ソート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、ソート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、ソート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。

ソート済みの一時表

次のような操作から、ソート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' subquery
- '<> ALL' subquery
- INTERSECT または EXCEPT
- UNION (ALL キーワードの指定なし)

ソート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、ソートに使用するキー列の数を示します。

  ```
  #Sort Key Columns = n
  ```

  ソート・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、実行時に最適なソート・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
  ```

- ソート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) 環境でのソートの場合は、実行されるソート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
  ```

- 次のステートメントは、ソートの結果をソート・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・ソートが指示されている場合は、データベース・マネージャーは、ソートの出力をメモリーに保管して、ソート結果を別の一時表には入れません。

- 次のステートメントは、ソート中に重複値を除去することを示しています。

  ```
  Duplicate Elimination
  ```

- ソート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プッシュダウン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。表関数は以下によって示されます。

```
Access User Defined Table Function
|  Name = schema.funcname
|  Specific Name = specificname
|  SQL Access Level = accesslevel
|  Language = lang
|  Parameter Style = parmstyle
|  Fenced                       Not Deterministic
|  Called on NULL Input         Disallow Parallel
|  Not Federated                Not Threadsafe
```

特定の名前は起動される表関数を一意的に識別します。残りの行は、機能の属性を詳細に示しています。

結合の情報: 次の 3 つのタイプの結合があります。

- 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
Merge Join
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
Left Outer Merge Join
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プッシュダウンされた場合、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、*composite inner* 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 を作成します。
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します。

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・ストリーム情報: 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な 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
  ```

- ソート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、および削除の情報: 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: Materialized Query Table = schema.name ID = ts,n
Update: Materialized Query Table = schema.name ID = ts,n
Delete: Materialized Query Table = 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
```

ブロックと行 ID の準備の情報: 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、修飾行およびブロックの ID (ID) をソートしておき、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 ID 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 ID 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いずれも、Index ORing を使用して修飾 ID のリストを作成します。

  ```
  Index ORing Preparation
  Block Index ORing 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 ID 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、オプティマイザーは索引の 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。

- ID 作成のもう 1 つの使用法は、以下のいずれかに示すように、リスト・プリフェッチ中に使用される入力データを作成することです。

  ```
  List Prefetch Preparation
  Block 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 ID 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のいずれかで開始されます。

  ```
  Index ANDing
  Block Index ANDing
  ```

  オプティマイザー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、ID 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 ID を判別します。下記のステートメントは、index ANDing 用に ID が処理されていることを示します。

  ```
  Index ANDing Bitmap Build Using Row IDs
  Index ANDing Bitmap Probe Using Row IDs
  Index ANDing Bitmap Build and Probe Using Row IDs
  Block Index ANDing Bitmap Build Using Block IDs
  Block Index ANDing Bitmap Build and Probe Using Block IDs
  Block Index ANDing Bitmap Build and Probe Using Row IDs
  Block Index ANDing Bitmap Probe Using Block IDs and Build Using Row IDs
  Block Index ANDing Bitmap Probe Using Block IDs
  Block Index ANDing Bitmap Probe Using Row IDs
  ```

オプティマイザー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 ID 作成の場合でも、リスト・プリフェッチ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
```

集約の情報: 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、集約操作がプッシュダウン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、実行される統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
```

並列処理の情報: SQL ステートメントを並列で実行する場合 (パーティション内並列処理またはパーティション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。

- パーティション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
  ```

- パーティション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 つ以上のデータベース・パーティションに送られて、実行されます。重要なサブセクションは、*コーディネーター・サブセクション* です。コーディネーター・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散は、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データベース・パーティションは、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、データベース・パーティション・グループ内にあるデータベース・パーティション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、事前に決められたデータベース・パーティション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、指定のデータベース・パーティション・グループで事前に決められたデータベース・パーティション番号に対応するデータベース・パーティション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データベース・パーティションに、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 つのデータベース・パーティション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のいずれかは、コーディネーター・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 Application Coordinator Node
  Directed to Local Coordinator Node
  ```

- 以下のステートメントは、リストされたすべてのデータベース・パーティション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 つのデータベース・パーティション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
  ```

- パーティション・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・パーティション表キューの場合は、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。

    すべての行がコーディネーター・ノードに送られます。

    ```
    Broadcast to Coordinator Node
    ```

    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・パーティションに、すべての行が送られます。

    ```
    Broadcast to All Nodes of Subsection n
    ```

    行にある値に基づいて、各行がデータベース・パーティションに送られます。

    ```
    Hash to Specific Node
    ```

    各行は、ステートメントが実行されている間に決定されたデータベース・パーティションに送られます。

    ```
    Send to Specific Node
    ```

    各行は、ランダムに決定されたデータベース・パーティションに送られます。

    ```
    Send to Random Node
    ```

  - ある種の状況では、データベース・パーティション表キューは、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
    ```

  - 表アクセスの際にプッシュダウン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ソートする場合は、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、ソート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
```

ソート・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
```

- パーティション・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
```

フェデレーテッド照会の情報: フェデレーテッド・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。

読み取られるデータ・ソースは以下のように指示されます。

```
Ship 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
```

データ・ソースで生じる挿入、更新、または削除操作は、適切なメッセージによって示されます。

```
Ship Distributed Insert #n
Ship Distributed Update #n
Ship Distributed Delete #n
```

表がデータ・ソースで明示的にロックされている場合、これは次のステートメントによって示されます。

```
Ship Distributed Lock Table #n
```

データ・ソースに対する DDL ステートメントは、 2 つの部分に分割されます。データ・ソースで呼び出される部分は、以下によって示されます。

```
Ship Distributed DDL Statement #n
```

フェデレーテッド・サーバーがパーティション・データベースである場合、 DDL ステートメントの一部はカタログ・ノードで実行する必要があります。このことは、次に示します。

```
Distributed DDL Statement #n Completion
```

各分散サブステートメントの詳細は、個別に指定されます。分散ステートメントのオプションは、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
```

- データ・ソースがリレーショナルである場合、サブステートメントの SQL は以下のように表示されます。

```
SQL Statement:
     statement
```

非リレーショナルのデータ・ソースは、以下によって示されます。

```
Non-Relational Data Source
```

- サブステートメント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。

```
Nicknames Referenced:
   schema.nickname ID = n
```

データ・ソースがリレーショナルである場合、ニックネームの基本表は以下のように示されます。

```
Base = baseschema.basetable
```

データ・ソースが非リレーショナルである場合、ニックネームのソース・ファイルは以下のように示されます。

```
Source File = filename
```

- サブステートメントを実行する前にフェデレーテッド・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
```

- サブステートメントを実行した後でデータ・ソースからフェデレーテッド・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 Explain 情報:

- 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 節が含まれる場合、次のテキストが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。

– ユニーク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ます。

– ソート中に重複を除去することができます。

- 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
  ```

  位置操作がフェデレーテッド・データ・ソースに対するものである場合、ステートメントは以下のようになります。

  ```
  Distributed 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
  ```

- ロールアウト最適化に適格なマルチディメンション・クラスタリング表に対する削除操作については、以下の情報が表示されます。

```
CELL DELETE with deferred cleanup
```

または

```
CELL DELETE with immediate cleanup
```

- 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プッシュダウンされる場合、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
```

- ストアード・プロシージャーが呼び出されている場合、以下の情報が表示されます。

```
Call Stored Procedure
|   Name = schema.funcname
|   Specific Name = specificname
|   SQL Access Level = accesslevel
|   Language = lang
|   Parameter Style = parmstyle
|   Expected Result Sets = n
|   Fenced                      Not Deterministic
|   Called on NULL Input        Disallow Parallel
|   Not Federated               Not Threadsafe
```

- 1 つ以上の LOB ロケーターが解放されている場合、以下の情報が表示されます。

```
Free LOB Locators
```

db2expln および dynexpln 出力の例

ここに示されている例は、 db2expln および dynexpln からの出力のレイアウトと形式を理解するために役立ちます。これらの例は、別段の説明がない限り、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: 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ハッシュ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・プランは各表が結合される際にリレーション・スキャンを行います。

```
******************** PACKAGE ****************************************

Package Name = "DOOLE"."EXAMPLE"  Version = ""

 Prep Date = 2002/01/04
 Prep Time = 14:05:00

 Bind Timestamp = 2002-01-04-14.05.00.415403

 Isolation Level          = Cursor Stability
 Blocking                 = Block Unambiguous Cursors
 Query Optimization Class = 5

 Partition Parallel       = No
 Intra-Partition Parallel = No
```

```
 SQL Path                 = "SYSIBM", "SYSFUN", "SYSPROC", "DOOLE"

-------------------- SECTION ---------------------------------------
Section = 1


SQL Statement:
  DECLARE EMPCUR CURSOR
  FOR
     SELECT e.lastname, e.job, d.deptname, d.location, p.projname
     FROM employee AS e, department AS d, project AS p
     WHERE e.workdept = d.deptno AND e.workdept = p.deptno


Estimated Cost        = 120.518692
Estimated Cardinality = 221.535980

(    6) Access Table Name = DOOLE.EMPLOYEE  ID = 2,5
        |  #Columns = 3
        |  Relation Scan
        |  |  Prefetch: Eligible
        |  Lock Intents
        |  |  Table: Intent Share
        |  |  Row  : Next Key Share
(    6) |  Process Build Table for Hash Join
(    2) Hash Join
        |  Estimated Build Size: 7111
        |  Estimated Probe Size: 9457
(    5)    Access Table Name = DOOLE.PROJECT  ID = 2,7
        |  |  #Columns = 2
        |  |  Relation Scan
        |  |  |  Prefetch: Eligible
        |  |  Lock Intents
        |  |  |  Table: Intent Share
        |  |  |  Row  : Next Key Share
(    5)    |  Process Build Table for Hash Join
(    3)    Hash Join
        |  |  Estimated Build Size: 5737
        |  |  Estimated Probe Size: 6421
(    4)    |  Access Table Name = DOOLE.DEPARTMENT  ID = 2,4
        |  |  |  #Columns = 3
        |  |  |  Relation Scan
        |  |  |  |  Prefetch: Eligible
        |  |  |  Lock Intents
        |  |  |  |  Table: Intent Share
        |  |  |  |  Row  : Next Key Share
(    4)    |  |  |  Process Probe Table for Hash Join
(    1) Return Data to Application
        |  #Columns = 5

End of section


Optimizer Plan:

              RETURN
              (   1)
                |
              HSJOIN
              (   2)
             /      \
         HSJOIN      TBSCAN
         (   3)      (   6)
        /      \       |
    TBSCAN    TBSCAN   Table:
    (   4)    (   5)   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、ハッシュ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: パーティション内並列処理による単一パーティションのプラン: この例は、最初の例と同じ 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"."EXAMPLE"  Version = ""

 Prep Date = 2002/01/04
 Prep Time = 14:12:38

 Bind Timestamp = 2002-01-04-14.12.38.732627

 Isolation Level          = Cursor Stability
 Blocking                 = Block Unambiguous Cursors
 Query Optimization Class = 5

 Partition Parallel       = No
 Intra-Partition Parallel = Yes (Bind Degree = 4)

 SQL Path                 = "SYSIBM", "SYSFUN", "SYSPROC", "DOOLE"

-------------------- SECTION ---------------------------------------
Section = 1


SQL Statement:
  DECLARE EMPCUR CURSOR
  FOR
     SELECT e.lastname, e.job, d.deptname, d.location, p.projname
     FROM employee AS e, department AS d, project AS p
     WHERE e.workdept = d.deptno AND e.workdept = p.deptno

Intra-Partition Parallelism Degree = 4

Estimated Cost        = 133.934692
Estimated Cardinality = 221.535980

(    2) Process Using 4 Subagents
(    7) |  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
(    7) |  |  Process Build Table for Hash Join
(    3) |  Hash Join
        |  |  Estimated Build Size: 7111
        |  |  Estimated Probe Size: 9457
(    6) |  |  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
(    6) |  |  |  Process Build Table for Hash Join
(    4) |  |  Hash Join
        |  |  |  Estimated Build Size: 5737
        |  |  |  Estimated Probe Size: 6421
(    5) |  |  |  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
(    5) |  |  |  |  Process Probe Table for Hash Join
(    2) |  Insert Into Asynchronous Local Table Queue  ID = q1
(    2) Access Local Table Queue  ID = q1  #Columns = 5
(    1) Return Data to Application
        |  #Columns = 5

End of section


Optimizer Plan:

              RETURN
              (   1)
                |
               LTQ
              (   2)
                |
              HSJOIN
              (   3)
             /      \
        HSJOIN         TBSCAN
        (   4)         (   7)
       /      \          |
   TBSCAN    TBSCAN   Table:
   (   5)    (   6)   DOOLE
     |         |      EMPLOYEE
 Table:      Table:
 DOOLE       DOOLE
 DEPARTMENT  PROJECT
```

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。

例 3: パーティション間並列処理による複数パーティションのプラン: この例は、最初の例と同じ 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・パーティションからなるパーティション・データベースでコンパイルされたものです。

```
******************** PACKAGE ************************************

Package Name = "DOOLE"."EXAMPLE"  Version = ""

 Prep Date = 2002/01/04
 Prep Time = 14:54:57

 Bind Timestamp = 2002-01-04-14.54.57.033666

 Isolation Level          = Cursor Stability
```

```
 Blocking                = Block Unambiguous Cursors
 Query Optimization Class = 5

 Partition Parallel      = Yes
 Intra-Partition Parallel = No

 SQL Path                = "SYSIBM", "SYSFUN", "SYSPROC", "DOOLE"

-------------------- SECTION -------------------------------------
Section = 1


SQL Statement:
  DECLARE EMPCUR CURSOR
  FOR
     SELECT e.lastname, e.job, d.deptname, d.location, p.projname
     FROM employee AS e, department AS d, project AS p
     WHERE e.workdept = d.deptno AND e.workdept = p.deptno


Estimated Cost        = 118.483406
Estimated Cardinality = 474.720032

        Coordinator Subsection:
(-----)    Distribute Subsection #2
           |  Broadcast to Node List
           |  |  Nodes = 10, 33, 55
(-----)    Distribute Subsection #3
           |  Broadcast to Node List
           |  |  Nodes = 10, 33, 55
(-----)    Distribute Subsection #1
           |  Broadcast to Node List
           |  |  Nodes = 10, 33, 55
(    2)    Access Table Queue  ID = q1  #Columns = 5
(    1)    Return Data to Application
           |  #Columns = 5

        Subsection #1:
(    8)    Access Table Queue  ID = q2  #Columns = 2
(    3)    Hash Join
           |  Estimated Build Size: 5737
           |  Estimated Probe Size: 8015
(    6)    |  Access Table Queue  ID = q3  #Columns = 3
(    4)    |  Hash Join
           |  |  Estimated Build Size: 5333
           |  |  Estimated Probe Size: 6421
(    5)    |  |  Access Table Name = DOOLE.DEPARTMENT  ID = 2,4
           |  |  |  #Columns = 3
           |  |  |  Relation Scan
           |  |  |  |  Prefetch: Eligible
           |  |  |  Lock Intents
           |  |  |  |  Table: Intent Share
           |  |  |  |  Row  : Next Key Share
(    5)    |  |  |  Process Probe Table for Hash Join
(    2)    Insert Into Asynchronous Table Queue  ID = q1
           |  Broadcast to Coordinator Node
           |  Rows Can Overflow to Temporary Table

        Subsection #2:
(    9)    Access Table Name = DOOLE.PROJECT  ID = 2,7
           |  #Columns = 2
           |  Relation Scan
           |  |  Prefetch: Eligible
           |  Lock Intents
           |  |  Table: Intent Share
           |  |  Row  : Next Key Share
(    9)    |  Insert Into Asynchronous Table Queue  ID = q2
```

```
            |  |  Hash to Specific Node
            |  |  Rows Can Overflow to Temporary Tables
(    8)     Insert Into Asynchronous Table Queue Completion  ID = q2

        Subsection #3:
(    7)     Access Table Name = DOOLE.EMPLOYEE  ID = 2,5
            |  #Columns = 3
            |  Relation Scan
            |  |  Prefetch: Eligible
            |  Lock Intents
            |  |  Table: Intent Share
            |  |  Row  : Next Key Share
(    7)     |  Insert Into Asynchronous Table Queue  ID = q3
            |  |  Hash to Specific Node
            |  |  Rows Can Overflow to Temporary Tables
(    6)     Insert Into Asynchronous Table Queue Completion  ID = q3

End of section


Optimizer Plan:

                RETURN
                (   1)
                  |
                 BTQ
                (   2)
                  |
                HSJOIN
                (   3)
               /      \
         HSJOIN          DTQ
         (   4)         (   8)
        /      \          |
    TBSCAN      DTQ     TBSCAN
    (   5)     (   6)   (   9)
      |          |       |
  Table:       TBSCAN   Table:
  DOOLE        (   7)   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。

- **コーディネーター・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 をスキャンし、ハッシュ結合を使って表キュー q3 からのデータと結合します。 2 番目のハッシュ結合は、DEPARTMENT 表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 を使ってコーディネーター・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、PROJECT 表をスキャンして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、EMPLOYEE 表をスキャンして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: パーティション間並列処理とパーティション内並列処理による複数パーティションのプラン: この例は、最初の例と同じ 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・パーティション (4-way SMP マシン上にある) から成るパーティション・データベースでコンパイルされたものです。

```
******************** PACKAGE ************************************

Package Name = "DOOLE"."EXAMPLE"  Version = ""

 Prep Date = 2002/01/04
 Prep Time = 14:58:35

 Bind Timestamp = 2002-01-04-14.58.35.169555

 Isolation Level          = Cursor Stability
 Blocking                 = Block Unambiguous Cursors
 Query Optimization Class = 5

 Partition Parallel       = Yes
 Intra-Partition Parallel = Yes (Bind Degree = 4)

 SQL Path                 = "SYSIBM", "SYSFUN", "SYSPROC", "DOOLE"

-------------------- SECTION -------------------------------------
Section = 1


SQL Statement:
  DECLARE EMPCUR CURSOR
  FOR
     SELECT e.lastname, e.job, d.deptname, d.location, p.projname
     FROM employee AS e, department AS d, project AS p
     WHERE e.workdept = d.deptno AND e.workdept = p.deptno

Intra-Partition Parallelism Degree = 4

Estimated Cost        = 145.198898
Estimated Cardinality = 474.720032

        Coordinator Subsection:
(-----)    Distribute Subsection #2
           |  Broadcast to Node List
           |  |  Nodes = 10, 33, 55
(-----)    Distribute Subsection #3
           |  Broadcast to Node List
           |  |  Nodes = 10, 33, 55
(-----)    Distribute Subsection #1
           |  Broadcast to Node List
           |  |  Nodes = 10, 33, 55
(    2)    Access Table Queue  ID = q1  #Columns = 5
(    1)    Return Data to Application
           |  #Columns = 5

        Subsection #1:
(    3)    Process Using 4 Subagents
(   10)    |  Access Table Queue  ID = q3  #Columns = 2
(    4)    |  Hash Join
           |  |  Estimated Build Size: 5737
           |  |  Estimated Probe Size: 8015
(    7)    |  |  Access Table Queue  ID = q5  #Columns = 3
(    5)    |  |  Hash Join
           |  |  |  Estimated Build Size: 5333
           |  |  |  Estimated Probe Size: 6421
(    6)    |  |  |  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
(    6)     |  |  |  |  Process Probe Table for Hash Join
(    3)     |  Insert Into Asynchronous Local Table Queue  ID = q2
(    3)    Access Local Table Queue  ID = q2  #Columns = 5
(    2)    Insert Into Asynchronous Table Queue  ID = q1
            |  Broadcast to Coordinator Node
            |  Rows Can Overflow to Temporary Table

        Subsection #2:
(   11)    Process Using 4 Subagents
(   12)    |  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
(   11)    |  Insert Into Asynchronous Local Table Queue  ID = q4
(   11)    Access Local Table Queue  ID = q4  #Columns = 2
(   10)    Insert Into Asynchronous Table Queue  ID = q3
           |  Hash to Specific Node
           |  Rows Can Overflow to Temporary Tables

        Subsection #3:
(    8)    Process Using 4 Subagents
(    9)    |  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
(    8)    |  Insert Into Asynchronous Local Table Queue  ID = q6
(    8)    Access Local Table Queue  ID = q6  #Columns = 3
(    7)    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)
                |
              HSJOIN
              (   4)
              /     \
          HSJOIN     DTQ
          (   5)    (  10)
          /     \     |
      TBSCAN    DTQ   LTQ
      (   6)   (   7) (  11)
        |        |     |
```

```
 Table:       LTQ      TBSCAN
 DOOLE       (   8)    (  12)
 DEPARTMENT    |         |
             TBSCAN    Table:
             (   9)    DOOLE
               |       PROJECT
            Table:
            DOOLE
            EMPLOYEE
```

このプランは、3 番目の例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 : フェデレーテッド・データベースのプラン: この例は、最初の例と同じ SQL ステートメントを示していますが、この照会は、フェデレーテッド・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 はフェデレーテッド・サーバーにあります。

```
******************** PACKAGE ****************************************

Package Name = "DOOLE"."EXAMPLE"  Version = ""

 Prep Date = 2002/01/11
 Prep Time = 13:52:48

 Bind Timestamp = 2002-01-11-13.52.48.325413

 Isolation Level          = Cursor Stability
 Blocking                 = Block Unambiguous Cursors
 Query Optimization Class = 5

 Partition Parallel       = No
 Intra-Partition Parallel = No

 SQL Path                 = "SYSIBM", "SYSFUN", "SYSPROC", "DOOLE"

-------------------- SECTION ---------------------------------------
Section = 1


SQL Statement:
  DECLARE EMPCUR CURSOR
  FOR
     SELECT e.lastname, e.job, d.deptname, d.location, p.projname
     FROM employee AS e, department AS d, project AS p
     WHERE e.workdept = d.deptno AND e.workdept = p.deptno


Estimated Cost        = 1804.625000
Estimated Cardinality = 112000.000000

(    7) Ship Distributed Subquery #2
        |  #Columns = 2
(    2) Hash Join
        |  Estimated Build Size: 48444
        |  Estimated Probe Size: 232571
(    6) |  Access Table Name = DOOLE.EMPLOYEE  ID = 2,5
        |  |  #Columns = 3
        |  |  Relation Scan
        |  |  |  Prefetch: Eligible
```

```
        |  |  Lock Intents
        |  |  |  Table: Intent Share
        |  |  |  Row  : Next Key Share
(    6) |  |  Process Build Table for Hash Join
(    3) |  Hash Join
        |  |  Estimated Build Size: 7111
        |  |  Estimated Probe Size: 64606
(    4) |  |  Ship Distributed Subquery #1
        |  |  |  #Columns = 3
(    1) Return Data to Application
        |  #Columns = 5

Distributed Substatement #1:
(    4) Server: REMOTE  (DB2/UDB 8.1)
        SQL Statement:

             SELECT A0."DEPTNO", A0."DEPTNAME", A0."LOCATION"
             FROM "DOOLE"."DEPARTMENT" A0

        Nicknames Referenced:
           DOOLE.DEPARTMENT  ID = 32768
              Base = DOOLE.DEPARTMENT
        #Output Columns = 3

Distributed Substatement #2:
(    7) Server: REMOTE  (DB2/UDB 8.1)
        SQL Statement:

             SELECT A0."DEPTNO", A0."PROJNAME"
             FROM "DOOLE"."PROJECT" A0

        Nicknames Referenced:
           DOOLE.PROJECT  ID = 32769
               Base = DOOLE.PROJECT
        #Output Columns = 2

End of section

Optimizer Plan:

               RETURN
               (   1)
                 |
               HSJOIN
               (   2)
              /      \
        HSJOIN          SHIP
        (   3)          (   7)
       /      \           |
    SHIP      TBSCAN   Nickname:
   (   4)     (   6)   DOOLE
     |          |      PROJECT
 Nickname:    Table:
 DOOLE        DOOLE
 DEPARTMENT   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、 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。データがフェデレーテッド・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。

例 6: XANDOR および XISCAN 演算子: この例は、XANDOR 演算子が、同じ表 *XISCANTABLE* 上で定義された XML データに関する 2 つの別個の索引 (*IDX1* および *IDX2*) の XISCAN スキャンを結合する方法を示しています。

```
IBM DB2 Database SQL Explain Tool

****************** DYNAMIC ***************************************

=================== STATEMENT ========================================

        Isolation Level          = Cursor Stability
        Blocking                 = Block Unambiguous Cursors
        Query Optimization Class = 5

        Partition Parallel       = No
        Intra-Partition Parallel = No

        SQL Path                 = "SYSIBM", "SYSFUN", "SYSPROC", "ATTALURI"


Query Statement:
  xquery
  for $c in db2-fn:xmlcolumn("XISCANTABLE.XMLCOL ")/a[@x="1" ]/b[@y=
          "2" ] return $c

Section Code Page = 819

Estimated Cost = 192.266113
Estimated Cardinality = 1.800000

(    6) Index ANDing and ORing over XML
          |  Xpath is
          |  |  /child::element(a)[./child::element(b)
          |  |  /attribute::attribute(y)(:Index Search over XML 1:)
          |  |   and ./attribute::attribute(x)(:Index Search over XML 2:)
          |  |  ]
          |  Index Search over XML 1
          |  |  Access Table Name = ATTALURI.XISCANTABLE
          |  |  |  Index Scan over XML:  Name = ATTALURI.IDX1  ID = 6
          |  |  |  |  Physical Index over XML
          |  |  |  |  Index Columns:
          |  |  |  |  |  1: XMLCOL (Ascending)
          |  |  |  #Key Columns = 4
          |  |  |  |  Start Key: Inclusive Value
          |  |  |  |  |  |  1: ?
          |  |  |  |  |  |  2: ?
          |  |  |  |  |  |  3: ?
          |  |  |  |  |  |  4: ?
          |  |  |  |  Stop Key: Inclusive Value
          |  |  |  |  |  |  1: ?
          |  |  |  |  |  |  2: ?
          |  |  |  |  Index-Only Access
          |  |  |  |  Index Prefetch: None
          |  |  |  |  Isolation Level: Uncommitted Read
          |  |  |  |  Lock Intents
          |  |  |  |  |  Table: Intent None
          |  |  |  |  |  Row  : None
          |  |  |  |  StopKey = StartKey
          |  |  |  |  Value Start Key = ?
          |  Index Search over XML 2
          |  |  Access Table Name = ATTALURI.XISCANTABLE
          |  |  |  Index Scan over XML:  Name = ATTALURI.IDX2  ID = 4
          |  |  |  |  Physical Index over XML
          |  |  |  |  Index Columns:
          |  |  |  |  |  1: XMLCOL (Ascending)
          |  |  |  #Key Columns = 4
          |  |  |  |  Start Key: Inclusive Value
          |  |  |  |  |  |  1: ?
          |  |  |  |  |  |  2: ?
```

```
        |  |  |  |  |  |  3: ?
        |  |  |  |  |  |  4: ?
        |  |  |  |  Stop Key: Inclusive Value
        |  |  |  |  |  |  1: ?
        |  |  |  |  |  |  2: ?
        |  |  |  Index-Only Access
        |  |  |  Index Prefetch: None
        |  |  |  Isolation Level: Uncommitted Read
        |  |  |  Lock Intents
        |  |  |  |  Table: Intent None
        |  |  |  |  Row  : None
        |  |  |  StopKey = StartKey
        |  |  |  Value Start Key = ?
(    5) Insert Into Sorted Temp Table  ID = t1
        |  #Columns = 1
        |  #Sort Key Columns = 1
        |  |  Key 1: (Ascending)
        |  Sortheap Allocation Parameters:
        |  |  #Rows     = 2
        |  |  Row Width = 16
        |  Piped
        |  Duplicate Elimination
(    4) List Prefetch Preparation
(    4) |  Access Table Name = ATTALURI.XISCANTABLE  ID = 2,16
        |  |  #Columns = 1
        |  |  Fetch Using Prefetched List
        |  |  |  Prefetch: Eligible
        |  |  Lock Intents
        |  |  |  Table: Intent Share
        |  |  |  Row  : Next Key Share
(    2) Nested Loop Join
        |  Piped Inner
(    9) |  XML Doc Navigation
        |  |  Navigator is
        |  |  |  /fn:root($CONTEXT_NODE$())/child::element(a)(:#Xpath Predicates = 1:)
        |  |  |  [./child::element(b)(:Output nodeSeqRef :)
        |  |  |  (:#Xpath Predicates = 1:)
        |  |  |  /attribute::attribute(y) and
        |  |  |  ./attribute::attribute(x)]
(    1) Iterate over XML sequence for Xquery bindout
(    1) Return Data to Application
        |  #Columns = 1

End of section


Optimizer Plan:

                                    RETURN
                                    (   1)
                                      |
                                    NLJOIN
                                    (   2)
                                   /      \
                                FETCH    XSCAN
                               (----)    (   9)
                               /            \
                          RIDSCN            Table:
                          (   4)               ATTALURI
                            |                  XISCANTABLE
                           SORT
                          (   5)
                            |
                         XANDOR
                         (   6)
                 /---------/     \--------\
            XISCAN                       XISCAN
            (   6)                       (   6)
           /       \                    /       \
```

```
   Index:         Table:           Index:        Table:
      ATTALURI       ATTALURI         ATTALURI      ATTALURI
      IDX1           XISCANTABLE      IDX2          XISCANTABLE
```

