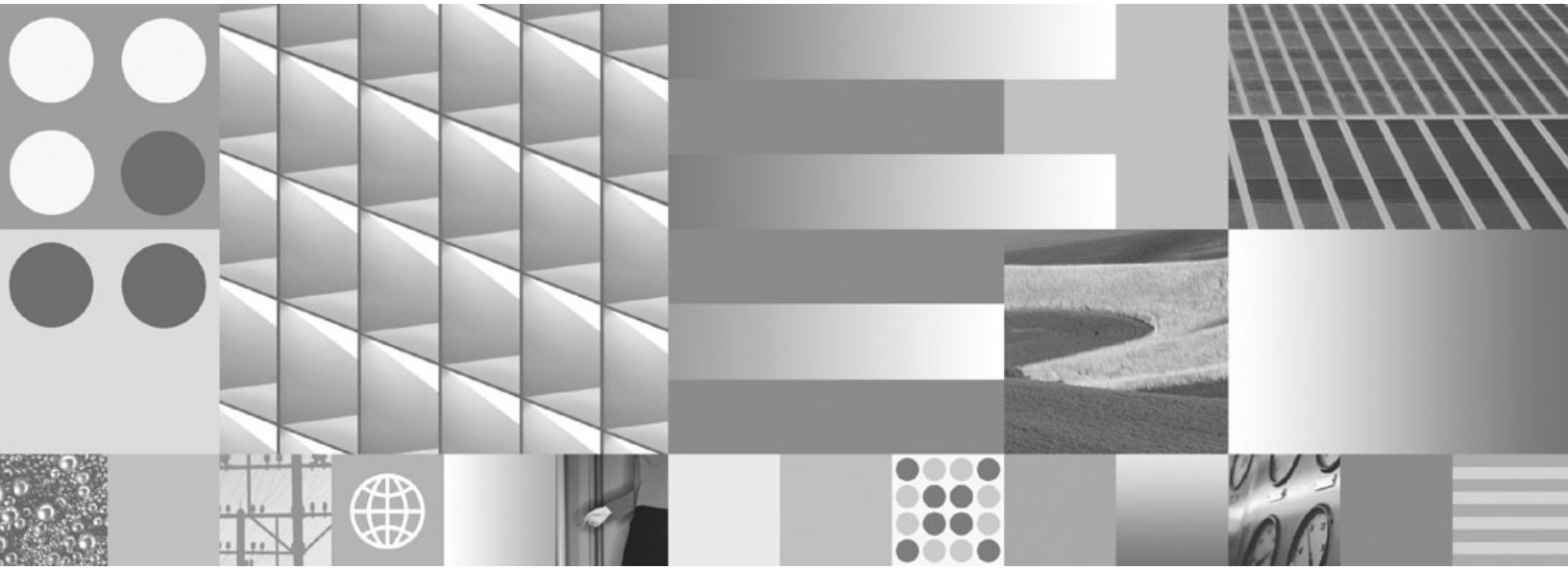
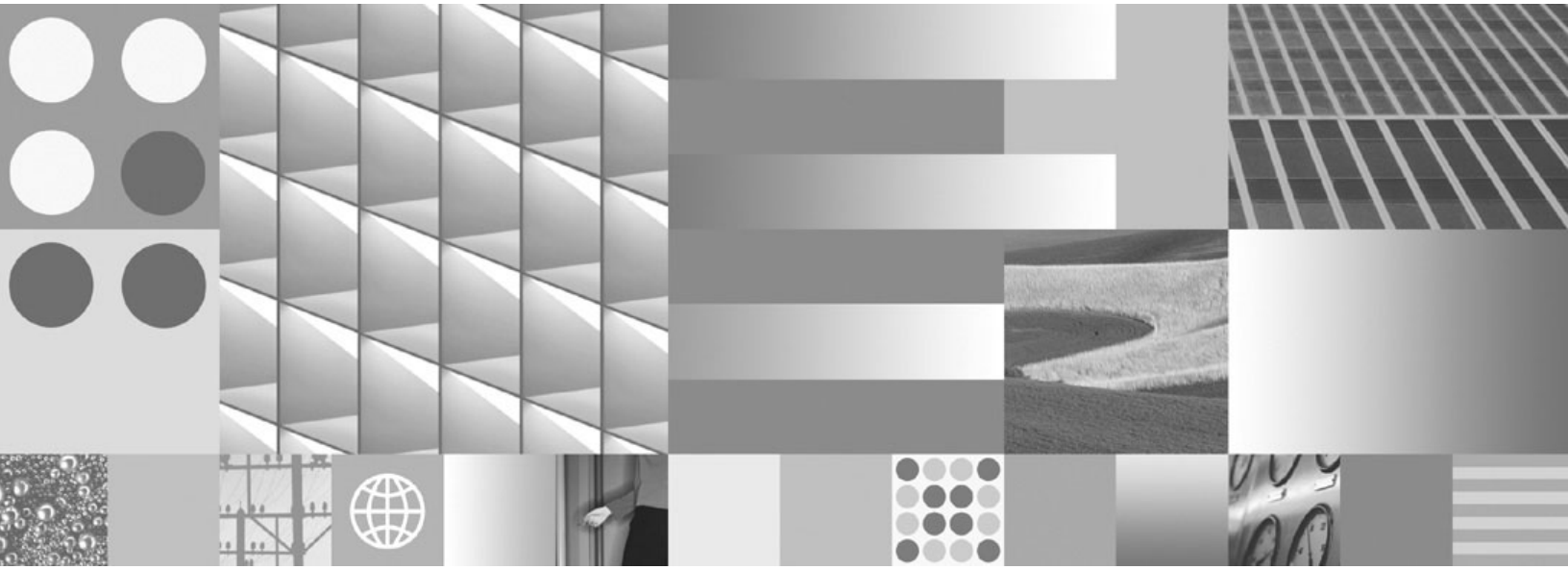
DB2 Version 9.5
for Linux, UNIX, and Windows



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

SC88-4259-01
(英文原典：SC23-5849-01)

DB2 Version 9.5
for Linux, UNIX, and Windows



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

SC88-4259-01
(英文原典：SC23-5849-01)

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、 697 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： SC23-5849-01
DB2 Version 9.5 for Linux, UNIX, and Windows
Data Servers, Databases, and Database Objects Guide

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 2008.3

目次

本書について

「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」には、DB2® リレーショナル・データベース管理システム (RDBMS) 製品の使用と管理に必要な情報が載せられています。本書には、データベースの計画および設計、データベース・オブジェクトのインプリメンテーションおよび管理に関する情報が含まれます。また、データベースの構成およびチューニングに関する参照情報も載せられています。

本書の対象読者

本書は主に、ローカルまたはリモート・クライアントによってアクセスされるデータベースを設計、インプリメント、および保守する必要のある、データベースおよびシステム管理者を対象としています。また、DB2 リレーショナル・データベース管理システムの管理および操作を理解する必要のあるプログラマーや他のユーザーにもご利用いただけます。

本書の構成

本書は、以下の 4 部で構成されています。

第 1 部 - データ・サーバー
このセクションでは、DB2 データ・サーバーを簡潔に説明します。データ・サーバーの容量の管理、AIX® 上の 64 ビット環境におけるラージ・ページのサポートについても説明します。さらに、単一のコンピューター上で複数の DB2 コピーを実行することに関する情報、データベース・システムの管理に役立つ自動フィーチャーについての情報、インスタンスの設計、作成、および操作に関する情報、および Lightweight Directory Access Protocol (LDAP) サーバーの構成に関するオプション情報も記載しています。

第 2 部 - データベース
このセクションでは、データベース、バッファー・プール、表スペース、およびスキーマの設計、作成、および保守について説明します。データベース・パーティションの詳細情報は、新しい「*パーティショニングおよびクラスタリング・ガイド*」に記載されています。

第 3 部 - データベース・オブジェクト
このセクションでは、次のデータベース・オブジェクトの設計、作成、および保守について説明します。表、制約、索引、トリガー、シーケンス、およびビュー。

第 4 章 - リファレンス
このセクションには、環境変数、レジストリー変数、および構成パラメーターを使用したデータベース・システムの構成およびチューニングに関する参照情報が載せられています。さまざまな命名規則および SQL と XML の制限もリストされています。

第 1 部 データ・サーバー

第 1 章 DB2 データ・サーバー

データ・サーバーは、構造化された情報を安全で効率的に管理できるようにするためのソフトウェア・サービスを提供します。 DB2 はハイブリッド・リレーショナル XML データ・サーバーです。

データ・サーバーとは DB2 データベース・エンジンがインストールされているマシンを指します。 DB2 エンジンはすべての機能を装備した堅固なデータベース管理システムであり、ここには実際のデータベース使用に基づいて最適化された SQL サポートと、データの管理に役立つツールが組み込まれています。

IBM では、すべての各種データ・サーバーにアクセス可能なデータ・サーバー・クライアントが組み込まれたいくつかのデータ・サーバー製品を提供しています。 DB2 データ・サーバー製品、使用可能なフィーチャー、および詳細説明と仕様の完全なリストについては、http://www-306.ibm.com/software/data/db2/9/ を参照してください。

データ・サーバー容量の管理

データ・サーバーの容量が、現在または将来の必要を満たさない場合、ディスク・スペースを追加して追加のコンテナーを作成するか、メモリーを追加することによってその容量を拡張することができます。これらの単純なストラテジーで、必要な容量を追加できない場合は、プロセッサーまたは物理パーティションを追加することを考慮してください。環境を変更してシステムを拡大または縮小するときは、そのような変更が、データのロード、またはデータベースのバックアップおよびリストアなどのデータベース手順に与える影響をよく知っている必要があります。

プロセッサーの追加

1 台のプロセッサーを単一パーティション・データベース構成で使用しており、しかもそれを最大限まで使用してしまっている場合、プロセッサーを追加するか、データベース・パーティションを追加したほうがよいでしょう。プロセッサーを追加することの利点は、処理力が増すことです。 SMP システムでは、プロセッサーはメモリーおよびストレージ・システム・リソースを共用します。すべてのプロセッサーが 1 つのシステムに収まっているため、システム間の通信回線、およびシステム間のタスクの調整などについての、オーバーヘッドとなる考慮事項が加わることがありません。ロード、バックアップ、およびリストアなどのユーティリティーは、追加のプロセッサーを利用することができます。

注: Solaris オペレーティング・システムのように、オペレーティング・システムによっては、プロセッサーを動的にオンラインまたはオフラインに調整することができるものがあります。

プロセッサーを追加する場合、使用されるプロセッサーの数を左右する、いくつかのデータベース構成パラメーターを検討し、変更してください。次のデータベース構成パラメーターは、使用するプロセッサー数を判別するもので、更新の必要の可能性があります。

- デフォルトのパーティション内並行度 (dft_degree)
- 最大並列処理の多重度 (max_querydegree)
- パーティション内並列処理機能の使用可能化 (intra_parallel)

また、アプリケーションが並列処理を実行する方法を決定するパラメーターを評価することも必要です。

通信に TCP/IP が使用されている環境では、 DB2TCPCONNMGRS レジストリー変数の値を考慮する必要があります。

物理パーティションの追加

データベース・マネージャーが現在パーティション・データベース環境にある場合、個別の単一プロセッサーまたはマルチプロセッサーの物理パーティションを追加することによって、データ・ストレージ・スペースを広げ、処理能力を高めることができます。各データベース・パーティション上のメモリーおよびストレージ・システム・リソースは、他のデータベース・パーティションと共用されません。データベース・パーティションの追加の結果、通信およびタスク調整の問題が発生することがありますが、この選択には、複数のシステム間でデータおよびユーザー・アクセスのバランスを取ることができる、という利点があります。データベース・マネージャーはこの環境をサポートします。

データベース・パーティションの追加は、データベース・マネージャー・システムの実行中でも停止中でも行えます。ただし、システムの実行中にデータベース・パーティションを追加すると、データベースがその新しいデータベース・パーティションにマイグレーションする前にシステムを一度停止し、再始動することが必要です。

新しいデータベース・パーティションを追加するときは、その処理が完了し、新しいサーバーが正常にシステムに組み込まれるまでは、新規データベース・パーティションを活用するデータベースのドロップまたは作成を行うことはできません。

64 ビット環境におけるラージ・ページのサポートの使用可能化 (AIX)

IBM® eServer™ pSeries® システムの POWER4™ プロセッサーでは、従来のページ・サイズ 4 KB に加えて、16 MB のページ・サイズがサポートされるようになりました。IBM DB2 Version 9.1 for AIX 64-bit Edition などの、集中的なメモリー・アクセスを必要とし、大量の仮想メモリーを使用するアプリケーションでは、このラージ・ページの使用によってパフォーマンスを向上できるでしょう。

注: ラージ・ページを使用可能にすると、セルフチューニング・メモリー・マネージャーは全体のデータベース・メモリー消費量を自動的には調整しなくなります。それで、この使用を考慮するのは、データベース・メモリー所要量が比較的に静的な、よく定義されたワークロードの場合だけに限るべきです。

1. vmo コマンドの実行方法に関する詳しい説明は、 AIX のマニュアルを参照してください。
2. メモリーを固定してラージ・ページをサポートするようにシステムを構成する場合には、十分な注意が必要です。あまり多くのメモリーを固定しすぎると、固定されていないメモリー・ページにかかるページング・アクティビティーの負荷が

大きくなります。ラージ・ページに物理メモリーを割り振りすぎると、4 KB ページのサポートにメモリーが不足して、かえってシステムのパフォーマンスを低下させることになります。

3. また、DB2_LARGE_PAGE_MEM レジストリー変数の設定によっても、メモリーは固定されます。

AIX オペレーティング・システム・コマンドを実行できる root 権限が必要です。

1. ラージ・ページがサポートされるように AIX サーバーを構成するには、次のフラグを立てて vmo コマンドを発行します。

   ```
   vmo -r -o lgpg_size=<LargePageSize> -o lgpg_regions=<LargePages>
   ```

   ここで、<LargePageSize> はハードウェアによってサポートされるラージ・ページのサイズ (バイト)、<LargePages> は予約するラージ・ページ数をそれぞれ指定します。 例えば、ラージ・ページのサポート用に 25 GB を割り振る必要がある場合は、次のコマンドを実行します。

   ```
   vmo -r -o lgpg_size=16777216 -o lgpg_regions=1600
   ```

2. bosboot コマンドを実行します。これによって次のシステム・ブート時に、先に実行した vmo コマンドが有効になります。
3. サーバーが立ち上がったなら、これを固定したメモリーに使用できるようにします。
   - 次のフラグを立てて vmo コマンドを発行します。

     ```
     vmo -o v_pinshm=1
     ```

   - db2set コマンドを使用して DB2_LARGE_PAGE_MEM レジストリー変数を「DB」に設定した後、DB2 を開始します。

     ```
     db2set DB2_LARGE_PAGE_MEM=DB
     db2start
     ```

第 2 章 複数の DB2 コピー

バージョン 9 以上では、同一コンピューターで複数の DB2 コピーをインストールおよび実行できます。 DB2 コピーとは、同じコンピューター上の特定のロケーションにある DB2 データベース製品の 1 つ以上のインストールのことです。各 DB2 バージョン 9 のコピーのコード・レベルは、同じであっても異なっていてもかまいません。

このことには、以下のような利点があります。

- 異なる DB2 バージョンを必要とするアプリケーションを、同じコンピューター上で同時に実行することができます。
- 異なる機能のための独立した DB2 製品のコピーを実行できます。
- 実動データベースをその後のバージョンの DB2 製品に移動する前に、同じコンピューターでテストすることができます。
- 独立系ソフトウェア・ベンダーの場合、DB2 サーバー製品をご自分の製品に組み込み、ユーザーからは DB2 データベースが分からないようにできます。COM+ アプリケーションでは一度に 1 つの Data Server Runtime Client しか使用できないため、COM+ アプリケーションでは Data Server Runtime Client の代わりに、IBM Data Server Driver for ODBC and CLI をアプリケーションとともに使用および配布してください。 IBM Data Server Driver for ODBC and CLI にはこの制限がありません。

デフォルトの IBM データベース・クライアント・インターフェース・コピー

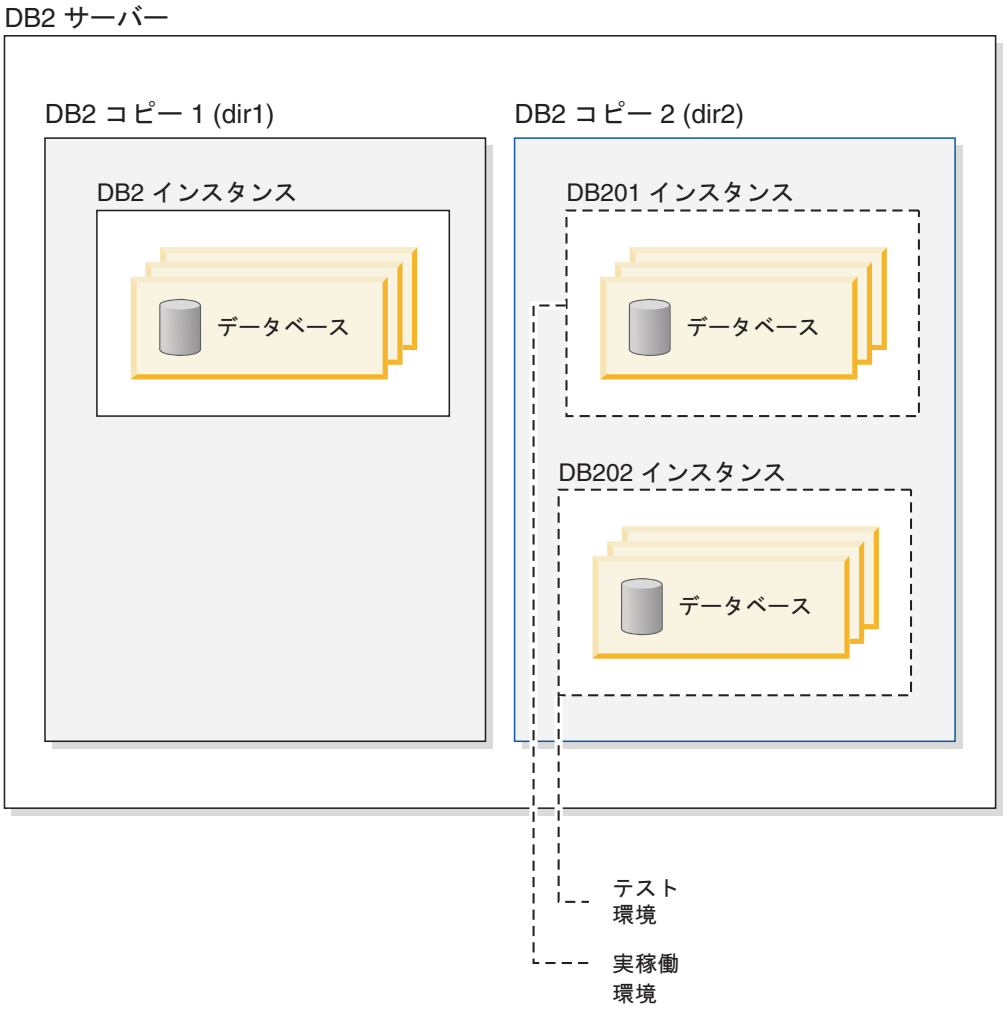
1 台のコンピューター上に、複数の DB2 コピーと、デフォルトの IBM データベース・クライアント・インターフェース・コピーを持つことができます。これによってクライアント・アプリケーションに、デフォルトでデータベースとのインターフェースに必要な ODBC ドライバー、CLI ドライバー、および .NET データ・プロバイダー・コードが提供されます。

バージョン 9.1 以降、DB2 コピーに IBM データベース・クライアント・インターフェース・コピーのコードが組み込まれるようになりました。バージョン 9.5 以降でインストールを選択できる新しい製品には、クライアント・アプリケーションとデータベースのインターフェースを可能にするために必要なコードが組み込まれています。この製品は、IBM Data Server Driver for ODBC, CLI, and .NET (DSDRIVER) です。バージョン 9.5 以降では、DSDRIVER を DB2 コピーのインストールとは別の IBM データ・サーバー・ドライバー・コピー上にインストールすることができます。

バージョン 9.1 以降、コンピューター上に複数の DB2 コピーをインストールできるようになりました。バージョン 9.5 以降では、コンピューター上に複数の IBM データベース・クライアント・インターフェース・コピーおよび複数の DB2 コピーをインストールすることが可能です。新しい DB2 コピーまたは新しい IBM デー

タ・サーバー・ドライバー・コピーをインストールするときに、デフォルトの DB2 コピーおよびデフォルトの IBM データベース・クライアント・インターフェース・コピーを変更する機会があります。

以下の図は、DB2 サーバーにインストールされている複数の DB2 コピーを示しています。これは DB2 データベース製品の任意の組み合わせです。



バージョン 8 およびバージョン 9 以降のコピーは、同じコンピューター上に共存させることができます。ただし、バージョン 8 がデフォルトの DB2 および IBM データベース・クライアント・インターフェース・コピーでなければなりません。インストール中に、デフォルトの DB2 コピーまたはデフォルトの IBM データベース・クライアント・インターフェース・コピーをバージョン 8 のコピーからバージョン 9 以降のコピーに変更することはできません。また、最初にバージョン 8 のコピーをマイグレーションまたはアンインストールしない限り、デフォルト・コピーを切り替えるためのコマンド db2swtch を後から実行することもできません。db2swtch コマンドを、バージョン 8 がシステム上に存在しているときに実行する場合、バージョン 8 がシステム上に見つかったためにデフォルトのコピーを変更できないことを示すエラー・メッセージを受け取ります。

複数の DB2 コピーまたは複数の IBM データ・サーバー・ドライバー・コピーをインストールした後、デフォルトの DB2 コピーまたはデフォルトの IBM® データベ

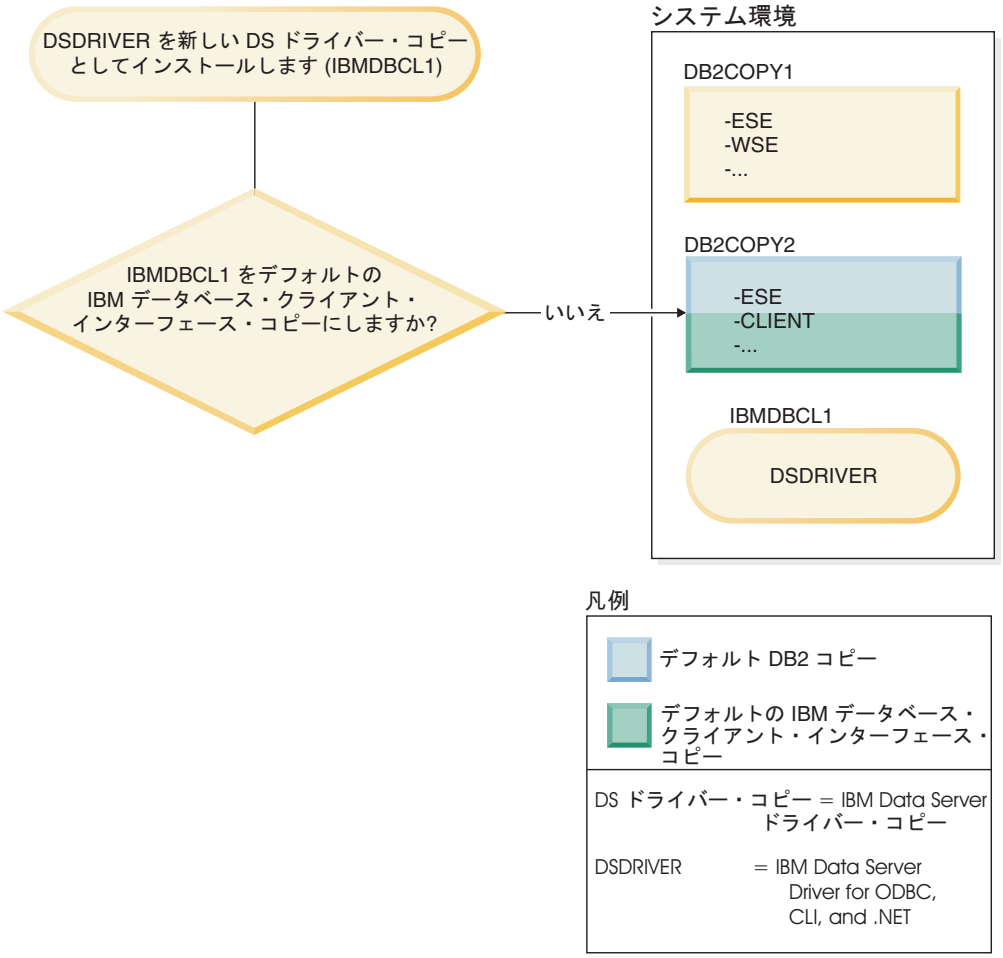
ース・クライアント・インターフェース・コピーのいずれかを変更したい場合があります。バージョン 8 がインストールされている場合、デフォルトの DB2 コピーを変更するにはその前に製品をアンインストールするか、バージョン 9 以降にマイグレーションする必要があり、デフォルトの IBM データベース・クライアント・インターフェース・コピーを変更するにはその前に製品をアンインストールするか、バージョン 9.5 以降にマイグレーションする必要があります。

クライアント・アプリケーションはいつでも、データ・サーバー・ドライバーがある場所 (DSDRIVER がインストールされているディレクトリー) に直行できます。

DB2 コピーまたは IBM データ・サーバー・ドライバー・コピーのいずれかをアンインストールする際、それがデフォルトの IBM データベース・クライアント・インターフェース・コピーだった場合、デフォルトは自動的に管理されます。選択したデフォルト・コピーが除去され、新しいデフォルトが自動的に選択されます。デフォルトの DB2 コピーをアンインストールするときに、それがシステムで最後 (唯一) の DB2 コピーではない場合、まずデフォルトを別の DB2 コピーに切り替えるように指示されます。

新規 IBM データベース・クライアント・インターフェース・コピーのインストール時のデフォルトの選択

バージョン 9.5 以降で、2 つの DB2 コピー (DB2COPY1 とDB2COPY2) をインストールしている場合のシナリオを考えてみましょう。 DB2COPY2 は、デフォルトの DB2 コピーであり、またデフォルトの IBM データベース・クライアント・インターフェース・コピーです。

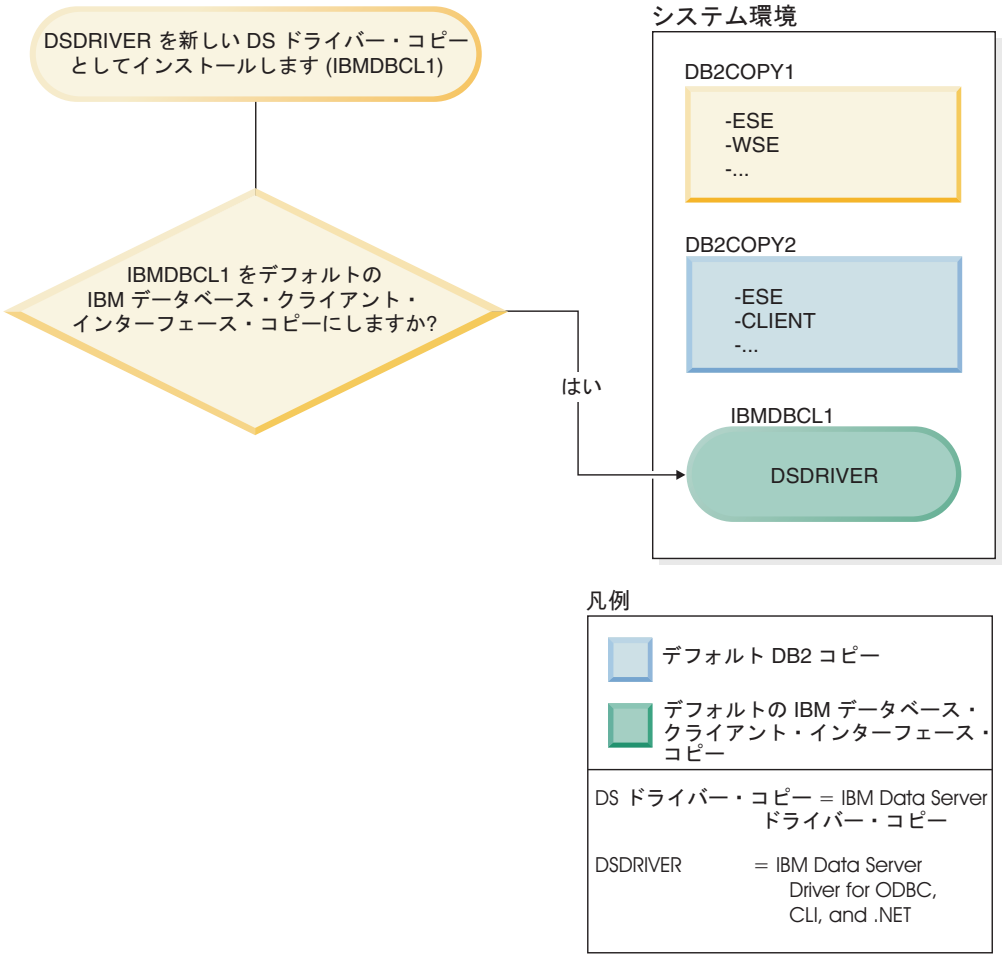


IBM Data Server Driver for ODBC, CLI, and .NET (DSDRIVER) を新規の IBM Data Server ドライバー・コピー上にインストールします。

新規の IBM Data Server ドライバー・コピー (IBMDBCL1) のインストール中に、この新規の IBM Data Server ドライバー・コピーをデフォルトの IBM データベース・クライアント・インターフェース・コピーにするかどうかを確認されます。

「いいえ」と応答した場合は、DB2COPY2 が引き続きデフォルトの IBM データベース・クライアント・インターフェース・コピーです。 (また、引き続きデフォルトの DB2 コピーでもあります。)

一方、同じシナリオですが、新規の IBM Data Server ドライバー・コピーをデフォルトの IBM データベース・クライアント・インターフェース・コピーにするかどうかを確認された際に「はい」と応答した場合を考えてみます。



この場合、IBMDBCL1 が、デフォルトの IBM データベース・クライアント・インターフェース・コピーになります。(DB2COPY2 が引き続きデフォルトの DB2 コピーです。)

複数の DB2 コピーを実行する場合の DAS の設定

バージョン 9 以降、複数の DB2 コピーを同じコンピューターで実行できるようになりました。これは、DB2 Administration Server (DAS) の動作に影響します。DAS は、データベース・マネージャー内の固有のコンポーネントであり、同じコンピューターにインストールされている DB2 コピーの数にかかわらず、アクティブにできるのは 1 つのバージョンのみという制限があります。このため、以下の制約事項および機能上必要となるものがあります。

サーバー上に存在できる DAS バージョンは 1 つだけであり、それはインスタンスを以下のように管理します。

- DAS がバージョン 9.1 またはバージョン 9.5 にある場合、それはバージョン 8 およびバージョン 9.1 またはバージョン 9.5 のインスタンスを管理できます。
- DAS がバージョン 8 にある場合、それはバージョン 8 のインスタンスのみを管理できます。バージョン 8 の DAS をマイグレーションするか、それをドロップしてバージョン 9.5 の新規 DAS を作成すると、バージョン 8 とバージョン 9.1

のインスタンスを管理できます。これは、コントロール・センターを使用してインスタンスを管理する場合にのみ必要です。

同じコンピューターにインストールされている DB2 コピーの数に関わらず、どの時点でも、あるコンピューター上で作成できる DAS は 1 つだけです。この DAS は、同じコンピューター上にあるすべての DB2 コピーによって使用されます。バージョン 9 以降では、 DAS は現在インストールされているどの DB2 コピーにも属することが可能です。

あるバージョン 9.5 コピーから別のバージョン 9.5 コピーに DAS を移動するには、dasupdt コマンドを使用します。バージョン 9.1 コピーからバージョン 9.5 コピーに DAS を移動する場合、dasupdt は使用できず、バージョン 9.1 からバージョン 9.5 にマイグレーションする必要があります。

Windows オペレーティング・システム上では、dasupdt コマンドを、DAS を同じバージョンの新規デフォルト DB2 コピーに移動する必要がある場合にも使用できます。

注:

- dasupdt コマンドを使用できるのは、同じリリースのさまざまな DB2 コピー間で (つまり、異なるフィックスパック間) で DAS を移動する場合だけです。 DAS のセットアップに使用することはできません。
- バージョン 9.1 からバージョン 9.5 の DAS にバージョン 8 からマイグレーションする場合は、dasmigr コマンドを使用します。
- DAS がセットアップされていない場合、通常の DAS セットアップ手順に従って DB2 コピーのいずれかで DAS をセットアップする必要があります。

複数の DB2 コピーを使用する場合のデフォルト・インスタンスの設定 (Windows)

バージョン 9.1 以降、DB2INSTANCE 環境は、環境で使用するように現在セットアップされている DB2 コピーに応じて設定されます。これを現行コピー内の別のインスタンスに明示的に設定しない場合、DB2INSTDEF プロファイル・レジストリー変数で指定されたデフォルト・インスタンスがデフォルトになります。

注: DB2INSTDEF は、使用中の現行の DB2 コピーに固有の、デフォルトのインスタンス変数です (つまり、各 DB2 コピーはそれぞれ固有の DB2INSTDEF を持ちます)。 DB2INSTANCE は、使用する現行のインスタンスに設定されます。次のとおりです。

- DB2INSTANCE が特定の DB2 コピーに設定されていない場合、DB2INSTDEF の値がその DB2 コピーに使用されます。
- DB2INSTANCE は、使用する DB2 コピーの下のインスタンスに対してのみ有効です。ただし、db2envar.bat コマンドを実行してコピーを切り替える場合、DB2INSTANCE は、最初に切り替えた DB2 コピーの DB2INSTDEF の値に更新されます。

すべてのグローバル・プロファイル・レジストリー変数は、SET VARIABLE=<variable_name> を使用して指定していなければ、DB2 コピーに固有です。

データベース・マネージャーの複数インスタンス

1 つのサーバーにデータベース・マネージャーの複数インスタンスを作成することができます。これはつまり、物理的に 1 つのコンピューターに同じ製品のインスタンスを複数作成し、それらを並行して稼働させることができるということです。これにより、複数の環境を柔軟に設定できます。

注: 2 つの異なる DB2 コピーに同じインスタンス名を使用することはできません。

次の環境を作成するために、複数のインスタンスが必要になる場合があります。

- 開発環境を実稼働環境から分離する。
- インスタンスがサービスする特定のアプリケーションの環境ごとに別個に調整する。
- 機密情報を管理担当者から保護する。例えば、給与計算データベースをそのインスタンスで保護し、他のインスタンスの所有者が給与計算データを見ることができないようにすることができます。

注:

- (UNIX® オペレーティング・システムのみ) 複数のインスタンス間での環境競合を防ぐために、各インスタンスのホーム・ディレクトリーをローカル・ファイル・システムに設定する必要があります。
- (Windows® プラットフォームのみ) インスタンスはノード・ディレクトリーにローカルまたはリモートのいずれかとしてカタログされます。デフォルト・インスタンスは DB2INSTANCE 環境変数で定義されます。データベースの作成、アプリケーションの強制終了、データベースのモニター、またはデータベース・マネージャー構成の更新などの、インスタンス・レベルでしか行うことのできない保守およびユーティリティー・タスクを実行するために、他のインスタンスに ATTACH (アタッチ) することができます。デフォルト・インスタンスにないインスタンスにアタッチしようとすると、そのインスタンスとの通信方法を判別するためにノード・ディレクトリーが使用されます。
- (すべてのプラットフォーム) DB2 データベース・プログラム・ファイルは 1 つのロケーションに物理的に保管され、各インスタンスは、それが属するコピーを指しているので、作成されるインスタンスごとにプログラム・ファイルが複写されるわけではありません。いくつかの関連するデータベースを、単一のインスタンス内に置くことができます。

複数インスタンス (Windows)

同じコンピューター上で、データベース・マネージャーの複数のインスタンスを実行することが可能です。それぞれのデータベース・マネージャーのインスタンスは独自のデータベースを保守し、独自のデータベース・マネージャー構成パラメーターを持っています。

注: インスタンスは、1 台のコンピューター上で、異なる DB2 コピーに属することもでき、これらのコピーは、データベース・マネージャーの異なるレベルに存在することが可能です。

データベース・マネージャーの 1 つのインスタンスは以下のもので構成されます。

- インスタンスを表す Windows サービス。サービスの名前は、インスタンス名と同じです。 (「サービス」パネルでの) サービスの表示名は、インスタンス名の前にストリング ″DB2 - ″ が付きます。例えば、インスタンス名が「DB2」であれば、「DB2」という名前の Windows サービスが存在し、その表示名は「DB2 - <DB2 コピー名> - DB2」です。

  注: Windows サービスは、クライアント・インスタンス用には作成されません。
- インスタンス・ディレクトリー。このディレクトリーには、データベース・マネージャー構成ファイル、システム・データベース・ディレクトリー、ノード・ディレクトリー、データベース接続サービス (DCS) ディレクトリー、およびインスタンスに関連したすべての診断ログとダンプ・ファイルが含まれます。デフォルトでは、インスタンス・ディレクトリーは SQLLIB ディレクトリー内のサブディレクトリーとなり、インスタンス名と同じ名前になります。例えば、インスタンス「DB2」のインスタンス・ディレクトリーは `C:¥SQLLIB¥DB2` です (`C:¥SQLLIB` はデータベース・マネージャーのインストール場所)。レジストリー変数 DB2INSTPROF を使用して、インスタンス・ディレクトリーのデフォルト場所を変更することができます。レジストリー変数 DB2INSTPROF の値を他の場所に設定した場合、インスタンス・ディレクトリーは DB2INSTPROF で指定されたディレクトリーの下に作成されます。例えば、`DB2INSTPROF=D:¥DB2PROFS` と設定した場合、インスタンス・ディレクトリーは `D:¥DB2PROFS¥DB2` となります。
  - db2set.exe -g コマンドを使用して、DB2INSTPROF を c:¥DB2PROFS に設定する。
  - DB2ICRT.exe コマンドを実行して、インスタンスを作成する。
- Windows オペレーティング・システムでインスタンスを作成するとき、インスタンスのディレクトリーや db2cli.ini ファイルといったユーザー・データ・ファイルのデフォルトの場所は、以下のディレクトリーになります。
  - Windows XP および Windows 2003 オペレーティング・システムでは、Documents and Settings¥All Users¥Application Data¥IBM¥DB2¥コピー名
  - Windows Vista オペレーティング・システムでは、ProgramData¥IBM¥DB2¥コピー名

DB2 コピーの更新 (Windows)

DB2 製品を更新する場合、既存の DB2 コピーを更新するか、または新規の DB2 コピーをインストールするかを指定する必要があります。 DB2 コピーを更新するには、「既存の処理」オプションを選択する必要があります。

複数の DB2 コピーを同時に更新することはできません。同じコンピューターにインストールされている可能性がある他の DB2 コピーを更新するには、インストールを再実行する必要があります。インストールでは、バージョン 8 またはバージョン 9.1 のコピーをマイグレーションするか (同じパス内で)、またはバージョン 8 のインストールを変更せずに新規のバージョン 9.1 またはバージョン 9.5 コピーをインストールするオプションが提供されます。

- マイグレーションを選択する場合、バージョン 8 のインストールは除去されます。

- 新規の DB2 コピーのインストールを選択する場合、db2ckmig および db2imigr コマンドを使用して後からインスタンスをマイグレーションすることを選択できます。

db2iupdt コマンドを使用して、DB2 インスタンスをバージョン 9.1 またはバージョン 9.5 の異なる DB2 コピー間で移動させたり、db2imigr コマンドを使用してバージョン 8 インスタンスをバージョン 9.1 またはバージョン 9.5 に移動させたりすることができます。

注:

- バージョン 7 とバージョン 9.1 またはバージョン 9.5 との共存はサポートされていません。
- 32 ビットの DB2 データ・サーバーと 64 ビットの DB2 データ・サーバーを同じ Windows X64 コンピューター上で共存させることはサポートされていません。

  バージョン 8 の 32 ビットの X64 DB2 インストールからバージョン 9.1 またはバージョン 9.5 の 64 ビットのインストールにマイグレーションすることはできません。その代わりに、バージョン 9.1 またはバージョン 9.5 の 32 ビットにマイグレーションし、X64 DB2 データ・サーバー・インストールを使用して 64 ビットにマイグレーションする必要があります。32 ビット版は除去されます。複数の 32 ビットの DB2 コピーをインストールしている場合、すべてのインスタンスを 1 つの DB2 コピーに移動して、コンピューターからこれらのコピーを除去する必要があります。
- バージョン 9.1 またはバージョン 9.5 の複数のコピーがある場合、インストール・オプションは、新規コピーのインストールか、DB2 の既存コピーの処理 (この場合、アップグレードまたは新規フィーチャーの追加が可能) のいずれかです。マイグレーションのアクションは、バージョン 9.1 またはバージョン 9.5 コピーに加えてバージョン 8 コピーもある場合にのみ選択可能です。
- バージョン 8 またはバージョン 9.1 がインストールされている場合、インストール・オプションは、既存のバージョン 8 またはバージョン 9.1 からバージョン 9.5 コピーへのマイグレーションか、新規 DB2 コピーのインストールのいずれかです。
- バージョン 7 以前がインストールされている場合、インストール時にはバージョン 9.1 またはバージョン 9.5 へのマイグレーションがサポートされていないことを示すメッセージが表示されます。新規の DB2 コピーは、バージョン 7 をアンインストールした後でなければインストールできません。言い換えれば、バージョン 7 とバージョン 9.1 またはバージョン 9.5 は共存させることができません。
- インスタンスをバージョン 9.1 またはバージョン 9.5 のコピーから別のコピーに移動させるには、db2iupdt コマンドを使用します。
- db2imigr コマンドを使用してインスタンスをバージョン 8 からマイグレーションする場合、ODBC データ・ソースはすべて再構成する必要があります。

複数のインスタンスの並行実行 (Windows)

複数のインスタンスを、同じ DB2 コピーまたは複数の異なる DB2 コピーで並行して実行することができます。

コマンド行を使用して、複数のインスタンスを同じ DB2 コピーで並行して実行するには:

1. 次のように入力して、開始する他のインスタンスの名前に DB2INSTANCE 変数を設定します。

   ```
   set db2instance=<another_instName>
   ```

2. db2start コマンドを入力して、インスタンスを開始します。

複数のインスタンスを複数の異なる DB2 コピーで並行して実行するには、以下のいずれかの方法を使用します。

- 「DB2 コマンド・ウィンドウ」を使用します。「スタート」→「すべてのプログラム」→「IBM DB2」→「*<DB2 コピー名>*」→「コマンド行ツール」→「DB2 コマンド・ウィンドウ」の順に選択します。「コマンド・ウィンドウ」は、選択した特定の DB2 コピー用の適切な環境変数で事前にセットアップされています。
- 「コマンド・ウィンドウ」から db2envar.bat を使用します。
  1. 「コマンド・ウィンドウ」をオープンします。
  2. アプリケーションで使用する DB2 コピーの完全修飾パスを使用して、db2envar.bat ファイルを実行します。

     ```
     <DB2 Copy install dir>¥bin¥db2envar.bat
     ```

特定の DB2 コピーに切り替えた後、前述のセクション「複数のインスタンスを同じ DB2 コピーで並行して実行するには」で指定された方法に従ってインスタンスを開始します。

同じまたは異なる DB2 コピー上のインスタンスでの作業

複数のインスタンスを、同じ DB2 コピーまたは複数の異なる DB2 コピーで並行して実行することができます。

同一の DB2 コピーでインスタンスに関する作業を行うには、以下のようにする必要があります。

1. すべてのインスタンスを同一の DB2 コピーに作成またはマイグレーションします。
2. そのインスタンスに対してコマンドを発行する前に、DB2INSTANCE 環境変数をその作業対象のインスタンス名に設定する必要があります。

あるインスタンスが別のデータベース・インスタンスにアクセスするのを防ぐために、インスタンス用のデータベース・ファイルはインスタンス名と同じ名前のディレクトリーの下に作成されます。例えば、インスタンス「DB2」用のデータベースを C: ドライブに作成する場合、データベース・ファイルは C:¥DB2 ディレクトリーの下に作成されます。同様に、インスタンス TEST 用のデータベースを C: ドライブに作成する場合、データベース・ファイルは C:¥TEST ディレクトリーの下に作

成されます。デフォルトで、その値は DB2 製品がインストールされているドライブのドライブ名になります。詳しくは、*dftdbpath* データベース・マネージャーの構成パラメーターを参照してください。

複数の DB2 コピーのあるシステムで特定の 1 つのインスタンスに関する作業を行うには、以下の方式のどちらかを使用します。

- 「コマンド・ウィンドウ」を次のように使用します。「スタート」→「すべてのプログラム」→「IBM DB2」→ *<DB2 コピー名>* →「コマンド行ツール」→「コマンド・ウィンドウ」。この「コマンド・ウィンドウ」は、選択した特定の DB2 コピー用に正しい環境変数で事前にセットアップされています。
- 「コマンド・ウィンドウ」から db2envar.bat を使用します。
  1. 「コマンド・ウィンドウ」をオープンします。
  2. アプリケーションで使用する DB2 コピーの完全修飾パスを使用して、db2envar.bat ファイルを実行します。

     ```
     <DB2 Copy install dir>¥bin¥db2envar.bat
     ```

第 3 章 オートノミック・コンピューティング

DB2 のオートノミック・コンピューティング環境は、自己構成、自己修復、自己最適化、および自己防御を行います。オートノミック・コンピューティングは、発生する状態を察知して応答することにより、コンピューター環境の管理という責任をデータベース管理者からテクノロジーへとシフトします。

以下の自動フィーチャーにより、データベース・システムの管理がサポートされます。

- セルフチューニング・メモリー
- 自動ストレージ
- (コンプレッション) ディクショナリーの自動作成 (ADC)
- 自動データベース・バックアップ
- 自動統計収集
- 構成アドバイザー
- ヘルス・モニター
- ユーティリティー・スロットル

自動フィーチャー

自動フィーチャーにより、データベース・システムの管理がサポートされます。自動フィーチャーを使用すれば、システムで自己診断することができ、履歴問題データと照合してリアルタイム・データを分析することにより問題が生じる前に問題を予測することができます。一部の自動ツールについては、ご使用のシステムに対して介入なしで変更するように構成して、サービスが中断されないようにすることができます。

データベースの作成時には、以下の自動フィーチャーのいくつかがデフォルトで使用可能になりますが、その他のフィーチャーは手動で使用可能にする必要があります。

セルフチューニング・メモリー (単一パーティション・データベースのみ)
セルフチューニング・メモリー・フィーチャーにより、メモリー構成のタスクが単純化されます。このフィーチャーでは、いくつかのメモリー構成パラメーターの値およびバッファー・プールのサイズを自動的に繰り返し調整してパフォーマンスを最適化することによって、ワークロードにおける大きな変化に対応します。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 (例えば、ソート機能、パッケージ・キャッシュ、ロック・リスト、およびバッファー・プール) の間で動的に分配します。データベースの作成後に、データベース構成パラメーター **self_tuning_mem** を OFF に設定することにより、セルフチューニング・メモリーを使用不可にすることができます。

自動ストレージ
自動ストレージ・フィーチャーは、表スペースのストレージ管理を単純化します。データベースを作成する際に、データベース・マネージャーが表スペ

ース・データを配置するストレージ・パスを指定してください。これで、表スペースを作成してデータを取り込む際に、データベース・マネージャーにより表スペース用のコンテナーとスペース割り振りが管理されます。

(コンプレッション) ディクショナリーの自動作成 (ADC)
コンプレッション・ディクショナリーは、COMPRESS 属性を YES に定義した表に対するデータの設定操作時に自動的に作成されます。ただし、コンプレッション・ディクショナリーがその物理表またはパーティション内にまだ存在せず、表が (例えば、挿入またはロード処理によって) 追加されたデータの結果として、サイズが約 1 MB に達した場合に、このディクショナリーが作成されて表に挿入されます。表の COMPRESS 属性が有効なままであれば、コンプレッション・ディクショナリーの作成が圧縮を受けた後に、すべてのデータが表に移動します。

自動データベース・バックアップ
データベースは、ハードウェアまたはソフトウェアのさまざまな障害のために使用不可能になることがあります。データベースの最新の全バックアップを持っておくことは、システムの災害時リカバリーを計画し、実装する上で不可欠です。災害時リカバリー計画の一部として自動データベース・バックアップを使用し、データベース・マネージャーが適切かつ定期的にデータベースをバックアップできるようにします。

自動統計収集
自動統計収集は、最新の表統計を保持するようにすることによって、データベース・パフォーマンスを向上するのに役立ちます。データベース・マネージャーは、ワークロードに必要な統計と更新の必要な統計を判断します。統計の収集は、非同期的に (バックグラウンドで) 行うことも、同期的に (SQL ステートメントがコンパイルされるときに実行時統計を収集することによって) 行うこともできます。それから DB2 オプティマイザーが、正確な統計に基づいてアクセス・プランを選択できます。データベースの作成後に、データベース構成パラメーター **auto_runstats** を OFF に設定することにより、自動統計収集を使用不可にすることができます。リアルタイム統計収集を有効にできるのは、自動統計収集が使用可能になっている場合に限ります。リアルタイム統計収集は、**auto_stmt_stats** 構成パラメーターで制御します。

構成アドバイザー
データベースを作成する際、このツールが自動的に実行され、データベース構成パラメーターおよびデフォルトのバッファー・プールのサイズ (IBMDEFAULTBP) を判別して設定します。これらの値は、システム・リソースおよびシステムの使用目的に基づいて選択されます。つまり、この最初の自動調整により、デフォルト値で作成できる同等のデータベースに比べて、ご使用のデータベースのパフォーマンスが向上します。さらに、データベース作成後にシステムを調整する時間が短くて済みます。構成アドバイザーは (データベースへのデータ取り込み後であれ) いつでも実行して推奨ツールを適用したり、現行のシステム特性に基づいてパフォーマンスを最適化できるよう一連の構成パラメーターを任意に適用させることができます。

ヘルス・モニター
ヘルス・モニターはサーバー・サイドのツールで、積極的に状態をモニターしたり、パフォーマンスの低下や停止の危険性があるデータベース環境を調

整します。ユーザーの側でいかなる形のアクティブなモニターも行わずに、正常性に関するさまざまな情報が提供されます。正常性のリスクが生じた場合、データベース・マネージャーはそのことを通知し、続行する方法もアドバイスします。ヘルス・モニターはスナップショット・モニターを使用してシステムに関する情報を収集しますが、パフォーマンス上の不利益をもたらすことはありません。さらに、情報を収集するためにスナップショット・モニター・スイッチをオンにすることもありません。

ユーティリティー・スロットル
このフィーチャーは保守ユーティリティーのパフォーマンス上の影響を調整し、実動期間中に、そうしたユーティリティーを同時に実行できるようにします。スロットル・ユーティリティーの*影響ポリシー* はデフォルトで定義済みですが、スロットル・ユーティリティーを実行する場合は、*影響優先順位* を設定する必要があります。スロットル・システムは、スロットル・ユーティリティーがその影響ポリシーに抵触することがない範囲で最大限頻繁に実行されるようにします。現在のところスロットル化できるのは、統計収集、バックアップ操作、リバランス操作、および非同期索引クリーンアップです。

自動保守

データベース・マネージャーには、データベースのバックアップを実行したり、統計を最新の状態に維持したり、必要に応じて表および索引を再編成したりするためのさまざまな自動保守機能が含まれています。データベースの保守は、データベースのパフォーマンスとリカバリー可能性を最適化しておく際に不可欠のアクティビティーです。

データベースの保守には、以下のアクティビティーの一部またはすべてが含まれます。

- **バックアップ。** データベースをバックアップする際には、データベース・マネージャーでは、元のデータに障害や損傷が起こった場合に備えて、データベースのデータのコピーをとって、それを別のメディアに保管します。自動データベース・バックアップ機能により、データベースが適切かつ定期的にバックアップされるようになるため、バックアップのタイミングについて心配したり、BACKUP コマンドの構文を覚えたりする必要がなくなります。
- **データのデフラグ (表または索引の再編成)。**この保守アクティビティーにより、表へのデータベース・マネージャーのアクセス効率が向上します。自動再編成機能により、表および索引のオフライン再編成が管理されるため、データをいつどのように再編成するかを心配しなくてすむようになります。
- **データ・アクセスの最適化 (統計の収集)。** データベース・マネージャーは、表のデータ、索引のデータ、あるいは表とその索引の両方のデータに関するシステム・カタログ統計を更新します。オプティマイザーはこれらの統計を使って、データにアクセスするにはどのパスを使用したらよいかを判断します。自動統計収集機能は、表に関する統計データを最新のものに維持することにより、データベースのパフォーマンスの向上を図ります。その目的は、オプティマイザーが正確な統計データに基づいてアクセス・プランを選択できるようにすることです。
- **統計のプロファイル。** 自動統計プロファイルは、統計データのうち古くなったもの、欠落しているもの、または正しくないものを検出したり、照会フィードバッ

クに基づいて統計プロファイルを生成したりすることによって、表統計データをいつどのように収集するかに関するアドバイスを提供します。

保守アクティビティーを実行するかどうか、そしていつ実行するかの判断には手間がかかる可能性がありますが、自動保守によりこの負担が取り除かれます。自動保守のフィーチャーの使用可能化は、自動保守データベース構成パラメーターを使用することによって簡単かつ柔軟に管理することができます。「自動保守の構成」ウィザードを使用して、保守の目的を指定することができます。データベース・マネージャーにより、保守アクティビティーを実行する必要があるかどうかがこれらの目的に基づいて判別され、次回の保守時間枠 (定義した時間枠) の中で、必要な保守アクティビティーのみが実行されます。

保守時間枠

保守時間枠は、自動保守アクティビティー (バックアップ、統計の収集、統計プロファイル、および再編成) を実行するために定義する時間枠です。オフライン時間枠は、データベースへのアクセスが利用できない時間である場合があります。オンライン時間枠は、ユーザーがデータベースに接続することが許可されている時間であることもあります。

保守時間枠は、タスク・スケジュールとは異なります。保守時間枠中でも、常に自動保守アクティビティーが実行されるとは限りません。実際は、システムに対して保守アクティビティーを実行する必要があるかどうかを、データベース・マネージャーが保守アクティビティーごとに評価します。保守要件が満たされていない場合に保守アクティビティーが実行されます。データベースがすでに適切に保守されていれば、保守アクティビティーは実行されません。

自動保守アクティビティーをいつ実行するかを検討してください。自動保守アクティビティーではシステム上のリソースが消費されるため、これらを実行するとデータベースのパフォーマンスに影響する可能性があります。これらのアクティビティーの一部では、表、索引、およびデータベースへのアクセスも制限されます。このため、データベース・マネージャーで保守アクティビティーの実行が可能な適切な時間枠を指定する必要があります。これらの期間は、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、オフラインおよびオンラインの保守時間枠として指定します。

オフライン保守アクティビティー

オフライン保守アクティビティー (オフライン・データベース・バックアップおよび表と索引の再編成) とは、オフライン保守時間枠でのみ実行可能な保守アクティビティーのことです。ユーザー・アクセスが影響を受ける範囲は、実行する保守アクティビティーによって異なります。

- オフライン・バックアップ中は、どのアプリケーションもデータベースに接続できません。そのとき接続されているアプリケーションがあれば、いずれも強制的に切断されます。
- オフライン表再編成またはオフライン索引再編成 (データのデフラグ) 中は、アプリケーションで表のデータにアクセスすることはできますが、更新することはできません。

オフライン保守アクティビティーは、指定された時間枠を超えるとしても完了するまで実行されます。内部スケジューリング・メカニズムでは、時間の

経過と共に、ジョブの完了時刻の最適な見積もり方法が学習されます。オフライン保守時間枠が短すぎるために、特定のデータベース・バックアップまたは再編成活動を実行できない場合には、その次の期間でスケジューラーはそのジョブを開始せず、ヘルス・モニターを通じてオフライン保守時間枠を長くすることが必要であるということを通知します。

オンライン保守アクティビティー

オンライン保守アクティビティー (自動統計収集とプロファイル、オンライン索引再編成、およびオンライン・データベース・バックアップ) とは、オンライン保守時間枠でのみ実行可能な保守アクティビティーのことです。オンライン保守アクティビティーを実行しても、そのとき接続されているどのアプリケーションも接続が維持され、また新規接続を確立することができます。システムへの影響を最小限に抑えるため、オンライン・データベース・バックアップと自動統計収集およびプロファイルは最適なユーティリティー・スロットル・メカニズムによってスロットルされます。

オンライン保守アクティビティーは、指定された時間枠を超えるとしても完了するまで実行されます。

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。

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。

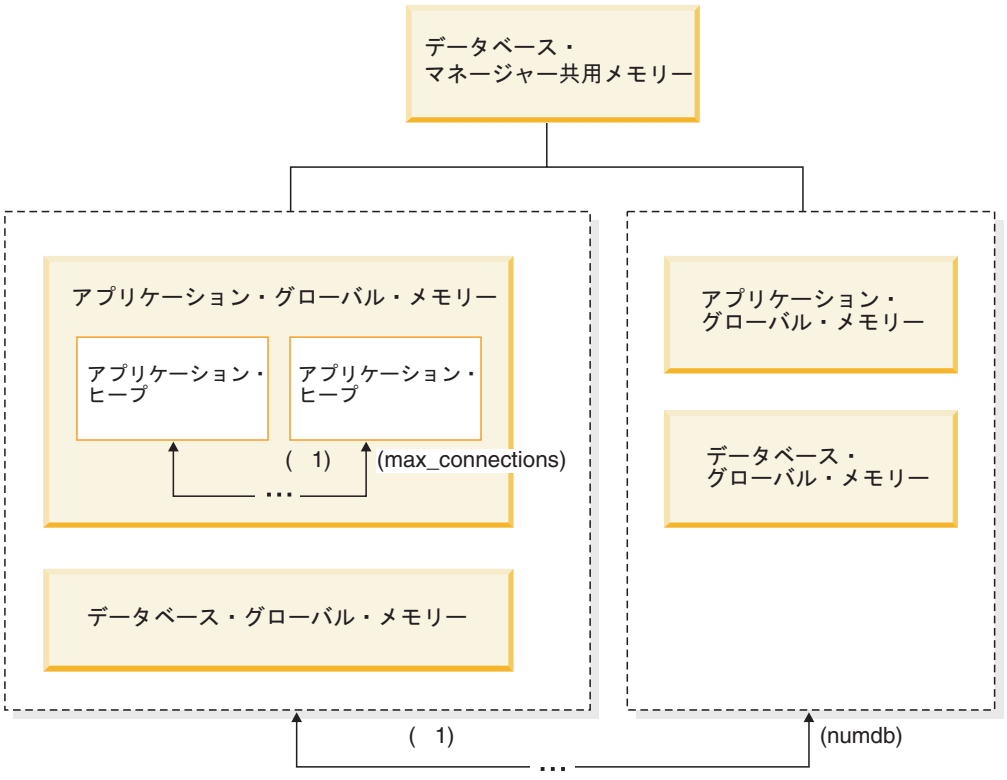


図 1. データベース・マネージャーによって使用されるメモリーのタイプ

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)

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。

操作の詳細、制限、およびメモリー・パラメーター間の相互作用

セルフチューニング・メモリーを使用可能にして、ほとんどのメモリー関連の構成パラメーターでデフォルトの AUTOMATIC 設定を使用できますが、操作の詳細、制限、および別のメモリー・パラメーターとの相互作用 (特に **instance_memory**、**database_memory**、および **appl_memory** の各パラメーター間の相互作用) について把握しておくと、それらの設定の管理を行いやすくなりますし、特定の状況で『メモリー不足』エラーが依然として生じ得る理由を理解する上でも役立つ場合があります。

目的

DB2 データベース・マネージャーは、基本的に以下の 2 つのタイプのメモリーを使用します。

- キャッシュ・ベース・メモリー。セルフチューニング・メモリー・マネージャー (STMM) によって制御され、種々のパフォーマンス・ヒープに配布されます。**database_memory** 構成パラメーターを使用すると、使用可能なキャッシュ・ベース・メモリーの最大量を制限したり、AUTOMATIC に設定し、セルフチューニング・メモリー・マネージャー (STMM) がキャッシュ・ベース・メモリーの全体量を管理するようにしたりできます。
- ファンクション・メモリー。アプリケーション・プログラムが使用します。**appl_memory** 構成パラメーターを使用して、アプリケーション・メモリーつまりファンクション・メモリーの最大量を制御します。このメモリーは、アプリケーション要求に応えるために DB2 データベース・エージェントによって割り振られます。デフォルトでは、この値は AUTOMATIC に設定されており、データベース・パーティションにより割り振られたメモリーの総量が、**instance_memory** 限度内である場合、アプリケーション・メモリー要求が許可されます。

プロセス

これまでのリリースでは、種々のオペレーティング・システムおよび DB2 のツールを使用してメモリーの様々な部分 (共用メモリー、専用メモリー、バッファー・プール・メモリー、locklist、sortheap など) を確認できましたが、DB2 データベース・マネージャーを使用してメモリー総量を確認することはほとんどできませんでした。いずれかのヒープがメモリー制限に達すると、アプリケーションのステートメントで「メモリー不足」エラー・メッセージが出て障害が起きました。DBA がそのヒープのメモリーを増やしてアプリケーションを再実行しても、別のヒープの別のステートメントで「メモリー不足」エラーが出るだけでした。現在では、デフォルトの AUTOMATIC 構成パラメーター設定を使用して、ファンクション・メモリー・ヒープの個々のハード上限を解除できるようになりました。

必要であれば (例えば、適切に動作していないデータベース・アプリケーションがかなり大量のメモリーを要求するというシナリオを回避する場合など)、**appl_memory** 構成パラメーターを使用して、アプリケーション・メモリー全体に対

する制限をデータベース・レベルに適用できます。また必要に応じて、個々のヒープ制限も適用できます。そのヒープの適切なデータベース構成パラメーターを AUTOMATIC 設定から修正値に変更すると、これを行えます。ファンクション・メモリー・ヒープすべてをデフォルトの AUTOMATIC 設定のままにして、**appl_memory** もデフォルトの AUTOMATIC 設定のままにすると、アプリケーション・メモリー使用量に関する唯一の制限は **instance_memory** 設定です。**instance_memory** も AUTOMATIC に設定すると、DB2 はメモリー使用量に関する上限を自動的に判別します。DBA では、使用されている **instance_memory** の総量、および現行の **instance_memory** 制限を、admin_get_dbp_mem_usage 表関数を使用して簡単に確認できます。

self_tuning_mem、instance_memory、database_memory、および appl_memory 構成パラメーター間の相互作用

セルフチューニング・メモリーが全面的に使用可能な場合 (**self_tuning_mem** が ON で、すべてのメモリー・パラメーターが AUTOMATIC に設定されている場合)、セルフチューニング・メモリー・マネージャーはシステム上で使用可能な空きメモリーを検査して、最適なパフォーマンスを得るためにキャッシュ・ベース・ヒープ専用にするメモリー量を決定します。キャッシュ・ベース・ヒープすべてが、**database_memory** サイズになります。 DB2 データベース・マネージャーの操作性と整合性を確保するために、キャッシュ・ベース・メモリー所要量に加えて、他のメモリーも必要となります。**instance_memory** とこうした 2 つのメモリー・コンシューマーとの差分を、アプリケーション・メモリー (**appl_memory**) で使用できます。アプリケーション・プログラムのファンクション・メモリーは、**instance_memory** 制限内にある限りは必要に応じて割り振られます。単一のアプリケーションが割り振ることができるメモリー量に関しては他に制限はありません。

またセルフチューニング・メモリー・マネージャーは、空きシステム・メモリーの残量、および空き **instance_memory** の残量を定期的に照会します。セルフチューニング・メモリー・マネージャーは (アプリケーションの障害を避けるため) パフォーマンス基準よりもアプリケーション所要量を重要視するので、アプリケーション・メモリー要求に対して十分な量の空きシステム・メモリーと **instance_memory** を確保するためにキャッシュ・ベース・ヒープを減らしてパフォーマンスを犠牲にします。アプリケーションが完了すると使用メモリーは解放されて、他のアプリケーションで再使用するか、セルフチューニング・メモリー・マネージャーで使用する **database_memory** 用に再利用する準備が整います。アプリケーション・アクティビティーが重たいためデータベース・システムのパフォーマンスが容認できないほど低下する場合、データベース・マネージャーで許可されるアプリケーション数を制御するか (例えば、接続コンセントレーターまたは DB2 9.5 のワークロード・マネージャー新機能を使用する)、システムに別のメモリー・リソースを追加することを考慮するのが役立つ場合があります。

制限 (「メモリー不足」エラーが依然として出る可能性がある場合)

アプリケーションが著しく大量のメモリーを急激に要求するなどメモリー使用量が急激に増えた際にそれに対応する十分な時間がセルフチューニング・メモリー・マネージャーにない場合、またはデータベース・ワークロードが急激に増える場合 (つまり同時にご使用のデータベースに多くの新しいアプリケーションが接続する場合)、依然として「メモリー不足」エラーが出される場合があります。こうした場

合、または大部分のアプリケーションが一定量のメモリーを使用していることを DBA が把握する場合、AUTOMATIC 設定ではなく **appl_memory** のハードコーディングされた値を使用するほうが良いことがあります。**appl_memory** がハード値 (例えば 2GB) に設定されていると、DB2 はその量を超えるアプリケーション・メモリー使用総量を許可しません。アプリケーション・メモリー使用総量が **appl_memory** 制限より少ない限りは、各アプリケーションは必要なメモリー量の使用を許可されます。**appl_memory** 制限または **instance_memory** 制限のどちらかに達すると、データベース・マネージャーがその制限に抵触するアプリケーション要求が出されると失敗し、適切な SQL コードが戻ります (実際に戻されるエラー・コードは、「メモリー不足」失敗が発生したアプリケーション操作の正確な位置によって異なります)。「メモリー不足」エラーが発生すると、DBA はエラー発生時に使用されていたメモリー量を判別するために db2diag.log を表示します。これは、調整が必要なメモリー・パラメーターがあるかどうかを見極めるのに役立ちます。

セルフチューニング・メモリーの使用可能化

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。

パーティション・データベース環境での自己チューニング・メモリー

パーティション・データベース環境で自己チューニング・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・データベースで自己チューニング・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・データベースで自己チューニング・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・データベースで自己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、変更を加えずに自己チューニング・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、自己チューニング・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しながらの自己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、自己チューニング・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、引き続き自己チューニング・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自己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、自己チューニング・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・パーティションで自己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・データベースで自己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、自己チューニング・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配分し、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。データ配分に偏りがあり、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。

メモリーおよびメモリー・ヒープの構成

単純化されたメモリー構成フィーチャーでは、メモリーに関連したほとんどの構成パラメーターで、デフォルトの AUTOMATIC 設定を使用する (したがって、調整がほとんど不要) ことにより、DB2 データ・サーバーで必要なメモリーおよびメモリー・ヒープを構成することができます。

単純化されたメモリー構成フィーチャーには、次の利点があります。

- **instance_memory** というパラメーターをただ 1 つ使用するだけで、データベース・マネージャーが専用および共有のメモリー・ヒープから割り振ることのできるすべてのメモリーを指定することができます。また、 **appl_memory** 構成パラメーターを使用することにより、DB2 データベース・エージェントによって割り振られるアプリケーション・メモリーの最大量を制御して、アプリケーション要求にサービスを提供することもできます。
- ファンクション・メモリーにのみ使用されるパラメーターを手動で調整する必要はありません。
- メモリー・ビジュアライザーを使用することにより、データベース・マネージャーの専用および共用メモリー・ヒープが現在消費しているメモリーの合計量を照会できます。 db2mtrk コマンドを使ってヒープの使用量をモニターしたり、ADMIN_GET_DBP_MEM_USAGE() 表関数を使って全体のメモリー使用量を照会したりすることもできます。
- デフォルトの DB2 構成では、調整がはるかに少なくて済み、作成した新規インスタンスに対して長所が活かされます。

次の表では、デフォルトの AUTOMATIC 設定となるメモリー構成パラメーターがリストされています。これらのパラメーターは必要に応じて動的に構成することも可能です。右端の列で説明されているように、AUTOMATIC 設定の意味が各パラメーターによって異なることに注意してください。

表 1. 値がデフォルトの AUTOMATIC になるメモリー構成パラメーター

構成パラメーターの名前	説明	AUTOMATIC 設定の意味
appl_memory	DB2 データベース・エージェントにより割り振られるアプリケーション・メモリーの最大量を制御して、アプリケーション要求にサービス提供します。	AUTOMATIC 設定は、データベース・パーティションにより割り振られたメモリーの総量が **instance_memory** 限度内である限り、すべてのアプリケーション・メモリー要求を許可します。

表 1. 値がデフォルトの AUTOMATIC になるメモリー構成パラメーター (続き)

構成パラメーターの名前	説明	AUTOMATIC 設定の意味
applheapsz	9.5 よりも前のバージョンでは、アプリケーションに対して動作する各データベース・エージェントが消費可能なアプリケーション・メモリーの量を示すものでした。バージョン 9.5 では、このパラメーターは、アプリケーション全体で消費可能なアプリケーション・メモリーの総量を示します。つまり、DPF、コンセントレーター、または SMP 構成では、以前のリリースで使用される **applheapsz** の値は、AUTOMATIC 設定を使用しない限り、増やすことが必要になる場合があることを意味します。	AUTOMATIC 設定は、**appl_memory** 限度または **instance_memory** 限度のいずれかに達するまで、アプリケーション・ヒープ・サイズが必要に応じて増加できるようにします。
database_memory (9.5 より前のバージョンでは、AUTOMATIC のデフォルト設定の適用は Windows プラットフォームと AIX プラットフォームに限られていました。バージョン 9.5 以降では AUTOMATIC はすべての DB2 サーバー製品のデフォルト設定となっています。)	データベース共有メモリー領域として予約されている共用メモリーの量を指定します。	使用可能になると、メモリー・チューナーはデータベースの全体のメモリー要件を決定し、データベース共用メモリーに割り振られるメモリー量を現行データベース要件に応じて増加または減少します。
dbheap	データベース・ヒープが使用する最大メモリーを決定します。	AUTOMATIC 設定は、**database_memory** 限度または **instance_memory** 限度のいずれかに達するまで、データベース・ヒープが必要に応じて増加できるようにします。
instance_memory	1 つのデータベース・パーティションに割り振ることができるメモリーの最大量を指定します。	AUTOMATIC 設定は、データベース・マネージャー・インスタンス全体によって消費される全体のメモリー量を特定の限度 (マシン上の物理 RAM の 75% から 95%) まで増加できるようにします。この限度は db2start の処理中に算出されます。
mon_heap_sz	データベース・システム・モニター・データに割り振られるメモリーの大きさ (ページ数) を決定します。	AUTOMATIC 設定は、**instance_memory** 限度に達するまで、モニター・ヒープが必要に応じて増加できるようにします。

表 1. 値がデフォルトの AUTOMATIC になるメモリー構成パラメーター (続き)

構成パラメーターの名前	説明	AUTOMATIC 設定の意味
stat_heap_sz	RUNSTATS コマンドを使って統計を収集するときに使用されるヒープの最大サイズを示します。	AUTOMATIC 設定は、**appl_memory** 限度または **instance_memory** 限度のいずれかに達するまで、統計ヒープ・サイズが必要に応じて増加できるようにします。
stmtheap	ステートメント・ヒープのサイズを指定します。これは SQL または XQuery ステートメントをコンパイルするための SQL または XQuery コンパイラー用のワークスペースとして使用されます。	AUTOMATIC 設定は、**appl_memory** 限度または **instance_memory** 限度のいずれかに達するまで、ステートメント・ヒープが必要に応じて増加できるようにします。

注: DBMCFG 管理ビューおよび DBCFG 管理ビューは、すべてのデータベース・パーティションで現在接続されているデータベースに関するデータベース・マネージャーの構成パラメーター情報をリトリーブします。 **mon_heap_sz**、**stmtheap**、および **stat_heap_sz** 構成パラメーターについては、このビュー上の DEFERRED_VALUE 列はデータベース活動化全体にわたっては持続しません。つまり、get dbm cfg show detail コマンドまたは get db cfg show detail コマンドを発行すると、この照会の出力には更新された (メモリー内の) 値が表示されます。

次の表では、構成パラメーターがインスタンスおよびデータベースのマイグレーションまたは作成時にデフォルトの AUTOMATIC 値に設定されるかどうかが示されています。

表 2. インスタンスおよびデータベースのマイグレーションおよび作成時に AUTOMATIC に設定される構成パラメーター

構成パラメーター	インスタンス・マイグレーションまたはインスタンス作成時に AUTOMATIC に設定	データベース・マイグレーション時に AUTOMATIC に設定	データベースの作成時に AUTOMATIC に設定
applheapsz[1]		X	X
dbheap		X	X
instance_memory	X		
mon_heap_sz[1]	X		
stat_heap_sz[1]		X	X
stmtheap[1]			X

メモリー構成の単純化への移行の一環として、以下のエレメントは推奨されなくなりました。

- 構成パラメーター **appgroup_mem_sz**、**groupheap_ratio**、および **app_ctl_heap_sz**。これらの構成パラメーターに代わって、新規の **appl_memory** 構成パラメーターが使用されるようになりました。

- db2mtrk メモリー・トラッカー・コマンドの **-p** パラメーター。このオプションは、専用エージェント・メモリー・ヒープをリストするものですが、すべてのアプリケーション・メモリー消費量をリストする **-a** パラメーターに置き換えられました。

メモリー・ビジュアライザーは、データベースによる最大アプリケーション・メモリー消費量を新規の **appl_memory** 構成パラメーターを使用して表示し、インスタンスによる最大メモリー消費量を更新された **instance_memory** 構成パラメーターを使用して表示します。また、メモリー・ビジュアライザーは、AUTOMATIC 設定を許可するすべての構成パラメーターの値も表示します。バージョン 9.5 データベースに関して、推奨されなくなった構成パラメーターの値はメモリー・ビジュアライザーで表示されませんが、以前のバージョンのデータベースでは、これらの値は表示されます。

instance_memory パラメーターを以下のリストで指定された値よりも大きい値に更新しようとすると、SQL5130N 戻りコードで失敗します。

- 4 GB (1 048 576 * 4 KB ページ) DB2 Express Edition および DB2 Express-C
- 16 GB (4 194 304 * 4 KB ページ) DB2 Workgroup Server Edition

高速コミュニケーション・マネージャー (FCM) 共有メモリーが割り振られている場合、システムの FCM 共有メモリー・サイズ全体のうちの各ローカル・データベース・パーティションの分が、そのデータベース・パーティションの **instance_memory** 限度になります。FCM メモリーの性質 (FCM バッファーの割り振りに失敗するとインスタンスがダウンする場合がある) のために、**instance_memory** 限度が原因で FCM メモリー要求が失敗することはありません。しかし、オペレーティング・システムがメモリーを割り振ることができない場合には失敗する可能性があります。 FCM メモリー要求のためにデータベース・パーティションが **instance_memory** 限度を超える場合、パーティションのメモリー使用量が **instance_memory** 限度より下のレベルに戻るまでは他のメモリー要求も失敗します。

エージェントおよびプロセス・モデルの構成

バージョン 9.5 では、プロセス・モデル関連パラメーターを構成するための、より柔軟で、あまり複雑でないメカニズムが提供されます。この構成の単純化により、これらのパラメーターに対する通常の調整が不要になり、また、これらのパラメーターの構成に必要な時間と労力が削減されます。また、新規値を有効にするために DB2 インスタンスをシャットダウンして再始動する必要もなくなりました。

エージェントおよびメモリーの動的および自動的な構成を可能にするには、インスタンスがアクティブにされるときに、メモリー・リソースを多少増やす必要があります。

エージェント、プロセス・モデル、およびメモリー構成

DB2 データ・サーバーは 32 ビットおよび 64 ビット・プラットフォーム上でマルチスレッド・アーキテクチャーを活用して、ユーザビリティーの向上、リソースの共用の改善、メモリー・フットプリントの削減、およびすべてのオペレーティング・システムで一貫したスレッド化アーキテクチャーなど、多数の利点を提供します。

複数パーティション間でのデータベースの構成

データベース・マネージャーでは、複数のパーティション間のすべてのデータベース構成エレメントの単一ビューを提供します。これは、各データベース・パーティションに対して db2_all コマンドを呼び出さなくても、すべてのデータベース・パーティション間のデータベース構成を更新またはリセットできることを意味します。

データベースが存在するパーティションから 1 つの SQL ステートメントを発行するか、1 つの管理コマンドを発行するだけで、複数パーティションのデータベース構成を更新できます。データベース構成を更新またはリセットする方法は、デフォルトではすべてのデータベース・パーティションに対して です。

コマンド・スクリプトおよびアプリケーションの後方互換性については、次の 3 つのオプションがあります。

- db2set コマンドを使用して、次のように **DB2_UPDDBCFG_SINGLE_DBPARTITION** レジストリー変数を TRUE に設定します。

  ```
  DB2_UPDDBCFG_SINGLE_DBPARTITION=TRUE
  ```

  注: レジストリー変数の設定は、ADMIN_CMD プロシージャーを使用して行われる UPDATE DATABASE CONFIGURATION または RESET DATABASE CONFIGURATION 要求には適用されません。

- UPDATE DATABASE CONFIGURATION または RESET DATABASE CONFIGURATION コマンド、あるいは ADMIN_CMD プロシージャーのいずれかで **DBPARTITIONNUM** パラメーターを使用します。例えば、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構成を更新するには、次のように ADMIN_CMD プロシージャーを呼び出します。

  ```
  CALL SYSPROC.ADMIN_CMD
    ('UPDATE DB CFG USING sortheap 1000')
  ```

  単一のデータベース・パーティションを更新するには、次のように ADMIN_CMD プロシージャーを呼び出します。

  ```
  CALL SYSPROC.ADMIN_CMD
    ('UPDATE DB CFG DBPARTITIONNUM 10 USING sortheap 1000')
  ```

- db2CfgSet API で **DBPARTITIONNUM** パラメーターを使用します。 **db2Cfg** 構造内の各フラグによって、データベース構成の値を単一のデータベース・パーティションに適用するかどうかが指示されます。フラグを設定する場合、**DBPARTITIONNUM** 値も指定する必要があり、例えば次のようにします。

  ```
  #define db2CfgSingleDbpartition          256
  ```

  データベース・マネージャーまたはデータベース構成パラメーターを設定する db2CfgSet API に対して、db2CfgSingleDbpartition 値を設定していない場合、**DB2_UPDDBCFG_SINGLE_DBPARTITION** レジストリー変数を TRUE に設定していないか、または *versionNumber* をバージョン 9.5 のバージョン番号より小さいものに設定していなければ、そのデータベース構成の値はすべてのデータベース・パーティションに適用されます。

データベースをバージョン 9.5 にマイグレーションする場合、データベース構成ファイルをマイグレーションする必要はありません。なぜなら、すべてのデータベー

ス構成パラメーターは同じ値を維持するからです。ただし、それ以降に出されるマイグレーション済みのデータベースに対するデータベース構成の更新またはリセット要求では、バージョン 9.5 の構成の更新またはリセット要求の方法が使用されます。

既存の更新またはリセットのコマンド・スクリプトの場合、上で説明したのと同じ規則が適用されます。つまり、バージョン 9.5 より前の方法を使用するか、UPDATE DATABASE CONFIGURATION または RESET DATABASE CONFIGURATION コマンドの **DBPARTITIONNUM** オプションを組み込むようにスクリプトを変更するか、あるいは **DB2_UPDDBCFG_SINGLE_DBPARTITION** レジストリー変数を設定することができます。

db2CfgSet API を呼び出す既存のアプリケーションの場合、バージョン 9.5 の方法を使用する必要があります。バージョン 9.5 より前の方法を使用する場合、**DB2_UPDDBCFG_SINGLE_DBPARTITION** レジストリー変数を設定できます。あるいは、新規の db2CfgSingleDbpartition フラグ、および特定のデータベース・パーティションのデータベース構成を更新またはリセットするための新規の **dbpartitionnum** フィールドを組み込み、バージョン 9.5 のバージョン番号を指定してこの API を呼び出すようにアプリケーションを変更することもできます。

注: データベース構成値に整合性がないことがわかる場合、それぞれのデータベース・パーティションを個別に更新またはリセットできます。

共有ファイル・ハンドル表

スレッド化されたデータベース・マネージャーにより、同じファイルに対して実行された入出力要求でファイルの再オープンとクローズが不要になるように、各データベースとその各データベースで動いているすべてのエージェントに対して、1 つの共有ファイル・ハンドル表が維持されます。

バージョン 9.5 よりも前のバージョンでは、ファイル・ハンドル表は、各 DB2 エージェントごとに分かれて保守されており、エージェントのファイル・ハンドル表の各サイズは、**maxfilop** 構成パラメーターにより制御されていました。バージョン 9.5 以降、データベース・マネージャーはデータベース全体で 1 つの共有ファイル・ハンドル表を保守するので、同じファイル・ハンドルが、同じデータベース・ファイルを処理するすべてのエージェント間で共有されます。結果的に、**maxfilop** 構成パラメーターは、この共有ファイル・ハンドル表のサイズを制御するために使用されます。

この変更により、**maxfilop** 構成パラメーターのデフォルト値、および最大値と最小値も新しくなりました。データベース・マイグレーション中に、**maxfilop** 構成パラメーターは、この新しいデフォルト値に自動的に設定されます。

fenced モード・プロセスでのベンダー・ライブラリー関数の実行

データベース・マネージャーは、データ圧縮、TSM のバックアップ、およびログ・データのアーカイブなどのタスクを実行する、fenced モード・プロセスのベンダー・ライブラリー関数をサポートします。

バージョン 9.5 より前は、ベンダー・ライブラリー関数、ベンダー・ユーティリティー、またはルーチンはエージェント・プロセス内で実行されていました。バージョン 9.5 以降は、DB2 データベース・マネージャー自体がマルチスレッド・アプリ

ケーションであるため、スレッド・セーフでなくなったベンダー・ライブラリー関数は、メモリーまたはスタックの破壊や、さらには DB2 データベース内のデータの破壊の原因になることがあります。これらの理由から、ベンダー・ユーティリティーを呼び出すごとに、新規の fenced モード・プロセスが作成されて、ベンダー・ライブラリー関数またはルーチンはその fenced モード・プロセス内で実行されます。これによって深刻なパフォーマンスの低下が生じることはありません。

注: fenced モード・フィーチャーは、Windows プラットフォームでは使用不可です。

自動ストレージ

自動ストレージにより、表スペースのストレージ管理が単純化されます。データベースを作成する際に、データベース・マネージャーが表スペース・データを配置するストレージ・パスを指定してください。これで、表スペースを作成してデータを取り込む際に、データベース・マネージャーにより表スペース用のコンテナーとスペース割り振りが管理されます。

自動ストレージの表スペース

自動ストレージが使用可能になっていないデータベースに表スペースを作成する際、MANAGED BY SYSTEM 節または MANAGED BY DATABASE 節を指定する必要があります。 . これらの節を使用すると、システム管理スペース (SMS) 表スペースまたはデータベース管理スペース (DMS) 表スペースが節に応じて作成されます。どちらの場合にもコンテナーの明示的なリストを提供する必要があります。

データベースで自動ストレージが使用可能になっている場合は、他の選択肢があります。つまり、MANAGED BY AUTOMATIC STORAGE 節を指定するか、または MANAGED BY 節を除外する (これは自動ストレージの使用を暗黙指定します) ことができます。 . データベース・マネージャーはコンテナーを自動的に割り当てるため、この場合コンテナー定義を指定する必要はありません。

以下は、自動ストレージの表スペースを作成するいくつかのステートメントの例です。

```
CREATE TABLESPACE TS1
CREATE TABLESPACE TS2 MANAGED BY AUTOMATIC STORAGE
CREATE TEMPORARY TABLESPACE TEMPTS
CREATE USER TEMPORARY TABLESPACE USRTMP MANAGED BY AUTOMATIC STORAGE
CREATE LONG TABLESPACE LONGTS
```

自動ストレージの表スペース・タイプは異なる表スペース・タイプに見えますが、これは実際既存の SMS タイプおよび DMS タイプを拡張したものです。表スペースを REGULAR または LARGE 表スペースで作成した場合、これはファイル・コンテナーを持つ DMS 表スペースとして作成されます。表スペースを USER または SYSTEM TEMPORARY 表スペースで作成した場合、これはディレクトリー・コンテナーを持つ SMS 表スペースとして作成されます。

注: この動作は将来のデータベース・マネージャーのバージョンでは変更される可能性があります。

これらのコンテナーに関連した名前の形式は次のとおりです。

storage path/*instance name*/NODE####/*database name*/T#######/C#######.*EXT*

説明:

storage path
データベースに関連付けられたストレージ・パス

instance name
データベースが作成されたインスタンス

database name
データベースの名前

NODE####
データベース・パーティション番号 (例えば、NODE0000)

T#######
表スペース ID (例えば T0000003)

C#######
コンテナー ID (例えば C0000012)

EXT 保管されているデータのタイプに基づく以下の拡張子

CAT システム・カタログの表スペース

TMP SYSTEM TEMPORARY 表スペース

UTM USER TEMPORARY 表スペース

USR ユーザーまたは REGULAR 表スペース

LRG LARGE 表スペース

自動ストレージの REGULAR 表スペースと LARGE 表スペース、および DMS 表スペースとの間の違い

自動ストレージの REGULAR 表スペースと LARGE 表スペースは、DMS 表スペースとして作成され、DMS 表スペースに関連付けられたすべての規則と動作が適用されます。ただし、以下の表で示すように、ストレージの管理方法に関して違いがあります。

表 3. 非自動ストレージ表スペースの管理と自動ストレージ表スペースの管理との間の違い

非自動ストレージ	自動ストレージ
表スペースの作成時に、コンテナーのリストを明示的に指定する必要がある。	表スペースの作成時にコンテナーのリストを指定できないが、その代わりに、データベース・マネージャーによりコンテナーが自動的に割り当てられて、割り振られる。
デフォルトで表スペースの自動サイズ変更がオフになる (AUTORESIZE が NO に設定される)。	デフォルトで表スペースの自動サイズ変更がオンになる (AUTORESIZE が YES に設定される)。
表スペースに初期サイズを指定するために、INITIALSIZE 節を使用できない。	表スペースに初期サイズを指定するために、INITIALSIZE 節を使用できる。
ALTER TABLESPACE ステートメントを使用 (ADD、DROP、BEGIN NEW STRIPE SET などを指定して) してコンテナー操作を実行できる。	データベース・マネージャーでスペースが管理されるため、コンテナー操作を実行できない。

表 3. 非自動ストレージ表スペースの管理と自動ストレージ表スペースの管理との間の違い (続き)

非自動ストレージ	自動ストレージ
リダイレクトされたリストア操作を使用して表スペースに関連付けられたコンテナーを再定義できる。	データベース・マネージャーでスペースが管理されるため、リダイレクトされたリストア操作を使用して表スペースに関連付けられたコンテナーを再定義することはできない。

上の表内で説明したように、自動ストレージの REGULAR 表スペースまたは LARGE 表スペースを作成する際に、次の例のように CREATE TABLESPACE ステートメントの一部として初期サイズを指定できます。

```
CREATE TABLESPACE TS1 INITIALSIZE 100 M
```

初期サイズを指定しない場合は、データベース・マネージャーにより、デフォルト値の 32 MB が使用されます。

特定のサイズで表スペースを作成するために、データベース・マネージャーによりストレージ・パス内にファイル・コンテナーが作成されます。パスの間でスペースの分配が均等でないと、同じサイズのコンテナーを作成できません。そのため、すべてのストレージ・パスのフリー・スペースを同じにすることが重要です。

表スペースに対して自動サイズ変更を使用可能にした場合は、その中のスペースが使用されるにつれ、データベース・マネージャーにより既存のコンテナーが自動的に拡張され、(ストライプ・セットの使用により) 新規コンテナーが追加されます。コンテナーが拡張されたり追加されたりするとしても、バランスの再調整は行われません。

表スペースの自動サイズ変更

自動サイズ変更の自動ストレージ表スペースが使用可能にされると、データベース・マネージャーはコンテナーの新規ストライプ・セットを追加することによって、フル・ファイル・システムの状態を自動的に処理することができます。

データベース・システムには、システム管理スペース (SMS) およびデータベース管理スペース (DMS) という 2 つの表スペース・タイプが存在することが可能です。SMS 表スペースに関連したコンテナーはファイル・システムのディレクトリーで、これらのディレクトリー内のファイルは、表スペース内のオブジェクトに連動して大きくなります。ファイルは、いずれかのコンテナーのファイル・システムの限界に達するか、またはデータベースの表スペース・サイズの限界に達するまで増大し続けます (を参照してください)。

DMS 表スペースはファイル・コンテナーまたはロー・デバイス・コンテナーで構成され、そのサイズはコンテナーが表スペースに割り当てられるときに設定されます。コンテナー内のスペースがすべて使用されている場合、表スペースはいっぱいであるとみなされます。ただし、SMS 表スペースに対しての場合とは異なり、ALTER TABLESPACE ステートメントを使用して表スペースにさらに多くのストレージ・スペースを与えることによって、コンテナーを追加または拡張することができます。また、DMS 表スペースには、*自動サイズ変更* というフィーチャーもあり、自動的にサイズ変更可能な DMS 表スペースのスペースが消費されるにつれ、データベース・システムは 1 つ以上のファイル・コンテナーによって表スペースを

拡張する場合があります。 SMS 表スペースにも自動的な増大の類似した機能がありますが、「自動サイズ変更」という用語は DMS に対してのみ使用されます。

表スペースの自動サイズ変更には次の影響があります。

- 自動サイズ変更が使用可能な表スペースには、その表スペースに関連付けられた、バージョン 8.2.1 またはそれ以前のリリースでは認識されないメタデータがあります。それらのバージョンで自動サイズ変更が使用可能な表スペースを持つデータベースの使用を試みると、障害が発生します (多くの場合 SQL0980C または SQL0902C エラーが返されます)。データベースに接続しようとしたり、データベースをリストアしようとしたりすると、エラーが送信される場合があります。自動サイズ変更の表スペースを使用可能にしている場合、それらの表スペースで「自動サイズ変更」機能を使用不可にするとメタデータが除去され、データベースをバージョン 8.2.1 以前のリリースで使用できるようになります。
- 自動サイズ変更フィーチャーを使用不可にすると、このフィーチャーを後で使用可能にした場合に、INCREASESIZE および MAXSIZE に関連付けられた値が失われます。
- このフィーチャーは、ロー・デバイス・コンテナーを使用する表スペースには使用可能にできず、また自動的にサイズ変更できる表スペースにロー・デバイス・コンテナーを追加できません。これらの操作を試みると、エラー (SQL0109N) が発生します。ロー・デバイス・コンテナーを追加する必要がある場合、まずフィーチャーを使用不可にする必要があります。
- リダイレクトされたリストア操作で、コンテナー定義をロー・デバイス・コンテナーが組み込まれるように変更することはできません。この種の操作を試みると、エラー (SQL0109N) が発生します。
- 最大サイズによってデータベース・マネージャーによる表スペースの自動増加に限界が設けられるため、ユーザーによる表スペースの増加にも限界が設けられます。つまり、表スペースにスペースを追加する操作を実行する際、操作実行後のサイズは最大サイズ以下でなければなりません。スペースの追加は、ALTER TABLESPACE ステートメントの ADD、EXTEND、RESIZE、または BEGIN NEW STRIPE SET 節の使用によって行えます。

自動サイズ変更フィーチャーの使用可能化および使用不可化

デフォルトでは、自動サイズ変更フィーチャーは DMS 表スペースで使用可能になっていません。以下のステートメントでは、自動サイズ変更が使用可能ではない DMS 表スペースが作成されます。

```
CREATE TABLESPACE DMS1 MANAGED BY DATABASE
  USING (FILE '/db2files/DMS1' 10 M)
```

自動サイズ変更フィーチャーを使用可能にするには、CREATE TABLESPACE ステートメントに AUTORESIZE YES 節を指定します。

```
CREATE TABLESPACE DMS1 MANAGED BY DATABASE
  USING (FILE '/db2files/DMS1' 10 M) AUTORESIZE YES
```

DMS 表スペースを作成した後に、AUTORESIZE 節を指定した ALTER TABLESPACE ステートメントを使用することによっても、自動サイズ変更フィーチャーを使用可能または使用不可にできます。

```
ALTER TABLESPACE DMS1 AUTORESIZE YES
ALTER TABLESPACE DMS1 AUTORESIZE NO
```

以下の MAXSIZE と INCREASESIZE という 2 つの他の属性も自動サイズ変更表スペースに関連しています。

最大サイズ (MAXSIZE)

CREATE TABLESPACE ステートメントの MAXSIZE 節は、表スペースの最大サイズを定義します。例えば、以下のステートメントは、(データベースに複数のデータベース・パーティションがある場合、1 つのデータベース・パーティションにつき) 100 メガバイトまで増やせる表スペースを作成します。

```
CREATE TABLESPACE DMS1 MANAGED BY DATABASE
  USING (FILE '/db2files/DMS1' 10 M)
       AUTORESIZE YES MAXSIZE 100 M
```

MAXSIZE NONE 節は、表スペースに最大限界がないことを指定します。表スペースの増大は、ファイル・システムの限界か、または表スペースの限界に達するまで続きます。SQL および XML の制限値を参照してください。 MAXSIZE 節を指定しないと、自動サイズ変更フィーチャーが使用可能になっている場合は、最大限界はありません。

以下の各例に示すように、ALTER TABLESPACE ステートメントを使用して、すでに自動サイズ変更が使用可能になっている表スペースの MAXSIZE の値を変更します。

```
ALTER TABLESPACE DMS1 MAXSIZE 1 G
ALTER TABLESPACE DMS1 MAXSIZE NONE
```

最大サイズを指定した場合、データベース・マネージャーはコンテナーの増加の整合性を保とうとするため、データベース・マネージャーが施行する実際の値は指定された値よりも若干小さくなる可能性があります。

増加サイズ (INCREASESIZE)

CREATE TABLESPACE ステートメントの INCREASESIZE 節は、表スペース内にフリー・エクステントがないが、1 つ以上のエクステントが要求された場合に表スペースを増やすために使用されるスペースの量を定義します。以下の各例に示すように、値は明示的なサイズまたはパーセントで指定できます。

```
CREATE TABLESPACE DMS1 MANAGED BY DATABASE
  USING (FILE '/db2files/DMS1' 10 M)
       AUTORESIZE YES INCREASESIZE 5 M

CREATE TABLESPACE DMS1 MANAGED BY DATABASE
  USING (FILE '/db2files/DMS1' 10 M)
       AUTORESIZE YES INCREASESIZE 50 PERCENT
```

パーセントの値は、表スペースを増やす必要が生じるたびに INCREASESIZE の値で指定された増加量が計算され、その時点の表スペース・サイズのパーセントに基づいて増加することを意味します。例えば、表スペースのサイズが 20 MB で INCREASESIZE の値が 50 % の場合、表スペースは最初に 10 MB 増加して (サイズが 30 MB になり)、次回は 15 MB 増加します。

自動サイズ変更フィーチャーを使用可能にした際に INCREASESIZE 節を指定しなかった場合は、データベース・マネージャーにより適切な使用値が決定されます

が、これは表スペースの存続期間中に変更される場合があります。AUTORESIZE や MAXSIZE と同様、ALTER TABLESPACE ステートメントを使用して INCREASESIZE の値を変更できます。

サイズの増加を指定した場合、データベース・マネージャーによって使用される実際の値が、指定した値と若干異なる場合があります。この使用値の調整は、表スペース内のコンテナー全体で増加の整合性を保つために行われます。

表スペースを拡張する方法

自動的にサイズを変更できる表スペースで、既存のスペースがすべて使用され、さらに多くのスペースが要求される場合、データベース・マネージャーは表スペースのサイズの増加を試みます。データベース・マネージャーは、バランスの再調整が発生しないように、表スペース内の拡張が可能なコンテナーを判別します。データベース・マネージャーは、表スペース・マップ (マップは表スペースのストレージ・レイアウトを記述する) の最新の範囲に存在するコンテナーのみを拡張し、またそれらのコンテナーをすべて同じ量で拡張します。

例えば、以下のステートメントを検討してみましょう。

```
CREATE TABLESPACE TS1 MANAGED BY DATABASE
  USING (FILE 'C:¥TS1CONT' 1000, FILE 'D:¥TS1CONT' 1000,
         FILE 'E:¥TS1CONT' 2000, FILE 'F:¥TS1CONT' 2000)
  EXTENTSIZE 4
  AUTORESIZE YES
```

データベース・マネージャーがメタデータのために各コンテナーの小さな部分 (1 つのエクステント) を使用することに注意して、下に示す CREATE TABLESPACE ステートメントに基づいて表スペースに対して作成された表スペース・マップを参照してください。(表スペース・マップは、表スペースのスナップショットからの出力の一部です)。

```
Table space map:

 Range  Stripe Stripe  Max         Max  Start  End    Adj.   Containers
 Number Set    Offset  Extent      Page Stripe Stripe
 [   0] [   0]      0     995      3983      0    248   0    4 (0,1,2,3)
 [   1] [   0]      0    1495      5983    249    498   0    2 (2,3)
```

表スペース・マップは、ID 2 または ID 3 を持つコンテナー (E:¥TS1CONT および F:¥TS1CONT) のみがマップの最新の範囲であることを示しています。そのため、データベース・マネージャーがこの表スペースのコンテナーを自動的に拡張するときには、これら 2 つのコンテナーのみが拡張されます。

注: コンテナーのサイズがすべて同じである表スペースを作成する場合、マップの範囲は 1 つだけになります。そのような場合、データベース・マネージャーはそれぞれのコンテナーを拡張します。拡張をコンテナーのサブセットのみに制限することを避けるには、同じサイズのコンテナーを持つ表スペースを作成してください。

前に説明したように、表スペースの最大サイズに対して限界を指定することもできますし、NONE の値を指定することもできます。後者の場合、増加の限界は設けられません。 NONE つまり限界なしを指定した場合は、ファイル・システムの限界または表スペースの限界によって上限が定義され、データベース・マネージャーでは、この上限を超える表スペースのサイズの増加が試みられません。ただし、その限界に達する前に、コンテナーを増やそうとしてもファイル・システムがいっぱい

であるために失敗することがあります。そうなると、データベース・マネージャーはそれ以上表スペースのサイズを増やさず、アプリケーションに「スペース不足」状態を返します。この状態を解決するには 2 つの方法があります。

- いっぱいになっているファイル・システムで使用可能なスペースの量を増やす。
- 当該のコンテナーが表スペース・マップの最新の範囲から外れるように、表スペースに対してコンテナー操作を実行する。これを行う最も簡単な方法は、新しいコンテナー・セットを持つ表スペースに新規ストライプ・セットを追加することで、またベスト・プラクティスは、コンテナーをすべて同じサイズにすることです。 BEGIN NEW STRIPE SET 節を指定した ALTER TABLESPACE ステートメントを使用して、新規ストライプ・セットを追加することができます。新規ストライプ・セットを追加することによって、表スペース・マップに新しい範囲が追加されます。新しい範囲が追加されると、データベース・マネージャーが自動的に拡張しようとするコンテナーはこの新規ストライプ・セット内に入り、古いコンテナーは変更されません。

  注: ユーザーが開始したコンテナー操作がペンディング状態であるか、または後続のバランスの再調整が進行中の場合、操作がコミットされるかまたはバランスの再調整が完了するまで自動サイズ変更フィーチャーは使用不可になります。

例えば、DMS 表スペースの場合に、表スペースに同じサイズの 3 つのコンテナーがあり、それぞれが独自のファイル・システムに存在するとします。表スペースに対して作業が行われるにつれ、データベース・マネージャーはこれら 3 つのコンテナーを自動的に拡張していきます。最終的に、ファイル・システムのいずれかがいっぱいになり、対応するコンテナーは増加しなくなります。ファイル・システム上でこれ以上使用可能なフリー・スペースを設けられない場合、当該のコンテナーが表スペース・マップの最新の範囲から外れるように、表スペースに対してコンテナー操作を実行する必要があります。その場合、2 つのコンテナーを指定して新規ストライプ・セットを追加するか (スペースがまだある各ファイル・システム上に1つ)、または指定するコンテナーを増やすかまたは減らすこともできます (この場合にも、追加される各コンテナーのサイズが同じになり、使用されているそれぞれのファイル・システム上に増加のための十分な余裕があるようにする)。データベース・マネージャーが表スペースのサイズを増加しようとすると、古いコンテナーの拡張を試みる代わりに、この新規ストライプ・セット内のコンテナーに対して拡張を試みます。

モニター

DMS 表スペースの自動サイズ変更に関する情報は、表スペースのモニターのスナップショット出力の一部として表示されます。次の例に示すように、増加サイズおよび最大サイズの値がこの出力内に含まれます。

```
Auto-resize enabled                     = Yes or No
Current tablespace size (bytes)         = ###
Maximum tablespace size (bytes)         = ### or NONE
Increase size (bytes)                   = ###
Increase size (percent)                 = ###
Time of last successful resize          = DD/MM/YYYY HH:MM:SS.SSSSSS
Last resize attempt failed              = Yes or No
```

自動ストレージ・データベース

データベース・マネージャーにより、すべてのデータベースがデフォルトでは「自動ストレージ」データベースとして作成されます。「自動ストレージ」データベースではないデータベースを作成するには、CREATE DATABASE コマンドを発行するときに AUTOMATIC STORAGE NO を指定します。

自動ストレージが使用可能なデータベースには 1 つ以上のストレージ・パスのセットが関連付けられています。表スペースは、*自動ストレージによる管理* を受けるものとして定義でき、そのコンテナーはストレージ・パスに基づいてデータベース・マネージャーにより割り当ておよび割り振りが行われます。

自動ストレージ用にデータベースを有効にできるのは、その作成時のみです。 同じように、本来自動ストレージを使用するように設計されたデータベースに対して、自動ストレージの無効化を行うことはできません。

デフォルトでは、すべてのデータベースが自動ストレージ・データベースとして作成されます。自動ストレージ・データベースではないデータベースを作成するには、CREATE DATABASE コマンドを発行するときに **AUTOMATIC STORAGE NO** を指定します。

自動ストレージを使用不可にする例を以下に示します。

```
CREATE DATABASE ASNODB1 AUTOMATIC STORAGE NO
CREATE DATABASE ASNODB2 AUTOMATIC STORAGE NO ON X:
```

明示的または暗黙的に使用可能にされる自動ストレージの例:

```
CREATE DATABASE DB1
CREATE DATABASE DB2 AUTOMATIC STORAGE YES ON X:
CREATE DATABASE DB3 ON /data/path1, /data/path2
CREATE DATABASE DB4 ON D:¥StoragePath DBPATH ON C:
```

使用される構文に基づいて、データベース・マネージャーはストレージ・ロケーションに関する以下の 2 つの情報を抽出します。

- データベース・パス (データベース・マネージャーがデータベース用のさまざまな制御ファイルを保管する場所):
  - **DBPATH ON** を指定した場合、これはデータベース・パスを指します。
  - **DBPATH ON** を指定しない場合、**ON** でリストされた最初のパスがデータベース・パス (およびストレージ・パス) を指します。
  - **DBPATH ON** と **ON** のどちらも指定しない場合は、データベース・パスの決定に **dftdbpath** データベース・マネージャー構成パラメーターが使用されます。
- ストレージ・パス (データベース・マネージャーが自動ストレージの表スペース・コンテナーを作成する場所):
  - **ON** を指定した場合は、リストされたすべてのパスがストレージ・パスになります。
  - **ON** を指定しない場合は、ストレージ・パスは **dftdbpath** データベース・マネージャー構成パラメーターの値に設定される 1 つのみになります。

上記の例に関して、使用されるデータベース・パスおよびストレージ・パスを以下の表に要約します。

表 4. 自動ストレージ・データベースとストレージ・パス

CREATE DATABASE コマンド	データベース・パス	ストレージ・パス
CREATE DATABASE DB1 AUTOMATIC STORAGE YES	**dftdbpath** 構成パラメーターの値	**dftdbpath** 構成パラメーターの値
CREATE DATABASE DB2 AUTOMATIC STORAGE YES ON X:	X:	X:
CREATE DATABASE DB3 ON /data/path1, /data/path2	/data/path1	/data/path1、/data/path2
CREATE DATABASE DB4 ON D:¥StoragePath DBPATH ON C:	C:	D:¥StoragePath

提供されるストレージ・パスが存在していなければならず、それらはアクセス可能でなければなりません。パーティション・データベース環境では、各データベース・パーティションに対して同じストレージ・パスが使用されます。データベース・パーティション式をストレージ・パス名の一部として使用しない場合には、特定のデータベース・パーティションに固有のストレージ・パスのセットを指定できません。データベース・パーティション式をストレージ・パス名の一部として使用すると、ストレージ・パスにデータベース・パーティション番号が反映され、各データベース・パーティションごとに異なるパス名が付けられます。

データベース・パーティション式を指示するには、引数 $N ($N の前に 1 つの半角ブランクあり) を使用します。データベース・パーティション式はストレージ・パス内のどこにでも使用でき、複数指定することも可能です。データベース・パーティション式はスペース文字で終了します。スペースの後に続く文字はすべて、データベース・パーティション式が評価された後、ストレージ・パスに付加されます。ストレージ・パス内でデータベース・パーティション式の後にスペース文字がない場合、ストリングの残りは式の一部であると見なされます。下の表に、$N 引数の有効な形式のみをリストします。演算子は左から右に向かって評価され、% はモジュラス演算子を表します。例の中のデータベース・パーティション番号は 10 です。

表 5. データベース・パーティション式

構文	例	値
[blank]$N	" $N"	10
[blank]$N+[number]	" $N+100"	110
[blank]$N%[number]	" $N%5"	0
[blank]$N+[number]%[number]	" $N+1%5"	1
[blank]$N%[number]+[number]	" $N%4+2"	4

データベース・パーティション式の使用例を以下に示します。

```
CREATE DATABASE TESTDB ON "/path1ForNode $N",
  "/path2ForNode $N" DBPATH ON "/dbpathForNodes"
```

パスの間にデータベース・パーティション式を埋め込んだ例を示します。

```
CREATE DATABASE TESTDB ON "/path1ForNode $N",
  "/path2ForNode $N suffix" DBPATH ON "/dbpathForNodes"
```

注: データベース・パーティション式は、**DBPATH ON**で明示的に指定しているか、最初のストレージ・パスにデータベース・パーティション式を使用して暗黙的に指定しているかに関係なく、データベース・パス内では無効です。

特定のデータベース・パーティションのストレージ・パスのフリー・スペースが計算されると、データベース・マネージャーは、ストレージ・パスの中に以下のディレクトリーまたはマウント・ポイントがないか調べ、見つかった最初のものを使用します。

```
storage path/instance name/NODE####/database name
storage path/instance name/NODE####
storage path/instance name
storage path
```

説明:

storage path
データベースに関連付けられたストレージ・パス

instance name
データベースが存在するインスタンス

NODE####
データベース・パーティション番号 (例えば、NODE0000 または NODE0001)

database name
データベースの名前

ファイル・システムをストレージ・パスの下の位置にマウントすることができます。データベース・マネージャーは、表スペース・コンテナーで使用可能なフリー・スペースの実際の量がストレージ・パスのディレクトリー自体に関連付けられている量と同じでない可能性があることを認識します。

1 つの物理コンピューターに 2 つの論理データベース・パーティションが存在し、1 つのストレージ・パス /db2data がある例について考えてみましょう。データベース・パーティションはそれぞれこのストレージ・パスを使用できますが、各パーティションごとに別のファイル・システムを作成して、データを各パーティションから分離することもできます。ファイル・システムは、/db2data/*instance*/NODE#### にマウントされます。ストレージ・パス上にコンテナーを作成してフリー・スペースを決定する際、データベース・マネージャーは、/db2data のフリー・スペース情報を取得しませんが、代わりに対応する /db2data/*instance*/NODE#### ディレクトリーのフリー・スペース情報を取得します。

データベースを作成するときには必ず 3 つのデフォルト表スペースが作成されます。 CREATE DATABASE コマンドの一部として明示的な表スペース定義を指定しない場合は、表スペースは自動ストレージの表スペースとして作成されます。

データベースを作成した後、次の例に示すように ALTER DATABASE ステートメントの ADD STORAGE 節を使用してデータベースに新規ストレージ・パスを追加できます。

```
ALTER DATABASE ADD STORAGE ON '/data/path3', '/data/path4'
```

自動ストレージを使用できるデータベースへの自動ストレージ・パスの追加

ALTER DATABASE を使用して、自動ストレージが有効になっているデータベースに自動ストレージ・パスを追加することができます。自動ストレージ用にデータベースを有効にできるのは、その作成時のみです。

複数パーティション・データベース環境用のストレージ・パスを追加する場合、各データベース・パーティションでそのストレージ・パスが存在する必要があります。指定されたパスがすべてのデータベース・パーティションに存在していない場合、ステートメントはロールバックされます。

既存のデータベースにストレージ・パスを追加するには、次のような ALTER DATABASE ステートメントを発行します。

```
ALTER DATABASE PATH pathname
```

自動ストレージの制約事項

自動ストレージを使用してデータベースを作成するかどうかを決定する際、いくつかの制約事項を考慮する必要があります。

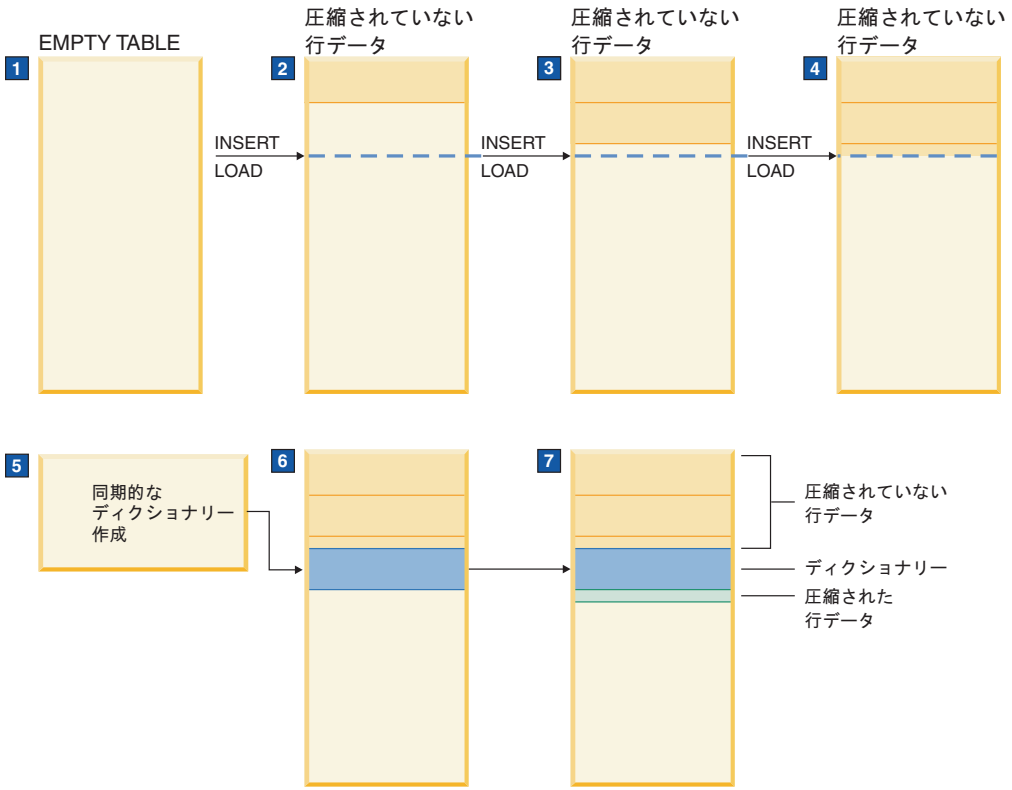
- データベースの作成後にデータベースで自動ストレージを使用不可または使用可能にすることはできません。
- ストレージ・パスは絶対パス名でなければなりません。 Windows オペレーティング・システムでは、ストレージ・パスにパスまたはドライブ名を使用できます。データベース・パスはドライブ名である必要があります。 最大パス長は 175 文字です。
- パーティション・データベースでは、各データベース・パーティションで同じストレージ・パスのセットを使用する必要があります (データベース・パーティション式を使用しない場合)。
- データベース・パーティション式は、CREATE DATABASE の **DBPATH ON** オプションを使用して明示的に指定しているか、最初のストレージ・パスにデータベース・パーティション式を使用して暗黙的に指定しているかに関係なく、データベース・パス内では無効です。

(コンプレッション) ディクショナリーの自動作成 (ADC)

コンプレッション・ディクショナリーは、スペースを空けて表により多くのデータを追加できるように、表に移動されたデータを圧縮するために使用されます。コンプレッション・ディクショナリーは、一定の条件を満たした場合、(ロード操作や挿入操作などの) データの設定操作中に自動的に作成され、表に挿入または追加されます。

(コンプレッション) ディクショナリーの自動作成 (ADC) は、表に対して COMPRESS 属性を定義していて、コンプレッション・ディクショナリーがその物理表またはパーティション内にまだ存在せず、またその表に十分なデータが存在している場合に、この表に対して実行されます。続いて、表に移動されたデータがコンプレッション・ディクショナリーを使用して圧縮されます (表の COMPRESS 属性が使用可能のままの場合)。

次の図は、コンプレッション・ディクショナリーが自動的に作成されるためのプロセスを示しています。



1. コンプレッション・ディクショナリーは、表が空のため作成されません。
2. 挿入またはロード操作を使用してデータが表に挿入されますが、非圧縮のままです。
3. さらに多くのデータが表に挿入またはロードされますが、非圧縮のままです。
4. しきい値に達すると、COMPRESS 属性が YES に設定されている場合には、ディクショナリー作成が自動的に起動されます。
5. ディクショナリーが作成されます。
6. ディクショナリーが表に追加されます。
7. これから先は、データは圧縮されます。

以下の表は、リリースごとの、コンプレッション・ディクショナリー作成の相違を示します。

表 6. リリースごとのコンプレッション・ディクショナリー作成の相違

コマンドおよび属性	バージョン 9.1	バージョン 9.5
RESETDICTIONARY オプションを指定した LOAD REPLACE コマンド	適用外。	表の COMPRESS 属性を YES に設定している場合で、1 行以上のデータが表にロードまたは挿入されている場合、既存のコンプレッション・ディクショナリーがあれば削除されて新規のコンプレッション・ディクショナリーが生成されます。
CREATE または ALTER TABLE ステートメントで、COMPRESS 属性が YES に設定されている	ディクショナリーの作成は自動ではありませんでした。表データを圧縮するには、表の再編成処理を使用して、コンプレッション・ディクショナリーを明示的に作成する必要がありました。	1 行以上のデータが表にロードまたは挿入されている場合、表の COMPRESS 属性を YES に設定すると、表が ADC の対象となります。
INSERT、LOAD INSERT、IMPORT INSERT、または REDISTRIBUTE コマンド	適用外。	表の COMPRESS 属性を YES に設定している場合で、表に十分なデータが入っている (つまり、しきい値を超えた) 場合、まだコンプレッション・ディクショナリーが作成されていない表には ADC が実行されます。 **注:** REDISTRIBUTE コマンドにより起動される ADC は、新規に追加されたデータベース・パーティションに限定されます。
KEEPDICTIONARY オプションを指定した REORG TABLE コマンド	表の COMPRESS 属性を YES に設定していて、コンプレッション・ディクショナリーがまだ表に存在していなかった場合、表に含まれるデータのボリュームに関係なく、コンプレッション・ディクショナリーの作成、表への挿入または追加が試みられていました。	表のサイズが ADC の表サイズしきい値と同じであり、しきい値を超えたときに表に十分なボリュームのデータが存在する場合のみ、表にディクショナリーが挿入されます。

データ行圧縮

データ行圧縮の目的は、ディスク・ストレージ・スペースの節約を実現することです。これにより、ディスク入出力も節約されます。また、さらに多くのデータをバッファー・プールにキャッシュできるので、バッファー・プールのヒット率も向上します。データ行圧縮は、静的なディクショナリー・ベースの圧縮アルゴリズムを使用して、行ごとにデータを圧縮します。

行レベルでのデータ圧縮では、1 つの行の複数の列値に渡る反復パターンを、より短いシンボル・ストリングで置き換えることが可能です。

注: データの圧縮および圧縮解除に必要な余分の CPU サイクルという形で、関連コストが生じます。データ行圧縮のストレージ節約およびパフォーマンスへの影響は、データベース内のデータの特性、データベースのレイアウトと調整、およびアプリケーション・ワークロードと関係しています。データ・ページ上のデータまたはログ・レコード内のデータのみが圧縮されます。

表データを圧縮するには、コンプレッション・ディクショナリーが表に対して存在している必要があり、CREATE TABLE または ALTER TABLE ステートメントの COMPRESS 属性を YES に設定する必要があり、また、十分なデータが表内に存在している必要もあります。これらの圧縮条件が表に存在している場合、INSERT ステートメントまたは LOAD INSERT、IMPORT INSERT、または REDISTRIBUTE コマンドを発行すると、表に追加されたデータが圧縮されます。

バージョン 9.5 では、表の COMPRESS 属性が YES に設定され、データ・コンプレッション・ディクショナリーが作成されると、データ行圧縮は自動的に使用可能になります。COMPRESS 属性を YES に設定して表を作成または変更している場合は、ユーザー側での手動操作もデータベース要求も必要ありません。つまり、データ・コンプレッション・ディクショナリーを作成するために、明示的な従来の (オフライン) 表再編成を実行する必要はありません。

注: COMPRESS 属性を YES に設定し、コンプレッション・ディクショナリーが存在している場合は、圧縮がインポートまたはロード操作による挿入を含め、行を挿入する操作すべてに適用されます。圧縮は表全体に対して使用可能にされますが、各行は個別に圧縮されます。そのため、圧縮された行と圧縮されていない行の両方を 1 つの表に同時に含めることも可能です。

コンプレッション・ディクショナリーを明示的に作成する (およびそれに続けて表を圧縮する) には、従来の (オフライン) 表再編成を実行します。表内に存在しているすべてのデータ行は、コンプレッション・ディクショナリーの作成に参加します。このコンプレッション・ディクショナリーは、表のデータ・オブジェクト部分に、表データ行と共に保管されます。

表を圧縮解除するには、表の COMPRESS 属性を NO に設定してから、従来の (オフライン) 表再編成を実行します。

制約事項

- データ行圧縮は、索引、long、LOB、および XML データ・オブジェクトには適用できません。
- 行圧縮には、表データのレプリケーション・サポートとの互換性はありません。
- RUNSTATS コマンドを使用して、行圧縮統計を生成することができます。この統計は、システム・カタログ表の SYSCAT.TABLES に保管されます。表に対する行の圧縮効果を見積もる圧縮見積もりオプションが、INSPECT ユーティリティーで使用できます。照会オプティマイザーのコスト計算モデルには、圧縮解除のコストが組み込まれています。
- 更新アクティビティーおよびデータ行内の更新変更の位置づけに応じて、ログ・スペースの消費量が増える場合があります。

- 行のサイズが大きくなる場合、新しいバージョンの行は現在のデータ・ページに収まらない可能性があります。この場合、行の新しいイメージは、オーバーフロー・ページに格納されます。この種のポインター・オーバーフロー・レコードの作成を最小限にするには、データ・ページにさらにフリー・スペースを追加することができます。例えば、圧縮せずに 5% のフリー・スペースを使用していた場合は、圧縮した 10% のフリー・スペースを割り振ってください。この推奨事項は、頻繁に更新されるデータの場合に特に重要です。

構成アドバイザー

構成アドバイザーを使用して、バッファー・プール・サイズ、データベース構成パラメーター、およびデータベース・マネージャー構成パラメーターの初期値の推奨値を取得することができます。

構成アドバイザーを使用するには、既存のデータベースに AUTOCONFIGURE コマンドを指定するか、または AUTOCONFIGURE を CREATE DATABASE コマンドのオプションとして指定します。データベースを構成するには、SYSADM、SYSCTRL、または SYSMAINT 権限が必要です。

推奨値は表示することができ、また CREATE DATABASE コマンドの APPLY オプションを使用して適用することもできます。推奨値は、ユーザーが提供する入力と、アドバイザーが収集するシステム情報に基づいています。

構成アドバイザーにより推奨される値は、インスタンスごとに 1 つのデータベースのみに関係します。このアドバイザーを複数のデータベースで使用する場合、各データベースは異なるインスタンスに属していなければなりません。

構成アドバイザーを使用して構成パラメーターを調整する

構成アドバイザーは、どの構成パラメーターを修正したらよいかを提案し、それらの値を提案することによって、インスタンスごとの単一データベースのパフォーマンスのチューニングとメモリー所要量のバランスをとるよう支援します。データベースを作成すると、構成アドバイザーが自動的に実行します。

このフィーチャーを使用不可にしたり、明示的に使用可能にするには、データベースを作成する前に次のようにして db2set コマンドを使用します。

```
db2set DB2_ENABLE_AUTOCONFIG_DEFAULT=NO
db2set DB2_ENABLE_AUTOCONFIG_DEFAULT=YES
```

いくつかの構成パラメーターに値を定義したり、これらのパラメーターのアプリケーションの適用範囲を決定するには、以下のオプションの 1 つを指定した AUTOCONFIGURE コマンドを使用します。

- NONE。適用される値がないことを意味します。
- DB ONLY。データベース構成値およびバッファー・プール値のみが適用されることを意味します。
- DB AND DBM。すべてのパラメーターとその値が適用されることを意味します。

注: CREATE DATABASE コマンドの実行時に構成アドバイザーが自動的に有効になるようにしていた場合でも、AUTOCONFIGURE コマンド・オプションを指定す

ることができます。 CREATE DATABASE コマンドの実行時に構成アドバイザーが有効になるようにしていなかった場合でも、構成アドバイザーを後から手動で実行することができます。

データベース構成推奨値の生成

データベースを作成すると、構成アドバイザーが自動的に実行します。構成アドバイザーを実行するには、コマンド行プロセッサー (CLP) で AUTOCONFIGURE コマンドを指定するか、または db2AutoConfig API を呼び出すこともできます。

CLP を使用して構成推奨値を要求するには、以下のコマンドを入力します。

```
AUTOCONFIGURE
     USING input_keyword param_value
     APPLY value
```

以下の AUTOCONFIGURE コマンドは、データベースの使用方法に関する入力に基づき、構成推奨値をリクエストしますが適用されない例です。

```
DB2 AUTOCONFIGURE USING
     MEM_PERCENT 60
     WORKLOAD_TYPE MIXED
     NUM_STMTS 500
     ADMIN_PRIORITY BOTH
     IS_POPULATED YES
     NUM_LOCAL_APPS 0
     NUM_REMOTE_APPS 20
     ISOLATION RR
     BP_RESIZEABLE YES
APPLY NONE
```

例: 構成アドバイザーを使用した構成推奨値の要求

このシナリオでは、推奨値を生成するための構成アドバイザーのコマンド行からの実行、および構成アドバイザーによって生成される出力を示します。

構成アドバイザーを実行するには、以下のようにします。

1. コマンド行で次のコマンドを指定して、PERSONL データベースに接続します。

   ```
   DB2 CONNECT TO PERSONL
   ```

2. データベースの使用法を指定して、CLP で AUTOCONFIGURE コマンドを実行します。以下の例で示すように、NONE という値を **APPLY** オプションに設定して、構成推奨値を表示するが、それらを適用しないことを指示します。

   ```
   DB2 AUTOCONFIGURE USING
        MEM_PERCENT 60
        WORKLOAD_TYPE MIXED
        NUM_STMTS 500
        ADMIN_PRIORITY BOTH
        IS_POPULATED YES
        NUM_LOCAL_APPS 0
        NUM_REMOTE_APPS 20
        ISOLATION RR
        BP_RESIZEABLE YES
   APPLY NONE
   ```

このコマンドのパラメーターの値に確信がない場合は、省略することができます。その場合は、デフォルトが使用されます。前の例で示したとおり、MEM_PERCENT、WORKLOAD_TYPE など最大 10 個のパラメーターを値なしで受け渡すことができます。

AUTOCONFIGURE コマンドによって生成される推奨値は、以下のように表形式で画面に表示されます。

表 7. 構成アドバイザーの出力例: 第 1 部

```
      Former and Applied Values for Database Manager Configuration
Description                                  Parameter   Current Value   Recommended Value
--------------------------------------------------------------------------------------------
Application support layer heap size (4KB)   (ASLHEAPSZ) = 15              15
No. of int. communication buffers(4KB)(FCM_NUM_BUFFERS) = AUTOMATIC       AUTOMATIC
Enable intra-partition parallelism     (INTRA_PARALLEL) = NO              NO
Maximum query degree of parallelism   (MAX_QUERYDEGREE) = ANY             1
Agent pool size                        (NUM_POOLAGENTS) = 100(calculated) 200
Initial number of agents in pool       (NUM_INITAGENTS) = 0               0
Max requester I/O block size (bytes)          (RQRIOBLK) = 32767           32767
Sort heap threshold (4KB)                  (SHEAPTHRES) = 0               0
```

表 8. 構成アドバイザーの出力例 (続き)

```
      Former and Applied Values for Database Configuration
Description                                  Parameter   Current Value   Recommended Value
--------------------------------------------------------------------------------------------
Default application heap (4KB)             (APPLHEAPSZ) = 256             256
Catalog cache size (4KB)              (CATALOGCACHE_SZ) = (MAXAPPLS*4)    260
Changed pages threshold                (CHNGPGS_THRESH) = 60              80
Database heap (4KB)                            (DBHEAP) = 1200            2791
Degree of parallelism                      (DFT_DEGREE) = 1               1
Default tablespace extentsize (pages)   (DFT_EXTENT_SZ) = 32              32
Default prefetch size (pages)         (DFT_PREFETCH_SZ) = AUTOMATIC       AUTOMATIC
Default query optimization class         (DFT_QUERYOPT) = 5               5
Max storage for lock list (4KB)              (LOCKLIST) = 100             AUTOMATIC
Log buffer size (4KB)                        (LOGBUFSZ) = 8               99
Log file size (4KB)                         (LOGFILSIZ) = 1000            1024
Number of primary log files                (LOGPRIMARY) = 3               8
Number of secondary log files               (LOGSECOND) = 2               3
Max number of active applications            (MAXAPPLS) = AUTOMATIC       AUTOMATIC
Percent. of lock lists per application       (MAXLOCKS) = 10              AUTOMATIC
Group commit count                          (MINCOMMIT) = 1               1
Number of asynchronous page cleaners   (NUM_IOCLEANERS) = 1               1
Number of I/O servers                   (NUM_IOSERVERS) = 3               4
Package cache size (4KB)                   (PCKCACHESZ) = (MAXAPPLS*8)    1533
Percent log file reclaimed before soft chckpt (SOFTMAX) = 100             320
Sort list heap (4KB)                         (SORTHEAP) = 256             AUTOMATIC
statement heap (4KB)                      (STMTHEAP) = 4096            4096
Statistics heap size (4KB)                (STAT_HEAP_SZ) = 4384            4384
Utilities heap size (4KB)                 (UTIL_HEAP_SZ) = 5000            113661
Self tuning memory                      (SELF_TUNING_MEM) = ON              ON
    Automatic runstats                   (AUTO_RUNSTATS) = ON              ON
Sort heap thres for shared sorts (4KB) (SHEAPTHRES_SHR) = 5000            AUTOMATIC
```

表 9. 構成アドバイザーの出力例 (続き)

```
      Former and Applied Values for Bufferpool(s)
Description                                  Parameter   Current Value   Recommended Value
--------------------------------------------------------------------------------------------
IBMDEFAULTBP                              Bufferpool size = -2              340985

DB210203I AUTOCONFIGURE completed successfully. Database manager or database
configuration values may have been changed. The instance must be restarted before
any changes come into effect. You may also want to rebind your packages after the
new configuration parameters take effect so that the new values will be used.
```

すべての推奨値に同意する場合は、AUTOCONFIGURE コマンドを再実行するが、その際、APPLY オプションを使用して推奨値を適用することを指定するか、または

UPDATE DATABASE MANAGER CONFIGURATION コマンドおよび UPDATE DATABASE CONFIGURATION コマンドを使用して個々の構成パラメーターを更新します。

ユーティリティー・スロットル

ユーティリティー・スロットルによって各種保守ユーティリティーのパフォーマンス上の影響が調整され、実動期間中に、これらのユーティリティーを同時に実行できるようにします。ユーティリティーのスロットル・モードでの実行を可能にする設定の影響ポリシーがデフォルトで定義済みですが、ユーティリティーの実行時にユーティリティーをスロットル化する場合は、各クリーナーがそのスロットル優先順位を示す設定の影響優先順位を設定する必要があります。

スロットル・システムは、スロットル・ユーティリティーがその影響ポリシーに抵触することがない範囲で最大限頻繁に実行されるようにします。スロットル化できるのは、統計収集、バックアップ操作、リバランス操作、および非同期索引クリーンアップです。

影響ポリシーは、**util_impact_lim** 構成パラメーターを設定して定義します。

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 (索引) 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。

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。

MDC 表の非同期索引クリーンアップ (Asynchronous index cleanup for MDC tables)

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 4 章 インスタンス

インスタンス とは、データベースをカタログし、構成パラメーターを設定するための、論理データベース・マネージャー環境です。必要に応じて、同一の物理サーバー上に複数のインスタンスを作成し、各インスタンスに固有のデータベース・サーバー環境を指定することができます。

注: Linux® および UNIX オペレーティング・システムへの非 root インストールの場合、単一インスタンスが DB2 製品のインストール中に作成されます。追加のインスタンスは作成することができません。

複数インスタンスを使用すると、次のことを行えます。

- 1 つのインスタンスを開発環境用に使用し、別のインスタンスを実稼働環境用に使用する。
- 特定の環境用にインスタンスを調整する。
- 機密情報へのアクセスを制限する。
- それぞれのインスタンスごとに SYSADM、SYSCTRL、および SYSMAINT 権限の割り当てを制御する。
- インスタンスごとにデータベース・マネージャーの構成を最適化する。
- インスタンスの失敗による影響を制限する。インスタンスが失敗した場合、1 つのインスタンスだけが影響を受けます。他のインスタンスは正常に機能し続けます。

複数インスタンスには以下のものが必要です。

- インスタンスごとに、追加のシステム・リソース (仮想メモリーとディスク・スペース)。
- 追加インスタンスを管理するための付加的な管理作業。

インスタンス・ディレクトリーには、データベース・インスタンスに関連するすべての情報が保管されます。インスタンス・ディレクトリーの位置を作成後に変更することはできません。インスタンス・ディレクトリーの内容は、以下のとおりです。

- データベース・マネージャー構成ファイル
- システム・データベース・ディレクトリー
- ノード・ディレクトリー
- ノード構成ファイル (db2nodes.cfg)
- デバッグ情報 (例外ダンプ、レジスター・ダンプ、 DB2 データベース・プロセス用の呼び出しスタックなど) が入った他のファイル。

用語:

ビット幅

仮想メモリーをアドレッシングするのに使用されるビットの数: 32 ビットおよび 64 ビットが最も一般的です。この用語は、インスタンス、アプリケ

ーション・コード、外部ルーチン・コードのビット幅を指すのに使用される場合があります。 32 ビット・アプリケーションは、32 ビット幅のアプリケーションと同じです。

32 ビット DB2 インスタンス
32 ビット共用ライブラリーおよび実行可能プログラムを含むすべての 32 ビット・バイナリーを含む DB2 インスタンス。

64 ビット DB2 インスタンス
64 ビット共用ライブラリーおよび実行可能プログラムを含み、またすべての 32 ビット・クライアント・アプリケーション・ライブラリー (クライアントとサーバーの両方に含まれる)、および 32 ビット外部ルーチン・サポート (サーバー・インスタンス上にのみ含まれる) を含む DB2 インスタンス。

インスタンスの設計

DB2 データベースは、データベース・サーバー上の DB2 インスタンス内に作成されます。同一の物理サーバー上に複数のインスタンスを作成すると、インスタンスごとに固有のデータベース・サーバー環境が提供されます。

例えば、テスト環境と実稼働環境を同一のマシン上で保守したり、アプリケーションごとにインスタンスを作成してから、各インスタンスをそれがサービス提供するアプリケーション専用に微調整できます。あるいは、機密データを保護するために、独自のインスタンス上に給与計算データベースを保管して、(同一サーバー上の) 他のインスタンスの所有者が給与計算データを閲覧できないようにすることができます。

インストール・プロセスにより、デフォルトの DB2 インスタンスが作成されます。これは、DB2INSTANCE 環境変数によって定義されます。これがほとんどの操作に使用されるインスタンスです。ただし、インストール後にインスタンスを作成 (または除去) することができます。

ご使用の環境に合ったインスタンスを判別して設計する場合、それぞれのインスタンスが 1 つ以上のデータベースに対するアクセスを制御することに注意してください。インスタンス内のすべてのデータベースには 1 つの固有名が割り当てられ、独自のシステム・カタログ表のセット (データベース内に作成されるオブジェクトを追跡するために使用) と独自の構成ファイルを持っています。また、それぞれのデータベースにはユーザーがそこに保管されたデータおよびデータベース・オブジェクトと対話する方法を管理する、許可された権限および特権の独自のセットがあります。 67 ページの図 2は、システム、インスタンス、およびデータベース間の階層関係を示しています。

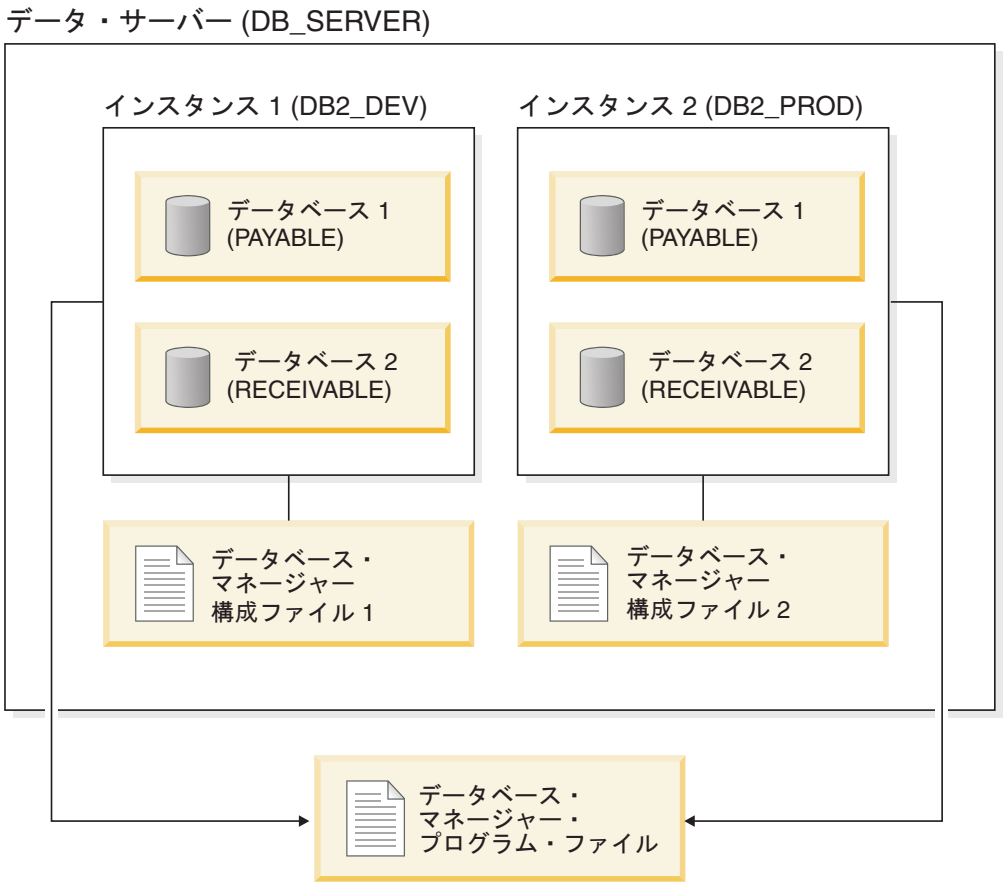


図 2. DB2 システム、インスタンス、およびデータベース間での階層関係

また、*DB2 管理サーバー* (DAS) と呼ばれるもう 1 つの特定のインスタンス・タイプも知っておく必要があります。 DAS は、他の DB2 サーバー上での管理タスクのみ支援するために使用される特殊な DB2 管理コントロール・ポイントです。クライアント構成アシスタントを使用して、リモート・データベースまたは DB2 製品に付属するグラフィック・ツール (例えば、コントロール・センターまたはタスク・センター) を見付ける場合に、DAS が実行している必要があります。複数のインスタンスが存在する場合でも、DB2 データベース・サーバー内には DAS は 1 つしかありません。

インスタンスが作成されたら、使用可能なその他のインスタンス (他のシステム上のインスタンスを含む) にアタッチできます。アタッチしたら、例えば、データベースの作成、データベースからのアプリケーションの強制終了、データベース・アクティビティーのモニター、または特定のインスタンスに関連付けられたデータベース・マネージャー構成ファイルの内容の変更などの、インスタンス・レベルでしか行うことができない保守およびユーティリティー・タスクを実行できます。

デフォルト・インスタンス

DB2 インストール手順の一部として、他のインスタンスに 『DB2』 という名前が付いていなければ、DB2 というデータベース・マネージャーの初期インスタンスを作成します。 DB2 バージョン 8 をインストールしていて、バージョン 9.1 またはバージョン 9.5 にアップグレードする場合、デフォルト・インスタンスは「DB2_01」です。

Linux およびUNIX では、命名規則の指針にはずれない範囲で初期インスタンスの名前を付けることができます。インスタンス名は、ディレクトリー構造を設定するために使用します。

このインスタンスをすぐに使えるようにするために、インストール中に次の設定がなされます。

- 環境変数 DB2INSTANCE が 『DB2』 に設定される。
- レジストリー変数 DB2INSTDEF が 『DB2』 に設定される。

これらの設定により、『DB2』 がデフォルト・インスタンスとして確立されます。デフォルトで使用されるインスタンスを変更することはできますが、最初に、追加インスタンスを作成する必要があります。

データベース・マネージャーを使用する前に、各ユーザーのデータベース環境を更新して、インスタンスにアクセスし、DB2 データベース・プログラムを実行できるようにします。このことはすべてのユーザー (管理ユーザーも含む) に当てはまります。

Linux およびUNIX オペレーティング・システムでは、データベース環境の設定に役立つサンプル・スクリプト・ファイルが提供されます。サンプル・ファイルは、Bourne または Korn シェルの場合は db2profile、 C シェルの場合は db2cshrc です。これらのスクリプトは、インスタンス所有者のホーム・ディレクトリー下の sqllib サブディレクトリーに配置されます。インスタンス所有者またはインスタンスの SYSADM グループに属するユーザーはだれでも、インスタンスの全ユーザーのスクリプトをカスタマイズできます。ユーザーごとにスクリプトをカスタマイズするには、sqllib/userprofile および sqllib/usercshrc を使用します。

ブランク・ファイル sqllib/userprofile および sqllib/usercshrc は、インスタンスの作成時に作成され、独自のインスタンス環境設定を追加できるようにします。DB2 フィックスパックをインストールする際、インスタンスの更新時に db2profile および db2cshrc ファイルは上書きされます。db2profile または db2cshrc スクリプト中の新しい環境設定を望まない場合は、対応する *user* スクリプト を使用してオーバーライドできます。このスクリプトは、ddb2profile または db2cshrc スクリプトの終了時に呼び出されます。インスタンスのマイグレーション (db2imigr コマンドを使用) 時に、user スクリプトは上書きコピーされるので、環境に対する変更は引き続き使用されます。

サンプル・スクリプトには、次の目的を持つステートメントが入っています。

- 既存の検索パスに以下のディレクトリーを追加して、ユーザーの「パス」を更新します。すなわち、インスタンス所有者のホーム・ディレクトリーの sqllib サブディレクトリー下にある bin、 adm、misc サブディレクトリー。
- DB2INSTANCE 環境変数にインスタンス名を設定します。

インスタンス・ディレクトリー

インスタンス・ディレクトリーには、データベース・インスタンスに関連するすべての情報が保管されます。インスタンス・ディレクトリーの位置を作成後に変更することはできません。

インスタンス・ディレクトリーの内容は、以下のとおりです。

- データベース・マネージャー構成ファイル
- システム・データベース・ディレクトリー
- ノード・ディレクトリー
- ノード構成ファイル (db2nodes.cfg)
- デバッグ情報 (例外ダンプ、レジスター・ダンプ、DB2 プロセス用の呼び出しスタックなど) が入った他のファイル。

Linux および UNIX オペレーティング・システムでは、インスタンス・ディレクトリーは INSTHOME/sqllib ディレクトリーにあります。ただし、INSTHOME は、インスタンス所有者のホーム・ディレクトリーです。命名規則の指針にはずれない範囲でデフォルト・インスタンスの名前を付けることができます。

Windows オペレーティング・システムでは、インスタンス・ディレクトリーは DB2 データベース製品がインストールされた /sqllib ディレクトリーの下にあります。インスタンス名はサービス名と同じなので、競合は発生しません。インスタンス名が別のサービス名と同じになることはありません。サービスを作成するための適切な許可が必要です。

パーティション・データベース環境におけるインスタンス・ディレクトリーは、インスタンスに属するすべてのデータベース・パーティション・サーバー間で共用されます。したがって、インスタンス・ディレクトリーは、インスタンス内のすべてのコンピューターがアクセスできるネットワーク共用ドライブ上に作成しなければなりません。

db2nodes.cfg

db2nodes.cfg ファイルを使用して、DB2 インスタンスに関与するデータベース・パーティション・サーバーを定義します。また、データベース・パーティション・サーバー通信に高速相互接続を使用する場合には、db2nodes.cfg ファイルを使用して、高速相互接続の IP アドレスまたはホスト名を指定します。

複数インスタンス (Linux、UNIX)

DB2 製品を root 特権を使用してインストールした場合は、1 つの Linux または UNIX オペレーティング・システム上に複数のインスタンスを作成することが可能です。各インスタンスは同時に実行されますが、それぞれは独立しています。したがって、一度にデータベース・マネージャーの 1 つのインスタンスでのみ作業できます。

注: 複数のインスタンス間での環境競合を防ぐために、各インスタンスごとにホーム・ディレクトリーを設定する必要があります。 ホーム・ディレクトリーが共有されている場合、エラーが戻されます。 各ホーム・ディレクトリーは、同じファイル・システム上にあっても構いませんし、別のファイル・システム上にあっても構いません。

インスタンス所有者およびシステム管理 (SYSADM) グループであるグループは、個々のインスタンスと関連付けられます。インスタンス所有者および SYSADM グループは、インスタンスの作成プロセス中に割り当てられます。 1 つのユーザー ID またはユーザー名が使用できるのは 1 つのインスタンスに対してのみで、そのユーザー ID またはユーザー名は、*インスタンス所有者* とも呼ばれます。

各インスタンス所有者は固有のホーム・ディレクトリーを持つ必要があります。インスタンスの実行に必要な構成ファイルはすべて、インスタンス所有者のユーザー ID またはユーザー名のホーム・ディレクトリーに作成されます。 インスタンス所有者のユーザー ID またはユーザー名をシステムから除去する必要が生じた場合、インスタンスに関連付けられたファイルと、そのインスタンスに保管されたデータへのアクセスを失うおそれがあります。このため、インスタンス所有者のユーザー ID またはユーザー名は、専らデータベース・マネージャーの実行だけに使用してください。

インスタンス所有者の 1 次グループも重要です。この 1 次グループは自動的にインスタンスのシステム管理グループになり、インスタンスに対する SYSADM 権限を取得します。インスタンス所有者の 1 次グループのメンバーである他のユーザー ID またはユーザー名も、このレベルの権限を取得します。この理由から、インスタンス所有者のユーザー ID またはユーザー名は、インスタンスの管理用に確保した 1 次グループに割り当てたいと思うかもしれません。(また、1 次グループは必ずインスタンス所有者のユーザー ID またはユーザー名に割り当てるようにします。割り当てなければ、システム・デフォルトの 1 次グループが使用されます。)

インスタンスのシステム管理グループにするグループがすでにある場合は、インスタンス所有者ユーザー ID またはユーザー名の作成時に、そのグループを 1 次グループとして割り当てるだけで済みます。インスタンスに対する管理権限を他のユーザーに付与するには、システム管理グループとして割り当てられたグループに該当するユーザーを追加します。

インスタンス間で SYSADM 権限を分離するには、インスタンス所有者ユーザー ID またはユーザー名ごとに異なる 1 次グループを使用します。ただし、複数インスタンスで共通の SYSADM 権限を持つことにした場合は、複数インスタンスに対して同じ 1 次グループを使用することができます。

複数インスタンス (Windows)

同じコンピューター上で、データベース・マネージャーの複数のインスタンスを実行することが可能です。それぞれのデータベース・マネージャーのインスタンスは独自のデータベースを保守し、独自のデータベース・マネージャー構成パラメーターを持っています。

注: インスタンスは、1 台のコンピューター上で、異なる DB2 コピーに属することもでき、これらのコピーは、データベース・マネージャーの異なるレベルに存在することが可能です。

データベース・マネージャーの 1 つのインスタンスは以下のもので構成されます。

- インスタンスを表す Windows サービス。サービスの名前は、インスタンス名と同じです。 (「サービス」パネルでの) サービスの表示名は、インスタンス名の前にストリング ″DB2 - ″ が付きます。例えば、インスタンス名が「DB2」であれば、「DB2」という名前の Windows サービスが存在し、その表示名は「DB2 - <DB2 コピー名> - DB2」です。

  注: Windows サービスは、クライアント・インスタンス用には作成されません。
- インスタンス・ディレクトリー。このディレクトリーには、データベース・マネージャー構成ファイル、システム・データベース・ディレクトリー、ノード・デ

ィレクトリー、データベース接続サービス (DCS) ディレクトリー、およびインスタンスに関連したすべての診断ログとダンプ・ファイルが含まれます。デフォルトでは、インスタンス・ディレクトリーは SQLLIB ディレクトリー内のサブディレクトリーとなり、インスタンス名と同じ名前になります。例えば、インスタンス「DB2」のインスタンス・ディレクトリーは C:¥SQLLIB¥DB2 です (C:¥SQLLIB はデータベース・マネージャーのインストール場所)。レジストリー変数 DB2INSTPROF を使用して、インスタンス・ディレクトリーのデフォルト場所を変更することができます。レジストリー変数 DB2INSTPROF の値を他の場所に設定した場合、インスタンス・ディレクトリーは DB2INSTPROF で指定されたディレクトリーの下に作成されます。例えば、DB2INSTPROF=D:¥DB2PROFS と設定した場合、インスタンス・ディレクトリーは D:¥DB2PROFS¥DB2 となります。

  – db2set.exe -g コマンドを使用して、DB2INSTPROF を c:¥DB2PROFS に設定する。
  – DB2ICRT.exe コマンドを実行して、インスタンスを作成する。
- Windows オペレーティング・システムでインスタンスを作成するとき、インスタンスのディレクトリーや db2cli.ini ファイルといったユーザー・データ・ファイルのデフォルトの場所は、以下のディレクトリーになります。
  – Windows XP および Windows 2003 オペレーティング・システムでは、Documents and Settings¥All Users¥Application Data¥IBM¥DB2¥コピー名
  – Windows Vista オペレーティング・システムでは、ProgramData¥IBM¥DB2¥コピー名

インスタンスの作成

データベース・マネージャーのインストール時にはインスタンスが 1 つ作成されますが、ビジネス・ニーズに応じて、さらに追加のインスタンスを作成する必要が生じるかもしれません。

前提条件

Windows の管理グループに属しているユーザー、あるいは Linux または UNIX プラットフォームの root 権限を持っているユーザーが、追加のインスタンスを作成できます。インスタンスの追加先のコンピューターは、インスタンス所有コンピューター (ノード 0) になります。インスタンスは、DB2 Administration Server が常駐するコンピューター上に必ず追加してください。インスタンス ID は root であってはならず、パスワードが失効していてもなりません。

制約事項

- Linux および UNIX オペレーティング・システム上では、追加のインスタンスを非 root インストールに対して作成することはできません。
- 既存のユーザー ID を使って DB2インスタンスを作成する場合、そのユーザー ID に関して以下の点を確認してください。
  – ロックされていない
  – パスワードが有効期限切れでない

コマンド行を使用してインスタンスを追加するには、以下のように入力します。

```
db2icrt <instance_name>
```

AIX サーバー上でインスタンスを作成する場合は、例えば以下のように、fenced ユーザー ID を提供する必要があります。

```
DB2DIR/instance/db2icrt -u db2fenc1 db2inst1
```

db2icrt コマンドを使用して別の DB2 インスタンスを追加するときには、インスタンス所有者のログイン名を提供する必要があり、オプションで、インスタンスの認証タイプを指定します。認証タイプは、そのインスタンスの下で作成されたすべてのデータベースに適用されます。認証タイプとは、ユーザーの認証が行われる場所を示すものです。

インスタンス・ディレクトリーの位置は、 DB2INSTPROF 環境変数を使って DB2PATH から変更することができます。その場合は、インスタンス・ディレクトリーの書き込みアクセスが必要です。 DB2PATH 以外のパスにディレクトリーを作成する場合、db2icrt コマンドを入力する前に、DB2INSTPROF を設定しなければなりません。

DB2 Enterprise Server Edition (ESE) の場合、追加する新規インスタンスがパーティション・データベース・システムであることを宣言する必要もあります。さらに、複数のデータベース・パーティションがある ESE インスタンスに対して作業を行うとき、および高速コミュニケーション・マネージャー (FCM) に対して作業を行うときには、インスタンスの作成時に追加の TCP/IP ポートを定義することにより、データベース・パーティション相互間に複数の接続を確立できます。

例えば、Windows オペレーティング・システムでは、-r <port range> パラメーターを指定して db2icrt コマンドを使用します。ポートの範囲は次のとおりです (base_port は FCM が使用できる最初のポート、end_port は FCM が使用できるポート番号範囲の最後のポートです)。

```
-r:<base_port,end_port>
```

インスタンスの変更

インスタンスは、後ほど製品をインストールしたり削除したりしても、できるだけ影響を受けないように設計されています。 Linux および UNIX では、実行可能プログラムまたはコンポーネントをインストールまたは除去した後にインスタンスを更新することができます。 Windows では、db2iupdt コマンドを実行します。

既存のインスタンスはたいてい、インストールまたは削除する製品の機能を自動的に継承するか、アクセスを失うかのどちらかです。しかし、特定の実行可能コードやコンポーネントがインストールまたは削除された場合、既存のインスタンスは自動的に新しいシステムの構成パラメーターを継承したり、すべての追加機能へのアクセスを取得したりするわけではありません。そのインスタンスは更新しなければなりません。

プログラム一時修正 (PTF) またはパッチをインストールしてデータベース・マネージャーを更新する場合は、db2iupdt コマンド (root インストールの場合) または db2nrupdt コマンド (非 root インストールの場合) を使って既存のすべてのデータベース・インスタンスを更新する必要があります。

インスタンスを変更または削除する前には、インスタンス内にあるインスタンス・サーバーとデータベース・パーティション・サーバーを理解しておく必要があります。

インスタンス構成の更新 (Linux、UNIX)

このトピックは、ルート・インスタンスにのみ該当します。ルート以外のインスタンスを更新するには、db2nrupdt コマンドを実行してください。

db2iupdt コマンドを実行すると、以下の操作が実行されて、指定したインスタンスが更新されます。

- インスタンス所有者のホーム・ディレクトリー下の sqllib サブディレクトリーにあるファイルを置き換える。
- ノード・タイプが変更されていれば、新しいデータベース・マネージャー構成ファイルが作成される。この作成は、既存のデータベース・マネージャー構成ファイルから関連する値を、新しいノード・タイプ用のデフォルトのデータベース・マネージャー構成ファイルとマージして行います。新しいデータベース・マネージャー構成ファイルが作成されると、古いファイルは、インスタンス所有者のホーム・ディレクトリーにある sqllib サブディレクトリーの backup サブディレクトリーにバックアップされます。

db2iupdt コマンドは、 AIX では /usr/opt/db2_09_05/instance/ ディレクトリーにあります。 db2iupdt コマンドは、HP-UX、Solaris、または Linux では /opt/IBM/db2/V9.5/instance/ ディレクトリーにあります。

コマンド行を使用してインスタンスを更新するには、以下のように入力します。

```
db2iupdt InstName
```

InstName はインスタンス所有者のログイン名です。

このコマンドに関連して、他にも次のオプション・パラメーターがあります。

-h または -?
このコマンドのヘルプ・メニューを表示します。

-d 問題判別中に使用するデバッグ・モードを設定します。

-a AuthType
インスタンスの認証タイプを指定します。有効な認証タイプは、SERVER、SERVER_ENCRYPT、または CLIENT です。指定しない場合、DB2 サーバーがインストールされていれば、デフォルトは SERVER に設定されます。 それ以外の場合は、CLIENT に設定されます。インスタンスの認証タイプは、インスタンスが所有するすべてのデータベースに適用されます。

-e 既存の各インスタンスを更新できます。既存のインスタンスをリストするには db2ilist を使用します。

-u Fenced ID
fenced ユーザー定義関数 (UDF) とストアード・プロシージャーを実行するユーザーの名前を指定します。 Data Server Client または DB2 Software

Developer's Kit をインストールする場合、これは必須ではありません。他の DB2 製品の場合、これは必須パラメーターです。注: Fenced ID には、″root″ または ″bin″ は使えません。

-k このパラメーターは、現在のインスタンス・タイプを保存します。このパラメーターを指定しない場合、現行インスタンスは次の順序で、使用可能な上位のインスタンス・タイプにアップグレードされます。

- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー
- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- クライアント

次に例を示します。

- インスタンスの作成後に DB2 Workgroup Server Edition または DB2 Enterprise Server Edition をインストールした場合は、次のコマンドを入力してインスタンスを更新してください。

```
db2iupdt -u db2fenc1 db2inst1
```

- インスタンスの作成後に DB2 Connect™ Enterprise Server Edition をインストールした場合は、インスタンス名を fenced ID としても使用できます。

```
db2iupdt -u db2inst1 db2inst1
```

- クライアント・インスタンスを更新するには、次のコマンドを呼び出します。

```
db2iupdt db2inst1
```

インスタンス構成の更新 (Windows)

Windows でインスタンス構成を更新するには、db2iupdt コマンドを使用します。

db2iupdt コマンドを実行すると、以下の操作が実行されて、指定したインスタンスが更新されます。

- インスタンス所有者のホーム・ディレクトリー下の sqllib サブディレクトリーにあるファイルを置き換える。
- ノード・タイプが変更されていれば、新しいデータベース・マネージャー構成ファイルが作成される。この作成は、既存のデータベース・マネージャー構成ファイルから関連する値を、新しいノード・タイプ用のデフォルトのデータベース・マネージャー構成ファイルとマージして行います。新しいデータベース・マネージャー構成ファイルが作成されると、古いファイルは、インスタンス所有者のホーム・ディレクトリーにある sqllib サブディレクトリーの backup サブディレクトリーにバックアップされます。

db2iupdt コマンドは、¥sqllib¥bin ディレクトリーにあります。

コマンドは、次のように使用します。

```
db2iupdt InstName
```

InstName はインスタンス所有者のログイン名です。

このコマンドに関連して、他にも次のオプション・パラメーターがあります。

/h: hostname
現行コンピューターに 1 つ以上の TCP/IP ホスト名がある場合に、デフォルトの TCP/IP ホスト名をオーバーライドします。

/p: instance profile path
更新されたインスタンスに対して新規のインスタンス・プロファイル・パスを指定します。

/r: baseport,endport
複数データベース・パーティションで稼働している時に、パーティション・データベース・インスタンスにより使用される TCP/IP ポートの範囲を指定します。

/u: username,password
DB2 サービスのアカウント名とパスワードを指定します。

インスタンスの操作

インスタンスを操作するときには、インスタンスの開始と停止、インスタンスへのアタッチ、およびインスタンスからのデタッチが可能です。

各インスタンスは、*インスタンス構成ファイル* (*データベース・マネージャー構成ファイル*としても知られる) で定義された SYSADM_GROUP に属するユーザーにより管理されます。ユーザー ID とユーザー・グループの作成方法は、それぞれの稼働環境ごとに異なります。

インスタンスの自動開始

Windows オペレーティング・システムでは、インストール時に作成されるデータベース・インスタンスはデフォルトで自動始動するよう設定されます。 db2icrt を使って作成されるインスタンスは、手動開始するよう設定されます。開始タイプを変更するには、「サービス」パネルに移動して、その DB2 サービスのプロパティーを変更する必要があります。

UNIX オペレーティング・システムで、各システムの再始動後にインスタンスを自動始動できるようにするには、次のコマンドを入力します。

```
db2iauto -on <instance name>
```

ここで <instance name> はインスタンスのログイン名です。 UNIX オペレーティング・システムで、システム再始動のたびにインスタンスを自動開始しないように設定するには、次のコマンドを入力します。

```
db2iauto -off <instance name>
```

<instance name> はインスタンスのログイン名です。

インスタンスの開始 (Linux、UNIX)

通常の業務の最中に DB2 データベースを開始または停止する必要が生じることがあります。例えば、インスタンス上のデータベースへの接続、アプリケーションのプリコンパイル、データベースへのパッケージのバインド、ホスト・データベースへのアクセスなどのタスクを実行する前に、インスタンスを開始しなければなりません。

Linux または UNIX システム上のインスタンスを開始する前に、

1. インスタンスに対する SYSADM、SYSCTRL、または SYSMAINT 権限を持つユーザー ID またはユーザー名を使ってログインします。あるいは、インスタンス所有者としてログインします。
2. 始動スクリプトを次のように実行します (INSTHOME は使用するインスタンスのホーム・ディレクトリー)。

```
. INSTHOME/sqllib/db2profile    (Bourne または Korn シェルの場合)
  source INSTHOME/sqllib/db2cshrc (C シェルの場合)
```

コマンド行を使用してインスタンスを開始するには、以下のように入力します。

```
db2start
```

注: インスタンスのデータベース・マネージャーを開始または停止するためのコマンドを実行すると、 DB2 データベース・マネージャーはそのコマンドを現行インスタンスに適用します。

インスタンスの開始 (Windows)

通常の業務の最中に DB2 インスタンスを開始または停止する必要が生じることがあります。例えば、インスタンス上のデータベースへの接続、アプリケーションのプリコンパイル、データベースへのパッケージのバインド、ホスト・データベースへのアクセスなどのタスクを実行する前に、インスタンスを開始しなければなりません。

db2start から DB2 データベース・インスタンスをサービスとして正常に起動するには、ユーザー・アカウントは、 Windows サービスを開始するために Windows オペレーティング・システムで定義された適切な特権を持っている必要があります。ユーザー・アカウントは、管理者、サーバー・オペレーター、またはパワー・ユーザーのいずれかのグループのメンバーです。拡張セキュリティーを使用可能にした場合、デフォルトでは DB2ADMNS および Administrators グループのメンバーだけがデータベースを始動できます。

コマンド行を使用してインスタンスを開始するには、以下のように入力します。

```
db2start
```

注: インスタンスのデータベース・マネージャーを開始または停止するためのコマンドを実行すると、 DB2 データベース・マネージャーはそのコマンドを現行インスタンスに適用します。

db2start コマンドを使用すると、DB2 データベース・インスタンスは Windows サービスとして起動します。 db2start を呼び出すときにスイッチ ″/D″ を指定すると、DB2 データベース・インスタンスを Windows 上でプロセスとして実行することができます。「コントロール パネル」または NET START コマンドを使用して、DB2 データベース・インスタンスをサービスとして開始することもできます。

パーティション・データベース環境で実行しているときには、各データベース・パーティション・サーバーは Windows サービスとして開始されます。パーティション・データベース環境で DB2 インスタンスをプロセスとして開始するために、「/D」スイッチを使用することはできません。

インスタンスとのアタッチおよびデタッチ

すべてのプラットフォームにおいて、データベース・マネージャーの (リモート・インスタンスを含む) 別のインスタンスにアタッチするには、ATTACH コマンドを使用します。インスタンスからデタッチするには、DETACH コマンドを使用します。

複数のインスタンスがすでに存在しなければなりません。

コマンド行を使用してインスタンスにアタッチするには、以下のように入力します。

```
db2 attach to <instance name>
```

例えば、以前にノード・ディレクトリーにカタログされた testdb2 というインスタンスにアタッチするには、次のようにします。

```
db2 attach to testdb2
```

例えば testdb2 インスタンスの保守に関連した作業を実行した後、コマンド行を使ってインスタンスからデタッチするには、次のように入力します。

```
db2 detach
```