例 7: XSCAN 演算子: この例は、XSCAN 演算子がアクセス・プランに現れる方法を示しています。この演算子は、ネスト・ループ結合演算子 (NLJOIN) により渡されるノード参照を処理します。アクセス・プランへの直接入力では示されません。

```
IBM DB2 Database SQL Explain Tool

******************** DYNAMIC ***************************************

=================== STATEMENT ============================================

        Isolation Level          = Cursor Stability
        Blocking                 = Block Unambiguous Cursors
        Query Optimization Class = 5

        Partition Parallel       = No
        Intra-Partition Parallel = No

        SQL Path                 = "SYSIBM", "SYSFUN", "SYSPROC", "ATTALURI"


Query Statement:
  xquery
  for $b in db2-fn:xmlcolumn("XISCANTABLE.XMLCOL" )//book[position()<=
          2] return $b

Section Code Page = 819

Estimated Cost = 779592.625000
Estimated Cardinality = 540000000.000000

(    4) Access Table Name = ATTALURI.XISCANTABLE  ID = 2,16
         |  #Columns = 1
         |  Relation Scan
         |  |  Prefetch: Eligible
         |  Lock Intents
         |  |  Table: Intent Share
         |  |  Row  : Next Key Share
(    3) Nested Loop Join
         |  Piped Inner
(    6)  |  XML Doc Navigation
         |  |  Navigator is
         |  |  |  /fn:root($CONTEXT_NODE$())/descendant-or-self::node()(:Output nodeSeqRef :)
         |  |  |
(    5)  |  Nested Loop Join
         |  |  Piped Inner
(   11)  |  |  XML Doc Navigation
         |  |  |  Navigator is
         |  |  |  |  /fn:root($CONTEXT_NODE$())/child::element(book)(:Output nodeSeqRef :)
         |  |  |  |
(   10)  |  |  Aggregation
         |  |  |  Column Function(s)
(    9)  |  |  Nested Loop Join
         |  |  |  Piped Inner
(   12)  |  |  |  Unnest input XML sequence into stream of items with item number
(    8)  |  |  Nested Loop Join
         |  |  |  Piped Inner
(   14)  |  |  |  Table Constructor
         |  |  |  |  1-Row(s)
(    2) Nested Loop Join
         |  Piped Inner
(   16)  |  Unnest input XML sequence into stream of items
```

```
(    1) Iterate over XML sequence for Xquery bindout
(    1) Return Data to Application
       |  #Columns = 1

End of section

Optimizer Plan:

               RETURN
               (   1)
                 |
               NLJOIN
               (   2)
             /-/      \-----\
         NLJOIN             *
         (   3)             *
       /       \            |
    TBSCAN      NLJOIN      TBSCAN
    (   4)      (   5)      (  16)
     |          /     \       |
   Table:     XSCAN   *      TFunc:
   ATTALURI   (   6)  *      SYSIBM
   XISCANTABLE        |      GENROW
                    NLJOIN
                    (   8)
                   /      \
               NLJOIN     FILTER
               (   9)     (  13)
              /      \       |
          GRPBY   TBSCAN  TBSCAN
          (  10)  (  12)  (  14)
            |       |       |
          XSCAN   TFunc:  TFunc:
          (  11)  SYSIBM  SYSIBM
                  GENROW  GENROW
```

例 8: XISCAN 演算子: この例は、表 *XISCANTABLE* に定義された XML データに対する索引 *IDX1* を XISCAN 演算子がスキャンする方法を示しています。

```
IBM DB2 Database SQL Explain Tool

****************** DYNAMIC ***************************************

=================== STATEMENT ============================================

        Isolation Level          = Cursor Stability
        Blocking                 = Block Unambiguous Cursors
        Query Optimization Class = 5

        Partition Parallel       = No
        Intra-Partition Parallel = No

        SQL Path                 = "SYSIBM", "SYSFUN", "SYSPROC", "ATTALURI"


Query Statement:
  xquery
  for $c in db2-fn:xmlcolumn("XISCANTABLE.XMLCOL ")/a[@x="1" ] return
          $c


Section Code Page = 819

Estimated Cost = 1666.833862
Estimated Cardinality = 18.000000

(    6) Access Table Name = ATTALURI.XISCANTABLE
```

```
        |  Index Scan over XML:  Name = ATTALURI.IDX1  ID = 4
        |  |  Physical Index over XML
        |  |  Index Columns:
        |  |  |  1: XMLCOL (Ascending)
        |  #Key Columns = 2
        |  |  Start Key: Inclusive Value
        |  |  |  |  1: ?
        |  |  |  |  2: ?
        |  |  Stop Key: Inclusive Value
        |  |  |  |  1: ?
        |  |  |  |  2: ?
        |  Index-Only Access
        |  Index Prefetch: None
        |  Isolation Level: Uncommitted Read
        |  Lock Intents
        |  |  Table: Intent None
        |  |  Row  : None
        |  StopKey = StartKey
        |  Value Start Key = ?
        |  Xpath is
        |  |  /child::element(a)/attribute::attribute(x)
(    5) Insert Into Sorted Temp Table  ID = t1
        |  #Columns = 1
        |  #Sort Key Columns = 1
        |  |  Key 1: (Ascending)
        |  Sortheap Allocation Parameters:
        |  |  #Rows     = 18
        |  |  Row Width = 16
        |  Piped
        |  Duplicate Elimination
(    4) List Prefetch Preparation
(    4) |  Access Table Name = ATTALURI.XISCANTABLE  ID = 2,16
        |  |  #Columns = 1
        |  |  Fetch Using Prefetched List
        |  |  |  Prefetch: Eligible
        |  |  Lock Intents
        |  |  |  Table: Intent Share
        |  |  |  Row  : Next Key Share
(    2) Nested Loop Join
        |  Piped Inner
(    7) |  XML Doc Navigation
        |  |  Navigator is
        |  |  |  /fn:root($CONTEXT_NODE$())/child::element(a)(:Output nodeSeqRef :)
        |  |  |  (:#Xpath Predicates = 1:)
        |  |  |  /attribute::attribute(x)
(    1) Iterate over XML sequence for Xquery bindout
(    1) Return Data to Application
        |  #Columns = 1

End of section


Optimizer Plan:

                    RETURN
                    (   1)
                      |
                    NLJOIN
                    (   2)
                   /      \
                FETCH     XSCAN
                (----)    (   7)
               /      \
           RIDSCN     Table:
           (   4)     ATTALURI
             |        XISCANTABLE
           SORT
```

```
        (   5)
          |
        XISCAN
        (   6)
       /      \
Index:        Table:
ATTALURI      ATTALURI
IDX1          XISCANTABLE
```

Explain 表および Explain 情報の編成

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 または XQuery 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。1 つの Explain インスタンス内でキャプチャーされた Explain 情報には、コンパイル環境ならびにコンパイルされる SQL または XQuery 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。 たとえば、Explain インスタンスは、以下のいずれかから成り立っています。

- 静的照会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 または XQuery ステートメント。 SQL ステートメント (XML データを照会するステートメントを含む) では、CALL、コンパウンド SQL (動的)、DELETE、INSERT、MERGE、REFRESH、SELECT、SET INTEGRITY、SELECT INTO、UPDATE、VALUES、および VALUES INTO ステートメントに関する Explain 情報をキャプチャーすることができます。 XQuery ステートメントでは、 XQUERY db2-fn:xmlcolumn および XQUERY db2-fn:sqlquery ステートメントに関する Explain 情報を取得することができます。

  注: REFRESH TABLE および SET INTEGRITY ステートメントは、静的にはコンパイルされず、動的にのみコンパイルされます。
- 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)

Explain 表の情報は、アクセス・プラン内の演算子とデータ・オブジェクトとの間の関連を反映します。次の図は、これらの表の間の関連を示します。

Explain 情報は、以下の表に保管されます。

表 65. Explain データを保管するリレーショナル表

表名	説明
EXPLAIN_ARGUMENT	個々の演算子に固有の特性がある場合、それを示します。
EXPLAIN_INSTANCE	すべての Explain 情報用の主コントロール表。 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 または XQuery 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	SQL または XQuery 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	照会コンパイラーが SQL または XQuery 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	特定の演算子によって適用される述部を識別します。

表 65. *Explain* データを保管するリレーショナル表 (続き)

表名	説明
EXPLAIN_STATEMENT	さまざまなレベルの Explain 情報に関する SQL または XQuery 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 または XQuery ステートメントと、アクセス・プランを選択するのにオプティマイザーで使用されるバージョンとが保管されます。 Explain スナップショットが要求されると、照会オプティマイザー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_STREAM	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。
EXPLAIN_DIAGNOSTIC	EXPLAIN_STATEMENT 表で EXPLAIN されたステートメントの特定のインスタンスに対して生成された各診断メッセージの項目を含みます。
EXPLAIN_DIAGNOSTIC_DATA	EXPLAIN_DIAGNOSTIC 表で記録された特定の診断メッセージのメッセージ・トークンを含みます。メッセージ・トークンは、メッセージを生成した SQL ステートメントの実行に固有の追加情報を提供します。
ADVISE_WORKLOAD	データベースへのワークロードを記述できます。 表の各行はワークロードにおける 1 つの SQL または XQuery 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。`db2advis` ツールは、この表を使ってワークロード情報を収集し、保管します。
ADVISE_INSTANCE	`db2advis` の実行に関する情報が入ります。これには開始時刻に関する情報も含まれます。 `db2advis` の実行ごとに 1 行が入ります。
ADVISE_INDEX	推奨索引に関する情報が格納されます。この表のデータは、照会コンパイラー、 `db2advis` ユーティリティー、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。 • 推奨索引を入手します。 • 提案された索引についての入力に基づき索引を評価します。
ADVISE_MQT	CREATE DDL、推奨される各 MQT を定義する照会、 XML 形式での COLSTATS (列情報) などの各 MQT の統計情報、NUMROWS などに加え、各 MQT のサンプリングされた統計を取得するためのサンプリング照会が入ります。
ADVISE_TABLE	推奨される MQT、MDC、およびデータベース・パーティションに関する設計アドバイザーの最終的な推奨値を使った、表作成の DDL を保管します。これは指定したオプションと生成された推奨値に応じて異なります。
ADVISE_PARTITION	`db2advis` によって生成および評価される仮想データベース・パーティションを保管します。

注: 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。 これらを作成するには、 *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。

DB2 バージョン 9.5 では、SYSPROC.SYSINSTALLOBJECTS プロシージャーを使用して、Explain 表を作成、ドロップ、および妥当性検査できるようになりました。

このプロシージャーにより、特定のスキーマと表スペースで Explain 表を作成できます。サンプルは EXPLAIN.DLL ファイル内にあります。

Explain 表は、複数のユーザーに共通にすることができます。ただし、Explain 表は、1 人のユーザーに対して定義して、それぞれの追加ユーザーに対しては、その定義済みの表を指すために同じ名前を使用して、別名を定義することができます。またはその代わりに、Explain 表を SYSTOOLS スキーマ下で定義することもできます。ユーザーのセッション ID (動的 SQL または XQuery ステートメントの場合)、またはステートメント許可 ID (静的 SQL または XQuery ステートメントの場合) の下に他の Explain 表または別名がない場合、Explain 機能のデフォルトは SYSTOOLS スキーマになります。共通の Explain 表を共用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。共通 Explain 表の読み取り許可も、通常は Explain 情報を分析するユーザーに限定するべきです。

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 または XQuery 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情報を記録します。

- 作成時刻
- オブジェクトの統計が最後に収集された時刻
- 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)
- プリフェッチ・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・クラスタリングの程度
- このオブジェクトの索引が使用するリーフ・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 または XQuery ステートメントを実行し、結果をユーザーに戻すために、データに対していくつかの操作を実行することができます。照会コンパイラーが、表スキャン、索引スキャン、ネスト・ループの結合、またはグループ化演算子など、必要な操作を判別します。

Explain 情報は、アクセス・プランで使用される演算子と各演算子に関する情報を示すだけでなく、アクセス・プランの累積効果も示します。

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、情報が収集された演算子に至るまでのコストが表示されます。

- 合計コスト (timeron)
- 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 (timeron)。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)。

timeron は、架空の相対計測単位です。 timeron は、オプティマイザーによって、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、timeron の見積もりコストが決定されるたびに、SQL または XQuery ステートメントの timeron メジャー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
- データの順序付けに使用された列 (オプティマイザーが、この配列を後続の演算子で使用できると判別した場合に行われます)
- 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)

インスタンスの Explain 情報

Explain インスタンス情報は EXPLAIN_INSTANCE 表に保管されます。インスタンス内の各照会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: 以下の項目によって提供される情報を用いると、それぞれの Explain インスタンスを一意的に識別し、照会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 が実行された照会ステートメントが入っているパッケージの名前
- Explain が実行された照会ステートメントが入っているパッケージの SQL スキーマ
- SQL ステートメントが入っていたパッケージのバージョン
- スナップショット情報が収集されたかどうか

環境設定: 照会コンパイラーが照会を最適化したデータベース・マネージャー環境の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。

- 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処理の多重度。

CURRENT DEGREE 特殊レジスター、DEGREE BIND 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処理の多重度を決めることができます。

- 照会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
- 照会のコンパイル時に指定されたカーソルの行ブロッキングのタイプ。
- 照会が実行される分離レベル
- 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。 Explain スナップショットがとられるときに、以下のパラメーターが記録されます。
  - ソート・ヒープ・サイズ (*sortheap*)
  - アクティブ・アプリケーションの平均数 (*avg_appls*)
  - データベース・ヒープ (*dbheap*)
  - ロック・リスト用最大ストレージ (*locklist*)
  - エスカレーション前のロック・リストの最大パーセント (*maxlocks*)
  - CPU 速度 (*cpuspeed*)
  - 通信スピード (*comm_bandwidth*)

ステートメントの識別: それぞれの Explain インスタンスごとに、複数の照会ステートメントが Explain されている場合があります。 Explain インスタンスを一意に識別する情報に加えて、次の情報は個々の照会ステートメントを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、位置指定 DELETE、位置指定 UPDATE、SET INTEGRITY
- SYSCAT.STATEMENTS カタログ・ビューに記録された、照会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 表内の QUERYTAG および QUERYNO フィールドには、Explain 処理の一部として設定されている ID が含まれています。 CLP または CLI セッション中にサブミットされた動的 Explain 照会ステートメントの場合、EXPLAIN MODE または EXPLAIN SNAPSHOT がアクティブになっていると、QUERYTAG が「CLP」か「CLI」に設定されます。この場合、各ステートメントごとに 1 以上ずつ大きくなっている番号のデフォルトが QUERYNO 値になります。その他の動的 Explain 照会ステートメント (CLP、CLI からでないもの、または EXPLAIN 照会ステートメントを使用しないもの) 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、*timeron* という架空の相対メジャー単位で示されます。経過時間の見積もりは、次の理由で提供されません。

- 照会オプティマイザーは経過時間ではなく、リソースの消費のみを見積もる。
- オプティマイザーは、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ありません。アクセス・プランの効率に影響を及ぼさない因数は無視します。実行時の因数の数は経過時間に影響します。これには、次のものが

含まれます。システムのワークロード、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、照会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 つのバージョンは、照会コンパイラーがアプリケーションから受信するコードです。もう 1 つのバージョンは、照会の内部コンパイラー表記からの逆変換です。この変換は他の照会ステートメントに似ているように見えますが、必ずしも正しい照会言語構文に従っているわけでも、内部表記の実際の内容全体を反映しているわけでもありません。この変換は、単に、 SQL および XQuery オプティマイザーがアクセス・プランを選択する元となる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。よりよい最適化のために SQL および XQuery コンパイラーが照会を書き直した方法を理解するには、ユーザー作成のステートメント・テキストを照会ステートメントの内部表記と比較してください。書き直されたステートメントは、トリガーや制約などのステートメントに影響を及ぼす、環境内の他のエレメントも示します。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。

$C*n* 派生列の名前。*n* は整数値を表します。

$CONSTRAINT$
コンパイル中に元の照会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.T*n*
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 および XQuery 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 および XQuery コンパイラーが追加した述部があっても、汎用には使用できない述部があることを示すタグ。内部述部は、トリガーおよび制約のために元の照会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。

$INTERNAL_XPATH$
単一の入力のアノテーションを付けられた XPath パターンをパラメーターとしてとり、パターンに一致する 1 つ以上の列のある表を戻す内部表関数を示します。

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照会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。

$WITH_CONTEXT$(...)
元の照会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、照会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。

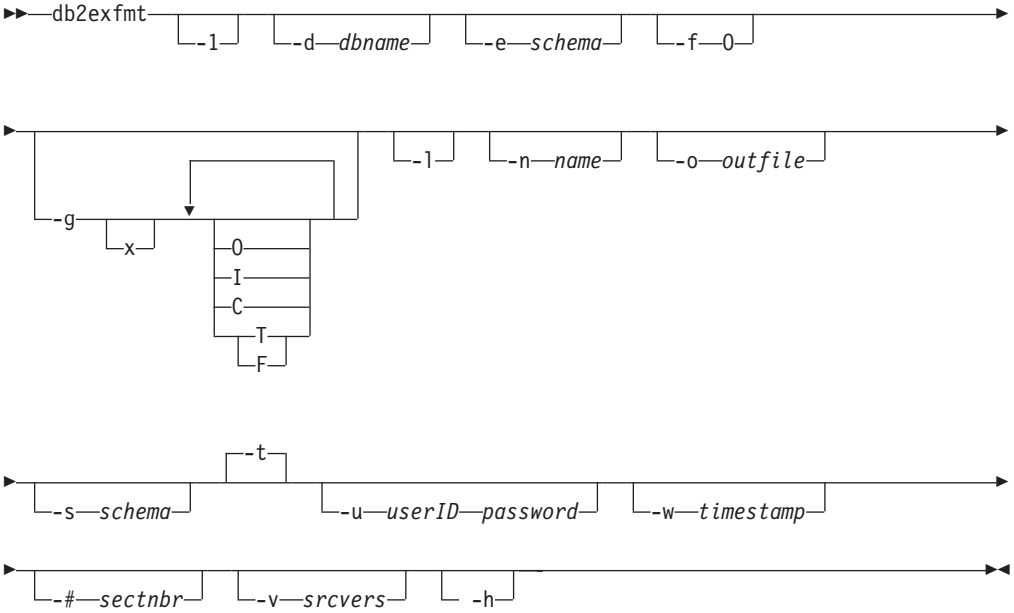
db2exfmt - Explain 表フォーマット

EXPLAIN 表の内容をフォーマットするには、db2exfmt ツールを使用します。このツールは、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にあります。 EXPLAIN スナップショットが使用可能であれば、このツールは EXPLAIN スナップショットの統計を使用します。

許可

このツールを使用するには、フォーマットする Explain 表に対する読み取りアクセスが必要です。

コマンド構文



コマンド・パラメーター

db2exfmt
オプションを指定しない場合、コマンドは対話モードに入り、ユーザーは入力するよう促されます。

-1
デフォルト `-e % -n % -s % -v % -w -1 -# 0` を使用します

Explain スキーマが提供されない場合、環境変数 $USER または $USERNAME の内容がデフォルトとして使用されます。この変数が見つからない場合は、ユーザーが Explain スキーマを提供するよう求められます。

-d *dbname*
パッケージを含むデータベースの名前。

-e *schema*
Explain 表 SQL スキーマ。

-f
フォーマット・フラグ。このリリースでは、値 O (演算子サマリー) だけがサポートされています。

-g グラフ・プラン。

x オプションを OFF にします (デフォルトは ON です)。

-g だけを指定した場合は、グラフが生成され、その後にすべての表に関するフォーマット済みの情報が生成されます。それ以外の場合は、以下の有効な値を任意に組み合わせて指定できます。

O グラフだけを生成します。表の内容はフォーマットしません。

T グラフ内の各演算子の下に合計コストを組み込みます。

F グラフ内の最初のタプルのコストを組み込みます。

I グラフ内の各演算子の下に I/O コストを組み込みます。

C グラフ内の各演算子の予期出力カーディナリティー (タプル数) を組み込みます。

これらのオプションを任意に組み合わせることができますが、相互に排他的な F と T は例外です。

-l パッケージ名の処理時に大文字小文字を区別します。

-n *name*
Explain 要求のソース名 (SOURCE_NAME)。

-s *schema*
Explain 要求のソースの SQL スキーマまたは修飾子 (SOURCE_SCHEMA)。

-o *outfile*
出力ファイル名。

-t 出力を端末に送信します。

-u *userID password*
データベースに接続時に、指定のユーザー ID とパスワードを使用します。

ユーザー ID とパスワードはいずれも、命名規則に従った有効な値でなければならず、データベースによって認識される値でなければなりません。

-w *timestamp*
Explain タイム・スタンプ。 -1 を指定すれば、最新の Explain 要求を取得できます。

-# *sectnbr*
ソース内のセクション番号。すべてのセクションを要求するには、ゼロを指定します。

-v *srcvers*
Explain 要求のソースのソース・バージョン (デフォルトは %)

-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、他のすべてのオプションは無視され、ヘルプ情報だけが表示されます。

使用上の注意

パラメーター値を指定しなかった場合や、指定内容が完全でない場合は、値の入力を促すプロンプトが表示されます。ただし、-h オプションと -l オプションは除きます。

Explain 表 SQL スキーマを指定しない場合は、環境変数 USER の値がデフォルトとして使用されます。この変数が見つからない場合は、Explain 表 SQL スキーマの指定を促すプロンプトが表示されます。

ソース名、ソース SQL スキーマ、Explain タイム・スタンプは、LIKE 述部形式で指定できます。この場合、パーセント記号 (%) と下線 (_) をパターン・マッチング文字として使用して、1 つの呼び出しで複数のソースを選択できます。 EXPLAIN された最新のステートメントの場合は、 Explain タイム・スタンプを -1 と指定できます。

-o をファイル名なしで指定し、-t を指定しない場合は、ファイル名の入力を促すプロンプトが表示されます (デフォルト名は db2exfmt.out)。 -o も -t も指定しない場合は、ファイル名の入力を促すプロンプトが表示されます (デフォルト・オプションは端末出力です)。 -o と -t の両方を指定した場合は、出力が端末に送信されます。

EXPLAIN スナップショットが使用可能であれば、db2exfmt コマンドは EXPLAIN スナップショットの統計を表示します。使用可能でない場合、db2exfmt は、EXPLAIN_OBJECT 表に保管された統計と、システム・カタログから直接取得されたいくつかの統計を表示します。

以下は EXPLAIN スナップショットの例です。

```
db2 explain plan with snapshot for query
db2exfmt
```

または、

```
db2 set current explain mode yes
db2 set current explain snapshot yes
run the query
db2exfmt
```

照会アクセス・プランの最適化

最適化クラス

SQL または XQuery 照会をコンパイルするときは、オプティマイザーによる、その照会のための最も効率的なアクセス・プランの選択方法を決定する最適化クラスを指定できます。最適化クラスは、照会のコンパイルで考慮される最適化の方針の数とタイプによって区別されます。個別に最適化手法を指定して照会の実行時のパフォーマンスを向上することはできますが、指定する最適化手法が多いほど、照会のコンパイルに要する時間とシステム・リソースは多くなります。

SQL または XQuery 照会をコンパイルするときは、以下のいずれかの最適化クラスを指定できます。

0- このクラスは、オプティマイザー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。この最適化クラスには、次の特性があります。

- オプティマイザーは、非均一分散統計を考慮しません。
- 基本照会書き直し規則のみを適用します。
- 貪欲型結合列挙を行います。
- ネストされたループ結合および索引スキャン・アクセス方式だけを使用可能にします。
- リスト・プリフェッチを、生成されたアクセス方式で使用されないようにします。
- スター型結合方式は考慮に入れません。

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 または XQuery 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- この最適化クラスには、次の特性があります。

- オプティマイザーは、非均一分散統計を考慮しません。
- 照会書き直し規則のサブセットのみを適用します。
- 貪欲型結合列挙を行います。
- リスト・プリフェッチを、生成されたアクセス方式で使用されないようにします。

最適化クラス 1 は、マージ・スキャン結合と表スキャン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、オプティマイザーに指示します。この最適化クラスには、次の特性があります。

- 使用可能な統計すべて (頻度と変位値の両方の非均一分散統計を含む) を使用します。
- 非常にまれな場合にしか当てはまらない、計算を多用する規則を除き、すべての照会再作成規則を適用します。これには、マテリアライズ照会表に対する照会の経路指定が含まれます。
- 貪欲型結合列挙を使用します。
- リスト・プリフェッチおよびマテリアライズ照会表の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ます。

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プランを考慮しないので、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。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、クラス 2 をお勧めします。このような環境では、特定の照会が繰り返されることはめ

ったにないので、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- このクラスは、適度の最適化を要求します。このクラスは、DB2 for MVS/ESA™、OS/390、または z/OS 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散統計を使用します。
- 「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します。
- 動的プログラミング結合列挙
  - 複合内部表の限定使用
  - 参照表に関係するスター・スキーマに対するデカルト積の限定使用
- リスト・プリフェッチ、索引 ANDing 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ます。

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。このクラスは、4 つ以上の結合を含む照会のアクセス・プランを改善します。しかし、オプティマイザーがより良いプランを考慮するのに失敗し、デフォルトの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、オプティマイザー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。 この最適化クラスには、次の特性があります。

- 使用可能な統計すべて (頻度と変位値の分散統計を含む) を使用します。
- 照会のマテリアライズ照会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します (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)。
- 動的プログラミング結合列挙
  - 複合内部表の限定使用
  - 参照表に関係するスター・スキーマに対するデカルト積の限定使用
- リスト・プリフェッチ、索引の AND 結合、およびマテリアライズ照会表経の経路指定を含めた広範囲のアクセス方式が考慮されます。

オプティマイザーが、複合動的 SQL または XQuery 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。

照会オプティマイザーが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、照会オプティマイザー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、オプティマイザー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 または XQuery 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、オプティマイザー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
- デカルト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、オプティマイザー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調べるには、 Explain とパフォーマンスの測定値を使用します。

最適化クラスの選択

最適化クラスを設定することには、特に以下の場合に最適化手法を明示的に指定できるという益があります。

- 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を管理する
- コンパイル時のデータベース・サーバー上のメモリー制限に対処する
- PREPARE などの照会のコンパイル時間を削減する

ほとんどのステートメントは、妥当な量のリソースの場合、デフォルトの照会最適化クラスである最適化クラス 5 を使用することによ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、実行される最適化の数によって影響を受けます。

照会最適化クラス 1、2、3、5、および 7 はすべて、汎用に適しています。クラス 0 の使用は、照会のコンパイル時間をさらに削減する必要があり、 SQL および XQuery ステートメントが非常に単純な場合にのみ考慮してください。

ヒント: 長い時間がかかる照会を分析するには、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してください。コンパイルにより多くの時間が必要な場合は、最適化のクラスを下げてください。実行により多くの時間が必要な場合は、より高い最適化クラスの使用を考慮してください。

最適化クラスを選択する際は、以下の一般指針を考慮してください。

- 始めは、デフォルトの照会最適化クラスのクラス 5 を使用してみます。
- デフォルト以外のクラスを使用する場合は、まず 1、2、3 のいずれかを試す。クラス 0、1、および 2 は、貪欲型結合列挙アルゴリズムを使用します。
- 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 を使用します。

- 実行時間が 1 秒未満の非常に短い照会には、低い最適化クラス (0 または 1) を使用します。この種の照会は、以下の特性をもつ傾向があります。
  - 一つか二つの表だけにアクセスします。
  - 単一の行または少しの行だけを取り出す。
  - 完全修飾したユニーク索引を使用します。

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、この種の照会の良い例です。

- 30 秒を上回る実行時間の長い照会には、高い最適化クラス (3、5、または 7) を使用します。
- クラス 3 以上は、動的プログラミング結合列挙アルゴリズムを使用します。このアルゴリズムではより多くの代替プランを考慮に入れようとするので、特に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。
- 最適化クラス 9 は、照会に対する特別な最適化要件がある場合にのみ使用します。

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は、より高度の最適化クラスを使用してください。

- 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 などのセット演算子が多い
- 修飾行が多い
- GROUP BY および HAVING 操作
- ネストした表式
- ビューの数が多い

完全に正規化されたデータベースに対する意思決定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。

照会生成プログラムによって生成された SQL および XQuery ステートメントには、もっと高度な照会最適化クラスを使用してください。多くの照会生成プログラムは非効率な照会を生成します。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・プランを選択しなければなりません。照会最適化クラス 2 以上を使用すると、このような SQL および XQuery 照会でも改善することができます。

注: フェデレーテッド・データベースの照会では、リモート・オプティマイザーに最適化クラスは適用されません。

最適化クラスの設定

最適化レベルを指定するときは、照会が静的または動的 SQL および XQuery ステートメントを使用しているか、また、同一の動的照会が繰り返し実行されるかどうかを考慮してください。静的 SQL および XQuery ステートメントの場合、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 および XQuery ステートメント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合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 または XQuery 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。このステートメントには、静的 SQL および XQuery ステートメントと同じ最適化レベルを使用することができます。

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、その確証がないか、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。

照会最適化クラスを指定するには、以下のステップに従ってください。

1. 以下のように、非公式に、あるいは正式なテストでパフォーマンス要因を分析します。
   - **動的**照会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。平均実行時間を見積もるには、以下の公式を使用します。