クライアント・アプリケーションとのアタッチおよびデタッチ

- クライアント・アプリケーションからインスタンスにアタッチするには、sqleatin API を呼び出します。
- クライアント・アプリケーションから、インスタンスをデタッチするには、sqledtin API を呼び出します。

同じまたは異なる DB2 コピー上のインスタンスでの作業

複数のインスタンスを、同じ DB2 コピーまたは複数の異なる DB2 コピーで並行して実行することができます。

同一の DB2 コピーでインスタンスに関する作業を行うには、以下のようにする必要があります。

1. すべてのインスタンスを同一の DB2 コピーに作成またはマイグレーションします。
2. そのインスタンスに対してコマンドを発行する前に、DB2INSTANCE 環境変数をその作業対象のインスタンス名に設定する必要があります。

あるインスタンスが別のデータベース・インスタンスにアクセスするのを防ぐために、インスタンス用のデータベース・ファイルはインスタンス名と同じ名前のディレクトリーの下に作成されます。例えば、インスタンス「DB2」用のデータベースを C: ドライブに作成する場合、データベース・ファイルは C:¥DB2 ディレクトリーの下に作成されます。同様に、インスタンス TEST 用のデータベースを C: ドライブに作成する場合、データベース・ファイルは C:¥TEST ディレクトリーの下に作成されます。デフォルトで、その値は DB2 製品がインストールされているドライブのドライブ名になります。詳しくは、*dftdbpath* データベース・マネージャーの構成パラメーターを参照してください。

複数の DB2 コピーのあるシステムで特定の 1 つのインスタンスに関する作業を行うには、以下の方式のどちらかを使用します。

- 「コマンド・ウィンドウ」を次のように使用します。「スタート」→「すべてのプログラム」→「IBM DB2」→ *<DB2 コピー名>* →「コマンド行ツール」→「コマンド・ウィンドウ」。この「コマンド・ウィンドウ」は、選択した特定の DB2 コピー用に正しい環境変数で事前にセットアップされています。
- 「コマンド・ウィンドウ」から db2envar.bat を使用します。
  1. 「コマンド・ウィンドウ」をオープンします。
  2. アプリケーションで使用する DB2 コピーの完全修飾パスを使用して、db2envar.bat ファイルを実行します。

     ```
     <DB2 Copy install dir>¥bin¥db2envar.bat
     ```

インスタンスの停止 (Linux、UNIX)

データベース・マネージャーの現在のインスタンスを停止する必要があるかもしれません。

Linux または UNIXシステム上のインスタンスを停止するには、以下を実行する必要があります。

1. インスタンスに対する SYSADM、SYSCTRL、または SYSMAINT 権限を持つユーザー ID またはユーザー名を使ってインスタンスにログインまたはアタッチします。あるいは、インスタンス所有者としてログインします。
2. 停止する特定のデータベースに接続された、すべてのアプリケーションおよびユーザーを表示します。重要なアプリケーションまたはクリティカルなアプリケーションが実行されていないことを確認するために、アプリケーションをリストします。リストを表示するには、SYSADM、SYSCTRL、または SYSMAINT 権限が必要です。
3. すべてのアプリケーションおよびユーザーにデータベースの使用を中断させます。ユーザーを強制終了するためには、SYSADM または SYSCTRL 権限が必要です。

db2stop コマンドはサーバーでのみ実行できます。このコマンドの実行中はデータベースの接続は許されません。インスタンス接続があると、インスタンスが停止する前に強制的にオフにされます。

注: コマンド行プロセッサーのセッションがインスタンスにアタッチされたら、terminate コマンドを実行し、各セッションを終了してから db2stop コマンドを実行します。db2stop コマンドを実行すると、 DB2INSTANCE 環境変数で定義されたインスタンスが停止します。

コマンド行を使用してインスタンスを停止するには、以下のように入力します。

```
db2stop
```

db2stop コマンドを使用して、パーティション・データベース環境内の個々のデータベース・パーティションを停止またはドロップすることができます。パーティション・データベース環境を使用し、以下のコマンドで論理パーティションをドロップする場合、

```
db2stop drop nodenum <0>
```

そのデータベースへのアクセスを試行しているユーザーがいないことを確認してください。そのようなユーザーが存在する場合、エラー・メッセージ SQL6030N が受信されます。

注: インスタンスのデータベース・マネージャーを開始または停止するためのコマンドを実行すると、 DB2 データベース・マネージャーはそのコマンドを現行インスタンスに適用します。詳しくは、『現行インスタンス環境変数の設定』を参照してください。

インスタンスの停止 (Windows)

データベース・マネージャーの現在のインスタンスを停止する必要があるかもしれません。

システムでインスタンスを停止するには、以下のことを行わなければなりません。

1. DB2 データベース・サービスの停止を行うユーザー・アカウントは、 Windows オペレーティング・システムで定義された適切な特権を持っている必要があります。ユーザー・アカウントは、管理者、サーバー・オペレーター、またはパワー・ユーザーのいずれかのグループのメンバーです。
2. 停止する特定のデータベースに接続された、すべてのアプリケーションおよびユーザーを表示します。重要なアプリケーションまたはクリティカルなアプリケーションが実行されていないことを確認するために、アプリケーションをリストします。リストを表示するには、SYSADM、SYSCTRL、または SYSMAINT 権限が必要です。
3. すべてのアプリケーションおよびユーザーにデータベースの使用を中断させます。ユーザーを強制終了するためには、SYSADM または SYSCTRL 権限が必要です。

db2stop コマンドはサーバーでのみ実行できます。このコマンドの実行中はデータベースの接続は許されません。インスタンス接続があると、DB2 データベース・サービスが停止する前に強制的にオフにされます。

注: コマンド行プロセッサーのセッションがインスタンスにアタッチされたら、terminate コマンドを実行し、各セッションを終了してから db2stop コマンドを実行します。db2stop コマンドを実行すると、 DB2INSTANCE 環境変数で定義されたインスタンスが停止します。

システムのインスタンスを停止するには、以下のいずれかの方法に従ってください。

- db2stop コマンドを使用して停止する。
- NET STOP コマンドを使用して停止する。
- アプリケーション内からインスタンスを停止する。

パーティション・データベース環境でデータベース・マネージャーを使用しているときには、各データベース・パーティション・サーバーがサービスとして開始されることに注意してください。それぞれのサービスを停止する必要があります。

注: インスタンスのデータベース・マネージャーを開始または停止するためのコマンドを実行すると、データベース・マネージャーはそのコマンドを現行インスタンスに適用します。詳しくは、『現行インスタンス環境変数の設定』を参照してください。

インスタンスのドロップ

このトピックは、すべてのプラットフォームのルート・インスタンスにのみ該当します。ルート以外のインスタンスをドロップするには、DB2製品をアンインストールする必要があります。

インスタンスを除去するには、コマンド行から以下のように入力します。

```
db2idrop <instance_name>
```

コマンド行を使用してインスタンスを除去する場合の、準備事項と詳細は以下のとおりです。

1. インスタンスを現在使用しているアプリケーションをすべて停止します。
2. それぞれの「コマンド・ウィンドウ」で terminate コマンドを実行して、コマンド行プロセッサーを停止します。
3. db2stop コマンドを実行してインスタンスを停止します。
4. DB2INSTPROF レジストリー変数で指示されたインスタンス・ディレクトリーをバックアップします。

   Linux および UNIXオペレーティング・システムでは、INSTHOME/sqllib ディレクトリー内のファイルをバックアップすることを考慮してください (INSTHOME はインスタンス所有者のホーム・ディレクトリー)。例えば、データベース・マネージャー構成ファイル db2systm、 db2nodes.cfg ファイル、ユーザー定義関数 (UDF)、または fenced ストアード・プロシージャー・アプリケーションを保管すると思うかもしれません。
5. (Linuxおよび UNIXオペレーティング・システムのみ) インスタンス所有者としてログオフします。
6. (Linux および UNIX オペレーティング・システム上のみ) root 権限を持つユーザーとしてログインします。
7. 次のように db2idrop コマンドを発行します。

   ```
   db2idrop InstName
   ```

   ここで、InstName は、ドロップされるインスタンスの名前です。

   このコマンドを使うと、インスタンスのリストからインスタンス項目が除去され、インスタンス・ディレクトリーが除去されます。
8. (Linuxおよび UNIXオペレーティング・システムのみ) オプションで、root 権限を持つユーザーとして、インスタンス所有者のユーザー ID とグループを除去することもできます (そのインスタンスにのみ使用されている場合)。インスタンスの再作成を計画している場合は、上記の除去は行わないでください。

   インスタンス所有者およびインスタンス所有者グループは他の目的で使用することもあるので、このステップはオプションです。

db2idrop コマンドを使うと、インスタンスのリストからインスタンス項目が除去され、インスタンス所有者のホーム・ディレクトリーにある sqllib サブディレクトリーが除去されます。

注: Linuxおよび UNIXオペレーティング・システムでは、db2idrop コマンドを使ってインスタンスをドロップしようとすると、sqllib サブディレクトリーは除去できないという内容のメッセージが生成され、adm サブディレクトリー内に拡張子 .nfs のファイルがいくつか生成されます。 adm サブディレクトリーは NFS でマウントされたシステムで、ファイルはサーバー上で制御されます。ディレクトリーがマウントされているファイル・サーバーから *.nfs ファイルを削除する必要があります。その後、sqllib サブディレクトリーを除去できます。

第 5 章 Lightweight Directory Access Protocol (LDAP)

Lightweight Directory Access Protocol (LDAP) は、ディレクトリー・サービスへの業界標準アクセス方式です。ディレクトリー・サービスとは、分散環境内にある複数のシステムおよびサービスについてのリソース情報を集めたリポジトリーです。クライアントとサーバーはディレクトリー・サービスを使用して、それらのリソースにアクセスします。

各データベース・サーバーのインスタンスは自らの存在を LDAP サーバーに公開するとともに、データベースの作成時にはデータベース情報を LDAP ディレクトリーへ送信します。クライアントがデータベースに接続すると、 LDAP ディレクトリーからそのサーバーのカタログ情報を取り出せます。各クライアントは、それぞれのマシンでローカルにカタログ情報を保管する必要はなくなります。クライアント・アプリケーションは、LDAP ディレクトリーの中で、データベースへ接続するのに必要な情報を探します。

キャッシュ・メカニズムが備えられているので、クライアントは LDAP ディレクトリー・サーバーを一度探すだけで済みます。 LDAP ディレクトリー・サーバーから情報を検索した後、 *dir_cache* データベース・マネージャー構成パラメーターおよび DB2LDAPCACHE レジストリー変数の値に基づいて、その情報はローカル・マシンに格納されるか、キャッシュに入れられます。データベース・マネージャー構成パラメーター *dir_cache* は、データベース、ノード、および DCS ディレクトリー・ファイルをメモリー・キャッシュに保管するために使用されます。ディレクトリー・キャッシュは、アプリケーションがクローズするまでアプリケーションによって使用されます。レジストリー変数 DB2LDAPCACHE は、データベース、ノード、および DCS ディレクトリー・ファイルをローカル・ディスク・キャッシュに保管するために使用されます。

- DB2LDAPCACHE=NO かつ *dir_cache*=NO の場合、情報は必ず LDAP から読み取られます。
- DB2LDAPCACHE=NO かつ *dir_cache*=YES の場合、 LDAP から一度情報が読み取られ、その情報は DB2 キャッシュに挿入されます。
- DB2LDAPCACHE=YES であるか、これが設定されない場合には、 LDAP から情報が一度読み取られ、その情報はローカル・データベース、ノード、および DCS ディレクトリーにキャッシュされます。

注: DB2LDAPCACHE レジストリー変数が適用されるのは、データベースとノード・ディレクトリーだけです。

LDAP 環境でのセキュリティーに関する考慮事項

LDAP ディレクトリーの情報にアクセスする前に、アプリケーションまたはユーザーは、LDAP サーバーによって認証されます。この認証作業のことを、LDAP サーバーに対するバインディング といいます。匿名のユーザーが LDAP ディレクトリーに格納されている情報を追加、削除、あるいは変更してしまわないように、その情報にアクセス・コントロールを設定することは重要です。

アクセス・コントロールは、デフォルトでコンテナー・レベルで継承され、適用できるものです。新しいオブジェクトが作成される場合、その新しいオブジェクトは、親オブジェクトと同じセキュリティー属性を継承します。コンテナー・オブジェクトにアクセス・コントロールを定義するには、 LDAP サーバー用の管理ツールを使用できます。

デフォルトでは、アクセス・コントロールは以下のように定義されています。

- LDAP 内のデータベースおよびノード項目については、すべて (つまり任意の匿名ユーザー) が読み取りアクセス権限を持っています。読み取り/書き込みアクセス権限を持っているのは、ディレクトリー管理者と、そのオブジェクトの所有者または作成者だけです。
- ユーザー・プロファイルについては、プロファイル所有者と、ディレクトリー管理者が読み取り/書き込みアクセス権限を持っています。他のユーザーのプロファイルにアクセスしようとしているユーザーは、ディレクトリー管理者権限を持っていなければ、そこにアクセスできません。

注: 許可検査は必ず LDAP サーバーによって実行されます。 DB2 によって実行されることはありません。 LDAP 許可検査は DB2 認証とは関係がありません。 SYSADM 権限を持つアカウントまたは許可 ID であっても、LDAP ディレクトリーに対するアクセス権は持っていない可能性があります。

LDAP コマンドまたは API を実行するときに、バインド識別名 (bindDN) とパスワードを指定しなかった場合、 DB2 はデフォルトの証明書を使用して LDAP サーバーにバインドします。この証明書の権限では要求したコマンドを実行できない場合もあります。その場合、エラーが戻されます。

DB2 コマンドまたは API では、 USER 節と PASSWORD 節を使用してユーザーの bindDN とパスワードを明示的に指定できます。

DB2 で使用する LDAP オブジェクト・クラスおよび属性

以下の表は、DB2 データベース・マネージャーで使用するオブジェクト・クラスを示しています。

表 10. cimManagedElement

クラス	**cimManagedElement**
Active Directory LDAP 表示名	適用されない
Active Directory 共通名 (cn)	適用されない
説明	IBM Schema で使用される多くのシステム管理オブジェクト・クラスの基本クラスを提供します。
SubClassOf	最上位
必須属性	
オプション属性	説明
タイプ	抽象型
OID (オブジェクト ID)	1.3.18.0.2.6.132
GUID (グローバル・ユニーク ID)	b3afd63f-5c5b-11d3-b818-002035559151

表 11. *cimSetting*

クラス	**cimSetting**
Active Directory LDAP 表示名	適用されない
Active Directory 共通名 (cn)	適用されない
説明	IBM Schema での構成および設定に使用する基本クラスを提供します。
SubClassOf	cimManagedElement
必須属性	
オプション属性	settingID
タイプ	抽象型
OID (オブジェクト ID)	1.3.18.0.2.6.131
GUID (グローバル・ユニーク ID)	b3afd64d-5c5b-11d3-b818-002035559151

表 12. *eProperty*

クラス	**eProperty**
Active Directory LDAP 表示名	ibm-eProperty
Active Directory 共通名 (cn)	ibm-eProperty
説明	ユーザーが選択できるプロパティーに、アプリケーション特有の設定を指定するときに使います。
SubClassOf	cimSetting
必須属性	
オプション属性	propertyType cisPropertyType cisProperty cesPropertyType cesProperty binPropertyType binProperty
タイプ	構造
OID (オブジェクト ID)	1.3.18.0.2.6.90
GUID (グローバル・ユニーク ID)	b3afd69c-5c5b-11d3-b818-002035559151

表 13. *DB2Node*

クラス	**DB2Node**
Active Directory LDAP 表示名	ibm-db2Node
Active Directory 共通名 (cn)	ibm-db2Node
説明	DB2 サーバーを表します。
SubClassOf	eSap / ServiceConnectionPoint
必須属性	db2nodeName

表 13. DB2Node (続き)

クラス	DB2Node
オプション属性	db2nodeAlias db2instanceName db2Type host / dNSHostName (注 2 参照) protocolInformation/ServiceBindingInformation
タイプ	構造
OID (オブジェクト ID)	1.3.18.0.2.6.116
GUID (グローバル・ユニーク ID)	b3afd65a-5c5b-11d3-b818-002035559151
特別な注意事項	1. *DB2Node* クラスは、 IBM Tivoli® Directory Server の下にある *eSap* オブジェクト・クラスと、Microsoft® Active Directory の下にある *ServiceConnectionPoint* オブジェクト・クラスから派生します。 2. *host* は IBM Tivoli Directory Server 環境で使用されます。*dNSHostName* 属性は Microsoft Active Directory で使用されます。 3. *protocolInformation* は IBM Tivoli Directory Server 環境のみで使用されます。 Microsoft Active Directory の場合は、ServiceConnectionPoint クラスを継承する *ServiceBindingInformation* 属性がプロトコル情報を収容するために使用されます。

DB2Node オブジェクトの *protocolInformation* 属性 (IBM Tivoli Directory Server で使用)、または *ServiceBindingInformation* 属性 (Microsoft Active Directory で使用) には、DB2 データベース・サーバーにバインドするための通信プロトコル情報が格納されます。これは、サポートされているネットワーク・プロトコルを記述するトークンから成ります。各トークンはセミコロンで区切られます。トークンとトークンの間にスペースはありません。アスタリスク (*) を使用して、オプション・パラメーターを指定できます。

TCP/IP のトークンは以下のとおりです。

- ″TCPIP″
- サーバーのホスト名または IP アドレス
- サービス名 (svcename) またはポート番号 (例: 50000)
- (任意指定) セキュリティー (″NONE″ または ″SOCKS″)

名前付きパイプのトークンは以下のとおりです。

- ″NPIPE″
- サーバーのコンピューター名
- サーバーのインスタンス名

表 14. *DB2Database*

クラス	**DB2Database**
Active Directory LDAP 表示名	ibm-db2Database
Active Directory 共通名 (cn)	ibm-db2Database
説明	DB2 データベースを表します。
SubClassOf	最上位
必須属性	db2databaseName db2nodePtr
オプション属性	db2databaseAlias db2additionalParameter db2ARLibrary db2authenticationLocation db2gwPtr db2databaseRelease DCEPrincipalName db2altgwPtr db2altnodePtr
タイプ	構造
OID (オブジェクト ID)	1.3.18.0.2.6.117
GUID (グローバル・ユニーク ID)	b3afd659-5c5b-11d3-b818-002035559151

表 15. *db2additionalParameters*

属性	**db2additionalParameters**
Active Directory LDAP 表示名	ibm-db2AdditionalParameters
Active Directory 共通名 (cn)	ibm-db2AdditionalParameters
説明	ホスト・データベース・サーバーへの接続時に使用する、任意の追加パラメーターを含んでいます。
構文	大文字小文字を無視したストリング
最大の長さ	1024
複合値	単一値
OID (オブジェクト ID)	1.3.18.0.2.4.426
GUID (グローバル・ユニーク ID)	b3afd315-5c5b-11d3-b818-002035559151

表 16. *db2authenticationLocation*

属性	**db2authenticationLocation**
Active Directory LDAP 表示名	ibm-db2AuthenticationLocation
Active Directory 共通名 (cn)	ibm-db2AuthenticationLocation
説明	認証が行われる場所を指定します。
構文	大文字小文字を無視したストリング

表 16. db2authenticationLocation (続き)

属性	db2authenticationLocation
最大の長さ	64
複合値	単一値
OID (オブジェクト ID)	1.3.18.0.2.4.425
GUID (グローバル・ユニーク ID)	b3afd317-5c5b-11d3-b818-002035559151
注意事項	有効な値は、CLIENT、SERVER、DCS、DCE、KERBEROS、SVRENCRYPT、または DCSENCRYPT です。

表 17. db2ARLibrary

属性	db2ARLibrary
Active Directory LDAP 表示名	ibm-db2ARLibrary
Active Directory 共通名 (cn)	ibm-db2ARLibrary
説明	アプリケーション・リクエスター・ライブラリーの名前
構文	大文字小文字を無視したストリング
最大の長さ	256
複合値	単一値
OID (オブジェクト ID)	1.3.18.0.2.4.427
GUID (グローバル・ユニーク ID)	b3afd316-5c5b-11d3-b818-002035559151

表 18. db2databaseAlias

属性	db2databaseAlias
Active Directory LDAP 表示名	ibm-db2DatabaseAlias
Active Directory 共通名 (cn)	ibm-db2DatabaseAlias
説明	データベース別名
構文	大文字小文字を無視したストリング
最大の長さ	1024
複合値	複合値
OID (オブジェクト ID)	1.3.18.0.2.4.422
GUID (グローバル・ユニーク ID)	b3afd318-5c5b-11d3-b818-002035559151

表 19. db2databaseName

属性	db2databaseName
Active Directory LDAP 表示名	ibm-db2DatabaseName
Active Directory 共通名 (cn)	ibm-db2DatabaseName
説明	データベース名
構文	大文字小文字を無視したストリング
最大の長さ	1024
複合値	単一値
OID (オブジェクト ID)	1.3.18.0.2.4.421
GUID (グローバル・ユニーク ID)	b3afd319-5c5b-11d3-b818-002035559151

表 20. *db2databaseRelease*

属性	**db2databaseRelease**
Active Directory LDAP 表示名	ibm-db2DatabaseRelease
Active Directory 共通名 (cn)	ibm-db2DatabaseRelease
説明	データベース・リリース番号
構文	大文字小文字を無視したストリング
最大の長さ	64
複合値	単一値
OID (オブジェクト ID)	1.3.18.0.2.4.429
GUID (グローバル・ユニーク ID)	b3afd31a-5c5b-11d3-b818-002035559151

表 21. *db2nodeAlias*

属性	**db2nodeAlias**
Active Directory LDAP 表示名	ibm-db2NodeAlias
Active Directory 共通名 (cn)	ibm-db2NodeAlias
説明	ノード別名
構文	大文字小文字を無視したストリング
最大の長さ	1024
複合値	複合値
OID (オブジェクト ID)	1.3.18.0.2.4.420
GUID (グローバル・ユニーク ID)	b3afd31d-5c5b-11d3-b818-002035559151

表 22. *db2nodeName*

属性	**db2nodeName**
Active Directory LDAP 表示名	ibm-db2NodeName
Active Directory 共通名 (cn)	ibm-db2NodeName
説明	ノード名
構文	大文字小文字を無視したストリング
最大の長さ	64
複合値	単一値
OID (オブジェクト ID)	1.3.18.0.2.4.419
GUID (グローバル・ユニーク ID)	b3afd31e-5c5b-11d3-b818-002035559151

表 23. *db2nodePtr*

属性	**db2nodePtr**
Active Directory LDAP 表示名	ibm-db2NodePtr
Active Directory 共通名 (cn)	ibm-db2NodePtr
説明	データベースを所有しているデータベース・サーバーを表す、ノード (DB2Node) オブジェクトへのポインター。
構文	識別名
最大の長さ	1000
複合値	単一値

表 23. db2nodePtr (続き)

属性	db2nodePtr
OID (オブジェクト ID)	1.3.18.0.2.4.423
GUID (グローバル・ユニーク ID)	b3afd31f-5c5b-11d3-b818-002035559151
特別な注意事項	このリレーションシップにより、クライアントは、データベースへ接続するためのプロトコル通信情報を取り出すことができます。

表 24. db2altnodePtr

属性	db2altnodePtr
Active Directory LDAP 表示名	ibm-db2AltNodePtr
Active Directory 共通名 (cn)	ibm-db2AltNodePtr
説明	代替データベース・サーバーを表すノード (DB2Node) オブジェクトへのポインター。
構文	識別名
最大の長さ	1000
複合値	複合値
OID (オブジェクト ID)	1.3.18.0.2.4.3093
GUID (グローバル・ユニーク ID)	5694e266-2059-4e32-971e-0778909e0e72

表 25. db2gwPtr

属性	db2gwPtr
Active Directory LDAP 表示名	ibm-db2GwPtr
Active Directory 共通名 (cn)	ibm-db2GwPtr
説明	データベースへのアクセス元となるゲートウェイ・サーバーを表す、ノード・オブジェクトへのポインター。
構文	識別名
最大の長さ	1000
複合値	単一値
OID (オブジェクト ID)	1.3.18.0.2.4.424
GUID (グローバル・ユニーク ID)	b3afd31b-5c5b-11d3-b818-002035559151

表 26. db2altgwPtr

属性	db2altgwPtr
Active Directory LDAP 表示名	ibm-db2AltGwPtr
Active Directory 共通名 (cn)	ibm-db2AltGwPtr
説明	代替ゲートウェイ・サーバーを表すノード・オブジェクトへのポインター。
構文	識別名
最大の長さ	1000
複合値	複合値
OID (オブジェクト ID)	1.3.18.0.2.4.3092
GUID (グローバル・ユニーク ID)	70ab425d-65cc-4d7f-91d8-084888b3a6db

表 27. *db2instanceName*

属性	**db2instanceName**
Active Directory LDAP 表示名	ibm-db2InstanceName
Active Directory 共通名 (cn)	ibm-db2InstanceName
説明	データベース・サーバー・インスタンスの名前
構文	大文字小文字を無視したストリング
最大の長さ	256
複合値	単一値
OID (オブジェクト ID)	1.3.18.0.2.4.428
GUID (グローバル・ユニーク ID)	b3afd31c-5c5b-11d3-b818-002035559151

表 28. *db2Type*

属性	**db2Type**
Active Directory LDAP 表示名	ibm-db2Type
Active Directory 共通名 (cn)	ibm-db2Type
説明	データベース・サーバーのタイプ
構文	大文字小文字を無視したストリング
最大の長さ	64
複合値	単一値
OID (オブジェクト ID)	1.3.18.0.2.4.418
GUID (グローバル・ユニーク ID)	b3afd320-5c5b-11d3-b818-002035559151
注意事項	有効なデータベース・サーバーのタイプは、 SERVER、MPP、および DCS です。

表 29. *DCEPrincipalName*

属性	**DCEPrincipalName**
Active Directory LDAP 表示名	ibm-DCEPrincipalName
Active Directory 共通名 (cn)	ibm-DCEPrincipalName
説明	DCE プリンシパル名
構文	大文字小文字を無視したストリング
最大の長さ	2048
複合値	単一値
OID (オブジェクト ID)	1.3.18.0.2.4.443
GUID (グローバル・ユニーク ID)	b3afd32d-5c5b-11d3-b818-002035559151

表 30. *cesProperty*

属性	**cesProperty**
Active Directory LDAP 表示名	ibm-cesProperty
Active Directory 共通名 (cn)	ibm-cesProperty

表 30. cesProperty (続き)

属性	cesProperty
説明	この属性の値を使用すれば、アプリケーション固有の設定に関する構成パラメーターを提供できます。たとえば、値に XML 形式のデータを含めることができます。この属性の値は、 cesPropertyType 属性値ではすべて同じ種類でなければなりません。
構文	大文字小文字を区別するストリング
最大の長さ	32700
複合値	複合値
OID (オブジェクト ID)	1.3.18.0.2.4.307
GUID (グローバル・ユニーク ID)	b3afd2d5-5c5b-11d3-b818-002035559151

表 31. cesPropertyType

属性	cesPropertyType
Active Directory LDAP 表示名	ibm-cesPropertyType
Active Directory 共通名 (cn)	ibm-cesPropertyType
説明	この属性の値を使用すれば、 cesProperty 属性のすべての値の構文、意味、その他の特性を記述できます。例えば、″XML″ という値を使用すると、 cesProperty 属性のすべての値が XML 構文としてエンコードされることを示します。
構文	大文字小文字を無視したストリング
最大の長さ	128
複合値	複合値
OID (オブジェクト ID)	1.3.18.0.2.4.308
GUID (グローバル・ユニーク ID)	b3afd2d6-5c5b-11d3-b818-002035559151

表 32. cisProperty

属性	cisProperty
Active Directory LDAP 表示名	ibm-cisProperty
Active Directory 共通名 (cn)	ibm-cisProperty
説明	この属性の値を使用すれば、アプリケーション固有の設定に関する構成パラメーターを提供できます。たとえば、値に INI ファイルを含めることができます。この属性の値は、 cisPropertyType 属性値ではすべて同じ種類でなければなりません。
構文	大文字小文字を無視したストリング
最大の長さ	32700
複合値	複合値
OID (オブジェクト ID)	1.3.18.0.2.4.309
GUID (グローバル・ユニーク ID)	b3afd2e0-5c5b-11d3-b818-002035559151

表 33. *cisPropertyType*

属性	**cisPropertyType**
Active Directory LDAP 表示名	ibm-cisPropertyType
Active Directory 共通名 (cn)	ibm-cisPropertyType
説明	この属性の値を使用すれば、 cisProperty 属性のすべての値の構文、意味、その他の特性を記述できます。例えば、″INI File″ という値を使用すると、 cisProperty 属性のすべての値が INI ファイルであることを示します。
構文	大文字小文字を無視したストリング
最大の長さ	128
複合値	複合値
OID (オブジェクト ID)	1.3.18.0.2.4.310
GUID (グローバル・ユニーク ID)	b3afd2e1-5c5b-11d3-b818-002035559151

表 34. *binProperty*

属性	**binProperty**
Active Directory LDAP 表示名	ibm-binProperty
Active Directory 共通名 (cn)	ibm-binProperty
説明	この属性の値を使用すれば、アプリケーション固有の設定に関する構成パラメーターを提供できます。たとえば、値にバイナリーでエンコードされた Lotus® 1-2-3 のプロパティーを含めることができます。この属性の値は、binPropertyType 属性値ではすべて同じ種類でなければなりません。
構文	バイナリー
最大の長さ	250000
複合値	複合値
OID (オブジェクト ID)	1.3.18.0.2.4.305
GUID (グローバル・ユニーク ID)	b3afd2ba-5c5b-11d3-b818-002035559151

表 35. *binPropertyType*

属性	**binPropertyType**
Active Directory LDAP 表示名	ibm-binPropertyType
Active Directory 共通名 (cn)	ibm-binPropertyType
説明	この属性の値を使用すれば、 binProperty 属性のすべての値の構文、意味、その他の特性を記述できます。例えば、″Lotus 123″ という値を使用すると、 binProperty 属性のすべての値がバイナリーでエンコードされた Lotus 1-2-3 のプロパティーであることを示します。
構文	大文字小文字を無視したストリング
最大の長さ	128
複合値	複合値
OID (オブジェクト ID)	1.3.18.0.2.4.306
GUID (グローバル・ユニーク ID)	b3afd2bb-5c5b-11d3-b818-002035559151

表 36. *PropertyType*

属性	PropertyType
Active Directory LDAP 表示名	ibm-propertyType
Active Directory 共通名 (cn)	ibm-propertyType
説明	この属性の値は、 eProperty オブジェクトの意味特性を記述します。
構文	大文字小文字を無視したストリング
最大の長さ	128
複合値	複合値
OID (オブジェクト ID)	1.3.18.0.2.4.320
GUID (グローバル・ユニーク ID)	b3afd4ed-5c5b-11d3-b818-002035559151

表 37. *settingID*

属性	settingID
Active Directory LDAP 表示名	適用されない
Active Directory 共通名 (cn)	適用されない
説明	命名属性。 eProperty などの cimSetting から派生したオブジェクト項目を示すために使用できます。
構文	大文字小文字を無視したストリング
最大の長さ	256
複合値	単一値
OID (オブジェクト ID)	1.3.18.0.2.4.325
GUID (グローバル・ユニーク ID)	b3afd596-5c5b-11d3-b818-002035559151

DB2 オブジェクト・クラスおよび属性を使用して LDAP ディレクトリー・スキーマを拡張する

LDAP ディレクトリー・スキーマは、オブジェクト・クラスおよび属性を定義し、情報を LDAP ディレクトリー項目に格納します。オブジェクト・クラスは、一連の必須および任意指定属性で構成されています。 LDAP ディレクトリーの各項目には、それぞれに関連付けられたオブジェクト・クラスがあります。

DB2 データベース・マネージャーが情報を LDAP へ格納する前に、LDAP サーバーのディレクトリー・スキーマには、DB2 データベース・システムで使用するオブジェクト・クラスと属性が含まれていなければなりません。新しいオブジェクト・クラスおよび属性を基本スキーマに追加する作業のことを、ディレクトリー・スキーマの拡張といいます。

注: IBM Tivoli Directory Server を使用している場合、DB2 UDB バージョン 8.1 またはそれ以前によって必要とされるオブジェクト・クラスおよび属性はすべて、基本スキーマに含まれます。この場合、DB2 オブジェクト・クラスと属性を使って基本スキーマを拡張する必要はありません。ただし、DB2 UDB バージョン 8.2 には、基本スキーマに含まれない新しい属性が 2 つあります。この場合、2 つの新しい DB2 データベース属性を使って基本スキーマを拡張する必要があります。

サポートされている LDAP クライアント構成およびサーバー構成

次の表では、サポートされている LDAP クライアント構成とサーバー構成をまとめています。

IBM Tivoli Directory Server は、LDAP バージョン 3 のサーバーであり、Windows、AIX、Solaris、Linux、HP-UX で使用できます。AIX と System i™ では基本オペレーティング・システムの中に組み込まれており、OS/390® Security Server にも用意されています。

DB2 データベースは、AIX、Solaris、HP-UX 11.11、Windows、Linux で IBM LDAP クライアントをサポートしています。

Microsoft Active Directory Server は、LDAP バージョン 3 のサーバーであり、Windows 2000 Server と Windows Server 2003 のオペレーティング・システム・ファミリーに組み込まれています。

Microsoft LDAP クライアントは、Windows オペレーティング・システムに組み込まれています。

表 38.

サポートされている LDAP クライアント構成およびサーバー構成	IBM Tivoli Directory Server	Microsoft Active Directory Server	Sun One LDAP サーバー
IBM LDAP クライアント	サポートされている	サポートされている	サポートされている
Microsoft LDAP/ADSI クライアント	サポートされている	サポートされている	サポートされている

注: Windows オペレーティング・システムで実行されている場合、DB2 データベース・マネージャーは、IBM LDAP クライアントまたは Microsoft LDAP クライアントの使用をサポートします。 IBM LDAP クライアントを明示的に選択するには、db2set コマンドを使用して DB2LDAP_CLIENT_PROVIDER レジストリー変数を ″IBM″ に設定します。 Microsoft LDAP クライアントは、Windows オペレーティング・システムに組み込まれています。

LDAP サポートと DB2 Connect

LDAP サポートがDB2 Connect ゲートウェイにおいて有効で、データベースがゲートウェイ・データベース・ディレクトリーでは見つからなかった場合、DB2 データベース・システムは LDAP を検索し、見つかった情報を保持しようとします。

LDAP でのホスト・データベースの登録

ホスト・データベースを LDAP で登録するときは、2 とおりの構成が可能です。つまり、ホスト・データベースに直接接続する構成と、ゲートウェイを経由してホスト・データベースに接続する構成です。

ホスト・データベースへの直接接続の場合は、ホスト・サーバーを LDAP に登録してから、そのホスト・サーバーのノード名を指定してホスト・データベースを

LDAP でカタログします。ゲートウェイを経由してホスト・データベースに接続する場合は、ゲートウェイ・サーバーを LDAP に登録してから、そのゲートウェイ・サーバーのノード名を指定して LDAP でホスト・データベースをカタログします。

LDAP サポートが DB2 Connect ゲートウェイにおいて有効で、データベースがゲートウェイ・データベース・ディレクトリーでは見つからなかった場合、 DB2 データベース・システムは LDAP を検索し、見つかった情報を保持しようとします。

上記の 2 つの場合の例として、次のような状況を考慮してみましょう。
NIAGARA_FALLS という名前のホスト・データベースがあるとします。このホスト・データベースは、TCP/IP を使用して着信接続を受け入れることができます。あるクライアントが DB2 Connect をインストールしていないためにそのホストへ直接接続できない場合は、 goto@niagara という名前のゲートウェイを使用してホストに接続します。

以下のステップを実行する必要があります。

1. TCP/IP 接続を可能にするため、ホスト・データベース・サーバーを LDAP に登録します。このサーバーの TCP/IP ホスト名は ″myhost″、ポート番号は ″446″ です。ステップ 1 と同じように、NODETYPE 節を ″DCS″ に設定して、これがホスト・データベース・サーバーであることを示します。

   ```
   db2 register ldap as nftcpip tcpip hostname myhost svcename 446
      remote mvssys instance mvsinst nodetype dcs
   ```

2. TCP/IP 接続を可能にするため、DB2 Connect ゲートウェイ・サーバーを LDAP に登録します。このゲートウェイ・サーバーの TCP/IP ホスト名は ″niagara″、ポート番号は ″50000″ です。

   ```
   db2 register ldap as whasf tcpip hostname niagara svcename 50000
      remote niagara instance goto nodetype server
   ```

3. TCP/IP 接続を使用してホスト・データベースを LDAP でカタログします。このホストのデータベース名は ″NIAGARA_FALLS″、データベース別名は ″nftcpip″ です。 GWNODE 節は、DB2 Connect ゲートウェイ・サーバーのノード名を指定するために使用されています。

   ```
   db2 catalog ldap database NIAGARA_FALLS as nftcpip at node nftcpip
      gwnode whasf authentication dcs
   ```

上記のように登録とカタログが完了した後、 TCP/IP を使用してホストに接続する場合は、″nftcpip″ に接続します。 DB2 Connect がクライアント・ワークステーションにインストールされていない場合、接続は TCP/IP によってゲートウェイ経由で確立されます。ゲートウェイからは、TCP/IP を使用してホストに接続します。

通常は、LDAP においてホスト・データベース情報を手動で構成できるので、それぞれのクライアントでは、各マシンのデータベースおよびノードをローカルに手動でカタログする必要はありません。次のように処理します。

1. ホスト・データベース・サーバーを LDAP に登録します。そのためには、ホスト・データベース・サーバーのリモート・コンピューター名、インスタンス名、およびノード・タイプを、REGISTER コマンドの REMOTE 節、INSTANCE 節、および NODETYPE 節にそれぞれ指定しなければなりません。 REMOTE 節は、ホスト・サーバー・マシンのホスト名と LU 名のどちらに設定しても構いません。 INSTANCE 節は、8 文字以下の文字ストリングで設定します。 (例えば、インスタンス名を ″DB2″ のように設定します。) NODE TYPE 節は必ず

″DCS″ に設定して、これがホスト・データベース・サーバーであることを示さなければなりません。ホスト・データベースを LDAP に登録します。

2. CATALOG LDAP DATABASE コマンドを使ってホスト・データベースを LDAP に登録します。さらに、PARMS 節を使用して、任意の DRDA® パラメーターを指定することができます。データベース認証タイプは ″DCS″ に設定してください。

IBM Tivoli Directory Server 用にディレクトリー・スキーマを拡張する

IBM Tivoli Directory Server を使用している場合、バージョン 8.2 より前の DB2 データベースで必要とされるオブジェクト・クラスおよび属性はすべて、基本スキーマに含まれます。

バージョン 8.2 に含まれる新しい DB2 データベース属性を使って基本スキーマを拡張するには、以下を実行します。

```
ldapmodify -c -h <machine_name>:389 -D <dn> -w <password> -f altgwnode.ldif
```

以下は、altgwnode.ldif ファイルの内容です。

```
dn:                cn=schema
changetype:        modify
add:               attributetypes
attributetypes:    (
 1.3.18.0.2.4.3092
 NAME 'db2altgwPtr'
 DESC 'DN pointer to DB2 alternate gateway (node) object'
 SYNTAX 1.3.6.1.4.1.1466.115.121.1.12)
-
add:               ibmattributetypes
ibmattributetypes: (
 1.3.18.0.2.4.3092
 DBNAME ('db2altgwPtr' 'db2altgwPtr')
 ACCESS-CLASS NORMAL
 LENGTH 1000)


dn:                cn=schema
changetype:        modify
add:               attributetypes
attributetypes:    (
 1.3.18.0.2.4.3093
 NAME 'db2altnodePtr'
 DESC 'DN pointer to DB2 alternate node object'
 SYNTAX 1.3.6.1.4.1.1466.115.121.1.12)
-
add:               ibmattributetypes
ibmattributetypes: (
 1.3.18.0.2.4.3093
 DBNAME ('db2altnodePtr' 'db2altnodePtr')
 ACCESS-CLASS NORMAL
 LENGTH 1000)


dn:            cn=schema
changetype:    modify
replace:       objectclasses
objectclasses: (
 1.3.18.0.2.6.117
 NAME 'DB2Database'
 DESC 'DB2 database'
 SUP cimSetting
 MUST ( db2databaseName $ db2nodePtr )
 MAY ( db2additionalParameters $ db2altgwPtr $ db2altnodePtr
    $ db2ARLibrary $ db2authenticationLocation $ db2databaseAlias
    $ db2databaseRelease $ db2gwPtr $ DCEPrincipalName ) )
```

altgwnode.ldif および altgwnode.readme ファイルは、 ftp://ftp.software.ibm.com/ps/products/db2/tools/ldap の URL にあります。

DB2 スキーマ定義を追加した後、すべての変更を有効にするために、 Directory Server を再始動する必要があります。

Netscape LDAP ディレクトリー・サポートおよび属性定義

Netscape LDAP Server のサポート・レベルは、バージョン 4.12 以降です。

Netscape LDAP Server バージョン 4.12 以降では、 Netscape Directory Server を使用すると、アプリケーションは属性およびオブジェクト・クラス定義を 2 つのファイル slapd.user_oc.conf および slapd.user_at.conf に追加することによって、スキーマ

を拡張できます。この 2 つのファイルは、<Netscape_install path>¥slapd-<machine_name>¥config ディレクトリーにあります。

注: Sun One Directory Server 5.0 を使用している場合、Sun One Directory Server のためのディレクトリー・スキーマの拡張に関するトピックを参照してください。

以下のようにして、 DB2 属性を slapd.user_at.conf に追加する必要があります。

```
###############################################################################
#
# IBM DB2 Database
# Attribute Definitions
#
# bin -> binary
# ces -> case exact string
# cis -> case insensitive string
# dn -> distinguished name
#
###############################################################################

attribute binProperty                    1.3.18.0.2.4.305     bin
attribute binPropertyType                1.3.18.0.2.4.306     cis
attribute cesProperty                    1.3.18.0.2.4.307     ces
attribute cesPropertyType                1.3.18.0.2.4.308     cis
attribute cisProperty                    1.3.18.0.2.4.309     cis
attribute cisPropertyType                1.3.18.0.2.4.310     cis
attribute propertyType                   1.3.18.0.2.4.320     cis
attribute systemName                     1.3.18.0.2.4.329     cis
attribute db2nodeName                    1.3.18.0.2.4.419     cis
attribute db2nodeAlias                   1.3.18.0.2.4.420     cis
attribute db2instanceName                1.3.18.0.2.4.428     cis
attribute db2Type                        1.3.18.0.2.4.418     cis
attribute db2databaseName                1.3.18.0.2.4.421     cis
attribute db2databaseAlias               1.3.18.0.2.4.422     cis
attribute db2nodePtr                     1.3.18.0.2.4.423     dn
attribute db2gwPtr                       1.3.18.0.2.4.424     dn
attribute db2additionalParameters        1.3.18.0.2.4.426     cis
attribute db2ARLibrary                   1.3.18.0.2.4.427     cis
attribute db2authenticationLocation      1.3.18.0.2.4.425     cis
attribute db2databaseRelease             1.3.18.0.2.4.429     cis
attribute DCEPrincipalName               1.3.18.0.2.4.443     cis
```

以下のようにして、 DB2 オブジェクト・クラスを slapd.user_oc.conf ファイルに追加する必要があります。

```
###############################################################################
#
# IBM DB2 Database
# Object Class Definitions
#
###############################################################################

objectclass eProperty
        oid 1.3.18.0.2.6.90
        requires
                objectClass
        allows
                cn,
                propertyType,
                binProperty,
                binPropertyType,
                cesProperty,
                cesPropertyType,
                cisProperty,
                cisPropertyType
```

```
objectclass eApplicationSystem
        oid 1.3.18.0.2.6.84
        requires
                objectClass,
                systemName


objectclass DB2Node
        oid 1.3.18.0.2.6.116
        requires
                objectClass,
                db2nodeName
        allows
                db2nodeAlias,
                host,
                db2instanceName,
                db2Type,
                description,
                protocolInformation

objectclass DB2Database
        oid 1.3.18.0.2.6.117
        requires
                objectClass,
                db2databaseName,
                db2nodePtr
        allows
                db2databaseAlias,
                description,
                db2gwPtr,
                db2additionalParameters,
                db2authenticationLocation,
                DCEPrincipalName,
                db2databaseRelease,
                db2ARLibrary
```

DB2 スキーマ定義を追加した後、すべての変更を有効にするために、 Directory Server を再始動する必要があります。

Sun One Directory Server 用にディレクトリー・スキーマを拡張する

Sun One Directory Server は、 Netscape または iPlanet ディレクトリー・サーバーとしても知られています。

ご使用の環境で Sun One Directory Server を利用するには、 60ibmdb2.ldif ファイルを以下のディレクトリー・サーバーに追加してください。

Windows では、iPlanet が C:¥iPlanet¥Servers にインストール済みの場合、上記のファイルを .¥sldap-<machine_name>¥config¥schema に追加します。

UNIX では、 iPlanet が /usr/iplanet/servers にインストール済みの場合、上記のファイルを ./slapd-<machine_name>/config/schema に追加します。

以下は、このファイルの内容です。

```
##############################################################################
# IBM DB2 Database
##############################################################################
dn: cn=schema
##############################################################################
```

```
# Attribute Definitions (Before V8.2)
########################################################################
attributetypes: ( 1.3.18.0.2.4.305  NAME 'binProperty'
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.5              X-ORIGIN 'IBM DB2' )
attributetypes: ( 1.3.18.0.2.4.306  NAME 'binPropertyType'
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.15             X-ORIGIN 'IBM DB2' )
attributetypes: ( 1.3.18.0.2.4.307  NAME 'cesProperty'
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.15             X-ORIGIN 'IBM DB2' )
attributetypes: ( 1.3.18.0.2.4.308  NAME 'cesPropertyType'
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.15             X-ORIGIN 'IBM DB2' )
attributetypes: ( 1.3.18.0.2.4.309  NAME 'cisProperty'
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.15             X-ORIGIN 'IBM DB2' )
attributetypes: ( 1.3.18.0.2.4.310  NAME 'cisPropertyType'
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.15             X-ORIGIN 'IBM DB2' )
attributetypes: ( 1.3.18.0.2.4.320  NAME 'propertyType'
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.15             X-ORIGIN 'IBM DB2' )
attributetypes: ( 1.3.18.0.2.4.329  NAME 'systemName'
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 SINGLE-VALUE X-ORIGIN 'IBM DB2' )
attributetypes: ( 1.3.18.0.2.4.419  NAME 'db2nodeName'
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 SINGLE-VALUE X-ORIGIN 'IBM DB2' )
attributetypes: ( 1.3.18.0.2.4.420  NAME 'db2nodeAlias'
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.15             X-ORIGIN 'IBM DB2' )
attributetypes: ( 1.3.18.0.2.4.428  NAME 'db2instanceName'
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 SINGLE-VALUE X-ORIGIN 'IBM DB2' )
attributetypes: ( 1.3.18.0.2.4.418  NAME 'db2Type'
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 SINGLE-VALUE X-ORIGIN 'IBM DB2' )
attributetypes: ( 1.3.18.0.2.4.421  NAME 'db2databaseName'
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 SINGLE-VALUE X-ORIGIN 'IBM DB2' )
attributetypes: ( 1.3.18.0.2.4.422  NAME 'db2databaseAlias'
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.15             X-ORIGIN 'IBM DB2' )
attributetypes: ( 1.3.18.0.2.4.426  NAME 'db2additionalParameters'
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 SINGLE-VALUE X-ORIGIN 'IBM DB2' )
attributetypes: ( 1.3.18.0.2.4.427  NAME 'db2ARLibrary'
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 SINGLE-VALUE X-ORIGIN 'IBM DB2' )
attributetypes: ( 1.3.18.0.2.4.425  NAME 'db2authenticationLocation'
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 SINGLE-VALUE X-ORIGIN 'IBM DB2' )
attributetypes: ( 1.3.18.0.2.4.429  NAME 'db2databaseRelease'
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 SINGLE-VALUE X-ORIGIN 'IBM DB2' )
attributetypes: ( 1.3.18.0.2.4.443  NAME 'DCEPrincipalName'
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 SINGLE-VALUE X-ORIGIN 'IBM DB2' )
attributetypes: ( 1.3.18.0.2.4.423  NAME 'db2nodePtr'
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.12 SINGLE-VALUE X-ORIGIN 'IBM DB2' )
attributetypes: ( 1.3.18.0.2.4.424  NAME 'db2gwPtr'
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.12 SINGLE-VALUE X-ORIGIN 'IBM DB2' )
########################################################################
# Attribute Definitions (V8.2 and later)
########################################################################
attributetypes: ( 1.3.18.0.2.4.3092 NAME 'db2altgwPtr'
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.12             X-ORIGIN 'IBM DB2' )
attributetypes: ( 1.3.18.0.2.4.3093 NAME 'db2altnodePtr'
   SYNTAX 1.3.6.1.4.1.1466.115.121.1.12             X-ORIGIN 'IBM DB2' )
########################################################################
# Object Class Definitions
# DB2Database for V8.2 has the above two new optional attributes.
########################################################################
objectClasses: ( 1.3.18.0.2.6.90  NAME 'eProperty'
   SUP top STRUCTURAL MAY ( cn $ propertyType $ binProperty
   $ binPropertyType $ cesProperty $ cesPropertyType $ cisProperty
   $ cisPropertyType ) X-ORIGIN 'IBM DB2' )
objectClasses: ( 1.3.18.0.2.6.84  NAME 'eApplicationSystem'
   SUP top STRUCTURAL MUST systemName
   X-ORIGIN 'IBM DB2' )
objectClasses: ( 1.3.18.0.2.6.116 NAME 'DB2Node'
   SUP top STRUCTURAL MUST db2nodeName MAY ( db2instanceName $ db2nodeAlias
   $ db2Type $ description $ host $ protocolInformation )
   X-ORIGIN 'IBM DB2' )
objectClasses: ( 1.3.18.0.2.6.117 NAME 'DB2Database'
```

```
SUP top STRUCTURAL MUST (db2databaseName $ db2nodePtr ) MAY
( db2additionalParameters $ db2altgwPtr $ db2altnodePtr $ db2ARLibrary
$ db2authenticationLocation $ db2databaseAlias $ db2databaseRelease
$ db2gwPtr $ DCEPrincipalName $ description )
X-ORIGIN 'IBM DB2' )
```

60ibmdb2.ldif および 60ibmdb2.readme ファイルは、 ftp://ftp.software.ibm.com/ps/products/db2/tools/ldap の URL にあります。

DB2 スキーマ定義を追加した後、すべての変更を有効にするために、 Directory Server を再始動する必要があります。

Windows Active Directory

Active Directory のサポート

DB2 データベース・サーバーは、 ibm_db2Node オブジェクトとして Active Directory に公開されます。 ibm_db2Node オブジェクト・クラスは、ServiceConnectionPoint (SCP) オブジェクト・クラスのサブクラスです。

各 ibm_db2Node オブジェクトにはプロトコル構成情報が含まれていて、クライアント・アプリケーションが DB2 データベース・サーバーへ接続できるようになっています。新しいデータベースが作成されると、そのデータベースは ibm_db2Node の下にある ibm_db2Database オブジェクトとして、 Active Directory に公開されます。

リモート・データベースに接続するときは、DB2 クライアントは LDAP インターフェースを使用して、Active Directory に対して ibm_db2Database オブジェクトについての照会を行います。データベース・サーバーへ接続するためのプロトコル通信 (バインド情報) は、 ibm_db2Database オブジェクトが作成された ibm_db2Node オブジェクトから取得されます。

ibm_db2Node および ibm_db2Database オブジェクトのプロパティー・ページは、ドメイン・コントローラーの *Active Directory* ユーザーとコンピュータ 管理コンソール (MMC) を使用して、表示または変更が可能です。プロパティー・ページをセットアップするには、以下のように regsrv32 コマンドを実行して、DB2 オブジェクトのためのプロパティー・ページを登録します。

```
regsvr32 %DB2PATH%¥bin¥db2ads.dll
```

オブジェクトは、ドメイン・コントローラーの *Active Directory* ユーザーとコンピュータ 管理コンソール (MMC) を使用して表示することができます。この管理ツールは、「スタート」->「プログラム」->「管理ツール」 ->「Active Directory ユーザーとコンピュータ」にあります。

注: コンピューター・オブジェクトの下の DB2 データベース・オブジェクトを表示するには、「表示」メニューから *Users, Groups, and Computers as containers* を選択する必要があります。

注: DB2 データベース・システムがドメイン・コントローラーにインストールされていない場合でも、ドメイン・コントローラーのローカル・ディレクトリーに、%DB2PATH%¥bin から db2ads.dll ファイルを、また %DB2PATH%¥msg¥locale-name からリソース DLL db2adsr.dll をコピーすることで、DB2 データベース・オブジェ

クトのプロパティー・ページを表示できます。 (コピーされたこの 2 つのファイルの保管場所は、PATH 環境変数に含まれるいずれかのディレクトリーでなければなりません。) その後、regsvr32 コマンドをローカル・ディレクトリーから実行し、DLL を登録します。

Active Directory を使用するよう DB2 データベース・マネージャーを構成する

Microsoft Active Directory へアクセスするためには、以下の条件を満たしていなければなりません。

1. DB2 データベースを実行するマシンは、 Windows 2000 または Windows Server 2003 ドメインに属している必要がある。
2. Microsoft LDAP クライアントがインストールされている。 Microsoft LDAP クライアントは、Windows 2000、Windows XP、および Windows Server 2003 オペレーティング・システムに含まれています。
3. LDAP サポートが有効になっている。Windows 2000、Windows XP、または Windows Server 2003 の場合、 LDAP サポートはインストール・プログラムによって有効になります。
4. DB2 データベース・システムを実行して Active Directory から情報を読み取るときは、ドメイン・ユーザー・アカウントにログオンする。

Active Directory のセキュリティーに関する考慮事項

Active Directory において、DB2 データベースとノードの各オブジェクトは、 DB2 サーバーがインストールされているマシンのコンピューター・オブジェクトの下に作成されます。データベース・サーバーを Active Directory に登録したり、データベースのカタログを Active Directory に作成するためには、コンピューター・オブジェクトの下でオブジェクトを作成または更新できるアクセス権限が必要です。

デフォルトでは、コンピューター・オブジェクトの下にあるオブジェクトは、認証を受けたすべてのユーザーが読み取ることができます。管理者 (Administrators、Domain Administrators、および Enterprise Administrators グループに属するユーザー) であれば更新することもできます。特定のユーザーまたはグループにアクセス権限を付与するには、 *Active Directory* ユーザーとコンピュータ 管理コンソール (MMC) を次のようにして使います。

1. *Active Directory* ユーザーとコンピュータ 管理ツールを開始します。

   (「スタート」->「プログラム」->「管理ツール」->「Active Directory ユーザーとコンピュータ」)
2. 「*表示 (View)*」から「*詳細 (Advanced Features)*」を選択します。
3. 「*コンピューター (Computers)*」コンテナーを選択します。
4. DB2 がインストールされているサーバー・マシンを表すコンピューター・オブジェクトの上を右クリックし、「*プロパティー (Properties)*」を選択します。
5. 「*セキュリティー (Security)*」タブを選択し、指定したユーザーまたはグループに必要なアクセス権限を追加します。

ユーザー・レベルの DB2 レジストリー変数および CLI 設定は、ユーザー・オブジェクトの下にある DB2 プロパティー・オブジェクトで保守されます。 DB2 レジス

トリー変数または CLI 設定をユーザー・レベルで設定するためには、そのユーザーに「ユーザー (User)」オブジェクトの下にオブジェクトを作成できるアクセス権限が必要です。

デフォルトでは、「ユーザー (User)」オブジェクトの下にオブジェクトを作成できるのは管理者だけです。 DB2 レジストリー変数または CLI 設定をユーザー・レベルで設定できるアクセス権限をユーザーに付与するには、*Active Directory* ユーザーとコンピュータ 管理コンソール (MMC) を次のようにして使います。

1. *Active Directory* ユーザーとコンピュータ 管理ツールを開始します。

   (「スタート」->「プログラム」->「管理ツール」->「Active Directory ユーザーとコンピュータ」)
2. 「ユーザー (Users)」コンテナーの下にあるユーザー・オブジェクトを選択します。
3. ユーザー・オブジェクトの上を右クリックし、「プロパティー *(Properties)*」を選択します。
4. 「セキュリティー *(Security)*」タブを選択します。
5. 「追加 (Add)」ボタンを使用して、ユーザー名をリストに追加します。
6. 「書き込み (Write)」および「すべての子オブジェクトの作成 (Create All Child Objects)」アクセスを付与します。
7. 「拡張 (Advanced)」設定を使用して、「このオブジェクトとすべての子オブジェクト (This object and all child objects)」に適用される許可を設定します。
8. 「継承可能な許可を親からこのオブジェクトへ継承するのを許可する (Allow inheritable permissions from parent to propagate to this object)」チェック・ボックスを選択します。

Active Directory 内の DB2 オブジェクト

DB2 データベース・マネージャーは、Active Directory 内の以下の 2 箇所にオブジェクトを作成します。

1. DB2 データベースおよびノード・オブジェクトが、 DB2 サーバーがインストールされているマシンのコンピューター・オブジェクトの下に作成されます。 Windows ドメインに属していない DB2 サーバー・マシンの場合は、 DB2 データベースおよびノード・オブジェクトは「システム (System)」コンテナーの下に作成されます。
2. ユーザー・レベルの DB2 レジストリー変数および CLI 設定が、「ユーザー (User)」オブジェクトの下にある DB2 プロパティー・オブジェクトに保管されます。これらのオブジェクトには、そのユーザーに固有の情報が入ります。

Active Directory 用にディレクトリー・スキーマを拡張する

DB2 データベース・マネージャーが Active Directory に情報を保管できるようにするためには、ディレクトリー・スキーマを拡張して、新しい DB2 データベース・オブジェクト・クラスおよび属性を組み込む必要があります。新しいオブジェクト・クラスおよび属性をディレクトリー・スキーマに追加する作業のことを、スキーマの拡張 といいます。

Windows ドメインに属するマシンに DB2 データベース・システムを初めてインストールする前に、 DB2 スキーマ・インストール・プログラム db2schex を実行して、 Active Directory のスキーマを拡張しておく必要があります。

db2schex プログラムは、製品 CD-ROM の x:¥db2¥windows¥utilities¥ という場所にあります (x: は CD-ROM ドライブです)。

コマンドは、次のように使用します。

```
db2schex
```

他にも、このコマンドに関連する以下のような任意指定の節があります。

-b UserDN
ユーザーの識別名を指定します。

-w Password
バインド・パスワードを指定します。

-u スキーマをアンインストールします。

-k エラーを無視して、アンインストールを強制的に続行します。

注:

1. UserDN とパスワードを指定しなかった場合、 db2schex は現在ログオンしているユーザーでバインドされます。
2. userDN 節には Windows ユーザー名を指定できます。
3. スキーマを更新するためには、Schema Administrators グループのメンバーであるか、スキーマを更新するための権限を委任されている必要があります。

ディレクトリー・スキーマを拡張するには、DB2 UDB バージョン 8.2 以降に用意されている db2schex コマンドを実行する必要があります。

DB2 データベース管理システムの以前のバージョンに用意されていた db2schex コマンドを既に実行していた場合は、その同じコマンドを DB2 UDB バージョン 8.2 以降で再び実行すると、以下の 2 つのオプション属性が ibm-db2Database クラスに追加されます。

```
ibm-db2AltGwPtr
ibm-db2NodePtr
```

Windows の DB2 データベース管理システムの以前のバージョンで db2schex コマンドを実行していなかった場合は、その同じコマンドを DB2 UDB バージョン 8.2 以降で実行すると、DB2 データベース・システムの LDAP サポートのためのすべてのクラスと属性が追加されます。

次に例を示します。

- DB2 データベース・スキーマをインストールする方法:

  ```
  db2schex
  ```

- DB2 データベース・スキーマをインストールし、バインド DN とパスワードを指定する方法:

  ```
  db2schex -b "cn=A Name,dc=toronto1,dc=ibm,dc=com"
           -w password
  ```

または

```
db2schex -b Administrator -w password
```

- DB2 データベース・スキーマをアンインストールする方法:

```
db2schex -u
```

- DB2 データベース・スキーマをアンインストールし、エラーを無視する方法:

```
db2schex -u -k
```

Active Directory 用の DB2 スキーマ・インストール・プログラムは、以下のタスクを実行します。

1. どのサーバーが Schema Master であるかを検出します。
2. Schema Master となっているドメイン・コントローラーにバインドします。
3. スキーマにクラスや属性を追加する権限をユーザーに付与します。
4. Schema Master を書き込み可能にします (つまり、レジストリー内の安全インターロックを取り除きます)。
5. すべての新しい属性を作成します。
6. すべての新しいオブジェクト・クラスを作成します。
7. エラーがあるかどうか検出し、エラーが発生している場合には、スキーマに加えられた変更をロールバックするようプログラムに指示します。

インストールが完了した後に LDAP サポートを使用可能にする

インストールの完了後に、LDAP サポートを使用可能にする必要があります。

各マシンで以下の手順を使用します。

1. LDAP サポート・バイナリー・ファイルをインストールするには、セットアップ・プログラムを実行し、カスタム・インストールで LDAP Directory Exploitation サポートを選択します。 セットアップ・プログラムによりバイナリー・ファイルがインストールされ、 DB2 プロファイルのレジストリー変数 DB2_ENABLE_LDAP が ″YES″ に設定されます。

   注: Windows および UNIX プラットフォームの場合、db2set コマンドを使用して DB2_ENABLE_LDAP レジストリー変数を ″YES″ に設定することにより、LDAP を明示的に有効にしなければなりません。
2. UNIX プラットフォームのみ: LDAP サーバーの TCP/IP ホスト名を宣言します。オプションとして、ポート番号を宣言することもできます。そのためのコマンドは、`db2set DB2LDAPHOST=<base_domain_name>[:port_number]` です (base_domain_name は LDAP サーバーの TCP/IP ホスト名、[:port_number] はポート番号です)。ポート番号を指定しない場合は、デフォルトの LDAP ポート (389) が使用されます。

   DB2 オブジェクトの探索は、LDAP 基本識別名 (baseDN) によって行われます。DB2SET コマンドを使用することにより、各マシンで LDAP 基本識別名を構成することができます。

   ```
   db2set DB2LDAP_BASEDN=<baseDN>
   ```

   ここで、baseDN は LDAP サーバーに定義されている LDAP 接尾部の名前です。この LDAP 接尾部は DB2 オブジェクトを収容するのに使用されます。

3. REGISTER LDAP AS コマンドを使用し、 DB2 サーバーの現在のインスタンスを LDAP に登録します。 例:

   ```
   db2 register ldap as <node-name> protocol tcpip
   ```

4. LDAP に登録するデータベースがあれば、CATALOG LDAP DATABASE コマンドを実行します。 例:

   ```
   db2 catalog ldap database <dbname> as <alias_dbname>
   ```

5. LDAP ユーザーの識別名 (DN) とパスワードを入力します。ただし、これらが必要になるのは、 DB2 ユーザー固有情報を保管するのに LDAP を使用する計画がある場合だけです。

IBM LDAP 環境での DB2 の構成

DB2 を IBM LDAP 環境で使用できるようにするには、各マシンで次の設定を構成しなければなりません。

- LDAP サポートが有効になっている。 Windows の場合、 LDAP サポートはインストール・プログラムによって有効になります。すべての Windows オペレーティング・システムは、デフォルトで Microsoft の LDAP クライアントを使用します。IBM LDAP クライアントを使用する場合には、 db2set コマンドを使用して、 DB2LDAP_CLIENT_PROVIDER レジストリー変数を「IBM」に設定する必要があります。
- LDAP サーバーの TCP/IP ホスト名とポート番号を構成する。これらの値は DB2LDAPHOST 応答キーワードを使用して自動インストール時に入力することもできますし、 db2set コマンドを使用して後から手動設定することもできます。

  ```
  db2set DB2LDAPHOST=<hostname[:port]>
  ```

  ここで、hostname は LDAP サーバーの TCP/IP ホスト名、[:port] はポート番号です。ポート番号が指定されなかった場合、 DB2 はデフォルトの LDAP ポート (389) を使用します。

  DB2 オブジェクトの探索は、LDAP 基本識別名 (baseDN) によって行われます。db2set コマンドを使用することにより、各マシンで LDAP 基本識別名を構成することができます。

  ```
  db2set DB2LDAP_BASEDN=<baseDN>
  ```

  ここで、baseDN は LDAP サーバーに定義されている LDAP 接尾部の名前です。この LDAP 接尾部は DB2 オブジェクトを収容するのに使用されます。
- LDAP ユーザーの識別名 (DN) とパスワードを構成する必要があります。ただし、これらが必要になるのは、 DB2 ユーザー固有情報を保管するのに LDAP を使用する計画がある場合だけです。

LDAP 項目の登録

インストール後の DB2 サーバーの登録

DB2 サーバー・インスタンスに接続するためにクライアント・アプリケーションが使用するプロトコル構成情報を公開するには、各 DB2 サーバー・インスタンスを LDAP に登録しておく必要があります。

データベース・サーバーのインスタンスを登録するときには、*ノード名* を指定する必要があります。このノード名は、クライアント・アプリケーションがサーバーに接続またはアタッチするときに、そのクライアント・アプリケーションによって使われます。 CATALOG LDAP NODE コマンドを使用して、もう一つ、 LDAP ノードの別名をカタログすることができます。

注: Windows 2000 ドメイン環境で作業している場合は、インストール時に DB2 サーバー・インスタンスが、次の情報とともに、自動的に Active Directory へ登録されます。

```
nodename: TCP/IP hostname
protocol type: TCP/IP
```

TCP/IP ホスト名が 9 文字以上の場合は、8 文字に切り捨てられます。

REGISTER コマンドは次のようになります。

```
db2 register db2 server in ldap
   as <ldap_node_name>
   protocol tcpip
```

protocol 節には、このデータベース・サーバーに接続するときに使用する通信プロトコルを指定します。

複数の物理マシンを含む DB2 Enterprise Server Edition のインスタンスを作成する場合、それぞれのマシンで一度 REGISTER コマンドを呼び出す必要があります。rah コマンドは、すべてのマシンで REGISTER コマンドを発行する際に使用します。

注: LDAP では名前は固有でなければならないので、各マシンに同じ ldap_node_name を使うことはできません。 REGISTER コマンドでは、各マシンのホスト名を、 ldap_node_name に置き換えることができます。例:

```
rah ">DB2 REGISTER DB2 SERVER IN LDAP AS <> PROTOCOL TCPIP"
```

″<>″ には、rah コマンドを実行している各マシンのホスト名が入ります。 めったにありませんが、複数の DB2 Enterprise Server Edition のインスタンスが存在する場合、rah コマンドでは、インスタンスとホスト索引の組み合わせをノード名として使用することができます。

REGISTER コマンドは、リモート DB2 サーバーに対して発行できます。そのためには、リモート・サーバーの登録時に、リモート・コンピューター名、インスタンス名、およびプロトコル構成パラメーターを指定しなければなりません。このコマンドは、次のように使います。

```
db2 register db2 server in ldap
   as <ldap_node_name>
   protocol tcpip
   hostname <host_name>
   svcename <tcpip_service_name>
   remote <remote_computer_name>
   instance <instance_name>
```

コンピューター名には、次の規則が使われます。

- TCP/IP を構成する場合、コンピューター名は TCP/IP ホスト名と同じでなければならない。

高可用性の環境あるいはフェイルオーバーの環境で実行し、通信プロトコルとして TCP/IP を使用する場合、クラスターの IP アドレスを使わなければなりません。クラスターの IP アドレスを使用するなら、クライアントは、マシンごとに個別の TCP/IP ノードをカタログすることなく、いずれかのマシンでサーバーに接続することができます。このクラスターの IP アドレスは、次に示すように、 hostname 節を使って指定します。

```
db2 register db2 server in ldap
   as <ldap_node_name>
   protocol tcpip
   hostname n.nn.nn.nn
```

n.nn.nn.nn はクラスターの IP アドレスです。

クライアント・アプリケーションから LDAP に DB2 サーバーを登録するには、db2LdapRegister API を呼び出します。

アタッチのためにノード別名をカタログする

DB2 サーバーのノード名については、サーバーを LDAP に登録するときに、指定しなければなりません。アプリケーションはこのノード名を使用して、データベース・サーバーにアタッチします。

ノード名がアプリケーション内でハードコーディングされる場合のように、別のノード名が必要であれば、CATALOG LDAP NODE コマンドを以下の例のように使用して変更してください。

```
db2 catalog ldap node <ldap_node_name>
   as <new_alias_name>
```

LDAP ノードをアンカタログするには、UNCATALOG LDAP NODE コマンドを以下のように使用します。

```
db2 uncatalog ldap node <ldap_node_name>
```

LDAP ディレクトリーへのデータベースの登録

インスタンス内でデータベースを作成するときに、そのデータベースは LDAP へ自動的に登録されます。登録することにより、クライアント・マシンでデータベースおよびノードをカタログせずに、データベースへリモート・クライアント接続できるようになります。 クライアントがデータベースへ接続しようとする場合に、ローカル・マシンのデータベース・ディレクトリーにそのデータベースが存在していなければ、 LDAP ディレクトリーが検索されます。

LDAP ディレクトリーにその名前が存在する場合、データベースはローカル・マシンで作成されますが、 LDAP ディレクトリーにおける名前の競合を警告する警告メッセージが戻されます。このため、LDAP ディレクトリーでは、手動でデータベースをカタログすることができます。ユーザーは、CATALOG LDAP DATABASE コマンドを使用して、リモート・サーバーのデータベースを LDAP に登録できます。リモート・データベースを登録するときには、リモート・データベース・サーバーを表す LDAP ノードの名前を指定します。リモート・データベース・サーバーは、データベースを登録する前に、 REGISTER DB2 SERVER IN LDAP コマンドを使用して、 LDAP に登録しなければなりません。データベースを手動で LDAP に登録するには、CATALOG LDAP DATABASE コマンドを使います。

```
db2 catalog ldap database <dbname>
   at node <node_name>
   with "My LDAP database"
```

クライアント・アプリケーションから LDAP にデータベースを登録するには、db2LdapCatalogDatabase API を呼び出します。

LDAP 項目の登録解除

DB2 サーバーの登録を解除する

LDAP から特定のインスタンスの登録を解除すると、そのインスタンスを参照するすべてのノード、あるいは、そのインスタンスを参照する別名、オブジェクト、およびデータベース・オブジェクトも除去されます。

ローカルまたはリモート・マシンのいずれでも、DB2 サーバーの登録を解除する場合には、そのサーバーの LDAP ノード名を指定する必要があります。

```
db2 deregister db2 server in ldap
   node <node_name>
```

クライアント・アプリケーションから LDAP の DB2 サーバーの登録を解除するには、db2LdapDeregister API を呼び出します。

DB2 サーバーの登録を解除すると、 DB2 サーバーのその同じインスタンスを参照する LDAP ノード項目および LDAP データベース項目も、すべてアンカタログされます。

LDAP ディレクトリーからのデータベースの登録解除

データベースをドロップしたり、データベースを所有するインスタンスを LDAP から登録解除すると、データベースも LDAP から自動的に登録解除されます。

データベースを LDAP から手動で登録解除する場合は、以下のコマンドを使用します。

```
db2 uncatalog ldap database <dbname>
```

クライアント・アプリケーションから LDAP のデータベースの登録を解除するには、db2LdapUncatalogDatabase API を呼び出します。

LDAP ユーザーの構成

LDAP ユーザーの作成

DB2 データベース・システムでは、 DB2 レジストリー変数と CLI 構成をユーザー・レベルで設定できます。 (これは Linux および UNIX プラットフォームでは使用できません。) ユーザー・レベルのサポートがあれば、マルチユーザー環境でユーザー固有の設定が可能です。 Windows Terminal Server はその一例です。ここでは、各ログオン・ユーザーが、システム環境や他のユーザーの環境を干渉することなく自分の環境をカスタマイズできます。

IBM Tivoli ディレクトリーを使用する場合、ユーザー・レベルの情報を LDAP に保管する前に、LDAP ユーザーを定義しなければなりません。 LDAP ユーザーを作成するには、ユーザー・オブジェクトのすべての属性を収容する LDIF ファイルを作成してから、LDIF インポート・ユーティリティーを実行し、そのオブジェクトを LDAP ディレクトリーにインポートします。 IBM Tivoli Directory Server 用の LDIF ユーティリティーは LDIF2DB です。

人物オブジェクトの属性を収めた LDIF ファイルは次のようなものです。

```
File name: newuser.ldif

dn: cn=Mary Burnnet, ou=DB2 Development, ou=Toronto, o=ibm, c=ca
objectclass: ePerson
cn: Mary Burnnet
sn: Burnnet
uid: mburnnet
userPassword: password
telephonenumber: 1-416-123-4567
facsimiletelephonenumber: 1-416-123-4568
title: Software Developer
```

IBM LDIF インポート・ユーティリティーを使用して LDIF ファイルをインポートする LDIF コマンドの例を、以下に示します。

```
LDIF2DB -i newuser.ldif
```

注:

1. LDIF2DB コマンドは LDAP サーバー・マシンから実行しなければなりません。
2. 必要なアクセス・コントロール・リスト (ACL) を LDAP ユーザー・オブジェクトに付与することにより、LDAP ユーザーが所有オブジェクトの追加、削除、読み取り、書き込みを行えるようにしなければなりません。ユーザー・オブジェクトについての ACL を付与するには、 LDAP Directory Server Web Administration ツールを使用します。

DB2 アプリケーション用 LDAP ユーザーの構成

Microsoft LDAP クライアントを使用する場合、 LDAP ユーザーはオペレーティング・システムのユーザー・アカウントと同一です。しかし、IBM LDAP クライアントを使用している場合、DB2 データベース・マネージャーを使用する前に、現行ログオン・ユーザーの LDAP ユーザー識別名 (DN) とパスワードを構成しなければなりません。

LDAP ユーザーの識別名 (DN) をパスワードを構成するには、db2ldcfg ユーティリティーを使用します。

```
db2ldcfg -u <userDN> -w <password> --> set the user's DN and password
         -r                        --> clear the user's DN and password
```

例:

```
db2ldcfg -u "cn=Mary Burnnet,ou=DB2 Development,ou=Toronto,o=ibm,c=ca"
         -w password
```

LDAP 環境でのユーザー・レベルでの DB2 レジストリー変数の設定

LDAP 環境では、DB2 プロファイルのレジストリー変数をユーザー・レベルで設定できます。これにより、それぞれの DB2 環境をカスタマイズすることができます。

DB2 プロファイルのレジストリー変数をユーザー・レベルで設定するには、 -ul オプションを使います。

```
db2set -ul <variable>=<value>
```

注: これは AIX や Solaris オペレーティング・システムではサポートされません。

DB2 はキャッシュ・メカニズムを備えています。 DB2 プロファイルのユーザー・レベルのレジストリー変数は、ローカル・マシンにキャッシュされます。 -ul パラメーターを指定すると、DB2 は、いつもキャッシュから DB2 レジストリー変数を読み取ります。キャッシュは、以下の場合に更新されます。

- DB2 レジストリー変数をユーザー・レベルで更新またはリセットした場合。
- LDAP プロファイル変数をユーザー・レベルで更新するためのコマンドは、次のとおりです。

  ```
  db2set -ur
  ```

LDAP サポートを使用不可にする

LDAP サポートを使用不可にするには、以下の手順に従います。

1. DB2 サーバーのインスタンスごとに、DB2 サーバーを LDAP から登録解除します。

   ```
   db2 deregister db2 server in ldap node <nodename>
   ```

2. DB2 プロファイルのレジストリー変数 DB2_ENABLE_LDAP を ″NO″ に設定します。

DB2 サーバー用のプロトコル情報を更新する

LDAP 内の DB2 サーバー情報は、最新のものにしておく必要があります。たとえば、プロトコル構成パラメーターまたはサーバーのネットワーク・アドレスを変更するときには、 LDAP を更新しなければなりません。

ローカル・マシンの LDAP にある DB2 サーバーを更新するには、次のコマンドを使用します。

```
db2 update ldap ...
```

更新可能なプロトコル構成パラメーターの例としては、TCP/IP ホスト名とサービス名またはポート番号のパラメーターがあります。

リモート DB2 サーバーのプロトコル構成パラメーターを更新するには、 node 節を指定した UPDATE LDAP コマンドを使います。

```
db2 update ldap
   node <node_name>
   hostname <host_name>
   svcename <tcpip_service_name>
```

他のサーバーへの LDAP クライアントの転送

システム障害時のクライアント・リルートと同じような機能が、 LDAP でも使用できます。

DB2_ENABLE_LDAP レジストリー変数を「Yes」に設定する必要があります。

LDAP 環境では、データベースおよびノード・ディレクトリーの情報がすべて LDAP サーバーで維持されます。クライアントは、LDAP ディレクトリーから情報を検索します。 DB2LDAPCACHE レジストリー変数が「Yes」に設定されている場合、この情報はローカル・データベースおよびノード・ディレクトリーで更新されます。

UPDATE ALTERNATE SERVER FOR LDAP DATABASE コマンドを使用して、LDAP で DB2 データベースを表すデータベースの代替サーバーを定義します。あるいは、クライアント・アプリケーションから db2LdapUpdateAlternateServerForDB API を呼び出して、LDAP 内のデータベースの代替サーバーを更新します。

この代替サーバー情報が確立されると、接続時にこの情報がクライアントに戻されます。

注: LDAP サーバーに保管されている代替サーバー情報と、データベース・サーバー・インスタンスに保管されている代替サーバー情報との同期を保持することを強くお勧めします。データベース・サーバー・インスタンスで UPDATE ALTERNATE SERVER FOR DATABASE コマンド (「FOR LDAP DATABASE」ではないことに注意) を発行すると、データベース・サーバー・インスタンスと LDAP サーバーの間の同期を確実にするのに役立ちます。

UPDATE ALTERNATE SERVER FOR DATABASE コマンドをサーバー・インスタンスで入力すると、サーバーで LDAP サポートが使用可能 (DB2_ENABLE_LDAP=Yes) になっていて、かつ (db2ldcfg が実行済みで) LDAP ユーザー ID とパスワードがすでにキャッシュされている場合、データベースの代替サーバーが LDAP サーバー上で自動的に (つまり暗黙的に) 更新されます。これは、UPDATE ALTERNATE SERVER FOR LDAP DATABASE を明示的に入力したかのように機能します。

UPDATE ALTERNATE SERVER FOR LDAP DATABASE コマンドをデータベース・サーバー・インスタンス以外のインスタンスから発行する場合は、UPDATE ALTERNATE SERVER FOR DATABASE コマンドを使用して、確実に代替サーバー情報がデータベース・サーバー・インスタンスでも同じように構成されるようにしてください。クライアントが最初にデータベース・サーバー・インスタンスに接続した後、データベース・サーバー・インスタンスから戻される代替サーバー情報は、LDAP サーバーで構成される情報より優先されます。データベース・サーバー・インスタンスに代替サーバー情報を構成しない場合、最初の接続後、クライアント・リルートは使用不可と見なされます。

LDAP 環境でのリモート・サーバーのアタッチ

LDAP 環境では、ATTACH コマンドで LDAP ノード名を使用することにより、リモート・データベース・サーバーへアタッチすることができます。例えば、db2 attach to <ldap_node_name> のようなコマンドを使用します。

クライアント・アプリケーションが特定のノードにアタッチまたはデータベースに初めて接続する場合、そのノードはローカル・ノード・ディレクトリーに含まれていないので、データベース・マネージャーは LDAP ディレクトリーの中でその宛先ノード項目を探します。 LDAP ディレクトリーでその項目が見つかったら、リモート・サーバーのプロトコル情報が取り出されます。データベースに接続し、LDAP ディレクトリーでその項目が見つかったら、データベース情報が取り出されます。この情報を使用して、データベース・マネージャーはローカル・マシンのデータベース項目およびノード項目を自動的にカタログします。次回にクライアント・アプリケーションが同じノードまたはデータベースにアタッチするときには、ローカル・データベース・ディレクトリーの情報が使われるので、 LDAP ディレクトリーを検索する必要はありません。

詳細情報: キャッシュ・メカニズムが備えられているので、クライアントは LDAP サーバーを一度だけ検索します。情報を検索すると、 *dir_cache* データベース・マネージャー構成パラメーターおよび DB2LDAPCACHE レジストリー変数の値に基づいて、その情報はローカル・マシンに格納されるか、キャッシュに入れられます。

- DB2LDAPCACHE=NO かつ dir_cache=NO の場合、情報は必ず LDAP から読み取られます。
- DB2LDAPCACHE=NO かつ dir_cache=YES の場合、 LDAP から一度情報が読み取られ、その情報は DB2(R) キャッシュに挿入されます。
- DB2LDAPCACHE=YES であるか、これが設定されない場合には、 LDAP サーバーから情報が一度読み取られ、その情報はローカル・データベース、ノード、および DCS ディレクトリーにキャッシュされます。

注: LDAP 情報のキャッシュには、ユーザー・レベルの CLI や DB2 プロファイル・レジストリー変数は含まれません。

ローカル・データベースおよびノード・ディレクトリーの LDAP 項目をリフレッシュする

DB2 データベース・システムは、クライアントが LDAP サーバーを検索する回数を削減するキャッシュ・メカニズムを提供します。

情報を検索すると、 dir_cache データベース・マネージャー構成パラメーターおよび DB2LDAPCACHE レジストリー変数の値に基づいて、その情報はローカル・マシンに格納されるか、キャッシュに入れられます。

- DB2LDAPCACHE=NO かつ dir_cache=NO の場合、情報は必ず LDAP から読み取られます。
- DB2LDAPCACHE=NO かつ dir_cache=YES の場合、 LDAP から一度情報が読み取られ、その情報は DB2 キャッシュに挿入されます。

- DB2LDAPCACHE=YES であるか、これが設定されない場合には、 LDAP サーバーから情報が一度読み取られ、その情報はローカル・データベース、ノード、および DCS ディレクトリーにキャッシュされます。

注: LDAP 情報のキャッシュには、ユーザー・レベルの CLI や DB2 プロファイル・レジストリー変数は含まれません。LDAP の情報は変更されることがあるため、ローカル・データベースおよびノード・ディレクトリーにキャッシュされた LDAP 項目をリフレッシュする必要が生じるかもしれません。これを行ういくつかの方法があります。

LDAP から取得されたすべてのローカル・データベースおよびノード項目をリフレッシュするには、以下のコマンドを使います。

```
db2 refresh ldap immediate
```

同様に、既存のローカル・データベースおよびノード項目をリフレッシュするとともに、LDAP から新規項目を追加するには、以下のコマンドを使用することができます。

```
db2 refresh ldap immediate all
```

IMMEDIATE ALL オプションを指定することによって、LDAP サーバーに含まれているすべての NODE および DB 項目がローカル・ディレクトリーに追加されます。

あるいは、次回のデータベース接続またはインスタンスのアタッチの際に、LDAP リソースを参照するデータベース項目を DB2 に強制的にリフレッシュさせるには、次のコマンドを使用します。

```
db2 refresh ldap database directory
```

同様に、次回のデータベース接続またはインスタンスのアタッチの際に、LDAP リソースを参照するノード項目を DB2 データベース・マネージャーに強制的にリフレッシュさせるには、次のコマンドを使用します。

```
db2 refresh ldap node directory
```

リフレッシュの一環として、ローカル・データベースおよびノード・ディレクトリーに保管されているすべての LDAP 項目が除去されます。次回にアプリケーションがデータベースまたはノードにアクセスすると、情報を LDAP から直接に読み取り、ローカル・データベースまたはノード・ディレクトリーに新しい項目を生成します。

リフレッシュを周期的に実行するには、以下を行います。

- 周期的に実行するリフレッシュをスケジュールする。
- システムのブート時に REFRESH コマンドを実行する。
- 使用可能な管理パッケージを使い、すべてのクライアント・マシンで REFRESH コマンドを呼び出す。
- DB2LDAPCACHE=″NO″ を設定して、 LDAP 情報がデータベース、ノード、および DCS ディレクトリーのキャッシュに入れられないようにする。

LDAP サーバーの検索

DB2 データベース・システムは現行の LDAP サーバーを検索します。しかし、複数の LDAP サーバーがある環境では、検索範囲を定義できます。

例えば、現行の LDAP サーバーで情報が見つからない場合、他のすべての LDAP サーバーの自動検索を指定するか、またはその代替手段として、検索範囲を現行の LDAP サーバー、またはローカル DB2 データベース・カタログだけに限定することができます。

検索範囲を設定すると、エンタープライズ全体でのデフォルト検索範囲が設定されます。この検索範囲は、DB2 データベース・プロファイルのレジストリー変数 DB2LDAP_SEARCH_SCOPE によって制御されます。検索範囲の値を設定するには、 db2set コマンドで -gl オプションを使用します。これは「LDAP 内でグローバル」という意味です。

```
db2set -gl db2ldap_search_scope=<value>
```

可能な値には、「local」、「domain」、あるいは「global」があります。設定しない場合、デフォルト値は「domain」です。この値は、検索範囲を現行 LDAP サーバーのディレクトリーに限定します。

例えば、新しいデータベースを作成したら、検索範囲を「global」に初期設定することができます。これにより、LDAP を使用するように構成された DB2 クライアントは、すべての LDAP サーバーを検索し、データベースを探すことができます。まずそれぞれのクライアントに接続またはアタッチし、各マシンでエントリーが記録されたら、キャッシングを使用可能にしてある場合は、検索範囲を「local」に変更することができます。「local」に変更すると、各クライアントはどの LDAP サーバーもスキャンしません。

注: LDAP 内のグローバル・レベルで設定できるレジストリー変数は、DB2 データベース・プロファイルのレジストリー変数 DB2LDAP_KEEP_CONNECTION および DB2LDAP_SEARCH_SCOPE のみです。

第 2 部 データベース

第 6 章 データベース

DB2 データベースは、*リレーショナル・データベース* です。このデータベースは、相互に関連する表にすべてのデータを格納します。データが共有され、重複が最小限にとどめられるように、表間のリレーションシップが確立されます。

リレーショナル・データベース は、1 つの表集合として扱われ、データのリレーショナル・モデルに従って操作されます。このデータベースには、データの保存、管理、およびアクセスに使用されるオブジェクトが一式揃っています。そのようなオブジェクトの例として、表、ビュー、索引、関数、トリガー、およびパッケージがあります。オブジェクトには、システムで定義するもの (システム定義オブジェクト) とユーザーが定義するもの (ユーザー定義オブジェクト) があります。

分散リレーショナル・データベース は、相互接続された異なるコンピューター・システムに分散している表集合と他のオブジェクトで構成されています。各コンピューター・システムには、その環境で表を管理するリレーショナル・データベース・マネージャーが 1 つあります。 これらのデータベース・マネージャーは、特定のデータベース・マネージャーが SQL ステートメントを別のコンピューター・システムで実行することができるような仕方で、相互に通信および調整を行います。

パーティション・リレーショナル・データベース は、データが複数のデータベース・パーティションにまたがって管理されるリレーショナル・データベースのことです。データベース・パーティション間のデータの分離は、ほとんどの SQL ステートメントではユーザーに認識されません。ただし、一部のデータ定義言語 (DDL) ステートメントでは、データベース・パーティション情報が考慮されます (CREATE DATABASE PARTITION GROUP など)。 DDL は、同じデータベース内のデータのリレーションシップを記述するために使用される SQL ステートメントのサブセットです。

フェデレーテッド・データベース は、データが複数のデータソース (分離リレーショナル・データベースなど) に保存されるリレーショナル・データベースのことです。データはあたかも単一の大容量のデータベースにあるかのように見え、従来の SQL 照会でアクセスすることができます。データに対する変更は、該当するデータ・ソースへ明示的に送られます。

データベースの設計

データベースを設計するときは、現実の業務システムをモデル化します。そのシステムには、エンティティーとその特性 (つまり、*属性*) のセットと、それらのエンティティー間のルールやリレーションシップ (関係) が含まれています。

最初のステップは、自分が表現したいシステムを記述することです。例えば、出版システムのデータベースを作成している場合、そのシステムには、書籍、著者、編集者、および発行者などのいくつかのタイプのエンティティーが含まれるでしょう。各エンティティーには、次のように、記録する必要のあるいくつかの情報 (つまり、属性) があります。

- *書籍:* タイトル、ISBN、発行日、場所、発行者、....

- *著者:* 氏名、住所、電話・FAX 番号、E-mail アドレス、....
- *編集者:* 氏名、住所、電話・FAX 番号、E-mail アドレス、....
- *発行者:* 氏名、住所、電話・FAX 番号、E-mail アドレス、....

データベースでこれらの種類のエンティティーとその属性を表すだけでなく、これらのエンティティーを互いに関連付ける方法も必要になります。例えば、書籍とその著者の関係、書籍/著者と編集者の関係、および書籍/著者と発行者の関係を表す必要があります。

データベースのこれらのエンティティー同士のリレーションシップには、3 つのタイプがあります。

1 対 1 のリレーションシップ
このタイプのリレーションシップでは、1 つのエンティティーの各インスタンスに別のエンティティーの 1 つ以上のインスタンスが関係しています。現行では、上述のシナリオに 1 対 1 のリレーションシップはありません。

1 対多のリレーションシップ
このタイプのリレーションシップでは、1 つのエンティティーの各インスタンスに別のエンティティーの 1 つ以上インスタンスが関係しています。例えば、一人の著者が複数の書籍を書いており、複数の書籍の著者が 1 人だけ、という場合などです。これが、リレーショナル・データベースでモデルとなるリレーションシップの最も一般的なタイプです。

多対多のリレーションシップ
このタイプのリレーションシップでは、ある特定のエンティティーの多くのインスタンスが、別のエンティティーの 1 つ以上のインスタンスに関係しています。例えば、ある書籍を共同執筆した複数の著者が、他に多くの書籍を書いている場合などです。

データベースは表で構成されているため、表の各セルが単一のビューを保持する、このデータに最も適した表集合を構築する必要があります。このタスクを行うために考えられる方法は、いくつもあります。データベースの設計担当者であれば、可能な範囲で最適な表集合を考案するのが仕事になります。

例えば、多くの列と行のある 1 つの表を作成して、そこにすべての情報を収めることもできます。しかし、この方法では一部の情報が反復してしまうでしょう。また、データ入力とデータ保守に多大な時間がかり、エラーも起こりやすくなります。この単一表設計とは対照的に、*リレーショナル・データベース* では、シンプルな表を複数持つことで、冗長を減らし、大きくて管理しにくい表で生じる問題を回避することができます。リレーショナル・データベースでは、複数の表に単一のタイプのエンティティーに関する情報が入ります。

また、リレーショナル・データベース内のデータは複数のユーザーがアクセスし、データを変更するので、その整合性を保守する必要もあります。データを共有する場合は常に、データベース表内の値の正確性を保証する必要があります。

行うことができる操作は、以下のとおりです。

- 分離レベルを使用して、データがアクセスされている間、他の処理からデータをどのようにロックまたは分離するかを決定する。

- 制約を定義してビジネス規則を強制することにより、データを保護し、データ間のリレーションシップを定義する。
- トリガーを作成し、複雑な、複数の表にまたがるデータの妥当性検査を行う。
- 一貫性のある状態にリストアできるようにデータを保護するためのリカバリー戦略をインプリメントする。

実際のデータベース設計は、ここで示したものよりもさらに複雑なタスクであり、スペース所要量、キー、索引、制約、セキュリティーと許可など、考慮すべき事項も多くあります。この情報については、DB2 インフォメーション・センター、およびこの主題に関する入手可能な多くの DB2 ブックで説明されています。

データベース・ディレクトリーおよびファイル

データベースを作成するとき、デフォルトの情報を含むデータベースに関する情報は、ディレクトリー階層内に保管されます。

階層ディレクトリー構造は、CREATE DATABASE コマンド内に提供する情報によって判別されるロケーションに作成されます。データベースを作成するときにディレクトリー・パスまたはドライブのロケーションを指定しない場合、デフォルトのロケーションが使用されます。データベースを作成するロケーションを明示的に示すことをお勧めします。

CREATE DATABASE コマンドのデータベース・パスに指定したディレクトリーに、インスタンスの名前を使用するサブディレクトリーが作成されます。このサブディレクトリーにより、同じディレクトリーの下に異なるインスタンスで作成されたデータベースが同じパスを使用しないようにします。インスタンス名のサブディレクトリーの下に、NODE0000 というサブディレクトリーが作成されます。このサブディレクトリーは、論理的にパーティション化されたデータベース環境内のデータベース・パーティションを区別します。ノード名のディレクトリーの下に、SQL00001 というサブディレクトリーが作成されます。このサブディレクトリーのこの名前はデータベース・トークンを使用し、作成されているデータベースを表します。 SQL00001 には、作成された 1 番目のデータベースに関連するオブジェクトが含まれ、 2 番目以降のデータベースについては SQL00002 というように、より大きな数値が順次与えられます。これらのサブディレクトリーは、CREATE DATABASE コマンドで指定したディレクトリー上のこのインスタンスで作成されたデータベースを区別します。

以下のように、ディレクトリー構造が表示されます。

```
<your_database_path>/<your_instance>/NODE0000/SQL00001/
```

データベース・ディレクトリーには、CREATE DATABASE コマンドの一部として作成される以下のファイルが含まれます。

- ファイル SQLBP.1 および SQLBP.2 には、バッファー・プール情報が含まれています。これらのファイルは、バックアップのため、相互に重複しています。
- ファイル SQLSPCS.1 および SQLSPCS.2 には、表スペース情報が含まれています。これらのファイルは、バックアップのため、相互に重複しています。
- SQLSGF.1 および SQLSGF.2 ファイルには、データベースの自動ストレージに関連したストレージ・パス情報が入っています。これらのファイルは、バックアップのため、相互に重複しています。

- SQLDBCONF ファイルには、データベース構成情報が入っています。このファイルを編集しないでください。

  注: 以前のリリースでは SQLDBCON ファイルが使用されており、そこには SQLDBCONF が壊れた場合に使用できるものと同様の情報が含まれています。構成パラメーターを変更するには、 UPDATE DATABASE CONFIGURATION および RESET DATABASE CONFIGURATION ステートメントを使用します。
- DB2RHIST.ASC ヒストリー・ファイルおよびそのバックアップ DB2RHIST.BAK には、バックアップ、リストア、表のロード、表の再編成、表スペースの変更、およびデータベースへの他の変更についての履歴情報が含まれています。

  DB2TSCHG.HIS ファイルには、ログ・ファイル・レベルでの表スペース変更の履歴が入っています。各ログ・ファイルごとに、DB2TSCHG.HIS には、ログ・ファイルに影響される表スペースを識別するのに役立つ情報が入っています。表スペース・リカバリーは、このファイルからの情報を使用して、表スペース・リカバリー中に処理するログ・ファイルを判別します。テキスト・エディターで、両方のヒストリー・ファイルの内容を調べることができます。
- ログ制御ファイル SQLOGCTL.LFH.1 とそのミラー・コピーである SQLOGCTL.LFH.2、および SQLOGMIR.LFH には、アクティブ・ログについての情報が入ります。

  リカバリー処理では、これらのファイルの情報を使用して、リカバリーのために戻るログ内の位置を判別します。 SQLOGDIR サブディレクトリーには、実際のログ・ファイルが入ります。

  注: このログ・サブディレクトリーはデータに使用されているディスクとは異なるディスクに配置することをお奨めします。こうすると、ディスクの問題が生じたときに、データとログの両方ではなく、データまたはログのいずれかに限定することができます。ログ・ファイルおよびデータベース・コンテナーは、同じディスク・ヘッドの移動で競合することはないため、パフォーマンスにも大きな利点となります。ログ・サブディレクトリーのロケーションを変更するには、*newlogpath* データベース構成パラメーターを変更します。
- SQLINSLK ファイルは、データベースが必ず 1 つのデータベース・マネージャー・インスタンスでしか使われないようにします。

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・モニターも作成されます。詳細デッドロック・イベント・モニター・ファイルは、カタログ・ノードのデータベース・ディレクトリーに保管されています。イベント・モニターが、出力するファイルの最大数に達した場合、イベント・モニターは非活動化され、メッセージが通知ログに書き込まれます。これは、イベント・モニターがディスク・スペースを使用し過ぎるのを防ぎます。不要になった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アクティブになります。

非自動ストレージ・データベースの SMS データベース・ディレクトリーについての追加情報

非自動ストレージ・データベースでは、作動データベースに必要なデフォルトのシステム管理スペース (SMS) 表スペースが、SQLT* サブディレクトリーにあります。デフォルトの表スペースは 3 つ作成されます。

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

各サブディレクトリーまたはコンテナーには、 SQLTAG.NAM というファイルが作成されています。このファイルは、サブディレクトリーに使用中のマークを付け、後続の表スペース作成で、これらのサブディレクトリーが使用されないようにします。

さらに、SQL*.DAT というファイルが、サブディレクトリーまたはコンテナーに含まれる、各表についての情報を保管します。アスタリスク (*) は、各表を示す固有の数字の集合で置き換えられます。各 SQL*.DAT ファイルごとに、表タイプ、表の再編成状況、または索引、LOB、または LONG フィールドがその表に存在するかどうかによって、以下のファイルが 1 つ以上存在することがあります。

- SQL*.BKM (MDC 表である場合、ブロック割り振り情報を含む)
- SQL*.LF (LONG VARCHAR または LONG VARGRAPHIC データを含む)
- SQL*.LB (BLOB、CLOB、または DBCLOB データを含む)
- SQL*.XDA (XML データを含む)
- SQL*.LBA (SQL*.LB ファイルについての割り当ておよびフリー・スペース情報を含む)
- SQL*.INX (索引表データを含む)
- SQL*.IN1 (索引表データを含む)
- SQL*.DTR (SQL*.DAT ファイルの再編成についての一時データを含む)
- SQL*.LFR (SQL*.LF ファイルの再編成についての一時データを含む)
- SQL*.RLB (SQL*.LB ファイルの再編成についての一時データを含む)
- SQL*.RBA (SQL*.LBA ファイルの再編成についての一時データを含む)

データベース構成ファイル

データベースごとにデータベース*構成ファイル* が作成されます。このファイルは、バージョン 8.2 より前では SQLDBCON と呼ばれ、バージョン 8.2 以降では、SQLDBCONF と呼ばれます。このファイルの作成は管理者のために行われます。

このファイルには、データベースの使用に影響を与える、次のような様々な*構成パラメーター* の値が入れられます。

- データベースの作成時に指定または使用されるパラメーター (データベース・コード・ページ、照合順序、DB2 データベース・リリース・レベルなど)

- データベースの現行の状態を示すパラメーター (バックアップ・ペンディング・フラグ、データベース一貫性フラグ、ロールフォワード操作ペンディング・フラグなど)
- データベースの操作時に使用されるシステム・リソースの量を定義するパラメーター (バッファー・プール・サイズ、データベース・ロギング、ソート・メモリー・サイズなど)

注: DB2 データベース・マネージャーが提供している以外の方法を使用して db2system、SQLDBCON (バージョン 8.2 よりも前)、または SQLDBCONF (バージョン 8.2 以降) ファイルを編集すると、データベースが使用不可能になることがあります。このため、これら文書化され、データベース・マネージャーによりサポートされている以外の方法で、ファイルを変更しないでください。

パフォーマンス上のヒント: 構成パラメーターの多くはデフォルト値が提供されていますが、データベースの最適なパフォーマンスを達成するためには構成パラメーターの更新が必要な場合があります。デフォルトでは、構成アドバイザーが、CREATE DATABASE コマンドの一部として呼び出され、一部のパラメーターの初期値がご使用の環境に合わせてあらかじめ構成されます。

複数パーティション・データベースの場合: 複数のデータベース・パーティションにわたって分散されたデータベースを持っている場合、構成ファイルは、すべてのデータベース・パーティションで同じものである必要があります。照会コンパイラーは、ローカル・ノードの構成ファイルの情報に基づいて分散 SQL ステートメントをコンパイルし、 SQL ステートメントのニーズを満足させるためのアクセス・プランを作成するので、これらの構成ファイルには整合性が必要です。データベース・パーティションごとに異なる構成ファイルを維持していると、どのデータベース・パーティションでステートメントが準備されたかによって、異なるアクセス・プランが作成される可能性があります。

ノード・ディレクトリー

データベース・マネージャーは、最初のデータベース・パーティションがカタログされるときに*ノード・ディレクトリー* を作成します。

データベース・パーティションをカタログするためには、CATALOG NODE コマンドを使用します。ローカルのノード・ディレクトリーの内容をリストするには、LIST NODE DIRECTORY コマンドを使用します。

ノード・ディレクトリーは、各データベース・クライアントごとに作成され維持されます。ディレクトリーには、そのクライアントがアクセスできる 1 つ以上のデータベースを持っている各リモート・ワークステーションごとに 1 つの項目が入っています。 DB2 クライアントは、データベース接続またはインスタンス接続が要求されると、ノード・ディレクトリーの中の通信エンドポイント情報を使用します。

ディレクトリーの中の項目には、クライアントからリモート・データベース・パーティションに通信するために使用される、通信プロトコルのタイプについての情報も含まれます。ローカル・データベース・パーティションをカタログすることによって、同じコンピューターにあるインスタンスに対する別名が作成されます。

ローカル・データベース・ディレクトリー

ローカル・データベース・ディレクトリー ・ファイルは、データベースがすでに定義されている各パス (もしくは Windows オペレーティング・システムの「ドライブ」) にあります。このディレクトリーには、そこからアクセスできるデータベースごとに 1 つの項目が入っています。

各項目には次のものが含まれています。

- CREATE DATABASE コマンドによって提供されたデータベース名
- データベースの別名 (別名が指定されない場合は、データベース名と同じ)
- データベースを説明する注釈 (CREATE DATABASE コマンドで提供されたもの)
- データベースのためのルート・ディレクトリーの名前
- その他のシステム情報

システム・データベース・ディレクトリー

システム・データベース・ディレクトリー・ファイルは、データベース・マネージャーの各インスタンスごとに存在し、このインスタンスに対してカタログされているデータベースごとに 1 つの項目が含まれています。

データベースは CREATE DATABASE コマンドの発行時に暗黙のうちにカタログされますが、 CATALOG DATABASE コマンドによって明示的にカタログすることもできます。

作成されたそれぞれのデータベースごとに 1 つの項目がディレクトリーに追加されますが、これには以下の情報が含まれます。

- CREATE DATABASE コマンドによって提供されたデータベース名
- データベースの別名 (別名が指定されない場合は、データベース名と同じ)
- CREATE DATABASE コマンドによって提供されたデータベース注釈
- ローカル・データベース・ディレクトリーの位置
- データベースが*間接* データベースであることを示す標識。これは、データベースが現行のデータベース・マネージャー・インスタンス上にあるという意味です。
- その他のシステム情報

UNIX プラットフォームおよびパーティション・データベース環境では、すべてのデータベース・パーティションが、同じシステム・データベース・ディレクトリー・ファイル sqldbdir (そのインスタンスのホーム・ディレクトリーの sqldbdir サブディレクトリーにある) に常にアクセスするようにしてください。同じ sqldbdir サブディレクトリーにあるシステム・データベース・ディレクトリーかシステム・インテンション・ファイル sqldbins のいずれかが、共有ファイル・システム上の別のファイルに飛ぶシンボリック・リンクである場合、予期しないエラーが発生する可能性があります。

ノード構成ファイルの作成

データベースをパーティション・データベース環境で操作する場合、 db2nodes.cfg というノード構成ファイルを作成する必要があります。

このファイルは、並列機能を持ったデータベース・マネージャーを複数データベース・パーティションにわたって開始する前に、そのインスタンスに対するホーム・

ディレクトリーの sqllib サブディレクトリーの中に配置されていなければなりません。このファイルには、1 つのインスタンスの中のすべてのデータベース・パーティションの構成情報が含まれ、そのインスタンスのすべてのデータベース・パーティションによって共用されます。

Windows での考慮事項

DB2 Enterprise Server Edition を Windows で使用している場合、インスタンス作成時にノード構成ファイルが作成されます。ノード構成ファイルは手動で作成したり変更したりしないでください。 db2ncrt コマンドを使用して、データベース・パーティション・サーバーをインスタンスに追加することができます。 db2ndrop コマンドを使用して、データベース・パーティション・サーバーをインスタンスからドロップすることができます。db2nchg コマンドを使用すれば、データベース・パーティション・サーバーの構成を変更することができます。たとえば、1 つのコンピューターから別のコンピューターへのデータベース・パーティション・サーバーの移動、 TCP/IP ホスト名の変更、または別の論理ポートやネットワーク名の選択を行うことができます。

注: インスタンスが削除された場合にデータの消失を避けるため、sqllib サブディレクトリーには、データベース・マネージャーによって作成されたもの以外のファイルまたはディレクトリーを作成しないでください。ただし、以下の 2 つの例外があります。システムがストアード・プロシージャーをサポートしている場合は、ストアード・プロシージャー・アプリケーションを sqllib サブディレクトリーの下の function サブディレクトリーに入れます。もう 1 つの例外は、ユーザー定義関数 (UDF) が作成される場合です。 UDF の実行可能コードは、同じディレクトリーに入れることが許されます。

ファイルには、1 つのインスタンスに属する各データベース・パーティションごとに 1 行が含まれます。それぞれの行は、以下の形式になっています。

```
dbpartitionnum hostname [logical-port [netname]]
```

トークンはブランクで区切られます。変数は、以下のとおりです。

dbpartitionnum

データベース・パーティションを固有に定義するデータベース・パーティション番号 (0 から 999 まで)。データベース・パーティション番号は、昇順でなければなりません。間の番号が抜けていてもかまいません。

いったんデータベース・パーティション番号が割り当てられると、それを変更することはできません。 (変更すると、データを分散する方法を指定する分散マップの中の情報が信用できないものになります。)

データベース・パーティションをドロップした場合、そのデータベース・パーティション番号は、追加する任意の新しいデータベース・パーティション用に再使用することができます。

データベース・パーティション番号は、データベース・ディレクトリー内にデータベース・パーティション名を生成するために使用されます。ノード名は、以下の形式になります。

```
NODE nnnn
```

nnnn はデータベース・パーティション番号で、左側はゼロで埋められます。このデータベース・パーティション番号は、 CREATE DATABASE コマンドおよび DROP DATABASE コマンドでも使用されます。

hostname

パーティション間通信のための IP アドレスのホスト名。ホスト名には、完全修飾名を使用します。 /etc/hosts ファイルも、完全修飾名を使用する必要があります。 db2nodes.cfg ファイルおよび /etc/hosts ファイルで完全修飾名を使用しない場合、エラー・メッセージ SQL30082N RC=3 を受け取る場合があります。

(netname が指定された場合は、例外です。この場合、netname がほとんどの通信で使用され、 hostname は db2start、db2stop、および db2_all でのみ使用されます。)

logical-port

このパラメーターの指定は任意であり、データベース・パーティションの論理ポート番号を指定します。この番号は、データベース・マネージャー・インスタンス名と一緒に使用され、etc/services ファイルの中の TCP/IP サービス名項目を識別します。

IP アドレスと論理ポートの組み合わせは、既知のアドレスとして使用され、データベース・パーティション間の通信接続をサポートするために、すべてのアプリケーションの間で固有のものでなければなりません。

各 hostname について、1 つの logical-port は、0 かまたはブランク (0 がデフォルト) でなければなりません。この logical-port に関連付けられるデータベース・パーティションは、クライアントが接続するホスト上のデフォルトのノードです。これは、db2profile スクリプト内の DB2NODE 環境変数か、または sqlesetc() API を使用してオーバーライドすることができます。

netname

このパラメーターの指定は任意であり、それぞれが独自のホスト名を持つ、複数のアクティブな TCP/IP インターフェースを持ったホストをサポートするために使用されます。

以下の例は、RS/6000® SP™ システムのための可能なノード構成ファイルを示したものであり、 SP2EN1 には複数の TCP/IP インターフェースと 2 つの論理パーティションがあり、 DB2 データベースのインターフェースとして SP2SW1 を使用しています。この例は、データベース・パーティション番号が (0 ではなく) 1 から始まり、 *dbpartitionnum* の順番は間の番号が抜けていることも示しています。

表 39. データベース・パーティション番号の例示表

dbpartitionnum	hostname	logical-port	netname
1	SP2EN1.mach1.xxx.com	0	SP2SW1
2	SP2EN1.mach1.xxx.com	1	SP2SW1
4	SP2EN2.mach1.xxx.com	0	
5	SP2EN3.mach1.xxx.com		

好みのエディターを使用して、db2nodes.cfg ファイルを更新することができます。(例外: Windows ではエディターは使用しないでください。) ただし、データベース・パーティションでは、ノード構成ファイルが db2start を出したときにロックされ、db2stop がデータベース・マネージャーを停止させた後でアンロックされることが必要なため、ファイル内の情報の整合性を注意して保護する必要があります。ファイルがロックされている場合、db2start コマンドで、必要に応じて、ファイルを更新することができます。例えば、RESTART オプションまたは ADDNODE オプションを指定して db2start を発行することができます

注: db2stop コマンドが失敗し、ノード構成ファイルがアンロックされない場合、アンロックするために db2stop FORCE を発行してください。

ノードおよびデータベースの構成ファイルの変更

データベース構成ファイルを更新するには、AUTOCONFIGURE コマンドに適切なオプションを付けて実行します。

構成アドバイザーは、どの構成パラメーターを修正したらよいかを提案し、それらの推奨値を示すことによって、インスタンスごとの単一データベースのパフォーマンスのチューニングとメモリー所要量のバランスをとるよう支援します。

いずれかのデータベース・パーティションを変更 (データベース・パーティションの追加・削除、または既存のデータベース・パーティションの移動) する計画を立てている場合には、ノード構成ファイルを更新する必要があります。データベースに対する変更を計画している場合、構成パラメーターの値を見直す必要があります。一部の値は、その使われ方に応じて、データベースに対して行われる変更の一環として、定期的に調整できます。

注: パラメーターを修正しても、以下の時点まで値は更新されません。

- データベース・パラメーターの場合、すべてのアプリケーションが切断された後で、そのデータベースに対する最初の新しい接続が行われるまで。
- データベース・マネージャー・パラメーターの場合、次回、そのインスタンスを停止して開始するまで。

構成アドバイザーによって推奨された値は、個別のワークロードとサーバーについての情報に基づいた値であるため、ほとんどのケースでデフォルト値よりもパフォーマンスが良くなります。ただし、この値は、指定されたデータベース・システムのパフォーマンスを改善するよう設計されたものであり、必ずしも最適なものではありません。これらの値は、最適なパフォーマンスを獲得するために、さらに調整を行うための出発点と考えてください。

バージョン 9.1 では、データベースを作成するときに構成アドバイザーが自動的に起動します。このフィーチャーを使用不可にしたり、明示的に使用可能にしたりするには、データベースを作成する前に db2set コマンドを使用します。次に例を示します。

```
db2set DB2_ENABLE_AUTOCONFIG_DEFAULT=NO
db2set DB2_ENABLE_AUTOCONFIG_DEFAULT=YES
```

デフォルトで使用可能に設定されている他のフィーチャーについて詳しくは、「 19 ページの『自動フィーチャー』」を参照してください。

コマンド行から構成アドバイザーを使用する場合は、AUTOCONFIGURE コマンドを使用します。

コマンド行を使用してデータベース・マネージャー構成の個々のパラメーターを更新するには、以下のようにします。

```
UPDATE DBM CFG FOR <database_alias>
   USING <config_keyword>=<value>
```

1 つのコマンドで 1 つまたは複数の <config_keyword>=<value> の組み合わせを更新できます。データベース・マネージャー構成ファイルに対する変更内容は、そのほとんどが、メモリーへのロード後にはじめて有効になります。サーバー構成パラメーターの場合は、START DATABASE MANAGER コマンドの実行時に有効になります。クライアント構成パラメーターの場合は、アプリケーションの再始動時に有効になります。

現行のデータベース・マネージャー構成パラメーターを表示したり印刷したりするには、GET DATABASE MANAGER CONFIGURATION コマンドを使用してください。

クライアント・アプリケーションから構成アドバイザーにアクセスするには、db2AutoConfig API を呼び出します。クライアント・アプリケーションからデータベース・マネージャー構成の個々のパラメーターまたはデータベース構成ファイルを更新するには、db2CfgSet API を呼び出します。

データベース・リカバリー・ログ

データベース・リカバリー・ログ は、新しい表の追加または既存の表に対する更新を含む、データベースに対して行われたすべての変更の記録を維持します。

このログはいくつかの*ログ・エクステント* からなり、それぞれのログ・エクステントは、*ログ・ファイル* と呼ばれる別個のファイルに入っています。

データベース・リカバリー・ログを使用して、障害 (例えば、システム電源異常またはアプリケーション・エラー) によって、データベースが矛盾した状態のままにならないようにすることができます。障害が発生した場合、すでに入力はされたがコミットされていない変更事項はロールバックされ、コミット済みのすべてのトランザクション (ディスクに物理的に書き込まれてはいないかもしれない) は再実行されます。これらのアクションによって、データベースの整合性が保たれます。

データベース・オブジェクトのスペース所要量

データベース・オブジェクトのサイズは、正確に見積もることができません。サイズの見積もりを難しくする原因は、ディスクのフラグメント化によって発生するオーバーヘッド、フリー・スペース、および可変長列の使用などです。これは、列タイプや行の長さが広い範囲で異なる可能性があるためです。

まずデータベースのサイズを見積もってから、テスト・データベースを作成し、それに標準的なデータを入れてみてください。それから db2look ユーティリティーを使用して、データベースのデータ定義ステートメントを生成します。

データベースのサイズ見積もりを行う時、以下の要素を考慮する必要があります。

- システム・カタログ表

- ユーザー表データ
- ロング・フィールド (LF) データ
- ラージ・オブジェクト (LOB) データ
- 索引スペース
- ログ・ファイルのスペース
- 一時ワークスペース

以下のものに必要なオーバーヘッドとスペース所要量も考慮してください。

- ローカル・データベース・ディレクトリーのファイル
- システム・データベース・ディレクトリーのファイル
- オペレーティング・システムに必要なファイル管理オーバーヘッド。これには次のものが含まれます。
  - ファイル・ブロック・サイズ
  - ディレクトリー制御スペース

ログ・ファイルのスペース所要量

ログ制御ファイルの場合、56 KB のスペースが必要です。

また、最低でも、アクティブ・ログ構成のために十分なスペースが必要です。以下のように計算することができます。

```
(logprimary + logsecond) * (logfilsiz + 2 ) * 4096
```

説明:

- *logprimary* は、データベースの構成ファイルに定義されている 1 次ログ・ファイルの数です。
- *logsecond* は、データベース構成ファイル内に定義されている 2 次ログ・ファイルの数です。この計算式で、*logsecond* を *-1* に設定することはできません。(*logsecond* を -1 に設定すると、無限のアクティブ・ログ・スペースを要求していることになります。)
- *logfilsiz* は、データベースの構成ファイルに定義されている各ログ・ファイルのページ数です。
- 2 は各ログ・ファイルに必要なヘッダー・ページの数です。
- 4096 は 1 ページのバイト数です。

データベースが循環ロギングのために使用される場合、この公式の結果は、ロギング用に割り振られる全スペースとなります。つまり、それを超えるスペースは割り振られず、どのログ・ファイルに対しても、ディスク・スペース不十分のエラーを受け取ることはありません。

データベースがロールフォワード・リカバリーのために使用される場合、次のような特別のログ・スペース所要量について考慮する必要があります。

- *logarchmeth1* 構成パラメーターを logretain に設定すると、ログ・ファイルがログ・パス・ディレクトリーにアーカイブされます。オンライン・ディスク・スペースは、ログ・ファイルを別のロケーションに移動しない限り、いつかいっぱいになります。

- *logarchmeth1* 構成パラメーターを userexit、DISK、または VENDOR に設定すると、ユーザー出口プログラムによって、アーカイブ・ログ・ファイルが別のロケーションに移動されます。次のもののために、さらに余分のログ・スペースが必要です。
  - ユーザー出口プログラムによって移動される前のオンライン・アーカイブ・ログ。
  - 将来の利用のために初期化されている新しいログ・ファイル。

データベースが無限ロギングのために使用される場合 (つまり、 *logsecond* を *-1* に設定する場合)、アーカイブ・ロギングを使用可能にするには、*logarchmeth1* 構成パラメーターは OFF または LOGRETAIN 以外の値に設定されなければなりません。データベース・マネージャーは、少なくとも、ログ・パス内の *logprimary* によって指定されたアクティブ・ログ・ファイルの数を維持します。したがって、上記の公式内の *logsecond* に -1 の値を使用しないでください。ログ・ファイルのアーカイブによって生じる遅延を許可するために、余分のディスク・スペースを提供するようにしてください。

ログ・パスをミラーリングする場合、見積もりログ・ファイルのスペース所要量を 2 倍にする必要があります。

Lightweight Directory Access Protocol (LDAP) ディレクトリー・サービス

ディレクトリー・サービスとは、分散環境内にある複数のシステムおよびサービスについてのリソース情報を集めたリポジトリーです。クライアントとサーバーはディレクトリー・サービスを使用して、それらのリソースにアクセスします。

クライアントおよびサーバーは、ディレクトリー・サービスを使用して、他のリソースにアクセスする方法を見つけます。分散環境にある、これら他のリソースについての情報を、ディレクトリー・サービス・リポジトリーに入力することが必要です。

Lightweight Directory Access Protocol (LDAP) は、ディレクトリー・サービスへの業界標準アクセス方式です。各データベース・サーバーのインスタンスは自らの存在を LDAP サーバーに公開するとともに、データベースの作成時にはデータベース情報を LDAP ディレクトリーへ送信します。クライアントがデータベースに接続すると、 LDAP ディレクトリーからそのサーバーのカタログ情報を取り出せます。各クライアントは、それぞれのコンピューターでローカルにカタログ情報を保管する必要はなくなります。クライアント・アプリケーションは、LDAP ディレクトリーの中で、データベースへ接続するのに必要な情報を探します。

注: LDAP サーバーに対するデータベース・サーバー・インスタンスのパブリッシングは、自動処理ではなく、管理者が手動で行う必要があります。

DB2 システムの管理者として、ディレクトリー・サービスを確立および保守できます。構成アシスタントは、このディレクトリー・サービスの保守を支援することができます。このディレクトリー・サービスは、Lightweight Directory Access Protocol (LDAP) ディレクトリー・サービスを通して DB2 データベース・マネージャーで使用できるようになります。LDAP ディレクトリー・サービスを使用するには、まず

DB2 データベース・マネージャーがサポートする LDAP サーバーが存在しており、ディレクトリー情報がそこに保管されるようにする必要があります。

注: Windows ドメイン環境で実行中の場合、LDAP サーバーは、Windows アクティブ・ディレクトリーに統合されているので、すでに使用可能になっています。その結果、Windows を実行するすべてのコンピューターで LDAP を使用できます。

LDAP ディレクトリーは、クライアントが非常に多いために各クライアント・コンピューターでローカル・ディレクトリー・カタログを更新するのが困難なエンタープライズ環境で役立ちます。この状況では、カタログ項目の保守が 1 か所で、つまり LDAP サーバー上でなされるように、ディレクトリー項目を LDAP サーバーに保管することを考慮してください。

データベースの作成

CREATE DATABASE コマンドを使用して、データベースを作成できます。クライアント・アプリケーションからデータベースを作成するには、sqlecrea API を呼び出します。

作成前に、データベースの内容、レイアウト、潜在的な増大度、および使用方法の設計に十分な時間をかけてください。

システム・カタログ・ビュー上の CREATETAB、BINDADD、CONNECT、IMPLICIT_SCHEMA、および SELECT の各データベース特権は、PUBLIC に自動的に付与されます。ただし、RESTRICTIVE オプションを使用すると、特権が自動的に PUBLIC に認可されることがなくなります。 RESTRICTIVE オプションの詳細については、CREATE DATABASE コマンドを参照してください。

データベースの作成時には、以下のタスクがそれぞれ実行されます。

- データベースで必要になるすべてのシステム・カタログ表を設定する。
- データベースのリカバリー・ログを割り当てる。
- データベースの構成ファイルを作成し、デフォルト値を設定する。
- データベースのユーティリティーをデータベースにバインドする。

コマンド行プロセッサーを使用してデータベースを作成するには、次のように入力します。

```
CREATE DATABASE <database name>
```

例えば、次のコマンドは、デフォルトの位置に PERSON1 と呼ばれるデータベースを、″Personnel DB for BSchiefer Co″ という関連する注釈を付けて作成します。

```
CREATE DATABASE personl
  WITH "Personnel DB for BSchiefer Co"
```

構成アドバイザー

構成アドバイザーは、どの構成パラメーターを修正したらよいかを提案し、それらの推奨値を示すことによって、インスタンスごとの単一データベースのパフォーマンスのチューニングとメモリー所要量のバランスをとるよう支援します。データベースを作成すると、構成アドバイザーが自動的に起動し

ます。このフィーチャーを使用不可にしたり、明示的に使用可能にしたりするには、データベースを作成する前に db2set コマンドを使用します。次に例を示します。

```
db2set DB2_ENABLE_AUTOCONFIG_DEFAULT=NO
db2set DB2_ENABLE_AUTOCONFIG_DEFAULT=YES
```

デフォルトで使用可能に設定されている他のフィーチャーについて詳しくは、「 19 ページの『自動フィーチャー』」を参照してください。

イベント・モニター

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・モニターも作成されます。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。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
```

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びアクティブになります。

リモート・データベース

データベースを別 (リモートも可能) のインスタンスに作成することができます。データベースを別の (リモート) データベース・パーティション・サーバーに作成するには、最初にそのサーバーに接続する必要があります。処理中に、次のコマンドによりデータベース接続が一時的に確立されます。

```
CREATE DATABASE <database name> AT DBPARTITIONNUM <options>
```

このタイプの環境では、デフォルト・インスタンス以外のインスタンス (リモート・インスタンスを含む) に対してインスタンス・レベルの管理を実行することができます。これを行うための指示については、db2iupdt (インスタンスの更新) コマンドを参照してください。

データベース・コード・ページ

デフォルトでは、データベースは UTF-8 (Unicode) コード・セットで作成されます。

データベースのデフォルトのコード・ページをオーバーライドするには、データベースの作成時に希望するコード・セットおよびテリトリーを指定する必要があります。コード・セットおよびテリトリーの設定については、CREATE DATABASE コマンドまたは sqlecrea API を参照してください。

自動ストレージ・データベース

データベース・マネージャーにより、すべてのデータベースがデフォルトでは「自動ストレージ」データベースとして作成されます。「自動ストレージ」データベースではないデータベースを作成するには、CREATE DATABASE コマンドを発行するときに AUTOMATIC STORAGE NO を指定します。

自動ストレージが使用可能なデータベースには 1 つ以上のストレージ・パスのセットが関連付けられています。表スペースは、*自動ストレージによる管理* を受けるものとして定義でき、そのコンテナーはストレージ・パスに基づいてデータベース・マネージャーにより割り当ておよび割り振りが行われます。

自動ストレージ用にデータベースを有効にできるのは、その作成時のみです。 同じように、本来自動ストレージを使用するように設計されたデータベースに対して、自動ストレージの無効化を行うことはできません。

デフォルトでは、すべてのデータベースが自動ストレージ・データベースとして作成されます。自動ストレージ・データベースではないデータベースを作成するには、CREATE DATABASE コマンドを発行するときに **AUTOMATIC STORAGE NO** を指定します。

自動ストレージを使用不可にする例を以下に示します。

```
CREATE DATABASE ASNODB1 AUTOMATIC STORAGE NO
CREATE DATABASE ASNODB2 AUTOMATIC STORAGE NO ON X:
```

明示的または暗黙的に使用可能にされる自動ストレージの例:

```
CREATE DATABASE DB1
CREATE DATABASE DB2 AUTOMATIC STORAGE YES ON X:
CREATE DATABASE DB3 ON /data/path1, /data/path2
CREATE DATABASE DB4 ON D:¥StoragePath DBPATH ON C:
```

使用される構文に基づいて、データベース・マネージャーはストレージ・ロケーションに関する以下の 2 つの情報を抽出します。

- データベース・パス (データベース・マネージャーがデータベース用のさまざまな制御ファイルを保管する場所):
  - **DBPATH ON** を指定した場合、これはデータベース・パスを指します。
  - **DBPATH ON** を指定しない場合、**ON** でリストされた最初のパスがデータベース・パス (およびストレージ・パス) を指します。
  - **DBPATH ON** と **ON** のどちらも指定しない場合は、データベース・パスの決定に **dftdbpath** データベース・マネージャー構成パラメーターが使用されます。
- ストレージ・パス (データベース・マネージャーが自動ストレージの表スペース・コンテナーを作成する場所):
  - **ON** を指定した場合は、リストされたすべてのパスがストレージ・パスになります。
  - **ON** を指定しない場合は、ストレージ・パスは **dftdbpath** データベース・マネージャー構成パラメーターの値に設定される 1 つのみになります。

上記の例に関して、使用されるデータベース・パスおよびストレージ・パスを以下の表に要約します。

表 40. 自動ストレージ・データベースとストレージ・パス

CREATE DATABASE コマンド	データベース・パス	ストレージ・パス
CREATE DATABASE DB1 AUTOMATIC STORAGE YES	**dftdbpath** 構成パラメーターの値	**dftdbpath** 構成パラメーターの値

表 40. 自動ストレージ・データベースとストレージ・パス (続き)

CREATE DATABASE コマンド	データベース・パス	ストレージ・パス
CREATE DATABASE DB2 AUTOMATIC STORAGE YES ON X:	X:	X:
CREATE DATABASE DB3 ON /data/path1, /data/path2	/data/path1	/data/path1、/data/path2
CREATE DATABASE DB4 ON D:¥StoragePath DBPATH ON C:	C:	D:¥StoragePath

提供されるストレージ・パスが存在していなければならず、それらはアクセス可能でなければなりません。パーティション・データベース環境では、各データベース・パーティションに対して同じストレージ・パスが使用されます。データベース・パーティション式をストレージ・パス名の一部として使用しない場合には、特定のデータベース・パーティションに固有のストレージ・パスのセットを指定できません。データベース・パーティション式をストレージ・パス名の一部として使用すると、ストレージ・パスにデータベース・パーティション番号が反映され、各データベース・パーティションごとに異なるパス名が付けられます。

データベース・パーティション式を指示するには、引数 $N ($N の前に 1 つの半角ブランクあり) を使用します。データベース・パーティション式はストレージ・パス内のどこにでも使用でき、複数指定することも可能です。データベース・パーティション式はスペース文字で終了します。スペースの後に続く文字はすべて、データベース・パーティション式が評価された後、ストレージ・パスに付加されます。ストレージ・パス内でデータベース・パーティション式の後にスペース文字がない場合、ストリングの残りは式の一部であると見なされます。下の表に、$N 引数の有効な形式のみをリストします。演算子は左から右に向かって評価され、% はモジュラス演算子を表します。例の中のデータベース・パーティション番号は 10 です。

表 41. データベース・パーティション式

構文	例	値
[blank]$N	" $N"	10
[blank]$N+[number]	" $N+100"	110
[blank]$N%[number]	" $N%5"	0
[blank]$N+[number]%[number]	" $N+1%5"	1
[blank]$N%[number]+[number]	" $N%4+2"	4

データベース・パーティション式の使用例を以下に示します。

```
CREATE DATABASE TESTDB ON "/path1ForNode $N",
  "/path2ForNode $N" DBPATH ON "/dbpathForNodes"
```

パスの間にデータベース・パーティション式を埋め込んだ例を示します。

```
CREATE DATABASE TESTDB ON "/path1ForNode $N",
  "/path2ForNode $N suffix" DBPATH ON "/dbpathForNodes"
```

注: データベース・パーティション式は、**DBPATH ON**で明示的に指定しているか、最初のストレージ・パスにデータベース・パーティション式を使用して暗黙的に指定しているかに関係なく、データベース・パス内では無効です。

特定のデータベース・パーティションのストレージ・パスのフリー・スペースが計算されると、データベース・マネージャーは、ストレージ・パスの中に以下のディレクトリーまたはマウント・ポイントがないか調べ、見つかった最初のものを使用します。

```
storage path/instance name/NODE####/database name
storage path/instance name/NODE####
storage path/instance name
storage path
```

説明:

storage path
データベースに関連付けられたストレージ・パス

instance name
データベースが存在するインスタンス

NODE####
データベース・パーティション番号 (例えば、NODE0000 または NODE0001)

database name
データベースの名前

ファイル・システムをストレージ・パスの下の位置にマウントすることができます。データベース・マネージャーは、表スペース・コンテナーで使用可能なフリー・スペースの実際の量がストレージ・パスのディレクトリー自体に関連付けられている量と同じでない可能性があることを認識します。

1 つの物理コンピューターに 2 つの論理データベース・パーティションが存在し、1 つのストレージ・パス /db2data がある例について考えてみましょう。データベース・パーティションはそれぞれこのストレージ・パスを使用できますが、各パーティションごとに別のファイル・システムを作成して、データを各パーティションから分離することもできます。ファイル・システムは、/db2data/*instance*/NODE#### にマウントされます。ストレージ・パス上にコンテナーを作成してフリー・スペースを決定する際、データベース・マネージャーは、/db2data のフリー・スペース情報を取得しませんが、代わりに対応する /db2data/*instance*/NODE#### ディレクトリーのフリー・スペース情報を取得します。

データベースを作成するときには必ず 3 つのデフォルト表スペースが作成されます。 CREATE DATABASE コマンドの一部として明示的な表スペース定義を指定しない場合は、表スペースは自動ストレージの表スペースとして作成されます。

データベースを作成した後、次の例に示すように ALTER DATABASE ステートメントの ADD STORAGE 節を使用してデータベースに新規ストレージ・パスを追加できます。

```
ALTER DATABASE ADD STORAGE ON '/data/path3', '/data/path4'
```

自動ストレージの制約事項

自動ストレージを使用してデータベースを作成するかどうかを決定する際、いくつかの制約事項を考慮する必要があります。

- データベースの作成後にデータベースで自動ストレージを使用不可または使用可能にすることはできません。
- ストレージ・パスは絶対パス名でなければなりません。 Windows オペレーティング・システムでは、ストレージ・パスにパスまたはドライブ名を使用できます。データベース・パスはドライブ名である必要があります。 最大パス長は 175 文字です。
- パーティション・データベースでは、各データベース・パーティションで同じストレージ・パスのセットを使用する必要があります (データベース・パーティション式を使用しない場合)。
- データベース・パーティション式は、CREATE DATABASE の **DBPATH ON** オプションを使用して明示的に指定しているか、最初のストレージ・パスにデータベース・パーティション式を使用して暗黙的に指定しているかに関係なく、データベース・パス内では無効です。

自動ストレージを使用できるデータベースへの自動ストレージ・パスの追加

ALTER DATABASE を使用して、自動ストレージが有効になっているデータベースに自動ストレージ・パスを追加することができます。自動ストレージ用にデータベースを有効にできるのは、その作成時のみです。

複数パーティション・データベース環境用のストレージ・パスを追加する場合、各データベース・パーティションでそのストレージ・パスが存在する必要があります。指定されたパスがすべてのデータベース・パーティションに存在していない場合、ステートメントはロールバックされます。

既存のデータベースにストレージ・パスを追加するには、次のような ALTER DATABASE ステートメントを発行します。

```
ALTER DATABASE PATH pathname
```

ストレージ・パスのモニター

データベースのスナップショットには、データベースに関連したストレージ・パスのリストが含まれます。

以下において、自動ストレージ・パスの数が 0 の場合、データベースで自動ストレージが使用可能になっていないことを表します。

```
Number of automatic storage paths  = ##
Automatic storage path             = <1st path>
Automatic storage path             = <2nd path>
...
```

バッファー・プールのモニター・スイッチがオンの場合、以下のエレメントも設定されます。

```
File system ID                     = 12345
File system free space (bytes)     = 20000000000
File system used space (bytes)     = 40000000000000
File system total space (bytes)    = 40020000000000
```

このデータはパス単位で設定されます。単一データベース・パーティション・システムではパスごと、複数データベース・パーティション環境では各データベース・パーティションごとです。

さらに、表スペースのスナップショット内に次の情報が設定されます。この情報は、表スペースが自動ストレージの表スペースとして作成されたかどうかを示します。

```
Using automatic storage            = Yes or No
```

RESTORE DATABASE の含意

RESTORE DATABASE コマンドは、バックアップ・イメージからデータベースをリストアするために使用されます。

リストア操作中に、データベース・パスのロケーションを選択したり、またデータベースに関連したストレージ・パスを再定義したりすることができます。データベース・パスとストレージ・パスは、RESTORE DATABASE コマンドで TO、ON、および DBPATH ON の組み合わせを使用することにより設定されます。

例えば、自動ストレージが使用可能なデータベースの有効な RESTORE コマンドには次のものがあります。

```
RESTORE DATABASE TEST1
RESTORE DATABASE TEST2 TO X:
RESTORE DATABASE TEST3 DBPATH ON D:
RESTORE DATABASE TEST3 ON /path1, /path2, /path3
RESTORE DATABASE TEST4 ON E:¥newpath1, F:¥newpath2 DBPATH ON D:
```

CREATE DATABASE コマンドと同様、データベース・マネージャーはストレージ・ロケーションに関する以下の 2 つの情報を抽出します。

- **データベース・パス** (データベース・マネージャーがデータベース用のさまざまな制御ファイルを保管する場所)
  - TO または DBPATH ON が指定される場合、これはデータベース・パスを指します。
  - ON が使用され、DBPATH ON が指定されない場合、ON でリストされる最初のパスはデータベース・パスとして使用されます (これは同時にストレージ・パスでもあります)。
  - TO、ON、または DBPATH ON のうちどの節も指定されない場合、*dftdbpath* データベース・マネージャー構成パラメーターがデータベース・パスを決定します。

  注: 同じ名前のデータベースがディスク上に存在する場合、データベース・パスは無視され、データベースは既存のデータベースと同じロケーションに置かれます。
- **ストレージ・パス** (データベース・マネージャーが自動ストレージの表スペース・コンテナーを作成する場所)
  - ON が指定される場合、リストされるパスはすべてストレージ・パスと見なされ、これらのパスはバックアップ・イメージの中に保管されるパスの代わりに使用されます。
  - ON が指定されない場合、ストレージ・パスに対して変更は行われません (バックアップ・イメージ内に保管されるストレージ・パスは維持されます)。

この概念を分かりやすくするために、上記と同じ 5 つの RESTORE コマンドの例が、それに対応するストレージ・パスとともに以下の表で示されています。

表 42. データベースとストレージ・パスに関するリストアの影響

RESTORE DATABASE コマンド	同じ名前のデータベースがディスク上に存在しない		同じ名前のデータベースがディスク上に存在する	
	データベース・パス	ストレージ・パス	データベース・パス	ストレージ・パス
RESTORE DATABASE TEST1	<*dftdbpath*>	バックアップ・イメージに定義されているストレージ・パスを使用する	既存のデータベースのデータベース・パスを使用する	バックアップ・イメージに定義されているストレージ・パスを使用する
RESTORE DATABASE TEST2 TO X:	X:	バックアップ・イメージに定義されているストレージ・パスを使用する	既存のデータベースのデータベース・パスを使用する	バックアップ・イメージに定義されているストレージ・パスを使用する
RESTORE DATABASE TEST3 DBPATH ON /db2/databases	/db2/databases	バックアップ・イメージに定義されているストレージ・パスを使用する	既存のデータベースのデータベース・パスを使用する	バックアップ・イメージに定義されているストレージ・パスを使用する
RESTORE DATABASE TEST4 ON /path1, /path2, /path3	/path1	/path1, /path2, /path3	既存のデータベースのデータベース・パスを使用する	/path1, /path2, /path3
RESTORE DATABASE TEST5 ON E:¥newpath1, F:¥newpath2 DBPATH ON D:	D:	E:¥newpath1、F:¥newpath2	既存のデータベースのデータベース・パスを使用する	E:¥newpath1、F:¥newpath2

ストレージ・パスがリストア操作の一部として再定義された場合、自動ストレージを使用するように定義されている表スペースは自動的に新規パスにリダイレクトされます。ただし、SET TABLESPACE CONTAINERS コマンドを使用して自動ストレージの表スペースに関連したコンテナーを明示的にリダイレクトすることはできません。このアクションは許可されていません。

バックアップ・イメージ内のデータベースで自動ストレージが使用可能になっているかどうかを表示するには、db2ckbkp コマンドの -s オプションを使用します。自動ストレージが使用可能な場合、データベースに関連したストレージ・パスが表示されます。

複数パーティション自動ストレージ対応のデータベースでは、RESTORE DATABASE コマンドに他の含意がいくつかあります。

1. データベースはすべてのデータベース・パーティション上で同じストレージ・パスのセットを使用しなければなりません。

2. 新規のストレージ・パスを使用した RESTORE コマンドの発行は、カタログ・データベース・パーティション上でのみ行えます。これにより、すべての非カタログ・データベース・パーティション上でデータベースの状態が RESTORE_PENDING に設定されます。

表 43. 複数パーティション・データベースでのリストアの含意

RESTORE DATABASE コマンド	データベース・パーティション # で発行される	同じ名前のデータベースがディスク上に存在しない		同じ名前のデータベースがディスク上に存在する (スケルトン・データベースを含む)	
		他のデータベース・パーティションでの結果	ストレージ・パス	他のデータベース・パーティションでの結果	ストレージ・パス
RESTORE DATABASE TEST1	カタログ・データベース・パーティション	カタログ・データベース・パーティション上で、ストレージ・パスを使用してバックアップ・イメージからスケルトン・データベースが作成される。他のデータベース・パーティションはすべて、RESTORE_PENDING 状態に置かれる。	バックアップ・イメージに定義されているストレージ・パスを使用する	なし。ストレージ・パスが変更されないため、他のデータベース・パーティションでは何も起こらない。	バックアップ・イメージに定義されているストレージ・パスを使用する
	非カタログ・データベース・パーティション	SQL2542N または SQL2551N が戻される。データベースが存在しない場合、カタログ・データベース・パーティションがまずリストアされなければならない。	N/A	なし。ストレージ・パスが変更されないため、他のデータベース・パーティションでは何も起こらない。	バックアップ・イメージに定義されているストレージ・パスを使用する

表 43. 複数パーティション・データベースでのリストアの含意 (続き)

RESTORE DATABASE コマンド	データベース・パーティション # で発行される	同じ名前のデータベースがディスク上に存在しない		同じ名前のデータベースがディスク上に存在する (スケルトン・データベースを含む)	
		他のデータベース・パーティションでの結果	ストレージ・パス	他のデータベース・パーティションでの結果	ストレージ・パス
RESTORE DATABASE TEST2 ON /path1, /path2, /path3	カタログ・データベース・パーティション	RESTORE コマンドで指定されたストレージ・パスを使用してスケルトン・データベースが作成される。他のデータベース・パーティションはすべて、RESTORE_ PENDING 状態に置かれる。	/path1, /path2, /path3		/path1, /path2, /path3
	非カタログ・データベース・パーティション	SQL1174N が戻される。データベースが存在しない場合、カタログ・データベース・パーティションがまずリストアされなければならない。非カタログ・データベース・パーティションの RESTORE にはストレージ・パスを指定できない。	N/A	SQL1172N が戻される。非カタログ・データベース・パーティションの RESTORE には新規のストレージ・パスを指定できない。	N/A

データベースのカタログ作成

新しいデータベースを作成すると、システム・データベース・ディレクトリーのファイルに自動的にカタログされます。また、CATALOG DATABASE コマンドを使って、システム・データベース・ディレクトリーのファイルにデータベースを明示的にカタログすることもできます。

CATALOG DATABASE コマンドを使えば、違う別名でデータベースをカタログしたり、 UNCATALOG DATABASE コマンドによって以前に削除したデータベース項目をカタログしたりすることが可能になります。

データベースは、データベース作成時に自動的にカタログされますが、それでもデータベースをカタログすることが必要になる場合があります。これを実行する場合、データベースが存在していることが必要です。

デフォルトで、データベース・ディレクトリーを含むディレクトリー・ファイルは、「ディレクトリー・キャッシュ・サポート (*dir_cache*)」構成パラメーターを使ってメモリーのキャッシュに入れられます。ディレクトリー・キャッシュが使用可能な場合は、別のアプリケーションがディレクトリーを変更しても (例えば、CATALOG DATABASE または UNCATALOG DATABASE コマンドを使用して)、ユーザーのアプリケーションが再始動されるまで、その変更は有効になりません。コマンド行プロセッサー・セッションが使用するディレクトリー・キャッシュをリフレッシュするには、TERMINATE コマンドを発行してください。

パーティション・データベースでは、ディレクトリー・ファイルのキャッシュは各データベース・パーティションで作成されます。

アプリケーション・レベルのキャッシュに加えて、データベース・マネージャー・レベルのキャッシュも、内部的なデータベース・マネージャーの検索に使用されます。この「共用」キャッシュをリフレッシュするには、db2stop および db2start コマンドを出してください。

コマンド行プロセッサーを使用して異なる別名でデータベースのカタログを作成するには、 CATALOG DATABASE コマンドを使用します。例えば、次のコマンド行プロセッサー・コマンドを使うと、PERSON1 データベースが HUMANRES としてカタログされます。

```
CATALOG DATABASE personl AS humanres
  WITH "Human Resources Database"
```

この場合、システム・データベース・ディレクトリー項目のデータベース別名は HUMANRES になります。これは、データベース名 (PERSON1) とは違うものです。

クライアント・アプリケーションからシステム・データベース・ディレクトリーにデータベースをカタログするには、sqlecadb API を呼び出します。

コマンド行プロセッサーを使用してデフォルト以外のインスタンス上でデータベースのカタログを作成するには、CATALOG DATABASE コマンドを使用します。次の例では、データベース B への接続は、INSTNC_C に対して行われます。インスタンス instnc_c を、このコマンド試行前に、ローカル・ノードとしてカタログしておくことが必要です。

```
CATALOG DATABASE b as b_on_ic AT NODE instnc_c
```

注: クライアント・ノードで CATALOG DATABASE コマンドを使うと、データベース・サーバー・コンピューターにあるデータベースをカタログすることもできます。

データベースへのユーティリティーのバインド

データベースの作成時に、データベース・マネージャーにより、db2ubind.lst および db2cli.lst 内のユーティリティーのデータベースへのバインドが試行されます。これらのファイルは、sqllib ディレクトリーの bnd サブディレクトリーに格納されています。

ユーティリティーをバインドすると、パッケージ が作成されます。これは、1 つのソース・ファイルからの特定の SQL および XQuery ステートメントを処理するのに必要な情報がすべて入れられているオブジェクトです。

注: クライアントからこれらのユーティリティーを使用する場合は、ユーティリティーを明示的にバインドする必要があります。sample データベースにパッケージを作成するには、これらのファイルが入っているディレクトリーを使用しなければなりません。バインド・ファイルは、 sqllib ディレクトリーの bnd サブディレクトリーの中にあります。

データベースにユーティリティーをバインドまたは再バインドするには、コマンド行から以下のコマンドを呼び出します。ここで、sample はデータベースの名前です。

```
connect to sample
bind @db2ubind.lst
```

データベース別名の作成

別名 は、表、ニックネーム、またはビューを間接的に参照して、SQL または XQuery ステートメントを表やビューの修飾名とは無関係なものにするための手段です。

表名やビュー名を変更しても、別名の定義を変えるだけで済みます。別名は他の別名に対して作成することもできます。別名は、ビューやトリガーの定義、また SQL または XQuery ステートメントの中で使用できます。ただし、既存の表名やビュー名を参照する表チェック制約の定義では使用できません。

別名は、定義時に存在していない表、ビュー、または別名に対しても定義できます。しかし、別名の含まれる SQL または XQuery ステートメントのコンパイル時には、存在していなければなりません。

別名は既存の表名が使用できる所ならどこにでも使用できます。また、別名のチェーンに循環参照または反復参照がない限り他の別名を参照することもできます。

既存の表、ビュー、別名と同じ別名を作成することはできません。また、別名は同じデータベース内の表しか参照できません。 CREATE TABLE ステートメントまたは CREATE VIEW ステートメントでは、同じスキーマ内の別名と同じ表名やビュー名は使用できません。

別名の作成には特別な権限は必要ありません。ただし、自分の現在の許可 ID が所有するスキーマ以外のスキーマに別名を作成する場合は、 DBADM 権限が必要です。

別名または別名が参照するオブジェクトがドロップされると、その別名に依存するすべてのパッケージは無効のマークが付けられ、その別名に依存するすべてのビューおよびトリガーは作動不能のマークが付けられます。

コマンド行を使用して別名を作成するには、以下のように入力します。

```
CREATE ALIAS <alias_name> FOR <table_name>
```

別名は、ステートメントのコンパイル時に表名やビュー名に置き換えられます。別名または別名のチェーンが表名やビュー名に置換できないと、エラーになります。例えば、WORKERS を EMPLOYEE の別名にした場合、コンパイル時には、

```
SELECT * FROM WORKERS
```

は、実際には次のものになります。

```
SELECT * FROM EMPLOYEE
```

次の SQL ステートメントは、EMPLOYEE 表に WORKERS という別名を作成するものです。

```
CREATE ALIAS WORKERS FOR EMPLOYEE
```

注: DB2 for OS/390 and z/Series は、ALIAS と SYNONYM という、別名についての 2 つの異なる概念を採用しています。これらの 2 つの概念は、DB2 データベースと以下の点で異なります。

- DB2 for OS/390 and z/Series の ALIAS
  - 作成者が特殊権限または特権を有していなければならない。
  - 他の別名を参照できない。
- DB2 for OS/390 and z/Series の SYNONYM
  - 作成者だけしか使用できない。
  - 常に修飾なしである。
  - 参照テーブルがドロップされると、ドロップされる。
  - ネーム・スペースを表またはビューと共用しない。

分散リレーショナル・データベースへの接続

分散リレーショナル・データベースは、正式なリクエスター/サーバー・プロトコルと機能に基づいて構築されます。

アプリケーション・リクエスター は、接続の両端のうち、アプリケーション側をサポートするものです。アプリケーション・リクエスターは、アプリケーションからのデータベース要求を分散データベース・ネットワークでの使用に適した通信プロトコルに変換します。これらの要求は、接続のもう一方の端のデータベース・サーバー によって受信され、処理されます。アプリケーション・リクエスターとデータベース・サーバーは連携して通信とロケーションに関する考慮事項を処理し、アプリケーションがローカル・データベースにアクセスしているのと変わりなく稼働できるようにします。

表やビューを参照する SQL ステートメントを実行できるようにするためには、その前に、データベース・マネージャーのアプリケーション・サーバーにアプリケー

ション・プロセスを接続しておく必要があります。 CONNECT ステートメントにより、アプリケーション・プロセスとそのサーバーの接続が確立されます。

CONNECT ステートメントには、次の 2 つのタイプがあります。

- CONNECT (タイプ 1) では、作業単位 (リモート作業単位) セマンティクスごとに 1 つのデータベースがサポートされます。
- CONNECT (タイプ 2) では、作業単位 (アプリケーション制御の分散作業単位) セマンティクスごとに複数のデータベースがサポートされます。

DB2 コール・レベル・インターフェース (CLI) および組み込み SQL は、*並行トランザクション* と呼ばれる接続モードに対応しています。このモードでは、複数の接続が可能で、各接続が独立したトランザクションになります。1 つのアプリケーションに、同じデータベースへの複数の同時接続を持たせることができます。

アプリケーション・サーバーは、プロセスが開始される環境に対してローカルでもリモートでもかまいません。アプリケーション・サーバーは、分散リレーショナル・データベースを使用していない環境でも存在しています。この環境には、CONNECT ステートメントに指定されるアプリケーション・サーバーを記述するローカル・ディレクトリーが組み込まれています。

アプリケーション・サーバーは、表やビューを参照するバインドされた形式の静的 SQL ステートメントを実行します。このバインドされたステートメントは、データベース・マネージャーがバインド操作でそれ以前に作成したパッケージから取り出されます。

ほとんどの場合、アプリケーション・サーバーに接続しているアプリケーションは、そのアプリケーション・サーバーのデータベース・マネージャーでサポートされているステートメントや節を使用できます。このことは、一部のステートメントや節をサポートしないデータベース・マネージャーのアプリケーション・リクエスターによってアプリケーションが実行される場合でも当てはまります。

分散リレーショナル・データベースのリモート作業単位

リモート作業単位機能 では、SQL ステートメントの準備と実行をリモートで行えます。

コンピューター・システム「A」のアプリケーション・プロセスは、コンピューター・システム「B」のアプリケーション・サーバーに接続することができ、1 つ以上の作業単位内で、「B」のオブジェクトを参照する静的または動的 SQL ステートメントをいくつでも実行することができます。B の作業単位が終了した後に、アプリケーション・プロセスはコンピューター・システム C のアプリケーション・サーバーに接続することができ、これをさらに広げていくことができます。

ほとんどの SQL ステートメントは、リモートで準備して実行することができますが、以下の制約事項があります。

- 1 つの SQL ステートメントで参照されるオブジェクトはすべて、同じアプリケーション・サーバーによって管理されなければなりません。
- 同じ作業単位にあるすべての SQL ステートメントは、同じアプリケーション・サーバーによって実行されなければなりません。

どの時点においても、アプリケーション・プロセスは次の 4 つの可能な*接続状態*のうちの 1 つになります。

- 接続可能で接続済み

  アプリケーション・プロセスがアプリケーション・サーバーに接続しており、CONNECT ステートメントが実行可能です。

  暗黙的接続が可能な場合:

  – CONNECT TO ステートメントもしくはオペランドなしの CONNECT ステートメントが正常に実行されると、アプリケーション・プロセスが接続可能で未接続状態からこの状態になります。
  – CONNECT RESET、DISCONNECT、SET CONNECTION、または RELEASE 以外の SQL ステートメントが発行されると、アプリケーション・プロセスが暗黙的接続可能状態からこの状態になることもあります。

  暗黙的接続が使用可能かどうかに関係なく、以下の場合にこの状態になります。

  – 接続可能で未接続状態から CONNECT TO ステートメントが正常に実行されたとき
  – 接続不可で接続済み状態から COMMIT ステートメントまたは ROLLBACK ステートメントが正常に実行された、もしくは強制ロールバックが発生したとき

- 接続不可で接続済み

  アプリケーション・プロセスがアプリケーション・サーバーに接続されていますが、アプリケーション・サーバーを変更するために CONNECT TO ステートメントを正常に実行することはできません。次のステートメント以外の SQL ステートメントを実行すると、アプリケーション・プロセスが接続可能で接続済み状態からこの状態になります: CONNECT TO、オペランドなしの CONNECT、CONNECT RESET、DISCONNECT、SET CONNECTION、RELEASE、COMMIT、または ROLLBACK。

- 接続可能で未接続

  アプリケーション・プロセスがアプリケーション・サーバーに接続されていません。実行可能な SQL ステートメントは、CONNECT TO だけです。それ以外はエラー (SQLSTATE 08003) が発生します。

  暗黙的接続が可能かどうかに関係なく、CONNECT TO ステートメントの発行時にエラーが起きた場合、もしくは、作業単位内でエラーが発生し、接続とロールバックが失われた場合に、アプリケーション・プロセスはこの状態になります。アプリケーション・プロセスが接続可能状態にない、もしくはサーバー名がローカル・ディレクトリーのリストにないという理由でエラーが発生しても、この状態に変わることはありません。

  暗黙的接続が可能ではない場合:

  – これが、アプリケーション・プロセスの初期状態です。
  – CONNECT RESET および DISCONNECT ステートメントで、この状態に変わります。

- 暗黙的接続可能 (暗黙的接続が使用可能な場合)

暗黙的接続が可能であれば、これがアプリケーション・プロセスの初期状態になります。 CONNECT RESET ステートメントで、この状態に変わります。接続不可で接続済み状態で COMMIT もしくは ROLLBACK ステートメントを発行し、次に接続可能で接続済み状態で DISCONNECT ステートメントが続く場合も、この状態になります。

暗黙的接続の可用性は、インストール・オプション、環境変数、および認証設定によって決まります。

連続して CONNECT ステートメントを実行してもエラーにはなりません。これは、CONNECT 自体がアプリケーション・プロセスを接続可能状態から解除することはないからです。ただし、連続して CONNECT RESET ステートメントを実行するとエラーになります。また、CONNECT TO、CONNECT RESET、オペランドなしの CONNECT、SET CONNECTION、RELEASE、COMMIT、または ROLLBACK 以外の SQL ステートメントを実行し、それから CONNECT TO ステートメントを実行すると、エラーになります。このエラーを避けるには、CONNECT RESET、DISCONNECT (COMMIT または ROLLBACK ステートメントが先行)、COMMIT、または ROLLBACK ステートメントを、CONNECT TO ステートメントより前に実行してください。

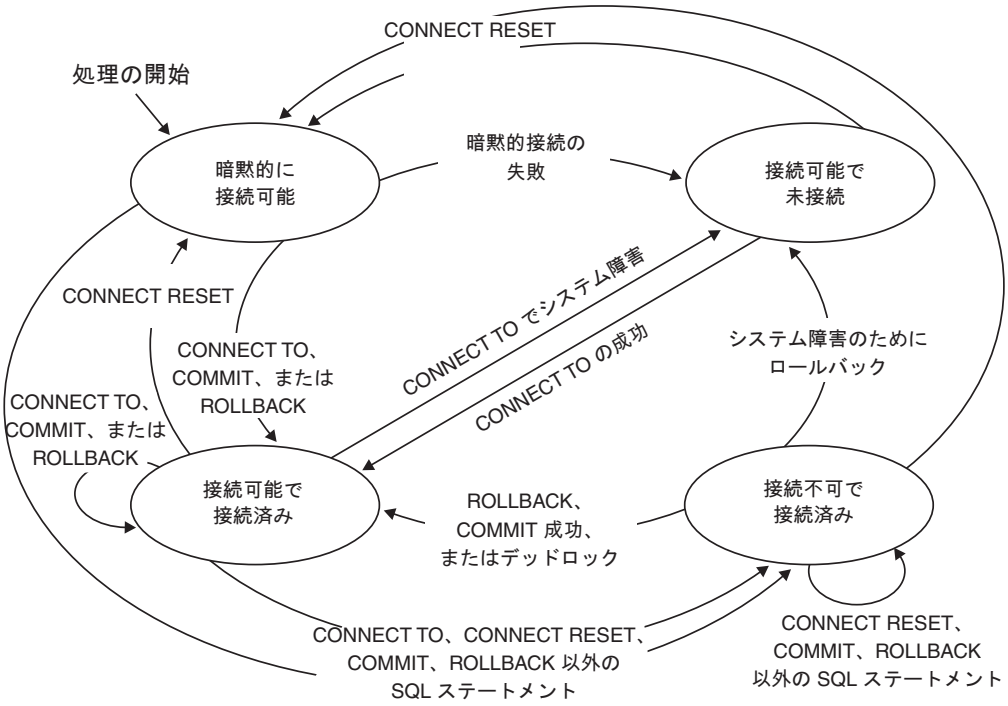


図 3. 暗黙的接続が可能な場合の接続状態遷移

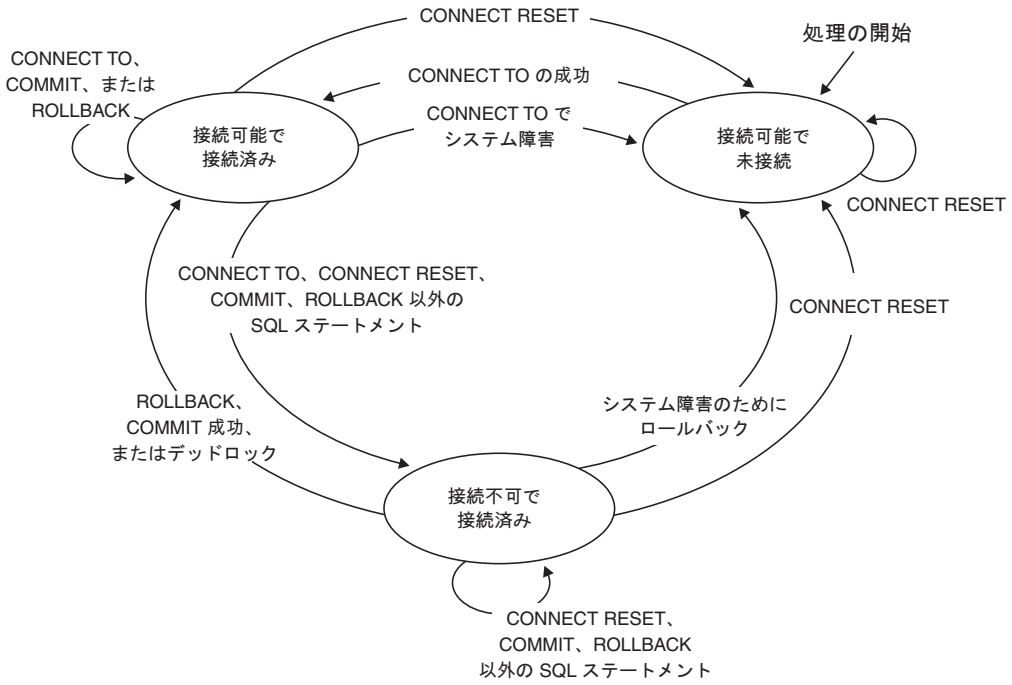


図 4. 暗黙的接続が可能でない場合の接続状態遷移

アプリケーション制御の分散作業単位

アプリケーション制御の分散作業単位機能 では、SQL ステートメントの準備と実行をリモートで行えます。

CONNECT または SET CONNECTION ステートメントを発行することで、コンピューター・システム「A」のアプリケーション・プロセスが、コンピューター・システム「B」のアプリケーション・サーバーに接続することができます。そうすると、アプリケーション・プロセスが、作業単位を終了させる前に「B」にあるオブジェクトを参照する静的および動的 SQL ステートメントをいくつでも実行することができます。1 つの SQL ステートメントで参照されるオブジェクトはすべて、同じアプリケーション・サーバーによって管理されなければなりません。ただし、リモート作業単位機能とは異なり、アプリケーション・サーバーはいくつでも同じ作業単位に加わることができます。コミットもしくはロールバック操作により、作業単位が終了します。

アプリケーション制御の分散作業単位では、タイプ 2 の接続が使用されます。*タイプ 2* 接続で、アプリケーション・プロセスを指定アプリケーション・サーバーに接続し、アプリケーション制御の分散作業単位の規則を設定します。

タイプ 2 のアプリケーション・プロセスは次のとおりです。

- 常時接続可能
- 接続済み状態または未接続状態のいずれか
- ゼロ以上の接続を持つ

アプリケーション・プロセスの各接続は、その接続のアプリケーション・サーバーのデータベース別名により一意的に識別されます。

個々の接続は、常に以下の接続状態のうちのいずれかになります。

- 現行で保持
- 現行で解放ペンディング
- 休止で保持
- 休止で解放ペンディング

タイプ 2 のアプリケーション・プロセスの初期状態は未接続で、接続は何もありません。 接続の初期状態は、現行で保持です。

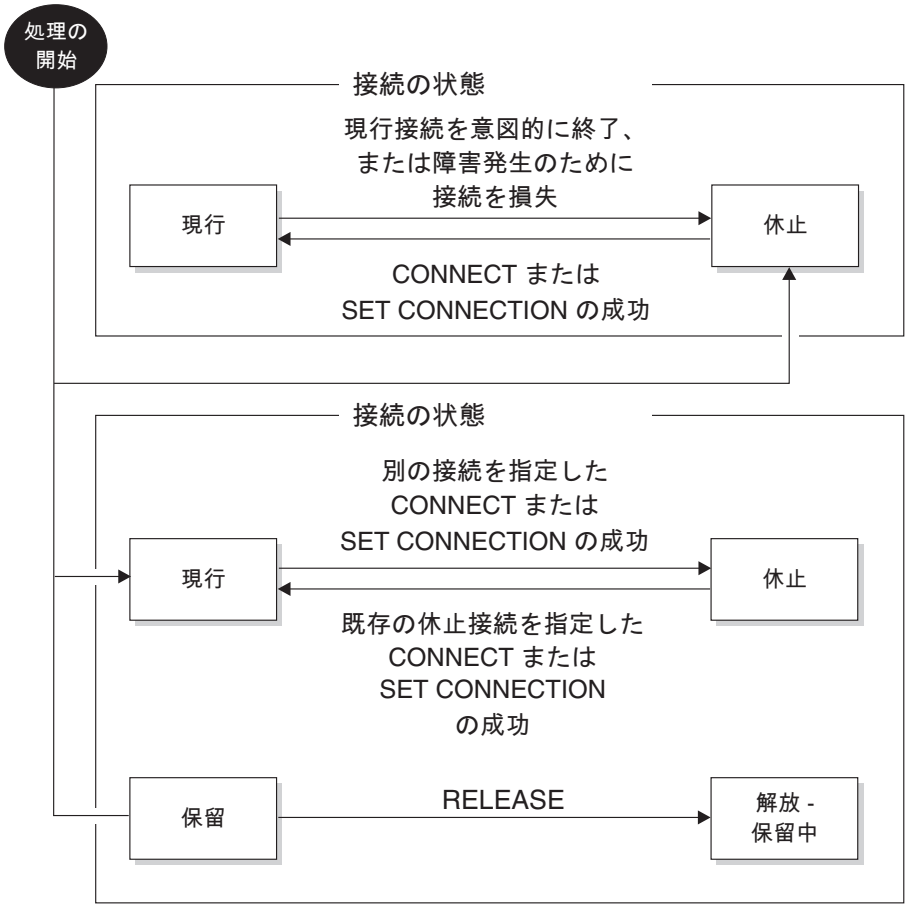


図 5. アプリケーション制御の分散作業単位の接続状態遷移

アプリケーション・プロセスの接続状態

CONNECT ステートメントの実行に適用される一定の規則があります。

以下の規則が CONNECT ステートメントの実行に適用されます。

- コンテキストは、同じアプリケーション・サーバーに対して同時に複数の接続を持つことができない。

- アプリケーション・プロセスが SET CONNECTION ステートメントを実行するときは、指定したロケーション名は、アプリケーション・プロセスの接続の中の既存の接続でなければならない。
- アプリケーション・プロセスが CONNECT ステートメントを実行するときに、SQLRULES(STD) オプションが有効である場合は、指定したサーバー名がアプリケーション・プロセスの接続の中の既存の接続であってはならない。SQLRULES オプションの説明については、 151 ページの『作業単位のセマンティクスを規制するオプション』 を参照してください。

アプリケーション・プロセスに現行接続がある場合、そのアプリケーション・プロセスは*接続済み* 状態です。CURRENT SERVER 特殊レジスターに、この現行接続のアプリケーション・サーバー名が入ります。アプリケーション・プロセスは、そのアプリケーション・サーバーが管理するオブジェクトを参照する SQL ステートメントを実行することができます。

CONNECT または SET CONNECTION ステートメントが正常に実行されると、未接続状態のアプリケーション・プロセスは、接続済み状態になります。接続がない状態で SQL ステートメントが発行される場合、DB2DBDFT 環境変数にデフォルトのデータベース名が設定されていると、暗黙的接続が作られます。

アプリケーション・プロセスに現行接続がない場合、そのアプリケーション・プロセスは*未接続* 状態です。実行可能な SQL ステートメントは、CONNECT、DISCONNECT ALL、DISCONNECT (データベースを指定)、SET CONNECTION、RELEASE、COMMIT、ROLLBACK、およびローカル SET ステートメントだけです。

現行接続を意図的に終わらせたとき、もしくは、SQL ステートメントが失敗して、アプリケーション・サーバーでロールバック操作が起き、接続が失われたときに、*接続済み状態* のアプリケーション・プロセスが*未接続状態* に変わります。DISCONNECT ステートメント、または接続が解放ペンディング状態にあるときに COMMIT ステートメントが正常に実行されると、接続は意図的に終了します。(DISCONNECT プリコンパイラー・オプションが AUTOMATIC に設定されていると、すべての接続が終了します。CONDITIONAL に設定されている場合は、オープン WITH HOLD カーソルを持たない接続がすべて終了します。)

接続状態

接続状態には、「保持状態および解放ペンディング状態」と「現行状態および休止状態」という 2 つのタイプがあります。

アプリケーション・プロセスが CONNECT ステートメントを実行し、サーバー名がアプリケーション・リクエスターには知られているもののアプリケーション・プロセスの既存の接続の中にない場合: (i) 現行接続は*休止接続状態* になり、サーバー名が接続のセットに加えられ、その新しい接続は*現行接続状態* および*保持接続状態* になります。

サーバー名がアプリケーション・プロセスの既存の接続の中にすでにあり、アプリケーションが SQLRULES(STD) オプションを指定してプリコンパイルされている場合、エラー (SQLSTATE 08002) が起きます。

保持状態および解放ペンディング状態 接続を保持状態か解放ペンディング状態にするのを制御するのは、RELEASE ステートメントです。*解放*ペンディング 状態とは、次の正常なコミット操作で接続の切断が起こることを意味しています。(ロールバックをしても接続には何も影響しません。) *保持* 状態とは、次のコミット操作で接続が切断されない ことを意味しています。

初期のすべての接続は保持状態で、RELEASE ステートメントを使って解放ペンディング状態にすることができます。いったん解放ペンディング状態になると、接続を保持状態に戻すことはできません。ROLLBACK ステートメントが発行された場合、またはコミット操作が正常に行われずにロールバック操作が発生した場合でも、接続は作業単位の境界を越えて解放ペンディング状態のままになります。

接続が解放のために明示的にマークされていなくても、コミット操作が DISCONNECT プリコンパイラー・オプションの条件を満たしていれば、コミット操作によって切断することができます。

現行状態および休止状態 これは、接続が保持状態または解放ペンディングのどちらであるかに関係なく、現行状態または休止状態にもなることができます。*現行* 状態にある接続は、この状態にある間に SQL ステートメントの実行に使用される接続のことです。*休止* 状態にある接続とは、現行でない接続のことです。

休止状態の接続で実行できる SQL ステートメントは、COMMIT、ROLLBACK、DISCONNECT、RELEASE のみです。 SET CONNECTION および CONNECT ステートメントは指定したサーバーの接続状態を現行状態に変え、既存の接続は休止状態になるか休止状態のままになります。どの時点でも、現行状態にある接続は 1 つだけです。同じ作業単位の中で休止接続が現行状態に変わると、すべてのロック、カーソル、作成済みステートメントの状態は最後にその接続が現行であったときと同じ状態になります。

接続の終了時

接続が終了すると、アプリケーション・プロセスが接続によって獲得していたすべてのリソース、および接続を確立し維持するために使用されたすべてのリソースが割り振り解除されます。例えば、アプリケーション・プロセスが RELEASE ステートメントを実行すると、その次のコミット操作で接続が終了するときにオープンしているカーソルがすべてクローズされます。

接続は、通信障害によっても終了することがあります。この接続が現行状態にあると、アプリケーション・プロセスは未接続状態になります。

アプリケーション・プロセスの接続はすべて、そのプロセスが終了するときに終了します。

作業単位のセマンティクスを規制するオプション

タイプ 2 接続管理のセマンティクスは、プリコンパイラーのオプション群によって決定されます。これらのオプションについて以下に要約します。デフォルト値は太字に下線を付けたテキストで示します。

- CONNECT (**1** | 2)。 CONNECT ステートメントが、タイプ 1 とタイプ 2 のどちらとして処理されるかを指定します。

- SQLRULES (**DB2** | STD)。タイプ 2 の CONNECT が、CONNECT による休止接続への切り替えを認める DB2 規則に従って処理されるのか、それともその切り替えを認めない SQL92 標準規則に従って処理されるのかを指定します。
- DISCONNECT (**EXPLICIT** | CONDITIONAL | AUTOMATIC)。以下のようにコミット操作の発生時に切断されるデータベース接続を指定します。
  - SQL の RELEASE ステートメントによって解放するよう明示的に指定されているデータベース接続 (EXPLICIT)
  - オープンされている WITH HOLD カーソルのないもの、および解放のためにマークされたもの (CONDITIONAL)
  - すべての接続 (AUTOMATIC)
- SYNCPOINT (**ONEPHASE** | TWOPHASE | NONE)。複数のデータベース接続にまたがってコミットまたはロールバックを調整する仕方を指定します。このオプションは無視され、後方互換性のためだけに含まれています。
  - 作業単位の中で更新を行えるのは 1 つのデータベースに対してだけです。他のデータベースはすべて読み取り専用です (ONEPHASE)。他のデータベースに対して更新しようとすると、エラー (SQLSTATE 25000) になります。
  - 実行時にトランザクション・マネージャー (TM) を使って、このプロトコルをサポートするデータベースの間で 2 フェーズ COMMIT を調整します (TWOPHASE)。
  - 2 フェーズ COMMIT を実行する TM を使用せず、単一更新・複数読み取りを強制しません (NONE)。 COMMIT または ROLLBACK ステートメントが実行されると、個々の COMMIT または ROLLBACK がすべてのデータベースに通知されます。 1 つ以上の ROLLBACK が失敗すると、エラー (SQLSTATE 58005) になります。 1 つ以上の COMMIT が失敗すると、別のエラー (SQLSTATE 40003) になります。

実行時に上記のいずれかのオプションをオーバーライドするには、SET CLIENT コマンドまたは sqlesetc アプリケーション・プログラミング・インターフェース (API) を使用してください。その現在の設定値は QUERY CLIENT コマンドまたは sqleqryc API を使用して取得できます。これらは SQL ステートメントではなく、さまざまなホスト言語およびコマンド行プロセッサー (CLP) で定義されている API であることに注意してください。

データ表記の考慮事項

システムが異なると、データを表現する方式も異なります。データをあるシステムから別のシステムへ移動する場合、データ変換が必要なことがあります。

DRDA をサポートする製品は、データを受け取る側のシステムで必要な変換を自動的に実行します。

数値データの変換を実行するには、データ・タイプと、そのデータ・タイプが送り側システムでどのように表現されるかという情報がシステムに必要です。文字ストリングの変換のためにはさらに情報が必要です。ストリングの変換は、データのコード・ページと、そのデータに対して実行する操作の両方に応じて変わります。文字変換は、IBM Character Data Representation Architecture (CDRA) に従って実行されます。文字変換の詳細については、「*Character Data Representation Architecture: Reference & Registry*」(SC09-2190-00) マニュアルを参照してください。

ローカルまたはシステムのデータベース・ディレクトリー・ファイルの表示

システム上のデータベースに関連した情報を表示するには、LIST DATABASE DIRECTORY コマンドを使用します。

ローカルまたはシステム・データベース・ディレクトリー・ファイルのどちらかを見る前に、インスタンスおよびデータベースを前もって作成しておく必要があります。

ローカル・データベース・ディレクトリー・ファイルの内容を見るためには、以下のコマンドを出します (ただし、<location> はデータベースの位置を指定します)。