     ```
     コンパイル時間 + すべての反復の実行時間の合計
     ----------------------------------------------------
                       反復回数
     ```

     この公式で、**反復回数**は照会ステートメントをコンパイルするたびに照会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: 初期コンパイルの後、動的 SQL および XQuery ステートメントは、環境の変化に伴って再コンパイルが必要になると、再コンパイルされます。照会ステートメントがキャッシュされた後で環境が変化しなければ、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、それを再度コンパイルする必要はありません。
   - **静的** SQL および XQuery ステートメントの場合は、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。

     静的 SQL および XQuery ステートメントのコンパイル時間にも関心があるとしても、ステートメントのコンパイル時間と実行時間の合計を、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、静的照会ステートメントは 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。
2. 最適化クラスを以下のとおりに指定します。

- **動的** SQL および XQuery ステートメントでは、 SQL ステートメント SET で設定される CURRENT QUERY OPTIMIZATION 特殊レジスターによって指定された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 または XQuery 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。

  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照会のデフォルト値は、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。クラス 5 はこのパラメーターのデフォルト値です。 BIND オプションおよび特殊レジスターのデフォルト値も、*dft_queryopt* データベース構成パラメーターから読み取られます。
- **静的** SQL および XQuery 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照会ステートメントに使用されます。この静的 SQL および XQuery ステートメントの最適化クラスを変更するには、BIND コマンドを使用します。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。

オプティマイザー・プロファイルとガイドラインの概要

DB2 オプティマイザーは、業界でも最高の性能を備えたコスト・ベースのオプティマイザーの 1 つです。とはいえ、まれにこのオプティマイザーも、必ずしも最適とは言えない実行プランを選択してしまうことがあります。DBA はデータベースに精通しているため、db2advis、RUNSTATS、db2expln、および最適化クラス設定などの機能を使用して、データベースのパフォーマンスをより高めるためのオプティマイザーの調整に役立てることができます。しかし、すべてのチューニング・オプションを使い尽くしても期待する結果が得られない場合には、DB2 オプティマイザーに明示的な最適化ガイドラインを提供できます。

最適化プロファイルは、1 つ以上の SQL ステートメントについての最適化ガイドラインを含めることのできる XML 文書です。各 SQL ステートメントとそれに関連付ける最適化ガイドラインとの間の対応は、SQL テキストおよび SQL ステートメントを明確に識別するために必要な他の関連する情報を使用して確立されます。411 ページの図 31 は、最適化プロファイルを使用してどのように最適化ガイドラインを DB2 オプティマイザーに渡すことができるのかを図で示しています。

```
<?xml version="1.0" encoding="UTF-8">

<OPTPROFILE VERSION="9.1.0.0">
<STMTPROFILE ID="Guidelines for TPCD Q9">
  <STMTKEY SCHEMA="TPCD">
    SELECT S.S_NAME, S.S_ADDRESS, S.S_PHONE, S.S_COMMENT
    FROM PARTS P, SUPPLIERS S, PARTSUPP PS
    WHERE P_PARTKEY = PS.PS_PARTKEY AND S.S_SUPPKEY = PS.PS_SUPPKEY AND P.P_SIZE = 39 AND P.P_TYPE = 'BRASS' AND
          S.S_NATION = 'MOROCCO' AND S.S_NATION IN ('MOROCCO', 'SPAIN') AND
          PS.PS_SUPPLYCOST = (SELECT MIN(PS1.PS_SUPPLYCOST)
                              FROM PARTSUPP PS1, SUPPLIERS S1
                              WHERE P.P_PARTKEY = PS1.PS_PARTKEY AND S1.S_SUPPKEY = PS1.PS_SUPPKEY AND
                              S1.S_NATION = S.S_NATION))
  </STMTKEY>
  <OPTGUIDELINES> <IXSCAN TABLE="S" INDEX="I_SUPPKEY"/> </OPTGUIDELINES>
</STMTPROFILE>
</OPTPROFILE>
```

図 31. 最適化プロファイルを使用したガイドラインの引き渡し

各 STMTPROFILE エレメントは、1 つのアプリケーション・ステートメントに対して一連の最適化ガイドラインを提供します。ターゲットとなるステートメントは、STMTKEY サブエレメントによって識別されます。次いで、最適化プロファイルはスキーマ修飾名を付けられて、データベースに挿入されます。挿入された最適化プロファイルは、BIND コマンドや PREPARE コマンドでこの修飾名が指定されたときに、ステートメントの最適化に対して施行されます。

最適化プロファイルを使用すると、アプリケーションやデータベースの構成を変更することなく、オプティマイザーに最適化ガイドラインを渡すことができます。行うことは、ただ単純な XML 文書を作成し、それをデータベースに挿入して、BIND コマンドや PREPARE コマンドでその最適化プロファイルの名前を指定するだけです。オプティマイザーにより最適化ガイドラインと該当するステートメントが、自動的にマッチングされます。

最適化ガイドラインは、包括的なものにする必要はありませんが、希望する実行プランをターゲットにしたものでなければなりません。DB2 オプティマイザーは、ガイドラインがあっても、他の候補となるアクセス・プランを選択肢に含んだまま、既存のコスト・ベースのメソッドを使用して機能します。最適化ガイドラインは、特定の表参照をターゲットにしていれば、通常の最適化パラメーターの設定をオーバーライドできません。つまり、表 A と B の間のマージ結合を指定する最適化ガイドラインであれば、最適化クラス 0 では無効です。

オプティマイザーは、無効な最適化ガイドラインや適用できない最適化ガイドラインは無視します。無視された最適化ガイドラインがある場合、実行プランは生成され、理由コード 13 で SQL0437W の警告が戻されます。その後で、最適化ガイドラインの処理に関する詳細な診断情報は、EXPLAIN ステートメントを使用して入手できます。

最適化ガイドライン

最適化ガイドラインのタイプと処理の概要: DB2 オプティマイザーは、ステートメントを 2 つのフェーズで最適化します。最適化ステートメントは、*照会書き直し最適化フェーズ* によって決定されます。このフェーズでは、元のステートメントが、プラン最適化フェーズでより容易に最適化できる、意味的に同等のステートメントにトランスフォームされます。*プラン最適化フェーズ*では、数多くの選択肢を列挙

して実行コストの見積もりが最小になるものを選択することによって、*最適化ステートメント*に最適なアクセス方式、結合方式、および結合順序が判別されます。

2 つの最適化フェーズで考慮される照会のトランスフォーメーション、アクセス方式、結合方式、結合順序、および他の最適化のための選択肢は、CURRENT QUERY OPTIMIZATION (特殊レジスター)、REOPT (バインド・オプション)、および *DB2_REDUCED_OPTIMIZATION* (レジストリー変数) などの様々な DB2 パラメーターによって制御されます。ステートメントの最適化で考慮される、一連の最適化のための選択肢は、検索スペースと呼ばれます。

以下のタイプのステートメント最適化ガイドラインがサポートされています。

- 一般最適化ガイドライン
- 照会書き直しガイドライン
- プラン最適化ガイドライン

最適化ガイドラインは、特定の順序で適用されます。一般最適化ガイドラインは、検索スペースに作用するため、最初に適用されます。照会書き直しガイドラインは、プラン最適化フェーズで最適化されたステートメントに作用するため、次に適用されます。最後にプラン最適化ガイドラインが適用されます。

一般最適化ガイドライン を使用すると、一般の最適化パラメーターの設定に作用することができます。*照会書き直しガイドライン* を使用すると、照会書き直し最適化フェーズで考慮されるトランスフォーメーションに作用することができます。*プラン最適化ガイドライン* を使用すると、プラン最適化フェーズで考慮されるアクセス方式、結合方式、および結合順序に作用することができます。

一般最適化ガイドライン: *一般最適化ガイドライン* を使用すると、一般の最適化パラメーターを設定できます。これらのガイドラインは、それぞれステートメント・レベルの有効範囲を持ちます。

照会書き直し最適化ガイドライン: *照会書き直しガイドライン* を使用すると、照会書き直し最適化フェーズで考慮されるトランスフォーメーションに作用することができます。照会書き直し最適化フェーズでは、元のステートメントを意味的に同等の*最適化ステートメント* にトランスフォームします。次いで、プラン最適化フェーズで、最適化ステートメントの最適な実行プランが決定されます。このため、照会書き直し最適化ガイドラインは、プラン最適化ガイドラインの適用可能性に影響を与える可能性があります。

各照会書き直し最適化ガイドラインは、オプティマイザーの照会トランスフォーメーション規則のいずれかに対応します。以下の照会トランスフォーメーション規則は、照会書き直し最適化ガイドラインの影響を受ける可能性があります。

- IN-LIST から結合へ
- 副照会から結合へ
- NOT-EXISTS 副照会からアンチ結合へ
- NOT-IN 副照会からアンチ結合へ

照会書き直し最適化ガイドラインは、常に適用可能なわけではありません。照会書き直し規則は、一度に 1 つ施行されます。つまり、後続の規則の前に施行されるある照会書き直し規則は、その後続の規則に関連付けられている照会書き直し最適化

ガイドラインに作用します。時に、環境の構成がいくつかの書き直し規則の動作に影響を与え、それが特定の規則の照会書き直し最適化ガイドラインの適用可能性に影響を与えます。毎回同じ結果を得るためには、照会書き直し規則を施行する前にいくつかの条件を満たす必要があります。照会書き直しコンポーネントが照会への規則の適用を試みる際に、その規則に関連付けられている条件が満たされていない場合、その規則では照会書き直し最適化ガイドラインが無視されます。照会書き直し最適化ガイドラインが適用できない場合に、そのガイドラインが使用可能化のガイドラインである場合は、理由コード 13 で SQL0437W エラー・メッセージが戻されます。照会書き直し最適化ガイドラインが適用できない場合に、ガイドラインが使用不可能化のガイドラインである場合は、エラー・メッセージは戻されません。この場合、規則は使用不可にされていたものとして扱われるため、照会書き直し規則は適用されません。

照会書き直し最適化ガイドラインは、ステートメント・レベルと述部レベルの 2 つのカテゴリーに分けることができます。ステートメント・レベルのカテゴリーは、すべての照会書き直し最適化ガイドラインでサポートされています。述部レベルのカテゴリーは、INLIST2JOIN でのみサポートされています。ステートメント・レベルの照会書き直し最適化ガイドラインは、照会全体に適用されます。述部レベルの照会書き直し最適化ガイドラインは、特定の述部にのみ適用されます。ステートメント・レベルと述部レベルの両方の照会書き直し最適化ガイドラインが指定された場合は、述部レベルのガイドラインが、その特定の述部に関してのみステートメント・レベルのガイドラインをオーバーライドします。

各照会書き直し最適化ガイドラインは、最適化ガイドライン・スキーマ内で、対応する *書き直し要求エレメント* によって表されます。

プラン最適化ガイドライン: プラン最適化ガイドラインは、ステートメントのアクセス方式、結合方式、結合順序、およびその他の実行プランの詳細が決定されるコスト・ベースの最適化のフェーズで適用されます。プラン最適化ガイドラインでは、実行プランのすべての面を指定する必要はありません。実行プランのうち、指定されなかった面については、コスト・ベースの方法でオプティマイザーによって決定されます。

プラン最適化ガイドラインには、2 つのカテゴリーがあります。

- *accessRequest* - アクセス要求は、ステートメントの表参照を満たすのに適したアクセス方式を指定します。
- *joinRequest* - 結合要求では、結合操作を実行するのに適した結合方式および順序を指定します。結合要求は、その他のアクセス要求または結合要求で構成されます。

アクセス要求の最適化ガイドラインは、オプティマイザーのデータ・アクセス方式 (表スキャン、索引スキャン、リスト・プリフェッチなど) に対応します。結合要求の最適化ガイドラインは、オプティマイザーの結合方式 (ネスト・ループ結合、ハッシュ結合、マージ結合など) に対応します。そのような方式については、「パフォーマンス・ガイド」で説明されています。

最適化ガイドラインでの表参照の形成: 本書において、*表参照* という語は、SQL ステートメント内またはビュー定義内のすべての表、ビュー、表式、または別名を意味します。最適化ガイドラインは、元のステートメントにある直接的な名前が使

用されていても、あるいは*最適化ステートメント* の表参照に関連付けられた固有な相関名が使用されていても、表参照を識別することができます。直接的な名前の連続である拡張名は、ビューに組み込まれている表参照を一意的に識別するのに役立ちます。ステートメント全体のコンテキストで固有ではない直接的な名前や拡張名を識別する最適化ガイドラインは、未確定と見なされ、適用されません。さらに、同一の表参照を識別する複数の最適化ガイドラインが存在する場合は、その表参照を識別するすべての最適化ガイドラインが競合していると見なされ、適用されません。続くセクションでは、最適化ガイドラインの技術的なこれらの特定の面について、さらに詳しく説明します。照会のトランスフォーメーションが行われる可能性があるため、直接的な名前または拡張名が最適化の間もそのまま存在するという保証はありません。そのような場合には、その表参照をターゲットにしているすべてのガイドラインが無視されることになります。

元のステートメントの直接的な名前を使用した表参照の識別

表参照を、その直接的な名前を使用して識別します。直接的な名前は、SQL ステートメント内で表が修飾されるときと同じ要領で指定されます。 SQL ID を指定するときの規則が TABLE 属性の値にも適用されます。

最適化ガイドラインの TABLE 属性の値は、ステートメントの直接的な名前とそれぞれ比較されます。このリリースの DB2 では、1 つの一致だけが許可されます。TABLE 属性値がスキーマ修飾されている場合は、値が、同等の直接的な修飾表名すべてと一致します。TABLE 属性値が未修飾の場合は、値が、同等の相関名または直接的な表名すべてに一致します (そのため、TABLE 属性値は、そのステートメントで有効なデフォルトのスキーマによって暗黙的に修飾されたものと見なされます)。これらの概念については、図 32 のサンプル・ステートメントで図解されています。ステートメントはデフォルトのスキーマ 『Tpcd』 を使用して最適化されるものと想定します。

```
SELECT S_NAME, S_ADDRESS, S_PHONE, S_COMMENT
FROM PARTS, SUPPLIERS, PARTSUPP PS
WHERE P_PARTKEY = PS.PS_PARTKEY AND S.S_SUPPKEY = PS.PS_SUPPKEY AND
   P_SIZE = 39 AND P_TYPE = 'BRASS';
```

図 32. 最適化ガイドラインの TABLE 属性値を使用して、ステートメントの直接的な名前をそれぞれ比較する

ステートメントの表参照を適切に識別する TABLE 属性値には、'"Tpcd".PARTS'、'PARTS'、'Parts' (ID に区切りが入っていないため、これは大文字に変換されます) があります。ステートメントの表参照の識別に失敗する TABLE 属性値には、'"Tpcd2"、SUPPLIERS'、'PARTSUPP' (直接的な名前ではない)、および 'Tpcd.PARTS' (ID Tpcd には区切りが入っていなければならず、そうでないと大文字に変換されてしまいます) があります。

直接的な名前は、元のステートメント、ビュー、SQL 関数、またはトリガーにあるどの表参照をターゲットにする際にも使用できます。

元のステートメントの直接的な名前を使用したビューの表参照の識別

最適化ガイドラインでは、拡張構文を使用してビューに組み込まれた表参照を識別することができます。

拡張構文は、元のステートメント、SQL 関数、またはトリガーのどの表参照をターゲットにする際にも使用できます。

最適化ステートメントの相関名を使用した表参照の識別

最適化ガイドラインでは、*最適化*ステートメント の表参照に関連付けられた固有の相関名を使用して表参照を識別することもできます。最適化ステートメントは、元のステートメントと意味的に同等なバージョンで、最適化の照会書き直しのフェーズで決定されるものです。最適化ステートメントは、EXPLAIN 表から取得できます。最適化ステートメントの表参照の識別には、最適化ガイドラインの TABID 属性が使用されます。

1 つの最適化ガイドラインに TABLE 属性と TABID 属性の両方が指定されている場合は、両方の属性が同じ表参照を識別している必要があります。そうでない場合、その最適化ガイドラインは無視されます。

注: 最適化ステートメントの相関名については、現在のところ、 DB2 の新しいリリースへのアップグレードを行っても変わらないという保証はありません。

未確定表参照

最適化ガイドラインが複数の直接的な名前または拡張名と一致する場合、その最適化ガイドラインは無効であると見なされ、適用されません。

これを解決するためには、固有の相関名を使用するようにビューを書き直すか、TABID 属性を使用することができます。

注: 最適化ステートメントの相関名はすべて固有であるため、TABID フィールドで識別された表参照は未確定になることがありません。

競合する最適化ガイドライン

同じ表参照を複数の最適化ガイドラインで識別することはできません。

複数のガイドラインが同じ表を参照する場合は、最初のガイドラインだけが適用されます。それ以外のガイドラインはすべて無視され、エラー・メッセージが出されます。

述部レベルで使用可能な OPTION を指定する照会書き直しの INLIST2JOIN 書き直し要求のエレメントを複数指定することには制限があります。このような INLIST2JOIN 書き直し要求のエレメントの述部レベルでの指定は、1 つの照会につき 1 つしかできません。

最適化ガイドラインが使用されていることの確認:

オプティマイザーは、毎回、最適化プロファイルまたは SQL 組み込み最適化ガイドライン (ステートメント・プロファイルとしても知られる) で指定された最適化ガイドラインに従うことを試みますが、無効なガイドラインや適用できないガイドラインについてはこれを拒否します。

EXPLAIN 機能を使用するためには、EXPLAIN 表が存在している必要があります。EXPLAIN 表を作成する DDL は EXPLAIN.DDL で、sqllib ディレクトリーの misc サブディレクトリーにあります。

有効な最適化ガイドラインが使用されていることを確認するには、次のようにします。

1. ステートメントで EXPLAIN コマンドを実行します。 最適化ガイドラインが、最適化プロファイルまたは SQL コメントのいずれかを使用するステートメントで施行された場合には、その最適化プロファイルの名前が、EXPLAIN ARGUMENTS 表に RETURN 演算子の引数として表示されます。最適化ガイドラインに含まれる SQL 組み込み最適化ガイドラインまたはステートメント・プロファイルが、現行のステートメントと一致する場合は、ステートメント・プロファイルの名前が RETURN 演算子の引数として表示されます。新しい 2 つの引数値のタイプは、OPT_PROF と STMTPROF です。
2. EXPLAIN されたステートメントの結果を調べてください。 以下の EXPLAIN 表に対する照会を変更して、EXPLAIN_REQUESTER、EXPLAIN_TIME、SOURCE_NAME、SOURCE_VERSION、および QUERYNO の特定の組み合わせで最適化プロファイル名やステートメント・プロファイル名を戻すようにできます。

```
SELECT VARCHAR(B.ARGUMENT_TYPE,   9) as TYPE,
     VARCHAR(B.ARGUMENT_VALUE, 24) as VALUE

FROM   EXPLAIN_STATEMENT A, EXPLAIN_ARGUMENT B

WHERE  A.EXPLAIN_REQUESTER = 'SIMMEN'
  AND  A.EXPLAIN_TIME      = '2003-09-08-16.01.04.108161'
       AND  A.SOURCE_NAME       = 'SQLC2E03'
  AND  A.SOURCE_VERSION    = ''
  AND  A.QUERYNO           = 1

  AND  A.EXPLAIN_REQUESTER = B.EXPLAIN_REQUESTER
  AND  A.EXPLAIN_TIME      = B.EXPLAIN_TIME
  AND  A.SOURCE_NAME       = B.SOURCE_NAME
  AND  A.SOURCE_SCHEMA     = B.SOURCE_SCHEMA
  AND  A.SOURCE_VERSION    = B.SOURCE_VERSION
  AND  A.EXPLAIN_LEVEL     = B.EXPLAIN_LEVEL
  AND  A.STMTNO            = B.STMTNO
  AND  A.SECTNO            = B.SECTNO

  AND  A.EXPLAIN_LEVEL     = 'P'

  AND  (B.ARGUMENT_TYPE  = 'OPT_PROF' OR ARGUMENT_TYPE = 'STMTPROF')
  AND  B.OPERATOR_ID = 1
```

図 33. 照会を使用して、EXPLAIN されたステートメントからの結果を戻す

最適化ガイドラインがアクティブで、EXPLAIN されたステートメントが最適化ガイドラインの STMTKEY エレメントに含まれているステートメントと一致する場合、図 33 にあるのと同様の照会では、 417 ページの図 34 にある出力が生成されます。STMTPROF 引数の値は、STMTPROFILE エレメントの ID 属性と同じです。

```
TYPE      VALUE
--------- -------------------------
OPT_PROF  NEWTON.PROFILE1
STMTPROF  Guidelines for TPCD Q9
```

図 34. EXPLAIN されたステートメントの結果を戻す照会の出力

最適化プロファイル

最適化プロファイルの分析: このセクションでは、最適化プロファイルの内容を紹介します。所定の DB2 リリースに有効な最適化プロファイルの内容は、現行最適化プロファイル・スキーマ (COPS) と呼ばれる XML スキーマによって記述されます。

DB2 の現行リリースの COPS は、424 ページの『現行の最適化プロファイル・スキーマ』にリストされています。

最適化プロファイルにはグローバル・ガイドラインを含めることができます。これは、プロファイルが有効である間、すべての DML ステートメントに適用されるものです。また、最適化プロファイルには、それぞれがパッケージ内の単一 DML ステートメントに適用される特定のガイドラインを含めることもできます。例:

- グローバル最適化ガイドラインを作成できます。これは、現行最適化プロファイルがアクティブである間に発生するすべてのステートメントについて、オプティマイザーがマテリアライズ照会表 Test.SumSales および Test.AvgSales を参照するように要求します。
- ステートメント最適化ガイドラインを作成できます。これは、オプティマイザーがターゲット・ステートメントを見つけるときに、I_SUPPKEY 索引を使用して SUPPLIERS 表にアクセスするように要求します。

したがって、最適化プロファイルには、2 つのタイプのガイドラインを指定できる主なセクションを定義する 2 つのエレメントが含まれます。つまり、1 つのグローバル *OPTGUIDELINES* エレメントと、任意の数の *STMTPROFILE* エレメントです。また、最適化プロファイルには *OPTPROFILE* エレメントを含まれています。これは、メタデータおよび処理ディレクティブを含むセクションを定義します。

418 ページの図 35 は、DB2 バージョン 9.1 に有効な最適化プロファイルの例を示しています。この最適化プロファイルには、グローバル最適化ガイドライン・セクションと 1 つのステートメント・プロファイル・セクションがあります。

```
<?xml version="1.0" encoding="UTF-8"?>
<OPTPROFILE VERSION="9.1.0.0">

  <!--
       Global optimization guidelines section.
       Optional but at most one.
  -->
  <OPTGUIDELINES>
    <MQT NAME="Test.AvgSales"/>
    <MQT NAME="Test.SumSales"/>
  </OPTGUIDELINES>

  <!--
       Statement profile section.
       Zero or more.
  -->
  <STMTPROFILE ID="Guidelines for TPCD Q9">
    <STMTKEY SCHEMA="TPCD">
      <![CDATA[SELECT S.S_NAME, S.S_ADDRESS, S.S_PHONE,
S.S_COMMENT FROM PARTS P, SUPPLIERS S, PARTSUPP PS
WHERE P_PARTKEY = PS.PS_PARTKEY AND S.S_SUPPKEY = PS.PS_SUPPKEY AND P.P_SIZE = 39
AND P.P_TYPE = 'BRASS' AND S.S_NATION = 'MOROCCO' AND S.S_NATION IN ('MOROCCO', 'SPAIN')
AND PS.PS_SUPPLYCOST = (SELECT MIN(PS1.PS_SUPPLYCOST) FROM PARTSUPP PS1, SUPPLIERS S1
WHERE P.P_PARTKEY = PS1.PS_PARTKEY AND S1.S_SUPPKEY = PS1.PS_SUPPKEY AND
S1.S_NATION = S.S_NATION)]]>
    </STMTKEY>
    <OPTGUIDELINES>
      <IXSCAN TABID="Q1" INDEX="I_SUPPKEY"/>
    </OPTGUIDELINES>
  </STMTPROFILE>

</OPTPROFILE>
```

図 35. 有効な最適化プロファイル

OPTPROFILE エレメント

最適化プロファイルは OPTPROFILE エレメントで始まります。 図 35 の例では、このエレメントは、最適化プロファイルのバージョンが 9.1 であることを示すバージョン属性で構成されます。

グローバル最適化ガイドライン・セクション

グローバル最適化ガイドラインは、最適化プロファイルが有効になっているすべてのステートメントに適用される最適化ガイドラインを指定します。グローバル最適化ガイドライン・セクションは、グローバル *OPTGUIDELINES* エレメントによって表されます。 図 35 に示される例では、このセクションには、最適化プロファイルが有効になっているステートメントに応答するためにマテリアライズ照会表 (MQT) Test.AvgSales および Test.SumSales を考慮しなければならないことを指定する、単一のグローバル最適化ガイドラインが含まれます。

ステートメント・プロファイル・セクション

ステートメント・プロファイルは、特定のステートメントに適用される最適化ガイドラインを定義します。最適化プロファイルにはゼロ個以上のステートメント・プロファイルを入れることができます。ステートメント・プロファイル・セクションは、*STMTPROFILE* エレメントによって表されます。 図 35 に示される例では、このセクションには、最適化プロファイルが有効になっている特定のステートメントのガイドラインが含まれます。

各ステートメント・プロファイルには、ステートメント・キーおよびステートメント・レベルの最適化ガイドラインが含まれます。これらは、*STMTKEY* および *OPTGUIDELINES* エレメントによって表されます。

ステートメント・キーは、ステートメント・レベルの最適化ガイドラインが適用されるステートメントを識別します。もう一度、 418 ページの図 35 の例を見ると、*STMTKEY* エレメントにはオリジナル・ステートメント・テキストと、ターゲットのステートメントを明白に識別するために必要なその他の情報が含まれています。

オプティマイザーは、ステートメント・キーを使用して、ステートメント・プロファイルと該当するステートメントを自動的に突き合わせます。この関係により、アプリケーションを変更しなくても、そのアプリケーション内のステートメントの最適化ガイドラインを提供することができます。

ステートメント・プロファイルのステートメント・レベルの最適化ガイドライン・セクションは、*OPTGUIDELINES* エレメントによって表されます。ステートメント・プロファイル内のステートメント・キーのマッチングが成功すると、オプティマイザーはステートメントの最適化時に関連したステートメント最適化ガイドラインを参照します。

ステートメント最適化ガイドライン・セクション

418 ページの図 35 の例では、ステートメント・プロファイル・セクションには、*OPTGUIDELINES* エレメントによって表されるステートメント最適化ガイドライン・セクションが含まれています。

このセクションは、参照されたステートメントに関する必要な照会実行プランの局面を指定します。これは、1 つ以上の *access* および *join* 要求で構成されます。これらの要求は、ステートメント内の表のアクセスおよび結合のための希望する方法を指定します。 418 ページの図 35 のガイドラインには、1 つのアクセス要求が含まれています。これは、ネストされた副選択で参照される SUPPLIERS 表が I_SUPPKEY という名前の索引を使用することを指定します。

418 ページの図 35 では、*indexScan* エレメントは、索引アクセス要求を示します。indexScan エレメントの *tableReference* エレメントは、SUPPLIERS 表がアクセス要求のターゲットになることを指定します。 *index* エレメントは、I_SUPPKEY 索引が使用されることを指定します。

ガイドラインは、希望する照会実行プランの一部を指定することだけを必要とします。オプティマイザーは、そのコスト・ベースのモデルを適用して、プランの残りの部分を選択することができます。

最適化プロファイルの作成:

最適化プロファイルは、現行の最適化プロファイル・スキーマに従って有効にする必要があります。

最適化プロファイルには多くのガイドラインを組み合わせて含めることができるため、この作業では、最適化プロファイルの作成に共通するステップだけを示します。

最適化プロファイルを作成するには、以下のようにします。

1. XML エディターを起動します。可能なら、スキーマ妥当性検査機能を持つものを使用してください。オプティマイザーは XML 妥当性検査を実行しません。
2. 意味のある名前を使用して新規の XML 文書を作成します。適用先のステートメントの有効範囲を記述する名前を指定することができます (例えば、inventory_db.xml)。
3. XML 宣言を文書に追加します。エンコード形式を指定しない場合、UTF-8 が想定されます。可能であれば、UTF-16 エンコード方式で文書を保管してください。このエンコード方式で処理すると、DB2 はより効率的に機能します。

   ```
   <?xml version="1.0" encoding="UTF-16"?>
   ```

4. 最適化プロファイル・セクションを文書に追加します。

   ```
   <OPTPROFILE VERSION="9.1.0.0">
   </OPTPROFILE>
   ```

5. 最適化プロファイル・エレメント内で、必要に応じてグローバル・ガイドラインまたはステートメント・レベル・ガイドラインを作成し、ファイルを保管します。

ステートメント最適化ガイドラインの作成:

以下のステップでは、ステートメント最適化ガイドラインの作成方法について説明します。

ステートメント・ガイドラインを挿入する 419 ページの『最適化プロファイルの作成』。

ステートメント最適化ガイドラインを作成するには、以下のようにします。

1. 他のすべてのチューニング・オプションを使い尽くします。「*パフォーマンス・ガイド*」を参照してください。例:
   a. データ分散統計が RUNSTATS ユーティリティーによって最近更新されたことを確認します。
   b. ワークロードに適切な最適化クラスを設定して、DB2 が実行されていることを確認します。
   c. オプティマイザーに、照会で参照される表にアクセスするための適切な索引があることを確認します。
2. 問題のあるステートメントに対して EXPLAIN 機能を実行し、出力を分析して、ガイドラインが適切であるかどうかを判別します。ステートメント最適化ガイドラインが役に立つと判断する場合には、先に進みます。
3. 以下のような照会を実行して、*オリジナル* ・ステートメントを取得します。

   ```
   SELECT STATEMENT TEXT
     FROM EXPLAIN_STATEMENT
     WHERE  EXPLAIN_LEVEL = '0' AND
       EXPLAIN_REQUESTER = 'SIMMEN' AND
       EXPLAIN_TIME      = '2003-09-08-16.01.04.108161' AND
       SOURCE_NAME       = 'SQLC2E03' AND
       SOURCE_VERSION    = '' AND
       QUERYNO           = 1
   ```

4. 最適化プロファイルを編集し、ステートメント・キーにステートメント・テキストを挿入してステートメント・プロファイルを作成します。例:

```
<STMTPROFILE ID="Guidelines for TPCD Q9">
   <STMTKEY SCHEMA="TPCD"><![CDATA[SELECT S.S_NAME, S.S_ADDRESS, S.S_PHONE,
     S.S_COMMENT
     FROM PARTS P, SUPPLIERS S, PARTSUPP PS
     WHERE P_PARTKEY = PS.PS_PARTKEY AND S.S_SUPPKEY = PS.PS_SUPPKEY
     AND P.P_SIZE  = 39 AND P.P_TYPE  = 'BRASS' AND S.S_NATION
     = 'MOROCCO' AND
     PS.PS_SUPPLYCOST = (SELECT MIN(PS1.PS_SUPPLYCOST)
     FROM PARTSUPP PS1, SUPPLIERS S1
     WHERE P.P_PARTKEY  = PS1.PS_PARTKEY AND S1.S_SUPPKEY = PS1.PS_SUPPKEY
     AND S1.S_NATION  = S.S_NATION)]]>
   </STMTKEY>
</STMTPROFILE>
```

5. ステートメント・キーの下で、ステートメント最適化ガイドラインを構成します。直接的な名前を使用して、アクセスおよび結合要求で使用されるオブジェクトを識別します。以下に、結合要求の例を示します。

```
<OPTGUIDELINES>
  <HSJOIN>
    <TBSCAN TABLE='PS1'/>
    <IXSCAN TABLE='S1'
            INDEX='I1'/>
  </HSJOIN>
</OPTGUIDELINES>
```

6. ファイルを妥当性検査して保管します。

結果を検査するには、『最適化プロファイルを使用するように DB2 を構成する』、『オプティマイザーが使用する最適化プロファイルの指定』、および 415 ページの『最適化ガイドラインが使用されていることの確認』にある手順に従ってください。希望する結果が得られない場合、ガイドラインを変更し (または実行計画にさらに局面を指定し)、 423 ページの『最適化プロファイルの変更』の説明に従って最適化プロファイルを更新します。必要に応じて繰り返します。

最適化プロファイルを使用するように DB2 を構成する:

最適化プロファイル・ファイルが構成され、その内容が現行最適化プロファイル・スキーマ (COPS) に対して妥当性検査された後、その内容を固有のスキーマ修飾名に関連付けて、SYSTOOLS.OPT_PROFILE 表に保管する必要があります。

最適化プロファイルを使用するように DB2 を構成するには、以下のようにします。

1. 449 ページの『SYSTOOLS.OPT_PROFILE 表』に示されているように、最適化プロファイル表を作成します。 表の各行には最適化プロファイルを 1 つ含めることができます。SCHEMA および NAME 列は最適化プロファイルを識別し、PROFILE 列には最適化プロファイルのテキストが含まれます。
2. オプション: データベース・セキュリティー要件を満たす権限を付与することができます。オプティマイザーは、権限セットに関係なく、表を読み取ることができます。
3. 使用する最適化プロファイルを表に挿入します。

オプティマイザーが使用する最適化プロファイルの指定: 最適化プロファイルがパッケージ・レベルで使用されることを指定するには、OPTPROFILE バインド・オプションを使用できます。最適化プロファイルがステートメント・レベルで使用されることを指定するには、CURRENT OPTIMIZATION PROFILE 特殊レジスターを使用できます。この特殊レジスターには、最適化のために動的に準備されたステート

メントによって使用される最適化プロファイルの修飾名が含まれます。 CLI アプリケーションの場合、CURRENTOPTIMIZATIONPROFILE 構成オプションを使用して、各接続にこの特殊レジスターを設定することができます。

また、OPTPROFILE バインド・オプションの設定は、CURRENT OPTIMIZATION PROFILE 特殊レジスターにデフォルトの 最適化プロファイルを指定します。デフォルトの優先順位の順序は、次のとおりです。

- OPTPROFILE バインド・オプションは、その他の設定に関係なく、すべての静的ステートメントに適用されます。
- 動的ステートメントの場合、CURRENT OPTIMIZATION PROFILE 特殊レジスターの値は、優先順位の低い順から高い順に、以下によって決定されます。
  - OPTPROFILE バインド・オプション
  - CURRENTOPTIMIZATIONPROFILE クライアント構成オプション
  - アプリケーション内の最新の SET CURRENT OPTIMIZATION PROFILE ステートメント

最適化プロファイルのパッケージへのバインディング:

BIND または PRECOMPILE コマンドを使用してパッケージを準備する場合、OPTPROFILE オプションを使用して、そのパッケージに使用する最適化プロファイルを指定できます。

この方法は最適化プロファイルを静的ステートメントに適用するための唯一の方法で、指定されたプロファイルはパッケージ内のすべての静的ステートメントに適用されます。また、この方法で指定される最適化プロファイルは、パッケージ内の動的ステートメントに使用されるデフォルト最適化プロファイルです。

API (例えば、sqlaprep) を使用するか、または CLP から SQLJ および組み込み SQL で最適化プロファイルをバインドできます。

例えば、CLP からインベントリー・データベース最適化プロファイルをインベントリー・アプリケーションにバインドするには、以下のようにします。

```
db2 prep inventapp.sqc bindfile optprofile NEWTON.INVENTDB
db2 bind inventapp.bnd
db2 connect reset
db2 terminate
xlc -I$HOME/sqllib/include -c inventapp.c -o inventapp.o
xlc -o inventapp inventapp.o -ldb2 -L$HOME/sqllib/lib
```

最適化プロファイルにスキーマ名を指定しない場合、QUALIFIER オプションが暗黙の修飾子として使用されます。

アプリケーション内での最適化プロファイルの設定:

アプリケーションでは、SET CURRENT OPTIMIZATION PROFILE ステートメントを使用して、動的ステートメントの現行最適化プロファイルの設定を制御できます。ステートメントで指定する最適化プロファイル名はスキーマ修飾名でなければなりません。スキーマ名を指定しない場合、CURRENT SCHEMA 特殊レジスターの値が暗黙スキーマ修飾子として使用されます。

指定される最適化プロファイルは、別の SET CURRENT OPTIMIZATION PROFILE ステートメントが見つかるまで、後続のすべての動的ステートメントに適用されます。静的ステートメントは、この設定が評価される前に前処理されてパックされるため、影響を受けません。

アプリケーション内で最適化プロファイルを設定するには、以下のようにします。

- SET CURRENT OPTIMIZATION PROFILE ステートメントは、アプリケーション全体を通じて使用できます。例えば、次のシーケンス内の最終ステートメントは、JON.SALES 最適化プロファイルに従って最適化されます。

```
EXEC SQL  SET CURRENT OPTIMIZATION PROFILE = 'NEWTON.INVENTDB';

/* The following statements are both optimized with 'NEWTON.INVENTDB' */
EXEC SQL PREPARE stmt FROM SELECT ... ;
EXEC SQL EXECUTE stmt;

EXEC SQL EXECUTE IMMEDIATE SELECT ... ;

EXEC SQL SET CURRENT OPTIMIZATION PROFILE = 'JON.SALES';

/* This statement is optimized with 'JON.SALES' */
EXEC SQL EXECUTE IMMEDIATE SELECT ... ;
```

図 36. SET CURRENT OPTIMIZATION PROFILE ステートメントを使用したプロファイルの変更

- アプリケーションの開始時に有効だったデフォルト最適化プロファイルをオプティマイザーが使用するようにする場合、ヌル値を指定します。例:

  ```
  EXEC SQL SET CURRENT OPTIMIZATION PROFILE = NULL;
  ```

- オプティマイザーが最適化プロファイルを使用しないようにする場合、空ストリング (’’) を指定します。例:

  ```
  EXEC SQL SET CURRENT OPTIMIZATION PROFILE = '';
  ```

- CLI アプリケーションの場合、CURRENTOPTIMIZATIONPROFILE パラメーターを db2cli.ini ファイルに追加できます。項目をファイルに追加するには、構成アシスタントまたは UPDATE CLI CONFIGURATION コマンドを使用します。例:

  ```
  DB2 UPDATE CLI CFG FOR SECTION SANFRAN USING CURRENTOPTIMIZATIONPROFILE JON.SALES
  ```

  この結果として、db2cli.ini ファイルに次の項目が生成されます。

  ```
  [SANFRAN]
  CURRENTOPTIMIZATIONPROFILE=JON.SALES
  ```

  注: アプリケーション内の SET CURRENT OPTIMIZATION PROFILE ステートメントは、この設定をオーバーライドします。

最適化プロファイルの変更:

最適化プロファイルを変更するには、その文書を編集し、それを現行最適化プロファイル・スキーマ (COPS) に対して妥当性検査して、SYSTOOLS.OPT_PROFILE 表内のオリジナル文書を新しいバージョンに置き換えます。ただし、最適化プロファイルが参照されると、それはコンパイルされて、メモリー内にキャッシュされます。これらの参照も除去する必要があります。 FLUSH OPTIMIZATION PROFILE CACHE ステートメントを使用すると、古い最適化プロファイルを最適化プロファ

イル・キャッシュから除去し、動的プラン・キャッシュにある古いプロファイルを使用して準備されたステートメントも無効にします。

最適化プロファイルを変更するには、以下のようにします。

1. 必要な変更を加えて最適化プロファイル・ファイルを編集し、XML を妥当性検査します。
2. SYSTOOLS.OPT_PROFILE 表内の行を新規プロファイルで更新します。
3. 450 ページの『最適化プロファイル・キャッシュをフラッシュするためのトリガー』を作成しなかった場合、ステートメント FLUSH OPTIMIZATION PROFILE CACHE を発行します。

   注: 最適化プロファイル・キャッシュをフラッシュすると、古い最適化プロファイルで準備された動的ステートメントも動的プラン・キャッシュで無効にされます。

その後、最適化プロファイルを参照すると、オプティマイザーは新しいプロファイルを読み取り、それを最適化プロファイル・キャッシュに再ロードします。また、古い最適化プロファイルで準備されたステートメントは論理的に無効化されるため、それらのステートメントに対する呼び出しは新規の最適化プロファイルで準備され、動的プラン・キャッシュに再キャッシュされます。

最適化プロファイルの削除:

必要ではない最適化プロファイルは、SYSTOOLS.OPT_PROFILE 表から削除することによって除去できます。最適化プロファイルが参照されると、それはコンパイルされて、メモリー内にキャッシュされます。そのため、オリジナル・プロファイルがすでに使用されている場合、最適化プロファイル・キャッシュからも削除された最適化プロファイルをフラッシュする必要があります。

最適化プロファイルを削除するには、以下のようにします。

1. SYSTOOLS.OPT_PROFILE 表から最適化プロファイルを削除します。例:

   ```
   DELETE FROM SYSTOOLS.OPT_PROFILE
     WHERE SCHEMA = 'NEWTON' AND NAME = 'INVENTDB';
   ```

2. 450 ページの『最適化プロファイル・キャッシュをフラッシュするためのトリガー』を作成しなかった場合、CLP から FLUSH OPTIMIZATION PROFILE CACHE コマンドを実行して、最適化プロファイル・キャッシュに含まれている可能性のある最適化プロファイルのすべてのバージョンをフラッシュします。

   注: また、このアクションにより、古い最適化プロファイルを使用して準備されたステートメントが動的プラン・キャッシュ内で無効にされます。

その後、その最適化プロファイルを参照すると、オプティマイザーは理由コード 13 の警告 SQL0437W を出します。

最適化プロファイルおよびガイドライン用の XML スキーマ

現行の最適化プロファイル・スキーマ:

以下は最適化プロファイルのスキーマです。

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified" version="1.0">
<!--*****************************************************************************************-->
<!-- IBM Confidential                                                                        -->
<!-- Licensed Materials - Property of IBM                                                    -->
<!-- (C) Copyright International Business Machines Corporation 2007. All rights reserved.    -->
<!-- U.S. Government Users Restricted Rights; Use, duplication or disclosure restricted by   -->
<!-- GSA ADP Schedule Contract with IBM Corp.                                                -->
<!--*****************************************************************************************-->
<!--*****************************************************************************************-->
<!-- Definition of the current optimization profile schema for V9.5.0.0                      -->
<!--                                                                                         -->
<!-- An optimization profile is composed of the following sections:                          -->
<!--                                                                                         -->
<!-- + A global optimization guidelines section (at most one) which defines optimization     -->
<!-- guidelines affecting any statement for which the optimization profile is in effect.     -->
<!--                                                                                         -->
<!-- + Zero or more statement profile sections, each of which defines optimization           -->
<!-- guidelines for a particular statement for which the optimization profile                -->
<!-- is in effect.                                                                           -->
<!--                                                                                         -->
<!-- The VERSION attribute indicates the version of this optimization profile                -->
<!-- schema.                                                                                 -->
<!--*****************************************************************************************-->
<xs:element name="OPTPROFILE">
   <xs:complexType>
      <xs:sequence>
         <!-- Global optimization guidelines section. At most one can be specified. -->
         <xs:element name="OPTGUIDELINES" type="globalOptimizationGuidelinesType" minOccurs="0"/>
         <!-- Statement profile section. Zero or more can be specified -->
         <xs:element name="STMTPROFILE" type="statementProfileType" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <!-- Version attribute is currently optional -->
      <xs:attribute name="VERSION" use="optional">
         <xs:simpleType>
            <xs:restriction base="xs:NMTOKEN">
               <xs:enumeration value="9.5.0.0"/>
            </xs:restriction>
         </xs:simpleType>
      </xs:attribute>
   </xs:complexType>
</xs:element>
<!--*****************************************************************************************-->
<!-- Global optimization guidelines supported in this version:                               -->
<!-- + MQTOptimizationChoices elements influence the MQTs considered by the optimizer.       -->
<!-- + computationalPartitionGroupOptimizationsChoices elements can affect repartitioning    -->
<!-- optimizations involving nicknames.                                                      -->
<!-- + The REOPT element can affect how statements involving variables are optimized.        -->
<!-- ***************************************************************************************** -->
<xs:complexType name="globalOptimizationGuidelinesType">
   <xs:sequence>
      <xs:group ref="MQTOptimizationChoices" />
      <xs:group ref="computationalPartitionGroupOptimizationChoices" />
      <xs:group ref="generalRequest"/>
   </xs:sequence>
</xs:complexType>
<!-- *****************************************************************************************-->
<!-- Elements for affecting materialized query table (MQT) optimization.                     -->
<!--                                                                                         -->
<!-- + MQTOPT - can be used to disable materialized query table (MQT) optimization.          -->
<!-- If disabled, the optimizer will not consider MQTs to optimize the statement.            -->
<!--                                                                                         -->
<!--  + MQT - multiple of these can be specified. Each specifies an MQT that should be       -->
<!-- considered for optimizing the statement. Only specified MQTs will be considered.        -->
<!--                                                                                         -->
<!--*****************************************************************************************-->
   <xs:group name="MQTOptimizationChoices">
   <xs:choice>
      <xs:element name="MQTOPT" minOccurs="0" maxOccurs="1">
         <xs:complexType>
            <xs:attribute name="OPTION" type="optionType" use="optional"/>
         </xs:complexType>
      </xs:element>
      <xs:element name="MQT" minOccurs="0" maxOccurs="unbounded">
         <xs:complexType>
            <xs:attribute name="NAME" type="xs:string" use="required"/>
         </xs:complexType>
      </xs:element>
   </xs:choice>
```

```
</xs:group>
<!-- ********************************************************************************************-->
<!-- Elements for affecting computational partition group (CPG) optimization.                    -->
<!--                                                                                             -->
<!-- + PARTOPT - can be used disable the computational partition group (CPG) optimization        -->
<!-- which is used to dynamically redistributes inputs to join, aggregation,                     -->
<!-- and union operations when those inputs are results of remote queries.                       -->
<!--                                                                                             -->
<!-- + PART - Define the partition groups to be used in CPG optimizations.                       -->
<!--                                                                                             -->
<!-- ********************************************************************************************-->
<xs:group name="computationalPartitionGroupOptimizationChoices">
   <xs:choice>
      <xs:element name="PARTOPT" minOccurs="0" maxOccurs="1">
         <xs:complexType>
            <xs:attribute name="OPTION" type="optionType" use="optional"/>
      </xs:complexType>
      </xs:element>
      <xs:element name="PART" minOccurs="0" maxOccurs="1">
         <xs:complexType>
            <xs:attribute name="NAME" type="xs:string" use="required"/>
         </xs:complexType>
      </xs:element>
   </xs:choice>
</xs:group>
<!-- ********************************************************************************************-->
<!-- Definition of a statement profile.                                                          -->
<!-- Comprised of a statement key and optimization guidelines.                                   -->
<!-- The statement key specifies semantic information used to identify the statement to          -->
<!-- which optimization guidelines apply. The optional ID attribute provides the statement       -->
<!-- profile with a name for use in EXPLAIN output.                                              -->
<!-- ********************************************************************************************-->
<xs:complexType name="statementProfileType">
   <xs:sequence>
      <!-- Statement key element -->
      <xs:element name="STMTKEY" type="statementKeyType"/>
      <xs:element name="OPTGUIDELINES" type="optGuidelinesType"/>
   </xs:sequence>
   <!-- ID attribute.Used in explain output to indicate the statement profile was used. -->
   <xs:attribute name="ID" type="xs:string" use="optional"/>
</xs:complexType>
<!--*********************************************************************************************-->
<!-- Definition of the statement key. The statement key provides semantic information used       -->
<!-- to identify the statement to which the optimization guidelines apply.                       -->
<!-- The statement key is comprised of:                                                          -->
<!-- + statement text (as written in the application)                                            -->
<!-- + default schema (for resolving unqualified table names in the statement)                   -->
<!-- + function path (for resolving unqualified types and functions in the statement)            -->
<!-- The statement text is provided as element data whereas the default schema and function      -->
<!-- path are provided via the SCHEMA and FUNCPATH elements, respectively.                       -->
<!--*********************************************************************************************-->
<xs:complexType name="statementKeyType" mixed="true">
   <xs:attribute name="SCHEMA" type="xs:string" use="optional"/>
   <xs:attribute name="FUNCPATH" type="xs:string" use="optional"/>
</xs:complexType>
<!--*********************************************************************************************-->
<!--                                                                                             -->
<!-- Optimization guideline elements can be chosen from general requests, rewrite               -->
<!-- requests access requests, or join requests.                                                 -->
<!--                                                                                             -->
<!-- General requests affect the search space which defines the alternative query                -->
<!-- transformations, access methods, join methods, join orders, and other optimizations,        -->
<!-- considered by the optimizer.                                                                -->
<!--                                                                                             -->
<!-- Rewrite requests affect the query transformations used in determining the optimized         -->
<!-- statement.                                                                                  -->
<!--                                                                                             -->
<!-- Access requests affect the access methods considered by the cost-based optimizer,          -->
<!-- and join requests affect the join methods and join order used in the execution plan.        -->
<!--                                                                                             -->
<!--*********************************************************************************************-->
<xs:element name="OPTGUIDELINES" type="optGuidelinesType"/>
<xs:complexType name="optGuidelinesType">
   <xs:sequence>
      <xs:group ref="generalRequest" minOccurs="0" maxOccurs="1"/>
      <xs:choice maxOccurs="unbounded">
         <xs:group ref="rewriteRequest" />
         <xs:group ref="accessRequest"/>
         <xs:group ref="joinRequest"/>
```

```
      </xs:choice>
   </xs:sequence>
</xs:complexType>
<!--****************************************************************************** -->
<!-- Choices of general request elements.                                         -->
<!-- REOPT can be used to override the setting of the REOPT bind option.          -->
<!--****************************************************************************** -->
<xs:group name="generalRequest">
   <xs:sequence>
      <xs:element name="REOPT" type="reoptType" minOccurs="0" maxOccurs="1"/>
      <xs:element name="DEGREE" type="degreeType" minOccurs="0" maxOccurs="1"/>
      <xs:element name="QRYOPT" type="qryoptType" minOccurs="0" maxOccurs="1"/>
      <xs:element name="RTS" type="rtsType" minOccurs="0" maxOccurs="1"/>
   </xs:sequence>
</xs:group>
<!--****************************************************************************--->
<!-- Choices of rewrite request elements.                                     -->
<!--****************************************************************************--->
<xs:group name="rewriteRequest">
   <xs:sequence>
      <xs:element name="INLIST2JOIN" type="inListToJoinType" minOccurs="0"/>
      <xs:element name="SUBQ2JOIN" type="subqueryToJoinType" minOccurs="0"/>
      <xs:element name="NOTEX2AJ" type="notExistsToAntiJoinType" minOccurs="0"/>
      <xs:element name="NOTIN2AJ" type="notInToAntiJoinType" minOccurs="0"/>
   </xs:sequence>
</xs:group>
<!--****************************************************************************** -->
<!-- Choices for access request elements.                                         -->
<!-- TBSCAN - table scan access request element                                   -->
<!-- IXSCAN - index scan access request element                                   -->
<!-- LPREFETCH - list prefetch access request element                             -->
<!-- IXAND - index ANDing access request element                                  -->
<!-- IXOR - index ORing access request element                                    -->
<!-- ACCESS - indicates the optimizer should choose the access method for the table -->
<!--****************************************************************************** -->
<xs:group name="accessRequest">
   <xs:choice>
      <xs:element name="TBSCAN" type="tableScanType"/>
      <xs:element name="IXSCAN" type="indexScanType"/>
      <xs:element name="LPREFETCH" type="listPrefetchType"/>
      <xs:element name="IXAND" type="indexAndingType"/>
      <xs:element name="IXOR" type="indexOringType"/>
      <xs:element name="ACCESS" type="anyAccessType"/>
   </xs:choice>
</xs:group>
<!--****************************************************************************** -->
<!-- Choices for join request elements.                                           -->
<!-- NLJOIN - nested-loops join request element                                   -->
<!-- MSJOIN - sort-merge join request element                                     -->
<!-- HSJOIN - hash join request element                                           -->
<!-- JOIN - indicates that the optimizer is to choose the join method.            -->
<!--****************************************************************************** -->
<xs:group name="joinRequest">
   <xs:choice>
      <xs:element name="NLJOIN" type="nestedLoopJoinType"/>
      <xs:element name="HSJOIN" type="hashJoinType"/>
      <xs:element name="MSJOIN" type="mergeJoinType"/>
      <xs:element name="JOIN" type="anyJoinType"/>
   </xs:choice>
</xs:group>
<!--****************************************************************************** -->
<!-- REOPT general request element. Can override REOPT setting at the package, db,  -->
<!-- dbm level.                                                                     -->
<!--****************************************************************************** -->
<xs:complexType name="reoptType">
   <xs:attribute name="VALUE" use="required">
      <xs:simpleType>
         <xs:restriction base="xs:string">
            <xs:enumeration value="ONCE"/>
            <xs:enumeration value="ALWAYS"/>
         </xs:restriction>
      </xs:simpleType>
   </xs:attribute>
</xs:complexType>
<!--****************************************************************************--->
<!-- RTS general request element to enable, disable or provide a time budget for  -->
<!-- real-time statistics collection.                                             -->
<!-- OPTION attribute allows enabling or disabling real-time statistics.          -->
<!-- TIME attribute provides a time budget in milliseconds for real-time statistics collection.-->
```

```
<!--************************************************************************************** -->
<xs:complexType name="rtsType">
   <xs:attribute name="OPTION" type="optionType" use="optional" default="ENABLE"/>
   <xs:attribute name="TIME" type="xs:nonNegativeInteger" use="optional"/>
</xs:complexType>
<!--**************************************************************************************-->
<!-- Definition of an "IN list to join" rewrite request                                 -->
<!-- OPTION attribute allows enabling or disabling the alternative.                       -->
<!-- TABLE attribute allows request to target IN list predicates applied to a             -->
<!-- specific table reference. COLUMN attribute allows request to target a specific IN list   -->
<!-- predicate.                                                                           -->
<!--************************************************************************************** -->
<xs:complexType name="inListToJoinType">
   <xs:attribute name="OPTION" type="optionType" use="optional" default="ENABLE"/>
   <xs:attribute name="TABLE" type="xs:string" use="optional"/>
   <xs:attribute name="COLUMN" type="xs:string" use="optional"/>
</xs:complexType>
<!--**************************************************************************************-->
<!-- Definition of a "subquery to join" rewrite request                                   -->
<!-- The OPTION attribute allows enabling or disabling the alternative.                   -->
<!--************************************************************************************** -->
<xs:complexType name="subqueryToJoinType">
   <xs:attribute name="OPTION" type="optionType" use="optional" default="ENABLE"/>
</xs:complexType>
<!--**************************************************************************************-->
<!-- Definition of a "not exists to anti-join" rewrite request                          -->
<!-- The OPTION attribute allows enabling or disabling the alternative.                   -->
<!--************************************************************************************** -->
<xs:complexType name="notExistsToAntiJoinType">
   <xs:attribute name="OPTION" type="optionType" use="optional" default="ENABLE"/>
</xs:complexType>
<!--**************************************************************************************-->
<!-- Definition of a "not IN to anti-join" rewrite request                              -->
<!-- The OPTION attribute allows enabling or disabling the alternative.                   -->
<!--**************************************************************************************-->
<xs:complexType name="notInToAntiJoinType">
   <xs:attribute name="OPTION" type="optionType" use="optional" default="ENABLE"/>
</xs:complexType>
<!--**************************************************************************************-->
<!-- Effectively the superclass from which all access request elements inherit.           -->
<!-- This type currently defines TABLE and TABID attributes, which can be used to tie an  -->
<!-- access request to a table reference in the query.                                    -->
<!-- The TABLE attribute value is used to identify a table reference using identifiers    -->
<!-- in the original SQL statement. The TABID attribute value is used to identify a table -->
<!-- referece using the unique correlation name provided via the                          -->
<!-- optimized statement. If both the TABLE and TABID attributes are specified, the TABID -->
<!-- field is ignored. The FIRST attribute indicates that the access should be the first  -->
<!-- access in the join sequence for the FROM clause.                                     -->
<!--************************************************************************************** -->
<xs:complexType name="accessType" abstract="true">
   <xs:attribute name="TABLE" type="xs:string" use="optional"/>
   <xs:attribute name="TABID" type="xs:string" use="optional"/>
   <xs:attribute name="FIRST" type="xs:string" use="optional" fixed="TRUE"/>
</xs:complexType>
<!--**************************************************************************************-->
<!-- Definition of an table scan access request method.                                -->
<!--**************************************************************************************-->
<xs:complexType name="tableScanType">
   <xs:complexContent>
      <xs:extension base="accessType"/>
   </xs:complexContent>
</xs:complexType>
<!-- **************************************************************************************-->
<!-- Definition of an index scan access request element. The index name is optional.      -->
<!--************************************************************************************** -->
<xs:complexType name="indexScanType">
   <xs:complexContent>
      <xs:extension base="accessType">
         <xs:attribute name="INDEX" type="xs:string" use="optional"/>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>
<!--**************************************************************************************-->
<!-- Definition of a list prefetch access request element. The index name is optional.    -->
<!--************************************************************************************** -->
<xs:complexType name="listPrefetchType">
   <xs:complexContent>
      <xs:extension base="accessType">
         <xs:attribute name="INDEX" type="xs:string" use="optional"/>
```