```
LIST DATABASE DIRECTORY ON <location>
```

システム・データベース・ディレクトリー・ファイルの内容を見るためには、データベース・ディレクトリー・ファイルの位置を*指定せずに* LIST DATABASE DIRECTORY コマンドを出します。

データベースのドロップ

データベースをドロップすることは、データベース内のすべてのオブジェクト、コンテナー、関連ファイルが削除されるため、広範囲に及ぶ影響があります。データベースをドロップすると、そのデータベースはデータベース・ディレクトリーからドロップ (アンカタログ) されます。

コマンド行を使用してデータベースをドロップするには、以下のように入力します。

```
DROP DATABASE <name>
```

次のコマンドは、SAMPLE というデータベースを削除するものです。

```
DROP DATABASE SAMPLE
```

注: SAMPLE データベースをドロップした後で再び必要であることが分かった場合には、これは再作成できます。

クライアント・アプリケーションからデータベースをドロップするには、sqledrpd API を呼び出します。指定されたデータベース・パーティション・サーバーのデータベースをドロップするには、sqledpan API を呼び出します。

別名のドロップ

別名をドロップする場合、別名の記述がカタログから削除され、別名を参照するパッケージおよびキャッシュ済みの動的照会は無効となり、この別名に依存するビューとトリガーもすべて作動不能とマークされます。

別名をドロップするには、次のようにコマンド行から DROP ステートメントを発行します。

```
DROP ALIAS EMPLOYEE-ALIAS
```

第 7 章 データベース・パーティション

データベース・パーティション は、データベースの一部であり、それ自体のデータ、索引、構成ファイル、およびトランザクション・ログからなります。データベース・パーティションは、ノードまたはデータベース・ノードと呼ばれる場合があります。パーティション・データベース環境とは、データベース・パーティション全体へのデータの配分をサポートするデータベースのインストール済み環境です。

データベース・パーティションについて詳しくは、「*パーティショニングおよびクラスタリング・ガイド*」を参照してください。

第 8 章 バッファー・プール

バッファー・プール とは、表および索引のデータがディスクから読み取られるときにそれらをキャッシュに入れるためにデータベース・マネージャーによって割り振られるメイン・メモリーの領域です。すべての DB2 データベースには、バッファー・プールが必要です。

新規データベースごとに、IBMDEFAULTBP というデフォルトの定義済みのバッファー・プールが作成されます。追加のバッファー・プールを CREATE BUFFERPOOL、DROP BUFFERPOOL、および ALTER BUFFERPOOL ステートメントを使用して、作成、ドロップ、および変更することができます。SYSCAT.BUFFERPOOLS カタログ・ビューで、データベース内に定義されたバッファー・プールの情報がアクセスされます。

バッファー・プールが使用される方法

表内のデータの行が最初にアクセスされると、データベース・マネージャーによってそのデータが含まれたページがバッファー・プールに配置されます。ページは、データベースがシャットダウンされるか、またはページによって占有されたスペースが別のページによって要求されるまで、バッファー・プール内に留まります。

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。データの整合性を維持するために、データベース・マネージャーでは、バッファー・プール内の特定のページを更新中のエージェントを一度に 1 つしか許可しません。特定のページの更新中は、このページは 1 つのエージェントによって排他的にアクセスされています。ページが読み取られている場合は、複数のエージェントによって同時に読み取られる場合があります。
- 「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。
- 変更されたページがディスクに書き出されると、そのページは「クリーン」になり、バッファー・プールに残されたままになることがあります。

データベースの調整の大部分で、データのバッファー・プールへの移動を制御する構成パラメーターの設定、およびデータのバッファーからディスクへの書き込みを伴います。ページが最新のエージェントによって必要とされない場合は、このページ・スペースは新規アプリケーションからの新規ページ要求に使用される可能性があります。この場合、データベース・マネージャーのパフォーマンスが、余分なディスク入出力によって低下します。

スナップショット・モニターを使用して、バッファー・プールのヒット率を算出することができます。このヒット率はバッファー・プールの調整に役立ちます。

バッファー・プールの設計

バッファー・プールのサイズはデータベース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。

新規のバッファー・プールを作成する前に、以下の項目を解決してください。

- 使用するバッファー・プールの名前
- バッファー・プールを即時に作成するのか、あるいは次にデータベースが非活動化および再活動化された直後に作成するのか
- バッファー・プールが存在する必要があるのは、すべてのデータベース・パーティションか、それともデータベース・パーティションのサブセットか
- どれくらいのページ・サイズがバッファー・プールに必要か。下記の 159 ページの『バッファー・プールのページ・サイズ』を参照してください。
- バッファー・プールが固定サイズかどうか、あるいはデータベース・マネージャーによってバッファー・プールのサイズがワークロードに応じて自動的に調整されるかどうか。バッファー・プールの作成時に SIZE パラメーターを未指定にしておくことにより、データベース・マネージャーによるバッファー・プールの自動調整が行われるようにすることをお勧めします。詳しくは、「CREATE BUFFERPOOL ステートメント」の SIZE パラメーターおよび 159 ページの『バッファー・プールのメモリーの考慮事項』を参照してください。
- バッファー・プールの一部をブロック・ベース入出力用に予約するかどうか。詳しくは、「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」を参照してください。

表スペースとバッファー・プールとの間のリレーションシップ

バッファー・プールを設計する場合は、表スペースとバッファー・プールとの間の関係を理解しておく必要があります。各表スペースは、それぞれ特定の 1 つのバッファー・プールに関連付けられます。 IBMDEFAULTBP がデフォルトのバッファー・プールです。また、データベース・マネージャーにより、システム・バッファー・プールの IBMSYSTEMBP4K、IBMSYSTEMBP8K、IBMSYSTEMBP16K、および IBMSYSTEMBP32K (以前「隠れたバッファー・プール」といわれていた) も割り振られます。別のバッファー・プールを表スペースと関連付けるためには、そのバッファー・プールが存在しており、この 2 つが同じページ・サイズを持っている必要があります。この関連付けは、表スペースの作成時に (CREATE TABLESPACE ステートメントを使用して) 定義されますが、後から (ALTER TABLESPACE ステートメントを使用して) 変更することができます。

複数のバッファー・プールを使うと、全体のパフォーマンスが向上するようにデータベースの使用するメモリーを構成することができます。例えば、ユーザーによってランダムにアクセスされる 1 つまたは複数の大きな (使用可能なメモリーより大きい) 表が入っている表スペースの場合、データ・ページをキャッシュしても有利ではないため、バッファー・プールのサイズを制限することができます。また、オンライン・トランザクション・アプリケーション用の表スペースに対してより大きなバッファー・プールを関連付けると、アプリケーションの使用するデータ・ページが長くキャッシュされて、応答時間が速くなる場合があります。新しいバッファー・プールを構成する場合には、注意が必要です。

バッファー・プールのページ・サイズ

デフォルトのバッファー・プールのページ・サイズは、CREATE DATABASE コマンドの使用時に設定されます。このデフォルトは、将来の CREATE BUFFERPOOL ステートメントおよび CREATE TABLESPACE ステートメントすべてのデフォルト・ページ・サイズになります。データベースの作成時にページ・サイズを指定しない場合、デフォルト・ページ・サイズは 4 KB です。

注: データベースで 8 KB、16 KB、または 32 KB のページ・サイズが必要であると判断した場合は、一致するページ・サイズのバッファー・プールを少なくとも 1 つ定義して、データベース内の表スペースに関連付ける必要があります。

しかし、システム・バッファー・プールとは異なる特性を持つバッファー・プールが必要になる場合もあります。データベース・マネージャーで使用するために、新規のバッファー・プールを作成できます。表スペースとバッファー・プールの変更を有効にするために、データベースを再始動しなければならない場合があります。バッファー・プール用に選択するページ・サイズは、表スペースに指定されたページ・サイズによって決定されます。バッファー・プールに使用するページ・サイズは、バッファー・プールを作成した後ではもはや変更できないため、このページ・サイズの選択は重要です。

バッファー・プールのメモリーの考慮事項

メモリーの要件

バッファー・プールを設計する際は、ご使用のマシンに搭載したメモリー、および同じマシン上のデータベース・マネージャーで同時に実行される他のアプリケーションによって必要とされるメモリーの量に基づくメモリー要件も考慮してください。アクセスされるすべてのデータを維持するための十分なメモリー量がない場合は、オペレーティング・システムのデータ・スワッピングが発生します。これは、他のデータに場所を開けるために一部のデータが一時ディスク・ストレージに書き込まれるか、またはスワップされるときに発生します。一時ディスク・ストレージ上のデータが必要になると、このデータはメイン・メモリーにスワップ・バックされます。

バッファー・プール・メモリー保護

バージョン 9.5 では、バッファー・プールのメモリー内のデータ・ページがストレージ・キーによって保護されます。このストレージ・キーが使用可能になるのは、これが DB2_MEMORY_PROTECT レジストリー変数によって明示的に使用可能にされており、AIX (5.3 TL06 5.4) が POWER6™ 上で稼働している場合に限られます。

バッファー・プール・メモリー保護は、エージェントごとのレベルで機能します。これは、特定のエージェントがアクセスを必要とするときにのみバッファー・プール・ページにアクセスするというものです。メモリー保護は、DB2 エンジン・スレッドがバッファー・プール・メモリーにいつアクセスしてよく、いつアクセスすべきでないかを識別することによって機能します。詳細については、160 ページの『バッファー・プール・メモリー保護 (POWER6 上で稼動する AIX)』 を参照してください。

Address Windowing Extensions (AWE) および Extended Storage (ESTORE)

注: ESTORE 関連のキーワード、モニター・エレメント、およびデータ構造を含む AWE と ESTORE の各フィーチャーが廃止されました。さらにメモリーを割り当てるには、64 ビット・ハードウェア用のオペレーティング・システム、および関連の DB2 製品へアップグレードする必要があります。また、この廃止された機能の参照を除去するために、アプリケーションとスクリプトを変更する必要もあります。

バッファー・プール・メモリー保護 (POWER6 上で稼動する AIX)

データベース・マネージャーでは、追加、変更、および削除をデータベース・データの大部分に適用するためにバッファー・プールが使用されます。 POWER6 上で稼動する AIX 5.3 TL06+ の場合、*ストレージ・キー* を使用してバッファー・プール・メモリーを保護することができます。

ストレージ・キー は IBM® Power6 プロセッサーおよび AIX® オペレーティング・システムの新規フィーチャーで、ハードウェア・キーをカーネル・スレッド・レベルで使用して一定の範囲のメモリー保護を可能にします。ストレージ・キー保護により、バッファー・プール・メモリーの破壊の問題が低減し、データベースを停止させるおそれがあるエラーが限定されます。プログラミングという手段によってバッファー・プールに不正にアクセスしようとすると、データベース・マネージャーによって検出および処理が可能なエラー条件が発生します。

注: バッファー・プール・メモリー保護は、エージェントごとのレベルで機能します。これは、特定のエージェントがアクセスを必要とするときにのみバッファー・プール・ページにアクセスするというものです。

データベース・マネージャーでバッファー・プール・メモリーへのアクセスを制限することによって、バッファー・プールが保護されます。エージェントでその処理の実行のためにバッファー・プールへのアクセスが必要な場合は、このエージェントによるバッファー・プール・メモリーへのアクセスが一時的に認可されます。エージェントでバッファー・プールへのアクセスが不要になると、このアクセスは取り消されます。これにより、エージェントによるバッファー・プールの内容の変更が絶対に必要な場合にしか許可されないため、バッファー・プールの破壊の可能性が低減します。バッファー・プール・メモリーへの不正アクセスにより、セグメンテーション・エラーが発生します。これらのエラーを診断するための db2diag、db2fodc、db2pdcfg、および db2support コマンドなどのツールが提供されます。

データベース・エンジンの回復力を向上させるために、このバッファー・プール・メモリー保護フィーチャーを完全に使用可能にするには、以下の DB2_MEMORY_PROTECT と DB2_THREAD_SUSPENSION レジストリー変数の両方を使用可能にする必要があります。

DB2_MEMORY_PROTECT レジストリー変数

このレジストリー変数により、バッファー・プール・メモリーの保護フィーチャーが使用可能または使用不可にされます。 DB2_MEMORY_PROTECT が使用可能に (YES に設定) されている場合、DB2 エンジン・スレッドがバッファー・プール・メモリーに不正にアクセスしようとすると、そのエンジン・スレッドはトラップします。デフォルトは NO です。

DB2_THREAD_SUSPENSION レジストリー変数

このレジストリー変数により、DB2 スレッド中断フィーチャーが使用可能または使用不可にされます。これにより、障害のあるエンジン・スレッド (つまり、ストレージ・キーで保護されたメモリーに不正にアクセスしようとしたスレッド) を中断することにより、 DB2 インスタンスによりトラップが維持されるかどうかを制御できます。 .

注: DB2_THREAD_SUSPENSION は、DB2_MEMORY_PROTECT が使用可能になっている場合にのみ使用可能にできます。 DB2_THREAD_SUSPENSION が使用可能になっている場合は、以下のようになります。

- スレッド内のどんな障害でも、ストレージ・キーを使用して保護された、スレッドにアクセス権のないメモリーにアクセスする試みで引き起こされたかどうかに関係なく、ストレージ・キーを使用して保護されたこのメモリーに対して、スレッドにアクセス権のない場所で障害が発生した場合、データベース・マネージャーにより、保護されたメモリーが破壊されていず、その結果 DB2 エンジンの実行を継続させることが保証されます。
- ユーザー定義関数を SQL なしで unfenced モードで実行した場合、バッファー・プール・メモリー保護違反が検出されると、データベース・マネージャーによってエラーが UDF の呼び出し元に返され、データベースは影響を受けずに実行し続けます。

バッファー・プールの作成

データベース・マネージャーによって使用される新しいバッファー・プールを定義するには、CREATE BUFFERPOOL ステートメントを使用します。

基本の CREATE BUFFERPOOL ステートメントの例は、以下のとおりです。

```
CREATE BUFFERPOOL <buffer pool name>
  PAGESIZE 4096
```

バッファー・プールは、使用可能な十分のメモリーがある場合は即時にアクティブになることができます。デフォルトでは、新規バッファー・プールは IMMEDIATE キーワードを使用して作成され、ほとんどのプラットフォーム上では、データベース・マネージャーがさらに多くのメモリーを取得することができます。予期される戻りは、メモリー割り振りの成功です。データベース・マネージャーがさらにメモリーを割り振ることができない場合にのみ、バッファー・プールが開始できず、これが後続のデータベース始動時に実行されることを述べる警告条件が戻されます。即時要求の場合、データベースを再始動する必要はありません。このステートメントがコミットされるとシステム・カタログ表にバッファー・プールが反映されますが、バッファー・プールがアクティブになるのは、次にデータベースが開始されてからです。他のオプションも含め、このステートメントについて詳しくは、『CREATE BUFFERPOOL ステートメント』を参照してください。

CREATE BUFFERPOOL DEFERRED を発行する場合、バッファー・プールは即時にはアクティブにならず、次のデータベース始動時に作成されます。新規表スペースが据え置きバッファー・プールを明示的に使用するように作成されている場合でも、データベースが再始動されるまで、その新規表スペースは既存のバッファー・プールを使用します。

作成されるすべてのバッファー・プールの合計に相当する十分な実メモリーがマシン上に必要です。オペレーティング・システムも、操作のためにいくらかのメモリーを必要とします。

コマンド行を使用してバッファー・プールを作成するには、以下のようにします。

1. 以下の SQL ステートメントを発行して、データベース内に既に存在するバッファー・プール名のリストを入手します。

   ```
   SELECT BPNAME FROM SYSCAT.BUFFERPOOLS
   ```

2. 結果として表示されたリストに現在存在していないバッファー・プール名を選択します。
3. 作成しようとしているバッファー・プールの特性を決定します。
4. CREATE BUFFERPOOL ステートメントを実行する適正な許可 ID があることを確認します。
5. CREATE BUFFERPOOL ステートメントを発行します。

パーティション・データベース上では、それぞれのデータベース・パーティション上で異なるサイズを指定するなど、作成するバッファー・プールをさまざまに定義することができます。デフォルトの ALL DBPARTITIONNUMS 節は、このバッファー・プールが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して作成されることを示します。

以下の例で、オプションの DATABASE PARTITION GROUP 節は、バッファー・プール定義が適用されるデータベース・パーティション・グループを指定します。

```
CREATE BUFFERPOOL <buffer pool name>
  PAGESIZE 4096
  DATABASE PARTITION GROUP <db partition group name>
```

このパラメーターを指定すると、そのデータベース・パーティション・グループ内のデータベース・パーティションに対してのみバッファー・プールが作成されます。各データベース・パーティション・グループがデータベース内に存在していなければなりません。 DATABASE PARTITION GROUP 節を指定しない場合、このバッファー・プールは、すべてのデータベース・パーティション (および後でデータベースに追加されるすべてのデータベース・パーティション) に対して作成されます。

詳しくは、『CREATE BUFFERPOOL ステートメント』を参照してください。

バッファー・プールの変更

さまざまな理由で (例えばセルフチューニング・メモリーを使用可能化するために)、バッファー・プールを変更する必要が生じるかもしれません。これを行うには、ALTER BUFFERPOOL ステートメントを使用します。

このステートメントの許可 ID には、SYSCTRL 権限または SYSADM 権限がなければなりません。

バッファー・プールを操作する際、以下のいずれかの作業を行う必要があるかもしれません。

- バッファー・プールのセルフチューニングの使用可能化。これにより、データベース・マネージャーはワークロードに応じてバッファー・プールのサイズを調整できるようになります。
- ブロック・ベース入出力のための、バッファー・プールのブロック領域の変更。
- 新規データベース・パーティション・グループへのこのバッファー・プール定義の追加。
- すべてのデータベース・パーティション、またはいくつかのデータベース・パーティションにおけるバッファー・プール・サイズの変更。

コマンド行を使用してバッファー・プールを変更するには、以下のようにします。

1. データベースに既に存在するバッファー・プール名のリストを取得するために、次のステートメントを発行します。

   ```
   SELECT BPNAME FROM SYSCAT.BUFFERPOOLS
   ```

2. 結果として表示されるリストからバッファー・プール名を選択します。
3. どのような変更を行う必要があるかを判断します。
4. ALTER BUFFERPOOL ステートメントを実行する適正な許可 ID があることを確認します。

   注: 2 つの重要なパラメーターは IMMEDIATE と DEFERRED です。 IMMEDIATE では、有効にするために次のデータベース活動化を待つ必要なしに、バッファー・プール・サイズが変更されます。新しいスペースを割り振るための十分なデータベース共用メモリーがない場合、ステートメントの実行は DEFERRED となります。

   DEFERRED を指定すると、データベースが再活動化されるまで、バッファー・プールの変更内容は適用されません。予約済みメモリー・スペースは不要です。活動化の際に、データベース・マネージャーによって必要なメモリーがシステムから割り振られます。

5. バッファー・プール・オブジェクトの 1 つの属性を変更するには、ALTER BUFFERPOOL ステートメントを使用します。例:

   ```
   ALTER BUFFERPOOL buffer pool name SIZE number of pages
   ```

- ここで、*buffer pool name* は、システム・カタログに記述されたバッファー・プールを識別する 1 部構成の名前です。
- *number of pages* は、この特定のバッファー・プールに割り振られる新しいページ数です。値 -1 も使用できます。この値は、データベース構成パラメーター **buffpage** で見つかった値をバッファー・プールのサイズにすることを示します。

さらに、ステートメントに DBPARTITIONNUM <db partition number> 節を含めることもできます。これは、バッファー・プール・サイズの変更対象となるデータベース・パーティションを指定します。この節を指定しない場合、SYSCAT.BUFFERPOOLDBPARTITIONS 内に例外項目が設定されているパーティションを除く、すべてのデータベース・パーティションでバッファー・プール・サイズが変更されます。データベース・パーティションに関するこの節の使用方法について、詳しくは ALTER BUFFERPOOL ステートメントを参照してください。

このステートメントの結果としてバッファー・プールが変更される場合、ステートメントのコミット時にシステム・カタログ表に変更内容が反映されます。ただし、

デフォルトの IMMEDIATE キーワードを使用して指定された ALTER BUFFERPOOL 要求が成功した場合を除いて、データベースが次回に開始されるまでは実際のバッファー・プールは変更されません。

作成されるすべてのバッファー・プールの合計に相当する十分な実メモリーがマシン上に必要です。さらに、データベース・マネージャーの残りの部分とアプリケーションのための十分な実メモリーも必要です。

バッファー・プールのドロップ

バッファー・プールをドロップするときには、それらのバッファー・プールに割り当てられた表スペースがないことを確認してください。IBMDEFAULTBP バッファー・プールをドロップすることはできません。

ディスク・ストレージは、データベースへの次の接続まで解放できません。実際には、データベースが停止するまで、ドロップされたバッファー・プールからストレージ・メモリーは解放されません。バッファー・プール・メモリーは、データベース・マネージャーによる使用のために、即時に解放されます。

バッファー・プールをドロップするには、以下のように DROP BUFFERPOOL ステートメントを使用できます。

```
DROP BUFFERPOOL <buffer pool name>
```

第 9 章 表スペース

表スペース は、表、索引、ラージ・オブジェクト、およびロング・データを含む、ストレージ構造です。表スペースは、データベース・パーティション・グループの中に常駐します。表スペースによって、データベースと表データのロケーションをコンテナーに直接割り当てることができます。 (コンテナーとしては、ディレクトリー名、装置名、ファイル名があります。) これによってパフォーマンスが改善され、構成の柔軟性が高くなります。

表スペースはデータベース・パーティション・グループの中にあるので、表の保持のために選択された表スペースは、その表のデータがデータベース・パーティション・グループ内の複数のデータベース・パーティションにわたって分配される方法を定義します。 1 つの表スペースが複数のコンテナーにわたる場合もあります。(1 つまたは複数の表スペースからの) 複数のコンテナーを、同じ物理ディスク (またはドライブ) 上に作成することも可能です。自動ストレージの表スペースを使用している場合は、この表スペースはデータベース・マネージャーによって処理されます。自動ストレージの表スペースを *使用していない* 場合、パフォーマンスを向上させるためには、各コンテナーごとに異なるディスクを使用する必要があります。

図 6 は、データベース内の表および表スペースと、そのデータベースに関連するコンテナーのリレーションシップを示しています。

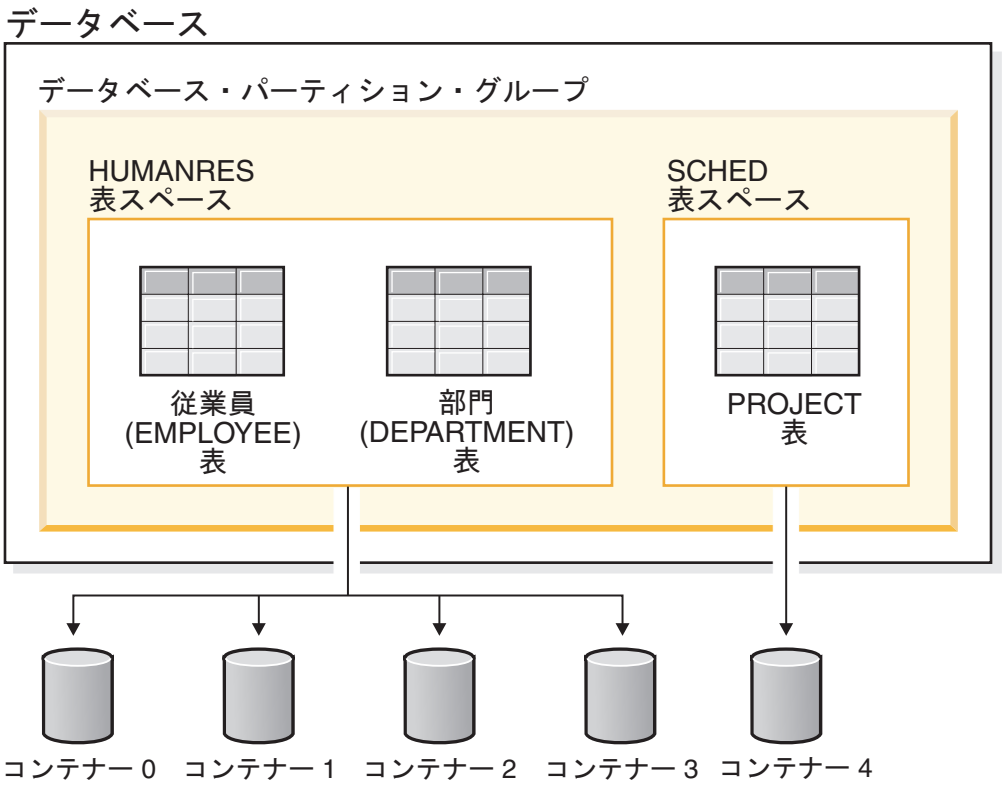


図 6. データベース内の表スペースと表

EMPLOYEE および DEPARTMENT 表は HUMANRES 表スペースにあり、これはコンテナー 0、1、2、および 3 にわたっています。 PROJECT 表は SCHED 表スペースにあり、コンテナー 4 に入っています。この例では、各コンテナーが別のディスクにあることを示しています。

データベース・マネージャーは、コンテナー間でデータ・ロードの平衡を取ろうとします。結果として、データを格納するのにすべてのコンテナーが使われます。別のコンテナーを使用する前に、データベース・マネージャーがコンテナーに書き込むページ数は、エクステント・サイズ と呼ばれます。データベース・マネージャーは、毎回最初のコンテナーから表データを格納し始めるとは限りません。

図 7 は、エクステント・サイズが 4 KB ページ 2 つ分の HUMANRES 表スペースを表しています。それぞれのページには、割り振りエクステントが小さく設定されているコンテナーが 4 つずつあります。 DEPARTMENT 表と EMPLOYEE 表は、どちらも 7 ページあり、 4 つのコンテナーすべてにわたっています。

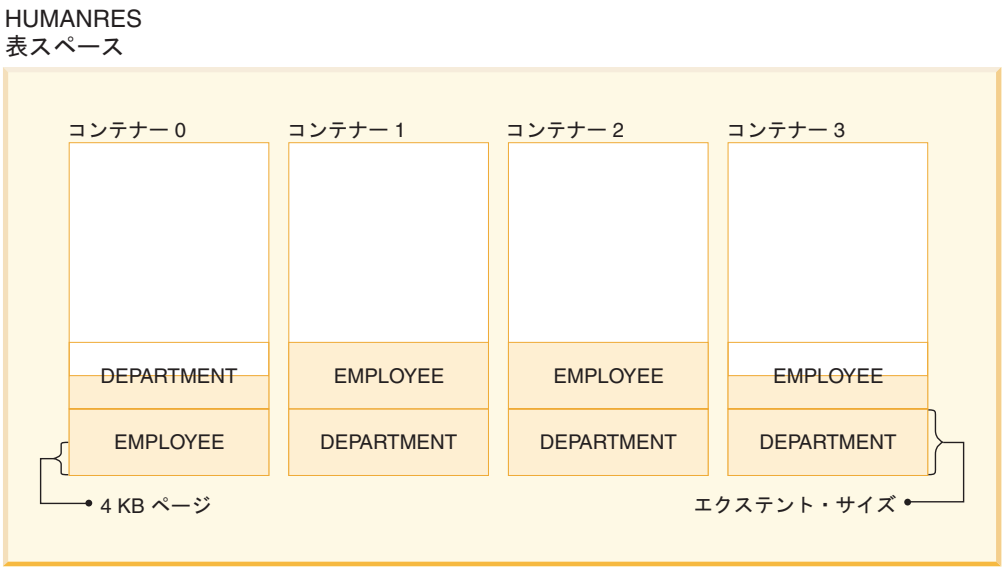


図 7. 表スペースの中のコンテナーとエクステント

表スペースの設計

表スペースは、データベース内のデータが、物理的にシステム上のどこに格納されるかを指定して、データベースと、実際のデータが存在するコンテナー・オブジェクトとの間の間接化の層を提供するために使用されます。

表スペースを作成する理由はいろいろありますが、その中には、リカバリーを可能にすること、およびさまざまなバッファー・プールにオブジェクトを分離できるようにするという点があります。自動ストレージを使用すると、表スペースの物理的なディスク・ロケーションや、コンテナーの物理的なロケーションを気にする必要がなくなります。データベース・マネージャーは表スペースにコンテナーを自動的に割り当てる、または作成します。

1 つのデータベースには、少なくとも以下の 3 つの表スペースが含まれている必要があります。

- 1 つのカタログ表スペース。これには、データベースのすべてのシステム・カタログ表が入ります。この表スペースは SYSCATSPACE と呼ばれ、省略できません。 IBMCATGROUP は、この表スペースに対するデフォルト・データベース・パーティション・グループです。
- 1 つまたは複数のユーザー表スペース。これには、ユーザー定義のすべての表が入ります。デフォルトには、1 つの表スペース USERSPACE1 が作成されます。IBMDEFAULTGROUP は、この表スペースに対するデフォルト・データベース・パーティション・グループです。

  表スペース名は、表の作成時に指定してください。指定しない場合、望みどおりの結果が得られないかもしれません。

  表のページ・サイズは、行サイズまたは列数のいずれかによって決定されます。行の許容最大長は、表が作成された表スペースのページ・サイズに依存しています。ページ・サイズに指定できる値は 4 KB、8 KB、16 KB、および 32 KB です。バージョン 9.1 より前のバージョンでは、4 KB がデフォルトのページ・サイズでした。バージョン 9.1 以降では、それ以外のいずれかのサポート値がデフォルトのページ・サイズになり得ます。デフォルトのページ・サイズは新規データベースの作成時に宣言されます。デフォルトのページ・サイズが宣言されると、表には 1 つのページ・サイズを持つ表スペース、LONG または LOB データにはページ・サイズの異なる別の表スペースを自由に作成できるようになります。 (SMS は複数の表スペースにまたがる表をサポートしませんが、 DMS はそのような表をサポートすることに留意してください。) 列の数または行のサイズが表スペースのページ・サイズの限界を超えると、エラーが戻されます (SQLSTATE 42997)。
- 1 つまたは複数の *TEMPORARY 表スペース*。一時表が入れられます。TEMPORARY 表スペースには、*SYSTEM TEMPORARY 表スペース*と*USER TEMPORARY 表スペース* があります。

  SYSTEM TEMPORARY 表スペースは、ソートや結合などの操作の実行中に、データベース・マネージャーに必要な一時データを保持します。これらのタイプの操作には、結果セットを処理するための余分のスペースが必要です。各データベースには、少なくとも 1 つの SYSTEM TEMPORARY 表スペースが必要です。デフォルトには、データベースの作成時に TEMPSPACE1 という 1 つの SYSTEM TEMPORARY 表スペースが作成されます。 IBMTEMPGROUP は、この表スペースに対するデフォルト・データベース・パーティション・グループです。

  USER TEMPORARY 表スペースは、DECLARE GLOBAL TEMPORARY TABLE ステートメントを使用して作成した表からの一時データを保持します。宣言済み一時表を定義できるようにするには、該当する USE 特権を使用して、1 つ以上の USER TEMPORARY 表スペースを作成する必要があります。 USE 特権を付与するには、GRANT ステートメントを使用します。 USER TEMPORARY 表スペースは、デフォルトではデータベース作成の時点で*作成されません* 。

  データベースが複数の TEMPORARY 表スペースを使用していて、新しい一時オブジェクトが必要になったときは、オプティマイザーが、このオブジェクトに適

したページ・サイズを選択します。そして、対応するページ・サイズを持つ TEMPORARY 表スペースにそのオブジェクトが割り振られます。同じページ・サイズの TEMPORARY 表スペースが複数存在する場合は、ラウンドロビン式に表スペースが選択されます。ほとんどの環境では、1 ページ・サイズの複数の TEMPORARY 表スペースを持つことは推奨されていません。

デフォルトよりも大きいページ・サイズで定義された表スペース内の表に対して照会が実行されると、照会が失敗する場合があります。これは、より大きいページ・サイズで定義された TEMPORARY 表スペースが存在しない場合に起こります。場合によっては、さらに大きいページ・サイズの TEMPORARY 表スペースを作成する必要があります (デフォルトのページ・サイズが 4 KB だった場合は、8 KB、16 KB、または 32 KB のページ・サイズの TEMPORARY 表スペースを作成する必要があるでしょう)。データ操作言語 (DML) ステートメントは、USER TEMPORARY 表スペース内の最大ページ・サイズと同じページ・サイズの TEMPORARY 表スペースがなければ失敗する可能性があります。

単一の SMS TEMPORARY 表スペースの定義では、大半のユーザー表スペースで使用されているページ・サイズと等しいページ・サイズを指定してください。そうすれば、一般的な環境とワークロードに適したサイズが得られます。

表スペースとデータベース・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、各表スペースが特定のデータベース・パーティション・グループと関連付けられます。これによって、表スペースの特性をそのデータベース・パーティション・グループ内の各データベース・パーティションに適用することができます。

データベース・パーティション・グループはすでに存在している必要があり (CREATE DATABASE PARTITION GROUP ステートメントを使用して定義される)、表スペースとデータベース・パーティション・グループの関連は、 CREATE TABLESPACE ステートメントを使用して表スペースが作成されるときに定義されます。

ALTER TABLESPACE ステートメントを使用して表スペースとデータベース・パーティション・グループの間の関連を変更することはできません。データベース・パーティション・グループ内の個々のデータベース・パーティションに対する表スペース仕様を変更できるだけです。単一パーティション・データベース環境では、各表スペースはデフォルト・データベース・パーティション・グループと関連付けられます。表スペースを定義するときのデフォルトのデータベース・パーティション・グループは IBMDEFAULTGROUP です。ただし、SYSTEM TEMPORARY 表スペースが定義されている場合は IBMTEMPGROUP が使用されます。

さまざまなタイプの表スペース

データベースには、少なくとも 3 つのタイプの表スペースが必要です。カタログ表スペース 1 つ、ユーザー表スペースが 1 つ以上、TEMPORARY 表スペースが 1 つ以上です。

表スペースには、次に示す 2 つの種類があります。 1 つのデータベースで両方を使用できます。

- システム管理スペース。オペレーティング・システムのファイル・マネージャーがストレージ・スペースを制御します。
- データベース管理スペース。データベース・マネージャーがストレージ・スペースを制御します。

パーティション・データベース環境では、カタログ・パーティションは 3 つのデフォルト表スペースすべてを含み、その他のデータベース・パーティションはそれぞれ TEMPSPACE1 と USERSPACE1 だけを含みます。

自動ストレージ表スペースも作成できます。これは、基礎表スペース・タイプとして SMS または DMS を使用します。実際のタイプ (SMS または DMS) は、そこに存在するデータのタイプに応じて、データベース・マネージャーが選択します (TEMPORARY 表スペースには SMS、それ以外の場合は DMS)。

システム管理スペース

SMS (システム管理スペース) 表スペースでは、オペレーティング・システムのファイル・システム・マネージャーが、表の保管されるスペースの割り振りと管理を行います。

ストレージ・モデルは一般的に、表オブジェクトを表す多くのファイルで構成されます。これらのファイルは、ファイル・システム・スペースに保管されます。 ユーザーがファイルのロケーションを決定し、データベース・マネージャーがそれらの名前を制御し、そしてファイル・システムがそれらを管理する責任を持ちます。データベース・マネージャーは、各ファイルに書き込まれるデータの量を制御することにより、それぞれの表スペース・コンテナーにデータを均等に分配します。

各表には、それと関連した少なくとも 1 つの SMS 物理ファイルがあります。

表スペースのデータは、システム内のすべてのコンテナーにエクステント単位でストライピングされます。エクステント は、データベースに定義される連続ページのグループです。ファイル拡張子は、ファイルに保管されているデータのタイプを示します。データを表スペースのすべてのコンテナーに均一に配布するために、表の開始エクステントは、すべてのコンテナーでラウンドロビン方式で配置されています。そのようなエクステントの分散は、データベースに小さな表がたくさん入っている場合に特に重要です。

SMS 表スペースでは、表のスペースは要求時に割り振られます。割り振られるスペースの量は、*multipage_alloc* データベース構成パラメーターの設定によって左右されます。この構成パラメーターが YES に設定されている場合、スペースが必要なときにはエクステント全体 (通常は複数のページで構成される) が割り振られます。それ以外の場合は、一度に 1 ページのスペースが割り振られます。

マルチページ・ファイル割り振りは、デフォルトで使用可能です。 multipage_alloc データベース構成パラメーターの値は、マルチページ・ファイル割り振りが使用可能かどうかを示します。

注: マルチページ・ファイル割り振りは、TEMPORARY 表スペースには適用されません。

複数ページ・ファイル割り振りは、表のデータおよび索引部分にのみ影響します。したがって、.LF、.LB、および .LBA ファイルが一度に 1 つのエクステントに拡張されることはありません。

SMS 表スペース内の 1 つのコンテナーのすべてのスペースが表に割り当てられると、他のコンテナーでスペースが残っていても、表スペースはいっぱいであると見なされます。まだコンテナーが 1 つもないデータベース・パーティションの SMS 表スペースにのみ、コンテナーを追加できます。

注: SMS 表スペースは、ファイル・システムのプリフェッチとキャッシングを利用できます。

SMS 表スペースは、CREATE DATABASE コマンドまたは CREATE TABLESPACE ステートメントの MANAGED BY SYSTEM オプションを使用して定義されます。 SMS 表スペースを設計するときは、かぎとなる以下の 2 つの要素を考慮に入れる必要があります。

- 表スペース用のコンテナー

  表スペースで使用するコンテナーの数を指定しなければなりません。 SMS 表スペースが作成された後ではコンテナーを追加または削除することができないため、使用するすべてのコンテナーを確認しておくことがきわめて重要です。パーティション・データベース環境では、SMS 表スペースのデータベース・パーティション・グループに新しいデータベース・パーティションが追加されたときに、ALTER TABLESPACE ステートメントを使用して、新しいデータベース・パーティションにコンテナーを追加することができます。

  SMS 表スペースの各コンテナーは、絶対または相対ディレクトリー名を識別します。これらのディレクトリーは、それぞれ別個のファイル・システム (または物理ディスク) に置くことができます。表スペースの最大サイズは次のように見積もることができます。

  ```
  number of containers * (maximum file system size
     supported by the operating system)
  ```

  この公式は、各コンテナーごとに別個のファイル・システムがマップされており、各ファイル・システムには使用できるスペースの上限があることを前提とします。現実にはそのようなことはあまりなく、多くの場合、表スペースの最大サイズはもっと小さくなります。データベース・オブジェクトのサイズにおける制限もあります。これは、表スペースの最大サイズに影響を与える場合があります。

  注: コンテナーを定義するときには注意が必要です。コンテナーにファイルやディレクトリーがすでに存在する場合は、エラー (SQL0298N) が戻されます。

- 表スペースのエクステント・サイズ

  エクステント・サイズの指定は、表スペースの作成時にしか行えません。後から変更することはできませんので、エクステント・サイズの適切な値を選択してください。

  表スペースを作成するときにエクステント・サイズを指定しなければ、データベース・マネージャーは、*dft_extent_sz* データベース構成パラメーターで定義され

ているデフォルト・エクステント・サイズを使って表スペースを作成します。この構成パラメーターは、データベースの作成時に指定した情報をもとに初期設定されます。 CREATE DATABASE コマンドで *dft_extent_sz* パラメーターを指定しなければ、デフォルト・エクステント・サイズは 32 に設定されます。

コンテナーの数および表スペースのエクステント・サイズに適切な値を選択するには、以下のことを理解しておく必要があります。

- 使用しているオペレーティング・システムでの、論理ファイル・システムのサイズ制限。

  例えば、一部のオペレーティング・システムでは、2 GB の制限があります。そのため、64 GB の表オブジェクトを希望する場合は、このタイプのシステムでは少なくとも 32 のコンテナーが必要です。

  表スペースを作成するときに、コンテナーが別々のファイル・システムに存在するよう指定すれば、データベースに格納できるデータ量を増やすことができます。
- データベース・マネージャーが、表スペースと関連したデータ・ファイルおよびコンテナーを管理する方法。

  最初の表データ・ファイル (SQL00002.DAT) は、その表スペースに指定された最初のコンテナーの中に作成され、このファイルは、エクステント・サイズまで拡張することが許されます。そのサイズまで達したら、データベース・マネージャーは次のコンテナーの SQL00002.DAT にデータを書き込みます。このプロセスは、すべてのコンテナーに SQL00002.DAT ファイルが入るまで続きます。その後、データベース・マネージャーは最初のコンテナーに戻ります。このプロセス (ストライピング という) は、コンテナーが満杯になるか (SQL0289N)、またはオペレーティング・システムがそれ以上スペースを割り振れなくなる (ディスク満杯エラー) まで、コンテナー・ディレクトリーにわたって続行されます。ストライピングは、ブロック・マップ・ファイル (SQLnnnnn.BKM)、索引オブジェクト、および表データの保管に使用されるその他のオブジェクトに適用されます。

  注: SMS 表スペースは、そのコンテナーのうちのいずれか 1 つが満杯になると、ただちに満杯になります。したがって、各コンテナーに同じ量のスペースを割り当てることが重要です。

  複数のコンテナーにわたってデータをより均等に分散させるのに役立てるため、データベース・マネージャーは、表の ID (上記の例では SQL00002.DAT) とコンテナー数を考慮することにより、最初に使用するコンテナーを判別しています。コンテナーは 0 から順番に番号が付けられます。

データベース管理スペース

DMS (データベース管理スペース) 表スペースでは、データベース・マネージャーがストレージ・スペースを制御します。

ストレージ・モデルは、限定された数の装置またはファイルからなり、そのスペースはデータベース・マネージャーが管理します。データベース管理者は、どの装置およびファイルを使用するかを決定し、それらの装置およびファイル上のスペース

を管理します。この表スペースは基本的に、データベース・マネージャーの必要を最もよく満たすように設計された特別な目的のファイル・システムを実現したものです。

DMS 表スペースと SMS 表スペースの相違点は、DMS 表スペースでは、表スペースの作成時にスペースが割り当てられることです。一方 SMS 表スペースでは、必要に応じてスペースが割り振られます。ユーザー定義の表およびデータが入っている DMS 表スペースは、あらゆる表データまたは索引データを格納する *REGULAR* 表スペースまたは *LARGE* 表スペースとして定義することができます。

DMS 表スペースおよびコンテナーを設計するときは、以下の要素を考慮してください。

- データベース・マネージャーは、ストライピングを使用して、すべてのコンテナーにわたって均等にデータを分散させるようにしています。(つまり、データを表スペースのすべてのコンテナーに均一に配布して、表のエクステントをすべてのコンテナーでラウンドロビン方式で配置します。)
- REGULAR 表スペースの最大サイズは 32 KB のページに対して 512 GB です。LARGE 表スペースの最大サイズは、16 TB です。これ以外のページ・サイズに対する REGULAR 表スペースの最大サイズについては、SQL および XML の制限値を参照してください。
- SMS 表スペースとは異なり、 DMS 表スペースを構成するコンテナーのサイズはすべて同じにする必要はありません。しかし、サイズが異なると、コンテナー間のストライピングが不均一になり、最適なパフォーマンスが得られるとは限らないため、通常は推奨されていません。いずれかのコンテナーが満杯である場合、DMS 表スペースは、他のコンテナーからの使用可能なフリー・スペースを使用します。
- スペースは事前割り振りされるため、表スペースを作成する前に、スペースが利用可能になっていなければなりません。デバイス・コンテナーを使用する場合は、コンテナーを定義するのに十分なスペースの装置が存在していなければなりません。それぞれの装置には、コンテナーを 1 つだけ定義することができます。スペースを無駄にしないために、装置のサイズとコンテナーのサイズが等しくなるようにしてください。例えば、ある装置に 5 000 ページが割り振られているのに、デバイス・コンテナーに 3 000 ページしか割り振らないと、その装置の 2 000 ページは使用できなくなります。
- デフォルトでは、すべてのコンテナー内の 1 つのエクステントがオーバーヘッドのために予約されます。フル・エクステントしか使用されないため、最適なスペース管理を実現するには、コンテナーを割り振るときに以下の公式を使って適切なサイズを決定してください。

  ```
  extent_size * (n + 1)
  ```

  extent_size は表スペース内の各エクステントのサイズ、 *n* はコンテナーに格納するエクステントの数を表します。
- DMS 表スペースの最小サイズは、5 つのエクステントです。 5 つのエクステントよりも小さい表スペースを作成しようとすると、エラー (SQL1422N) が生じます。
  - 表スペース内のエクステント 3 つはオーバーヘッドのために確保されています。

  - いずれのユーザー表データを格納するにも、最低で 2 つのエクステントが必要です。 (これらのエクステントは 1 つの表の正規データを格納するためのものです。専用のエクステントを必要とする、索引、ロング・フィールド、またはラージ・オブジェクト・データのためではありません。)
- デバイス・コンテナーは、物理ボリュームではなく、「文字固有のインターフェース」を持つ論理ボリュームを使用しなければなりません。
- DMS 表スペースでは、装置の代わりにファイルを使用することができます。デフォルトの表スペース属性 (バージョン 9.5 では NO FILE SYSTEM CACHING) では、装置のセットアップをしないで済むという利点を持ちながら、装置に近い機能をファイルが実行できます。詳しくは、 196 ページの『ファイル・システム・キャッシングを使用しない表スペース』を参照してください。
- 実際のワークロードに LOB または LONG VARCHAR データが含まれる場合、ファイル・システムのキャッシュによってより高いパフォーマンスが得られる場合があります。

  注: LOB および LONG VARCHAR はデータベース・マネージャーのバッファー・プールには入れられません。
- オペレーティング・システムによっては、 2 GB より大きいサイズの物理装置を持つことができるものがあります。物理装置を複数の論理装置に分割して、オペレーティング・システムによって許されるサイズより大きなコンテナーがないようにする必要があります。

注: SMS 表スペースと同様、 DMS ファイル・コンテナーもファイル・システムのプリフェッチとキャッシングを利用できます。ただし、ロー・デバイス・コンテナーを使用する DMS 表スペースでは利用できません。

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

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

CREATE TABLESPACE ステートメントは、データベースで新しい表スペースを作成し、この表スペースにコンテナーを割り当て、カタログに表スペース定義と属性を記録します。表スペースの作成時、エクステント・サイズは連続するページの数として定義されます。エクステントは、表スペース内のスペース割り当ての単位です。エクステント内のページを使用できる表やオブジェクト (索引など) は、そのエクステントごとに 1 つだけです。表スペースで作成されるすべてのオブジェクトには、論理表スペース・アドレス・マップでエクステントが割り当てられます。エクステントの割り当ては、スペース・マップ・ページにより管理されます。

論理表スペース・アドレス・マップの先頭のエクステントは、表スペースのヘッダーで、これには内部制御情報が含まれます。 2 番目のエクステントは、表スペースの SMP の最初のエクステントです。 SMP エクステントは、表スペースで等インターバルに分散されます。それぞれの SMP エクステントは、現行の SMP エクステントから次の SMP エクステントへのエクステントのビットマップです。このビットマップは、どの中間エクステントが使用中かをトラッキングするのに使用されます。

SMP に続くエクステントは、表スペースのオブジェクト表です。オブジェクト表は、表スペースにどのユーザー・オブジェクトが存在するか、またどこに最初のエクステント・マップ・ページ (EMP) エクステントが配置されているかをトラッキングする内部表です。各オブジェクトには、それぞれ EMP があり、これは、論理表スペース・アドレス・マップに保管されているオブジェクトの各ページへのマップを提供します。 図 8 は、論理表スペース・アドレス・マップでエクステントが割り当てられる方法を示しています。

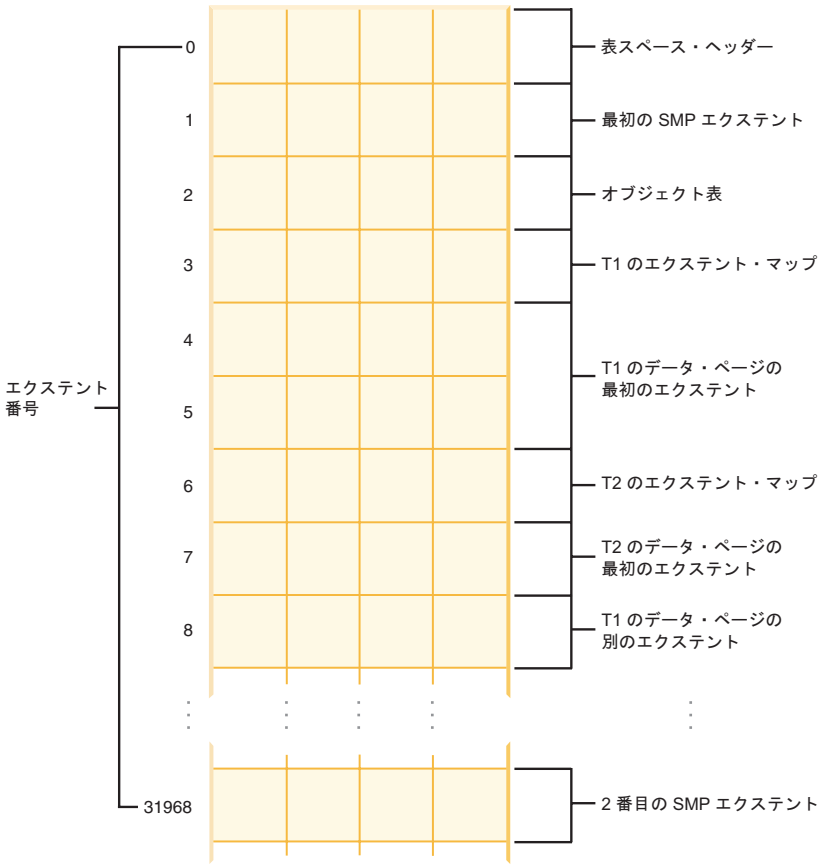


図 8. 論理表スペース・アドレス・マップ

DMS 表スペース・マップ:

表スペース・マップは DMS 表スペースを表すデータベース・マネージャーの内部表記であり、表スペースにおける論理ページ・ロケーションから物理ページ・ロケーションへの変換を記述しています。このトピックでは、表スペース・マップが役立つ理由と、表スペース・マップの情報がどこから得られるかを説明します。

パーティション・データベースでは、 DMS 表スペース内の各ページに 0 から N-1 までの番号が論理的に付けられています (N は、表スペース内の使用できるページ数)。

DMS 表スペース内のページは、エクステント・サイズに基づいて各エクステントにグループ分けされ、表スペース管理の観点から言うと、すべてのオブジェクト割り振りはエクステント・ベースで行われます。つまり、表があるエクステントの半分のページしか使っていなくても、そのエクステント全体が使用中で、そのオブジェクトに所有されていると見なされます。デフォルトでは、コンテナー・タグを保持するために 1 つのエクステントが使用され、このエクステント内のすべてのページはデータを保持できません。ただし、レジストリー変数 DB2_USE_PAGE_CONTAINER_TAG をオンにすれば、コンテナー・タグに使用されるのは 1 ページだけになります。

176 ページの図 9 に、DMS 表スペースの論理アドレス・マップを示します。

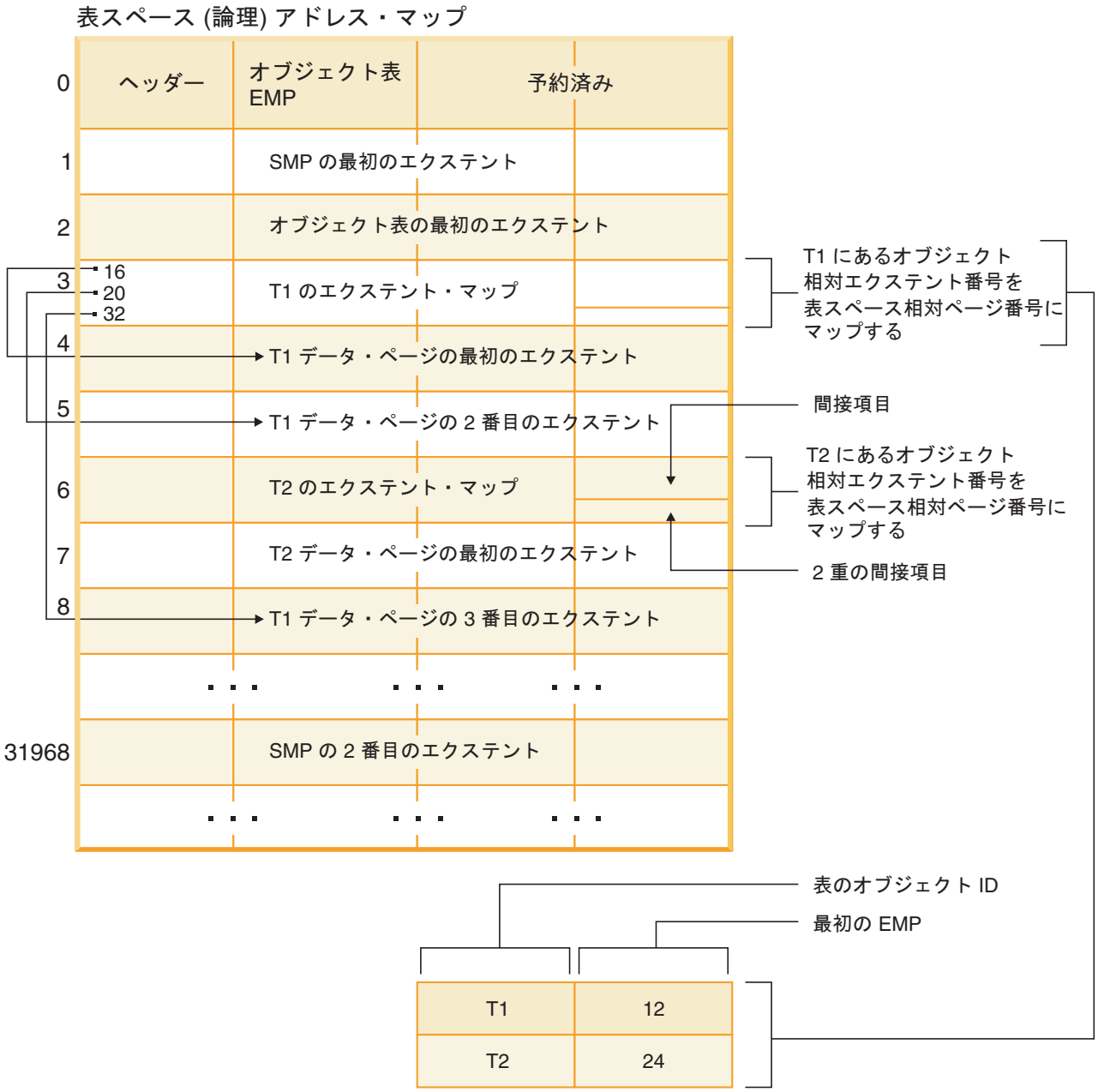


図 9. DMS 表スペース

表スペースのアドレス・マップには 2 種類のマップ・ページがあります。エクステント・マップ・ページ (EMP) とスペース・マップ・ページの 2 種類です。

オブジェクト表は内部リレーショナル表で、オブジェクト ID を表の最初の EMP エクステントのロケーションにマップします。この EMP エクステントは、直接的にも間接的にも、オブジェクトにあるすべてのエクステントをマップします。各 EMP には、項目の配列が含まれています。各項目は、オブジェクト相対エクステント番号を、オブジェクト・エクステントが配置されている表スペース相対ページ番号にマップします。直接 EMP 項目は、オブジェクト相対アドレスを表スペース相対アドレスに直接マップします。最初の EMP エクステントにある最後の EMP ページには、間接項目が含まれます。間接 EMP 項目は EMP ページにマップし、続

いて EMP ページがオブジェクト・ページにマップします。最初の EMP エクステントにある最後の EMP ページの末尾の 16 個の項目には、 2 重の間接項目が含まれます。

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

コンテナー内のスペースはエクステント単位で割り振られるため、完全なエクステントを形成しないページは使用されません。例えば、205 ページからなるコンテナーがあり、エクステント・サイズが 10 である場合、 1 つのエクステントがタグ用に使用され、 19 個のエクステントにデータが保持されます (残る 5 ページは無駄になります)。

DMS 表スペースにただ 1 つのコンテナーが含まれている場合、論理ページ番号からディスク上の物理ロケーションへの変換プロセスは単純で、ページ 0、1、2 はディスク上に同じ順番で格納されます。

複数のコンテナーが存在し、各コンテナーのサイズが同じである場合にも、変換プロセスはかなり単純です。表スペースの最初のエクステント (ページ 0 から [エクステント・サイズ - 1] ページまでを含む) は最初のコンテナーに格納され、2 番目のエクステントは 2 番目のコンテナーに格納されます (以下同様)。最後のコンテナーが終わると、最初のコンテナーから再び処理が繰り返されます。この循環的プロセスにより、データのバランスが取られます。

さまざまなサイズのコンテナーが含まれる表スペースの場合、より大きなコンテナーの余分のスペースが利用されないため、各コンテナーを順番に移動する単純な方式は使用できません。このような場合、表スペース・マップが役立ちます。表スペース・マップは、表スペース内でエクステントがどのように配置されるかを管理し、物理コンテナー内のすべてのエクステントが利用されるようにします。

注: 以下の例では、コンテナー・サイズを決定する上でコンテナー・タグのサイズは考慮されていません。また、目的は単に例を示すことなので、かなり小さなコンテナー・サイズを使っていますが、そのサイズをお勧めするという意味ではありませんのでご注意ください。さらに、 1 つの表スペース内でいろいろなサイズのコンテナーを使っていますが、実際には、同じサイズのコンテナーを使用することをお勧めします。

例 1:

表スペースに 3 つのコンテナーがあり、各コンテナーには 80 個の使用可能なページが含まれ、表スペースのエクステント・サイズは 20 です。したがって各コンテナーには 4 つのエクステント (80/20) が含まれ、合計で 12 個のエクステントが存在します。これらのエクステントは、 178 ページの図 10 のようにディスクに配置されます。

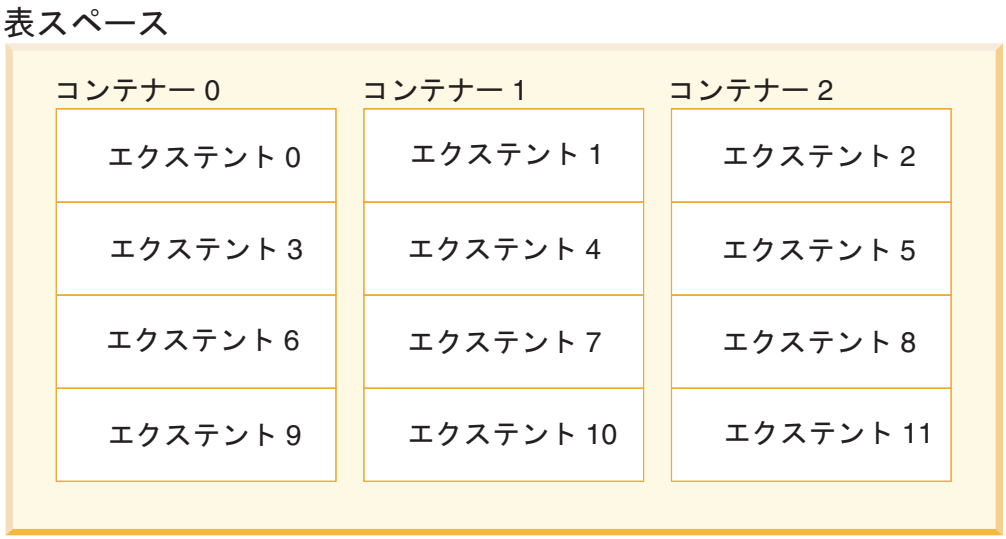


図 10. 3 つのコンテナー、12 のエクステントが含まれる表スペース

表スペース・マップを表示するには、スナップショット・モニターを使って表スペースのスナップショットを取ってください。例 1 では 3 つのコンテナーのサイズが等しく、表スペース・マップは以下のようになります。

```
Range   Stripe Stripe Max    Max  Start  End  Adj.  Containers
Number  Set    Offset Extent   Page  Stripe  Stripe
  [0]    [0]    0   11   239    0    3  0  3 (0, 1, 2)
```

範囲 (range) とはマップの一部であり、ストライプ内の隣接する範囲の中にすべて同じコンテナー・セットが含まれます。例 1 では、すべてのストライプ (0 から 3 まで) に 3 つのコンテナー (0、1、2) からなる同じセットが含まれ、これが 1 つの範囲と見なされます。

表スペース・マップの見出しは、Range Number (範囲番号)、 Stripe Set (ストライプ・セット)、Stripe Offset (ストライプ・オフセット)、 Maximum extent number addressed by the range (範囲でアドレス指定される最大エクステント番号)、Maximum page number addressed by the range (範囲でアドレス指定される最大ページ番号)、 Start Stripe (開始ストライプ)、End Stripe (終了ストライプ)、 Range adjustment (範囲調整)、および Container list (コンテナー・リスト) です。これらについては、例 2 で詳しく説明されます。

この表スペースは 179 ページの図 11 のように表すこともできます。この図では、各垂直線がコンテナー、各水平線がストライプ、各セル番号がエクステントです。

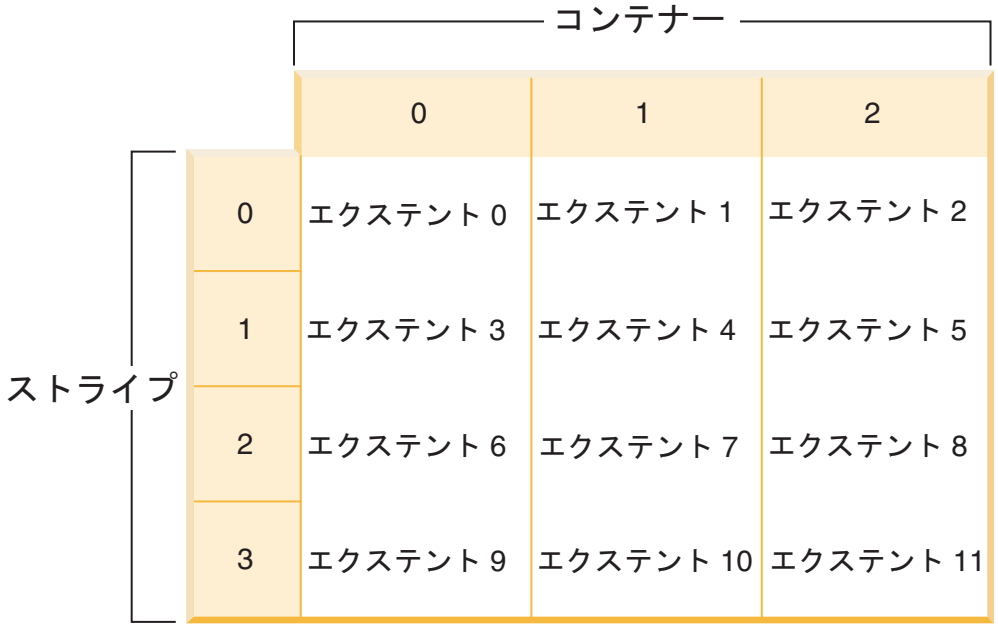


図 11. 3 つのコンテナー、 12 のエクステントが含まれる表スペース (ストライプを強調した場合)

例 2:

表スペースには 2 つのコンテナーがあり、最初のコンテナーのサイズは 100 ページ、2 番目のサイズは 50 ページ、そしてエクステント・サイズは 25 です。つまり最初のコンテナーには 4 つのエクステント、 2 番目のコンテナーには 2 つのエクステントが含まれます。この表スペースは、図 12 のように表すことができます。

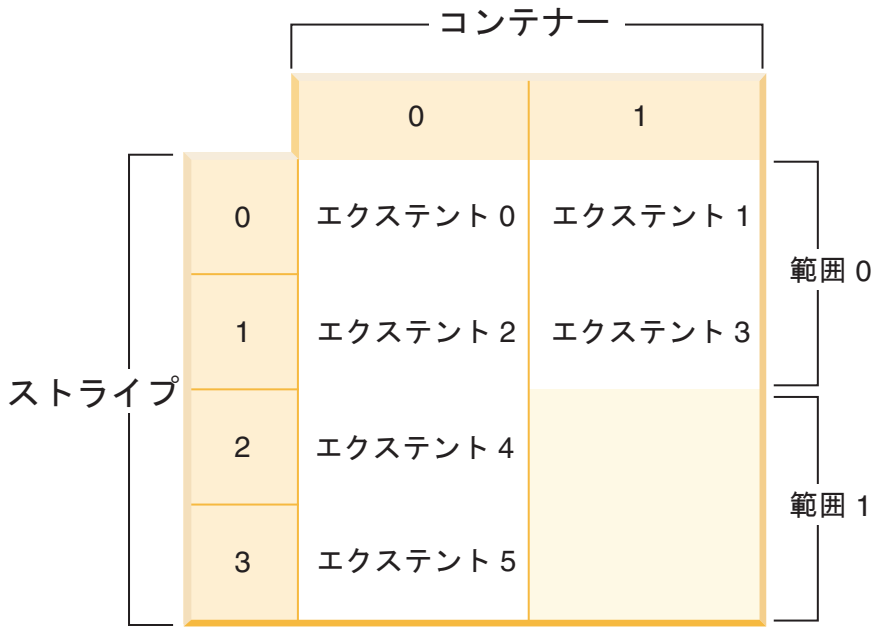


図 12. 2 つのコンテナーが含まれる表スペース (範囲を強調した図)

ストライプ 0 および 1 には両方のコンテナー (0、1) が含まれますが、ストライプ 2 および 3 には最初のコンテナー (0) しか含まれません。この 2 つのストライプ・セットが、それぞれ範囲になります。表スペース・マップは、以下の表スペース・スナップショットのようになります。

```
Range   Stripe Stripe Max    Max  Start  End  Adj.  Containers
Number  Set    Offset Extent   Page  Stripe  Stripe
  [0]    [0]    0    3    99    0    1  0  2 (0, 1)
  [1]    [0]    0    5   149    2    3  0  1 (0)
```

最初の範囲には 4 つのエクステントが存在するので、この範囲でアドレス指定される最大エクステント番号 (Max Extent) は 3 です。各エクステントには 25 ページが含まれるので、最初の範囲には全部で 100 ページが存在します。ページの番号付けもまた 0 から始まるので、この範囲でアドレス指定される最大ページ番号 (Max Page) は 99 になります。この範囲の最初のストライプ (開始ストライプ) は 0、最後のストライプ (終了ストライプ) は 1 です。この範囲には 0 と 1 というコンテナーがあります。ストライプ・オフセットは、ストライプ・セットの最初のストライプです (この場合はただ 1 つのストライプ・セットしか存在しないので 0)。範囲調整 (Adj.) とは、表スペースの中でデータをリバランスするときに使われるオフセットです。 (表スペースでスペースが追加またはドロップされるとき、リバランスが行われる場合があります。) リバランスが行われない限り、この値は常に 0 です。

2 番目の範囲には 2 つのエクステントがあり、前の範囲内でアドレス指定される最大エクステント番号が 3 であるため、この範囲でアドレス指定される最大エクステント番号は 5 です。2 番目の範囲には全部で 50 ページ (2 エクステント × 25 ページ) 存在し、前の範囲内でアドレス指定される最大ページ番号が 99 であるため、この範囲でアドレス指定される最大ページ番号は 149 です。この範囲はストライプ 2 で始まり、ストライプ 3 で終わります。

自動ストレージの表スペース

自動ストレージが使用可能になっていないデータベースに表スペースを作成する際、MANAGED BY SYSTEM 節または MANAGED BY DATABASE 節を指定する必要があります。 . これらの節を使用すると、システム管理スペース (SMS) 表スペースまたはデータベース管理スペース (DMS) 表スペースが節に応じて作成されます。どちらの場合にもコンテナーの明示的なリストを提供する必要があります。

データベースで自動ストレージが使用可能になっている場合は、他の選択肢があります。つまり、MANAGED BY AUTOMATIC STORAGE 節を指定するか、または MANAGED BY 節を除外する (これは自動ストレージの使用を暗黙指定します) ことができます。 . データベース・マネージャーはコンテナーを自動的に割り当てるため、この場合コンテナー定義を指定する必要はありません。

以下は、自動ストレージの表スペースを作成するいくつかのステートメントの例です。

```
CREATE TABLESPACE TS1
CREATE TABLESPACE TS2 MANAGED BY AUTOMATIC STORAGE
CREATE TEMPORARY TABLESPACE TEMPTS
CREATE USER TEMPORARY TABLESPACE USRTMP MANAGED BY AUTOMATIC STORAGE
CREATE LONG TABLESPACE LONGTS
```

自動ストレージの表スペース・タイプは異なる表スペース・タイプに見えますが、これは実際既存の SMS タイプおよび DMS タイプを拡張したものです。表スペースを REGULAR または LARGE 表スペースで作成した場合、これはファイル・コンテナーを持つ DMS 表スペースとして作成されます。表スペースを USER または SYSTEM TEMPORARY 表スペースで作成した場合、これはディレクトリー・コンテナーを持つ SMS 表スペースとして作成されます。

注: この動作は将来のデータベース・マネージャーのバージョンでは変更される可能性があります。

これらのコンテナーに関連した名前の形式は次のとおりです。

storage path/*instance name*/NODE####/*database name*/T#######/C#######.*EXT*

説明:

storage path
: データベースに関連付けられたストレージ・パス

instance name
: データベースが作成されたインスタンス

database name
: データベースの名前

NODE####
: データベース・パーティション番号 (例えば、NODE0000)

T#######
: 表スペース ID (例えば T0000003)

C#######
: コンテナー ID (例えば C0000012)

EXT
: 保管されているデータのタイプに基づく以下の拡張子

  CAT システム・カタログの表スペース

  TMP SYSTEM TEMPORARY 表スペース

  UTM USER TEMPORARY 表スペース

  USR ユーザーまたは REGULAR 表スペース

  LRG LARGE 表スペース

自動ストレージの REGULAR 表スペースと LARGE 表スペース、および DMS 表スペースとの間の違い

自動ストレージの REGULAR 表スペースと LARGE 表スペースは、DMS 表スペースとして作成され、DMS 表スペースに関連付けられたすべての規則と動作が適用されます。ただし、以下の表で示すように、ストレージの管理方法に関して違いがあります。

表 44. 非自動ストレージ表スペースの管理と自動ストレージ表スペースの管理との間の違い

非自動ストレージ	自動ストレージ
表スペースの作成時に、コンテナーのリストを明示的に指定する必要がある。	表スペースの作成時にコンテナーのリストを指定できないが、その代わりに、データベース・マネージャーによりコンテナーが自動的に割り当てられて、割り振られる。
デフォルトで表スペースの自動サイズ変更がオフになる (AUTORESIZE が NO に設定される)。	デフォルトで表スペースの自動サイズ変更がオンになる (AUTORESIZE が YES に設定される)。
表スペースに初期サイズを指定するために、INITIALSIZE 節を使用できない。	表スペースに初期サイズを指定するために、INITIALSIZE 節を使用できる。
ALTER TABLESPACE ステートメントを使用 (ADD、DROP、BEGIN NEW STRIPE SET などを指定して) してコンテナー操作を実行できる。	データベース・マネージャーでスペースが管理されるため、コンテナー操作を実行できない。
リダイレクトされたリストア操作を使用して表スペースに関連付けられたコンテナーを再定義できる。	データベース・マネージャーでスペースが管理されるため、リダイレクトされたリストア操作を使用して表スペースに関連付けられたコンテナーを再定義することはできない。

上の表内で説明したように、自動ストレージの REGULAR 表スペースまたは LARGE 表スペースを作成する際に、次の例のように CREATE TABLESPACE ステートメントの一部として初期サイズを指定できます。

```
CREATE TABLESPACE TS1 INITIALSIZE 100 M
```

初期サイズを指定しない場合は、データベース・マネージャーにより、デフォルト値の 32 MB が使用されます。

特定のサイズで表スペースを作成するために、データベース・マネージャーによりストレージ・パス内にファイル・コンテナーが作成されます。パスの間でスペースの分配が均等でないと、同じサイズのコンテナーを作成できません。そのため、すべてのストレージ・パスのフリー・スペースを同じにすることが重要です。

表スペースに対して自動サイズ変更を使用可能にした場合は、その中のスペースが使用されるにつれ、データベース・マネージャーにより既存のコンテナーが自動的に拡張され、(ストライプ・セットの使用により) 新規コンテナーが追加されます。コンテナーが拡張されたり追加されたりするとしても、バランスの再調整は行われません。

TEMPORARY 表スペース

SYSTEM TEMPORARY 表スペースは、ソートや結合などの操作の実行中に、データベース・マネージャーに必要な一時データを保持します。

これらのタイプの操作には、結果セットを処理するための余分のスペースが必要です。各データベースには、少なくとも 1 つの SYSTEM TEMPORARY 表スペースが必要です。デフォルトには、データベースの作成時に TEMPSPACE1 という 1 つ

の SYSTEM TEMPORARY 表スペースが作成されます。 IBMTEMPGROUP は、この表スペースに対するデフォルト・データベース・パーティション・グループです。

USER TEMPORARY 表スペースは、DECLARE GLOBAL TEMPORARY TABLE ステートメントを使用して作成した表からの一時データを保持します。宣言済み一時表を定義できるようにするには、該当する USE 特権を使用して、1 つ以上の USER TEMPORARY 表スペースを作成する必要があります。 USE 特権を付与するには、GRANT ステートメントを使用します。 USER TEMPORARY 表スペースは、デフォルトではデータベース作成の時点で*作成されません*。

単一の SMS TEMPORARY 表スペースの定義では、大半の REGULAR 表スペースで使用されているページ・サイズと等しいページ・サイズを指定することが推奨されています。そうすれば、一般的な環境と処理に適したサイズが得られます。しかし、TEMPORARY 表スペースの構成や処理を変えるとよい結果が得られる場合もあります。以下の点を考慮してください。

- 一時表はたいてい、一まとまりに順次アクセスされます。つまり、まとまった行が挿入されたり、ひとかたまりの順次行がフェッチされたりします。このため、比較的大きなページ・サイズを指定すると、特定量のデータを読み取るために必要な論理/物理ページの入出力要求が少なくなるので、一般的にパフォーマンスは向上します。ただし、平均的な一時表行サイズが 255 で除算したページ・サイズより小さい場合には、必ずしもパフォーマンスが向上するとは限りません。各ページには、ページ・サイズに関係なく最大 255 行を配置できます。例えば、15 バイト行の一時表が必要になる照会では、 4 KB の TEMPORARY 表スペース・ページ・サイズを使ったほうが効率がよくなります。該当する 255 行すべてを 4 KB ページ内に入れることができるからです。 8 KB (またはそれ以上) のページ・サイズを使用すると、それぞれの一時表ページごとに少なくとも 4 KB (またはそれ以上) のバイトのスペースが無駄になります。したがって、必要な入出力要求の数は減りません。
- データベース内の REGULAR 表スペースの 50 % を超える部分が同じページ・サイズを使用する場合は、 TEMPORARY 表スペースの定義で同じページ・サイズを指定するほうが得策かもしれません。そのような指定を行えば TEMPORARY 表スペースは、 REGULAR 表スペースの大部分または全部と同じバッファー・プール・スペースを共用できるからです。この結果、バッファー・プールのチューニングが簡単になります。
- TEMPORARY 表スペースを使って表を再編成する場合、 TEMPORARY 表スペースのページ・サイズは表のページ・サイズと一致しなければなりません。このため、異なるページ・サイズごとに定義された TEMPORARY 表スペースが必要です。それら個々のページ・サイズは、 TEMPORARY 表スペースを使って再編成できる既存の表によって使用されます。

  表を、ターゲット表スペースで直接再編成すれば、 TEMPORARY 表スペースを使わずに再編成を行うことができます。言うまでもなく、このタイプの再編成では、ターゲット表スペースに再編成プロセス用の余分のスペースが必要になります。
- 実際の作業環境のために SMS SYSTEM TEMPORARY 表スペースの中のシステム一時表を頼りにしている場合には、レジストリー変数 DB2_SMS_TRUNC_TMPTABLE_THRESH を使用することも考慮できます。シス

テム一時表は、ファイル・サイズ 0 に切り捨てられます。DB2_SMS_TRUNC_TMPTABLE_THRESH を使用してこのファイル・サイズをゼロ以外のままにしておくと、システム一時表の作成と切り捨てが繰り返されるパフォーマンス上のコストを防ぐことができます。新しいシステム一時表の必要には、パフォーマンス上のコストが伴う可能性があります。このレジストリー変数を使用するなら、システム上でシステム一時表を 0 (ゼロ) でないままにすることにより、システム一時表の作成と切り捨てを繰り返すことによるパフォーマンス上のコストを避けることができます。

- ページ・サイズの異なる複数の TEMPORARY 表スペースが存在すれば、オプティマイザーは、最も多くの行を保持できるバッファー・プール (ほとんどの場合、最大のバッファー・プール) を持つ TEMPORARY 表スペースを選択します。その場合はたいてい、TEMPORARY 表スペースの 1 つに十分のバッファー・プールを割り当て、他の TEMPORARY 表スペースには小さめのバッファー・プールを割り当てるのが賢明です。そのようなバッファー・プール割り当ては、メイン・メモリーの使用効率を向上させるのに役立ちます。例えば、カタログ表スペースが 4 KB ページを使用し、残りの表スペースが 8 KB ページを使用する場合、最適な TEMPORARY 表スペースの構成として、 1 つの 8 KB TEMPORARY 表スペースに大きなバッファー・プールを指定し、 1 つの 4 KB TEMPORARY 表スペースに小さなバッファー・プールを指定することが考えられます。
- 一般に、単一ページ・サイズの TEMPORARY 表スペースを複数定義しても、特に利点はありません。
- TEMPORARY 表スペースに関してはたいてい、DMS よりも SMS を選択するほうが優れています。その理由は、以下のとおりです。
  - DMS を使用する場合は、 SMS を使用する場合よりも、一時表の作成時に多くのオーバーヘッドがあります。
  - SMS ではディスク・スペースをオンデマンドで割り振りますが、 DMS では事前に割り振っておく必要があります。事前割り振りは問題になる場合があります。例えば、 TEMPORARY 表スペースに保持する一時データのストレージ要件がピーク時に非常に高く、ストレージ要件の平均はとても低いという場合があります。 DMS では、ピーク時のストレージ要件は事前割り振りしておく必要がありますが、 SMS では、オフピーク時に余分のディスク・スペースを他の目的で使用することができます。
  - データベース・マネージャーは、一時表ページをディスクに書き出さずに、メモリー内に維持しようとします。結果として、DMS を使用してもパフォーマンス上の効果はあまり期待できません。

SYSTEM TEMPORARY 表スペース・ページ・サイズが要件を満たしていることの確認:

より大きなレコード ID (RID) を使用すると、照会の結果セット内または位置指定の更新の行サイズが増えます。結果セット内の行サイズが既存の SYSTEM TEMPORARY 表スペースの行の最大長の制限に近い場合、より大きなページ・サイズを使って SYSTEM TEMPORARY 表スペースを作成する必要があるかもしれません。

前提条件

必要に応じて SYSTEM TEMPORARY 表スペースを作成するために、SYSCTRL または SYSADM 権限を持っていることを確認してください。

手順

照会または位置指定の更新に対して、SYSTEM TEMPORARY 表スペースの最大ページ・サイズが十分に大きいことを確認するには、以下のようにします。

1. 照会または位置指定の更新の結果セット内の最大行サイズを判別します。表の作成に使用した DDL ステートメントを使って、照会をモニターするか、最大行サイズを計算します。
2. 以下の例のように LIST TABLESPACES コマンドを使って表スペースをリストします。

```
  db2 LIST TABLESPACES SHOW DETAIL
...
 Tablespace ID                        = 1
 Name                                 = TEMPSPACE1
 Type                                 = System managed space
 Contents                             = System Temporary data
 State                                = 0x0000
   Detailed explanation:
     Normal
 Total pages                          = 10
 Useable pages                        = 10
 Used pages                           = 10
 Free pages                           = Not applicable
 High water mark (pages)              = Not applicable
 Page size (bytes)                    = 4096
 Extent size (pages)                  = 32
 Prefetch size (pages)                = 320
 Number of containers                 = 10
...
```

   出力の中で、Contents (内容) フィールドの値が「System Temporary data」である表スペースを探すことにより、SYSTEM TEMPORARY 表スペースを識別できます。それぞれの SYSTEM TEMPORARY 表スペースのページ・サイズ、および照会や更新の中で参照される表が作成された場所の表スペースのページ・サイズを書き留めておきます。
3. 以下のようにして、結果セット内の最も大きい行サイズが SYSTEM TEMPORARY 表スペースのページ・サイズに適合するかどうかを検査します。

```
maximum_row_size > maximum_row_length - 8 bytes (structure overhead in
                                                 single partition)
maximum_row_size > maximum_row_length - 16 bytes (structure overhead in DPF)
```

   ここで、maximum_row_size は結果セットの最大行サイズ、maximum_row_length はすべての SYSTEM TEMPORARY 表スペースの中で最も大きいページ・サイズに基づく可能な最大長です。表スペース・ページ・サイズから行の最大長を決定するには、「*SQL リファレンス 第 1 巻*」の『SQL と XML の制限値』を参照してください。

   最大行サイズが計算値より小さい場合、照会は DB2 UDB バージョン 8 のときと同じように実行されるため、このタスクをこれ以上進める必要はありません。
4. 表が作成された場所の表スペース・ページ・サイズに比べて少なくとも 1 ページ分だけ大きい SYSTEM TEMPORARY 表が存在しない場合、このサイズの SYSTEM TEMPORARY 表スペースを作成してください。例えば Windows オペ

レーティング・システムでは、ページ・サイズ 4 KB の表スペースで表を作成した場合、以下のように、ページ・サイズ 8 KB の追加の SYSTEM TEMPORARY 表スペースを作成します。

```
db2 CREATE SYSTEM TEMPORARY TABLESPACE tmp_tbsp
      PAGESIZE 8K
      MANAGED BY SYSTEM
      USING ('d:¥tmp_tbsp','e:¥tmp_tbsp')
```

表スペースのページ・サイズが 32 KB である場合、SYSTEM TEMPORARY 表スペースのページに適合するために、照会の中で選択される情報を減らすか、照会を分割することができます。例えば、表のすべての列を選択する代わりに、本当に必要な列だけを選択するか、特定の列のサブストリングだけを選択することにより、ページ・サイズ制限の超過を防ぐことができます。

SMS 表スペースと DMS 表スペースの比較

データを格納するために使用する表スペースの種類を決定する際には、いくつかのトレードオフについて考慮する必要があります。

SMS 表スペースの利点:

- スペースが必要になる時点まで、スペースがシステムによって割り振られることはありません。
- コンテナーの事前定義が不要なため、表スペースの作成に必要な初期作業が少なくてすみます。
- 範囲パーティション・データに対して作成した索引は、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。

DMS 表スペースの利点:

- 表スペースのサイズは、 ALTER TABLESPACE ステートメントを使用してコンテナーを追加または拡張することによって増加できます。既存のデータは、最適な入出力効率を保つために、新しいコンテナーのセットにわたって自動的にリバランスすることができます。
- 格納するデータのタイプに基づいて、表を複数の表スペースに分割することができます。
  - ロング・フィールド (LF) およびラージ・オブジェクト (LOB) データ
  - 索引
  - 通常表データ

  パフォーマンスを改善するため、または 1 つの表に格納できるデータの量を増やすために、表データを分割することができます。例えば、4 KB のページ・サイズを持つ大きい表スペースを使用する場合には、2 TB の通常の表データのテーブルと、2 TB の索引データを含む別の表スペースと、さらに 2 TB のロング・データを含む別の表スペースを持つことができます。そうしないで、これらの 3 つのタイプのデータを 1 つの表スペースに格納したとすると、合計スペースは 2 TB に制限されることになります。もっと大きなページ・サイズを使用すると、さらに多くのデータを格納できます。データベース・マネージャーのページ・サイズ制限の完全なリストについては、関連リンクを参照してください。
- 範囲パーティション・データに対して作成した索引は、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。

- ディスク上のデータのロケーションの制御は、オペレーティング・システムがこれをサポートしていれば可能です。
- 一般的に、十分に調整した DMS 表スペースの方が SMS 表スペースよりも性能が優れています。

注: パフォーマンス・センシティブ・アプリケーション、特に多数の挿入操作を含むものには、DMS 表スペースの使用をお勧めします。

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

一般的に、個人用の小さなデータベースでは、SMS 表スペースを使用して管理するのが一番簡単です。一方、サイズが大きく、これからも拡張するデータベースの場合は、 SMS 表スペースを TEMPORARY 表スペースやカタログ表スペースとしてのみ使用し、 DMS 表スペースは分割して、各表に複数のコンテナーを割り当てるのが効果的です。さらに、ロング・フィールド (LF) データおよび索引はそれぞれ独自の表スペースに保管するとよいでしょう。

デバイス・コンテナーを使って DMS 表スペースを使用する場合は、使用環境を調整および管理する必要があります。

SMS および DMS ワークロードに関する考慮事項

使用する表スペースのタイプ、および指定するページ・サイズを決定する際には、実際の環境でデータベース・マネージャーが管理している基本的なワークロードのタイプが影響する場合があります。

オンライン・トランザクション処理 (OLTP) のワークロードの特徴は、データに対してランダム・アクセスを行う必要があり、多くの場合、頻繁に挿入や更新を行ったり、または通常小さなデータ・セットを戻すような照会をするトランザクションです。アクセスがランダムであり、そのアクセスが 1 ないし数ページに対するものであるとすれば、プリフェッチが行われる可能性はあまりありません。

デバイス・コンテナーを使用する DMS 表スペースは、この状態において最適に実行されます。最高のパフォーマンスが必要なければ、ファイル・コンテナーを使用した DMS 表スペースまたは SMS 表スペースもまた、 OLTP ワークロードに対する適切な選択です。 FILE SYSTEM CACHING をオフにして、ファイル・コンテナーを持つ DMS 表スペースを使用すると、DMS ロー表スペース・コンテナーと同等のレベルで実行できることに注意してください。順次入出力が少ないか、まったくない場合、 CREATE TABLESPACE ステートメントの EXTENTSIZE および PREFETCHSIZE パラメーターの設定値は、入出力の効率にとって重要ではありません。ただし、十分なページ・クリーナー数を設定し、 *chngpgs_thresh* 構成パラメーターを使用することが重要です。

照会ワークロードの特徴は、データに対して順次アクセスまたは部分的な順次アクセスを行う必要のある、通常大きなデータ・セットを戻すトランザクションです。複数のデバイス・コンテナーを使用する DMS 表スペースで、しかも各コンテナーが別々のディスクにある場合には、並列プリフェッチが最も効率的に行われる可能性があります。 CREATE TABLESPACE ステートメントの PREFETCHSIZE パラメーターの値は、 EXTENTSIZE パラメーターの値にデバイス・コンテナーの数を乗算した値に設定しなければなりません。別の方法として、プリフェッチ・サイズに -1 を指定すると、データベース・マネージャーは適切なプリフェッチ・サイズを自動的に選択します。これによって、データベース・マネージャーは、すべてのコンテナーから並列にプリフェッチすることができます。コンテナー数が変更される場合、または多少なりとも積極的にプリフェッチする必要がある場合、 ALTER TABLESPACE ステートメントを使用して、それに応じて PREFETCHSIZE 値を変更することができます。

照会ワークロードの代わりの方法として、ファイル・システムが独自のプリフェッチを使用している場合には、ファイルを使用することもできます。ファイルは、ファイル・コンテナーを使用した DMS タイプ、または SMS タイプのいずれかが可能です。 SMS を使用する場合、入出力並列処理を達成するために、ディレクトリー・コンテナーを別々の物理ディスクにマップする必要があることに注意してください。

ワークロードが混在している場合には、 OLTP ワークロードのために単一の入出力要求をできるだけ効率的にすると同時に、照会ワークロードのために並列入出力の効率を最大化することが目標となります。

表スペースのページ・サイズを決定するための考慮事項は次のとおりです。

- 行のランダム読み取りおよび書き込み操作を実行する OLTP アプリケーションについては、不必要な行にはバッファー・プールのスペースを費やさないため、通常はページ・サイズは小さい方が望ましいです。
- 一度に多くの連続した行にアクセスする意思決定支援システム (DSS) アプリケーションについては、指定された数の行を読み取るのに必要な入出力要求が減るので、通常はページ・サイズは大きい方が望ましいです。
- ページ・サイズが大きいと、索引のレベルの数を減らすことができます。
- 大きいページは、長い行をサポートします。
- デフォルトの 4 KB ページでは、表は 500 列に制限されますが、より大きなページ・サイズ (8 KB、16 KB、32 KB) は 1012 列をサポートします。
- 表スペースに使用できる最大サイズは、表スペースのページ・サイズに比例します。

SMS および DMS 装置に関する考慮事項

表スペース・コンテナー用にファイル・システムのファイルを使用するか装置を使用するかを検討する際、考慮すべきオプションがいくつかあります。それはデータのバッファリングと LOB または LONG データを使用するかどうかです。

- **データのバッファリング**

  ディスクから読み取られる表データは通常、データベースのバッファー・プールで使用可能です。アプリケーションが実際にページを使用してしまうよりも前に、特に別のデータ・ページでバッファー・プール・スペースが必要な場合に、

データ・ページをバッファー・プールから解放できることもあります。システム管理スペース (SMS) またはデータベース管理スペース (DMS) ファイル・コンテナーを使用する表スペースについては、上記のファイル・システム・キャッシングにより入出力は必要なくなることもあります (ファイル・システム・キャッシングが使用されない場合、入出力は必要です)。

データベース管理スペース (DMS) を使用している表スペースは、ファイル・システムもキャッシュも使用しません。このような場合には、データベース・バッファー・プールのサイズを大きくし、ファイル・システム・キャッシュのサイズを小さくすることによって、デバイス・コンテナーを使用する DMS 表スペースは二重バッファリングを使用しないという事実を相殺することができます。

デバイス・コンテナーを使用する DMS 表スペースの入出力が同等の SMS 表スペースの入出力と比較して大きくなっていることを、システム・レベルのモニター・ツールが示す場合には、この差は、二重バッファリングが原因である可能性があります。

- **LOB データまたは LONG データの使用**

  アプリケーションが LOB データまたは LONG データのいずれかをリトリーブする場合には、データベース・マネージャーは、データをそのバッファーにキャッシュしません。アプリケーションがこれらのページの 1 つを要求するたびに、データベース・マネージャーはディスクからリトリーブしなければなりません。LOB または LONG データが SMS または DMS ファイル・コンテナーに格納される場合、ファイル・システム・キャッシュでバッファリングが行われ、結果としてパフォーマンスが向上することがあります。

  システム・カタログにはいくつかの LOB 列が含まれているので、システム・カタログは SMS 表スペースまたは DMS ファイル表スペースに入れておく必要があります。

表の表スペースを選択する際の考慮事項

表を表スペースにマップする方法を決定する際は、表の分散、表に含まれるデータの量とタイプ、および管理上の問題を考慮する必要があります。

表の分散

最低でも、選択する表スペースが、希望する分散を使用するデータベース・パーティション・グループ内にあるようにする必要があります。

表内のデータの量

1 つの表スペースの中に多くの小さな表を保管する計画である場合、その表スペースに対して SMS を使用することを検討してください。入出力とスペース管理を効率的に行う DMS の利点は、小さな表ではそれほど重要ではありません。 SMS の利点の方が (必要な場合に限り)、小さな表の場合はより魅力的です。表の 1 つがより大きいか、または表内のデータにより速くアクセスする必要がある場合には、小さなエクステント・サイズを持つ DMS 表スペースを検討してください。

非常に大きな表の場合は、それぞれに別個の表スペースを使用し、小さな表はすべて 1 つの表スペースにまとめるのが良いでしょう。このように分けると、表スペースの使用方法に基づいて適切なエクステント・サイズを選択することもできます。

表の中のデータのタイプ

例えば、あまり頻繁に使用されない履歴データの入った表がある場合、このデータに対する照会については、応答時間が長くてもよいとエンド・ユーザーは考えるかもしれません。その場合、履歴データ表に別の表スペースを使用して、その表スペースをアクセス速度が遅く、費用のかからない物理装置に割り当てるのも一案です。

あるいは、データが快適に使用できなければならなかったり迅速な応答が求められるいくつかの重要な表は、別扱いにするという方法もあります。そのような表は、そうした重要なデータ要件をサポートできる高速の物理装置に割り当てられた表スペースに入れておきます。

また、DMS 表スペースを使用して、表データを 3 つの異なる表スペースに分けることもできます。つまり 1 つは索引データ用、1 つはラージ・オブジェクト (LOB) およびロング・フィールド (LF) データ用、残る 1 つは通常表データ用です。これにより、データに最適な表スペース特性、およびそれらの表スペースをサポートする物理装置を選択することができます。例えば、利用可能な最高速の装置に索引データを入れると、パフォーマンスはかなり向上します。複数の DMS 表スペースにまたがって 1 つの表を分割する場合、ロールフォワード・リカバリーが使用可能であれば、これらの表スペースをまとめてバックアップおよびリストアすることを考慮してください。 SMS 表スペースは、複数の表スペースにまたがるこのようなタイプのデータ分散をサポートしません。

管理上の問題

一部の管理機能は、データベースや表のレベルではなく、表スペースのレベルで実行できます。例えば、データベースではなく表スペースのバックアップをとれば、時間とリソースの節約になります。こうすれば、大量の変更がある表スペースを頻繁にバックアップする一方、変更が非常に少ない表スペースは時折バックアップするだけにできます。

データベースや表スペースはリストアすることができます。互いに無関係な表が表スペースを共有していない場合、データベースのごく一部だけをリストアして、コストを減らすことができます。

互いに関連する表は一まとまりの表スペースに一緒に入れておくのがよいでしょう。そうした表は参照制約によって関連付けられる場合もあれば、定義された他の業務制約によって関連付けられる場合もあります。

ある特定の表を頻繁にドロップおよび再定義する必要がある場合、表をドロップするよりは DMS 表スペースをドロップする方がより効率的であるため、その表を独自の表スペースの中に定義したほうがよい場合があります。

表スペースの自動サイズ変更

自動サイズ変更の自動ストレージ表スペースが使用可能にされると、データベース・マネージャーはコンテナーの新規ストライプ・セットを追加することによって、フル・ファイル・システムの状態を自動的に処理することができます。

データベース・システムには、システム管理スペース (SMS) およびデータベース管理スペース (DMS) という 2 つの表スペース・タイプが存在することが可能です。SMS 表スペースに関連したコンテナーはファイル・システムのディレクトリーで、これらのディレクトリー内のファイルは、表スペース内のオブジェクトに連動して大きくなります。ファイルは、いずれかのコンテナーのファイル・システムの限界に達するか、またはデータベースの表スペース・サイズの限界に達するまで増大し続けます (を参照してください)。

DMS 表スペースはファイル・コンテナーまたはロー・デバイス・コンテナーで構成され、そのサイズはコンテナーが表スペースに割り当てられるときに設定されます。コンテナー内のスペースがすべて使用されている場合、表スペースはいっぱいであるとみなされます。ただし、SMS 表スペースに対しての場合とは異なり、ALTER TABLESPACE ステートメントを使用して表スペースにさらに多くのストレージ・スペースを与えることによって、コンテナーを追加または拡張することができます。また、DMS 表スペースには、*自動サイズ変更* というフィーチャーもあり、自動的にサイズ変更可能な DMS 表スペースのスペースが消費されるにつれ、データベース・システムは 1 つ以上のファイル・コンテナーによって表スペースを拡張する場合があります。 SMS 表スペースにも自動的な増大の類似した機能がありますが、「自動サイズ変更」という用語は DMS に対してのみ使用されます。

表スペースの自動サイズ変更には次の影響があります。

- 自動サイズ変更が使用可能な表スペースには、その表スペースに関連付けられた、バージョン 8.2.1 またはそれ以前のリリースでは認識されないメタデータがあります。それらのバージョンで自動サイズ変更が使用可能な表スペースを持つデータベースの使用を試みると、障害が発生します (多くの場合 SQL0980C または SQL0902C エラーが返されます)。データベースに接続しようとしたり、データベースをリストアしようとしたりすると、エラーが送信される場合があります。自動サイズ変更の表スペースを使用可能にしている場合、それらの表スペースで「自動サイズ変更」機能を使用不可にするとメタデータが除去され、データベースをバージョン 8.2.1 以前のリリースで使用できるようになります。
- 自動サイズ変更フィーチャーを使用不可にすると、このフィーチャーを後で使用可能にした場合に、INCREASESIZE および MAXSIZE に関連付けられた値が失われます。
- このフィーチャーは、ロー・デバイス・コンテナーを使用する表スペースには使用可能にできず、また自動的にサイズ変更できる表スペースにロー・デバイス・コンテナーを追加できません。これらの操作を試みると、エラー (SQL0109N) が発生します。ロー・デバイス・コンテナーを追加する必要がある場合、まずフィーチャーを使用不可にする必要があります。
- リダイレクトされたリストア操作で、コンテナー定義をロー・デバイス・コンテナーが組み込まれるように変更することはできません。この種の操作を試みると、エラー (SQL0109N) が発生します。

- 最大サイズによってデータベース・マネージャーによる表スペースの自動増加に限界が設けられるため、ユーザーによる表スペースの増加にも限界が設けられます。つまり、表スペースにスペースを追加する操作を実行する際、操作実行後のサイズは最大サイズ以下でなければなりません。スペースの追加は、ALTER TABLESPACE ステートメントの ADD、EXTEND、RESIZE、または BEGIN NEW STRIPE SET 節の使用によって行えます。

自動サイズ変更フィーチャーの使用可能化および使用不可化

デフォルトでは、自動サイズ変更フィーチャーは DMS 表スペースで使用可能になっていません。以下のステートメントでは、自動サイズ変更が使用可能ではない DMS 表スペースが作成されます。

```
CREATE TABLESPACE DMS1 MANAGED BY DATABASE
  USING (FILE '/db2files/DMS1' 10 M)
```

自動サイズ変更フィーチャーを使用可能にするには、CREATE TABLESPACE ステートメントに AUTORESIZE YES 節を指定します。

```
CREATE TABLESPACE DMS1 MANAGED BY DATABASE
  USING (FILE '/db2files/DMS1' 10 M) AUTORESIZE YES
```

DMS 表スペースを作成した後に、AUTORESIZE 節を指定した ALTER TABLESPACE ステートメントを使用することによっても、自動サイズ変更フィーチャーを使用可能または使用不可にできます。

```
ALTER TABLESPACE DMS1 AUTORESIZE YES
ALTER TABLESPACE DMS1 AUTORESIZE NO
```

以下の MAXSIZE と INCREASESIZE という 2 つの他の属性も自動サイズ変更表スペースに関連しています。

最大サイズ (MAXSIZE)

CREATE TABLESPACE ステートメントの MAXSIZE 節は、表スペースの最大サイズを定義します。例えば、以下のステートメントは、(データベースに複数のデータベース・パーティションがある場合、1 つのデータベース・パーティションにつき) 100 メガバイトまで増やせる表スペースを作成します。

```
CREATE TABLESPACE DMS1 MANAGED BY DATABASE
  USING (FILE '/db2files/DMS1' 10 M)
       AUTORESIZE YES MAXSIZE 100 M
```

MAXSIZE NONE 節は、表スペースに最大限界がないことを指定します。表スペースの増大は、ファイル・システムの限界か、または表スペースの限界に達するまで続きます。SQL および XML の制限値を参照してください。 MAXSIZE 節を指定しないと、自動サイズ変更フィーチャーが使用可能になっている場合は、最大限界はありません。

以下の各例に示すように、ALTER TABLESPACE ステートメントを使用して、すでに自動サイズ変更が使用可能になっている表スペースの MAXSIZE の値を変更します。

```
ALTER TABLESPACE DMS1 MAXSIZE 1 G
ALTER TABLESPACE DMS1 MAXSIZE NONE
```

最大サイズを指定した場合、データベース・マネージャーはコンテナーの増加の整合性を保とうとするため、データベース・マネージャーが施行する実際の値は指定された値よりも若干小さくなる可能性があります。

増加サイズ (INCREASESIZE)

CREATE TABLESPACE ステートメントの INCREASESIZE 節は、表スペース内にフリー・エクステントがないが、1 つ以上のエクステントが要求された場合に表スペースを増やすために使用されるスペースの量を定義します。以下の各例に示すように、値は明示的なサイズまたはパーセントで指定できます。

```
CREATE TABLESPACE DMS1 MANAGED BY DATABASE
  USING (FILE '/db2files/DMS1' 10 M)
       AUTORESIZE YES INCREASESIZE 5 M

CREATE TABLESPACE DMS1 MANAGED BY DATABASE
  USING (FILE '/db2files/DMS1' 10 M)
       AUTORESIZE YES INCREASESIZE 50 PERCENT
```

パーセントの値は、表スペースを増やす必要が生じるたびに INCREASESIZE の値で指定された増加量が計算され、その時点の表スペース・サイズのパーセントに基づいて増加することを意味します。例えば、表スペースのサイズが 20 MB で INCREASESIZE の値が 50 % の場合、表スペースは最初に 10 MB 増加して (サイズが 30 MB になり)、次回は 15 MB 増加します。

自動サイズ変更フィーチャーを使用可能にした際に INCREASESIZE 節を指定しなかった場合は、データベース・マネージャーにより適切な使用値が決定されますが、これは表スペースの存続期間中に変更される場合があります。AUTORESIZE や MAXSIZE と同様、ALTER TABLESPACE ステートメントを使用して INCREASESIZE の値を変更できます。

サイズの増加を指定した場合、データベース・マネージャーによって使用される実際の値が、指定した値と若干異なる場合があります。この使用値の調整は、表スペース内のコンテナー全体で増加の整合性を保つために行われます。

表スペースを拡張する方法

自動的にサイズを変更できる表スペースで、既存のスペースがすべて使用され、さらに多くのスペースが要求される場合、データベース・マネージャーは表スペースのサイズの増加を試みます。データベース・マネージャーは、バランスの再調整が発生しないように、表スペース内の拡張が可能なコンテナーを判別します。データベース・マネージャーは、表スペース・マップ (マップは表スペースのストレージ・レイアウトを記述する) の最新の範囲に存在するコンテナーのみを拡張し、またそれらのコンテナーをすべて同じ量で拡張します。

例えば、以下のステートメントを検討してみましょう。

```
CREATE TABLESPACE TS1 MANAGED BY DATABASE
  USING (FILE 'C:¥TS1CONT' 1000, FILE 'D:¥TS1CONT' 1000,
         FILE 'E:¥TS1CONT' 2000, FILE 'F:¥TS1CONT' 2000)
  EXTENTSIZE 4
  AUTORESIZE YES
```

データベース・マネージャーがメタデータのために各コンテナーの小さな部分 (1 つのエクステント) を使用することに注意して、下に示す CREATE TABLESPACE

ステートメントに基づいて表スペースに対して作成された表スペース・マップを参照してください。(表スペース・マップは、表スペースのスナップショットからの出力の一部です)。

```
Table space map:

 Range  Stripe Stripe  Max         Max  Start  End    Adj.   Containers
 Number Set    Offset  Extent      Page Stripe Stripe
 [   0] [   0]      0     995      3983      0    248   0    4 (0,1,2,3)
 [   1] [   0]      0    1495      5983    249    498   0    2 (2,3)
```

表スペース・マップは、ID 2 または ID 3 を持つコンテナー (E:¥TS1CONT および F:¥TS1CONT) のみがマップの最新の範囲であることを示しています。そのため、データベース・マネージャーがこの表スペースのコンテナーを自動的に拡張するときには、これら 2 つのコンテナーのみが拡張されます。

注: コンテナーのサイズがすべて同じである表スペースを作成する場合、マップの範囲は 1 つだけになります。そのような場合、データベース・マネージャーはそれぞれのコンテナーを拡張します。拡張をコンテナーのサブセットのみに制限することを避けるには、同じサイズのコンテナーを持つ表スペースを作成してください。

前に説明したように、表スペースの最大サイズに対して限界を指定することもできますし、NONE の値を指定することもできます。後者の場合、増加の限界は設けられません。 NONE つまり限界なしを指定した場合は、ファイル・システムの限界または表スペースの限界によって上限が定義され、データベース・マネージャーでは、この上限を超える表スペースのサイズの増加が試みられません。ただし、その限界に達する前に、コンテナーを増やそうとしてもファイル・システムがいっぱいであるために失敗することがあります。そうなると、データベース・マネージャーはそれ以上表スペースのサイズを増やさず、アプリケーションに「スペース不足」状態を返します。この状態を解決するには 2 つの方法があります。

- いっぱいになっているファイル・システムで使用可能なスペースの量を増やす。
- 当該のコンテナーが表スペース・マップの最新の範囲から外れるように、表スペースに対してコンテナー操作を実行する。これを行う最も簡単な方法は、新しいコンテナー・セットを持つ表スペースに新規ストライプ・セットを追加することで、またベスト・プラクティスは、コンテナーをすべて同じサイズにすることです。 BEGIN NEW STRIPE SET 節を指定した ALTER TABLESPACE ステートメントを使用して、新規ストライプ・セットを追加することができます。新規ストライプ・セットを追加することによって、表スペース・マップに新しい範囲が追加されます。新しい範囲が追加されると、データベース・マネージャーが自動的に拡張しようとするコンテナーはこの新規ストライプ・セット内に入り、古いコンテナーは変更されません。

  注: ユーザーが開始したコンテナー操作がペンディング状態であるか、または後続のバランスの再調整が進行中の場合、操作がコミットされるかまたはバランスの再調整が完了するまで自動サイズ変更フィーチャーは使用不可になります。

例えば、DMS 表スペースの場合に、表スペースに同じサイズの 3 つのコンテナーがあり、それぞれが独自のファイル・システムに存在するとします。表スペースに対して作業が行われるにつれ、データベース・マネージャーはこれら 3 つのコンテナーを自動的に拡張していきます。最終的に、ファイル・システムのいずれかがいっぱいになり、対応するコンテナーは増加しなくなります。ファイル・システム上

でこれ以上使用可能なフリー・スペースを設けられない場合、当該のコンテナーが表スペース・マップの最新の範囲から外れるように、表スペースに対してコンテナー操作を実行する必要があります。その場合、2 つのコンテナーを指定して新規ストライプ・セットを追加するか (スペースがまだある各ファイル・システム上に1つ)、または指定するコンテナーを増やすかまたは減らすこともできます (この場合にも、追加される各コンテナーのサイズが同じになり、使用されているそれぞれのファイル・システム上に増加のための十分な余裕があるようにする)。データベース・マネージャーが表スペースのサイズを増加しようとすると、古いコンテナーの拡張を試みる代わりに、この新規ストライプ・セット内のコンテナーに対して拡張を試みます。

モニター

DMS 表スペースの自動サイズ変更に関する情報は、表スペースのモニターのスナップショット出力の一部として表示されます。次の例に示すように、増加サイズおよび最大サイズの値がこの出力内に含まれます。

```
Auto-resize enabled                    = Yes or No
Current tablespace size (bytes)        = ###
Maximum tablespace size (bytes)        = ### or NONE
Increase size (bytes)                  = ###
Increase size (percent)                = ###
Time of last successful resize         = DD/MM/YYYY HH:MM:SS.SSSSSS
Last resize attempt failed             = Yes or No
```

コンテナーを追加またはドロップした後の自動プリフェッチ・サイズ調整

データベース・マネージャーは、バージョン 8.2 (またはそれ以降) で作成した表スペースのデフォルトが自動プリフェッチ・サイズになるようにセットアップされています。

コンテナーを追加またはドロップした後に表スペースのプリフェッチ・サイズの更新を忘れる可能性がある場合、データベース・マネージャーがプリフェッチ・サイズを自動的に決めるようにすることを考慮できます。プリフェッチ・サイズの更新を忘れると、データベースのパフォーマンスが大幅に低下することがあります。

データベース・マネージャーは以下の公式を使用して、表スペースのプリフェッチ・サイズを計算します。

プリフェッチ・サイズ = (コンテナーの数) X (コンテナーごとの物理スピンドルの数) X エクステント・サイズ

表スペースのプリフェッチ・サイズを AUTOMATIC に設定しない方法には、以下の 3 つがあります。

- 特定のプリフェッチ・サイズで表スペースを作成する。プリフェッチ・サイズの値を手動で選択するということは、表スペースに関連したコンテナーの数が調整されたときにはいつでも、自分で忘れずにプリフェッチ・サイズを調整するということです (調整が必要な場合)。
- 表スペースを作成する際にはプリフェッチ・サイズを使用せず、 *dft_prefetch_sz* データベース構成パラメーターを AUTOMATIC ではない値に設定する。表スペースを作成するときにプリフェッチ・サイズを明示的に指定しない場合、データ

ベース・マネージャーはこのパラメーターを調べます。AUTOMATIC 以外の値が検出されると、その値がデフォルトのプリフェッチ・サイズとして使用されます。表スペースに関連したコンテナーの数が調整されたときにはいつでも、自分で忘れずにプリフェッチ・サイズを調整しなければなりません (調整が必要な場合)。

- ALTER TABLESPACE ステートメントを使用して、プリフェッチ・サイズを手動で変更する。

DB2_PARALLEL_IO の使用

プリフェッチ要求は、表スペースの並列処理に基づいて、要求がプリフェッチ・キューにサブミットされる前に、複数の小さなプリフェッチ要求に分割されます。コンテナーごとの物理スピンドル数を定義し、表スペース上の並列入出力に影響を与えるために、DB2_PARALLEL_IO レジストリー変数が使用されます。並列入出力がオフであると、表スペースの並列処理はコンテナーの数と等しくなります。並列入出力がオンであると、表スペースの並列処理は、コンテナーの数に DB2_PARALLEL_IO レジストリー変数で指定した値を乗算した結果と等しくなります。 (つまり、表スペースの並列処理は、プリフェッチ・サイズを表スペースのエクステント・サイズで割った値と等しくなります。)

以下の例は、 DB2_PARALLEL_IO レジストリー変数がプリフェッチ・サイズにどのように影響を与えるかを示しています。 (以下の表スペースはすべて、AUTOMATIC プリフェッチ・サイズが指定されているものと想定してください。)

- DB2_PARALLEL_IO=*
  - すべての表スペースはデフォルト (各コンテナーのスピンドル数が 6) を使用します。並列入出力がオンであると、プリフェッチ・サイズは 6 倍大きくなります。
  - すべての表スペースでは、並列入出力がオンになります。プリフェッチ要求は複数の小さな要求に分割されます。それぞれの要求は、プリフェッチ・サイズをエクステント・サイズで割った値と等しくなります (または、コンテナー数にスピンドル数を乗じた値に等しくなります)。
- DB2_PARALLEL_IO=*:3
  - すべての表スペースは、コンテナーごとのスピンドル数として 3 を使用します。
  - すべての表スペースでは、並列入出力がオンになります。
- DB2_PARALLEL_IO=*:3,1:1
  - すべての表スペースはコンテナーごとのスピンドル数として 3 を使用しますが、表スペース 1 は例外で 1 を使用します。
  - すべての表スペースでは、並列入出力がオンになります。

ファイル・システム・キャッシングを使用しない表スペース

UNIX、Linux、および Windows でバッファリングのない I/O を使用可能または使用不可にする際に推奨される方式は、表スペース・レベルで行う方式です。

この方式では、特定の表スペースにバッファリングのない I/O を使用可能または使用不可にすると同時に、データベースの物理レイアウトに依存することを避けることができます。また、データベース・マネージャーが、各ファイルで最適な I/O (バッファリングありまたはバッファリングなし) を判別することができます。

バッファリングのない I/O を使用可能にし、特定の表スペースでファイル・キャッシングを使用不可にするには、NO FILE SYSTEM CACHING 節を使用します。バッファリングのない I/O が使用可能にされている場合、データベース・マネージャーはプラットフォームに基づいて、直接 I/O (DIO) と並行 I/O (CIO) のどちらを使用するかを自動的に判別します。 CIO の方がパフォーマンスが向上するため、データベース・マネージャーは CIO がサポートされていれば CIO を使用します。どちらを使用するかを指定するユーザー・インターフェースはありません。

バッファリングのない I/O から最大の効果を引き出すため、バッファー・プールのサイズを増やす必要がある可能性もあります。ただし、セルフチューニング・メモリー・マネージャーが使用可能にされ、バッファー・プール・サイズが AUTOMATIC に設定されている場合、データベース・マネージャーはバッファー・プール・サイズを最適のパフォーマンスにセルフチューニングします。このフィーチャーは、バージョン 9 より前は使用できなかったことに注意してください。

ファイル・システム・キャッシングを使用不可または使用可能にするには、CREATE TABLESPACE または ALTER TABLESPACE ステートメントでそれぞれ NO FILE SYSTEM CACHING または FILE SYSTEM CACHING 節を指定します。デフォルト設定は、どちらの節も指定されていない場合に使用されます。 ALTER TABLESPACE の場合、新しいキャッシング・ポリシーを有効にするためにデータベースへの既存の接続を終了する必要があります。

注: 属性がデフォルトから FILE SYSTEM CACHING または NO FILE SYSTEM CACHING に変更されると、それをデフォルトに戻すメカニズムはありません。

ファイル・システム・キャッシングを使用可能および使用不可にするこの方式は、表スペース・レベルで I/O モード (バッファリングありまたはバッファリングなし) を制御することを可能にします。

注: ロング・フィールド (LF) データおよびラージ・オブジェクト (LOB) データへの I/O アクセスは、対象となる表スペースの設定に関係なく、SMS および DMS コンテナーの両方でバッファーに入れられることに注意してください。

GET SNAPSHOT FOR TABLESPACES コマンドを使用すると、ファイル・システム・キャッシングに関する節の現在の設定を照会することができます。例えば、以下に示すのは、DB2 GET SNAPSHOT FOR TABLEPSACES ON db1 出力の断片です。

```
Tablespace name                            = USERSPACE1
Tablespace ID                              = 2
Tablespace Type                            = database managed space
Tablespace Content Type                    = All permanent data. Large table space.
Tablespace Page size (bytes)               = 4096
Tablespace Extent size (pages)             = 32
Automatic Prefetch size enabled            = Yes
Buffer pool ID currently in use            = 1
Buffer pool ID next startup                = 1
Using automatic storage                    = Yes
Auto-resize enabled                        = Yes
File system caching                        = No
```

```
Tablespace State                          = 0x'00000000'
 Detailed explanation:
   Normal
Tablespace Prefetch size (pages)          = 32
Total number of pages                     = 256
```

UNIX、Linux、および Windows でバッファリングのない I/O を使用可能または使用不可にする別の方式

一部の UNIX プラットフォームでは、MOUNT オプションを使用することにより、ファイル・システム・レベルのファイル・システム・キャッシングを使用不可にすることができます。詳しくは、オペレーティング・システムの資料を参照してください。ただし、表スペース・レベルとファイル・システム・レベルでファイル・システム・キャッシングを使用できないようにした場合の違いを理解することが重要です。表スペース・レベルでは、ファイルを開く際にファイル・システム・キャッシングを使用するかどうかをデータベース・マネージャーが制御します。ファイル・システム・レベルでは、特定のファイル・システムに存在するすべてのファイルが、ファイル・システム・キャッシングなしで開かれます。 AIX などの一部のプラットフォームでは、このフィーチャーを使用する前提となる特定の要件があります (例えば、読み取りおよび書き込みアクセスのシリアライゼーション)。データベース・マネージャーがこれらの要件に準拠していて、ターゲット・ファイル・システムに非 DB2 ファイルが含まれている場合は、このフィーチャーを使用可能にする前に、要件についてオペレーティング・システムの資料を参照してください。

注: バージョン 8.1 FixPak 4 で導入された、現在は推奨されていないレジストリー変数 DB2_DIRECT_IO により、AIX JFS2 上のロング・フィールド・データ、ラージ・オブジェクト・データ、および TEMPORARY 表スペースを除くすべての SMS コンテナーでファイル・システム・キャッシングが使用不可になります。バージョン 9.1 以降でこのレジストリー変数を設定することは、NO FILE SYSTEM CACHING 節を指定してすべての表スペース、SMS および DMS を変更することと等価です。ただし、DB2_DIRECT_IO を使用することはお勧めできません。この変数は、今後のリリースで削除されます。その代わりとして、表スペースのレベルで NO FILE SYSTEM CACHING を使用可能にしてください。

Windows でバッファリングのない I/O を使用可能/使用不可にする別の方式

以前のリリースでは、パフォーマンス・レジストリー変数 DB2NTNOCACHE を使用して、すべてのDB2 ファイルについてファイル・システム・キャッシングを使用不可にすることができました。そのようにすれば、より多くのメモリーがデータベースに利用できるようになるため、バッファー・プールやソート・ヒープの量を増やすことができます。DB2NTNOCACHE は、バージョン 9.5 では推奨されておらず、今後のリリースでは除去される可能性があります。 DB2NTNOCACHE と NO FILE SYSTEM CACHING 節の使用の違いは、表スペースを選択してキャッシングを使用不可にできるかどうかです。バージョン 9.5 からは、NO FILE SYSTEM CACHING がデフォルトとして使用されているため、FILE SYSTEM CACHING を明示的に指定しない限り、インスタンスに新規に作成された表スペースしか含まれない場合は、このレジストリー変数を設定してファイル・システム・キャッシングをインスタンス全体に渡って使用不可にする必要はなくなりました。

パフォーマンスの考慮事項

バッファリングのない I/O は、本来はパフォーマンス向上のために使用されます。ただし場合によっては、小さいバッファー・プール・サイズと小さいファイル・システム・キャッシュの組み合わせによって (そればかりとは限りませんが)、パフォーマンスの低下が引き起こされることがあります。パフォーマンスを改善するための提案を以下に示します。

- セルフチューニング・メモリー・マネージャーが使用可能でない場合は、それを使用可能にして、`ALTER BUFFERPOOL <name> SIZE AUTOMATIC` を使用して、バッファー・プール・サイズを自動に設定します。これによって、データベース・マネージャーは、バッファー・プール・サイズを自己調整できます。
- セルフチューニング・メモリー・マネージャーを使用可能にしない場合は、パフォーマンスが改善されるまで、10 または 20 パーセントずつバッファー・プール・サイズを大きくします。
- セルフチューニング・メモリー・マネージャーを使用可能にしない場合は、「FILE SYSTEM CACHING」を使用するように表スペースを変更します。これにより、バッファリングのない I/O が基本的に使用不可になり、コンテナー・アクセスはバッファー I/O に戻ります。

実動システムでこれを実装する前に、制御された環境でパフォーマンスの調整をテストしてください。

表スペース・コンテナー用にファイル・システムのファイルを使用するか、装置を使用するかを選択するときは、ファイル・システム・キャッシングについて考慮する必要があります。ファイル・システム・キャッシングは、次のように実行されます。

- DMS ファイル・コンテナー (およびすべての SMS コンテナー) の場合、オペレーティング・システムはファイル・システム・キャッシュ中のページをキャッシュすることがある (表スペースに NO FILESYSTEM CACHING が定義されていない場合)
- DMS デバイス・コンテナー表スペースの場合、オペレーティング・システムはファイル・システム・キャッシュ中のページをキャッシュしない

新規表スペース・コンテナー用のデフォルトのファイル・システム・キャッシング・メカニズムとして CIO/DIO を使用する

ほとんどの AIX、Linux、Solaris、およびWindows プラットフォーム上で新規に作成された表スペース・コンテナーのデフォルトの入出力メカニズムは CIO/DIO (並行 I/O または直接 I/O) です。このデフォルトを指定すると、多量のトランザクション処理ワークロードおよびロールバックに対してバッファー付き I/O を超えてスループットが増加します。

FILE SYSTEM CACHING または NO FILE SYSTEM CACHING 属性は、入出力操作がファイル・システム・レベルでキャッシュに入れられるかどうかを指定します。

- FILE SYSTEM CACHING は、ターゲット表スペースでのすべての入出力操作がファイル・システム・レベルでキャッシュに入れられることを指定します。
- NO FILE SYSTEM CACHING は、すべての入出力操作がファイル・システム・レベルのキャッシュを迂回することを示します。

注: DMS 表スペースを使用する場合、ロング・フィールド (LF) データとラージ・オブジェクト (LOB) データ用に別個の表スペースを使用して、REGULAR 表スペースが影響を受けないようにする必要があります。 (SMS 表スペースの場合、CIO/DIO (NO FILE SYSTEM CACHING) 属性は使用不可にされます。)

以下のインターフェースには FILE SYSTEM CACHING 属性が含まれます。

- CREATE TABLESPACE ステートメント
- CREATE DATABASE コマンド
- sqlecrea() API (SQLETSDESC 構造の *sqlfscaching* フィールドを使用)

この属性が CREATE TABLESPACE ステートメント、または CREATE DATABASE コマンドで指定されていない場合、データベース・マネージャーはプラットフォームおよびファイル・システムのタイプに基づくデフォルトの動作を使用して要求を処理します。正確な動作については、『ファイル・システム・キャッシュ構成』を参照してください。 sqlecrea() API の場合、フィールド *sqlfscaching* の `0x2` の値が、データベース・マネージャーにデフォルト設定を使用するように指示します。

以下のツールが現在、FILE SYSTEM CACHING 属性の値を解釈することに注意してください。

- GET SNAPSHOT FOR TABLESPACES コマンド
- db2pd –tablespaces コマンド
- db2look –d <dbname> –l コマンド

db2look では、FILE SYSTEM CACHING 属性が指定されていない場合、出力にはこの属性は含まれません。

例

データベースおよび関連したすべての表スペース・コンテナーが AIX JFS ファイル・システム上にあり、次のステートメントが発行されたとします。

```
DB2 CREATE TABLESPACE JFS2
```

前のバージョンでは、属性が指定されていない場合、データベース・マネージャーは I/O メカニズムにバッファー付き I/O (FILE SYSTEM CACHING) を使用しました。バージョン 9.5 では、データベース・マネージャーは NO FILE SYSTEM CACHING を使用します。

ファイル・システム・キャッシュ構成

オペレーティング・システムは、デフォルトでは、ディスクとの間で読み書きされるファイル・データをキャッシュに入れます。

標準的な読み取り操作では、物理ディスク・アクセスにより、データがディスクからファイル・システム・キャッシュに読み取られ、そのデータがキャッシュからアプリケーション・バッファーにコピーされます。同様に、書き込み操作では、物理ディスク・アクセスにより、データがアプリケーション・バッファーからファイル・システム・キャッシュにコピーされ、そのデータがキャッシュから物理ディスクにコピーされます。 CREATE TABLESPACE ステートメントの FILE SYSTEM CACHING 節には、このファイル・システム・レベルでのデータ・キャッシング動

作が反映されます。データベース・マネージャーは、自身のデータ・キャッシングをバッファー・プールを使って管理します。バッファー・プールのサイズが適切に調整されていれば、ファイル・システム・レベルのキャッシングは必要ありません。

注: データベース・マネージャーは、ページのキャッシュを無効化することによって、既にほとんどの DB2 データのキャッシングを防止していますが、AIX の一時データと LOB は例外です。

ファイル・システム・レベルとバッファー・プールの両方でキャッシングを行うと、二重キャッシングに余分の CPU サイクルが必要なため、性能が低下する場合があります。この二重キャッシングを回避するため、大抵のファイル・システムにはファイル・システム・レベルでのキャッシングを使用不可にするフィーチャーがあります。これは一般に、バッファリングのない *I/O* と呼ばれています。UNIX では、このフィーチャーは一般に、*Direct I/O (DIO)* として知られています。Windows では、これは FILE_FLAG_NO_BUFFERING フラグを立ててファイルを開くことに相当します。さらに、IBM JFS2 や Symantec VERITAS VxFS などの一部のファイル・システムは、拡張された Direct I/O である高性能な *Concurrent I/O (CIO)* フィーチャーもサポートしています。データベース・マネージャーは、NO FILE SYSTEM CACHING 表スペース節を使用してこのフィーチャーをサポートしています。これが設定されているとき、データベース・マネージャーは、CIO が存在するファイル・システムで、自動的にこのフィーチャーを利用します。このフィーチャーは、ファイル・システム・キャッシュのメモリー要件を削減して他の用途に使用できるメモリーを増やすのに役立つことがあります。

バージョン 9.5 より前は、NO FILE SYSTEM CACHING も FILE SYSTEM CACHING も指定されていない場合は、キーワード FILE SYSTEM CACHING が暗黙指定されていました。バージョン 9.5 では、どちらのキーワードも指定されていない場合、デフォルトの NO FILE SYSTEM CACHING が使用されます。この変更は、新規に作成された表スペースにのみ影響します。バージョン 9.5 より前に作成された既存の表スペースは、影響を受けません。この変更は、AIX、Linux、Solaris、および Windows に適用されますが、以下の例外についてはデフォルトの動作は FILE SYSTEM CACHING のままです。

- AIX JFS
- Solaris 非 VxFS
- Linux for System z™
- すべての SMS TEMPORARY 表スペース・ファイル
- SMS 永続表スペース・ファイル。ただし、ロング・フィールド (LF) データ・ファイルとラージ・オブジェクト (LOB) データ・ファイルを除く。

デフォルト設定をオーバーライドするには、FILE SYSTEM CACHING または NO FILE SYSTEM CACHING を指定します。

サポートされる構成

202 ページの表 45 は、ファイル・システム・キャッシングなしで表スペースを使用する際にサポートされる構成を示しています。これは以下の点も示しています。(a) それぞれの場合に DIO と拡張 DIO のどちらが使用されるか。(b)表スペースでプラ

ットフォームおよびファイル・システム・タイプに応じて NO FILE SYSTEM CACHING と FILE SYSTEM CACHING のどちらも指定されない場合のデフォルトの動作。

表 45. ファイル・システム・キャッシングを使用しない表スペースのサポートされる構成

プラットフォーム	ファイル・システムのタイプおよび必要な最小レベル	NO FILE SYSTEM CACHING が指定されているときに、データベース・マネージャーがサブミットした DIO 要求と CIO 要求	NO FILE SYSTEM CACHING と FILE SYSTEM CACHING のどちらも指定されない場合のデフォルトの動作
AIX 5.3+	Journal File System (JFS)	DIO	FILE SYSTEM CACHING (注 1 を参照。)
AIX 5.3+	Concurrent Journal File System (JFS2)	CIO	NO FILE SYSTEM CACHING
AIX 5.3+	VERITAS Storage Foundation for DB2 4.1 (VxFS)	CIO	NO FILE SYSTEM CACHING
HP-UX 11i (PA-RISC)	VERITAS Storage Foundation 4.1 (VxFS)	CIO	FILE SYSTEM CACHING
HP-UX バージョン 11i v2 (Itanium®)	VERITAS Storage Foundation 4.1 (VxFS)	CIO	FILE SYSTEM CACHING
Solaris 9	UNIX File System (UFS)	DIO	FILE SYSTEM CACHING (注 2 を参照。)
Solaris 10	UNIX File System (UFS)	CIO	FILE SYSTEM CACHING (注 2 を参照。)
Solaris 9、10	VERITAS Storage Foundation for DB2 4.1 (VxFS)	CIO	NO FILE SYSTEM CACHING
Linux ディストリビューション SLES 9+ および RHEL 4+ (対応アーキテクチャー: x86、x86_64、 IA64、 POWER™)	ext2、ext3、reiserfs	DIO	NO FILE SYSTEM CACHING
Linux ディストリビューション SLES 9+ および RHEL 4+ (対応アーキテクチャー: x86、x86_64、 IA64、 POWER)	VERITAS Storage Foundation 4.1 (VxFS)	CIO	NO FILE SYSTEM CACHING
Linux ディストリビューション SLES 9+ および RHEL 4+ (対応アーキテクチャー: zSeries®)	ファイバー・チャネル・プロトコル (FCP) を使用する SCSI (Small Computer System Interface) ディスク上の ext2、ext3、または reiserfs	DIO	FILE SYSTEM CACHING

表 45. ファイル・システム・キャッシングを使用しない表スペースのサポートされる構成 (続き)

プラットフォーム	ファイル・システムのタイプおよび必要な最小レベル	NO FILE SYSTEM CACHING が指定されているときに、データベース・マネージャーがサブミットした DIO 要求と CIO 要求	NO FILE SYSTEM CACHING と FILE SYSTEM CACHING のどちらも指定されない場合のデフォルトの動作
Windows	特定の要件なし。DB2 がサポートするすべてのファイル・システムを処理可能	DIO	NO FILE SYSTEM CACHING

注:

1. AIX JFS では、FILE SYSTEM CACHING がデフォルトです。
2. Solaris UFS では、FILE SYSTEM CACHING がデフォルトです。
3. データベース・マネージャー対応 VERITAS Storage Foundation では、オペレーティング・システムの前提条件が異なることがあります。上記にリストしたプラットフォームは、現行のリリースでサポートされているプラットフォームです。前提条件に関する情報については、VERITAS Storage Foundation for DB2 のサポートにご相談ください。
4. 上記の最小レベルの代わりに SFDB2 5.0 を使用する場合は、SFDB2 5.0 MP1 RP1 リリースを使用する必要があります。このリリースには、5.0 バージョンに固有のフィックスが含まれています。
5. データベース・マネージャーでデフォルト設定に対して NO FILE SYSTEM CACHING を選択したくない場合は、関連する SQL、コマンド、または API で FILE SYSTEM CACHING を指定してください。

例

例 1: デフォルトでは、この新規表スペースはバッファリングのない I/O を使用して作成されます。NO FILE SYSTEM CACHING 節が暗黙指定されています。

```
CREATE TABLESPACE table space name ...
```

例 2: 以下のステートメントの NO FILE SYSTEM CACHING 節は、この特定の表スペースではファイル・システム・レベルのキャッシングが OFF になることを示しています。

```
CREATE TABLESPACE table space name ... NO FILE SYSTEM CACHING
```

例 3: 以下のステートメントは、既存の表スペースについてファイル・システム・レベルのキャッシングを使用不可にします。

```
ALTER TABLESPACE table space name ... NO FILE SYSTEM CACHING
```

例 4: 以下のステートメントは、既存の表スペースについてファイル・システム・レベルのキャッシングを使用可能にします。

```
ALTER TABLESPACE table space name ... FILE SYSTEM CACHING
```

表スペースのエクステント・サイズ

表スペースのエクステント・サイズは、次のコンテナーに書き込まれるまでに 1 つのコンテナーに書き込まれる表データのページ数を表します。

エクステント・サイズを選択するときには、次のことを考慮してください。

- 表スペースの中の表のサイズとタイプ。

  DMS 表スペースの中のスペースは、 1 つの表に対して一度に 1 エクステントずつ割り当てられます。表に入力が行われ、エクステントがいっぱいになると、新しいエクステントが割り当てられます。 DMS 表スペース・コンテナー・ストレージは事前予約済みです。つまり、コンテナーを使い切るまで新しいエクステントが割り振られます。

  SMS 表スペースの中のスペースは、 1 つの表に対して一度に 1 エクステントずつ、または一度に 1 ページずつ割り振られます。表にデータが入れられ、エクステントまたはページがいっぱいになると、ファイル・システム内のすべてのエクステントまたはページが使用されるまで、新しいエクステントまたはページが割り振られます。 SMS 表スペースを使用している場合は、複数ページ・ファイル割り振りが可能です。複数ページ・ファイル割り振りの場合、一度に 1 ページずつではなく複数のエクステントを割り振ることができます。

  マルチページ・ファイル割り振りは、デフォルトで使用可能です。 *multipage_alloc* データベース構成パラメーターの値は、マルチページ・ファイル割り振りが使用可能かどうかを示します。

  注: マルチページ・ファイル割り振りは、TEMPORARY 表スペースには適用されません。

  1 つの表は、以下の別個の表オブジェクトからなります。
  - 1 つのデータ・オブジェクト。これは、正規の列データが保管される場所です。
  - 1 つの索引オブジェクト。表に定義されたすべての索引がここに保管されます。
  - 1 つのロング・フィールド (LF) データ・オブジェクト。表に 1 つまたは複数の LONG フィールド・データがあれば、それらの列はここに保管されます。
  - 2 つのラージ・オブジェクト (LOB) データ・オブジェクト。表に 1 つまたは複数の LOB 列がある場合、それらの列が以下の 2 つの表オブジェクトに保管されます。
    - LOB データ用の 1 つの表オブジェクト。
    - LOB データを記述するメタデータ用の 2 番目の表オブジェクト。
  - マルチディメンション・クラスタリング (MDC) 表のブロック・マップ・オブジェクト。
  - XML 文書を保管する別の XDA オブジェクト。

  それぞれの表オブジェクトは別々に保管され、それぞれが必要に応じて新しいエクステントを割り当てます。また、それぞれの DMS 表オブジェクトは、 (その表オブジェクトに属する表スペース内のすべてのエクステントを記述する) エクステント・マップというメタデータ・オブジェクトとも関連付けられます。エク

ステント・マップ用のスペースも一度に 1 エクステントずつ割り当てられます。したがって、DMS 表スペース内のオブジェクトに対するスペースの初期割り振りは、エクステント 2 個になります。 (SMS 表スペース内のオブジェクトのためのスペースの初期割り振りは 1 ページです。)

1 つの DMS 表スペース内に多くの小さな表がある場合は、比較的少ないデータ量を保管するのに、比較的大きな量のスペースを割り振ることになります。このような場合には、小さなエクステント・サイズを指定する必要があります。一方で、急速に大きくなっていく大きな表に対して小さなエクステント・サイズの DMS 表スペースを使用するなら、追加のエクステントの頻繁な割り振りに伴う不必要なオーバーヘッドが生じる可能性があります。

- 表に対するアクセスの種類。

  表へのアクセスに、大量のデータを処理する照会やトランザクションが数多く使用される場合には、表データのプリフェッチ処理を行うことにより、パフォーマンスの大幅な向上が期待できます。

- エクステントの必要最小数。

  表スペース用の 5 個のエクステントが入るスペースがコンテナーの中になければ、表スペースは作成されません。

表スペースのページ・サイズ

表スペースを設計する際は、ページ・サイズを考慮する必要があります。

ページ・サイズの制限には、4 KB、8 KB、16 KB、または 32 KB が使用できます。これらページ・サイズはそれぞれに、各表スペース・タイプについての最大が定められており、それを守らなければなりません。

表 46 に、各種の表スペース・タイプに対するページ・サイズごとの制限を示します。

表 46. 表スペースのページ・サイズごとの制限

表スペース・タイプ (ギガバイト単位)	**4K ページ・サイズ制限**	**8K ページ・サイズ制限**	**16K ページ・サイズ制限**	**32K ページ・サイズ制限**
SMS 表スペース	64	128	256	512
DMS 表スペース (通常)	64	128	256	512
DMS 表スペース (大)	2048	4096	8192	16 384
自動ストレージの表スペース (通常)	64	128	256	512
自動ストレージの表スペース (大)	2048	4096	8192	16 384
TEMPORARY 表スペース	64	128	256	512

各種表スペースでのデータベースおよび索引のページ・サイズ制限については、SQL および XML の制限値でデータベース・マネージャーのページ・サイズごとの制限を参照してください。

SYSTEM TEMPORARY 表スペースの最大ページ・サイズが照会または位置指定 UPDATE を行うのに十分な大きさであるようにする方法は、「*マイグレーション・ガイド*」の『SYSTEM TEMPORARY 表スペース・ページ・サイズが要件を満たしていることの確認』を参照してください。

表スペースのディスク入出力

表スペースのタイプと設計によって、その表スペースに対して実行される入出力の効率が決まります。

表スペースの設計と使用に関する他の問題について考慮するときは、まず以下の概念を理解しておく必要があります。

大ブロック読み取り
単一の要求で複数ページ (通常 1 エクステント) を検索する読み取り。一度に複数ページを読み取ることは、それぞれのページを別個に読み取るより効率的です。

プリフェッチ
照会でページが参照されるのに先立って行うページの読み取り。全体的な目標は、応答時間を削減することです。ページのプリフェッチを照会の実行と非同期に行うことができれば、この目標を達成することができます。最適な応答時間は、 CPU または入出力サブシステムのいずれかが最大容量で操作されている場合に達成されます。

ページ・クリーニング
ページの読み取りおよび変更が行われると、これらのページはデータベース・バッファー・プールの中に累積されます。ページが読み込まれるときには、バッファー・プール・ページの中に読み込まれます。バッファー・プールが変更されたページでいっぱいである場合は、それらの変更されたページのいずれかをディスクに書き出さないと、新しいページを読み込むことができません。バッファー・プールがいっぱいになるのを防ぐために、ページ・クリーナー・エージェントは変更されたページを書き出して、読み取り要求の際にバッファー・プール・ページが利用できる状態にします。

データベース・マネージャーは、大ブロック読み取りが効率的であるときには常にこれを行います。通常これは、順次または部分的に順次であるデータを検索する場合に行われます。 1 回の読み取り操作で読み取られるデータの量はエクステント・サイズによって決まり、エクステント・サイズが大きければ大きいほど、1 回に読み取られるページ数が多くなります。

ディスクから、バッファー・プール内の連続ページにページを読み取ることができる場合、順次プリフェッチ・パフォーマンスをさらに向上させることができます。デフォルトでは、バッファー・プールはページ・ベースであるため、ディスクから連続ページに読み取るときに連続ページのセットを検出できるという保証はありません。ブロック・ベースのバッファー・プールは、ページ域だけでなく、連続ページのセット用のブロック域も含まれているため、この目的のために、ブロック・ベースのバッファー・プールを使用することができます。連続ページセットそれぞれ

をブロックといい、それぞれのブロックに含まれるページ数をブロック・サイズといいます。それぞれのブロック内のページ数だけでなく、ページ域およびブロック域のサイズが構成可能です。

エクステントがディスク上に格納される方法は、入出力の効率に影響を与えます。デバイス・コンテナーを使用する DMS 表スペースの場合、データはディスク上で連続する傾向にあり、最小のシーク時間とディスク待ち時間で読み取ることができます。しかし、ファイルを使用する場合は、DMS 表スペースで使用するために事前割り振りされた大きなファイルも、特にクリーンなファイル・スペースにファイルが割り振られた場合には、ディスク上で連続しやすくなります。しかし、データがファイル・システムによって分割され、ディスク上の複数のロケーションに保管される可能性があります。これは、SMS 表スペースを使用し、ファイルが一度に 1 ページずつ拡張され、フラグメント化が起こりやすい場合に最もよく発生します。

CREATE TABLESPACE または ALTER TABLESPACE ステートメント上の PREFETCHSIZE オプションを変更することによって、プリフェッチの程度を制御することができます。あるいは、プリフェッチ・サイズを AUTOMATIC に設定して、データベース・マネージャーが使用に最適なサイズを自動的に選択するようにすることもできます。 (データベース内のすべての表スペースのデフォルト値は、データベース構成パラメーター *dft_prefetch_sz* によって設定されます。) PREFETCHSIZE パラメーターは、プリフェッチが起動されたときに、どれだけのページ数を読み取るかをデータベース・マネージャーに指示します。 PREFETCHSIZE を CREATE TABLESPACE ステートメントの EXTENTSIZE パラメーターの倍数に設定すると、複数のエクステントを並列して読み取らせることができます。 (データベース内のすべての表スペースのデフォルト値は、データベース構成パラメーター *dft_extent_sz* によって設定されます。) EXTENTSIZE パラメーターは、次のコンテナーにスキップするまでに、あるコンテナーに書き込まれる 4 KB ページの数を指定します。

例えば、3 つの装置を使用した表スペースがあるとします。 PREFETCHSIZE が EXTENTSIZE の 3 倍になるよう設定すれば、データベース・マネージャーは各装置から大ブロックを並列して読み取ることができ、それによって入出力のスループットがかなり増加します。なお、ここでは、各装置が別々の物理装置であり、コントローラーが各装置からのデータ・ストリームを処理するために十分な処理能力を持っていることを想定しています。データベース・マネージャーは、照会速度、バッファー・プール使用率、およびその他の要因に基づいて、実行時にプリフェッチ・パラメーターを動的に調整しなければならなくなる場合があることに注意してください。

一部のファイル・システム (AIX のジャーナル・ファイル・システムなど) は、独自のプリフェッチ方式を使用します。場合によっては、データベース・マネージャー・プリフェッチよりもファイル・システムのプリフェッチの方が大きな値に設定されます。この結果、ファイル・コンテナーを使用する SMS および DMS 表スペースのプリフェッチの方が、装置を使用する DMS 表スペースのプリフェッチよりも速く実行されるように見えることがあります。しかしこれは、おそらくファイル・システム内で付加的なレベルでのプリフェッチが行われているためにそう見えているに過ぎません。 DMS 表スペースは、同等のどの構成よりも速く実行できるはずです。

プリフェッチまたは均一な読み取りが効率的に行われるようにするために、十分な数のクリーン・バッファー・プール・ページが存在しなければなりません。例えば、表スペースから 3 エクステントを読み取り、読み取られる各ページごとにバッファー・プールから変更ページを書き出さなければならない並列プリフェッチ要求が存在するとします。この並行プリフェッチ要求の効率が下がって、照会に対応できなくなる可能性があります。ページ・クリーナーは、プリフェッチ要求を満たす十分な数で構成されている必要があります。

初期の表スペースの定義

データベースが作成されるときに、次の 3 つの表スペースが定義されます。(1) システム・カタログ表に対する SYSCATSPACE、(2) データベースの処理中に作成されたシステム一時表に対する TEMPSPACE1、(3) ユーザー定義の表および索引に対する USERSPACE1

注: データベースを初めて作成する時点では、 USER TEMPORARY 表スペースは作成されません。

他で指定されていない限り、3 つのデフォルトの表スペースは自動ストレージによって管理されます。

CREATE DATABASE コマンドを使用して、デフォルト・バッファー・プールおよび初期表スペースのページ・サイズを指定できます。このデフォルトは、将来の CREATE BUFFERPOOL ステートメントおよび CREATE TABLESPACE ステートメントすべてのデフォルト・ページ・サイズにもなります。データベースの作成時にページ・サイズを指定しない場合、デフォルト・ページ・サイズは 4 KB です。

コマンド行を使用して初期表スペースを定義するには、以下のように入力します。

```
CREATE DATABASE <name>
  CATALOG TABLESPACE
    MANAGED BY SYSTEM USING ('<path>')
    EXTENTSIZE <value> PREFETCHSIZE <value>
  USER TABLESPACE
    MANAGED BY DATABASE USING (FILE'<path>' 5000,
                               FILE'<path>' 5000)
    EXTENTSIZE <value> PREFETCHSIZE <value>
  TEMPORARY TABLESPACE
    MANAGED BY SYSTEM USING ('<path>')
  WITH "<comment>"
```

表スペースにデフォルト定義を使用しない場合は、 CREATE DATABASE コマンドでこれらの特性を指定することができます。例えば、次のコマンドは Windows 上にデータベースを作成するために使用するものです。

```
CREATE DATABASE PERSONL
  CATALOG TABLESPACE
    MANAGED BY SYSTEM USING ('d:¥pcatalog','e:¥pcatalog')
    EXTENTSIZE 16 PREFETCHSIZE 32
  USER TABLESPACE
    MANAGED BY DATABASE USING (FILE'd:¥db2data¥personl' 5000,
                               FILE'd:¥db2data¥personl' 5000)
    EXTENTSIZE 32 PREFETCHSIZE 64
  TEMPORARY TABLESPACE
    MANAGED BY SYSTEM USING ('f:¥db2temp¥personl')
  WITH "Personnel DB for BSchiefer Co"
```

この例では、最初の表スペースのそれぞれの定義が明示的に提供されています。デフォルト定義を使用したくない表スペースの表スペース定義を指定するだけで済みます。

注: パーティション・データベース環境で作業している場合、特定のデータベース・パーティションに対して、コンテナーを作成したり割り当てたりすることはできません。まずデフォルトのユーザー表スペースおよび、SYSTEM TEMPORARY 表スペースで、データベースを作成する必要があります。次に、CREATE TABLESPACE ステートメントを使用して、必要な表スペースを作成できます。最後に、デフォルトの表スペースをドロップします。

CREATE DATABASE コマンドの MANAGED BY 句のコーディングは、 CREATE TABLESPACE ステートメントの MANAGED BY 句と同じ形式になります。

必要に応じて、さらにユーザー表スペースと TEMPORARY 表スペースを余分に追加することができます。カタログ表スペース SYSCATSPACE をドロップしたり、もう 1 つ作成したりすることはできません。また、ページ・サイズが 4 KB の少なくとも 1 つの SYSTEM TEMPORARY 表スペースが常に必要です。その他の SYSTEM TEMPORARY 表スペースは作成できます。表スペースのページ・サイズやエクステント・サイズを、作成後に変更することもできません。

DMS 直接ディスク・アクセス・デバイスのアタッチ

データを保管するためにコンテナーを処理している場合、データベース・マネージャーは、直接ディスク・アクセス (ロー I/O) をサポートしています。

このタイプのサポートにより、直接ディスク・アクセス (ロー) デバイスを DB2 データベース・システムにアタッチすることができます。

表スペース作成時に参照する、コンテナーのデバイス名またはファイル名を知っておくことが必要です。表スペースに割り振られる各デバイスまたはファイル名と関連するスペースの量も知っておくべきです。コンテナーに対する読み取りおよび書き込みの正しい許可が必要です。

直接ディスク・アクセスを識別するための物理的また論理的方法は、オペレーティング・システムによって異なります。

- Windows オペレーティング・システムの場合:

  物理ハード・ディスクを指定するには、次の構文を使用します。

  ¥¥.¥PhysicalDriveN

  ここで N は、システムにある物理ドライブのいずれかを表します。この場合、N を 0、1、2、または他の正の整数に置き換えることができます。

  ¥¥.¥PhysicalDrive5

  論理ドライブ、つまり未フォーマットのデータベース・パーティションを指定するには、次の構文を使用します。

  ¥¥.¥N:

ここで N: は、システムにある論理ドライブ名を表します。例えば、N: を E: または他のドライブ名で置き換えることができます。ドライブを識別するための文字の使用によって課せられる制限をなくすには、論理ドライブでグローバル・ユニーク ID (GUID) を使用できます。

Windows の場合、DMS ロー・デバイス表スペース・コンテナーの指定の方式は新しくなっています。ボリューム (つまり基本ディスク・データベース・パーティションまたは動的ボリューム) には、その作成時にグローバル・ユニーク ID (GUID) が割り当てられます。 GUID は、表スペース定義でコンテナーを指定する際に、デバイス ID として使用できます。 GUID はすべてのシステムで固有です。つまり、複数パーティション・データベースでは、ディスク・パーティション定義が同じであっても、GUID は各データベース・パーティションで異なるということです。

Windows システムで定義されるすべてのディスク・ボリュームで GUID を表示しやすくするため、 *db2listvolumes.exe* というツールを使用できます (Windows オペレーティング・システムのみ)。このツールは、ツールが実行する現行ディレクトリーで 2 つのファイルを作成します。 1 つは `volumes.xml` というファイルで、XML が使用できるブラウザーで、 XML でエンコードされた各ディスク・ボリュームを簡単に表示するための情報が入っています。 2 つ目のファイルは `tablespace.ddl` で、表スペース・コンテナーを指定するために必要な構文が入っています。表スペース定義に必要な残りの情報が入るように、このファイルを更新する必要があります。 db2listvolumes コマンドは、コマンド行引数を必要としません。

- Linux および UNIX プラットフォームでは、論理ボリュームを利用すると、実際には不連続の物理データベース・パーティション上、さらには複数の物理ボリューム上に存在しているディスク領域を、ユーザーおよびアプリケーションからは、単一で連続する拡張可能なディスク・ボリュームのように見せることができます。また、論理ボリュームは、単一のボリューム・グループ内に入っていなければなりません。 1 つのボリューム・グループにつき 256 の論理ボリュームという制限があります。 1 つのボリューム・グループあたりの物理ボリュームの制限は 32 です。 mklv コマンドを使用して、追加の論理ボリュームを作成することができます。このコマンドにより、論理ボリュームの名前を指定し、その論理ボリュームのために割り振られる論理パーティションの数やロケーションを含む、論理ボリュームの特性を定義することができます。

  論理ボリュームを作成した後で、chlv コマンドでその名前や特性を変更することができ、 extendlv でその論理ボリュームに割り振られている論理パーティションの数を増やすことができます。さらに大きい数が指定されていない限り、作成の時点での論理ボリュームのデフォルトの最大サイズは 512 論理パーティションです。この制限をオーバーライドするには、chlv コマンドを使用します。

  AIX では、論理ボリューム・ストレージの確立と制御を可能にするオペレーティング・システム・コマンドのセット、ライブラリー・サブルーチン、および他のツールは、論理ボリューム・マネージャー (LVM) と呼ばれています。LVM は、ストレージ・スペースのより単純で柔軟な論理ビューと実際の物理的なディスクとの間のデータをマップすることにより、ディスク・リソースを制御します。

mklv および他の論理ボリューム・コマンド、また LVM の詳細については、「*AIX 5L バージョン 5.2 システム・マネージメント・コンセプト:オペレーティング・システムおよびデバイス*」を参照してください。

DMS ダイレクト・ディスク・アクセスの構成とセットアップ (Linux)

データを保管するためにコンテナーを処理している場合、データベース・マネージャーはブロック・デバイス・インターフェース (つまりロー I/O) を使用した直接ディスク (ロー) アクセスをサポートしています。

Linux でロー I/O をセットアップする前に、1 つ以上の空き IDE または SCSI ディスク・データベース・パーティションが必要です。表スペースの作成時にディスク・パーティションを参照するには、ディスク・パーティションの名前、および表スペースに割り振られるディスク・パーティションに関連するスペースの量を知っておくべきです。

Linux 環境で作業するときには、以下の情報を使用してください。 Linux/390 では、データベース・マネージャーはダイレクト・ディスク・アクセス装置をサポートしていません。

Linux 上でロー I/O を構成するには、次のようにします。

この例では、使用されるロー・データベース区分は /dev/sda5 です。この区分には、重要なデータは含まれていないと想定します。

1. このデータベース・パーティションの 4096 バイト・ページの数を計算します。端数が出た場合は切り捨てます。例:

```
# fdisk /dev/sda
Command (m for help): p

Disk /dev/sda: 255 heads, 63 sectors, 1106 cylinders
Units = cylinders of 16065 * 512 bytes
```

表 47. Linux ロー I/O 計算

デバイス・ブート	開始	終了	ブロック	ID	システム
/dev/sda1	1	523	4200997	83	Linux
/dev/sda2	524	1106	4682947+	5	Extended
/dev/sda5	524	1106	4682947	83	Linux

```
Command (m for help): q
#
```

/dev/sda5 のページ数は、次のようになります。

```
num_pages = floor( (4682947 * 1024)/4096 )
num_pages = 1170736
```

2. ディスク・パーティション名を指定して、表スペースを作成します。例:

```
CREATE TABLESPACE dms1
MANAGED BY DATABASE
USING (DEVICE '/dev/sda5' 1170736)
```

3. 接合ポイント (またはボリューム・マウント・ポイント) を使用して論理パーティションを指定するには、RAW パーティションを別の NTFS フォーマットのボリュームに接合ポイントとしてマウントし、次いで NTFS ボリューム上の接合ポイントへのパスをコンテナー・パスとして指定します。例:

```
CREATE TABLESPACE TS4
   MANAGED BY DATABASE USING (DEVICE 'C:¥JUNCTION¥DISK_1' 10000,
      DEVICE 'C:¥JUNCTION¥DISK_2' 10000)
```

データベース・マネージャーはまずパーティションを照会して、それにファイル・システムがあるかどうかを確認します。存在する場合には、パーティションは RAW デバイスとして扱われず、データベース・マネージャーは通常のファイル・システム入出力操作をそのパーティションに対して実行します。

ロー・デバイスの表スペースは、データベース・マネージャーでサポートされているその他すべてのページ・サイズでもサポートされています。

バージョン 9 より前の Linux 版では、ロー・コントローラー・ユーティリティーを使用した直接ディスク・アクセスが使用されていました。この方法は現在は推奨されておらず、使用すべきではありません。データベース・マネージャーでは、Linux オペレーティング・システムがこの方式をサポートする場合には、この方式を使用することができますが、db2diag.log には、推奨されないことを示すメッセージが表示されます。

以前の方式では、ディスク・パーティションをロー・コントローラーに「バインド」し、CREATE TABLESPACE コマンドを使用してそのロー・コントローラーをデータベース・マネージャーに指定することが必要でした。

```
CREATE TABLESPACE dms1
MANAGED BY DATABASE
USING (DEVICE '/dev/raw/raw1' 1170736)
```

表スペースの作成

非自動ストレージ表スペースの場合、表スペース作成時に参照する、コンテナーのデバイス名またはファイル名を知っておくことが必要です。

表スペースに割り振られる各デバイスまたはファイル名と関連する空き容量も知っておくべきです。自動ストレージ表スペースの場合、データベース・マネージャーはデータベースに関連付けられたストレージ・パスに基づいて、表スペースにコンテナーを割り当てます。

表スペースは、データベース・システムによって使用される物理的なストレージ・デバイスと、データの保管に使用される論理的なコンテナーまたは表とを関連付けます。

データベースの中に表スペースを作成すると、その表スペースにコンテナーが割り当てられ、その定義と属性がデータベース・システム・カタログに記録されます。このようにして、その表スペースに表を作成できるようになります。表スペース作成時に参照する、コンテナーのデバイス名またはファイル名を知っておくことが必要です。表スペースに割り振られる各デバイスまたはファイル名と関連する空き容量も知っておくべきです。

データベースの作成時には、3 つの初期表スペースが作成されます。 3 つの初期表スペースのページ・サイズは、CREATE DATABASE コマンドの使用時に設定または受け入れられたデフォルトに基づきます。このデフォルトは、将来の CREATE BUFFERPOOL ステートメントおよび CREATE TABLESPACE ステートメントすべてのデフォルト・ページ・サイズにもなります。データベースの作成時にページ・サイズを指定しない場合、デフォルト・ページ・サイズは 4 KB です。表スペースの作成時にページ・サイズを指定しない場合、デフォルト・ページ・サイズはデータベースの作成時に設定されたサイズになります。

コマンド行を使用して SMS 表スペースを作成するには、以下のように入力します。

```
CREATE TABLESPACE <NAME>
  MANAGED BY SYSTEM
  USING ('<path>')
```

コマンド行を使用して DMS 表スペースを作成するには、以下のように入力します。

```
CREATE TABLESPACE <NAME>
  MANAGED BY DATABASE
  USING (FILE'<path>' <size>)
```

コマンド行を使用して自動ストレージ表スペースを作成するには、以下のいずれかのステートメントを入力します。

```
CREATE TABLESPACE <NAME>
```

```
CREATE TABLESPACE <NAME>
  MANAGED BY AUTOMATIC STORAGE
```

次の SQL ステートメントは、別個の 3 つのドライブの 3 つのディレクトリーを使用して、 Windows 上で SMS 表スペースを作成するものです。

```
CREATE TABLESPACE RESOURCE
  MANAGED BY SYSTEM
  USING ('d:¥acc_tbsp', 'e:¥acc_tbsp', 'f:¥acc_tbsp')
```

次の SQL ステートメントは、それぞれ 5,000 ページの 2 つのファイル・コンテナーを使用して、 DMS 表スペースを作成します。

```
CREATE TABLESPACE RESOURCE
  MANAGED BY DATABASE
  USING (FILE'd:¥db2data¥acc_tbsp' 5000,
         FILE'e:¥db2data¥acc_tbsp' 5000)
```

上記の 2 つの例では、コンテナーに明示的な名前が提供されました。しかし、相対コンテナー名を指定する場合、コンテナーはデータベース・パス配下に作成されたサブディレクトリーの中に作成されます。

表スペース・コンテナーを作成するとき、存在しないディレクトリー・レベルがあればデータベース・マネージャーによって作成されます。例えば、コンテナーの指定が /project/user_data/container1 であり、ディレクトリー /project が存在しない場合、データベース・マネージャーはディレクトリーの /project および /project/user_data を作成します。

データベース・マネージャーが作成するディレクトリーは、すべて PERMISSION 700 付きで作成されます。つまり、インスタンスの所有者だけに読み取り、書き込

み、および実行のアクセス権があります。このアクセス権を持つのはインスタンスの所有者のみであるため、複数のインスタンスを作成している間は以下のシナリオが発生する可能性があります。

- 上記の例と同じディレクトリー構造を使用して、ディレクトリー・レベル /project/user_data が存在しないと仮定します。
- user1 は user1 という名前のインスタンスをデフォルトで作成して、データベースを作成した後、/project/user_data/container1 をコンテナーの 1 つとする表スペースを作成します。
- user2 は user2 という名前のインスタンスをデフォルトで作成して、データベースを作成した後、/project/user_data/container2 をコンテナーの 1 つとする表スペースを作成しようとします。

最初の要求によりデータベース・マネージャーはディレクトリー・レベル /project/user_data を PERMISSION 700 で作成したので、user2 にはこれらのディレクトリー・レベルに対するアクセス権がないので、それらのディレクトリー内に container2 を作成できません。この場合、CREATE TABLESPACE 操作は失敗します。

この競合を解決するには、次の 2 つの方法があります。

1. 表スペースを作成する前に、ディレクトリー /project/user_data を作成して、user1 と user2 の両方が表スペースを作成するために必要なアクセスに関する許可を設定します。表スペース・ディレクトリーのすべてのレベルが存在する場合、データベース・マネージャーはアクセス権を変更しません。
2. user1 が /project/user_data/container1 を作成した後で、 /project/user_data に関する許可を、user2 が表スペースを作成するために必要なアクセス権に設定します。

サブディレクトリーがデータベース・マネージャーによって作成される場合は、表スペースがドロップされると、そのサブディレクトリーもデータベース・マネージャーによって削除される場合があります。

このシナリオでは、表スペースが特定のデータベース・パーティション・グループに関連付けられていないことを前提としています。デフォルトのデータベース・パーティション・グループである IBMDEFAULTGROUP は、以下のパラメーターがステートメント内に指定されていない場合に使用されます。

```
IN database_partition_group_name
```

以下の SQL ステートメントは、それぞれ 10 000 ページの 3 つの論理ボリュームを使用して AIX システム上に DMS 表スペースを作成し、それらの入出力特性を指定します。

```
CREATE TABLESPACE RESOURCE
  MANAGED BY DATABASE
  USING (DEVICE '/dev/rdblv6' 10000,
         DEVICE '/dev/rdblv7' 10000,
         DEVICE '/dev/rdblv8' 10000)
  OVERHEAD 7.5
  TRANSFERRATE 0.06
```

この SQL ステートメントで指定している UNIX デバイスは、すでに存在しているものであり、インスタンス所有者および SYSADM グループが書き込みを行えるようでなければなりません。

以下の例は、UNIX 複数パーティション・データベース内の ODDGROUP と呼ばれるデータベース・パーティション・グループ上に DMS 表スペースを作成します。ODDGROUP は、 CREATE DATABASE PARTITION GROUP ステートメントを使用してあらかじめ作成されていなければなりません。この場合、ODDGROUP データベース・パーティション・グループは、1、3、および 5 の番号が付いたデータベース・パーティションから成っていると想定されています。すべてのデータベース・パーティション上で、10 000 の 4 KB ページについて、デバイス /dev/hdisk0 を使用します。さらに、40 000 の 4 KB ページから成るデータベース・パーティションごとに 1 つのデバイスを宣言します。

```
CREATE TABLESPACE PLANS IN ODDGROUP
  MANAGED BY DATABASE
  USING (DEVICE '/dev/HDISK0' 10000, DEVICE '/dev/n1hd01' 40000)
        ON DBPARTITIONNUM 1
       (DEVICE '/dev/HDISK0' 10000, DEVICE '/dev/n3hd03' 40000)
        ON DBPARTITIONNUM 3
       (DEVICE '/dev/HDISK0' 10000, DEVICE '/dev/n5hd05' 40000)
        ON DBPARTITIONNUM 5
```

データベース・マネージャーで順次プリフェッチ機能を使用すると、並列入出力が使われ、順次入出力のパフォーマンスを大幅に向上させることができます。

デフォルトの 4 KB サイズより大きいページ・サイズを使用する表スペースを作成することもできます。次の SQL ステートメントは、Linux および UNIX システム上でページ・サイズが 8 KB の SMS 表スペースを作成するものです。

```
CREATE TABLESPACE SMS8K
  PAGESIZE 8192
  MANAGED BY SYSTEM
  USING ('FSMS_8K_1')
  BUFFERPOOL BUFFPOOL8K
```

関連したバッファー・プールのページ・サイズも、同じ 8 KB でなければならないことに注意してください。

作成された表スペースは、それが参照するバッファー・プールが活動化されるまで使用できません。

ALTER TABLESPACE ステートメントを使うと、 DMS 表スペースでコンテナーを追加、ドロップ、またはサイズ変更したり、表スペースの PREFETCHSIZE、OVERHEAD、および TRANSFERRATE の設定値を変更したりすることができます。表スペース・ステートメントを発行するトランザクションは、システム・カタログの競合を避けるために、ALTER TABLESPACE SQL ステートメントの後できるだけ早くコミットする必要があります。

注: PREFETCHSIZE の値は、EXTENTSIZE の値の倍数でなければなりません。例えば、EXTENTSIZE が 10 なら PREFETCHSIZE は 20 や 30 などにする必要があります。表スペースを作成するときには、以下の式を使用してプリフェッチ・サイズを手動で作成してください。

```
プリフェッチ・サイズ = (コンテナーの数) X (コンテナーごとの物理スピンドルの数) X エクステント・サイズ
```

PREFETCHSIZE を AUTOMATIC に設定することによって、データベース・マネージャーがプリフェッチ・サイズを自動的に決めるようにすることも検討してください。

直接 I/O (DIO) は、ファイル・システム・レベルでのキャッシングをバイパスするので、メモリーのパフォーマンスを改善します。この処理は CPU オーバーヘッドを削減して、データベース・インスタンスがより多くのメモリーを使用できるようにします。

並行 I/O (CIO) は DIO の利点を含むと共に、書き込みアクセスのシリアライゼーションを軽減します。

AIX では DIO および CIO がサポートされます。HP-UX、Solaris、Linux、および Windows オペレーティング・システムでは DIO（のみ）がサポートされます。

キーワード NO FILE SYSTEM CACHING および FILE SYSTEM CACHING は、各表スペースで DIO または CIO のどちらを使用するかを指定するための、CREATE および ALTER TABLESPACE の SQL ステートメントの一部です。 NO FILE SYSTEM CACHING が有効なとき、データベース・マネージャーは可能な場合に常に並行 I/O (CIO) の使用を試みます。 CIO がサポートされていない場合 (例えば、JFS が使用されている場合)、DIO が代わりに使用されます。

CREATE TABLESPACE ステートメントを実行すると、ドロップされた表のリカバリー・フィーチャーがデフォルトでオンになります。このフィーチャーにより、表スペース・レベルのリストアおよびロールフォワード操作を使用して、ドロップされた表データをリカバリーすることができます。これはデータベース・レベルのリカバリーより速く、ユーザーもデータベースをそのまま使用できるため、有用です。

しかし、ドロップされた表のリカバリー・フィーチャーは、リカバリー対象のドロップ表操作が多数ある場合、または履歴ファイルが非常に大規模である場合、順方向リカバリーのパフォーマンスに影響を与える可能性があります。

非常に多くのドロップ表操作を実行することを計画している場合は、このフィーチャーを無効にして、循環ロギングを使用するか、またはドロップされた表のリカバリーを実行しないようにします。このフィーチャーを無効にするには、CREATE TABLESPACE ステートメントの発行時に、DROPPED TABLE RECOVERY オプションを明示的に OFF に設定することができます。代替方法として、ALTER TABLESPACE ステートメントを使用して、既存の表スペースのドロップされた表のリカバリー・フィーチャーをオフにすることができます。

SYSTEM TEMPORARY 表スペースの作成

SYSTEM TEMPORARY 表スペースは、システム一時表を格納するのに使用されます。システム一時表を格納できるのは SYSTEM TEMPORARY 表スペースだけなので、データベースは常に少なくとも 1 つの SYSTEM TEMPORARY 表スペースを持っていなければなりません。

データベースを作成すると、定義される 3 つのデフォルト表スペースのうち 1 つが、 ″TEMPSPACE1″ という SYSTEM TEMPORARY 表スペースになります。

別の SYSTEM TEMPORARY 表スペースを作成するには、CREATE TABLESPACE ステートメントを使用します。例:

```
CREATE SYSTEM TEMPORARY TABLESPACE tmp_tbsp
  MANAGED BY SYSTEM
  USING ('d:¥tmp_tbsp','e:¥tmp_tbsp')
```

各ページ・サイズの表スペースが少なくとも 1 つ必要です。

SYSTEM TEMPORARY 表スペースの作成時に指定できるデータベース・パーティション・グループは、 IBMTEMPGROUP だけです。

USER TEMPORARY 表スペースの作成

USER TEMPORARY 表スペースは、デフォルトではデータベース作成の時点で作成されません。アプリケーション・プログラムが一時表を使用する必要がある場合、一時表が置かれる USER TEMPORARY 表スペースを作成する必要があります。

REGULAR 表スペースと同様、USER TEMPORARY 表スペースは、IBMTEMPGROUP 以外の任意のデータベース・パーティション・グループに作成できます。 IBMDEFAULTGROUP は、ユーザー一時表の作成時に使用されるデフォルトのデータベース・パーティション・グループです。

DECLARE GLOBAL TEMPORARY TABLE ステートメントは、宣言された一時表を、USER TEMPORARY 表スペース内で使用できるように定義します。

ユーザーの TEMPORARY 表スペースを作成するには、CREATE TABLESPACE ステートメントを使用します。

```
CREATE USER TEMPORARY TABLESPACE usr_tbsp
  MANAGED BY DATABASE
  USING (FILE 'd:¥db2data¥user_tbsp' 5000,
  FILE 'e:¥db2data¥user_tbsp' 5000)
```

表スペースの変更

コマンド行を使用して表スペースを変更するには、ALTER TABLESPACE ステートメントを使用します。

SMS、DMS、および自動ストレージ・コンテナーを変更することができます。また、表スペースの名前変更およびオフライン・モードからオンライン・モードへの切り換えもできます。

SMS 表スペースの変更

SMS 表スペースの場合、単一パーティション・データベースに追加できるコンテナーは 1 つのみ、パーティション・データベースに追加できるコンテナーは 1 つ以上です。

これを行うプロセスは、218 ページの『DMS コンテナーの追加または拡張』に説明されているものと同じです。

DMS 表スペースの変更

DMS 表スペースの場合、コンテナーの追加、拡張、リバランス、サイズ変更、ドロップ、または削減を行うことができます。

DMS コンテナーの追加または拡張

DMS 表スペース (つまり、MANAGED BY DATABASE 節を使用して作成されたもの) のサイズは、 1 つ以上のコンテナーを表スペースに追加することによって増やすことができます。

表スペースに新しいコンテナーを追加したり、既存のコンテナーを拡張したりすると、表スペースのリバランスという処理が発生する場合があります。バランスの再調整のプロセスには、表スペース・エクステントをある場所から別の場所に移動することが含まれます。このプロセスの最中、データは表スペース内にストライピングされた状態に保たれます。バランスの再調整は、必ずしもすべてのコンテナーにわたって発生するわけではなく、既存のコンテナー構成、新規コンテナーのサイズ、表スペースがどの程度満杯になっているかなど、多くの要素に依存しています。

コンテナーが既存の表スペースに追加される時には、コンテナーがストライプ 0 で開始されないように追加される場合があります。 174 ページの『DMS 表スペース・マップ』にその説明があります。マップ内のどこで開始するかはデータベース・マネージャーにより決定され、それは追加されるコンテナーのサイズに基づいてなされます。追加されるコンテナーが十分に大きくなければ、そのコンテナーがマップの最後のストライプで終わるように配置されます。それが十分に大きい場合には、ストライプ 0 で開始するように配置されます。

新規コンテナーを追加し、新規ストライプ・セットを作成する場合には、バランスの再調整は発生しません。 ALTER TABLESPACE ステートメントで BEGIN NEW STRIPE SET 節を使用して、新規ストライプ・セットを作成します。さらに、ALTER TABLESPACE ステートメントで ADD TO STRIPE SET 節を使用して、既存のストライプ・セットにコンテナーを追加することもできます。

バランスの再調整中も、表スペースへのアクセスは制限されません。複数のコンテナーを追加する必要がある場合は、それらを同時に追加しなければなりません。

コマンド行を使用して DMS 表スペースをコンテナーに追加するには、以下のようにします。

```
ALTER TABLESPACE <name>
   ADD (DEVICE '<path>' <size>, FILE '<filename>' <size>)
```

以下の例は、Linux および UNIX システム上にある表スペースに対して、2 つの新しいデバイス・コンテナー (それぞれが 10 000 ページのもの) を追加する方法を示したものです。

```
ALTER TABLESPACE RESOURCE
  ADD (DEVICE '/dev/rhd9'  10000,
       DEVICE  '/dev/rhd10' 10000)
```

ALTER TABLESPACE ステートメントを使用すると、パフォーマンスに影響を及ぼす可能性のある、表スペースの他の特性を変更することができます。

DMS コンテナーのリバランス

ALTER TABLESPACE ステートメントを使用すれば、既存の表スペースにコンテナーを追加したり、コンテナーを拡張して記憶容量を増やしたりできます。

コンテナーを自動ストレージ表スペースに手動で追加することはできません。データベース・マネージャーはコンテナーを必要に応じて自動的に拡張または追加します。

表スペースを作成すると、その表スペース・マップが作成され、すべての初期コンテナーがストライプ 0 で開始するようにラインアップします。したがって、個々の表スペース・コンテナーが満杯になるまで、データはすべての表スペース・コンテナー間で均等にストライピングされます。 (例 1 を参照。)

既存のコンテナーよりも小さいコンテナーを追加すると、データの分散が不均等になります。この結果、等しいサイズのコンテナーの場合よりも、データの事前取り出しなどの並列入出力の実行の効率が低下します。

表スペースに新しいコンテナーを追加したり、既存のコンテナーを拡張したりすると、表スペース・データのリバランスという処理が発生する場合があります。

再平衡化

コンテナーの追加または拡張を実行したときのリバランスの処理では、表スペースのエクステントが 1 つのロケーションから別のロケーションに移動します。目的は、表スペース内のデータのストライピングをきちんと調整することです。

リバランスの処理中も、表スペースへのアクセスは制限されません。通常どおり、オブジェクトのドロップ、作成、データ挿入、照会ができます。しかし、リバランスの処理は、パフォーマンスに大きな影響を与えます。複数のコンテナーを追加する必要があって、しかもコンテナーのリバランスを実行するつもりであれば、 1 つの ALTER TABLESPACE ステートメントですべてのコンテナーを同時に追加するべきです。そうすれば、データベース・マネージャーがデータのリバランスを何度も行う必要はなくなります。

リバランスの処理では、表スペースの最高水準点が重要な役割を果たします。この場合の最高水準点とは、表スペース内で割り振られている最高ページのページ番号です。例えば、 1000 ページの表スペースがあって、エクステントのサイズが 10 であれば、エクステントの数は 100 になります。その表スペース内で割り振られている最高エクステントが 42 番目のエクステントであれば、最高水準点は、42 * 10 = 420 ページということになります。これは、使用済みページの数と同じではありません。最高水準点よりも低い位置にあるエクステントの中には、再利用のために解放されるものもあるからです。

このリバランスが始まる前に、コンテナーの変更内容に基づいて、新しい表スペース・マップが作成されます。リバランサーとは、現在のマップで決められているロケーションから、新しいマップで決められているロケーションにエクステントを移す処理です。リバランサーの処理は、エクステント 0 から始まり、エクステントを 1 つずつ移していき、最後に最高水準点を含んでいるエクステントを移します。エクステントを移すたびに、現在のマップが新しいマップに合わせて 1 箇所ずつ変更されていきます。リバランス処理の完了時には、現在のマップと新しいマップは、最高水準点を含んでいるストライプの位置までまったく同じになっているはずです。現在のマップは、新しいマップとまったく同じになり、リバランスの処理は完了します。現在のマップ内のエクステントのロケーションが新しいマップ内のロケーションと同じ場合は、エクステントの移動は行われず、 I/O も発生しません。

新しいコンテナーを追加した場合、新しいマップ内でのそのコンテナーの位置は、そのコンテナーのサイズと、そのストライプ・セット内の他のコンテナーのサイズによって決まります。追加するコンテナーが、ストライプ・セット内の最初のストライプから始まり、ストライプ・セット内の最後のストライプで終わる (または最後のストライプを超える) だけのサイズを持っていれば、そのコンテナーはそのような形で配置されます (例 2 を参照)。そのコンテナーがそれだけのサイズを持っていなければ、マップ内では、そのコンテナーがストライプ・セット内の最後のストライプで終わるように配置されます (例 4を参照)。このような動作になっているのは、リバランスを必要とするデータの量を最小限に抑えるためです。

注: 以下の例では、コンテナー・サイズを決定する上でコンテナー・タグのサイズは考慮されていません。また、目的は単に例を示すことなので、かなり小さなコンテナー・サイズを使っていますが、そのサイズをお勧めするという意味ではありませんのでご注意ください。さらに、 1 つの表スペース内でいろいろなサイズのコンテナーを使っていますが、実際には、同じサイズのコンテナーを使用することをお勧めします。

例 1:

3 つのコンテナーを含んだ表スペースを作成するとします。エクステントのサイズは 10、 3 つのコンテナーのサイズはそれぞれ 60 ページ、40 ページ、 80 ページ (6 エクステント、4 エクステント、 8 エクステント) です。この場合にマップに基づいて作成される表スペースを図にすると、 221 ページの図 13 のようになります。

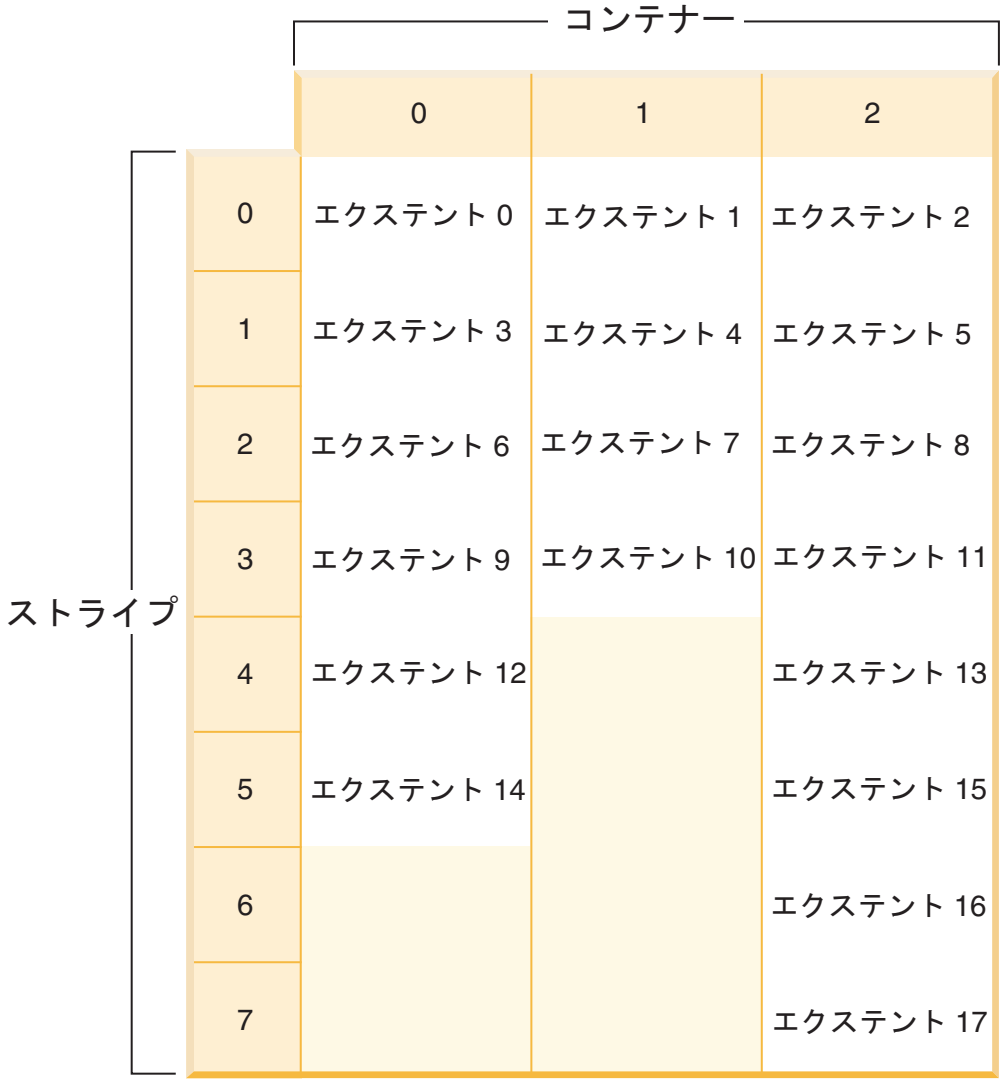


図 13. 3 つのコンテナー、18 のエクステントが含まれる表スペース

表スペースのスナップショットから分かるとおり、対応する表スペース・マップは、次のようになります。

```
Range     Stripe  Stripe  Max        Max    Start    End    Adj.   Containers
Number    Set     Offset  Extent     Page   Stripe   Stripe
   [0]      [0]      0      11       119       0        3    0   3 (0, 1, 2)
   [1]      [0]      0      15       159       4        5    0   2 (0, 2)
   [2]      [0]      0      17       179       6        7    0   1 (2)
```

表スペース・マップの見出しは、Range Number (範囲番号)、 Stripe Set (ストライプ・セット)、Stripe Offset (ストライプ・オフセット)、 Maximum extent number addressed by the range (範囲でアドレス指定される最大エクステント番号)、Maximum page number addressed by the range (範囲でアドレス指定される最大ページ番号)、 Start Stripe (開始ストライプ)、End Stripe (終了ストライプ)、 Range adjustment (範囲調整)、および Container list (コンテナー・リスト) です。

例 2:

例 1 の表スペースに 80 ページのコンテナーを追加するとしましょう。この場合、追加するコンテナーは、最初のストライプ (ストライプ 0) から始まり、最後のストライプ (ストライプ 7) で終わるだけのサイズになっています。したがって、このコンテナーは、最初のストライプから始まる位置に配置されます。結果として生成される表スペースは、 図 14 の図のようになります。

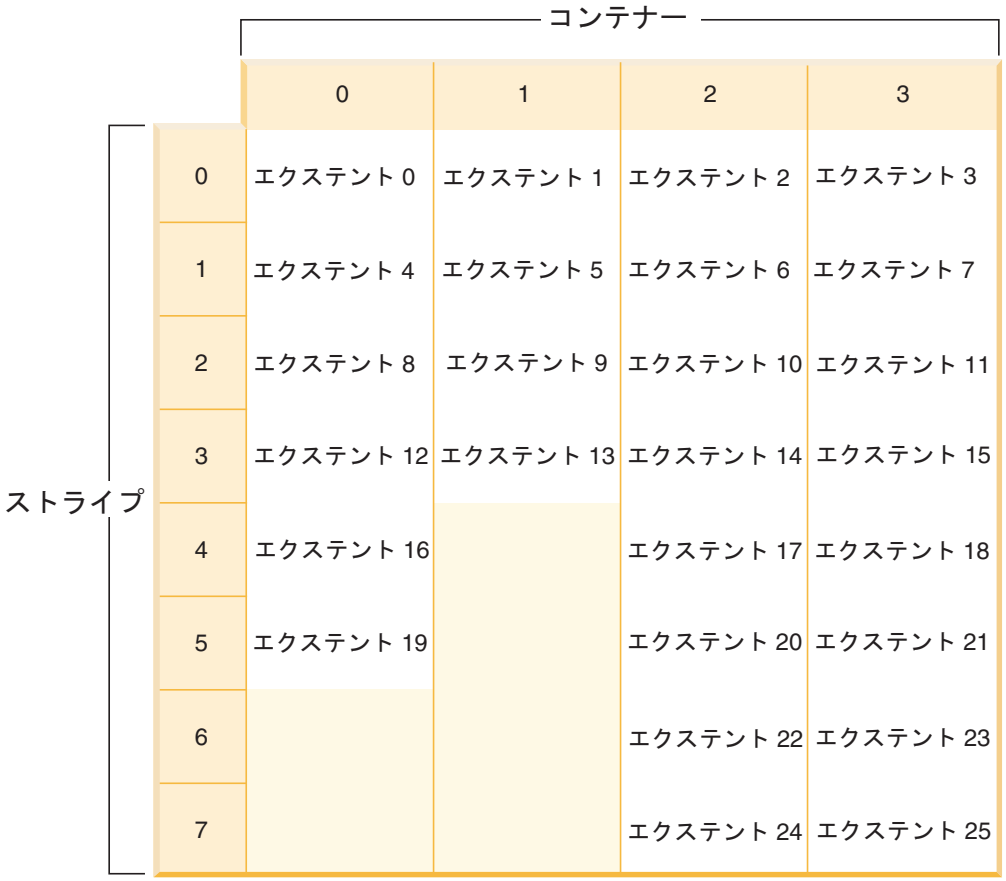


図 14. 4 つのコンテナー、26 のエクステントが含まれる表スペース

表スペースのスナップショットから分かるとおり、対応する表スペース・マップは、次のようになります。

```
Range     Stripe  Stripe  Max        Max    Start    End    Adj.   Containers
Number    Set     Offset  Extent     Page   Stripe   Stripe
   [0]       [0]      0      15      159         0       3    0   4 (0, 1, 2, 3)
   [1]       [0]      0      21      219         4       5    0   3 (0, 2, 3)
   [2]       [0]      0      25      259         6       7    0   2 (2, 3)
```

ここで、最高水準点がエクステント 14 にあるとすれば、リバランサーの処理は、エクステント 0 から始まり、エクステント 14 までのすべてのエクステントが移動することになります。両方のマップのエクステント 0 のロケーションは同じなので、このエクステントは、動かす必要がありません。同じことがエクステント 1 と 2 についても言えます。一方、エクステント 3 は動かす必要があります。したがって、古いロケーション (コンテナー 0 の 2 番目のエクステント) から読み取られて、新しいロケーション (コンテナー 3 の 1 番目のエクステント) に書き込まれま

す。このあと、エクステント 14 までのすべてのエクステントが移動します。エクステント 14 が移動した時点で、現在のマップが新しいマップと同じになって、リバランサーの処理が終了します。

この場合、新しく追加するスペースをすべて最高水準点の後に入れるようにマップを変更すれば、リバランスの処理は不要になり、すべてのスペースがすぐに使用できる状態になります。一方、一部のスペースだけを最高水準点の後に入れるようにマップを変更すれば、最高水準点よりも上の位置にあるストライプ内のスペースだけがすぐに使用できる状態になります。残りのスペースは、リバランスの処理が完了するまで使用できません。

コンテナーを拡張する場合も、リバランサーの動作は同じです。ストライプ・セット内の最後のストライプを超えてコンテナーを拡張する場合は、それに合わせてストライプ・セットが拡張され、その後のストライプ・セットもシフトアウトすることになります。したがって、コンテナーが後続のストライプ・セットに入り込むことはありません。

例 3:

例 1 の表スペースを再び取り上げましょう。コンテナー 1 を 40 ページから 80 ページに拡張すると、新しい表スペースは、 224 ページの図 15 のようになります。

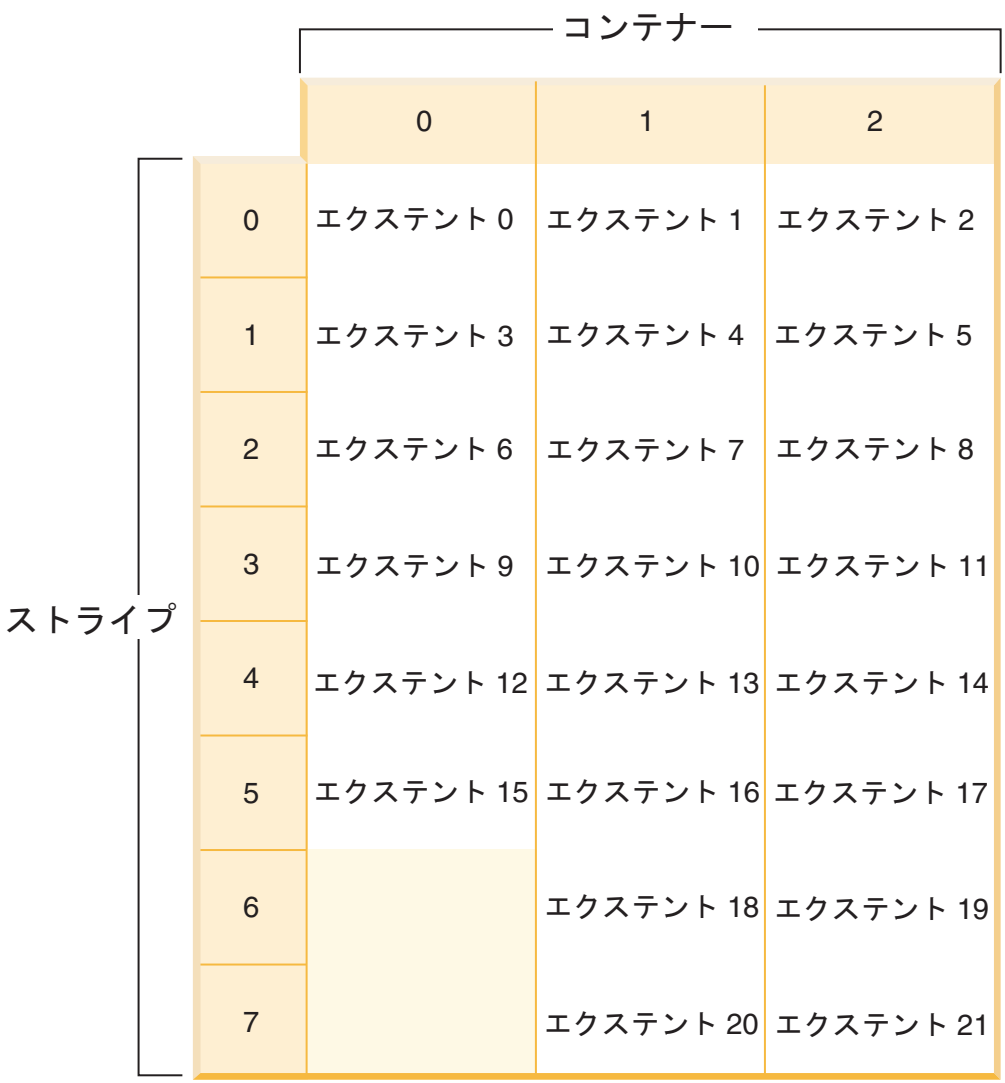


図 15. 3 つのコンテナー、22 のエクステントが含まれる表スペース

表スペースのスナップショットから分かるとおり、対応する表スペース・マップは、次のようになります。

```
Range     Stripe  Stripe  Max        Max    Start    End    Adj.   Containers
Number    Set     Offset  Extent     Page   Stripe   Stripe
   [0]       [0]      0      17      179       0        5    0   3 (0, 1, 2)
   [1]       [0]      0      21      219       6        7    0   2 (1, 2)
```

例 4:

例 1 の表スペースをさらに取り上げましょう。その表スペースに 50 ページ (5 エクステント) のコンテナーを追加すると、そのコンテナーは新しいマップに次のような要領で追加されます。この場合のコンテナーは、最初のストライプ (ストライプ 0) から始まり、最後のストライプ (ストライプ 7) で終わる (または最後のストライプを超える) だけのサイズになっていないので、最後のストライプで終わるような位置に配置されます。 (225 ページの図 16 を参照。)

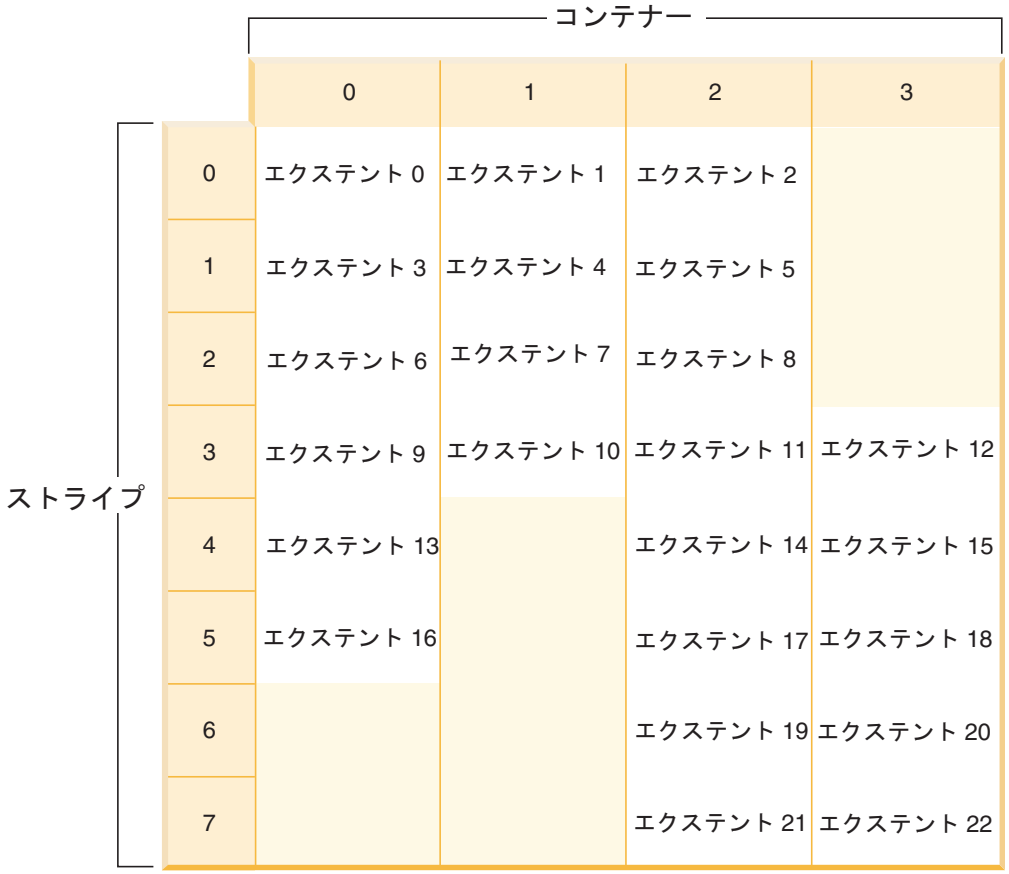


図 16. 4 つのコンテナー、23 のエクステントが含まれる表スペース

表スペースのスナップショットから分かるとおり、対応する表スペース・マップは、次のようになります。

```
Range     Stripe  Stripe  Max        Max    Start    End    Adj.   Containers
Number    Set     Offset  Extent     Page   Stripe   Stripe
   [0]      [0]       0       8       89        0       2    0   3 (0, 1, 2)
   [1]      [0]       0      12      129        3       3    0   4 (0, 1, 2, 3)
   [2]      [0]       0      18      189        4       5    0   3 (0, 2, 3)
   [3]      [0]       0      22      229        6       7    0   2 (2, 3)
```

コンテナーを拡張するには、 ALTER TABLESPACE ステートメントで EXTEND 節か RESIZE 節を使用します。コンテナーを追加して、データのリバランスの処理を実行するには、 ALTER TABLESPACE ステートメントで ADD 節を使用します。複数のストライプ・セットがある表スペースにコンテナーを追加する場合は、どのストライプ・セットに追加するかを指定できます。そのためには、 ALTER TABLESPACE ステートメントで ADD TO STRIPE SET 節を使用します。ストライプ・セットを指定しないと、デフォルトの動作として、現在のストライプ・セットにコンテナーが追加されます。現在のストライプ・セットとは、最後に作成されたストライプ・セットであり、最後にスペースが追加されたストライプ・セットではありません。

ストライプ・セットを変更すると、そのストライプ・セットと後続のストライプ・セットに対して、リバランスの処理が実行される場合があります。

リバランスの処理の進行状況をモニターするには、表スペースのスナップショットを使用します。表スペースのスナップショットからは、リバランスの開始時刻、移動したエクステントの数、移動する必要があるエクステントの数など、リバランスの処理についての情報が得られます。

リバランスを回避する方法 (ストライプ・セットの使用)

コンテナーの追加または拡張を実行する場合でも、表スペースの最高水準点を超えた位置にスペースを追加するのであれば、リバランスの処理は実行されません。

コンテナーを追加する場合は、最高水準点の下の位置にスペースを追加するケースがほとんどです。したがって、コンテナーを追加したときには、リバランスの処理が必要になるケースが多いわけです。しかし、最高水準点よりも上の位置に新しいコンテナーを強制的に追加するための方法があります。この方法を使用すれば、表スペースのコンテンツのリバランスの処理を回避できます。このような方法の利点は、新しいコンテナーをすぐに使用できるということです。リバランスの処理を回避するためのオプションは、コンテナーを追加する場合にのみ使用できるもので、既存のコンテナーを拡張する場合には使用できません。コンテナーを拡張する場合は、最高水準点よりも上の位置にスペースを追加するときに限り、リバランスの処理が回避されます。例えば、同じサイズのコンテナーがいくつかある状況で、それぞれのコンテナーを同じ量だけ拡張する場合、エクステントの相対位置は変わらないため、リバランスの処理は発生しません。

表スペースにコンテナーを追加するときにリバランスの処理を回避するには、新しいストライプ・セット を追加します。表スペース内のいくつかのコンテナーにまたがってデータがストライピングされている場合、それらのコンテナーをまとめてストライプ・セットと呼んで、同じ表スペース内の他のコンテナーとは別個に扱うようにしています。そうすれば、既存のストライプ・セット内の既存のコンテナーには触れずに、その新しいストライプ・セットにコンテナーを追加できます。

コンテナーを追加して、リバランスの処理を回避するには、 ALTER TABLESPACE ステートメントで BEGIN NEW STRIPE SET 節を使用します。

例 5:

3 つのコンテナーを含んだ表スペースを作成するとします。エクステントのサイズは 10、 3 つのコンテナーのサイズはそれぞれ 30 ページ、40 ページ、 40 ページ (3 エクステント、4 エクステント、 4 エクステント) です。この場合の表スペースを図にすると、 227 ページの図 17 のようになります。

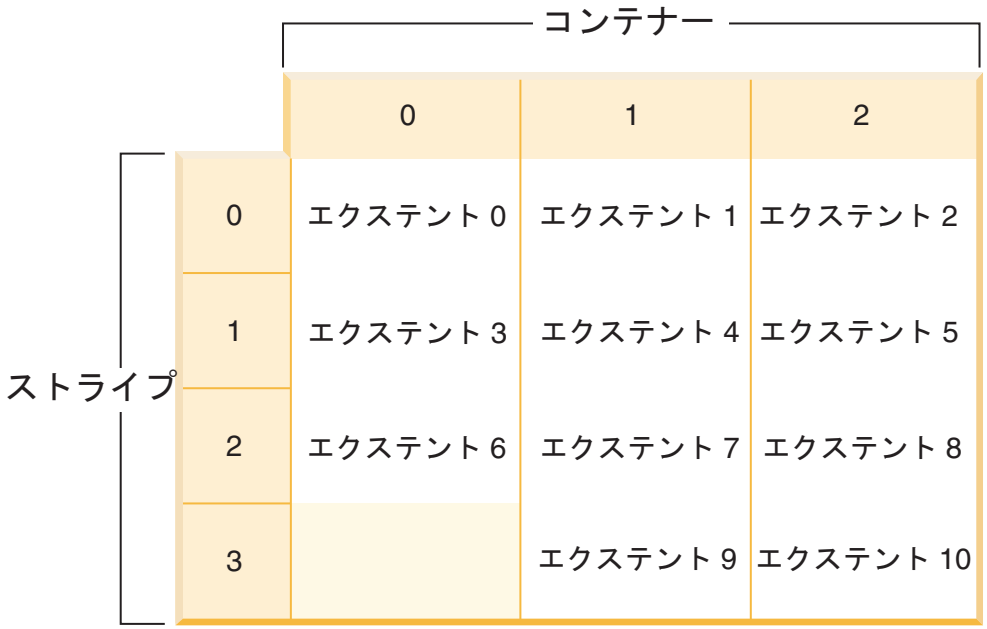


図 17. 3 つのコンテナー、11 のエクステントが含まれる表スペース

表スペースのスナップショットから分かるとおり、対応する表スペース・マップは、次のようになります。

```
Range     Stripe  Stripe  Max        Max    Start    End    Adj.   Containers
Number    Set     Offset  Extent     Page   Stripe   Stripe
   [0]       [0]      0       8       89        0        2    0   3 (0, 1, 2)
   [1]       [0]      0      10      109        3        3    0   2 (1, 2)
```

例 6:

BEGIN NEW STRIPE SET 節を指定して、 30 ページと 40 ページ (それぞれ 3 エクステントと 4 エクステント) の 2 つの新しいコンテナーを 1 つずつ追加する場合、既存の範囲は影響を受けずに、新しい範囲セットが作成されます。この新しい範囲セットがストライプ・セットであり、最後に作成されたストライプ・セットが現在のストライプ・セットということになります。その 2 つのコンテナーを追加した後の表スペースは、 228 ページの図 18 のようになります。

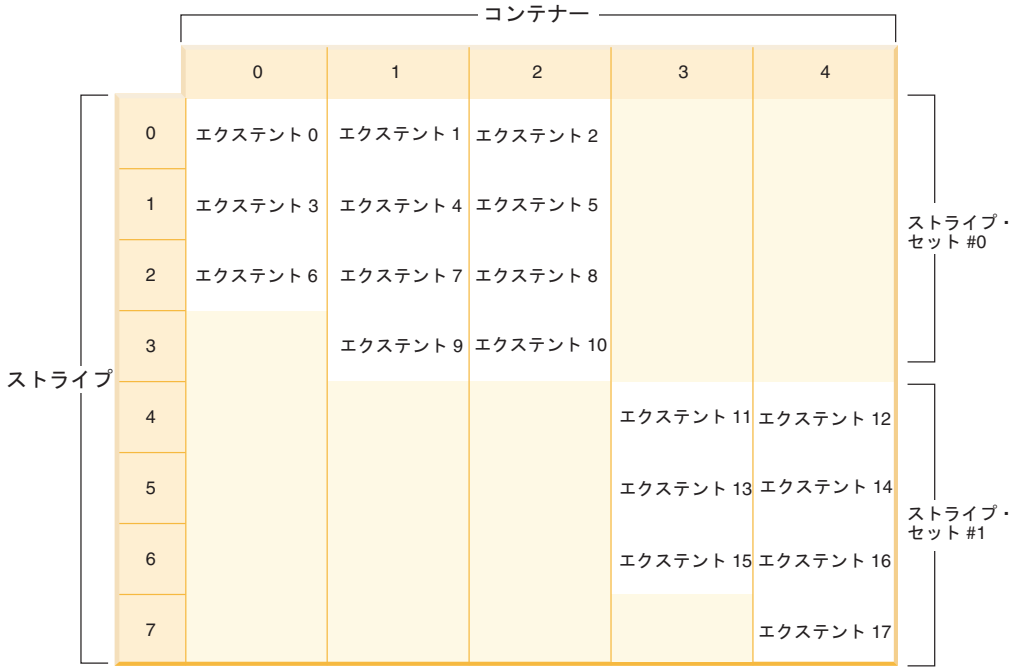


図 18. 2 つのストライプ・セットを含む表スペース

表スペースのスナップショットから分かるとおり、対応する表スペース・マップは、次のようになります。

```
Range     Stripe  Stripe  Max        Max    Start    End    Adj.   Containers
Number    Set     Offset  Extent     Page   Stripe   Stripe
   [0]       [0]      0       8       89        0        2    0   3 (0, 1, 2)
   [1]       [0]      0      10      109        3        3    0   2 (1, 2)
   [2]       [1]      4      16      169        4        6    0   2 (3, 4)
   [3]       [1]      4      17      179        7        7    0   1 (4)
```

表スペースに新しいコンテナーを追加するときに、 ADD 節に TO STRIPE SET 節を指定しない 場合は、コンテナーが現在のストライプ・セット (最高のストライプ・セット) に追加されます。 ADD TO STRIPE SET 節を使用すれば、表スペース内の指定したストライプ・セットにコンテナーを追加できます。もちろん、有効なストライプ・セットを指定する必要があります。

データベース・マネージャーは、表スペース・マップを使用してストライプ・セットをトラッキングします。したがって、新しいコンテナーを追加して、リバランスの処理を回避すると、通常は、コンテナーのリバランスの処理を実行した場合よりも、マップのサイズが早く大きくなっていきます。表スペース・マップのサイズが大きくなりすぎた場合は、コンテナーを追加しようとしたときに、エラー SQL0259N を受け取ることになります。

DMS コンテナーのサイズ変更

自動ストレージ表スペース内でコンテナーを手動でサイズ変更することはできません。

データベース・マネージャーはコンテナーを必要に応じて自動的に拡張または追加します。ただし、DMS 表スペース (つまり、MANAGED BY DATABASE 節を使用して作成されたもの) のコンテナーをサイズ変更することができます。

それぞれのロー・デバイスは、1 つのコンテナーとしてのみ使用できます。ロー・デバイスのサイズは、それが作成された後は固定されます。 RESIZE や EXTEND オプションを使用してロー・デバイス・コンテナーを増やすことを考慮している時には、まずロー・デバイスのサイズをチェックして、デバイス・コンテナーのサイズをロー・デバイスのサイズより大きく増やそうとすることが決してないようにしなければなりません。

さらに、DMS 表スペースから既存のコンテナーをドロップしたり、 DMS 表スペースにある既存のコンテナーのサイズを縮小したり、さらにすべてのコンテナーにわたるデータのバランスの再調整を必要とせずに DMS 表スペースに新規コンテナーを追加したりもできます。

既存の表スペース・コンテナーのドロップや既存のコンテナーのサイズの縮小が可能なのは、ドロップされるエクステントまたはサイズが縮小されるエクステントの数が、表スペースの「最高水準点」を超えた位置にあるフリー・エクステントの数以下である場合に限られます。この場合の最高水準点とは、表スペース内で割り振られている最高ページのページ番号です。このマークは、表スペースの使用済みページの数と同じではありません。最高水準点よりも低い位置にあるエクステントの中には、再利用のために入手できるものもあるからです。

最高水準点まで (それを含む) のエクステントはすべて、表スペース内の同じ論理位置にとどまっている必要があるため、表スペースの最高水準点より上にあるフリー・エクステントの数は重要です。その操作によって生成される表スペースには、データをすべて収容するだけのスペースが必要になります。十分のフリー・スペースがない場合には、エラー・メッセージ (SQL20170N, SQLSTATE 57059) が出されます。

コンテナーをドロップするには、ALTER TABLESPACE ステートメントで DROP オプションを使用します。例:

```
ALTER TABLESPACE TS1 DROP (FILE 'file1', DEVICE '/dev/rdisk1')
```

既存のコンテナーのサイズを縮小するには、RESIZE オプションまたは REDUCE オプションを使用します。 RESIZE オプションを使用する時、ステートメントの一部としてリストされているすべてのコンテナーは、サイズを増加させるか、サイズを縮小させるかの一方しか行えません。同じステートメント内で、幾つかのコンテナーを増やして、他のコンテナーを減らすことはできません。コンテナーのサイズの新しい下限がわかっている場合には、サイズ変更方式 (RESIZE) を利用します。コンテナーの現在のサイズがわからない場合 (または考慮していない場合) には、縮小方式（REDUCE）を利用します。

コマンド行を使用して DMS 表スペース中の 1 つ以上のコンテナーのサイズを減らすには、以下のようにします。

```
ALTER TABLESPACE <name>
   REDUCE (FILE '<filename>' <size>)
```

以下の例は、Windows ベースのシステム上の表スペース中にある、ファイル・コンテナー (既に 1 000 ページあるもの) のサイズを縮小する方法を示したものです。

```
ALTER TABLESPACE PAYROLL
  REDUCE (FILE 'd:¥hldr¥finance'  200)
```

このアクションを実行すると、ファイルは 1 000 ページから 800 ページにサイズが減少します。

コマンド行を使用して DMS 表スペース中の 1 つ以上のコンテナーのサイズを増やすには、以下のようにします。

```
ALTER TABLESPACE <name>
   RESIZE (DEVICE '<path>' <size>)
```

以下の例は、Linux および UNIX システム上の表スペース中にある、2 つのデバイス・コンテナー (それぞれが 1 000 ページのもの) のサイズを増やす方法を示したものです。

```
 ALTER TABLESPACE HISTORY
   RESIZE (DEVICE '/dev/rhd7'  2000,
           DEVICE  '/dev/rhd8' 2000)
```

このアクションを実行すると、2 つのデバイスは 1 000 ページから 2 000 ページにサイズが増えます。すべてのコンテナーにわたって表スペースのコンテナーのバランスが再調整されます。バランスの再調整中も、表スペースへのアクセスは制限されません。

コマンド行を使用して DMS 表スペース中の 1 つ以上のコンテナーを拡張するには、以下のようにします。

```
ALTER TABLESPACE <name>
   EXTEND (FILE '<filename>' <size>)
```

以下の例は、Windows ベースのシステム上の表スペース中にある、ファイル・コンテナー (それぞれが 1 000 ページのもの) のサイズを増やす方法を示したものです。

```
ALTER TABLESPACE PERSNEL
   EXTEND (FILE 'e:¥wrkhist1'  200
           FILE  'f:¥wrkhist2'  200)
```

このアクションを実行すると、2 つのファイルは 1 000 ページから 1 200 ページにサイズが増えます。すべてのコンテナーにわたって表スペースのコンテナーのバランスが再調整されます。バランスの再調整中も、表スペースへのアクセスは制限されません。

DMS コンテナー (ファイルとロー・デバイス・コンテナーの両方) の追加または変更は、プリフェッチャーを使用して並列で実行されます。このような作成またはサイズ変更コンテナー操作の並列処理を高めるために、システムで実行されるプリフェッチャーの数を増やすことができます。並列で行われないプロセスは、これらのアクションのロギングと、コンテナーの作成の場合は、コンテナーのタグ付けだけです。

注: CREATE TABLESPACE または ALTER TABLESPACE ステートメントの並列処理 (新しいコンテナーの既存の表スペースへの追加に関して) を最大限にするには、プリフェッチャーの数が、追加されるコンテナーの数と等しいかそれ以上であることを確かめてください。プリフェッチャーの数は、*num_ioservers* データベース構成パラメーターにより制御されます。データベースは、新規パラメーター値を有

効にするためには、停止しなければなりません。つまり、すべてのアプリケーションおよびユーザーは、変更を有効にするためにデータベースから切断する必要があります。

ALTER TABLESPACE ステートメントを使用すると、パフォーマンスに影響を及ぼす可能性のある、表スペースの他の特性を変更することができます。

DMS コンテナーのドロップまたは削減

DMS 表スペースでは、ALTER TABLESPACE ステートメントを使用して、表スペースからコンテナーをドロップしたり、コンテナーのサイズを縮小することができます。

コンテナーのドロップや縮小が可能なのは、その操作によってドロップされるエクステントの数が、表スペースの最高水準点を超えた位置にあるフリー・エクステントの数以下である場合に限られます。要するに、その操作によってページ番号は変更できないため、最高水準点までのエクステントはすべて、表スペース内の同じ論理位置にとどまっている必要があるわけです。したがって、その操作によって生成される表スペースには、最高水準点までのデータをすべて収容するだけのスペースが必要になります。十分なフリー・スペースがない状況では、ステートメントの実行時にすぐにエラーを受け取ることになります。

この場合の最高水準点とは、表スペース内で割り振られている最高ページのページ番号です。例えば、 1000 ページの表スペースがあって、エクステントのサイズが 10 であれば、エクステントの数は 100 になります。その表スペース内で割り振られている最高エクステントが 42 番目のエクステントであれば、最高水準点は、42 * 10 = 420 ページということになります。これは、使用済みページの数と同じではありません。最高水準点よりも低い位置にあるエクステントの中には、再利用のために解放されるものもあるからです。

コンテナーのドロップや縮小を実行したときに、表スペースからドロップされるスペース内にデータが入っていると、リバランスという処理が発生します。このリバランスが始まる前に、コンテナーの変更内容に基づいて、新しい表スペース・マップが作成されます。リバランサーとは、現在のマップで決められているロケーションから、新しいマップで決められているロケーションにエクステントを移す処理です。リバランサーの処理は、最高水準点を含んでいるエクステントから始まり、エクステントを 1 つずつ移していき、最後にエクステント 0 を移します。エクステントを移すたびに、現在のマップが新しいマップに合わせて 1 箇所ずつ変更されていきます。現在のマップ内のエクステントのロケーションが新しいマップ内のロケーションと同じ場合は、エクステントの移動は行われず、 I/O も発生しません。この場合のリバランスは、割り振られている最高エクステントから始まり、表スペース内の最初のエクステントで終わるため、*リバース・リバランス* と呼ばれています(その逆に、コンテナーの追加または拡張の後に表スペースにスペースが追加される場合は、*フォワード・リバランス* が発生することになります)。

コンテナーをドロップすると、残りのコンテナーの番号が再割り振りされ、 0 から 1 ずつ増えていくようにコンテナー ID がきれいに調整されます。ストライプ・セット内のすべてのコンテナーをドロップした場合は、そのストライプ・セットがマップから削除され、そのマップ内の残りのストライプ・セットがすべてシフトダウンして、ストライプ・セット番号の抜けがないように調整されます。

注: 以下の例では、コンテナー・サイズを決定する上でコンテナー・タグのサイズは考慮されていません。また、目的は単に例を示すことなので、かなり小さなコンテナー・サイズを使っていますが、そのサイズをお勧めするという意味ではありませんのでご注意ください。さらに、 1 つの表スペース内でいろいろなサイズのコンテナーを使っていますが、これも単に例を示すという目的なので、実際には、同じサイズのコンテナーを使用することをお勧めします。

例えば、 3 つのコンテナーが入っている表スペースがあって、エクステントのサイズが 10 だとしましょう。 3 つのコンテナーのサイズは、それぞれ 20 ページ、50 ページ、50 ページ (2 エクステント、5 エクステント、5 エクステント) です。この表スペースの図を図 19 に示します。

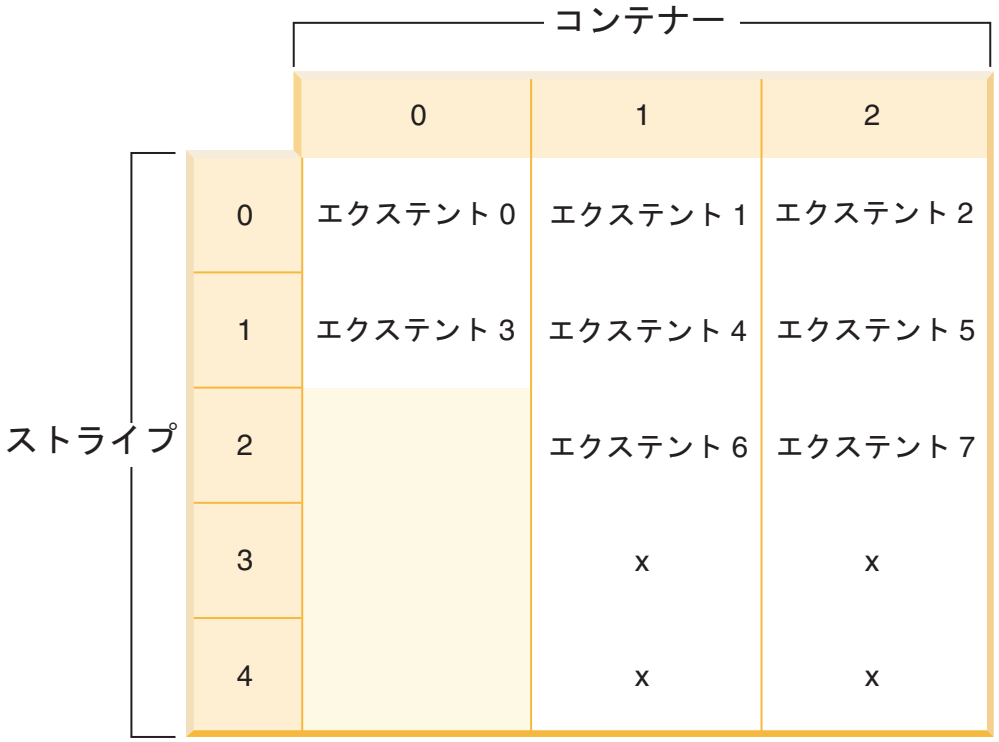


図 19. 12 個のエクステントを含む表スペース (そのうち 4 個のエクステントにはデータが入っていない)

X というマークは、エクステントは存在するものの、そのエクステントにデータが入っていないという意味です。

2 つのエクステントが入っているコンテナー 0 をドロップするには、最高水準点よりも上の位置に、少なくとも 2 つのフリー・エクステントがなければなりません。この場合の最高水準点はエクステント 7 なので、 4 つのフリー・エクステントが残っています。したがって、コンテナー 0 のドロップは可能です。

表スペースのスナップショットから分かるとおり、対応する表スペース・マップは、次のようになります。

```
Range     Stripe  Stripe  Max        Max    Start    End    Adj.   Containers
Number    Set     Offset  Extent     Page   Stripe   Stripe
   [0]       [0]       0       5       59        0        1    0   3 (0, 1, 2)
   [1]       [0]       0      11      119        2        4    0   2 (1, 2)
```

ドロップ操作の後、表スペースはコンテナー 0 とコンテナー 1 だけになります。新しい表スペースの図を図 20 に示します。

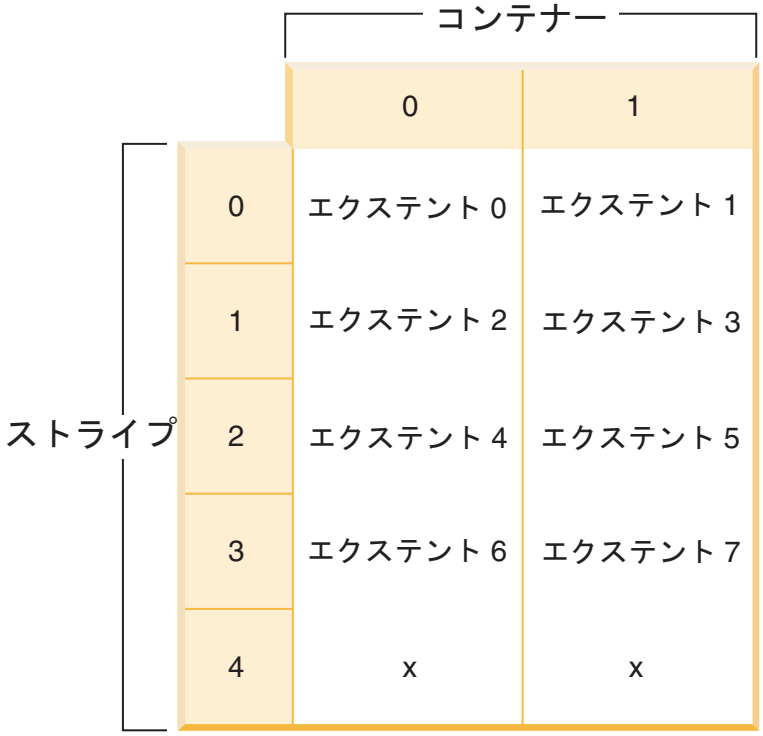


図 20. コンテナーをドロップした後の表スペース

表スペースのスナップショットから分かるとおり、対応する表スペース・マップは、次のようになります。

```
Range     Stripe  Stripe  Max        Max    Start    End    Adj.   Containers
Number    Set     Offset  Extent     Page   Stripe   Stripe
   [0]       [0]       0       9       99        0        4    0   2 (0, 1)
```

コンテナーのサイズを縮小する場合も、リバランサーの動作は同じです。

コンテナーを縮小するには、 ALTER TABLESPACE ステートメントで REDUCE オプションか RESIZE オプションを使用します。コンテナーをドロップするには、ALTER TABLESPACE ステートメントで DROP オプションを使用します。

自動ストレージの表スペースの変更

自動ストレージ表スペースの場合、コンテナーのサイズの削減のみを行うことができます。

これを行うプロセスは、 228 ページの『DMS コンテナーのサイズ変更』に説明されているものと同じです。

表スペースの名前変更

表スペースの名前変更には、RENAME TABLESPACE ステートメントを使用します。

SYSCATSPACE 表スペースの名前は変更できません。「ロールフォワード・ペンディング」または「ロールフォワード進行中」状態の表スペースの名前を変更することはできません。

バックアップを取った後に名前変更した表スペースをリストアする際には、RESTORE DATABASE コマンド中で新しい表スペースの名前を使用しなければなりません。変更前の表スペース名を使用しても、検出されません。同様に、ROLLFORWARD DATABASE コマンドを使用して表スペースをロールフォワードする場合も、必ず新しい名前を使用するようにしてください。変更前の表スペース名を使用しても、検出されません。

既存の表スペースに含まれる、個々のオブジェクトについて考慮する必要なしに、表スペースの名前を新たに付けることができます。表スペースの名前を変更すると、その表スペースを参照するカタログ・レコードもすべて変更されます。

オフラインからオンラインへの表スペースの切り替え

表スペースに関連付けられたコンテナーがすでにアクセス可能になっている場合、ALTER TABLESPACE ステートメントの SWITCH ONLINE 節を使用して、表スペースから OFFLINE 状態を除去できます。

表スペースは、データベースがまだ起動し作動している間に、OFFLINE 状態を除去します。

この節を使用する代わりに、データベースからすべてのアプリケーションを切断し、再びアプリケーションをデータベースに接続しなおすこともできます。これで、表スペースから OFFLINE 状態が除去されます。

コマンド行を使用して表スペースから OFFLINE 状態を除去するには、以下のように入力します。

```
db2 ALTER TABLESPACE <name>
   SWITCH ONLINE
```

データが RAID 装置にある場合の表スペース・パフォーマンスの最適化

データが RAID (新磁気ディスク制御機構) 装置に置かれている場合に、パフォーマンスを最適化するには以下の指針に従ってください。

1. 一連の RAID 装置上に表スペースを作成する場合、指定の表スペース (SMS または DMS) のコンテナーを別々の装置上に作成してください。

   146 GB のディスクが 15 個 (RAID-5 アレイ 3 つと、各アレイに 5 つのディスク) 構成されているという例について考えてみます。フォーマット設定後、各

ディスクでは約 136 GB のデータを保持できます。したがって、各アレイは約 544 GB (4 アクティブ・ディスク x 136 GB) を格納できます。300 GB のストレージを要求する表スペースがある場合、3 つのコンテナーを作成し、各コンテナーをそれぞれ異なる装置上に配置します。各コンテナーは 1 つの装置上で 100 GB (300 GB/3) を使用し、追加の表スペース用として各装置には 444 GB (544 GB - 100 GB) が残ります。

2. 表スペース用に適切なエクステント・サイズを選択します。表スペース用のエクステント・サイズとは、データベース・マネージャーが次のコンテナーに書き込むまでに 1 つのコンテナーに書き込むデータ量のことです。理想としては、エクステント・サイズはディスクの基礎となるセグメント・サイズの倍数にしてください。このセグメント・サイズとは、ディスク・コントローラーが次の物理ディスクに書き込むまでに 1 つの物理ディスクに書き込むデータ量のことです。セグメント・サイズの倍数であるエクステント・サイズを選択すると、プリフェッチにおける並列順次読み取りなどの複数のエクステント・ベースの操作が同一の物理ディスクで競合しないようにできます。また、ページ・サイズの倍数であるエクステント・サイズを選択してください。

   この例でセグメント・サイズが 64 KB で、ページ・サイズが 16 KB の場合、適切なエクステント・サイズは 256 KB となる可能性があります。

3. DB2_PARALLEL_IO レジストリー変数を使用して、すべての表スペースの並列入出力を使用可能にしたり、コンテナー当たりの物理ディスク数を指定したりします。

   この例の場合、DB2_PARALLEL_IO = *:4 を設定します。

   表スペースのプリフェッチ・サイズを AUTOMATIC に設定すると、データベース・マネージャーはプリフェッチ・サイズ値を判別するために、DB2_PARALLEL_IO に指定した物理ディスク数の値を使用します。プリフェッチ・サイズが AUTOMATIC に設定されていない場合には、RAID ストライプ・サイズを考慮に入れてプリフェッチ・サイズを手動で設定できます。RAID ストライプ・サイズとは、セグメント・サイズをアクティブなディスク数で乗算した値です。以下の条件を満たすプリフェッチ・サイズ値を選択します。

   - RAID ストライプ・サイズに RAID 並列装置の数を乗算した値に等しい (または、この積の整数倍にする)。
   - エクステント・サイズの整数倍。

   この例の場合、プリフェッチ・サイズを 768 KB に設定できます。この値は、RAID ストライプ・サイズ (256 KB) に RAID 並列装置の数 (3) を乗算した値と等しくなります。またエクステント・サイズ (256 KB) の倍数でもあります。このプリフェッチ・サイズを選択すると、単一のプリフェッチがすべてのアレイのすべてのディスクで行われることになります。ワークロードに関係するのが主に順次スキャンであるためプリフェッチャーをより積極的に動作させるには、この値の倍数 (1536 KB (768 KB x 2) など) を代わりに使用できます。

4. DB2_USE_PAGE_CONTAINER_TAG レジストリー変数は設定しないでください。既に説明したように、表スペースを作成する場合、RAID ストライプ・サイズと同じか、その倍数のエクステント・サイズを指定してください。ただし、DB2_USE_PAGE_CONTAINER_TAG を ON に設定すると、1 ページのコンテナ

ー・タグが使用され、エクステントは RAID ストライプと同列にはなりません。その結果、入出力要求中に最適と思える数よりも多くの物理ディスクにアクセスする必要があるかもしれません。

表スペースのドロップ

表スペースをドロップすると、その表スペース内のすべてのデータは削除し、コンテナーを解放し、カタログ項目を除去し、そして、その表スペースの中に定義されたすべてのオブジェクトをドロップするか無効のマークを付けることになります。

表スペースをドロップすることによって、空の表スペース内のコンテナーを再使用することができますが、コンテナーを再使用する前に、 DROP TABLESPACE ステートメントをコミットしなければなりません。

ユーザー表スペースのドロップ

該当するシングル・ユーザー表スペース内の索引および LOB データを含むすべての表データが入ったユーザー表スペースをドロップできます。また、いくつかの表スペースにわたる表を持つようなユーザー表スペースもドロップできます。つまり、1 つの表スペースに表データ、別の表スペースに索引、3 番目の表スペースに LOB を持つような場合、単一のステートメントで これらの 3 つの表スペースすべてを同時にドロップする必要があります。複数の表スペースにまたがる表を含む表スペースは、一つのステートメントに含まれる必要があります。さもなければ、ドロップ要求は失敗します。

コマンド行を使用してユーザー表スペースをドロップするには、以下のように入力します。

```
DROP TABLESPACE <name>
```

以下の SQL ステートメントは、表スペース ACCOUNTING をドロップします。

```
DROP TABLESPACE ACCOUNTING
```

USER TEMPORARY 表スペースのドロップ

USER TEMPORARY 表スペースをドロップできるのは、その表スペースに現行で定義されている、宣言済み一時表がない場合に限ります。表スペースをドロップするときに、その表スペース内の宣言済み一時表のドロップが試みられることはまったくありません。

注: 宣言済み一時表を宣言しているアプリケーションがデータベースから切断されると、宣言済み一時表は暗黙的にドロップされます。

SYSTEM TEMPORARY 表スペースのドロップ

まず SYSTEM TEMPORARY 表スペースをもう 1 つ作成してからでなければ、 4 KB のページ・サイズを持つ SYSTEM TEMPORARY 表スペースをドロップすることはできません。データベースは常に少なくとも 1 つの 4 KB のページ・サイズを持つ SYSTEM TEMPORARY 表スペースを持っていなければならないため、新規 SYSTEM TEMPORARY 表スペースは、4 KB のページ・サイズを持っている必要があります。例えば、4 KB のページ・サイズを持つ単一 SYSTEM TEMPORARY 表スペースを持っていて、そこにコンテナーを追加しようとしており、その表スペースが SMS 表スペ

ースであるならば、まず適切なコンテナー数を持つ新しい 4 KB のページ・サイズの SYSTEM TEMPORARY 表スペースを追加してから、古い SYSTEM TEMPORARY 表スペースをドロップしなければなりません。(DMS を使用している場合には、表スペースをドロップして再作成することなく、コンテナーを追加することができます。)

デフォルトの表スペース・ページ・サイズは、データベースが作成されたときのページ・サイズ (デフォルトでは 4 KB ですが、8 KB、16 KB、または 32 KB も可能) です。

次は、SYSTEM TEMPORARY 表スペースを作成するためのステートメントです。

```
CREATE SYSTEM TEMPORARY TABLESPACE <name>
  MANAGED BY SYSTEM USING ('<directories>')
```

続いて、コマンド行を使用してシステム表スペースをドロップするには、以下のように入力します。

```
DROP TABLESPACE <name>
```

以下の SQL は、TEMPSPACE2 と呼ばれる新しい SYSTEM TEMPORARY 表スペースを作成します。

```
CREATE SYSTEM TEMPORARY TABLESPACE TEMPSPACE2
  MANAGED BY SYSTEM USING ('d:¥systemp2')
```

TEMPSPACE2 が作成されれば、以下のコマンドを使用して、元の SYSTEM TEMPORARY 表スペース TEMPSPACE1 をドロップすることができます。

```
DROP TABLESPACE TEMPSPACE1
```

第 10 章 スキーマ

スキーマ とは、名前を持つオブジェクトの集合のことです。これにより、それらのオブジェクトを論理的にグループ化できます。スキーマは、名前修飾子でもあります。これにより、複数のオブジェクトに対して同じ自然名を使用しながら、それらのオブジェクトに対するあいまい参照を防ぐことができます。

例えば、「INTERNAL」および「EXTERNAL」というスキーマ名によって、2 つの異なる SALES 表を識別することが容易になります (INTERNAL.SALES、EXTERNAL.SALES)。

スキーマによって、複数のアプリケーションがネーム・スペースの衝突を生じることなく、単一のデータベースにデータを保管できるようにもなります。

スキーマと *XML* スキーマ とは別個のものなので混同しないでください。後者は、XML 文書の構造を記述し、その内容を妥当性検査するための標準です。

表、ビュー、ニックネーム、トリガー、関数、パッケージ、および他のオブジェクトをスキーマに入れることができます。スキーマ自体が 1 つのデータベース・オブジェクトです。現行ユーザーを指定するか、またはスキーマ所有者と記録された指定の許可 ID を指定した CREATE SCHEMA ステートメントを使用して、スキーマは明示的に作成されます。また、ユーザーが IMPLICIT_SCHEMA 権限を持っている場合には、他のオブジェクトを作成する際に暗黙的に作成することもできます。

スキーマ名 は、 2 つの部分から成るオブジェクト名の高位の部分として使用されます。オブジェクトを作成する際にスキーマを使用して固有に修飾すると、オブジェクトはこのスキーマに割り当てられます。オブジェクトを作成する際にスキーマ名を指定しないと、デフォルトのスキーマ名 (CURRENT SCHEMA 特殊レジスターで指定されたもの) が使用されます。

例えば、DBADM 権限を有するユーザーが、ユーザー A に対して C と呼ばれるスキーマを作成するとします。

```
CREATE SCHEMA C AUTHORIZATION A
```

次にユーザー A は、以下のステートメントを出して、スキーマ C 内に X という名前の表を作成することができます (ただし、ユーザー A が CREATETAB データベース権限をもつことを前提とします)。

```
CREATE TABLE C.X (COL1 INT)
```

予約済みのスキーマ名があります。例えば、組み込み関数は SYSIBM スキーマに属し、プリインストールされたユーザー定義関数は SYSFUN スキーマに属します。

データベースが作成される場合に、それが RESTRICTIVE オプションを使用して作成されるのではない場合は、すべてのユーザーが IMPLICIT_SCHEMA 権限を持ちます。この権限を使用して、ユーザーは、まだ存在していないスキーマ名を持つオブジェクトを作成するときに、常に暗黙にスキーマを作成します。スキーマが暗黙的に作成されるときは、CREATEIN 特権が付与されます。この特権により、どのようなユーザーも、そのスキーマに他のオブジェクトを作成することができます。別

名、特殊タイプ、関数、およびトリガーなどのオブジェクトの作成能力は、暗黙的に作成されるスキーマにまで拡張されます。暗黙的に作成されるスキーマについてのデフォルトの特権には、旧バージョンとの後方互換性があります。

IMPLICIT_SCHEMA 権限が PUBLIC から取り消される場合、スキーマは、CREATE SCHEMA ステートメントを使用して明示的に作成されるか、または IMPLICIT_SCHEMA 権限が与えられているユーザー (例えば、 DBADM 権限のあるユーザー) によって暗黙的に作成されます。 PUBLIC から IMPLICIT_SCHEMA 権限を取り消すことは、スキーマ名の使用に対する制御が増す一方で、既存のアプリケーションがオブジェクトの作成を試みる時に許可エラーが生じる可能性があります。

スキーマには特権もあるので、スキーマ所有者がその特権を使用すれば、どのようなユーザーがスキーマ中のオブジェクトを作成、変更、コピー、およびドロップする権限をもつかを制御することができます。これにより、データベース内にあるオブジェクトのサブセットの操作を制御できます。当初、スキーマの所有者にはスキーマに関するこれらのすべての特権が与えられ、それらの特権を他のユーザーに付与することもできます。暗黙的に作成されたスキーマはシステムによって所有され、当初、そのようなスキーマにオブジェクトを作成する権限がすべてのユーザーに与えられます。 SYSADM または DBADM 権限を有するユーザーは、任意のスキーマでユーザーが保持する特権を変更することができます。したがって、任意のスキーマ (暗黙的に作成されたスキーマであっても) のオブジェクトを作成、変更、コピー、およびドロップするためのアクセスを制御することができます。

スキーマの設計

データを表に編成すれば、表や他の関連オブジェクトを一緒にグループ化する利点があります。これは、CREATE SCHEMA ステートメントを使用して、スキーマを定義することによって行うことができます。

スキーマについての情報は、接続するデータベースのシステム・カタログ表に保持されます。他のオブジェクトが作成されると、それらのオブジェクトを作成するスキーマ内に置くことができますが、1 つのオブジェクトは 1 つのスキーマにしか存在できないことに注意してください。

スキーマはディレクトリーと比較できます。現行スキーマは現行ディレクトリーに対応します。この例えでいうと、SET SCHEMA は change directory コマンドに相当します。

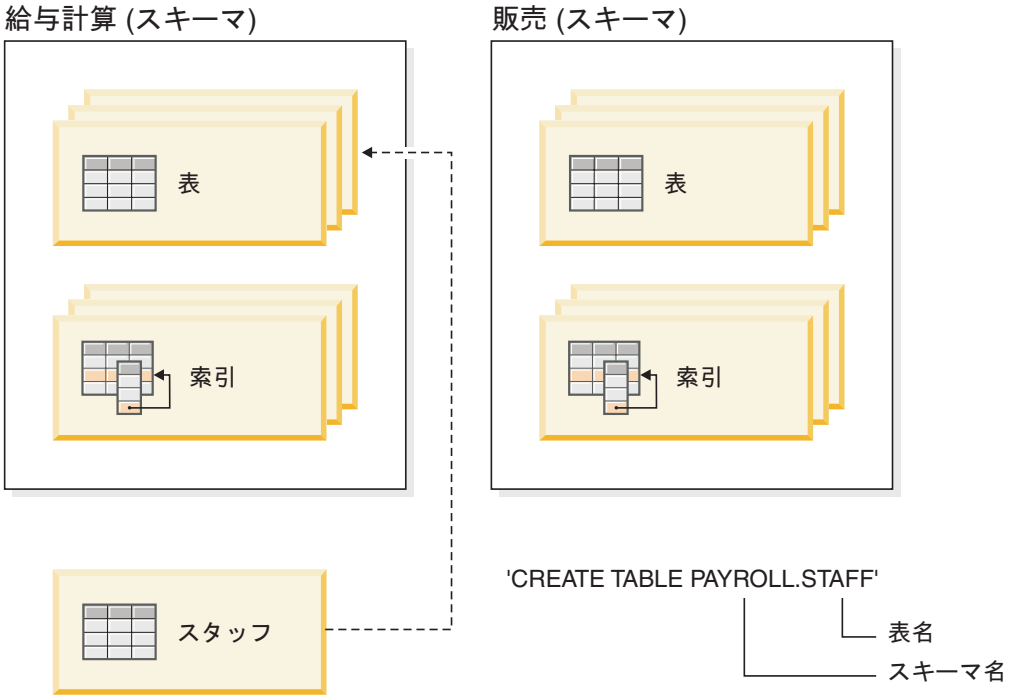
重要: 許可 ID とスキーマの間には、デフォルトの CURRENT SCHEMA 設定 (以下で説明) を除いて関係はないことを理解しておくのは重要です。

データベースおよび表を設計するときに、システム内のスキーマ (その名前および各スキーマに関連付けられるオブジェクトを含む) を考慮する必要もあります。

データベース内のほとんどのオブジェクトは、2 つの部分から成る固有名が割り当てられます。最初 (左端) の部分は修飾子またはスキーマと呼ばれ、2 番目 (右端) の部分は単純 (または非修飾) 名と呼ばれます。構文的には、これらの 2 つの部分はピリオドで区切られた単一の文字ストリングとして連結されます。スキーマ名によって修飾できるオブジェクト (表、索引、ビュー、ユーザー定義のデータ・タイ

プ、ユーザー定義関数、ニックネーム、パッケージ、またはトリガーなど) が初めて作成されるとき、それはその名前の修飾子に基づいて特定のスキーマに割り当てられます。

例えば、次の図は、表作成プロセス中に表が特定のスキーマに割り当てられる方法を示しています。



また、ユーザーに正しい権限と指示を与えるために、スキーマ・アクセスが付与される方法を理解しておく必要もあります。

スキーマ名

新規スキーマを作成する場合、名前はカタログ内ですでに記述されているスキーマ名を表してはなりません。また、名前を ″SYS″ で始めることはできません。その他の制限および推奨事項については、 244 ページの『スキーマ名の制限および推奨事項』を参照してください。

スキーマへのアクセス

スキーマはデータベース内での固有性を強制するために使用されるため、スキーマ内のオブジェクトに対する非修飾アクセスは許可されません。 2 人のユーザーが 2 つの表 (または他のオブジェクト) を同時に作成できるかどうかを考慮する際に、このことが明らかになります。スキーマを使用して固有性を強制しないと、 3 人目のユーザーが表を照会しようとする場合にあいまいさが生じます。もっと詳しく制限されていなければ、どの表を使用すべきかを判別できません。

CREATE SCHEMA ステートメントの一部として作成されたどのオブジェクトの定義者も、スキーマの所有者になります。この所有者は、他のユーザーに対して、スキーマ特権の GRANT および REVOKE を行うことができます。

ユーザーが SYSADM 権限または DBADM 権限を持っている場合、そのユーザーは、任意の有効な名前でスキーマを作成することができます。データベースが作成されるときに、IMPLICIT_SCHEMA 権限が PUBLIC (つまり、すべてのユーザー) に付与されます。

ユーザーが IMPLICIT_SCHEMA または DBADM 権限を持っていない場合、ユーザー自身の許可 ID と同じ名前のスキーマのみを作成することができます。

デフォルト・スキーマ

スキーマまたは修飾子が作成されるオブジェクトの名前の一部として指定されていない場合、そのオブジェクトは CURRENT SCHEMA 特殊レジスターで示されるとおり、デフォルト・スキーマに割り当てられます。この特殊レジスターのデフォルト値は、セッション許可 ID の値です。

デフォルト・スキーマは、動的ステートメント内の非修飾オブジェクト参照によって必要とされます。 CURRENT SCHEMA 特殊レジスターをデフォルトにしたいスキーマに設定することにより、特定の DB2 接続用のデフォルト・スキーマを設定できます。この特殊レジスターを設定するために、指定された許可は必要ありません。そのため、どのユーザーでも CURRENT SCHEMA を設定できます。

SET SCHEMA ステートメントの構文は次のとおりです。

```
SET SCHEMA = <schema-name>
```

このステートメントは対話式に発行するか、またはアプリケーション内から発行することができます。 CURRENT SCHEMA 特殊レジスターの初期値は、現行セッション・ユーザーの許可 ID と同じになります。詳しくは、SET SCHEMA ステートメントを参照してください。

注:

- その他の方法でも、接続時にデフォルト・スキーマを設定できます。例えば、CLI/ODBC アプリケーションの場合は cli.ini ファイルを使用し、JDBC アプリケーション・プログラミング・インターフェースの場合は接続プロパティーを使用します。
- デフォルト・スキーマのレコードはシステム・カタログ内に作成されませんが、スキーマまたは修飾子が作成されるオブジェクトの名前の一部として指定されていない場合にはデータベース・マネージャーが (CURRENT SCHEMA 特殊レジスターから) 取得できる値としてのみ存在します。

暗黙作成

IMPLICIT_SCHEMA 権限がある場合には、スキーマを暗黙的に作成できます。この権限を使用して、すでに存在していないスキーマ名を持つオブジェクトを作成するときには常に暗黙にスキーマを作成できます。オブジェクトを作成するユーザーが IMPLICIT_SCHEMA 権限を保持していれば、スキーマ内のデータ・オブジェクトが初めて作成されるときにスキーマが暗黙的に作成されることがしばしばあります。

明示作成

また、CREATE SCHEMA および DROP SCHEMA ステートメントをコマンド行またはアプリケーション・プログラムから実行することにより、スキ

ーマを明示的に作成およびドロップすることができます。詳しくは、CREATE SCHEMA および DROP SCHEMA ステートメントを参照してください。

スキーマ別の表およびビューの別名

表名またはビュー名に対する制限の一部としてスキーマ名を入力しなくても、別のユーザーが表またはビューにアクセスできるようにするには、そのユーザーについて別名を設定する必要があります。別名の定義では、完全修飾表名またはビュー名 (ユーザーのスキーマを含む) を定義します。そうすれば、ユーザーは別名を使用して照会するだけで済みます。別名は、ユーザーのスキーマによって別名定義の一部として完全修飾されています。

スキーマ別のオブジェクトのグループ化

データベース・オブジェクトは、1 つの ID で構成されているか、 2 つの ID で構成されるスキーマ*修飾オブジェクト* です。スキーマ修飾オブジェクトのスキーマまたは高位部分は、データベースの中のオブジェクトを分類またはグループ化するための手段を提供します。表、ビュー、別名、特殊タイプ、関数、索引、パッケージ、またはトリガーが作成されると、それがスキーマに割り当てられます。この割り当ては、明示的または暗黙的のいずれかで行われます。

ステートメント中のオブジェクトを参照するときに 2 部から成るオブジェクト名の高位部分を使用する場合は、スキーマを明示的に使用することになります。例えば、USER A がスキーマ C の CREATE TABLE ステートメントを次のように発行します。

```
CREATE TABLE C.X (COL1 INT)
```

2 部から成るオブジェクト名の高位部分を使用しない場合は、スキーマを暗黙的に使用することになります。スキーマを暗黙的に使用すると、オブジェクト名の高位部分を構成するのに使用されるスキーマ名を識別するために、 CURRENT SCHEMA 特殊レジスターが使用されます。 CURRENT SCHEMA の初期値は、現行セッション・ユーザーの許可 ID です。これを現行セッション中に変更する場合は、 SET SCHEMA ステートメントを使用して特殊レジスターを別のスキーマ名に設定することができます。

データベース作成時、一部のオブジェクトは、特定のスキーマ内に作成されてシステム・カタログ表に保管されます。

オブジェクトの作成先のスキーマを明示的に指定する必要はありません。このように指定しない場合は、ステートメントの許可 ID が使用されます。例えば、次の CREATE TABLE ステートメントの場合、デフォルトのスキーマ名は現在ログオンしている許可 ID (すなわち CURRENT SCHEMA 特殊レジスター値) になります。

```
CREATE TABLE X (COL1 INT)
```

動的 SQL および XQuery ステートメントでは、通常、修飾なしのオブジェクト名参照を暗黙的に修飾するために CURRENT SCHEMA 特殊レジスター値を使用します。

独自のオブジェクトを作成する前に、それらをデフォルトのスキーマの中に作成するか、または論理的にオブジェクトをグループ化する別のスキーマを使用するかを

検討しなければなりません。共用されるオブジェクトを作成している場合は、別のスキーマ名を使用すると、非常に便利です。

スキーマ名の制限および推奨事項

スキーマに名前を付けるときに認識しておくべき制限と推奨事項がいくつかあります。

- ユーザー定義タイプ (UDT) には、SQL および XML の制限値にリストされたスキーマの長さを超えるスキーマ名を指定することはできません。
- 次のスキーマ名は予約語なので、使用できません。
  SYSCAT、SYSFUN、SYSIBM、SYSSTAT、SYSPROC。
- マイグレーションの問題を未然に防ぐために、SYS で始まるスキーマ名を使用しないでください。データベース・マネージャーでは SYS で始まるスキーマ名を使用して、トリガー、ユーザー定義タイプ、またはユーザー定義関数を作成できません。
- SESSION をスキーマ名として使用しないようお奨めします。宣言される一時表は、SESSION によって修飾しなければなりません。したがって、アプリケーションに永続表と同じ名前の一時表を宣言させることができますが、その場合、アプリケーション・ロジックが複雑になりすぎる可能性があります。宣言される一時表を扱う場合以外は、スキーマ SESSION の使用を避けてください。

スキーマの作成

スキーマを使用すれば、複数のオブジェクトの作成時にそれらのオブジェクトをグループ化できます。各オブジェクトはただ 1 つのスキーマにだけ属することができます。スキーマを作成するには CREATE SCHEMA ステートメントを使用します。スキーマについての情報は、接続先のデータベースのシステム・カタログ表に保持されます。

スキーマを作成して、オプションで別のユーザーをスキーマ所有者に設定するには、SYSADM または DBADM 権限が必要です。このどちらの権限も保持していない場合であっても、ユーザーの許可 ID を使ってスキーマを作成することは可能です。 CREATE SCHEMA ステートメントによって作成されるすべてのオブジェクトの定義者は、スキーマ所有者になります。この所有者は、他のユーザーに対して、スキーマ特権の GRANT および REVOKE を行うことができます。

コマンド行からスキーマを作成するには、次のステートメントを入力します。

```
CREATE SCHEMA <schema-name> [ AUTHORIZATION <schema-owner-name> ]
```

ここで、<schema-name> はスキーマの名前です。この名前は、カタログにすでに記録されているスキーマとは異なる固有の名前でなければなりません。また、SYS で始まる名前は指定できません。オプションの AUTHORIZATION 節を指定すると、<schema-owner-name> がスキーマ所有者になります。この節を指定しない場合、このステートメントを発行した許可 ID がスキーマ所有者になります。

詳しくは、CREATE SCHEMA ステートメントを参照してください。『『スキーマ名の制限および推奨事項』』もご覧ください。

スキーマのコピー

db2move ユーティリティーおよび ADMIN_COPY_SCHEMA プロシージャーにより、データベース・スキーマのコピーを迅速に作成できます。モデル・スキーマを確立すると、新しいバージョンを作成するためのテンプレートとしてそれを使用できます。

同じデータベース内の単一のスキーマをコピーするには、ADMIN_COPY_SCHEMA プロシージャーを使用します。単一スキーマまたは複数のスキーマをソース・データベースからターゲット・データベースへコピーするには、`-co COPY` アクションを指定して db2move ユーティリティーを使用します。ソース・スキーマからのほとんどのデータベース・オブジェクトは、新しいスキーマの下のターゲット・データベースにコピーされます。

トラブルシューティングのヒント

ADMIN_COPY_SCHEMA プロシージャーと db2move ユーティリティーはどちらも LOAD コマンドを呼び出します。ロードの処理中、データベース・ターゲット・オブジェクトのある表スペースはバックアップ・ペンディング状態になります。

ADMIN_COPY_SCHEMA プロシージャー

上記の注で説明したとおり、プロシージャーに COPYNO オプションを指定して使用すると、ターゲット・オブジェクトがある表スペースをバックアップ・ペンディング状態にします。表スペースの SET INTEGRITY ペンディング状態を解除するために、このプロシージャーは SET INTEGRITY ステートメントを発行します。ターゲット表オブジェクトに参照制約が定義されている状態では、ターゲット表も SET INTEGRITY ペンディング状態になります。表スペースはすでにバックアップ・ペンディング状態なので、ADMIN_COPY_SCHEMA プロシージャーが SET INTEGRITY ステートメントを発行する試みは失敗します。

この状態を解決するには、BACKUP DATABASE コマンドを実行して、影響を受ける表スペースのバックアップ・ペンディング状態を解除します。次いで、このプロシージャーによって生成されたエラー表の **Statement_text** 列を参照し、SET INTEGRITY ペンディング状態にある表のリストを見つけます。次に、リストされている各表に SET INTEGRITY ステートメントを発行し、各表の SET INTEGRITY ペンディング状態を解除します。

db2move ユーティリティー

このユーティリティーは、以下のタイプを除き、許容されるスキーマ・オブジェクトをすべてコピーしようとします。

- 表階層
- ステージング表 (複数パーティション・データベース環境でのロード・ユーティリティーではサポートされません)
- jars (Java™ ルーチン・アーカイブ)
- ニックネーム
- パッケージ
- ビュー階層
- オブジェクト特権 (新規オブジェクトはすべてデフォルト許可で作成されます。)

- 統計 (新規オブジェクトに統計情報は含まれません。)
- 索引拡張 (ユーザー定義構造化タイプに関連)
- ユーザー定義構造化タイプおよびそのトランスフォーム関数

サポートされないタイプに関するエラー

サポートされないタイプのいずれかのオブジェクトがソース・スキーマで検出されると、サポートされないオブジェクト・タイプが検出されたことを示すエントリーがエラー・ファイルに書き込まれます。 COPY 操作は引き続き正常に行われます。ログに記録されたエントリーは、この操作ではコピーされないオブジェクトについて伝えるためのものです。

スキーマを伴わないオブジェクト

表スペースなどの、スキーマを伴わないオブジェクト、およびイベント・モニターは、スキーマのコピー操作時には操作されません。これらは、スキーマのコピー操作が呼び出される前に、ターゲット・データベース上で作成する必要があります。

複製された表

複製された表をコピーする場合、表の新規コピーはレプリケーションには使用できません。表は通常表として再作成されます。

異なるインスタンス

ソース・データベースは、ターゲット・データベースと同じインスタンス内にない場合は、カタログされる必要があります。

SCHEMA_MAP オプション

SCHEMA_MAP オプションを使用して、ターゲット・データベース上で異なるスキーマ名を指定する場合、元のスキーマ名を新しいスキーマ名に置き換えるために、スキーマのコピー操作はオブジェクト定義ステートメントの最小限の構文解析のみを実行します。例えば、SQL プロシージャーの内容の中に表示されるオリジナル・スキーマのインスタンスは新しいスキーマ名に置き換えられません。したがって、スキーマのコピー操作はそれらのオブジェクトを再作成できない場合があります。コピー操作の完了後に、エラー・ファイル内の DDL を使用して、それらの失敗したオブジェクトを手動で再作成することができます。

オブジェクト間の相互依存

スキーマのコピー操作は、それらのオブジェクト間の相互依存性を満たす順序でオブジェクトを再作成しようとします。例えば、表 T1 にユーザー定義関数 U1 を参照する列が含まれる場合、T1 を再作成する前に U1 を再作成します。ただし、プロシージャーの従属情報がカタログ内で前もって使用可能なわけではないため、プロシージャーを再作成する際に、スキーマのコピー操作では最初にすべてのプロシージャーを再作成し、それから失敗したものを再作成しようとします (それらのプロシージャーが前の試行時に正常に作成されたプロシージャーに依存していれば、それ以降の試行時には正常に再作成されると想定しています)。それ以降の試行時に 1 つ以上のプロシージャーを正常に再作成できる限り、この操作は失敗したプロシージャーを引き続き再作成しようとします。プロシージャーの再作成試行時に、エラー (および DDL) がエラー・ファイルに毎回記録されます。エラー・ファイルに同じプロシージャーの項目が多数表示されることがありますが、それらのプロシージャーはそれ以降の試行時に正常に再作成されている可能性があり

ますスキーマのコピー操作の完了時に SYSCAT.PROCEDURES 表を照会して、エラー・ファイルにリストされているそれらのプロシージャーが正常に再作成されたかどうか判断する必要があります。

詳細については、ADMIN_COPY_SCHEMA プロシージャーおよび db2move ユーティリティーを参照してください。

ADMIN_COPY_SCHEMA プロシージャーを使用したスキーマ・コピーの例

同じデータベース内の単一のスキーマをコピーするには、以下のようにして ADMIN_COPY SCHEMA プロシージャーを使用します。

```
DB2 "SELECT SUBSTR(OBJECT_SCHEMA,1, 8)
  AS OBJECT_SCHEMA, SUBSTR(OBJECT_NAME,1, 15)
  AS OBJECT_NAME, SQLCODE, SQLSTATE, ERROR_TIMESTAMP, SUBSTR(DIAGTEXT,1, 80)
  AS DIAGTEXT, SUBSTR(STATEMENT,1, 80)
  AS STATEMENT FROM COPYERRSCH.COPYERRTAB"

CALL SYSPROC.ADMIN_COPY_SCHEMA('SOURCE_SCHEMA', 'TARGET_SCHEMA',
  'COPY', NULL, 'SOURCETS1 , SOURCETS2', 'TARGETTS1, TARGETTS2,
  SYS_ANY', 'ERRORSCHEMA', 'ERRORNAME')
```

この SELECT ステートメントの出力は次のようになります。

```
OBJECT_SCHEMA OBJECT_NAME     SQLCODE     SQLSTATE ERROR_TIMESTAMP
------------- --------------- ----------- -------- --------------------------
SALES         EXPLAIN_STREAM         -290 55039    2006-03-18-03.22.34.810346

DIAGTEXT
--------------------------------------------------------------------------------
[IBM][CLI Driver][DB2/LINUXX8664] SQL0290N  Table space access is not allowed.

STATEMENT
--------------------------------------------------------------------------------
set integrity for "SALES  "."ADVISE_INDEX" , "SALES"."ADVISE_MQT" , "SALES"."

1 record(s) selected.
```

db2move ユーティリティーを使用したスキーマ・コピーの例

1 つまたは複数のスキーマをソース・データベースからターゲット・データベースにコピーするには、-co COPY アクションを指定して db2move ユーティリティーを使用します。モデル・スキーマを確立すると、新しいバージョンを作成するためのテンプレートとしてそれを使用できます。

例 1: -c COPY オプションの使用

-co COPY オプションを指定した以下の db2move の例は、スキーマ BAR をサンプル・データベースからターゲット・データベースにコピーして、それを FOO に名前変更します。

```
db2move sample COPY -sn BAR -co target_db  target schema_map
  "((BAR,FOO))" -u userid -p password
```

新規の (ターゲット) スキーマ・オブジェクトは、ソース・スキーマのオブジェクトと同じオブジェクト名を使用して作成されますが、修飾子はターゲット・スキーマのものが使用されます。表のコピーを作成する際には、ソー

ス表のデータを共にコピーすることも、しないこともできます。ソース・データベースおよびターゲット・データベースは、別々のシステムに存在できます。

例 2: COPY 操作中に表スペース名のマッピングを指定する

以下の例は、db2move COPY 操作時にソース・システムからの表スペースの代わりに使用される、特定の表スペース名マッピングを指定する方法を示しています。 SYS_ANY キーワードを指定して、ターゲット表スペースを、デフォルトの表スペース選択アルゴリズムを使用して選択することを指示できます。この場合、db2move ユーティリティーは、ターゲットとして使用可能な任意の表スペースを選択します。

```
db2move sample COPY -sn BAR -co target_db  target schema_map
  "((BAR,FOO))" tablespace_map "(SYS_ANY)" -u userid -p password
```

SYS_ANY キーワードは、すべての表スペースに使用できます。または、一部の表スペースに固有のマッピングを指定し、残りにはデフォルトの表スペース選択アルゴリズムを指定できます。

```
db2move sample COPY -sn BAR -co target_db  target schema_map "
  ((BAR,FOO))" tablespace_map "((TS1, TS2),(TS3, TS4), SYS_ANY)"
  -u userid -p password
```

これは、表スペース TS1 が TS2 にマップされ、TS3 が TS4 にマップされ、残りの表スペースがデフォルトの表スペース選択アルゴリズムを使用することを示しています。

例 3: COPY 操作後のオブジェクト所有者の変更

正常に COPY を実行した後に、ターゲット・スキーマで作成された新しい各オブジェクトの所有者を変更することができます。ターゲット・オブジェクトのデフォルト所有者は接続ユーザーです。以下のようなオプションを指定すると、新しい所有者に所有権が移転します。

```
db2move sample COPY -sn BAR -co target_db  target schema_map
  "((BAR,FOO))" tablespace_map "(SYS_ANY)" owner jrichards
  -u userid -p password
```

ターゲット・オブジェクトの新規所有者は jrichards です。

ソース・スキーマとターゲット・スキーマが別々のシステムにある場合、ターゲット・システム側で db2move ユーティリティーを呼び出す必要があります。あるデータベースから別のデータベースにスキーマをコピーする場合、このアクションを行うには、ソース・データベースからコピーされるスキーマ名のリスト (コンマで区切られたもの) およびターゲット・データベース名が必要です。

スキーマをコピーするには、OS のコマンド・プロンプトから db2move を次のように発行します。

```
db2move <dbname> COPY -co <COPY- options>
 -u <userid> -p <password>
```

スキーマのコピー操作が失敗した場合の再開方法

db2move COPY 操作時に発生したエラーは、コピーするオブジェクトのタイプまたは COPY 操作に失敗したフェーズ (つまり、オブジェクトの再作成フェーズ、またはデータのロード・フェーズ) に応じてさまざまな方法で処理できます。

db2move ユーティリティーは、メッセージ・ファイルおよびエラー・ファイルを使用して、エラーやメッセージをユーザーに報告します。スキーマのコピー操作では、COPYSCHEMA_<timestamp>.MSG メッセージ・ファイル、および COPYSCHEMA_<timestamp>.err エラー・ファイルが使用されます。これらのファイルは、現行作業ディレクトリーに作成されます。現在時刻がファイル名に付加されるため、ファイルの固有性が保証されます。これらのメッセージ・ファイルおよびエラー・ファイルが必要ではなくなった場合、それらを削除するかどうかはユーザーが決定することになります。

注: 複数の db2move インスタンスを同時に実行することができます。COPY オプションは SQLCODES を戻しません。これは db2move の動作と一貫性のあるものです。

オブジェクト・タイプ

コピーするオブジェクトのタイプは、物理オブジェクトとビジネス・オブジェクトという 2 つのタイプのいずれかに分類できます。

物理オブジェクトとは、表、索引、およびユーザー定義の構造化タイプなど、コンテナー内に物理的に存在するオブジェクトのことです。ビジネス・オブジェクトとは、ビュー、ユーザー定義構造化タイプ (UDT)、および別名などの、コンテナー内に常駐しないカタログ・オブジェクトのことです。

物理オブジェクトの再作成時にエラーが発生すると、このユーティリティーはロールバックを行いますが、論理オブジェクトの再作成時のエラーではロールバックは行いません。

スキーマのコピー操作の再開

ロード操作の失敗の原因となった問題 (エラー・ファイルで説明) を解決した後、以下の構文に示されているように、コピーしてデータを追加する表を指定するために -tf オプション (LOADTABLE.err ファイル名を渡す) を使用して db2move -COPY コマンドを再発行することができます。

```
db2move sourcedb COPY -tf LOADTABLE.err -co TARGET_DB mytarget_db
  -mode load_only
```

以下の構文で示されているように、-tn オプションを使用して表の名前を手動で入力することもできます。

```
db2move sourcedb COPY -tn "FOO"."TABLE1","FOO 1"."TAB 444",
  -co TARGET_DB mytarget_db -mode load_only
```

注: load_only モードでは、-tn または -tf オプションを使って少なくとも 1 つの表を入力する必要があります。

例

db2move COPY スキーマ操作時に発生したエラーは、コピーするオブジェクトのタイプまたは COPY 操作に失敗したフェーズに応じてさまざまな方法で処理できます。

db2move ユーティリティーは、スキーマのコピーのエラーおよびメッセージを、次のメッセージ・ファイルおよびエラー・ファイルに報告します。

- COPYSCHEMA <timestamp>.MSG メッセージ・ファイル
- COPYSCHEMA_<timestamp>.err エラー・ファイル

これらのファイルは、現行作業ディレクトリーに作成されます。現在時刻がファイル名に付加されるため、ファイルの固有性が保証されます。これらのメッセージ・ファイルおよびエラー・ファイルが必要ではなくなった場合、それらを削除する必要があります。

注: 複数の db2move インスタンスを同時に実行することができます。COPY オプションは SQLCODES を戻しません。これは db2move の動作と一貫性のあるものです。

例 1: 物理オブジェクトに関連したスキーマのコピー・エラー

ターゲット・データベース上での物理オブジェクトの再作成時に発生した失敗は、エラー・ファイル COPYSCHEMA_<timestamp>.err に書き込まれます。エラー・ファイルには、失敗した各オブジェクトについて、オブジェクト名、オブジェクト・タイプ、DDL テキスト、タイム・スタンプ、およびストリング形式の SQLCA (SQLCA フィールド名、続いてそのデータ値) などの情報が含まれています。

COPYSCHEMA_<timestamp>.err エラー・ファイルの出力例は以下のとおりです。

```
1. schema: FOO.T1
Type:      TABLE
Error Msg: SQL0104N An unexpected token 'FOO.T1'...
Timestamp: 2005-05-18-14.08.35.65
DDL:       create view FOO.v1

2. schema:  FOO.T3
Type:      TABLE
Error Msg: SQL0204N FOO.V1 is an undefined name.
Timestamp: 2005-05-18-14.08.35.68
DDL:       create table FOO.T3
```

物理オブジェクトの作成に関するエラーが再作成フェーズの最後で、かつ、ロード・フェーズを試行する前に書き込まれた場合、db2move ユーティリティーは失敗し、エラーが戻されます。ターゲット・データベースでのオブジェクトの作成はすべてロールバックされ、内部で作成された表はすべてソース・データベースにおいてクリーンアップされます。ロールバックは、最初の失敗の後ではなく、各オブジェクトの再作成を試みた後に再作成フェーズの最後で発生します。これは、起こりうるすべてのエラーをエラー・ファイルにまとめるためです。これによって、db2move 操作を再開する前に問題を修正する機会が与えられます。失敗がなければ、エラー・ファイルは削除されます。

例 2: ビジネス・オブジェクトに関連したスキーマのコピー・エラー

ターゲット・データベースでビジネス・オブジェクトの再作成中にエラーが発生しても、db2move ユーティリティーが失敗することはありません。代わりに、それらの失敗は COPYSCHEMA_<timestamp>.err エラー・ファイルに書き込まれます。db2move ユーティリティーの完了時に、失敗について調べ、問題に対処し、作成に失敗したオブジェクトを手動で再作成することができます (便宜のために、DDL がエラー・ファイル内で提供されます)。

db2move が、ロード・ユーティリティーを使用して表データの再挿入を試行している際にエラーが発生した場合でも、db2move ユーティリティーは失敗しません。代わりに、失敗に関する一般情報は COPYSCHEMA_<timestamp>.err ファイル (オブジェクト名、オブジェクト・タイプ、DDL テキスト、タイム・スタンプ、SQLCA など) に書き込まれ、表の完全修飾名は別のファイル LOADTABLE_<timestamp>.err に書き込まれます。各表は、db2move -tf オプションの形式に合わせて 1 行ごとにリストされ、以下のようになります。

```
"FOO"."TABLE1"
"FOO 1"."TAB 444"
```

例 3: 他のタイプの db2move の失敗

メモリー・エラーやファイル・システム・エラーなどの内部操作によって、db2move ユーティリティーが失敗することがあります。

DDL の再作成フェーズ中に内部操作の失敗が発生した場合、正常に作成されたオブジェクトはすべてターゲット・スキーマからロールバックされ、内部で作成された表 (DMT 表や db2look 表など) はすべて、ソース・データベース上でクリーンアップされます。

ロード・フェーズ中に内部操作の失敗が発生した場合は、正常に作成されたオブジェクトはすべてターゲット・スキーマに残ります。ロード操作時に失敗が発生したすべての表、およびまだロードされていないすべての表は、LOADTABLE.err エラー・ファイルに書き込まれます。その後、例 2 で説明されているように、LOADTABLE.err を使用して db2move COPY コマンドを発行することができます。db2move ユーティリティーが異常終了 (例えば、システム・クラッシュ、ユーティリティー・トラップ、ユーティリティーの強制終了など) した場合、ロードする必要がある表についての情報は失われます。この場合、ADMIN_DROP_SCHEMA プロシージャーを使用してターゲット・スキーマをドロップしてから、db2move COPY コマンドを再発行することができます。

スキーマのコピー操作の試行時に発生するエラーの種類にかかわらず、どの場合でも、ADMIN_DROP_SCHEMA プロシージャーを使用してターゲット・スキーマをドロップし、db2move COPY コマンドを再発行することができます。

スキーマのドロップ

スキーマをドロップする前に、そのスキーマの中にあったオブジェクトをすべてドロップするか、別のスキーマに移動しなければなりません。 DROP ステートメントを試みているときには、スキーマ名がカタログの中になければなりません。そうでない場合、エラーが戻されます。

コマンド行を使用してスキーマをドロップするには、以下のように入力します。

```
DROP SCHEMA <name> RESTRICT
```

以下の例では、″joeschma″ というスキーマがドロップされます。

```
DROP SCHEMA joeschma RESTRICT
```

RESTRICT キーワードは、データベースから削除するよう指定されたスキーマにはオブジェクトを定義できないという規則を強制します。このキーワードは必ず指定する必要があります。

第 3 部 データベース・オブジェクト

論理データベースの設計は、データベース・オブジェクトの定義から成ります。

DB2 データベース内には以下のデータベース・オブジェクトを作成することができます。

- 表
- 制約
- 索引
- トリガー
- シーケンス
- ビュー

これらのデータベース・オブジェクトは、グラフィカル・ユーザー・インターフェースを使用するか、またはステートメントを明示的に実行することによって作成できます。これらのデータベース・オブジェクトの作成に使用されるステートメントはデータ定義言語 (DDL) ステートメントと呼ばれ、一般的には CREATE または ALTER というキーワードが接頭部に付きます。

各データベース・オブジェクトが提供するフィーチャーや機能を理解することは、時間とともに起こる拡大や成長に対応するための柔軟性を維持しつつ、現行ビジネスのデータ・ストレージの必要を満たす優れたデータベース設計をインプリメントするために重要です。

第 11 章 表

表とは、データベース・マネージャーによって維持される論理構造です。表は列と行で構成されます。

列と行の交点すべてには、*値* と呼ばれる特定のデータ項目があります。*列* は、同一のタイプまたはそのいずれかのサブタイプの値の集合です。*行* は、*n* 番目の値が、表の *n* 番目の列の値であるような、一続きに配置された値です。

アプリケーション・プログラムでは、行が表に登録された順序を判別できますが、行の実際の順序はデータベース・マネージャーによって判別され、通常制御することはできません。マルチディメンション・クラスタリング (MDC) では、行間の実際の順序ではなく、ある種のクラスタリング (群/集合) の感覚が提供されます。

表のタイプ

環境に応じて、データを保管するために DB2 データベースに 1 つ以上の表を作成する必要があります。表を作成するときは、表の各列に内容のタイプを指定します。また、ビジネス・ルールを施行するために、その他の特性、例えば主キー、チェック制約なども定義します。さらに、表を作成するときは、要件に最も合ったタイプを検討する必要もあります。

(宣言済み) グローバル一時表を除く、以下のタイプの表のすべてが CREATE TABLE ステートメントを使用して作成できます。

基本表 このタイプの表では永続データが保持されます。

通常表 このタイプの表はヒープとしてインプリメントされます。索引付き通常表は、「汎用の」表です。

付加モードの表
このタイプの表は、主に INSERT 用に最適化された通常表です。通常表は、ALTER TABLE ステートメントを使用して付加モードになります。付加モードは、特定の索引にクラスタリングすることが重要ではなく、挿入率が高く、また表に対する削除がそれほど多くないか、または全くない表に最適です。

結果表 このタイプの表は、照会の要求に応じるためにデータベース・マネージャーが 1 つ以上の表から選択または生成した行の集まりで構成されます。

サマリー表
このタイプの表は、表の中のデータの判別にも使われる照会によって定義される表です。サマリー表を使って、照会のパフォーマンスを向上させることができます。照会の一部はサマリー表を使って解決できるとデータベース・マネージャーが判断した場合、データベース・マネージャーは、サマリー表を使用するように照会を書き換えることができます。この決定は、CURRENT REFRESH AGE および CURRENT QUERY OPTIMIZATION 特殊レジスターなどの、データベース構成の設定値に基づいてなされます。

型付き表

表では、列ごとにデータ・タイプを個別に定義する (つまり、型をユーザー定義の構造化タイプの属性に基づいたものにする) ことができます。このような表は、*型付き表* と呼ばれます。ユーザー定義の構造化タイプは、タイプ階層の一部にすることができます。*サブタイプ* は、*スーパータイプ* から属性を継承します。同様に、型付き表は表階層の一部にすることができます。*副表* は、*スーパー表* から列を継承します。*サブタイプ* という用語は、タイプ階層において 1 つのユーザー定義の構造化タイプおよびその下にあるすべてのユーザー定義の構造化タイプを指して用いられることに注意してください。構造化タイプ T の*厳密な意味でのサブタイプ* とは、タイプ階層で T の下にある構造化タイプのことです。同様に、*副表* という用語は、表階層において 1 つの型付き表およびその下にあるすべての型付き表を指して用いられます。表 T の*厳密な意味での副表* とは、表階層において T の下にある表のことです。

(宣言済み) グローバル一時表

このタイプの表はユーザー定義の一時表とも呼ばれ、データベース内のデータを扱うアプリケーションによって使用されます。データの操作の結果は、一時的に表に保管する必要があります。 USER TEMPORARY 表スペースは、グローバル一時表が作成される前に存在していなければなりません。

注: グローバル一時表の記述は、システム・カタログには現れません。したがって、この表を他のアプリケーションのために保持したり、他のアプリケーションと共用したりすることはできません。この表を使用するアプリケーションが終了したりデータベースから切断されたりすると、表の中のデータはすべて削除され、表は暗黙的にドロップされます。

グローバル一時表は、以下のものをサポートしません。

- LOB タイプの列 (または LOB に基づく特殊タイプの列)
- ユーザー定義タイプの列
- LONG VARCHAR 列
- XML 列

このタイプの表は、DECLARE GLOBAL TEMPORARY TABLE ステートメントで作成され、1 つのアプリケーションのために一時データを保持するときに使います。この表は、アプリケーションがデータベースから切断されるときに、暗黙的にドロップされます。

マルチディメンション・クラスタリング (MDC) 表

このタイプの表は、複数のキーまたはディメンションで同時に物理的なクラスタリングが行われる表としてインプリメントされます。 MDC 表は、データウェアハウジングや大規模データベース環境で使用されます。通常表における索引のクラスタリングは、データの 1 ディメンションだけのクラスタリングをサポートします。 MDC 表では、複数のディメンション間でのデータ・クラスタリングを利用できます。

注: MDC は一種の表ですが、パーティション表と共存させることができ、またパーティション表そのものにもできます。つまり、この表は他のタイプの表と相互に排他的ではありません。また、パーティション表は APPEND 表にすることもできます。他の組み合わせ (例えば MDC と

APPEND、RCT と他の表、あるいはその他の組み合わせ) は許可されません。また MDC では、複合ディメンション内での*保証された*クラスタリング が提供されます。なお、クラスタリング索引を持つ通常表では、クラスタリングがデータベース・マネージャーによって試みられますが保証はされず、通常時間の経過とともに劣化します。

範囲クラスター表 (RCT)
このタイプの表は、データの順次クラスターとしてインプリメントされ、高速かつ直接のアクセスを可能にします。表内の各レコードは事前定義されたレコード ID (RID) を持ちます。この ID は、表の中でレコードを探索するために内部で使用されるものです。 RTC 表は、表内の 1 つ以上の列でデータが密にクラスタリングされている場合に使用できます。この表では、列内の最大値と最小値によって、有効な値の範囲が定義されます。表内のレコードへは、これらの列を使用してアクセスします。つまり、これが RCT 表の事前定義されたレコード ID (RID) の性質を利用する最良の方法です。

パーティション化された表
このタイプの表は、複数のデータ・パーティションに分割されたデータとともにインプリメントされます。データは、表の表パーティション・キー列にある値に従って分割されます。パーティション表によって、通常の表よりも表データをより容易にロールインおよびロールアウトでき、管理も容易で、索引を柔軟に配置し、照会処理をより効果的に行うことができます。

データを保持する各表について、使用できるどの表タイプが要件に対して最適かを考慮してください。例えば、緩くクラスター化されている (単調増加しない) データ・レコードがある場合は、通常表と索引を使用することを考慮してください。キー内に重複する (固有でない) 値を持つデータ・レコードがある場合は、範囲クラスター表を使用しないでください。使用する範囲クラスター表に一定量のディスク・ストレージを事前割り当てする余裕がない場合は、このタイプの表は使用しないでください。これらの要素は、範囲クラスター表として使用できるデータがあるかどうかを判断するのに役立ちます。

表の設計

表を設計する際に、一定の概念に精通し、表およびユーザー・データのスペース所要量を判別し、さらに特定のフィーチャー (スペース圧縮やオプティミスティック・ロックなど) を使用するかどうか判別する必要があります。

パーティション表を設計する場合は、以下のようなパーティションの概念を十分理解しておく必要があります。

- データ編成スキーム
- 表パーティション・キー
- データ・パーティション間でのデータの配分に使用されるキー
- MDC ディメンションに使用されるキー

これらやその他のパーティションの概念については、287 ページの『表パーティション化およびデータ編成スキーム』を参照してください。

表の設計概念

表を設計するときは、関連するいくつかの概念を十分理解している必要があります。

列データ・タイプの指定

列を定義する際には、作成する表の各列ごとに、列の名前を指定し、列に含まれるデータの種類 (つまりデータ・タイプ) を定義して、データの長さを定義する必要があります。

バイナリー・データとして保管される文字データ

短精度整数

このデータ・タイプは、精度 15 ビットの 2 進整数値を保管するために使用されます。短精度整数の値の範囲は -32 768 から +32 767 までです。短精度整数データ・タイプは、最小のストレージ・スペースを使って数値を保管します (保管されるそれぞれの値ごとに 2 バイトのスペースが必要)。表定義で短精度整数の列を宣言するには、SMALLINT という用語を使用します。

整数 このデータ・タイプは、精度 31 バイトの 2 進整数値を保管するために使用されます。整数データ・タイプは短精度整数データ・タイプと比べて 2 倍のストレージ・スペースを必要としますが (保管されるそれぞれの値ごとに 4 バイトのスペースが必要)、値の範囲はより広範です。整数値の範囲は -2 147 483 648 から +2 147 483 647 までです。表定義で整数の列を宣言するには、INTEGER および INT という用語を使用できます。

多倍長整数

このデータ・タイプは、64 ビット整数をサポートするプラットフォームにおいて精度 63 ビットの 2 進整数値を保管するために使用されます。多倍長整数として保管された大きな数値の処理は、10 進値として保管された同様の数値の処理よりも効率的です。さらに、多倍長整数値を使用した計算は、REAL 値や DOUBLE 値を使用した計算よりも正確です。

このデータ・タイプは、短精度整数データ・タイプと比べて 4 倍のストレージ・スペースを必要とします (保管されるそれぞれの値ごとに 8 バイトのスペースが必要)。多倍長整数の範囲は -9 223 372 036 854 775 808 から +9 223 372 036 854 775 807 までです。表定義で多倍長整数の列を宣言するには、BIGINT という用語を使用します。

10 進数

このデータ・タイプは、整数部分と小数部分からなる数値を保管するために使用されます。2 つの部分は結合されてパック 10 進数フォーマットで保管されます。 10 進数データ・タイプを宣言する際には、精度 (総桁数) とスケール (数値の小数部分に使用される桁数) を常に指定する必要があります。 10 進数の精度の範囲は 1 から 31 までです。 10 進値の保管に必要なストレージ・スペースの量は、「*精度*÷ 2 (切り捨て) + 1 = *必要なバイト・スペース*」という方程式を使って算出できます。

例えば、DECIMAL(8,2) 値は 5 バイトのストレージ・スペースを必要とします (8 ÷ 2 = 4、4 + 1 = 5)。 DECIMAL(7,2) 値は 4 バイトのストレージ・スペースを必要とします (7 ÷ 2 = 3.5 (3 に切り捨て)、3 + 1 = 4)。

表定義で 10 進数の列を宣言するには、
DECIMAL、DEC、NUMERIC、NUM のどの用語でも使用できます。

注: 10 進数の列定義で精度とスケールの値が指定されない場合には、デフォルトで、精度値 5、スケール値 0 がそれぞれ使用されます (つまり 3 バイトのストレージ・スペースが必要です)。

単精度浮動小数点

このデータ・タイプは、実数の 32 ビット近似値を保管するために使用されます。単精度浮動小数点データ・タイプが必要とするストレージ・スペースの量は整数データ・タイプと同じですが (保管される値ごとに 4 バイトのスペースが必要)、単精度浮動小数点の方が数値の範囲がかなり大きく、$10E^{-38}$ から $10E^{+38}$ まで可能です。

表定義で単精度浮動小数点の列を宣言するには、REAL および FLOAT という用語を使用できます。ただし、FLOAT という用語を使用する場合、列の長さは 1 から 24 の範囲で指定する必要があります。FLOAT は、単精度および倍精度の両方の浮動小数点データ・タイプを表すために使用できます。どちらのデータ・タイプが実際に使用されるかは、指定される長さによって決まります。

倍精度浮動小数点

倍精度浮動小数点データ・タイプは、実数の 64 ビット近似値を保管するために使用されます。倍精度浮動小数点データ・タイプが必要とするストレージ・スペースの量は多倍長整数データ・タイプと同じですが (保管される値ごとに 8 バイトのスペースが必要)、倍精度浮動小数点の数値の範囲が最も大きく、-1.79760^{+308} から $-2.225E^{-307}$ まで、0、および $2.225E^{-307}$ から $-1.79769E^{+308}$ までが可能です。

固定長文字ストリング

このデータ・タイプは、254 文字を超えない特定の長さの文字データおよび文字ストリング・データを保管するために使用されます。表定義で固定長文字ストリングの列を宣言するには、CHARACTER および CHAR という用語を使用できます。固定長文字ストリング・データ・タイプを宣言する際には、保管される文字ストリング・データの長さを常に指定する必要があります。 固定長文字ストリング値の保管に必要なストレージ・スペースの量は、「*固定長* x 1 = *必要なバイト・スペース*」という方程式を使って判別できます。例えば、CHAR(8) 値は 8 バイトのストレージを必要とします。

注: 固定長文字ストリング・データ・タイプを使用する場合、実際のデータの長さが列定義の際に指定された長さよりもかなり短いならば、ストレージ・スペースが浪費される可能性があります。例えば、CHAR(20) と定義された列に値 YES および NO を保管する場

合などです。したがって、固定長文字ストリング列に指定する固定長は、その列に保管されるデータの実際の長さにできるだけ近づける必要があります。

可変長文字データ

このデータ・タイプは、長さが固定されない文字ストリング・データを保管するために使用されます。可変長文字ストリング・データの長さは 32 672 文字まで可能です。ただし、実際に許容される長さは、1 つの表スペース・ページに納まる範囲のデータでなければならないという制約があります。つまり、4K のページを使用する表スペースに格納される表の場合、可変長文字ストリング・データの長さは 4 092 文字を超えることができません。8K のページを使用する表スペースに格納される表の場合、可変長文字ストリング・データの長さは 8 188 文字を超えることができません (32K までこれと同様です)。表スペースはデフォルトで 4K のページとして作成されるため、可変長文字ストリング・データ・タイプを使って 4 092 文字を超えるストリングを保管する場合には、より大きなページ・サイズの表スペースを明示的に作成する必要があります。

注:

- さらに、文字ストリング・データを格納するための十分なスペースが表の行に必要です。言い換えると、表内の他の列のストレージ要件を文字ストリング・データのストレージ要件に付け加える必要があります。必要なストレージ・スペースの総量は、表スペースのページ・サイズを超えてはなりません。
- 可変長ストリング・データ値が更新されて新しい値が元の値より大きくなった場合には、その値を含むレコードが表内の別のページに移されます。このようなレコードをポインター・レコードといいます。 1 つのデータ・レコードの処理に複数のページを検索する必要があるため、ポインター・レコードの数が多すぎるとパフォーマンスが著しく低下する可能性があります。

表定義で可変長文字ストリングの列を宣言するには、CHARACTER VARYING、CHAR VARYING、および VARCHAR という用語を使用できます。可変長文字ストリング列を定義する際には、その列に保管される最大文字数の予測を宣言に含めて指定する必要があります。それ以降、この列に保管される文字ストリング・データ値は、指定された最大の長さ以下でなければなりません。これより長い場合、データは保管されず、エラーが戻されます。

可変長文字ストリング値の保管に必要なストレージ・スペースの量は、「(*ストリングの長さ* x 1) + 4 = *必要なバイト・スペース*」という方程式を使って判別できます。したがって、VARCHAR(30) データ・タイプを使用して 30 文字からなる文字ストリングを保管する場合、長さ 30 という値を指定すると 34 バイトのストレージ・スペースが必要になります。 (このデータ・タイプを使用するすべての文字ストリングは、30 文字以下の長さでなければなりません。)

可変長の長文字データ

また、長さが固定されないストリング・データを保管するために可変長の長文字ストリング・データ・タイプを使用することもできます。このデータ・

タイプは、4K ページの表スペースに格納される表に、32 700 文字以下の長さの文字ストリングを保管するために使用されます。言い換えると、可変長の長文字ストリングを使用する場合には、可変長文字ストリング・データに適用されるページ・サイズおよび文字ストリング・データ長に関する制限が当てはまりません。

表定義で可変長の長文字ストリング列を宣言するには、LONG VARCHAR という用語を使用します。可変長の文字ストリング値の保管に必要なストレージ・スペースの量は、「(*ストリングの長さ* x 1) + 24 = *必要なバイト・スペース*」という方程式を使って判別できます。

注: 表定義で列を宣言する際には、あらゆる文字ストリング・データ・タイプに関して FOR BIT DATA 節を使用できます。この節を使用すると、データ交換操作中にコード・ページ変換が実行されず、データそのものがバイナリー (ビット) データとして処理および比較されます。

文字ラージ・オブジェクト (CLOBS)

CLOB (文字ラージ・オブジェクト) 値の長さは、最大で 2 ギガバイト (2 147 483 647 バイト) まで可能です。 CLOB は、大きな SBCS、または SBCS と MBCS の混合である文字ベースのデータ (例えば 1 つの文字セットで作成された文書) を保管するために使用され、SBCS または混合コード・ページが関連付けられます。

バイナリー・データとして保管される可変長の文字 (ラージ・オブジェクト LOBS およびバイナリー・ラージ・オブジェクト BLOB)

ラージ・オブジェクト という用語および一般的な頭字語 LOB は、BLOB、CLOB、または DBCLOB データ・タイプを指します。 LOB 値には、『可変長文字データ』で説明されているような LONG VARCHAR 値に適用される制限が当てはまります。これらの制限は、LOB ストリングの長さ属性が 254 バイト以下の場合でも該当します。このデータ・タイプは、長さが固定されないバイナリー・ストリング・データを保管するために使用されます。従来型ではないデータ (文書、グラフィック・イメージ、写真、オーディオ、ビデオなど) の保管によく使用されます。

注: SQL では、バイナリー・ラージ・オブジェクト・データを他のデータと同じ方法で操作することはできません。例えば、バイナリー・ラージ・オブジェクト値はソートできません。

Unicode データ

サポートされているデータ・タイプはすべて、Unicode データベースでもサポートされます。特に、GRAPHIC ストリング・データが Unicode データベースでサポートされており、UCS-2 エンコードで格納されます。 SBCS クライアントを含む各クライアントは、Unicode データベースへの接続時に UCS-2 エンコードの GRAPHIC ストリング・データ・タイプを処理できます。

日時データ (タイム・スタンプ)

日付 データ・タイプは、有効なカレンダー・データを示す 3 つの部分からなる値 (年、月、日) を保管するために使用されます。年の部分の範囲は 0001 から 9999 まで、月の範囲は 1 から 12 まで、日の範囲は 1 から n (28、29、30、または 31) までです。 n は月に依存し、うるう年かどうかにも依存します。外部的には、日付データ・タイプは長さ 10 の固定長文字ス

トリング・データ・タイプとして認識されます。ただし、内部的には、日付データ・タイプはかなり少ないストレージ・スペースを必要とします。日付値はパックされたストリングとして保管されるため、値ごとに 4 バイトのスペースだけが必要です。表定義で日付の列を宣言するには、DATE という用語を使用します。

時刻データ・タイプは、一日の中で 24 時間クロック形式の有効な時刻を示す 3 つの部分からなる値 (時間、分、秒) を保管するために使用されます。時間の部分の範囲は 0 から 24 まで、分の範囲は 0 から 59 まで、秒の範囲は 0 から 59 までです。 (時間の部分を 24 に設定する場合には、分および秒の部分を 0 に設定する必要があります。) 外部的には、時刻データ・タイプは長さ 8 の固定長文字ストリング・データ・タイプとして認識されます。ただし、時刻値は日付値と同様に、パックされたストリングとして保管されます。この場合、保管される時刻値ごとに 3 バイトのスペースが必要です。表定義で時刻の列を宣言するには、TIME という用語を使用します。

日付と同様に、時刻の表記方法は世界中でさまざまに異なります。このため、時刻値の形式もまた、使用されるデータベースに関連したテリトリー・コードに応じて異なります。 表 48 は、さまざまな時刻形式と、それらのストリング表記例を示しています。

表 48. 日付形式 (YYYY = 年、MM = 月、DD = 日)

形式の名前	省略形	日付ストリング形式
国際標準化機構	ISO	YYYY-MM-DD
IBM USA 標準規格	USA	MM/DD/YYYY
IBM 欧州標準規格	EUR	MM/DD/YYYY
日本工業規格	JIS	YYYY-MM-DD
サイト固有	LOC	データベースのテリトリー・コードに基づく

数値データ

すべての数値には符号と精度があります。数値がゼロの場合、その符号は正と見なされます。精度とは、符号を除くビット数つまり桁数です。
CREATE TABLE ステートメントに関する説明のデータ・タイプのセクションを参照してください。

通貨データ

バージョン 9.5 では 10 進浮動小数点データ・タイプ DECFLOAT が導入されています。これは、厳密な 10 進数値を扱うビジネス・アプリケーション (例えば金融アプリケーション) で使用できる便利なデータ・タイプです。 10 進数データの 2 進数近似値を提供するバイナリー浮動小数点データ・タイプ (REAL および DOUBLE) は、このようなアプリケーションでは不適切です。 DECFLOAT は DECIMAL の正確さと FLOAT のパフォーマンス上の利点を合わせ持つため、通貨値を操作するアプリケーションで役立ちます。

XML データ

XML データ・タイプは、XML 値を保管する表の列を定義するために使用されます。保管されるすべての XML 値は、整形式 XML 文書でなければ

なりません。このネイティブ XML データ・タイプが導入されたことにより、整形式 XML 文書をネイティブな階層形式で他のリレーショナル・データとともにデータベースに保管できます。

生成列

生成列は表に定義されます。生成列に格納される値は、式を使って計算された値です。挿入操作や更新操作で指定された値ではありません。

作成する表で、特定の式や述部を頻繁に使用することが分かっている場合、その表に 1 つまたは複数の生成列を追加することができます。生成列を使用すると、表データを照会する際のパフォーマンスを向上させることができます。

例えば、パフォーマンスが重要な場合、式の評価の費用を高める恐れのある要因が 2 つあります。

1. 照会中に式の評価を何度も行わなければならないこと。
2. 計算が複雑であること。

照会のパフォーマンスを向上させるには、式の結果を入れるために、列をもう 1 つ定義することができます。そうすれば、同じ式を含んだ照会を発行するときに、生成列を直接に使用できます。あるいは、オプティマイザーの照会書き直しコンポーネントで、その式を生成列に置き換えることもできます。

照会時に複数の表のデータを結合する場合、生成列を追加すると、オプティマイザーがより良い結合の方針を選択できるかもしれません。

生成列は、照会のパフォーマンスを向上させるために使用します。したがって、おそらく、生成列を追加するのは、表を作成したり、表にデータを読み込んだ後になるでしょう。

例

以下に、CREATE TABLE ステートメントを使って生成列を定義する方法の例を示します。

```
CREATE TABLE t1 (c1 INT,
                 c2 DOUBLE,
                 c3 DOUBLE GENERATED ALWAYS AS (c1 + c2)
                 c4 GENERATED ALWAYS AS
                    (CASE WHEN c1 > c2 THEN 1 ELSE NULL END))
```

この表を作成した後で、生成列を使用して索引を作成できます。例:

```
CREATE INDEX i1 ON t1(c4)
```

生成列を照会に利用できます。例:

```
SELECT COUNT(*) FROM t1 WHERE c1 > c2
```

これは、以下のように書き換えることができます。

```
SELECT COUNT(*) FROM t1 WHERE c4 IS NOT NULL
```

別の例を以下に示します。

```
SELECT c1 + c2 FROM t1 WHERE (c1 + c2) * c1 > 100
```

これは、以下のように書き換えることができます。

```
SELECT c3 FROM t1 WHERE c3 * c1 > 100
```

自動番号付けと ID 列

ID 列を使用すると、表に追加される個々の行に対し、固有の数値を DB2 が自動的に生成します。

表に追加する個々の行を固有に識別する必要がある場合、その表を作成する際に、ID 列を表に追加できます。表に追加する個々の行の数値が固有であることを保証するには、ID 列にユニーク索引を作成するか、またはその列を主キーとして宣言する必要があります。

ID 列を使用する他の例としては、注文番号、従業員番号、在庫番号、事例番号などがあります。ALWAYS または BY DEFAULT を指定して、 DB2 データベース・マネージャーによって ID 列の値が生成されるようにすることができます。

GENERATED ALWAYS として定義された ID 列は、常に DB2 データベース・マネージャーが生成する値に指定されます。アプリケーションが明示的に値を指定することはできません。 ID 列を GENERATED BY DEFAULT として定義すると、アプリケーションが明示的に ID 列の値を指定できます。アプリケーションが値を指定しないと、DB2 が値を生成します。アプリケーションが値を制御するので、値が固有であることを DB2 が保証することはできません。GENERATED BY DEFAULT 節は、既存の表の内容をコピーする目的でデータ伝搬を行う場合や、表のアンロードや再ロードを行う場合に使用します。

いったん作成した後で、表の記述を変更して、ID 列を組み込むことはできません。

行が表に、指定された明示的な ID 列値で挿入される場合、次の内部生成される値は更新されず、表内の既存の値と競合する可能性があります。ID 列の値の固有性が、その ID 列に定義されている主キーまたはユニーク索引によって強制される場合、重複値があるとエラー・メッセージが生成されます。

新しい表に ID 列を定義するには、CREATE TABLE ステートメントに AS IDENTITY 節を使用します。

例

以下に、CREATE TABLE ステートメントを使って ID 列を定義する方法の例を示します。

```
CREATE TABLE table (col1 INT,
                    col2 DOUBLE,
                    col3 INT NOT NULL GENERATED ALWAYS AS IDENTITY
                                      (START WITH 100, INCREMENT BY 5))
```

この例では、3 番目の列が ID 列です。この列で使用される値を指定して、追加される個々の行を固有に識別することもできます。この例の場合、最初に入力される行については、値 『100』 が ID 列に入れられます。この表に行が追加されるごとに、値は 5 ずつ増えます。

制約、デフォルト、NULL 設定による列データの制約

多くの場合、データは特定の制約事項や規則に従う必要があります。そのような制約事項は、情報の 1 つの断片 (例えば形式やシーケンス番号) に個別に適用される場合もあれば、情報の複数の断片に適用される場合もあります。

列データ値の NULL 可能性

NULL 値は不明な状態を表します。デフォルトでは、すべての組み込みデータ・タイプで NULL 値の存在がサポートされます。しかし、ビジネス・ルールによっては、いくつかの列に必ず値を入れる必要が生じるかもしれません (例えば緊急時の情報)。このような場合、NOT NULL 制約を使用すれば、表の特定の列に決して NULL 値を割り当てられないように設定できます。 NOT NULL 制約が特定の列について定義されていると、その列に NULL 値を入れようとする挿入または更新操作は失敗します。

デフォルトの列データ値

必ず値を提供するよう求めるビジネス・ルールがあるのと同様に、どんな値が必要かを指定するビジネス・ルールもあります (例えば従業員の性別は M (男性) または F (女性) のどちらかである必要があります)。列デフォルト制約を使用すれば、表の特定の列に関して具体的な列値を持たない行が表に追加された場合、事前定義値がその列に必ず割り当てられるように設定できます。列に割り当てるデフォルト値として、NULL、列データ・タイプと互換性のある制約値、またはデータベース・マネージャーによって提供される値が可能です。詳しくは、『デフォルトの列およびデータ・タイプの定義』を参照してください。

キー キーとは、特定の行データの識別またはアクセスに使用できる、表または索引の中の 1 つの列または列セットです。任意の列をキーに含めることができ、同じ列を複数のキーに含めることもできます。単一の列からなるキーをアトミック・キーといいます。複数の列からなるキーを複合キーといいます。アトミックまたは複合という属性に加えて、制約をインプリメントするためのキーの使用方法に応じて、次のようにキーを分類できます。

- ユニーク・キーは、ユニーク制約をインプリメントするために使用されます。
- 主キーは、エンティティー整合性制約をインプリメントするために使用されます。 (主キーは NULL 値をサポートしない特殊な種類のユニーク・キーです。)
- 外部キーは、参照整合性制約をインプリメントするために使用されます。(外部キーは主キーまたはユニーク・キーを参照する必要があります。外部キーには、対応する索引がありません。)

ほとんどの場合、キーは表、索引、または参照制約定義を宣言するときに指定されます。

制約 制約とは、表の中で挿入、削除、または更新できる値を制限する規則です。制約には、チェック制約、主キー制約、参照制約、ユニーク制約、ユニーク・キー制約、外部キー制約、インフォメーショナル制約があります。それぞれの種類の制約について、詳しくは『 303 ページの『第 12 章 制約』』または『 303 ページの『制約のタイプ』』を参照してください。

デフォルトの列およびデータ・タイプの定義:

列およびデータ・タイプには、デフォルト値が事前に定義されている、あるいは割り当てられているものがあります。

例えば、各種データ・タイプのデフォルトの列値は次のとおりです。

- *NULL*
- *0:* 短精度整数、整数、10 進数、単精度浮動小数点、および倍精度浮動小数点で使用されます。
- *ブランク:* 固定長文字ストリングおよび固定長 2 バイト文字ストリングで使用されます。
- *長さがゼロのストリング:* 可変長文字ストリング、バイナリー・ラージ・オブジェクト、文字ラージ・オブジェクト、および 2 バイト文字ラージ・オブジェクトで使用されます。
- *日付:* これは、行が挿入された時点のシステム日付です (これは CURRENT_DATE 特殊レジスターから取得されます)。既存の表に日付列が追加されると、既存の行に日付 0001 年 1 月 1 日 (January, 01, 0001) が割り当てられます。
- *時刻またはタイム・スタンプ:* これは、ステートメントが挿入された時点のシステム時刻またはシステム日時です (これは CURRENT_TIME 特殊レジスターから取得されます)。既存の表に時刻列が追加されると、既存の行に時刻 00:00:00 か、または日付 0001 年 1 月 1 日 (January, 01, 0001) と時刻 00:00:00 を含むタイム・スタンプが割り当てられます。

  注: 任意のステートメントが 1 つ発行された場合、そのすべての行が同じデフォルト時刻/タイムスタンプの値を取得します。
- *ユーザー定義の特殊データ・タイプ:* これは、ユーザー定義の特殊データ・タイプの基本データ・タイプに関するシステム定義のデフォルト値です (ユーザー定義の特殊データ・タイプにキャストされる)。

主キー、参照整合性、チェック、およびユニーク制約

制約とは、表の中で挿入、削除、または更新できる値を制限する規則です。

主キー制約

主キー制約とは、ユニーク制約と同じプロパティーを持つ、列または列の組み合わせです。主キー制約と外部キー制約を使用して、表間のリレーションシップを定義できます。

参照整合性 (または外部キー) 制約

外部キー制約 (参照制約または参照整合性制約とも呼ばれる) は、 1 つ以上の表の中の 1 つ以上の列での値に関する論理規則です。例えば、表集合は企業の製造業者に関する情報を共有します。場合によっては、製造業者の名前が変わることもあります。表の製造業者の ID が、製造業者情報の製造業者 ID と一致していなければならないことを示す参照制約を定義できます。この制約により、製造業者情報が失われてしまう結果になりかねない挿入、更新、削除操作が抑止されます。

チェック制約

(表) チェック制約は、特定の表に追加されるデータに制約を設定します。

ユニーク制約

ユニーク制約 (ユニーク・キー制約とも呼ばれる) は、表の中の 1 つ以上の列での重複値を禁止する規則です。ユニーク制約では、ユニーク・キーと主キーがサポートされています。

Unicode 表およびデータに関する考慮事項

Unicode 文字エンコード規格は、固定長の文字エンコード方式です。これには、世界で実際に使われているほとんどすべての言語の文字が含まれています。

Unicode 表およびデータに関する考慮事項について詳しくは、以下を参照してください。

- 「*国際化対応ガイド*」の『Unicode 文字のエンコード』
- 「*国際化対応ガイド*」の『照合順序に基づく文字比較』
- 「*国際化対応ガイド*」の『テリトリー・コードによる日時の形式』
- 「*国際化対応ガイド*」の『ユーロを使用可能なコード・ページ遷移表ファイル』

Unicode に関する追加情報は、最新版の「*Unicode Standard*」、および Unicode Consortium の Web サイト www.unicode.org をご覧ください。

表のスペース所要量

表を設計する際に、スペース所要量を考慮する必要があります。

ロング・フィールド (LF) データ

ロング・フィールド (LF) データは、他のデータ・タイプのストレージ・スペースとは異なる、別個の表オブジェクトに格納します。

データが格納される 32-KB 域は、「2 の累乗」× 512 バイトのサイズのセグメントに分割されます。 (このため、これらのセグメントは、512 バイト、1024 バイト、2048 バイトというように、最高 32768 バイトまでが可能です。)

ロング・フィールド・データ・タイプ (LONG VARCHAR または LONG VARGRAPHIC) は、フリー・スペースを容易に再利用できるような方法で保管されます。割り振りとフリー・スペースに関する情報は 4 KB の割り振りページに格納されますが、オブジェクト中はさほど頻繁には見られません。

オブジェクト内の未使用スペースの量は、ロング・フィールド・データのサイズと、そのサイズがデータのすべてのオカレンスを通じてある程度一定しているかどうかによって決まります。 255 バイトを超えるデータ入力項目の場合、未使用のスペースは、ロング・フィールド・データのサイズの 50% に達することもあります。

文字データの長さがページ・サイズより短く、データの残りの部分と合わせてもページ・サイズ内に収まる場合には、LONG VARCHAR または LONG VARGRAPHIC の代わりに、CHAR、GRAPHIC、VARCHAR、または VARGRAPHIC の各データ・タイプを使用すべきです。

ラージ・オブジェクト (LOB) データ

ラージ・オブジェクト (LOB) データは、他のデータ・タイプのストレージ・スペースとは構造が異なる、2 つの別個の表オブジェクトに格納しま

す。 LOB データに必要なスペースを見積もるには、これらのデータ・タイプで定義されたデータを格納するための、次のような 2 つの表オブジェクトについて考慮する必要があります。

- *LOB データ・オブジェクト:* データが格納される 64 MB 域は、「2 の累乗」× 1024 バイトのサイズのセグメントに分割されます。 (つまり、このセグメントの大きさは、1024 バイト、2048 バイト、 4096 バイトというようにして、最高 64 MB までとなります。)

  LOB データに使用されるディスク・スペースの量を少なくするために、CREATE TABLE および ALTER TABLE ステートメントの lob-options 節で COMPACT オプションを指定することができます。 COMPACT オプションを指定すると、LOB データは小さなセグメントに分割され、必要なディスク・スペースの量を最小限にとどめることができます。これはデータ圧縮を伴わず、単に最も近い 1 KB 境界になるよう、最小限のスペースを使用しているだけです。 LOB 値に付加する場合、 COMPACT オプションを指定するとパフォーマンスが低下する可能性があります。

  LOB データ・オブジェクトでのフリー・スペース量は、更新および削除の量と挿入する LOB 値のサイズによって変わります。

- *LOB 割り振りオブジェクト:*割り振りとフリー・スペースに関する情報は、実際のデータとは別の 4 KB 割り振りページに格納されます。使用される 4 KB ページの数は、ラージ・オブジェクト・データに割り振られるデータの量 (未使用スペース含む) に依存しています。オーバーヘッドは次のように計算されます。 64 GB ごとに 4 KB ページ 1 個 + 8 MB ごとに 4 KB ページ 1 個。

  文字データの長さがページ・サイズより短く、データの残りの部分と合わせてもページ・サイズ内に収まる場合には、BLOB、CLOB、または DBCLOB の代わりに、CHAR、GRAPHIC、VARCHAR、または VARGRAPHIC の各データ・タイプを使用すべきです。

システム・カタログ表

データベースが作成されると、システム・カタログ表も作成されます。システム表は、データベース・オブジェクトと特権がデータベースに追加されるたびに増加します。

最初の時点では、約 3.5 MB のディスク・スペースが使用されます。

カタログ表に割り当てられるスペースの量は、表スペースのタイプとカタログ表が含まれる表スペースのエクステント・サイズによって異なります。例えば、エクステント・サイズが 32 の DMS 表スペースが使用される場合、最初の時点ではカタログ表に 20 MB のスペースが割り振られます。注: 複数パーティションを持つデータベースの場合、カタログ表は CREATE DATABASE コマンドの発行元のデータベース・パーティション上にのみ存在します。カタログ表のディスク・スペースは、そのデータベース・パーティションのためだけに必要です。

一時表 一部のステートメントには、処理のための一時表が必要です (メモリー中で行えないソート操作のための作業ファイルなど)。これらの一時表にはディスク・スペースが必要です。必要なスペースの量は、照会のサイズ、数、および性質、また戻される表のサイズに応じて異なります。

実際の作業環境はそれぞれに異なっているため、一時表のスペース要件を見積もることは困難です。例えば、さまざまなシステム一時表の存在期間が長いために、 SYSTEM TEMPORARY 表スペースに割り振られるスペースの量が実際に使用されるより多いように思える場合があります。このことは、DB2_SMS_TRUNC_TMPTABLE_THRESH レジストリー変数を使用した場合に発生することがあります。

データベース・システム・モニターおよび表スペース照会 API を使用して、通常の操作で使用されるワークスペース量を追跡することができます。

DB2_OPT_MAX_TEMP_SIZE レジストリー変数を使用すると、照会で使用される TEMPORARY 表スペースの量を制限することができます。

表のページ・サイズ

表データの行は、ページと呼ばれるブロックに格納されます。使用できるページのサイズは、4 KB、8 KB、16 KB、32 KB の 4 種類です。表データ・ページには、LONG VARCHAR、LONG VARGRAPHIC、BLOB、CLOB、または DCLOB データ・タイプとして定義された列のデータは含まれません。ただし、それらの列の記述子は、表データ・ページの行に含まれます。

ページ・サイズが 4 KB、8 KB、16 KB、または 32 KB のバッファー・プールまたは表スペースを作成することができます。特定サイズの表スペース内に作成される表にはすべて、一致するページ・サイズが指定されます。単一の表または索引オブジェクトのサイズは、 (32 KB のページ・サイズを想定すると) 最大 512 GB まで可能です。

使用するページ・サイズが 8 KB、16 KB、または 32 KB の場合、使用できる列は最大で 1012 です。ページ・サイズが 4 KB の場合、使用できる列の数は最大で 500 です。ページあたり使用できる行の数は、ページ・サイズにかかわらず最大で 255 です。

行の最大長は、使用するページ・サイズに応じて以下のように異なります。

- ページ・サイズが 4 KB の場合、行の最大長は 4005 バイトです。
- ページ・サイズが 8 KB の場合、行の最大長は 8101 バイトです。
- ページ・サイズが 16 KB の場合、行の最大長は 16293 バイトです。
- ページ・サイズが 32 KB の場合、行の最大長は 32677 バイトです。

表スペースのページ・サイズを決定する際は、以下のことを考慮してください。

- 行のランダム読み取りおよび書き込み操作を実行する OLTP アプリケーションについては、不必要な行に使用するバッファー・プールのスペースが少なくなるので、通常はページ・サイズは小さい方が望ましいです。
- 一度に多くの連続した行にアクセスする DSS アプリケーションについては、指定された数の行を読み取るのに必要な入出力要求が減るので、通常はページ・サイズは大きい方が望ましいです。しかし、これには例外があります。行のサイズが `pagesize / 255` より小さい場合、各ページには無駄なスペースが生じます (1 ページの行数は最大で 255 です)。この場合、ページ・サイズは小さい方が適切でしょう。

ページ・サイズが大きいと、索引のレベルの数を減らすことができます。大きいページは、長い行をサポートします。デフォルトの 4 KB ページを使用する場合、表の列は 500 に制限されます。より大きなページ・サイズ (8 KB、16 KB、32 KB) は 1012 列をサポートします。表スペースに使用できる最大サイズは、表スペースのページ・サイズに比例します。

ユーザー表データのスペース所要量

デフォルトでは、表データは 4 KB のページに格納されます。各ページ (ページ・サイズは関係ない) には、68 バイトからなるデータベース・マネージャー用のオーバーヘッドが含まれます。そのため、ユーザー・データ (つまり行) を入れるのに 4028 バイトが残されています。ただし 4 KB のページ上のいずれの行も、長さ 4005 バイトを超えてはなりません。 1 つの行が複数のページにまたがることはありません。ページ・サイズが 4 KB の場合、使用できる列の数は最大で 500 です。

表データ・ページには、 LONG VARCHAR、 LONG VARGRAPHIC、 BLOB、CLOB、および DBCLOB データ・タイプとして定義された列のデータは含まれません。ただし、それらの列の記述子は、表データ・ページの行に含まれます。

通常、行は先頭一致順で正規の表に挿入されます。ファイルの中から、新しい行を保持できる最初の使用可能なスペースが (フリー・スペース・マップを使って) 検索されます。行が更新されると、それを含むページ上に十分なスペースがある限り、行は元の場所のままで更新されます。この場合には、オリジナルの行ロケーションにレコードが 1 つ作成され、更新された行の表ファイルの中の新しいロケーションを指し示します。

ALTER TABLE APPEND ON ステートメントが発行されると、データは常に追加され、データ・ページのフリー・スペースについての情報は保持されません。

表にクラスター索引が定義されているなら、データベース・マネージャーは、そのクラスター索引のキー順に従ってデータを物理的にクラスター化することを試みます。その表に行が挿入されると、データベース・マネージャーは、まずクラスター索引の中からそのキー値を検索します。キー値が見つかったなら、データベース・マネージャーはそのキーの指し示すデータ・ページにレコードを挿入することを試みます。キー値が見つからない場合は、その 1 つ上のキー値を使用し、この値を持つレコードを含むページにレコードを挿入します。表のターゲット・ページに十分なスペースがない場合には、その近くにあるページからスペースを検索するため、フリー・スペース・マップが使用されます。時間とともにデータ・ページのスペースはすべて使用され、レコードは表の中のターゲット・ページからますます遠い位置に入れられるようになります。やがて表データはクラスター化されていないと見なされるようになったなら、表の再編成を使用することによりクラスター化されたデータの配列をリストアできます。

表がマルチディメンション・クラスタリング (MDC) 表の場合、データベース・マネージャーは、 1 つ以上の定義済みディメンションまたはクラスター索引に従って、レコードが常に物理的にクラスター化されることを保証します。 MDC 表が特定のディメンションで定義されているなら、各ディメンションごとにブロック索引が作成され、セル (ディメンション値の固有の組み合わせ) をブロックにマップする複合ブロック索引が作成されます。この複合ブロック索引は、特定のレコードがど

のセルに属するか、また表の中でそのセルに属するレコードを含むブロックまたはエクステントを正確に決定するために使用されます。その結果、レコード挿入時にデータベース・マネージャーは、ディメンション値が同じであるレコードを含むブロックのリストを複合ブロック索引から検索し、スペース検索の対象をそれらのブロックだけに限定します。セルがまだ存在していない場合、またはそのセルの既存のブロック中に十分なスペースがない場合には、別のブロックがそのセルに割り当てられ、そこにレコードが挿入されます。その場合も、ブロック内で使用可能なスペースを素早く見つけるために、フリー・スペース・マップが使用されます。

データベース内にある各ユーザー表の 4 KB ページ数を見積もる場合、以下のように計算します。

```
ROUND DOWN(4028/(average row size + 10)) = records_per_page
```

上の結果を以下の式に代入します。

```
(number_of_records/records_per_page) * 1.1 = number_of_pages
```

ここで平均行サイズは平均列サイズの合計です。また 1.1 はオーバーヘッドに起因する値です。

注: この公式はあくまでも見積もりの算出です。フラグメント化やオーバーフロー・レコードのためにレコード長にばらつきがある場合、この見積もりの精度は低下します。

ページ・サイズが 8 KB、16 KB、または 32 KB のバッファー・プールまたは表スペースを作成するためのオプションもあります。特定サイズの表スペース内に作成される表にはすべて、一致するページ・サイズが指定されます。単一の表または索引オブジェクトのサイズは、 (32 KB のページ・サイズを想定すると) 最大 512 GB まで可能です。ページ・サイズが 8 KB、16 KB、または 32 KB の場合、使用できる列は最大で 1012 です。 4 KB ページ・サイズの場合、列の最大数は 500 です。行の最大長は、ページ・サイズに応じて以下のように異なります。

- ページ・サイズが 4 KB の場合、行の最大長は 4005 バイトです。
- ページ・サイズが 8 KB の場合、行の最大長は 8101 バイトです。
- ページ・サイズが 16 KB の場合、行の最大長は 16 293 バイトです。
- ページ・サイズが 32 KB の場合、行の最大長は 32 677 バイトです。

ページ・サイズを大きくすると、索引のレベルの数を減少させることができます。行のランダム読み取りおよび書き込みを行う OLTP (オンライン・トランザクション処理) アプリケーションを使用する場合は、不必要な行に費やされるバッファー・スペースが少なくなるので、ページ・サイズは小さい方が望ましいです。一度に多くの連続した行にアクセスする DSS (意思決定支援システム) アプリケーションを使用する場合は、指定された数の行を読み取るのに必要な入出力要求の数が減るので、ページ・サイズは大きい方が望ましいです。

バックアップ・イメージを異なるページ・サイズにリストアすることはできません。

755 を超える列を表す IXF データ・ファイルをインポートすることはできません。

宣言済み一時表は、独自の USER TEMPORARY 表スペース・タイプの中にのみ作成されます。デフォルトの USER TEMPORARY 表スペースはありません。一時表には LONG データを入れることはできません。これらの表は、アプリケーションがデータベースから切断すると暗黙的にドロップされます。これらの表のスペース所要量を見積もる際には、この点を考慮に入れる必要があります。

表のスペース圧縮

データ行、NULL 値、およびシステム・デフォルト値の圧縮のようなフィーチャーを使用すると、表をディスクに保管するときに占有されるスペースをより少なくすることができるかもしれません。データ圧縮を使用するとデータ保管の際に使用されるデータベース・ページが少なくなるため、ディスク・ストレージ・スペースを節約することができます。 1 ページに保管できる論理データが増えるので、同じ量の論理データにアクセスするために読み取るページ数が減ります。これは、圧縮を使用することによってディスク入出力を減らせることをも意味します。バッファー・プールにキャッシュできる論理データが増えるので、入出力の速度も上がります。

データベース・システムにデータ圧縮をインプリメントするためのメソッドは 2 つあります。

(スペース) 値圧縮

このメソッドは、データを表現するために使用されるスペースや、データベース管理システム (DBMS) がデータ保管のために内部的に使用するストレージ構造を最大限利用します。値圧縮では、値の重複項目を除去し、1 つのコピーだけを保管します。保管された値への参照がある場合、保管されたコピーはその場所を追跡します。

テーブルの作成時、オプションの VALUE COMPRESSION 節を使用して、表が表レベル、そしておそらく列レベルで行フォーマットを節約するスペースを使用していることを指定します。

VALUE COMPRESSION が使用されると、NULL および定義済み可変長データ・タイプ (VARCHAR、 VARGRAPHICS、 LONG VARCHAR、 LONG VARGRAPHIC、 BLOB、 CLOB、および DBCLOB) に割り当てられた、長さがゼロのデータは、ディスク上に保管されません。これらのデータ・タイプに関連したオーバーヘッド値のみがディスク・スペースを使用します。

VALUE COMPRESSION が使用されると、COMPRESS SYSTEM DEFAULT オプションを使用して、さらにディスク・スペース使用量を削減することができます。挿入または更新値が、列のデータ・タイプのシステム・デフォルト値に等しい場合、使用されるディスク・スペースは最小になります。デフォルト値はディスク上に保管されません。 COMPRESS SYSTEM DEFAULT をサポートするデータ・タイプには、すべての数値タイプ列、固定長文字、および固定長 GRAPHIC ストリング・データ・タイプが含まれます。これは、ゼロおよびブランクは圧縮できるということです。

(データ) 行圧縮

このメソッドは、1 つの行の複数の列値に渡る反復パターンをより短いシンボル・ストリングで置き換えることによってデータ行を圧縮します。データ行圧縮の目的は、ディスク・ストレージ・スペースの節約を実現することで

す。データ行圧縮は、ディスク入出力の節約にもなります。また、さらに多くのデータをバッファー・プールにキャッシュできるので、バッファー・プールのヒット率も向上します。とはいえ、データの圧縮および圧縮解除に必要な余分の CPU サイクルという形で、関連コストが生じます。データ行圧縮のストレージ節約およびパフォーマンスへの影響は、データベース内のデータの特性、データベースのレイアウトと調整、およびアプリケーション・ワークロードと関係しています。データ・ページ上のデータまたはログ・レコード内のデータのみが圧縮されます。

データ行圧縮は、静的なディクショナリー・ベースの圧縮アルゴリズムを使用して、行ごとにデータを圧縮します。行レベルでのデータ圧縮では、1 つの行の複数の列値に渡る反復パターンを、より短いシンボル・ストリングで置き換えることが可能です。表データを圧縮するには、表 COMPRESS 属性を YES に設定する必要があり、コンプレッション・ディクショナリーが表に対して存在している必要があります。

コンプレッション・ディクショナリーを作成する (およびそれに続けて表を圧縮する) には、従来の (オフライン) 表再編成を実行します。また、コンプレッション・ディクショナリーは、IMPORT、LOAD INSERT、および LOAD REPLACE を含む INSERT 操作、およびいくつかの REDISTRIBUTE 操作で作成されます。表内に存在しているすべてのデータ行は、コンプレッション・ディクショナリーの作成に参加します。このコンプレッション・ディクショナリーは、表のデータ・オブジェクト部分に、表データ行と共に保管されます。

表を圧縮解除するには、表の COMPRESS 属性を NO に設定してから、従来の (オフライン) 表再編成を実行します。

ページに挿入されているデータ行は、表の COMPRESS 属性が YES に設定されていて、ディクショナリーが存在する場合に圧縮できます。これは、インポートまたはロード操作による挿入を含め、行を挿入する操作すべてに適用されます。圧縮は表全体に対して使用可能にされますが、各行は個別に圧縮されます。そのため、圧縮された行と圧縮されていない行の両方を 1 つの表に同時に含めることも可能です。

データ行圧縮は、索引、long、LOB、および XML データ・オブジェクトには適用できません。

行圧縮には、表データのレプリケーション・サポートとの互換性はありません。

行圧縮統計は、RUNSTATS コマンドを使用して生成することができ、システム・カタログ表 SYSCAT.TABLES に保管されます。圧縮見積もりオプションは、INSPECT ユーティリティーで使用できます。照会オプティマイザーのコスト計算モデルには、圧縮解除のコストが組み込まれています。

UPDATE アクティビティーおよびデータ行内の更新変更の位置づけに応じて、ログの消費量が増える場合があります。ロギングおよび列の順序付けの更新については、274 ページの『更新ロギングを最小化するための列の順序付け』を参照してください。

更新の結果として行のサイズが大きくなる場合、新しいバージョンの行は現在のデータ・ページに収まらない可能性があります。行の新しいイメージは、そこにではなく、オーバーフロー・ページに格納されます。この種のポ

インター・オーバーフロー・レコードの作成を最小限にするには、データ・ページにさらにフリー・スペースを追加すればよいのです。例えば、圧縮せずに 5% のフリー・スペースを使用していた場合は、圧縮した 10% のフリー・スペースを割り振ってください。 この推奨事項は、頻繁に更新されるデータの場合に特に重要です。

既存の表のスペース圧縮

VALUE COMPRESSION 節を指定すると、スペース圧縮ができるように既存表の行フォーマットを変更できます。スペース圧縮を許可する列のバイト・カウントの合計は、スペース圧縮を許可しない列のバイト・カウントの合計を超える可能性があります。これは、バイト・カウントの合計が、表スペース内の表で許容される行の長さを超えないかぎり、そう言えます。例えば、許容される行の長さは、4 KB のページ・サイズを持つ表スペースでは 4005 バイトです。許容される行の長さを超える場合には、エラー・メッセージ SQL0670N が戻ります。バイト・カウントの公式は、CREATE TABLE ステートメントの一部として文書化されています。

同様に、VALUE COMPRESSION 節を削除すると、スペース圧縮がそれまで許可されていた表の行をスペース圧縮を許容しないように変更できます。列のバイト・カウントの合計に関して同じ条件が適用され、エラー・メッセージ SQL0670N が必要に応じて戻されます。

表のスペース圧縮を考慮すべきかどうかを判断するために、ほとんどがシステム・デフォルト値に等しい値、または NULL になっている表は、新しく行フォーマットを行う利点があるということを知っておくとよいでしょう。例えば、INTEGER 列と 90% の列が 0 の値 (データ・タイプ INTEGER のデフォルト値) または NULL になっているところでは、この表に加えてこの列を圧縮すると、新しく行フォーマットを行う利点が得られ、多くのディスク・スペースを節約します。

表を変更する時には、VALUE COMPRESSION 節を使用して、表が表レベルで (おそらく列レベルでも) スペース行フォーマットを使用しているということを指定することができます。 ACTIVATE VALUE COMPRESSION を使用して、表がスペース節約手法を使用することを指定するか、または DEACTIVATE VALUE COMPRESSION を使用して、表がその表のデータではもはやスペース節約手法を使用しないことを指定します。

DEACTIVATE VALUE COMPRESSION を使用する場合は、そのことにより、その表の列に関連したすべての COMPRESS SYSTEM DEFAULT オプションを暗黙的に使用不可にします。

表を新しい行フォーマットに変更した後は、それ以降に挿入、ロード、または更新されるすべての行は、新しい行フォーマットを持ちます。すべての行を新しい行フォーマットに変更するために、行フォーマットを変更する前に、表の再編成を実行するか、既存の行に更新操作を行う必要があります。

更新ロギングを最小化するための列の順序付け

CREATE TABLE ステートメントを使用して列を定義する際は、特に更新が集中するワークロードに関して、列の順序を考慮してください。頻繁に更新される列は、グループにまとめ、表定義の最後もしくは最後の方に定義してください。このようにすると、パフォーマンスが向上し、ログに記録されるバイト数が減り、書き込ま

れるログ・ページの数が少なくなるほか、多くの更新を実行するトランザクションに必要とされるアクティブなログのスペース要件も小さくできます。

データベース・マネージャーは、UPDATE ステートメントの SET 節で指定されている列の値が変更されていることを自動的に想定することはしません。索引の保守や、ロギングに必要な行の数を制限するために、データベースは、新しい列値と古い列値を比較して、列が変更されているかどうかを判別します。値が変更されている列だけが更新されているものとして扱われます。データがデータ行の外部に保管されている列 (long、LOB、ADT、および XML 列タイプ) や、レジストリー変数 DB2ASSUMEUPDATE が使用可能になっている場合の固定長列の場合は、この UPDATE の動作に例外が発生します。これらの例外が発生した場合、列の値は変更されているものと見なされるため、新しい列値と古い列値の比較は行われません。

UPDATE ログ・レコードには、3 つの異なるタイプがあります。

- 更新前および更新後の行イメージの完全なロギング。更新前および更新後の行のイメージがすべてログに記録されます。これは、DATA CAPTURE CHANGES が使用可能になっている表でのみ実行されるロギングのタイプで、結果のバイト数が行の更新のロギングの中で最も大きいロギングです。
- 完全 XOR ロギング。更新前の行イメージと更新後の行イメージの XOR の差で、変更されている箇所の先頭のバイトから短いほうの行の端まで、そして長い方の行の残りすべてのバイト数。このロギングの結果は、更新前と更新後のイメージの完全なロギングに比べて記録されるログのバイト数が少なくなり、ログ・レコード・ヘッダー情報を越えたデータのバイト数が最も長い行イメージのサイズになります。
- 部分 XOR ロギング。更新前の行イメージと更新後の行イメージの XOR の差で、変更されている箇所の先頭のバイトから変更されている箇所の最後のバイトまで。バイト位置は、列の先頭のバイトか最後のバイトのいずれかです。このロギングは、結果のログのバイト数が最も小さいロギングで、行の更新に関するログ・レコードの中で最も効率的なタイプです。

表で DATA CAPTURE CHANGES が使用可能になっていない場合、更新のログで記録されるデータの量は以下の点に依存します。

- 更新される列 (COLNO) の隣接性
- 更新される列が固定長か可変長か
- 行圧縮 (COMPRESS YES) が使用可能か

行の合計長が変わらない場合は、行の圧縮が使用可能であっても、データベース・マネージャーは最適な部分 XOR ログ・レコードを計算して書き込みます。

行の合計長が変わる場合 (更新される列が可変長で、行圧縮が使用可能な場合に一般的) は、データベース・マネージャーは、変更された箇所の先頭のバイトを判別して、完全 XOR ログ・レコードを書き込みます。

データ行圧縮

データ行圧縮の目的は、ディスク・ストレージ・スペースの節約を実現することです。これにより、ディスク入出力も節約されます。また、さらに多くのデータをバッファー・プールにキャッシュできるので、バッファー・プールのヒット率も向上

します。データ行圧縮は、静的なディクショナリー・ベースの圧縮アルゴリズムを使用して、行ごとにデータを圧縮します。

行レベルでのデータ圧縮では、1 つの行の複数の列値に渡る反復パターンを、より短いシンボル・ストリングで置き換えることが可能です。

注: データの圧縮および圧縮解除に必要な余分の CPU サイクルという形で、関連コストが生じます。データ行圧縮のストレージ節約およびパフォーマンスへの影響は、データベース内のデータの特性、データベースのレイアウトと調整、およびアプリケーション・ワークロードと関係しています。データ・ページ上のデータまたはログ・レコード内のデータのみが圧縮されます。

表データを圧縮するには、コンプレッション・ディクショナリーが表に対して存在している必要があり、CREATE TABLE または ALTER TABLE ステートメントの COMPRESS 属性を YES に設定する必要があり、また、十分なデータが表内に存在している必要もあります。これらの圧縮条件が表に存在している場合、INSERT ステートメントまたは LOAD INSERT、IMPORT INSERT、または REDISTRIBUTE コマンドを発行すると、表に追加されたデータが圧縮されます。

バージョン 9.5 では、表の COMPRESS 属性が YES に設定され、データ・コンプレッション・ディクショナリーが作成されると、データ行圧縮は自動的に使用可能になります。COMPRESS 属性を YES に設定して表を作成または変更している場合は、ユーザー側での手動操作もデータベース要求も必要ありません。つまり、データ・コンプレッション・ディクショナリーを作成するために、明示的な従来の (オフライン) 表再編成を実行する必要はありません。

注: COMPRESS 属性を YES に設定し、コンプレッション・ディクショナリーが存在している場合は、圧縮がインポートまたはロード操作による挿入を含め、行を挿入する操作すべてに適用されます。圧縮は表全体に対して使用可能にされますが、各行は個別に圧縮されます。そのため、圧縮された行と圧縮されていない行の両方を 1 つの表に同時に含めることも可能です。

コンプレッション・ディクショナリーを明示的に作成する (およびそれに続けて表を圧縮する) には、従来の (オフライン) 表再編成を実行します。表内に存在しているすべてのデータ行は、コンプレッション・ディクショナリーの作成に参加します。このコンプレッション・ディクショナリーは、表のデータ・オブジェクト部分に、表データ行と共に保管されます。

表を圧縮解除するには、表の COMPRESS 属性を NO に設定してから、従来の (オフライン) 表再編成を実行します。

制約事項

- データ行圧縮は、索引、long、LOB、および XML データ・オブジェクトには適用できません。
- 行圧縮には、表データのレプリケーション・サポートとの互換性はありません。
- RUNSTATS コマンドを使用して、行圧縮統計を生成することができます。この統計は、システム・カタログ表の SYSCAT.TABLES に保管されます。表に対する行の圧縮効果を見積もる圧縮見積もりオプションが、INSPECT ユーティリティーで使用できます。照会オプティマイザーのコスト計算モデルには、圧縮解除のコストが組み込まれています。

- 更新アクティビティーおよびデータ行内の更新変更の位置づけに応じて、ログ・スペースの消費量が増える場合があります。
- 行のサイズが大きくなる場合、新しいバージョンの行は現在のデータ・ページに収まらない可能性があります。この場合、行の新しいイメージは、オーバーフロー・ページに格納されます。この種のポインター・オーバーフロー・レコードの作成を最小限にするには、データ・ページにさらにフリー・スペースを追加することができます。例えば、圧縮せずに 5% のフリー・スペースを使用していた場合は、圧縮した 10% のフリー・スペースを割り振ってください。この推奨事項は、頻繁に更新されるデータの場合に特に重要です。

オプティミスティック・ロック

バージョン 9.5 では、拡張オプティミスティック・ロック・サポートは SQL データベース・アプリケーションのための技法を提供しており、この技法では行の選択と更新または削除の間の行ロックを保持しません。

アンロックされた行は更新または削除前に変更される可能性はまずないと想定して、アプリケーションを楽観的に書くことができます。行が変更される場合、更新または削除は失敗し、アプリケーションの論理はそのような障害を、例えば選択を再試行することによって処理できます。

この拡張オプティミスティック・ロックの利点は、他のアプリケーションが同じ行を同時に読み取り/書き込みできるので、並行性が向上することです。ビジネス・トランザクションがデータベース・トランザクションに相関していない 3 層環境では、このオプティミスティック・ロック技法が使用されます。なぜなら、こうした環境ではロックをビジネス・トランザクション間で維持できないからです。

オプティミスティック・ロック

オプティミスティック・ロック とは、行を選択してから行を更新または削除するまでの間に行ロックを保持しない SQL データベース・アプリケーションのための手法です。

アプリケーションは、アンロックされた行が更新または削除操作の前に変更されることはないという楽観的な仮定の下で作成されます。行が変更されると更新または削除は失敗します。アプリケーション・ロジックはその種の失敗を、選択を再試行するなどして処理します。オプティミスティック・ロックの 1 つの利点は、並行性の向上です。別のアプリケーションはその行の読み取り/書き込みが行えるからです。 1 つの欠点は、アプリケーション内での再試行ロジックが増えることです。ビジネス・トランザクションにデータベース・トランザクションとの相関がない 3 層環境ではオプティミスティック・ロックの手法が使用されます。ビジネス・トランザクションを越えてロックを維持することができないからです。

DB2 アプリケーションは現在、探索済み UPDATE ステートメントを作成することにより、*値によるオプティミスティック・ロック* を使用できるようにします。このステートメントは、選択されたものと全く同じ値を持つ行を検索します。行の列値が変更されると探索済み UPDATE は失敗します。しかし、値によるオプティミスティック・ロックにはいくつかの欠点があります。

- オプティミスティック・ロックの使用時の条件である *false positives* が識別される。これにより最初に再度選択されないかぎり、選択されて以降変更された 行は

更新できない。 (これは、最初に再度選択されないかぎり、選択されて以降未変更の 行を更新することができない条件である *false negatives* と対比することができます。)

- アプリケーションによる UPDATE 検索条件の作成が複雑である
- DB2 サーバーが値に基づいてターゲット行を検索するのは非能率的である
- 一部のクライアントとデータベースのデータ・タイプ (例えばタイム・スタンプ) が一致しないため、すべての列を探索済み UPDATE で使用できるわけではない

バージョン 9.5 では、*false positive* を持たない、より簡単かつ高速なオプティミスティック・ロックのサポートが追加されています。このサポートは、以下の新しい SQL 関数、式、およびフィーチャーを使って追加されます。

- 行 ID (RID_BIT または RID) 組み込み関数
- ROW CHANGE TOKEN 式
- 時間に基づく更新の検出
- 暗黙的な隠し列

このプログラミング・モデルを使用するアプリケーションにとって、拡張オプティミスティック・ロック・フィーチャーが有効です。このプログラミング・モデルを使用しないアプリケーションは、オプティミスティック・ロック・アプリケーションと見なされず、引き続き今までどおりに機能することに注意してください。

行 ID (RID_BIT または RID) 組み込み関数

この組み込み関数は SELECT リストまたは述部ステートメントで使用できます。例えば、述部 WHERE RID_BIT(tab)=? では、行を効率的に見つけるために、RID_BIT equals 述部が新規の直接アクセス方式としてインプリメントされます。以前は、選択されたすべての列値を述部に追加し、単一行だけを修飾する一部の固有列の組み合わせに依存することにより、効率の劣るアクセス方式で「値付きオプティミスティック・ロック」と言われる値が行われていました。

ROW CHANGE TOKEN 式

この新しい式では、トークンを BIGINT として返します。トークンは、行の変更順序内の相対点を表します。アプリケーションは行の現行の行変更トークン値と、行が最後にフェッチされたときに保管された行変更トークン値とを比較して、行が変更されたかどうかを判別することができます。

時間に基づく更新の検出:

このフィーチャーは、ROW CHANGE TIMESTAMP 式を使って SQL に追加されます。このフィーチャーをサポートするには、タイム・スタンプ値を保管するために、表内に新規生成の行変更タイム・スタンプ列を定義する必要があります。これは ALTER TABLE ステートメントを使って既存の表に追加するか、または新規表を作成するときに行変更タイム・スタンプ列を定義することができます。行変更タイム・スタンプ列の存在はオプティミスティック・ロックの動作にも影響を与えます。つまり、行変更トークンの細分性をページ・レベルから行レベルに上げるためにこの列が使用され、これによってオプティミスティック・ロック・アプリケーションに大きな益がもたらされることがあります。このフィーチャーは DB2 for z/OS® にも追加されています。

暗黙的な隠し列:

互換性のために、このフィーチャーは行変更タイム・スタンプ列を既存の表とアプリケーションに簡単に採用できるようにします。暗黙的な列リストを使用する場合、暗黙的な隠し列は外部化されません。例:

- 表に対して SELECT * を発行しても、結果表に暗黙的な隠し列は戻されません。
- 列リストを持たない INSERT ステートメントは暗黙的な隠し列の値を予期しませんが、列で NULL を許可するかまたは列が別のデフォルト値を持つように定義する必要があります。

注: オプティミスティック・ロックの用語、例えば *オプティミスティック並行性制御*、ペシミスティック・ロック、*ROWID*、および*更新検出* などの定義については、DB2 の用語集を参照してください。

オプティミスティック・ロックの制限と考慮事項

このトピックでは、オプティミスティック・ロックに関して考慮する必要のある制限事項をリストします。

- ROW CHANGE TIMESTAMP 列は、以下のキー、列、および名前ではサポートされません (使用した場合、sqlstate 429BV が戻されます)。
  - 主キー
  - 外部キー
  - マルチディメンション・クラスタリング (MDC) 列
  - 範囲パーティション列
  - データベース・ハッシュ・パーティション・キー
  - DETERMINED BY 制約列
  - ニックネーム
- RID() 関数は Database Partitioning Feature (DPF) 構成ではサポートされません。
- オプティミスティック・ロック・シナリオでは、フェッチ操作と更新操作の間にオンラインまたはオフラインで表の再編成 (REORG) を実行すると更新に失敗する場合がありますが、これは通常のアプリケーション再試行ロジックで処理できます。
- バージョン 9.5 では、IMPLICITLY HIDDEN 属性はオプティミスティック・ロック用の ROW CHANGE TIMESTAMP 列だけに制限されています。
- 既存の表に ROW CHANGE TIMESTAMP 列が追加された場合、すべての行がマテリアライズされたことが保証されるまで、インプレース再編成は制限されます (このエラーが発生すると SQL2219、理由コード 13 が戻されます)。 LOAD REPLACE コマンドまたは従来型の表の REORG を使用すれば、これが可能です。これにより、*false positive* を防ぐことができます。 ROW CHANGE TIMESTAMP 列を使って作成された表には、この制限がありません。

暗黙的な隠し列についての考慮事項

IMPLICITLY HIDDEN として定義された列は、SELECT リスト内で * が指定された照会の結果表には含まれません。ただし、照会の中で暗黙的な隠し列を明示的に参照することができます。

挿入時に列リストが指定されない場合、挿入の VALUES 節または SELECT LIST にこの列を含めることはできません (一般的に、これは生成される列、省略可能な列、または NULL 可能な列でなければなりません)。

例えば、SELECT リストや照会の述部の中で暗黙的な隠し列を参照することができます。さらに、CREATE INDEX ステートメント、ALTER TABLE ステートメント、INSERT ステートメント、MERGE ステートメント、または UPDATE ステートメントの中で暗黙的な隠し列を明示的に参照することができます。参照制約の中で暗黙的な隠し列を参照することもできます。列リストを含まない REFERENCES 節は、親表の主キーを暗黙的に参照します。親表の主キーに、暗黙的な隠し列として定義された列を含めることが可能です。そのような参照制約は許容されます。

- マテリアライズ照会定義の全選択の SELECT リストで暗黙的な隠し列が明示的に参照される場合は、その列がマテリアライズ照会表に含まれます。そうでない場合は、暗黙的な隠し列を含む表を参照するマテリアライズ照会表には暗黙的な隠し列が含まれません。
- ビュー定義 (CREATE VIEW ステートメント) の全選択の SELECT リストで暗黙的な隠し列が明示的に参照される場合は、その列がビューに含まれます (ただし、ビューの列は「隠し列」とは見なされません)。そうでない場合は、暗黙的な隠し列を含む表を参照するビューには暗黙的な隠し列が含まれません。

ラベル・ベースのアクセス制御 (LBAC) についての考慮事項

列が LBAC で保護されている場合、その列に対するユーザー・アクセスは、LBAC ポリシーおよびユーザーのセキュリティー・ラベルによって決定されます。この保護が行変更タイム・スタンプ列に適用される場合、その列に由来する ROW CHANGE TIMESTAMP 式および ROW CHANGE TOKEN 式を介して、その列の参照にまで保護が拡張されます。

このため、表のセキュリティー・ポリシーを決定する際には、オプティミスティック・ロックまたは時間に基づく更新検出を必要に応じて使用するすべてのユーザーが行変更タイム・スタンプ列にアクセスできることを確認してください。行変更タイム・スタンプ列が存在しない場合、ROW CHANGE TOKEN 式を LBAC によってブロックできないことに注意してください。ただし、表を修正して行変更タイム・スタンプ列を追加した場合には、LBAC に関するすべての考慮事項が該当するようになります。

行変更トークンの細分性および *false negative*

RID_BIT() 組み込み関数と行変更トークンは、オプティミスティック・ロックで唯一の要件です。ただし、表のスキーマもオプティミスティック・ロックの動作に影響を与えます。

例えば、以下のステートメント節のいずれかを使用して行変更タイム・スタンプ列を定義すると、DB2 サーバーは行が最後に変更された時刻 (または最初に挿入された時刻) を保管します。これは、行に対して最後に行われた変更のタイム・スタンプをキャプチャーする手段となります。これがタイム・スタンプ列であり、ユーザー提供の入力値の受け入れに GENERATED BY DEFAULT 節が使用されない限り、データベース・マネージャーによって保守されます。

```
GENERATED ALWAYS FOR EACH ROW ON UPDATE AS ROW CHANGE TIMESTAMP
GENERATED BY DEFAULT FOR EACH ROW ON UPDATE AS ROW CHANGE TIMESTAMP
```

したがって、アプリケーションが表で新規の ROW CHANGE TOKEN 式を使用するときには、次の 2 つの可能性を考慮する必要があります。

- *表に行変更タイム・スタンプ列が存在しない:* A ROW CHANGE TOKEN 式は、同じページ上にあるすべての行によって共有される派生 BIGINT 値を戻します。ページ上の行の 1 つが更新されると、同じページにあるすべての行の行変更トークンも変更されます。このことは、別の行に対して変更が行われると更新が失敗する可能性があることを意味します。この特性を、false negative といいます。

  注: このモードは、アプリケーションが *false negative* を許容し、追加ストレージを ROW CHANGE TIMESTAMP 列の各行に追加しない場合にのみ使用してください。
- *表に行変更タイム・スタンプ列が存在する:* ROW CHANGE TOKEN 式は、列のタイム・スタンプ値から派生する BIGINT 値を戻します。この場合、(まれではありますが) *false negative* が発生する場合があります。 *false negative* が生じ得るのは、表が再編成または再分散されたときに、行が移動され、アプリケーションが以前の RID_BIT() 値を使用するような場合です。

時間に基づく更新の検出

アプリケーションによっては、特定の時刻範囲でのデータベース更新を認識しなければならないものがあります。それはデータのレプリケーション、監査シナリオ、その他に使用される場合があります。新規の ROW CHANGE TIMESTAMP 式はこの情報を提供します。

この式は行が最後に変更された時間を表すタイム・スタンプを、CURRENT TIMESTAMP に似た現地時間で表現して戻します。行が更新されている場合、最新の更新時間がその行に反映されます。それ以外の場合、値は行の元の挿入に相当します。

ROW CHANGE TIMESTAMP の値はデータベースの表または表パーティションごとの行ごとに固有です。つまり、データベース・パーティションにつきすべての行が固有であるわけではなく、同じ表内の行のみが固有です。この値は行の変更順序を表します。より最近変更された行は、それよりも前に変更された行よりも必ず時刻が後の値を持っています。この値は、後の値が常に前の値より時刻が後になるため、以下の場合システム・クロックと同期しなくなることがあります。

- システム・クロックが変更される。
- 行変更タイム・スタンプ列が GENERATED BY DEFAULT (データ伝搬のみを意図) で、行に同期しない値が提供されている。

ROW CHANGE TIMESTAMP 式を使用するための前提条件は、表に行変更タイム・スタンプ列が定義されていなければならないというものです。すべての行は挿入または最後に更新された時間のタイム・スタンプを返します。行変更タイム・スタンプ列を表に組み込む方法として、次の 2 つがあります。

- *表が行変更タイム・スタンプ列を組み込んで作成されている。* ROW CHANGE TIMESTAMP 式は列の値を戻します。このカテゴリーの場合、タイム・スタンプは正確です。行変更タイム・スタンプは、データベースによって生成される場合、通常挿入の速度と DST 調整を含む可能なクロック操作によって制限を受けます。

- *表が行変更タイム・スタンプ列を組み込んで作成されていない* が、後で ALTER TABLE ステートメントで追加された。 ROW CHANGE TIMESTAMP 式は列の値を戻します。このカテゴリーの場合、古い (変更前の) 行が初めて更新されるか、またはオフライン表の再編成が実行されるまで、その行には実際のタイム・スタンプは含まれません。

  注: タイム・スタンプはデータベースにおいて実際の更新が行われたおおよその時間で、システム・クロックの時間を基準としており、データベース/表パーティション内でタイム・スタンプを繰り返すことはできないという制限を考慮に入れています。実際に通常は、更新時間を非常に正確に表したものとなっています。行変更タイム・スタンプは、データベースによって生成される場合、通常挿入の速度と DST 調整を含む可能なクロック操作によって制限を受けます。

  ALTER TABLE ステートメント以降に更新されていない行は、列にデフォルト・タイプの値を戻します。これは、1 年 1 月 1 日午前 0 時です。更新された行だけが固有のタイム・スタンプを持つことになります。オフライン表の再編成によってマテリアライズされたタイム・スタンプを持つ行は、表の再編成時に生成された固有のタイム・スタンプを戻します。 INPLACE オプションを使用した再編成ではスキーマ変更がマテリアライズされないため、こうした再編成では不十分です。

いずれの場合も、行のタイム・スタンプは、再配布が実行される場合にも更新されることがあります。再配布中に行があるデータベース・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る場合、ターゲットで固有であることが保証されている新規のタイム・スタンプが生成されなければなりません。

ROW CHANGE TIMESTAMP について生成される時刻値

パーティションごとに固有値が強制されるため、行変更タイム・スタンプ列について生成された正確な値に関するいくつかの境界条件があります。

システム・クロックが DB2 サーバー上でクロックの修正または夏時間調整時刻ポリシーのために、過去に合わせて調整される場合、そのタイム・スタンプは、システム・クロックの現行値または CURRENT TIMESTAMP 特殊レジスターと比べて、時間的に進んでいる可能性があります。このことは、タイム・スタンプがシステム・クロックの調整前に生成された場合に生じます。つまり、調整された時間より後は、固有性を維持するためにタイム・スタンプは常に昇順で生成されます。

REORG 操作または LOAD 操作の一部として表に追加された列でタイム・スタンプが生成されると、初期タイム・スタンプ値から始まって順次、ユーティリティーの処理の特定の時点でタイム・スタンプが生成されます。ユーティリティーがタイム・スタンプの細分度より速く行を処理できる (つまり、1 秒あたり百万行を超える) 場合、一部の行で生成される値も、システム・クロックまたは CURRENT TIMESTAMP 特殊レジスターと比べて、時間的に進んでいる可能性があります。

どの場合も、システム・クロックが行変更タイム・スタンプの値に追いつくと、行が挿入された時刻の近似値があるはずです。そうした時刻まで、タイム・スタンプはタイム・スタンプ・タイプで許可される最も細かい細分度で昇順で生成されます。

RID_BIT() および RID() 組み込み関数

RID_BIT() および行変更トークン を表の中のすべての行について選択できます。SELECT (選択) は、アプリケーションで必要とされる分離レベルで行うことができます。

アプリケーションは以下を検索することにより、オプティミスティック・ロックを使用して同じ (未変更の) 行を UPDATE (更新) できます。

- 更新ターゲット行に直接アクセスする (スキャンしない) RID_BIT()
- これが同じ未変更の行であることを確認する行変更トークン

この更新 (または削除) は、同じ作業単位内で、または接続境界を超える場合であっても、選択後の任意の時点で行うことができます。ある時点で特定の行について上記の 2 つの値を取得していることだけがその要件です。

オプティミスティック・ロックは“WebSphere 指向のプログラミング・モデル”で使用されます。例えば、Microsoft .NET はこのモデルを使用して、以下のようにして、SELECT ステートメントを処理し、その後に UPDATE または DELETE ステートメントを処理します。

- データベース・サーバーに接続し、希望する行を表から SELECT (選択) します。
- データベースから切断するか、行ロックを解放します。そのようにして、他のアプリケーションは、そのアプリケーションによって保持されたロックおよびリソースによる並行性の競合を発生させることなく、データの読み取り、更新、削除、および挿入を行うことができます。 (分離「非コミット読み取り」により、高度な並行性が許可され、かつ他のアプリケーションが更新および削除トランザクションを COMMIT (コミット) すると想定して、このオプティミスティック・ロック・アプリケーションは更新値を読み取り、楽観的に検索された更新/削除は成功します)。
- SELECT (選択) された行データに対していくつかのローカル計算を実行します。
- データベース・サーバーに再接続し、1 つ以上の特定のターゲット行で UPDATE または DELETE を検索します (さらに、ターゲット行が変更されている場合は、失敗した UPDATE または DELETE ステートメントを処理します)。

このプログラミング・モデルを使用するアプリケーションにとって、拡張オプティミスティック・ロック・フィーチャーが有効です。このプログラミング・モデルを使用しないアプリケーションは、オプティミスティック・ロック・アプリケーションと見なされず、引き続き今までどおりに機能することに注意してください。

RID_BIT() および RID() 組み込み関数フィーチャー

拡張オプティミスティック・ロックおよび更新検出用にインプリメントされた新規フィーチャーは以下のとおりです。

RID_BIT(<table designator>)

行のレコード ID (RID) を VARCHAR(16) FOR BIT DATA として戻す新規の組み込み関数。

注: DB2 for z/OS は、組み込み関数 RID を戻りタイプ BIGINT でインプリメントしますが、これは Linux、UNIX、およびWindows RID には十分な

大きさではありません。互換性を保つために、この RID() 組み込み関数は、RID_BIT() に加えて BIGINT を戻します。

この RID() 組み込み関数は DPF 環境では機能しません。また表のバージョン情報を含んでいません。それ以外は、RID_BIT と同じように機能します。 z/OS サーバーに移植されるアプリケーションをコーディングする場合にのみ、この組み込み関数を使用する必要があります。このトピックでは、必要な箇所を除いて、RID_BIT にのみ言及しています。

RID_BIT() 組み込み関数
この組み込み関数は SELECT リストまたは述部ステートメントで使用できます。例えば、述部 `WHERE RID_BIT(tab)=?` では、行を効率的に見つけるために、RID_BIT equals 述部が新規の直接アクセス方式としてインプリメントされます。以前は、選択されたすべての列値を述部に追加し、単一行だけを修飾する一部の固有列の組み合わせに依存することにより、効率の劣るアクセス方式で*値付きオプティミスティック・ロック* と言われる値が行われていました。

ROW CHANGE TOKEN FOR <table designator>
トークンを BIGINT として戻す新しい式。トークンは、行の変更順序内の相対点を表します。アプリケーションは行の現行の行変更トークン値と、行が最後にフェッチされたときに保管された行変更トークン値とを比較して、行が変更されたかどうかを判別することができます。

ROW CHANGE TIMESTAMP 列
以下のいずれかとして定義できる、デフォルト・タイプ TIMESTAMP の GENERATED 列。

```
GENERATED ALWAYS FOR EACH ROW ON UPDATE
AS ROW CHANGE TIMESTAMP
```

または (データ伝搬またはアンロードおよび再ロード操作にのみ推奨されます)

```
GENERATED BY DEFAULT FOR EACH ROW ON UPDATE
AS ROW CHANGE TIMESTAMP
```

この列のデータは行が変更されるたびに変化します。この列が定義されるとき、ROW CHANGE TOKEN 値はこの列から派生します。 GENERATED ALWAYS が使用される場合、データベース・マネージャーはこの値がデータベース・パーティション内または表パーティション内で固有であり、*false positives* の可能性がないことを確かめます。

最初の 2 つのエレメント、RID_BIT および ROW CHANGE TOKEN を使用するために、その他の変更をデータベース・スキーマに加える必要はありません。ただし、ROW CHANGE TIMESTAMP 列がない場合、行変更トークンは同じページ上のすべての行によって共有されることに注意してください。ページ上の行を更新すると、同じページに保管された他の行について *false negatives* が発生することがあります。この列を使用する場合、ROW CHANGE TOKEN はタイム・スタンプから派生し、表またはデータベース・パーティション内のその他の行とは共有されません。 280 ページの『行変更トークンの細分性および *false negative*』 を参照。

時間に基づく更新の検出フィーチャー

表指定子によって識別される表の中の行が最後に変更された時刻を表すタイム・スタンプ値を戻す新しい式。

```
ROW CHANGE TIMESTAMP FOR <table designator>
```

ROW CHANGE TIMESTAMP 式は、ROW CHANGE TIMESTAMP 列を持たない表についてはサポートされていません。

ROW CHANGE TIMESTAMP 式は時間に基づく更新検出のシナリオにのみ使用されます。ここでは、表指定子によって識別される表について行変更タイム・スタンプ列が定義されている必要があります。この列はデータベース・マネージャーによって管理され、ROW CHANGE TIMESTAMP 式によって戻されるタイム・スタンプ値を保管するために使用されます。このタイム・スタンプは、各データベース・パーティションの行ごとにデータベースによって割り当てられるときに固有であることが保証されるという点で、CURRENT TIMESTAMP とは異なります。これは、挿入または更新される個々の行の変更時刻のローカル・タイム・スタンプ近似値です。

注: これらの 2 つのフィーチャー (つまり、RID_BIT() および RID() 組み込み関数と時間に基づく更新検出フィーチャー) には相互関係があるにもかかわらず、ROW CHANGE TOKEN と ROW CHANGE TIMESTAMP 式の使用は交換可能でないこと、特に ROW CHANGE TIMESTAMP 式はオプティミスティック・ロックの使用の一部ではないことに注意するのは重要です。

オプティミスティック・ロックの使用可能化の計画

オプティミスティック・ロックの新規の SQL 式および属性は、関係する表に対して DDL 変更を加えずに使用できるため、テスト・アプリケーションでオプティミスティック・ロックを簡単に試すことができます。

DDL 変更がない場合、オプティミスティック・ロック・アプリケーションは DDL 変更がある場合よりも *false negatives* が発生することがあります。 false negative は非常に多くの再試行を繰り返すことがあるため、false negative が発生するアプリケーションは実稼働環境で正しくスケールされない可能性があります。そのため、false negative を回避するには、オプティミスティック・ロック・ターゲット表が次のいずれかになっている必要があります。

- ROW CHANGE TIMESTAMP 列とともに作成されている。
- ROW CHANGE TIMESTAMP 列を含むように変更されている。

推奨されている DDL 変更を実行すると、false negative の数はかなり減るはずです。様々な行で作動する同時アプリケーションではなく、再編成などの表レベルの操作によってのみ、false negative は発生します。

一般に、データベース・マネージャーは false negative (例えば、オンラインまたはオフライン再編成) を許可しており、行変更タイム・スタンプ列が存在すればページまたは行レベルの細分度が使用されるかどうかを判別できます。また、SYSCAT.COLUMNS を照会して、ROWCHANGETIMESTAMP 列に YES が指定されている行を持つ表を調べることができます。

アプリケーションおよびデータベースを徹底的に分析すると、この DDL が必要ないことを示す場合があります。例えば、ページごとに 1 行が存在する場合、または

同一データ・ページ上で更新および削除操作がめったに行われないか、全く行われない場合などです。そうした分析は例外です。

更新タイム・スタンプ検出を使用する場合、表の DDL を変更し、場合によっては表を再編成して値をマテリアライズしなければならないことがあります。こうした変更が実動データベースに悪影響を与えることが懸念される場合、最初にテスト環境で変更を試す必要があります。例えば、余分な列があると、行サイズの制限およびプランの選択に影響を与えることがあります。

知っておくべき条件

- システム・クロックおよびタイム・スタンプ値の細分度について知っておくべき条件があります。表に ROW CHANGE TIMESTAMP 列がある場合、挿入または更新後に、新規行にはそのデータベース・パーティション上のその表に固有の ROW CHANGE TIMESTAMP 値があります。
- 固有であることを保証するために、システム・クロックが過去にさかのぼって調整されているか、またはデータの更新あるいは挿入がタイム・スタンプの細分度より速く発生しているかどうかに関係なく、行の生成済みタイム・スタンプは常に大きくなります。そのため、ROW CHANGE TIMESTAMP は今後、システム時刻および DB2 の CURRENT TIMESTAMP 特殊レジスターと比較される可能性があります。システム・クロックの同期が完全にずれたり、データベース・マネージャーが 1 秒に百万行を超えるペースで挿入または更新を行っているのでない限り、これは通常は実際の時刻に非常に近くなります。 CURRENT TIMESTAMP とは対照的に、この値も更新の時刻に行ごとに生成されるため、通常は CURRENT TIMESTAMP よりはるかに近くなります。CURRENT TIMESTAMP はステートメント全体について 1 回生成されますが、これは複雑さや影響を受ける行数に応じて完了するのに非常に長い時間がかかることがあります。

アプリケーションにおけるオプティミスティック・ロックの使用可能化

アプリケーションでオプティミスティック・ロックのサポートを使用可能にするために、実行する必要のあるいくつかのステップがあります。

1. 最初の照会で、処理する必要のある各行の行 ID (283 ページの『RID_BIT() および RID() 組み込み関数』を使用) および行変更トークンを SELECT します。
2. 行ロックを解放し、他のアプリケーションが表に対して SELECT、INSERT、UPDATE、および DELETE を実行できるようにします。
3. 検索条件に行 ID と行変更トークンを使用して、ターゲット行に対して検索 UPDATE または DELETE を実行します。ロックを解除された行は、元の SELECT ステートメント以来、未変更であるとの楽観的な前提に基づいてこれを行います。
4. 行が変更されていた場合は、UPDATE 操作が失敗し、アプリケーションのロジックでこの失敗を処理しなければなりません。例えば、アプリケーションが SELECT および UPDATE 操作を再試行します。

上記のステップを実行した後、次のようにします。

- アプリケーションの実行する再試行の回数が予想回数あるいは望ましい回数よりも多いようであれば、行変更タイム・スタンプ列を表に追加することにより、

RID_BIT 関数で識別された行に変更が加えられる場合に限って、単に行変更トークンを無効にし、同じデータ・ページに対する他のアクティビティーについてはこれが生じないようにします。

- 一定の時刻範囲に挿入または更新された行を調べるには、表を作成または変更して、これに行変更タイム・スタンプ列を含めます。この列はデータベース・マネージャーによって自動的に保守され、列名または ROW CHANGE TIMESTAMP 式のいずれかを使用して照会することができます。
- 行変更タイム・スタンプ列にのみ当てはまる点として、列を IMPLICTLY HIDDEN 属性を使って定義すると、表の列への暗黙的な参照がある場合に、この列は外部化されません。しかし、暗黙的な隠し列を SQL ステートメントで明示的に参照することはいつでも可能です。これは、表に列を追加すると、暗黙的な列のリストを使用する既存のアプリケーションに障害が起こる場合に、有用なものとなり得ます。

表パーティション化およびデータ編成スキーム

表のパーティション化とは、表の 1 つ以上のパーティション列にある値に従って表データを複数のデータ・パーティションに分割するデータ編成スキームのことです。指定された表のデータは、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。これらのオブジェクトは複数の表スペースに存在することが可能です。

表のパーティション化とデータ編成スキームについてのすべての詳細は、『パーティショニングおよびクラスタリング・ガイド』を参照してください。

表の作成

表に保管されるデータの変更、およびデータへのアクセスは、データベース・マネージャーによって制御されます。 CREATE TABLE ステートメントを使用すれば、表を作成することができます。複雑なステートメントを使用して、表のあらゆる属性や特性を定義できます。ただし、すべての点でデフォルトを使用するならば、表を作成するステートメントは非常に簡単です。

```
CREATE TABLE <table name> (<column name> <data type> <column options>,
                           (<column name> <data type> <column options>, ...)
```

ここで、<table name> に修飾子を含めても含めなくてもかまいません。この名前は、システム・カタログ内のすべての表、ビュー、および別名と比較して固有でなければなりません。また、SYSIBM、SYSCAT、SYSFUN、または SYSSTAT という名前にすることはできません。

<column name> は、表内の列の名前を指定します。この名前を修飾することはできず、表内の他の列と比べて固有でなければなりません。

ある列の属性を詳細に定義するために、列オプション <column options> を指定することもできます。オプションには、列が NULL 値になるのを防ぐ NOT NULL のほかに、LOB データ・タイプ用の特殊なオプション、参照型の列の SCOPE、列に関する制約、列のデフォルトなどがあります。詳しくは、CREATE TABLE ステートメントを参照してください。

グローバル一時表の宣言

アプリケーション内からグローバル一時表を作成するには、DECLARE GLOBAL TEMPORARY TABLE ステートメントを使用します。

グローバル一時表は、ユーザー定義の一時表とも呼ばれ、データベース内のデータを扱うアプリケーションによって使用されます。データの操作の結果は、一時的に表に保管する必要があります。 USER TEMPORARY 表スペースは、グローバル一時表が作成される前に存在していなければなりません。

注: グローバル一時表の記述は、システム・カタログには現れません。したがって、この表を他のアプリケーションのために保持したり、他のアプリケーションと共用したりすることはできません。この表を使用するアプリケーションが終了したりデータベースから切断されたりすると、表の中のデータはすべて削除され、表は暗黙的にドロップされます。
グローバル一時表は、以下のものをサポートしません。

- LOB タイプの列 (または LOB に基づく特殊タイプの列)
- ユーザー定義タイプの列
- LONG VARCHAR 列
- XML 列

例

```
DECLARE GLOBAL TEMPORARY TABLE gbl_temp
   LIKE empltabl
   ON COMMIT DELETE ROWS
   NOT LOGGED
   IN usr_tbsp
```

このステートメントにより、gbl_temp というグローバル一時表が作成されます。この表の列は、名前と記述が empltabl の列と完全に一致するように定義されています。暗黙定義には、列名、データ・タイプ、NULL 可能特性、および列のデフォルト値の属性だけが含まれます。他のすべての列属性 (ユニーク制約、外部キー制約、トリガー、索引を含む) は、定義されていません。 COMMIT 操作を実行すると、表で WITH HOLD カーソルがオープンしていなければ、表の中のデータはすべて削除されます。ユーザー一時表に対する変更内容はログに記録されません。グローバル一時表は、指定された USER TEMPORARY 表スペースに置かれます。この表スペースがないと、この表の宣言は失敗します。

ROLLBACK または ROLLBACK TO SAVEPOINT がこの表の作成時に指定される場合、表中のすべての行を削除する (DELETE ROWS、これがデフォルトです) か、表の行を保存する (PRESERVE ROWS) ことを指定できます。

この表は、アプリケーションがデータベースから切断されるときに、暗黙的にドロップされます。

既存の表の類似表の作成

ATTACH PARTITION 節を指定して ALTER TABLE ステートメントを発行したときにターゲット表の特性がソースの特性と十分一致しないときには、ソース表を新規作成する必要があるかもしれません。ソース表を新規作成する前にまず、既存のソース表とターゲット表の間のミスマッチを訂正してみることができます。

表を作成するには、ステートメントの許可 ID によって保持される特権に、以下の権限および特権の少なくとも 1 つが含まれていなければなりません。

- データベースに対する CREATETAB 権限、表スペースに対する USE 特権、および以下のいずれか。
  - データベースに対する IMPLICIT_SCHEMA 権限 (表の暗黙または明示スキーマ名がない場合)
  - スキーマに対する CREATEIN 特権 (表のスキーマ名が既存のスキーマを指す場合)
- SYSADM または DBADM 権限

ミスマッチの訂正に失敗した場合には、エラー SQL20408N または SQL20307N が戻されます。

新規ソース表を作成するには、以下のようにします。

1. db2look コマンドを使用して、ターゲット表と同じ表を作成するための CREATE TABLE ステートメントを生成する。

   ```
   db2look -d <source database name> -t <target database name> -e -p
   ```

2. db2look 出力からパーティション節を除去し、作成された表の名前を新規の名前 (この例では sourceC) に変更する。
3. 次に、LOAD FROM CURSOR コマンドを使用して、元のソース表にあるすべてのデータを、新しく作成されたソース表 sourceC にロードする。

   ```
   DECLARE mycurs CURSOR FOR SELECT * FROM source
   LOAD FROM mycurs OF CURSOR REPLACE INTO sourceC
   ```

   元のデータに sourceC 表の定義に適合しないためにこのコマンドが失敗する場合、 sourceC に転送される際に、元の表のデータを変換する必要があります。
4. データが正常に sourceC にコピーされた後、ALTER TABLE target ...ATTACH sourceC ステートメントをサブミットする。

ステージング・データ用の表の作成

ステージング表 は、据え置きマテリアライズ照会表の増分保守サポートを可能にします。ステージング表は、マテリアライズ照会表を基礎表の内容で同期するためにマテリアライズ照会表に適用する必要がある変更を収集します。ステージング表を使用するなら、マテリアライズ照会表の即時リフレッシュが要求された時に、即時保守操作により引き起こされる高いロック競合を除去します。さらに、マテリアライズ照会表は、もはや REFRESH TABLE が実行される時ごとに全体を再生成する必要はありません。

マテリアライズ照会表は、複雑な照会をする時の応答時間を向上させる強力な手段であり、とりわけ以下の操作のいくつかを必要とする照会に威力を発揮します。

- 1 ディメンション以上の集約データ
- 表のグループの結合データおよび集約データ
- 通常アクセスされるデータのサブセットからのデータ
- 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

以下は、ステージング表に関連する重要な制約事項です。

1. ステージング表が作成されるマテリアライズ照会表を定義するために使用される照会は、増分的に保守していけるものでなければなりません。すなわちその照会は、即時リフレッシュ・オプションを持つマテリアライズ照会表と同じ規則に従う必要があります。
2. 据え置きリフレッシュだけが、サポートするステージング表を持つことができます。さらに照会は、そのステージング表に関連したマテリアライズ照会表を定義します。マテリアライズ照会表は、REFRESH DEFERRED で定義しなければなりません。
3. ステージング表を使用してリフレッシュする時は、現時点のリフレッシュのみがサポートされます。
4. パーティション化された階層表、およびパーティション化された型付き表はサポートされていません。 (パーティション表は、CREATE TABLE ステートメントの PARTITION BY 節での指定に基づいてデータが複数のストレージ・オブジェクトにパーティション化されている表のことです。)

不整合、不完全、またはペンディング状態のステージング表は、いくつかの操作を行わない限り、関連したマテリアライズ照会表を増分的にリフレッシュするために使用することはできません。それらの操作は、ステージング表の内容を、それに関連したマテリアライズ照会表およびその基礎表と整合させ、ステージング表をペンディング状態から解除します。マテリアライズ照会表のリフレッシュに続き、そのステージング表の内容はクリアされ、ステージング表は正常状態に設定されます。ステージング表は、SET INTEGRITY ステートメントに適切なオプションを付けて使用することにより、意図的に整理することもできます。整理すると、そのステージング表は不整合状態に変更されます。例えば、以下のステートメントを実行すると、STAGTAB1 というステージング表を強制的に整理します。

```
SET INTEGRITY FOR STAGTAB1 PRUNE;
```

ステージング表が作成される時、それはペンディング状態に置かれ、さらに表が基礎表および関連したマテリアライズ照会表の内容に関して不整合または不完全であることを示す標識を持ちます。そのステージング表は、その基礎表から変更内容の収集を開始するために、ペンディングおよび不整合状態から解除される必要があります。ペンディング状態にある時には、ステージング表に関連する基礎表に対するいづれの変更も失敗し、関連したマテリアライズ照会表をリフレッシュしようとしても失敗します。

ステージング表のペンディング状態を解除するには、いくつかの方法があります。

- SET INTEGRITY FOR <ステージング表名> STAGING IMMEDIATE UNCHECKED
- SET INTEGRITY FOR <ステージング表> IMMEDIATE CHECKED

表の変更

このセクションには、表を変更する方法についてのトピックが含まれています。

マテリアライズ照会表のプロパティーの変更

多少の制約事項はありますが、マテリアライズ照会表を正規表に変更したり、正規表をマテリアライズ照会表に変更したりできます。他の表タイプは変更できません。変更できるのは正規表とマテリアライズ照会表だけです。例えば、複製マテリアライズ照会表から正規表に (あるいはその逆に) 変更することはできません。

正規表をマテリアライズ照会表に変更すると、その表は整合性設定ペンディング状態になります。このように変更した場合、マテリアライズ照会表の定義を全選択したものは変更前の表の定義と一致していなければなりません。つまり、以下の点を満たしていなければなりません。

- 列の数が同じでなければならない。
- 列名と位置が一致しなければならない。
- データ・タイプが同一でなければならない。

マテリアライズ照会表が元の表で定義されている場合、元の表自体を変更してマテリアライズ照会表にすることはできません。元の表にトリガー、チェック制約、参照制約、または定義済みのユニーク索引がある場合、その表をマテリアライズ照会表に変更することはできません。マテリアライズ照会表を定義するために表の特性を変更する場合、同じ ALTER TABLE ステートメントを使って、別の方法で表を変更することはできません。

正規表をマテリアライズ照会表に変更する場合、マテリアライズ照会表の定義の全選択では、元の表を直接参照したり、あるいはビュー、別名、またはマテリアライズ照会表を介して間接的に参照することはできません。

マテリアライズ照会表を正規表に変更するには、以下のようにします。

```
ALTER TABLE sumtable
   SET SUMMARY AS DEFINITION ONLY
```

正規表をマテリアライズ照会表に変更するには、以下のようにします。

```
ALTER TABLE regtable
  SET SUMMARY AS <fullselect>
```

正規表をマテリアライズ照会表に変更する際の全選択に関する制約事項は、CREATE SUMMARY TABLE ステートメントを使用してサマリー表を作成する際の制約事項によく似ています。

マテリアライズ照会表のデータのリフレッシュ

REFRESH TABLE ステートメントを使用して、 1 つ以上のマテリアライズ照会表のデータをリフレッシュできます。このステートメントは、アプリケーション・プログラムに組み込むこともできますし、動的に出すこともできます。このステートメントを使用するには、SYSADM または DBADM 権限か、更新する表に対する CONTROL 特権が必要です。

次の例は、マテリアライズ照会表のデータをリフレッシュする方法を示しています。

```
REFRESH TABLE SUMTAB1
```

列プロパティーの変更

列プロパティーを変更するには、ALTER TABLE ステートメントを使用します。列プロパティーには、NULL 可能、LOB オプション、有効範囲、制約および圧縮属性、データ・タイプなどがあります。詳しくは、ALTER TABLE ステートメントを参照してください。

表を変更するには、変更する表に対し、次の中から少なくとも 1 つの特権を持っている必要があります。

- ALTER 特権
- CONTROL 特権
- SYSADM または DBADM 権限
- 表のスキーマに対する ALTERIN 特権

既存の列の定義を変更する、表の列の変更時に SQL を編集しテストする、または表の列の変更時に関連したオブジェクトを検証するには、DBADM 権限が必要です。

例えば、コマンド行から以下のように入力します。

```
ALTER TABLE EMPLOYEE
   ALTER COLUMN WORKDEPT
      SET DEFAULT '123'
```

列の追加とドロップ

既存の表に列を追加したり、既存の表から列をドロップするには、それぞれ ADD COLUMN 節または DROP COLUMN 節を指定した ALTER TABLE ステートメントを使用します。表は型付き表であってはなりません。

表にすでに存在するすべての行では、新しい列の値はデフォルト値に設定されます。新しい列は表の最後の列になります。つまり、元の列数が *n* であれば、追加される列の番号は *n*+1 となります。新しい列を追加することにより、すべての列の合計バイト数が最大レコード・サイズを超えてはなりません。

列を追加するには、次のようなステートメントを発行します。

```
ALTER TABLE SALES
  ADD COLUMN SOLD_QTY
  SMALLINT NOT NULL DEFAULT 0
```

列を削除またはドロップするには、次のようなステートメントを発行します。

```
ALTER TABLE SALES
  DROP COLUMN SOLD_QTY
```

列定義の DEFAULT 節の変更

DEFAULT 節は、INSERT で値が提供されない場合に列のデフォルト値を提供します。または、INSERT や UPDATE で DEFAULT として指定されることもあります。 DEFAULT キーワードの後に特定のデフォルト値が指定されない場合、データ・タイプに応じてデフォルト値が決定されます。列が XML または構造化タイプとして定義されている場合には、DEFAULT 節は指定できません。

列定義で DEFAULT を省略した場合、『 265 ページの『デフォルトの列およびデータ・タイプの定義』』で説明されているように、列のデフォルトとして NULL 値が使用されます。

DEFAULT キーワードを使って指定できる値の種類については、ALTER TABLE ステートメントを参照してください。

列の生成または ID プロパティーの変更

ALTER TABLE ステートメントの ALTER COLUMN 節を使用して、表の列の生成または ID プロパティーを追加およびドロップできます。

次のいずれかを行うことができます。

- 既存の非生成列に対して作業を行うとき、生成式属性を追加できます。その結果、変更された列は生成列となります。
- 既存の生成列に対して作業を行うとき、生成式属性をドロップできます。その結果、変更された列は通常の、非生成列となります。
- 既存の非 ID 列に対して作業を行うとき、ID 属性を追加できます。その結果、変更された列は ID 列となります。
- 既存の ID 列に対して作業を行うとき、ID 属性をドロップできます。その結果、変更された列は通常の、非生成、非 ID 列となります。
- 既存の生成列に対して作業を行うとき、生成列を GENERATED ALWAYS から GENERATED BY DEFAULT の状態に変更できます。その逆も可能です。つまり、生成列を GENERATED BY DEFAULT から GENERATED ALWAYS の状態に変更できます。これが可能なのは、生成列に対して作業を行うときだけです。
- ユーザー定義のデフォルト列からデフォルト属性をドロップできます。これを行うと、新規のデフォルト値は NULL となります。
- 同じ ALTER COLUMN ステートメントで、デフォルト、ID、または生成属性をドロップしてから、新規のデフォルト、ID、または生成属性を設定できます。
- CREATE TABLE および ALTER TABLE ステートメントの両方で、『ALWAYS』 は GENERATED 節の中のオプションのワードです。つまり、ALTER TABLE ステートメント内で使用されるとき、GENERATED ALWAYS は GENERATED と同等になります。

列定義の修正

列をドロップしたり、列のタイプや属性を変更するには、ALTER TABLE ステートメントを使用します。例えば、既存の VARCHAR または VARGRAPHIC 列の長さを増やすことができます。文字数は、使用されるページ・サイズに従属する値まで増やすことができます。

列に関連付けられたデフォルト値を変更するには、いったん新しいデフォルト値を定義すれば、デフォルトの使用が指示されている後続のあらゆる SQL 操作で、その列の新しい値が使用されます。その新しい値は、割り当て規則に準拠している必要があり、 CREATE TABLE ステートメントに関して説明したものと同じ制約が課せられます。

注: 生成列のデフォルト値をこのステートメントで変更することはできません。

SQL を使用してこれらの表属性を変更する場合、表をドロップしてから再作成する必要はなくなりました。これは、オブジェクト従属関係が存在する場合には複雑になる、時間のかかるプロセスです。

コマンド行を使用して既存の表の列の長さおよびタイプを修正するには、以下のように入力します。

```
ALTER TABLE <table_name>
   ALTER COLUMN <column_name>
   <modification_type>
```

例えば、列を 4000 文字まで増やすには、次のような形式で指定します。

```
ALTER TABLE t1
   ALTER COLUMN colnam1
   SET DATA TYPE VARCHAR(4000)
```

別の例として、列が新しい VARGRAPHIC 値を持つことができるようにするには、次のようなステートメントを使用します。

```
ALTER TABLE t1
   ALTER COLUMN colnam2
   SET DATA TYPE VARGRAPHIC(2000)
```

型付き表の列を変更することはできません。しかし、有効範囲がまだ定義されていない既存の参照タイプ列に、有効範囲を追加することは可能です。例:

```
ALTER TABLE t1
   ALTER COLUMN colnamt1
   ADD SCOPE typtab1
```

コマンド行を使用して既存の表の列のデフォルト値を修正するには、以下のように入力します。

```
ALTER TABLE <table_name>
   ALTER COLUMN <column_name>
   SET DEFAULT 'new_default_value'
```

例えば、列のデフォルト値を変更するには、次のような形式で指定します。

```
ALTER TABLE t1
   ALTER COLUMN colnam1
   SET DEFAULT '123'
```

表と列の名前変更

RENAME ステートメントを使用すれば、既存の表の名前を変更できます。列の名前を変更するには、ALTER TABLE ステートメントを使用します。

表の名前を変更する際には、既存の定義 (ビューまたはマテリアライズ照会表)、トリガー、SQL 関数、または制約の中でソース表が参照されてはなりません。さらに、(ID 列を除く) 生成列や、親表または従属表が存在してはなりません。新しい表名を反映するようにカタログ項目が更新されます。詳細情報と例については、RENAME ステートメントを参照してください。

既存の列定義の変更については、『 292 ページの『列プロパティーの変更』』および ALTER TABLE ステートメントを参照してください。

作動不能サマリー表の回復

サマリー表は、基本表での SELECT 特権を取り消されると、*作動不能* になります。

次のステップは、作動不能サマリー表を回復するのに役に立ちます。

- サマリー表を作成するために最初に使用されたステートメントを判別する。この情報は SYSCAT.VIEW カタログ・ビューの TEXT 列から獲得することができます。
- CREATE SUMMARY TABLE ステートメントおよび同じ名前と同じ定義を使用して、サマリー表を再作成する。
- GRANT ステートメントを使用して、サマリー表に以前に付与されていたすべての特権を再度付与する。 (作動不能サマリー表に付与されていたすべての特権は取り消されていることに注意してください。)

作動不能サマリー表を回復したくない場合は、 DROP TABLE ステートメントを使用してそのサマリー表を明示的にドロップするか、または同じ名前と別の定義を使用して新規のサマリー表を作成することができます。

作動不能サマリー表は、SYSCAT.TABLES および SYSCAT.VIEWS カタログ・ビューにしか項目がありません。 SYSCAT.VIEWDEP、SYSCAT.TABAUTH、SYSCAT.COLUMNS、および SYSCAT.COLAUTH カタログ・ビューのすべての項目が除去されます。

表定義の表示

SYSCAT.COLUMNS カタログ・ビューを使用すれば、表の定義を表示できます。各行は、表、ビュー、またはニックネームに対して定義された列を表します。列のデータを表示するには、SELECT ステートメントを使用してください。

表またはビューの別名

別名 とは、表またはビューの代替名です。既存の表またはビューを参照できる 場合に、これを使用して表またはビューを参照することができます。

別名は、どのようなコンテキストでも使用できるというわけではありません。例えば、チェック制約のチェック条件では使用できません。別名は、宣言済み一時表を参照することはできません。

表やビューのように、別名も作成やドロップができ、関連するコメントを付けることができます。ただし、表とは異なり、別名はチェーニング と呼ばれるプロセスの中で互いを参照できます。別名は公開して参照される名前ですので、これを使用するために特別な権限や特権を必要とするわけではありません。しかし、別名によって参照される表やビューにアクセスするためには、これらオブジェクトに関連する許可が必要です。

データベース別名およびネットワーク別名などといった、他のタイプの別名もあります。別名は、フェデレーテッド・システム上のデータ表またはビューを参照するニックネーム に対して作成することもできます。

表のドロップ

表をドロップするには、DROP TABLE ステートメントを使用できます。表がドロップされると、その表についての情報が含まれる SYSCAT.TABLES システム・カタログ・ビュー内の行がドロップされます。その表に依存する他のオブジェクトがあれば、それらも影響を受けます。

例:

- すべての列名はドロップされます。
- その表の列について作成された索引はドロップされます。
- その表に基づくすべてのビューには作動不能のマークが付けられます。
- ドロップされた表と従属ビューに対するすべての特権が暗黙のうちに取り消されます。
- その表が親表または従属表となっている参照制約がすべてドロップされます。
- ドロップされた表に依存するすべてのパッケージおよびキャッシュに入った動的 SQL および XQuery ステートメントは、無効のマークが付けられ、従属オブジェクトが再作成されるまで、そのままの状態になります。これには、ドロップされる階層内の副表の上にあるスーパー表に依存しているパッケージが含まれます。
- 参照の有効範囲として、ドロップされた表を定義しているすべての参照列は、「有効範囲解除」されます。
- その表の別名定義は影響を受けません。別名は「未定義」状態となります。
- ドロップされた表に依存しているすべてのトリガーには、作動不能のマークが付けられます。

コマンド行を使用して表をドロップするには、以下のように入力します。

```
DROP TABLE <table_name>
```

次のステートメントは、DEPARTMENT という表をドロップするものです。

```
DROP TABLE DEPARTMENT
```

個々の表に副表がある場合、その表はドロップできません。ただし、次の例に示すとおり、表階層内の表はすべて、単一の DROP TABLE HIERARCHY ステートメントを使ってドロップできます。

```
DROP TABLE HIERARCHY person
```

DROP TABLE HIERARCHY ステートメントでは、ドロップする階層のルート表を指定しなければなりません。

表階層のドロップと特定表のドロップを比較した場合、次のような相違があります。

- DROP TABLE HIERARCHY は、個々の DROP 表ステートメントで活動化される削除トリガーを活動化しない。例えば、個々の副表をドロップすると、そのスーパー表に対する削除トリガーが活動化されます。
- DROP TABLE HIERARCHY は、ドロップされた表の個々の行についてログ項目を作成しない。代わりに、階層のドロップは単一のイベントとしてログ記録されます。

マテリアライズ照会またはステージング表のドロップ

マテリアライズ照会表またはステージング表は変更できませんが、ドロップすることはできます。この表を参照しているすべての索引、主キー、外部キー、およびチェック制約がドロップされます。この表を参照するすべてのビューおよびトリガーは、作動不能になります。ドロップされたオブジェクトまたは作動不能とマークされたオブジェクトに依存するすべてのパッケージは、無効になります。

コマンド行を使用してマテリアライズ照会表またはステージング表をドロップするには、以下のように入力します。

```
DROP TABLE <table_name>
```

次のステートメントは、マテリアライズ照会表 XT をドロップするものです。

```
DROP TABLE XT
```

マテリアライズ照会表は、 DROP TABLE ステートメントを使用して明示的にドロップすることもできますし、基礎表のいずれかがドロップされる場合は暗黙的にドロップすることもできます。

ステージング表は、 DROP TABLE ステートメントを使用して明示的にドロップすることもできますし、関連したマテリアライズ照会表がドロップされるときに暗黙的にドロップすることもできます。

表のシナリオおよび例

このセクションでは、表のシナリオおよび例を示します。

シナリオ: オプティミスティック・ロックおよび時間に基づく検出

以下の 3 つのシナリオでは、アプリケーションでオプティミスティック・ロックを使用可能化およびインプリメントする方法を示します。時間に基づく検出を使用する場合と使用しない場合、および暗黙的な隠し列を使用する場合と使用しない場合を考慮します。

シナリオ: アプリケーション・プログラムでのオプティミスティック・ロックの使用

このシナリオでは、アプリケーション・プログラムでオプティミスティック・ロックをインプリメントする方法を、6 つの異なるシナリオにわたって説明します。

オプティミスティック・ロックを使用可能化するよう設計されたアプリケーション内の以下のような一連のイベントを考えます。

```
SELECT QUANTITY, row change token FOR STOCK, RID_BIT(STOCK)
 INTO :h_quantity, :h_rct, :h_rid
 FROM STOCK WHERE PARTNUM = 3500
```

このシナリオでは、アプリケーション・ロジックが各行を読み取ります。『 286 ページの『アプリケーションにおけるオプティミスティック・ロックの使用可能化』』で説明されているように、このアプリケーションではオプティミスティック・ロックが使用可能化されているため、ホスト変数 :h_rid に保管された RID_BIT() 値、およびホスト変数 :h_rct に保管された行変更トークン値が選択リストに含まれます。

オプティミスティック・ロックを使用可能にすると、アプリケーションは、更新または削除のターゲット行をロックで保護しなくても、それらの行は変更されないとオプティミスティック (楽観的) に想定します。データベースの並行性を改善するために、アプリケーションは以下のいずれかの方法を使用して行ロックを除去します。

- 作業単位のコミット (この場合、行ロックが除去されます)
- WITH RELEASE 節を使用したカーソルのクローズ (この場合、行ロックが除去されます)
- より低い分離レベルを次のように使用する:
  - CURSOR STABILITY (CS)。この場合、次の行または結果表の末尾にカーソルがフェッチした後は、行がロックされません。
  - UNCOMMITED READ (UR)。この場合、すべての非コミット・データは新しい (非コミット) 行変更トークン値を持ちます。非コミット・データがロールバックされた場合、古いコミット済み行変更トークンは別の値になります。

    注: 通常は更新がロールバックされないと想定すると、UR を使用すれば最大の並行性が得られます。
- データベースから切断して、アプリケーション用のすべての DB2サーバー・リソースを解放する。 (この方式は .NET アプリケーションによってよく使用されます。)

次のように、アプリケーションはいくつかの行を処理して、いずれか 1 つをオプティミスティックな方法で更新することを決定します。

```
UPDATE STOCK SET QUANTITY = QUANTITY – 1
 WHERE row change token FOR STOCK = :h_rct AND
 RID_BIT(STOCK) = :h_rid
```

UPDATE ステートメントは、上記の SELECT ステートメントで識別される行を更新します。

次のように、探索型 UPDATE 述部が表への直接フェッチとして計画されています。

```
RID_BIT(STOCK) = :h_rid
```

直接フェッチは、DB2オプティマイザーにかかるコストが小さい、単純で非常に効率的なアクセス・プランです。 RID_BIT() 述部で行を検出できない場合、行はすでに削除済みで、行が見つからないために更新が失敗します。

RID_BIT() 述部で行が見つかる場合、行変更トークンが未変更であれば、述部の行変更トークン FOR STOCK = :h_rct によって行が検出されます。 SELECT 以後に行変更トークンが変更された場合、行が見つからないために探索型 UPDATE が失敗します。

299 ページの表 49 は、オプティミスティック・ロックを使用可能にした場合に考えられるシナリオの一覧です。

表 49. オプティミスティック・ロックを使用可能にした場合に考えられるシナリオ

シナリオ ID	アクション	結果
シナリオ 1	行変更タイム・スタンプ列が表に定義され、行を変更したアプリケーションは他に存在しない。	:h_rid によって識別される行に対する行変更トークン述部が成功し、更新が成功します。
シナリオ 2	ROW CHANGE TIMESTAMP が表に定義されている。選択の後、更新 (およびコミット) の前に、別のアプリケーションが行を更新し、行変更タイム・スタンプ列も更新された。	行変更トークン述部では、選択時に行のタイム・スタンプから生成されたトークンと、現在の行のタイム・スタンプ・トークン値との比較に失敗します。このため UPDATE ステートメントは行の検出に失敗します。
シナリオ 3	ROW CHANGE TIMESTAMP が表に定義されている。別のアプリケーションが行を更新したため、その行変更トークンが新しくなった。このアプリケーションは分離レベル UR で行を選択し、新しい非コミット行変更トークンを取得する。	このアプリケーションは UPDATE を実行しますが、別のアプリケーションが行ロックを解除するまでロック待機します。新しいトークンを持つ変更内容が別のアプリケーションによってコミットされた場合、行変更トークン述部は成功し、UPDATE も成功します。別のアプリケーションが古いトークンにロールバックした場合、行変更トークン述部は失敗し、UPDATE は行を検出できません。
シナリオ 4	行変更タイム・スタンプ列が表に定義されていない。選択の後、更新の前に、同じページ上の別の行が更新、削除、または挿入された。	ページ上の全行に関する行変更トークン値が変更されたため、行変更トークン述部はトークンの比較に失敗します。実際には行が変更されていなくても、UPDATE ステートメントは行の検出に失敗します。 もし行変更タイム・スタンプ列を追加していれば、この false negative シナリオでは UPDATE は失敗しなかったはずです。
シナリオ 5	行変更タイム・スタンプ列を持つように表が修正された。選択によって戻された行は、この修正後にまだ変更されていない。別のアプリケーションがこの行を更新し、その際、現在のタイム・スタンプを持つ行変更タイム・スタンプ列をその行に追加した。	行変更トークン述部は、以前に生成されたトークンと、行変更タイム・スタンプ列から作成されたトークン値との比較に失敗します。このため UPDATE ステートメントは行を検出できません。対象の行は実際に変更されているため、これは false negative シナリオではありません。
シナリオ 6	選択の後、更新の前に、表が再編成された。 :h_rid で識別される行 ID を使用しても、行を検出できないか、異なるトークンを持つ行であるため、更新は失敗します。このような false negative は、たとえ行変更タイム・スタンプ列が行に存在しても、防ぐことができません。	行そのものは再編成によって更新されませんが、述部の RID_BIT 部分は再編成後に元の行を識別できません。

シナリオ: 暗黙的な隠し列を使用したオプティミスティック・ロック

以下のシナリオでは、暗黙的な隠し列 (つまり IMPLICITLY HIDDEN 属性を使って定義された列) を使用してアプリケーション・プログラムでオプティミスティック・ロックをインプリメントする方法について説明します。

これらのシナリオでは、表 SALARY_INFO に 3 つの列が定義され、最初の列は暗黙的に隠された ROW CHANGE TIMESTAMP 列で、その値は常に生成されると想定します。

シナリオ 1:

次のステートメントでは、暗黙的な隠し列が列リストで明示的に参照され、その値が VALUES 節で提供されます。

```
INSERT INTO SALARY_INFO (UPDATE_TIME, LEVEL, SALARY)
  VALUES (DEFAULT, 2, 30000)
```

シナリオ 2:

次の INSERT ステートメントは暗黙的な列リストを使用します。暗黙的な列リストには暗黙的な隠し列が含まれないため、VALUES 節には他の 2 列の値だけが含まれます。

```
INSERT INTO SALARY_INFO
  VALUES (2, 30000)
```

この場合、デフォルト値を提供するために列 UPDATE_TIME を定義する必要があります。そのデフォルト値は、挿入される行に使用されます。

シナリオ 3:

次のステートメントでは、暗黙的な隠し列が選択リストで明示的に参照され、その値が結果セットに表示されます。

```
SELECT UPDATE_TIME, LEVEL, SALARY FROM SALARY_INFO
  WHERE LEVEL = 2

  UPDATE_TIME        LEVEL    SALARY
  -------------------------- ----------- -----------
  2006-11-28-10.43.27.560841      2    30000
```

シナリオ 4:

次のステートメントでは * 表記を使って列リストが暗黙的に生成されます。暗黙的な隠し列は結果セットに表示されません。

```
SELECT * FROM SALARY_INFO
  WHERE LEVEL = 2

  LEVEL    SALARY
  ----------- -----------
       2    30000
```

シナリオ 5:

次のステートメントでは * 表記を使って列リストが暗黙的に生成されます。さらに、ROW CHANGE TIMESTAMP FOR 式を使用することによって暗黙的な隠し列の値も表示されます。

```
SELECT ROW CHANGE TIMESTAMP FOR SALARY_INFO
  AS ROW_CHANGE_STAMP, SALARY_INFO.*
  FROM SALARY_INFO WHERE LEVEL = 2
```

結果表はシナリオ 3 と同様です (列 UPDATE_TIME が ROW_CHANGE_STAMP になります)。

シナリオ: 時間に基づく更新の検出

このシナリオでは、タイム・スタンプによる更新検出を使ってアプリケーション・プログラムでオプティミスティック・ロックをインプリメントする方法を、3 つの異なるシナリオにわたって説明します。

このシナリオでは、最近 30 日間に変更されたすべての行をアプリケーションが選択します。

```
SELECT * FROM TAB WHERE
  ROW CHANGE TIMESTAMP FOR TAB <=
  CURRENT TIMESTAMP AND
  ROW CHANGE TIMESTAMP FOR TAB >=
  CURRENT TIMESTAMP - 30 days;
```

シナリオ 1:

行変更タイム・スタンプ列が表に定義されていない。ステートメントは SQL20431N とともに失敗します。この SQL 式は、行変更タイム・スタンプ列が定義された表でのみサポートされます。

注: このシナリオは z/OSでは機能します。

シナリオ 2:

次のように、表の作成時に行変更タイム・スタンプ列が定義された。

```
CREATE TABLE TAB ( ..., RCT TIMESTAMP NOT NULL
                        GENERATED ALWAYS
                        FOR EACH ROW ON UPDATE AS
                        ROW CHANGE TIMESTAMP)
```

このステートメントによって、最近 30 日間に挿入または更新されたすべての行が戻されます。

シナリオ 3:

最近 30 日間のある時点で、次のような ALTER TABLE ステートメントを使って行変更タイム・スタンプ列が表に追加された。

```
ALTER TABLE TAB ADD COLUMN RCT TIMESTAMP NOT NULL
                                GENERATED ALWAYS
                                FOR EACH ROW ON UPDATE AS
                                ROW CHANGE TIMESTAMP
```

このステートメントによって表のすべての行が戻されます。 ALTER TABLE ステートメント以後に変更されていない行は、ALTER TABLE ステートメント自体のタイム・スタンプをデフォルト値として使用します。 ALTER TABLE ステートメント以後に変更された他のすべての行は、固有のタイム・スタンプを持ちます。

第 12 章 制約

どの業務でも、データが特定の制限または規則に従っていなければならない場合があります。例えば、従業員番号が固有でなければならない、などです。データベース・マネージャーは、このような規則を強制する手段として*制約* を提供します。

以下のタイプの制約が使用できます。

- NOT NULL 制約
- ユニーク (またはユニーク・キー) 制約
- 主キー制約
- 外部キー (または参照整合性) 制約
- (表) チェック制約
- インフォメーショナル制約

制約は表のみに関連付けられ、表の作成プロセスの一部として (CREATE TABLE ステートメントを使用して) 定義されるか、または表の作成後に (ALTER TABLE ステートメントを使用して) 表の定義に追加されます。ALTER TABLE ステートメントを使用して、制約を変更することができます。たいていの場合、既存の制約はいつでもドロップできます。この操作は、表の構造や、そこに格納されているデータには影響を与えません。

注: 表オブジェクトに関連付けられるのはユニーク制約とプライマリー制約のみで、これらはしばしば 1 つ以上のユニーク索引または主キー索引を使用することによって強制されます。

制約のタイプ

制約 は、最適化を目的として使用される規則です。

制約には次の 5 つのタイプがあります。

- *NOT NULL 制約*。これは、表の中の 1 つ以上の列に NULL 値が入力されないようにする規則です。
- *ユニーク制約* (*ユニーク・キー制約* とも呼ばれる)。これは、表の中の 1 つ以上の列での重複値を禁止する規則です。ユニーク制約では、ユニーク・キーと主キーがサポートされています。例えば、2 つの製造業者に同じ製造業者 ID が与えられないようにするために、製造業者表の製造業者 ID にユニーク制約を定義することができます。
- *主キー制約*。これは、ユニーク制約と同じプロパティーを持つ、列または列の組み合わせです。主キー制約と外部キー制約を使用して、表間のリレーションシップを定義できます。
- *外部キー制約* (*参照制約* または *参照整合性制約* とも呼ばれる)。これは、1 つ以上の表の中の 1 つ以上の列での値に関する論理規則です。例えば、表集合は企業の製造業者に関する情報を共有します。場合によっては、製造業者の名前が変わることもあります。表の製造業者の ID が、製造業者情報の製造業者 ID と一致

していなければならないことを示す参照制約を定義できます。この制約により、製造業者情報が失われてしまう結果になりかねない挿入、更新、削除操作が抑止されます。

- *(表) チェック制約* (単に*チェック制約* と呼ぶこともある)。これは、特定の表に追加されるデータに制約を設定します。例えば、表チェック制約では、個人情報の入った表で給与データが追加または更新される場合に、常に従業員の給与レベルが $20,000 より低くならないようにできます。

インフォメーショナル制約。これは制約の特定のタイプの属性ですが、データベース・マネージャーでは実施されません。

NOT NULL 制約

NOT NULL 制約は、NULL 値が列に入力されないようにします。

NULL 値は、不明の状態を表すためにデータベース内で使用されます。デフォルトでは、データベース・マネージャーに提供されるすべての組み込みデータ・タイプが NULL 値の存在をサポートします。ただし、ビジネス・ルールによっては、値を常に提供しなければならないことを指示するものがあります (例えば、すべての従業員が緊急時連絡先情報を通知する必要があるなど)。表の指定された列に NULL 値が割り当てられないことを保証するために、NOT NULL 制約が使用されます。NOT NULL 制約が特定の列について定義されていると、その列に NULL 値を入れようとする挿入または更新操作は失敗します。

制約は特定の表にのみ適用されるため、通常、制約は表作成プロセス中に表の属性と共に定義されます。次の CREATE TABLE ステートメントは、NOT NULL 制約が特定の列に定義される方法を示しています。

```
CREATE TABLE EMPLOYEES    (. . .
                           EMERGENCY_PHONE    CHAR(14)  NOT NULL,
                           . . .
                           );
```

ユニーク制約

ユニーク制約 は、列セットにある値が固有となり、表のすべての行が非 NULL となるようにします。ユニーク制約で指定される列は、 NOT NULL と定義されていなければなりません。ユニーク制約の列に変更が加えられるときに、データベース・マネージャーはユニーク索引を使用してキーを強制的にユニークにします。

ユニーク制約は、UNIQUE 節を使用して、CREATE TABLE または ALTER TABLE ステートメントで定義できます。例えば、部署表での典型的なユニーク制約では、部署番号が固有かつ非 NULL でなければなりません。

305 ページの図 21 は、表にユニーク制約が存在する場合、表に重複レコードが追加されないことを示しています。

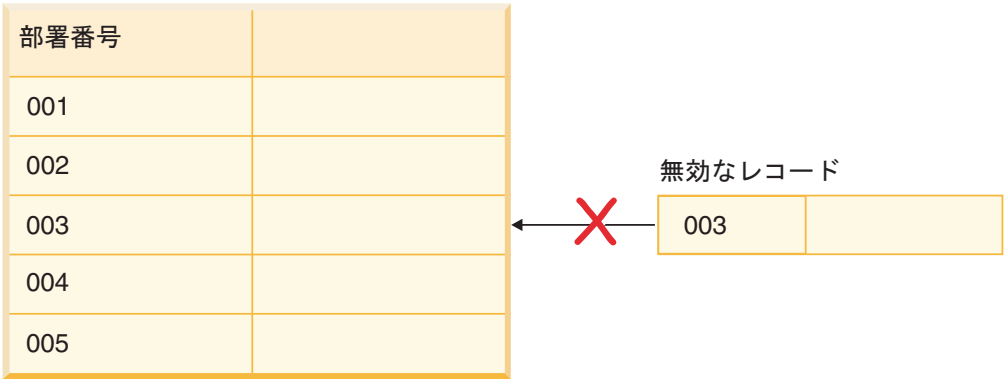


図 21. ユニーク制約は、データが重複しないようにするためのものです。

データベース・マネージャーは、挿入および更新操作中に制約を強制し、データ整合性を保証します。

1 つの表で、任意の数のユニーク制約を定義できます。ただし、複数のユニーク制約を主キーと定義することはできません。 1 つの表で、同じ列セット上に複数のユニーク制約を持つことはできません。

参照制約の外部キーによって参照されるユニーク制約を、*親キー* といいます。

- CREATE TABLE ステートメントでユニーク制約が定義されると、データベース・マネージャーによってユニーク索引が自動的に作成され、 1 次索引、またはユニークかつシステムで必須の索引と指定されます。
- ユニーク制約が ALTER TABLE ステートメントで定義されており、索引が同じ列にある場合、その索引はユニークかつシステムで必須と指定されます。そのような索引が存在しない場合、データベース・マネージャーによってユニーク索引が自動的に作成されて、1 次索引、またはユニークかつシステムで必須の索引と指定されます。

注: ユニーク制約を定義することとユニーク索引を作成することは別です。どちらの操作も強制的にユニークにしますが、ユニーク索引では NULL 可能列を使用できるので、一般的に親キーとして使用することができません。

主キー制約

主キー制約と外部キー制約を使用して、表間のリレーションシップを定義できます。

主キーとは、ユニーク制約と同じプロパティーを持つ、列または列の組み合わせです。主キーは表の行を識別するために使用するため、固有であり、かつ NOT NULL 属性を持っていなければなりません。 1 つの表は複数の主キーを持つことはできませんが、複数のユニーク・キーを持つことはできます。主キーはオプションであり、表の作成時または変更時に定義できます。これらには、データのエクスポート時または再編成時にデータを配列するという利点もあります。

(表) チェック制約

チェック制約 (*表チェック制約* とも呼ばれる) は、表の各行にある 1 つ以上の列で許可される値を指定するためのデータベース規則です。チェック制約の指定は、制限付きフォームの検索条件を使って行われます。

外部キー (参照) 制約

外部キー制約 (*参照制約* または*参照整合性制約* とも呼ばれる) により、表間および表内での必須リレーションシップを定義することができます。

例えば、典型的な外部キー制約は、従業員表の全従業員が、部署表に定義されているとおりに既存の部署のメンバーでなければならない、というものです。

参照整合性 とは、すべての外部キーのすべての値が有効であるデータベースの状態です。*外部キー* は、親表の行の 1 つ以上の主キーまたはユニーク・キー値と値が一致している必要のある、表内の列または列セットです。*参照制約* は、以下の条件のいずれかが当てはまる場合にのみ、外部キーの値が有効になる規則です。

- それらが親キーの値となっている。
- ある外部キーのコンポーネントは NULL である。

このリレーションシップを確立するには、従業員表の部署番号を外部キーとして定義し、部署表の部署番号を主キーとして定義できます。

307 ページの図 22 は、表間に外部キー制約が存在する場合、表に無効キーのレコードが追加されないようにする方法を示しています。

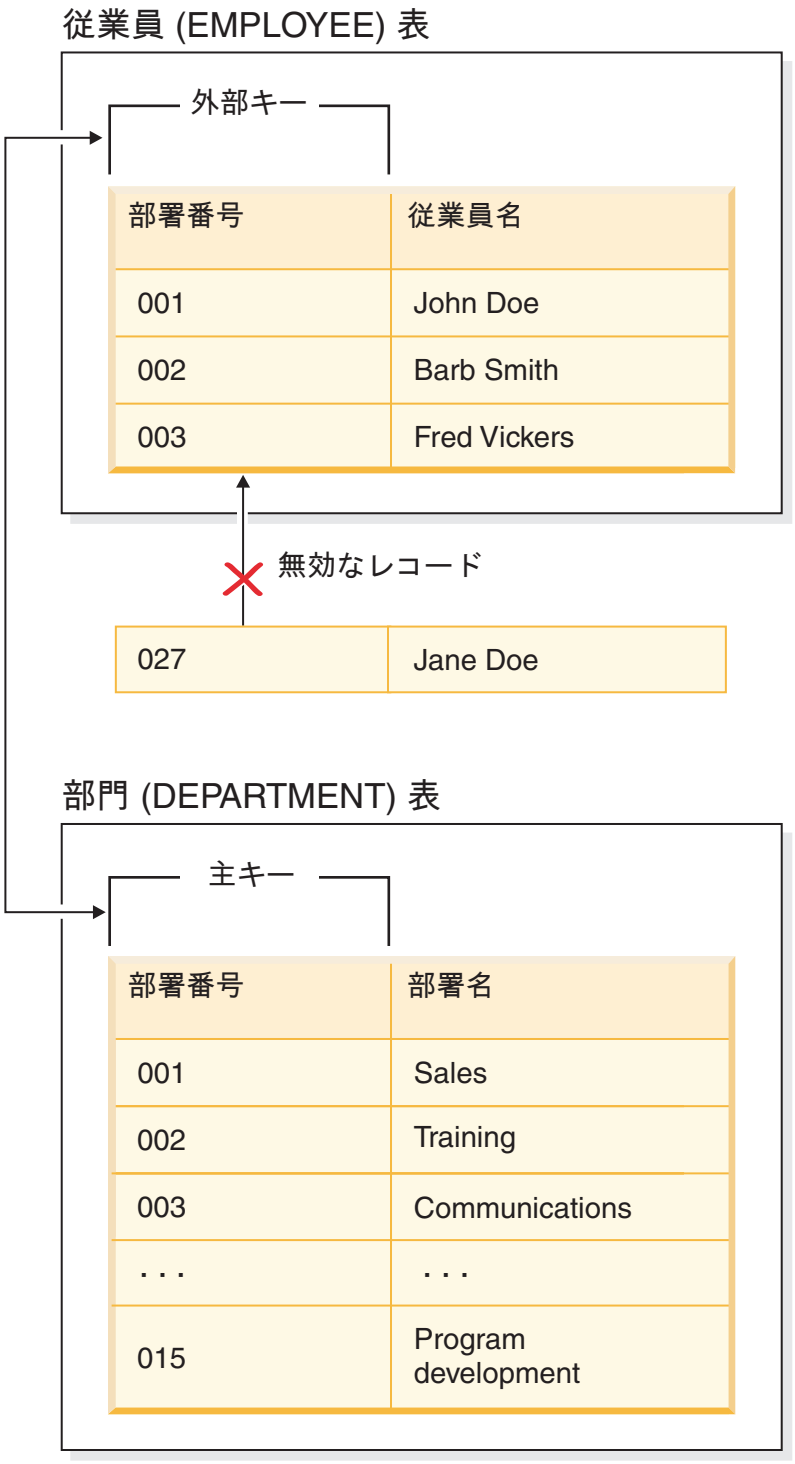


図 22. 外部キー制約と主キー制約

親キーの入った表を参照制約の *親表* といい、外部キーの入った表を表の *従属* といいます。

参照制約は、CREATE TABLE ステートメントまたは ALTER TABLE ステートメントで定義できます。参照制約は、INSERT、UPDATE、DELETE、ALTER

TABLE、MERGE、ADD CONSTRAINT、および SET INTEGRITY ステートメントの実行中に、データベース・マネージャーによって実施されます。

参照整合性規則には、以下の用語が関係します。

表 50. 参照整合性の用語

概念	用語
親キー	親キーとは、参照制約の主キーまたはユニーク・キーのことです。
親行	1 つ以上の従属行を持つ行。
親表	参照制約の親キーの入った表。 1 つの表を、任意の数の参照制約における親にすることができます。参照制約の中の親である表が、参照制約の中の従属になることもできます。
従属表	定義において 1 つ以上の参照制約の入った表。表を、任意の数の参照制約において従属表にすることができます。参照制約の中の従属である表が、参照制約の中の親になることもできます。
下層表	ある表が表 T の下層であるとは、それが T の従属であるか、または T の従属の下層であるということです。
従属行	1 つ以上の親行を持つ行。
下層行	ある行が行 r の下層であるとは、それが r の従属であるか、または r の従属の下層であるということです。
参照循環	セット内の各表がそれ自身の下層であるような、参照制約のセット。
自己参照表	同じ参照制約内の親および従属である表。この制約を、*自己参照制約* といいます。
自己参照行	それ自身の親である行。

参照制約の目的は、確実に表のリレーションシップが維持され、データ入力規則が順守されるようにすることです。つまり、参照制約が実施されていれば、子表の外部キー列に非 NULL 値が存在し、それに一致する値が対応する親表の親キーに存在する場合に、子表の各行にそれぞれ対応する行が親表に存在することが、データベース・マネージャーによって保証されます。

参照制約を侵すような形で SQL 操作によるデータの変更が試みられると、外部キー (または参照) 制約違反が発生する可能性があります。データベース・マネージャーは、各参照制約に関連した規則のセットを実施することによってこの種の状況を処理します。この規則のセットは次のもので構成されます。

- 挿入規則
- 更新規則
- 削除規則

参照整合性を侵すような形で SQL 操作によるデータの変更が試みられると、参照制約違反が発生する可能性があります。例:

- 挿入操作によって子表にデータの行を追加しようとしたときに、子表の外部キー列の値が、対応する親表の親キーの値と一致しない場合。
- 更新操作によって子表の外部キー列の値を別の値に変更しようとしたときに、その変更後の値が、対応する親表の親キーの値と一致しない場合。
- 更新操作によって親表の親キーの値を別の値に変更しようとしたときに、その変更後の値が、子表の外部キー列の値と一致しない場合。

- 削除操作によって親表からレコードを除去しようとしたときに、子表の外部キー列にそれと一致する値がある場合。

データベース・マネージャーは、各参照制約に関連した規則のセットを実施することによってこの種の状況を処理します。この規則のセットは次のもので構成されます。

- 挿入規則
- 更新規則
- 削除規則

挿入規則

参照制約の挿入規則とは、外部キーの非 NULL の挿入値が、親表の親キーの何らかの値に一致しなければならないというものです。複合外部キーの値は、値のいずれかの部分が NULL であれば、NULL になります。これは、外部キーが指定された場合は暗黙の規則になります。

更新規則

参照制約の更新規則は、参照制約を定義するときに指定されます。選択項目には NO ACTION と RESTRICT があります。更新規則は、親の行または従属表の行を更新するときに適用されます。

親行の場合、親キーの列内の値を更新するときに以下の規則が適用されます。

- 従属表の行がキーの元の値と一致し、しかも更新規則が RESTRICT である場合、その更新は拒否されます。
- 更新ステートメントの完了時に (AFTER トリガーを除く) 従属表の行に対応する親キーがなく、しかも更新規則が NO ACTION である場合、その更新は拒否されます。

更新規則が RESTRICT で、 1 つまたは複数の従属行が存在する場合には、親のユニーク・キーの値は変更できません。ただし、NO ACTION の更新規則では、更新ステートメントの完了時にすべての子が親キーを持つ場合、親のユニーク・キーを更新することができます。 NULL 以外の外部キーの更新値は、関連する親表の主キーの値に等しくなければなりません。

また参照制約の更新規則として NO ACTION または RESTRICT を使用すると、制約がいつ適用されるかが決まります。 RESTRICT の更新規則は、 CASCADE や SET NULL などの変更規則を伴う参照制約を含む他のすべての制約の前に適用されます。 NO ACTION の更新規則は、他の参照制約の後で適用されます。戻される SQLSTATE は、更新規則が RESTRICT か NO ACTION かによって異なります。

従属行の場合、外部キーを指定した際の暗黙的な更新規則は NO ACTION です。NO ACTION は、更新ステートメントの完了時に、外部キーの NULL 以外の更新値が親表の親キーの何らかの値に一致していなければならないことを意味します。

複合外部キーの値は、値のいずれかの部分が NULL であれば、NULL になります。

削除規則

参照制約の削除規則は、参照制約を定義するときに指定されます。選択項目としては、NO ACTION、RESTRICT、CASCADE、または SET NULL があります。 SET NULL を指定できるのは、外部キーのいずれかの列で NULL 値が可能な場合だけです。

指定する表または指定するビューの基本表が親である場合、削除のために選択する行は RESTRICT の削除規則との関係において従属であってはならず、 DELETE は RESTRICT の削除規則との関係において従属である下層行にカスケードしてはなりません。

削除操作が RESTRICT の削除規則によって禁止されていなければ、選択された行は削除されます。選択された行の従属行もすべて影響を受けます。

- 削除規則が SET NULL の関係において、すべての従属行の外部キーの NULL 可能列は、 NULL 値に設定されます。
- 削除規則が CASCADE のリレーションシップにおいて、すべての従属行も削除され、上記の規則はこれらの行にも適用されます。

他の参照制約が実施された後で、非 NULL の外部キーが既存の親行を指すようにするために、 NO ACTION の削除規則が検査されます。

参照制約の削除規則は、親表の*行* が削除されるときにのみ適用されます。より厳密に言うと、親表の*行* が削除または伝搬された削除操作 (以下で定義する)の対象であり、その行に参照制約の従属表内の従属がある場合にのみ、その規則が適用されます。例えば、P が親表、D が従属表、そして p が削除または伝搬削除操作の対象である親行を指すとします。削除規則は以下のように機能します。

- RESTRICT または NO ACTION では、エラーが発生し、行は削除されません。
- CASCADE では、削除操作が表 D の従属に伝搬されます。
- SET NULL では、表 D 内の p の各従属の外部キーの各 NULL 可能列が NULL に設定されます。

P での削除操作に関係する可能性のあるすべての表を、P への*連結削除* といいます。したがって、表が P の従属である場合、または P から削除操作のカスケードが行われる表の従属である場合、表は表 P への連結削除になります。

連結削除のリレーションシップには以下の制約事項があります。

- 複数の表の参照サイクル中で表が自身への連結削除である場合、そのサイクルに RESTRICT または SET NULL の削除規則が含まれていてはなりません。
- 表は、CASCADE リレーションシップ (自己参照、または別の表を参照) 内の従属表であってはならず、RESTRICT または SET NULL の削除規則に対する自己参照リレーションシップを持っていてもなりません。
- 表が、オーバーラップする外部キーを持つような複数のリレーションシップに基づく別の表への連結削除である場合、それらのリレーションシップの削除規則は同じものでなければならず、いずれも SET NULL であってはなりません。
- 表と別の表の間に複数のリレーションシップに基づく連結削除が設定されており、それらのリレーションシップのうちのいずれかに SET NULL の削除規則が

指定されている場合、そのリレーションシップの外部キー定義に分散キーまたは MDC キー列が含まれていてはならず、データ・パーティション・キー列または RCT キー列を追加してもなりません。

- 2 つの表が CASCADE リレーションシップによる同じ表への連結削除である場合、連結削除のパスが削除規則 RESTRICT または SET NULL で終わっていれば、この 2 つの表は相互に対して連結削除とはなりません。

インフォメーショナル制約

インフォメーショナル制約 は、データへのアクセスを改善するために SQL コンパイラーが使用できる制約の属性です。インフォメーショナル制約はデータベース・マネージャーによって適用されず、データの追加検査を行うために使用されることもありません。この制約は、照会パフォーマンスを改善するために使用されます。

インフォメーショナル制約は、CREATE TABLE または ALTER TABLE ステートメントを使用して定義されます。まず、参照整合性またはチェック制約を追加し、その後、それらに制約属性を関連付けて、データベース・マネージャーがその制約を適用するかどうか、またその制約が照会の最適化のために使用されるかどうかを指定します。

制約の設計

制約を設計および作成する際、さまざまな種類の制約を識別しやすくする命名規則を使用するのが適切です。エラーが発生した場合の診断を考えると、これは特に重要です。

以下のタイプの制約を設計できます。

- NOT NULL 制約
- ユニーク制約
- 主キー制約
- (表) チェック制約
- 外部キー (参照) 制約
- インフォメーショナル制約

ユニーク制約の設計

ユニーク制約 は、指定されたキー内のそれぞれの値が固有のものになるようにします。 1 つの表は、1 つのユニーク制約を主キーとして定義して、任意の数のユニーク制約を持つことができます。

制約事項

- 副表にユニーク制約を定義することはできません。
- 1 つの表当たり 1 つの主キーだけが可能です。

CREATE TABLE または ALTER TABLE ステートメントの UNIQUE 節を使用してユニーク制約を定義します。ユニーク・キーは複数の列で構成できます。 1 つの表上で、複数のユニーク制約が許されます。

いったん確立されると、INSERT または UPDATE ステートメントが表内のデータを修正するときに、データベース・マネージャーによって、そのユニーク制約が自動的に施行されます。ユニーク制約は、ユニーク索引を通して施行されます。

ユニーク制約が ALTER TABLE ステートメントに定義され、そのユニーク・キーの同じ列セットに索引が存在する場合、その索引はユニーク索引となり、制約によって使用されます。

任意のユニーク制約を 1 つ選んで、それを主キーとして使用することができます。主キーは、(他のユニーク制約と一緒に) 参照制約の中の親キーとして使用することができます。主キーを定義するには、 CREATE TABLE または ALTER TABLE ステートメントで PRIMARY KEY 節を使います。主キーには、複数の列を含めることができます。

1 次索引は、主キーの値が固有のものとなるように強制します。主キーを指定して表を作成すると、データベース・マネージャーはそのキーに対する 1 次索引を作成します。

ユニーク制約として使用される索引に対するパフォーマンス上のヒントには、以下のものがあります。

索引のある空の表の初期ロードを実行するときは、 IMPORT よりも LOAD のほうがパフォーマンスはよくなります。これは、LOAD の INSERT または REPLACE のいずれのモードを使用していても同じです。索引のある既存の表に、大量のデータを追加するときには (IMPORT INSERT または LOAD INSERT を使用)、IMPORT よりも LOAD のほうがパフォーマンスは多少よくなります。IMPORT コマンドを使用して最初の大量のデータのロードを行う場合は、データがインポートまたはロードされた後でユニーク・キーを作成してください。こうすれば、表のロード時に索引保守のためのオーバーヘッドを避けることができます。さらに、索引が使用する記憶域を最小限にすることができます。REPLACE モードでロード・ユーティリティーを使用している場合は、データをロードする前にユニーク・キーを作成してください。この場合、ロード中に索引を作成するほうが、ロードの後で CREATE INDEX ステートメントを使用するより効率的です。

主キー制約の設計

それぞれの表は、1 つの主キーを持つことができます。主キーとは、ユニーク制約と同じプロパティーを持つ、列または列の組み合わせです。主キー制約と外部キー制約を使用して、表間のリレーションシップを定義できます。

主キーは表の行を識別するために使用するため、固有でなければならず、追加または削除は少なくなければなりません。 1 つの表は複数の主キーを持つことはできませんが、複数のユニーク・キーを持つことはできます。主キーはオプションであり、表の作成時または変更時に PRIMARY KEY 節を使ってこれを定義できます。これらには、データのエクスポート時または再編成時にデータを配列するという利点もあります。

主キー制約はユニーク制約と同じように設計されます (『 311 ページの『ユニーク制約の設計』』を参照)。唯一の違いとして、1 つの表に多数のユニーク制約を設定できますが、主キー制約は 1 つだけが可能です。

注: 複合主キーに基づいて主キー制約を設定できます。

チェック制約の設計

チェック制約を作成する際には、(1) すべての行がチェック制約を満たす、(2) いくつか、またはすべての行がチェック制約を満たさない、のいずれかの状況が発生します。

すべての行がチェック制約を満たす

すべての行がチェック制約に適合する場合には、チェック制約が正常に作成されます。その後、このビジネス・ルールの制約に適合しないデータを使って挿入または更新しようとすると、その試行は拒否されます。

いくつか、またはすべての行がチェック制約を満たさない

チェック制約に適合しない行が存在する場合には、チェック制約が作成されません (つまり ALTER TABLE ステートメントが失敗します)。新しい制約を EMPLOYEE 表に追加する ALTER TABLE ステートメントが以下に示されています。チェック制約の名前は CHECK_JOB です。 INSERT または UPDATE ステートメントが失敗した場合、どの制約に違反したかを通知する際に、データベース・マネージャーはこの名前を使用します。表のチェック制約を定義するために CHECK 節が使われています。

```
ALTER TABLE EMPLOYEE
  ADD CONSTRAINT check_job
  CHECK (JOB IN ('Engineer', 'Sales', 'Manager'));
```

表がすでに定義されていたため、ALTER TABLE ステートメントが使用されました。定義される制約に違反する値が EMPLOYEE 表に存在する場合、ALTER STATEMENT は正常に完了しません。

チェック制約や他の種類の制約を使ってビジネス・ルールをインプリメントしていくうちに、それらの制約を変更する必要が生じるかもしれません。組織のビジネス・ルールが変更された場合には、特にその可能性があります。チェック制約を変更する必要がある場合には、必ず制約をドロップして、新しい制約を再作成しなければなりません。チェック制約は任意の時点でドロップできます。ドロップ操作は、表または表内のデータに影響を与えません。チェック制約をドロップする際には、制約によって実行されていたデータ検証が無効になることに注意してください。

チェック制約と BEFORE トリガーの比較

データの整合性を保持するためにトリガーを使用するか、それともチェック制約を使用するかを検討するときは、チェック制約の間の違いを考慮する必要があります。

リレーショナル・データベース内のデータは複数のユーザーがアクセスし、データを変更するので、その整合性を保守する必要があります。データを共有する場合は常に、データベース内の値の正確性を保証する必要があります。

チェック制約

(表) チェック制約は、特定の表に追加されるデータに制約を設定します。表チェック制約は、表の列で使用できる値について、データ・タイプによる制限よりもさらに詳しい制限を定義するために使用します。表チェック制約

は、範囲のチェック、または同じ表中の同じ行にある他の値との関係に基づくチェックという形式で行うことができます。

データを使用するすべてのアプリケーションに対して規則を適用するには、表チェック制約を使用して表中で使用できるデータを制限してください。表チェック制約により、利用の幅が広がるとともに保守しやすくなります。

チェック制約の実施はデータ整合性を保守するために重要ですが、大量のデータが変更されると必ず一定量のオーバーヘッドが発生し、パフォーマンスに影響が出る場合があります。

BEFORE トリガー

更新または挿入の前に実行されるトリガーを使用すると、更新中または挿入中の値を、データベースが実際に修正される*前に* 修正することができます。これは、アプリケーションからの入力 (ユーザーから見たデータ) を希望の内部データベース形式に変換するために使用できます。また、ユーザー定義の関数により他の非データベース操作を活動化する際にも、BEFORE トリガーを使用することができます。

修正に加えて、BEFORE トリガーの一般的な使用法として SIGNAL 節を使用したデータ検査があります。

BEFORE トリガーとチェック制約の間には、データ検査に使用される場合、次の 2 つの違いがあります。

1. BEFORE トリガーはチェック制約とは異なり、同じ表中の同じ行にある他の値へのアクセスは制限されません。
2. LOAD 操作後の表に対する SET INTEGRITY 操作中は、トリガー (BEFORE トリガーを含む) は実行されません。しかし、チェック制約は検査されます。

外部キー (参照) 制約の設計

参照整合性 は、表定義と列定義に外部キー (または参照) 制約を追加すると課され、すべての外部キー列に索引が作成されます。索引および外部キー制約が定義されると、表および列内のデータへの変更は、定義済み制約に対してチェックされます。要求アクションの完了は、制約検査の結果に依存します。

参照制約は、 CREATE TABLE または ALTER TABLE ステートメントの FOREIGN KEY 節および REFERENCES 節を使用して確立されます。型付き表に対する参照制約、または型付き表である親表に対する参照制約からの影響があります。

外部キーの指定によって、1 つの表の行内または 2 つの表の行間の値について、制約が施行されます。データベース・マネージャーは、表定義で指定された制約を検査し、それに応じてリレーションシップを維持します。その目標は、1 つのデータベース・オブジェクトが別のオブジェクトを参照するときに、パフォーマンスを低下させることなく、いつでも整合性が保たれているようにすることです。

例えば、主キーと外部キーは、それぞれ部署番号の列を持ちます。 EMPLOYEE 表の場合、列名は WORKDEPT であり、DEPARTMENT 表の場合、列名は DEPTNO です。この 2 つの表の間のリレーションシップは、次の制約によって定義されています。

- EMPLOYEE 表の各従業員の部署番号は 1 つだけであり、その番号は DEPARTMENT 表の中にも存在しています。
- EMPLOYEE 表の各行は、DEPARTMENT 表の 1 つの行だけと関連があります。表同士の間には、一意のリレーションシップがあります。
- EMPLOYEE 表の行のうち WORKDEPT の値が NULL 値でないものは、それぞれ DEPARTMENT 表の DEPTNO 列の 1 つの行と関連しています。
- DEPARTMENT 表は親表であり、EMPLOYEE 表は従属表です。

親表である DEPARTMENT を定義するステートメントは、次のとおりです。

```
CREATE TABLE DEPARTMENT
       (DEPTNO    CHAR(3)     NOT NULL,
        DEPTNAME  VARCHAR(29) NOT NULL,
        MGRNO     CHAR(6),
        ADMRDEPT  CHAR(3)     NOT NULL,
        LOCATION  CHAR(16),
           PRIMARY KEY (DEPTNO))
    IN RESOURCE
```

従属表である EMPLOYEE を定義するステートメントは、次のとおりです。

```
CREATE TABLE EMPLOYEE
       (EMPNO     CHAR(6)     NOT NULL PRIMARY KEY,
        FIRSTNME  VARCHAR(12) NOT NULL,
        LASTNAME  VARCHAR(15) NOT NULL,
        WORKDEPT  CHAR(3),
        PHONENO   CHAR(4),
        PHOTO     BLOB(10m)   NOT NULL,
           FOREIGN KEY DEPT (WORKDEPT)
           REFERENCES DEPARTMENT ON DELETE NO ACTION)
    IN RESOURCE
```

DEPTNO 列を DEPARTMENT 表の主キーとして指定し、 WORKDEPT を EMPLOYEE 表の外部キーとして指定すると、 WORKDEPT 値についての参照制約を定義することになります。この制約は、2 つの表の間での値の参照整合性を施行するものとなります。この場合、EMPLOYEE 表に追加される従業員の部署番号は、 DEPARTMENT 表の中にあるものでなければなりません。

EMPLOYEE 表の参照制約の削除規則は、NO ACTION です。つまり、DEPARTMENT 表から部署を削除しようとしても、その部署に従業員がいれば削除はできません。

この例では CREATE TABLE ステートメントを使って参照制約を追加していますが、 ALTER TABLE ステートメントを使うこともできます。

別の例: 前の例の中で使用されたのと同じ表定義が使用されます。また、DEPARTMENT 表は、EMPLOYEE 表より前に作成されます。各部署にはマネージャーがおり、そのマネージャーは EMPLOYEE 表にリストされています。DEPARTMENT 表の MGRNO 番号は、実際には EMPLOYEE 表の外部キーになっています。この参照サイクルのために、この制約にはわずかに問題があります。外部キーは後で追加することができます。また、CREATE SCHEMA ステートメントを使用して、EMPLOYEE 表と DEPARTMENT 表の両方を同時に作成することもできます。

318 ページの『参照制約内の外部キー』 も参照してください。

トリガーと参照制約の相互作用の例

更新操作を行うと、トリガーと参照制約およびチェック制約の相互作用が発生することがあります。

図 23 とその後の説明は、データベースのデータを更新するステートメントに対して行われる典型的な処理を示しています。

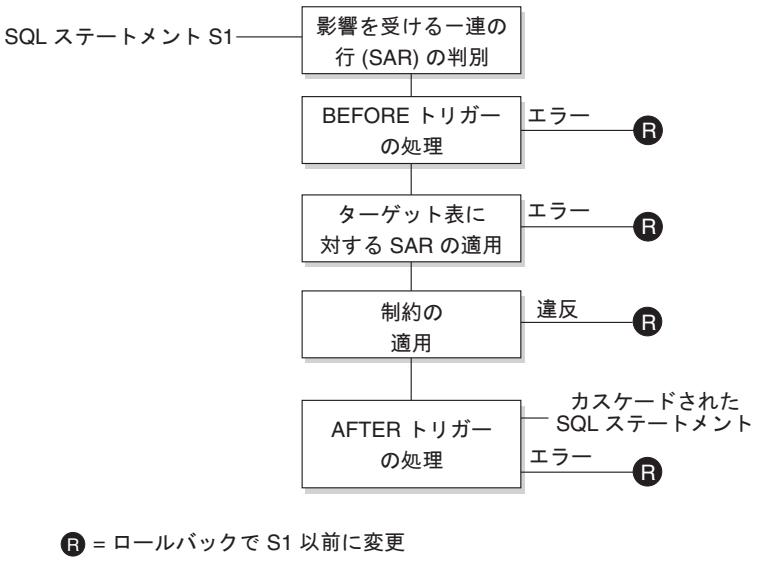


図 23. 関連するトリガーと制約を伴うステートメントの処理

図 23 は、表を更新するステートメントの一般的な処理の順序を示しています。ここでは、BEFORE トリガー、参照制約、チェック制約、および AFTER トリガーがカスケードしている表を想定しています。図 23 に示されているボックスやその他の項目について、以下に説明します。

- ステートメント S_1

  これは、プロセスを開始する DELETE、INSERT、または UPDATE ステートメントです。ステートメント S_1 は、この説明においてサブジェクト表 と呼ばれる表 (または表に対する更新可能なビュー) を指定しています。

- 影響を受ける一連の行の判別

  このステップは、CASCADE および SET NULL の参照制約の削除規則と、AFTER トリガーからのカスケード・ステートメントに対して繰り返されるプロセスの開始点です。

  このステップの目的は、そのステートメントで*影響を受ける一連の行* を判別することです。含まれる行の集合は、ステートメントに基づいて、以下のようになります。

  – DELETE の場合、ステートメントの検索条件を満たしているすべての行 (位置指定 DELETE の場合は現在行)
  – INSERT の場合、VALUES 節または全選択によって指定される行
  – UPDATE の場合、検索条件を満たしているすべての行 (位置指定 UPDATE の場合は現在行)

影響を受ける一連の行が空の場合、BEFORE トリガー、サブジェクト表に適用される変更、またはステートメントの処理に対する制約はありません。

- BEFORE トリガーの処理

  BEFORE トリガーの処理はすべて作成の昇順で行われます。各 BEFORE トリガーは、影響を受ける一連の行内の各行ごとに 1 回ずつトリガー・アクションを処理します。

  トリガー・アクションの処理の過程でエラーが生じることがあり、そのような場合には元のステートメント S_1 の結果としての変更内容 (これまでの) がすべてロールバックされます。

  BEFORE トリガーがない場合、または影響を受ける一連の行が空の場合、このステップはスキップされます。

- サブジェクト表への影響を受ける一連の行の適用

  データベース内のサブジェクト表への実際の削除、挿入、または更新は影響を受ける一連の行を使用して適用されます。

  影響を受ける一連の行の適用時にエラーが生じることがあり (ユニーク索引のあるロケーションに重複するキーを持つ行を挿入しようとした場合など)、そのような場合は元のステートメント S_1 の結果としての変更内容 (これまでの) がすべてロールバックされます。

- 制約の適用

  影響を受ける一連の行が空でない場合には、サブジェクト表に関連した制約が適用されます。この制約には、ユニーク制約、ユニーク索引、参照制約、チェック制約、ビューに対する WITH CHECK OPTION に関連した検査などがあります。カスケード削除規則または NULL 設定のある参照制約では、追加のトリガーが活動化されることがあります。

  何らかの制約または WITH CHECK OPTION に違反するとエラーが発生し、S_1 の結果として行われた変更 (その時点までの) はロールバックされます。

- AFTER トリガーの処理

  S_1 によって活動化された AFTER トリガーは、すべて作成の昇順に処理されます。

  FOR EACH STATEMENT トリガーでは、影響を受ける一連の行が空の場合にも、1 回だけトリガー・アクションが処理されます。 FOR EACH ROW トリガーでは、影響を受ける一連の行内の各行ごとに 1 回ずつトリガー・アクションが処理されます。

  トリガー・アクションの処理の過程でエラーが生じることがあり、そのような場合は元の S_1 の結果としての変更内容 (これまでの) がすべてロールバックされます。

トリガーのトリガー・アクションには、トリガーによって実行される DELETE、INSERT、または UPDATE などのステートメントが入っている場合があります。この説明では、そのような各ステートメントは、*カスケードしたステートメント* と見なされます。

カスケードしたステートメントは、AFTER トリガーのトリガー・アクションの一部として処理される DELETE、INSERT、または UPDATE ステートメントです。そのステートメントによって、カスケード・レベルのトリガー処理が開始されます。これは、新しい S_1 としてトリガー・ステートメントを割り当てて、ここで説明した手順をすべて再帰的に実行することと見なすことができます。

各 S_1 ごとに起動されるすべての AFTER トリガーによって実行されるすべてのステートメントの処理が完了すると、元の S_1 の処理が完了します。

- R = 変更を S_1 の前までロールバックする操作

  制約違反も含めて、処理中にエラーが発生すると、元のステートメント S_1 の結果として直接または間接になされたすべての変更がロールバックされます。その場合、データベースは、元のステートメント S_1 の実行直前と同じ状態に戻ります。

参照制約内の外部キー

外部キーは、同じ表または別の表内の主キーまたはユニーク・キーを参照します。外部キーを割り当てると、指定した参照制約にしたがって参照整合性が維持されます。

外部キーを定義するには、 CREATE TABLE または ALTER TABLE ステートメントで FOREIGN KEY 節を使います。外部キーにより、その表は別の表 (親表という) に従属することになります。任意の表の外部キーを構成する列または列セットの値は、その親表のユニーク・キーまたは主キーの値と一致していなければなりません。

外部キーの中の列の数は、親表の対応する基本制約またはユニーク制約 (親キーと呼ばれる) の中の列の数と同じでなければなりません。さらに、キー列定義の対応する部分は、それぞれ同じデータ・タイプ、同じ長さでなければなりません。外部キーには、*制約名* を割り当てることができます。名前は、割り当てなくても自動的に割り当てられます。使いやすさのためには、自分で*制約名* を割り当て、システムが生成した名前は使用しないようにすることをお勧めします。

複合外部キーの値は、外部キーの各列の値が親キーの対応する列の値と**等しければ**、親キーの値と一致します。 NULL 値が含まれる外部キーは、親キーが定義上 NULL 値を持つことができないため、親キーの値と一致することはありません。しかし、外部キーの NULL 値は、NULL 値ではないどの部分の値とも無関係に常に有効です。

外部キー定義に適用される規則は、次のとおりです。

- 1 つの表に複数の外部キーが可能です。
- 外部キーのいずれかの部分が NULL 可能なら、その外部キーは NULL 可能です。
- 外部キーのいずれかの部分が NULL ならば、その外部キーの値は NULL です。

外部キーを処理するときには、以下の操作を行えます。

- ゼロ個以上の外部キーを持つ表を作成すること。
- 表の作成時または変更時に外部キーを定義すること。
- 表の変更時に外部キーをドロップすること。

ユーティリティー操作に対する表制約の影響

ロードされる表に参照整合性制約がある場合、ロード・ユーティリティーはその表を SET INTEGRITY ペンディング状態にして、ロードされた行の参照整合性を検証するためにその表に対して SET INTEGRITY ステートメントを実行する必要があることを通知します。ロード・ユーティリティーの完了後に、SET INTEGRITY ステートメントを発行し、ロードされた行に対して参照整合性検査を実行して、表を SET INTEGRITY ペンディング状態から変更する必要があります。

例えば、DEPARTMENT 表と EMPLOYEE 表だけが SET INTEGRITY ペンディング状態になっている場合には、以下のステートメントを実行することができます。

```
SET INTEGRITY FOR DEPARTMENT ALLOW WRITE ACCESS,
   EMPLOYEE ALLOW WRITE ACCESS,
   IMMEDIATE CHECKED FOR EXCEPTION IN DEPARTMENT,
   USE DEPARTMENT_EX,
   IN EMPLOYEE USE EMPLOYEE_EX
```

インポート・ユーティリティーは、参照制約によって次のような影響を受けます。

- オブジェクト表にそれ自体以外の従属表がある場合、 REPLACE および REPLACE CREATE の機能は使用できません。

  これらの関数を使用する場合は、まずその表が親表となっている外部キーをすべてドロップしてください。インポートが終了したら、ALTER TABLE ステートメントで外部キーを再作成してください。
- 自己参照制約が入っている表へのインポートが成功するかどうかは、行をインポートする順番によります。

オブジェクトを変更するときのステートメント従属関係

ステートメントの従属関係には、パッケージ、およびキャッシュに入った動的 SQL および XQuery ステートメントが含まれます。*パッケージ* とは、データベース・オブジェクトの 1 つで、データベース・マネージャーが、特定のアプリケーション・プログラムにとって最も効率的な方法でデータにアクセスするのに必要な情報が入っています。*バインディング* とは、データベース・マネージャーが、アプリケーションの実行時にデータベースにアクセスするのに必要なパッケージを作成するプロセスです。

パッケージおよびキャッシュに入った動的 SQL および XQuery ステートメントは、さまざまなタイプのオブジェクトに従属することができます。

そうしたオブジェクトは、明示的に参照できます。 SQL SELECT ステートメントに含める表やユーザー定義関数などはその例です。また、オブジェクトの暗黙的な参照も可能です。例えば、親表の行の削除時に、**参照制約** の違反がないかどうかの検査が必要な従属表がこれに該当します。パッケージはさらに、パッケージ作成者に付与される特権にも依存しています。

パッケージ、またはキャッシュに入った動的照会ステートメントがオブジェクトに依存しており、そのオブジェクトがドロップされた場合は、そのパッケージまたはキャッシュに入った動的照会ステートメントは「無効」状態になります。パッケージがユーザー定義関数に従属し、その関数がドロップされると、パッケージは次のような「作動不能」状態になります。

- キャッシュに入れられた動的 SQL または XQuery ステートメント (無効状態) は再び、次に使用する際に自動的に最適化されます。ステートメントに必要なオブジェクトがドロップされている場合に動的 SQL または XQuery ステートメントを実行すると、失敗してエラー・メッセージが表示されることがあります。
- 無効状態にあるパッケージは、次に使用する際に暗黙的に再バインドされます。そのようなパッケージは明示的に再バインドすることもできます。トリガーがドロップされたためにパッケージに無効のマークが付けられた場合、再バインド・パッケージはトリガーを呼び出さなくなります。
- 作動不能のパッケージは、明示的に再バインドした後でなければ使用できません。

フェデレーテッド・データベースのオブジェクトには、同様の従属関係があります。例えば、サーバーをドロップすると、そのサーバーに関連付けられたニックネームを参照するパッケージ、またはキャッシュに入れられた動的 SQL は無効になります。

ある場合には、パッケージを再バインドできないことがあります。例えば、表がドロップされたのに再作成されない場合は、パッケージを再バインドできません。この場合、オブジェクトを再作成するか、ドロップされたオブジェクトをアプリケーションが使用しないようにアプリケーションを変更しなければなりません。

その他のほとんどの場合 (例えば**制約** の 1 つがドロップされた場合) には、パッケージを再バインドすることが可能です。

以下のシステム・カタログ・ビューは、パッケージの状態およびパッケージの従属関係を判別するのに役立ちます。

- SYSCAT.PACKAGEAUTH
- SYSCAT.PACKAGEDEP
- SYSCAT.PACKAGES

インフォメーショナル制約の設計

レコードが挿入または更新されたときにデータベース・マネージャーが制約を実施すると、システムのオーバーヘッドが増加する場合があります。参照整合性制約を持つ多数のレコードをロードするときは特にそう言えます。レコードを表に挿入する前に情報の検証がアプリケーションによって既に行われている場合は、通常の制約ではなく、インフォメーショナル制約を使用する方がより効率的かもしれません。

インフォメーショナル制約はデータが準拠する規則をデータベース・マネージャーに伝えますが、データベース・マネージャーはその規則を実施しません。ただし、この情報は DB2 オプティマイザーによって利用されます。これにより、SQL 照会のパフォーマンスが向上する可能性があります。

以下の例では、インフォメーショナル制約の使い方とその機能を例示しています。この単純な表には、応募者の年齢と性別に関する情報が含まれています。

```
CREATE TABLE APPLICANTS
  (
   AP_NO INT NOT NULL,
     GENDER  CHAR(1) NOT NULL,
CONSTRAINT GENDEROK
   CHECK (GENDER IN ('M', 'F'))
   NOT ENFORCED
   ENABLE QUERY OPTIMIZATION,
      AGE  INT NOT NULL,
CONSTRAINT AGEOK
   CHECK (AGE BETWEEN 1 AND 80)
   NOT ENFORCED
   ENABLE QUERY OPTIMIZATION,
 );
```

この例には、列の制約の動作を変える 2 つの節が含まれています。最初のものは NOT ENFORCED というオプションです。これは、データが挿入または更新されるときにこの列のチェックを実施しないようにデータベース・マネージャーに指示します。

2 番目は ENABLE QUERY OPTIMIZATION というオプションです。これは、SELECT ステートメントがこの表に対して実行されるときにデータベース・マネージャーによって使用されます。この値を指定すると、データベース・マネージャーは SQL を最適化するときに制約内の情報を利用します。

表に NOT ENFORCED オプションが含まれていると、本来の挿入ステートメントとは異なる動作をする場合があります。以下の SQL を APPLICANTS 表に対して実行してもエラーは発生しません。

```
INSERT INTO APPLICANTS VALUES
 (1, 'M', 54),
 (2, 'F', 38),
 (3, 'M', 21),
 (4, 'F', 89),
 (5, 'C', 10),
 (6, 'S',100),
```

応募者番号 5 の性別が子 (child) を表す (C) となっており、応募者番号 6 の性別も不自然で、AGE 列の年齢の限度も超えています。どちらの場合も、制約は NOT ENFORCED なので、データベース・マネージャーは挿入操作を許容します。表に対して SELECT ステートメントを実行すると、結果は次のようになります。

```
SELECT * FROM APPLICANTS
  WHERE GENDER = 'C';

APPLICANT  GENDER  AGE
---------  ------  ---

0 record(s) selected.
```

データベース・マネージャーは表の中で 'C' という値を見つけたにもかかわらず、不正確な応答を照会に戻しています。しかし、この列の制約によって、'M' または 'F' だけが有効な値であることがデータベース・マネージャーに伝えられます。ENABLE QUERY OPTIMIZATION キーワードも、ステートメントを最適化すると

きにデータベース・マネージャーがこの制約情報を利用できるようにしています。そのような動作を望まない場合は、ALTER TABLE ステートメントを使って次のように制約を変更する必要があります。

```
ALTER TABLE APPLICANTS
  ALTER CHECK AGEOK DISABLE QUERY OPTIMIZATION
```

この照会が再発行されると、データベース・マネージャーは次の正しい結果を戻します。

```
SELECT * FROM APPLICANTS
  WHERE SEC = 'C';

APPLICANT GENDER  AGE
---------  ------  ---
        5  C       10

1 record(s) selected.
```

インフォメーショナル制約の使用が最善となるのは、アプリケーション・プログラムがデータの挿入および更新を行っている唯一のアプリケーションであることが確実であるシナリオです。すべての情報 (性別や年齢など) がアプリケーションによって事前にチェック済みの場合、インフォメーショナル制約を使用することでより高速なパフォーマンスを実現でき、努力の重複を回避することができます。さらに、インフォメーショナル制約の使用は、データウェアハウスの設計においても有効です。

制約の作成および変更

制約は、ALTER TABLE ステートメントを使用して既存の表に追加することができます。

制約名は、ALTER TABLE ステートメント内に指定した他のどの制約とも同じにすることはできず、その表内で固有のものでなければなりません (これには、定義されたすべての参照整合性制約の名前も含まれます)。ステートメントが正常実行される前に、既存のデータが新しい条件に対してチェックされます。

ユニーク制約の作成および変更

ユニーク制約を既存の表に追加することができます。制約名は、ALTER TABLE ステートメント内に指定した他のどの制約とも同じにすることはできず、その表内で固有のものでなければなりません (これには、定義されたすべての参照整合性制約の名前も含まれます)。ステートメントが正常実行される前に、既存のデータが新しい条件に対してチェックされます。

コマンド行を使用してユニーク制約を定義するには、ALTER TABLE ステートメントの ADD CONSTRAINT オプションを使用します。例えば、以下のステートメントは、表内の従業員を固有のものとして識別するための新しい方法を表す、 EMPLOYEE 表に対するユニーク制約を追加します。

```
ALTER TABLE EMPLOYEE
   ADD CONSTRAINT NEWID UNIQUE(EMPNO,HIREDATE)
```

この制約を変更するには、これをドロップしてから再作成することが必要な場合があります。

主キー制約の作成および変更

主キー制約を既存の表に追加することができます。制約名は、その表内で固有のものでなければなりません (これには、定義されたすべての参照整合性制約の名前も含まれます)。ステートメントが正常実行される前に、既存のデータが新しい条件に対してチェックされます。

コマンド行を使用して主キーを追加するには、以下のように入力します。

```
ALTER TABLE <name>
   ADD CONSTRAINT <column_name>
   PRIMARY KEY <column_name>
```

既存の制約を変更することはできません。主キーとして別の列または列セットを定義するには、最初に既存の主キー定義をドロップしてから再作成する必要があります。

チェック制約の作成および変更

表チェック制約が追加されると、表の挿入または更新を行うパッケージおよびキャッシュに入った動的 SQL には、無効のマークが付けられる場合があります。

コマンド行を使用して表チェック制約を追加するには、以下のように入力します。

```
ALTER TABLE EMPLOYEE
   ADD CONSTRAINT REVENUE CHECK (SALARY + COMM > 25000)
```

この制約を変更するには、これをドロップしてから再作成することが必要な場合があります。

外部キー (参照) 制約の作成および変更

外部キーは、別の表のデータ値への参照です。外部キー制約にはさまざまなタイプがあります。

外部キーが表に追加されると、以下のステートメントが含まれるパッケージまたはキャッシュに入った動的 SQL には、無効のマークが付けられる場合があります。

- 外部キーが入っている表の挿入または更新を行うステートメント
- 親表の更新または削除を行うステートメント

コマンド行を使用して外部キーを追加するには、以下のように入力します。

```
ALTER TABLE <name>
   ADD CONSTRAINT <column_name>
    FOREIGN KEY <column_name>
    ON DELETE <action_type>
    ON UPDATE <action_type>
```

以下の例は、表に主キーと外部キーを追加するための ALTER TABLE ステートメントを示しています。

```
ALTER TABLE PROJECT
  ADD CONSTRAINT PROJECT_KEY
      PRIMARY KEY (PROJNO)
ALTER TABLE EMP_ACT
  ADD CONSTRAINT ACTIVITY_KEY
      PRIMARY KEY (EMPNO, PROJNO, ACTNO)
  ADD CONSTRAINT ACT_EMP_REF
      FOREIGN KEY (EMPNO)
      REFERENCES EMPLOYEE
      ON DELETE  RESTRICT
```

```
ADD CONSTRAINT ACT_PROJ_REF
    FOREIGN KEY (PROJNO)
    REFERENCES PROJECT
    ON DELETE CASCADE
```

この制約を変更するには、これをドロップしてから再作成することが必要な場合があります。

インフォメーショナル制約の作成および変更

照会のパフォーマンスを向上させるために、インフォメーショナル制約を表に追加することができます。NOT ENFORCED オプションを DDL に指定する場合、CREATE TABLE または ALTER TABLE ステートメントを使用してインフォメーショナル制約を追加します。

制約事項: 表にインフォメーショナル制約を定義した後は、インフォメーショナル制約を除去してからでないと、その表の列名を変更することができません。

コマンド行を使用して表にインフォメーショナル制約を指定するには、新規表に対して以下のコマンドを入力します。

```
ALTER TABLE <name> <constraint attributes> NOT ENFORCED
```

ENFORCED または NOT ENFORCED: 挿入、更新、削除などの通常の操作中に、データベース・マネージャーによって制約が課せられるかどうかを指定します。

- 機能従属関係に ENFORCED を指定することはできません (SQLSTATE 42621)。
- この制約に適合することが分かっている表データだけに、NOT ENFORCED を指定してください。データが実際には制約に準拠していない場合、照会結果が予測不能になる可能性があります。

この制約を変更するには、これをドロップしてから再作成することが必要な場合があります。

表の制約定義の表示

表の制約定義は、SYSCAT.INDEXES および SYSCAT.REFERENCES カタログ・ビューにあります。

SYSCAT.INDEXES ビューの UNIQUERULE 列は、索引の特性を示します。この列の値が P の場合、索引は主キーです。U の場合、索引はユニーク索引です (しかし、主キーではありません)。

SYSCAT.REFERENCES カタログ・ビューには、参照整合性 (外部キー) 制約情報が含まれています。

制約のドロップ

表チェック制約を明示的にドロップするには、ALTER TABLE ステートメントを使います。また、DROP TABLE ステートメントを使うと、暗黙のうちにチェック制約がドロップされます。

制約をドロップするには、DROP または DROP CONSTRAINT 節を指定して ALTER TABLE ステートメントを使用してください。これによって変更を反映した列が含まれる表のバインドとアクセスの続行が可能になります。表上のすべてのユニーク制約の名前は、SYSCAT.INDEXES システム・カタログ・ビューの中にあります。

ユニーク制約のドロップ

ALTER TABLE ステートメントを使用して、明示的にユニーク制約をドロップすることができます。

ALTER TABLE ステートメントの DROP UNIQUE 節は、ユニーク制約 constraint-name の定義、およびこのユニーク制約に従属するすべての参照制約をドロップします。 constraint-name には、既存のユニーク制約を指定する必要があります。

```
ALTER TABLE <table-name>
  DROP UNIQUE <constraint-name>
```

このユニーク制約をドロップすると、その制約を使用しているすべてのパッケージまたはキャッシュに入った動的 SQL を無効にします。

主キー制約のドロップ

主キー制約をドロップするには、ALTER TABLE ステートメントの DROP PRIMARY KEY 節を使用します。

ALTER TABLE ステートメントの DROP PRIMARY KEY 節は、主キーの定義、およびその主キーに従属するすべての参照制約をドロップします。表には主キーがなければなりません。コマンド行を使用して主キーをドロップするには、以下のように入力します。

```
ALTER TABLE <table-name>
  DROP PRIMARY KEY
```

(表) チェック制約のドロップ

チェック制約をドロップすると、その表への INSERT または UPDATE 依存関係を持つすべてのパッケージおよびキャッシュに入った動的ステートメントは、無効にされます。表内のすべてのチェック制約の名前は、SYSCAT.CHECKS カタログ・ビューの中にあります。システム生成された名前を持つ表チェック制約をドロップする際には、SYSCAT.CHECKS カタログ・ビューで名前を確認してください。

次のステートメントは、チェック制約 constraint-name をドロップするものです。 constraint-name には、表に定義されている既存のチェック制約を指定する必要があります。コマンド行を使用して表チェック制約をドロップするには、以下のようにします。

```
ALTER TABLE <table_name>
  DROP <check_constraint_name>
```

外部キー (参照) 制約のドロップ

外部キー制約をドロップするには、ALTER TABLE ステートメントの DROP CONSTRAINT 節を使用します。

ALTER TABLE ステートメントの DROP CONSTRAINT 節は、制約 *constraint-name* をドロップします。 *constraint-name* には、表に定義されて

いる既存の外部キー制約、主キー、またはユニーク制約のいずれかを指定する必要があります。コマンド行を使用して外部キーをドロップするには、以下のように入力します。

```
ALTER TABLE <table-name>
  DROP FOREIGN KEY <foreign_key_name>
```

以下の例は、 ALTER TABLE ステートメントの DROP PRIMARY KEY および DROP FOREIGN KEY 節を使用して、表の主キーと外部キーをドロップします。

```
ALTER TABLE EMP_ACT
  DROP PRIMARY KEY
  DROP FOREIGN KEY ACT_EMP_REF
  DROP FOREIGN KEY ACT_PROJ_REF
ALTER TABLE PROJECT
  DROP PRIMARY KEY
```

外部キー制約がドロップされると、以下のものを含むパッケージまたはキャッシュに入った動的ステートメントには、無効のマークが付けられる場合があります。

- 外部キーが入っている表の挿入または更新を行うステートメント
- 親表の更新または削除を行うステートメント

第 13 章 索引

索引 は 1 つ以上のキーのセットであり、それぞれのキーは表の行を指しています。 SQL *オプティマイザー* は表のデータにアクセスする効率的な方法を自動的に選択します。オプティマイザーはデータへの一番速いアクセス・パスを判別するときに、索引を考慮に入れます。

注: すべての索引が表の行を指示するわけではありません。MDC ブロック索引は、データのエクステント (または、ブロック) を指します。 XML データ用の XML 索引では、特定の XML パターン式を使用して、単一の列に格納された XML 文書内にあるパスおよび値の索引付けを行います。その列のデータ・タイプは XML でなければなりません。 MDC ブロック索引と XML 索引の両方が、システム生成の索引です。

索引は、以下の目的でデータベース・マネージャーによって使用されます。

- パフォーマンスを改善する。ほとんどの場合、索引を使うとデータへのアクセスが速くなります。索引をビューに対して作成することはできませんが、ビューのベースとなる表に対して索引を作成することで、そのビューに対する操作のパフォーマンスを改善できる場合があります。
- 固有性を確保する。ユニーク索引を持つ表に、同一のキーを持つ行を含めることはできません。

表にデータを追加すると、その表や追加データに対して別の操作が実行されていないかぎり、データは表の下部に追加されます。データに順序はありません。 特定の行データを検索するときは、表の各行の最初から最後までチェックする必要があります。 特定の順序で表内のデータにアクセスするには、索引を使用します。

データの行に列値を使用することにより、その行全体を識別することができます。行を識別するため、1 つ以上の列が必要な場合があります。そのような列のことを *キー* といいます。 1 つの列を複数のキーで使用できます。

索引はキー内の値によって配列されます。キーはユニーク・キーか非ユニーク・キーのどちらかになります。各表には少なくとも 1 つのユニーク・キーが必要ですが、他にも非ユニーク・キーをいくつか含めることができます。各索引のキーは 1 つだけです。 例えば、索引に従業員 ID 番号 (固有) を使用し、別の索引に部門番号 (非固有) を使用できます。

例

328 ページの図 24 の表 A には、表の従業員番号に基づいた索引があります。このキー値は、表の行を指すポインターの機能を提供します。例えば、従業員番号 19 は、従業員 KMP を指しています。 索引では、ポインターを介してデータへのパスを作成できるので、効率よく表の行にアクセスできます。

ユニーク索引は、索引キーが必ず固有になるようにするために作成できます。*索引キー* は、索引が定義されている列または列の順序付き集合です。ユニーク索引を使用すると、索引になっている列にある索引キーごとの値または列の値が必ず固有のものとなります。

図 24 は、索引と表のリレーションシップを示しています。

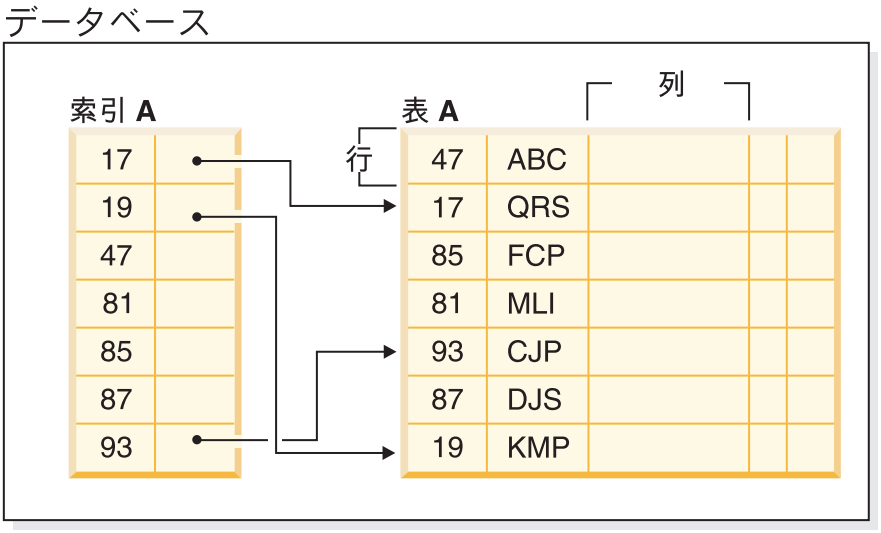


図 24. 索引と表の関係

329 ページの図 25 では、データベース・オブジェクトのいくつかの関連を図示しています。この図はまた、表、索引、ロング・データが表スペースに保管されている様子も示しています。

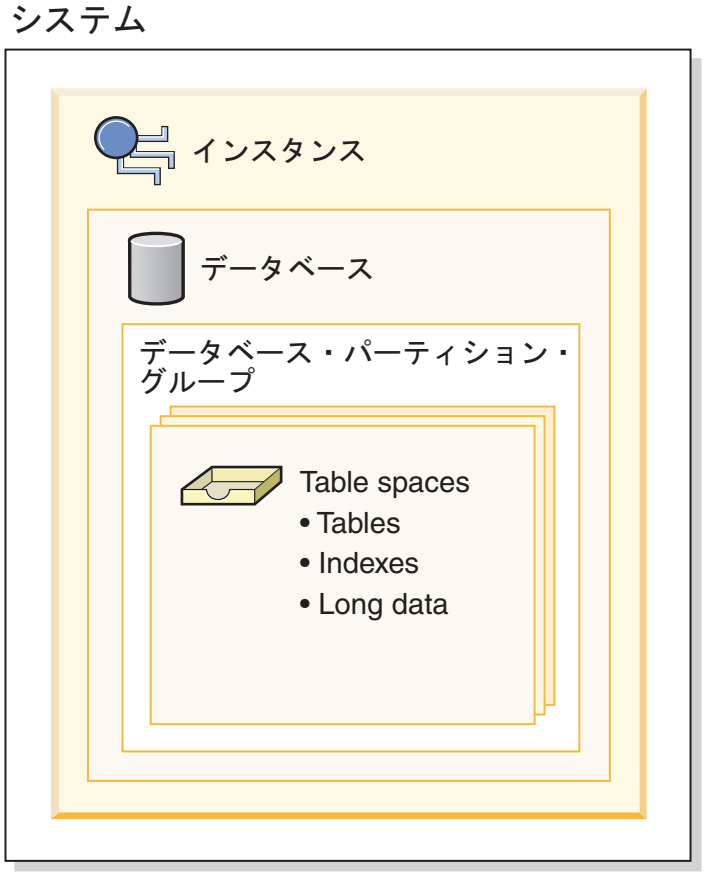


図 25. 一部のデータベース・オブジェクトの相互関係

索引のタイプ

索引には、ユニーク索引、非ユニーク索引、クラスター索引、非クラスター索引、およびマルチディメンション・クラスタリング (MDC) 表用のシステム生成のブロック索引という 5 つのタイプがあります。

ユニーク索引と非ユニーク索引

ユニーク索引は、表内のデータの 2 つの行が同一のキー値を持たないようにすることによりデータ整合性の維持に貢献する索引です。

すでにデータが入っている表に対してユニーク索引を作成しようとすると、索引を構成する列 (複数の場合もある) の値の固有性が検査されます。表の中に重複キー値を持つ行が含まれていると、索引の作成プロセスは失敗します。表に対してユニーク索引が定義されると、この索引内でキーが追加または変更されるときには必ず、固有性が強制されます。 (いくつか例を挙げると INSERT、UPDATE、LOAD、IMPORT、SET INTEGRITY などがあります。) データ値の固有性を強制するだけでなく、ユニーク索引を使用することによって照会処理中のデータ・リトリーブ・パフォーマンスを改善することもできます。

一方、非ユニーク索引は、それが関連付けられる表に対して制約を適用するために使用されることはありません。その代わり、非ユニーク索引は、頻繁に使用されるデータ値のソート順序を維持することにより照会のパフォーマンスを改善するためだけに使用されます。

クラスター索引と非クラスター索引

索引の体系は、クラスターと非クラスターに分類されます。クラスター索引とは、データ・ページ内の行の順序が索引内の行の順序と一致している索引です。このため、特定の表においてクラスター索引は 1 つしか含められないのに対し、非クラスター索引は表内に多数含めることができます。一部のリレーショナル・データベース管理システムにおいて、クラスター索引のリーフ・ノードは、どこか別の場所にあるデータのポインターではなく、実際のデータと対応します。

クラスター索引と非クラスター索引の両方で、索引構造内にキーとレコード ID のみが含まれます。レコード ID により、データ・ページ内の行が必ず指示されます。クラスター索引と非クラスター索引の唯一の違いは、データベース・マネージャーにより、対応するキーが索引ページに出現する順序と同じ順序でデータをデータ・ページに保持しようと試みられる点です。このため、データベース・マネージャーでは、行を類似のキーで同じページへ挿入しようと試みられます。表が再編成されると、行は索引キーの順序でデータ・ページに挿入されます。

選択された索引に関する表を再編成すると、データが再クラスター化されます。クラスター索引は、表内のデータのより良い順次アクセスを可能にするので、範囲述部のある列に最も役立ちます。このようにすると、類似値が同一のデータ・ページにあるため、ページ・フェッチは少なくなります。

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。

クラスター索引によるパフォーマンスの改善

クラスタリング索引を使用するとほとんどの場合、照会操作のパフォーマンスが改善されます。これは、クラスタリング索引がページに格納されているデータへのアクセス・パスをより線形化するためです。さらに、同様の索引キー値を持つ行が一緒に格納されるため、クラスタリング索引が使用されるときのプリフェッチが、通常はより効率的になります。

ただし、クラスタリング索引は、CREATE TABLE ステートメントで使用される表定義の一部として指定することができません。その代わり、クラスタリング索引は、CLUSTER オプションを指定して CREATE INDEX ステートメントを実行することによってのみ作成されます。その後、ALTER TABLE ステートメントを使用して、表に対して作成されたクラスタリング索引に対応する主キーを追加する必要があります。このクラスタリング索引が、表の主キー索引として使用されます。

注: 表内の PCTFREE を ALTER TABLE ステートメントを使用して適切な値に設定すると、行を類似の値でページに挿入するための適切なフリー・スペースを残しておくことによって、表がクラスタリングされた状態で維持されるために役立つ可能性があります。詳しくは、『ALTER TABLE ステートメント』および『表および索引の再編成の必要性を少なくする』を参照してください。

一般的に、クラスタリング索引がユニーク索引であれば、クラスタリングの保守はより効果的に行えます。

主キーまたはユニーク・キー制約とユニーク索引の違い

覚えておくべき重要な点として、1 次ユニーク・キー制約とユニーク索引に大きな違いはありません。データベース・マネージャーはユニーク索引と NOT NULL 制約を組み合わせて使うことにより、主キーおよびユニーク・キー制約のリレーショナル・データベース概念を実現します。したがって、ユニーク索引は NULL 値を許可するので、単独で主キー制約を適用することはありません。 (NULL 値は不明な値を表すものの、索引に関して言えば、他の NULL 値と同等として扱われます。)

そのため、ユニーク索引が 1 つの列で構成される場合、NULL 値は 1 つしか許容されません。 NULL 値が複数存在すると、ユニーク制約違反になります。同様にユニーク索引が複数の列で構成される場合も、値と NULL の特定の組み合わせは一度しか使用できません。

双方向インデックス

双方向インデックスでは、デフォルトで順方向と逆方向の両方のスキャンが可能です。 CREATE INDEX ステートメントの ALLOW REVERSE SCANS 節は、順方向と逆方向両方の索引スキャンが行えるようにします。つまり、索引スキャンを索引作成時に定義した順序で行うことも、あるいはその反対の順序 (逆方向) で行うこともできます。このオプションによって、次のことが可能になります。

- MIN 関数および MAX 関数を容易化する
- 前のキーをフェッチする
- データベース・マネージャーが逆方向スキャンのために一時表を作成しなくてもすむようにする
- 冗長な逆順の索引をなくす

DISALLOW REVERSE SCANS を指定すると、索引を逆順でスキャンすることはできません。 (しかし、この索引は、物理的には ALLOW REVERSE SCANS 索引と全く同じになります。)

索引の設計

索引は、通常、表へのアクセスを高速にするために使用されます。しかし索引は、論理データ設計の点でも役立ちます。

例えば、ユニーク索引を使うと、列に重複値を入力できなくなるため、表内の行が同じものになることはありません。また、列の値を昇順または降順に配列するためにも、索引を作成することができます。

注: 索引を作成するときに覚えておくべき点があります。それは、索引によって読み取りパフォーマンスは改善される可能性があっても、書き込みパフォーマンスにはマイナスの影響が出るという点です。これが生じるのは、データベース・マネージャーが表に行を書き込むたびに、影響を受ける索引もすべて更新しなければならないためです。したがって、索引の作成は、パフォーマンスが全体的に改善されることが明らかな場合にのみ行ってください。

索引を作成するときは、表の構造や、索引に対して最も頻繁に実行される照会のタイプも考慮に入れる必要があります。例えば、頻繁に発行される照会の WHERE 節に現れる列などは、索引を作成することが得策です。しかし、それほど頻繁に実行されない照会の場合は、索引が、INSERT や UPDATE ステートメントの実行によりパフォーマンスに与える悪影響の方が、その利点を上回る可能性があります。

同様に、頻繁に発行される照会の GROUP BY 節に関係する列の場合も索引を作成することによってパフォーマンスが改善されるかもしれません。行をグループ化するために使用される値の数が、グループ化される行の数と比べて少ないような場合は特にそうです。

索引を設計するときのガイドラインと考慮事項

- *索引* は、列ごとに表の中に定義されます。索引は、表の作成者、または特定の列では直接アクセスが必要であることについて知っているユーザーが作成できます。 1 次索引のキーは、ユーザー定義の索引がすでに存在しているのでない限り、主キーに対して自動的に作成されます。
- *索引キー* は 1 つの列または複数の列の集合のことであり、そこに索引が定義され、索引の有効性が判別されます。索引キーを構成する列の順序は、索引キーの作成に影響を与えることはありませんが、索引を使用するかどうかを決定するときに、オプティマイザーに影響を与える可能性があります。
- 1 つの表に対して、索引はいくつでも定義でき、こうして照会のパフォーマンスを高めることができます。索引マネージャーは更新、削除、および挿入操作中、索引を保守する必要があります。更新事項の多い表に対してたくさんの索引を作成すると、要求の処理がスローダウンする可能性があります。同様に、索引キーが大きくなっても、要求の処理速度が低下することがあります。したがって、索引の使用は、頻繁にアクセスするための利点があるということが明らかな場合だけにしてください。
- ユニーク索引キーの一部ではないものの、その索引で保管され保守される列データは、組み込み列と呼ばれます。組み込み列を指定できるのは、ユニーク索引についてだけです。組み込み列のある索引を作成しているときは、ユニーク・キー列だけがソートされ、固有のものであると見なされます。列の組み込みを使用することにより、データ・リトリーブのための索引のみアクセスを使用可能にすることができ、こうしてパフォーマンスを改善することができます。
- 索引を作成する表が空であっても、索引は作成されますが、表がロードされるか行が挿入される時点まで索引項目は作成されません。表が空でない場合、CREATE INDEX ステートメントの処理中に索引項目がデータベース・マネージャーによって作成されます。
- *クラスタリング索引* では、データベース・マネージャーにより、表の新しい行が (索引によって定義される) 近似したキー値を持つ既存の行と物理的に近い位置に配置されるように試みられます。
- *主キーの索引* をクラスタリング索引にする場合は、CREATE TABLE ステートメントで主キーを指定しないでください。主キーが作成されると、関連する索引を変更することはできなくなります。主キー節を指定しないで CREATE TABLE を発行します。その後、クラスタリング属性を指定して CREATE INDEX を発行します。最後に、ALTER TABLE ステートメントを使用して、作成されたばかりの索引に対応する主キーを追加します。この索引が、主キーの索引として使用されます。

- 索引はディスク・スペースを消費します。ディスク・スペースの量は、キー列の長さや索引付けされる行の数によって異なります。索引のサイズは、表にデータを挿入すればするほど大きくなります。したがって、データベースのサイズを計画するときは、索引付けされるデータの量を考慮に入れてください。索引付けをサイジングするときの考慮事項としては次のものが含まれます。
  - 主キーおよびユニーク・キー制約を適用すると必ず、システム生成のユニーク索引が作成されます。
  - MDC 表の作成では、システム生成のブロック索引も作成されます。
  - XML 列があると必ず、システム生成の索引が作成されます。
  - 通常、外部キー制約の列に対して索引を作成するとプラスの影響が現れます。

  注: 1 つの索引あたりの列の最大数は 64 です。ただし、型付き表の索引を作成する場合は、1 つの索引あたりの列の最大数は 63 です。すべてのオーバーヘッドも含めた索引キーの最大長は、*indexpagesize*/4 です。 1 つの表で許可される最大索引数は 32 767 です。索引キーの最大長は、ページ・サイズあたりの索引キーの長さの限界以下でなければなりません。列の保管時の長さについては、『CREATE TABLE ステートメント』を参照してください。キーの長さの限界については、『SQL と XQuery の制限値』のトピックを参照してください。

  注:

索引設計のためのツール

表を作成した後、データベース・マネージャーがそこからデータをリトリーブする速度について考える必要があります。設計アドバイザーまたは db2advis コマンドの使用は、索引を設計する上で役立ちます。

有用な索引を表に対して作成することにより、照会のパフォーマンスを著しく改善することができます。本の索引と同じように、表に対する索引も特定の情報を素早く、最小限の検索によって見つけることができるようにします。索引を使って特定の行を表からリトリーブすることで、データベース・マネージャーが実行する必要のある高コストの入出力操作の数を減らすことができます。こうして、索引を使用することによって、すべての一致が見つかるまでデータベース・マネージャーがすべてのデータ・ページを徹底的に検索するのではなく、比較的少ないデータ・ページの読み取りを行うことによって行を見つけることができます。

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)
- 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
- ワークロード・パフォーマンスのチューニング

索引のスペース所要量

索引を設計するときは、そのスペース所要量について留意する必要があります。

各索引に必要なスペースは、次のようにして見積もることができます。

```
(average index key size + index key overhead) * number of rows * 2
```

説明:

- 「(average index key size (平均索引キー・サイズ)」は、索引キーの中の各列のバイト・カウントです。 (VARCHAR および VARGRAPHIC 列の平均の列サイズを見積もるには、現行データ・サイズの平均に 2 バイトを加えます。宣言されている最大サイズは使用しません。)
- 「index key overhead (索引キーのオーバーヘッド)」は、索引が作成される表のタイプに依存します。 LARGE 表の場合は、XML 索引の有無に関係なく値は 11 です。ただし、表がパーティション化されている場合の値は 13 です。
  REGULAR 表の場合は、XML 索引がないときの値は 9、XML 索引があるときの値は 11 です。パーティション化されている REGULAR 表の場合は、すべて値は 11 です。
- 「2」倍しているのは、ノンリーフ・ページやフリー・スペースなどのオーバーヘッドのためです。

注:

1. NULL 値が可能な列の場合は、NULL 値標識のために 1 バイトをさらに追加してください。
2. マルチディメンション・クラスタリング (MDC) 表に対して内部的に作成されたブロック索引の場合、「number of rows (行数)」を「number of blocks (ブロック数)」で置き換えることになります。

XML 列の各索引に必要なスペースは、次のようにして見積もることができます。

```
(average index key size + index key overhead) * number of indexed nodes * 2
```

説明:

- 「number of blocks (平均索引キー・サイズ)」は、索引を構成しているキー・パーツの合計です。 XML 索引は、いくつかの XML キー・パーツと値 (sql-data-type) で構成されています。

  ```
  fixed overhead + variable overhead + byte count of sql-data-type
  ```

  説明:

  – 「fixed overhead (固定オーバーヘッド)」は 14 バイトです。

– 「fixed overhead (可変オーバーヘッド)」は、索引付きノードの平均の深さに 4 バイトを加算した値です。
– sql-data-type 値のバイト・カウントは、SQL と同じ規則に従います。

- 「number of indexed nodes (索引ノード数)」は、挿入される文書の数に、索引定義の XML パターン式 (XMLPATTERN) を満たすサンプル文書内のノードの数を乗算した値です。

バージョン 8 より前に作成された索引 (タイプ 1 索引) は、バージョン 8 以降に作成された索引 (タイプ 2 索引) と異なります。表にどのタイプの索引が存在するかを判別するには、ADMIN_GET_TAB_INFO 表関数を使用してください。タイプ 1 索引をタイプ 2 索引に変換するには、REORG INDEXES .... CONVERT コマンドを使用します。

REORG INDEXES コマンドを使用するときは、索引が格納される表スペースに十分なフリー・スペースを確保しておいてください。フリー・スペースの量は、現行の索引サイズと同等であるべきです。 ALLOW WRITE ACCESS オプションを使って索引を再編成する場合は、追加スペースが必要になるかもしれません。その追加スペースは、索引の再編成中に索引に影響を与えるアクティビティーのログのためのものです。

索引の作成時には一時スペースが必要になります。索引作成時に必要な一時スペースの最大量は、次のようにして見積もることができます。

```
(average index key size + index key overhead) * number of rows * 3.2
```

または

```
(average index key size + index key overhead) * number of indexed nodes * 3.2
```

ここで「3.2」は索引オーバーヘッドおよび索引の作成時のソートに必要なスペースに起因する値です。

注: 非ユニーク索引の場合、重複キー項目を保管するために、5 バイトだけが必要です。上記に示した見積もりは、重複がないものと想定しています。 1 つの索引を保管するために必要なスペースは、上記の公式では余分に見積もってしまう場合があります。

「number of index nodes (索引ノード数)」が 64 KB のデータを超える場合は、挿入時に一時スペースが必要です。一時スペースの量は、次のようにして見積もることができます。

```
(average index key size) * number of indexed nodes * 1.2
```

リーフ・ページごとのキーの数を見積もるために、以下の 2 つの公式を使用できます (2 番目の方がより正確な見積もりが可能です)。これらの見積もりの精度は、平均値がどの程度うまく実際のデータを反映しているかに大きく依存しています。

注: SMS 表スペースの場合、リーフ・ページに必要な最小スペースは 12 KB です。 DMS 表スペースの場合、最小は 1 エクステントです。

- リーフ・ページ当たりのキーの数の平均値は、おおよそ以下のように見積もることができます。

```
(.9 * (U - (M*2))) * (D + 1)
----------------------------
      K + 7 + (5 * D)
```

説明:

– U (ページでの使用可能なスペース) は、ページ・サイズから 100 を引いた数値とほぼ等しい。ページ・サイズが 4096 の場合、U は 3996。

– M = U / (9 + *minimumKeySize*)

– D = キー値当たりの重複の平均数

– K = *averageKeySize*

minimumKeySize と *averageKeysize* は、それぞれ NULL 可能なキー部分のために余分に 1 バイト、および各可変長キー部分のために余分に 2 バイトが必要であることに注意してください。

組み込み列がある場合は、それらを *minimumKeySize* と *averageKeySize* 用に計算に入れる必要があります。

「minimumKeySize」は、索引を構成しているキー・パーツの合計です。

```
fixed overhead + variable overhead + byte count of sql-data-type
```

説明:

– 「fixed overhead (固定オーバーヘッド)」は 13 バイトです。

– 「fixed overhead (可変オーバーヘッド)」は、索引付きノードの最小の深さに 4 バイトを加算した値です。

– sql-data-type 値のバイト・カウントは、SQL と同じ規則に従います。

索引作成中に、デフォルト値の 10 % 以外の空きパーセント値が指定されている場合、 .9 は、(100 - pctfree)/100 の値で置き換えることができます。

- リーフ・ページ当たりのキーの数の平均値をより正確に見積もるには、以下のようにします。

```
L = number of leaf pages = X / (avg number of keys on leaf page)
```

ここで X は表の中の行の総数です。

XML 列の索引の場合、X は列内にある索引付きノードの総数です。

次のようにして索引の元のサイズを見積もることができます。

```
(L + 2L/(average number of keys on leaf page)) * pagesize
```

DMS 表スペースの場合、1 つの表上のすべての索引について合計サイズを算出し、索引が存在する表スペースのエクステント・サイズの倍数に切り上げます。

挿入 (INSERT) /更新 (UPDATE) アクティビティーによる索引の増加のための追加スペースを設ける必要がありますが、これによりページの分割が生じる場合があります。

元の索引サイズのより正確な見積もりと、索引の中のレベルの数の見積もりを出すには、以下の計算式を使用します。 (これは、索引定義で組み込み列が使用さ

れている場合は特に注意を引くものとなります。) ノンリーフ・ページ当たりのキーの数の平均値は、おおよそ以下のようになります。

```
(.9 * (U - (M*2))) * (D + 1)
----------------------------
      K + 13 + (9 * D)
```

説明:

- U (ページでの使用可能なスペース) は、ページ・サイズから 100 を引いた数値とほぼ等しい。ページ・サイズが 4096 の場合、U は 3996。
- D はノンリーフ・ページ上のキー値あたりの平均重複数 (この数はリーフ・ページ上での数よりもずっと小さいので、この値を 0 に設定して計算を単純化することもできます)。
- M = U / (9 + ノンリーフ・ページの *minimumKeySize*)
- K = ノンリーフ・ページの *averageKeySize*

ノンリーフ・ページの *minimumKeySize* および *averageKeySize* は、組み込み列が存在する場合を除いて、リーフ・ページのものと同じです。組み込み列は、ノンリーフ・ページ上には保管されません。

`(100 - pctfree)/100` の値が `.9` より大きくなければ、この値を `.9` で置き換えることはできません。なぜなら、索引作成中に最大 10 % のフリー・スペースがノンリーフ・ページに残されるからです。

ノンリーフ・ページの数は、次のようにして見積もることができます。

```
if L > 1 then {P++; Z++}
While (Y > 1)
{
   P = P + Y
   Y = Y / N
  Z++
}
```

説明:

- P はページの数 (初期値は 0)。
- L はリーフ・ページの数。
- N は各ノンリーフ・ページのキーの数。
- Y = L / N
- Z は索引ツリーのレベルの数 (初期値は 1)。

ページの合計数は、

```
T = (L + P + 2) * 1.0002
```

追加の 0.02 % は、スペース・マップ・ページを含むオーバーヘッドのためのものです。

索引を作成するために必要なスペースの量は次のように見積もります。

```
T * pagesize
```

索引の作成

索引は、列内の値を昇順または降順で配列するように作成することができます。索引の作成には、CREATE INDEX ステートメント、DB2 設計アドバイザー、または db2advis 設計アドバイザー・コマンドを使用できます。

例えば、コマンド行から CREATE INDEX ステートメントを使用して索引を作成するには、以下のように入力します。

```
CREATE UNIQUE INDEX EMP_IX
  ON EMPLOYEE(EMPNO)
  INCLUDE(FIRSTNAME, JOB)
```

ユニーク索引だけに適用される INCLUDE 節では、一連の索引キー列に追加列を付加することを指定します。この節に組み込まれる列は、固有性を強制するためには使用されません。これらの組み込み列は、索引のみのアクセスを行うことにより、照会のパフォーマンスを向上させることがあります。このオプションにより、次のことが可能になる場合があります。

- さらに照会を行うためにデータ・ページにアクセスする必要がなくなる。
- 重複する索引が除去される。

この索引がある表に対して SELECT EMPNO, FIRSTNAME, JOB FROM EMPLOYEE が発行された場合、データ・ページを読み取ることなく、索引から、必要なデータをすべて検索することができます。これにより、パフォーマンスが向上します。

注: バージョン 8 以降で作成した索引 (タイプ 2 索引といいます) では、行の削除または更新時にキーが削除済みとしてマークされるだけです。これを疑似削除済みキーといいます。削除または更新がコミットされてしばらくしてからクリーンアップがなされるまで、このようなキーはページから物理的には除去されません。そうしたクリーンアップは、キーが削除済みとしてマークされたページを変更する後続のトランザクションによりなされるかもしれません。疑似削除済みキーのクリーンアップは、REORG INDEXES ユーティリティーの CLEANUP ONLY ALL オプションを使用して、明示的に起動することができます。

1 つのパーティション・データベース環境内の表に対する索引は、同じ CREATE INDEX ステートメントを使用して作成されます。索引内のデータは、その表の分散キーに基づいて分散されます。配分が行われると、データベース・パーティション・グループ内の各データベース・パーティションに B+ ツリーが作成されます。各 B+ ツリーは、そのデータベース・パーティションに属する表の一部に索引を付けます。複数パーティション・データベース上で定義されるユニーク索引内の列は、分散キーの列のスーパーセットでなければなりません。

注: バージョン 9.5 で、Solaris プラットフォーム上では、CREATE INDEX ステートメントは RAW デバイスを使用している場合は停止します。 Sun によりこの問題は修正され、カーネル・パッチでフィックスがリリースされる予定です。

索引の変更

索引を変更する場合は、最初に索引をドロップしてから、索引を再度作成する必要があります。ALTER INDEX ステートメントはありません。

例えば、キー列のリストに列を追加する場合は、前の定義をドロップし、新規の索引を作成しなければなりません。COMMENT ステートメントを使用して、索引の目的を記述するコメントを追加することはできます。

索引の名前変更

RENAME ステートメントを使用して、既存の索引を名前変更することができます。

既存の索引を名前変更するには、コマンド行から次のステートメントを発行します。