```
      </xs:extension>
   </xs:complexContent>
</xs:complexType>
<!--*****************************************************************************************-->
<!-- Definition of an index ANDing access request element.                                   -->
<!-- A single index scan be specified via the INDEX attribute. Multiple indexes              -->
<!-- can be specified via INDEX elements. The index element specification supersedes the     -->
<!-- attribute specification. If a single index is specified, the optimizer will use the     -->
<!-- index as the first index of the index ANDing access method and will choose addi-        -->
<!-- tional indexes using cost. If multiple indexes are specified the optimizer will         -->
<!-- use exactly those indexes in the specified order. If no indexes are specified           -->
<!-- via either the INDEX attribute or INDEX elements, then the optimizer will choose        -->
<!-- all indexes based upon cost.                                                            -->
<!--***************************************************************************************** -->
<xs:complexType name="indexAndingType">
   <xs:complexContent>
      <xs:extension base="accessType">
         <xs:sequence minOccurs="0">
            <xs:element name="INDEX" type="indexType" minOccurs="2" maxOccurs="unbounded"/>
         </xs:sequence>
         <xs:attribute name="INDEX" type="xs:string" use="optional"/>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>
<!--*****************************************************************************************-->
<!-- Definition of an INDEX element method. Index set is optional. If specified,            -->
<!-- at least 2 are required.                                                                -->
<!--***************************************************************************************** -->
<xs:complexType name="indexType">
   <xs:attribute name="IXNAME" type="xs:string" use="optional"/>
</xs:complexType>
<!--*****************************************************************************************-->
<!-- Use for derived table access or other cases where the access method is not of           -->
<!-- consequence.                                                                            -->
<!--*****************************************************************************************-->
<xs:complexType name="anyAccessType">
   <xs:complexContent>
      <xs:extension base="accessType"/>
   </xs:complexContent>
</xs:complexType>
<!--*****************************************************************************************-->
<!-- Definition of an index ORing access                                                     -->
<!-- Cannot specify more details (e.g indexes). Optimizer will choose the details based      -->
<!-- upon cost.                                                                              -->
<!--***************************************************************************************** -->
<xs:complexType name="indexOringType">
   <xs:complexContent>
      <xs:extension base="accessType"/>
   </xs:complexContent>
</xs:complexType>
<!--*****************************************************************************************-->
<!-- Effectively the super class from which join request elements inherit.                   -->
<!-- This type currently defines join element inputs and also the FIRST attribute.           -->
<!-- A join request must have exactly two nested sub-elements. The sub-elements can be       -->
<!-- either an access request or another join request. The first sub-element represents      -->
<!-- outer table of the join operation while the second element represents the inner         -->
<!-- table. The FIRST attribute indicates that the join result should be the first join      -->
<!-- relative to other tables in the same FROM clause.                                       -->
<!--***************************************************************************************** -->
<xs:complexType name="joinType" abstract="true">
   <xs:choice minOccurs="2" maxOccurs="2">
      <xs:group ref="accessRequest"/>
      <xs:group ref="joinRequest"/>
   </xs:choice>
   <xs:attribute name="FIRST" type="xs:string" use="optional" fixed="TRUE"/>
</xs:complexType>
<!--***************************************************************************************** -->
<!-- Definition of nested loop join access request. Subclass of joinType.                    -->
<!-- Does not add any elements or attributes.                                                -->
<!--***************************************************************************************** -->
<xs:complexType name="nestedLoopJoinType">
   <xs:complexContent>
      <xs:extension base="joinType"/>
   </xs:complexContent>
</xs:complexType>
<!--*****************************************************************************************-->
<!-- Definition of merge join access request. Subclass of joinType.                          -->
<!-- Does not add any elements or attributes.                                                -->
<!--***************************************************************************************** -->
```

```
<xs:complexType name="mergeJoinType">
   <xs:complexContent>
      <xs:extension base="joinType"/>
   </xs:complexContent>
</xs:complexType>
<!--******************************************************************************** -->
<!-- Definition of hash join access request. Subclass of joinType.                  -->
<!-- Does not add any elements or attributes.                                       -->
<!--******************************************************************************** -->
<xs:complexType name="hashJoinType">
   <xs:complexContent>
      <xs:extension base="joinType"/>
   </xs:complexContent>
</xs:complexType>
<!--******************************************************************************** -->
<!-- Any join is a subclass of binary join. Does not extend it in any way.          -->
<!-- Does not add any elements or attributes.                                       -->
<!--******************************************************************************** -->
<xs:complexType name="anyJoinType">
   <xs:complexContent>
      <xs:extension base="joinType"/>
   </xs:complexContent>
</xs:complexType>
<!--*************************************************************************************************-->
<!-- Allowable values for an OPTION attribute.                                                       -->
<!--*************************************************************************************** -->
<xs:simpleType name="optionType">
   <xs:restriction base="xs:string">
      <xs:enumeration value="ENABLE"/>
      <xs:enumeration value="DISABLE"/>
   </xs:restriction>
</xs:simpleType>
<!--*************************************************************************************************-->
<!-- Definition of the qryopt type: the only values allowed are 0, 1, 2, 3, 5, 7 and 9       -->
<!--*************************************************************************************************-->
<xs:complexType name="qryoptType">
   <xs:attribute name="VALUE" use="required">
      <xs:simpleType>
         <xs:restriction base="xs:string">
            <xs:enumeration value="0"/>
            <xs:enumeration value="1"/>
            <xs:enumeration value="2"/>
            <xs:enumeration value="3"/>
            <xs:enumeration value="5"/>
            <xs:enumeration value="7"/>
            <xs:enumeration value="9"/>
         </xs:restriction>
      </xs:simpleType>
   </xs:attribute>
</xs:complexType>
<!--*************************************************************************************************-->
<!-- Definition of the degree type: any number between 1 and 32767 or the strings ANY or -1  -->
<!--******************************************************************************** -->
<xs:simpleType name="intStringType">
   <xs:union>
      <xs:simpleType>
         <xs:restriction base="xs:integer">
            <xs:minInclusive value="1"></xs:minInclusive>
            <xs:maxInclusive value="32767"></xs:maxInclusive>
         </xs:restriction>
      </xs:simpleType>
      <xs:simpleType>
         <xs:restriction base="xs:string">
            <xs:enumeration value="ANY"/>
            <xs:enumeration value="-1"/>
         </xs:restriction>
      </xs:simpleType>
   </xs:union>
</xs:simpleType>
<xs:complexType name="degreeType">
 <xs:attribute name="VALUE" type="intStringType"></xs:attribute>
</xs:complexType>
</xs:schema>
```

OPTPROFILE エレメント用の XML スキーマ: OPTPROFILE エレメントは、最適化プロファイルのルートです。 OPTPROFILE エレメントは、次のように定義します。

XML Schema

```
<xs:element name="OPTPROFILE">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="OPTGUIDELINES" type="globalOptimizationGuidelinesType"
           minOccurs="0"/>
         <xs:element name="STMTPROFILE" type="statementProfileType" minOccurs="0"
           maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="VERSION" use="optional">
         <xs:simpleType>
            <xs:restriction base="xs:NMTOKEN">
               <xs:enumeration value="9.1.0.0"/>
            </xs:restriction>
         </xs:simpleType>
      </xs:attribute>
   </xs:complexType>
</xs:element>
```

説明

オプションの OPTGUIDELINES サブエレメントは、最適化プロファイルのグローバル最適化ガイドラインのセクションを定義します。各 STMTPROFILE サブエレメントは、ステートメント・プロファイル・セクションを定義します。 VERSION 属性は、特定の最適化プロファイルの作成および検証時に照らし合わせる現在の最適化プロファイル・スキーマを識別します。

グローバル OPTGUIDELINES エレメントの XML スキーマ: タイプ *globalOptimizationGuidelinesType* は、グローバル OPTGUIDELINES エレメントの形式を定義します。

XML Schema

```
<xs:complexType name="globalOptimizationGuidelinesType">
   <xs:sequence>
      <xs:group ref="MQTOptimizationChoices"/>
      <xs:group ref="computationalPartitionGroupOptimizationChoices"/>
      <xs:group ref="generalRequest"/>
   </xs:sequence>
</xs:complexType>
```

説明

グローバル最適化ガイドラインの定義には、グループ MQTOptimizationChoices のエレメント、グループ computationalPartitionGroupChoices のエレメント、または REOPT エレメントを使用できます。

432 ページの『MQT 最適化の選択項目』に定義されている MQTOptimizationChoices グループ・エレメントを使用して、MQT 置換に作用することができます。 432 ページの『計算パーティション・グループの最適化の選択項目』に定義されている computationalPartitionGroupOptimizationChoices グループ・エレメントを使用して、計算パーティション・グループの最適化に作用することができます。

計算パーティション・グループの最適化の一環として、リモート・データ・ソースから読み取られたデータの動的再分散が行われます。これは、パーティション化されたフェデレーテッド・データベース構成においてのみ当てはまります。

433 ページの『REOPT グローバル最適化ガイドライン』に定義されている REOPT エレメントを使用して、変数を参照するステートメントの最適化がいつ実行されるかを制御することができます。

MQT 最適化の選択項目: グループ *MQTOptimizationChoices* は、マテリアライズ照会表 (MQT) の最適化に作用するのに使用できる一連のエレメントを定義します。具体的には、エレメントを使用して、MQT 置換の考慮を有効または無効することや、オプティマイザーで考慮される MQT の完全セットを指定することができます。

XML Schema

```
<xs:group name="MQTOptimizationChoices">
   <xs:choice>
      <xs:element name="MQTOPT" minOccurs="0" maxOccurs="1">
         <xs:complexType>
            <xs:attribute name="OPTION" type="optionType" use="optional"/>
         </xs:complexType>
      </xs:element>
      <xs:element name="MQT" minOccurs="0" maxOccurs="unbounded">
         <xs:complexType>
            <xs:attribute name="NAME" type="xs:string" use="required"/>
         </xs:complexType>
      </xs:element>
   </xs:choice>
</xs:group>
```

説明

MQTOPT エレメントは、MQT 最適化の考慮を有効または無効にするのに使用されます。 OPTION 属性は、値 ENABLE または DISABLE を指定できます。 OPTION 属性のデフォルト値は ENABLE です。

あるいは、ゼロ個以上の MQT エレメントを用意することもできます。 MQT エレメントの NAME 属性は、オプティマイザーで考慮の対象とされる MQT を識別します。 NAME 属性内の MQT への参照の作成に関する規則は、直接的な表名への参照の作成の規則と同じです。 1 つ以上の MQT エレメントを指定した場合、それらの MQT だけが、オプティマイザーで考慮の対象となります。 1 つ以上の MQT を指定して MQT 置換を実行するかどうかの決定は、やはりコスト・ベースの決定になります。

例

以下の例は、MQT の最適化を無効にする方法を示しています。

```
   <OPTGUIDELINES>
         <MQTOPT OPTION='DISABLE'/>
</OPTGUIDELINES>
```

以下の例は、Tpcd.PARTSMQT および COLLEGE.STUDENTS という MQT にのみ MQT の最適化を限定する方法を示しています。

```
   <OPTGUIDELINES>
         <MQT NAME='Tpcd.PARTSMQT'/>
         <MQT NAME='COLLEGE.STUDENTS'/>
<OPTGUIDELINES>
```

計算パーティション・グループの最適化の選択項目: グループ *computationalPartitionGroupOptimizationChoices* は、計算パーティション・グループ

の最適化に作用するのに使用できる一連のエレメントを定義します。具体的には、エレメントを使用して、計算グループの最適化を有効または無効にすることや、計算パーティション・グループの最適化に使用する特定のパーティション・グループを指定することができます。

XML Schema

```
<xs:group name="computationalPartitionGroupOptimizationChoices">
   <xs:choice>
      <xs:element name="PARTOPT" minOccurs="0" maxOccurs="1">
         <xs:complexType>
            <xs:attribute name="OPTION" type="optionType" use="optional"/>
         </xs:complexType>
      </xs:element>
      <xs:element name="PART" minOccurs="0" maxOccurs="1">
         <xs:complexType>
            <xs:attribute name="NAME" type="xs:string" use="required"/>
         </xs:complexType>
      </xs:element>
   </xs:choice>
</xs:group>
```

説明

PARTOPT エレメントは、計算パーティション・グループの最適化の考慮を有効または無効にするのに使用されます。 OPTION 属性は、値 ENABLE または DISABLE を指定できます。 OPTION 属性のデフォルト値は ENABLE です。

あるいは、PART エレメントを使用して、計算パーティション・グループの最適化に使用するパーティション・グループを指定することもできます。 PART エレメントの NAME 属性は、パーティション・グループを識別しますが、これは既存のパーティション・グループを識別しなければなりません。指定のパーティション・グループを使用した動的再分散の実行の決定は、コスト・ベースの決定になります。

例

以下の例は、計算パーティション・グループの最適化を無効にする方法を示しています。

```
<OPTGUIDELINES>
      <PARTOPT OPTION='DISABLE'/>
</OPTGUIDELINES>
```

以下の例は、計算パーティション・グループの最適化に関してパーティション・グループ WORKPART を使用することを指示する方法を示しています。

```
<OPTGUIDELINES>
      <MQT NAME='Tpcd.PARTSMQT'/>
      <PART NAME='WORKPART'/>
<OPTGUIDELINES>
```

REOPT グローバル最適化ガイドライン: REOPT グローバル最適化ガイドラインは、変数 (ホスト変数、パラメーター・マーカー、グローバル変数または特殊レジスター) を含む DML ステートメントの最適化がいつ実行されるかを制御します。REOPT エレメントの説明および構文は、グローバル最適化ガイドラインとステートメント最適化ガイドラインのどちらでも同じです。 438 ページの『REOPT 要求』を参照してください。

STMTPROFILE エレメントの XML スキーマ: タイプ *statementProfileType* は、STMTPROFILE エレメントの形式を以下のように定義します。

```
XML Schema

   <xs:complexType name="statementProfileType">
      <xs:sequence>
         <xs:element name="STMTKEY" type="statementKeyType"/>
         <xs:element name="OPTGUIDELINES" type="optGuidelinesType"/>
      </xs:sequence>
      <xs:attribute name="ID" type="xs:string" use="optional"/>
   </xs:complexType>
```

説明

ステートメント・プロファイルは、最適化プロファイルが有効にされた特定のステートメントの最適化ガイドラインを指定します。ステートメント・プロファイルは、以下の 2 つの部分で構成されます。

- ステートメント・キー - 最適化プロファイルをアプリケーション内の複数のステートメントに対して有効にすることができます。オプティマイザーはステートメント・キーを使用して、各ステートメント・プロファイルをアプリケーションの該当するステートメントに対して自動的に突き合わせます。ユーザーはこの技法を使えば、アプリケーションを編集しなくても、ステートメントに最適化ガイドラインを提供することができます。

  ステートメント・キーは、アプリケーションに記載されているステートメントのテキストと、該当するアプリケーション・ステートメントを厳密に識別するのに必要なその他の情報で構成されます。 『STMTKEY エレメントの XML スキーマ』で説明されている STMTKEY サブエレメントが、ステートメント・キーです。
- ステートメント・レベル最適化ガイドライン - ステートメント・プロファイルのこのセクションは、ステートメント・キーで識別されるステートメントに有効な最適化ガイドラインを指定します。 436 ページの『ステートメント・レベルの OPTGUIDELINES エレメントの XML スキーマ』を参照してください。
- ステートメント・プロファイル名 - 診断用出力で使用されるユーザー提供の名前。これは、ステートメントの最適化で使用する特定のステートメント・プロファイルを指示します。

STMTKEY エレメントの XML スキーマ: タイプ *statementKeyType* は、STMTKEY エレメントの形式を定義します。

```
XML Schema

   <xs:complexType name="statementKeyType" mixed="true">
      <xs:attribute name="SCHEMA" type="xs:string" use="optional"/>
      <xs:attribute name="FUNCPATH" type="xs:string" use="optional"/>
   </xs:complexType>
   </xs:schema>
```

説明

ステートメント・キーのステートメント・テキスト部分は、STMTKEY エレメントの開始と終了のタグにはさまれたデータとして組み込まれます。

このオプションの SCHEMA 属性を使用して、ステートメント・キーのデフォルトのスキーマ部分を指定することができます。

オプションの FUNCPATH 属性を使用して、ステートメント・キーの関数パス部分を指定することができます。複数の関数パスを、それぞれコンマで区切って、この関数パス内に組み込むことができます。指定する関数パスは、コンパイル・キーに指定されている関数パスと正確に一致していなければなりません。

例

コンパイル・キーのデフォルト・スキーマが COLLEGE、関数パスが SYSIBM,SYSFUN,SYSPROC,DAVE であれば、以下のサンプル・ステートメント・キーは、ステートメント ″select * from orders where foo(orderkey) > 20″ に一致します。

```
<STMTKEY SCHEMA='COLLEGE' FUNCPATH='SYSIBM,SYSFUN,SYSPROC,DAVE'>
<![CDATA[select * from orders" where foo(orderkey) > 20]]>
</STMTKEY>
```

注: CDATA セクション (<![CDATA[で始まり、]]> で終わります) がステートメント・テキストで使用されています。これは、ステートメント・テキストに XML 特殊文字 '>' が含まれているためです。

ステートメント・キーおよびコンパイル・キーのマッチング: ステートメント・レベルの最適化ガイドラインが適用される特定のアプリケーション・ステートメントを識別するには、ステートメント・キーを使用します。 DB2 での静的または動的な SQL ステートメントのコンパイル時には、そのステートメントがコンパイラーによってどのように意味論的に解釈されるかについては、特定のパラメーター (特殊レジスター、およびバインド・オプションまたはプリコンパイル・オプションによって設定されます) の設定が作用します。 SQL ステートメントと、そのような特定の SQL コンパイラー・パラメーターの設定が一緒になって、コンパイル・キーと呼ばれるものを形成します。ステートメント・キーのそれぞれの部分は、コンパイル・キーの部分に対応します。

ステートメント・キー は、以下の部分で構成されます。

- ステートメント・テキスト - アプリケーションで作成されたとおりのステートメント・テキスト。
- デフォルト・スキーマ - 非修飾の表名用の暗黙の数量詞として使用されるスキーマ名。この部分はオプションですが、ステートメント・テキスト内に非修飾の表名がある場合は指定する必要があります。
- 関数パス - 非修飾関数およびデータ・タイプ参照の解決時に使用される関数パス。この部分はオプションですが、ステートメント内に非修飾のユーザー定義関数やユーザー定義タイプがある場合は指定する必要があります。

DB2 での SQL ステートメントのコンパイル時に、アクティブな最適化プロファイルが検出されると、その最適化プロファイルの各ステートメント・キーと現在のコンパイル・キーとのマッチングが試行されます。ステートメント・キーの各指定部分が、コンパイル・キーの対応部分と一致する場合には、ステートメント・キーとコンパイル・キーは一致したことになります。ステートメント・キーの一部を指定しなかった場合、その省略された部分はデフォルトで一致したものとみなされま

す。つまり、ステートメント・キーの各未指定部分は、ワイルドカードとして扱われて、任意のコンパイル・キーの対応部分にマッチングされるということです。

DB2 で、現在のコンパイル・キーに一致するステートメント・キーが検出されると、検索は停止されます。したがって、ステートメント・キーが現在のコンパイル・キーに一致する最適化プロファイル内に複数のステートメント・プロファイルがあった場合、そのようなステートメント・プロファイルのうちの最初のもの (文書の順序に基づいて) だけが使用されます。しかも、このような場合、エラーや警告は出されません。

ステートメント・レベルの OPTGUIDELINES エレメントの XML スキーマ: ステートメント・プロファイルの OPTGUIDELINES エレメントは、関連したステートメント・キーによって識別されるステートメントに有効な最適化ガイドラインを定義します。 OPTGUIDELINES エレメントは、タイプ *optGuidelinesType* で以下のように定義されます。

```
XML Schema

   <xs:element name="OPTGUIDELINES" type="optGuidelinesType"/>
   <xs:complexType name="optGuidelinesType">
      <xs:sequence>
         <xs:group ref="general request" minOccurs="0" maxOccurs="1"/>
         <xs:choice maxOccurs="unbounded">
            <xs:group ref="rewriteRequest"/>
            <xs:group ref="accessRequest"/>
            <xs:group ref="joinRequest"/>
         </xs:choice>
     </xs:sequence>
   </xs:complexType>
```

説明

optGuidelinesType グループは、OPTGUIDELINES エレメントの一連の有効なサブエレメントを定義します。それぞれのサブエレメントは、DB2 オプティマイザーによって最適化ガイドラインとして理解されます。サブエレメントは、一般要求エレメント、書き直し要求エレメント、アクセス要求エレメント、または結合要求エレメントのいずれかとして分類することができます。

- 一般要求エレメントは、一般最適化ガイドラインを指定するために使用されます。一般最適化ガイドラインを使用して、オプティマイザーの検索スペースを変更することができます。
- 書き直し要求エレメントは、照会書き直し最適化ガイドラインを指定するために使用されます。照会書き直しガイドラインは、最適化されたステートメントの判別で適用された照会のトランスフォーメーションに作用するために使用できます。
- アクセスおよび結合要求エレメントは、プラン最適化ガイドラインです。プラン最適化ガイドラインは、最適化されたステートメントの実行プランで使用されるアクセス方式、結合方式、および結合順序に作用するために使用できます。

注: ステートメント・プロファイルで指定される最適化ガイドラインは、最適化プロファイルのグローバル・セクションで指定される最適化ガイドラインよりも優先します。

一般最適化ガイドライン用の XML スキーマ: 一般最適化ガイドラインは、最適化プロセスの特定の段階固有のものではないガイドラインを指定するのに使用されま

す。一般最適化ガイドラインを使用して、オプティマイザーの検索スペースを変更することができます。このガイドラインは、*generalRequest* グループで構成されます。

```
<!--****************************************************************************** --> ¥
<!-- Choices of general request elements.                                         --> ¥
<!-- REOPT can be used to override the setting of the REOPT bind option.          --> ¥
<!--****************************************************************************** --> ¥
<xs:group name="generalRequest">
   <xs:sequence>
      <xs:element name="REOPT" type="reoptType" minOccurs="0" maxOccurs="1"/>
      <xs:element name="DEGREE" type="degreeType" minOccurs="0" maxOccurs="1"/>
      <xs:element name="QRYOPT" type="qryoptType" minOccurs="0" maxOccurs="1"/>
      <xs:element name="RTS" type="rtsType" minOccurs="0" maxOccurs="1"/>
   </xs:sequence>
</xs:group>
```

説明

一般要求エレメントを使用して、一般最適化ガイドラインを定義することができます。一般最適化ガイドラインは、最適化検索スペースに作用するため、書き直しやコスト・ベースの最適化ガイドラインの適用度にも作用します。

DEGREE 要求: DEGREE 一般要求エレメントを使用して、バインド・オプション、dft_degree データベース構成パラメーター、または前の SET CURRENT DEGREE ステートメントの設定をオーバーライドすることができます。 DEGREE 一般要求が考慮の対象となるのは、データベース・マネージャーがパーティション内並列処理用に構成されている場合だけであり、データベース・マネージャーがパーティション内並列処理用に構成されていない場合は警告が戻されます。 DEGREE 一般要求エレメントは、degreeType によって次のように定義されます。

XML Schema

```
<xs:simpleType name="intStringType">
   <xs:union>
      <xs:simpleType>
         <xs:restriction base="xs:integer">
            <xs:minInclusive value="1"></xs:minInclusive>
            <xs:maxInclusive value="32767"></xs:maxInclusive>
         </xs:restriction>
      </xs:simpleType>
      <xs:simpleType>
         <xs:restriction base="xs:string">
            <xs:enumeration value="ANY"/>
            <xs:enumeration value="-1"/>
         </xs:restriction>
      </xs:simpleType>
   </xs:union>
</xs:simpleType>

<xs:complexType name="degreeType">
   <xs:attribute name="VALUE"
      type="intStringType"></xs:attribute>
</xs:complexType>
```

説明

DEGREE 一般要求エレメントの必須の VALUE 属性は、DEGREE オプションの設定を指定します。この属性は、1 から 32767 までの整数値、またはストリング値 ″-1″ または ″ANY″ をとることができます。値 -1 (つまり、″ANY″ と同等) は、使用する並列処理の度合いは、DB2 によって決定されることを示します。値 1 は、

パーティション内並列処理を照会で使用してはならないことを示します。

QRYOPT 要求: QRYOPT 一般要求エレメントを使用して、バインド・オプション、dft_qryopt データベース構成パラメーター、または前の SET CURRENT QUERY OPTIMIZATION ステートメントの設定をオーバーライドすることができます。 QRYOPT 一般要求エレメントは、qryoptType によって次のように定義されます。

XML Schema

```
<xs:complexType name="qryoptType">
   <xs:attribute name="VALUE" use="required">
      <xs:simpleType>
         <xs:restriction base="xs:string">
            <xs:enumeration value="0"/>
            <xs:enumeration value="1"/>
            <xs:enumeration value="2"/>
            <xs:enumeration value="3"/>
            <xs:enumeration value="5"/>
            <xs:enumeration value="7"/>
            <xs:enumeration value="9"/>
         </xs:restriction>
      </xs:simpleType>
   </xs:attribute>
</xs:complexType>
```

説明

QRYOPT 一般要求エレメントの必須の VALUE 属性は、DEGREE オプションの設定を指定します。この属性は、リストから任意の値 (0、1、2、3、5、7 および 9) をとることができます。指定可能な各設定の動作に関する詳細な情報は、「パフォーマンス・ガイド 」に記載されています。

REOPT 要求: REOPT 一般要求エレメントを使用して、REOPT バインド・オプションの設定をオーバーライドすることができます。REOPT バインド・オプションは、パラメーター・マーカーまたはホスト変数を使用するステートメントの最適化に作用します。 REOPT 一般要求エレメントは、reoptType によって次のように定義されます。

XML Schema

```
<xs:complexType name="reoptType">
   <xs:attribute name="VALUE" use="required">
      <xs:simpleType>
         <xs:restriction base="xs:string">
            <xs:enumeration value="ONCE"/>
            <xs:enumeration value="ALWAYS"/>
         </xs:restriction>
      </xs:simpleType>
   </xs:attribute>
</xs:complexType>
```

説明

REOPT 一般要求エレメントの必須の VALUE 属性は、REOPT オプションの設定を指定します。この属性は、値 ONCE または値 ALWAYS をとることができます。値 ONCE は、最初の一連のホスト変数またはパラメーター・マーカー値用にステートメントを最適化することを指示します。値 ALWAYS は、一連のホスト変数またはパラメーター・マーカー値ごとに、ステートメントを最適化することを指示しま

す。指定可能な各 REOPT バインド・オプションの設定の動作に関する詳細な情報は、「*パフォーマンス・ガイド* 」に記載されています。

RTS 要求:

RTS 一般要求エレメントを使用して、リアルタイム統計収集を有効または無効にすることができます。また、リアルタイム統計収集が行われる時間を制限するためにも使用できます。特定の照会またはワークロードでは、ステートメント・コンパイル時に余分なオーバーヘッドが生じないようにするために、リアルタイム統計収集を無効にするか、それに費やす時間を制限することが望ましい場合もあります。

```
<!--*******************************************************************************************--> ¥
<!-- RTS general request element to enable, disable or provide a time budget for              --> ¥
<!-- real-time statistics collection.                                                         --> ¥
<!-- OPTION attribute allows enabling or disabling real-time statistics.                      --> ¥
<!-- TIME attribute provides a time budget in milliseconds for real-time statistics collection.--> ¥
<!--******************************************************************************************* --> ¥
<xs:complexType name="rtsType">
   <xs:attribute name="OPTION" type="optionType" use="optional" default="ENABLE"/>
   <xs:attribute name="TIME" type="xs:nonNegativeInteger" use="optional"/>
</xs:complexType>
```

説明

RTS 一般要求エレメントには、2 つのオプション属性があります。

- OPTION 属性は、リアルタイム統計収集を有効または無効にするために使用します。可能な値は、ENABLE または DISABLE です。オプションが指定されない場合のデフォルトは、ENABLE です。
- TIME 属性は、単一ステートメントで、ステートメント・コンパイル時にリアルタイム統計収集に費やす最大時間をミリ秒単位で指定します。

OPTION 属性で ENABLE を指定する場合、対応する構成パラメーターで自動統計収集およびリアルタイム統計を有効にする必要があります。これらを有効にしなかった場合、最適化ガイドラインが適用されず、警告メッセージ SQL0437W (理由コード 13) を受け取ります。

例えば以下の RTS 要求は、リアルタイム統計収集を有効にして、リアルタイム統計収集の時間を 3.5 秒に制限します。

```
<RTS OPTION="ENABLE" TIME="350" />
```

照会書き直しガイドラインの XML スキーマ: 照会書き直しガイドラインは、照会書き直し最適化フェーズに作用します。これは、*rewriteRequest* グループで構成されます。

rewriteRequest グループは、一連の有効な書き直し要求エレメントの選択項目を定義します。書き直し要求は、*INLIST2JOIN*、*SUBQ2JOIN*、*NOTEX2AJ*、および *NOTIN2AJ* です。

```
XML Schema

   <xs:group name="rewriteRequest">
      <xs:sequence>
         <xs:element name="INLIST2JOIN" type="inListToJoinType" minOccurs="0"/>
         <xs:element name="SUBQ2JOIN"   type="subqueryToJoinType" minOccurs="0"/>
         <xs:element name="NOTEX2AJ"    type="notExistsToAntiJoinType" minOccurs="0"/>
         <xs:element name="NOTIN2AJ"    type="notInToAntiJoinType" minOccurs="0"/>
      </xs:sequence>
   </xs:group>
```

説明

- INLIST2JOIN: IN-LIST から結合への書き直しトランスフォーメーションを有効または無効にします。これはステートメント・レベル・ガイドラインとして使用され、IN-LIST から結合へのトランスフォーメーションが考慮されるように、照会内のすべての IN-LIST 述部を有効または無効にします。また、述部レベル・ガイドラインとしても使用され、IN-LIST から結合へのトランスフォーメーションが考慮されるように、指定された IN-LIST 述部を有効または無効にします。ステートメント・レベルと述部レベルの両方のガイドラインが指定されている場合、述部ガイドラインはステートメント・レベル・ガイドラインをオーバーライドします。
- SUBQ2JOIN: 副照会から結合への書き直しトランスフォーメーションを有効または無効にします。これはステートメント・レベル・ガイドラインとしてのみ使用され、副照会から結合への書き直しトランスフォーメーションが考慮されるように、照会内のすべての副照会を有効または無効にします。
- NOTEX2AJ: NOT-EXISTS からアンチ結合への書き直しトランスフォーメーションを有効または無効にします。これはステートメント・レベル・ガイドラインとしてのみ使用され、NOT-EXISTS からアンチ結合への書き直しトランスフォーメーションが考慮されるように、照会内のすべての NOT-EXISTS 副照会を有効または無効にします。
- NOTIN2AJ: NOT-IN からアンチ結合への書き直しトランスフォーメーションを有効または無効にします。これはステートメント・レベル・ガイドラインとしてのみ使用され、NOT-IN からアンチ結合への書き直しトランスフォーメーションが考慮されるように、照会内のすべての NOT-IN 副照会を有効または無効にします。

IN-LIST から結合への書き直し要求: *INLIST2JOIN* 要求エレメントは、IN-LIST 述部から結合への書き直しトランスフォーメーションを有効または無効にするために使用できます。これは、ステートメント・レベル・ガイドラインまたは述部レベル・ガイドラインとして指定できます。述部レベル・ガイドラインの場合、オプション ENABLE を指定した 1 つの最適化ガイドラインだけが照会内で許可されます。

これは、複合タイプ *inListToJoinType* によって以下のように定義されます。

```
XML Schema

   <xs:complexType name="inListToJoinType">
      <xs:attribute name="OPTION" type="optionType" use="optional" default="ENABLE"/>
      <xs:attribute name="TABLE"  type="xs:string"  use="optional"/>
      <xs:attribute name="COLUMN" type="xs:string"  use="optional"/>
   </xs:complexType>
```

説明

INLIST2JOIN エレメントには 3 つのオプション属性があり、サブエレメントはありません。 OPTION 属性にはタイプ *optionType* があります。このタイプには 『ENABLE』 または 『DISABLE』 という値があります。 OPTION 属性が指定されない場合、デフォルト値は 『ENABLE』 です。表名属性および列名属性は、指定された表および列を含む IN-LIST 述部を指定するために使用されます。表名属性および列名属性が指定されていないか、または表名属性と列名属性の両方が空ストリング 『』 として指定されている場合、ステートメント・レベル・ガイドラインとして扱われます。表名が指定されているか、表名と列名の両方が指定されている

場合、述部レベル・ガイドラインとして扱われます。表名が指定されていないかまたは空ストリング 『』 として指定され、列名が指定されている場合、SQLO437W が理由コード 13 で出され、最適化ガイドラインは無視されます。

NOT-EXISTS からアンチ結合への書き直し要求: *NOTEX2AJ* 要求エレメントを使用して、NOT-EXISTS からアンチ結合への書き直しトランスフォーメーションを有効または無効にすることができます。これは、ステートメント・レベルのガイドラインとしてのみ指定することができます。

これは、複合タイプ *notExisitsToAntiJoinType* によって以下のように定義されます。

XML Schema

```
<xs:complexType name="notExistsToAntiJoinType">
   <xs:attribute name="OPTION" type="optionType" use="optional" default="ENABLE"/>
</xs:complexType>
```

説明

NOTEX2AJ エレメントには、1 つのオプション属性がありますが、サブエレメントはありません。 OPTION 属性にはタイプ *optionType* があります。このタイプには 『ENABLE』 または 『DISABLE』 という値があります。 OPTION 属性が指定されない場合、デフォルト値は 『ENABLE』 です。

NOT-IN からアンチ結合への書き直し要求: *NOTIN2AJ* 要求エレメントを使用して、NOT-IN からアンチ結合への書き直しトランスフォーメーションを有効または無効にすることができます。これは、ステートメント・レベルのガイドラインとしてのみ指定することができます。

これは、複合タイプ *notInToAntiJoinType* によって以下のように定義されます。

XML Schema

```
<xs:complexType name="notInToAntiJoinType">
   <xs:attribute name="OPTION" type="optionType" use="optional" default="ENABLE"/>
</xs:complexType>
```

説明

NOTIN2AJ エレメントには、1 つのオプション属性がありますが、サブエレメントはありません。 OPTION 属性にはタイプ *optionType* があります。このタイプには 『ENABLE』 または 『DISABLE』 という値があります。 OPTION 属性が指定されない場合、デフォルト値は 『ENABLE』 です。

副照会から結合への書き直し要求: *SUBQ2JOIN* 要求エレメントを使用して、副照会から結合への書き直しトランスフォーメーションを有効または無効にすることができます。これは、ステートメント・レベルのガイドラインとしてのみ指定することができます。

これは、複合タイプ *subqueryToJoinType* によって以下のように定義されます。

XML Schema

```
<xs:complexType name="subqueryToJoinType">
   <xs:attribute name="OPTION" type="optionType" use="optional" default="ENABLE"/>
</xs:complexType>
```

説明

SUBQ2JOIN エレメントには、1 つのオプション属性がありますが、サブエレメントはありません。 OPTION 属性にはタイプ *optionType* があります。このタイプには 『ENABLE』 または 『DISABLE』 という値があります。 OPTION 属性が指定されない場合、デフォルト値は 『ENABLE』 です。

プラン最適化ガイドライン用の XML スキーマ: プラン最適化ガイドラインは、次のようにアクセス要求と結合要求に分けることができます。

- アクセス要求 - アクセス要求は、表参照に適したアクセス方式を指定します。
- 結合要求 - 結合要求は、結合操作の実行に適した方式とシーケンスを指定します。結合要求は、その他のアクセス要求または結合要求で構成されます。

アクセス要求の選択項目の大半は、たとえば、表スキャン、索引スキャン、およびリスト・プリフェッチなどの、オプティマイザーのデータ・アクセス方式に対応します。使用可能な結合要求の大半は、ネスト・ループ結合、ハッシュ結合、およびマージ結合などの、オプティマイザーの結合方式に対応します。そのような方式については、「パフォーマンス・ガイド」で説明されています。このセクションでは、プラン最適化に作用するのに使用できるアクセス要求エレメントおよび結合要求エレメントを 1 つずつ詳述します。

アクセス要求: *accessRequest* グループは、アクセス要求エレメントの一連の有効な選択項目を定義します。アクセス要求は、ステートメント内の表参照を満たすのに適した方式を指定します。

XML Schema

```
<xs:group name="accessRequest">
   <xs:choice>
      <xs:element name="TBSCAN" type="tableScanType"/>
      <xs:element name="IXSCAN" type="indexScanType"/>
      <xs:element name="LPREFETCH" type="listPrefetchType"/>
      <xs:element name="IXAND" type="indexAndingType"/>
      <xs:element name="IXOR" type="indexOringType"/>
      <xs:element name="ACCESS" type="anyAccessType"/>
   </xs:choice>
</xs:group>
```

説明

- TBSCAN、IXSCAN、LPREFETCH、IXAND、および IXOR

  上記のエレメントは、DB2 データ・アクセス方式に対応しますが、ステートメント内で参照されるローカル表にのみ適用されます。これらのエレメントは、ニックネーム (リモート表) または派生表 (副選択の結果) を参照することはできません。

- ACCESS

  このエレメントが使用されるのは、アクセス方式とは対照的に、結合順序が最大の懸案事項である場合です。ターゲット表参照が派生表であるときは、このエレメントを使用する必要があります。オプティマイザーは、コストを使用して、ターゲット表参照用のアクセス方式を選択します。

アクセスのタイプ: TBSCAN、IXSCAN、LPREFETCH、IXAND、IXOR、および ACCESS エレメントに共通する局面は、抽象タイプ accessType によって以下のように定義されます。

XML Schema

```
<xs:complexType name="accessType" abstract="true">
   <xs:attribute name="TABLE" type="xs:string" use="optional"/>
   <xs:attribute name="TABID" type="xs:string" use="optional"/>
   <xs:attribute name="FIRST" type="xs:string" use="optional" fixed="TRUE"/>
</xs:complexType>
```

説明

すべてのアクセス要求エレメントは、複合タイプ *accessType* を拡張します。そのようなエレメントはいずれも、*TABLE* または *TABID* 属性を使用して、ターゲット表参照を指定する必要があります。 413 ページの『最適化ガイドラインでの表参照の形成』 には、アクセス要求エレメントから正しい表参照を作成する方法が説明されています。 そのようなエレメントは、オプションの *FIRST* 属性を指定することもできます。 FIRST 属性を指定する場合、その値は TRUE でなければなりません。アクセス要求エレメントに FIRST 属性を追加すると、アクセス要求のターゲットである表を、対応する FROM 節の結合シーケンス内の最初の表とするような実行プランが必要であることを指示します。 FIRST 属性を指定できるアクセスまたは結合要求は、FROM 節あたり 1 つだけです。 1 つの同じ FROM 節の表をターゲットとする複数のアクセスまたは結合要求が FIRST 属性を指定した場合、そのような要求のうちの最初のもの以外はすべて無視されて、理由コード 13 の警告 SQL0437W が出されます。

任意のアクセス要求: *ACCESS* アクセス要求エレメントを使用して、オプティマイザーが表へのアクセスに適した方式をコスト・ベース方式で選択するように要求することができます。通常、この選択項目を使用するのは、ローカル表をステートメント内の他の表に結合する方法を指示するだけのアクセス要求を指定する場合です。派生表への参照を行うときは、このアクセス要求エレメントを使用する必要があります。派生表は、別の副選択の結果です。 ACCESS アクセス要求エレメントは、複合タイプ *anyAccessType* によって以下のように定義されます。

XML Schema

```
<xs:complexType name="anyAccessType">
   <xs:complexContent>
      <xs:extension base="accessType"/>
   </xs:complexContent>
</xs:complexType>
```

説明

複合タイプ *anyAccessType* は、抽象タイプ *accessType* の単なる拡張です。新規のエレメントや属性は追加されていません。

索引 ANDing アクセス要求: *indexAnding* アクセス要求エレメントを使用して、オプティマイザーがローカル表にアクセスする際に索引 ANDing データ・アクセス方式を使用するように要求することができます。これは、複合タイプ *indexAndingType* によって以下のように定義されます。

XML Schema

```
<xs:complexType name="indexAndingType">
```

```
    <xs:complexContent>
       <xs:extension base="accessType">
          <xs:sequence minOccurs="0">
             <xs:element name="INDEX" type="indexType" minOccurs="2" maxOccurs="unbounded"/>
          </xs:sequence>
          <xs:attribute name="INDEX" type="xs:string" use="optional"/>
       </xs:extension>
    </xs:complexContent>
 </xs:complexType>

    <xs:complexType name="indexType">
    <xs:attribute name="IXNAME" type="xs:string" use="optional"/>
 </xs:complexType>
```

説明

複合タイプ *indexAndingType* は、オプションの *INDEX* 属性およびオプションの INDEX サブエレメントを追加して、抽象タイプ *localAccessType* を拡張します。 INDEX 属性を使用して、索引の ANDing 操作で使用される最初の索引を指定することができます。 INDEX 属性を使用すると、オプティマイザーはコスト・ベース方式で追加の索引およびアクセス・シーケンスを選択します。 INDEX エレメントを使用して、正確な一連の索引およびアクセス・シーケンスを指定することができます。 INDEX サブエレメントの出現順は、個々の索引スキャンを実行すべき順序を示しています。 INDEX サブエレメントを指定すると、それによって INDEX 属性の指定が置き換えられます。

- 索引を指定しないと、オプティマイザーは索引およびアクセス・シーケンスを両方ともコスト・ベース方式で選択します。
- 属性またはサブエレメントを使用して索引を指定する場合、その索引は TABLE または TABID 属性で識別される表で定義されている索引を識別する必要があります。
- 定義された索引が表にない場合、アクセス要求は無視されて、理由コード 13 の警告 SQL0437W が出されます。

索引 ANDing アクセス要求では、レコード索引の前にブロック索引が置かれている必要があります。この要件が満たされないと、理由コード 13 の警告 SQL0437W が出されます。索引 ANDing アクセス方式では、索引付けが可能な述部が各索引に少なくとも 1 つ必要です。必須の述部が存在しないために索引 ANDing が適格ではない場合、アクセス要求は無視されて、理由コード 13 の警告 SQL0437W が出されます。ステートメントに関して有効な検索スペース内に索引 ANDing データ・アクセス方式がない場合、アクセス要求は無視されて、理由コード 13 の警告 SQL0437W が出されます。

索引 ORing アクセス要求: *IXOR* アクセス要求エレメントを使用して、オプティマイザーがローカル表にアクセスする際に索引 ORing データ・アクセス方式を使用するように要求することができます。これは、複合タイプ *indexOringType* によって以下のように定義されます。

XML Schema

```
<xs:complexType name="indexOringType">
   <xs:complexContent>
      <xs:extension base="accessType"/>
   </xs:complexContent>
</xs:complexType>
```

説明

複合タイプ *indexOringType* は、抽象タイプ *accessType* の単なる拡張です。新規のエレメントや属性は追加されていません。ステートメントに関して有効な検索スペース内に索引 ORing アクセス方式がない場合、アクセス要求は無視されて、理由コード 13 の警告 SQL0437W が出されます。オプティマイザーは、索引 ORing 操作で使用する述部および索引をコスト・ベース方式で選択します。索引 ORing アクセス方式では、少なくとも 1 つの索引付けが可能な IN 述部、または索引付けおよび論理 OR 演算による結合が可能な語を伴う述部が必要です。必須の述部または索引が存在しないために索引 ORing が適格ではない場合、アクセス要求は無視されて、理由コード 13 の警告 SQL0437W が出されます。

索引スキャン・アクセス要求: *IXSCAN* アクセス要求エレメントを使用して、オプティマイザーがローカル表にアクセスする際に索引スキャンを使用するように要求することができます。これは、複合タイプ *indexScanType* によって以下のように定義されます。

XML Schema

```
<xs:complexType name="indexScanType">
   <xs:complexContent>
      <xs:extension base="accessType">
         <xs:attribute name="INDEX" type="xs:string" use="optional"/>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>
```

説明

複合タイプ *indexScanType* は、オプションの *INDEX* 属性を追加して、抽象 accessType を拡張します。 INDEX 属性は、表へのアクセスで使用される索引の名前を指定します。

- ステートメントに関して有効な検索スペース内に索引スキャン・アクセス方式がない場合、アクセス要求は無視されて、理由コード 13 の警告 SQL0437W が出されます。
- INDEX 属性を指定する場合、TABLE または TABID 属性で識別される表で定義されている索引を識別する必要があります。
- 索引が存在しない場合、アクセス要求は無視されて、理由コード 13 の警告 SQL0437W が出されます。
- INDEX 属性を指定しないと、オプティマイザーはコスト・ベース方式で索引を選択します。
- ターゲット表で索引が定義されていない場合、アクセス要求は無視されて、理由コード 13 の警告 SQL0437W が出されます。

リスト・プリフェッチ・アクセス要求: *listPrefetch* エレメントを使用して、オプティマイザーがローカル表にアクセスする際にリスト・プリフェッチ索引スキャンを使用するように要求することができます。これは、複合タイプ *listPrefetchType* によって以下のように定義されます。

XML Schema

```
<xs:complexType name="listPrefetchType">
   <xs:complexContent>
      <xs:extension base="accessType">
```

```
        <xs:attribute name="INDEX" type="xs:string" use="optional"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
```

説明

複合タイプ *listPrefetchType* は、オプションの *INDEX* 属性を追加して、抽象タイプ *accessType* を拡張します。 INDEX 属性は、表へのアクセスで使用される索引の名前を指定します。

- ステートメントに関して有効な検索スペース内にリスト・プリフェッチ・アクセス方式がない場合、アクセス要求は無視されて、理由コード 13 の警告 SQL0437W が出されます。
- リスト・プリフェッチ・アクセス方式では、索引付けが可能な述部が少なくとも 1 つ必要です。
- 必須の述部が存在しないためにリスト・プリフェッチが適格ではない場合、アクセス要求は無視されて、理由コード 13 の警告 SQL0437W が出されます。
- INDEX 属性を指定する場合、TABLE または TABID 属性で指定される表で定義されている索引を識別する必要があります。
- 索引が存在しない場合、アクセス要求は無視されて、理由コード 13 の警告 SQL0437W が出されます。
- INDEX 属性を指定しないと、オプティマイザーはコスト・ベース方式で索引を選択します。
- 適切に定義された索引がターゲット表にない場合、アクセス要求は無視されて、理由コード 13 の警告 SQL0437W が出されます。

表スキャン・アクセス要求: *TBSCAN* アクセス要求エレメントを使用して、オプティマイザーがローカル表にアクセスする際に順次表を使用ように要求することができます。これは、複合タイプ *tableScanType* によって以下のように定義されます。

XML Schema

```
  <xs:complexType name="tableScanType">
    <xs:complexContent>
      <xs:extension base="accessType"/>
    </xs:complexContent>
  </xs:complexType>
```

説明

複合タイプ *tableScanType* は、上記の抽象タイプ *accessType* の単なる拡張です。新規のエレメントや属性は追加されていません。ステートメントに関して有効な検索スペース内に表スキャン・アクセス方式がない場合、アクセス要求は無視されて、理由コード 13 の警告 SQL0437W が出されます。

結合要求: *joinRequest* グループは、結合要求エレメントの一連の有効な選択項目を定義します。結合要求は、2 つの表の結合用として望ましい方式を指定します。

XML Schema

```
  <xs:group name="joinRequest">
    <xs:choice>
      <xs:element name="NLJOIN" type="nestedLoopJoinType"/>
      <xs:element name="HSJOIN" type="hashJoinType"/>
```

```
        <xs:element name="MSJOIN" type="mergeJoinType"/>
        <xs:element name="JOIN" type="anyJoinType"/>
     </xs:choice>
  </xs:group>
```

説明

NLJOIN、*MSJOIN*、および *HSJOIN* 結合要求エレメントは、それぞれネスト・ループ、マージ、およびハッシュ結合方式に対応します。「パフォーマンス・ガイド 」に、これらの結合方式に関する適格の要件と実行の特性がさらに詳しく説明されています。 *JOIN* 結合要求エレメントは、オプティマイザーが結合方式を自由に選択できることを示します。この選択項目を使用するのは、特定の結合順序を第一の懸案事項として指定する場合です。

どの結合要求エレメントにも、結合操作の入力表を表す 2 つのサブエレメントが入っています。結合要求はまた、オプションの FIRST 属性を指定することもできます。

任意の結合要求: *JOIN* 結合要求エレメントを使用して、オプティマイザーがオプティマイザーによって選択された任意の結合方式を使用して 2 つの表を、特定の順序で結合するように要求することができます。入力表は、アクセス要求サブエレメントでの指定どおりに、ローカル表または派生表のいずれでも良く、結合要求サブエレメントでの指定どおりに結合操作の結果になります。 *JOIN* 結合要求エレメントは、複合タイプ *anyJoinType* によって以下のように定義されます。

XML Schema

```
  <xs:complexType name="anyJoinType">
     <xs:complexContent>
        <xs:extension base="joinType"/>
     </xs:complexContent>
  </xs:complexType>
```

説明

複合タイプ *anyJoinType* は、抽象タイプ *joinType* の単なる拡張です。新規のエレメントや属性は追加されていません。

ハッシュ結合要求: *HSJOIN* 結合要求エレメントを使用して、オプティマイザーがハッシュ結合方式を使用して 2 つの表を結合するように要求することができます。入力表は、アクセス要求サブエレメントでの指定どおりに、ローカル表または派生表のいずれでも良く、結合要求サブエレメントでの指定どおりに結合操作の結果になります。 *HSJOIN* 結合要求エレメントは、複合タイプ *hashJoinType* によって以下のように定義されます。

XML Schema

```
  <xs:complexType name="hashJoinType">
     <xs:complexContent>
        <xs:extension base="joinType"/>
     </xs:complexContent>
  </xs:complexType>
```

説明

複合タイプ *hashJoinType* は、抽象タイプ *joinType* の単なる拡張です。新規のエレメントや属性は追加されていません。ステートメントに関して有効な検索スペース

内にハッシュ結合方式がない場合、結合要求は無視されて、理由コード 13 の警告 SQL0437W が出されます。

マージ結合要求: *MSJOIN* 結合要求エレメントを使用して、オプティマイザーがマージ結合方式を使用して 2 つの表を結合するように要求することができます。入力表は、アクセス要求サブエレメントでの指定どおりに、ローカル表または派生表のいずれでも良く、結合要求サブエレメントでの指定どおりに結合操作の結果になります。 *MSJOIN* 結合要求エレメントは、複合タイプ *mergeJoinType* によって以下のように定義されます。

XML Schema

```
<xs:complexType name="mergeJoinType">
   <xs:complexContent>
      <xs:extension base="joinType"/>
   </xs:complexContent>
</xs:complexType>
```

説明

複合タイプ *mergeJoinType* は、抽象タイプ *joinType* の単なる拡張です。新規のエレメントや属性は追加されていません。ステートメントに関して有効な検索スペース内にマージ結合方式がない場合、結合要求は無視されて、理由コード 13 の警告 SQL0437W が出されます。

ネスト・ループ結合要求: *NLJOIN* 結合要求エレメントを使用して、オプティマイザーがネスト・ループ結合方式を使用して 2 つの表を結合するように要求することができます。入力表は、アクセス要求サブエレメントでの指定どおりに、ローカル表または派生表のいずれでも良く、結合要求サブエレメントでの指定どおりに結合操作の結果になります。 *NLJOIN* 結合要求エレメントは、複合タイプ *nestedLoopJoinType* によって以下のように定義されます。

XML Schema

```
<xs:complexType name="nestedLoopJoinType">
   <xs:complexContent>
      <xs:extension base="joinType"/>
   </xs:complexContent>
</xs:complexType>
```

説明

複合タイプ *nestedLoopJoinType* は、抽象タイプ *joinType* の単なる拡張です。新規のエレメントや属性は追加されていません。ステートメントに関して有効な検索スペース内にネスト・ループ結合方式がない場合、結合要求は無視されて、理由コード 13 の警告 SQL0437W が出されます。

結合要求のタイプ: すべての結合要求エレメントに共通する局面は、抽象タイプ *joinType* によって以下のように定義されます。

XML Schema

```
<xs:complexType name="joinType" abstract="true">
   <xs:choice minOccurs="2" maxOccurs="2">
      <xs:group ref="accessRequest"/>
      <xs:group ref="joinRequest"/>
   </xs:choice>
   <xs:attribute name="FIRST" type="xs:string" use="optional" fixed="TRUE"/>
</xs:complexType>
```

説明

joinType を拡張する結合要求エレメントには、正確に 2 つのサブエレメントがなければなりません。そのサブエレメントのどちらも、グループ *accessRequest* から選択されるアクセス要求エレメント、またはグループ *joinRequest* から選択される別の結合要求エレメントにすることができます。結合要求に最初に現れるサブエレメントが、結合操作の外部表を指定し、2 番目のエレメントが内部表を指定します。
FIRST 属性を指定する場合、その値は TRUE でなければなりません。結合要求エレメントに FIRST 属性を追加すると、結合要求のターゲットである表を、対応する FROM 節の結合シーケンスの最外部の表とするような実行プランが必要であることを指示します。 FIRST 属性を指定できるアクセスまたは結合要求は、FROM 節あたり 1 つだけです。 1 つの同じ FROM 節の表をターゲットとする複数のアクセスまたは結合要求が FIRST 属性を指定する場合、そのような要求のうちの最初のもの以外はすべて無視されて、理由コード 13 の警告 SQL0437W が出されます。

SYSTOOLS.OPT_PROFILE 表

SYSTOOLS.OPT_PROFILE 表には、すべての最適化プロファイル・ファイルが含まれています。表を作成するには、次の 2 つの方法があります。

- 次のようにして、SYSINSTALLOBJECTS プロシージャーを呼び出します。

```
db2 "call sysinstallobjects('opt_profiles', 'c', '', '')"
```

- 次のようにして CREATE TABLE コマンドを発行します。

```
CREATE TABLE SYSTOOLS.OPT_PROFILE (
            SCHEMA  VARCHAR(128) NOT NULL,
            NAME    VARCHAR(128) NOT NULL,
            PROFILE BLOB (2M)    NOT NULL,
            PRIMARY KEY ( SCHEMA, NAME )
          );
```

SYSTOOLS.OPT_PROFILE 表の列は次のように定義されます。

SCHEMA

- 最適化プロファイルのスキーマ修飾子を指定します。スキーマ名には最高 30 文字の英数字または下線文字を含めることができますが、それを以下の示すように VARCHAR(128) として定義できます。

名前

- 最適化プロファイルのベース名を指定します。名前には最高 128 文字の英数字または下線文字を含めることができます。

PROFILE

- 最適化プロファイルを定義する XML 文書。

注:

1. 最適化プロファイルは複数の異なるコンテキストで参照できます。SYSTOOLS.OPT_PROFILE 表の NAME および SCHEMA 列は共に最適化プロファイル名の 2 つの部分を指定します。 PROFILE 列には、最適化プロファイルを定義する XML 文書が含まれます。
2. 表スペースまたはパーティション・グループの制限はありません。

最適化プロファイル・キャッシュをフラッシュするためのトリガー:
SYSTOOLS.OPT_PROFILE 表内の項目が更新または削除されるときに、プロファイル・キャッシュが自動的にフラッシュされることを確実にするには、以下の SQL プロシージャーおよびトリガーを作成する必要があります。

```
CREATE PROCEDURE SYSTOOLS.OPT_FLUSH_CACHE( IN SCHEMA VARCHAR(128),
                                            IN NAME VARCHAR(128) )
LANGUAGE SQL
MODIFIES SQL DATA
BEGIN ATOMIC
 -- FLUSH stmt (33) + quoted schema (130) + dot (1) + quoted name (130) = 294
 DECLARE FSTMT VARCHAR(294) DEFAULT 'FLUSH OPTIMIZATION PROFILE CACHE '; --

 IF NAME IS NOT NULL THEN
   IF SCHEMA IS NOT NULL THEN
     SET FSTMT = FSTMT || '"' || SCHEMA || '".'; --
   END IF; --

   SET FSTMT = FSTMT || '"' || NAME || '"'; --

   EXECUTE IMMEDIATE FSTMT; --
 END IF; --
END;

CREATE TRIGGER SYSTOOLS.OPT_PROFILE_UTRIG AFTER UPDATE ON SYSTOOLS.OPT_PROFILE
REFERENCING OLD AS O
FOR EACH ROW
  CALL SYSTOOLS.OPT_FLUSH_CACHE( O.SCHEMA, O.NAME );

CREATE TRIGGER SYSTOOLS.OPT_PROFILE_DTRIG AFTER DELETE ON SYSTOOLS.OPT_PROFILE
REFERENCING OLD AS O
FOR EACH ROW
  CALL SYSTOOLS.OPT_FLUSH_CACHE( O.SCHEMA, O.NAME );
```

SYSTOOLS.OPT_PROFILE 表の管理: 最適化プロファイル・ファイルは、固有のスキーマ修飾名に関連付けて、SYSTOOLS.OPT_PROFILE 表に保管する必要があります。 LOAD、IMPORT、および EXPORT コマンドを使用して、その表内のファイルを管理できます。例えば、IMPORT コマンドを任意の DB2 クライアントから使用して、データをファイルから SYSTOOLS.OPT_PROFILE 表に挿入するかまたは更新できます。 EXPORT コマンドを使用して、プロファイルを OPT_PROFILE 表からファイルに取り出すことができます。

以下は、3 つの新しい行を別個の入力ファイルから SYSTOOLS.OPT_PROFILE 表に挿入する例です。ここでは、ファイルが現行ディレクトリーにあることを想定しています。

1. 個々の行にスキーマ、名前、およびファイル名を指定して入力ファイル (例えば、profiledata) を作成します。

   ```
   "ROBERT","PROF1","ROBERT.PROF1.xml"
   "ROBERT","PROF2","ROBERT.PROF2.xml"
   "DAVID", "PROF1","DAVID.PROF1.xml"
   ```

2. IMPORT コマンドを実行します。

   ```
   IMPORT FROM profiledata OF DEL MODIFIED BY LOBSINFILE INSERT INTO SYSTOOLS.OPT_PROFILE
   ```

既存の行を更新するには、まずそれらを削除して上記のように再び挿入するか、または INSERT_UPDATE オプションに IMPORT を指定して使用できます。

```
IMPORT FROM profiledata OF DEL MODIFIED BY LOBSINFILE
  INSERT_UPDATE INTO SYSTOOLS.OPT_PROFILE
```

プロファイルから ROBERT.PROF1 を ROBERT.PROF1.xml に取り出すには、以下のようにします。この場合は、プロファイルが 32,700 バイトより小さいことを想定しています。