```
RENAME INDEX <source index name> TO <target index name>
```

- <source index name> は、名前変更する既存の索引の名前です。名前 (スキーマ名を含む) は、データベースに既に存在する索引を指定していなければなりません。宣言済みのグローバル一時表上の索引の名前は指定できません。スキーマ名は SYSIBM、SYSCAT、SYSFUN、または SYSSTAT であってはなりません。
- <target index name> は、索引の新しい名前をスキーマ名を付けずに指定します。ソース・オブジェクトのスキーマ名が、オブジェクトの新しい名前の修飾に使用されます。修飾名は、データベースに既に存在する索引を指定するものであってはなりません。

索引を名前変更するとき、システム生成索引をソース索引にすることはできません。ステートメントが正常に実行されると、新規索引名を反映してシステム・カタログ表が更新されます。

索引の再作成

完全にはログ記録されていない索引作成のロールフォワードなどの特定のデータベース操作により、索引オブジェクトが無効になる場合があります。これは索引がロールフォワード操作中に作成されないことが原因です。索引オブジェクトは、その中に索引を再作成することでリカバリーすることができます。

データベース・マネージャーは、索引が無効になったことを検出すると、自動的にその再作成を試みます。再作成の実行時には、データベースの **indexrec** パラメーターまたはデータベース・マネージャー構成ファイルにより制御されます。これには、次に示す 5 つを設定できます。

- SYSTEM
- RESTART
- RESTART_NO_REDO
- ACCESS
- ACCESS_NO_REDO

RESTART_NO_REDO と ACCESS_NO_REDO は、RESTART と ACCESS に似ています。

NO_REDO オプションは、CREATE INDEX などの元の操作中に索引が完全にログ記録されていた場合でも、ロールフォワード中には索引は再作成されないが、再始動時または最初のアクセス時のいずれかには作成されることを意味します。詳しくは、**indexrec** パラメーターを参照してください。

データベースの再始動のタイミングが重要でない場合は、データベースを整合状態に戻す処理の一環として、無効な索引を再作成するほうが優れています。このアプローチを使用する場合、データベースの再始動に必要な時間は、索引の再作成処理のせいで長くなりますが、データベースが整合状態に戻った後は、通常の処理に影響が生じることはありません。

一方、索引がアクセスされるときに索引を再作成するなら、データベースを再始動するために要する時間はより早くなりますが、索引を再作成する結果として、予期しない応答時間の低下が生じる可能性があります。例えば、無効な索引を持つ表にアクセスするユーザーは、索引が再作成されるのを待たなければならないかもしれません。さらに、無効な索引の再作成の後、予期しないロックが獲得されて、長期に渡って保持される場合があります。これは特に、索引の再作成の発生原因となったトランザクションが終了 (すなわち、加えられた変更をコミットまたはロールバック) しない場合にそう言えます。

索引のドロップ

索引定義の索引の節は変更できません。索引をドロップしてから再び作成してください。 (索引をドロップしても、他のオブジェクトがドロップされることはありません。しかし、場合によっては一部のパッケージが無効になることがあります。) . DROP ステートメントを使用して索引をドロップします。

主キーまたはユニーク・キーの索引は、明示的にドロップすることはできません。索引をドロップするには、次の方法のいずれかを使用してください。

- 主キーまたはユニーク・キーに対して、 1 次索引またはユニーク制約が自動的に作成されていた場合、主キーまたはユニーク・キーをドロップすると、索引がドロップされることになります。ドロップは、ALTER TABLE ステートメントによって行われます。
- 1 次索引またはユニーク制約がユーザー定義であった場合、ALTER TABLE を使用して、まず主キーまたはユニーク・キーをドロップしなければなりません。主キーまたはユニーク・キーがドロップされた後は、索引は 1 次索引またはユニーク索引とは考えられなくなるため、明示してドロップすることができます。

コマンド行を使用して索引をドロップするには、以下のように入力します。

```
DROP INDEX <index_name>
```

次のステートメントは、PH という索引をドロップするものです。

```
DROP INDEX PH
```

ドロップされる索引に依存するパッケージ、およびキャッシュに入った動的 SQL および XQuery ステートメントには、無効のマークが付けられます。アプリケーション・プログラムは、索引の追加やドロップによる変更事項には影響されません。

第 14 章 トリガー

トリガー は、指定した表に対する挿入、更新、または削除操作への応答として実行されるアクションのセットを定義します。このような SQL 操作が実行されるとき、トリガーが*起動*されるといいます。トリガーはオプションであり、 CREATE TRIGGER ステートメントを使用して定義されます。

データ整合性規則を実施するために、参照制約およびチェック制約とともにトリガーを使用できます。また、トリガーを使用して、他の表への更新を行ったり、挿入または更新される行の値を自動的に生成またはトランスフォームできます。あるいは、関数を呼び出してタスク (アラートを発するなど) を実行することもできます。

トリガーは、*移り変わる* ビジネス規則を定義および実施するための便利な機構です。この規則は、さまざまな状態のデータ (例えば、昇給率が 10 % を超えることのできない給与など) を扱う規則です。

トリガーを使用すると、ビジネス規則を実施する論理をデータベース内に置くことができます。つまり、アプリケーションがそれらの規則の実施を担当しないということです。すべての表に対してロジックを一カ所に集中すれば、ロジックの変更時にアプリケーション・プログラムへの変更が必要ないため、簡単に保守を行えるようになります。

トリガーを作成する際に、以下を指定します。

- サブジェクト*表*。これは、トリガーが定義される表を指定します。
- *トリガー・イベント*。これは、サブジェクト表を変更する特定の SQL 操作を定義します。イベントには、挿入、更新、または削除操作があります。
- *トリガー起動時間*。これは、トリガー・イベントが発生する前か後のどちらで、トリガーを活動化するかを指定します。

トリガーを活動化するステートメントには、*影響を受ける行のセット* が組み込まれています。これらは、挿入、更新、または削除されるサブジェクト表の行です。*トリガー細分性* では、トリガーのアクションを、ステートメントで 1 回実行するか、または影響を受ける行ごとに 1 回実行するかを指定します。

トリガー・アクション は、オプションの検索条件、およびトリガーが起動されると必ず実行されるステートメントのセットで構成されます。ステートメントが実行されるのは、検索条件が true と評価された場合だけです。トリガー起動時間がトリガー・イベントの前の場合、トリガー・アクションに、SELECT ステートメント、遷移変数を設定するステートメント、および SQL 状態をシグナル通知するステートメントを組み入れることができます。トリガー起動時間がトリガー・イベントの後の場合、トリガー・アクションに、SELECT、 INSERT 、UPDATE、DELETE ステートメント、または SQL 状態をシグナル通知するステートメントを組み入れることができます。

トリガー・アクションでは、*遷移変数* を使用して、影響を受ける行のセット内の値を参照できます。遷移変数は、サブジェクト表の列の名前を使用します。この名前は、参照するのが古い値 (更新前) か新しい値 (更新後) かを識別するために指定さ

れた名前によって修飾されます。 BEFORE、INSERT、または UPDATE トリガーで、 SET Variable ステートメントを使用して新しい値を変更することもできます。

影響を受ける行のセット内の値を参照する別の方法は、*遷移表* を使用することです。遷移表では、サブジェクト表の列の名前も使用しますが、名前を指定することにより、影響を受ける行の完全なセットを表として扱うことができます。遷移表は、AFTER トリガーでしか使用できません (つまり、BEFORE および INSTEAD OF トリガーでは使用できません)。また、古い値と新しい値に別々の遷移表を定義することができます。

表、イベント (INSERT、UPDATE、DELETE、INSTEAD OF)、または起動時間 (BEFORE、AFTER) の組み合わせに対して複数のトリガーを指定することができます。特定の表、イベント、および起動時間に対して複数のトリガーが存在する場合、トリガーが活動化される順序は、作成された順序と同じです。そのため、一番あとに作成されたトリガーが、最後に活動化されます。

トリガーの活動化では、*トリガー・カスケード* が行われる場合があります。これは、ステートメントを実行するあるトリガーを活動化することにより、そのステートメントによって、他のトリガーが活動化されるか、または同じトリガーが再度活動化された結果です。トリガー・アクションによって、削除の参照整合性規則の適用の結果である更新が行われることもあります。これにより、今度は、追加トリガーの活動化が行われる場合があります。トリガー・カスケードでは、トリガーおよび参照整合性の削除規則のチェーンが活動化され、単一の INSERT、UPDATE、または DELETE ステートメントの結果として、データベースへの大幅な変更が行われる場合があります。

複数のトリガーが同じオブジェクトに対する挿入、更新、または削除操作を行う場合、アクセスの競合を解決するために一時表などの競合解決機構が使用されます。これは、パーティション・データベース環境では特に、パフォーマンスに大きな影響を与えることがあります。

トリガーのタイプ

トリガー は、指定した表に対する挿入、更新、または削除操作への応答として実行されるアクションのセットを定義します。このような SQL 操作が実行されるとき、トリガーが*起動*されるといいます。トリガーはオプションであり、 CREATE TRIGGER ステートメントを使用して定義されます。

データ整合性規則を実施するために、参照制約およびチェック制約とともにトリガーを使用できます。また、トリガーを使用して、他の表への更新を行ったり、挿入または更新される行の値を自動的に生成またはトランスフォームできます。あるいは、関数を呼び出してタスク (アラートを発するなど) を実行することもできます。

以下のタイプのトリガーがサポートされています。

BEFORE トリガー
更新、または挿入の前に実行します。更新または挿入中の値を、データベースが実際に修正される前に修正することができます。更新または挿入の前に実行するトリガーは、以下のいくつかの用途に使用できます。

- データベース内で実際に値の更新または挿入を行う前に、値のチェックや変更を行う。これは、画面上でユーザーが見ているものから何らかの内部データベース形式に、データを変換しなければならない場合に便利です。
- ユーザー定義関数でコーディングされている、他の非データベース操作を実行する。

BEFORE DELETE トリガー
削除の前に実行します。値を検査します (必要に応じてエラーを通知します)。

AFTER トリガー
更新、挿入、削除の後に実行します。更新または挿入の後に実行するトリガーは、以下のいくつかの用途に使用できます。

- 他の表のデータを更新する。この機能は、データ間のリレーションシップを保守したり、監査証跡情報を維持したりする際に便利です。
- その表または他の表内にある、他のデータに対してチェックを行う。この機能は、参照整合性制約が適切でない場合、または表チェック制約によってチェックが現在の表だけに制限されている場合に、データ整合性を確保するのに役立ちます。
- ユーザー定義関数でコーディングされている、非データベース操作を実行する。この機能は、アラートを出す場合や、そのデータベース以外の場所にある情報を更新する場合に役立ちます。

INSTEAD OF トリガー
ビューが複雑であるため、挿入、更新、および削除操作を固有にサポートできない場合に、それらのビューに対して操作を実行する方法を記述します。このトリガーにより、アプリケーションはすべての SQL 操作 (挿入、削除、更新、および選択) のための単独インターフェースとしてビューを使用できます。

BEFORE トリガー

更新または挿入の前に実行されるトリガーを使用すると、更新中または挿入中の値を、データベースが実際に修正される前に修正することができます。これは、アプリケーションからの入力 (ユーザーから見たデータ) を希望の内部データベース形式に変換するために使用できます。

また、ユーザー定義の関数により他の非データベース操作を活動化する際にも、この BEFORE トリガーを使用することができます。

BEFORE DELETE トリガーは、削除操作の前に実行します。このトリガーによって値が検査され、必要な場合はエラーが返されます。

例

次の例では、複雑なデフォルトで DELETE TRIGGER が定義されています。

```
CREATE TRIGGER trigger1
  BEFORE UPDATE ON table1
  REFERENCING NEW AS N
  WHEN (N.expected_delivery_date IS NULL)
  SET N.expected_delivery_date = N.order_date + 5 days;
```

次の例では、参照保全制約ではない表間制約で DELETE TRIGGER が定義されています。

```
CREATE TRIGGER trigger2
  BEFORE UPDATE ON table2
  REFERENCING NEW AS N
  WHEN (n.salary > (SELECT maxsalary FROM salaryguide WHERE rank = n.position))
  SIGNAL SQLSTATE '78000' SET MESSAGE_TEXT = 'Salary out of range');
```

AFTER トリガー

更新、挿入、削除の後に実行されるトリガーにはいくつかの使用法があります。

- トリガーにより、同じ表または別の表にあるデータを更新、挿入、削除できます。これは、データ間リレーションシップの保守、または監査証跡情報の維持に役立ちます。
- トリガーにより、表の中の残りのデータ、または別の表のデータの値をチェックできます。これは、表の中の他の行または他の表からデータが参照されているために参照整合性制約やチェック制約が使用できない場合に役立ちます。
- トリガーによりユーザー定義関数を使用して、非データベース操作を開始できます。これは、アラート発行やデータベース外部の情報の更新に役立ちます。

例

次の例は、新しい従業員が雇用されたときに、従業員数を増やす AFTER トリガーを表します。

```
CREATE TRIGGER NEW_HIRE
  AFTER INSERT ON EMPLOYEE
  FOR EACH ROW
  UPDATE COMPANY_STATS SET NBEMP = NBEMP + 1
```

INSTEAD OF トリガー

INSTEAD OF トリガーは、複雑なビューに対して挿入、更新、および削除操作を実行する方法を記述します。 INSTEAD OF トリガーにより、アプリケーションはすべての SQL 操作 (挿入、削除、更新、および選択) のための単独インターフェースとしてビューを使用できます。

通常、INSTEAD OF トリガーはビュー本体で適用されている論理の逆のものを含んでいます。例えば、列のソース表から列を復号するビューについて考えます。このビューの INSTEAD OF トリガーはデータを暗号化し、それをソース表に挿入します。こうして、対称操作を実行します。

INSTEAD OF トリガーを使用すると、指定に対して要求された変更操作がトリガー論理で置き換えられ、ビューに代わって操作を実行します。アプリケーション側から見ると、これは透過的に行われるため、すべての操作がビューに対して実行されているように見えます。対象となる特定のビューに対して実行される操作ごとに、1 つの INSTEAD OF トリガーだけが許可されます。

ビュー自体は非型付きビューまたは非型付きビューに対する別名でなければなりません。さらに、WITH CHECK OPTION (対称ビュー) を使用して定義されたビューや、対称ビューが直接的または間接的に定義されているビューであってはなりません。

例

以下の例は、定義されたビュー (EMPV) に対して INSERT、UPDATE、および DELETE のための論理を提供する 3 つの INSTEAD OF トリガーを示します。ビュー EMPV にはその from 節からの結合が含まれているため、変更操作を固有にサポートすることはできません。

```
CREATE VIEW EMPV(EMPNO, FIRSTNME, MIDINIT, LASTNAME, PHONENO,
                 HIREDATE, DEPTNAME)
AS SELECT EMPNO, FIRSTNME, MIDINIT, LASTNAME, PHONENO,
          HIREDATE, DEPTNAME
          FROM EMPLOYEE, DEPARTMENT WHERE
               EMPLOYEE.WORKDEPT = DEPARTMENT.DEPTNO

CREATE TRIGGER EMPV_INSERT INSTEAD OF INSERT ON EMPV
REFERENCING NEW AS NEWEMP FOR EACH ROW
INSERT INTO EMPLOYEE (EMPNO, FIRSTNME, MIDINIT, LASTNAME,
                      WORKDEPT, PHONENO, HIREDATE)
        VALUES(EMPNO, FIRSTNME, MIDINIT, LASTNAME,
               COALESCE((SELECT DEPTNO FROM DEPARTMENT AS D
                         WHERE D.DEPTNAME = NEWEMP.DEPTNAME),
                        RAISE_ERROR('70001', 'Unknown dept name')),
               PHONENO, HIREDATE)

CREATE TRIGGER EMPV_UPDATE INSTEAD OF UPDATE ON EMPV
REFERENCING NEW AS NEWEMP OLD AS OLDEMP
  FOR EACH ROW
BEGIN ATOMIC
 VALUES(CASE WHEN NEWEMP.EMPNO = OLDEMP.EMPNO THEN 0
             ELSE RAISE_ERROR('70002', 'Must not change EMPNO') END);
 UPDATE EMPLOYEE AS E
   SET (FIRSTNME, MIDINIT, LASTNAME, WORKDEPT, PHONENO, HIREDATE)
     = (NEWEMP.FIRSTNME, NEWEMP.MIDINIT, NEWEMP.LASTNAME,
               COALESCE((SELECT DEPTNO FROM DEPARTMENT AS D
                         WHERE D.DEPTNAME = NEWEMP.DEPTNAME),
                        RAISE_ERROR ('70001', 'Unknown dept name')),
               NEWEMP.PHONENO, NEWEMP.HIREDATE)
 WHERE NEWEMP.EMPNO = E.EMPNO;
END

CREATE TRIGGER EMPV_DELETE INSTEAD OF DELETE ON EMPV
REFERENCING OLD AS OLDEMP FOR EACH ROW
DELETE FROM EMPLOYEE AS E WHERE E.EMPNO = OLDEMP.EMPNO
```

トリガーの設計

トリガーを作成したなら、それを表と関連付ける必要があります。INSTEAD OF トリガーを作成した場合は、それをビューと関連付ける必要があります。この表またはビューは、トリガーのターゲット表 と呼ばれます。変更操作という用語は、ターゲット表の状態に対する何らかの変更を意味します。

変更操作 は、以下のいずれかによって開始されます。

- INSERT ステートメント
- UPDATE ステートメント、または UPDATE を実行する参照制約
- DELETE ステートメント、または DELETE を実行する参照制約
- MERGE ステートメント

各トリガーをこれら 3 つのタイプの変更操作の 1 つと関連付ける必要があります。この関連付けは、その特定のトリガーの*トリガー・イベント* と呼ばれます。

さらに、トリガー・イベントが発生する際にトリガーによって実行されるトリガー・アクション というアクションを定義する必要もあります。トリガー・アクションは 1 つ以上のステートメントから構成され、データベース・マネージャーがトリガー・イベントを実行する前または後に実行されます。トリガー・イベントが発生すると、データベース・マネージャーはサブジェクト表の中から変更操作の影響される行のセットを判別して、トリガーを実行します。

トリガーを作成する際のガイドライン:

トリガーを作成する際には、以下のような属性および振る舞いを宣言する必要があります。

- トリガーの名前
- 対象表の名前
- トリガー起動時間 (変更操作実行の BEFORE または AFTER)
- トリガー・イベント (INSERT、DELETE、または UPDATE)
- 以前の遷移変数の値 (存在する場合)
- 新しい遷移変数の値 (存在する場合)
- 以前の遷移表の値 (存在する場合)
- 新しい遷移表の値 (存在する場合)
- 細分性 (FOR EACH STATEMENT または FOR EACH ROW)
- トリガーのトリガー・アクション (トリガー・アクション条件とトリガー・ステートメントを含む)
- トリガー・イベントがトリガー列リストに対する UPDATE の場合、UPDATE ステートメントで特定の列が指定された場合にのみにトリガーを起動するかどうか

複数のトリガーの設計:

CREATE TRIGGER ステートメントを使用してトリガーを定義すると、この作成時間はデータベース内にタイム・スタンプの形式で登録されます。このタイム・スタンプの値は、同時に実行すべきトリガーが複数存在した際に、トリガーの活動化を順序付けするために引き続き使用されます。例えばタイム・スタンプは、同一対象表に同じイベントと同じ活動化時間を持つトリガーが複数存在する場合に使用されます。また、トリガー・アクションによって直接的または間接的に (これは、他の参照制約によって、反復的に行われることを意味します) 発生したトリガー・イベントおよび参照制約アクションによって活動化された、 1 つ以上の AFTER または INSTEAD OF トリガーが存在する場合にも使用されます。

次の 2 つのトリガーを考えてください。

```
CREATE TRIGGER NEW_HIRED
  AFTER INSERT ON EMPLOYEE
  FOR EACH ROW
  BEGIN ATOMIC
    UPDATE COMPANY_STATS
    SET NBEMP = NBEMP + 1;
  END

CREATE TRIGGER NEW_HIRED_DEPT
   AFTER INSERT ON EMPLOYEE
   REFERENCING NEW AS EMP
   FOR EACH ROW
     BEGIN ATOMIC
```

```
  UPDATE DEPTS
  SET NBEMP = NBEMP + 1
  WHERE DEPT_ID = EMP.DEPT_ID;
END
```

上記のトリガーは、employee 表で INSERT 操作を実行すると活動化されます。この場合、トリガー作成のタイム・スタンプは、上の 2 つのトリガーのうちどちらが最初に活動化されるかを定義します。

トリガーの活動化は、タイム・スタンプ値の昇順で処理されます。したがって、データベースに新しく追加されたトリガーは、事前に定義されている他のすべてのトリガーの後で実行されます。

旧トリガーは新規トリガーの前に活動化され、新規トリガーがデータベースに影響を及ぼす変更に対して*増分の* 加算として使用できるようにします。例えば、トリガー T1 によって起動されるステートメントが新しい行を表 T に挿入したとします。その後、T1 の後に実行されるトリガー T2 によって起動されるステートメントを使用して、T 中の同じ行を特定の値で更新できます。トリガーの起動順は予測可能なため、1 つの表に複数のトリガーを設けても、古いトリガーによって既に変更されている表に対して新しいトリガーが作用することがわかります。

参照制約とのトリガーの対話:

トリガー・イベントは参照制約が原因で変更されることがあります。例えば、DEPT と EMP という 2 つの表があるとすれば、 DEPT を削除および更新すると参照整合性制約により EMP も伝搬して削除および更新され、EMP で定義された削除および更新トリガーは、DEPT で定義された参照制約の結果として活動化されます。 EMP でのトリガーはその活動化時間にしたがって、 EMP 内の行の削除 (ON DELETE CASCADE の場合) または更新 (ON DELETE SET NULL の場合) の前 (BEFORE) か後 (AFTER) に実行されます。

トリガー起動条件の指定 (起動させるステートメントまたはイベント)

トリガーはどれもあるイベントと関連しています。トリガーは、それに対応するイベントがデータベースで発生すると活動化されます。このトリガー・イベントは、指定されたアクション、すなわち UPDATE、INSERT、または DELETE ステートメント (参照制約のアクションにより発生する操作を含む) がターゲット表に対して実行される際に発生します。

例:

```
CREATE TRIGGER NEW_HIRE
  AFTER INSERT ON EMPLOYEE
  FOR EACH ROW
  UPDATE COMPANY_STATS SET NBEMP = NBEMP + 1
```

上のステートメントは、挿入操作が表 employee で行われる際に活動化されるトリガー new_hire を定義します。

どのトリガー・イベントも (したがってどのトリガーも)、 1 つだけのターゲット表と 1 つだけの変更操作に関連付けることができます。以下のような変更操作があります。

挿入操作

挿入操作は INSERT ステートメントまたは MERGE ステートメントの挿入操作によってのみ行われます。したがって、LOAD コマンドなどの INSERT を使用しないユーティリティーを用いてデータをロードすると、トリガーは活動化されません。

削除操作

削除操作は、DELETE ステートメントか、MERGE ステートメントの削除操作か、ON DELETE CASCADE の参照制約規則の結果として実行されます。

更新操作

更新操作は、UPDATE ステートメントか、MERGE ステートメントの更新操作か、ON DELETE SET NULL の参照制約規則の結果として実行されます。

トリガー・イベントが更新操作である場合、そのイベントはターゲット表の特定の列と関連させることができます。この場合のトリガーは、更新操作が特定の列のどれかを更新しようとする場合に限り活動化されます。これにより、トリガーを活動化するイベントをさらに細分化することができます。

例えば、次のトリガー REORDER は、更新操作を表 PARTS の列 ON_HAND か MAX_STOCKED で実行する場合に限り活動化します。

```
CREATE TRIGGER REORDER
  AFTER UPDATE OF ON_HAND, MAX_STOCKED ON PARTS
  REFERENCING NEW AS N_ROW
  FOR EACH ROW
  WHEN (N_ROW.ON_HAND < 0.10 * N_ROW.MAX_STOCKED)
  BEGIN ATOMIC
  VALUES(ISSUE_SHIP_REQUEST(N_ROW.MAX_STOCKED -
                            N_ROW.ON_HAND,
                            N_ROW.PARTNO));
  END
```

トリガーは、活動化されると次のような細分性のレベルに従って実行されます。

FOR EACH ROW

影響される行の数と同じ回数だけ実行されます。トリガー・アクションに影響される特定の行を参照する必要がある場合、 FOR EACH ROW 細分性を使用します。この例として、AFTER UPDATE トリガーで更新行の新しい値と古い値を比較することが挙げられます。

FOR EACH STATEMENT

トリガー・イベントに対して一度だけ実行されます。

影響を受ける行のセットが空の場合 (すなわち、WHERE 節が行を限定しなかった検索済み UPDATE または DELETE の場合)、FOR EACH ROW トリガーは実行されません。ただし、FOR EACH STATEMENT トリガーはやはり一度実行されます。

例えば、FOR EACH ROW を使用して従業員数の計算を維持することができます。

```
CREATE TRIGGER NEW_HIRED
  AFTER INSERT ON EMPLOYEE
  FOR EACH ROW
  UPDATE COMPANY_STATS SET NBEMP = NBEMP + 1
```

FOR EACH STATEMENT の細分性を使用して更新を行っても同じ結果が得られます。

```
CREATE TRIGGER NEW_HIRED
  AFTER INSERT ON EMPLOYEE
  REFERENCING NEW_TABLE AS NEWEMPS
  FOR EACH STATEMENT
  UPDATE COMPANY_STATS
  SET NBEMP = NBEMP + (SELECT COUNT(*) FROM NEWEMPS)
```

注:

- FOR EACH STATEMENT の細分性は、 BEFORE トリガーにはサポートされません。
- トリガーの最大ネスト・レベルは 16 です。つまり、トリガー起動をカスケードする最大数が 16 となります。トリガー起動とは、トリガー・イベント時 (表の列内または広くは表へのデータの挿入、更新、または削除など) のトリガーの起動のことです。

トリガー起動のタイミングの指定 (BEFORE 節、AFTER 節、および INSTEAD OF 節)

トリガー起動時間 は、トリガー・イベントから見て、いつトリガーを活動化するかを指定します。

指定できる活動化のタイミングは、以下に示すように、
BEFORE、AFTER、INSTEAD OF の 3 つです。

- 活動化時間が BEFORE の場合、トリガー・アクションは、トリガー・イベントが実行される前に影響される行のそれぞれに対して活動化されます。そのため、BEFORE トリガーがそれぞれの行に対する実行を完了した後にのみ、サブジェクト表は変更されます。 BEFORE トリガーは、FOR EACH ROW の細分性を持たなければならないことに注意してください。
- 活動化時間が AFTER の場合、トリガー・アクションは、影響される行のそれぞれに対して、またはステートメントに対して、トリガーの細分性に従って活動化されます。これはトリガー・イベントが完了した後、またトリガー・イベントによって影響される可能性がある制約 (参照制約のアクションも含む) すべてをデータベース・マネージャーがチェックした後で起きます。 AFTER トリガーは、FOR EACH ROW か FOR EACH STATEMENT のどちらかの細分性を持つことができます。

  例えば、次のトリガーの活動化時間は employee での INSERT 操作の後 (AFTER) です。

  ```
  CREATE TRIGGER NEW_HIRE
    AFTER INSERT ON EMPLOYEE
    FOR EACH ROW
    UPDATE COMPANY_STATS SET NBEMP = NBEMP + 1
  ```

- 活動化時間が INSTEAD OF の場合、トリガー・アクションは、トリガー・イベントを実行する代わりに、影響される行のセットにある行のそれぞれに対して活動化されます。 INSTEAD OF トリガーは、FOR EACH ROW の細分性を持たなければなりません。また、サブジェクト表はビューでなければなりません。その他のトリガーはサブジェクト表としてビューを使用することはできません。

以下の図に、BEFORE トリガーと AFTER トリガーの実行モデルを示します。

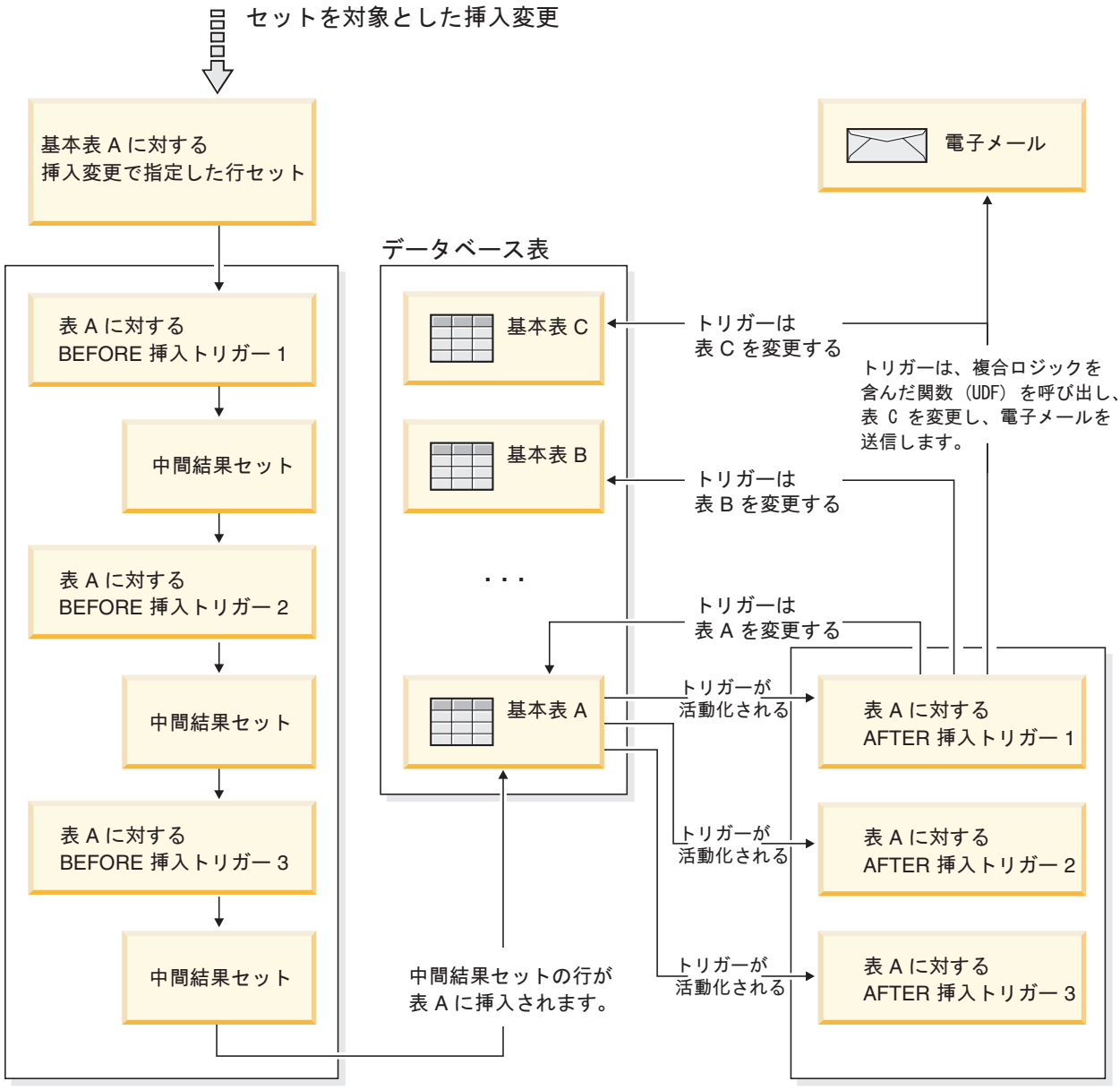


図 26. トリガー実行モデル

BEFORE トリガーと AFTER トリガーの両方を持つ特定の表と、これらのトリガーに関連した変更イベントについては、まずすべての BEFORE トリガーが活動化されます。そのイベントについて最初に活動化された BEFORE トリガーは、操作のターゲットになっている行セットに対して、ロジックで指定されているすべての更新変更を実行します。その BEFORE トリガーの出力を次の BEFORE トリガーが入力として受け取ります。そのイベントによって活動化されたすべての BEFORE トリガーが起動されると、中間結果セット、つまり、トリガー・イベント操作のターゲットになっている行に対する BEFORE トリガーの変更の結果が表に適用されます。その後、イベントに関連した各 AFTER トリガーが起動されます。 AFTER トリガーでは、同じ表を変更することも、別の表を変更することも、データベースの外部のアクションを実行することもできます。

トリガーの活動化時間が異なると、トリガーの目的も異なります。根本的に、BEFORE トリガーはデータベース管理システムの制約付きサブシステムに対する拡張です。したがって、通常は次のような目的で使用します。

- 入力データの検査を行う
- 新しく挿入された行に対して値を自動的に生成する
- 相互参照のために他の表から読み込む

BEFORE トリガーはトリガー・イベントがデータベースに適用される前に活動化されるので、これを使用してデータベースをさらに修正することはできません。そのため、BEFORE トリガーの活動化は、整合性に関する制約をチェックした後に行われます。

逆に、AFTER トリガーは、特殊なイベントが起こるたびにデータベースで実行されるアプリケーション・ロジックのモジュールと見なすことができます。 AFTER トリガーは、アプリケーションの一部として常に一定の状態でデータベースを参照します。また、整合性に関する制約が検証された後に実行されます。したがって、このトリガーは主にアプリケーションでも実行できる操作を行うために使用できます。例:

- データベース内であとに続く変更操作を行う。
- データベースの外側で、警告のサポートなどのアクションを行う。トリガーがロールバックされても、データベースの外側で行われるアクションはロールバックされないことに注意してください。

対照的に、INSTEAD OF トリガーは、それが定義されているビューの逆の操作の記述と考えることができます。例えば、ビュー内の選択リストに表に関する式が含まれる場合、 INSTEAD OF INSERT トリガーの本体の INSERT ステートメントには逆の式が含まれます。

BEFORE、AFTER、および INSTEAD OF トリガーにはさまざまな特質のものがあるため、それら BEFORE、AFTER、および INSTEAD OF の各トリガーで起動されるアクションを定義するために SQL 操作のさまざまなセットを使用できます。例えば、更新操作は BEFORE トリガーでは実行できません。これは、トリガー・アクションにおいて整合性に関する制約が違反されないという保証がないためです。同様に、BEFORE、AFTER、および INSTEAD OF トリガーでは、さまざまなトリガーの細分性がサポートされています。

すべてのトリガーのトリガー SQL ステートメントは、動的コンパウンド・ステートメントとすることができます。しかし、BEFORE トリガーにはいくつかの制約事項があります。このトリガーには以下の SQL ステートメントを含めることはできません。

- UPDATE
- DELETE
- INSERT
- MERGE

トリガー・アクション起動の条件定義 (WHEN 節)

トリガーを活動化すると、それに関連するトリガー・アクションが実行されます。すべてのトリガーには、次のような 2 つのコンポーネントを順々に持つトリガー・アクションが 1 つだけあります。その 2 つのコンポーネントとは、オプションの *トリガー・アクション条件* または WHEN 節、および *トリガー・ステートメント (複数可)* のセットです。

トリガー・アクション条件 は、検索条件を指定するトリガー・アクションの任意指定の節です。トリガー・アクション内でステートメントを実行するためには、検索条件は真と評価されなければなりません。 WHEN 節を省略すると、トリガー・アクション内のステートメントは常に実行されます。

トリガー・アクション条件は、FOR EACH ROW トリガーの場合にはそれぞれの行に対して一度評価され、 FOR EACH STATEMENT トリガーの場合にはステートメントに対して一度評価されます。

さらに WHEN 節は、トリガーに代わって活動化されるアクションを正しく調整するために使用できるように制御されます。例えばこの節は、入ってくる値がある一定の範囲の内側か外側になる場合だけトリガー・アクションが活動化されるという設定をすることで、データ従属の規則を強制するような場合に役立ちます。

トリガーを活動化すると、それに関連するトリガー・アクションが実行されます。すべてのトリガーには、次のような 2 つのコンポーネントを順々に持つトリガー・アクションが 1 つだけあります。

トリガー・アクション条件は、トリガー・アクションが実行中の行やステートメントに対してトリガー・ステートメントのセットが実行されるかどうかを定義します。トリガー・ステートメントのセットは、トリガー・イベントが発生した結果としてトリガーによりデータベースで実行される一連のアクションを定義します。

例えば以下のトリガー・アクションは、 on_hand 列の値が max_stocked 列の値の 10 % より小さい行に対してのみ、トリガー・ステートメントのセットが活動化されることを指定します。この場合、トリガー・ステートメントのセットが issue_ship_request 関数を呼び出します。

```
CREATE TRIGGER REORDER
  AFTER UPDATE OF ON_HAND, MAX_STOCKED ON PARTS
  REFERENCING NEW AS N_ROW
  FOR EACH ROW

  WHEN (N_ROW.ON_HAND < 0.10 * N_ROW.MAX_STOCKED)
  BEGIN ATOMIC
    VALUES(ISSUE_SHIP_REQUEST(N_ROW.MAX_STOCKED -
                              N_ROW.ON_HAND,
                              N_ROW.PARTNO));
  END
```

トリガー・ステートメントのセットは、トリガーの活動化により行われる実際のアクションを実行します。すべての SQL 操作がすべてのトリガーに有効となるわけではありません。トリガー起動時間が BEFORE か AFTER かにより、トリガー・ステートメントとして適切な操作の種類が異なる場合があります。

ほとんどの場合、トリガー・ステートメントが負の戻りコードを戻すと、トリガー・ステートメントはトリガーおよび参照制約のすべてのアクションとともにロー

ルバックされます。失敗した、トリガー・ステートメントからは、トリガー名、SQLCODE、SQLSTATE、およびトークンの大部分がエラー・メッセージとして戻されます。

トリガーでサポートされる SQL PL ステートメント

すべてのトリガーのトリガー SQL ステートメントは、動的コンパウンド・ステートメントとすることができます。

すなわち、トリガー SQL ステートメントは、以下のエレメント 1 つ以上を含めることができます。

- CALL ステートメント
- DECLARE 変数ステートメント
- SET 変数ステートメント
- WHILE ループ
- FOR ループ
- IF ステートメント
- SIGNAL ステートメント
- ITERATE ステートメント
- LEAVE ステートメント
- GET DIGNOSTIC ステートメント
- 全選択

ただし、AFTER トリガーと INSTEAD OF トリガーに限っては、以下の SQL ステートメントを 1 つ以上含めることができます。

- UPDATE ステートメント
- DELETE ステートメント
- INSERT ステートメント
- MERGE ステートメント

遷移変数を使用してトリガーの中で古い列値と新しい列値にアクセスする

FOR EACH ROW トリガーをインプリメントする際には、トリガーが実行される一連の影響行の列の値を参照する必要があるかもしれません。データベース内の表 (主題表を含む) の列を参照するには、正規の SELECT ステートメントを使用できることに注意してください。

FOR EACH ROW トリガーは現在実行中の行の列を、 CREATE TRIGGER ステートメントの REFERENCING 節で指定できる 2 つの遷移変数を使用して参照します。遷移変数には、相関名とともに OLD および NEW として指定される 2 種類があります。この分類には次のような意味があります。

OLD AS 相関名
行の元の状態 (つまりトリガー・アクションがデータベースに適用される前の状態) を収集する相関名を指定します。

NEW AS 相関名

トリガー・アクションがデータベースに適用される際に、データベースの行を更新するために使用される (または使用された) 値を収集する相関名を指定します。

次の例を考慮してください。

```
CREATE TRIGGER REORDER
  AFTER UPDATE OF ON_HAND, MAX_STOCKED ON PARTS
  REFERENCING NEW AS N_ROW
  FOR EACH ROW
  WHEN (N_ROW.ON_HAND < 0.10 * N_ROW.MAX_STOCKED
  AND N_ROW.ORDER_PENDING = 'N')
  BEGIN ATOMIC
    VALUES(ISSUE_SHIP_REQUEST(N_ROW.MAX_STOCKED -
                              N_ROW.ON_HAND,
                              N_ROW.PARTNO));
    UPDATE PARTS SET PARTS.ORDER_PENDING = 'Y'
    WHERE PARTS.PARTNO = N_ROW.PARTNO;
  END
```

上で説明した遷移変数の OLD および NEW の定義により、すべての遷移変数がすべてのトリガーに対して定義できるわけではないことが分かります。遷移変数は、次のようなトリガー・イベントの種類に基づいて定義することができます。

UPDATE

UPDATE トリガーは、OLD と NEW の両方の遷移変数を参照できます。

INSERT

INSERT トリガーは、NEW 遷移変数のみを参照できます。これは INSERT 操作の活動化の前に、影響される行がデータベースに存在しないためです。すなわち、トリガー・アクションがデータベースに適用される前の古い値を定義する行の元の状態がありません。

DELETE

DELETE トリガーは OLD 遷移変数のみを参照できます。これは、削除操作で指定された新しい値がないためです。

注: 遷移変数は FOR EACH ROW トリガーに対してのみ指定できます。 FOR EACH STATEMENT トリガーでは遷移変数を参照しても、影響される行のうち遷移変数が参照中の行を指定することはできません。代わりに、CREATE TRIGGER ステートメントの OLD TABLE および NEW TABLE 節を使用して、新規行と旧行のセットを参照してください。これらの節について詳しくは、CREATE TRIGGER ステートメントを参照してください。

遷移表を使用した古い表と新しい表の結果セットの参照

FOR EACH ROW と FOR EACH STATEMENT の両方のトリガーでは、影響される行の全セットを参照しなければならないことがあります。これは例えば、トリガー本体が影響される行のセットに集約 (何らかの列値の MAX、MIN、または AVG など) を適用する必要がある場合に当てはまります。

トリガーは、影響される行のセットを CREATE TRIGGER ステートメントの REFERENCING 節で指定できる 2 つの遷移表を使用して参照します。遷移表には、遷移変数のように OLD_TABLE および NEW_TABLE として表名とともに指定される 2 種類があります。この 2 種類の遷移表の意味は次のとおりです。

OLD_TABLE AS 表名

影響される行のセットの元の状態 (トリガー SQL 操作がデータベースに適用される前の状態) を収集する表の名前を指定します。

NEW_TABLE AS 表名

トリガー・アクションがデータベースに適用される際に、データベースの行を更新するために使用される値を収集する表の名前を指定します。

例:

```
CREATE TRIGGER REORDER
  AFTER UPDATE OF ON_HAND, MAX_STOCKED ON PARTS
  REFERENCING NEW_TABLE AS N_TABLE
  NEW AS N_ROW
  FOR EACH ROW
  WHEN ((SELECT AVG (ON_HAND) FROM N_TABLE) > 35)
  BEGIN ATOMIC
    VALUES(INFORM_SUPERVISOR(N_ROW.PARTNO,
                             N_ROW.MAX_STOCKED,
                             N_ROW.ON_HAND));
  END
```

NEW_TABLE は、FOR EACH ROW トリガーにおいても、更新された行の全セットを常に持つことに注意してください。トリガーはそれが定義される表上で実行されると、 NEW_TABLE にはそのトリガーを活動化したステートメントから変更された行が入ります。ただし、トリガー内のステートメントによって変更された行は入りません。これはトリガーの活動化を分離させるためです。

遷移表は読み取り専用です。遷移表は、トリガー・イベントに対して定義できる遷移変数の種類を定義するのと同じ、次のような規則で定義できます。

UPDATE

UPDATE トリガーは、 OLD_TABLE と NEW_TABLE の両方の遷移表を参照できます。

INSERT

INSERT トリガーは、NEW_TABLE 遷移表のみを参照できます。これは、INSERT 操作の活動化の前に、影響される行がデータベースに存在しないためです。すなわち、トリガー・アクションがデータベースに適用される前の古い値を定義している行の元の状態がありません。

DELETE

DELETE トリガーは、OLD_TABLE 遷移表のみを参照できます。これは、削除操作で指定された新しい値がないためです。

注: 遷移表が AFTER トリガーの FOR EACH ROW と FOR EACH STATEMENT の両方の細分性に対して指定できることは知っておく必要があります。

OLD_TABLE と NEW_TABLE の表名の有効範囲はトリガー本体です。この有効範囲では、表名はスキーマ内にある同一の非修飾*表名* を持つ他のすべての表の名前より優先されます。したがって、例えば OLD_TABLE や NEW_TABLE の表名が X の場合、SELECT ステートメントの FROM 節で X (すなわち非修飾の X) を参照することで、トリガー作成者のスキーマ内に X という名の表があったとしても遷移表を常に参照します。この場合、ユーザーはスキーマ内の表 X を参照するために完全修飾名を使わなければなりません。

トリガーの作成

トリガーは、指定した表または型付き表に対する INSERT、UPDATE、DELETE 節と一緒に実行される (つまり、それらの節によって起動される) 一連のアクションを定義します。

トリガーは、以下のような場合に使用します。

- 入力データの妥当性検査
- 新しく挿入された行の値を生成する
- 相互参照のために他の表から読み込む
- 監査履歴のために他の表に書き込む

トリガーを使えば、一般的な保全規則や業務規則をサポートできます。例えば、トリガーによって、注文に応じる前に顧客のクレジット限度を調べたり、サマリー・データ表を更新したりできます。

利点:

- アプリケーション開発がより速くなります。トリガーはデータベースの中に保管されるため、実行されるアクションをアプリケーションごとにコーディングする必要がありません。
- 保守が簡単: 一度トリガーを定義すると、トリガーを作成した表にアクセスするたびに、そのトリガーが自動的に呼び出されます。
- 業務規則がグローバルに適用されます。業務方針が変わった場合、各アプリケーション・プログラムを変更しなくても、トリガーを変更するだけで済みます。

制約事項:

- トリガーにニックネームを使用することはできません。
- BEFORE トリガーの場合は、 ID 列以外の生成列の列名を、トリガー・アクションによって指定することはできません。したがって、生成される識別値は BEFORE トリガーに認識されます。

アトミック・トリガーを作成する際には、ステートメント終了文字に注意する必要があります。コマンド行プロセッサーは、デフォルトではステートメント終了マーカーを「;」と見なします。「;」以外の文字を使用するには、スクリプト内でステートメント終了文字を手動で編集して、アトミック・トリガーを作成しなければなりません。例えば、「;」を「#」などの別の特殊文字で置き換えます。また、CREATE TRIGGER DDL の前に次のコードを付けることもできます。

```
--#SET TERMINATOR @
```

CLP の終止符を即時に変更するには、次の構文を使用すると元の設定に戻ります。

```
--#SET TERMINATOR
```

コマンド行からトリガーを作成するには、以下のように入力します。

```
db2 -td <delimiter> -vf <script>
```

ここで <delimiter> は代わりに使用するステートメント終了文字で、<script> は新しい <delimiter> を終了文字として記述したスクリプトになります。

コマンド行からトリガーを作成するには、以下のように入力します。

```
CREATE TRIGGER <name>
   <action> ON <table_name>
   <operation>
   <triggered_action>
```

次のステートメントは、新人が採用されるたびに従業員の数を増やすトリガーを作成します。これによって、EMPLOYEE 表に行が追加されるたびに、COMPANY_STATS 表の従業員数 (NBEMP) 列に 1 が加算されます。

```
CREATE TRIGGER NEW_HIRED
   AFTER INSERT ON EMPLOYEE
   FOR EACH ROW
   UPDATE COMPANY_STATS SET NBEMP = NBEMP+1;
```

トリガー本体には、INSERT、検索条件つきの UPDATE、検索条件つきの DELETE、全選択、SET 変数、および SIGNAL SQLSTATE のうちの 1 つ以上のステートメントを含めることができます。トリガーは、それが参照する INSERT、UPDATE、または DELETE ステートメントの前または後に起動できます。

トリガーの変更およびドロップ

トリガーは変更できません。まずドロップしてから、義務付けられた新規定義に従って作成し直す必要があります。

トリガーの従属関係

- 他のオブジェクトに対するトリガーの従属関係は、すべて SYSCAT.TRIGDEP システム・カタログ・ビューに記録されます。トリガーはさまざまなオブジェクトに従属する可能性があります。
- トリガーの従属先のオブジェクトをドロップすると、トリガーは機能しなくなりますが、その定義はシステム・カタログ・ビュー内に残ります。そのトリガーを再び有効にするには、システム・カタログ・ビューからその定義を取り出し、新しい CREATE TRIGGER ステートメントをサブミットしてください。
- トリガーをドロップすると、その説明は SYSCAT.TRIGGERS システム・カタログ・ビューから削除され、その従属関係もすべて SYSCAT.TRIGDEP システム・カタログ・ビューから削除されます。トリガーに UPDATE、INSERT、または DELETE 従属性のあるパッケージは、すべて無効になります。
- ビューがトリガーに従属している場合に、それを作動不能にすると、トリガーにも作動不能のマークが付けられます。作動不能のマークが付けられたトリガーに従属するパッケージがあれば、すべて無効にされます。

DROP TRIGGER ステートメントを使用すれば、トリガー・オブジェクトをドロップできますが、この手順では、以下のように従属パッケージに無効のマークが付けられます。

- 明示列リストなしの更新トリガーがドロップされると、ターゲット表上での更新使用を持つパッケージは無効にされます。

- 列リストつきの更新トリガーがドロップされると、ターゲット表上での更新使用を持つパッケージは、そのパッケージが CREATE TRIGGER ステートメントの column-name リストの中の少なくとも 1 つの列上での更新使用も持っている場合にのみ無効にされます。
- 挿入トリガーがドロップされると、ターゲット表上での挿入使用を持つパッケージが無効にされます。
- 削除トリガーがドロップされると、ターゲット表上での削除使用を持つパッケージが無効にされます。

パッケージは、アプリケーション・プログラムが明示的にバインドまたは再バインドされるか、あるいは、そのパッケージが実行され、データベース・マネージャーが自動的に再バインドを行うまで、無効のままとなります。

トリガーおよびトリガー使用の例

トリガーと参照制約の相互作用の例

更新操作を行うと、トリガーと参照制約およびチェック制約の相互作用が発生することがあります。

316 ページの図 23 とその後の説明は、データベースのデータを更新するステートメントに対して行われる典型的な処理を示しています。

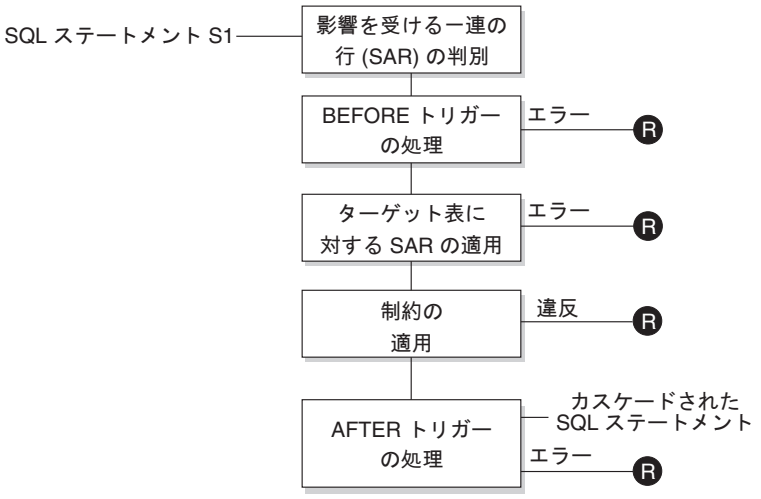


図 27. 関連するトリガーと制約を伴うステートメントの処理

316 ページの図 23 は、表を更新するステートメントの一般的な処理の順序を示しています。ここでは、BEFORE トリガー、参照制約、チェック制約、および AFTER トリガーがカスケードしている表を想定しています。 316 ページの図 23 に示されているボックスやその他の項目について、以下に説明します。

- ステートメント S_1

これは、プロセスを開始する DELETE、INSERT、または UPDATE ステートメントです。ステートメント S_1 は、この説明においてサブジェクト表 と呼ばれる表 (または表に対する更新可能なビュー) を指定しています。

- 影響を受ける一連の行の判別

  このステップは、CASCADE および SET NULL の参照制約の削除規則と、AFTER トリガーからのカスケード・ステートメントに対して繰り返されるプロセスの開始点です。

  このステップの目的は、そのステートメントで影響を受ける一連の行 を判別することです。含まれる行の集合は、ステートメントに基づいて、以下のようになります。

  – DELETE の場合、ステートメントの検索条件を満たしているすべての行 (位置指定 DELETE の場合は現在行)
  – INSERT の場合、VALUES 節または全選択によって指定される行
  – UPDATE の場合、検索条件を満たしているすべての行 (位置指定 UPDATE の場合は現在行)

  影響を受ける一連の行が空の場合、BEFORE トリガー、サブジェクト表に適用される変更、またはステートメントの処理に対する制約はありません。

- BEFORE トリガーの処理

  BEFORE トリガーの処理はすべて作成の昇順で行われます。各 BEFORE トリガーは、影響を受ける一連の行内の各行ごとに 1 回ずつトリガー・アクションを処理します。

  トリガー・アクションの処理の過程でエラーが生じることがあり、そのような場合には元のステートメント S_1 の結果としての変更内容 (これまでの) がすべてロールバックされます。

  BEFORE トリガーがない場合、または影響を受ける一連の行が空の場合、このステップはスキップされます。

- サブジェクト表への影響を受ける一連の行の適用

  データベース内のサブジェクト表への実際の削除、挿入、または更新は影響を受ける一連の行を使用して適用されます。

  影響を受ける一連の行の適用時にエラーが生じることがあり (ユニーク索引のあるロケーションに重複するキーを持つ行を挿入しようとした場合など)、そのような場合は元のステートメント S_1 の結果としての変更内容 (これまでの) がすべてロールバックされます。

- 制約の適用

  影響を受ける一連の行が空でない場合には、サブジェクト表に関連した制約が適用されます。この制約には、ユニーク制約、ユニーク索引、参照制約、チェック制約、ビューに対する WITH CHECK OPTION に関連した検査などがあります。カスケード削除規則または NULL 設定のある参照制約では、追加のトリガーが活動化されることがあります。

何らかの制約または WITH CHECK OPTION に違反するとエラーが発生し、S_1 の結果として行われた変更 (その時点までの) はロールバックされます。

- AFTER トリガーの処理

  S_1 によって活動化された AFTER トリガーは、すべて作成の昇順に処理されます。

  FOR EACH STATEMENT トリガーでは、影響を受ける一連の行が空の場合にも、1 回だけトリガー・アクションが処理されます。 FOR EACH ROW トリガーでは、影響を受ける一連の行内の各行ごとに 1 回ずつトリガー・アクションが処理されます。

  トリガー・アクションの処理の過程でエラーが生じることがあり、そのような場合は元の S_1 の結果としての変更内容 (これまでの) がすべてロールバックされます。

  トリガーのトリガー・アクションには、トリガーによって実行される DELETE、INSERT、または UPDATE などのステートメントが入っている場合があります。この説明では、そのような各ステートメントは、*カスケードしたステートメント* と見なされます。

  カスケードしたステートメントは、AFTER トリガーのトリガー・アクションの一部として処理される DELETE、INSERT、または UPDATE ステートメントです。そのステートメントによって、カスケード・レベルのトリガー処理が開始されます。これは、新しい S_1 としてトリガー・ステートメントを割り当てて、ここで説明した手順をすべて再帰的に実行することと見なすことができます。

  各 S_1 ごとに起動されるすべての AFTER トリガーによって実行されるすべてのステートメントの処理が完了すると、元の S_1 の処理が完了します。

- R = 変更を S_1 の前までロールバックする操作

  制約違反も含めて、処理中にエラーが発生すると、元のステートメント S_1 の結果として直接または間接になされたすべての変更がロールバックされます。その場合、データベースは、元のステートメント S_1 の実行直前と同じ状態に戻ります。

トリガーを使用したアクションの定義例

総管理者が、72 時間以内に別々の表に 3 つ以上の苦情を送ってきたカスタマーの名前を維持したいとします。また、顧客名がこの表に複数回挿入されたら必ず総管理者に知らせるようにしたいと仮定します。

このようなアクションを定義するには、次のように定義します。

- An UNHAPPY_CUSTOMERS table:

```
CREATE TABLE UNHAPPY_CUSTOMERS (
  NAME           VARCHAR (30),
  EMAIL_ADDRESS  VARCHAR (200),
  INSERTION_DATE DATE)
```

- 3 日以内に 3 つ以上のメッセージを受信した場合に、 UNHAPPY_CUSTOMERS 内に行を自動的に挿入するトリガー (NAME 列と E_MAIL_ADDRESS 列を含む CUSTOMERS 表があることを前提とする)。

```
CREATE TRIGGER STORE_UNHAPPY_CUST
  AFTER INSERT ON ELECTRONIC_MAIL
  REFERENCING NEW AS N
  FOR EACH ROW
  WHEN (3 <= (SELECT COUNT(*)
              FROM ELECTRONIC_MAIL
              WHERE SENDER = N.SENDER
                AND SENDING_DATE(MESSAGE) > CURRENT DATE - 3 DAYS)
       )
  BEGIN ATOMIC
    INSERT INTO UNHAPPY_CUSTOMERS
    VALUES ((SELECT NAME
    FROM CUSTOMERS
    WHERE EMAIL_ADDRESS = N.SENDER), N.SENDER, CURRENT DATE);
  END
```

- 同じカスタマーが複数回 UNHAPPY_CUSTOMERS に挿入された場合に総管理者に通知を送るトリガー (2 文字のストリングを入力とする SEND_NOTE 関数があることを前提とする)。

```
CREATE TRIGGER INFORM_GEN_MGR
  AFTER INSERT ON UNHAPPY_CUSTOMERS
  REFERENCING NEW AS N
  FOR EACH ROW
  WHEN (1 <(SELECT COUNT(*)
            FROM UNHAPPY_CUSTOMERS
            WHERE EMAIL_ADDRESS = N.EMAIL_ADDRESS)
       )
  BEGIN ATOMIC
    VALUES(SEND_NOTE('Check customer:' CONCAT N.NAME,
                     'bigboss@vnet.ibm.com'));
  END
```

トリガーを使用した業務規則の定義例

お客様の苦情を扱う電子メールは、マーケティング管理者の Mr. Nelson にカーボン・コピー (CC) のリストで提出しなければならないという方針が会社にあるとします。

これは規則であるため、制約として表現するほうがよいかもしれません。以下のような方法があります (この場合は、これをチェックする CC_LIST UDF の存在が前提になる)。

```
ALTER TABLE ELECTRONIC_MAIL ADD
  CHECK (SUBJECT <> 'Customer complaint' OR
         CONTAINS (CC_LIST(MESSAGE), 'nelson@vnet.ibm.com') = 1)
```

ただしこの制約により、マーケティング管理者に CC リストで提出しないお客様の苦情を扱う電子メールは挿入できなくなります。このことは、この会社の業務規則の目的ではないことは明らかです。その目的とは、マーケティング管理者にはコピーされていないお客様の苦情を扱う電子メールをマーケティング管理者に転送することです。このような業務規則は、宣言上の制約により表すことのできないアクションを行うことを要求するので、トリガーを使用してのみ表すことができます。トリガーは、E_MAIL タイプのパラメーターと文字ストリングを持つ SEND_NOTE 関数があると仮定します。

```
CREATE TRIGGER INFORM_MANAGER
  AFTER INSERT ON ELECTRONIC_MAIL
  REFERENCING NEW AS N
  FOR EACH ROW
  WHEN (N.SUBJECT = 'Customer complaint' AND
```

```
  CONTAINS (CC_LIST(MESSAGE), 'nelson@vnet.ibm.com') = 0)
BEGIN ATOMIC
  VALUES(SEND_NOTE(N.MESSAGE, 'nelson@vnet.ibm.com'));
END
```

トリガーを使用した表への操作の防止例

配信不能だった E メールが ELECTRONIC_MAIL という名前の表に保管されないようにするとします。そのようにするには、特定の SQL INSERT ステートメントを実行しないようにする必要があります。

それには次の 2 とおりの方法があります。

- 電子メールの件名が *undelivered mail* のときは必ずエラーを返す BEFORE トリガーを定義する。

```
CREATE TRIGGER BLOCK_INSERT
  NO CASCADE BEFORE INSERT ON ELECTRONIC_MAIL
  REFERENCING NEW AS N
  FOR EACH ROW
  WHEN (SUBJECT(N.MESSAGE) = 'undelivered mail')
  BEGIN ATOMIC
    SIGNAL SQLSTATE '85101'
      SET MESSAGE_TEXT = ('Attempt to insert undelivered mail');
    END
```

- 新しい列 SUBJECT の値を *undelivered mail* と異なるものにするチェック制約を定義する。

```
ALTER TABLE ELECTRONIC_MAIL
  ADD CONSTRAINT NO_UNDELIVERED
  CHECK (SUBJECT <> 'undelivered mail')
```

第 15 章 シーケンス

シーケンス とは、伝票番号など値の自動生成を可能にするデータベース・オブジェクトです。シーケンスは、ユニーク・キー値を生成するタスクに最も適しています。アプリケーションはシーケンスを使用して、番号の追跡に使用される列値から発生し得る並行性とパフォーマンス上の問題を回避することができます。データベース外で作成された数値と比べて、シーケンスの利点は、データベース・サーバーが生成された数値を追跡できることにあります。異常終了して再始動しても、重複した番号が生成されることはありません。

生成されるシーケンス番号のプロパティー

- 値は位取りがゼロの数値データ・タイプになります。このようなデータ・タイプは SMALLINT、BIGINT、INTEGER、および DECIMAL です。
- 連続値は、指定した整数増分値によって異なる場合があります。デフォルト増分値は 1 です。
- カウンター値はリカバリー可能です。カウンター値は、リカバリーが要求されたときにログから再構成されます。
- パフォーマンスを上げるため、値をキャッシュに入れることができます。値を事前割り振りしてキャッシュに保管しておくと、シーケンスのために値を生成するとき、ログへの非同期入出力が少なくなります。システム障害が発生した場合、使用されていないキャッシュ値はすべて失われたものとみなされます。CACHE に指定された値は、失われる可能性のあるシーケンス値の最大数です。

シーケンスで使用できる式には、以下の 2 つがあります。

- **NEXT VALUE 式:** 指定されたシーケンスの次の値を返します。NEXT VALUE 式がシーケンスの名前を指定していれば、新しいシーケンス番号が生成されます。ただし、1 つの照会の中で同じシーケンス名を指定している NEXT VALUE 式のインスタンスが複数ある場合、シーケンスのカウンターは結果の行ごとに 1 つだけ増加し、NEXT VALUE のすべてのインスタンスが結果の各行に同じ値を戻します。
- **PREVIOUS VALUE 式:** 現行アプリケーション・プロセス内の直前のステートメントに指定されたシーケンスに対して最後に生成された値を返します。つまり、ある特定の接続では別の接続が NEXT VALUE を呼び出しても、PREVIOUS VALUE は定数のままです。

これらの式の詳細と例については、『』「*SQL リファレンス 第 1 巻*」の『シーケンス参照』を参照してください。

シーケンスの設計

シーケンスを設計する際、ID 列とシーケンスの違い、およびご使用の環境にどちらが適切であるかを考慮する必要があります。シーケンスを使用する決定をする場合、使用可能なオプションおよびパラメーターについてよく知っている必要があります。

シーケンスを設計する前に、 366 ページの『シーケンスと ID 列の比較』を参照してください。

シーケンスは、セットアップと作成が単純であることに加え、値を生成する上での柔軟性を高めるための、以下のようなさまざまな追加のオプションがあります。

- さまざまなデータ・タイプ (SMALLINT、INTEGER、BIGINT、DECIMAL) から選択する
- 開始値を変更する (START WITH)
- シーケンスの増分 (増加する値または減少する値の指定を含む) を変更する (INCREMENT BY)
- シーケンスの番号付けが開始および停止する最小および最大値を設定する (MINVALUE/MAXVALUE)
- シーケンスが再度開始できるように値の折り返しを許可したり、循環を不許可にしたりする (CYCLE/NO CYCLE)
- パフォーマンスを改善するためにシーケンス値のキャッシングを許可したり、キャッシングを不許可にしたりする (CACHE/NO CACHE)

シーケンスが生成された後であっても、これらの値の多くは変更できます。例えば、曜日によって異なる開始値を設定することができます。シーケンスを使用する別の実際的な例として、銀行小切手の生成と処理があります。銀行小切手番号のシーケンスは極めて重要で、シーケンス番号のバッチが失われたり破損したりすると深刻な事態になります。

パフォーマンスを改善するため、CACHE オプションを意識し、活用する必要もあります。このオプションは、カタログに戻り別のシーケンスのセットを生成する前に、システムがいくつのシーケンス値を生成する必要があるかをデータベース・マネージャーに知らせます。デフォルト CACHE 値は、指定しなかった場合、20 です。デフォルトを例として取り上げると、最初のシーケンス値が要求されたときに、データベース・マネージャーは自動的に 20 個の順次値 (1、2、....、20) をメモリー内に生成します。新しいシーケンス番号が必要になるたびに、このメモリー・キャッシュの値を使用して次の値を戻します。このキャッシュの値がすべて使用されると、データベース・マネージャーは次の 21 個の値 (20、22....、40) を生成します。

シーケンス番号のキャッシングをインプリメントすることにより、データベース・マネージャーが頻繁にカタログ表にアクセスして次の値を取得する必要がなくなります。これにより、シーケンス番号の取得に関連したオーバーヘッドは削減されますが、システム障害が発生した場合、またはシステムがシャットダウンした場合には、シーケンスにギャップが発生する可能性もあります。例えば、シーケンス・キャッシュを 100 に設定すると決定した場合、データベース・マネージャーはこれらの数の 100 個の値をキャッシュに入れ、さらに次の値のシーケンスは 201 から開始することを示すようにシステム・カタログを設定します。データベースがシャットダウンした場合、次の一連のシーケンス番号は 201 から開始します。 101 から 200 までの生成された数値については、使用されなかった場合には一連のシーケンスから失われてしまいます。生成された値のギャップがご使用のアプリケーションで容認できない場合、システムのオーバーヘッドは大きくなってしまいますが、キャッシング値を NO CACHE に設定する必要があります。

使用可能なすべてのオプションおよび関連した値についての詳細は、CREATE SEQUENCE ステートメントを参照してください。

シーケンスの動作の管理

アプリケーションのニーズを満たすように、シーケンスの動作を調整することができます。 CREATE SEQUENCE ステートメントを発行して新しいシーケンスを作成する場合、および既存のシーケンスに対して ALTER SEQUENCE ステートメントを発行する場合は、シーケンスの属性を変更します。

指定可能なシーケンスの属性のいくつかを以下に示します。

データ・タイプ
CREATE SEQUENCE ステートメントの AS 節は、シーケンスの数値データ・タイプを指定します。データ・タイプにより、シーケンスの使用可能な最小値と最大値が決まります。データ・タイプごとの最小値と最大値は、SQL および XML の制限値にリストされています。シーケンスのデータ・タイプを変更することはできません。代わりに、DROP SEQUENCE ステートメントを発行してシーケンスをドロップし、新しいデータ・タイプを指定して CREATE SEQUENCE ステートメントを発行する必要があります。

開始値 CREATE SEQUENCE ステートメントの START WITH 節は、シーケンスの初期値を設定します。 ALTER SEQUENCE ステートメントの RESTART WITH 節は、シーケンスの値を指定値にリセットします。

最小値 MINVALUE 節は、シーケンスの最小値を設定します。

最大値 MAXVALUE 節は、シーケンスの最大値を設定します。

増分値 INCREMENT BY 節は、各 NEXT VALUE 式がシーケンスの現行値に追加する値を設定します。シーケンスの値を減らすには、負の値を指定します。

シーケンス循環
CYCLE 節は、シーケンスの値が最小値または最大値に達したとき、次の NEXT VALUE 式でそれぞれ最小値または最大値を初期値に戻します。

注: CYCLE は、ユニークな数値が必要でない場合、またはシーケンスがいったん循環したら古いシーケンス値が使用されなくなることが確実である場合にのみ、使用してください。

例えば、各 NEXT VALUE 式で開始時の最小値が 0、最大値が 1000、増分値が 2 で、最大値に達したときに最小値に戻る id_values というシーケンスを作成するには、次のステートメントを発行します。

```
CREATE SEQUENCE id_values
  START WITH 0
  INCREMENT BY 2
  MAXVALUE 1000
  CYCLE
```

アプリケーション・パフォーマンスおよびシーケンス

ID 列のように、シーケンスを使用して値を生成する場合、一般に、他の方法と比べてアプリケーションのパフォーマンスが向上します。シーケンスに代わる別の方法として、現行値を保管する単一列表を作成し、トリガーを使用して、またはアプリケーションの制御下でその値を増やす方法があります。ただし、単一列表にアプリ

ケーションが並行してアクセスする分散環境では、順番に表にアクセスすることを強制するために必要になるロックが、パフォーマンスに大きく影響します。

シーケンスは、単一列表に関連して発生する可能性のあるロックに関する問題を回避し、シーケンス値をメモリーにキャッシュして応答時間を改善することができます。シーケンスを使用するアプリケーションのパフォーマンスを最大にするには、シーケンスが適切な量のシーケンス値を確実にキャッシュするようにします。CREATE SEQUENCE ステートメントおよび ALTER SEQUENCE ステートメントの CACHE 節は、データベース・マネージャーが生成してメモリーに保管するシーケンス値の最大数を指定します。

シーケンスで順序正しく値を生成する必要があり、システム障害やデータベース非活動化などでその順序が途切れないようにする場合、 ORDER および NO CACHE 節を CREATE SEQUENCE ステートメントで使用します。 NO CACHE 節は、生成された値が途切れないことを保証します。この場合、シーケンスが新しい値を生成するたびにデータベース・ログに書き込むため、アプリケーションのパフォーマンスが低下します。トランザクションがロールバックし、要求したシーケンス値を実際には使用しないために、依然としてギャップが存在する場合があることに注意してください。

シーケンスと ID 列の比較

シーケンスと ID 列は DB2 アプリケーションに対して同じような目的を果たすために使用されているように見えますが、重要な違いがあります。 ID 列は、LOAD ユーティリティーを使用して単一表の列の値を自動的に生成します。シーケンスは、CREATE SEQUENCE ステートメントを使用して SQL ステートメントで使用できる順次値を要求時に生成します。

ID 列 データベース・マネージャーによって、表に追加される各行に対してそれぞれユニークな数値が自動生成されるようにすることができます。表の作成中に、表に追加する各行を一意に識別する必要があることが分かったら、CREATE TABLE ステートメントの一部として表定義に ID 列を追加することができます。

```
CREATE TABLE <table name>
  (<column name 1> INT,
   <column name 2>, DOUBLE,
   <column name 3> INT NOT NULL GENERATED ALWAYS AS IDENTITY
     (START WITH <value 1>, INCREMENT BY <value 2>))
```

この例では、3 番目の列が ID 列になっています。定義可能な属性の 1 つは、行が追加されたときに各行を一意に定義するための列で使用される値です。INCREMENT BY 節に続く値は、ID 列の内容の後続値が、表に追加される行ごとにいくつ増分されるかを示しています。

ID プロパティーは、作成後も、ALTER TABLE ステートメントを使用して変更したり除去したりすることができます。さらに、ALTER TABLE ステートメントで、ほかの列の ID プロパティーを追加することができます。

シーケンス
値の自動生成が可能です。シーケンスは、ユニーク・キー値を生成するタスクに最も適しています。アプリケーションはシーケンスを使用すると、他の方法で固有のカウンターを生成することによって発生する可能性のある、並

行性およびパフォーマンス上の問題を回避することができます。ID 列とは異なり、シーケンスは特定の表列に関連付けられることはありません。また、固有の表列にバインドされて、その表列からしかアクセスできなくなることもありません。

シーケンスは、作成後に変更することができます。そのため、制限なしに、またはユーザー定義の制限まで、値を増分または減分して値を生成することが可能です。ユーザー定義の制限に達した場合は、停止して最初に戻り、再度開始します。シーケンスは、単一パーティションのデータベースでのみサポートされています。

以下の例は、orderseq というシーケンスの作成方法を示しています。

```
CREATE SEQUENCE orderseq
  START WITH 1
  INCREMENT BY 1
  NOMAXVALUE
  NOCYCLE
  CACHE 50
```

この例では、シーケンスは 1 から始まり、1 ずつ増えていき、上限はありません。上限が割り当てられていないため、先頭に戻って循環することはありません。 CACHE パラメーターは、データベース・マネージャーが事前割り当てし、メモリーに保管するシーケンス値の最大値を指定します。

シーケンスの作成

シーケンスを作成するには、CREATE SEQUENCE ステートメントを使用します。ID 列属性とは異なり、シーケンスは特定の表列に関連付けられることはありません。また、固有の表列にバインドされて、その表列からしかアクセスできなくなることもありません。

NEXT VALUE または PREVIOUS VALUE 式が使用できる場合は、いくつかの制約事項があります。以下の方法のいずれかでシーケンスが値を生成するよう、シーケンスを作成または変更することができます。

- バインドなしで単調に (一定量ごとの変化で) 増分または減分する
- ユーザー定義の制限まで単調増分または減分して終了する
- ユーザー定義の制限まで単調増分または減分し、先頭に戻って循環する

注: シーケンスを使用するデータベースをリカバリーする際には注意してください: データベースの外部で使用されるシーケンス値 (銀行小切手に使用されるシーケンス番号など) の場合、データベースをデータベース障害前の時点までリカバリーすると、いくつかのシーケンスで重複値が生成される場合があります。値の重複を回避するため、データベースの外部のシーケンス値を使用するデータベースを以前の状態にリカバリーしないでください。

すべてのオプションでデフォルトを使用して order_seq というシーケンスを作成するには、アプリケーション・プログラム内で、または動的 SQL ステートメントを使用して以下のステートメントを発行します。

```
CREATE SEQUENCE order_seq
```

このシーケンスは 1 から開始し、1 ずつ増加します。上限はありません。

101 から開始し 200 で終わる銀行小切手のバッチの処理では以下の例のようになります。最初のオーダーは 1 から 100 まででした。このシーケンスは 101 から開始し、1 ずつ増加します。上限は 200 です。NOCYCLE は、重複した小切手番号が生成されないように指定されています。 CACHE パラメーターに関連する数値は、データベース・マネージャーが事前割り当てし、メモリーに保管するシーケンス値の最大数を指定します。

```
CREATE SEQUENCE order_seq
   START WITH 101
   INCREMENT BY 1
   MAXVALUE 200
   NOCYCLE
   CACHE 25
```

これらのオプションやその他のオプション、および許可要件については、CREATE SEQUENCE ステートメントを参照してください。

順次値の生成

順次値を生成することは、一般的なデータベース・アプリケーション開発の問題です。この問題を解決する最善の方法は、SQL でシーケンスとシーケンス式を使用することです。各シーケンス は、固有の名前が付けられたデータベース・オブジェクトであり、シーケンス式によってのみアクセスできます。

シーケンス式 には、 PREVIOUS VALUE 式と NEXT VALUE 式の 2 つがあります。 PREVIOUS VALUE 式は、指定されたシーケンスに対してアプリケーション・プロセスで生成された最新の値を返します。 PREVIOUS VALUE 式と同じステートメントで発生する NEXT VALUE 式は、そのステートメントの PREVIOUS VALUE 式で生成された値に対して影響を与えません。 NEXT VALUE シーケンス式は、シーケンスの値を増やして、そのシーケンスの新しい値を返します。

シーケンスを作成するには、CREATE SEQUENCE ステートメントを発行します。例えば、デフォルトの属性を使用して id_values というシーケンスを作成するには、次のステートメントを発行します。

```
CREATE SEQUENCE id_values
```

アプリケーション・セッションで、シーケンスの最初の値を生成するには、次のように NEXT VALUE 式を使用して VALUES ステートメントを発行します。

```
VALUES NEXT VALUE FOR id_values

1-----------
          1
            1 record(s) selected.
```

シーケンスの次の値で列の値を更新するには、次のように UPDATE ステートメントに NEXT VALUE 式を組み込みます。

```
UPDATE staff
  SET id = NEXT VALUE FOR id_values
  WHERE id = 350
```

シーケンスの次の値を使用して新しい行を表に挿入するには、次のように INSERT ステートメントに NEXT VALUE 式を組み込みます。

```
INSERT INTO staff (id, name, dept, job)
  VALUES (NEXT VALUE FOR id_values, 'Kandil', 51, 'Mgr')
```

ID 列またはシーケンスを使用する際の判断

ID 列とシーケンスは類似していますが、異なる点もあります。それぞれの特性は、データベースおよびアプリケーションの設計時に利用することができます。

ご使用のデータベース設計およびそのデータベースを使用するアプリケーションに合わせて、いつ ID 列を使用し、いつシーケンスを使用するべきかを判別する際に、以下に挙げる特性が役立つでしょう。

ID 列の特性

- ID 列は、1 つの表の値を自動的に生成します。
- ID 列が GENERATED ALWAYS として定義されている場合、使用される値は常にデータベース・マネージャーによって生成されます。表の内容の変更中にアプリケーションで独自の値を指定することはできません。
- 行を挿入した後に、IDENTITY_VAL_LOCAL() 関数を使用するか、SELECT FROM INSERT ステートメントを使って挿入行から ID 列を選び直すかして、生成された ID 値を検索することができます。
- LOAD ユーティリティーで、IDENTITY 値を生成することができます。

シーケンスの特性

- シーケンスは、どれか 1 つの表に限定されているものではありません。
- シーケンスは、SQL ステートメントまたは XQuery ステートメントで使用できる順次値を生成します。

  シーケンスはどのアプリケーションでも使用できるため、指定されたシーケンス内の次の値、およびステートメントの実行前に生成された値の検索を制御するための 2 つの式があります。 PREVIOUS VALUE 式は、現行セッション内の直前のステートメントに指定されたシーケンスについて最後に生成された値を返します。 NEXT VALUE 式は、指定されたシーケンスの次の値を返します。これらの式を使用すると、複数の表内の複数の SQL および XQuery ステートメントで同じ値を使用できるようになります。

シーケンスの変更

ALTER SEQUENCE ステートメントで、既存のシーケンスの属性を変更します。

変更可能なシーケンスの属性

- 今後の値の間の増分を変更
- 新しい最小値または最大値を確立
- キャッシュ済みシーケンス番号の数を変更
- シーケンスが循環するかどうかを変更
- 要求の順序でシーケンス番号が生成されるかどうかを変更
- シーケンスを再始動

シーケンス作成の一部ではないタスクが 2 つあります。以下のものが該当します。

- **RESTART:** シーケンスを、そのシーケンスの作成時に開始値として暗黙的または明示的に指定された値にリセットします。

- **RESTART WITH <numeric-constant>:** シーケンスを数値定数にリセットします。数値定数は、小数点以下に非ゼロ桁がない正または負の値です。

シーケンスを再始動、または CYCLE に変更した後、重複するシーケンス番号が生成される可能性があります。今後のシーケンス番号だけが ALTER SEQUENCE ステートメントによって影響を受けます。

シーケンスのデータ・タイプは変更できません。その代わり、現行シーケンスをドロップして、新しいデータ・タイプを指定した新しいシーケンスを作成する必要があります。

シーケンスが変更されると、データベース・マネージャーで使用されていないキャッシュ済みシーケンスの値はすべて失われます。

シーケンス定義の表示

PREVIOUS VALUE オプションを使用する VALUES ステートメントを使用して、シーケンスに関連付けられた参照情報を表示したり、シーケンス自体を表示したりします。

シーケンスの現行値を表示するには、PREVIOUS VALUE 式を使用して VALUES ステートメントを発行します。

```
VALUES PREVIOUS VALUE FOR id_values

1-----------
          1
            1 record(s) selected.
```

シーケンスの現行値は繰り返し検索することができます。シーケンスが返す値は、NEXT VALUE 式を発行するまで変わりません。以下の例では、現行接続の NEXT VALUE 式がシーケンスの値を増やすまで、PREVIOUS VALUE 式は値 1 を返します。

```
VALUES PREVIOUS VALUE FOR id_values

1-----------
          1
            1 record(s) selected.
VALUES PREVIOUS VALUE FOR id_values

1-----------
          1
            1 record(s) selected.
VALUES NEXT VALUE FOR id_values

1-----------
          2

            1 record(s) selected.
VALUES PREVIOUS VALUE FOR id_values

1-----------
          2

            1 record(s) selected.
```

これは、別の接続が同時にシーケンス値を消費している場合においても当てはまります。

シーケンスのドロップ

シーケンスを削除するには、DROP ステートメントを使用します。

シーケンスをドロップする場合、ステートメントの許可 ID は SYSADM または DBADM 権限を保持している必要があります。

以下を実行すると、特定のシーケンスをドロップすることができます。

```
DROP SEQUENCE <sequence_name>
```

<sequence_name> はドロップするシーケンス名で、ここには、既存のシーケンスを正しく識別するための暗黙的または明示的なスキーマ名が入ります。

ID 列のためにシステム作成されたシーケンスは、DROP SEQUENCE ステートメントを使用してドロップすることはできません。

シーケンスをドロップすると、そのシーケンスの特権もすべてドロップされます。

シーケンスのコーディング方法の例

作成されたアプリケーションの多くは、送り状番号、カスタマー番号、および新しい項目が必要になるたびに 1 つずつ増えていく他のオブジェクトを追跡するために、シーケンス番号の使用を必要とします。データベース・マネージャーは、ID 列を使用して表の中の値を自動的に増やしていくことができます。この手法は個々の表には適していますが、複数の表にわたって使用する必要のある固有値を生成するには、最善の方法とは言えない場合もあります。

シーケンス・オブジェクトでは、プログラマーの制御下で増加していく値を作成することができ、多くの表で使用することができます。次の例では、カスタマー番号用に作成されている整数データ・タイプを使ったシーケンス番号が示されています。

```
CREATE SEQUENCE customer_no AS INTEGER
```

デフォルトでは、シーケンス番号は 1 から始まり、一度に 1 つずつ増加し、INTEGER データ・タイプです。アプリケーションは、NEXT VALUE 関数を使用して、シーケンスの次の値を得る必要があります。この関数は、後に続く SQL ステートメントに使用できるシーケンスの次の値を生成します。

```
VALUES NEXT VALUE FOR customer_no
```

プログラマーは、VALUES 関数で次の番号を生成するのではなく、INSERT ステートメント内でこの式を使用することもできます。例えば、カスタマー表の最初の列にカスタマー番号があったなら、INSERT ステートメントを以下のように記述することも可能です。

```
INSERT INTO customers VALUES
  (NEXT VALUE FOR customer_no, 'comment', ...)
```

このシーケンス番号をほかの表に対する挿入で使う必要がある場合は、PREVIOUS VALUE 関数を使用して、直前に生成された値を取得することができます。例えば、作成したばかりのカスタマー番号を後続の送り状レコードで使う必要がある場合、次に示すように SQL に PREVIOUS VALUE 関数を組み込むことができます。

```
INSERT INTO invoices
  (34,PREVIOUS VALUE FOR customer_no, 234.44, ...)
```

PREVIOUS VALUE 関数は、アプリケーション内で複数回使用することができ、そのアプリケーションで生成された最後の値だけを戻します。後続のトランザクションにより、すでにシーケンスが別の値に増えている可能性がありますが、生成された最新の番号が常に表示されることになります。

シーケンス参照

sequence-reference:

nextval-expression:

prevval-expression:

NEXT VALUE FOR *sequence-name*
NEXT VALUE 式は、 *sequence-name* で指定されたシーケンスの次の値を生成して返します。

PREVIOUS VALUE FOR *sequence-name*
PREVIOUS VALUE 式は、現行アプリケーション・プロセス内の直前のステートメントに指定されたシーケンスについて最後に生成された値を返します。この値は、シーケンスの名前が指定されている PREVIOUS VALUE 式を使用して、繰り返し参照することができます。単一ステートメント内に同じシーケンスを指定している PREVIOUS VALUE 式のインスタンスが複数存在する可能性があり、それらはすべて同じ値を返します。パーティション・データベース環境では、最も新しく生成された値が PREVIOUS VALUE 式によって返されない場合があります。

同じシーケンス名が指定されている NEXT VALUE 式がすでに現行アプリケーション・プロセスで、現在または前のトランザクションで参照されている場合のみ、 PREVIOUS VALUE 式を使用できます (SQLSTATE 51035)。

注

- NEXT VALUE 式がシーケンスの名前を指定していれば、そのシーケンスの新しい値が生成されます。ただし、照会の中に同じシーケンス名を指定している

NEXT VALUE 式のインスタンスが複数ある場合、シーケンスのカウンターは結果の行ごとに 1 つずつ増えていき、 NEXT VALUE のすべてのインスタンスが結果の行に同じ値を戻します。

- 以下に示すように、同じシーケンス番号は、先頭の行の NEXT VALUE 式 (これはシーケンス値を生成します) およびその他の行の PREVIOUS VALUE 式 (PREVIOUS VALUE のインスタンスは現在のセッションで最後に生成されたシーケンス値を参照します) を使用してシーケンス番号を参照することによって、 2 つの異なる表内のユニーク・キー値として使用することができます。

```
INSERT INTO order(orderno, cutno)
  VALUES (NEXT VALUE FOR order_seq, 123456);

INSERT INTO line_item (orderno, partno, quantity)
  VALUES (PREVIOUS VALUE FOR order_seq, 987654, 1);
```

- NEXT VALUE 式と PREVIOUS VALUE 式は、以下の位置に指定できます。
  - select-statement または SELECT INTO ステートメント (ステートメントに DISTINCT キーワード、 GROUP BY 節、ORDER BY 節、UNION キーワード、INTERSECT キーワード、または EXCEPT キーワードが入っていなければ、select-clause 内)
  - INSERT ステートメント (VALUES 節内)
  - INSERT ステートメント (全選択の select-clause 内)
  - UPDATE ステートメント (SET 節内 (検索条件付き、または位置指定 UPDATE ステートメントのいずれか)、ただし NEXT VALUE は SET 節にある式の全選択の select-clause 内に指定できない)
  - SET 変数ステートメント (式の全選択の select-clause 内を除きます。トリガー内に NEXT VALUE 式を指定することができますが、PREVIOUS VALUE 式は指定できません)。
  - VALUES INTO ステートメント (式の全選択の select-clause 内)
  - CREATE PROCEDURE ステートメント (SQL プロシージャーのルーチン本体内)
  - トリガー・アクション内の CREATE TRIGGER ステートメント (NEXT VALUE 式は指定できるが、 PREVIOUS VALUE 式は指定できない)
- NEXT VALUE 式と PREVIOUS VALUE 式は、以下の位置には指定できません。
  - 完全外部結合の結合条件
  - CREATE TABLE または ALTER TABLE ステートメント内の列の DEFAULT 値
  - CREATE TABLE または ALTER TABLE ステートメント内の生成された列定義
  - CREATE TABLE または ALTER TABLE ステートメント内のサマリー表定義
  - CHECK 制約の条件
  - CREATE TRIGGER ステートメント (NEXT VALUE 式は指定できるが、 PREVIOUS VALUE 式は指定できない)
  - CREATE VIEW ステートメント
  - CREATE METHOD ステートメント
  - CREATE FUNCTION ステートメント

- XMLQUERY、XMLEXISTS、または XMLTABLE 式の引数リスト

- また、以下の位置に NEXT VALUE 式を指定することはできません (SQLSTATE 428F9)。
  - CASE 式
  - 総計関数のパラメーター・リスト
  - それ以前に明示的に許可されていない場合、コンテキスト内の副照会
  - 外部 SELECT に DISTINCT 演算子を備えた SELECT ステートメント
  - 結合の結合条件
  - 外部 SELECT に GROUP BY 節を備えた SELECT ステートメント
  - 外部 SELECT が UNION、INTERSECT、または EXCEPT セット演算子を使用して他の SELECT ステートメントと組み合わされている場合の SELECT ステートメント
  - ネストされた表の式
  - 表関数のパラメーター・リスト
  - 最外部の SELECT ステートメントか、DELETE または UPDATE ステートメントの WHERE 節
  - 最外部の SELECT ステートメントの ORDER BY 節
  - UPDATE ステートメントの SET 節にある式の全選択の select-clause
  - SQL ルーチンにおける IF、 WHILE、 DO...UNTIL、または CASE ステートメント
- シーケンスについて値が生成されると、その値を再使用できなくなるため、次に値が要求されたときに新しい値が生成されます。 NEXT VALUE 式が組み込まれているステートメントが失敗した場合やロールバックされた場合でも、これが当てはまります。

  列の VALUES リストにある NEXT VALUE 式が INSERT ステートメントに組み込まれており、 INSERT 実行中のある時点でエラー (次のシーケンス値を生成しているときの問題、あるいは別の列の値に問題があると考えられる) が起こった場合、挿入は失敗し (SQLSTATE 23505)、シーケンスについて生成した値は再使用できないものと見なされます。場合によっては、同じ INSERT ステートメントを再発行することによって、正しく動作します。

  たとえば、NEXT VALUE が使用されていた列のユニーク索引が存在する結果としてエラーが起こり、すでに生成されているシーケンス値がその索引に存在するとします。シーケンスについて生成される次の値は、索引には存在しない値になることが考えられるため、後続の INSERT が正しく動作します。
- シーケンスの値の生成において、そのシーケンスが最大値 (または降順シーケンスの最小値) に達し、循環が許可されていない場合、エラーが起こります (SQLSTATE 23522)。この場合、ユーザーはシーケンスを ALTER して許容値の範囲を拡張、またはシーケンスの循環を可能にでき、あるいは値の範囲がより大きな、異なるデータ・タイプを持つ新しいシーケンスを DROP および CREATE することができます。

  たとえば、シーケンスがデータ・タイプ SMALLINT で定義されていて、その結果、そのシーケンスが割り当て可能な値を使い果たしてしまうことがあります。

シーケンスを新しい定義で DROP および再作成して、そのシーケンスを INTEGER として再定義しなければならない場合があります。

- カーソルの SELECT ステートメント内の NEXT VALUE に対する参照は、結果表の行について生成される値を参照します。データベースから取り出される行ごとに NEXT VALUE 式のシーケンス値が生成されます。クライアントでブロッキングを行うと、サーバーで FETCH ステートメントの処理の前に値が生成されることがあります。この状況は、結果表の行がブロッキングされている場合に生じることがあります。クライアント・アプリケーションが、データベースでマテリアライズされている行をすべて明示的に FETCH しないと、 (マテリアライズされている行のうち戻されなかったものの) シーケンス値が生成されません。
- カーソルの SELECT ステートメント内の PREVIOUS VALUE に対する参照は、そのカーソルをオープンする前に、生成された指定シーケンスの値を参照します。しかしながら、カーソルをクローズすると、後続するステートメント内の、PREVIOUS VALUE によって戻される指定シーケンスの値に影響が生じることがあります。このことは、カーソルを再オープンした同じステートメントの場合でも生じることがあります。カーソルの SELECT ステートメントに入っている NEXT VALUE に対する参照中のシーケンス名が同じである場合はこのようになります。
- ***互換性***
  - 以前のバージョンの DB2 との互換性:
    - NEXTVAL と PREVVAL は、NEXT VALUE と PREVIOUS VALUE の代わりに指定できます。
  - IBM IDS との互換性:
    - NEXT VALUE FOR *sequence-name* の代わりに *sequence-name*.NEXTVAL を指定できます。
    - PREVIOUS VALUE FOR *sequence-name* の代わりに *sequence-name*.CURRVAL を指定できます。

例

″order″ という表があり、 ″order_seq″ という以下のようなシーケンスが作成されると想定します。

```
CREATE SEQUENCE order_seq
  START WITH 1
  INCREMENT BY 1
  NO MAXVALUE
  NO CYCLE
  CACHE 24
```

以下は、NEXT VALUE 式で ″order_seq″ シーケンス番号を生成する方法例を示しています。

```
INSERT INTO order(orderno, custno)
  VALUES (NEXT VALUE FOR order_seq, 123456);
```

または

```
UPDATE order
  SET orderno = NEXT VALUE FOR order_seq
  WHERE custno = 123456;
```

または

```
VALUES NEXT VALUE FOR order_seq INTO :hv_seq;
```

第 16 章 ビュー

ビュー は、データを保守せずに表すための効率的な方法です。ビューは実際の表ではなく、また永続ストレージを必要とすることもありません。「仮想表」が作成され、使用されます。

ビュー により、1 つ以上の表にあるデータをさまざまな方法で見ることができます。つまり、ビューとは、結果表に名前を付けて指定したものです。この指定は、ビューが SQL ステートメントで参照されるときにいつも実行される SELECT ステートメントのことです。ビューには表と同じく列と行があります。ビューはすべて、データ・リトリーブにおいて表と同じように使用することができます。挿入、更新、または削除の操作でビューが使用されるかどうかは、その定義により異なります。

ビューには、ベースとなっている表の列または行のすべてまたは一部を含めることができます。例えば、ビューの中で部署表と従業員表を結合して、特定の部署の従業員をすべてリストすることができます。

図 28 は、表とビューの関連を示しています。

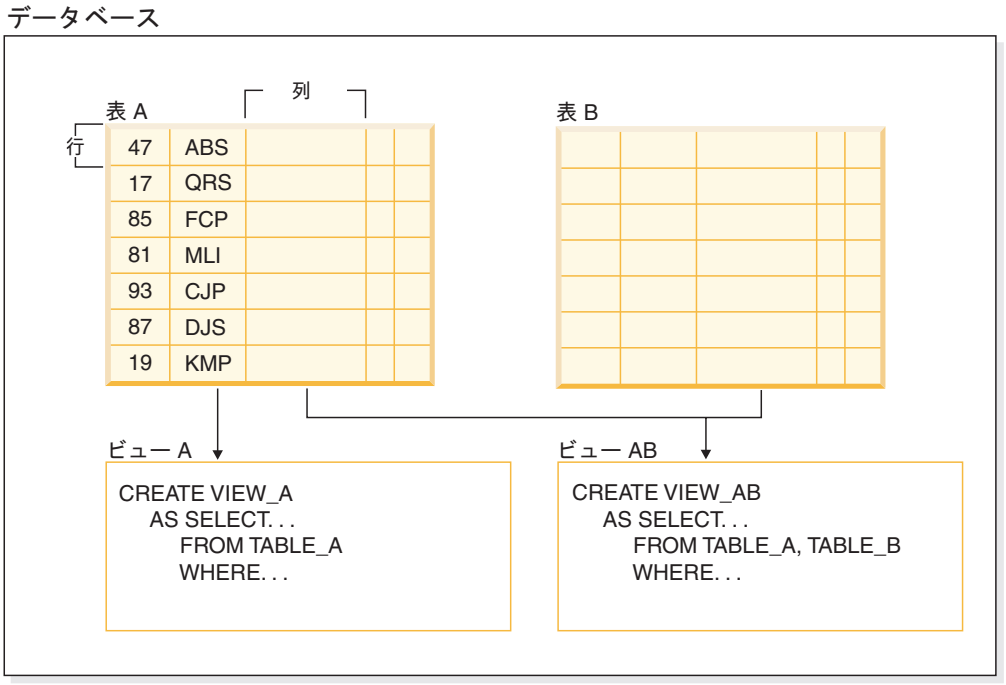


図 28. 表とビューの関係

ビューを使用して、機密データへのアクセスを制御することができます。なぜなら、ビューによって複数のユーザーが同じデータを異なる表示で見ることができるからです。例えば、数人のユーザーが、従業員に関するデータの表にアクセスしているとします。管理職は、自分の部門の従業員のデータは見ることができますが、他の部門の従業員のデータは見ることができません。人事部のユーザーは、すべて

の従業員の雇用日付を見ることができますが給料は見えません。経理課のユーザーは、給料を見ることはできますが雇用日付は見えません。こうしたユーザーはそれぞれ表から派生したビューで作業します。各ビューは、1 つの表のように表示され、それぞれ固有の名前があります。

ビューの列が基本表の列から直接に派生している場合、そのビューの列は表の列に適用されるあらゆる制約を継承します。例えば、ビューにその表の外部キーが入っている場合、そのビューを使用する挿入および更新操作は表と同じ参照制約に従います。また、ビューの表が親表である場合、そのビューを使用する削除および更新操作は、表の削除および更新操作と同じ規則に従います。

ビューでは、列ごとにデータ・タイプを結果表から派生させる (つまり、型をユーザー定義の構造化タイプの属性に基づいたものにする) ことができます。このようなビューを*型付きビュー* と呼びます。型付き表と同様に、型付きビューもビュー階層の一部になることができます。*サブビュー* は、*スーパービュー* から列を継承します。 *サブビュー* という語は、型付きビュー、およびビュー階層でその下にあるすべての型付きビューに当てはまります。ビュー V の*厳密な意味*でのサブビューとは、型付きビュー階層で V の下にあるビューのことです。

ビューが作動不能になる場合があります (表がドロップされた場合など)。これが発生すると、そのビューは SQL 操作では使えなくなります。

ビューの設計

ビュー により、1 つ以上の表にあるデータをさまざまな方法で見ることができます。つまり、ビューとは、結果表に名前を付けて指定したものです。

この指定は、ビューが SQL ステートメントで参照されるときにいつも実行される SELECT ステートメントのことです。ビューには、基本表と同じように列と行があります。ビューはすべて、データ・リトリーブにおいて表と同じように使用することができます。挿入、更新、または削除の操作でビューが使用されるかどうかは、その定義により異なります。

ビューは、可能な操作ごとに分類されます。分類には以下のものがあります。

- 削除可能
- 更新可能
- 挿入可能
- 読み取り専用

ビュー型は、その更新機能に合わせて設定されます。この種別は、ビューに対して許可されている SQL 操作の種類を示しています。

参照制約およびチェック制約は、それぞれ独立して扱われます。これらの制約は、ビューの種別に影響を与えることはありません。

例えば、参照制約のために、表に行を挿入することができない場合があります。その表を使用してビューを作成すると、そのビューを使って行を挿入することもできなくなります。しかし、ビューが挿入可能ビューのルールをすべて満たしているなら、挿入可能なビューとみなされます。これは、挿入制約が表に基づいており、ビュー定義に基づいてはいないからです。

詳しくは、CREATE VIEW ステートメントを参照してください。

システム・カタログ・ビュー

データベース・マネージャーは、その制御下のデータに関する情報の組み込まれた一連の表とビューを管理しています。これらの表とビューをまとめて、システム・カタログ と呼びます。

このシステム・カタログには、表、ビュー、索引、パッケージ、および関数といったデータベース・オブジェクトの論理および物理構造に関する情報が含まれています。統計情報もあります。データベース・マネージャーは、システム・カタログの情報が常に正確であるように管理します。

システム・カタログ・ビューは、ほかのデータベースのビューと類似しています。システム・カタログ・ビューのデータを照会するために、SQL ステートメントを使用することができます。更新可能なシステム・カタログ・ビューのセットを使用して、そのシステム・カタログの特定の値を変更することができます。

チェック・オプションのあるビュー

WITH CHECK OPTION で定義されているビューは、そのビューの SELECT ステートメントに対して、修正または挿入される行を強制します。このチェック・オプションのあるビューはシンメトリック・ビュー とも呼ばれています。例えば、部門 10 の従業員しか戻さないシンメトリック・ビューでは、ほかの部門の従業員の挿入は行えません。そのため、このオプションによりデータベースで変更されているデータの整合性が確保され、INSERT または UPDATE 操作中に条件の違反があるとエラーが戻されます。

アプリケーションの規則で表チェック制約として定義できないものがある場合や、使用するすべてのデータには当てはまらない規則がある場合には、アプリケーションのロジックへの規則の組み込みに代わる方法があります。データの条件を WHERE 節または WITH CHECK OPTION 節の一部として指定した、表のビューを作成することができます。このビュー定義は、アプリケーションに有効なデータ・セットに対して、検索できるデータを制限します。さらに、ビューが更新可能な場合は、WITH CHECK OPTION 節がアプリケーションに適用できる行の更新、挿入、および削除を制限します。

以下のビューでは、WITH CHECK OPTION を指定してはなりません。

- 読み取り専用オプションで定義されているビュー (読み取り専用ビュー)
- NODENUMBER または PARTITION 関数、非決定的関数 (RAND など)、もしくは外部アクションのある関数を参照するビュー
- 型付きビュー

例 1

以下に、WITH CHECK OPTION を使用したビュー定義の例を示します。条件が常に確実にチェックされるためには、このオプションは必須です。このビューでは、DEPT が必ず 10 になります。これにより、DEPT 列に対する値の入力が制限されます。新しい値の挿入にビューが使用される場合、WITH CHECK OPTION が常に強制されます。

```
CREATE VIEW EMP_VIEW2
  (EMPNO, EMPNAME, DEPTNO, JOBTITLE, HIREDATE)
AS SELECT ID, NAME, DEPT, JOB, HIREDATE FROM EMPLOYEE
  WHERE DEPT=10
WITH CHECK OPTION;
```

このビューが INSERT ステートメントで使用される場合、DEPTNO 列の値が 10 でないとその行は拒否されます。WITH CHECK OPTION が指定されていないと、変更中のデータ検証は行われませんので、注意してください。

このビューが SELECT ステートメントで使用されると、条件 (WHERE 節) が呼び出されて、データでその条件にマッチする行のみが結果表に入ります。言い換えるなら、WITH CHECK OPTION は、SELECT ステートメントの結果には影響を及ぼしません。

例 2

ビューを使うと、表データのうちアプリケーション・プログラムで利用できるサブセットを作成し、挿入または更新するデータの妥当性検査を実行することができます。ビューの列名は、元の表の対応する列名と違うものにすることができます。例:

```
CREATE VIEW <name> (<column>, <column>, <column>)
   SELECT <column_name> FROM <table_name>
   WITH CHECK OPTION
```

例 3

ビューを使うと、プログラムやエンド・ユーザー照会で表データを見る方法の点で柔軟性が高くなります。

以下の SQL ステートメントは、EMPLOYEE 表に 1 つのビューを作成します。このビューは部門 A00 のすべての従業員を、従業員番号と電話番号の情報とともにリストします。

```
CREATE VIEW EMP_VIEW (DA00NAME, DA00NUM, PHONENO)
   AS SELECT LASTNAME, EMPNO, PHONENO FROM EMPLOYEE
   WHERE WORKDEPT = 'A00'
   WITH CHECK OPTION
```

このステートメントの最初の行では、ビューの名前を指定し、その列を定義しています。 EMP_VIEW という名前は、 SYSCAT.TABLES の中のそのスキーマ内で固有のものでなければなりません。ビュー名は表名の 1 つとして表示されますが、データは含まれていません。このビューの列は DA00NAME、 DA00NUM、および PHONENO の 3 つであり、それぞれ、 EMPLOYEE 表の LASTNAME、EMPNO、および PHONENO の列に対応するものです。最初の行に指定する列名は、SELECT ステートメントの選択リストに 1 対 1 に対応します。列名を指定しないと、ビューの列名には SELECT ステートメントの結果表の列と同じ名前が使われます。

第 2 行は、データベースから選択する値について記述する SELECT ステートメントです。これには、ALL、DISTINCT、FROM、WHERE、GROUP BY、および HAVING という節を含めることができます。ビューのための列を選択する元のデータ・オブジェクトの名前を、 FROM 節の後に指定します。

例 4

WITH CHECK OPTION 節は、ビューに対して更新する行または挿入する行をビュー定義に照らしてチェックし、定義に従っていない場合にはリジェクトすることを指定するものです。これによってデータ整合性は向上しますが、余分な処理が必要になります。この節を省略すると、挿入する行または更新する行がビュー定義に照らしてチェックされることはありません。

次の SQL ステートメントは、EMPLOYEE 表に基づく同じビューを、 SELECT AS 節を使って作成するものです。

```
CREATE VIEW EMP_VIEW
   SELECT LASTNAME AS DA00NAME,
          EMPNO AS DA00NUM,
          PHONENO
   FROM EMPLOYEE
   WHERE WORKDEPT = 'A00'
   WITH CHECK OPTION
```

この例では、EMPLOYEE 表には給与に関する情報が入っているかもしれませんが、そうした情報は誰もが利用できるようにすべきではありません。しかし、従業員の電話番号はだれでもアクセスできるようにすべきです。このような場合は、LASTNAME 列と PHONENO 列だけからなるビューを作成できます。ビューへのアクセスは PUBLIC に与えるようにし、EMPLOYEE 表全体へのアクセスは、給与情報を見る権限のある人だけに制限するようにします。

ネストされたビュー定義

別のビューをベースにしているビューの場合、評価する必要のある述部の数は、WITH CHECK OPTION 指定に基づきます。

ビューが WITH CHECK OPTION なしで定義されている場合、そのビューの定義が、挿入や更新操作のデータ妥当性検査で使用されることはありません。しかし、ビューが直接または間接的に WITH CHECK OPTION で定義された別のビューに依存している場合、そのスーパービューの定義は挿入や更新操作のデータ妥当性検査で使用されます。

WITH CASCADED CHECK OPTION または単なる WITH CHECK OPTION (CASCADED は WITH CHECK OPTION のデフォルト値) でビューが定義されている場合、そのビューの定義は挿入や更新操作の検査で使用されます。さらにビューは、そのビューが依存している更新可能のビューから検索条件を継承します。これらの条件は、それらのビューに WITH CHECK OPTION が組み込まれていない場合でも継承されます。ビューに対する、またはそのビューに依存しているほかのビューに対する挿入または更新操作に適用される制約に準拠するため、それら継承された条件は掛け合わされます。

例えば、ビュー V2 がビュー V1 をベースにしており、V2 の検査オプションが WITH CASCADED CHECK OPTION と定義されている場合、INSERT および UPDATE ステートメントがビュー V2 に対して実行されると、両方のビューの述部が評価されます。

```
CREATE VIEW EMP_VIEW2 AS
  SELECT EMPNO, EMPNAME, DEPTNO FROM EMP
    WHERE DEPTNO = 10
WITH CHECK OPTION;
```

次に示す例では、WITH CASCADED CHECK OPTION を使用した CREATE VIEW ステートメントを示しています。ビュー EMP_VIEW3 は、ビュー EMP_VIEW2 をベースにして作成され、ビュー EMP_VIEW2 は WITH CHECK OPTION を指定して作成されています。EMP_VIEW3 に対してレコードを挿入または更新したい場合、そのレコードには値 DEPTNO=10 および EMPNO=20 が必要です。

```
CREATE VIEW EMP_VIEW3 AS
  SELECT EMPNO, EMPNAME, DEPTNO FROM EMP_VIEW2
    WHERE EMPNO > 20
WITH CASCADED CHECK OPTION;
```

注: EMP_VIEW2 に WITH CHECK OPTION が組み込まれていなくても、EMP_VIEW3 に対する挿入または更新操作で条件 DEPTNO=10 が強制されます。

WITH LOCAL CHECK OPTION は、ビュー作成時にも指定することができます。ビューが LOCAL CHECK OPTION で定義されると、挿入や更新操作の検査でそのビューの定義が使用されます。ただし、そのビューが依存している更新可能なビューからの検索条件を継承することはありません。

削除可能ビュー

ビューの定義方法によって、ビューを削除可能にすることができます。削除可能ビューとは、DELETE ステートメントを正常に発行することができるビューのことです。

ビューが削除可能とみなされるには、以下に示すように従う必要のある規則がいくつかあります。

- 外部全選択の各 FROM 節には、表 (OUTER 節なし)、削除可能ビュー (OUTER 節なし)、削除可能なネストした表式、または削除可能な共通表式のいずれかが 1 つだけ指定されている。
- データベース・マネージャーが、ビュー定義を使用して表の削除する行を派生させることができる必要がある。ある操作を行うとこれができなくなります。
  - GROUP BY 節または列関数を使用して複数行を 1 つにグループ化すると、元の行が失われてビューが削除不可になります。
  - 同様に行が VALUES から派生している場合、削除する表はありません。この場合もビューは削除可能になりません。
- 外部全選択で、GROUP BY または HAVING 節が使われていない。
- 外部全選択で、選択リストに列関数が組み込まれていない。
- 外部全選択で、UNION ALL を除いてセット演算 (UNION、EXCEPT、または INTERSECT) が使われていない。
- UNION ALL のオペランドの表が同じ表であってはならず、各オペランドが削除可能でなければならない。
- 外部全選択の選択リストに DISTINCT が含まれていない。

ビューが削除可能ビューとみなされるには、上記の規則をすべて満たしている必要があります。例えば、次のビューは削除可能です。削除可能ビューの規則にすべて従っています。

```
CREATE VIEW deletable_view
  (number, date, start, end)
AS
  SELECT number, date, start, end
  FROM employee.summary
  WHERE date='01012007'
```

挿入可能ビュー

挿入可能ビューでは、ビュー定義を使用して行を挿入することができます。挿入操作の INSTEAD OF トリガーがビューに対して定義されている場合、またはビューの少なくとも 1 つの列が更新可能である場合 (更新の INSTEAD OF トリガーとは無関係)、ビューの全選択に UNION ALL が含まれていないなら、そのビューは挿入可能です。基礎となる表のうちの 1 つの表のチェック制約を特定の行が正確に満たしている場合にのみ、その行をビュー (UNION ALL を含む) に挿入できます。更新不可の列が含まれているビューに挿入するには、それらの列を列リストから除外する必要があります。

以下に示すビューは、挿入可能ビューです。ただし、この例では、ビューを挿入しようとすると失敗します。これは、その表に NULL 値を受け入れない列があるためです。これらの列のいくつかは、ビュー定義には示されません。このビューを使用して値を挿入しようとすると、データベース・マネージャーが NOT NULL 列に NULL 値を挿入しようとします。このアクションは許可されていません。

```
CREATE VIEW insertable_view
  (number, name, quantity)
AS
SELECT number, name, quantify FROM ace.supplies
```

注: この表で定義されている制約は、この表をベースにしているビューを使用して実行できる操作とは無関係です。

更新可能ビュー

更新可能ビューとは、特殊な削除可能ビューです。削除可能ビューの少なくとも 1 つの列が更新可能であると、そのビューは更新可能ビューになります。

以下の規則がすべて当てはまる場合、ビューの列が更新可能になります。

- ビューが削除可能である。
- 列の解決結果が表の列 (間接参照操作は使用しない) となり、READ ONLY オプションが指定されていない。
- ビューの全選択に UNION ALL が含まれる場合、 UNION ALL のオペランドの対応するすべての列のデータ・タイプおよびデフォルト値が正確に一致している (長さ、または精度と位取りを含む)。

次の例では、更新できない定数値を使用しています。しかし、ビューが削除可能ビューであり、その列の少なくとも 1 つは更新可能になっています。そのため、これは更新可能ビューになります。

```
CREATE VIEW updatable_view
  (number, current_date, current_time, temperature)
AS
  SELECT number, CURRENT DATE, CURRENT TIME, temperature)
FROM weather.forecast
WHERE number = 300
```

読み取り専用ビュー

ビューが削除可能でも、更新可能でも、挿入可能でもない 場合、それは*読み取り専用* ビューになります。削除可能ビューの少なくとも 1 つの規則に従っていないビューである場合、それは読み取り専用になります。

SYSCAT.VIEWS カタログ・ビューの READONLY 列は、ビューが読み取り専用 (R) であることを示しています。

次に示す例は、DISTINCT 節が使用され、SQL ステートメントに複数の表が関係しているので、削除可能ビューではありません。

```
CREATE VIEW read_only_view
  (name, phone, address)
AS
SELECT DISTINCT viewname, viewphone, viewaddress
FROM employee.history adam, employer.dept sales
WHERE adam.id = sales.id
```

ビューの作成

ビューは 1 つ以上の表、ニックネーム、またはビューから導出されるもので、データの検索時には表と交換可能なものとして使用されます。ビューの中に表示されるデータに変更が加えられると、表そのもののデータも変更されます。表、ニックネーム、またはビューの基礎となるビューが、ビューを作成する前に既に存在していなければなりません。

ビューを作成することによって、重要データについてはアクセスを限定し、その他のデータについては一般にアクセスを許可することができます。

挿入先のビューにおいて、ビュー定義の選択リストに直接または間接的に表の ID 列の名前が含まれている場合は、INSERT ステートメントが表の ID 列を直接参照する場合と同じ規則が適用されます。

上記のようなビューの使用に加えて、次のような目的でビューを使用することができます。

- アプリケーション・プログラムに影響を及ぼさずに表を変更する。このことは、基礎表に基づいてビューを作成することによって生じます。基礎表を使用するアプリケーションは、新しいビューの作成によって影響を受けることはありません。新しいアプリケーションは、基礎表を使用するアプリケーションとは異なる目的のために作成されたビューを使用できます。
- 列の中の値を合計する、最大値を選択する、または、それらの値を平均する。
- 1 つまたは複数のデータ・ソースの中の情報へのアクセスを提供する。 CREATE VIEW ステートメント内のニックネームを参照し、複数ロケーション/グローバル・ビュー (ビューは異なるシステム上にある複数データ・ソース内の情報を結合できる) を作成することができます。

  標準の CREATE VIEW 構文を使ってニックネームを参照するビューを作成すると、基礎オブジェクトまたはデータ・ソースのオブジェクトへのアクセスにビュー作成者認証 ID ではなくビュー・ユーザーの認証 ID が使用されるという事実に注意を促す警告が表示されます。この警告を表示しないようにするには、FEDERATED キーワードを使用します。

型付きビューは、事前定義された構造化タイプに基づいています。 CREATE VIEW ステートメントを使用して、型付きビューを作成することができます。

ビューを作成する代わりにネストした表式または共通表式を使うこともできます。その場合、カタログ参照が少なくなり、パフォーマンスは高くなります。

CREATE VIEW ステートメントの例を以下に示します。基礎表 EMPLOYEE には、SALARY および COMM という名前の列があります。セキュリティー上の理由で、このビューは、ID、NAME、DEPT、JOB、および HIREDATE の各列から作成されます。さらに、DEPT 列へのアクセスは制限されます。この定義は、DEPTNO が 10 である部門に所属する従業員の情報のみを表示します。

```
CREATE VIEW EMP_VIEW1
  (EMPID, EMPNAME, DEPTNO, JOBTITLE, HIREDATE)
  AS SELECT ID, NAME, DEPT, JOB, HIREDATE FROM EMPLOYEE
    WHERE DEPT=10;
```

ビューの定義を行ったあと、アクセス権を指定できます。これにより、アクセス可能な表のビューは制限されるため、データ・セキュリティーを確保できます。上記のように、ビューには、特定の行へのアクセスを制限するための WHERE 節を含めたり、データの特定の列へのアクセスを制限するための列のサブセットを含めたりすることができます。

ビューの列名は、基本表の列名と一致していなくても構いません。表名には、ビュー名と同様、関連スキーマがあります。

ビューの定義が完了すると、(制限はありますが)
SELECT、INSERT、UPDATE、DELETE などのステートメントで使用できます。
DBA は、このビューに対して、表より高いレベルの特権を特定のユーザーのグループに付与することができます。

ユーザー定義関数 (UDF) を使用するビューの作成

UDF を使用するビューを一度作成すると、そのビューは、同じ名前を持つ他の UDF を後で作成した場合でも、そのビューが存在するかぎりこの同じ UDF を必ず使用します。新しい UDF を採用する場合には、ビューを再作成する必要があります。

次の SQL ステートメントは、定義内に関数の含まれるビューを作成するものです。

```
CREATE VIEW EMPLOYEE_PENSION (NAME, PENSION)
  AS SELECT NAME, PENSION(HIREDATE,BIRTHDATE,SALARY,BONUS)
  FROM EMPLOYEE
```

UDF 関数の PENSION によって、従業員が現時点で受け取ることのできる年金が計算されます。その計算には、HIREDATE、BIRTHDATE、SALARY、および BONUS が使われます。

型付きビューの変更

型付きビューの特定のプロパティーは、ビューをドロップまたは再作成する必要なく変更できます。そのようなプロパティーの 1 つとして、型付きビューの参照列への有効範囲の追加があります。

ALTER VIEW ステートメントは、有効範囲を追加するよう参照タイプ列を変更することによって、既存の型付きビュー定義を変更します。 DROP ステートメントは型付きビューを削除します。また、以下のいずれかを実行することもできます。

- INSTEAD OF トリガーを使用して型付きビューの内容を変更する。
- 統計収集を使用可能にするように型付きビューを変更する。

型付きビューの基礎となる内容を変更すると、トリガーの使用が必要になります。型付きビューに加えるその他の変更では、型付きビューをドロップした後に再作成する必要があります。

ALTER VIEW ステートメントの列名のデータ・タイプは、 REF (型付き表名または型付きビュー名のタイプ) でなければなりません。

パッケージ、またはキャッシュに入った動的ステートメントに無効のマークが付けられても、表および索引のような他のデータベース・オブジェクトは影響されません。

コマンド行を使用して型付きビューを変更するには、以下のように入力します。

```
ALTER VIEW <view_name> ALTER <column_name>
   ADD SCOPE <typed table or view name>
```

作動不能ビューの回復

作動不能ビュー とは、SQL ステートメントで使用できなくなったビューのことです。

ビューは、次のような場合に*作動不能* になることがあります。

- 基礎表での特権が取り消された結果。
- 表、別名、または関数がドロップされる場合。
- スーパービューが作動不能になる場合。 (スーパービューとは、別の型付きビュー (サブビュー) がそれを基にする型付きビューのことです。)
- 従属するビューがドロップされる場合

次のステップは、作動不能ビューを回復するのに役に立ちます。

1. ビューを作成するために最初に使用した SQL ステートメントを判別する。この情報は SYSCAT.VIEW カタログ・ビューの TEXT 列から獲得することができます。
2. 現行スキーマに QUALIFIER 列の内容を設定する。
3. 関数パスに FUNC_PATH 列の内容を設定する。
4. CREATE VIEW ステートメントを使用して、同じビュー名と同じ定義でビューを再作成する。
5. GRANT ステートメントを使用して、ビューに以前に付与されていたすべての特権を再度付与する。 (作動不能ビューに付与されていたすべての特権は取り消されていることに注意してください。)

作動不能ビューを回復したくない場合は、 DROP VIEW ステートメントを使用してそのビューを明示的にドロップするか、または同じ名前と別の定義を使用して新規のビューを作成することができます。

作動不能ビューは、SYSCAT.TABLES および SYSCAT.VIEWS カタログ・ビューにしか項目がありません。 SYSCAT.VIEWDEP、SYSCAT.TABAUTH、SYSCAT.COLUMNS、および SYSCAT.COLAUTH カタログ・ビューのすべての項目が除去されます。

ビューのドロップ

DROP VIEW ステートメントを使用してビューをドロップします。ドロップするビューに依存するビューがあれば、それらは作動不能になります。

コマンド行を使用してビューをドロップするには、以下のように入力します。

```
DROP VIEW <view_name>
```

以下の例は、EMP_VIEW という名前のビューをドロップする方法を示したものです。

```
DROP VIEW EMP_VIEW
```

表階層の場合と同様、階層のルート・ビューを指定することで、次の例に示すとおり、1 つのステートメント内でビュー階層全体をドロップすることができます。

```
DROP VIEW HIERARCHY VPerson
```

第 4 部 リファレンス

第 17 章 命名規則への準拠

命名規則

すべてのオブジェクト、ユーザー、およびグループの命名には規則があります これらの規則には、作業しているプラットフォームに特有のものもあります。

たとえば、名前に大文字と小文字を使用することに関連した規則があります。

- UNIX プラットフォームでは、名前は小文字でなければなりません。
- Windows プラットフォームでは、名前は大文字でも、小文字でも、大/小文字混合でも構いません。

特に指定がない限り、名前には以下の文字を含めることができます。

- A から Z までの文字。名前に使用されるとき、多くの場合 A から Z は小文字から大文字に変換されます。
- 0 から 9 の数字。
- ! % ( ) { } . – ^ ~ _ (アンダースコアー) @、#、$、およびスペース。
- ¥ (円記号)。

名前の先頭に、数値または下線文字を使用することはできません。

表、ビュー、列、索引、または許可 ID の名前には、SQL 予約語を使用しないでください。

ご使用のオペレーティング・システム、および DB2 データベースを操作している場所によって、異なる働きをする特殊文字が他にもあります。 それらの文字は正常に機能する可能性がありますが、必ず機能するという保証はありません。データベース内のオブジェクトを命名する際には、これら他の特殊文字を使用することはお奨めしません。

ユーザーおよびグループ名は、関連したシステムによって特定のオペレーション・システムに強制された規則にも従う必要があります。例えば、Linux および UNIX プラットフォームでは、ユーザー名および 1 次グループ名に許可されている文字は、名前として、a から z までの小文字、0 から 9 までの数字、および _ (下線) である必要があります (ただし、0 から 9 までの数字で始まらないこと)。

各長さは、SQL および XML の制限値にリストされた長さ以下である必要があります。

他にも、オブジェクト命名規則、NLS 環境での命名規則、および Unicode 環境での命名規則を考慮する必要があります。

AUTHID ID に関する制約事項: DB2 データベース・システムのバージョン 9.5 以降では、128 バイトの許可 ID を付けることが可能ですが、許可 ID がオペレーティング・システムのユーザー ID またはグループ名として解釈される場合は、オペレーティング・システムの命名上の制約事項が適用されます (例えば、8 または 30 文字のユーザー ID と 30 文字のグループ名の制限)。このため、128 バイトの許可

ID を付与できますが、この許可 ID を持つユーザーとしては接続することができません。独自のセキュリティー・プラグインを作成した場合は、許可 ID の拡張されたサイズを最大限に活用することができます。例えば、セキュリティー・プラグインに 30 バイトのユーザー ID を与えて、接続可能な認証中に、セキュリティー・プラグインが 128 バイトの許可 ID を返すことができます。

DB2 オブジェクト命名規則

すべてのオブジェクトは、一般的な命名規則に従います。さらに、いくつかのオブジェクトには、表に示されているような追加の制約事項があります。

表 51. データベース、データベース別名、およびインスタンスの命名規則

オブジェクト	ガイドライン
• データベース • データベース別名 • インスタンス	• データベース名は、それらがカタログされるロケーションにおいて固有でなければなりません。このロケーションとは、Linux および UNIX インプリメンテーションではディレクトリー・パス、Windows インプリメンテーションでは論理ディスクです。 • データベース別名は、システム・データベース・ディレクトリー内で固有でなければなりません。新規データベースが作成されると、デフォルトでは別名としてデータベース名が設定されます。そのため、たとえその名前のデータベースが存在しないとしても、データベース別名として既に存在する名前を使ってデータベースを作成することはできません。 • データベース、データベース別名、およびインスタンス名の長さは、SQL および XML の制限値にリストされた長さ以下である必要があります。 • Windows では、インスタンスはサービス名と同じ名前を持つことができません。 **注:** 通信環境でデータベースを使用する場合、問題を未然に防ぐために、特殊文字 @、#、および $ はデータベース名に使用しないでください。さらに、これらの特殊文字はすべてのキーボードに共通ではないので、他の言語でデータベースを使用する場合にも使用しないでください。

表 52. データベース・オブジェクトの命名規則

オブジェクト	ガイドライン
• 別名 • 監査ポリシー • バッファー・プール • 列 • イベント・モニター • 索引 • メソッド • ノード・グループ • パッケージ • パッケージ・バージョン • ロール • スキーマ • ストアード・プロシージャー • 表 • 表スペース • トリガー • トラステッド・コンテキスト • UDF • UDT • ビュー	これらのオブジェクトの長さは、SQL および XML の制限値にリストされた長さ以下である必要があります。オブジェクト名。次のものを含めることができます。 • 有効なアクセント付き文字 (ö など) • マルチバイト文字。マルチバイト・スペースは除く (マルチバイト環境の場合) パッケージ名およびパッケージ・バージョンにはピリオド (.)、ハイフン (-)、およびコロン (:) もまた含めることができます。

表 53. フェデレーテッド・データベース・オブジェクトの命名規則

オブジェクト	ガイドライン
• 関数マッピング • SPECIFICATION ONLY 指定の索引 • ニックネーム • サーバー • タイプ・マッピング • ユーザー・マッピング • ラッパー	これらのオブジェクトの長さは、SQL および XML の制限値にリストされた長さ以下である必要があります。フェデレーテッド・データベース・オブジェクトの名前には、次のものも含めることができます。 • 有効なアクセント付き文字 (ö など) • マルチバイト文字。マルチバイト・スペースは除く (マルチバイト環境の場合)

区切り ID およびオブジェクト名

キーワードを使用することができます。キーワードが SQL キーワードとしても解釈されるコンテキストで使用される場合には、それを区切り ID として指定する必要があります。

区切り ID を使用することによって、上記の命名規則に違反するオブジェクトを作成することは可能ですが、そのオブジェクトを続けて使おうとするとエラーになります。例えば、名前に + または - 記号が含まれている列を作成し、その列を索引の列として使おうとすると、索引の表を再編成する段階で問題が起きてしまいます。

スキーマ名の追加情報

- ユーザー定義タイプ (UDT) には、SQL および XML の制限値にリストされた長さを超えるスキーマ名を指定することはできません。
- 次のスキーマ名は予約語なので、使用しないでください。SYSCAT、SYSFUN、SYSIBM、SYSSTAT。
- マイグレーションの問題を未然に防ぐために、SYS で始まるスキーマ名を使用しないでください。データベース・マネージャーでは SYS で始まるスキーマ名を使用して、トリガー、ユーザー定義タイプ、またはユーザー定義関数を作成できません。
- SESSION をスキーマ名として使用しないようお奨めします。宣言される一時表は、SESSION によって修飾しなければなりません。したがって、アプリケーションに永続表と同じ名前の一時表を宣言させることができますが、その場合、アプリケーション・ロジックが複雑になりすぎる可能性があります。宣言される一時表を扱う場合以外は、スキーマ SESSION の使用を避けてください。

区切り ID およびオブジェクト名

キーワードを使用することができます。キーワードが SQL キーワードとしても解釈されるコンテキストで使用される場合には、それを区切り ID として指定する必要があります。

区切り ID を使用することによって、上記の命名規則に違反するオブジェクトを作成することは可能ですが、そのオブジェクトを続けて使おうとするとエラーになります。例えば、名前に + または - 記号が含まれている列を作成し、その列を索引の列として使おうとすると、索引の表を再編成する段階で問題が起きてしまいます。

ユーザー、ユーザー ID、およびグループの命名規則

ユーザー、ユーザー ID、およびグループ名は、命名のガイドラインに従う必要があります。

表 54. ユーザー、ユーザー ID、およびグループの命名規則

オブジェクト	ガイドライン
• グループ名 • ユーザー名 • ユーザー ID	• グループ名は、SQL および XML の制限値にリストされたグループ名の長さ以下である必要があります。 • Linux および UNIX オペレーティング・システムでは、ユーザー ID は最大 8 文字です。 • Windows 上のユーザー名は、最大 30 文字です。 • クライアント認証を使用しない場合は、Windows に接続している、ユーザー名が SQL および XML の制限値にリストされたユーザー名の長さを超える Windows 以外の 32 ビット・クライアントがサポートされます (ユーザー名およびパスワードが明示的に指定される場合)。 • 名前および ID には、次の禁止事項があります。 – USERS、 ADMINS、 GUESTS、 PUBLIC、 LOCAL、または任意の SQL 予約語であってはならない。 – IBM、SQL、または SYS で始めてはならない。

注:

1. いくつかのオペレーティング・システムでは、大文字小文字の区別があるユーザー ID およびパスワードを使用できます。それが使用できるかどうかは、ご使用のオペレーティング・システムの資料で確認してください。
2. 正常実行された CONNECT または ATTACH から戻された許可 ID は、SQL および XML の制限値にリストされた許可名の長さで切り捨てられます。省略符号 (...) が許可 ID に追加され、 SQLWARN フィールドに切り捨てられたことを示す警告が入ります。
3. ユーザー ID およびパスワードの末尾ブランクは、除去されます。

NLS 環境での命名規則

データベース名の中で使用できる基本文字セットは、単一バイトの大文字および小文字のローマ字 (A...Z、a...z)、アラビア数字 (0...9) および下線文字 (_) から構成されます。

ホストのデータベース製品との互換性を保つために、 3 つの特殊文字 (#、@、および $) がこのリストに加えられています。特殊文字 #、@、および $ は NLS ホスト (EBCDIC) 不変文字セットに組み込まれていないため、これらを NLS 環境で使用する場合は注意してください。使用されているコード・ページに応じて、拡張文字セットの文字も使用できます。複数コード・ページ環境でデータベースを使用し

ている場合、使用する拡張文字セットのどのエレメントもすべてのコード・ページによってサポートされていることを確認する必要があります。

データベース・オブジェクト (表やビューなど) に命名するときは、プログラム・ラベル、ホスト変数、カーソル、および拡張文字セットからのエレメント (たとえば、発音区別符号付きの文字) も使用することができます。厳密にどの文字を使用できるかは、使用しているコード・ページによって異なります。

DBCS ID の拡張文字セット定義: DBCS 環境では、拡張文字セットは、基本文字セットにあるすべての文字、および次のような文字から構成されます。

- おのおのの DBCS コード・ページにある 2 バイト文字は、すべて (2 バイトのスペースは除く) 有効な文字です。
- 2 バイトのスペースは、特殊文字です。
- 各混合コード・ページで使用できる 1 バイト文字は、次のように様々なカテゴリーに割り当てられます。

カテゴリー	各混合コード・ページにある有効なコード・ポイント
数字	x30-39
文字	x23-24、x40-5A、x61-7A、 xA6-DF (コード・ページ 932 および 942 の場合のみ A6-DF)
特殊文字	その他のすべての有効な 1 バイト文字のコード・ポイント

Unicode 環境での命名規則

Unicode データベースでは、ID はすべてマルチバイトの UTF-8 です。そのため、DB2 データベース・システムによって拡張文字セット (たとえば、アクセント付き文字、マルチバイト文字) での文字の使用が許可されている箇所では、 ID として任意の UCS-2 文字を使用することができます。

クライアントは、それぞれの環境でサポートされている任意の文字を入力することができます。その後、ID に入れられたすべての文字がデータベース・マネージャーによって UTF-8 へ変換されます。 Unicode データベースで ID に各国語の文字を指定するときには、以下の 2 つの点を考慮する必要があります。

- 非 ASCII 文字にはそれぞれ 2 から 4 バイトが必要。そのため、*n* バイトの ID は、ASCII 文字と非 ASCII 文字の比率に応じて、 *n*/4 文字から *n* 文字の範囲を保持できます。非 ASCII 文字 (アクセントなど) が 1 文字か 2 文字しか含まれない場合、その上限は *n* 文字に近くなりますが、完全に非 ASCII である ID の場合 (たとえば、日本語の場合)、使用できるのは *n*/4 から *n*/3 文字だけです。
- ID を別のクライアント環境から入力するのであれば、そのクライアントで使用できる文字の共通サブセットを使用して、 ID を定義しなければならない。例えば、Unicode データベースが Latin-1、アラビア語、および日本語環境からアクセスされる場合、すべての ID を実際に ASCII に限定する必要があります。

第 18 章 SQL および XML の制限値

以下の表は、SQL と XML の具体的な制限値を示しています。最も制限が厳しい場合に準拠することによって、容易に移植できるアプリケーション・プログラムを設計することができます。

表 55 は、制限値をバイト単位でリストしています。 ID の作成時に、アプリケーション・コード・ページからデータベース・コード・ページに変換された後に、これらの制限が課されます。また、データベースからの ID の検索時に、データベース・コード・ページからアプリケーション・コード・ページに変換された後にも、これらの制限が課されます。これらのいずれかのプロセスの中で ID 長さ限界を超えた場合には、切り捨てが生じるか、またはエラーが戻されます。

文字の長さ制限は、データベースのコード・ページとアプリケーションのコード・ページに応じて変わります。例えば、UTF-8 文字の幅は 1 から 4 バイトの範囲にわたるため、限界が 128 バイトの Unicode 表における ID の文字の長さ制限は、どんな文字が使用されるかによって 32 から 128 文字の範囲になります。名前の長さが、データベース・コード・ページへの変換後にこの表の限界を超えるような ID を作成しようとした場合には、エラーが戻されます。

ID 名を保管するアプリケーションは、コード・ページ変換が生じた後に ID のサイズが大きくなる可能性に対処できなければなりません。 ID がカタログから検索されるとき、それらはアプリケーション・コード・ページに変換されます。データベース・コード・ページからアプリケーション・コード・ページに変換されると、結果として ID は、表のバイト限界よりも長くなってしまう可能性があります。アプリケーションによって宣言されるホスト変数が、コード・ページ変換後に ID 全体を格納できない場合、それは切り捨てられます。それが受け入れられない場合には、ID 名全体を受け入れられるように、ホスト変数のサイズを大きくすることができます。

DB2 ユーティリティーのデータ検索、およびユーザー指定コード・ページへのデータの変換にも、同じ規則が適用されます。エクスポートなどの DB2 ユーティリティーがデータを検索し、(エクスポートの CODEPAGE 修飾子または DB2CODEPAGE レジストリー変数を使用して) ユーザー指定コード・ページへの変換を課す場合に、コード・ページ変換のために以下の表に明記されている限界を超えて ID が拡張すると、エラーが戻されるか、または ID が切り捨てられる可能性があります。

表 55. ID の長さの制限値

説明	バイト単位の最大値
別名	128
属性名	128
監査ポリシー名	128
許可名 (1 バイト文字のみ可)	128
バッファー・プール名	18
列名[2]	128

表 55. ID の長さの制限値 (続き)

説明	バイト単位の最大値
制約名	128
相関名	128
カーソル名	128
データ・パーティション名	128
データ・ソース列名	255
データ・ソース索引名	128
データ・ソース名	128
データ・ソース表名 (リモート表名)	128
データベース・パーティション・グループ名	128
データベース・パーティション名	128
イベント・モニター名	128
外部プログラム名	128
関数マッピング名	128
グループ名	128
ホスト ID[1]	255
データ・ソースのユーザー (リモート許可名) の ID	128
SQL プロシージャー中の ID (条件名、FOR ループ ID、ラベル、結果セット・ロケーター、ステートメント名、変数名)	128
索引名	128
索引拡張名	18
SPECIFICATION ONLY 指定の索引名	128
ラベル名	128
ネーム・スペースの URI (Uniform Resource Identifier)	1000
ニックネーム	128
パッケージ名	128
パッケージ・バージョン ID	64
パラメーター名	128
データ・ソースにアクセスするパスワード	32
プロシージャー名	128
ロール名	128
セーブポイント名	128
スキーマ名[2]	128
セキュリティー・ラベル・コンポーネント名	128
セキュリティー・ラベル名	128
セキュリティー・ポリシー名	128
シーケンス名	128
サーバー名 (データベース別名)	8
特定名	128
SQL 条件名	128

表 55. ID の長さの制限値 (続き)

説明	バイト単位の最大値
SQL 変数名	128
ステートメント名	128
表名	128
表スペース名	18
トランスフォーム・グループ名	18
トリガー名	128
トラステッド・コンテキスト名	128
タイプ・マッピング名	18
ユーザー定義関数名	128
ユーザー定義メソッド名	128
ユーザー定義タイプ名[2]	128
ビュー名	128
ラッパー名	128
XML エレメント名、属性名、接頭部名	1000
XML スキーマ・ロケーションの URI (Uniform Resource Identifier)	1000

注:

1. ホスト言語コンパイラーによっては、変数名に関してより厳しい制限がある場合があります。
2. SQLDA 構造が 30 バイトの列名を格納するように制限されている場合には、 18 バイトのユーザー定義タイプ名、およびユーザー定義タイプのための 8 バイトのスキーマ名。DESCRIBE ステートメントでは SQLDA が使用されるので、DESCRIBE ステートメントを使用して列またはユーザー定義タイプ名情報を取得する組み込み SQL アプリケーションは、これらの制限に従う必要があります。

表 56. 数値の制限値

説明	制限値
SMALLINT (短精度整数) の最小値	-32,768
SMALLINT (短精度整数) の最大値	+32,767
INTEGER (整数) の最小値	-2,147,483,648
INTEGER (整数) の最大値	+2,147,483,647
BIGINT (64 ビット整数) の最小値	-9,223,372,036,854,775,808
BIGINT (64 ビット整数) の最大値	+9,223,372,036,854,775,807
10 進数の精度の最大値	31
REAL 値の最大指数 (E_{max})	38
REAL (実数) の最小値	-3.402E+38
REAL (実数) の最大値	+3.402E+38
REAL 値の最小指数 (E_{min})	-37

表 56. 数値の制限値 (続き)

説明	制限値
REAL (実数) の正の最大値	+1.175E-37
REAL (実数) の負の最大値	-1.175E-37
DOUBLE 値の最大指数 (E_{max})	308
DOUBLE (倍精度) の最小値	-1.79769E+308
DOUBLE (倍精度) の最大値	+1.79769E+308
DOUBLE 値の最小指数 (E_{min})	-307
DOUBLE (倍精度) の正の最小値	+2.225E-307
DOUBLE (倍精度) の負の最大値	-2.225E-307
DECFLOAT(16) 値の最大指数 (E_{max})	384
最小 DECFLOAT(16) 値[1]	-9.999999999999999E+384
最大 DECFLOAT(16) 値	9.999999999999999E+384
DECFLOAT(16) 値の最小指数 (E_{min})	-383
最小の正 DECFLOAT(16) 値	1.000000000000000E-383
最大の負 DECFLOAT(16) 値	-1.000000000000000E-383
DECFLOAT(34) 値の最大指数 (E_{max})	6144
最小 DECFLOAT(34) 値[1]	-9.999999999999999999999999999999999E+6144
最大 DECFLOAT(34) 値	9.999999999999999999999999999999999E+6144
DECFLOAT(34) 値の最小指数 (E_{min})	-6143
最小の正 DECFLOAT(34) 値	1.000000000000000000000000000000000E-6143
最大の負 DECFLOAT(34) 値	-1.000000000000000000000000000000000E-6143

注:

1. 正規の 10 進浮動小数点数には限界があります。有効な 10 進浮動小数点値には、特殊値 NAN、-NAN、SNAN、-SNAN、INFINITY、および -INFINITY が含まれます。加えて、有効な値には非正規数が含まれます。

   非正規数は、調整された指数が E_{min} よりも小さいゼロ以外の数値です。非正規数では、指数の最小値は E_{min} - (*precision*-1) で、これを E_{tiny} といいます。ここで、*precision* は処理精度 (16 または 34) です。すなわち、非正規数では数値の範囲が、DECFLOAT(16) では 15 桁、DECFLOAT(34) では 33 桁、ゼロに近い側にそれぞれ広がります。非正規数は、その最大桁数が処理精度 (16 または 34) よりも小さいという点で正規の数と異なっています。 10 進浮動小数点数は、正規の数を表現する際と同じほど正確には非正規数を表現することができません。 DECFLOAT(34) の正の最小非正規数は $1x10^{-6176}$ で、これは 1 桁しかありません。一方、DECFLOAT(34) の正の最小正規数は $1.000000000000000000000000000000000x10^{-6143}$ であり、これは 34 桁あります。DECFLOAT(16) の正の最小非正規数は $1x10^{-398}$ です。

表 57. ストリングの制限値

説明	制限値
CHAR の最大長 (バイト単位)	254
VARCHAR の最大長 (バイト単位)	32 672
LONG VARCHAR の最大長 (バイト単位)	32 700
CLOB の最大長 (バイト単位)	2 147 483 647
シリアライズ XML の最大長 (バイト単位)	2 147 483 647
GRAPHIC の最大長 (2 バイト文字単位)	127
VARGRAPHIC の最大長 (2 バイト文字単位)	16,336
LONG VARGRAPHIC の最大長 (2 バイト文字単位)	16,350
DBCLOB の最大長 (2 バイト文字単位)	1,073,741,823
BLOB の最大長 (バイト単位)	2 147 483 647
文字定数の最大長	32 672
GRAPHIC 定数の最大長	16,336
連結後の文字ストリングの最大長	2 147 483 647
連結後の GRAPHIC ストリングの最大長	1,073,741,823
連結後のバイナリー・ストリングの最大長	2 147 483 647
16 進定数の最大桁数	32 672
実行時の構造化タイプ列オブジェクトの最大インスタンス (ギガバイト単位)	1
カタログ・コメントの最大サイズ (バイト単位)	254

表 58. XML 制限

説明	制限値
XML 文書の最大の深さ (レベル数)	125
XML スキーマ文書の最大サイズ (バイト単位)	31,457,280

表 59. 日付/時刻の制限値

説明	制限値
DATE (日付) の最小値	0001-01-01
DATE (日付) の最大値	9999-12-31
TIME (時刻) の最小値	00:00:00
TIME (時刻) の最大値	24:00:00
TIMESTAMP (タイム・スタンプ) の最小値	0001-01-01-00.00.00.000000
TIMESTAMP (タイム・スタンプ) の最大値	9999-12-31-24.00.00.000000

表 60. データベース・マネージャーの制限値

説明	制限値
表とビュー	
表の列の最大数 [7]	1012
ビューの列の最大数[1]	5000

表 60. データベース・マネージャーの制限値 (続き)

説明	制限値
ニックネームによって参照されるデータ・ソース表またはビューにある列の最大数	5000
分散キーの列の最大数[5]	500
すべてのオーバーヘッドを含む行の最大長[2] [7]	32,677
非パーティション表の、データベース・パーティション当たりの行の最大数	128 x 10^{10}
データ・パーティションの、データベース・パーティション当たりの行の最大数	128 x 10^{10}
REGULAR 表スペース内のデータベース・パーティション当たりの表の最大サイズ (ギガバイト単位)[3] [7]	512
LARGE DMS 表スペース内のデータベース・パーティション当たりの表の最大サイズ (ギガバイト単位)[7]	16,384
1 つの表のデータ・パーティションの最大数	32,767
表パーティション列の最大数	16
制約	
表に対する制約の最大数	ストレージ
ユニーク (UNIQUE) 制約の列の最大数 (ユニーク索引によってサポートされる)	64
ユニーク (UNIQUE) 制約の列の結合後の最大長 (ユニーク索引によってサポートされる) (バイト単位)[9]	8192
外部キーで参照される列の最大数	64
外部キーで参照される列の結合後の最大長 (バイト単位)[9]	8192
チェック制約の指定の最大長 (バイト単位)	65,535
トリガー	
トリガーのカスケード実行時の最大の深さ	16
ユーザー定義タイプ	
構造化タイプ内の属性の最大数	4082
索引	
表の索引の最大数	32,767 またはストレージのサイズによる
索引キーの列の最大数	64
すべてのオーバーヘッドを含む索引キーの最大長[7] [9]	*indexpagesize*/4
変数の索引キー部分の最大長 (バイト単位)[8]	1022 またはストレージのサイズによる
SMS 表スペース内のデータベース・パーティション当たりの索引の最大サイズ (ギガバイト単位)[7]	16,384
REGULAR DMS 表スペース内のデータベース・パーティション当たりの索引の最大サイズ (ギガバイト単位)[7]	512
LARGE DMS 表スペース内のデータベース・パーティション当たりの索引の最大サイズ (ギガバイト単位)[7]	16,384
データベース・パーティション当たりの XML データに対する索引の最大サイズ (テラバイト単位)	2

表 60. データベース・マネージャーの制限値 (続き)

説明	制限値
XML データに対する索引 に対する索引の変数の索引キー部分の最大長 (バイト単位)[7]	*pagesize*/4 - 207
SQL	
SQL ステートメントの最大全長 (バイト単位)	2,097,152
SQL ステートメントまたはビューで参照される表の最大数	ストレージ
SQL ステートメント中のホスト変数参照の最大数	32,767
ステートメント中の定数の最大数	ストレージ
選択リスト内のエレメントの最大数[7]	1012
WHERE または HAVING 節の述部の最大数	ストレージ
GROUP BY 節中の列の最大数[7]	1012
GROUP BY 節中の列の合計長の最大値 (バイト単位)[7]	32,677
ORDER BY 節中の列の最大数[7]	1012
ORDER BY 節中の列の合計長の最大値 (バイト単位)[7]	32,677
副照会ネストの最大レベル	ストレージ
単一のステートメント中の副照会の最大数	ストレージ
挿入操作内の値の最大数[7]	1012
単一の更新操作中の SET 節の最大数[7]	1012
ルーチン	
プロシージャー中のパラメーターの最大数	32,767
ユーザー定義関数のパラメーターの最大数	90
ルーチンのネスト・レベルの最大数	64
SQL パス内のスキーマの最大数	64
SQL パスの最大長 (バイト単位)	2048
アプリケーション	
プリコンパイル済みプログラム中のホスト変数宣言の最大数[3]	ストレージ
ホスト変数値の最大長 (バイト単位)	2 147 483 647
プログラムで宣言できるカーソルの最大数	ストレージ
作業単位で変更される行の最大数	ストレージ
一度にオープンできるカーソルの最大数	ストレージ
DB2 クライアント内のプロセス当たりの接続の最大数	512
トランザクション内の同時にオープンできる LOB ロケーターの最大数	32,100
SQLDA の最大サイズ (バイト単位)	ストレージ
準備されるステートメントの最大数	ストレージ
並行性	
サーバーの同時ユーザーの最大数[4]	64,000
インスタンス当たりの並行ユーザーの最大数	64,000
データベース当たりの並行アプリケーションの最大数	60,000
インスタンス当たりの並行使用可能なデータベースの最大数	256

表 60. データベース・マネージャーの制限値 (続き)

説明	制限値
モニター	
同時に起動できるイベント・モニターの最大数	32
セキュリティー	
タイプ・セットまたはツリーのセキュリティー・ラベル・コンポーネント内のエレメントの最大数	64
タイプ配列のセキュリティー・ラベル・コンポーネント内のエレメントの最大数	65,535
セキュリティー・ポリシー内のセキュリティー・ラベル・コンポーネントの最大数	16
データベース	
最大データベース・パーティション番号	999
表スペース	
データベース内の表スペースの最大数	32,768
SMS 表スペース内の表の最大数	65,534
REGULAR DMS 表スペースの最大サイズ (ギガバイト単位) [3] [7]	512
LARGE DMS 表スペースの最大サイズ (テラバイト単位)[3] [7]	16
一時 DMS 表スペースの最大サイズ (テラバイト単位) [37]	16
DMS 表スペース内の表オブジェクトの最大数[6]	51,000
自動ストレージ・データベース内のストレージ・パスの最大数	128
自動ストレージ・データベースに関連付けられているストレージ・パスの最大長 (バイト単位)	175
バッファー・プール	
32 ビット・リリースでのバッファー・プールの最大 NPAGES	1,048,576
64 ビット・リリースでのバッファー・プールの最大 NPAGES	2,147,483,647
すべてのバッファー・プール・スロットの最大合計サイズ (4K)	2,147,483,646

表 60. データベース・マネージャーの制限値 (続き)

説明	制限値
注: 1. この最大値は、 CREATE VIEW ステートメントで結合を使うことによって達成できます。そのようなビューからの選択は、選択リスト内のエレメントの最大数の制限値によって制限されます。 2. BLOB、CLOB、LONG VARCHAR、DBCLOB、および LONG VARGRAPHIC の列の実際のデータは、このカウントには含まれません。しかし、そのデータの格納場所についての情報のために、行内に一定のスペースが確保されます。 3. 示されている数値は、アーキテクチャー上の制限値であり、近似値です。実際の制限値はもっと小さい場合があります。 4. 実際の値はデータベース・マネージャーの構成パラメーター **max_connections** および **max_coordagents** によって制御されます。 5. これはアーキテクチャー上の制限値です。実際上の制限値としては、索引キーの列の最大数に関する制限値を使用する必要があります。 6. 表オブジェクトには、データ、索引、LONG VARCHAR または VARGRAPHIC 列、および LOB 列が組み入れられます。 表データと同じ表スペース中にある表オブジェクトは、制限値に対して余分のオブジェクトとしてカウントされません。しかし、表データとは異なる表スペースにある各表オブジェクトは、表オブジェクトがある表スペース中の表当たりの表オブジェクト・タイプの制限値に対して、実際に 1 個のオブジェクトとしてカウントされます。 7. ページ・サイズ固有の値については、表 61を参照してください。 8. これには、オーバーヘッドがすべて組み入れられており、最も長い索引キーによってのみ制限されます (バイト単位)。索引キー部分の数が増えるにつれて、各キー部分の最大長も増加します。 9. 索引オプションによっては、最大値がそれより小さな値になることもあります。	

表 61. データベース・マネージャーのページ・サイズ固有の制限値

説明	4K ページ・サイズ制限	8K ページ・サイズ制限	16K ページ・サイズ制限	32K ページ・サイズ制限
表の列の最大数	500	1012	1012	1012
すべてのオーバーヘッドを含む行の最大長	4005	8101	16,293	32,677
REGULAR 表スペース内のデータベース・パーティション当たりの表の最大サイズ (ギガバイト単位)	64	128	256	512
LARGE DMS 表スペース内のデータベース・パーティション当たりの表の最大サイズ (ギガバイト単位)	2048	4096	8192	16,384
すべてのオーバーヘッドを含む索引キーの最大長 (バイト単位)	1024	2048	4096	8192

表 61. データベース・マネージャーのページ・サイズ固有の制限値 (続き)

説明	4K ページ・サイズ制限	8K ページ・サイズ制限	16K ページ・サイズ制限	32K ページ・サイズ制限
SMS 表スペース内のデータベース・パーティション当たりの索引の最大サイズ (ギガバイト単位)	2048	4096	8192	16,384
REGULAR DMS 表スペース内のデータベース・パーティション当たりの索引の最大サイズ (ギガバイト単位)	64	128	256	512
LARGE DMS 表スペース内のデータベース・パーティション当たりの索引の最大サイズ (ギガバイト単位)	2048	4096	8192	16,384
データベース・パーティション当たりの XML データに対する索引の最大サイズ (テラバイト単位)	2	2	2	2
通常 DMS 表スペース中の最大サイズ (ギガバイト単位)	64	128	256	512
LARGE DMS 表スペースの最大サイズ (ギガバイト単位)	2048	4096	8192	16,384
一時 DMS 表スペースの最大サイズ (テラバイト単位)	2	4	8	16
選択リスト内のエレメントの最大数	500	1012	1012	1012
GROUP BY 節中の列の最大数	500	1012	1012	1012
GROUP BY 節中の列の合計長の最大値 (バイト単位)	4005	8101	16,293	32,677
ORDER BY 節中の列の最大数	500	1012	1012	1012
ORDER BY 節中の列の合計長の最大値 (バイト単位)	4005	8101	16,293	32,677
挿入操作内の値の最大数	500	1012	1012	1012
単一の更新操作中の SET 節の最大数	500	1012	1012	1012

第 19 章 レジストリー変数と環境変数

環境変数およびプロファイル・レジストリー

環境変数およびレジストリー変数は、データベース環境を制御します。

構成アシスタント (db2ca) を使って、レジストリー変数と構成パラメーターを構成することができます。

DB2 データベース・プロファイル・レジストリーが使用可能になる前は、 (例えば) Windows ワークステーションの環境変数を変更するためには環境変数を変更して再始動する必要がありました。いくつかの例外はありますが、現在、環境は DB2 プロファイル・レジストリーに保管されているレジストリー変数によって制御されます。 UNIX オペレーティング・システムで特定インスタンスに関するシステム管理 (SYSADM) 権限を持つユーザーは、そのインスタンスのレジストリー値を更新することができます。 Windows では、プロファイル・レジストリー変数を更新するには、ローカル管理者権限または SYSADM 権限が必要です。以下の条件があります。

- 拡張セキュリティーが使用可能になっている場合、SYSADM ユーザーは DB2ADMNS グループに属している必要があります。
- 拡張セキュリティーが使用可能になっていない場合、Windows レジストリーで適切な許可が付与されていれば、SYSADM ユーザーは更新を行えます。

再始動せずにレジストリー変数を更新するには、 db2set コマンドを使用します。この情報は、即時にプロファイル・レジストリーの中に保管されます。ただし変更内容は、現在実行中の DB2 アプリケーションまたはユーザーに影響はありません。DB2 レジストリーは、変更を行った後に開始した DB2 サーバー・インスタンスと DB2 アプリケーションに更新情報を適用します。

注: DB2 環境変数 **DB2INSTANCE** および **DB2NODE** は、 DB2 プロファイル・レジストリーに保管されない場合があります。一部のオペレーティング・システムでは、これらの環境変数を更新するために set コマンドを使用する必要があります。変更は、次回システムを再始動するまで有効です。 Linux および UNIX プラットフォームでは、set コマンドの代わりに export コマンドを使用できます。

プロファイル・レジストリーを使用することによって、環境変数の中央制御が可能になります。さまざまなレベルのサポートが、さまざまなプロファイルを通して提供されるようになっています。環境変数のリモートでの管理も、DB2 Administration Server を使用すれば可能です。

以下の 4 つのプロファイル・レジストリーがあります。

- DB2 インスタンス・レベル・プロファイル・レジストリー。 DB2 環境変数の大多数は、このレジストリーの中に置かれます。特定のインスタンスについての環境変数の設定値が、このレジストリーに保持されます。このレベルに定義された値は、グローバル・レベルの設定値をオーバーライドします。

- DB2 グローバル・レベル・プロファイル・レジストリー。特定のインスタンスごとに 1 つの環境変数が設定されるのではない場合、このレジストリーが使用されます。このレジストリーは、特定の DB2 ESE のコピー (そのインストール・パスには 1 つのグローバル・レベルのプロファイルが存在する) に関するすべてのインスタンスに対して可視となります。
- DB2 インスタンス・ノード・レベル・プロファイル・レジストリー。このレジストリー・レベルには、パーティション・データベース環境のデータベース・パーティションに固有の変数設定値が含まれます。このレベルで定義された値は、インスタンス・レベルおよびグローバル・レベルの設定値をオーバーライドします。
- DB2 インスタンス・プロファイル・レジストリー。このレジストリーには、現行コピーと関連付けられたすべてのインスタンス名のリストが含まれます。インストールにはそれぞれ固有のリストがあります。 db2ilist を実行すれば、システムで使用できるすべてのインスタンスがリストされます。

DB2 は、レジストリー値と環境変数をチェックし、それらを以下の順序で解決することによって、操作環境を構成します。

1. set コマンドを使用して設定された環境変数。 (あるいは、UNIX プラットフォームでは export コマンド。)
2. インスタンス・ノード・レベル・プロファイルを使用して設定されたレジストリー値 (コマンド db2set -i <instance name> <nodenum> を使用)。
3. インスタンス・レベル・プロファイルを使用して設定されたレジストリー値 (コマンド db2set -i を使用)。
4. グローバル・レベル・プロファイルを使用して設定されたレジストリー値 (コマンド db2set -g を使用)。

インスタンス・レベル・プロファイル・レジストリー

パーティション・データベース環境で作業を行っている場合、UNIX と Windows では、いくつかの違いがあります。これらの違いについて、以下に例を示します。

「red」、「white」、および「blue」で識別される 3 つの物理データベース・パーティションを持つ、パーティション・データベース環境があると想定します。 UNIX プラットフォームでは、インスタンス所有者がいずれかのデータベース・パーティションから次を実行した場合、

```
db2set -i FOO=BAR
```

または

```
db2set FOO=BAR        (「-i」が暗黙指定されます)
```

FOO の値は現行インスタンスのすべてのノード (「red」、「white」、および「blue」) に対して有効になります。

UNIX プラットフォームでは、インスタンス・レベル・プロファイル・レジストリーは、 sqllib ディレクトリー内のテキスト・ファイルに保管されます。パーティション・データベース環境では、sqllib ディレクトリーはすべての物理データベース・パーティションによって共用されているファイル・システムにあります。

Windows プラットフォームでは、ユーザーが「red」から同じコマンドを実行した場合、 FOO の値は現行インスタンスの「red」でのみ可視になります。 DB2 データベース・マネージャーはインスタンス・レベル・プロファイル・レジストリーを Windows レジストリーに保管します。物理データベース・パーティション間に共用はありません。レジストリー変数をすべての物理コンピューターに設定するには、以下のように「rah」コマンドを使用します。

```
rah db2set -i FOO=BAR
```

rah はリモートで db2set コマンドを「red」、「white」、および「blue」上で実行します。

DB2REMOTEPREG を使用して、インスタンスを所有していないコンピューター上のレジストリー変数を構成することにより、インスタンスを所有しているコンピューター上のレジストリー変数を参照することができます。これにより、インスタンスを所有しているコンピューター上のレジストリー変数が、インスタンス内のすべてのコンピューター間で共用される環境が効果的に作成されます。

上に示された例を使用し、「red」が所有コンピューターと想定すると、次を行うことで、「red」上のレジストリー変数を共有するために、**DB2REMOTEPREG** が「white」および「blue」コンピューターに設定されます。

```
(red の場合) 何もしません
(white および blue の場合) db2set DB2REMOTEPREG=¥¥red
```

DB2REMOTEPREG の設定値は、設定後に変更してはいけません。

以下に REMOTEPREG がどのように動作するかを示します。

DB2 データベース・マネージャーが Windows でレジストリー変数を読み取る場合、最初に **DB2REMOTEPREG** 値を読み取ります。 **DB2REMOTEPREG** が設定されている場合、**DB2REMOTEPREG** 変数に指定されているコンピューター名のリモート・コンピューター上のレジストリーをオープンします。その後のレジストリー変数の読み取りおよび更新は、指定されたリモート・コンピューターにリダイレクトされます。

リモート・レジストリーへのアクセスには、ターゲット・コンピューターで Remote Registry Service が実行されている必要があります。また、ユーザーのログオン・アカウントおよびすべての DB2 サービス・ログオン・アカウントに、リモート・レジストリーへの十分なアクセス権限が必要です。そのため、**DB2REMOTEPREG** を使用するには、Windows ドメイン環境で操作を行う必要があります。これによって、必要なレジストリー・アクセスが、ドメイン・アカウントに付与されます。

Microsoft Cluster Server (MSCS) には考慮事項があります。**DB2REMOTEPREG** を MSCS 環境では使用できません。すべてのコンピューターが同じ MSCS クラスターに属する MSCS 構成で稼働している場合、レジストリー変数はクラスター・レジストリー内に保持されます。そのため、それらはすでに同じ MSCS クラスター内のすべてのコンピューター間で共用されており、この場合は **DB2REMOTEPREG** を使用する必要はありません。

データベース・パーティションが複数の MSCS クラスターにまたがって存在するような、マルチパーティション・フェイルオーバー環境で稼働している場合、インス

タンスを所有しているコンピューターのレジストリー変数はこのクラスター・レジストリーにあるため、インスタンスを所有しているコンピューターを指すのに、**DB2REMOTEPREG** は使用できません。

レジストリー変数と環境変数の宣言、表示、変更、リセット、および削除

特定のレジストリー変数はすべて、DB2 データベース・プロファイル・レジストリーで定義することをぜひお勧めします。 DB2 変数がレジストリー以外で設定されていれば、その変数をリモート管理することはできません。変数値を有効にするには、ワークステーションをリブートしなければなりません。

db2set コマンドは、レジストリー変数および環境変数のローカルでの宣言をサポートします。

コマンドに対するヘルプ情報を表示するには、以下を使用します。

```
db2set -?
```

サポートされるすべてのレジストリー変数の完全なセットをリストするには、以下を使用します。

```
db2set -lr
```

現在のインスタンスまたはデフォルトのインスタンスに定義されているすべてのレジストリー変数をリストするには、以下を使用します。

```
db2set
```

プロファイル・レジストリーに定義されているすべてのレジストリー変数をリストするには、以下を使用します。

```
db2set -all
```

現在のインスタンスまたはデフォルトのインスタンス内のレジストリー変数の値を表示するには、以下を使用します。

```
db2set registry_variable_name
```

全レベルのレジストリー変数の値を表示するには、以下を使用します。

```
db2set registry_variable_name -all
```

現在のインスタンスまたはデフォルトのインスタンス内のレジストリー変数の値を変更するには、以下を使用します。

```
db2set registry_variable_name=new_value
```

インスタンス内のすべてのデータベースについて、レジストリー変数のデフォルトを変更するには、以下を使用します。

```
db2set registry_variable_name=new_value
    -i instance_name
```

インスタンス内の特定のデータベース・パーティションのレジストリー変数のデフォルトを変更するには、以下を使用します。

```
db2set registry_variable_name=new_value
    -i instance_name database_partition_number
```

このシステム内の特定のインストールに関連したすべてのインスタンス内のレジストリー変数のデフォルトを変更するには、以下を使用します。

```
db2set registry_variable_name=new_value -g
```

DB2_WORKLOAD などの集約レジストリー変数を使用して SAP 環境用のレジストリー変数を構成する場合、以下を使用して変数を設定できます。

```
db2set DB2_WORKLOAD=SAP
```

Lightweight Directory Access Protocol (LDAP) を使用している場合、LDAP で以下を使用してレジストリー変数を設定できます。

- LDAP でレジストリー変数をユーザー・レベルで設定するには、以下を使用します。

  ```
  db2set -ul
  ```

- LDAP でレジストリー変数をグローバル・レベルで設定するには、以下を使用します。

  ```
  db2set -gl user_name
  ```

LDAP 環境で実行している場合、その有効範囲がディレクトリー・パーティションまたは Windows ドメインに属するすべてのサーバーとすべてのユーザーに渡る、グローバルなものとなるように DB2 レジストリー変数値を設定することができます。現在のところ、LDAP グローバル・レベルで設定できる DB2 レジストリー変数は、 **DB2LDAP_KEEP_CONNECTION** と **DB2LDAP_SEARCH_SCOPE** の 2 つのみです。

例えば、LDAP のグローバル・レベルで検索の有効範囲の値を設定するには、以下を使用します。

```
db2set -gl db2ldap_search_scope = value
```

value には ″local″、″domain″、または ″global″ を指定できます。

注:

1. 複数のユーザーが db2set コマンドを使用して DB2 profile.env ファイルを同時に、またはほとんど同じ瞬間に更新する場合、profile.env ファイルのサイズは小さくなってゼロになります。 db2set -all の出力も矛盾する値を表示します。
2. 同一の DB2 ESE のインストール単位に関係しているすべてのインスタンスに適用される DB2 レジストリー変数の設定に使用される -g オプションと、特に LDAP グローバル・レベルで使用される -gl オプションの間には相違点があります。
3. ユーザー・レベルのレジストリー変数は、 LDAP 環境で Windows を実行する場合にのみサポートされます。
4. ユーザー・レベルの変数設定値には、ユーザー固有の変数設定値が含まれます。ユーザー・レベルでのすべての変更点は、LDAP ディレクトリーに書き込まれます。
5. パラメーター ″-i″、″-g″、″-gl″、″-ul″ は、同じコマンド内では同時に使用できません。
6. 一部の変数は、デフォルトで常にグローバル・レベル・プロファイルに設定されます (グローバルとは、同一 DB2 コピー上で稼働するすべてのインスタンスの

間で変数が共有されるという意味です)。そのようなパラメーターをインスタンスまたはデータベース・パーティション・レベル・プロファイルで設定することはできません。例えば、**DB2SYSTEM** および **DB2INSTDEF** があります。

7. UNIX では、インスタンスのレジストリー値を変更するためには、システム管理 (SYSADM) 権限を持っていなければなりません。グローバル・レベル・レジストリー中のパラメーターを変更できるのは、 root 権限を持つユーザーのみです。

1 つのインスタンスの 1 つのレジストリー変数をリセットして、グローバル・プロファイル・レジストリーの中のデフォルトに戻すには、以下を使用します。

```
db2set -r registry_variable_name
```

1 つのインスタンス内のデータベース・パーティションの 1 つのレジストリー変数をリセットして、グローバル・プロファイル・レジストリーの中のデフォルトに戻すには、以下を使用します。

```
db2set -r registry_variable_name database_partition_number
```

指定レベルの変数値を削除するには、同じコマンド構文を使って変数を設定できますが、変数値には何も指定しません。例えば、データベース・パーティション・レベルの変数設定値を削除するには、次のように入力します。

```
db2set registry_variable_name= -i instance_name
   database_partition_number
```

変数値を削除してその使用を制限する場合、その値が上位のプロファイル・レベルで定義されていれば、次のように入力します。

```
db2set registry_variable_name= -null -r instance_name
```

このコマンドを使用すると、指定するパラメーターの設定値が削除され、上位レベルのプロファイルでこの変数の値 (この場合は DB2 グローバル・レベル・プロファイル) を変更できなくなります。ただし、指定する変数を下位レベルのプロファイル (この場合は DB2 データベース・パーティション・レベル・プロファイル) で引き続き設定することはできます。

Windows 上の環境変数の設定

Windows オペレーティング・システムには 1 つのシステム環境変数 **DB2INSTANCE** があり、この値はプロファイル・レジストリーの外部でしか設定できません。ただし、**DB2INSTANCE** の設定は必須ではありません。DB2 プロファイル・レジストリー変数 **DB2INSTDEF** をグローバル・レベル・プロファイルで設定すれば、**DB2INSTANCE** が定義されていない場合に使用するインスタンス名を指定することができます。

Windows 上の DB2 Enterprise Server Edition サーバーには、プロファイル・レジストリーの外部でしか設定できない 2 つのシステム環境変数 **DB2INSTANCE** および **DB2NODE** があります。 **DB2INSTANCE** の設定は必須ではありません。 DB2 プロファイル・レジストリー変数 **DB2INSTDEF** をグローバル・レベル・プロファイルで設定すれば、**DB2INSTANCE** が定義されていない場合に使用するインスタンス名を指定することができます。

DB2NODE 環境変数は、コンピューター内のターゲット論理ノードへの要求を経路指定するために使用します。この環境変数は、DB2 プロファイル・レジストリーの中ではなく、アプリケーションまたはコマンドが発行されるセッションで設定しなければなりません。この変数を指定しない場合、ターゲット論理ノードはデフォルトとして、コンピューター上でゼロ (0) に定義された論理ノードに設定されます。

環境変数の設定値を判別するためには、echo コマンドを使用します。例えば、**DB2PATH** 環境変数の値を検査するには、以下のように入力します。

```
echo %db2path%
```

DB2 環境変数 **DB2INSTANCE** および **DB2NODE** を次のように設定できます (以下では **DB2INSTANCE** について説明します)。

- 「マイ コンピュータ」を右クリックして「プロパティ」を選択します。
- 「詳細設定」タブを選択し、「環境変数」をクリックします。そして、次のようにします。
  1. **DB2INSTANCE** 変数が存在しない場合は、次のようにします。
     a. 「新規」をクリックします。
     b. 「変数名」フィールドに **DB2INSTANCE** と入力します。
     c. 「変数値」フィールドにインスタンス名 (例えば db2inst) を入力します。
  2. **DB2INSTANCE** 変数が既に存在している場合は、以下のようにして新しい値を追加します。
     a. **DB2INSTANCE** 環境変数を選択します。
     b. 「値」フィールドをインスタンス名 (例えば、db2inst) に変更します。
  3. これらの変更を有効にするには、ご使用のシステムを再起動してください。

注: 環境変数 **DB2INSTANCE** もセッション (プロセス) レベルで設定できます。例えば、TEST という 2 番目の DB2 インスタンスを開始する場合、コマンド・ウィンドウで以下のコマンドを発行します。

```
set DB2INSTANCE=TEST
db2start
```

C シェルのコマンド・ウィンドウから、以下のコマンドを発行します。

```
setenv DB2INSTANCE TEST
```

プロファイル・レジストリーは、以下のように配置されます。

- Windows オペレーティング・システム・レジストリーの中の DB2 インスタンス・レベル・プロファイル・レジストリーは、以下のパスを使用して配置されます。

  ```
  ¥HKEY_LOCAL_computer¥SOFTWARE¥IBM¥DB2¥PROFILES¥instance_name
  ```

  注: *instance_name* は、DB2 インスタンスの名前です。
- Windows レジストリーの中の DB2 グローバル・レベル・プロファイル・レジストリーは、以下のパスを使用して配置されます。

  ```
  ¥HKEY_LOCAL_computer¥SOFTWARE¥IBM¥DB2¥GLOBAL_PROFILE
  ```
- Windows レジストリーの中の DB2 インスタンス・ノード・レベル・プロファイル・レジストリーは、以下のパスを使用して配置されます。

...¥SOFTWARE¥IBM¥DB2¥PROFILES¥*instance_name*¥NODES¥*node_number*

注: *instance_name* と *node_number* は、作業しているデータベース・パーティションに固有のものです。

- DB2 インスタンス・プロファイル・レジストリーは必要ありません。システム内の各 DB2 インスタンスごとに、1 つのキーが以下のパスに作成されます。

  ¥HKEY_LOCAL_computer¥SOFTWARE¥IBM¥DB2¥PROFILES¥*instance_name*

  インスタンスのリストを表示するには、PROFILES キーの下にあるキーを数えます。

Linux および UNIX オペレーティング・システム上の環境変数の設定

UNIX オペレーティング・システムでは、システム環境変数 **DB2INSTANCE** を設定しなければなりません。

db2profile (Bourne シェルまたは Korn シェルの場合) および db2cshrc (C シェルの場合) というスクリプトが、データベース環境のセットアップを援助するために例として提供されています。これらのファイルは insthome/sqllib の中にあります (ただし、 `insthome` はインスタンス所有者のホーム・ディレクトリーです)。

これらのスクリプトには、以下のためのステートメントが入っています。

- 以下のディレクトリーにユーザーのパスを更新します。
  - insthome/sqllib/bin
  - insthome/sqllib/adm
  - insthome/sqllib/misc
- 実行のために、**DB2INSTANCE** をデフォルトのローカル `instance_name` に設定します。

注: PATH および **DB2INSTANCE** を除いて、サポートされる変数は DB2 プロファイル・レジストリーで設定しなければなりません。DB2 データベース・マネージャーで設定されない変数を設定するには、スクリプト・ファイル userprofile および usercshrc で定義する必要があります。

インスタンス所有者または SYSADM ユーザーは、インスタンスのすべてのユーザーのためにこれらのスクリプトをカスタマイズすることができます。あるいは、ユーザーがスクリプトをコピーしてカスタマイズした後、スクリプトを直接呼び出すか、または自分の .profile または .login ファイルに追加することができます。

現行セッションについての環境変数を変更するには、以下のようなコマンドを出します。

- Korn シェルの場合、

  ```
  DB2INSTANCE=<inst1>
  export DB2INSTANCE
  ```

- Bourne シェルの場合、

  ```
  export DB2INSTANCE=<inst1>
  ```

- C シェルの場合、

```
setenv DB2INSTANCE <inst1>
```

DB2 プロファイル・レジストリーが正しく管理されるようにするためには、 UNIX オペレーティング・システム上で、以下のファイル所有権規則に従わなければなりません。

- DB2 インスタンス・レベル・プロファイル・レジストリー・ファイルは、以下のもとに配置されます。

  ```
  INSTHOME/sqllib/profile.env
  ```

  このファイルのアクセス許可と所有権は、以下のようにする必要があります。

  ```
  -rw-rw-r-- <db2inst1> <db2iadm1> profile.env
  ```

  ここで、<db2inst1> はインスタンス所有者、<db2iadm1> はインスタンス所有者のグループです。

  INSTHOME は、インスタンス所有者のホーム・パスです。
- DB2 グローバル・レベル・プロファイル・レジストリーは、以下のもとに配置されます。

  ```
  <installation path>/default.env
  ```

  すべての Linux および UNIX プラットフォームで、この位置にあります。

  このファイルのアクセス許可と所有権は、以下のようにする必要があります。

  ```
  -rw-rw-r-- root <group> default.env
  ```

  ここで <group> は、グループ ID 0 のグループ名です。例えば、AIX 上ではこれは「system」です。

  ルート・インストールのグローバル・レジストリー変数を変更するには、root 権限でログオンする必要があります。
- DB2 インスタンス・ノード・レベル・プロファイル・レジストリーは、以下のもとに配置されます。

  ```
  INSTHOME/sqllib/nodes/<node_number>.env
  ```

  ディレクトリーとこのファイルのアクセス許可と所有権は、以下のようにする必要があります。

  ```
  drwxrwsr-w <Instance_Owner> <Instance_Owner_Group> nodes

  -rw-rw-r-- <Instance_Owner> <Instance_Owner_Group> <node_number>.env
  ```

  INSTHOME は、インスタンス所有者のホーム・パスです。
- DB2 インスタンス・プロファイル・レジストリーは、以下のもとに配置されます。

  ```
  <installation path>/profiles.reg
  ```

  すべての Linux および UNIX プラットフォームで、この位置にあります。

  このファイルのアクセス許可と所有権は、以下のようにする必要があります。

  ```
  -rw-r--r-- root system profiles.reg
  ```

現行インスタンス環境変数の設定

インスタンスのデータベース・マネージャーを開始または停止するためのコマンドを実行すると、 DB2 はそのコマンドを現行インスタンスに適用します。DB2 は以下のように現行インスタンスを判別します。

- 現行セッションに **DB2INSTANCE** 環境変数が設定されていれば、その値が現行インスタンスです。**DB2INSTANCE** 環境変数を設定するには、次のように入力します。

  ```
  set db2instance=<new_instance_name>
  ```

- **DB2INSTANCE** が現行セッション用に設定されていない場合、DB2 データベース・マネージャーはシステム環境変数から **DB2INSTANCE** 環境変数用の設定値を使用します。Windows では、システム環境変数はシステム環境レジストリー内で設定されます。
- **DB2INSTANCE** がまったく設定されていなければ、DB2 データベース・マネージャーはレジストリー変数 **DB2INSTDEF** を使用します。

  DB2INSTDEF レジストリー変数をレジストリーのグローバル・レベルで設定するには、次のように入力します。

  ```
  db2set db2instdef=<new_instance_name> -g
  ```

現行セッションに適用されるインスタンスを判別するには、次のように入力します。

```
db2 get instance
```

集約レジストリー変数

集約レジストリー変数を使用すると、いくつかのレジストリー変数を、別のレジストリー変数名で識別される 1 つの構成としてグループ化することができます。グループの一部である各レジストリー変数には、事前定義された設定があります。集約レジストリー変数には 1 つの値が与えられます。この値は、複数のレジストリー変数の宣言として解釈されます。

集約レジストリー変数には、幅広い運用目的のためにレジストリー構成を簡単にするという意図があります。

有効な集約レジストリー変数は **DB2_WORKLOAD** だけです。

この変数の有効な値は以下のとおりです。

- SAP
- 1C
- TPM

集約レジストリー変数を介して暗黙的に構成されるレジストリー変数はすべて、明示的に定義することも可能です。パフォーマンスまたは診断のテストを行う際には、集約レジストリー変数の使用によってすでに値が与えられているレジストリー変数を明示的に設定すると便利です。集約によって暗黙的に構成される変数を明示的に設定することを、変数を「オーバーライドする」と言います。

明示的に設定したレジストリー変数を集約レジストリー変数を使用して変更しようとする場合、警告が出されて明示的に設定された値が保持されます。この警告は、明示的な値が維持されることを知らせます。最初に集約レジストリー変数が使用され、次いで明示的なレジストリー変数を指定すると、警告は出ません。

集約レジストリー変数の設定によって構成されるレジストリー変数は、どれも表示されません。ただし、それぞれの変数ごとに明示的に要求した場合は表示されます。集約レジストリー変数を照会すると、表示を要求された変数に割り当てられた値だけが表示されます。ほとんどのユーザーは、個々の変数の値に注意する必要はありません。

以下の例は、集約レジストリー変数の使用と、レジストリー変数の明示的な設定との間の相互作用を示しています。例えば、**DB2_WORKLOAD** 集約レジストリー変数を SAP に設定し、 **DB2_SKIPDELETED** レジストリー変数を NO にオーバーライドしたとします。`db2set` を入力すると、以下の結果を受け取ります。

```
DB2_WORKLOAD=SAP
DB2_SKIPDELETED=NO
```

別の状況で、**DB2ENVLIST** を設定し、 **DB2_WORKLOAD** 集約レジストリー変数を SAP に設定し、 **DB2_SKIPDELETED** レジストリー変数を NO にオーバーライドしたとします。(ここで、**DB2_SKIPDELETED** レジストリー変数は、SAP 環境を形成するグループの一部であると想定します。) さらに、集約レジストリー変数の設定によって自動的に構成されたレジストリー変数は、値の隣の大括弧の中に集約の名前が表示されます。 **DB2_SKIPDELETED** レジストリー変数は「`NO`」の値を表示し、その値の隣に「`[O]`」が表示されます。

DB2_WORKLOAD に関連した構成が必要なくなった場合、次のコマンドを使用して集約レジストリー変数の値を削除することによって、グループ内の各レジストリー変数の暗黙的な値を使用不可にすることができます。

```
db2set DB2_WORKLOAD=
```

DB2_WORKLOAD 集約レジストリー変数の値を削除したら、データベースを再始動します。データベースが再始動した後、集約レジストリー変数によって暗黙的に構成されたレジストリー変数は有効ではなくなります。集約レジストリー変数の値を削除する方法は、個々のレジストリー変数の削除の場合と同じです。

集約レジストリー変数の値を削除しても、明示的に設定されたレジストリー変数の値は削除されません。レジストリー変数が、使用不可にされているグループ定義のメンバーであることは、問題にはなりません。レジストリー変数の明示的な設定は維持されます。

DB2_WORKLOAD 集約レジストリー変数のメンバーである各レジストリー変数の値を確認する必要が生じる場合があります。例えば、**DB2_WORKLOAD** を SAP に構成した場合に使用される可能性のある値を確認することができます。**DB2_WORKLOAD=**SAP の場合に使用される可能性のある値を確認するには、`db2set -gd DB2_WORKLOAD=SAP` を実行します。グループに含まれているレジストリー変数が明示的に定義されていなかったと想定した場合、以下のレジストリー変数とそのデフォルト値のリストが戻されます。

```
db2set -gd DB2_WORKLOAD=SAP
DB2_RR_TO_RS=YES
DB2_REDUCED_OPTIMIZATION=4,INDEX,JOIN,NO_TQ_FACT,NO_HSJN_BUILD_FACT,
  STARJN_CARD_SKEW,NO_SORT_MGJOIN,CART OFF
DB2_MINIMIZE_LISTPREFETCH=YES
DB2_INLIST_TO_NLJN=YES
DB2_ANTIJOIN=EXTEND
DB2_IMPLICIT_UNICODE=YES
DB2_EVALUNCOMMITTED=YES
DB2_INTERESTING_KEYS=YES
DB2_SKIPINSERTED=YES
DB2_MDC_ROLLOUT=DEFER
DB2_OBJECT_TABLE_ENTRIES=65532
DB2NOTIFYVERBOSE=YES
DB2_OPTPROFILE=YES
DB2_VIEW_REOPT_VALUES=YES
DB2_OPT_MAX_TEMP_SIZE=10240
DB2_TRUNCATE_REUSESTORAGE=IMPORT
DB2COMM=TCPIP
DB2_SET_MAX_CONTAINER_SIZE=20000000000
```

DB2 レジストリー変数と環境変数

DB2 では多くのレジストリー変数と環境変数が用意されており、稼働状態にするにはこれらについて知っている必要があります。

サポートされているすべてのレジストリー変数を表示するには、以下のコマンドを実行してください。

```
db2set -lr
```

現行またはデフォルト・インスタンスの変数の値を変更するには、以下のコマンドを実行してください。

```
db2set registry_variable_name=new_value
```

DB2 環境変数 **DB2INSTANCE**、**DB2NODE**、**DB2PATH**、および **DB2INSTPROF** が DB2 プロファイル・レジストリーに保管されるかどうかは、オペレーティング・システムによって異なります。これらの環境変数を更新するには、`set` コマンドを使用します。変更は、ローカル (現行) コマンド・プロンプトのみで有効で、次回システムをリブートするまで有効です。 UNIX プラットフォームでは、`export` コマンドを `set` コマンドの代わりに使用できます。

変更されたレジストリー変数の値は、 db2start コマンドを実行する前に設定しなければなりません。

注: レジストリー変数が引数としてブール値を必要とする場合、値の YES、1、および ON はすべて同等であり、値の NO、0、および OFF もすべて同等です。どの変数についても、適切な同等値のいずれかを指定することができます。

以下の表は、カテゴリーごとにすべてのレジストリー変数をリストしています。

表 62. レジストリー変数と環境変数の要約

変数のカテゴリー	レジストリー変数または環境変数の名前
一般	DB2ACCOUNT DB2BIDI DB2_CAPTURE_LOCKTIMEOUT DB2CODEPAGE DB2_COLLECT_TS_REC_INFO DB2_CONNRETRIES_INTERVAL DB2CONSOLECP DB2COUNTRY DB2DBDFT DB2DBMSADDR DB2DISCOVERYTIME DB2FFDC DB2FODC DB2_FORCE_APP_ON_MAX_LOG DB2GRAPHICUNICODESERVER DB2INCLUDE DB2INSTDEF DB2INSTOWNER DB2_LIC_STAT_SIZE DB2LOCALE DB2_MAX_CLIENT_CONNRETRIES DB2_OBJECT_TABLE_ENTRIES DB2_SYSTEM_MONITOR_SETTINGS DB2TERRITORY DB2_VIEW_REOPT_VALUES

表 62. レジストリー変数と環境変数の要約 (続き)

変数のカテゴリー	レジストリー変数または環境変数の名前
システム環境	DB2_ALTERNATE_GROUP_LOOKUP DB2_CLP_EDITOR DB2_CLP_HISTSIZE DB2CONNECT_IN_APP_PROCESS DB2_COPY_NAME DB2DBMSADDR DB2_DIAGPATH DB2DOMAINLIST DB2ENVLIST DB2INSTANCE DB2INSTPROF DB2LDAPSecurityConfig DB2LIBPATH DB2LOGINRESTRICTIONS DB2NODE DB2OPTIONS DB2_PARALLEL_IO DB2PATH DB2PROCESSORS DB2RCMD_LEGACY_MODE DB2SYSTEM DB2_UPDDBCFG_SINGLE_DBPARTITION DB2_USE_PAGE_CONTAINER_TAG DB2_WORKLOAD
通信	DB2CHECKCLIENTINTERVAL DB2COMM DB2FCMCOMM DB2_FORCE_NLS_CACHE DB2RSHCMD DB2RSHTIMEOUT DB2SORCVBUF DB2SOSNDBUF DB2TCP_CLIENT_CONTIMEOUT DB2TCP_CLIENT_RCVTIMEOUT DB2TCPCONNMGRS
コマンド行	DB2BQTIME DB2BQTRY DB2_CLPPROMPT DB2IQTIME DB2RQTIME
パーティション・データベース環境	DB2CHGPWD_EEE DB2_NUM_FAILOVER_NODES DB2_PARTITIONEDLOAD_DEFAULT DB2PORTRANGE

表 62. レジストリー変数と環境変数の要約 (続き)

変数のカテゴリー	レジストリー変数または環境変数の名前
照会コンパイラー	DB2_ANTIJOIN DB2_INLIST_TO_NLJN DB2_LIKE_VARCHAR DB2_MINIMIZE_LISTPREFETCH DB2_NEW_CORR_SQ_FF DB2_OPT_MAX_TEMP_SIZE DB2_REDUCED_OPTIMIZATION DB2_SELECTIVITY DB2_SQLROUTINE_PREPOPTS

表 62. レジストリー変数と環境変数の要約 (続き)

変数のカテゴリー	レジストリー変数または環境変数の名前
パフォーマンス	DB2_ALLOCATION_SIZE DB2_APM_PERFORMANCE DB2ASSUMEUPDATE DB2_ASYNC_IO_MAXFILOP DB2_AVOID_PREFETCH DB2BPVARS DB2CHKPTR DB2CHKSQLDA DB2_EVALUNCOMMITTED DB2_EXTENDED_IN_TO_JOIN DB2_EXTENDED_IO_FEATURES DB2_EXTENDED_OPTIMIZATION DB2_IO_PRIORITY_SETTING DB2_IO_PRIORITY_SETTING DB2_KEEP_AS_AND_DMS_CONTAINERS_OPEN DB2_KEEPTABLELOCK DB2_LARGE_PAGE_MEM DB2_LOGGER_NON_BUFFERED_IO DB2MAXFSCRSEARCH DB2_MAX_INACT_STMTS DB2_MAX_NON_TABLE_LOCKS DB2_MDC_ROLLOUT DB2MEMDISCLAIM DB2MEMMAXFREE DB2_MEM_TUNING_RANGE DB2_MMAP_READ DB2_MMAP_WRITE DB2_NO_FORK_CHECK DB2NTMEMSIZE DB2NTNOCACHE DB2NTPRICLASS DB2NTWORKSET DB2_OVERRIDE_BPF DB2_PINNED_BP DB2PRIORITIES DB2_RESOURCE_POLICY DB2_SET_MAX_CONTAINER_SIZE DB2_SKIPDELETED DB2_SKIPINSERTED DB2_SMS_TRUNC_TMPTABLE_THRESH DB2_SORT_AFTER_TQ DB2_SELUDI_COMM_BUFFER DB2_TRUSTED_BINDIN DB2_USE_ALTERNATE_PAGE_CLEANING

表 62. レジストリー変数と環境変数の要約 (続き)

変数のカテゴリー	レジストリー変数または環境変数の名前
その他	DB2ADMINSERVER DB2CLIINIPATH DB2_COMMIT_ON_EXIT DB2_CREATE_DB_ON_PATH DB2DEFPREP DB2_DISABLE_FLUSH_LOG DB2_DISPATCHER_PEEKTIMEOUT DB2_DJ_INI DB2DMNBCKCTLR DB2_DOCHOST DB2_DOCPORT DB2_ENABLE_AUTOCONFIG_DEFAULT DB2_ENABLE_LDAP DB2_EVMON_EVENT_LIST_SIZE DB2_EVMON_STMT_FILTER DB2_EXTSECURITY DB2_FALLBACK DB2_FMP_COMM_HEAPSZ DB2_GRP_LOOKUP DB2_HADR_BUF_SIZE DB2_HADR_NO_IP_CHECK DB2_HADR_PEER_WAIT_LIMIT DB2LDAP_BASEDN DB2LDAPCACHE DB2LDAP_CLIENT_PROVIDER DB2LDAPHOST DB2LDAP_KEEP_CONNECTION DB2LDAP_SEARCH_SCOPE DB2_LOAD_COPY_NO_OVERRIDE DB2LOADREC DB2LOCK_TO_RB DB2_MAP_XML_AS_CLOB_FOR_DLC DB2_MAX_LOB_BLOCK_SIZE DB2_MEMORY_PROTECT DB2NOEXITLIST DB2_NUM_CKPW_DAEMONS DB2_OPTSTATS_LOG DB2REMOTEPREG DB2_RESOLVE_CALL_CONFLICT DB2ROUTINE_DEBUG DB2SATELLITEID DB2_SERVER_CONTIMEOUT DB2SORT DB2_THREAD_SUSPENSION DB2_TRUNCATE_REUSESTORAGE DB2_USE_DB2JCCT2_JROUTINE DB2_UTIL_MSGPATH DB2_VENDOR_INI DB2_XBSA_LIBRARY

汎用レジストリー変数

DB2ACCOUNT

- オペレーティング・システム: すべて
- デフォルト = NULL
- この変数は、リモート・ホストに送信される会計情報ストリングを定義します。詳細は、「DB2 Connect ユーザーズ・ガイド」を参照してください。

DB2BIDI

- オペレーティング・システム: すべて
- デフォルト = NO。値: YES または NO
- この変数は双方向サポートを可能にします。 **DB2CODEPAGE** 変数は、使用するコード・ページを宣言する場合に使用します。

DB2_CAPTURE_LOCKTIMEOUT

- オペレーティング・システム: すべて
- デフォルト = NULL。値: ON または NULL
- この変数は、ロックのタイムアウトが発生したときに、それに関する記述情報をログに記録するよう指定します。ログされた情報は次のことを識別します。ロック・タイムアウトの原因となったロック競合に関係したキー・アプリケーション、ロック・タイムアウト時に実行していたこれらのアプリケーションに関する詳細、および競合を引き起こしたロックに関する詳細。ロック・リクエスター (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ついての情報がキャプチャーされます。テキスト・レポートが書き込まれ、各ロック・タイムアウト時にファイルに保管されます。

  ファイルは、次の命名規則を使用して作成されます。
  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、*AGENTID* はエージェント ID、*yyyy-mm-dd-hh-mm-ss* は年、月、日、時、分、および秒で構成されるタイム・スタンプです。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。

  ファイルのロケーションは、*diagpath* データベース構成パラメーターに設定される値に基づきます。*diagpath* が設定されない場合、ファイルは次のディレクトリーの 1 つに配置されます。

  – Windows 環境では、以下のとおりです。
    - **DB2INSTPROF** 環境変数を設定しない場合は、情報は *x*:¥*SQLLIB*¥*DB2INSTANCE* に書き込まれます。ここで *x* はドライブ参照、*SQLLIB* は **DB2PATH** レジストリー変数に指定したディレクトリー、そして、*DB2INSTANCE* はインスタンス所有者の名前です。
    - **DB2INSTPROF** 環境変数を設定した場合、情報は *x*:¥*DB2INSTPROF*¥*DB2INSTANCE* に書き込まれます。ここで、*x* は

ドライブ参照、*DB2INSTPROF* はインスタンス・プロファイル・ディレクトリーの名前、そして *DB2INSTANCE* はインスタンス所有者の名前です。

– Linux および UNIX 環境では、情報は *INSTHOME*/sqllib/db2dump に書き込まれます。ここで、*INSTHOME* はインスタンスのホーム・ディレクトリーです。

ロック・タイムアウト・レポート・ファイルが必要なくなったら、これを削除してください。レポート・ファイルは他の診断ログと同じ場所に配置されるため、ディレクトリーがフルになると、DB2 システムはシャットダウンすることがあります。ロック・タイムアウト・レポート・ファイルを保持する必要がある場合、DB2 ログが保管されている所と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。

DB2CODEPAGE

- オペレーティング・システム: すべて
- デフォルト: オペレーティング・システムの指定どおりに言語 ID から得られます。
- この変数は、データベース・クライアント・アプリケーションのために DB2 に提示されるデータのコード・ページを指定します。 (設定するように) DB2 のマニュアルに明確に記述されているか、または DB2 サービスで求められない限り、 **DB2CODEPAGE** は設定しないでください。 オペレーティング・システムでサポートされていない値に **DB2CODEPAGE** を設定すると、その結果は予測できなくなります。通常、DB2 はコード・ページ情報を自動的にオペレーティング・システムから得るので、**DB2CODEPAGE** を設定する必要はありません。

  注: Windows は ANSII コード・ページの代わりに Unicode コード・ページを報告しない (Windows の地域設定で) ので、Windows アプリケーションは Unicode クライアントとして動作しません。この動作をオーバーライドするには、**DB2CODEPAGE** レジストリー変数を 1208 (Unicode コード・ページ用) に設定して、アプリケーションが Unicode アプリケーションとして動作するようにします。

DB2_COLLECT_TS_REC_INFO

- オペレーティング・システム: すべて
- デフォルト = ON。値: ON または OFF
- この変数は、表スペースに影響するログ・レコードがログ・ファイルに含まれているかどうかに関わりなく、表スペースのロールフォワード時に DB2 がすべてのログ・ファイルを処理するかどうかを指定します。この表スペースに影響するいずれのログ・レコードも含まれていないことが明確なログ・ファイルをスキップするには、この変数を ON に設定してください。 **DB2_COLLECT_TS_REC_INFO** はログ・ファイルを作成および使用する前に設定し、ログ・ファイルのスキップに必要な情報を収集できるようにします。

DB2_CONNRETRIES_INTERVAL

- オペレーティング・システム: すべて
- デフォルト = 設定しない。値: 秒数 (整数)

- この変数は、自動クライアント・リルート機能での接続の連続再試行間のスリープ時間を秒単位で指定します。この変数と **DB2_MAX_CLIENT CONNRETRIES** を併用することにより、自動クライアント・リルートの再試行の動作を構成できます。

  DB2_MAX_CLIENT_CONNRETRIES が設定されて **DB2_CONNRETRIES_INTERVAL** が設定されない場合、**DB2_CONNRETRIES_INTERVAL** はデフォルトで 30 になります。**DB2_MAX_CLIENT_CONNRETRIES** が設定されないで **DB2_CONNRETRIES_INTERVAL** が設定された場合、**DB2_MAX_CLIENT_CONNRETRIES** はデフォルトで 10 になります。**DB2_MAX_CLIENT_CONNRETRIES** も **DB2_CONNRETRIES_INTERVAL** も設定されない場合、自動クライアント・リルート・フィーチャーはデフォルトの動作に戻り、最大 10 分間、データベースへの接続を繰り返し再試行します。

DB2CONSOLECP

- オペレーティング・システム: Windows
- デフォルト = null。値: すべての有効なコード・ページ値
- DB2 メッセージ・テキストを表示するためのコード・ページを指定します。指定すると、この値はオペレーティング・システムのコード・ページ設定より優先されます。

DB2COUNTRY

- オペレーティング・システム: Windows
- デフォルト = null。値: すべての有効な country、territory、または region コードの数値
- この変数は、クライアント・アプリケーションの country、territory、または region コードを指定します。指定すると、この値はオペレーティング・システムの設定より優先されます。

注: DB2COUNTRY は、推奨されておらず、今後のリリースでは除去される可能性があります。代わりに DB2TERRITORY を使用してください。この変数は、**DB2COUNTRY** と同じ値を受け入れます。

DB2DBDFT

- オペレーティング・システム: すべて
- デフォルト = NULL
- この変数は、暗黙接続に使用するデータベースのデータベース別名を指定します。アプリケーションがデータベースに接続していないが SQL または XQuery ステートメントが発行されている場合、デフォルト・データベースで **DB2DBDFT** 環境変数が定義されていれば、暗黙接続が行われます。

DB2DBMSADDR

- オペレーティング・システム: Windows 32 ビット
- デフォルト = 0x20000000。値: 0x20000000 から 0xB0000000 (増分は 0x10000)

- この変数は、デフォルトのデータベース・マネージャー共用メモリーのアドレスを 16 進形式で指定します。共用メモリーのアドレス衝突のために db2start が失敗する場合、このレジストリー変数は強制的にデータベース・マネージャー・インスタンスに変更され、別のアドレスでその共用メモリーに割り当てられます。

DB2DISCOVERYTIME

- オペレーティング・システム: Windows
- デフォルト = 40 秒。最小 = 20 秒
- この変数は、SEARCH ディスカバリーが DB2 システムを探索する時間を指定します。

DB2FFDC

- オペレーティング・システム: すべて
- デフォルト: ON。値: ON、CORE:OFF
- この変数は、コア・ファイルの生成を非アクティブにする機能を提供します。デフォルトでは、このレジストリー変数は `ON` に設定されます。このレジストリー変数を設定しなかったり、`CORE:OFF` 以外の値に設定したりすると、DB2 サーバーが異常終了した場合に、コア・ファイルが生成されることがあります。

  問題判別に使用され、DIAGPATH に作成されるコア・ファイルには、DB2 終了プロセスのプロセス・イメージ全体が含まれています。コア・ファイルはかなり大きくなる可能性があるので、使用可能なファイル・システム・スペースを考慮すべきです。サイズは、問題発生時の DB2 の構成およびプロセスの状態によって異なります。

  Linux プラットフォームでは、デフォルトのコア・ファイルのサイズ制限が `0` (つまり `ulimit -c`) に設定されます。この設定では、コア・ファイルは生成されません。 Linux プラットフォーム上でコア・ファイルが作成されるようにするには、値を unlimited に設定します。

注: DB2FFDC は、バージョン 9.5 では推奨されておらず、今後のリリースでは除去される予定です。新しいレジストリー変数 **DB2FODC** は、**DB2FFDC** の機能を取り込んでいます。

DB2FODC

- オペレーティング・システム: すべて
- デフォルト: すべての FODC パラメーターの連結 (下記参照)
  - UNIX の場合: ″CORELIMIT=*val* DUMPCORE=ON DUMPDIR=*diagpath*″
  - Windows の場合: ″DUMPCORE=ON DUMPDIR=*diagpath*″

  各パラメーターはスペースで区切ります。
- このレジストリー変数は、FODC (First Occurrence Data Collection) で使用されるトラブルシューティング関連の一連のパラメーターを制御します。障害状態時のデータ収集のさまざまな側面を制御するには、**DB2FODC** を使用します。

このレジストリー変数は、DB2 インスタンスの始動時に 1 回読み取られます。FODC パラメーターのオンライン更新を実行するには、db2pdcfg ツールを使用します。各リブート時の構成が同じになるようにするには、**DB2FODC** レジストリー変数を使用します。パラメーターをすべて指定する必要はありません。また、パラメーターを特定の順序で指定する必要もありません。指定していないパラメーターには、デフォルト値が割り当てられます。例えば、コア・ファイルはダンプしない、しかし他のパラメーターについてはデフォルト動作が必要である、という場合は、次のコマンドを発行します。

```
db2set DB2FODC="DUMPCORE=OFF"
```

パラメーター:

CORELIMIT

- オペレーティング・システム: UNIX
- デフォルト=現在の ulimit 設定。値: 0 から無制限
- このオプションは、作成されるコア・ファイルの最大サイズをギガバイト単位で指定します。この値は現在のコア・ファイル・サイズの制限設定をオーバーライドします。コア・ファイルはかなり大きくなる可能性があるので、使用可能なファイル・システム・スペースを考慮すべきです。サイズは、問題発生時の DB2 の構成およびプロセスの状態によって異なります。

  CORELIMIT が設定されている場合、DB2 はこの値を使用して、現在のユーザー・コア制限 (ulimit) 設定をオーバーライドしてコア・ファイルを生成します。

  CORELIMIT が設定されていない場合、DB2 はコア・ファイルのサイズを、現在の ulimit 設定と同じ値に設定します。 1 つの例外は AIX の場合で、「無制限」という ulimit 設定は、DB2 サーバー・プロセスの場合に限り、8 GB の値でオーバーライドされます。8 GB より大きいコア・ダンプが必要な場合、ulimit を RAM のサイズなどの適切な大きさの値に設定するか、または十分な大きさの値で CORELIMIT を設定します。

  注: ユーザー・コア制限または CORELIMIT に対するどんな変更も、DB2 インスタンスの次のリサイクルまでは有効になりません。

DUMPCORE

- オペレーティング・システム: Linux、Solaris、AIX
- デフォルト = ON。値: ON または OFF
- このオプションは、コア・ファイルの生成を行うかどうかを指定します。問題判別に使用され、diagpath に作成されるコア・ファイルには、DB2 終了プロセスのプロセス・イメージ全体が含まれています。ただし、コア・ファイル・ダンプが実際に行われるかどうかは、CORELIMIT パラメーターの ulimit の

現在の設定と値によります。また、一部のオペレーティング・システムにはコア・ダンプの構成設定もあります。これはアプリケーション・コア・ダンプの動作を指示することがあります。

コア・ファイル・ダンプを使用不可にするために推奨される方法は、DUMPCORE を OFF に設定することです。

DUMPDIR

- オペレーティング・システム: すべて
- デフォルト=*diagpath* ディレクトリー。*diagpath* が定義されていない場合はデフォルトの診断ディレクトリー。値: ディレクトリーへのパス
- このオプションは、コア・ファイル作成用のディレクトリーの絶対パス名を指定します。このオプションは、コア・ファイルだけでなく、FODC パッケージの外部に格納する必要がある他の大規模バイナリー・ダンプにも使用できます。

DB2_FORCE_APP_ON_MAX_LOG

- オペレーティング・システム: すべて
- デフォルト = TRUE。値: TRUE または FALSE
- *max_log* 構成パラメーター値を超過したときの反応を指定します。 TRUE に設定すると、アプリケーションはデータベースを強制終了し、作業単位をロールバックします。

  FALSE に設定すると、現行のステートメントは失敗します。アプリケーションは、作業単位内のそれ以前のステートメントで完了した作業をコミットできます。または、作業をロールバックして作業単位を取り消すこともできます。

DB2GRAPHICUNICODESERVER

- オペレーティング・システム: すべて
- デフォルト = OFF。値: ON または OFF
- このレジストリー変数は、GRAPHIC データを Unicode データベースに挿入するために作成された既存のアプリケーションを受け入れるために使用されます。このレジストリー変数を使用するのは、sqldbchar (GRAPHIC) データを、クライアントのコード・ページの代わりに Unicode で送信するアプリケーションにのみ必要です。(sqldbchar は、単一の 2 バイト文字を保持できる C および C++ の SQL データ・タイプでサポートされています。) ON に設定すると、データベースに対して、GRAPHIC データが Unicode で送信されてくることを伝え、アプリケーションは GRAPHIC データを Unicode で受信することを期待します。

DB2INCLUDE

- オペレーティング・システム: すべて
- デフォルト = 現行ディレクトリー
- DB2 PREP 処理において、 SQL INCLUDE テキスト・ファイル・ステートメントの処理時に使用されるパスを指定します。これによって、INCLUDE ファイルが検出されるディレクトリーのリストが提供されま

す。プリコンパイルされるさまざまな言語での **DB2INCLUDE** の使用法については、「組み込み SQL アプリケーションの開発」を参照してください。

DB2INSTDEF

- オペレーティング・システム: Windows
- デフォルト = DB2
- この変数は、**DB2INSTANCE** が定義されていない場合に使用される値を設定します。

DB2INSTOWNER

- オペレーティング・システム: Windows
- デフォルト = NULL
- インスタンスの初回作成時に DB2 プロファイル登録で作成されるレジストリー変数。この変数は、インスタンスを所有するマシンの名前に設定されます。

DB2_LIC_STAT_SIZE

- オペレーティング・システム: すべて
- デフォルト = null。範囲: 0 から 32767
- この変数は、システムのライセンス統計が入っているファイルの最大サイズ (MB 単位) を決定します。値がゼロの場合、ライセンスの統計収集はオフになります。認識または定義されていない場合、この変数はデフォルト (無制限) に設定されます。統計は、ライセンス・センターを使って表示されます。

DB2LOCALE

- オペレーティング・システム: すべて
- デフォルト = NO。値: YES または NO
- この変数は、DB2 を呼び出した後にデフォルトの「C」プロセス・ロケールがデフォルトの「C」ロケールにリストアされるかどうか、および DB2 関数を呼び出した後にプロセス・ロケールをリストアして元の「C」に戻すかどうかを指定します。元のロケールが「C」ではない場合、このレジストリー変数は無視されます。

DB2_MAX_CLIENT_CONNRETRIES

- オペレーティング・システム: すべて
- デフォルト = 設定しない。値: 接続再試行の最大回数 (整数)
- この変数は、自動クライアント・リルート機能が試みる接続再試行の最大回数を指定します。この変数と **DB2_CONNRETRIES_INTERVAL** を併用することにより、自動クライアント・リルートの再試行の動作を構成できます。

  DB2_MAX_CLIENT_CONNRETRIES が設定されて
  DB2_CONNRETRIES_INTERVAL が設定されない場合、
  DB2_CONNRETRIES_INTERVAL はデフォルトで 30 になります。
  DB2_MAX_CLIENT_CONNRETRIES が設定されないで
  DB2_CONNRETRIES_INTERVAL が設定された場合、

DB2_MAX_CLIENT_CONNRETRIES はデフォルトで 10 になります。**DB2_MAX_CLIENT_CONNRETRIES** も **DB2_CONNRETRIES_INTERVAL** も設定されない場合、自動クライアント・リルート・フィーチャーはデフォルトの動作に戻り、最大 10 分間、データベースへの接続を繰り返し再試行します。

DB2_OBJECT_TABLE_ENTRIES

- オペレーティング・システム: すべて
- デフォルト = 0。値: 0 から 65532

  ご使用のシステムで実際に可能な最大値は、ページ・サイズとエクステント・サイズによって異なります。ただし、65532 を超えることはできません。
- この変数は、1 つの表スペースに入ることが予想されるオブジェクトの数を指定します。 DMS 表スペースに大量のオブジェクト (たとえば、1000 以上) が作成されることが分かっている場合は、表スペースを作成する前に、このレジストリー変数を適切な数値に設定します。これにより、表スペースの作成時のオブジェクト・メタデータ用に連続するストレージが予約されます。連続するストレージを予約すると、メタデータ内の項目を更新する操作 (例えば、CREATE INDEX、IMPORT REPLACE) がオンライン・バックアップによってブロックされる可能性は減ります。また、表スペースの開始時にメタデータが保管されるので、表スペースのサイズ変更が容易になります。

  表スペースの初期サイズが十分でないために連続するストレージを予約できない場合は、追加スペースの予約を指定せずに表スペース作成が続行します。

DB2_SYSTEM_MONITOR_SETTINGS

- オペレーティング・システム: すべて
- このレジストリー変数は、DB2 モニターのさまざまな側面の動作を変更できる一連のパラメーターを制御します。各パラメーターは、次の例のように、セミコロンで区切ります。

  ```
  db2set DB2_SYSTEM_MONITOR_SETTINGS=OLD_CPU_USAGE:TRUE;
   DISABLE_CPU_USAGE:TRUE
  ```

  DB2_SYSTEM_MONITOR_SETTINGS を設定するたびに、各パラメーターを明示的に設定する必要があります。この変数を設定するときに指定されなかったパラメーターは、デフォルト値に戻ります。したがって、次の例の場合、

  ```
  db2set DB2_SYSTEM_MONITOR_SETTINGS=DISABLE_CPU_USAGE:TRUE
  ```

  注: 現在、このレジストリー変数には Linux の設定値しか用意されていません。他のオペレーティング・システムの設定は、将来のリリースで追加されます。
  OLD_CPU_USAGE はデフォルト設定に戻ります。
- パラメーター:

  OLD_CPU_USAGE

  – オペレーティング・システム: Linux

- 値: TRUE/ON、FALSE/OFF
- RHEL4 および SLES9 でのデフォルト値: TRUE (注: OLD_CPU_USAGE に FALSE を設定すると、無視されて以前の動作のみが使用されます。)
- RHEL5、SLES10、およびその他でのデフォルト値: FALSE
- このパラメーターは、Linux プラットフォームでインスタンスが CPU 使用時間を取得する方式を制御します。 TRUE に設定すると、以前の CPU 使用時間取得方式が使用されます。この方式では、システム使用時間とユーザー CPU 使用時間の両方が返されますが、そのために CPU 消費が増加します (つまり、オーバーヘッドが大きくなります)。FALSE に設定すると、新しい CPU 使用状況取得方式が使用されます。この方式は、ユーザーの CPU 使用状況の値だけを返しますが、オーバーヘッドが小さいので高速です。

DISABLE_CPU_USAGE

- オペレーティング・システム: Linux
- 値: TRUE/ON、FALSE/OFF
- RHEL4 および SLES9 でのデフォルト値: TRUE
- RHEL5、SLES10、およびその他でのデフォルト値: FALSE
- このパラメーターで、CPU 使用状況を読み取るかどうかを決定できます。 DISABLE_CPU_USAGE を有効 (TRUE に設定) すると、CPU 使用状況は読み取られないので、CPU 使用状況の取得時に発生することがあるオーバーヘッドを回避できます。

DB2TERRITORY

- オペレーティング・システム: すべて
- デフォルト = オペレーティング・システムの指定どおりに言語 ID から得られます。
- この変数は、クライアント・アプリケーションの region および territory コードを指定します。これは、日付と時刻の形式に影響します。

DB2_VIEW_REOPT_VALUES

- オペレーティング・システム: すべて
- デフォルト = NO。値: YES、NO
- この変数を使用すると、すべてのユーザーは、ステートメントを EXPLAIN するときに、再最適化された SQL または XQuery ステートメントのキャッシュされた値を EXPLAIN_PREDICATE 表に保管できます。この変数を NO に設定すると、これらの値を EXPLAIN_PREDICATE 表に保管できるのは、DBADM だけになります。

システム環境変数

DB2_ALTERNATE_GROUP_LOOKUP

- オペレーティング・システム: AIX
- デフォルト = NULL。値: NULL または GETGRSET

- この変数は、オペレーティング・システムが提供する代替ソースからグループ情報を取得することを、DB2 に許可します。 AIX では、関数 getgrset が使用されます。これにより、Loadable Authentication Modules を介してローカル・ファイル以外からグループを取得できるようになります。

DB2_CLP_EDITOR

- オペレーティング・システム: すべて
- デフォルト = 「メモ帳」(Windows プラットフォーム)、vi (UNIX)。値: オペレーティング・システム・パスにある任意の有効なエディター

  注: このレジストリー変数は、インストール時にデフォルト値に設定されません。その代わり、このレジストリー変数が設定されていない場合、この変数を利用するコードがデフォルト値を使用します。
- この変数は、EDIT コマンドの実行時に使用するエディターを決定します。 CLP 対話式セッションから EDIT コマンドを実行すると、ユーザー指定コマンドでプリロード済みのエディターが起動し、編集と実行が可能になります。

DB2_CLP_HISTSIZE

- オペレーティング・システム: すべて
- デフォルト = 20。値: 1 から 500 まで

  注: このレジストリー変数は、インストール時にデフォルト値に設定されません。このレジストリー変数が設定されていない場合、または有効な範囲外の値に設定されている場合、この変数を利用するコードがデフォルト値の 20 を使用します。
- この変数は、CLP 対話式セッションでコマンド履歴に保管されるコマンドの数を決定します。コマンド履歴はメモリー内で保持されるため、この変数の値を高くしすぎると、セッション内で実行されるコマンドの数および長さによってはパフォーマンスが低下する場合があります。

DB2CONNECT_IN_APP_PROCESS

- オペレーティング・システム: すべて
- デフォルト = YES。値: YES または NO
- この変数を NO に設定すると、 DB2 Enterprise Server Edition マシン上のローカル DB2 Connect クライアントはエージェント内で強制的に実行されます。エージェント内で実行する利点としては、ローカル・クライアントをモニターできることと、ローカル・クライアントが SYSPLEX サポートを使用できることがあります。

DB2_COPY_NAME

- オペレーティング・システム: Windows
- デフォルト = マシンにインストールされている DB2 のデフォルト・コピーの名前。値: マシンにインストールされている DB2 のコピーの名前。名前の長さは最大で 128 文字です。
- **DB2_COPY_NAME** 変数は、現在使用中の DB2 のコピーの名前を保管します。マシンに複数の DB2 のコピーがインストールされている場合は、**DB2_COPY_NAME** を使用して DB2 の別のコピーに切り替えるこ

とはできません。その場合、現在使用中のコピーを変更するには、コマンド [INSTALLPATH]¥bin¥db2envar.bat を実行する必要があります。

DB2DBMSADDR

- オペレーティング・システム: Linux on x86 および Linux on zSeries (31 ビット)
- デフォルト = NULL。値: 範囲 0x09000000 から 0xB0000000、増分 0x10000 の仮想アドレス
- **DB2DBMSADDR** レジストリー変数は、デフォルトのデータベース共用メモリーのアドレスを 16 進形式で指定します。

  注: アドレスを間違うと、DB2 システムで重大な問題 (DB2 インスタンスを始動できない、データベースに接続できない、など) が発生する可能性があります。アドレスを誤ると、すでに使用中かまたは他で使用するために予定されていたメモリー内の領域と競合する場合があります。この問題を解決するには、以下のコマンドを使用して **DB2DBMSADDR** レジストリー変数を NULL にリセットします。

  ```
  db2set DB2DBMSADDR=
  ```

  この変数を使用して、DB2 プロセスのアドレス・スペース・レイアウトを調整することができます。この変数により、インスタンスの共用メモリーのロケーションが仮想アドレス 0x10000000 の現在のロケーションから新規の値に変更されます。

DB2_DIAGPATH

- オペレーティング・システム: すべて
- デフォルト = デフォルト値は、UNIX および Linux オペレーティング・システムではインスタンスの db2dump ディレクトリー、Windows オペレーティング・システムではインスタンスの db2 ディレクトリーです。
- このパラメーターは、ODBC および DB2 CLI アプリケーションにのみ適用されます。

  このパラメーターを使用して、DB2 診断情報のための完全修飾パスを指定できます。このディレクトリーには、ご使用のプラットフォームに応じて、ダンプ・ファイル、トラップ・ファイル、エラー・ログ・ファイル、通知ファイルおよびアラート・ログ・ファイルが入れられます。

  この環境変数を設定した場合に、その環境の有効範囲内にある ODBC および CLI アプリケーションで得られる効果は、 DB2 データベース・マネージャー構成パラメーター *diagpath* を設定した場合、および CLI/ODBC 構成キーワード **DiagPath** を設定した場合と同様です。

DB2DOMAINLIST

- オペレーティング・システム: すべて
- デフォルト = NULL。値: コンマ (『,』) 区切りの Windows ドメイン・ネームのリスト

- この変数は、1 つ以上の Windows ドメインを定義します。リストは、要求ユーザー ID が認証されるドメインを定義するもので、サーバーで維持されます。これらのドメインに属しているユーザーの接続要求のみが受け入れられます。

  この変数は、CLIENT 認証がデータベース・マネージャー構成で設定されている場合にのみ有効です。この変数は、Windows ドメイン環境で Windows デスクトップからのシングル・サインオンが要求されている場合に必要です。

  DB2 サーバー V7.1 以降は **DB2DOMAINLIST** をサポートしますが、純粋な Windows ドメイン環境に限定されます。V8 FixPak 15 および V9 FixPak 3 からは、クライアントかサーバーのどちらかが Windows 環境で実行されていれば、**DB2DOMAINLIST** がサポートされます。

DB2ENVLIST

- オペレーティング・システム: UNIX
- デフォルト = NULL
- この変数は、ストアード・プロシージャーまたはユーザー定義関数の特定の変数名をリストします。デフォルトでは、db2start コマンドは、接頭部が **DB2** または **db2** になっているユーザー環境変数を除いて、すべてのユーザー環境変数をフィルターに掛けて除去します。特定の環境変数をストアード・プロシージャーまたはユーザー定義関数のどちらかに渡さなければならない場合、 **DB2ENVLIST** 環境変数にその変数名をリストできます。その場合、各変数を 1 つまたは複数のスペースで区切ります。

DB2INSTANCE

- オペレーティング・システム: すべて
- デフォルト =**DB2INSTDEF** (Windows 32 ビット・オペレーティング・システムの場合)。
- この環境変数は、デフォルトでアクティブになるインスタンスを指定します。 UNIX では、ユーザーは **DB2INSTANCE** に値を指定する必要があります。

DB2INSTPROF

- オペレーティング・システム: Windows
- デフォルト = Documents and Settings¥All Users¥Application Data¥IBM¥DB2¥<コピー名> (Windows XP、Windows 2003)、ProgramData¥IBM¥DB2¥<コピー名> (Windows Vista)
- この環境変数は、Windows オペレーティング・システムでのインスタンス・ディレクトリーの場所を指定します。バージョン 9.5 からは、sqllib ディレクトリーの下にインスタンス・ディレクトリー (および他のユーザー・データ・ファイル) を置くことはできません。

DB2LDAPSecurityConfig

- オペレーティング・システム: すべて
- デフォルト = NULL。値: IBM LDAP セキュリティー・プラグイン構成ファイルの有効な名前とパス

- この変数は、IBM LDAP セキュリティー・プラグイン構成ファイルの場所を指定するのに使用されます。この変数が設定されていない場合、IBM LDAP セキュリティー・プラグイン構成ファイルの名前は IBMLDAPSecurity.ini になり、このファイルは以下の場所のいずれかにあります。
  - Linux および UNIX オペレーティング・システム: INSTHOME/sqllib/cfg/
  - Windows オペレーティング・システム: %DB2PATH%¥cfg¥

  Windows オペレーティング・システムでは、グローバル・システム環境で設定し、DB2 サービスからその値を参照できるようにしておく必要があります。

DB2LIBPATH

- オペレーティング・システム: UNIX
- デフォルト = NULL
- DB2 はその共有ライブラリー・パスを構成します。PATH をエンジンのライブラリー・パスに追加する場合は、(例えば、AIX ではユーザー定義関数には LIBPATH 内に特定項目が必要) **DB2LIBPATH** を設定する必要があります。実際の **DB2LIBPATH** の値は、 DB2 構成共有ライブラリー・パスの最後に付加されます。

DB2LOGINRESTRICTIONS

- オペレーティング・システム: AIX
- デフォルト = LOCAL。値: LOCAL、REMOTE、SU、NONE
- このレジストリー変数により、loginrestrictions() という AIX オペレーティング・システム API を使用できます。この API は、ユーザーがシステムへのアクセスを許可されるかどうかを決定します。この API を呼び出すことにより、DB2 データベース・セキュリティーは、オペレーティング・システムによって指定されたログイン制限を施行できます。このレジストリー変数を使用する際に、この API にサブミットできる値がいくつかあります。それらの値は次のとおりです。
  - REMOTE

    DB2 は、*rlogind* または *telnetd* プログラムによってリモート・ログインにアカウントを使用できることを確認するために、ログイン制限だけを施行します。
  - SU

    DB2 V9.1 は、su コマンドを使用でき、また現行プロセスが su コマンドを呼び出すことができるグループ ID を持つことを確認するために、su 制限だけを施行します。
  - NONE

    DB2 は、ログイン制限を施行しません。
  - LOCAL (または変数を設定しない)

DB2 は、このアカウントにローカル・ログインが許可されることを確認するために、ログイン制限だけを施行します。これは、ログイン時の通常の振る舞いです。

これらのオプションのいずれを設定したかに関わらず、指定された特権を持つユーザー・アカウントまたは ID は、サーバー上でローカルに、またリモート・クライアントから、DB2 を正常に使用できます。
loginrestrictions() API の説明については、 AIX 資料を参照してください。

DB2NODE

- オペレーティング・システム: すべて
- デフォルト = NULL。値: 1 から 999
- アタッチまたは接続するデータベース・パーティション・サーバーのターゲット論理ノードを指定します。この変数を指定しない場合、ターゲット論理ノードはデフォルトとして、マシン上のポート 0 に定義された論理ノードに設定されます。パーティション・データベース環境では、接続設定がトラステッド接続の獲得に影響を与えることがあります。例えば、**DB2NODE** 変数に設定されたノードでの接続の確立のためには中間ノード (ホップ・ノード) を経由する必要がある場合、この接続をトラステッド接続としてマーク付けできるかどうかを判別するための評価の際に判断されるのは、その中間ノードの IP アドレスと、ホップ・ノードと接続ノード間の通信に使用される通信プロトコルです。つまり、接続元となるノードは考慮されません。むしろ、ホップ・ノードが考慮されます。

DB2OPTIONS

- オペレーティング・システム: すべて
- デフォルト = NULL
- コマンド行プロセッサーのオプションを設定するのに使用します。

DB2_PARALLEL_IO

- オペレーティング・システム: すべて
- デフォルト = NULL。値: *TablespaceID*:[*n*],... - 定義済み表スペースのコンマで区切られたリスト (その数値の表スペース ID で識別される)。表スペースのプリフェッチ・サイズが AUTOMATIC の場合、表スペース ID、続けてコロン、コンテナー当たりのディスク数 (*n*) を指定することで、DB2 データベース・マネージャーにその表スペースのコンテナー当たりのディスク数を指示することができます。*n* を指定しない場合、デフォルトは 6 です。

  TablespaceID をアスタリスク (*) と置き換えて、すべての表スペースを指定することができます。例えば、**DB2_PARALLEL_IO** =* の場合、すべての表スペースは、コンテナー当たりのディスク数として 6 を使用します。アスタリスク (*) と表スペース ID の両方を指定した場合、表スペース ID の設定が優先されます。例えば、**DB2_PARALLEL_IO** =*,1:3 の場合、すべての表スペースには、コンテナー当たりのディスク数として 6 を使用します。ただし、表スペース 1 の場合は 3 を使用します。
- このレジストリー変数は、DB2 が表スペースの入出力並列処理を計算する方法を変更するために使用します。入出力並列処理を有効にすると (複

数のコンテナーを使用することにより暗黙的に、または **DB2_PARALLEL_IO** を設定することにより明示的に)、その結果としてプリフェッチ要求の正しい数が発行されます。各プリフェッチ要求は、ページのエクステントの要求です。例えば、表スペースには 2 つのコンテナーがあり、プリフェッチ・サイズはエクステント・サイズの 4 倍です。レジストリー変数が設定される場合、この表スペースに対するプリフェッチ要求は 4 つの要求 (要求ごとに 1 つのエクステント) に分けられ、4 つのプリフェッチャーが並列で要求を処理する可能性があります。

表スペース内の個々のコンテナーが複数の物理ディスクを介してストライピングされる場合に、または表スペースのコンテナーが複数の物理ディスクから成る単一の RAID 装置に作成される場合に、レジストリー変数を設定する場合もあるでしょう。

このレジストリー変数を設定しない場合、表スペースの並列処理の多重度は表スペースのコンテナーの数です。例えば、**DB2_PARALLEL_IO** が NULL に設定されており、表スペースに 4 つのコンテナーがある場合、4 つのエクステント・サイズのプリフェッチ要求が発行されます。または、表スペースには 2 つのコンテナーがあり、プリフェッチ・サイズがエクステント・サイズの 4 倍である場合、この表スペースに対するプリフェッチ要求は、2 つの要求 (要求ごとに 2 つのエクステント) に分けられます。

このレジストリー変数が設定され、表のプリフェッチ・サイズが AUTOMATIC ではない場合、表スペースの並列処理の度合いは、エクステント・サイズにより除算されたプリフェッチ・サイズになります。例えば、プリフェッチ・サイズが 160 ページでエクステント・サイズが 32 ページの表スペースに **DB2_PARALLEL_IO** を設定した場合、5 つのエクステント・サイズ・プリフェッチ要求が発行されます。

このレジストリー変数が設定されており、表スペースのプリフェッチ・サイズが AUTOMATIC の場合、DB2 が次の式を使用して、表スペースのプリフェッチ・サイズを自動的に計算します。

```
プリフェッチ・サイズ =
(コンテナー数) * (コンテナー当たりのディスク数)
  * エクステント・サイズ
```

コロンの後の数値は、DB2 により上記の式の (コンテナー当たりのディスク数) に使用されます。アスタリスクだけが使用されており、数値が指定されていない場合には、デフォルトの各コンテナーのディスク数 6 が使用されます。

以下の表は、使用可能なさまざまなオプションと、各状況で並列処理を計算する方法を要約しています。

表 63. 並列処理の計算方法

表スペースのプリフェッチ・サイズ	DB2_PARALLEL_IO 設定	並列処理は以下に等しくなる
AUTOMATIC (プリフェッチ・サイズ = コンテナーの数 * 1 * エクステント・サイズ)	設定されない	コンテナーの数
AUTOMATIC (プリフェッチ・サイズ = コンテナー数 * 6 * エクステント・サイズ)	*表スペース ID*	コンテナー数 * 6
AUTOMATIC (プリフェッチ・サイズ = コンテナー数 * *n* * エクステント・サイズ)	*表スペース ID*:*n*	コンテナー数 * n
AUTOMATIC ではない	設定されない	コンテナーの数
AUTOMATIC ではない	*表スペース ID*	プリフェッチ・サイズ/エクステント・サイズ
AUTOMATIC ではない	*表スペース ID*:*n*	プリフェッチ・サイズ/エクステント・サイズ

たとえば、3 つの表スペースがあり、その ID がそれぞれ 3、4、5 であるとします。そのエクステント・サイズはすべて 4096 バイトであり、それぞれ 2 つのコンテナーを持っています。表スペース 3 と 4 のプリフェッチ・サイズは両方とも AUTOMATIC であり、表スペース 5 のプリフェッチ・サイズは 16384 バイトです。 **DB2_PARALLEL_IO**=*:5,4:10 と設定したとすると、表スペースの並列処理は以下のように導き出されることになります。

- 表スペース 3: n の値 (コンテナー当たりのディスク数) は 5、エクステント・サイズ =4096、コンテナー数 =2、およびプリフェッチ・サイズは AUTOMATIC です。したがって、プリフェッチ・サイズは 2*5*4096 であり、並列処理 = コンテナー数 * n =2*5=10 です。
- 表スペース 4: n の値 (コンテナー当たりのディスク数) は、この表スペースでは特に 10 に設定されます。エクステント・サイズ =4096、コンテナー数 =2、n=10、およびプリフェッチ・サイズは AUTOMATIC です。したがって、プリフェッチ・サイズ =2*10*4096、並列処理 = コンテナー数 *n=2*10=20 です。
- 表スペース 5: n の値は 5 のままですが、プリフェッチ・サイズは AUTOMATIC ではないので、影響はありません。エクステント・サイズ =4096、コンテナー数 =2、およびプリフェッチ・サイズ =16384 です。したがって、並列処理 = プリフェッチ・サイズ/エクステント・サイズ =16384/4096=4 です。

いくつかのシナリオでは、この変数を使用した結果、ディスクの競合が発生する場合があります。例えば、表スペースに 2 つのコンテナーがあり、2 つのコンテナーのそれぞれが専用の単一ディスクを持つ場合、レジストリー変数を設定すると、 2 つのプリフェッチャーが 2 つのディスクのそれぞれに同時にアクセスするので、それらのディスクにおける競合が

生じることがあります。ただし、2 つのコンテナーのそれぞれが複数のディスクを介してストライピングされた場合、レジストリー変数を設定すると、すぐに 4 つの異なるディスクへのアクセスが許可される可能性があります。

このレジストリー変数に対する変更を有効にするには、 db2stop コマンドを出して db2start コマンドを入力します。

DB2PATH

- オペレーティング・システム: Windows
- デフォルト = (オペレーティング・システムによって異なる)
- この環境変数は、Windows 32 ビットのオペレーティング・システムにおいて、この製品がインストールされるディレクトリーを指定するために使用されます。

DB2PROCESSORS

- オペレーティング・システム: Windows
- デフォルト= NULL。値: 0-n-1 (ここで n はプロセッサーの数)
- この変数は、特定の db2syscs プロセスに対してプロセス・アフィニティー・マスクを設定します。複数の論理ノードを実行している環境では、論理ノードを 1 つのプロセッサーまたは複数のプロセッサーの集合に関連付けるためにこの変数が使用されます。

  この変数が指定されると、DB2 は SetProcessAffinityMask() API を発行します。指定されない場合は、db2syscs プロセスがサーバー上のすべてのプロセッサーに関連付けられます。

DB2RCMD_LEGACY_MODE

- オペレーティング・システム: Windows
- デフォルト = NULL。値: YES、ON、TRUE、または 1。あるいは NO、OFF、FALSE、または 0
- この変数は、DB2 リモート・コマンド・サービスの拡張セキュリティーを使用可能または使用不可にすることを、ユーザーに許可します。 DB2 リモート・コマンド・サービスを安全に実行するには、**DB2RCMD_LEGACY_MODE** を NO、OFF、FALSE、0、NULL のいずれかに設定します。レガシー・モード (拡張セキュリティーなし) で実行するには、 **DB2RCMD_LEGACY_MODE** を YES、 ON、TRUE、1 のいずれかに設定します。セキュア・モードは、ドメイン・コントローラーがWindows 2000 以降を実行している場合のみ使用可能です。

  注: DB2RCMD_LEGACY_MODE を YES、ON、TRUE、1 のいずれかに設定すると、DB2 リモート・コマンド・サービスに送信されるすべての要求は、リクエスターのコンテキストで処理されます。この処理を進めるには、マシンとサービスのいずれかまたは両方のログオン・アカウントにクライアントの偽名を使用できるようにする必要があります。そのためには、ドメイン・コントローラーでのマシンとサービスのログオン・アカウントを使用可能にします。

注: **DB2RCMD_LEGACY_MODE** を NO、OFF、FALSE、 0 のいずれかに設定する場合、自分の代わりに DB2 リモート・コマンド・サービスにコマンドを実行させるには、SYSADM 権限が必要です。

DB2SYSTEM

- オペレーティング・システム: Windows および UNIX
- デフォルト = NULL
- DB2 サーバー・システムを識別するためにユーザーおよびデータベース管理者が使用する名前を指定します。可能であれば、この名前はご使用のネットワーク内で固有である必要があります。

  この名前は、コントロール・センターのオブジェクト・ツリーのシステム・レベルで表示されるので、管理者がコントロール・センターから管理できるサーバー・システムを識別する場合に役立ちます。

  構成アシスタントの「ネットワークの検索」機能を使用すると、DB2 ディスカバリーがこの名前を戻し、その結果がオブジェクト・ツリーのシステム・レベルで表示されます。この名前は、ユーザーがアクセスするデータベースが入っているシステムを識別するのに役立ちます。**DB2SYSTEM** の値は、インストール時に次のように設定されます。

  – Windows では、セットアップ・プログラムによって Windows システム用に指定されているコンピューター名に等しく設定されます。
  – UNIX システムでは、 UNIX システムの TCP/IP ホスト名に等しい名前に設定されます。

DB2_UPDDBCFG_SINGLE_DBPARTITION

- オペレーティング・システム: すべて
- デフォルト = 設定しない。値: 0/FALSE/NO、1/TRUE/YES
- このレジストリー変数では、1、TRUE、YES のいずれかに設定すると、データベースの更新およびリセットが特定のパーティションのみに作用することを指定できます。この変数を設定しない場合、更新および要求はバージョン 9.5 の動作に従います。
- バージョン 9.5 から、パーティション節を指定しない場合は、データベース構成の更新または変更はすべてのデータベース・パーティションにわたって作用します。 **DB2_UPDDBCFG_SINGLE_DBPARTITION** では、以前のバージョンの DB2 の動作に戻すことが可能です。つまり、ローカル・データベース・パーティションまたは **DB2NODE** レジストリー変数で設定されたデータベース・パーティションにのみデータベース構成の更新を適用できます。こうすることで、この動作を必要とする既存のコマンド・スクリプトやアプリケーションに対して、後方互換性をサポートできます。

注: この変数は、ADMIN_CMD ルーチンの呼び出しによって行われる更新要求やリセット要求には適用されません。

DB2_USE_PAGE_CONTAINER_TAG

- オペレーティング・システム: すべて
- デフォルト = NULL。値: ON、NULL

- デフォルトでは、DB2 はコンテナー・タグを各 DMS コンテナー (それがファイルでも装置でも) の最初のエクステントに保管します。コンテナー・タグは、コンテナーのメタデータです。 DB2 バージョン 8.1 より前には、コンテナー・タグは単一のページに保管されたので、コンテナー内でより少ないスペースだけを必要としました。継続してコンテナー・タグを単一のページに保管するには、
  DB2_USE_PAGE_CONTAINER_TAG を ON に設定します。

  しかし、コンテナーに RAID 装置を使用する場合にこのレジストリー変数を ON に設定すると、入出力パフォーマンスは低下することがあります。 RAID 装置ではエクステント・サイズが RAID ストライプ・サイズと等しいかその倍数の表スペースを作成するので、
  DB2_USE_PAGE_CONTAINER_TAG を ON に設定すると、エクステントが RAID ストライプときれいに一致しなくなります。 その結果、入出力要求は最適な場合よりも多くの物理ディスクにアクセスしなければならないことがあります。 非常に厳しいスペース制約があるか、またはバージョン 8 以前のデータベースと動作が一貫している必要があるのでない限り、このレジストリー変数を使用可能にしないことを強くお勧めします。

  このレジストリー変数に対する変更を有効にするには、 db2stop コマンドを出して db2start コマンドを入力します。

DB2_WORKLOAD

- オペレーティング・システム: すべて
- デフォルト=設定しない。値: 1C、TPM、SAP
- **DB2_WORKLOAD** の各値は、事前定義された設定値を持ついくつかのレジストリー変数を固有にグループ化したものです。
- 有効な値は以下のとおりです。

  1C この設定は、1C アプリケーション用のデータベースでレジストリー変数のセットを構成する場合に使用します。

  TPM この設定は、Tivoli Provisioning Manager 用のデータベースでレジストリー変数のセットを構成する場合に使用します。

  SAP この設定は、SAP 環境用のデータベースでレジストリー変数のセットを構成する場合に使用します。

  DB2_WORKLOAD=SAP を設定した場合、ユーザー表スペース SYSTOOLSPACE と USER TEMPORARY 表スペース SYSTOOLSTMPSPACE は自動的には作成されません。自動作成される表のこれらの表スペースは、以下のウィザード、ユーティリティー、または関数によって使用されます。

  - 自動保守
  - 設計アドバイザー
  - コントロール・センターのデータベース情報パネル
  - SYSINSTALLOBJECTS ストアード・プロシージャー (表スペースの入力パラメーターが指定されていない場合)
  - GET_DBSIZE_INFO ストアード・プロシージャー

SYSTOOLSPACE および SYSTOOLSTMPSPACE 表スペースがないと、これらのウィザード、ユーティリティー、または関数を使用できません。

ウィザード、ユーティリティー、または関数を使用するには、以下のいずれかを行います。

– ツールが必要とするオブジェクトを保持するための SYSTOOLSPACE 表スペースを手動で作成します (パーティション・データベース環境では、この表スペースをカタログ・パーティション上に作成します)。例:

```
CREATE REGULAR TABLESPACE SYSTOOLSPACE
IN IBMCATGROUP
MANAGED BY SYSTEM
USING ('SYSTOOLSPACE')
```

– 有効な表スペースを指定し、SYSINSTALLOBJECTS ストアード・プロシージャーを呼び出してツールのオブジェクトを作成し、特定のツールの ID を指定する。 SYSINSTALLOBJECTS によって表スペースが作成されます。オブジェクト用に SYSTOOLSSPACE を使用しない場合は、ユーザー定義の別の表スペースを指定してください。

これらの選択肢の少なくとも 1 つを完了した後、SYSTOOLSTMPSPACE TEMPORARY 表スペースを作成します (このときも、パーティション・データベース環境で作業する場合はカタログ・パーティションに作成します)。例:

```
CREATE USER TEMPORARY TABLESPACE SYSTOOLSTMPSPACE
IN IBMCATGROUP
MANAGED BY SYSTEM
USING ('SYSTOOLSTMPSPACE')
```

表スペース SYSTOOLSPACE および TEMPORARY 表スペース SYSTOOLSTMPSPACE を作成すると、前述のウィザード、ユーティリティー、または関数が使用可能になります。

通信変数

DB2CHECKCLIENTINTERVAL

- オペレーティング・システム: すべて、ただしサーバーのみ
- デフォルト = 50。値: ゼロ以上の数値。
- この変数は、TCP/IP クライアント接続検査の頻度を指定します。照会完了まで待つ代わりに、クライアント終了を早期に検出できるようにします。この変数が 0 に設定されている場合、検査は実行されません。

  低い値ほどチェックが頻繁になります。低い頻度の場合は 100、中位の頻度の場合は 50、高い頻度の場合は 10 を目安にしてください。 値は内部 DB2 メトリックで測られます。値は線形スケールを表します。つまり、値が 50 から 100 に増えると、間隔は 2 倍になります。データベース要求の実行中に、クライアントの状況を頻繁にチェックすればするほど、照会が完了するまでの時間が長くなります。 DB2 のワークロードが重い

(内部要求が多い) 場合、**DB2CHECKCLIENTINTERVAL** に低い値を設定すると、ワークロードが軽い状況よりも、パフォーマンスに重大な影響があります。

DB2 Universal Database™ バージョン 8.1.4 以降、**DB2CHECKCLIENTINTERVAL** のデフォルト値は 50 です。バージョン 8.1.4 より前は、デフォルト値が 0 です。

DB2COMM

- オペレーティング・システム: すべて、ただしサーバーのみ
- デフォルト = NULL。値: NPIPE、TCPIP、SSL
- この変数は、データベース・マネージャーを開始したときに開始されるコミュニケーション・マネージャーを指定します。この変数を指定しないと、サーバーではどの DB2 コミュニケーション・マネージャーも開始されません。

DB2FCMCOMM

- オペレーティング・システム: サポートされる DB2 Enterprise Server Edition プラットフォームのすべて
- デフォルト = TCPIP4。値: TCPIP4 または TCPIP6
- この変数は、db2nodes.cfg ファイル内のホスト名がどのように解決されるかを指定します。すべてのホスト名は IPv4 または IPv6 として解決されます。ホスト名の代わりに IP アドレスが db2nodes.cfg に指定されている場合は、IP の形式から IPv4 が使用されているか IPv6 が使用されているかがわかります。 **DB2FCMCOMM** が設定されていない場合、IPv4 のデフォルト設定値は、IPv4 ホストのみ開始できることを意味します。

  注: db2nodes.cfg で指定されたホスト名から解決した IP の形式や db2nodes.cfg で直接指定されている IP の形式が **DB2FCMCOMM** の設定と一致しない場合は、db2start が失敗します。

DB2_FORCE_NLS_CACHE

- オペレーティング・システム: AIX、HP_UX、Solaris
- デフォルト = FALSE。値: TRUE または FALSE
- この変数は、マルチスレッド・アプリケーションにおいてロック競合が起きないようにするために使用します。レジストリー変数が TRUE の場合、スレッドが初めてコード・ページとテリトリー・コードの情報にアクセスする際にそれらの情報が保管されます。その時点以降、この情報を要求する他のスレッドにこのキャッシュ情報が使用されます。したがってロック競合は除かれ、特定の状態下でパフォーマンスが向上することになります。アプリケーションにより接続間のロケール設定が変更される場合、この設定値は使用できません。マルチスレッド・アプリケーションでロケール設定を変更するのはスレッド・セーフ ではないので普通はこの変更は行われません。したがってこのような状態は考慮する必要がないと思われます。

DB2RSHCMD

- オペレーティング・システム: UNIX

- デフォルト = rsh (HP-UX では remsh)。値は rsh、remsh、または ssh までの絶対パス名です。
- デフォルトでは、DB2 データベース・システムはリモート・データベース・パーティションの開始時に rsh を通信プロトコルとして使用し、db2_all スクリプトを使用してすべてのデータベース・パーティションでユーティリティーおよびコマンドを実行します。 例えば、このレジストリー変数を ssh の絶対パス名に設定すると、 DB2 データベース製品は要求されたユーティリティーおよびコマンドを実行するための通信プロトコルとして ssh を使用します。また、適切なデフォルトのパラメーターを使用して、リモート・コマンド・プログラムを呼び出すスクリプトの絶対パス名に設定することもできます。この変数が必要なのは、パーティション・データベースの場合、または DB2 製品のインストール先とは異なるサーバーから db2start コマンドを実行する単一パーティション環境の場合に限られます。 インスタンス所有者は、指定されたリモート・シェル・プログラムを使用して、各 DB2 データベース・ノードから他の DB2 データベース・ノードへ、追加の検査または認証 (つまり、パスワードまたはパスワード句) を要求するプロンプトが出されることなく、ログインできなければなりません。

DB2RSHCMD レジストリー変数を設定して ssh シェルを DB2 と共に使用する方法については、「Configure DB2 Universal Database for UNIX to use OpenSSH」というホワイト・ペーパーを参照してください。

DB2RSHTIMEOUT

- オペレーティング・システム: UNIX
- デフォルト = 30 秒。値: 1 から 120
- この変数が適用できるのは、**DB2RSHCMD** が非 NULL 値に設定されている場合のみです。このレジストリー変数を使用して、DB2 データベース・システムが任意のリモート・コマンドを待機するタイムアウト期間を制御します。このタイムアウト期間が過ぎても応答を受け取らない場合には、リモート・データベース・パーティションに到達できず、操作が失敗したとみなされます。

  注: 指定される時間値は、リモート・コマンドを実行するために必要な時間ではありません。要求を認証するために必要な時間です。

DB2SORCVBUF

- オペレーティング・システム: すべて
- デフォルト = 65 536
- TCP/IP 受信バッファーの値を指定します。

DB2SOSNDBUF

- オペレーティング・システム: すべて
- デフォルト = 65 536
- TCP/IP 送信バッファーの値を指定します。

DB2TCP_CLIENT_CONTIMEOUT

- オペレーティング・システム: すべて、ただしクライアントのみ
- デフォルト = 0 (タイムアウトなし)。値: 0 から 32 767 秒

- **DB2TCP_CLIENT_CONTIMEOUT** レジストリー変数は、クライアントが TCP/IP 接続操作の完了を待つ秒数を指定します。指定された秒数内に接続が確立されない場合、DB2 データベース・マネージャーはエラー `-30081 selectForConnectTimeout` を戻します。

  レジストリー変数が設定されていないか、または 0 に設定されている場合は、タイムアウトはありません。

  注: オペレーティング・システムにも接続タイムアウト値があります。これは、**DB2TCP_CLIENT_CONTIMEOUT** を使って設定したタイムアウトより前に実施される場合があります。 例えば、AIX はデフォルト *tcp_keepinit*=150 (0.5 秒単位) を持ち、75 秒後に接続が終了します。

DB2TCP_CLIENT_RCVTIMEOUT

- オペレーティング・システム: すべて、ただしクライアントのみ
- デフォルト = 0 (タイムアウトなし)。値: 0 から 32 767 秒
- **DB2TCP_CLIENT_RCVTIMEOUT** レジストリー変数は、クライアントが TCP/IP 上のデータを受信する操作を待つ秒数を指定します。指定された秒数内にサーバーからデータが受信されない場合、DB2 データベース・マネージャーはエラー `-30081 selectForRecvTimeout` を戻します。

  レジストリー変数が設定されていないか、または 0 に設定されている場合は、タイムアウトはありません。

  注: DB2TCP_CLIENT_RCVTIMEOUT の設定は、CLI 受信タイムアウト設定でオーバーライドできます。

DB2TCPCONNMGRS

- オペレーティング・システム: すべて
- デフォルト = 1 (シリアル・マシンの場合); 最大で 16 つの接続マネージャーに切り上げられたプロセッサー数の平方根 (SMP マシンの場合)。値: 1 から 16
- このレジストリー変数が設定されていない場合、デフォルトの数の接続マネージャーが作成されます。このレジストリー変数が設定されていれば、ここで割り当てた値がデフォルト値を上書きします。指定された数 (最大 16) の TCP/IP 接続マネージャーが作成されます。 1 より小さい値が指定された場合、 **DB2TCPCONNMGRS** には値 1 が設定され、値が範囲外であることを示す警告がログに記録されます。 16 より大きい値が指定された場合、 **DB2TCPCONNMGRS** には値 16 が設定され、値が範囲外であることを示す警告がログに記録されます。 1 から 16 の値を指定した場合、その値がそのまま使用されます。複数の接続マネージャーが作成されれば、複数のクライアント接続を同時に受け取る場合の接続スループットが向上するはずです。ユーザーが SMP マシン上で実行している場合、または **DB2TCPCONNMGRS** レジストリー変数を変更した場合、追加の TCP/IP 接続マネージャー・プロセス (UNIX の場合) またはスレッド (Windows オペレーティング・システムの場合) がある場合があります。追加のプロセスまたはスレッドでは、追加のストレージを必要とします。

注: 接続マネージャーの数を 1 に設定すると、多くのユーザーを持つ、または頻繁に接続と切断が繰り返される、あるいはその両方が原因でシステムのリモート接続上でパフォーマンスの低下が生じます。

コマンド行変数

DB2BQTIME

- オペレーティング・システム: すべて
- デフォルト = 1 秒。最大値: 1 秒
- この変数は、コマンド行プロセッサー・フロントエンドが、バックエンド・プロセスがアクティブであるかどうかを検査してバックエンド・プロセスへの接続を確立する前にスリープする時間を指定します。

DB2BQTRY

- オペレーティング・システム: すべて
- デフォルト = 60 再試行。最小値: 0 再試行
- この変数は、コマンド行プロセッサー・フロントエンド・プロセスが、バックエンド・プロセスがすでにアクティブであるかどうかを判別しようとする回数を指定します。これは、**DB2BQTIME** と一緒に働きます。

DB2_CLPPROMPT

- オペレーティング・システム: すべて
- デフォルト=なし (定義されていない場合、デフォルトの CLP 対話式プロンプトとして「db2 =>」が使用されます)。値: %i、%d、%ia、%da、または %n のトークンをゼロ個以上含み、100 未満の長さの任意のテキスト・ストリング。デフォルト CLP 対話式プロンプト (db2 =>) を明示的に変更する必要がない限り、この変数をユーザーが設定する必要はありません。
- このレジストリー変数を使用すると、コマンド行プロセッサー (CLP) 対話式モードで使用されるプロンプトをユーザーが定義できます。この変数は、オプション・トークン %i、%d、%ia、%da、または %n をゼロ個以上含み、長さが 100 文字未満の任意のテキスト・ストリングに設定できます。CLP 対話式モードで実行する場合、使用するプロンプトは **DB2_CLPPROMPT** レジストリー変数に指定された文字列を取り、トークン %i、%d、%ia、%da、または %n をそれぞれ、現行アタッチ・インスタンスのローカル別名、現行データベース接続のローカル別名、現行アタッチ・インスタンスの許可 ID、現行データベース接続の許可 ID、および改行 (すなわち復帰) で置き換えることによって構成されます。

  注:

  1. **DB2_CLPPROMPT** レジストリー変数が CLP 対話モード内で変更された場合、 CLP 対話モードが閉じて再オープンされるまで、**DB2_CLPPROMPT** の新しい値は有効になりません。
  2. インスタンス接続が存在していない場合は、 %ia は空ストリングで置き換えられ、 %i は **DB2INSTANCE** レジストリー変数の値で置き換えられます。 Windows プラットフォームの場合のみ、**DB2INSTANCE** 変数が設定されていない場合、 %i が **DB2INSTDEF**

レジストリー変数の値によって置き換えられます。これらの変数のいずれも設定されていない場合は、%i が空ストリングで置き換えられます。

3. データベース接続が存在していない場合、 %da は空ストリングで置き換えられ、 %d は **DB2DBDFT** レジストリー変数の値で置き換えられます。 **DB2DBDFT** 変数が設定されていない場合、 %d は空ストリングで置き換えられます。
4. 対話式入力プロンプトは常に、許可 ID、データベース名、およびインスタンス名を大文字で表します。

DB2IQTIME

- オペレーティング・システム: すべて
- デフォルト = 5 秒。最小値: 1 秒
- この変数は、コマンド行プロセッサー・バックエンド・プロセスが、フロントエンド・プロセスがコマンドを渡すのを入力キューで待機する時間を指定します。

DB2RQTIME

- オペレーティング・システム: すべて
- デフォルト = 5 秒。最小値: 1 秒
- この変数は、コマンド行プロセッサー・バックエンド・プロセスが、フロントエンド・プロセスからの要求を待機する時間を指定します。

パーティション・データベース環境変数

DB2CHGPWD_EEE

- オペレーティング・システム: DB2ESE on AIX、Linux、およびWindows
- デフォルト = NULL。値: YES または NO
- この変数は、AIX または Windows ESE システムでのパスワード変更を他のユーザーに許可するかどうかを指定します。すべてのデータベース・パーティションまたはノードのパスワードは、 Windows ドメイン・コントローラ (Windows の場合) または LDAP (AIX の場合) を使って集中保守する必要があります。集中保守しないと、すべてのデータベース・パーティションまたはノード間でパスワードが一致しなくなるおそれがあります。その結果、ユーザーが変更を加えるために接続するデータベース・パーティションでのみパスワードが変更される可能性があります。

DB2_NUM_FAILOVER_NODES

- オペレーティング・システム: すべて
- デフォルト = 2。値: 0 からデータベース・パーティションの必要数まで
- **DB2_NUM_FAILOVER_NODES** を設定して、フェイルオーバーの際にマシン上に開始する必要があるであろう追加のデータベース・パーティションの数を指定します。

  DB2 データベースの高可用性ソリューションでは、データベース・サーバーで障害が発生した際に、障害が発生したマシンのデータベース・パーティションを別のマシンで再始動できます。高速コミュニケーション・マ

ネージャー (FCM) は、**DB2_NUM_FAILOVER_NODES** を使用して、このフェイルオーバーを機能させるために各マシンでどれほどのメモリーを予約しておくかを計算します。

例えば、次の構成を考えてみましょう。

– マシン A には 1 と 2 という 2 つのデータベース・パーティションがあります。
– マシン B には 3 と 4 という 2 つのデータベース・パーティションがあります。
– A と B 両方のマシンで **DB2_NUM_FAILOVER_NODES** は 2 に設定されています。

DB2START の際、FCM は、A と B の両方に最大 4 つのデータベース・パーティションを管理できるだけの十分なメモリーを予約して、一方のマシンで障害が発生しても、もう一方のマシンで障害が発生したマシンの 2 つのデータベース・パーティションを再始動できるようにしておきます。マシン A で障害が発生した場合は、マシン B でデータベース・パーティション 1 と 2 を再始動できます。マシン B で障害が発生した場合は、マシン A でデータベース・パーティション 3 と 4 を再始動できます。

DB2_PARTITIONEDLOAD_DEFAULT

- オペレーティング・システム: サポートされる ESE プラットフォームのすべて
- デフォルト = YES。値: YES または NO
- **DB2_PARTITIONEDLOAD_DEFAULT** レジストリー変数によって、ESE に特定のロード・オプションを指定しない場合、ユーザーは ESE 環境内のロード・ユーティリティーのデフォルト動作を変更できます。デフォルト値は YES であり、これは ESE 環境で ESE に特定のロード・オプションを指定しない場合に、ロードがターゲット表が定義されているすべてのデータベース・パーティション上で試行されることを指定します。値が NO であると、ロードはロード・ユーティリティーが現在接続されているデータベース・パーティション上だけで試行されます。

  注: この変数は、推奨されておらず、今後のリリースでは除去される可能性があります。 LOAD コマンドに、同じ動作を実現するために使用できるさまざまなオプションがあります。次のコマンドを LOAD コマンドとともに指定することにより、この変数に NO を設定した場合と同じ結果を得ることができます。`PARTITIONED DB CONFIG MODE LOAD_ONLY OUTPUT_DBPARTNUMS x`。ここで、x はデータをロードするパーティションのパーティション番号です。

DB2PORTRANGE

- オペレーティング・システム: Windows
- 値: nnnn:nnnn
- この値は、FCM によって使用される TCP/IP ポート範囲に設定されるので、別のマシン上に作成される追加のデータベース・パーティションも同じポート範囲になります。

照会コンパイラー変数

DB2_ANTIJOIN

- オペレーティング・システム: すべて
- デフォルト = NO (ESE 環境)、デフォルト = YES (非 ESE 環境)、値: YES、NO、または EXTEND
- DB2 Enterprise Server Edition の場合に YES が指定されていると、オプティマイザーは、「NOT EXISTS」副照会を DB2 がより効率的に処理できるアンチ結合に変換する機会を探ります。非 ESE 環境で NO が指定されていると、オプティマイザーは「NOT EXISTS」副照会をアンチ結合に変換する機会を制限します。

  ESE と NON-ESE の両方の環境で、EXTEND を指定すると、オプティマイザーは「NOT IN」と「NOT EXISTS」の両方の副照会をアンチ結合に変換する機会を探ります。

DB2_INLIST_TO_NLJN

- オペレーティング・システム: すべて
- デフォルト = NO。値: YES または NO
- 状況によっては、SQL および XQuery コンパイラーは IN リスト述部を結合に書き換えることができます。たとえば、次のような照会は、

  ```
  SELECT *
   FROM EMPLOYEE
   WHERE DEPTNO IN ('D11', 'D21', 'E21')
  ```

  次のように書くことができます。

  ```
  SELECT *
   FROM EMPLOYEE, (VALUES 'D11', 'D21', 'E21) AS V(DNO)
   WHERE DEPTNO = V.DNO
  ```

  DEPTNO に索引がある場合、この書き換えはより良いパフォーマンスを提供することがあります。値のリストが最初にアクセスされて、結合述部に適用する索引を使用して NESTED LOOP 結合で EMPLOYEE に結合されます。

  場合によっては、オプティマイザーが照会の書き換えに最適な結合メソッドを判別するための正確な情報を持っていないことがあります。 IN リストにパラメーター・マーカーまたはホスト変数が含まれる場合に、このことが生じることがあります。これらはオプティマイザーがカタログ統計を使用して選択度を判別することを妨げます。このレジストリー変数は、ホスト変数やパラメーター・マーカーを使った場合でも、オプティマイザーが値のリストを結合するために、 IN リストを結合内の内部表として与える表を使用して、NESTED LOOP 結合を優先的に使用するようにします。

  注: DB2 照会コンパイラー変数 **DB2_MINIMIZE_LISTPREFETCH** および **DB2_INLIST_TO_NLJN** の両方またはいずれかが YES に設定されると、REOPT(ONCE) が指定されていても、アクティブ状態のままになります。

DB2_LIKE_VARCHAR

- オペレーティング・システム: すべて
- デフォルト = Y,Y
- サブエレメント統計の使用を制御します。これらの統計は、データにブランクで区切られた一連のサブフィールドまたはサブエレメント形式の構造がある場合、列内のデータ内容に関する統計です。サブエレメント統計の収集はオプションで、RUNSTATS コマンドまたは API 内のオプションによって制御されます。

  この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 は文字のストリングです。

  このレジストリー変数の使用方法を示す構文は次のとおりです。

  ```
  db2set DB2_LIKE_VARCHAR=[Y|N|S|num1] [,Y|N|S|num2]
  ```

  説明

  – コンマの前にある項、または述部の右辺で唯一の項は、以下の意味になります。ただしこれが有効なのは、2 番目の項が N に指定されているか、または列に正の値のサブエレメント統計がない場合のみです。
    - S - オプティマイザーは % 文字で囲まれたストリングの長さに基づいて、列を形成するために連結する一連のエレメントの各エレメントの長さを見積もる。
    - Y - デフォルト。アルゴリズム・パラメーターのデフォルト値 1.9 を使用する。アルゴリズム・パラメーターで可変長サブエレメント・アルゴリズムを使用する。
    - N - 固定長サブエレメント・アルゴリズムを使用する。
    - num1 - 可変長サブエレメント・アルゴリズムにより、アルゴリズム・パラメーターとして num1 の値を使用する。

  – コンマの後の項は以下のような意味がありますが、正の値のサブエレメント統計を持つ列に対してのみです。
    - N - サブエレメント統計を使用しない。最初の用語が有効になります。
    - Y - デフォルト。正の値のサブエレメント統計を持つ列の場合に、アルゴリズム・パラメーターのデフォルト値 1.9 と一緒にサブエレメント統計を使用する可変長サブエレメント・アルゴリズムを使用する。
    - num2 - 正の値のサブエレメント統計を持つ列の場合に、アルゴリズム・パラメーターとして num2 の値と一緒にサブエレメント統計を使用する可変長サブエレメント・アルゴリズムを使用する。

DB2_MINIMIZE_LISTPREFETCH

- オペレーティング・システム: すべて
- デフォルト = NO。値: YES または NO
- リスト・プリフェッチは特殊な表アクセス方式です。索引から対象データのあるRID を検索して、それらをページ番号でソートしてからデータ・

ページをプリフェッチします。場合によっては、オプティマイザーがリスト・プリフェッチが良いアクセス方式であるかどうかを判別するための正確な情報を持っていないことがあります。オプティマイザーがカタログ統計を使用して選択度を判別することを妨げるパラメーター・マーカーまたはホスト変数が、述部選択度に含まれる場合にこのことが生じることがあります。

このレジストリー変数は、そのような状況でオプティマイザーがリスト・プリフェッチを検討することを禁止します。

注: DB2 照会コンパイラー変数 **DB2_MINIMIZE_LISTPREFETCH** および **DB2_INLIST_TO_NLJN** の両方またはいずれかが YES に設定されると、REOPT(ONCE) が指定されていても、アクティブ状態のままになります。

DB2_NEW_CORR_SQ_FF

- オペレーティング・システム: すべて
- デフォルト = OFF。値: ON または OFF
- ON に設定すると、照会オプティマイザーが特定の副照会述部について計算した選択度の値に影響します。このパラメーターを使用すると、副照会の SELECT リストで MIN または MAX 集約関数を使用する等価副照会述部の選択値の正確度を高めることができます。例:

```
SELECT * FROM T WHERE
T.COL = (SELECT MIN(T.COL)
FROM T WHERE ...)
```

DB2_OPT_MAX_TEMP_SIZE

- オペレーティング・システム: すべて
- デフォルト= NULL。値: すべての TEMPORARY 表スペースで照会が使用できるスペースの量 (メガバイト単位)
- TEMPORARY 表スペースで照会が使用できるスペースの量を制限します。 **DB2_OPT_MAX_TEMP_SIZE** を設定すると、オプティマイザーはこれを設定しない場合に比べて高コストのプランを選択する可能性がありますが、そのプランが TEMPORARY 表スペースで使用するスペースは小さくなります。 **DB2_OPT_MAX_TEMP_SIZE** を設定する場合は、TEMPORARY 表スペースの使用を制限する必要性と、これを設定したために選択されるプランの効率のバランスを取るようにしてください。

  DB2_WORKLOAD=SAP に設定すると、**DB2_OPT_MAX_TEMP_SIZE** は自動的に 10240 (10 GB) に設定されます。

  DB2_OPT_MAX_TEMP_SIZE に設定した値を超える TEMPORARY 表スペースを使用する照会を実行する場合、照会は失敗しませんが、すべてのリソースが使用可能とは限らないので最適パフォーマンスにならない可能性があるという警告が出されます。

  オプティマイザーが検討する操作のうち、**DB2_OPT_MAX_TEMP_SIZE** で設定された制限の影響を受けるものは、以下のとおりです。

  – ORDER BY、DISTINCT、GROUP BY、MERGE SCAN 結合、NESTED LOOP 結合などの操作での明示的ソート。

– 明示的一時表
– ハッシュ結合および DUPLICATE MERGE 結合での暗黙的一時表

DB2_REDUCED_OPTIMIZATION

- オペレーティング・システム: すべて
- デフォルト = NO。値: NO、YES、任意の整数、DISABLE、NO_SORT_NLJOIN、または NO_SORT_MGJOIN
- このレジストリー変数によって、最適化機能を削減したり、最適化機能を指定した最適化レベルに固定して使用するように要求することができます。使用される最適化手法の数を削減する場合、最適化の際に使用される時間およびリソースも削減されます。

  注: 最適化に必要な時間とリソースをより削減できるかもしれませんが、一方で最適ではないアクセス・プランが生成されるリスクは増えます。このレジストリー変数は、IBM またはそのパートナーから指示された場合にのみ使用します。

  – NO に設定した場合

    オプティマイザーは最適化手法を変更しません。

  – YES に設定した場合

    最適化レベルが 5 (デフォルト) もしくは 5 未満の場合に、通常はより良いアクセス・プランを生成することがなく、相当量の準備時間とリソースを消費する、いくつかの最適化手法をオプティマイザーは使用不可にします。

    最適化レベルがちょうど 5 の場合、さらにいくつかの手法を制限または使用不可にします。その結果、オプティマイザーによる最適化に必要な時間とリソースをより削減できるかもしれませんが、一方で最適ではないアクセス・プランが生成されるリスクは増えます。最適化レベルが 5 未満の場合、これらの技法のいくつかは最初から無効であることがあります。しかしそれらが有効であれば、有効のままとなります。しかしそれらが有効であれば、有効のままとなります。

  – 任意の整数に設定した場合

    YES と同じ効果があり、さらにレベル 5 で最適化された動的に準備された照会のために以下の追加の動作が伴います。いずれかの照会ブロック内にある結合の合計数が設定値を超える場合、上記のレベル 5 最適化レベルについての説明で示したような、追加の最適化手法を使用不可にする代わりに、オプティマイザーは貪欲型結合列挙に切り替えます。これは、照会が最適化レベル 2 に類似したレベルで最適化されることを暗黙に示します。

  – DISABLE に設定した場合

    最適化レベル 5 での動的照会の時には、この **DB2_REDUCED_OPTIMIZATION** 変数の指定がなくとも、オプティマイザーは動的に最適化レベルを下げることがあります。この設定値はこの動作を使用不可にして、オプティマイザーがレベル 5 の最適化を完全に実行することを要求します。

– NO_SORT_NLJOIN に設定した場合

オプティマイザーは、NESTED LOOP 結合 (NLJOIN) の場合に、強制的にソートを行う照会プランは生成しません。このようなタイプのソートは、パフォーマンスの改善に役立つことがあります。ということは、NO_SORT_NLJOIN オプションを使用するときは、パフォーマンスに大きく影響することがあるので注意が必要です。

– NO_SORT_MGJOIN に設定した場合

オプティマイザーは、MERGE SCAN 結合 (MSJOIN) の場合に、強制的にソートを行う照会プランは生成しません。このようなタイプのソートは、パフォーマンスの改善に役立つことがあります。ということは、NO_SORT_MGJOIN オプションを使用するときは、パフォーマンスに大きく影響することがあるので注意が必要です。

最適化レベル 5 での動的な最適化レベルの引き下げは、**DB2_REDUCED_OPTIMIZATION** を YES に設定したときの最適化レベルがちょうど 5 の場合について説明された動作、および整数の設定値について説明された動作よりも優先されることに注意してください。

DB2_SELECTIVITY

- オペレーティング・システム: すべて
- デフォルト = NO。値: YES または NO
- このレジストリー変数は、 SQL ステートメント内の検索条件で、SELECTIVITY 文節が使用できる場所を制御します。

  このレジストリー変数が YES に設定された場合、以下の述部に SELECTIVITY 節を設定可能です。
  – 少なくとも 1 つの式がホスト変数を含む基本述部
  – 一致条件、述部条件、またはエスケープ条件にホスト変数が含まれる LIKE 述部

DB2_SQLROUTINE_PREPOPTS

- オペレーティング・システム: すべて
- デフォルト = 空ストリング。値は以下のようになります。
  – BLOCKING {UNAMBIG | ALL | NO}
  – DATETIME {DEF | USA | EUR | ISO | JIS | LOC}
  – DEGREE {1 | *degree-of-parallelism* | ANY}
  – DYNAMICRULES {BIND | INVOKEBIND | DEFINEBIND | RUN | INVOKERUN | DEFINERUN}
  – EXPLAIN {NO | YES | ALL}
  – EXPLSNAP {NO | YES | ALL}
  – FEDERATED {NO | YES}
  – INSERT {DEF | BUF}
  – ISOLATION {CS | RR | UR | RS | NC}
  – QUERYOPT *最適化レベル*

– REOPT {NONE | ONCE | ALWAYS}
– VALIDATE {RUN | BIND}

- **DB2_SQLROUTINE_PREPOPTS** レジストリー変数を使用すると、 SQL および XQuery プロシージャーのプリコンパイル・オプションと BIND オプションをカスタマイズできます。この変数を設定する際は、次のように、各オプションをスペースで区切ります。

  ```
  db2set DB2_SQLROUTINE_PREPOPTS="BLOCKING ALL VALIDATE RUN"
  ```

  各オプションとその設定についての完全な説明は、『BIND コマンド』を参照してください。

個々のプロシージャーを選択して、インスタンスを再始動させずに **DB2_SQLROUTINE_PREPOPTS** と同じ結果を得るためには、SET_ROUTINE_OPTS プロシージャーを使用します。

パフォーマンス変数

DB2_ALLOCATION_SIZE

- オペレーティング・システム: すべて
- デフォルト= 128 KB。範囲: 64 KB から 256 MB
- バッファー・プールのメモリー割り振りのサイズを指定します。

  このレジストリー変数に高い値を設定することの潜在的な利点は、バッファー・プールの希望のメモリー量に達するために必要な割り振りの回数が少なくてすむということです。

  このレジストリー変数に高い値を設定する際にかかる潜在的な損失は、バッファー・プールが割り振りサイズの倍数で変更されない場合にメモリーが無駄になる点です。たとえば、**DB2_ALLOCATION_SIZE** の値が 8 MB で、バッファー・プールが 4 MB ずつ削減される場合、8 MB セグメント全体は解放できないので、この 4 MB は無駄になります。

注: DB2_ALLOCATION_SIZE は、推奨されておらず、今後のリリースでは除去される可能性があります。

DB2_APM_PERFORMANCE

- オペレーティング・システム: すべて
- デフォルト = OFF。値: ON または OFF
- 照会キャッシュ (パッケージ・キャッシュ) の動作に影響するアクセス・プラン・マネージャー (APM) 内でパフォーマンス関連の変更を使用可能にするには、この変数を ON に設定します。 これらの設定値は通常、実動システムには勧められません。これらはパッケージ外キャッシュ・エラー、メモリー使用量の増加、またはその両方など、いくらかの制限を生じさせます。

  DB2_APM_PERFORMANCE を ON に設定すると、NO PACKAGE LOCK モードも使用可能になります。このモードは、グローバル照会キャッシュがパッケージ・ロックを使用しないで操作できるようにします。パッケージ・ロックは、キャッシュされたパッケージ項目が除去されない

ように保護する内部システム・ロックです。 NO PACKAGE LOCKモードの場合、パフォーマンスはいくらか改善されることがありますが、一部のデータベース操作はできなくなります。これらの禁止される操作には、パッケージを無効にする操作、パッケージを操作不能にする操作、PRECOMPILE、BIND、および REBIND が含まれます。

DB2ASSUMEUPDATE

- オペレーティング・システム: すべて
- デフォルト = OFF。値: ON または OFF
- 使用可能になっていると、この変数により DB2 データベース・システムは、UPDATE ステートメントで提供されるすべての固定長の列が変更中であると想定することができます。これにより、DB2 データベース・システムが既存の列値と新規値を比較して列が実際に変更中かどうかを判別する必要がなくなります。列が更新用に (たとえば SET 文節で) 提供されており、しかし実際にはその列が変更されていないときにこのレジストリー変数を使用すると、ロギングや索引の保守が余分に発生する可能性があります。

  DB2ASSUMEUPDATE レジストリー変数の活動化は、db2start コマンド上では有効です。

DB2_ASYNC_IO_MAXFILOP

- オペレーティング・システム: すべて
- デフォルト = *maxfilop* 構成パラメーターの値。値: *maxfilop* の値から *max_int* の値
- **DB2_ASYNC_IO_MAXFILOP** は、推奨されておらず、今後のリリースでは除去される可能性があります。この変数は、スレッド化されたデータベース・マネージャーによって保守される共用ファイル・ハンドル表のゆえに、廃止されました。詳しくは、共有ファイル・ハンドル表を参照してください。

  DB2_ASYNC_IO_MAXFILOP は、バージョン 9.5 で設定することはできますが、効果はありません。それぞれのデータベースごとにオープンしておけるファイル・ハンドルの数を制限するには、 *maxfilop* 構成パラメーターを参照してください。

DB2_AVOID_PREFETCH

- オペレーティング・システム: すべて
- デフォルト = OFF。値: ON または OFF
- クラッシュ・リカバリーにおいて、プリフェッチを使用するかを指定します。 **DB2_AVOID_PREFETCH** =ON の場合は、プリフェッチは使用されません。

DB2BPVARS

- オペレーティング・システム: 各パラメーターに指定されたとおり
- デフォルト = パス
- 2 種類のパラメーターを使用してバッファー・プールを調整できます。パラメーターの 1 つの種類は Windows だけで使用可能なもので、バッファー・プールが特定のタイプのコンテナーの分散読み取りを使用すること

を指定します。他の種類のパラメーターは、すべてのプラットフォームで使用可能なもので、プリフェッチ動作に影響を与えます。

複数のパラメーターは ASCII ファイル内で、各行に 1 つずつ parameter=value の形式で指定できます。例として、bpvars.vars という名前のファイルには以下の行が含まれます。

```
NO_NT_SCATTER = 1
NUMPREFETCHQUEUES = 2
```

bpvars.vars が F:¥vars¥ に保管されていると想定すると、これらの変数を設定するには以下のコマンドを実行します。

```
db2set DB2BPVARS=F:¥vars¥bpvars.vars
```

分散読み取りパラメーター

分散読み取りパラメーターは、それぞれのコンテナー・タイプに対する順次プリフェッチが大量に行われ、 **DB2NTNOCACHE** をすでに ON に設定したシステムで推奨されます。 これらのパラメーターは、Windows プラットフォームだけで使用可能であり、NT_SCATTER_DMSFILE、NT_SCATTER_DMSDEVICE、および NT_SCATTER_SMS です。 NO_NT_SCATTER パラメーターを指定すると、すべてのコンテナーで分散読み取りを明示的に禁止します。 指定の種類のすべてのコンテナーで分散読み取りをオンに切り替えるには、特定のパラメーターを使用します。上記の個々のパラメーターのデフォルトはゼロ (OFF) で、可能な値はゼロ (OFF) および 1 (ON) です。

注: 分散読み取りをオンに切り替えることができるのは、**DB2NTNOCACHE** を ON に設定して Windows ファイルのキャッシングをオフにした場合だけです。 **DB2NTNOCACHE** の設定が OFF に設定されているか未設定の場合は、コンテナーの分散読み取りを試みると管理通知ログに警告メッセージが書き込まれ、分散読み取りは使用不可のままになります。

プリフェッチ調整パラメーター

プリフェッチ調整パラメーターは、NUMPREFETCHQUEUES および PREFETCHQUEUESIZE です。これらのパラメーターはどのプラットフォームでも使用可能であり、バッファー・プール・データ・プリフェッチの改善のために使用することができます。例えば、目的の PREFETCHSIZE が複数の PREFETCHSIZE/EXTENTSIZE プリフェッチ要求に分割されている順次プリフェッチについて考えます。 この場合、要求はプリフェッチ・キューに入れられ、そこから入出力サーバーがディスパッチされて非同期入出力を実行します。デフォルトでは、DB2 データベース・マネージャーはデータベース・パーティションごとにサイズ max(200,2*NUM_IOSERVERS) の 1 つのキューを保守します。一部の環境では、キューの増加またはキューのサイズの変更、もしくはその両方により、パフォーマンスが改善されます。プリフェッチ・キューの数は最大で入出力サーバーの数の半分とします。これらのパラメーターを設定するとき、現行ユーザー数などのワークロード特性と同様に、

PREFETCHSIZE、EXTENTSIZE、NUM_IOSERVERS などのパラメーター、およびバッファー・プール・サイズを検討してください。

デフォルト値が環境に対して小さすぎると考えられる場合、最初に少しだけ値を増加します。例えば、NUMPREFETCHQUEUES=4 および PREFETCHQUEUESIZE=200 と設定できます。これらのパラメーターの変更を制御された方法で行い、変更の効果を観察および評価できるようにします。

NUMPREFETCHQUEUES では、デフォルトは 1 で、値の範囲は 1 から NUM_IOSERVERS となります。 NUMPREFETCHQUEUES を 1 未満に設定した場合、それは 1 に調整されます。その値を NUM_IOSERVERS よりも大きく設定した場合、それは NUM_IOSERVERS に調整されます。

PREFETCHQUEUESIZE の場合、デフォルト値は max (200,2*NUM_IOSERVERS) です。値の範囲は 1 から 32767 です。 PREFETCHQUEUESIZE を 1 未満に設定した場合、それはデフォルト値に調整されます。 その値を 32 767 より大きく設定した場合、それは 32 767 に調整されます。

注: DB2BPVARS は、推奨されておらず、今後のリリースでは除去される可能性があります。

DB2CHKPTR

- オペレーティング・システム: すべて
- デフォルト = OFF。値: ON または OFF
- 入力のポインター検査が必要であるかどうかを指定します。

DB2CHKSQLDA

- オペレーティング・システム: すべて
- デフォルト = ON。値: ON または OFF
- 入力の SQLDA 検査が必要であるかどうかを指定します。

DB2_EVALUNCOMMITTED

- オペレーティング・システム: すべて
- デフォルト = OFF。値: ON、OFF
- 使用可能になっていると、この変数は、可能な場合に、表アクセス・スキャンまたは索引アクセス・スキャンが、データ・レコードが述部評価を満たしたことがわかるまで行ロックを据え置くかまたは回避できるようにします。

  この変数を使用可能にすると、非コミット・データで述部評価が行われる場合があります。

  DB2_EVALUNCOMMITTED は、カーソル固定分離レベルかまたは読み取り固定分離レベルのいずれかを使用するステートメントに対してのみ適用可能です。 索引スキャンの場合、索引はタイプ 2 索引でなければなりません。

さらに、削除された行は表スキャンのアクセス時に無条件でスキップされますが、削除されたキーは、タイプ 2 索引のスキャンでは、レジストリー変数 **DB2_SKIPDELETED** も設定されていなければスキップされません。

DB2_EVALUNCOMMITTED レジストリー変数の変更は、db2start コマンドによって有効になります。 据え置きロッキングを適用可能とするかどうかに関する判断は、ステートメントのコンパイル時またはバインド時に行われます。

DB2_EXTENDED_IN_TO_JOIN

- オペレーティング・システム: すべて
- デフォルト = <設定しない> または <NULL> 値。デフォルトの in-to-join およびバイナリー・サーチの両方の機能が使用可能です。値: IN2JOIN_OFF、BINSEARCH_OFF。
- IN 検索では、効率のよい二分木検索アルゴリズムが使用されます。この検索方法はデフォルトで使用可能になっていて、IN リスト述部を指定した照会のコンパイルおよび実行のたびに採用されます。そのため、ユーザーには認識されないことを意図しています。新しい動作によって、特定のまれな照会があった場合に、それ自身のパフォーマンス上の問題が生じる可能性があります。そのような事態を診断して修復するために、環境変数 **DB2_EXTENDED_IN_TO_JOIN** を使用して、IN 検索アルゴリズムの動作を診断することができます。必要があれば、二分木検索の動作を使用不可にすることができます。つまり、IN 検索を従来の検索アルゴリズムに戻します。
- パラメーター IN2JOIN_OFF は、デフォルトの in-to-join 動作を使用不可にして、以前の IN リスト計画機能に戻します。デフォルトのバイナリー・サーチ機能はこれまでどおり使用可能です。 BINSEARCH_OFF は、バイナリー・サーチの検索引数述部インプリメンテーションを使用不可にしますが、in-to-join 機能は使用可能のままになります。この 2 つのパラメーターを任意の順序で使用すれば、デフォルトの in-to-join 動作と、デフォルトのバイナリー・サーチ検索引数述部の両方のインプリメンテーションが使用不可になります。

DB2_EXTENDED_IO_FEATURES

- オペレーティング・システム: AIX
- デフォルト = OFF。値: ON、OFF
- 入出力パフォーマンスを拡張する機能を有効にするには、この変数を ON に設定します。 この拡張の一環として、高優先順位入出力の待ち時間が削減される以外に、メモリー・キャッシュのヒット率も向上します。これらのフィーチャーは、特定の組み合わせのソフトウェアとハードウェア構成の場合のみ使用可能です。他の構成に対してこの変数を ON に設定しても、DB2 データベース管理システムまたはオペレーティング・システムでは無視されます。最小構成要件は、以下のとおりです。
  - データベースのバージョン: DB2 V9.1
  - ロー・デバイスをデータベース・コンテナーに使用する必要があります (ファイル・システム上のコンテナーはサポートされません)。

– オペレーティング・システム: AIX 5.3 TL4
– ストレージ・サブシステム: Shark DS8000™ は、すべての拡張入出力パフォーマンス・フィーチャーをサポートします。セットアップおよび前提条件の詳細は、Shark DS8000 の資料を参照してください。

HIGH、MEDIUM、および LOW のデフォルトの入出力優先順位設定はそれぞれ 3、8、および 12 です。**DB2_IO_PRIORITY_SETTING** レジストリー変数を使用すれば、これらの設定を変更することができます。

DB2_EXTENDED_OPTIMIZATION

- オペレーティング・システム: すべて
- デフォルト = OFF。値: ON、OFF、または ENHANCED_MULTIPLE_DISTINCT
- この変数は、照会パフォーマンスの向上のために、照会オプティマイザーが最適化拡張を使用するかどうかを指定します。 ON および ENHANCED_MULTIPLE_DISTINCT の値は、それぞれ異なる最適化拡張を指定します。両方を使用することを望む場合は、コンマ区切りのリストを使用してください。

  1 つの単一選択操作に複数の特殊集約操作が関与する場合や、データベース・パーティション数に対するプロセッサーの比率が低い (例えば、1 以下の比率) の場合、ENHANCED_MULTIPLE_DISTINCT 値によって照会のパフォーマンスが改善されることがあります。この設定は、対称型マルチプロセッサー (SMP) を持たない DPF (データベース・パーティション機能) 環境で使用してください。

  最適化拡張を使用しても、必ずしもすべての環境で照会のパフォーマンスが向上するわけではありません。それぞれの照会パフォーマンスが向上するかどうかを判別するために、テストを行う必要があります。

DB2_HASH_JOIN

- オペレーティング・システム: すべて
- デフォルト = YES。値: YES または NO
- アクセス・プランをコンパイルするときに、可能な結合メソッドとしてハッシュ結合を指定します。最高のパフォーマンスを得るには、**DB2_HASH_JOIN** を調整する必要があります。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* 構成パラメーターに使用可能なメモリーの最大量を見積もり、それから *sortheap* 構成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値を大きくしてください。ただし *sheapthres* 構成パラメーターで指定した制限に達しないようにします。

  注: DB2_HASH_JOIN は、バージョン 9.5 より推奨されなくなり、今後のリリースでは除去される予定です。

DB2_IO_PRIORITY_SETTING

- オペレーティング・システム: AIX

- 値: HIGH:#,MEDIUM:#,LOW:#。ここで # は 1 から 15 までのいずれかの数値を指定できます。
- この変数は、**DB2_EXTENDED_IO_FEATURES** レジストリー変数と組み合わせて使用します。 このレジストリー変数は、DB2 データベース・システムのデフォルトの HIGH、MEDIUM、および LOW 入出力優先順位設定 (それぞれ3、8、および 12) をオーバーライドする手段を提供します。 このレジストリー変数は、インスタンスの開始前に設定する必要があります。変更した場合は、インスタンスの再始動が必要です。このレジストリー変数を設定するだけでは拡張入出力機能は有効にならず、有効にするには **DB2_EXTENDED_IO_FEATURES** を設定する必要があることに注意してください。 **DB2_EXTENDED_IO_FEATURES** に対するすべてのシステム要件は、このレジストリー変数にも適用されます。

DB2_KEEP_AS_AND_DMS_CONTAINERS_OPEN

- オペレーティング・システム: すべて
- デフォルト: NO。値: YES または NO
- この変数を ON に設定するとき、各 DMS 表スペースのコンテナーには、データベースが非活動状態になるまで開かれたままになるファイル・ハンドルがあります。コンテナーを開くオーバーヘッドが除去されるので、照会のパフォーマンスが向上する可能性があります。このレジストリーは、純粋な DMS 環境でのみ使用してください。そうでない場合、SMS 表スペースに対する照会のパフォーマンスは悪影響を受ける場合があります。

DB2_KEEPTABLELOCK

- オペレーティング・システム: すべて
- デフォルト: OFF。値: ON、TRANSACTION、OFF、CONNECTION
- この変数が ON または TRANSACTION に設定されていると、この変数によって DB2 データベース・システムは、非コミット読み取りまたはカーソル固定のいずれかの分離レベルのカーソルがクローズされるときでも、表ロックについては引き続き保持されるようになります。保持される表ロックは、読み取り固定スキャンまたは反復可能読み取りスキャンの場合に解放されるように、トランザクションの終了時に解放されます。

  この変数が CONNECTION に設定されていると、アプリケーションがトランザクションをロールバックするか、接続がリセットされるまで、表ロックはアプリケーションに対して解放されています。表ロックはコミット間で保持され続け、表ロックをドロップするアプリケーション要求はデータベースによって無視されます。表ロックは、アプリケーションに割り振られたままです。したがって、アプリケーションが表ロックを再び要求するとき、ロックはすでに使用可能です。

  この最適化を活用できるアプリケーションのワークロードの場合、パフォーマンスは向上します。ただし、同時に実行されるその他のアプリケーションのワークロードが、影響を受ける場合があります。その他のアプリケーションは、並行性が低下する結果となる指定された表へのアクセスをブロックされる可能性があります。DB2 SQL のカタログ表は、この設定の

影響を受けません。CONNECTION 設定には、ON または TRANSACTION 設定で説明されている動作も含まれます。

このレジストリー変数はステートメントのコンパイル時またはバインド時にチェックされます。

DB2_LARGE_PAGE_MEM

- オペレーティング・システム: AIX、Linux、Windows Server 2003
- デフォルト = NULL。値: 該当するすべてのメモリー領域がラージ・ページ・メモリーを使用する場合は * を使用します。それ以外の場合は、ラージ・ページ・メモリーを使用する特定のメモリー領域をコンマで区切られたリストで指定します。使用可能な領域はオペレーティング・システムによって異なります。 AIX 上では、 DB、DBMS、FCM、または PRIVATE の領域を指定できます。Linux 上では、DB の領域を指定できます。Windows Server 2003 上では、DB の領域を指定できます。
- **DB2_LARGE_PAGE_MEM** レジストリー変数は、ラージ・ページ・サポートを使用可能にするために使用します。
  DB2_LARGE_PAGE_MEM=DB を設定すると、データベース共有メモリー領域のラージ・ページ・メモリーが使用可能になります。

  ラージ・ページの使用は主に、高性能コンピューティング・アプリケーションのパフォーマンスの向上を意図したものです。集中的なメモリー・アクセスを必要とし、大量の仮想メモリーを使用するアプリケーションでは、このラージ・ページの使用によってパフォーマンスを向上できる場合があります。 DB2 データベース・システムでラージ・ページを使用できるようにするには、まずオペレーティング・システムがラージ・ページを使用できるように構成する必要があります。

  ラージ専用ページを使用可能にすると、DB2 データベース・システムのメモリー使用量がかなり増加します。各 DB2 エージェントが最低 1 つの物理メモリー・ラージ・ページ (16 MB) を消費するためです。 64 ビット DB2 for AIX 上でエージェント専用メモリー用にラージ・ページを使用可能にするには (**DB2_LARGE_PAGE_MEM**=PRIVATE 設定)、オペレーティング・システム上でラージ・ページを構成することに加えて、以下の条件を満たさなければなりません。

  – インスタンス所有者が CAP_BYPASS_RAC_VMM および CAP_PROPAGATE 機能を所有していなければならない。

  AIX の場合、この変数を DB に設定すると、データベース共用メモリーのセルフチューニング (**database_memory** 構成パラメーターを AUTOMATIC に設定することによって活動化される) を使用可能にできなくなります。

  AIX 5L™ の場合は、この変数を FCM に設定することができます。FCM メモリーは独自のメモリー・セット内に置かれるため、**DB2_LARGE_PAGE_MEM** レジストリー変数の値に FCM キーワードを追加して、FCM メモリーがラージ・ページを使用できるようにする必要があります。

Linux の場合は、*libcap.so.1* ライブラリーの可用性に関する追加要件があります。このオプションが有効であるためには、このライブラリーがインストールされていなければなりません。このオプションがオンになっていて、このライブラリーがシステム上にない場合、DB2 データベースは大容量のカーネル・ページを使用不可にして、それらがない場合と同様に機能し続けます。

Linux では、大容量カーネル・ページが使用可能かどうかを検査するために、次のコマンドを発行します。

```
cat /proc/meminfo
```

ラージ・カーネル・ページが使用可能である場合は、次の 3 行が表示されます (サーバー上に構成されているメモリーの量によって数値は異なります)。

```
HugePages_Total:   200
HugePages_Free:    200
Hugepagesize:    16384 kB
```

これらの行が表示されない場合、または HugePages_Total が 0 である場合は、オペレーティング・システムまたはカーネルを構成する必要があります。

Windows では、システム上で使用可能なラージ・ページ・メモリーの量は、すべての使用可能メモリーよりも少なくなります。システムがいくらかの時間稼働した後、メモリーをフラグメント化できるようになり、ラージ・ページ・メモリーの量が減少します。

DB2_LOGGER_NON_BUFFERED_IO

- オペレーティング・システム: すべて
- デフォルト = OFF。値: ON、OFF
- この変数は、ログ・ファイル・システム上での直接入出力を使用可能にします。

DB2_LOGGER_NON_BUFFERED_IO レジストリー変数は、DB2 バージョン 9.5 フィックスパック 1 リリースから使用できるようになりました。

DB2MAXFSCRSEARCH

- オペレーティング・システム: すべて
- デフォルト = 5。値: -1、1 から 33 554
- 表へのレコードの追加時に、検索するフリー・スペース制御レコード (FSCR) の数を指定します。デフォルトでは、5 つの FSCR を検索します。この値の変更は、スペース再利用で挿入速度の平衡を取れるようにします。スペース再利用の最適化のためには大きな値を使用します。挿入速度の最適化のためには小さな値を使用します。値を -1 に設定すると、データベース・マネージャーはすべての FSCR を強制的に検索します。

DB2_MAX_INACT_STMTS

- オペレーティング・システム: すべて
- デフォルト= 設定しない。値: 最大 4 GB

- この変数は、1 つのアプリケーションに保持される非アクティブ・ステートメントの数に関するデフォルトの限度をオーバーライドします。別の値を選択して、非アクティブ・ステートメント情報用に使用されるシステム・モニター・ヒープの量を増減できます。デフォルトの限度は 250 です。

  アプリケーションの作業単位に含まれるステートメント数が非常に多い場合や、多数のアプリケーションが並行して実行されている場合は、システム・モニター・ヒープが使い尽くされることがあります。

DB2_MAX_NON_TABLE_LOCKS

- オペレーティング・システム: すべて
- デフォルト = YES。値: 説明を参照
- この変数は、トランザクションによってすべてが解放される前に持つことのできる、 NON 表ロックの最大数を定義します。 NON 表ロックとは、トランザクションが使用を終えてもハッシュ・テーブルやトランザクション・チェーンに保持されている表ロックのことです。トランザクションが同じ表に何回もアクセスすることはよくあるので、ロックを保持してその状態を NON に変更することでパフォーマンスが改善されます。

  最良の結果を得るためのこの変数の推奨値は、接続によってアクセスが予想される表の数の最大数です。ユーザー定義値が指定されない場合、デフォルト値は以下のとおりです。locklist サイズが次のもの以上の場合、

  ```
  SQLP_THRESHOLD_VAL_OF_LRG_LOCKLIST_SZ_FOR_MAX_NON_LOCKS
  ```

  (現行は 8000)、デフォルト値は次のようになります。

  ```
  SQLP_DEFAULT_MAX_NON_TABLE_LOCKS_LARGE
  ```

  (現行は 150) それ以外の場合、デフォルト値は次のようになります。

  ```
  SQLP_DEFAULT_MAX_NON_TABLE_LOCKS_SMALL
  ```

  (現行は 0)

DB2_MDC_ROLLOUT

- オペレーティング・システム: すべて
- デフォルト=IMMEDIATE。値: IMMEDIATE、OFF、または DEFER
- この変数は、MDC 表からの削除の「ロールアウト」という、パフォーマンス強化を使用可能にします。ロールアウトは、MDC 表内の行を削除する高速の手段です。このとき、すべてのセル (ディメンション値の論理積) は、1 つの検索 DELETE ステートメントで削除されます。この利点は、ロギングが削減され、処理がより効率的に行われることです。
- この変数設定の場合に生じる可能性のある 3 つの結果は次のとおりです。
  - ロールアウトなし - OFF を指定した場合
  - 即時ロールアウト - IMMEDIATE を指定した場合
  - 索引の据え置きクリーンアップ付きのロールアウト - DEFER を指定した場合

- 始動後に値を変更した場合に、ステートメントを新たにコンパイルすると、新しいレジストリー値の設定が順守されます。パッケージ・キャッシュ内にあるステートメントの場合、そのステートメントの再コンパイル時点まで、削除処理中の変更は行われません。 SET CURRENT MDC ROLLOUT MODE ステートメントは、アプリケーション接続レベルの **DB2_MDC_ROLLOUT** の値をオーバーライドします。

DB2MEMDISCLAIM

- オペレーティング・システム: すべて
- デフォルト = YES。値: YES または NO
- DB2 データベース・システム・プロセスが使用するメモリーには、ページング・スペースが関連することがあります。このページング・スペースは、関連するメモリーが解放された後も予約されたままになる場合があります。これは、オペレーティング・システムの (調整可能な) 仮想メモリー管理割り振りポリシーによって異なります。 **DB2MEMDISCLAIM** レジストリー変数は、解放されたメモリーと予約ページング・スペースとの関連付けをオペレーティング・システムが解除することを DB2 エージェントが明示的に要求するかどうかを制御します。

  DB2MEMDISCLAIM を YES に設定すると、ページング・スペースの所要量が少なくなり、場合によっては、ページングに起因するディスク活動も減少します。 **DB2MEMDISCLAIM** を NO に設定すると、ページング・スペースの所要量が増え、場合によっては、ページングに起因するディスク活動も増加します。ページング・スペースが多い場合や、ページングが行われないほど実メモリーが十分にある場合などは、 NO を設定してもパフォーマンスはわずかしか向上しません。

DB2MEMMAXFREE

- オペレーティング・システム: すべて
- デフォルト = NULL。値: 0 から 2^{32}-1 バイト
- 未使用メモリーがオペレーティング・システムに戻される前に、DB2 データベース・システム・プロセスによって保持される未使用の専用メモリーの最大バイト数を指定します。

  DB2MEMMAXFREE が設定されない場合、DB2 データベース・システム・プロセスは、未使用の専用メモリーの最大 20% (現在消費されている専用メモリー容量に基づいて) を確保したうえで、メモリーを解放してオペレーティング・システムに戻します。

注: DB2MEMMAXFREE は、推奨されておらず、今後のリリースでは除去される予定です。現在データベース・マネージャーはスレッド化エンジン・モデルを使用するので、この変数は今後は必要なくなります。この変数を設定しないでください。設定すると、パフォーマンスが低下し、予期しない動作を引き起こす可能性があります。

DB2_MEM_TUNING_RANGE

- オペレーティング・システム: AIX、Windows
- デフォルト = NULL。値: 一連のパーセンテージ n, m (n=*minfree*、m=*maxfree*)

- DB2 インスタンスがフリーにしておく物理メモリーの量は重要です。なぜなら、同一マシンで実行中の他のアプリケーションが使用できるメモリー量は、ここから決まるからです。データベース共用メモリーのセルフチューニングを使用可能にすると、あるインスタンスがフリーにしておく物理メモリーの量は、そのインスタンス (およびそのアクティブ・データベース) でのメモリーの必要性によって決まります。インスタンスは、追加メモリーが緊急に必要になると、システムの空き物理メモリーが *minfree* で指定されたパーセンテージに達するまでメモリーを割り振ります。メモリーをあまり必要としないときは、インスタンスはフリーにしておく物理メモリーの量を、*maxfree* でパーセンテージとして指定された量まで増やします。したがって、要件として、*minfree* に設定する値は *maxfree* の値よりも小さくなければなりません。

  この変数が設定されない場合、DB2 データベース・マネージャーは *minfree* と *maxfree* の値を、サーバーのメモリー量に基づいて計算します。セルフチューニング・メモリー・マネージャー (STMM) を実行している場合に、*database_memory* を AUTOMATIC に設定し、空き物理メモリー量の不足に関連した問題が起きているのでない限り、この変数を設定しない ことをお勧めします。

DB2_MMAP_READ

- オペレーティング・システム: AIX
- デフォルト = OFF。値: ON または OFF
- この変数は、**DB2_MMAP_WRITE** と一緒に使用して、入出力の代替方法として DB2 データベース・システムが mmap を使用できるようにします。

  これらの変数が ON に設定されている場合、 DB2 バッファー・プールとの間で読み書きされるデータは AIX メモリー・キャッシュをバイパスします。 使用する DB2 バッファー・プールが比較的小さい場合は、バッファー・プールのサイズを増やすのではなく、**DB2_MMAP_READ** と **DB2_MMAP_WRITE** を OFF に設定することによって AIX メモリー・キャッシングを活用する必要があります。

DB2_MMAP_WRITE

- オペレーティング・システム: AIX
- デフォルト = OFF。値: ON または OFF
- この変数は、**DB2_MMAP_READ** と一緒に使用して、入出力の代替方法として DB2 データベース・システムが mmap を使用できるようにします。

  これらの変数が ON に設定されている場合、 DB2 バッファー・プールとの間で読み書きされるデータは AIX メモリー・キャッシュをバイパスします。 使用する DB2 バッファー・プールが比較的小さい場合は、バッファー・プールのサイズを増やすのではなく、**DB2_MMAP_READ** と **DB2_MMAP_WRITE** を OFF に設定することによって AIX メモリー・キャッシングを活用する必要があります。

DB2_NO_FORK_CHECK

- オペレーティング・システム: UNIX
- デフォルト = OFF。値: ON または OFF
- この変数を使用可能にすると、DB2 ランタイム・クライアントは、現行プロセスがフォーク呼び出しの結果であるかどうかを判別するチェックを最小化します。これにより、fork() API を使用しない DB2 アプリケーションのパフォーマンスは改善されます。

注: この変数は、推奨されておらず、今後のリリースでは除去される予定です。これが不要になるのは、プロセスを開始または新たに fork するときには、現在のプロセス ID (pid) がキャッシュされるからです。

DB2NTMEMSIZE

- オペレーティング・システム: Windows
- デフォルト = (メモリー・セグメントにより異なる)
- Windows では、プロセス間でアドレスの一致を保証するために DLL 初期設定時にすべての共用メモリー・セグメントを予約する必要があります。必要に応じて **DB2NTMEMSIZE** により Windows の DB2 デフォルトをオーバーライドできます。 ほとんどの状態では、デフォルト値で十分なはずです。メモリー・セグメント、デフォルト・サイズ、およびオーバーライド・オプションは以下のとおりです:

  1. 並列 FCM バッファー: デフォルトのサイズ 512 MB (32 ビット・プラットフォーム)、4.5 GB (64 ビット・プラットフォーム) 。オーバーライド・オプションは FCM:<バイト数> です。
  2. fenced モード・コミュニケーション: デフォルトのサイズ 96 MB (32 ビット・プラットフォーム)、512 MB (64 ビット・プラットフォーム)。オーバーライド・オプションは APLD:<バイト数> です。

  オーバーライド・オプションをセミコロン (;) で区切って、複数のセグメントをオーバーライドできます。例えば、32 ビット・バージョンの DB2 で FCM バッファーを 1 GB に制限し、fenced ストアード・プロシージャーを 256 MB に制限するには、以下を使用します。

  ```
  db2set DB2NTMEMSIZE=FCM:1073741824;APLD:268435456
  ```

DB2NTNOCACHE

- オペレーティング・システム: Windows
- デフォルト = OFF。値: ON または OFF
- **DB2NTNOCACHE** レジストリー変数は、DB2 データベース・システムがデータベース・ファイルを NOCACHE オプションを指定してオープンするかどうかを指定します。 **DB2NTNOCACHE**=ON の場合は、ファイル・システムのキャッシュは除去されます。 **DB2NTNOCACHE**=OFF の場合、オペレーティング・システムは DB2 ファイルをキャッシュに入れます。 これは、長いフィールドまたは LOB を含んでいるファイルを除くすべてのデータに適用されます。システム・キャッシュを除去すると、より多くのメモリーがデータベースに利用できるようになるため、バッファー・プールやソート・ヒープの量を増やすことができます。

  Windows では、デフォルトの動作として、ファイルをオープンするときにキャッシュに入れられます。ファイル内の 1 GB ごとに 1 MB がシス

テム・プールから予約されます。このレジストリー変数を使用して、キャッシュに関する (文書化されていない) 192 MB 制限をオーバーライドします。キャッシュの限界に達すると、リソース不足を示すエラーが表示されます。

注: DB2NTNOCACHE は、バージョン 8.2 より推奨されなくなり、今後のリリースでは除去される予定です。表スペース・コンテナーで同じ成果を得るには、CREATE TABLESPACE および ALTER TABLESPACE SQL ステートメントを使用します。

DB2NTPRICLASS

- オペレーティング・システム: Windows
- デフォルト = NULL。値: R、H、(他の任意の値)
- DB2 インスタンスの優先度クラスを設定します (プログラム DB2SYSCS.EXE)。次の 3 つの優先度クラスがあります。
  - NORMAL_PRIORITY_CLASS (デフォルトの優先度クラス)
  - REALTIME_PRIORITY_CLASS (「R」を使って設定)
  - HIGH_PRIORITY_CLASS (「H」を使って設定)

  この変数を、個々のスレッド優先順位 (**DB2PRIORITIES** を使って設定) と一緒に使って、システム中の他のスレッドに関連した DB2 スレッドの絶対優先順位を決定します。

  注: DB2NTPRICLASS は推奨されていないため、サービスの推奨時にのみ使用してください。エージェントの優先順位およびプリフェッチの優先順位を調整するには、DB2 サービス・クラスを使用します。この変数を使用する際には、注意する必要があります。誤用すると、システム・パフォーマンス全体に悪い影響を及ぼす可能性があります。

  詳細は、Win32 資料の SetPriorityClass() API を参照してください。

DB2NTWORKSET

- オペレーティング・システム: Windows
- デフォルト = 1,1
- DB2 データベース・マネージャーに利用できる最小および最大の実効ページ・セットを変更するために使用されます。デフォルトでは、Windows でページングが行われていない場合は、プロセスの実効ページ・セットは必要なだけ大きくすることができます。ただし、ページングが発生しているときは、プロセスが持つことができる最大の実効ページ・セットは約 1 MB です。 **DB2NTWORKSET** を使えば、このデフォルトの動作をオーバーライドできます。

  DB2NTWORKSET の指定は、**DB2NTWORKSET**=min, max の構文を使用します。ここで、min と max はメガバイト単位で表されます。

DB2_OVERRIDE_BPF

- オペレーティング・システム: すべて
- デフォルト= 設定しない。値: 正数のページ数、または <entry>[;<entry>...] (<entry>=< バッファー・プール ID>,<ページ数>)

- この変数は、データベースの活動化、ロールフォワード・リカバリー、またはクラッシュ・リカバリーの際に作成されるバッファー・プールのサイズをページ数で指定します。これが役立つのは、メモリー制約のためにデータベースの活動化、ロールフォワード・リカバリー、またはクラッシュ・リカバリーの途中で障害が発生する場合です。メモリーの制約は、実メモリーの不足により生じることも稀にありますが、たいていはデータベース・マネージャーがラージ・バッファー・プールを割り当てようとしたことが原因で、誤った構成のバッファー・プールがあったために起こります。例えば、データベース・マネージャーによって最小のバッファー・プール (16 ページ) も起動しない場合は、この環境変数を使用してさらに小さいページ数を指定してみてください。この変数に指定された値は、現行バッファー・プール・サイズをオーバーライドします。

  バッファー・プールのすべてまたはサブセットのサイズを一時的に変更して始動できるように、以下も使用できます。 <entry>[<entry>...]
  (<entry>=<バッファー・プール ID>,<ページの数>)

DB2_PINNED_BP

- オペレーティング・システム: AIX、HP-UX、Linux
- デフォルト = NO。値: YES または NO
- この変数は、一部のオペレーティング・システムで、データベースに関連したデータベース・グローバル・メモリー (バッファー・プールを含む) をメイン・メモリーに指定するために使用されます。データベース・グローバル・メモリーをシステム・メイン・メモリーに保持することにより、データベース・パフォーマンスがより一貫性のあるものになります。

  たとえば、バッファー・プールがシステム・メイン・メモリーからスワップアウトされる場合、データベース・パフォーマンスは低下します。バッファー・プールをシステム・メモリーに保持することによってディスク入出力が減ると、データベース・パフォーマンスは改善されます。別のアプリケーションがより多くのメイン・メモリーを要求した場合、システム・メイン・メモリー所要量に応じて、データベース・グローバル・メモリーをメイン・メモリーからスワップアウトできます。

  Linux の場合は、このレジストリー変数の変更に加えて、ライブラリー *libcap.so.1* も必要になります。

  64 ビット DB2 for AIX の場合、この変数を YES に設定すると、データベース共用メモリーのセルフチューニング (*database_memory* 構成パラメーターを AUTOMATIC に設定することによって活動化される) を使用可能にできなくなります。

  64 ビット環境での HP-UX では、このレジストリー環境を変更する他に、 DB2 インスタンス・グループに MLOCK 特権を与えなければなりません。これを行うには、ルート・アクセス権限を持つユーザーが以下の処置を実行します。

  1. DB2 インスタンス・グループを `/etc/privgroup` ファイルに追加します。たとえば、DB2 インスタンス・グループが `db2iadm1` グループに属している場合、次の行を `/etc/privgroup` ファイルに追加します。

```
db2iadm1 MLOCK
```

2. 次のコマンドを発行します。

```
setprivgrp -f /etc/privgroup
```

DB2PRIORITIES

- オペレーティング・システム: すべて
- 値の設定はプラットフォームにより異なります
- DB2 プロセスとスレッドの優先順位を制御します。

注: DB2PRIORITIES は推奨されていないため、サービスの推奨時にのみ使用してください。エージェントの優先順位およびプリフェッチの優先順位を調整するには、DB2 サービス・クラスを使用します。

DB2_RESOURCE_POLICY

- オペレーティング・システム: AIX 5 以上、すべての Linux、ただし zSeries (32 ビット) を除く、 Windows Server 2003 またはそれ以上
- デフォルト = 設定しない。値: 構成ファイルへの有効なパス
- リソース・ポリシーを定義し、DB2 データベースが使用するオペレーティング・システム・リソースを制限できます。または、特定のオペレーティング・システム・リソースを特定の DB2 データベースに割り当てるための規則を含んでいます。例えば、AIX、Linux、または Windows オペレーティング・システムの場合、このレジストリー変数を使用して DB2 データベース・システムが使用する処理プログラムのセットを制限できます。リソース制御の範囲は、オペレーティング・システムによって異なります。

  AIX NUMA および Linux NUMA を使用できるマシンでは、ポリシーを定義して、DB2 データベース・システムが使用するリソース・セットを指定できます。リソース・セット・バインディングを使用すると、各 DB2 プロセスが個別に特定のリソース・セットにバインドされます。パフォーマンス調整の状況によっては、この方法が役立つ場合があります。

  このレジストリー変数を設定して、DB2 プロセスをオペレーティング・システム・リソースにバインドするときに使用するポリシーが定義された構成ファイルへのパスを指定できます。リソース・ポリシーを使用することにより、 DB2 データベース・システムを制限するオペレーティング・システム・リソースのセットを指定できます。各 DB2 プロセスはこのセットの単一リソースにバインドされます。リソース割り当ては循環ラウンドロビン方式で行われます。

  構成ファイルの例:

  例 1: すべての DB2 プロセスを CPU 1 または 3 のいずれかにバインドする。

```
<RESOURCE_POLICY>
 <GLOBAL_RESOURCE_POLICY>
  <METHOD>CPU</METHOD>
  <RESOURCE_BINDING>
   <RESOURCE>1</RESOURCE>
  </RESOURCE_BINDING>
  <RESOURCE_BINDING>
   <RESOURCE>3</RESOURCE>
  </RESOURCE_BINDING>
 </GLOBAL_RESOURCE_POLICY>
```

```
 </RESOURCE_POLICY>
```

例 2: (AIX のみ) DB2 プロセスを次のリソース・セットのいずれかにバインドする。
sys/node.03.00000、sys/node.03.00001、sys/node.03.00002、
sys/node.03.00003

```
 <RESOURCE_POLICY>
  <GLOBAL_RESOURCE_POLICY>
   <METHOD>RSET</METHOD>
   <RESOURCE_BINDING>
    <RESOURCE>sys/node.05.00000</RESOURCE>
   </RESOURCE_BINDING>
   <RESOURCE_BINDING>
    <RESOURCE>sys/node.05.00001</RESOURCE>
   </RESOURCE_BINDING>
   <RESOURCE_BINDING>
    <RESOURCE>sys/node.05.00002</RESOURCE>
   </RESOURCE_BINDING>
   <RESOURCE_BINDING>
    <RESOURCE>sys/node.05.00003</RESOURCE>
   </RESOURCE_BINDING>
  </GLOBAL_RESOURCE_POLICY>
 </RESOURCE_POLICY>
```

注意: RSET メソッドを使用するには CAP_NUMA_ATTACH 機能が必要です。
(AIX のみ)

例 3: (Linux のみ) SAMPLE データベースに関連付けられているバッファー・プール ID 2 および 3 からのすべてのメモリーを NUMA ノード 3 にバインドする。
また、NUMA ノード 3 へのバインディングに全データベース・メモリーの 80 パーセントを使い、20 パーセントは非バッファー・プール固有メモリーとしてすべてのノードの間でストライピングされるものとして残します。

```
 <RESOURCE_POLICY>
  <DATABASE_RESOURCE_POLICY>
   <DBNAME>sample</DBNAME>
   <METHOD>NODEMASK</METHOD>
   <RESOURCE_BINDING>
    <RESOURCE>3</RESOURCE>
    <DBMEM_PERCENTAGE>80</DBMEM_PERCENTAGE>
    <BUFFERPOOL_BINDING>
     <BUFFERPOOL_ID>2</BUFFERPOOL_ID>
     <BUFFERPOOL_ID>3</BUFFERPOOL_ID>
    </BUFFERPOOL_BINDING>
   </RESOURCE_BINDING>
  </DATABASE_RESOURCE_POLICY>
 </RESOURCE_POLICY>
```

注: RSET メソッドを使用するには CAP_NUMA_ATTACH 機能が必要であるため、このメソッドは Linux ではサポートされていません。

DB2_RESOURCE_POLICY レジストリー変数で指定された構成ファイルは、SCHEDULING_POLICY 要素を受け入れます。 プラットフォームによっては、SCHEDULING_POLICY 要素を使用して、次のものを選択できます。

– DB2 サーバーが使用するオペレーティング・システムのスケジューリング・ポリシー

  DB2 on AIX および DB2 on Windows では、**DB2NTPRICLASS** レジストリー変数を使用して、オペレーティング・システムのスケジューリング・ポリシーを設定できます。

– 個々の DB2 サーバー・エージェントが使用するオペレーティング・システム優先順位

別の方法として、レジストリー変数の **DB2PRIORITIES** と **DB2NTPRICLASS** を使用して、オペレーティング・システムのスケジューリング・ポリシーの制御と DB2 エージェントの優先順位の設定を行うこともできます。 なお、リソース・ポリシー構成ファイル内の SCHEDULING_POLICY 要素の仕様上、スケジューリング・ポリシーとそれに関連したエージェントの優先順位の両方を 1 つの場所で指定する必要があります。

例 1: AIX SCHED_FIFO2 スケジューリング・ポリシーの選択と db2 ログ書き込み処理および db2 ログ読み取り処理の優先順位の指定

```
<RESOURCE_POLICY>
 <SCHEDULING_POLICY>
  <POLICY_TYPE>SCHED_FIFO2</POLICY_TYPE>
  <PRIORITY_VALUE>60</PRIORITY_VALUE>

  <EDU_PRIORITY>
   <EDU_NAME>db2loggr</EDU_NAME>
   <PRIORITY_VALUE>56</PRIORITY_VALUE>
  </EDU_PRIORITY>

  <EDU_PRIORITY>
   <EDU_NAME>db2loggw</EDU_NAME>
   <PRIORITY_VALUE>56</PRIORITY_VALUE>
  </EDU_PRIORITY>
 </SCHEDULING_POLICY>
</RESOURCE_POLICY>
```

例 2: DB2NTPRICLASS=H の置き換え (Windows)

```
<RESOURCE_POLICY>
 <SCHEDULING_POLICY>
  <POLICY_TYPE>HIGH_PRIORITY_CLASS</POLICY_TYPE>
 </SCHEDULING_POLICY>
</RESOURCE_POLICY>
```

DB2_SET_MAX_CONTAINER_SIZE

- オペレーティング・システム: すべて
- デフォルト = 設定しない。値: -1、65 536 バイトより大きい任意の正整数
- このレジストリー変数を使うと、AutoResize フィーチャーが有効になった自動ストレージ表スペース用の個々のコンテナーのサイズを制限することができます。

  注: DB2_SET_MAX_CONTAINER_SIZE は、バイト、キロバイト、またはメガバイトで指定できますが、db2set はその値をバイトで表示します。
- 値を -1 に設定すると、コンテナーのサイズに対する制限はなくなります。

DB2_SKIPDELETED

- オペレーティング・システム: すべて
- デフォルト = OFF。値: ON または OFF
- この変数が使用可能になっていると、カーソル固定分離レベルまたは読み取り固定分離レベルのいずれかを使用するステートメントは、索引アクセス中に削除されたキー、および表アクセス中に削除された行を無条件でス

キップすることができます。 **DB2_EVALUNCOMMITTED** を使用可能にすると、削除された行は自動的にスキップされますが、非コミットの実際には削除されていないタイプ 2 索引のキーは、**DB2_SKIPDELETED** も使用可能にしていなければスキップされません。

このレジストリー変数は DB2 カタログ表上のカーソルの動作には影響を与えません。

このレジストリー変数は、db2start コマンドによって活動化されます。

DB2_SKIPINSERTED

- オペレーティング・システム: すべて
- デフォルト = OFF。値: ON または OFF
- **DB2_SKIPINSERTED** レジストリー変数が使用可能になっていると、カーソル固定分離レベルまたは読み取り固定分離レベルのいずれかを使用するステートメントは、非コミットの挿入行を、それらが挿入されなかったかのようにスキップすることができます。このレジストリー変数は DB2 カタログ表上のカーソルの動作には影響を与えません。このレジストリー変数は、データベースの始動時に活動化されますが、非コミットの挿入行のスキップは、ステートメントのコンパイル時またはバインド時に決定されます。

  注: 挿入行のスキップの動作には、ロールアウト・クリーンアップが保留になっている表との互換性がありません。その結果、スキャナーが RID に対するロックを待機しても、その RID はロールアウト済みブロックの一部であることを発見するだけです。

DB2_SMS_TRUNC_TMPTABLE_THRESH

- オペレーティング・システム: すべて
- デフォルト = 0。値: -1、0 から n。(n = SMS 表スペース・コンテナー内の一時表ごとに保持されるエクステントの数)
- この変数は、一時表を表すファイルを SMS 表スペースに保持する際の、最小ファイル・サイズしきい値を指定します。

  デフォルトで、この変数は 0 に設定されています。これは、特殊しきい値処理が実行されないことを示します。 その代わり、一時表が不要になると、そのファイルは切り捨てられて 0 エクステントになります。

  この変数の値が 0 より大きい場合は、大きいほうのファイルが保持されます。これにより、一時表が使用されるごとのファイルのドロップおよび再作成に関係するシステム・オーバーヘッドが若干減少します。

  この変数が -1 に設定されている場合は、ファイルが切り捨てられることはなく、ファイルはシステム・リソースによる制限を除いて無制限に増大します。

DB2_SORT_AFTER_TQ

- オペレーティング・システム: すべて
- デフォルト = NO。値: YES または NO

- 受信終了時にデータをソートすることが必要で、受信ノード数が送信ノード数と等しい場合、パーティション・データベース環境内の指示された表キューをオプティマイザーが処理する方法を指定します。

  DB2_SORT_AFTER_TQ=NO の場合、オプティマイザーは送信終了時には行のソートを、受信終了時には行のマージを行う傾向があります。

  DB2_SORT_AFTER_TQ=YES の場合、オプティマイザーはソートをしないで行を送信し、すべての行を受信した後の受信終了時にもマージを行わない傾向があります。

DB2_SELUDI_COMM_BUFFER

- オペレーティング・システム: すべて
- デフォルト = OFF。値: ON または OFF
- この変数は、UPDATE、INSERT、または DELETE (UDI) 照会から、カーソルを SELECT でブロックする処理中に使用されます。使用可能になっていると、このレジストリー変数は照会の結果が一時表に保管されないようにします。その代わりに、UDI 照会からカーソルを SELECT でブロックするための OPEN 処理中に、DB2 データベース・システムは照会の結果全体を通信バッファー・メモリー領域に直接バッファーしようとします。

  注: 通信バッファー・スペースが照会の結果全体を保持できるほど十分には大きくない場合、SQLCODE -906 エラーが発行され、トランザクションはロールバックされます。ローカル・アプリケーションおよびリモート・アプリケーションそれぞれの通信バッファー・メモリー領域のサイズの調整については、 *aslheapsz* および *rqrioblk* データベース・マネージャー構成パラメーターを参照してください。

  パーティション・データベース環境の場合や、パーティション内並列処理が使用可能である場合には、このレジストリー変数はサポートされません。

DB2_TRUSTED_BINDIN

- オペレーティング・システム: すべて
- デフォルト = OFF。値 : OFF、ON、CHECK
- **DB2_TRUSTED_BINDIN** が使用可能になっていると、組み込み unfenced ストアード・プロシージャー内にホスト変数を含む照会ステートメントの実行速度が増します。

  この変数が使用可能になっていると、組み込み unfenced ストアード・プロシージャー内に含まれる SQL および XQuery ステートメントのバインド中に外部 SQLDA フォーマットから内部 DB2 フォーマットへの変換は行われません。これにより、組み込み SQL および XQuery ステートメントの処理速度が増します。

  この変数が使用可能になっている場合、以下のデータ・タイプは組み込み unfenced ストアード・プロシージャーでサポートされません。

  – SQL_TYP_DATE

- SQL_TYP_TIME
- SQL_TYP_STAMP
- SQL_TYP_CGSTR
- SQL_TYP_BLOB
- SQL_TYP_CLOB
- SQL_TYP_DBCLOB
- SQL_TYP_CSTR
- SQL_TYP_LSTR
- SQL_TYP_BLOB_LOCATOR
- SQL_TYP_CLOB_LOCATOR
- SQL_TYP_DCLOB_LOCATOR
- SQL_TYP_BLOB_FILE
- SQL_TYP_CLOB_FILE
- SQL_TYP_DCLOB_FILE
- SQL_TYP_BLOB_FILE_OBSOLETE
- SQL_TYP_CLOB_FILE_OBSOLETE
- SQL_TYP_DCLOB_FILE_OBSOLETE

これらのデータ・タイプが見つかると、SQLCODE -804、SQLSTATE 07002 エラーが戻されます。

注: 入力ホスト変数のデータ・タイプと長さは、対応するエレメントの内部データ・タイプと長さとに正確に一致していなければなりません。ホスト変数の場合、この要件は常に満たされます。しかし、パラメーター・マーカーの場合、データ・タイプが一致しているか確認する必要があります。データ・タイプと長さがすべての入力ホスト変数と一致しているかどうかの確認に CHECK オプションを使用できますが、しかしこのオプションは大抵パフォーマンスを低下させます。

注: DB2_TRUSTED_BINDIN は、推奨されておらず、今後のリリースでは除去される予定です。

DB2_USE_ALTERNATE_PAGE_CLEANING

- オペレーティング・システム: すべて
- デフォルト = 設定しない。値: ON または OFF
- この変数は、DB2 データベースがページ・クリーニング・アルゴリズムの代替方式、またはデフォルトのページ・クリーニング方式を使用するかどうかを指定します。この変数が ON に設定されると、DB2 システムは、変更されたページをディスクに書き込み、LSN_GAP を保持し、積極的に対象を検索します。これを行うことにより、ページ・クリーナーは使用可能なディスク I/O 帯域幅をより効果的に使用できます。この変数が ON に設定されると、*chngpgs_thresh* データベース構成パラメーターはページ・クリーナー・アクティビティーを制御しないので、関係がなくなります。

その他の変数

DB2ADMINSERVER

- オペレーティング・システム: Windows および UNIX
- デフォルト = NULL
- DB2 Administration Server を指定します。

DB2CLIINIPATH

- オペレーティング・システム: すべて
- デフォルト = NULL
- DB2 CLI/ODBC 構成ファイル (db2cli.ini) のデフォルト・パスをオーバーライドし、クライアントの異なる位置を指定するために使用されます。ここで指定される値は、クライアント・システム上の有効なパスでなければなりません。

DB2_COMMIT_ON_EXIT

- オペレーティング・システム: UNIX
- デフォルト = OFF。値: OFF/NO/0 または ON/YES/1
- UNIX プラットフォームでは、バージョン 8 より前の DB2 が、残った未完了トランザクションをアプリケーションの正常終了時には自動コミットしていました。バージョン 8 でこの動作が変更され、未完了トランザクションは終了時に自動ロールバックされるようになりました。このレジストリー変数を使用することにより、以前の動作に依存する組み込み SQL アプリケーションを使用しているユーザーは、バージョン 9 でもそのアプリケーションを継続して使用できます。このレジストリー変数は、JDBC、CLI、および ODBC アプリケーションには影響を与えません。

  このレジストリー変数は推奨されていません。出口点コミットの動作は将来のリリースでサポートされなくなりますので注意してください。バージョン 9 より前に開発されたアプリケーションをこの機能に依存したまま使用し続けるか検討し、必要であればアプリケーションに適した明示的 COMMIT または ROLLBACK ステートメントを追加してください。このレジストリー変数をオンにする場合は、明示的には COMMIT を実行できない新規アプリケーションを出口の前にインプリメントしないように注意してください。

  通常は、このレジストリー変数はデフォルト設定のままにしておきます。

DB2_CREATE_DB_ON_PATH

- オペレーティング・システム: Windows
- デフォルト = NULL。値: YES または NO
- データベース・パスとして (ドライブと同様に) パスを使用するサポートを有効にするには、このレジストリー変数を YES に設定します。**DB2_CREATE_DB_ON_PATH** の設定は、データベースの作成時、データベース・マネージャー構成パラメーター *dftdbpath* の設定時、およびデータベースのリストア時にチェックされます。完全修飾データベース・パスは、最大で 215 文字の長さにすることができます。

DB2_CREATE_DB_ON_PATH が設定されておらず (または NO に設定されており)、データベースの作成またはリストア時にデータベース・パスのパスを指定すると、エラー SQL1052N が戻されます。

DB2_CREATE_DB_ON_PATH が設定されておらず (または NO に設定されており)、*dftdbpath* データベース・マネージャー構成パラメーターを更新すると、エラー SQL5136N が戻されます。

注意:
新規データベースの作成にパス・サポートを使用した場合、バージョン 9.1 より前に、db2DbDirGetNextEntry() API (またはその旧バージョン) を使用して作成されたアプリケーションは、正しく機能しない可能性があります。さまざまなシナリオおよび適切なアクションの詳細については、http://www.ibm.com/software/data/db2/udb/support/ を参照してください。

DB2DEFPREP

- オペレーティング・システム: すべて
- デフォルト = NO。値: ALL、YES、または NO
- DEFERRED_PREPARE プリコンパイル・オプションが利用可能になる前にプリコンパイルされたアプリケーションのために、このオプションの実行時の動作をシミュレートします。例えば、DB2 V2.1.1 またはそれ以前のアプリケーションを DB2 V2.1.2 以降の環境で実行するときは、**DB2DEFPREP** を使用して、望ましい「据え置き準備」動作を指示することができます。

注: DB2DEFPREP は推奨されておらず、今後のリリースでは除去される予定です。この変数は、ユーザーが、DEFERRED_PREPARE プリコンパイル・オプションを使用できない古いバージョンの DB2 を使用している場合にのみ、必要になります。

DB2_DISABLE_FLUSH_LOG

- オペレーティング・システム: すべて
- デフォルト = OFF。値: ON または OFF
- オンライン・バックアップの完了時に、アクティブ・ログ・ファイルのクローズを無効にするかどうかを指定します。

  オンライン・バックアップの完了時に、最後のアクティブ・ログ・ファイルは切り捨てられ、クローズされ、アーカイブできるようになります。これにより、オンライン・バックアップに、リカバリーに使用できるアーカイブ・ログの完全セットが揃うことになります。ログ・シーケンス番号 (LSN) のスペース部分の浪費が心配な場合は、最後のアクティブ・ログ・ファイルのクローズを無効にすることができます。アクティブ・ログ・ファイルが切り捨てられるたびに、LSN は切り捨てられたスペースに見合う量だけ増えます。かなりの数のオンライン・バックアップを毎日実行する場合は、最後のアクティブ・ログ・ファイルのクローズを無効にすることができます。

  オンライン・バックアップの完了後間もなく、ログ・フル・メッセージを受け取ることが分かった場合は、最後のアクティブ・ログ・ファイルのク

ローズを無効にすることができます。ログ・ファイルが切り捨てられる場合、予約済みのアクティブ・ログ・スペースは、切り捨てられたログのサイズに見合う量だけ増えます。アクティブ・ログ・スペースは、切り捨てられたログ・ファイルが再利用されると解放されます。再利用は、ログ・ファイルが非アクティブになった後、間もなく実行されます。これら 2 つのイベント間の短い間隔内に、ログ・フル・メッセージを受け取る場合があります。

ログを含むどのバックアップにおいても、バックアップにログを含めるためにアクティブ・ログ・ファイルは切り捨ててクローズする必要があるので、このレジストリー変数は無視されます。

DB2CONNECT_DISCONNECT_ON_INTERRUPT

- オペレーティング・システム: すべて
- デフォルト = NO。値: YES/TRUE/1 または NO/FALSE/0
- YES (TRUE または 1) に設定すると、この変数は、中断が発生したときにバージョン 8 (以上) の DB2 Universal Database z/OS サーバーへの接続を即時に切断することを指定します。この変数は以下の構成で使用できます。
  - DB2 クライアントをバージョン 8 (以上) の DB2 UDB z/OS サーバーと共に実行している。この場合、クライアント上の **DB2CONNECT_DISCONNECT_ON_INTERRUPT** を YES に設定します。
  - バージョン 8 (以上) の DB2 UDB z/OS サーバーに通ずる DB2 Connect ゲートウェイを経由して、DB2 クライアントを実行している。この場合、ゲートウェイ上の **DB2CONNECT_DISCONNECT_ON_INTERRUPT** を YES に設定します。

DB2_DISPATCHER_PEEKTIMEOUT

- オペレーティング・システム: すべて
- デフォルト = 1。値: 0 から 32767 秒。0 は即時にタイムアウトになることを示します。
- **DB2_DISPATCHER_PEEKTIMEOUT** により、クライアントをエージェントに引き渡す前に、ディスパッチャーがクライアントの接続要求を待機する時間 (秒単位) を調整できます。ほとんどの場合、このレジストリー変数は調整する必要がありません。このレジストリー変数は、DB2 Connect 接続コンセントレーターが使用可能になっているインスタンスだけに影響を与えます。

  このレジストリー変数および **DB2_SERVER_CONTIMEOUT** レジストリー変数はどちらも、接続時の新規クライアントの取り扱いを構成します。多くの低速のクライアントがインスタンスに接続している場合、ディスパッチャーは各クライアントをタイムアウトするまで最大で 1 秒待機できますが、数多くのクライアントが同時に接続している場合は、結果としてディスパッチャーがボトルネックになります。複数のアクティブ・データベースがあるインスタンスで、非常に低速の接続時には、**DB2_DISPATCHER_PEEKTIMEOUT** を 0 に引き下げることができま

す。 **DB2_DISPATCHER_PEEKTIMEOUT** を引き下げると、ディスパッチャーは既にその場にあるクライアントの接続要求のみを考慮するようになります。つまり、ディスパッチャーは接続要求の着信を待機することはありません。無効値が設定された場合、デフォルト値が使用されます。このレジストリー変数は、動的ではありません。

DB2_DJ_INI

- オペレーティング・システム: すべて
- デフォルト =
  - UNIX: db2_instance_directory/cfg/db2dj.ini
  - Windows: db2_install_directory¥cfg¥db2dj.ini
- フェデレーション構成ファイルの絶対パス名を指定します。たとえば、`db2set DB2_DJ_INI=$HOME/sqllib/cfg/my_db2dj.ini` のようにします。このファイルには、データ・ソース環境変数の設定値が含まれています。これらの環境変数は、Informix® ラッパー、および WebSphere® Federation Server によって提供されるラッパーによって使用されます。

  次に示すのは、フェデレーション構成ファイルの例です。