```
EXPORT TO ROBERT.PROF1.xml OF DEL SELECT PROFILE FROM SYSTOOLS.OPT_PROFILE
  WHERE SCHEMA='ROBERT' AND NAME='PROF1'
```

32,700 バイトを超えるデータをエクスポートする場合、または詳細については、「コマンド・リファレンス 」の EXPORT コマンドに関する資料を参照してください。

照会の最適化に影響を与える構成パラメーター

構成パラメーターの中には、 SQL または XQuery 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パーティション・データベース環境に適用されるものですが、一部にはパーティション・データベース環境のみに適用されるものもあります。ハードウェアが同機種のパーティション・データベース環境の場合は、各パラメーターに使用する値をすべてのデータベース・パーティションで同じにする必要があります。

注: 構成パラメーターを動的に変更する場合、パッケージ・キャッシュにある以前のアクセス・プランのために、オプティマイザーが変更されたパラメーター値を即時には読み取れない可能性があります。パッケージ・キャッシュをリセットするには、 FLUSH PACKAGE CACHE コマンドを実行してください。

フェデレーテッド・システムで、照会の大部分がニックネームにアクセスする場合、環境を変更する前に、送信する照会のタイプを評価してください。例えば、フェデレーテッド・データベースでは、 DBMS およびフェデレーテッド・システム内のデータといったデータ・ソースから、バッファー・プールがページをキャッシュに入れることはありません。このため、バッファーのサイズを増やしても、オプティマイザーがニックネームを含む照会のアクセス・プランを選択する際に、追加のアクセス・プランの選択肢を考慮するとは限りません。しかし、オプティマイザーは、データ・ソース表をローカルにマテリアライズすることが、最もコストを低くする手段、またはソート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、使用可能なリソースを増やすことで、パフォーマンスが向上します。

以下の構成パラメーターまたは要因は、 SQL または XQuery 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ます。

- 作成時、または変更時に指定したバッファー・プールのサイズ。

  オプティマイザーはアクセス・プランを選択するとき、ディスクからページをバッファー・プールにフェッチするときの入出力コストを考慮し、照会を満たすために必要な入出力の数を見積もります。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。

  オプティマイザーは、SYSCAT.BUFFERPOOLS システム・カタログ表の、およびパーティション・データベース環境では SYSCAT.BUFFERPOOLDBPARTITIONS 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
– 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

- デフォルトのパーティション内並行度 (dft_degree)

  dft_degree 構成パラメーターでは、 CURRENT DEGREE 特殊レジスターおよび DEGREE BIND オプションのデフォルト値を指定することにより、並列処理を指定します。値 1 は、パーティション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、パーティション内並列処理の多重度をオプティマイザーが決定することを意味します。

  注: パーティション内並列処理は、*intra_parallel* データベース・マネージャー構成パラメーターを設定して使用可能にしない限り行われません。

- デフォルトの照会最適化クラス (dft_queryopt)

  照会最適化クラスは SQL または XQuery 照会のコンパイル時に指定することができますが、デフォルトの照会最適化クラスを設定することもできます。

- アクティブ・アプリケーションの平均数 (avg_appls)

  オプティマイザーは、選択されたアクセス・プランの実行時にどれだけのバッファー・プールが使用可能かを見積もる助けとして、 *avg_appls* パラメーターを使用します。このパラメーターに高い値を指定すると、オプティマイザーがバッファー・プールをより控えめに使用するアクセス・プランを選択するという影響が出る可能性があります。値を 1 に指定すると、オプティマイザーはバッファー・プール全体がアプリケーションによって使用可能と見なします。

- ソート・ヒープ・サイズ (sortheap)

  ソートされる行が、ソート・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、複数のソート・パスが実行され、各パスごとに行全体のうちの 1 つのサブセットがソートされることになります。各ソート・パスは、バッファー・プール内のシステム一時表に保管され、ディスクに書き込むこともできます。すべてのソート・パスが完了すると、それらのソート済みサブセットはマージされ、ソート済みの単一の行集合になります。最終的にソートされたデータ・リストを保管するためにシステム一時表が必要ではない場合は、ソートは「パイプ処理」されるものと見なされます。つまり、ソートの結果を 1 つの順次アクセスで読み取ることができます。パイプ処理されたソートは、パイプ処理ではないソート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。

  アクセス・プランを選択するとき、オプティマイザーはソート操作のコストを見積もります。それには、ソート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。

  – ソートするデータの量を見積もる。
  – *sortheap* パラメーターを見て、ソート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。

- ロック・リスト用最大ストレージ (locklist) およびエスカレーション前のロック・リストの最大パーセント (maxlocks)

分離レベルが**反復可能読み取り (RR)** である場合、オプティマイザーは *locklist* 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エスカレーションされる可能性があるかどうかを判別します。オプティマイザーは、表アクセスに関してロック・エスカレーションが起きると見積もった場合、照会実行時のロック・エスカレーションのオーバーヘッドを生じさせる代わりに、そのアクセス・プランには表レベル・ロックを選択します。

- CPU 速度 (cpuspeed)

  オプティマイザーは CPU 速度を使用して、特定の操作を実行するコストを見積もります。CPU コストの見積もり、およびさまざまな入出力コストの見積もりは、照会に対して最適のアクセス・プランを選択するのに役立ちます。

  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マイグレーションした時点で、自動的に適切な値に設定されます。このパラメーターは、テスト・システムにおいて実稼働環境のモデル化を行っている場合か、ハードウェア変更の影響を見積もっている場合でない限り、調整しないでください。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と、その環境のために選択される可能性のあるアクセス・プランを検出することができます。データベース・マネージャーがこの自動構成パラメーターの値を再計算するようにするには、-1 に設定してください。

- 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、オプティマイザー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が、複合 SQL または XQuery 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警告を受け取ることがあります。例えば、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、要求した量より少ないことを示す場合があります。

- 通信スピード (comm_bandwidth)

  通信スピードは、オプティマイザーがアクセス・プランを決めるのに使用されます。オプティマイザーはこのパラメーターの値を使用して、パーティション・データベース環境のデータベース・パーティション・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。

- アプリケーション・ヒープ・サイズ (applheapsz)

  大規模なスキーマには、アプリケーション・ヒープ内に十分な空間が必要です。

照会の最適化に影響を与えるデータベース・パーティション・グループ

パーティション・データベース環境では、オプティマイザーは表のコロケーションを認識し、照会に対する最適のアクセス・プランを判別する際にそのコロケーションを使用します。表が頻繁に結合照会に関係する場合、それらの表は、結合される各表にある行が同じデータベース・パーティションにあるように、パーティショ

ン・データベース環境のデータベース・パーティション間で分割する必要があります。結合操作を実行するときに、結合される両方の表にあるデータのコロケーションによって、データをあるデータベース・パーティションから別のデータベース・パーティションに移動されなくなります。同じデータベース・パーティション・グループに両方の表を置き、表のデータが確実に同じパーティションに入れられるようにしてください。

パーティション・データベース環境では、表のサイズによって、データをより多くのデータベース・パーティションに分散すると、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、および結合が必要かどうかといった照会のタイプはすべて照会のコストに影響を与えます。

複数述部の列相関

アプリケーションに 2 つ以上の結合述部が 2 つの表を結合するような結合で構成される照会がある場合があります。照会が別の表の似ていて関連した列の間のリレーションシップを決定する必要がある場合、これ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を作るメーカーのことを考えてみましょう。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
```

オプティマイザーは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と想定します。つまり、それぞれの色ごとにすべてのレベルの伸縮性があり、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あると想定します。それから、表ごとに伸縮性のレベルの数と異なる色の数に基づくカタログ統計情報を使用して、述部の対の全体的な選択可能性を見積もります。この見積もりに基づいて、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によって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。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。こうなるとオプティマイザーは最良のプランを選択することができない場合があります。たとえば、マージ結合の方が速いのに、ネスト・ループ結合プランを選択してしまうかもしれません。

DB2 のオプティマイザーは、列に対して索引を定義する場合、あるいは適切な列のグループ列統計を収集して保守する場合、結合述部の相関の検出と補正を試みます。

たとえば、上記の伸縮性の例では、以下の索引の一方または両方を定義できます。

```
IX1 PRODUCT.COLOR, PRODUCT.ELASTICITY
```

または

```
IX2 RAWMATERIAL.COLOR, RAWMATERIAL.ELASTICITY
```

あるいは、両方とも定義します。

オプティマイザーが相関を検出できるように、この索引の非組み込み列は相関する列だけでなければなりません。索引だけのスキャンができるように、索引に組み込み列を含めることもできます。結合述部に 3 つ以上の相関列がある場合、そのすべての列を含む索引を定義してください。

オプティマイザーが索引キーのカーディナリティーを考慮して相関を検出するために満たさなければならない条件の 1 つとして、個々の列中の固有値の数が、同じ表の個々の列の固有値の数より大きくなければなりません。たとえば、上記のように IX1 と IX2 を定義したとします。PRODUCT.COLOR 中の固有値の数が RAWMATERIAL.COLOR 中の固有値の数より少なく、PRODUCT.ELASTICITY 中の固有値の数が RAWMATERIAL.ELASTICITY 中の固有値の数より少ない場合、オプティマイザーは IX2 を使用して相関を検出します。列のカーディナリティーを比較すると、PRODUCT 列中の固有値は RAWMATERIAL 列中の固有値に含まれる可能性が高いですが、確実ではありません。あるドメインが別のドメインを想定する可能性をさらに高めるために、オプティマイザーは HIGH2KEY および LOW2KEY 統計でこれらの索引列を比較することもあります。

該当する索引の作成後に、表に関する統計が正確で最新であることを確認してください。

オプティマイザーはユニーク索引の統計表中の FIRSTnKEYCARD および FULLKEYCARD 列の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。

別の方法としては、列のセットに関する列グループ統計を収集することができます。上記の伸縮性の例では、 PRODUCT.COLOR、PRODUCT.ELASTICITY または RAWMATERIAL.COLOR、RAWMATERIAL.ELASTCITY (あるいはその両方) 列に関する統計を収集できます。

列グループ統計は、RUNSTATS の「ON COLUMNS」オプションを使用して収集します。たとえば、PRODUCT.COLOR および PRODUCT.ELASTICITY に関する列グループ統計を収集するには、次の RUNSTATS コマンドを発行します。

```
RUNSTATS ON TABLE product ON COLUMNS ((color, elasticity))
```

単純な等価述部の相関

JOIN 述部相関に加えて、オプティマイザーはタイプ COL = の単純な等価述部に関する相関も管理します。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、YEAR、COLOR、および STYLE (セダン、ステーション・ワゴン、スポーツ・カーなど) 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 の索引が存在するか、列グループ統計が収集される場合、オプティマイザーはその索引または列の統計情報を使用して、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。述部がローカル等価述部の場合、オプティマイザーが調整を行うためのユニーク索引を持つ必要はありません。

索引および列のグループ統計を使用してグループ化 KEYCARD を計算する

照会においてデータを特定の方法でグループ化する必要がある場合、オプティマイザーは個々のグループの数、つまり*グループ化 KEYCARD* を計算する必要があります。グループ化要件は、GROUP BY または DISTINCT などの操作の結果として生じます。

次の照会を考えてみましょう。

```
SELECT DEPTNO, YEARS, AVG(SALARY)
  FROM EMPLOYEE
 GROUP BY DEPTNO, MGR, YEAR_HIRED
```

索引または列のグループ統計を使用しない場合、オプティマイザーによって見積もられるグループの数 (この場合は、戻される行の数も) は、DEPTNO、MGR、および YEAR_HIRED の個々の値の数の積になります。この見積もりは、グループ化キー列が独立していることを前提としています。ただし、各管理者が 1 つの部門だけを管理している場合には、この前提事項に該当しない可能性があります。さらに、各部門が従業員の雇用を毎年行っていない可能性もあります。このように、DEPTNO、MGR、および YEAR_HIRED の個々の値の積は、個々のグループの実際の数よりも多く見積もられている可能性があります。

ここで、索引が次のように定義されているとします。

```
INDEX IX1:  DEPTNO, MGR, YEAR_HIRED
```

この場合、IX1 の FULLKEYCARD は、上の照会の個々のグループの実際数をオプティマイザーに提供します。

別の索引定義について考えてみてください。

```
INDEX IX2:  DEPTNO, MGR, YEAR_HIRED, COMM
```

IX2 の FIRST3KEYCARD は (DEPTNO、MGR、YEAR_HIRED) の個々のグループの数を示すため、IX2 もグループ化 KEYCARD を計算するのに役立ちます。

索引統計 (FIRST2KEYCARD、FIRST3KEYCARD、FIRST4KEYCARD、および FULLKEYCARD) に加えて、グループ化 KEYCARD を同様の方法で計算するため

に、オプティマイザーは列グループ統計を活用する場合もあります。以下のように、DEPTNO、MGR、および YEAR_HIRED で収集された列グループ統計は、上で示した IX1 および IX2 と同じ効果を提供します。

```
RUNSTATS ON TABLE EMPLOYEE ON COLUMNS ((DEPTNO, MGR, YEAR_HIRED))
```

グループ化キーが 5 つ以上の列で構成されている場合、列グループ統計の収集が適している場合があることに注意してください。これは RUNSTATS が、最初の 4 つの列に関する統計、および指定された索引の完全索引キー列の統計のみを収集するためです。

統計ビュー

統計ビューでは、オプティマイザーがより正確に基数推定値を計算することを可能にします。 カーディナリティー見積もりとは、述部が適用されるかまたは集約が実行された後に、オプティマイザーが統計を使用して部分的な照会結果のサイズを判別するプロセスのことです。基数推定の正確性は、述部および使用可能な統計に依存しています。統計は、列内のデータ値の分布を表すために使用でき、これによりデータ値が不均一に分散している場合の基数推定を向上させることができます。また統計は、列セットにおける明確な値の数を表すためにも使用でき、これにより列が統計的に相関がある場合の基数推定を向上させることができます。ただし、複雑な関係は統計で表すことができない場合があります。相関およびスキュー属性 (例えば、*make* = 'Honda' AND *model* = 'Accord') を含む述部または集合体のフィルタリング効果、式結果 (例えば、*price* > MSRP + Dealer_markup) との比較、複数の表にわたる関係 (例えば、*product.name* = 'Alloy wheels' および *product.key* = *sales.product_key*)、または 独立した属性および単純な比較操作を行う述部または集合以外のものが、この複雑な関係に含まれます。

統計ビューとは、関連した統計値と共に表示されるビューのことです。このビューを使用して、ビューの定義が照会の定義とオーバーラップする照会の基数推定を向上させることができます。これが強力な機能である理由は、このビューが、1 つ以上の表が関係した複雑な (相関の可能性がある) 述部セットが指定された照会で、基数推定値を決定するための正確な統計値をオプティマイザーに提供することにあります。

統計ビューを、最適化する対象の照会で直接参照する必要はありません。多くの場合、ビューの統計は、その定義が照会の定義とオーバーラップする場合に活用することができます。この新しい機能を活用するには、ALTER VIEW ステートメントを使用する最適化でビューが使用可能で、ビュー上の統計についてシステム・カタログ表にデータが取り込まれている必要があります。

統計ビューの使用

照会を最適化するために統計を使用するには、最適化でビューが使用可能でなければなりません。 最適化で使用可能なビューは、*統計ビュー*です。統計ビューではないビューは、最適化では無効であると見なされます。*通常ビュー*という用語は、最適化では無効であるビューのことを指します。ビューは、最初に作成された時は、最適化で無効になっています。

- 最適化のためにビューを有効にするには、ビューと、ビューを定義する基となる表の両方に対する ALTER 特権が必要です。

- ビューのために RUNSTATS を呼び出すには、次のいずれかの権限が必要です。
  - SYSADM
  - SYSCTRL
  - SYSMAINT
  - DBADM
  - ビューの CONTROL 特権

  加えて、ビューから行をアクセスするための適切な特権が必要です。具体的には、ビューの定義で参照される表、ビュー、またはニックネームごとに、次のいずれかの特権が必要です。
  - SYSADM
  - DBADM
  - CONTROL
  - SELECT

以下のいずれかが真の場合、ビューを最適化のために有効にすることはできません。

- ビューが直接または間接に MQT を参照する(MQT または統計ビューは統計ビューを参照できます)。
- 動作不能ビューである。
- 型付きビューである。
- 同じ ALTER VIEW ステートメントの中に別のビュー変更がある。

最適化を有効とするために変更されるビューの定義が、以下の条件のいずれかに該当する場合、ALTER VIEW ENABLE OPTIMIZATION は警告と共に正常に完了しますが、オプティマイザーはその統計を活用しません。

- 総計機能または明確な操作が含まれる。
- union、except、または intersect 操作が含まれる。
- スカラー集合 (OLAP) 関数が含まれる。

1. 最適化のためにビューを有効にします。

   ALTER VIEW ステートメントで新しい ENABLE OPTIMIZATION 節を使用すると、ビューを最適化のために有効とすることができます。最適化のために有効となったビューは、ALTER VIEW ステートメントで DISABLE OPTIMIZATION 節を使用すると、後で最適化のために無効にすることができます。たとえば、最適化のためにビュー myview を有効にするには、次のように入力します。

   ```
   ALTER VIEW myview ENABLE QUERY OPTIMIZATION
   ```

   最適化のために有効にしたビューについては、対応する SYSTABLES 項目の PROPERTY 列の文字位置 13 が 'Y' になっています。最適化のために無効とされたビューについては、対応する SYSTABLES 項目 のPROPERTY 列の文字位置 13 がブランクになっています。
2. RUNSTATS を実行します。 たとえば、ビュー myview 上で統計を収集するには、次のように入力します。

   ```
   RUNSTATS ON TABLE db2dba.myview
   ```

統計サンプリングを使用して、分散統計を含む、ビュー統計を収集するには、行レベルのサンプリングを使用する行の 10% で、次のように入力します。

```
RUNSTATS ON TABLE db2dba.myview WITH DISTRIBUTION TABLESAMPLE BERNOULLI (10)
```

注: DB2 のバージョン 9.1 の前は、統計ビューで RUNSTATS を実行すると手動更新のためのカタログ統計表が準備されるだけで、統計は収集されませんでした。DB2 Version 9.1では、統計ビューで RUNSTATS を実行すると、統計が収集されます。これは、ビューによって戻されるデータの量によっては、RUNSTATS が以前よりも長い時間を要する場合があることを意味します。

3. ビュー統計を更新すると、ビューの定義をオーバーラップする照会のプランが変化する場合があります。これらの照会が静的 SQL パッケージの一部である場合、新しい統計の結果として得られるプランを利用するため、これらのパッケージを再バインドする必要があります。

最適化に関連するビュー統計

最適化では、CARD や COLCARD など、統計ビューを定義する照会のデータ分布を特徴づける統計値のみが、オプティマイザーによって考慮されます。収集してオプティマイザーで活用できるのは、ビュー・レコードに関連付けられている次の統計だけです。

表統計 (SYSCAT.TABLES、SYSSTAT.TABLES)

- CARD - ビューの結果における行数です。

列統計 (SYSCAT.COLUMNS、SYSSTAT.COLUMNS)

- COLCARD - ビューの結果における列の特殊な値の数です。
- AVGCOLLEN - ビューの結果における平均の列長です。
- HIGH2KEY - ビューの結果における列の 2 番目に大きい値です。
- LOW2KEY - ビューの結果における列の 2 番目に小さい値です。
- NUMNULLS -ビューの結果の列の NULL の数です。
- SUB_COUNT - ビューの結果列における平均のサブエレメントの数です。
- SUB_DELIM_LENGTH - 各サブエレメントを分離する各区切り文字の平均長です。

列分散統計 (SYSCAT.COLDIST、SYSSTAT.COLDIST)

- DISTCOUNT - タイプが Q の場合、COLVALUE 統計値以下の特殊な値の数です。
- SEQNO - 表内の行を一意的に識別することを助けるためのシーケンス番号の頻度ランキングです。
- COLVALUE - 頻度または変位値統計を収集する対象となるデータ値です。
- VALCOUNT - ビュー列においてデータ値が発生する頻度、または変位値に対しては、データ値 (COLVALUE)以下の値の数。

データ分布を表現しない統計、例えば NPAGES、および FPAGES、は収集されますが、オプティマイザーでは無視されます。

シナリオ: 統計ビューを使用してカーディナリティー推定値を向上させる

データウェアハウスでは、ディメンション表データは静的ですが、ファクト表情報はしばしば大きく動的に変化します。これは、次元属性データがファクト表属性データと正または負の相関関係にある可能性を意味します。現在、オプティマイザーで使用可能な従来の基本表統計では、表間のリレーションシップを識別することは許可されていません。統計ビュー (および MQT) 上での列と表の分散統計を使用して、オプティマイザーに必要な情報を提供し、これらのタイプのカーディナリティー推定エラーを修正することができます。

各年の 7 月期に販売したゴルフ・クラブについて、年間の売り上げを計算する次の照会を考慮します。

```
SELECT  sum(f.sales_price), d2.year
FROM  product d1, period d2, daily_sales f
WHERE  d1.prodkey = f.prodkey
    AND  d2.perkey = f.perkey
    AND  d1.item_desc = 'golf club'
    AND  d2.month = 'JUL'
GROUP BY  d2.year
```

オプティマイザーが、PRODUCT および DAILY_SALES を含む半結合、または PERIOD や DAILY_SALES を含む半結合が最も選択的かどうかを判別できるのであれば、場合によっては、この照会ではスター型結合照会の実行プランが最も選択的です。効率的なスター型結合プランを生成するには、オプティマイザーが、索引 ANDing 操作の外部レグのために最も選択的な半結合を選択できる必要があります。

データウェアハウスに、在庫がなくなった製品のレコードが含まれることが多くあります。これが原因で、結合後の PRODUCT 列 の分布が結合前の分布と大きく異なって表示されることがあります。高精度の情報が欠落しているため、オプティマイザーは基本表の統計のみに基づいてローカル述部の選択度を判別します。そのため、 *item_desc = 'golf club'* 述部の選択度に関して、オプティマイザーが過度に楽観的になる可能性があります。

たとえば、ゴルフ・クラブ が従来製造された製品の 1% に相当し、かつ最近の売り上げの 20% を占める場合、オプティマイザーは、*item_desc = 'golf club'* の選択度を過剰に推定します。結合後の *item_desc* の分布記述統計がないからです。また、セールスが 12 カ月がすべて同程度だと思われる場合、*month = 'JUL'* 述部の選択度は約 8% 前後です。したがって、*item_desc = 'golf club'* 述部の選択度推定におけるエラーは、オプティマイザーが PRODUCT と DAILY_SALES の半結合を、スター型結合プランの索引 ANDing 操作の外部レグとして見かけ上、より選択的に実行する誤りの原因となります。

次の例は、このタイプの問題を解決するための、統計ビューのセットアップ方法を段階的に説明しています。

以下は STORE、 CUSTOMER、PRODUCT、PROMOTION、および PERIOD がディメンション表にあり、DAILY_SALES がファクト表にある典型的なデータウェアハウスのデータベースです。

表 66. STORE (63 行)

列	storekey	store_number	city	state	district	...
属性	整数	char(2)	char(20)	char(5)	char(14)	...
	非 NULL					
	主キー					

表 67. CUSTOMER (1,000,000 行)

列	custkey	名前	住所	年齢	性別	...
属性	整数	char(30)	char(40)	smallint	char(1)	...
	非 NULL					
	主キー					

表 68. PRODUCT (19,450 行)

列	prodkey	カテゴリー	item_desc	価格	コスト	...
属性	整数	整数	char(30)	decimal(11)	decimal(11)	...
	非 NULL					
	主キー					

表 69. PROMOTION (35 行)

列	promokey	promotype	promodesc	promovalue	...
属性	整数	整数	char(30)	decimal(5)	...
	非 NULL				
	主キー				

表 70. PERIOD (2922 行)

列	perkey	calendar_date	month	period	year	...
属性	整数	日付	char(3)	smallint	smallint	...
	非 NULL					
	主キー					

表 71. DAILY_SALES (754,069,426 行)

列	storekey	custkey	prodkey	promokey	perkey	sales_price	...
属性	整数	整数	整数	整数	整数	decimal(11)	...

再び訪問した消費者に対して以前と同じ製品の割引を提案した場合、その製品を再び買う可能性について、会社のマネージャーが調査すると想定します。加えて、国内に 18 のロケーションがある「01」ストアにのみ調査をするとします。 462 ページの表 72 は、プロモーションの割合でアノテーションを付けられた、選択可能なプロモーションの異なったカテゴリーを示しています。

表 72. PROMOTION (35 行)

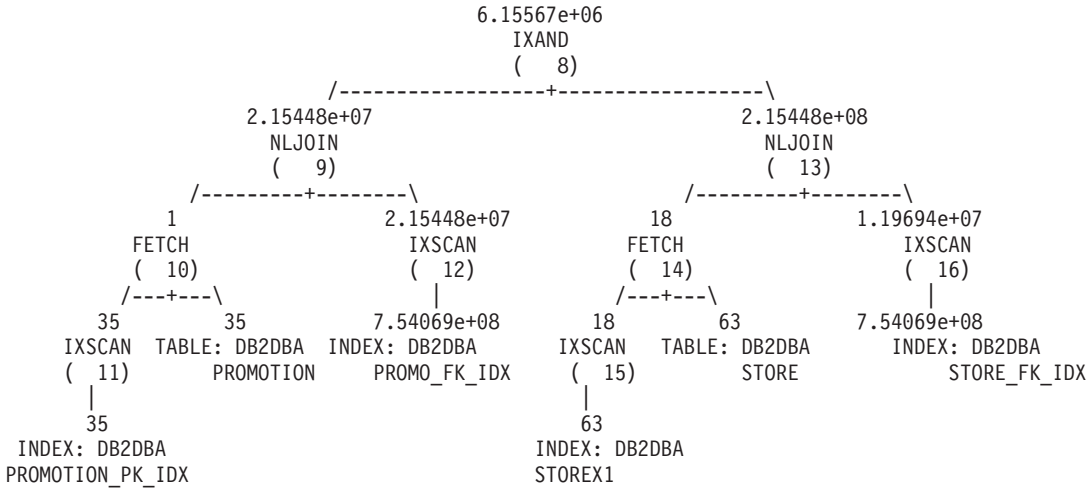
promotype	promodesc	COUNT(promotype)	プロモーションの割合
1	登録済みのお客様	1	2.86%
2	クーポン	15	42.86%
3	広告	5	14.29%
4	マネージャーのお勧め	3	8.57%
5	過剰在庫アイテム	4	11.43%
6	エンド・アイル・ディスプレイ	7	20.00%

この表は、提供された全 35 種類のプロモーションのわずか 2.86% だけが、登録済みのお客様用の割引からもたらされていることを示しています。(1÷35 ≈ 0.0286)

次の照会は実行され、12,889,514 カウントが返されます。

```
SELECT  count(*)
FROM  store d1, promotion d2, daily_sales f
WHERE  d1.storekey = f.storekey
    AND  d2.promokey = f.promokey
    AND  d1.store_number = '01'
    AND  d2.promotype = 1
```

この照会はオプティマイザーによって生成された次のプランにしたがって実行されます。この図の各ノードにおいて、上部の数はカーディナリティー推定値を、2 番目の行は演算子タイプを、括弧中の番号は演算子 ID を表しています。



番号 9 のネストされたループ結合において、オプティマイザーは、販売商品の約 2.86% が割引価格で同じ商品を買いにお客様が戻ってきた結果であると推定します。(2.15448e+07 ÷7.54069e+08 ≈ 0.0286) これは、PROMOTION 表と DAILY_SALES 表を結合する前後で同じパーセンテージであることに注意してください。 表 73 は、結合の前後のカーディナリティー推定値とその割合 (フィルタリング効果) を要約します。

表 73. カーディナリティーは、DAILY_SALES との結合の前後を推定します。

	結合前		結合後	
述部	**カウント**	**行修飾の割合**	**カウント**	**行修飾の割合**
store_number = '01'	18	28.57%	2.15448e+08	28.57%
promotype=1	1	2.86%	2.15448e+07	2.86%

promotype = 1 の確率は、*store_number = '01'* の確率より小さいので、オプティマイザーは PROMOTION と DAILY_SALES 間の半結合をスター型結合プランの索引 ANDing 操作の外部レグとして選択します。これは、タイプ 1 のプロモーションを使用すると、約 6,155,670 の商品販売の推定カウントとなります。2.09 という因数から誤ったカーディナリティー推定をしてしまいます (12,889,514 ÷ 6,155,670 ≈ 2.09)。

オプティマイザーが、2 つの述部に適合するレコードの実際数の半分しか推定できないのはなぜでしょうか。「01」ストアは全ストアの約 28.57% に相当します。他のストアが、「01」ストアより多くの売り上げがある場合はどうでしょうか (28.57% 未満の場合)。あるいは、もし「01」ストアが製品の大部分を売り上げた場合はどうでしょうか (28.57% より多い場合)。同様に、 462 ページの表 73 に示されるタイプ 1 のプロモーションを利用した商品販売の 2.86% は誤りの原因になる可能性があります。 DAILY_SALES の実際の割合は推定された数と大きく異なる場合があります。

オプティマイザーが見積もりを訂正し過大評価や過小評価を削減するために統計ビューを使用することができます。最初に前述の照会内の各半結合を表す 2 つの統計ビューを作成する必要があります。最初の統計ビューは、すべての日常販売活動におけるプロモーション・タイプの分布を提供します。 2 番目の統計ビューは、すべての日常販売活動におけるプロモーション・タイプの分布を表します。各統計ビューが特定のストア・ナンバーやプロモーション・タイプに関する分布情報を提供できるという点に注意してください。この例では、10% のサンプル率を利用して、各ビューのための DAILY_SALES のレコードを取得し、グローバルな一時表に保管しています。それから、必要な統計を収集するためにこれらの表を照会し、2 つの統計ビューを更新します。

1. STORE と DAILY_SALES の結合を表すビューを作成します。

```
CREATE  view sv_store_dailysales as
   (SELECT  s.*
    FROM  store s, daily_sales ds
    WHERE  s.storekey = ds.storekey)
```

2. PROMOTION と DAILY_SALES の結合を表すビューを作成します。

```
CREATE  view sv_promotion_dailysales as
   (SELECT  p.*
    FROM  promotion.p, daily_sales ds
    WHERE  p.promokey = ds.promokey)
```

3. 照会最適化を使用可能にしてビューを統計ビューにします。
   -
     ```
     ALTER VIEW  sv_store_dailysales  ENABLE QUERY OPTIMIZATION
     ```
   -
     ```
     ALTER VIEW  sv_promotion_dailysales  ENABLE QUERY OPTIMIZATION
     ```
4. RUNSTATS を実行して、ビュー上の統計を収集します。
   -
     ```
     RUNSTATS  on table db2dba.sv_store_dailysales  WITH DISTRIBUTION
     ```
   -
     ```
     RUNSTATS  on table db2dba.sv_promotion_dailysales WITH DISTRIBUTION
     ```
5. 再び照会を実行し、再最適化できるようにします。再最適化の際に、オプティマイザーは、SV_STORE_DAILYSALES と SV_PROMOTION_DAILYSALES をこ

の照会と突き合わせ、ビュー統計を使用してファクト表とディメンション表間の半結合のカーディナリティー推定を調整します。これにより、これらの統計なしで選択された半結合の元の順序を逆転させます。新規プランは次のとおりです。

```
                                1.04627e+07
                                  IXAND
                                  (   8)
                        /-----------------+-----------------\
                    6.99152e+07                             1.12845e+08
                      NLJOIN                                  NLJOIN
                      (   9)                                  (  13)
                  /---------+--------\                  /---------+--------\
                18                3.88418e+06           1                 1.12845e+08
               FETCH                IXSCAN             FETCH                IXSCAN
               (  10)               (  12)             (  14)               (  16)
              /---+---\               |               /---+---\               |
            18          63          7.54069e+08      35          35          7.54069e+08
          IXSCAN   TABLE:DB2DBA     INDEX: DB2DBA   IXSCAN    TABLE: DB2DBA    INDEX: DB2DBA DB2DBA
         (  11)      STORE           STORE_FK_IDX    (  15)       PROMOTION        PROMO_FK_IDX
            |                                          |
            63                                         35
        INDEX: DB2DBA                              INDEX: DB2DBA
        STOREX1                                    PROMOTION_PK_IDX
```

今回は STORE と DAILY_SALES 間の半結合が索引 ANDing プランの外部レグ上で実行されていることに注意してください。2 つの統計ビューは、本質的にオプティマイザーに対して store_number = '01' の述部は promotype = 1 の述部より多くの行をフィルタリングするように指示するからです。 表 74 は、半結合ごとの結合の前後のカーディナリティー推定値とその割合 (フィルタリング効果) を要約します。

今回は STORE と DAILY_SALES 間の半結合が索引 ANDing プランの外部レグ上で実行されていることに注意してください。2 つの統計ビューは、本質的にオプティマイザーに対して store_number = '01' の述部は promotype = 1 の述部より多くの行をフィルタリングするように指示するからです。 今回オプティマイザーは、約 10,462,700 の商品販売があると見積もります。この推定は 1.23 の要因から導き出され (12,889,514 ÷ 10,462,700 ≈ 1.23)、これは 462 ページの表 73 の推定において大きな改善です。 表 74 は、各述部のため結合の前後に、カーディナリティー推定値とフィルタリング効果を要約します。

表 74. カーディナリティーは、DAILY_SALES との結合の前後を推定します。

	結合前		**結合後** *統計ビューなしの場合*		**結合後** *統計ビューがある場合*	
述部	**カウント**	**行修飾の割合**	**カウント**	**行修飾の割合**	**カウント**	**行修飾の割合**
store_number = '01'	18	28.57%	2.15448e+08	28.57%	6.99152e+07	9.27%
promotype=1	1	2.86%	2.15448e+07	2.86%	1.12845e+08	14.96%

今回オプティマイザーは、約 10,462,700 の商品販売があると見積もります。この推定は 1.23 の要因から導き出され、これは統計ビューがない推定において大きな改善です。

第 6 部 付録

付録 A. DB2 技術情報の概説

DB2 技術情報は、以下のツールと方法を介して利用できます。

- DB2 インフォメーション・センター
  - トピック (タスク、概念、およびリファレンス・トピック)
  - DB2 ツールのヘルプ
  - サンプル・プログラム
  - チュートリアル
- DB2 資料
  - PDF ファイル (ダウンロード可能)
  - PDF ファイル (DB2 PDF DVD に含まれる)
  - 印刷資料
- コマンド行ヘルプ
  - コマンド・ヘルプ
  - メッセージ・ヘルプ

注: DB2 インフォメーション・センターのトピックは、PDF やハードコピー資料よりも頻繁に更新されます。最新の情報を入手するには、資料の更新が発行されたときにそれをインストールするか、ibm.com® にある DB2 インフォメーション・センターを参照してください。

技術資料、ホワイト・ペーパー、IBM Redbooks® 資料などのその他の DB2 技術情報には、オンライン (ibm.com) でアクセスできます。 DB2 Information Management ソフトウェア・ライブラリー・サイト (http://www.ibm.com/software/data/sw-library/) にアクセスしてください。

資料についてのフィードバック

DB2 の資料についてのお客様からの貴重なご意見をお待ちしています。DB2 の資料を改善するための提案については、db2docs@ca.ibm.com まで E メールを送信してください。DB2 の資料チームは、お客様からのフィードバックすべてに目を通しますが、直接お客様に返答することはありません。お客様が関心をお持ちの内容について、可能な限り具体的な例を提供してください。特定のトピックまたはヘルプ・ファイルについてのフィードバックを提供する場合は、そのトピック・タイトルおよび URL を含めてください。

DB2 お客様サポートに連絡する場合には、この E メール・アドレスを使用しないでください。資料を参照しても、DB2 の技術的な問題が解決しない場合は、お近くの IBM サービス・センターにお問い合わせください。

DB2 テクニカル・ライブラリー (ハードコピーまたは PDF 形式)

以下の表は、DB2 ライブラリーについて説明しています。DB2 ライブラリーに関する詳細な説明については、www.ibm.com/shop/publications/order にある IBM Publications Center にアクセスしてください。英語の DB2 バージョン 9.5 のマニュアル (PDF 形式) とその翻訳版は、 www.ibm.com/support/docview.wss?rs=71&uid=swg2700947 からダウンロードできます。

この表には印刷資料が入手可能かどうかが示されていますが、国または地域によっては入手できない場合があります。

表 75. DB2 の技術情報

資料名	資料番号	印刷資料が入手可能かどうか
管理 API リファレンス	SC88-4431-00	入手可能
管理ルーチンおよびビュー	SC88-4435-00	入手不可
コール・レベル・インターフェース ガイドおよびリファレンス 第 1 巻	SC88-4433-00	入手可能
コール・レベル・インターフェース ガイドおよびリファレンス 第 2 巻	SC88-4434-00	入手可能
コマンド・リファレンス	SC88-4432-00	入手可能
データ移動ユーティリティー ガイドおよびリファレンス	SC88-4421-00	入手可能
データ・リカバリーと高可用性 ガイドおよびリファレンス	SC88-4423-00	入手可能
データ・サーバー、データベース、およびデータベース・オブジェクトのガイド	SC88-4259-00	入手可能
データベース・セキュリティー・ガイド	SC88-4418-00	入手可能
ADO.NET および OLE DB アプリケーションの開発	SC88-4425-00	入手可能
組み込み SQL アプリケーションの開発	SC88-4426-00	入手可能
Java アプリケーションの開発	SC88-4427-00	入手可能
Perl および PHP アプリケーションの開発	SC88-4428-00	入手不可
SQL および 外部ルーチンの開発	SC88-4429-00	入手可能
データベース・アプリケーション開発の基礎	GC88-4430-00	入手可能
DB2 インストールおよび管理概説 (Linux および Windows 版)	GC88-4439-00	入手可能
国際化対応ガイド	SC88-4420-00	入手可能

表 75. DB2 の技術情報 (続き)

資料名	資料番号	印刷資料が入手可能かどうか
メッセージ・リファレンス 第 1 巻	GI88-4109-00	入手不可
メッセージ・リファレンス 第 2 巻	GI88-4110-00	入手不可
マイグレーション・ガイド	GC88-4438-00	入手可能
Net Search Extender 管理およびユーザーズ・ガイド **注:** この資料の内容は、DB2 インフォメーション・センターには含まれていません。	SC88-4630-00	入手可能
パーティションおよびクラスタリングのガイド	SC88-4419-00	入手可能
Query Patroller 管理およびユーザーズ・ガイド	SC88-4611-00	入手可能
IBM データ・サーバー・クライアント機能 概説およびインストール	GC88-4441-00	入手不可
DB2 サーバー機能 概説およびインストール	GC88-4440-00	入手可能
Spatial Extender and Geodetic Data Management Feature ユーザーズ・ガイドおよびリファレンス	SC88-4629-00	入手可能
SQL リファレンス 第 1 巻	SC88-4436-00	入手可能
SQL リファレンス 第 2 巻	SC88-4437-00	入手可能
システム・モニター ガイドおよびリファレンス	SC88-4422-00	入手可能
テキスト検索ガイド	SC88-4424-00	入手可能
問題判別ガイド	GI88-4108-00	入手不可
データベース・パフォーマンスのチューニング	SC88-4417-00	入手可能
Visual Explain チュートリアル	SC88-4449-00	入手不可
新機能	SC88-4445-00	入手可能
ワークロード・マネージャー ガイドおよびリファレンス	SC88-4446-00	入手可能
pureXML ガイド	SC88-4447-00	入手可能
XQuery リファレンス	SC88-4448-00	入手不可

表 76. DB2 Connect 固有の技術情報

資料名	資料番号	印刷資料が入手可能かどうか
DB2 Connect Personal Edition 概説およびインストール	GC88-4443-00	入手可能

表 76. DB2 Connect 固有の技術情報 (続き)

資料名	資料番号	印刷資料が入手可能かどうか
DB2 Connect サーバー機能 概説およびインストール	GC88-4444-00	入手可能
DB2 Connect ユーザーズ・ガイド	SC88-4442-00	入手可能

表 77. Information Integration の技術情報

資料名	資料番号	印刷資料が入手可能かどうか
Information Integration: フェデレーテッド・システム 管理ガイド	SC88-4166-01	入手可能
Information Integration: レプリケーションおよびイベント・パブリッシングのための ASNCLP プログラム・リファレンス	SC88-4167-02	入手可能
Information Integration: フェデレーテッド・データ・ソース構成ガイド	SC88-4185-01	入手不可
Information Integration: SQL レプリケーション ガイドおよびリファレンス	SC88-4168-01	入手可能
Information Integration: レプリケーションとイベント・パブリッシング 概説	GC88-4187-01	入手可能

DB2 の印刷資料の注文方法

DB2 の印刷資料が必要な場合、オンラインで購入することができますが、すべての国および地域で購入できるわけではありません。 DB2 の印刷資料については、IBM 営業担当員にお問い合わせください。 *DB2 PDF* ドキュメンテーション DVD の一部のソフトコピー・ブックは、印刷資料では入手できないことに留意してください。例えば、「*DB2* メッセージ・リファレンス」はどちらの巻も印刷資料としては入手できません。

DB2 PDF ドキュメンテーション DVD で利用できる DB2 の印刷資料の大半は、IBM に有償で注文することができます。国または地域によっては、資料を IBM Publications Center からオンラインで注文することもできます。お客様の国または地域でオンライン注文が利用できない場合、DB2 の印刷資料については、IBM 営業担当員にお問い合わせください。 DB2 PDF ドキュメンテーション DVD に収録されている資料の中には、印刷資料として提供されていないものもあります。

注: 最新で完全な DB2 資料は、DB2 インフォメーション・センター (http://publib.boulder.ibm.com/infocenter/db2luw/v9r5) で参照することができます。

DB2 の印刷資料は以下の方法で注文することができます。

- 日本 IBM 発行のマニュアルはインターネット経由でご購入いただけます。詳しくは http://www.ibm.com/shop/publications/order の「ご注文について」をご覧ください。資料の注文情報にアクセスするには、お客様の国、地域、または言語を選択してください。その後、各ロケーションにおける注文についての指示に従ってください。
- DB2 の印刷資料を IBM 営業担当員に注文するには、以下のようにします。
  1. 以下の Web サイトのいずれかから、営業担当員の連絡先情報を見つけてください。
     - IBM Directory of world wide contacts (www.ibm.com/planetwide)
     - IBM Publications Web サイト (http://www.ibm.com/shop/publications/order) 国、地域、または言語を選択し、お客様の所在地に該当する Publications ホーム・ページにアクセスしてください。このページから、「このサイトについて」のリンクにアクセスしてください。
  2. 電話をご利用の場合は、DB2 資料の注文であることをご指定ください。
  3. 担当者に、注文する資料のタイトルと資料番号をお伝えください。タイトルと資料番号は、468 ページの『DB2 テクニカル・ライブラリー (ハードコピーまたは PDF 形式)』でご確認いただけます。

コマンド行プロセッサーから SQL 状態ヘルプを表示する

DB2 は、SQL ステートメントの結果の原因になったと考えられる条件の SQLSTATE 値を戻します。 SQLSTATE ヘルプは、SQL 状態および SQL 状態クラス・コードの意味を説明します。

SQL 状態ヘルプを呼び出すには、コマンド行プロセッサーを開いて以下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。

例えば、`? 08003` を指定すると SQL 状態 08003 のヘルプが表示され、 `? 08` を指定するとクラス・コード 08 のヘルプが表示されます。

異なるバージョンの DB2 インフォメーション・センターへのアクセス

DB2 バージョン 9.5 のトピックを扱っている DB2 インフォメーション・センターの URL は、http://publib.boulder.ibm.com/infocenter/db2luw/v9r5/です。

DB2 バージョン 9 のトピックを扱っている DB2 インフォメーション・センターの URL は http://publib.boulder.ibm.com/infocenter/db2luw/v9/です。

DB2 バージョン 8 のトピックについては、バージョン 8 のインフォメーション・センターの URL http://publib.boulder.ibm.com/infocenter/db2luw/v8/にアクセスしてください。

DB2 インフォメーション・センターにおける特定の言語でのトピックの表示

DB2 インフォメーション・センターでは、ブラウザーの設定で指定した言語でのトピックの表示が試みられます。トピックがその指定言語に翻訳されていない場合は、 DB2 インフォメーション・センターでは英語でトピックが表示されます。

- Internet Explorer Web ブラウザーで、指定どおりの言語でトピックを表示するには、以下のようにします。
  1. Internet Explorer の「**ツール**」 -> 「**インターネット オプション**」 -> 「**言語...**」ボタンをクリックします。「言語の優先順位」ウィンドウがオープンします。
  2. 該当する言語が、言語リストの先頭の項目に指定されていることを確認します。
     - リストに新しい言語を追加するには、「**追加...**」ボタンをクリックします。

       注: 言語を追加しても、特定の言語でトピックを表示するのに必要なフォントがコンピューターに備えられているとはかぎりません。
     - リストの先頭に新しい言語を移動するには、その言語を選択してから、その言語が言語リストに先頭に行くまで「**上に移動**」ボタンをクリックします。
  3. ブラウザー・キャッシュを消去してから、ページをリフレッシュし、使用する言語で DB2 インフォメーション・センターを表示します。
- Firefox または Mozilla Web ブラウザーの場合に、使いたい言語でトピックを表示するには、以下のようにします。
  1. 「**ツール**」 -> 「**オプション**」 -> 「**詳細**」 ダイアログの「**言語**」セクションにあるボタンを選択します。「設定」ウィンドウに「言語」パネルが表示されます。
  2. 該当する言語が、言語リストの先頭の項目に指定されていることを確認します。
     - リストに新しい言語を追加するには、「**追加...**」ボタンをクリックしてから、「言語を追加」ウィンドウで言語を選択します。
     - リストの先頭に新しい言語を移動するには、その言語を選択してから、その言語が言語リストに先頭に行くまで「**上に移動**」ボタンをクリックします。
  3. ブラウザー・キャッシュを消去してから、ページをリフレッシュし、使用する言語で DB2 インフォメーション・センターを表示します。

ブラウザーとオペレーティング・システムの組み合わせによっては、オペレーティング・システムの地域の設定も希望のロケールと言語に変更しなければならない場合があります。

コンピューターまたはイントラネット・サーバーにインストールされた DB2 インフォメーション・センターの更新

DB2 インフォメーション・センターをローカルにインストールしている場合は、IBM から提供される更新をダウンロードおよびインストールすることができます。

ローカルにインストールされた DB2 インフォメーション・センターを更新するには、以下のことを行う必要があります。

1. コンピューター上の DB2 インフォメーション・センターを停止し、インフォメーション・センターをスタンドアロン・モードで再始動します。インフォメーション・センターをスタンドアロン・モードで実行すると、ネットワーク上の他のユーザーがそのインフォメーション・センターにアクセスできなくなります。これで、更新をダウンロードして適用できるようになります。
2. 「更新」機能を使用することにより、どんな更新が利用できるかを確認します。インストールする更新がある場合は、「更新」機能を使用してそれをダウンロードおよびインストールできます。

   注: ご使用の環境において、インターネットに接続されていないマシンに DB2 インフォメーション・センターの更新をインストールする必要がある場合は、インターネットに接続されていて DB2 インフォメーション・センターがインストールされているマシンを使用して、更新サイトをローカル・ファイル・システムにミラーリングする必要があります。ネットワーク上の多数のユーザーが資料の更新をインストールする場合にも、更新サイトをローカルにミラーリングして、更新サイト用のプロキシーを作成することにより、個々のユーザーが更新を実行するのに要する時間を短縮できます。
   更新パッケージが入手可能な場合、「更新」機能を使用してパッケージをダウンロードします。ただし、「更新」機能は、スタンドアロン・モードでのみ使用できます。
3. スタンドアロンのインフォメーション・センターを停止し、コンピューター上の DB2 インフォメーション・センターを再開します。

注: Windows Vista の場合、下記のコマンドは管理者として実行する必要があります。完全な管理者特権でコマンド・プロンプトまたはグラフィカル・ツールを起動するには、ショートカットを右クリックしてから、 **「管理者として実行」**を選択します。

コンピューターまたはイントラネット・サーバーにインストール済みの DB2 インフォメーション・センターを更新するには、以下のようにします。

1. DB2 インフォメーション・センターを停止します。
   - Windows では、**「スタート」→「コントロール パネル」→「管理ツール」→「サービス」**をクリックします。次に、**「DB2 インフォメーション・センター」**サービスを右クリックして**「停止」**を選択します。
   - Linux では、以下のコマンドを入力します。

     ```
     /etc/init.d/db2icdv95 stop
     ```
2. インフォメーション・センターをスタンドアロン・モードで開始します。
   - Windows の場合:
     a. コマンド・ウィンドウを開きます。

b. インフォメーション・センターがインストールされているパスにナビゲートします。デフォルトでは、DB2 インフォメーション・センターは <Program Files>¥IBM¥DB2 Information Center¥Version 9.5 ディレクトリーにインストールされています (<Program Files> は「Program Files」ディレクトリーのロケーション)。

c. インストール・ディレクトリーの doc¥bin ディレクトリーにナビゲートします。

d. 次のように help_start.bat ファイルを実行します。

```
help_start.bat
```

- Linux の場合:

a. インフォメーション・センターがインストールされているパスにナビゲートします。デフォルトでは、DB2 インフォメーション・センターは /opt/ibm/db2ic/V9.5 ディレクトリーにインストールされています。

b. インストール・ディレクトリーの doc/bin ディレクトリーにナビゲートします。

c. 次のように help_start スクリプトを実行します。

```
help_start
```

システムのデフォルト Web ブラウザーが起動し、スタンドアロンのインフォメーション・センターが表示されます。

3. 「更新」ボタン () をクリックします。インフォメーション・センターの右側のパネルで、 「更新の検索 (Find Updates)」をクリックします。既存の文書に対する更新のリストが表示されます。
4. ダウンロード・プロセスを開始するには、ダウンロードする更新をチェックして選択し、「更新のインストール (Install Updates)」をクリックします。
5. ダウンロードおよびインストール・プロセスが完了したら、「完了」をクリックします。
6. スタンドアロンのインフォメーション・センターを停止します。
   - Windows の場合は、インストール・ディレクトリーの doc¥bin ディレクトリーにナビゲートしてから、次のように help_end.bat ファイルを実行します。

     ```
     help_end.bat
     ```

     注: help_end バッチ・ファイルには、help_start バッチ・ファイルを使用して開始したプロセスを安全に終了するのに必要なコマンドが含まれています。Ctrl-C または他の方法を使用して、help_start.bat を終了しないでください。
   - Linux の場合は、インストール・ディレクトリーの doc/bin ディレクトリーにナビゲートしてから、次のように help_end スクリプトを実行します。

     ```
     help_end
     ```

     注: help_end スクリプトには、help_start スクリプトを使用して開始したプロセスを安全に終了するのに必要なコマンドが含まれています。他の方法を使用して、help_start スクリプトを終了しないでください。
7. DB2 インフォメーション・センターを再開します。

- Windows では、「**スタート**」→「**コントロール パネル**」→「**管理ツール**」→「**サービス**」をクリックします。次に、「**DB2 インフォメーション・センター**」サービスを右クリックして「**開始**」を選択します。
- Linux では、以下のコマンドを入力します。

  ```
  /etc/init.d/db2icdv95 start
  ```

更新された DB2 インフォメーション・センターに、更新された新しいトピックが表示されます。

DB2 チュートリアル

DB2 チュートリアルは、DB2 製品のさまざまな機能について学習するのを支援します。この演習をとおして段階的に学習することができます。

はじめに

インフォメーション・センター (http://publib.boulder.ibm.com/infocenter/db2help/) から、このチュートリアルの XHTML 版を表示できます。

演習の中で、サンプル・データまたはサンプル・コードを使用する場合があります。個々のタスクの前提条件については、チュートリアルを参照してください。

DB2 チュートリアル

チュートリアルを表示するには、タイトルをクリックします。

「*pureXML ガイド* 」の『**pureXML™**』
: XML データを保管し、ネイティブ XML データ・ストアに対して基本的な操作を実行できるように、DB2 データベースをセットアップします。

「*Visual Explain チュートリアル*」の『**Visual Explain**』
: Visual Explain を使用して、パフォーマンスを向上させるために SQL ステートメントを分析し、最適化し、調整します。

DB2 トラブルシューティング情報

DB2 製品を使用する際に役立つ、トラブルシューティングおよび問題判別に関する広範囲な情報を利用できます。

DB2 ドキュメンテーション
: トラブルシューティング情報は、DB2 問題判別ガイド、または DB2 インフォメーション・センターの「サポートおよびトラブルシューティング」セクションにあります。ここには、DB2 診断ツールおよびユーティリティーを使用して、問題を切り分けて識別する方法、最も頻繁に起こる幾つかの問題に対するソリューションについての情報、および DB2 製品を使用する際に発生する可能性のある問題の解決方法についての他のアドバイスがあります。

DB2 Technical Support の Web サイト
: 現在問題が発生していて、考えられる原因とソリューションを検索したい場合は、DB2 Technical Support の Web サイトを参照してください。
Technical Support サイトには、最新の DB2 資料、TechNotes、プログラム

診断依頼書 (APAR またはバグ修正)、フィックスパック、およびその他のリソースへのリンクが用意されています。この知識ベースを活用して、問題に対する有効なソリューションを探し出すことができます。

DB2 Technical Support の Web サイト (http://www.ibm.com/software/data/db2/udb/support.html) にアクセスしてください。

ご利用条件

これらの資料は、以下の条件に同意していただける場合に限りご使用いただけます。

個人使用: これらの資料は、すべての著作権表示その他の所有権表示をしていただくことを条件に、非商業的な個人による使用目的に限り複製することができます。ただし、IBM の明示的な承諾をえずに、これらの資料またはその一部について、二次的著作物を作成したり、配布 (頒布、送信を含む) または表示 (上映を含む) することはできません。

商業的使用: これらの資料は、すべての著作権表示その他の所有権表示をしていただくことを条件に、お客様の企業内に限り、複製、配布、および表示することができます。 ただし、IBM の明示的な承諾をえずにこれらの資料の二次的著作物を作成したり、お客様の企業外で資料またはその一部を複製、配布、または表示することはできません。

ここで明示的に許可されているもの以外に、資料や資料内に含まれる情報、データ、ソフトウェア、またはその他の知的所有権に対するいかなる許可、ライセンス、または権利を明示的にも黙示的にも付与するものではありません。

資料の使用が IBM の利益を損なうと判断された場合や、上記の条件が適切に守られていないと判断された場合、IBM はいつでも自らの判断により、ここで与えた許可を撤回できるものとさせていただきます。

お客様がこの情報をダウンロード、輸出、または再輸出する際には、米国のすべての輸出入関連法規を含む、すべての関連法規を遵守するものとします。

IBM は、これらの資料の内容についていかなる保証もしません。これらの資料は、特定物として現存するままの状態で提供され、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任なしで提供されます。

付録 B. 特記事項

本書は米国 IBM が提供する製品およびサービスについて作成したものです。

本書に記載の製品、サービス、または機能が日本においては提供されていない場合があります。日本で利用可能な製品、サービス、および機能については、日本 IBM の営業担当員にお尋ねください。本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービスのみが使用可能であることを意味するものではありません。これらに代えて、IBM の知的所有権を侵害することのない、機能的に同等の製品、プログラム、またはサービスを使用することができます。ただし、IBM 以外の製品とプログラムの操作またはサービスの評価および検証は、お客様の責任で行っていただきます。

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む) を保有している場合があります。本書の提供は、お客様にこれらの特許権について実施権を許諾することを意味するものではありません。実施権についてのお問い合わせは、書面にて下記宛先にお送りください。

〒106-8711
東京都港区六本木 3-2-12
日本アイ・ビー・エム株式会社
法務・知的財産
知的財産権ライセンス渉外

以下の保証は、国または地域の法律に沿わない場合は、適用されません。 IBM およびその直接または間接の子会社は、本書を特定物として現存するままの状態で提供し、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。国または地域によっては、法律の強行規定により、保証責任の制限が禁じられる場合、強行規定の制限を受けるものとします。

この情報には、技術的に不適切な記述や誤植を含む場合があります。本書は定期的に見直され、必要な変更は本書の次版に組み込まれます。 IBM は予告なしに、随時、この文書に記載されている製品またはプログラムに対して、改良または変更を行うことがあります。

本書は、IBM 以外の Web サイトおよびリソースへのリンクまたは参照を含む場合があります。 IBM は、本書より参照もしくはアクセスできる、または本書からリンクされた IBM 以外の Web サイトもしくは第三者のリソースに対して一切の責任を負いません。 IBM 以外の Web サイトにリンクが張られていることにより IBM が当該 Web サイトを推奨するものではなく、またその内容、使用もしくはサイトの所有者について IBM が責任を負うことを意味するものではありません。 また、IBM は、お客様が IBM Web サイトから第三者の存在を知ることになった場合にも (もしくは、IBM Web サイトから第三者へのリンクを使用した場合にも) 、お客様と第三者との間のいかなる取引に対しても一切責任を負いません。 従って、お客様は、IBM が上記の外部サイトまたはリソースの利用について責任を負うものではなく、また、外部サイトまたはリソースからアクセス可能なコンテンツ、サービス、

製品、またはその他の資料一切に対して IBM が責任を負うものではないことを承諾し、同意するものとします。 第三者により提供されるソフトウェアには、そのソフトウェアと共に提供される固有の使用条件が適用されます。

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 Limited
Office of the Lab Director
8200 Warden Avenue
Markham, Ontario
L6G 1C7
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・プログラムのいかなる部分、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

DB2 バージョン 9.5 ドキュメンテーション・ライブラリーの資料に記載されている会社名、製品名、またはサービス名は、 IBM Corporation の商標である可能性があります。 IBM Corporation の商標については、http://www.ibm.com/legal/copytrade.shtml を参照してください。

以下は、それぞれ各社の商標または登録商標です。

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。

Intel®、 Intel ロゴ、Intel Inside® ロゴ、Intel Centrino®、Intel Centrino ロゴ、Celeron®、Intel Xeon®、 Intel SpeedStep®、 Itanium® および Pentium® は、Intel 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です。

UNIX は、The Open Group の米国およびその他の国における登録商標です。

Linux は、Linus Torvalds の米国およびその他の国における商標です。

Adobe®、Adobe ロゴ、PostScript®、PostScript ロゴは、Adobe Systems Incorporated の米国およびその他の国における登録商標または商標です。

他の会社名、製品名およびサービス名等はそれぞれ各社の商標です。

索引

日本語, 数字, 英字, 特殊文字の順に配列されています。なお, 濁音と半濁音は清音と同等に扱われています。

[ア行]

[カ行]

[サ行]

[タ行]

[ナ行]

[ハ行]

［マ行］

［ヤ行］

［ラ行］

[ワ行]

A

B

C

D

E

F

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z

Printed in Japan

SC88-4417-00

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

Spine information:

DB2 Version 9.5 for Linux, UNIX, and Windows

データベース・パフォーマンスのチューニング