  ```
  INFORMIXDIR=/informix/client_sdk
  INFORMIXSERVER=inf93
  ORACLE_HOME=/usr/oracle9i
  SYBASE=/sybase/V12
  SYBASE_OCS=OCS-12_5
  ```

  db2dj.ini ファイルには、以下の制約事項が適用されます。

  - *evname=value* の形式で入力しなければなりません。 evname は環境変数の名前で、value はその値です。
  - 環境変数名の最大長は 255 バイトです。
  - 環境変数値の最大長は 765 バイトです。

  この変数は、データベース・マネージャー・パラメーター FEDERATED が YES に設定されていない場合に無視されます。

DB2DMNBCKCTLR

- オペレーティング・システム: Windows
- デフォルト = NULL。値: ? またはドメイン・ネーム
- DB2 サーバーがバックアップ・ドメイン・コントローラーになっている場合、そのドメイン・ネームが分かれば、**DB2DMNBCKCTLR**=DOMAIN_NAME と設定します。 DOMAIN_NAME は大文字でなければなりません。ローカル・マシンがバックアップ・ドメイン・コントローラーになっているドメインを DB2 が判別するには、**DB2DMNBCKCTLR**=? と設定します。 **DB2DMNBCKCTLR** プロファイル変数を設定しなかったりブランクに設定したりすると、 DB2 は 1 次ドメイン・コントローラーで認証を実行します。

  注: デフォルトでは、DB2 は既存のバックアップ・ドメイン・コントローラーを使用しません。バックアップ・ドメイン・コントローラーは 1 次ドメイン・コントローラーと同期しないことがあり、セキュリティーが危険にさらされる可能性があるためです。 1 次ドメイン・コントローラ

ーのセキュリティー・データベースが更新されたが、その変更内容がバックアップ・ドメイン・コントローラーに伝搬していない場合に、同期しなくなることがあります。この事態は、ネットワーク待ち時間が生じた場合やコンピューターのブラウザー・サービスが作動可能でない場合に起こることがあります。

注: DB2DMNBCKCTLR は推奨されておらず、今後のリリースでは除去される予定です。 Active Directory ではバックアップ・ドメイン・コントローラーがなくなったため、この変数は必要なくなりました。

DB2_DOCHOST

- オペレーティング・システム: すべて
- デフォルト= 設定しない (ただし、DB2 は引き続き http://publib.boulder.ibm.com/infocenter/db2luw/v9 にある IBM Web サイトから Information Center へのアクセスを試みます)。値: http://*hostname* (*hostname*= 有効なホスト名または IP アドレス)
- 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ホスト名を指定します。「DB2 セットアップ」ウィザードで自動構成オプションを選択した場合には、 DB2 インフォメーション・センターのインストール中に、この変数は自動的に設定されます。

DB2_DOCPORT

- オペレーティング・システム: すべて
- デフォルト = NULL。値: 任意の有効なポート番号
- DB2 ヘルプ・システムが DB2 資料を表示するときに使用するポート番号を指定します。「DB2 セットアップ」ウィザードで自動構成オプションを選択した場合には、 DB2 インフォメーション・センターのインストール中に、この変数は自動的に設定されます。

DB2_ENABLE_AUTOCONFIG_DEFAULT

- オペレーティング・システム: すべて
- デフォルト = NULL。値: YES または NO
- この変数は、データベースの作成時に構成アドバイザーが自動的に実行されるかどうかを制御します。 **DB2_ENABLE_AUTOCONFIG_DEFAULT** が設定されない場合 (NULL の場合)、変数が YES に設定される場合と同じ効果があり、データベースの作成時に構成アドバイザーが実行されます。この変数の設定後にインスタンスを再始動する必要はありません。AUTOCONFIGURE コマンドまたは CREATE DB AUTOCONFIGURE を実行する場合、これらのコマンドは **DB2_ENABLE_AUTOCONFIG_DEFAULT** の設定をオーバーライドします。

DB2_ENABLE_LDAP

- オペレーティング・システム: すべて
- デフォルト = NO。値: YES または NO
- Lightweight Directory Access Protocol (LDAP) を使用するかどうかを指定します。 LDAP は、ディレクトリー・サービスへのアクセス方式の 1 つです。

DB2_EVMON_EVENT_LIST_SIZE

- オペレーティング・システム: すべて
- デフォルト=0 (制限なし)。値: KB/Kb/kb、MB/Mb/mb、または GB/Gb/gb で指定した値; この変数には固定された上限はありませんが、モニター・ヒープから使用可能なメモリーの量による限界があります。
- このレジストリー変数は、特定のイベント・モニターへの書き込みを待機させるためにキューに入れることができる最大バイト数を指定します。キューのサイズがこの限度に達すると、イベント・モニター・レコードを送信しようとしているエージェントは、キューのサイズがこのしきい値よりも小さくなるまで待機します。

注: モニター・ヒープから割り振りを得られないアクティビティー・レコードはドロップされます。アクティビティー・レコードのドロップが起こらないようにするには、*mon_heap_sz* 構成パラメーターを AUTOMATIC に設定します。*mon_heap_sz* を特定の値に設定している場合は、**DB2_EVMON_EVENT_LIST_SIZE** の値がこの値よりも小さくなるようにしてください。ただし、モニター・ヒープは他のモニター・エレメントのトラッキングにも使用されるため、この処置を行ったからといってアクティビティー・レコードのドロップがまったく起こらないとは限りません。

DB2_EVMON_STMT_FILTER

- オペレーティング・システム: すべて
- デフォルト = 設定しない。値:
  - すべて: すべてのステートメント・イベント・モニターの出力をフィルターに掛けます。このオプションは、排他的です。
  - '*nameA nameB nameC*': ストリングのそれぞれの名前は、フィルターに掛けるレコードのイベント・モニターの名前を表します。複数の名前を指定する場合は、それぞれの名前をブランク 1 つで区切ってください。 DB2 は、入力された名前をすべて大文字にします。それぞれ最大 18 バイトの名前を最大 32 個まで指定できます。
  - '*nameA:op1,op2 nameB:op1,op2 nameC:op1*': ストリングのそれぞれの名前は、フィルターに掛けるレコードのイベント・モニターの名前を表します。各オプション (op1、op2、など) は、特定の SQL 操作にマップする整数値を表します。整数値を指定することにより、どの規則をどのイベント・モニターに適用するかを決定できます。
- **DB2_EVMON_STMT_FILTER** を使用すると、ステートメント・イベント・モニターによって書き込まれるレコードの数を削減できます。このレジストリー変数を設定すると、以下の SQL 操作のレコードだけが、指定されたイベント・モニターに書き込まれるようになります。

表 64.

SQL 操作	整数値マッピング
SELECT	15
EXECUTE	2
EXECUTE_IMMEDIATE	3
CLOSE	6

表 64. (続き)

SQL 操作	整数値マッピング
STATIC COMMIT	8
STATIC ROLLBACK	9
CALL	12
PRE_EXEC	17

その他のすべての操作は、ステートメント・イベント・モニターの出力には表示されなくなります。イベント・モニターへレコードを書き込む操作のセットをカスタマイズするには、整数値を使用します。

例 1:

```
db2set DB2_EVMON_STMT_FILTER= 'mon1 monitor3'
```

この例では、mon1 および monitor3 イベント・モニターが、限定されたアプリケーション要求のリストに関してレコードを受け取ります。たとえば、mon1 ステートメント・イベント・モニターによってモニターされているアプリケーションが動的 SQL ステートメントを作成し、そのステートメントに基づいてカーソルを開き、そのカーソルから 10,000 行をフェッチした後、カーソルのクローズ要求を発行する場合、mon1 イベント・モニターの出力にはクローズ要求のレコードのみが表示されます。

例 2:

```
db2set DB2_EVMON_STMT_FILTER='evmon1:3,8 evmon2:9,15
```

この例では、evmon1 および evmon2 が、限定されたアプリケーション要求のリストに関してレコードを受け取ります。例えば、evmon1 ステートメント・イベント・モニターによってモニターされているアプリケーションが CREATE ステートメントを発行する場合、即時実行操作および静的コミット操作のみが evmon1 イベント・モニター出力に表示されます。evmon2 ステートメント・イベント・モニターによってモニターされているアプリケーションが、選択ロールバックと静的ロールバックの両方を含む SQL を実行する場合、これらの 2 つの操作のみが evmon2 イベント・モニター出力に表示されます。

注: データベース・システム・モニター定数の定義については、sqlmon.h ヘッダー・ファイルを参照してください。

DB2_EXTSECURITY

- オペレーティング・システム: Windows
- デフォルト = ON。値: ON または OFF
- DB2 システム・ファイルをロックすることにより、 DB2 への無許可アクセスを防ぎます。問題を未然に防ぐため、このレジストリー変数はオフにしないでください。

DB2_FALLBACK

- オペレーティング・システム: Windows
- デフォルト = OFF。値: ON または OFF

- この変数を使用することによって、フォールバック処理中に強制的にすべてのデータベース接続を切断できます。これは、Microsoft Cluster Server (MSCS) がある Windows 環境で、フェイルオーバー・サポートと一緒に使用されます。 **DB2_FALLBACK** が未設定、または OFF に設定されていて、フォールバックの間データベースが接続されている場合、DB2 リソースをオフラインにすることはできません。つまり、フォールバック処理は失敗します。

DB2_FMP_COMM_HEAPSZ

- オペレーティング・システム: Windows、UNIX
- デフォルト = 20 MB または 10 の fenced ルーチンを実行するのに十分なスペース (大きいほう)。 AIX では、デフォルトは 256 MB です。
- この変数は、ストアード・プロシージャーやユーザー定義関数の呼び出しのような、 fenced ルーチンの呼び出しに使用されるプールのサイズを指定します (4 KB ページ単位)。各 fenced ルーチンが使用するスペースは、*aslheapsz* 構成パラメーターの値の 2 倍です。

  システムで多くの fenced ルーチンを実行している場合には、この変数の値を増やす必要があるかもしれません。実行している fenced ルーチンがとても少ない場合には、変数の値を減らすことができます。

  この値を 0 に設定すると設定なしと見なされ、fenced ルーチンを呼び出すことができなくなります。このことは、自動データベース保守機能 (自動バックアップ、統計収集、および REORG) が fenced ルーチン・インフラストラクチャーに依存しているため、ヘルス・モニターと自動データベース保守機能が使用不能になるという意味でもあります。

DB2_GRP_LOOKUP

- オペレーティング・システム: Windows
- デフォルト = NULL。値:
  LOCAL、DOMAIN、TOKEN、TOKENLOCAL、TOKENDOMAIN
- この変数は、ユーザーが所属するグループを列挙するために、どの Windows セキュリティー機構を使用するかを指定します。

DB2_HADR_BUF_SIZE

- オペレーティング・システム: すべて
- デフォルト = 2**logbufsz*
- この変数では、スタンバイ・ログ受信バッファー・サイズをログ・ページの単位で指定します。この変数を設定しないと、DB2 はプライマリー側の *logbufsz* 構成パラメーター値の 2 倍の値をスタンバイ受信バッファー・サイズに使用します。この変数はスタンバイ・インスタンス内で設定します。 1 次データベースは、この変数を無視します。

  HADR 同期モード (*hadr_syncmode* データベース構成パラメーター) が ASYNC に設定されている場合、ピア状態中に、スタンバイ側の処理が遅いと、プライマリー上での送信操作が停止し、それによってプライマリー上のトランザクション処理がブロックされることがあります。デフォルトより大きいログ受信バッファーは、スタンバイ・データベース上に構成することができ、このバッファーにはより多くの未処理ログ・データが入り

ます。これにより、プライマリーで処理するトランザクションが待たされず、スタンバイがログ・データを取り込むよりも、プライマリーがログ・データを生成する方が速い状態が短時間起こりえます。

DB2_HADR_NO_IP_CHECK

- オペレーティング・システム: すべて
- デフォルト: OFF。値: ON |OFF
- HADR 接続で IP 検査を迂回するかどうかを指定します。
- この変数は、主に、ネットワーク・アドレス変換 (NAT) 環境で HADR 接続の IP クロス・チェックをバイパスする場合に使用されます。この変数を指定すると、HADR 構成の正常性検査の効果が弱まるため、他の環境でのこの変数の使用は推奨されていません。デフォルトでは、HADR 接続が確立されるときに、ローカル・ホストとリモート・ホストのパラメーターの構成の整合性が検証されます。クロス・チェックのため、ホスト名は IP アドレスにマップされます。次の 2 つの検査が実行されます。
  - プライマリーの HADR_LOCAL_HOST パラメーター = スタンバイの HADR_REMOTE_HOST パラメーター
  - プライマリーの HADR_REMOTE_HOST パラメーター = スタンバイの HADR_LOCAL_HOST パラメーター

  検査が失敗すると、接続はクローズされます。

  このパラメーターをオンにすると、IP 検査は実行されなくなります。

DB2_HADR_PEER_WAIT_LIMIT

- オペレーティング・システム: すべて
- デフォルト = 0 (制限なしを意味する)。値: 0 以上、最大符号なしの 32 ビット整数以下
- DB2 バージョン 9.5 フィックスパック 1 から、レジストリー変数 *DB2_HADR_PEER_WAIT_LIMIT* が設定されているとき、HADR の 1 次データベースは、スタンバイへのログ・レプリケーションのために 1 次データベースへのロギングが指定の秒数ブロックされた場合にピア状態から解除されるようになりました。この限度に到達すると、1 次データベースはスタンバイ・データベースへの接続を切断します。ピア・ウィンドウが使用できない場合、1 次データベースは切断状態になり、ロギングが再開します。ピア・ウィンドウが使用できる場合、1 次データベースは切断ピア状態になり、その状態では、ロギングはブロックされたままになります。1 次データベースは、再接続またはピア・ウィンドウの終了時に、切断ピア状態のままになります。1 次データベースが切断ピア状態になると、ロギングが再開します。このパラメーターは、スタンバイ・データベースには影響を与えません。ただし、1 次データベースとスタンバイ・データベースとの両方で同じ値を使用することが推奨されています。無効な値 (数値または負の数値ではない) は「0」と解釈され、制限がないことを意味します。このパラメーターは静的です。このパラメーターを有効にするには、データベース・インスタンスを再始動する必要があります。

DB2LDAP_BASEDN

- オペレーティング・システム: すべて
- デフォルト = NULL。値: 任意の有効なベース識別ドメイン名。

- これが設定されると、DB2 の LDAP オブジェクトは、指定されたベース識別名の下、以下の条件で LDAP ディレクトリーに保管されます。

```
CN=System
CN=IBM
CN=DB2
```

  これを Microsoft Active Directory Server で設定する場合は、CN=DB2、CN=IBM、および CN=System がこの識別名の下で定義されていることを確認してください。

DB2LDAPCACHE

- オペレーティング・システム: すべて
- デフォルト = YES。値: YES または NO
- LDAP キャッシュを使用可能にするかどうかを指定します。このキャッシュは、ローカル・マシン上のデータベース、ノード、および DCS ディレクトリーのカタログを作成するのに使用します。

  確実にキャッシュ内の項目を最新のものにするには、以下を行います。

```
REFRESH LDAP IMMEDIATE ALL
```

  このコマンドは、データベース・ディレクトリーとノード・ディレクトリーにおいて、正しくない項目の更新および除去を実行します。

DB2LDAP_CLIENT_PROVIDER

- オペレーティング・システム: Windows
- デフォルト = NULL (使用可能であれば Microsoft が使用されます。そうでなければ IBM が使用されます。) 値: IBM または Microsoft
- Windows 環境で稼働している場合、 DB2 は LDAP ディレクトリーへアクセスするために、 Microsoft LDAP クライアントか IBM LDAP クライアントのいずれかの使用をサポートしています。このレジストリー変数は、 DB2 が使用する LDAP クライアントを明示的に選択するのに使用します。

  注: このレジストリー変数の現行値を表示するには、以下の `db2set` コマンドを使用します。

```
db2set DB2LDAP_CLIENT_PROVIDER
```

DB2LDAPHOST

- オペレーティング・システム: すべて
- デフォルト = NULL。値: 任意の有効なホスト名
- LDAP ディレクトリーの位置のホスト名を指定します。

DB2LDAP_KEEP_CONNECTION

- オペレーティング・システム: すべて
- デフォルト = YES。値: YES または NO
- DB2 がその内部 LDAP 接続ハンドルをキャッシュするかどうかを指定します。この変数を NO に設定すると、DB2 は LDAP 接続ハンドルをディレクトリー・サーバーにキャッシュしません。この場合、パフォーマンスにマイナスの影響を及ぼす可能性がありますが、同時にアクティブな、

ディレクトリー・サーバーへの LDAP クライアント接続の数を最少にすることが必要である場合は、**DB2LDAP_KEEP_CONNECTION** を NO に設定するとよいでしょう。

最高のパフォーマンスを得るために、この変数はデフォルトで YES に設定されています。

DB2LDAP_KEEP_CONNECTION レジストリー変数は、グローバル・レベル・プロファイル・レジストリー変数として LDAP にインプリメントされているだけなので、次のように db2set コマンドに -gl オプションを指定して、この変数を設定しなければなりません。

```
db2set -gl DB2LDAP_KEEP_CONNECTION=NO
```

DB2LDAP_SEARCH_SCOPE

- オペレーティング・システム: すべて
- デフォルト = DOMAIN。値: LOCAL、DOMAIN、または GLOBAL
- Lightweight Directory Access Protocol (LDAP) のデータベース・パーティションまたはドメインで検出された情報の検索範囲を指定します。LOCAL を指定すると、LDAP ディレクトリー内の探索は使用不可になります。 DOMAIN を指定すると、現行ディレクトリー・パーティションの LDAP 内だけを探索します。 GLOBAL を指定すると、オブジェクトが見つかるまで全ディレクトリー・パーティション内の LDAP を探索します。

DB2_LOAD_COPY_NO_OVERRIDE

- オペレーティング・システム: すべて
- デフォルト = NONRECOVERABLE。値: COPY YES または NONRECOVERABLE
- この変数は、任意の LOAD COPY NO を、変数の値に応じて、LOAD COPY YES または NONRECOVERABLE のいずれかに変換します。この変数は HADR 1 次データベースと標準 (非 HADR) データベースに適用可能です。 HADR スタンバイ・データベース上では無視されます。HADR 1 次データベース上では、この変数が設定されていないと、LOAD COPY NO が LOAD NONRECOVERABLE に変換されます。この変数の値は、COPY YES 節と同じ構文を使用して、リカバリー不能ロードまたはコピー宛先のいずれかを指定します。

DB2LOADREC

- オペレーティング・システム: すべて
- デフォルト = NULL
- ロールフォワード時にロード・コピーの位置をオーバーライドするために使用されます。ユーザーがロード・コピーの物理的な位置を変更している場合には、ロールフォワードを出す前に **DB2LOADREC** を設定しておく必要があります。

DB2LOCK_TO_RB

- オペレーティング・システム: すべて
- デフォルト = NULL。値: STATEMENT

- ロック・タイムアウトの場合にトランザクション全体をロールバックするか、または現行のステートメントだけをロールバックするかを指定します。 **DB2LOCK_TO_RB** が STATEMENT に設定されていると、ロック・タイムアウトによってロールバックされるのは、現行のステートメントだけになります。その他の設定では、トランザクション全体がロールバックされます。

DB2_MAP_XML_AS_CLOB_FOR_DLC

- オペレーティング・システム: すべて
- デフォルト = NO。値: YES または NO
- **DB2_MAP_XML_AS_CLOB_FOR_DLC** レジストリー変数は、XML をデータ・タイプとしてサポートしないクライアント (または DRDA Application Requestor) のために、XML 値のデフォルトの DESCRIBE および FETCH 動作をオーバーライドする機能を提供します。デフォルト値は NO で、これはこれらのクライアントに対し、XML 値の DESCRIBE が BLOB(2GB) を戻すこと、および XML 値の FETCH が UTF-8 のエンコード方式を示す XML 宣言を含む BLOB に対する暗黙の XML シリアライゼーションを生じさせることを指定します。

  値が YES の場合、XML 値の DESCRIBE は CLOB(2GB) を戻し、XML 値の FETCH は XML 宣言を含まない CLOB に対する暗黙の XML シリアライゼーションを生じさせます。

注: DB2_MAP_XML_AS_CLOB_FOR_DLC は、推奨されておらず、今後のリリースでは除去される予定です。XML 値にアクセスする既存の DB2 アプリケーションのほとんどは XML 対応クライアントを使用して XML 値にアクセスするため、この変数は必要なくなりました。

DB2_MAX_LOB_BLOCK_SIZE

- オペレーティング・システム: すべて
- デフォルト = 0 (制限なし)。値: 0 から 21487483647
- 1 つのブロックで戻される最大の LOB または XML データの量を設定します。これはハードの最大ではありません。データ取得中にサーバー上でこの最大に到達すると、サーバーはコマンドの応答 (たとえば FETCH など) をクライアントに生成する前に、現在行の書き込みを終了します。

DB2_MEMORY_PROTECT

- オペレーティング・システム: ストレージ・キーをサポートする AIX
- デフォルト = NO。値: NO または YES
- このレジストリー変数は、ストレージ・キーを使用して無効なメモリー・アクセスによるバッファー・プール内のデータの破損を防ぐ、メモリー保護フィーチャーを有効にします。メモリー保護は、DB2 エンジン・スレッドがバッファー・プール・メモリーにいつアクセスしてよく、いつアクセスすべきでないかを識別することによって機能します。
  DB2_MEMORY_PROTECT が YES に設定されている場合、DB2 エンジン・スレッドがバッファー・プール・メモリーに不正にアクセスしようとするときにはいつでも、そのエンジン・スレッドはトラップします。

トラップ回復フィーチャー **DB2_THREAD_SUSPENSION** をフィーチャーさせるためには、**DB2_MEMORY_PROTECT** を有効にする必要があります。

注: **DB2_LGPAGE_BP** が YES に設定されている場合、メモリー保護を使用することはできません。**DB2_MEMORY_PROTECT** が YES に設定されている場合でも、DB2 はバッファー・プール・メモリーの保護に失敗し、この機能を使用不可にします。

DB2NOEXITLIST

- オペレーティング・システム: すべて
- デフォルト = OFF。値: ON または OFF
- この変数は、DB2 に出口リスト・ハンドラーをロードさせず、**DB2_COMMIT_ON_EXIT** レジストリー変数の設定にかかわらずアプリケーション終了時にコミットを実行させないことを示します。

  DB2NOEXITLIST がオフになっていて **DB2_COMMIT_ON_EXIT** がオンになっている場合は、組み込み SQL アプリケーションのすべての未完了トランザクションが自動的にコミットされます。推奨されているのは、アプリケーション出口で明示的に COMMIT や ROLLBACK ステートメントを追加するやり方です。

  アプリケーションが終了する前にアプリケーションに動的に DB2 ライブラリーをロードおよびアンロードさせると、DB2 出口ハンドラーを呼び出す際に破損が生じる可能性があります。この破損は、アプリケーションがメモリー内に存在しない関数の呼び出しを試行するために発生することがあります。このような状態にならないようにするためには、**DB2NOEXITLIST** レジストリー変数を設定してください。

DB2_NUM_CKPW_DAEMONS

- オペレーティング・システム: UNIX
- デフォルト = 3。値: 0 から 100
- **DB2_NUM_CKPW_DAEMONS** レジストリー変数を使用することにより、構成可能な数のチェック・パスワード・デーモンを開始することができます。デーモンは db2start の実行中 (これはルートとして実行される) に作成され、チェック・パスワード要求を処理します。
  DB2_NUM_CKPW_DAEMONS の設定値を大きくすると、データベース接続の確立に必要な時間は減ります。しかしこれは、多数の接続が同時に行われ、認証にコストがかかるようなシナリオでのみ有効です。

  DB2_NUM_CKPW_DAEMONS を 0 に設定するとチェック・パスワード・デーモンは使用不可になります。そして DB2 データベース・マネージャーは個別にプログラムを開始し、必要になったときに毎回パスワードをチェックするようになります (バージョン 7 の動作)。

DB2_OPTSTATS_LOG

- オペレーティング・システム: すべて
- デフォルト = 設定しない (詳細は以下を参照してください)。値 = OFF、ON {NUM | SIZE | NAME | DIR}

- **DB2_OPTSTATS_LOG** は、統計収集関連のアクティビティーをモニターおよび分析するために使用する統計イベント・ロギング・ファイルの属性を指定します。 **DB2_OPTSTATS_LOG** を設定しない場合や、ON に設定した場合は、統計イベントのロギングが使用可能になり、システム・パフォーマンスのモニターや、問題を判別しやすくするための履歴の維持ができるようになります。ログ・レコードは、ファイルがいっぱいになるまで、1 つ目のログ・ファイルに書き込まれます。それ以降のレコードは、その次に使用できるログ・ファイルに書き込まれます。ファイルの数が最大数に達すると、一番古いログ・ファイルに新しいレコードが上書きされます。システム・リソースの消費が大きな懸念となる場合は、このレジストリー変数を OFF に設定して使用不可にしてください。

  統計イベント・ロギングを明示的に使用可能にする場合 (ON に設定する場合) には、ユーザーが変更可能なオプションがいくつもあります。

  - NUM: 回転ログ・ファイルの最大数。デフォルト = 5。値: 1 から 15
  - SIZE: 回転ログ・ファイルの最大サイズ。(各回転ファイルのサイズは SIZE/NUM になります。)デフォルト=100 Mb。値: 1 Mb から 4096 Mb
  - NAME: 回転ログ・ファイルのベース名。デフォルト = db2optstats.<number>.log。例えば、db2optstats.0.log、db2optstats.1.log など。
  - DIR: 回転ログ・ファイルのベース・ディレクトリー。デフォルト = $DIAGPATH/events

  値を指定できるオプションの数に制限はありません。ただし、統計のロギングを使用可能にする場合は最初の値を必ず ON にしてください。例えば、ログ・ファイルの最大数を 6、ファイルの最大サイズを 25 MB、基本ファイル名を mystatslog、ディレクトリーを mystats に指定して統計のロギングを使用可能にする場合は、次のコマンドを発行します。

  ```
  db2set DB2_OPTSTATS_LOG=ON,NUM=6,SIZE=25,NAME=mystatslog,DIR=mystats
  ```

DB2REMOTEPREG

- オペレーティング・システム: Windows
- デフォルト = NULL。値: 任意の有効な Windows マシン名
- DB2 インスタンス・プロファイルおよび DB2 インスタンスの Win32 登録リストが入っているリモート・マシン名を指定します。
  DB2REMOTEPREG の値は、DB2 のインストール後にただ一度だけ設定し、変更すべきではありません。この変数の使用には十分な注意が必要です。

DB2_RESOLVE_CALL_CONFLICT

- オペレーティング・システム: AIX、HP-UX、Solaris、Linux、Windows
- デフォルト = YES。値: YES、NO
- トリガーのコンテキストで SQLCODE -746 エラーが出ないようにします。トリガーで CALL ステートメントを発行すると、実行時に SQLCODE SQL0746 が出されます。 SQL0746 エラーは、トリガーにより呼び出されたプロシージャーが、呼び出すステートメントのコンテキスト内で、以前に変更された表にアクセスしないようにします。この変数を設定すると、DB2 データベース・マネージャーは、CALL ステートメン

トを実行する前に、トリガーについての SQL 標準規則に準拠して表のすべての変更が強制的に完了するようにします。

DB2_RESOLVE_CALL_CONFLICT をリセットする前にインスタンスを停止した後、それを再始動する必要があります。その後、トリガーの呼び出しの原因となるすべてのパッケージを再バインドします。 SQL プロシージャーを再バインドするには、CALL SYSPROC.REBIND_ROUTINE_PACKAGE ('P','*procedureschema.procedurename*','CONSERVATIVE'); を使用します。

DB2_RESOLVE_CALL_CONFLICT には、パフォーマンスに影響を与える可能性があることに留意する必要があります。
DB2_RESOLVE_CALL_CONFLICT を YES に設定すると、DB2 データベース・マネージャーが必要に応じて一時表を注入することによって、読み取りおよび書き込みでの競合の可能性をすべて解決します。これはせいぜい 1 つの一時表が注入されるだけなので、通常影響は小さなものです。トリガー・ステートメントにより 1 行 (または少数の行) しか変更されていないので、これは OLTP 環境には少しの影響しかありません。通常、TEMPORARY 表スペースに SMS (システム管理スペース) を使用するための一般推奨に従う場合、**DB2_RESOLVE_CALL_CONFLICT** を設定することのパフォーマンスへの影響は低いと予想されます。

DB2ROUTINE_DEBUG

- オペレーティング・システム: AIX および Windows
- デフォルト = OFF。値: ON または OFF
- Java ストアード・プロシージャー用のデバッグ機能を使用可能にするかどうかを指定します。 Java ストアード・プロシージャーをデバッグしない場合は、デフォルト OFF を使用します。デバッグを使用可能にすると、パフォーマンス上の影響があります。

注: DB2ROUTINE_DEBUG は推奨されておらず、今後のリリースでは除去される予定です。ストアード・プロシージャー・デバッガーは、Unified Debugger に置き換えられました。

DB2SATELLITEID

- オペレーティング・システム: すべて
- デフォルト = NULL。値: サテライト制御データベースで宣言されている有効なサテライト ID
- サテライトが同期するときに、サテライト制御サーバーに渡されるサテライト ID を指定します。この変数に値が指定されない場合は、ログオン ID がサテライト ID として使用されます。

DB2_SERVER_CONTIMEOUT

- オペレーティング・システム: すべて
- デフォルト = 180。値: 0 から 32767 秒
- このレジストリー変数および **DB2_DISPATCHER_PEEKTIMEOUT** レジストリー変数はどちらも、接続時の新規クライアントの取り扱いを構成します。 **DB2_SERVER_CONTIMEOUT** により、接続を終了する前に、エージェントがクライアントの接続要求を待機する時間 (秒単位) を調整で

きます。ほとんどの場合、このレジストリー変数は調整する必要がありません。しかし、DB2 クライアントが接続時にサーバーによって頻繁にタイムアウトになってしまう場合は、**DB2_SERVER_CONTIMEOUT** にさらに高い値を設定して、タイムアウト期間を延長することができます。無効値が設定された場合、デフォルト値が使用されます。このレジストリー変数は、動的ではありません。

DB2SORT

- オペレーティング・システム: すべて、ただしサーバーのみ
- デフォルト = NULL
- この変数は、ロード・ユーティリティーが実行時にロードするライブラリーの位置を指定します。このライブラリーには、索引付きデータのソートに使用される関数の入り口点が入っています。表索引の生成時にロード・ユーティリティーと共にベンダー提供のソート用製品を利用するときは、**DB2SORT** を使用します。提供されるパスは、データベース・サーバーとの関係で表される必要があります。

DB2_THREAD_SUSPENSION

- オペレーティング・システム: ストレージ・キーをサポートする AIX
- デフォルト = OFF。値: ON または OFF
- このレジストリー変数は、DB2 スレッド中断フィーチャーを使用可能にするかどうかを指定します。これにより、障害のあるエンジン・スレッド (ストレージ・キーで保護されたメモリーに不正にアクセスしようとしたスレッド) を中断することにより、 DB2 インスタンスがトラップを維持するかどうかを制御できます。

  注: DB2_THREAD_SUSPENSION は、レジストリー変数 **DB2_MEMORY_PROTECT** が YES に設定されている場合にのみ使用可能にできます。

DB2_TRUNCATE_REUSESTORAGE

- オペレーティング・システム: すべて
- デフォルト = NULL (設定しない)。値: IMPORT、import
- この変数を使用して、IMPORT with REPLACE コマンドと BACKUP ... ONLINE コマンドの間のロック競合を解決することができます。状況によっては、オンライン・バックアップ操作と切り捨て操作を同時に実行できない場合があります。その場合には **DB2_TRUNCATE_REUSESTORAGE** を IMPORT または import に設定することができます。そうすると、データ、索引、長いフィールド、ラージ・オブジェクト、およびブロック・マップ (多次元クラスタリング表の場合) を含むオブジェクトの物理的な切り捨てはスキップされ、論理的な切り捨てのみが実行されます。つまり、IMPORT with REPLACE コマンドは表を空にします。これによってオブジェクトの論理サイズは減りますが、ディスク上のストレージは割り振られたままの状態となります。

  このレジストリー変数は動的な変数です。これを設定または設定解除するときにはインスタンスを停止して開始する必要はありません。 **DB2_TRUNCATE_REUSESTORAGE** の設定はオンライン・バックアップの開始前に行い、設定解除はオンライン・バックアップの完了後に行え

ます。複数パーティション環境では、レジストリー変数はその変数が設定されるノードでのみアクティブになります。
DB2_TRUNCATE_REUSESTORAGE は DMS 永続オブジェクトに対してのみ有効になります。

SAP 環境で **DB2_WORKLOAD**=SAP が設定されると、このレジストリー変数のデフォルト値は IMPORT になります。

DB2_USE_DB2JCCT2_JROUTINE

- オペレーティング・システム: すべて
- デフォルト=設定しない。値: ON/YES/1/TRUE または OFF/NO/0/FALSE
- Java ストアード・プロシージャーおよびユーザー定義関数用のデフォルト・ドライバーは IBM Data Server Driver for JDBC and SQLJ です。非推奨のドライバーである、Java ルーチンに対する SQL 要求にサービスを提供するための DB2 JDBC Type 2 Driver for Linux, UNIX, and Windows を使用することを望む場合は、
DB2_USE_DB2JCCT2_JROUTINE を OFF、NO、0、または FALSE のいずれかに設定してください。

DB2_UTIL_MSGPATH

- オペレーティング・システム: すべて
- デフォルト=*instanceName*/tmp ディレクトリー
- **DB2_UTIL_MSGPATH** レジストリー変数は、SYSPROC.ADMIN_CMD プロシージャー、 SYSPROC.ADMIN_REMOVE_MSGS プロシージャー、および SYSPROC.ADMIN_GET_MSGS UDF と併用されます。これは、インスタンス・レベルで適用されます。 fenced ユーザー ID がファイルの読み取り、書き込み、およびその削除を行えるサーバー上のディレクトリー・パスを指し示すように **DB2_UTIL_MSGPATH** を設定できます。このディレクトリーはすべてのコーディネーター・パーティションからアクセス可能でなければならず、ユーティリティー・メッセージ・ファイルを入れるための十分なスペースが必要です。

  このパスが設定されない場合、デフォルトとして *instanceName*/tmp ディレクトリーが使用されます (DB2 のアンインストール時に *instanceName*/tmp がクリーンアップされることに注意してください)。

  このパスが変更される場合、前の設定が指し示していたディレクトリー内に存在するファイルは自動的には移動または削除されません。古いパスの下で作成されたメッセージの内容を検索する場合、これらのメッセージ (その先頭にはユーティリティー名が付けられ、その末尾にはユーザー ID が付けられる) は **DB2_UTIL_MSGPATH** が指し示す新規のディレクトリーに手動で移動させる必要があります。新規ロケーションに次のユーティリティー・メッセージ・ファイルが作成され、読み取られ、クリーンアップされます。

  DB2_UTIL_MSGPATH ディレクトリーの下にあるファイルはユーティリティー固有のファイルであり、トランザクションに依存するファイルではありません。それらはバックアップ・イメージの一部ではありません。
DB2_UTIL_MSGPATH ディレクトリーの下にあるファイルはユーザーに

よって管理されます。これは、ユーザーが SYSPROC.ADMIN_REMOVE_UTILMSG プロシージャーを使ってメッセージ・ファイルを削除できることを意味します。これらのファイルは DB2 のアンインストールでクリーンアップされません。

DB2_VENDOR_INI

- オペレーティング・システム: AIX、HP-UX、Solaris および Windows
- デフォルト = NULL。値: 任意の有効なパスおよびファイル。
- すべてのベンダー特定の環境設定を含むファイルを示します。データベース・マネージャーが開始した時に、値が読み取られます。

注: DB2_VENDOR_INI は、バージョン 9.5 より推奨されなくなり、今後のリリースでは除去される予定です。この環境変数の設定は、代わりに **DB2_DJ_INI** 変数が指定するファイル内に記述することができます。

DB2_XBSA_LIBRARY

- オペレーティング・システム: AIX、HP-UX、Solaris および Windows
- デフォルト = NULL。値: 任意の有効なパスおよびファイル。
- ベンダーの提供する XBSA ライブラリーを示します。 AIX で、共用オブジェクトが `shr.o` という名前でない場合は、設定にそのオブジェクトを組み込む必要があります。HP-UX、Solaris、および Windows では、共用オブジェクト名は必要ありません。たとえば、Legato's NetWorker Business Suite Module for DB2 を使用するには、レジストリー変数を次のように設定します。

  ```
  db2set DB2_XSBA_LIBRARY="/usr/lib/libxdb2.a(bsashr10.o)"
  ```

  XBSA インターフェースは、BACKUP DATABASE または RESTORE DATABASE コマンドから呼び出すことができます。例:

  ```
  db2 backup db sample use XBSA
  db2 restore db sample use XBSA
  ```

第 20 章 構成パラメーター

DB2 インスタンスまたはデータベースが作成されると、デフォルトのパラメーター値を持つ、対応する構成ファイルが作成されます。これらのパラメーター値を変更して、パフォーマンスを向上させ、インスタンスまたはデータベースの他の特性を改善することができます。

データベース・マネージャーは、パラメーターのデフォルト値に基づいてディスク・スペースおよびメモリーを割り振ります。この割り振りでは、ある程度のニーズは満たせるかもしれませんが、デフォルト値を使用して最高のパフォーマンスを引き出すことはできない場合もあります。

構成ファイル には、DB2 データベース製品および個々のデータベースに割り振られているリソースや、診断レベルなどの値を定義するパラメーターが含まれています。構成ファイルには次の 2 つのタイプがあります。

- DB2 インスタンスごとに存在するデータベース・マネージャー構成ファイル。
- データベースごとに存在するデータベース構成ファイル。

データベース・マネージャー構成ファイル は、 DB2 インスタンスの作成時に作成されます。これに含まれるパラメーターは、インスタンスの一部であるデータベースに関係なく、インスタンス・レベルでシステム・リソースに影響します。システムの構成によっては、これらのパラメーターの多くの値をシステム・デフォルト値から変更して、パフォーマンスを向上させたり容量を増やしたりすることができます。

それぞれのクライアントごとにも 1 つのデータベース・マネージャー構成ファイルがあります。このファイルには、特定のワークステーションのクライアント・イネーブラーに関する情報が入っています。サーバーに使用可能なパラメーターのサブセットは、クライアントにも適用できます。

データベース・マネージャーの構成パラメーターは、`db2systm` というファイルに保管されます。このファイルは、データベース・マネージャーのインスタンス作成時に作成されます。 Linux および UNIX 環境では、このファイルはデータベース・マネージャーのインスタンス用サブディレクトリー `sqllib` にあります。Windows では、このファイルは、デフォルト解釈として、 `sqllib` ディレクトリーのインスタンス用サブディレクトリーに入っています。DB2INSTPROF 変数が設定されている場合は、このファイルは、 DB2INSTPROF 変数が指定するディレクトリーの `instance` サブディレクトリーに入っています。

バージョン 9.5 では、グローバル・レベル (db2set -g) における DB2INSTPROF の暗黙的なデフォルト値は、DB2INSTPROF が設定されていない場合でも、下記の新しい場所に保管されます。

- Vista 環境では `ProgramData¥IBM¥DB2¥<DB2COPYNAME>`
- Windows 2003/XP 環境では `Documents and Settings¥All Users¥Application Data¥IBM¥DB2¥<Copy Name>`

実行時データ・ファイルの場所を指定するその他のプロファイル・レジストリー変数は、DB2INSTPROF の値を照会する必要があります。これには、次のような変数があります。

- DB2CLINIPATH
- DIAGPATH
- SPM_LOG_PATH

データベース構成パラメーターは、バージョン 8.2 よりも前のバージョンで作成されたデータベースの場合 SQLDBCON というファイル内に保管されますが、バージョン 8.2 以降で作成されたデータベースについては、データベース構成パラメーターは、すべて SQLDBCONF というファイル内に保管されます。これらのファイルは直接編集できないので、提供されている API、または API を呼び出すツールを使って、変更や表示を行ってください。

パーティション・データベース環境では、このファイルは、どのデータベース・パーティション・サーバーも同じファイルにアクセスすることができるように、ファイル共用システムに常駐します。データベース・マネージャーの構成は、すべてのデータベース・パーティション・サーバーで同じです。

パラメーターの多くは、データベース・マネージャーの単一のインスタンスに割り振られるシステム・リソースの量を制御したり、あるいは、環境上の考慮事項に基づいて、データベース・マネージャーのセットアップや異なる通信サブシステムを構成します。その他に、情報提供だけを目的とした、変更不能のパラメーターがあります。これらのすべてのパラメーターには、データベース・マネージャーのインスタンスに保管されているシングル・データベースとは関係なくグローバルに適用されるものです。

データベース構成ファイル は、データベースの作成時に作成され、データベースが常駐している場所に保管されます。データベースごとに 1 つの構成ファイルがあります。このデータベース構成ファイルで、データベースに割り当てるリソースの量を指定します。パラメーターの多くの値を変更して、パフォーマンスを向上させたり容量を増やしたりすることができます。特定のデータベースでの活動のタイプによっては、異なった変更が必要になることがあります。

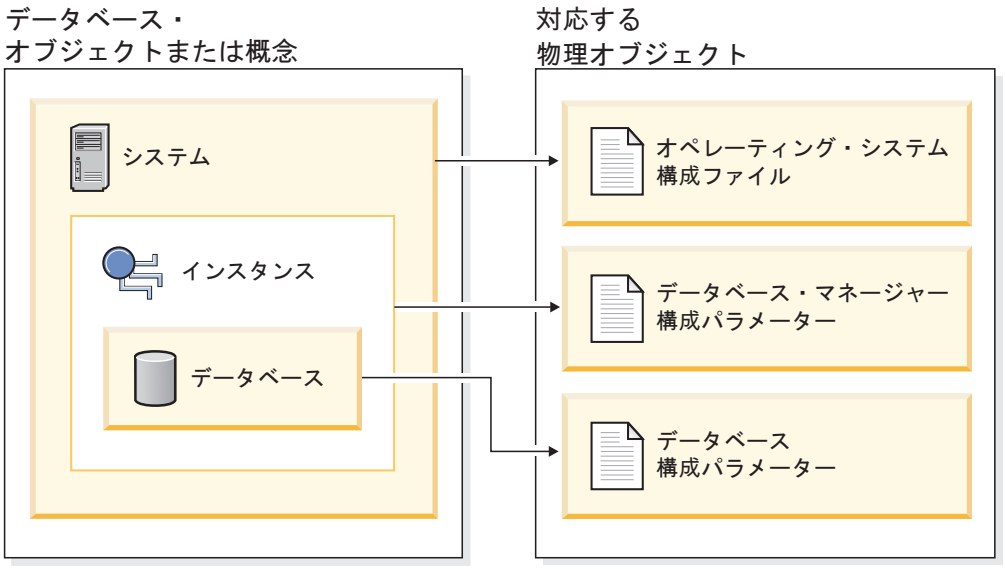


図 29. データベース・オブジェクトと構成ファイルの関係

構成パラメーターによる DB2 データベース・マネージャーの構成

データベース・マネージャーは、パラメーターのデフォルト値に基づいてディスク・スペースおよびメモリーを割り振ります。この割り振りでは、ある程度のニーズは満たせるかもしれませんが、デフォルト値を使用して最高のパフォーマンスを引き出すことはできない場合もあります。

デフォルト値は、比較的小さなメモリー・リソースを備えたマシンに合わせて設定されており、データベース・サーバー専用に設定されています。したがって、次の環境では、これらの値を修正する必要があります。

- データベースが大きい
- 接続の数が多い
- 特定のアプリケーションでハイパフォーマンスが求められている
- 固有の照会またはトランザクション・ロードまたはタイプを使用する

それぞれのトランザクション処理環境は、ある 1 つの面または複数の面において固有です。デフォルト構成を使用した場合、このような相違点によりデータベース・マネージャーのパフォーマンスが大きな影響を受けることがあります。このため、ユーザーの環境に合わせて構成を調整するよう強くお勧めします。

構成の調整を始めるに際しては、「構成アドバイザー」または AUTOCONFIGURE コマンドを使用することをお勧めします。「構成アドバイザー」または AUTOCONFIGURE コマンドは、ワークロードの特性に関する質問に対するユーザーからの回答を基にしてパラメーターの値を生成します。

いくつかの構成パラメーターを AUTOMATIC に設定することによって、データベース・マネージャーがこれらのパラメーターを自動的に管理して、現在のリソース要件を反映するようにできます。現在の内部設定を保守しながら、構成パラメータ

ーの AUTOMATIC 設定をオフにするには、UPDATE DATABASE CONFIGURATION コマンドの MANUAL キーワードを使用します。データベース・マネージャーがこれらパラメーターの値を更新した場合、get db/dbm cfg show detail コマンドによって新しい値が表示されます。

個々のデータベース用のパラメーターは、SQLDBCONF という構成ファイルに保管されます。このファイルは、データベースの他の制御ファイルとともに SQLnnnnn ディレクトリーに保管されています。 nnnnn はこのデータベースの作成時に割り当てられた番号です。各データベースにはそれぞれ構成ファイルがあり、パラメーターの多くはそのデータベースに割り振られるリソースの量を指定します。ファイルには、データベースの状況を示すフラグと共に、記述情報も入っています。

警告: データベース・マネージャーが提供している以外の方法を使用して db2systm、SQLDBCON、または SQLDBCONF を編集すると、データベースが使用不能になることがあります。これら文書化され、データベース・マネージャーによりサポートされている以外の方法で、ファイルを変更しないでください。

パーティション・データベース環境では、データベース・パーティションごとに別個の SQLDBCONF ファイルが存在します。 SQLDBCONF ファイルにある値は、データベース・パーティションごとに同じ値にも異なる値にもできますが、同機種環境では、構成パラメーター値はすべてのデータベース・パーティションで同じ値にすることが推奨されています。通常、異なるデータベース構成パラメーターの設定を必要とするカタログ・ノードが存在する可能性があり、また他の各データ・パーティションにそれらのマシン・タイプや他の情報に基づく、また別の値が設定されることもあります。

コマンド行プロセッサーを使用して、構成パラメーターを更新します。

設定を変更するためのコマンドは、次のように入力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の場合:

- GET DATABASE MANAGER CONFIGURATION (または GET DBM CFG)
- UPDATE DATABASE MANAGER CONFIGURATION (または UPDATE DBM CFG)
- RESET DATABASE MANAGER CONFIGURATION (または RESET DBM CFG)。すべてのデータベース・マネージャーのパラメーターをデフォルト値にリセットします。
- AUTOCONFIGURE。

データベース構成パラメーターの場合:

- GET DATABASE CONFIGURATION (または GET DB CFG)
- UPDATE DATABASE CONFIGURATION (または UPDATE DB CFG)
- RESET DATABASE CONFIGURATION (または RESET DB CFG)。すべてのデータベースのパラメーターをデフォルト値にリセットします。
- AUTOCONFIGURE。

アプリケーション・プログラミング・インターフェース (API) を使用して、構成パラメーターを更新します。

API は、アプリケーションまたはホスト言語プログラムから呼び出すことができます。次の DB2 API を呼び出して、構成パラメーターを表示または更新します。

- db2AutoConfig - 構成アドバイザーにアクセス
- db2CfgGet - データベース・マネージャーまたはデータベース構成パラメーターの取得
- db2CfgSet - データベース・マネージャーまたはデータベース構成パラメーターの設定

構成アシスタントを使用して、構成パラメーターを更新します。

構成アシスタントは、クライアント上のデータベース・マネージャー構成パラメーターを設定するために使用することもできます。その他のパラメーターはオンラインで変更できます。これらのパラメーターは、*構成可能なオンライン構成パラメーター* と呼ばれます。

更新された構成値を表示する

一部のデータベース・マネージャー構成パラメーターの場合、新しいパラメーター値を反映させるために、データベース・マネージャーを一度停止 (`db2stop`) して再始動 (`db2start`) する必要があります。

一部のデータベース・パラメーターの場合、データベースが再活動化されるか、またはオフラインからオンラインへ切り替えられるときにのみ変更が有効になります。これらの場合、まずすべてのアプリケーションをデータベースから切断する必要があります。 (データベースが活動化しているか、またはオフラインからオンラインへ切り替えられる場合は、いったん非活動化し、それから活動化し直す必要があります。) その後は、データベースに最初に接続した時点で、変更が反映されます。

インスタンスにアタッチしている間に、構成可能なオンラインのデータベース・マネージャー構成パラメーターの設定を変更する場合、UPDATE DBM CFG コマンドのデフォルトの動作は、即時に変更を適用することになっています。変更を即時に適用させたくない場合には、UPDATE DBM CFG コマンドで DEFERRED オプションを使用します。

データベース・マネージャーの構成パラメーターをオンラインで変更するには、以下のようにします。

```
db2 attach to <instance-name>
db2 update dbm cfg using <parameter-name> <value>
db2 detach
```

クライアントの場合、データベース・マネージャー構成パラメーターの変更は、クライアントがサーバーに次回接続したときに反映されます。

接続中に、構成可能なオンラインのデータベース構成パラメーターを変更する場合には、デフォルトの動作は、可能であればどこであってもオンラインで変更を適用することになっています。パラメーターの場合、スペースの割り振りに関係したオーバーヘッドが生じるため、設定を変更しても、それが反映されるまでにかなりの時間がかかることがあることに注意してください。コマンド行プロセッサーからオンラインで構成パラメーターを変更するには、データベースへの接続が必要です。データベースの構成パラメーターをオンラインで変更するには、以下のようにします。

```
db2 connect to <dbname>
db2 update db cfg using <parameter-name> <parameter-value>
db2 connect reset
```

それぞれの構成可能なオンラインの構成パラメーターには、そのパラメーターと関連した*伝搬*クラスがあります。伝搬クラスは、構成パラメーターへの変更が有効であることを予期できるときを示します。以下の 3 つの伝搬クラスがあります。

- **immediate**: コマンドまたは API 呼び出しで即時に変更するパラメーター。たとえば、*diaglevel* には、immediate という伝搬クラスがあります。
- **Statement boundary**: ステートメントおよびステートメントなどの境界で変更するパラメーター。たとえば、*sortheap* という値を変更する場合、すべての新規要求は、新規値を使用して開始します。
- **Transaction boundary**: トランザクション境界で変更するパラメーター。たとえば、*dl_expint* の新規値は、 COMMIT ステートメントの後に更新されます。

新しいパラメーター値は実際にはすぐに反映されませんが、(GET DATABASE MANAGER CONFIGURATION または GET DATABASE CONFIGURATION コマンドを使用して) パラメーターの設定値を表示すると、必ず最新の更新内容が表示されます。これらのコマンドで SHOW DETAIL 文節を使用してパラメーターの設定値を表示すると、メモリー内の最新の更新内容と値の両方が表示されます。

データベース構成パラメーターを更新した後に、アプリケーションを再バインドする

データベース構成パラメーターをいくつか変更しただけで、 SQL および XQuery オプティマイザーが選択するアクセス・プランに影響が出ます。これらのパラメーターのいずれかを変更した場合には、 SQL および XQuery ステートメントに最適なアクセス・プランが使用されるように、アプリケーションの再バインドを考慮してください。いずれかのパラメーターがオンラインで変更 (例えば、UPDATE DATABASE CONFIGURATION IMMEDIATE コマンドを使用して) されている場合、SQL および XQuery オプティマイザーは、新しい照会ステートメント用の新しいアクセス・プランを選択します。 ただし、照会ステートメントのキャッシュ内の既存の内容はパージされません。その照会キャッシュの内容を消去するには、FLUSH PACKAGE CACHE ステートメントを使用します。

注: ヘルプまたはその他の DB2 の資料では、構成パラメーター (たとえば、*userexit*) の大多数の許容値が「Yes」か「No」、あるいは「On」か「Off」であると記述されています。混乱しないように、「Yes」は「On」と同じであり、「No」は「Off」と同じものと見なします。

構成パラメーターのサマリー

以下の表には、データベース・サーバー用のデータベース・マネージャー およびデータベース 構成ファイルのパラメーターがリストされています。データベース・マネージャーおよびデータベースの構成パラメーターを変更する際は、各パラメーターの詳細情報も考慮に入れてください。デフォルトを含む個々のオペレーティング環境情報については、それぞれのパラメーターの説明に述べてあります。

データベース・マネージャー構成パラメーター・サマリー

一部のデータベース・マネージャー構成パラメーターの場合、新しいパラメーター値を反映させるために、データベース・マネージャーを一度停止 (db2stop) してから再始動 (db2start) する必要があります。その他のパラメーターはオンラインで変更できます。これらのパラメーターは、*構成可能なオンライン構成パラメーター* と呼ばれます。インスタンスにアタッチしている間に、構成可能なオンラインのデータベース・マネージャー構成パラメーターの設定を変更する場合、UPDATE DBM CFG コマンドのデフォルトの動作は、即時に変更を適用します。変更を即時に適用させたくない場合には、UPDATE DBM CFG コマンドで DEFERRED オプションを使用します。

以下の表の中の「自動」という列は、パラメーターが UPDATE DBM CFG コマンド上の AUTOMATIC キーワードをサポートするかどうかを示しています。

パラメーターを自動に更新する場合、AUTOMATIC キーワードに加えて開始値を指定することもできます。値は、それぞれのパラメーターごとに別の意味を持つ場合があり、ある場合には適用できないことがあるので注意してください。値を指定する前に、パラメーターの資料を読んで、値が何を表しているかを判別してください。次の例では、*num_poolagents* が AUTOMATIC に更新され、DB2 はプールするアイドル・エージェントの最小数として 20 を使用します。

```
db2 update dbm cfg using num_poolagents 20 automatic
```

AUTOMATIC フィーチャーを設定解除するには、パラメーターの値を更新するか、または MANUAL キーワードを使用することができます。パラメーターを MANUAL に更新すると、そのパラメーターは自動でなくなり、現行値 (get db/dbm cfg show detail コマンドで `Current Value` 列に表示される) に設定されます。

「パフォーマンスへの影響」の欄は、各パラメーターのシステム性能に与える影響の重大度を示しています。この欄について完全に正確な情報を記述することはできないので、大まかな情報として扱ってください。

- **高** - パラメーターがパフォーマンスに大きな影響を与える可能性があることを示します。これらのパラメーターの値を決定するときには注意が必要です。これは、場合によっては、提供されているデフォルト値を受け入れることを意味します。
- **中** - パラメーターがパフォーマンスに何らかの影響を与える可能性があることを示します。これらのパラメーターをどの程度調整する必要があるかは、ユーザーの環境および必要によって異なります。
- **低** - このパラメーターはパフォーマンスへの影響が低いことを示します。
- **なし** - このパラメーターはパフォーマンスへの直接の影響がないことを示します。これらのパラメーターをパフォーマンス向上のために調整する必要はありませんが、通信サポートなど、システム構成のその他の局面でこれらのパラメーターが非常に重要になる可能性があります。

「トークン」、「トークン値」、および「データ・タイプ」列は、 db2CfgGet または db2CfgSet API を呼び出す際に必要となる情報を提供します。 この情報には、構成パラメーター ID、*db2CfgParam* データ構造の *token* エレメントの項目、およびその構造に渡される値のデータ・タイプが含まれます。

表 65. 構成可能データベース・マネージャー構成パラメーター

パラメーター	オンラインで構成可能	自動	パフォーマンスへの影響	トークン	トークン値	データ・タイプ	追加情報
agent_stack_sz	なし	なし	低	SQLF_KTN_AGENT_STACK_SZ	61	Uint16	520 ページの『agent_stack_sz - エージェント・スタック・サイズ』
agentpri	なし	なし	高	SQLF_KTN_AGENTPRI	26	Sint16	522 ページの『agentpri - エージェントの優先順位』
aslheapsz	なし	なし	高	SQLF_KTN_ASLHEAPSZ	15	Uint32	524 ページの『aslheapsz - アプリケーション・サポート層ヒープ・サイズ』
audit_buf_sz	なし	なし	高	SQLF_KTN_AUDIT_BUF_SZ	312	Sint32	526 ページの『audit_buf_sz - 監査バッファー・サイズ』
authentication[1]	なし	なし	低	SQLF_KTN_AUTHENTICATION	78	Uint16	527 ページの『authentication - 認証タイプ』
catalog_Noauth	あり	なし	なし	SQLF_KTN_CATALOG_NOAUTH	314	Uint16	528 ページの『catalog_noauth - 権限なしで許可されるカタログ』
clnt_krb_plugin	なし	なし	なし	SQLF_KTN_CLNT_KRB_PLUGIN	812	char(33)	529 ページの『clnt_krb_plugin - クライアント Kerberos プラグイン』
clnt_pw_plugin	なし	なし	なし	SQLF_KTN_CLNT_PW_PLUGIN	811	char(33)	529 ページの『clnt_pw_plugin - クライアント・ユーザー ID パスワード・プラグイン』
cluster_mgr	なし	なし	なし	SQLF_KTN_CLUSTER_MGR	920	char(262)	530 ページの『cluster_mgr - クラスター・マネージャー名』
comm_bandwidth	あり	なし	中	SQLF_KTN_COMM_BANDWIDTH	307	float	530 ページの『comm_bandwidth - 通信スピード』
conn_elapse	あり	なし	中	SQLF_KTN_CONN_ELAPSE	508	Uint16	531 ページの『conn_elapse - 接続経過時間』
cpuspeed	あり	なし	高	SQLF_KTN_CPUSPEED	42	float	531 ページの『cpuspeed - CPU 速度』
dft_account_str	あり	なし	なし	SQLF_KTN_DFT_ACCOUNT_STR	28	char(25)	532 ページの『dft_account_str - デフォルト・チャージバック・アカウント』
dft_monswitches • *dft_mon_bufpool* • *dft_mon_lock* • *dft_mon_sort* • *dft_mon_stmt* • *dft_mon_table* • *dft_mon_timestamp* • *dft_mon_uow*	あり	なし	中	SQLF_KTN_DFT_MONSWITCHES[2] • SQLF_KTN_DFT_MON_BUFPOOL • SQLF_KTN_DFT_MON_LOCK • SQLF_KTN_DFT_MON_SORT • SQLF_KTN_DFT_MON_STMT • SQLF_KTN_DFT_MON_TABLE • SQLF_KTN_DFT_MON_TIMESTAMP • SQLF_KTN_DFT_MON_UOW	29 • 33 • 34 • 35 • 31 • 32 • 36 • 30	Uint16 • Uint16 • Uint16 • Uint16 • Uint16 • Uint16 • Uint16 • Uint16	533 ページの『dft_monswitches - デフォルトのデータベース・システム・モニター・スイッチ』
dftdbpath	あり	なし	なし	SQLF_KTN_DFTDBPATH	27	char(215)	534 ページの『dftdbpath - デフォルト・データベース・パス』
diaglevel	あり	なし	低	SQLF_KTN_DIAGLEVEL	64	Uint16	535 ページの『diaglevel - 診断エラー・キャプチャー・レベル』
diagpath	あり	なし	なし	SQLF_KTN_DIAGPATH	65	char(215)	536 ページの『diagpath - 診断データ・ディレクトリー・パス』

表 65. 構成可能データベース・マネージャー構成パラメーター (続き)

パラメーター	オンラインで構成可能	自動	パフォーマンスへの影響	トークン	トークン値	データ・タイプ	追加情報
dir_cache	なし	なし	中	SQLF_KTN_DIR_CACHE	40	Uint16	538 ページの『dir_cache - ディレクトリー・キャッシュ・サポート』
discover[3]	なし	なし	中	SQLF_KTN_DISCOVER	304	Uint16	539 ページの『discover - ディスカバリー・モード』
discover_inst	あり	なし	低	SQLF_KTN_DISCOVER_INST	308	Uint16	540 ページの『discover_inst - サーバー・インスタンスのディスカバリー』
fcm_num_buffers	あり	あり	中	SQLF_KTN_FCM_NUM_ BUFFERS	503	Uint32	540 ページの『fcm_num_buffers - FCM バッファー数』
fcm_num_ channels	あり	あり	中	SQLF_KTN_FCM_NUM_ CHANNELS	902	Uint32	541 ページの『fcm_num_channels - FCM チャネル数』
fed_noauth	あり	なし	なし	SQLF_KTN_FED_NOAUTH	806	Uint16	542 ページの『fed_noauth - フェデレーテッド・データベース認証をバイパス』
federated	あり	なし	中	SQLF_KTN_FEDERATED	604	Sint16	543 ページの『federated - フェデレーテッド・データベース・システム・サポート』
federated_async	あり	あり	中	SQLF_KTN_FEDERATED_ ASYNC	849	Sint32	543 ページの『federated_async - 照会ごとの非同期 TQ の最大数構成パラメーター』
fenced_pool	あり	あり	中	SQLF_KTN_FENCED_POOL	80	Sint32	544 ページの『fenced_pool - fenced プロセスの最大数』
group_plugin	なし	なし	なし	SQLF_KTN_GROUP_PLUGIN	810	char(33)	546 ページの『group_plugin - グループ・プラグイン』
health_mon	あり	なし	低	SQLF_KTN_HEALTH_MON	804	Uint16	546 ページの『health_mon - ヘルス・モニター』
indexrec[4]	あり	なし	中	SQLF_KTN_INDEXREC	20	Uint16	547 ページの『indexrec - 索引再作成時点』
instance_memory	あり	あり	中	SQLF_KTN_INSTANCE_ MEMORY	803	Uint64	549 ページの『instance_memory - インスタンス・メモリー』
intra_parallel	なし	なし	高	SQLF_KTN_INTRA_PARALLEL	306	Sint16	552 ページの『intra_parallel - パーティション内並列処理機能の使用可能化』
java_heap_sz	なし	なし	高	SQLF_KTN_JAVA_HEAP_SZ	310	Sint32	552 ページの『java_heap_sz - Java インタープリター最大ヒープ・サイズ』
jdk_path	なし	なし	なし	SQLF_KTN_JDK_PATH	311	char(255)	553 ページの『jdk_path - Software Developer's Kit for Java インストール・パス』
keepfenced	なし	なし	中	SQLF_KTN_KEEPFENCED	81	Uint16	554 ページの『keepfenced - fenced プロセスの維持』
local_gssplugin	なし	なし	なし	SQLF_KTN_LOCAL_ GSSPLUGIN	816	char(33)	555 ページの『local_gssplugin - ローカル・インスタンス・レベル許可に使用する GSS API プラグイン』
max_connections	あり	あり	中	SQLF_KTN_MAX_ CONNECTIONS	802	Sint32	555 ページの『max_connections - クライアント接続の最大数』
max_connretries	あり	なし	中	SQLF_KTN_MAX_ CONNRETRIES	509	Uint16	556 ページの『max_connretries - ノード接続再試行回数』
max_coordagents	あり	あり	中	SQLF_KTN_MAX_ COORDAGENTS	501	Sint32	557 ページの『max_coordagents - コーディネーター・エージェントの最大数』
max_querydegree	あり	なし	高	SQLF_KTN_MAX_ QUERYDEGREE	303	Sint32	558 ページの『max_querydegree - 照会の最大並列処理多重度』

表 65. 構成可能データベース・マネージャー構成パラメーター (続き)

パラメーター	オンラインで構成可能	自動	パフォーマンスへの影響	トークン	トークン値	データ・タイプ	追加情報
max_time_diff	なし	なし	中	SQLF_KTN_MAX_TIME_DIFF	510	Uint16	559 ページの『max_time_diff - ノード間最大時差』
mon_heap_sz	あり	あり	低	SQLF_KTN_MON_HEAP_SZ	79	Uint16	561 ページの『mon_heap_sz - データベース・システム・モニター・ヒープ・サイズ』
notifylevel	あり	なし	低	SQLF_KTN_NOTIFYLEVEL	605	Sint16	563 ページの『notifylevel - 通知レベル』
num_initagents	なし	なし	中	SQLF_KTN_NUM_ INITAGENTS	500	Uint32	564 ページの『num_initagents - プール内エージェントの初期数』
num_initfenced	なし	なし	中	SQLF_KTN_NUM_ INITFENCED	601	Sint32	565 ページの『num_initfenced - fenced プロセスの初期数』
num_poolagents	あり	あり	高	SQLF_KTN_NUM_ POOLAGENTS	502	Sint32	565 ページの『num_poolagents - エージェント・プール・サイズ』
numdb	なし	なし	低	SQLF_KTN_NUMDB	6	Uint16	567 ページの『numdb - ホストおよび System i データベースを含めた並行アクティブ・データベースの最大数』
query_heap_sz	なし	なし	中	SQLF_KTN_QUERY_HEAP_SZ	49	Sint32	567 ページの『query_heap_sz - 照会ヒープ・サイズ』
resync_interval	なし	なし	なし	SQLF_KTN_RESYNC_ INTERVAL	68	Uint16	569 ページの『resync_interval - トランザクション再同期インターバル』
rqrioblk	なし	なし	高	SQLF_KTN_RQRIOBLK	1	Uint16	569 ページの『rqrioblk - クライアント入出力ブロック・サイズ』
sheapthres	なし	なし	高	SQLF_KTN_SHEAPTHRES	21	Uint32	571 ページの『sheapthres - ソート・ヒープしきい値』
spm_log_file_sz	なし	なし	低	SQLF_KTN_SPM_LOG_FILE_ SZ	90	Sint32	572 ページの『spm_log_file_sz - 同期点マネージャー・ログ・ファイル・サイズ』
spm_log_path	なし	なし	中	SQLF_KTN_SPM_LOG_PATH	313	char(226)	573 ページの『spm_log_path - 同期点マネージャー・ログ・ファイル・パス』
spm_max_resync	なし	なし	低	SQLF_KTN_SPM_MAX_ RESYNC	91	Sint32	574 ページの『spm_max_resync - 同期点マネージャー再同期エージェント数の限度』
spm_name	なし	なし	なし	SQLF_KTN_SPM_NAME	92	char(8)	574 ページの『spm_name - 同期点マネージャー名』
srvcon_auth	なし	なし	なし	SQLF_KTN_SRVCON_AUTH	815	Uint16	574 ページの『srvcon_auth - サーバーでの着信接続の認証タイプ』
srvcon_gssplugin_ list	なし	なし	なし	SQLF_KTN_SRVCON_ GSSPLUGIN_ LIST	814	char(256)	575 ページの『srvcon_gssplugin_list - サーバーでの着信接続用の GSS API プラグインのリスト』
srv_plugin_mode	なし	なし	なし	SQLF_KTN_SRV_PLUGIN_ MODE	809	Uint16	576 ページの『srv_plugin_mode - サーバー・プラグイン・モード』
srvcon_pw_plugin	なし	なし	なし	SQLF_KTN_SRVCON_PW_ PLUGIN	813	char(33)	576 ページの『srvcon_pw_plugin - サーバーでの着信接続用のユーザー ID-パスワード・プラグイン』

表 65. 構成可能データベース・マネージャー構成パラメーター (続き)

パラメーター	オンラインで構成可能	自動	パフォーマンスへの影響	トークン	トークン値	データ・タイプ	追加情報
start_stop_time	あり	なし	低	SQLF_KTN_START_STOP_TIME	511	Uint16	577 ページの『start_stop_time - タイムアウトの開始および停止』
svcename	なし	なし	なし	SQLF_KTN_SVCENAME	24	char(14)	578 ページの『svcename - TCP/IP サービス名』
sysadm_group	なし	なし	なし	SQLF_KTN_SYSADM_GROUP	39	char(128)	578 ページの『sysadm_group - システム管理権限グループ名』
sysctrl_group	なし	なし	なし	SQLF_KTN_SYSCTRL_GROUP	63	char(128)	579 ページの『sysctrl_group - システム制御権限グループ名』
sysmaint_group	なし	なし	なし	SQLF_KTN_SYSMAINT_GROUP	62	char(128)	580 ページの『sysmaint_group - システム保守権限グループ名』
sysmon_group	なし	なし	なし	SQLF_KTN_SYSMON GROUP	808	char(128)	581 ページの『sysmon_group - システム・モニター権限グループ名』
tm_database	なし	なし	なし	SQLF_KTN_TM_DATABASE	67	char(8)	582 ページの『tm_database - トランザクション・マネージャー・データベース名』
tp_mon_name	なし	なし	なし	SQLF_KTN_TP_MON_NAME	66	char(19)	582 ページの『tp_mon_name - トランザクション・プロセッサー・モニター名』
trust_allclnts[5]	なし	なし	なし	SQLF_KTN_TRUST_ALLCLNTS	301	Uint16	584 ページの『trust_allclnts - 全クライアントのトラステッド化』
trust_clntauth	なし	なし	なし	SQLF_KTN_TRUST_CLNTAUTH	302	Uint16	585 ページの『trust_clntauth - トラステッド・クライアント認証』
util_impact_lim	あり	なし	高	SQLF_KTN_UTIL_IMPACT_LIM	807	Uint32	586 ページの『util_impact_lim - インスタンス影響ポリシー』

表 65. 構成可能データベース・マネージャー構成パラメーター (続き)

パラメーター	オンラインで構成可能	自動	パフォーマンスへの影響	トークン	トークン値	データ・タイプ	追加情報

注:

1. 有効な値 (sqlenv.h で定義):

```
SQL_AUTHENTICATION_SERVER (0)
SQL_AUTHENTICATION_CLIENT (1)
SQL_AUTHENTICATION_DCS (2)
SQL_AUTHENTICATION_DCE (3)
SQL_AUTHENTICATION_SVR_ENCRYPT (4)
SQL_AUTHENTICATION_DCS_ENCRYPT (5)
SQL_AUTHENTICATION_DCE_SVR_ENC (6)
SQL_AUTHENTICATION_KERBEROS (7)
SQL_AUTHENTICATION_KRB_SVR_ENC (8)
SQL_AUTHENTICATION_GSSPLUGIN (9)
SQL_AUTHENTICATION_GSS_SVR_ENC (10)
SQL_AUTHENTICATION_DATAENC (11)
SQL_AUTHENTICATION_DATAENC_CMP (12)
SQL_AUTHENTICATION_NOT_SPEC (255)
```

2.

```
Bit 1 (xxxx xxx1): dft_mon_uow
Bit 2 (xxxx xx1x): dft_mon_stmt
Bit 3 (xxxx x1xx): dft_mon_table
Bit 4 (xxxx 1xxx): dft_mon_buffpool
Bit 5 (xxx1 xxxx): dft_mon_lock
Bit 6 (xx1x xxxx): dft_mon_sort
Bit 7 (x1xx xxxx): dft_mon_timestamp
```

3. 有効な値 (sqlutil.h で定義):

```
SQLF_DSCVR_KNOWN (1)
SQLF_DSCVR_SEARCH (2)
```

4. 有効な値 (sqlutil.h で定義):

```
SQLF_INX_REC_SYSTEM (0)
SQLF_INX_REC_REFERENCE (1)
```

5. 有効な値 (sqlutil.h で定義):

```
SQLF_TRUST_ALLCLNTS_NO (0)
SQLF_TRUST_ALLCLNTS_YES (1)
SQLF_TRUST_ALLCLNTS_DRDAONLY (2)
```

表 66. 情報提供用のデータベース・マネージャー構成パラメーター

パラメーター	トークン	トークン値	データ・タイプ	追加情報
nodetype[1]	SQLF_KTN_NODETYPE	100	Uint16	562 ページの『nodetype - マシン・ノード・タイプ』
release	SQLF_KTN_RELEASE	101	Uint16	569 ページの『release - 構成ファイル・リリース・レベル』

注:

1. 有効な値 (sqlutil.h で定義):

```
SQLF_NT_STANDALONE (0)
SQLF_NT_SERVER (1)
SQLF_NT_REQUESTOR (2)
SQLF_NT_STAND_REQ (3)
SQLF_NT_MPP (4)
SQLF_NT_SATELLITE (5)
```

データベース構成パラメーター・サマリー

次の表は、データベース構成ファイルに入っているパラメーターをリストしています。データベース構成パラメーターを変更する際は、パラメーターの詳細情報も考慮に入れてください。

一部のデータベース構成パラメーターの場合、データベースが再活動化されるときにのみ変更が有効になります。これらの場合、まずすべてのアプリケーションをデータベースから切断する必要があります。 (データベースが活動化している場合は、いったん非活動化し、それから活動化し直す必要があります。) 次にデータベースに接続する際に、変更が有効になります。その他のパラメーターはオンラインで変更できます。これらのパラメーターは、*構成可能なオンライン構成パラメーター* と呼ばれます。

「自動」、「パフォーマンスへの影響」、「トークン」、「トークン値」、「データ・タイプ」の各列の説明については、上記の『データベース・マネージャー構成パラメーター・サマリー』セクションを参照してください。

UPDATE DB CFG コマンドでも AUTOMATIC キーワードがサポートされています。次の例では、*database_memory* は AUTOMATIC に更新され、データベース・マネージャーはこのパラメーターに追加の変更を加える際に 20000 を開始値として使用します。

```
db2 update db cfg using for sample using database_memory 20000 automatic
```

バージョン 9.5 からは、db2_all コマンドを発行しなくても、あるいは各パーティションを個別に更新またはリセットしなくても、いくつか、あるいはすべてのプラットフォームに渡ってデータベース構成パラメーター値を更新およびリセットすることができます。詳しくは、『複数パーティション間でのデータベースの構成』を参照してください。

表 67. 構成可能なデータベース構成パラメーター

パラメーター	オンラインで構成可能	自動	パフォーマンスへの影響	トークン	トークン値	データ・タイプ	追加情報
alt_collate	なし	なし	なし	SQLF_DBTN_ALT_COLLATE	809	Uint32	587 ページの『alt_collate - 代替照合シーケンス』
applheapsz	あり	あり	中	SQLF_DBTN_APPLHEAPSZ	51	Uint16	591 ページの『applheapsz - アプリケーション・ヒープ・サイズ』
appl_memory	あり	あり	中	SQLF_DBTN_APPL_MEMORY	904	Uint64	591 ページの『appl_memory - アプリケーション・メモリー構成パラメーター』
archretrydelay	あり	なし	なし	SQLF_DBTN_ ARCHRETRYDELAY	828	Uint16	592 ページの『archretrydelay - エラー時のアーカイブ再試行遅延』

表 67. 構成可能なデータベース構成パラメーター (続き)

パラメーター	オンラインで構成可能	自動	パフォーマンスへの影響	トークン	トークン値	データ・タイプ	追加情報
• *auto_maint* • *auto_db_backup* • *auto_tbl_maint* • *auto_runstats* • *auto_stats_prof* • *auto_stmt_stats* • *auto_prof_upd* • *auto_reorg*	あり	なし	中	• SQLF_DBTN_AUTO_MAINT • SQLF_DBTN_AUTO_DB_BACKUP • SQLF_DBTN_AUTO_TBL_MAINT • SQLF_DBTN_AUTO_RUNSTATS • SQLF_DBTN_AUTO_STATS_PROF • SQLF_DBTN_AUTO_STMT_STATS • SQLF_DBTN_AUTO_PROF_UPD • SQLF_DBTN_AUTO_REORG	• 831 • 833 • 835 • 837 • 839 • 905 • 844 • 841	Uint16	593 ページの『auto_maint - 自動保守』
auto_del_rec_obj	あり	なし	中	SQLF_DBTN_AUTO_DEL_REC_OBJ	912	Uint16	593 ページの『auto_del_rec_obj - リカバリー・オブジェクトの自動削除構成パラメーター』
autorestart	あり	なし	低	SQLF_DBTN_AUTO_RESTART	25	Uint16	596 ページの『autorestart - 自動再始動使用可能』
avg_appls	あり	あり	高	SQLF_DBTN_AVG_APPLS	47	Uint16	596 ページの『avg_appls - アクティブ・アプリケーションの平均数』
blk_log_dsk_ful	あり	なし	なし	SQLF_DBTN_BLK_LOG_DSK_FUL	804	Uint16	598 ページの『blk_log_dsk_ful - ログ・ディスク・フルによるアプリケーション中断』
catalogcache_sz	あり	なし	中	SQLF_DBTN_CATALOGCACHE_SZ	56	Sint32	598 ページの『catalogcache_sz - カタログ・キャッシュ・サイズ』
chngpgs_thresh	なし	なし	高	SQLF_DBTN_CHNGPGS_THRESH	38	Uint16	600 ページの『chngpgs_thresh - 変更済みページしきい値』
database_memory	あり	あり	中	SQLF_DBTN_DATABASE_MEMORY	803	Uint64	603 ページの『database_memory - データベース共用メモリー・サイズ』
dbheap	あり	あり	中	SQLF_DBTN_DB_HEAP	58	Uint64	606 ページの『dbheap - データベース・ヒープ』
db_mem_thresh	あり	なし	低	SQLF_DBTN_DB_MEM_THRESH	849	Uint16	605 ページの『db_mem_thresh - データベース・メモリーしきい値』
decflt_rounding	なし	なし	なし	SQLF_DBTN_DECFLT_ROUNDING	913	Unit16	608 ページの『decflt_rounding - 10 進浮動小数点丸め構成パラメーター』
dft_degree	あり	なし	高	SQLF_DBTN_DFT_DEGREE	301	Sint32	609 ページの『dft_degree - デフォルト多重度』
dft_extent_sz	あり	なし	中	SQLF_DBTN_DFT_EXTENT_SZ	54	Uint32	610 ページの『dft_extent_sz - 表スペースのデフォルトのエクステント・サイズ』
dft_loadrec_ses	あり	なし	中	SQLF_DBTN_DFT_LOADREC_SES	42	Sint16	610 ページの『dft_loadrec_ses - ロード・リカバリー・セッションのデフォルト数』
dft_mttb_types	なし	なし	なし	SQLF_DBTN_DFT_MTTB_TYPES	843	Uint32	611 ページの『dft_mttb_types - 最適化用デフォルト保守表タイプ』

表 67. 構成可能なデータベース構成パラメーター (続き)

パラメーター	オンラインで構成可能	自動	パフォーマンスへの影響	トークン	トークン値	データ・タイプ	追加情報
dft_prefetch_sz	あり	あり	中	SQLF_DBTN_DFT_PREFETCH_SZ	40	Sint16	611 ページの『dft_prefetch_sz - デフォルト・プリフェッチ・サイズ』
dft_queryopt	あり	なし	中	SQLF_DBTN_DFT_QUERYOPT	57	Sint32	613 ページの『dft_queryopt - デフォルト照会最適化クラス』
dft_refresh_age	なし	なし	中	SQLF_DBTN_DFT_REFRESH_AGE	702	char(22)	613 ページの『dft_refresh_age - デフォルトの REFRESH AGE』
dft_sqlmathwarn	なし	なし	なし	SQLF_DBTN_DFT_SQLMATHWARN	309	Sint16	614 ページの『dft_sqlmathwarn - 演算例外時継続』
discover_db	あり	なし	中	SQLF_DBTN_DISCOVER	308	Uint16	615 ページの『discover_db - データベース・ディスカバリー』
dlchktime	あり	なし	中	SQLF_DBTN_DLCHKTIME	9	Uint32	616 ページの『dlchktime - デッドロック・チェック・インターバル』
dyn_query_mgmt	なし	なし	低	SQLF_DBTN_DYN_QUERY_MGMT	604	Uint16	617 ページの『dyn_query_mgmt - 動的 SQL および XQuery 照会管理』
enable_xmlchar	あり	なし	なし	SQLF_DBTN_ENABLE_XMLCHAR	853	Uint32	617 ページの『enable_xmlchar - XML への変換の有効化構成パラメーター』
failarchpath	あり	なし	なし	SQLF_DBTN_FAILARCHPATH	826	char(243)	618 ページの『failarchpath - フェイルオーバー・ログ・アーカイブ・パス』
hadr_local_host	なし	なし	なし	SQLF_DBTN_HADR_LOCAL_HOST	811	char(256)	619 ページの『hadr_local_host - HADR ローカル・ホスト名』
hadr_local_svc	なし	なし	なし	SQLF_DBTN_HADR_LOCAL_SVC	812	char(41)	620 ページの『hadr_local_svc - HADR ローカル・サービス名』
hadr_peer_ window	なし	なし	低 (注 4 を参照)	SQLF_DBTN_HADR_PEER_WINDOW	914	Uint32	620 ページの『hadr_peer_window - HADR ピア・ウィンドウ構成パラメーター』
hadr_remote_host	なし	なし	なし	SQLF_DBTN_HADR_REMOTE_HOST	813	char(256)	620 ページの『hadr_remote_host - HADR リモート・ホスト名』
hadr_remote_inst	なし	なし	なし	SQLF_DBTN_HADR_REMOTE_INST	815	char(9)	621 ページの『hadr_remote_inst - リモート・サーバーの HADR インスタンス名』
hadr_remote_svc	なし	なし	なし	SQLF_DBTN_HADR_REMOTE_SVC	814	char(41)	621 ページの『hadr_remote_svc - HADR リモート・サービス名』
hadr_syncmode	なし	なし	なし	SQLF_DBTN_HADR_SYNCMODE	817	Uint32	622 ページの『hadr_syncmode - 対等状態にあるログ書き込みのための HADR 同期モード』
hadr_timeout	なし	なし	なし	SQLF_DBTN_HADR_TIMEOUT	816	Uint32	623 ページの『hadr_timeout - HADR タイムアウト値』
indexrec[2]	あり	なし	中	SQLF_DBTN_INDEXREC	30	Uint16	547 ページの『indexrec - 索引再作成時点』
locklist	あり	あり	高 (エスカレーションに影響するとき)	SQLF_DBTN_LOCK_LIST	704	Uint64	624 ページの『locklist - ロック・リスト用最大ストレージ』
locktimeout	なし	なし	中	SQLF_DBTN_LOCKTIMEOUT	34	Sint16	627 ページの『locktimeout - ロック・タイムアウト』
logarchmeth1	あり	なし	なし	SQLF_DBTN_LOGARCHMETH1	822	char(252)	628 ページの『logarchmeth1 - 1 次ログ・アーカイブ方式』

表 67. 構成可能なデータベース構成パラメーター (続き)

パラメーター	オンラインで構成可能	自動	パフォーマンスへの影響	トークン	トークン値	データ・タイプ	追加情報
logarchmeth2	あり	なし	なし	SQLF_DBTN_LOGARCHMETH2	823	char(252)	629 ページの『logarchmeth2 - 2 次ログ・アーカイブ方式』
logarchopt1	あり	なし	なし	SQLF_DBTN_LOGARCHOPT1	824	char(243)	631 ページの『logarchopt1 - 1 次ログ・アーカイブ・オプション』
logarchopt2	あり	なし	なし	SQLF_DBTN_LOGARCHOPT2	825	char(243)	631 ページの『logarchopt2 - 2 次ログ・アーカイブ・オプション』
logbufsz	なし	なし	高	SQLF_DBTN_LOGBUFSZ	33	Uint16	632 ページの『logbufsz - ログ・バッファー・サイズ』
logfilsiz	なし	なし	中	SQLF_DBTN_LOGFIL_SIZ	92	Uint32	632 ページの『logfilsiz - ログ・ファイルのサイズ』
logindexbuild	あり	あり	なし	SQLF_DBTN_LOGINDEXBUILD	818	Uint32	634 ページの『logindexbuild - 作成されるログ索引ページ』
logprimary	なし	なし	中	SQLF_DBTN_LOGPRIMARY	16	Uint16	635 ページの『logprimary - 1 次ログ・ファイル数』
logretain[3]	なし	なし	低	SQLF_DBTN_LOG_RETAIN	23	Uint16	636 ページの『logretain - ログ保持使用可能』
logsecond	あり	なし	中	SQLF_DBTN_LOGSECOND	17	Uint16	637 ページの『logsecond - 2 次ログ・ファイル数』
max_log	あり	あり		SQLF_DBTN_MAX_LOG	807	Uint16	638 ページの『max_log - トランザクション当たりの最大ログ』
maxappls	あり	あり	中	SQLF_DBTN_MAXAPPLS	6	Uint16	639 ページの『maxappls - アクティブ・アプリケーションの最大数』
maxfilop	あり	なし	中	SQLF_DBTN_MAXFILOP	3	Uint16	640 ページの『maxfilop - アプリケーション単位の最大データベース・ファイル・オープン数』
maxlocks	あり	あり	高 (エスカレーションに影響するとき)	SQLF_DBTN_MAXLOCKS	15	Uint16	641 ページの『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』
min_dec_div_3	なし	なし	高	SQLF_DBTN_MIN_DEC_DIV_3	605	Sint32	643 ページの『min_dec_div_3 - 10 進数除算の位取り 3』
mincommit	あり	なし	高	SQLF_DBTN_MINCOMMIT	32	Uint16	644 ページの『mincommit - グループへのコミット数』
mirrorlogpath	なし	なし	低	SQLF_DBTN_MIRRORLOGPATH	806	char(242)	646 ページの『mirrorlogpath - ミラー・ログ・パス』
newlogpath	なし	なし	低	SQLF_DBTN_NEWLOGPATH	20	char(242)	647 ページの『newlogpath - データベース・ログ・パスの変更』
num_db_backups	あり	なし	なし	SQLF_DBTN_NUM_DB_BACKUPS	601	Uint16	649 ページの『num_db_backups - データベース・バックアップの数』
num_freqvalues	あり	なし	低	SQLF_DBTN_NUM_FREQVALUES	36	Uint16	650 ページの『num_freqvalues - 保存される高頻度値の数』
num_iocleaners	なし	あり	高	SQLF_DBTN_NUM_IOCLEANERS	37	Uint16	651 ページの『num_iocleaners - 非同期ページ・クリーナーの数』
num_ioservers	なし	あり	高	SQLF_DBTN_NUM_IOSERVERS	39	Uint16	652 ページの『num_ioservers - 入出力サーバーの数』
num_log_span	あり	あり		SQLF_DBTN_NUM_LOG_SPAN	808	Uint16	653 ページの『num_log_span - ログ・スパンの数』

表 67. 構成可能なデータベース構成パラメーター (続き)

パラメーター	オンラインで構成可能	自動	パフォーマンスへの影響	トークン	トークン値	データ・タイプ	追加情報
num_quantiles	あり	なし	低	SQLF_DBTN_NUM_QUANTILES	48	Uint16	654 ページの『num_quantiles - 列の変位値の数』
numarchretry	あり	なし	なし	SQLF_DBTN_NUMARCHRETRY	827	Uint16	655 ページの『numarchretry - エラー時の再試行数』
overflowlogpath	なし	なし	中	SQLF_DBTN_OVERFLOWLOGPATH	805	char(242)	656 ページの『overflowlogpath - オーバーフロー・ログ・パス』
pckcachesz	あり	あり	高	SQLF_DBTN_PCKCACHE_SZ	505	Uint32	658 ページの『pckcachesz - パッケージ・キャッシュ・サイズ』
rec_his_retentn	なし	なし	なし	SQLF_DBTN_REC_HIS_RETENTN	43	Sint16	660 ページの『rec_his_retentn - リカバリー履歴保持期間』
self_tuning_mem	あり	なし	高	SQLF_DBTN_SELF_TUNING_MEM	848	Uint16	662 ページの『self_tuning_mem-セルフチューニング・メモリー』
seqdetect	あり	なし	高	SQLF_DBTN_SEQDETECT	41	Uint16	664 ページの『seqdetect - 順次検出フラグ』
sheapthres_shr	あり	あり	高	SQLF_DBTN_SHEAPTHRES_SHR	802	Uint32	664 ページの『sheapthres_shr - 共用ソートのソート・ヒープのしきい値』
softmax	なし	なし	中	SQLF_DBTN_SOFTMAX	5	Uint16	666 ページの『softmax - リカバリー範囲およびソフト・チェックポイント・インターバル』
sortheap	あり	あり	高	SQLF_DBTN_SORT_HEAP	52	Uint32	668 ページの『sortheap - ソート・ヒープ・サイズ』
stat_heap_sz	あり	あり	低	SQLF_DBTN_STAT_HEAP_SZ	45	Uint32	669 ページの『stat_heap_sz - 統計ヒープ・サイズ』
stmtheap	あり	あり	中	SQLF_DBTN_STMT_HEAP	821	Uint32	670 ページの『stmtheap - ステートメント・ヒープ・サイズ』
trackmod	なし	なし	低	SQLF_DBTN_TRACKMOD	703	Uint16	671 ページの『trackmod - 変更ページの追跡使用可能化』
tsm_mgmtclass	あり	なし	なし	SQLF_DBTN_TSM_MGMTCLASS	307	char(30)	671 ページの『tsm_mgmtclass - Tivoli Storage Manager 管理クラス』
tsm_nodename	あり	なし	なし	SQLF_DBTN_TSM_NODENAME	306	char(64)	672 ページの『tsm_nodename - Tivoli Storage Manager ノード名』
tsm_owner	あり	なし	なし	SQLF_DBTN_TSM_OWNER	305	char(64)	672 ページの『tsm_owner - Tivoli Storage Manager 所有者名』
tsm_password	あり	なし	なし	SQLF_DBTN_TSM_PASSWORD	501	char(64)	673 ページの『tsm_password - Tivoli Storage Manager パスワード』
userexit	なし	なし	低	SQLF_DBTN_USER_EXIT	24	Uint16	673 ページの『userexit - ユーザー出口使用可能』
util_heap_sz	あり	なし	低	SQLF_DBTN_UTIL_HEAP_SZ	55	Uint32	674 ページの『util_heap_sz - ユーティリティー・ヒープ・サイズ』
vendoropt	あり	なし	なし	SQLF_DBTN_VENDOROPT	829	char(242)	675 ページの『vendoropt - ベンダー・オプション』<
wlm_collect_int	あり	なし	低	SQLF_DBTN_WLM_COLLECT_INT	907	Sint32	587 ページの『wlm_collect_int - ワークロード管理収集間隔構成パラメーター』

表 67. 構成可能なデータベース構成パラメーター (続き)

パラメーター	オンラインで構成可能	自動	パフォーマンスへの影響	トークン	トークン値	データ・タイプ	追加情報

注: SQLF_DBTN_AUTONOMIC_SWITCHES のビットは、いくつかの自動保守構成パラメーターのデフォルト設定を示します。この複合パラメーターを構成する個々のビットは、以下のとおりです。
1.
`Default => Bit 1 on (xxxx xxxx xxxx xxx1): auto_maint`
`Bit 2 off (xxxx xxxx xxxx xx0x): auto_db_backup`
`Bit 3 on (xxxx xxxx xxxx x1xx): auto_tbl_maint`
`Bit 4 on (xxxx xxxx xxxx 1xxx): auto_runstats`
`Bit 5 off (xxxx xxxx xxx0 xxxx): auto_stats_prof`
`Bit 6 off (xxxx xxxx xx0x xxxx): auto_prof_upd`
`Bit 7 off (xxxx xxxx x0xx xxxx): auto_reorg`
`Bit 8 off (xxxx xxxx 0xxx xxxx): auto_storage`
`Bit 9 off (xxxx xxx0 xxxx xxxx): auto_stmt_stats`
`0 0 0 D`
 	
`Maximum => Bit 1 on (xxxx xxxx xxxx xxx1): auto_maint`
`Bit 2 off (xxxx xxxx xxxx xx1x): auto_db_backup`
`Bit 3 on (xxxx xxxx xxxx x1xx): auto_tbl_maint`
`Bit 4 on (xxxx xxxx xxxx 1xxx): auto_runstats`
`Bit 5 off (xxxx xxxx xxx1 xxxx): auto_stats_prof`
`Bit 6 off (xxxx xxxx xx1x xxxx): auto_prof_upd`
`Bit 7 off (xxxx xxxx x1xx xxxx): auto_reorg`
`Bit 8 off (xxxx xxxx 1xxx xxxx): auto_storage`
`Bit 9 off (xxxx xxx1 xxxx xxxx): auto_stmt_stats`
`0 1 F F`
2. 有効な値 (`sqlutil.h` で定義):
`SQLF_INX_REC_SYSTEM (0)`
`SQLF_INX_REC_REFERENCE (1)`
`SQLF_INX_REC_RESTART (2)`
3. 有効な値 (`sqlutil.h` で定義):
`SQLF_LOGRETAIN_NO (0)`
`SQLF_LOGRETAIN_RECOVERY (1)`
`SQLF_LOGRETAIN_CAPTURE (2)`
4. *hadr_peer_window* パラメーターをゼロ以外の時刻値に設定した場合、1 次データベースが切断されたピア状態にあるときに、スタンバイ・データベースに接続されていないにもかかわらず、スタンバイ・データベースからの確認を待機中であるために、トランザクションを続行しているかのように見える場合があります。							

表 68. 情報提供用のデータベース構成パラメーター

パラメーター	トークン	トークン値	データ・タイプ	追加情報
backup_pending	SQLF_DBTN_BACKUP_PENDING	112	Uint16	597 ページの『backup_pending - バックアップ・ペンディング標識』
codepage	SQLF_DBTN_CODEPAGE	101	Uint16	601 ページの『codepage - データベースのコード・ページ』
codeset	SQLF_DBTN_CODESET	120	char(9) [注 1]	601 ページの『codeset - データベース用コード・セット』
collate_info	SQLF_DBTN_COLLATE_INFO	44	char(260)	601 ページの『collate_info - 照合情報』
country/region	SQLF_DBTN_COUNTRY	100	Uint16	602 ページの『country/region - データベース・テリトリー・コード』
database_consistent	SQLF_DBTN_CONSISTENT	111	Uint16	602 ページの『database_consistent - データベースの整合性』
database_level	SQLF_DBTN_DATABASE_LEVEL	124	Uint16	603 ページの『database_level - データベース・リリース・レベル』
hadr_db_role	SQLF_DBTN_HADR_DB_ROLE	810	Uint32	619 ページの『hadr_db_role - HADR データベース役割』
log_retain_status	SQLF_DBTN_LOG_RETAIN_STATUS	114	Uint16	628 ページの『log_retain_status - ログ保持状況表示』

表 68. 情報提供用のデータベース構成パラメーター (続き)

パラメーター	トークン	トークン値	データ・タイプ	追加情報
loghead	SQLF_DBTN_LOGHEAD	105	char(12)	634 ページの『loghead - 最初のアクティブ・ログ・ファイル』
logpath	SQLF_DBTN_LOGPATH	103	char(242)	634 ページの『logpath - ログ・ファイルのロケーション』
multipage_alloc	SQLF_DBTN_MULTIPAGE_ALLOC	506	Uint16	647 ページの『multipage_alloc - マルチページ・ファイル割り振り使用可能』
numsegs	SQLF_DBTN_NUMSEGS	122	Uint16	656 ページの『numsegs - SMS コンテナーのデフォルト数』
pagesize	SQLF_DBTN_PAGESIZE	846	Uint32	657 ページの『pagesize - データベース・デフォルト・ページ・サイズ』
release	SQLF_DBTN_RELEASE	102	Uint16	569 ページの『release - 構成ファイル・リリース・レベル』
restore_pending	SQLF_DBTN_RESTORE_PENDING	503	Uint16	661 ページの『restore_pending - リストア・ペンディング』
restrict_access	SQLF_DBTN_RESTRICT_ACCESS	852	Sint32	661 ページの『restrict_access - データベース制限付きアクセス構成パラメーター』
rollfwd_pending	SQLF_DBTN_ROLLFWD_PENDING	113	Uint16	661 ページの『rollfwd_pending - ロールフォワード・ペンディング標識』
territory	SQLF_DBTN_TERRITORY	121	char(5) [注 2]	671 ページの『territory - データベース・テリトリー』
user_exit_status	SQLF_DBTN_USER_EXIT_STATUS	115	Uint16	673 ページの『user_exit_status - ユーザー出口状況標識』

注:

1. HP-UX、Linux および Solaris オペレーティング・システムの場合は char(17)。
2. HP-UX、Linux および Solaris オペレーティング・システムの場合は char(33)。

DB2 Administration Server (DAS) 構成パラメーターのサマリー

表 69. *DAS 構成パラメーター*

パラメーター	パラメーター・タイプ	追加情報
authentication	構成可能	675 ページの『authentication - 認証タイプ DAS』
contact_host	オンラインで構成可能	676 ページの『contact_host - 連絡先リストのロケーション』
das_codepage	オンラインで構成可能	676 ページの『das_codepage - DAS コード・ページ』
das_territory	オンラインで構成可能	677 ページの『das_territory - DAS テリトリー』
dasadm_group	構成可能	677 ページの『dasadm_group - DAS 管理者権限グループ名』
db2system	オンラインで構成可能	678 ページの『db2system - DB2 サーバー・システムの名前』
discover	オンラインで構成可能	678 ページの『discover - DAS ディスカバリー・モード』
exec_exp_task	構成可能	679 ページの『exec_exp_task - 有効期限切れタスクの実行』
jdk_64_path	オンラインで構成可能	623 ページの『jdk_64_path - 64 ビット Software Developer's Kit for Java インストール・パス DAS』
jdk_path	オンラインで構成可能	680 ページの『jdk_path - Software Developer's Kit for Java インストール・パス DAS』
sched_enable	構成可能	680 ページの『sched_enable - スケジューラー・モード』
sched_userid	通知	681 ページの『sched_userid - スケジューラー・ユーザー ID』
smtp_server	オンラインで構成可能	681 ページの『smtp_server - SMTP サーバー』
toolscat_db	構成可能	681 ページの『toolscat_db - ツール・カタログ・データベース』
toolscat_inst	構成可能	682 ページの『toolscat_inst - ツール・カタログ・データベース・インスタンス』
toolscat_schema	構成可能	682 ページの『toolscat_schema - ツール・カタログ・データベース・スキーマ』

構成パラメーターのセクション見出し

各構成パラメーターの説明には、該当する場合、以下のセクション見出しのいくつかまたはすべてが含まれます。場合によっては、セクション見出しが一方しか示されないこともあります。例えば、[範囲] が指定されている場合は、有効な値は不要です。多くの場合、これらの見出しは説明を要しません。

表 70.

セクション見出し	説明と可能な値
構成タイプ	可能な値は以下のとおりです。 • データベース・マネージャー • データベース • DB2 Administration Server
適用	該当する場合、構成パラメーターが適用されるデータ・サーバー・タイプがリストされます。可能な値は以下のとおりです。 • クライアント • ローカルおよびリモート・クライアントを持つデータベース・サーバー • ローカル・クライアントを持つデータベース・サーバー • DB2 Administration Server • OLAP 関数 • ローカルとリモート・クライアントを持つパーティション・データベース・サーバー • フェデレーションが有効化された場合、ローカルとリモート・クライアントを持つパーティション・データベース・サーバー。 • ローカル・クライアントを持つサテライト・データベース・サーバー
パラメーター・タイプ	可能な値は以下のとおりです。 • 構成可能 (変更を有効にするには、データベース・マネージャーを再始動する必要がある) • オンラインで構成可能 (データベース・マネージャーを再始動せずに、オンラインで動的に更新することが可能) • 通知 (値は通知のためのみで、更新はできない)
デフォルト [範囲]	該当する場合、NULL 値または自動設定を含むデフォルト値と可能な範囲がリストされます。範囲がプラットフォームごとに異なる場合は、値はプラットフォームまたはプラットフォーム・タイプ (例えば、32 ビット・プラットフォームや 64 ビット・プラットフォーム) ごとにリストされます。多くの場合、デフォルト値は範囲の一部としては示されません。
単位	該当する場合、計測単位がリストされます。可能な値は以下のとおりです。 • バイト • カウンター • MB/秒 • ミリ秒 • 分 • ページ (4 KB) • パーセント • 秒
有効な値	該当する場合、有効な値がリストされます。この見出しとデフォルトの [範囲] 見出しは、いずれか一方しか示されません。
例	該当する場合、例がリストされます。
伝搬クラス	該当する場合、可能な値は次のとおりです。 • 即時 • ステートメント境界
割り振られるタイミング	該当する場合、構成パラメーターがデータベース・マネージャーによって割り振られるタイミングが示されます。

表 70. (続き)

セクション見出し	説明と可能な値
解放されるタイミング	該当する場合、構成パラメーターがデータベース・マネージャーによって解放されるタイミングが示されます。
制約事項	該当する場合、構成パラメーターに当てはまるすべての制約事項がリストされます。
制限	該当する場合、構成パラメーターに当てはまるすべての制限がリストされます。
推奨事項	該当する場合、構成パラメーターに当てはまるすべての推奨事項がリストされます。
使用上の注意	該当する場合、構成パラメーターに当てはまるすべての使用上の注意がリストされます。

エージェントの数に影響を与える構成パラメーター

データベース・エージェントとその管理方法に関連するデータベース・マネージャー構成パラメーターは多数あります。

以下のデータベース・マネージャー構成パラメーターは、作成されるデータベース・エージェントの数、およびそれらが管理される方法を決定します。

- 「エージェント・プール・サイズ」(*num_poolagents*): システム内で使用可能な状態に保たれる、プール対象のアイドル・エージェントの合計数。このパラメーターのデフォルト値は 100, AUTOMATIC です。
- 「プール内エージェントの初期数」(*num_initagents*): データベース・マネージャーの開始時に、この値に基づいて作業エージェントのプールが作成されます。これによって、初期の照会のパフォーマンスが速くなります。作業エージェントはすべてアイドル・エージェントとして開始します。
- 「最大接続数」(*max_connections*): 各データベース・パーティションごとにデータベース・マネージャー・システムに許容される接続の最大数を指定します。
- 「コーディネーター・エージェントの最大数」(*max_coordagents*): パーティション・データベース環境、および**接続コンセントレーター**使用可能時に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では、この値によってコーディネーター・エージェント数が制限されます。

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。たとえば、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・プールに取り出す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移行した時点で、自動的に適切な値に設定されます。このパラメーターは、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。たとえば、SQLCODE +437 (SQLSTATE 01602) により、ステートメントのコンパイルに使った最適化の量が、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。

max_coordagents および max_connections を構成する際の制限と動作

バージョン 9.5 では、*max_coordagents* および *max_connections* パラメーターのデフォルトは AUTOMATIC で、*max_coordagents* が 200 に設定され、*max_connections* は -1 に設定されます (つまり、*max_coordagents* の値に設定されます)。これらの設定値により、コンセントレーターは OFF に設定されます。

max_coordagents または *max_connections* をオンラインで構成する際、認識しておくべき制約事項と動作がいくつかあります。

- *max_coordagents* の値が大きくなると、設定は直ちに有効になり、新しい要求は新しいコーディネーター・エージェントの作成を許可されます。値が小さくなると、コーディネーター・エージェントの数は直ちに減らされません。コーディネーター・エージェントの数は増えなくなり、既存のコーディネーター・エージェントは現在の一連の作業を終えると、コーディネーター・エージェントの全体数を減らすために終了する可能性があります。コーディネーター・エージェントを必要とする作業に対する新しい要求は、コーディネーター・エージェントの総数が新しい値以下に減って、コーディネーター・エージェントが解放されるまでは処理されません。
- *max_connections* の値が大きくなると、設定は直ちに有効になり、このパラメーターによって以前ブロックされていた新しい接続は許可されます。この値が小さくなると、データベース・マネージャーは既存の接続を積極的に終了せず、既存の接続が終了して、値が新規の最大値を下回るまで新しい接続は許可されません。
- *max_connections* が -1 (デフォルト) に設定されると、許可される接続の最大数は *max_coordagents* と同じになり、*max_coordagents* がオフラインまたはオンラインで更新されると許可される接続の最大数も同様に更新されます。

max_coordagents または *max_connections* の値をオンラインで変更する場合、接続コンセントレーターが OFF のときに ON にしたり、ON のときに OFF にするような切り換えが発生するような変更は加えることができません。例えば、DB2START

時に *max_coordagents* が *max_connections* 未満 (コンセントレーターは ON) の場合、これら 2 つのパラメーターにオンラインで行うすべての更新で、*max_coordagents* < *max_connections* という関係が維持されている必要があります。同様に、DB2START 時に *max_coordagents* が *max_connections* 以上 (コンセントレーターは OFF) の場合、オンラインで行うすべての更新で、この関係が維持されている必要があります。

この種の更新をオンラインで実行するとき、データベース・マネージャーは操作を失敗させず、更新を遅延させます。 IMMEDIATE を指定してデータベース・マネージャー構成パラメーターを更新しようとしてもできない場合と同様に、警告の SQL1362W メッセージが戻されます。

max_coordagents または *max_connections* を AUTOMATIC に設定すると、以下の動作が予想されます。

- どちらのパラメーターも、開始値および AUTOMATIC 設定を使用して構成できます。例えば、次のコマンドにより、*max_coordagents* パラメーターに値 200 および AUTOMATIC が関連付けられます。

  ```
  UPDATE DBM CONFIG USING max_coordagents 200 AUTOMATIC
  ```

  これらのパラメーターには必ず 1 つの値が関連付けられます。これはデフォルトとして設定された値か、自分で指定した値のいずれかです。いずれかのパラメーターを更新するときに AUTOMATIC のみを指定する (つまり値を指定しない) 場合、このパラメーターに関連付けられた値があるなら、その値がそのまま使用されます。 AUTOMATIC 設定だけが影響を受けます。

  注: コンセントレーターが ON の場合、これら 2 つの構成パラメーターに割り当てられた値は、パラメーターが AUTOMATIC に設定されている場合でも重要です。

- 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、接続とコーディネーター・エージェントの数がワークロードに見合うように、必要に応じて大きくなることを許可します。ただし、以下の注意点があります。

  1. コンセントレーターが OFF の場合、データベース・マネージャーは 1 対 1 の比率を維持します。すべての接続について、コーディネーター・エージェントは *1* つ のみとなります。
  2. コンセントレーターが ON の場合、データベース・マネージャーは接続に対するコーディネーター・エージェントの比率を、パラメーターの値によって設定された比率に維持しようとします。

     注:

     – 比率を維持するために使用される方法は干渉的にならないように設計されており、比率を完全に維持することを保証するものではありません。このシナリオでは、新しい接続は必ず許可されます。ただし、使用可能なコーディネーター・エージェントを得るために待機しなければならないことがあります。新しいコーディネーター・エージェントは比率を維持するために、必要に応じて作成されます。接続が終了すると、データベース・マネージャーはコーディネーター・エージェントも終了させて、比率を維持する場合があります。

– データベース・マネージャーは、ユーザーが設定した比率を下げることはしません。設定した *max_coordagents* と *max_connections* の初期値は下限とみなされます。

- これら両方のパラメーターの現行値および遅延された値は、CLP や API など、さまざまな方法で表示できます。表示される値は、必ずユーザーが設定した値です。例えば、次のコマンドが発行されて、インスタンス上で作業を実行する 30 件の同時接続が開始された場合でも、*max_connections* および *max_coordagents* に表示される値は 20、AUTOMATIC のままです。

```
UPDATE DBM CFG USING max_connections 20 AUTOMATIC,
  max_coordagents 20 AUTOMATIC
```

  現在モニター・エレメントを実行している接続とコーディネーター・エージェントの実数を判別するために、ヘルス・モニターを使用することもできます。

- *max_connections* が AUTOMATIC に設定されていて値が *max_coordagents* よりも大きく (このためコンセントレーターが ON になっている)、*max_coordagents* が AUTOMATIC に設定されていない場合、データベース・マネージャーは許可する接続の数を無制限とし、これらの接続は限られた数のコーディネーター・エージェントを使用することになります。

  注: 接続は、使用可能なコーディネーター・エージェントを待機する場合があります。

max_coordagents および max_connections 構成パラメーターに AUTOMATIC オプションを使用することは、次の 2 つのシナリオでのみ有効です。 :

1. どちらのパラメーターも AUTOMATIC に設定されている。
2. *max_connections* が AUTOMATIC に設定されており、*max_coordagents* が AUTOMATIC に設定されていない状態でコンセントレーターが使用可能である。

これらのパラメーターに AUTOMATIC を使用したそれ以外の構成は、すべてブロックされて、これら 2 つのパラメーターで AUTOMATIC が有効な設定を説明した理由コードとともに、SQL6112N が返されます。

データベース・マネージャー構成パラメーター

agent_stack_sz - エージェント・スタック・サイズ

このパラメーターは、DB2 によって各エージェントに割り振られる仮想メモリーを決定します。

構成タイプ

データベース・マネージャー

適用

- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- ローカル・クライアントを持つデータベース・サーバー
- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ
構成可能

デフォルト [範囲]

Linux (32 ビット)
256 [16 – 1024]

Linux (64 ビット) および UNIX
1024 [256 – 32768]

Windows
16 [8 – 1000]

単位 ページ (4 KB)

割り振られるタイミング
エージェントが、アプリケーションの作業を行うように初期化されるとき

解放されるタイミング
エージェントがアプリケーションの作業を完了するとき

このパラメーターを使うと、特定のアプリケーションでのサーバーのメモリー使用率を最適化できます。単純照会に使用されるスペースと比較して、照会がより複雑になればより多くのスタック・スペースが使用されます。

このパラメーターは、Windows 環境でそれぞれのエージェントごとに初期コミット済みスタック・サイズを設定する場合に使用します。デフォルトでは、それぞれのエージェント・スタックは、最大でデフォルトの予約スタック・サイズである 256 KB (64 4 KB ページ) まで増やせます。ほとんどのデータベース操作には、この上限で十分です。 UNIX と Linux の場合、*agent_stack_sz* は、2 の累乗の次の最も近い大きな値に切り上げられます。 UNIX のデフォルトの設定値は、大部分のワークロードに十分対応できるものにしてください。

ただし、準備中の SQL または XQuery ステートメントが大きい場合は、エージェントがスタック・スペースを使い尽くす可能性があり、システムがオーバーフロー例外 (0xC00000FD) を生成することになります。こうなったときは、エラーがリカバリー不能のため、サーバーはシャットダウンします。

注: バージョン 9.5 以降では、sqlcode-973 がスタック・オーバーフロー例外の代わりに返されます。

エージェント・スタック・サイズは、*agent_stack_sz* をデフォルトの予約スタック・サイズ 64 ページよりも大きい値に設定することで、大きくすることができます。*agent_stack_sz* の値は、デフォルトの予約スタック・サイズよりも大きくなると、Windows オペレーティング・システムによって、その値に最も近い 1 MB の倍数に丸められることに注意してください。したがって、エージェント・スタック・サイズを 128 4 KB ページに設定すると、実際にはそれぞれのエージェントごとに 1 MB が予約されることになります。*agent_stack_sz* の値をデフォルトの予約スタック・サイズよりも小さい値に設定した場合は、必要なら、スタックが最大でデフォルトの予約スタック・サイズまで大きくなるため、デフォルトの予約スタック・サイズが最大限度に影響することはありません。この場合は、*agent_stack_sz* の値は、エージェントが作成されるときのスタック用初期コミット済みメモリーになります。

デフォルトの予約スタック・サイズは、db2syscs.exe ファイルに関するヘッダー情報を変更する db2hdr ユーティリティーを使用して変更できます。デフォルトの予約スタック・サイズを変更すると、*agent_stack_sz* の変更では、エージェントのスタック・サイズが影響を受けるだけであるのに対して、すべてのスレッドに影響が生じます。db2hdr ユーティリティーを使用してデフォルトのスタック・サイズを変更すると、より優れた細分性が得られ、そのためにスタック・サイズを最小所要スタック・サイズに設定できるという利点があります。ただし、db2syscs.exe に対する変更を有効にするためには、DB2 をいったん停止してから、再始動する必要があります。

推奨: 32 ビット環境で大規模または複雑な XML データを処理する場合、*agent_stack_sz* を少なくとも 256 4 KB ページに更新する必要があります。非常に複雑な XML スキーマは、スキーマ登録時または XML 文書の妥当性検査時にスタック・オーバーフロー例外を避けるために、*agent_stack_sz* をほぼ限度いっぱいに設定することが必要になる場合があります。

環境が以下のいずれかにあてはまる場合は、他のクライアントがより多くのアドレス・スペースを使用できるようにするために、スタック・サイズを減らすことができます。

- 複雑な照会を持たない、単純なアプリケーション (たとえば単純な OLTP) だけがある場合
- 相対的に多くの (たとえば 100 より多い) 並行クライアントを必要とする場合

Windows の場合、エージェント・スタック・サイズおよび並行クライアントの数は、逆比例の関係にあります。大きなスタック・サイズは、実行できる並行クライアントの潜在数を減らします。こうなるのは、Windows プラットフォームではアドレス・スペースが限定されているためです。

agentpri - エージェントの優先順位

このパラメーターはバージョン 9.5 では推奨されませんが、バージョン 9.5 より前のデータ・サーバーおよびクライアントでは引き続き使用されています。この構成パラメーターに指定されるすべての値は、前の各バージョンで動作したのとまったく同様に引き続き動作し、このパラメーターは完全にサポートされ続けます。このパラメーターがワークロード管理 (WLM) に使用される場合は、WLM サービス・クラス・エージェントの優先順位は無視されます。

注: 以下の情報は、バージョン 9.5 より前のデータ・サーバーおよびクライアントだけに当てはまります。

このパラメーターは、オペレーティング・システム・スケジューラーによって、すべてのエージェントと、データベース・マネージャー・インスタンス・プロセスおよびスレッドの両方に与えられる優先順位をコントロールします。この優先順位は、データベース・マネージャー・プロセス、エージェント、他のプロセスに関連するスレッド、およびマシンで実行中のスレッドに対する CPU 時間の割り当て方法を決定します。

構成タイプ

データベース・マネージャー

適用

- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- ローカル・クライアントを持つデータベース・サーバー
- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ
構成可能

デフォルト [範囲]

AIX -1 (system) [41 - 125]

他の UNIX
-1 (system) [41 - 128]

Windows
-1 (system) [0 - 6]

Solaris -1 (system) [0 - 59]

パラメーターを -1 または system に設定すると、特殊なアクションは取られず、データベース・マネージャーは、オペレーティング・システムがすべてのプロセスとスレッドをスケジュールする通常の方法でスケジュールされます。パラメーターを -1 または system 以外の値に設定すると、データベース・マネージャーは、そのパラメーターの値に設定される静的優先順位で、そのプロセスとスレッドを作成します。したがって、このパラメーターを使用すると、データベース・マネージャー・プロセスおよびスレッド (パーティション・データベース環境では、これにコーディネーター・エージェントとサブエージェント、並列システム・コントローラー、および FCM デーモンも含まれます) をマシンで実行する優先順位をコントロールできます。

このパラメーターを使用して、データベース・マネージャーのスループットを増やすことができます。このパラメーターの設定値は、データベース・マネージャーが稼働しているオペレーティング・システムによって異なります。例えば、Linux または UNIX 環境では、数値が低いほど優先順位が高くなります。パラメーターを 41 から 125 の間の値に設定すると、データベース・マネージャーはそのパラメーターの値に設定される UNIX の静的優先順位で、そのエージェントを作成します。このことは Linux および UNIX の環境では重要です。数値を小さくするとデータベース・マネージャーの優先順位が高くなりますが、他のプロセス (アプリケーションとユーザーの両方) で十分な CPU 時間を獲得できないため遅延が起きることがあるからです。このパラメーターの設定値と、マシン上で予期される他の活動との平衡を取る必要があります。

制約事項:

- エージェントの優先順位フィーチャーが (DB2 サービス・クラスのオペレーティング・システム (例えば、AIX) との関連付けによって) サービス・クラスに使用可能にされると、*agentpri* データベース構成パラメーターはオーバーライドされます。サービス・クラスが使用不可にされると、どのステートメントも (別の ALTER サービス・クラス・ステートメントでさえも) 処理することはできません。

- AIX WLM に対するアプリケーション・タグの最大長は 30 文字です。 30 文字より長い Outbound Correlator (アウトバウンド相関関係子) を設定すると、AIX WLM によって拒否されます。
- このパラメーターを Linux および UNIX プラットフォーム上のデフォルト以外の値に設定すると、ガバナーを使用してエージェントの優先順位を変更することができなくなります。
- Solaris オペレーティング・システムでは、デフォルト値 (-1) を変更しないでください。デフォルト値を変更すると、DB2 プロセスの優先順位がリアルタイムに設定されるため、システムの利用可能なすべてのリソースが占有される可能性があります。

推奨: 最初は、デフォルト値を使用してください。デフォルト値では、他のユーザー/アプリケーションへの応答時間とデータベース・マネージャー・スループットの間の適切な折衷案を提供します。

データベースのパフォーマンスが問題になる場合は、ベンチマークの手法を使って、このパラメーターの最適な設定値を決定できます。データベース・マネージャーの優先順位を上げるときは、注意する必要があります。特に CPU 使用率が非常に高いときは、他のユーザー・プロセスのパフォーマンスが著しく低下する可能性があるからです。データベース・マネージャー・プロセスおよびスレッドの優先順位を上げると、有効なパフォーマンスの効果を得られる可能性があります。

aslheapsz - アプリケーション・サポート層ヒープ・サイズ

アプリケーション・サポート層ヒープは、ローカル・アプリケーションとその関連エージェントの間の通信バッファーを示します。このバッファーはデータベース・マネージャー・エージェントが開始するたびに共用メモリーとして割り振られます。

構成タイプ
データベース・マネージャー

適用
- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- ローカル・クライアントを持つデータベース・サーバー
- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ
構成可能

デフォルト [範囲]
15 [1 - 524 288]

単位 ページ (4 KB)

割り振られるタイミング
データベース・マネージャー・エージェント・プロセスがローカル・アプリケーションで開始するとき

解放されるタイミング
データベース・マネージャー・エージェント・プロセスが終了したとき

データベース・マネージャーへの要求、またはそれに関連する応答がバッファーに適合しない場合は、バッファーが 2 つ以上の送受信の組み合わせに分割されます。このバッファーのサイズは、単一の送受信の組み合わせを使用する大多数の要求を処理するように、設定する必要があります。要求のサイズは、次のものを保存するのに必要なストレージに基づいて決まります。

- 入力 SQLDA
- SQLVAR 内のすべての関連データ
- 出力 SQLDA
- 通常は 250 バイトを超えないその他のフィールド

このパラメーターは、この通信バッファーだけでなく、それ以外に次の 2 つの目的でも使用されます。

- ブロック・カーソルがオープンされているとき、入出力ブロック・サイズを決定するのに使用されます。ブロック・カーソル用のこのメモリーは、アプリケーションの専用アドレス・スペースから割り振られるので、それぞれのアプリケーション・プログラムごとに割り振る専用メモリーの最適量を決定する必要があります。データ・サーバー・ランタイム・クライアントがアプリケーションの専用メモリーからブロック・カーソル用のスペースを割り振れない場合は、非ブロッキング・カーソルがオープンされます。
- これは、エージェント・プロセスと db2fmp プロセスの間の通信のサイズを決定するのに使用されます (db2fmp プロセスは、ユーザー定義関数でも fenced ストアード・プロシージャーでも構いません)。システム上でアクティブな各 db2fmp プロセスまたはスレッドのために、共用メモリーから何バイトかのメモリーが割り振られます。

ローカル・アプリケーションから送信されたデータは、データベース・マネージャーによって、照会ヒープから割り振られた連続したメモリーのセットに受信されます。 *aslheapsz* パラメーターは、照会ヒープの初期サイズ (ローカルおよびリモート・クライアントの両方について) の決定に使用されます。照会ヒープの最大サイズは、*query_heap_sz* パラメーターによって定義されます。

推奨: アプリケーションの要求が通常は少なく、そのアプリケーションが、メモリーに制約があるシステムで実行されている場合は、このパラメーターの値を減らすことが必要になる可能性があります。通常は非常に大きい照会をしていて複数の送受信要求を必要とし、またシステムがメモリーによる制約を受けていない場合は、このパラメーターの値を増やすことが必要になる可能性があります。

次の公式を使用して、*aslheapsz* の最小ページ数を計算します。

```
aslheapsz >= ( sizeof(入力 SQLDA)
             + sizeof(各入力 SQLVAR)
             + sizeof(出力 SQLDA)
             + 250 ) / 4096
```

ただし、sizeof(x) は、特定の入力値または出力値のページ数を計算する x のサイズ (バイト数) です。

ブロック・カーソルの数および潜在的なサイズに対するこのパラメーターの影響についても考慮する必要があります。転送される行の数が多い、またはそのサイズが大きい場合 (例えば、データの量が 4096 バイトより大の場合) は、行ブロックが大

きければ、パフォーマンスの向上がもたらされる可能性もあります。ただし、レコード・ブロックを大きくすると、それぞれの接続ごとの作業セット・メモリーのサイズも増大するので、トレードオフが必要になります。

また、レコード・ブロックが大きくなれば、フェッチ要求も実際にアプリケーションで必要とされる数よりも多くなる可能性もあります。フェッチ要求の数は、アプリケーションの SELECT ステートメントの OPTIMIZE FOR 節を使ってコントロールできます。

audit_buf_sz - 監査バッファー・サイズ

このパラメーターは、データベースを監査するときに使用されるバッファーのサイズを指定します。

構成タイプ
データベース・マネージャー

適用

- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- ローカル・クライアントを持つデータベース・サーバー
- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ
構成可能

デフォルト [範囲]
0 [0 - 65 000]

単位 ページ (4 KB)

割り振られるタイミング
DB2 の開始時

解放されるタイミング
DB2 の終了時

このパラメーターのデフォルト値はゼロ (0) です。 値がゼロ (0) である場合、監査バッファーは使用されません。値がゼロ (0) よりも大きい場合、監査機能によって監査レコードが生成されるときに、そのレコードが置かれる監査バッファー用にスペースが割り振られます。 4 KB ページの倍数の値が監査バッファーに割り振られたスペースの量です。監査バッファーは動的には割り振られません。このパラメーターに新しい値を指定してそれを有効にするには、 DB2 をいったん終了してから再始動する必要があります。

このパラメーターをデフォルト値からゼロ (0) よりも大きい値に変更すると、監査機能は、監査レコードを生成するステートメントの実行とは非同期にレコードをディスクに書き込みます。これは、パラメーター値をゼロ (0) のままにしておくよりも DB2 のパフォーマンスを向上させます。値をゼロ (0) にすると、監査機能は、監査レコードを生成するステートメントの実行と同期して (同時に) レコードをディスクに書き込みます。監査時の同期操作は、DB2 で実行しているアプリケーションのパフォーマンスを低下させます。

authentication - 認証タイプ

このパラメーターは、ユーザーの認証が行われる方法とその場所を指定します。

構成タイプ

データベース・マネージャー

適用

- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- クライアント
- ローカル・クライアントを持つデータベース・サーバー
- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ

構成可能

デフォルト [範囲]

SERVER [CLIENT; SERVER; SERVER_ENCRYPT; DATA_ENCRYPT; DATA_ENCRYPT_CMP; KERBEROS; KRB_SERVER_ENCRYPT; GSSPLUGIN; GSS_SERVER_ENCRYPT]

認証が SERVER の場合は、サーバー上で認証を行うために、ユーザー ID とパスワードがクライアントからサーバーに送信されます。値 SERVER_ENCRYPT は、ネットワークを介して送信されるパスワードが暗号化されることを除き、SERVER と同じように機能します。

DATA_ENCRYPT の値は、暗号化された SERVER 認証スキーマおよびユーザー・データの暗号化をサーバーが受け入れることを意味します。認証は SERVER_ENCRYPT の場合と全く同様に機能します。

この認証タイプを使用するとき、以下のユーザー・データが暗号化されます。

- SQL ステートメント
- SQL プログラム変数データ
- SQL ステートメントを処理するサーバーの出力データ (データについての説明を含む)
- 照会から生じる応答セット・データの一部またはすべて
- ラージ・オブジェクト (LOB) ストリーム
- SQLDA 記述子

DATA_ENCRYPT_CMP の値は、暗号化された SERVER 認証スキーマおよびユーザー・データの暗号化をサーバーが受け入れることを意味します。さらに、この認証タイプでは DATA_ENCRYPT 認証タイプをサポートしない前の製品との互換性があります。これらの製品は、SERVER_ENCRYPT 認証タイプを使って、暗号化ユーザー・データがない状態での接続を許可されます。新しい認証タイプをサポートしている製品は、これを使用する必要があります。この認証タイプは、サーバーのデータベース・マネージャー構成ファイル内のみで有効であり、CATALOG DATABASE コマンドで使用するときには無効です。

注: (「標準への準拠」のトピックに定義されている) 標準に準拠した構成では、サポートされる唯一の値は SERVER です。

値 CLIENT は、すべての認証がクライアントで行われることを示します。この場合はサーバーで認証を行う必要はありません。

KERBEROS の値は、認証が、Kerberos セキュリティー・プロトコルを使用して Kerberos サーバー上で行われることを意味します。 Kerberos セキュリティー・システムをサポートするサーバーおよびクライアントの認証タイプが KRB_SERVER_ENCRYPT の場合、有効なシステム認証タイプは KERBEROS です。クライアントが Kerberos セキュリティー・システムをサポートしない場合のシステム認証タイプは、事実上 SERVER_ENCRYPT に相当します。

値 GSSPLUGIN は、外部の GSSAPI ベースのセキュリティー・メカニズムを使用して認証が実行されることを意味します。 GSSPLUGIN セキュリティー・メカニズムをサポートするサーバーおよびクライアントの認証タイプが GSS_SERVER_ENCRYPT の場合、有効なシステム認証タイプは GSSPLUGIN です (つまり、クライアントがサーバーのプラグインのいずれかをサポートしている場合)。クライアントが GSSPLUGIN セキュリティー・メカニズムをサポートしない場合のシステム認証タイプは、事実上 SERVER_ENCRYPT に相当します。

推奨: 一般に、ローカル・クライアントにはデフォルト値 (SERVER) が適切です。データベース・サーバーにリモート・クライアントが接続している場合、ユーザーのパスワードを保護するために SERVER_ENCRYPT という値が推奨されます。

catalog_noauth - 権限なしで許可されるカタログ

このパラメーターは、ユーザーがデータベースとノード、または DCS と ODBC ディレクトリーを SYSADM 権限なしでカタログおよびアンカタログできるようにするかを指定します。

構成タイプ
データベース・マネージャー

適用
- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- クライアント
- ローカル・クライアントを持つデータベース・サーバー
- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ
オンラインで構成可能

伝搬クラス
即時

デフォルト [範囲]

ローカルおよびリモート・クライアントを持つデータベース・サーバー
NO [NO (0) — YES (1)]

クライアント; ローカル・クライアントを持つデータベース・サーバー
YES [NO (0) — YES (1)]

パラメーターのデフォルト値 (0) は、SYSADM 権限が必要なことを示します。このパラメーターが 1 (yes) に設定された場合、SYSADM 権限は必要なくなります。

clnt_krb_plugin - クライアント Kerberos プラグイン

このパラメーターでは、クライアント側の認証とローカル許可に使用するデフォルト Kerberos プラグイン・ライブラリーの名前を指定します。

構成タイプ
データベース・マネージャー

適用

- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- クライアント
- ローカル・クライアントを持つデータベース・サーバー
- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ
構成可能

デフォルト [範囲]
Null または IBMkrb5 [任意の有効ストリング]

デフォルトの値は、Linux および UNIX システムでは NULL で、Windows オペレーティング・システムでは `IBMkrb5` です。このプラグインは、クライアントが KERBEROS 認証を使用して認証されるか、またはローカル許可が実行され、DBM CFG の認証タイプが KERBEROS のときに使用されます。

clnt_pw_plugin - クライアント・ユーザー ID パスワード・プラグイン

このパラメーターでは、クライアント側の認証とローカル許可に使用するユーザー ID-パスワード・プラグイン・ライブラリーの名前を指定します。

構成タイプ
データベース・マネージャー

適用

- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- クライアント
- ローカル・クライアントを持つデータベース・サーバー
- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ
構成可能

デフォルト [範囲]
Null [任意の有効ストリング]

デフォルトの値は NULL で、DB2 提供のユーザー ID-パスワード・プラグイン・ライブラリーが使用されます。このプラグインは、クライアントが CLIENT 認証を使用して認証されるか、またはローカル許可が実行され、DBM CFG の認証タイプが CLIENT、SERVER、SERVER_ENCRYPT、または DATA_ENCRYPT のときに使用されます。非 root インストールについては、DB2 のユーザー ID およびパスワード・プラグイン・ライブラリーが使用される場合は、DB2 製品を使用する前に、db2rfe コマンドを実行する必要があります。

cluster_mgr - クラスター・マネージャー名

このパラメーターを使用すれば、データベース・マネージャーによる指定したクラスター・マネージャーに対するクラスターの構成変更の増分についての通信が可能になります。

構成タイプ
データベース・マネージャー

適用

- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- ローカル・クライアントを持つデータベース・サーバー
- ローカルとリモート・クライアントを持つ複数パーティション・データベース・サーバー

パラメーター・タイプ
通知

デフォルト
デフォルトなし

有効な値

- TSA

このパラメーターは、DB2 ハイ・アベイラビリティー・インスタンス構成ユーティリティー (db2haicu) を使用した高可用性クラスター構成時に設定します。

comm_bandwidth - 通信スピード

このパラメーターは、データベース・パーティション・サーバー間の帯域幅を指定することにより、照会オプティマイザーがアクセス・パスを決定するのに役立ちます。

構成タイプ
データベース・マネージャー

適用 ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ
オンラインで構成可能

伝搬クラス
ステートメント境界

デフォルト [範囲]
-1 [.1 - 100 000]

値が -1 の場合は、パラメーター値がデフォルトにリセットされます。デフォルト値は、基礎となる通信アダプターの速度に基づいて計算されます。ギガビット・イーサネットを使用するシステムでは、値 100 が期待されます。

単位 MB/秒

通信スピード用に計算された値 (MB/秒単位) は、照会オプティマイザーによって、パーティション・データベース・システムのデータベース・パーティション・サーバー間の特定の操作の実行のコストを見積もるのに使用されます。オプティマイザーがクライアントとサーバーの間の通信のコストをモデル化することはないので、このパラメーターには、データベース・パーティション・サーバー間の公称帯域幅のみが反映される必要があります。

この値を明示的に設定すると、テスト・システム上に実稼働環境をモデル化したり、ハードウェアのアップグレードの影響を調査したりすることができます。

推奨: 異なる環境をモデル化する場合にのみ、このパラメーターを調整する必要があります。

通信スピードは、オプティマイザーがアクセス・パスの決定にあたって使用します。このパラメーターを変更した場合は、アプリケーションの再バインド (REBIND PACKAGE コマンドを使用) を考慮してください。

conn_elapse - 接続経過時間

このパラメーターは、TCP/IP 接続が 2 つのデータベース・パーティション・サーバー間に設定される必要がある時間 (秒数) を指定します。

構成タイプ
データベース・マネージャー

適用 ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ
オンラインで構成可能

伝搬クラス
即時

デフォルト [範囲]
10 [0–100]

単位 秒

このパラメーターによって指定された時間内に接続の試みが成功すれば、通信が確立されます。接続に失敗した場合は、通信を確立する試みがもう一度行われます。*max_connretries* パラメーターによって指定された回数だけ接続が試みられ、しかもそのすべてがタイムアウトになった場合は、エラーになります。

cpuspeed - CPU 速度

このパラメーターはデータベースがインストールされたマシンの CPU 速度を反映します。

構成タイプ
データベース・マネージャー

適用
- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- ローカル・クライアントを持つデータベース・サーバー
- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ
オンラインで構成可能

伝搬クラス
ステートメント境界

デフォルト [範囲]
-1 [$1x10^{-10}$ — 1]。値を -1 にすると、パラメーター値は、測定プログラムの実行に基づいてリセットされます。

単位 ミリ秒

このプログラムは、IBM RS/6000 モデル 530H 用のデータがファイル内にないか、またはご使用のマシン用のデータがファイル内にない場合で、ベンチマーク結果が利用できないときに実行されます。

この値を明示的に設定すると、テスト・システム上に実稼働環境をモデル化したり、ハードウェアのアップグレードの影響を調査したりすることができます。この値を -1 に設定すると、*cpuspeed* が再計算されます。

推奨: 異なる環境をモデル化する場合にのみ、このパラメーターを調整する必要があります。

CPU 速度は、オプティマイザーが、アクセス・パスの決定で使用します。このパラメーターを変更した場合は、アプリケーションの再バインド (REBIND PACKAGE コマンドを使用) を考慮してください。

dft_account_str - デフォルト・チャージバック・アカウント

このパラメーターは、アカウント ID のデフォルトの接尾部の役割を果たします。

構成タイプ
データベース・マネージャー

適用
- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- クライアント
- ローカル・クライアントを持つデータベース・サーバー
- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ
オンラインで構成可能

伝搬クラス
即時

デフォルト [範囲]
Null [任意の有効ストリング]

アプリケーション接続要求のそれぞれについて、DB2 Connect が生成した接頭部とユーザーが指定した接尾部からなるアカウント ID が、アプリケーション・リクエスターから DRDA アプリケーション・サーバーへ送信されます。このアカウント情報は、リソースの使用状況を各ユーザー・アクセスに関連付けるためのメカニズムとして、システム管理者に提供されます。

注: このパラメーターは DB2 Connect にのみ適用できます。

接尾部は、アプリケーション・プログラムが呼び出す `sqlesact()` API、またはユーザー設定の環境変数 DB2ACCOUNT によって与えられます。接尾部が API と環境変数のどちらからも与えられない場合、DB2 Connect はデフォルトの接尾部としてこのパラメーターの値を使用します。このパラメーターが特に有用なのは、会計情報ストリングを DB2 Connect に転送できる機能のない以前のデータベース・クライアント (バージョン 2 よりも前のクライアント) の場合です。

推奨: 以下を使用して、この会計情報ストリングを設定してください。

- 英字 (A から Z)
- 数字 (0 から 9)
- 下線 (_)

dft_monswitches - デフォルトのデータベース・システム・モニター・スイッチ

このパラメーターは、パラメーターの各ビットで内部的に表されるいくつかのスイッチを設定できます。

構成タイプ
データベース・マネージャー

適用

- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- ローカル・クライアントを持つデータベース・サーバー
- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ
オンラインで構成可能

伝搬クラス
即時

注: dft_mon_xxxx スイッチの設定値を変更する前にインスタンスに明示的に ATTACH した場合、この変更は直ちに有効になります。そうでない場合は、設定は次回インスタンスが再始動したときに有効になります。

デフォルト
デフォルトでオンの dft_mon_timestamp を除いて、すべてのスイッチがオフ

このパラメーターは、以下のパラメーターを設定することによって、それぞれのスイッチを別個に独立して更新できる点で、ユニークなパラメーターです。

dft_mon_uow
スナップショット・モニターの作業単位 (UOW) スイッチのデフォルト値

dft_mon_stmt
スナップショット・モニターのステートメント・スイッチのデフォルト値

dft_mon_table
スナップショット・モニターの表スイッチのデフォルト値

dft_mon_bufpool
スナップショット・モニターのバッファー・プール・スイッチのデフォルト値

dft_mon_lock
スナップショット・モニターのロック・スイッチのデフォルト値

dft_mon_sort
スナップショット・モニターのソート・スイッチのデフォルト値

dft_mon_timestamp
スナップショット・モニターのタイム・スタンプ・スイッチのデフォルト値

推奨: どのスイッチも (ただし、dft_mon_timestamp を除く)、それを ON にした場合は、そのスイッチに関連するモニター・データの収集を、データベース・マネージャーに指示します。多くのモニター・データを収集すると、データベース・マネージャーのオーバーヘッドが増加し、システム・パフォーマンスに影響を与えます。 dft_mon_timestamp スイッチを OFF にすることが、CPU 使用率が 100% に近付くにつれて重要になります。こうなったときは、タイム・スタンプを発行するのに必要な CPU 時間が大幅に増大します。さらに、タイム・スタンプ・スイッチを OFF にすると、モニター・スイッチのコントロール下にある他のデータの合計コストが大幅に削減されます。

すべてのモニター・アプリケーションは、アプリケーションがその最初のモニター要求を出したときに、これらのデフォルト・スイッチ設定を継承します (たとえば、スイッチを設定し、イベント・モニターを活動化し、スナップショットを取る)。構成ファイル内のスイッチは、データベース・マネージャーの始動時からデータの収集を開始する場合にのみ、オンに設定してください。 (オンに設定しない場合、各モニター・アプリケーションはそれ自体のスイッチを設定することができ、収集したデータはそのスイッチが設定された時刻に関連付けられるようになります。)

dftdbpath - デフォルト・データベース・パス

このパラメーターには、データベース・マネージャーのもとでデータベースを作成するために使用されるデフォルト・ファイル・パスが含まれています。データベースの作成時にパスを指定しなかった場合、データベースは、*dftdbpath* パラメーターで指定されたパスに作成されます。

構成タイプ
データベース・マネージャー

適用

- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- ローカル・クライアントを持つデータベース・サーバー
- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ
オンラインで構成可能

伝搬クラス
即時

デフォルト [範囲]

UNIX インスタンス所有者のホーム・ディレクトリー [任意の既存のパス]

Windows
DB2 がインストールされているドライブ [任意の既存のパス]

パーティション・データベース環境では、データベースが作成されるパスは、NFS マウント・パス (Linux および UNIX プラットフォームの場合) でもネットワーク・ドライブ (Windows 環境の場合) でもないようにする必要があります。指定のパスが各データベース・パーティション・サーバーに物理的に存在している必要があります。混乱を避けるためには、それぞれのデータベース・パーティション・サーバー上にローカルにマウントされているパスを指定するのが一番良い方法です。パスの最大長は 205 文字です。システムは、パスの終わりにデータベース・パーティション名を追加します。

データベースは巨大なサイズになる場合があり、また多くのユーザーがデータベースを作成することも (環境と目的による) あるため、すべてのデータベースを指定のロケーションに作成して格納することができるようにすると便利です。また、整合性やバックアップおよびリカバリーのためにも、データベースをアプリケーションとデータから分離しておくことも有益です。

Linux および UNIX 環境の場合、*dftdbpath* 名の長さは、215 文字以下の有効で絶対的なパス名でなければなりません。Windows の場合は、*dftdbpath* にはドライブ名が指定でき、オプションで後にコロンを続けても構いません。

推奨: 可能であれば、ボリュームの大きいデータベースは、オペレーティング・システム・ファイルやデータベース・ログなど、頻繁にアクセスされるデータとは異なるディスクに配置してください。

diaglevel - 診断エラー・キャプチャー・レベル

このパラメーターは、db2diag.log ファイルに記録される診断エラーのタイプを指定します。

構成タイプ
データベース・マネージャー

適用

- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- クライアント
- ローカル・クライアントを持つデータベース・サーバー
- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ
オンラインで構成可能

伝搬クラス
即時

デフォルト [範囲]
3 [0 — 4]

このパラメーターの有効な値は、次のとおりです。

- **0** - キャプチャーされる診断データなし
- **1** - 重大エラーのみ
- **2** - すべてのエラー
- **3** - すべてのエラーおよび警告
- **4** - すべてのエラー、警告、および通知メッセージ

diagpath 構成パラメーターを使って、*diaglevel* パラメーターの値に基づいて生成される可能性があるエラー・ファイル、アラート・ログ・ファイル、およびダンプ・ファイルが置かれるディレクトリーを指定します。

推奨: 問題の解決に役立つ追加の問題判別データを収集する場合は、このパラメーターの値を大きくすることができます。

diagpath - 診断データ・ディレクトリー・パス

このパラメーターを使用して、DB2 診断情報のための完全修飾パスを指定できます。

構成タイプ
データベース・マネージャー

適用

- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- クライアント
- ローカル・クライアントを持つデータベース・サーバー
- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ
オンラインで構成可能

伝搬クラス
即時

デフォルト [範囲]
NULL [任意の有効なパス名]

このディレクトリーには、ご使用のプラットフォームに応じて、ダンプ・ファイル、トラップ・ファイル、エラー・ログ・ファイル、通知ファイル、アラート・ログ・ファイル、および FODC (First Occurrence Data Collection: 最初のオカレンスのデータ収集) パッケージが入れられます。

このパラメーターが null の場合は、診断情報が、以下のいずれかのディレクトリーまたはフォルダーにあるファイルに書き込まれます。

- Windows 環境では、以下のとおりです。
  - DB2INSTPROF 環境変数を設定しない場合は、情報は *x*:¥*SQLLIB*¥*DB2INSTANCE* に書き込まれます。ここで *x* はドライブ参照、*SQLLIB* は **DB2PATH** レジストリー変数に指定したディレクトリー、および *DB2INSTANCE* はインスタンス所有者の名前です。
  - **DB2INSTPROF** 環境変数を設定する場合は、情報は *x*:¥*DB2INSTPROF*¥*DB2INSTANCE* に書き込まれます。ここで *x* はドライブ参照、*DB2INSTPROF* はインスタンス・プロファイル・ディレクトリーの名前、および *DB2INSTANCE* はインスタンスの名前です。
  - ユーザー・データ・ファイル (例えばインスタンス・ディレクトリーの下のファイルなど) は、次のように、コードがインストールされている場所とは別の場所に書き込まれます。
    - Windows Vista 環境では、ユーザー・データ・ファイルは ProgramData¥IBM¥DB2¥ に書き込まれます。
    - Windows 2003 と XP 環境では、ユーザー・データ・ファイルは Documents and Settings¥All Users¥Application Data¥IBM¥DB2¥*Copy Name* に書き込まれます。ここで *Copy Name* は、ご使用の DB2 コピーの名前です。
- Linux および UNIX 環境では、情報は *INSTHOME*/sqllib/db2dump に書き込まれます。ここで、*INSTHOME* はインスタンスのホーム・ディレクトリーです。

バージョン 9.5 では、グローバル・レベルの **DB2INSTPROF** のデフォルト値は、上記の新しい場所に保管されます。実行時データ・ファイルの場所を指定するその他のプロファイル・レジストリー変数は、**DB2INSTPROF** の値を照会する必要があります。他の変数には、以下のものがあります。

- **DB2CLIINIPATH**
- **DIAGPATH**
- **SPM_LOG_PATH**

推奨: *diagpath* 構成パラメーターのデフォルト設定を使用するか、複数のインスタンスの場合は、*diagpath* 値を 1 つの場所にまとめて使用してください。

パーティション・データベース環境では、*diagpath* パラメーターはロギングによるパフォーマンスを最高にするために、ホストのローカル・ストレージを使用する必要があります。これにより、それぞれの物理パーティションに、別々のロギングおよび診断ディレクトリーが作成されます。PD_GET_DIAG_HIST 表関数を使用して、別のパーティションからログ・レコードを取り出し、PD_GET_LOG_MSGS 表関数を使用してすべてのパーティションからの通知ログを取り出すことができます。

dir_cache - ディレクトリー・キャッシュ・サポート

このパラメーターは、データベース、ノード、および DCS ディレクトリー・ファイルをメモリーにキャッシュするかどうかを決定します。

構成タイプ

データベース・マネージャー

適用

- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- クライアント
- ローカル・クライアントを持つデータベース・サーバー
- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ

構成可能

デフォルト [範囲]

Yes [Yes; No]

割り振られるタイミング

- アプリケーションが最初の接続を発行したとき、アプリケーション・キャッシュが割り振られます。
- データベース・マネージャー・インスタンスが開始されたとき (db2start)、サーバー・ディレクトリー・キャッシュが割り振られます。

解放されるタイミング

- アプリケーションが処理を終了したとき、アプリケーション・キャッシュが解放されます。
- データベース・マネージャー・インスタンスが停止したとき (db2stop)、サーバー・ディレクトリー・キャッシュが解放されます。

ディレクトリー・キャッシュを使用すると、ディレクトリー・ファイル入出力がなくなり、ディレクトリー情報を得るためのディレクトリー検索が最小化されるため、接続のコストが低減します。 ディレクトリー・キャッシュには次の 2 つのタイプがあります。

- 各アプリケーションについて、アプリケーションを実行中のマシン上に割り振られ使用される、アプリケーション・ディレクトリー・キャッシュ
- 内部データベース・マネージャー・プロセスのいくつかについて割り振られ使用される、サーバー・ディレクトリー・キャッシュ

アプリケーション・ディレクトリー・キャッシュでは、アプリケーションがその最初の接続を発行したとき、各ディレクトリー・ファイルが読み取られ、情報がこのアプリケーションのために専用メモリーにキャッシュされます。 キャッシュは、後続の接続要求でアプリケーション処理によって使用され、アプリケーション処理が終了するまで保持されます。データベースがアプリケーション・ディレクトリー・キャッシュに見つからない場合、ディレクトリー・ファイルが検索されますが、キャッシュは更新されません。 アプリケーションがディレクトリー項目を変更した場合、そのアプリケーションで次の接続が発行されると、このアプリケーションのキャッシュはリフレッシュされます。他のアプリケーションのアプリケーション・デ

ィレクトリー・キャッシュはリフレッシュされません。アプリケーション処理が終了すると、キャッシュは解放されます。 (コマンド行プロセッサー・セッションが使用するディレクトリー・キャッシュをリフレッシュするには、 db2 terminate コマンドを発行してください。)

サーバー・ディレクトリー・キャッシュでは、データベース・マネージャー・インスタンスが開始されると (db2start)、それぞれのディレクトリー・ファイルが読み取られ、情報が共用メモリーにキャッシュされます。このキャッシュは、インスタンスが停止するまで、(db2stop) 保持されます。ディレクトリー項目がこのキャッシュに見つからない場合、ディレクトリー・ファイルが検索されて情報が検索されます。このサーバー・ディレクトリー・キャッシュは、インスタンスの実行中はリフレッシュされません。

推奨: ディレクトリー・ファイルの変更が頻繁ではなく、パフォーマンスが重要な場合は、ディレクトリー・キャッシュを使用してください。

また、リモート・クライアントでは、ディレクトリー・キャッシュは、アプリケーションがいくつかの異なる接続要求を発行する場合に役立ちます。 この場合、キャッシュを行うと、1 つのアプリケーションがディレクトリー・ファイルを読む必要回数が減少します。

ディレクトリー・キャッシュは、データベース・システム・モニター・スナップショットの取得のパフォーマンスも向上させます。さらに、データベースの別名を使用する代わりに、スナップショット呼び出しでデータベース名を明示的に参照することもできます。

注: ディレクトリー・キャッシュがオンで、データベース・マネージャーの開始後に、データベースがカタログ、アンカタログ、作成、またはドロップされる場合、スナップショット呼び出しを実行したときに、エラーが起きる場合があります。

discover - ディスカバリー・モード

このパラメーターは、クライアントが行えるディスカバリー要求があれば、その種類を決定します。

構成タイプ

データベース・マネージャー

適用

- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- クライアント
- ローカル・クライアントを持つデータベース・サーバー
- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ

構成可能

デフォルト [範囲]

SEARCH [DISABLE, KNOWN, SEARCH]

クライアント側の観点からは、次のいずれかが行われます。

- *discover* = SEARCH の場合、クライアントは、ネットワーク上の DB2 サーバー・システムを見つけるための SEARCH ディスカバリー要求を発行できます。SEARCH ディスカバリーは、KNOWN ディスカバリーが提供する機能のスーパーセットを提供します。 *discover* = SEARCH の場合、クライアントは SEARCH ディスカバリー要求および KNOWN ディスカバリー要求の両方を発行することができます。
- *discover* = KNOWN の場合、クライアントは KNOWN ディスカバリー要求のみを発行することができます。特定のシステム上の Administration Server に関する接続情報を指定すると、DB2 システム上のすべてのインスタンスおよびデータベース情報がクライアントに返されます。
- *discover* = DISABLE の場合、クライアントではディスカバリーが使用不可になります。

デフォルトのディスカバリー・モードは SEARCH です。

discover_inst - サーバー・インスタンスのディスカバリー

このパラメーターは、このインスタンスが DB2 ディスカバリーによって検出できるかどうかを指定します。

構成タイプ
データベース・マネージャー

適用
- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- クライアント
- ローカル・クライアントを持つデータベース・サーバー
- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ
オンラインで構成可能

伝搬クラス
即時

デフォルト [範囲]
ENABLE [ENABLE, DISABLE]

このパラメーターのデフォルトは「enable」で、これはインスタンスを検出できますが、「disable」を指定すると、インスタンスは検出できません。

fcm_num_buffers - FCM バッファー数

このパラメーターは、データベース・サーバー間とデータベース・サーバー内の両方での内部通信 (メッセージ) 用として使用される 4KB バッファーの数を指定します。

構成タイプ
データベース・マネージャー

適用
- ローカルおよびリモート・クライアントを持つデータベース・サーバー

- ローカル・クライアントを持つデータベース・サーバー
- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ
オンラインで構成可能

伝搬クラス
即時

デフォルト [範囲]

32 ビット・プラットフォーム
Automatic [128 - 65 300]

64 ビット・プラットフォーム
Automatic [128 - 524 288]

- ローカルとリモート・クライアントを持つデータベース・サーバー。デフォルトは、1024。
- ローカル・クライアントを持つデータベース・サーバー。デフォルトは、512。
- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー。デフォルトは、4096。

単一パーティション・データベース・システムでは、 *intra_parallel* パラメーターがアクティブでない場合、このパラメーターは使用されません。

AUTOMATIC に設定されている場合、FCM はリソースの使用状況をモニターして、30 分間使用されていないリソースがあれば徐々に解放していきます。インスタンスの開始時に指定されたリソース数をデータベース・マネージャーが割り振れない場合には、インスタンスを開始できるようになるまで、構成値を徐々に減らします。

同じマシン上に複数の論理ノードがある場合は、このパラメーターの値を大きくする必要がある場合もあります。また、システム上のユーザーの数、システム上のデータベース・パーティション・サーバーの数、または複雑なアプリケーションのためにメッセージ・バッファーを使い尽くした場合は、このパラメーターの値を大きくしなければならない場合があります。

複数の論理ノードを使用している場合は、*fcm_num_buffers* バッファーの 1 つのプールが同じマシン上の複数の論理ノードすべてで共有されます。プールのサイズは、*fcm_num_buffers* 値にその物理マシン上の論理ノードの数を乗算して決定します。使用中の値をもう一度調べて、複数の論理ノードがあるマシン上で割り振られる FCM バッファーの合計数を考慮してください。

fcm_num_channels - FCM チャネル数

このパラメーターは、各データベース・パーティションの FCM チャネル数を指定します。

構成タイプ
データベース・マネージャー

適用

- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- ローカル・クライアントを持つデータベース・サーバー
- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー
- ローカル・クライアントを持つサテライト・データベース・サーバー

パラメーター・タイプ
オンラインで構成可能

伝搬クラス
即時

デフォルト [範囲]

UNIX 32 ビット・プラットフォーム
自動。開始値は 256、512、2 048 [128 - 120 000]

UNIX 64 ビット・プラットフォーム
自動。開始値は 256、512、2 048 [128 - 524 288]

Windows 32 ビット
自動。開始値は 10 000 [128 - 120 000]

Windows 64 ビット
自動。開始値は 256、512、2 048 [128 - 524 288]

- ローカルとリモート・クライアントを持つデータベース・サーバーの場合、開始値は 512 です。
- ローカル・クライアントを持つデータベース・サーバーの場合、開始値は 256 です。
- ローカルとリモート・クライアントを持つパーティション・データベース環境サーバーの場合、開始値は 2 048 です。

非パーティションのデータベース環境では、*intra_parallel* パラメーターがアクティブでないと、*fcm_num_channels* を使用できません。

FCM チャネルは、DB2 エンジンで実行される EDU 間の論理通信エンドポイントです。両方の制御フロー (要求と応答) とデータ・フロー (表キュー・データ) はチャネルにより、パーティション間のデータを転送します。

AUTOMATIC に設定されている場合、FCM はチャネルの使用状況をモニターし、要件の変化に応じて徐々にリソースの割り振りや解放を行います。

fed_noauth - フェデレーテッド・データベース認証をバイパス

このパラメーターは、フェデレーテッド認証をインスタンスでバイパスするかどうかを決定します。

構成タイプ
データベース・マネージャー

適用

- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- ローカル・クライアントを持つデータベース・サーバー

- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ
オンラインで構成可能

伝搬クラス
即時

デフォルト [範囲]
No [Yes; No]

fed_noauth が *yes* に設定され、 *authentication* が *server* または *server_encrypt* に設定され、*federated* が *yes* に設定されていると、インスタンスでの認証はバイパスされます。認証はデータ・ソースで行われると見なされます。 *fed_noauth* が *yes* に設定されているときは、注意してください。認証はクライアントでも、DB2 でも行われません。SYSADM 認証名を知っているユーザーであれば、このフェデレーテッド・サーバーの SYSADM 権限を持つことができます。

federated - フェデレーテッド・データベース・システム・サポート

このパラメーターは、データ・ソース (DB2 ファミリーおよび Oracle など) によって管理されるデータに対する分散要求をアプリケーションが実行できるようにする機能を使用可能または使用不能にします。

構成タイプ
データベース・マネージャー

適用
- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- ローカル・クライアントを持つデータベース・サーバー
- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ
構成可能

デフォルト [範囲]
No [Yes; No]

federated_async - 照会ごとの非同期 TQ の最大数構成パラメーター

このパラメーターは、フェデレーテッド・サーバーがサポートするアクセス・プラン内の非同期表キュー (ATQ) の最大数を決定します。

構成タイプ
データベース・マネージャー

適用
- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- ローカル・クライアントを持つデータベース・サーバー

- フェデレーションが有効化された場合、ローカルとリモート・クライアントを持つパーティション・データベース・サーバー。

パラメーター・タイプ
オンラインで構成可能

デフォルト [範囲]
0 [0 - 32,767, -1, ANY]

ANY か -1 を指定すると、オプティマイザーがそのアクセス・プランに対する ATQ の数を決定します。オプティマイザーは、プラン内の適格な SHIP またはリモート・プッシュダウン演算子すべてに ATQ を割り当てます。DB2_MAX_ASYNC_REQUESTS_PER_QUERY サーバー・オプションに指定される値は、非同期要求の数を制限します。

推奨 *federated_async* 構成パラメーターは、特殊レジスターおよびバインド・オプションにデフォルトまたは開始値を指定します。このパラメーターの値は、CURRENT FEDERATED ASYNCHRONY 特殊レジスター、FEDERATED_ASYNCHRONY バインド・オプション、または FEDERATED_ASYNCHRONY プリコンパイル・オプションの値の設定を増減させることによってオーバーライドできます。

特殊レジスターまたはバインド・オプションにより *federated_async* 構成パラメーターがオーバーライドされない場合、このパラメーターの値は、フェデレーテッド・サーバーが許可するアクセス・プランの ATQ の最大数を決定します。特殊レジスターまたはバインド・オプションによりこのパラメーターがオーバーライドされる場合、特殊レジスターまたはバインド・オプションの値は、プランの ATQ の最大数を決定します。

federated_async 構成パラメーターの値を変更した場合、現行の作業単位がコミットすると直ちに動的ステートメントに影響します。後続の動的ステートメントは、自動的に新しい値を認識します。フェデレーテッド・データベースの再始動は必要ありません。組み込み SQL パッケージは、*federated_async* 構成パラメーターの値が変更されても、無効化も暗黙的再バインドも行われません。

federated_async 構成パラメーターに新しい値を指定して、静的 SQL ステートメントに影響を与えようとする場合、パッケージを再バインドする必要があります。

fenced_pool - fenced プロセスの最大数

スレッド化 db2fmp プロセス (スレッド・セーフ・ストアード・プロシージャーおよび UDF を実行するプロセス) の場合、このパラメーターは、それぞれの db2fmp プロセス内でキャッシュに入れられるスレッドの数を表します。非スレッド化 db2fmp プロセスの場合は、このパラメーターではキャッシュに入れられるプロセスの数を表します。

構成タイプ
データベース・マネージャー

適用

- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- ローカル・クライアントを持つデータベース・サーバー

- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ
オンラインで構成可能

デフォルト [範囲]
-1 (*max_coordagents*), Automatic [-1; 0–64 000]

単位 カウンター

制約事項:

- このパラメーターが動的に更新され、値が小さくなると、データベース・マネージャーは db2fmp スレッドまたはプロセスを積極的に終了せず、キャッシュされた db2fmp の数を新しい値まで減らすために、これらが使用されるごとにそれらのキャッシングを停止します。
- このパラメーターが動的に更新され、値が大きくなると、さらに db2fmp スレッドおよびプロセスが作成されたときにデータベース・マネージャーはそれをキャッシュに入れます。
- このパラメーターを -1 (デフォルト) に設定すると、*max_coordagents* 構成パラメーターの値とみなされます。*max_coordagents* の値のみが想定されており、自動設定や動作は想定されていないことに注意してください。
- このパラメーターが AUTOMATIC (これもデフォルト) に設定されると、以下のようになります。
  - データベース・マネージャーは、キャッシュされる db2fmp スレッドおよびプロセスの数の増加を、コーディネーター・エージェントの最高水準点に基づいて許可します。特にこのパラメーターの自動の動作は、この数の増加を、データベース・マネージャーが開始されてからこれまでに同時に登録されていたコーディネーター・エージェントの最大数に応じて許可します。
  - このパラメーターに割り当てられた値は、キャッシュに入れる db2fmp スレッドおよびプロセスの数の下限を表します。

推奨: fenced ストアード・プロシージャーまたはユーザー定義関数を使用している環境の場合は、インスタンス上で実行する最大数の同時ストアード・プロシージャーおよび UDF を処理するために、適切な数の db2fmp プロセスが確実に使用可能であるよう、このパラメーターを使用することができます。これにより、ストアード・プロシージャーおよび UDF の実行の一環として新しい fenced モード・プロセスを作成する必要がないようにすることができます。

デフォルト値で db2fmp プロセスに付与されるシステム・リソースが適切でない場合には、データベース・マネージャーのパフォーマンスに影響を生じるため、ご使用の環境ではデフォルト値が適切でない場合は、下記を使用すれば、このパラメーターを調整するための手掛かりが得られます。

```
fenced_pool = # of applications allowed to make stored procedure and
UDF calls at one time
```

keepfenced が YES に設定されている場合は、fenced ルーチン呼び出しが処理され、エージェントに戻された後も、キャッシュ・プール内に作成されたそれぞれの db2fmp プロセスは存在し、システム・リソースを使用し続けます。

keepfenced が NO に設定されている場合は、非スレッド化 db2fmp プロセスは、実行を完了した時点で終了するので、キャッシュ・プールはありません。マルチスレッド化 db2fmp プロセスは存在し続けますが、これらのプロセスにプールされるスレッドはありません。つまり、*keepfenced* が NO に設定されている場合でも、スレッド化 C db2fmp プロセスを 1 つと、スレッド化 Java db2fmp プロセスを 1 つ、システム上に持つことができることを示します。

以前のバージョンでは、このパラメーターは *maxdari* という名前で呼ばれていました。

group_plugin - グループ・プラグイン

このパラメーターでは、グループ・プラグイン・ライブラリーの名前を指定します。

構成タイプ
データベース・マネージャー

適用
- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- クライアント
- ローカル・クライアントを持つデータベース・サーバー
- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ
構成可能

デフォルト [範囲]
Null [任意の有効ストリング]

デフォルトの値は NULL で、DB2 はオペレーティング・システムのグループ・ルックアップを使用します。このプラグインはすべてのグループ参照数に使用されます。非 root インストールについては、DB2 のユーザー ID およびパスワード・プラグイン・ライブラリーが使用される場合は、DB2 製品を使用する前に、db2rfe コマンドを実行する必要があります。

health_mon - ヘルス・モニター

このパラメーターを使用すると、インスタンス、その関連データベース、およびデータベース・オブジェクトを健全性を表すさまざまな指標に従ってモニターするかどうか指定できます。

構成タイプ
データベース・マネージャー

パラメーター・タイプ
オンラインで構成可能

伝搬クラス
即時

デフォルト [範囲]
On [On; Off]

関連パラメーター

health_mon をオンにすると (デフォルト)、ユーザーが選択したオブジェクトの健全性についての情報を、エージェントが収集します。ユーザーが設定したしきい値を基にして、オブジェクトが不健全な位置にあると見なされた場合は、通知が送信され、自動的にアクションが取られます。 *health_mon* をオフにした場合、オブジェクトの健全性はモニターされません。

ヘルス・センターまたは CLP を使用して、モニターするインスタンスおよびデータベース・オブジェクトを選択できます。また、ヘルス・モニターによって収集されたデータに基づいて、通知の送信先をどこにし、どのようなアクションを取る必要があるかについても指定できます。

indexrec - 索引再作成時点

このパラメーターでは、データベース・マネージャーがいつ無効な索引の再作成を試みるか、さらにスタンバイ・データベースでの DB2 ロールフォワードまたは HADR ログ再生の間に索引作成を再実行するかどうかを指示します。

構成タイプ

データベースおよびデータベース・マネージャー

適用

- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- ローカル・クライアントを持つデータベース・サーバー
- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ

オンラインで構成可能

伝搬クラス

即時

デフォルト [範囲]

UNIX データベース・マネージャー

restart [restart; restart_no_redo; access; access_no_redo]

Windows データベース・マネージャー

restart [restart; restart_no_redo; access; access_no_redo]

データベース

Use system setting [system; restart; restart_no_redo; access; access_no_redo]

このパラメーターには、次に示す 5 つを設定できます。

SYSTEM

データベース・マネージャーの構成ファイルに指定されているシステム設定*を使用* して、無効な索引の再作成がいつ行われるか、DB2 ロールフォワードまたは HADR ログ再生の間に索引作成ログ・レコードが再実行されるかどうかを指定します。(注: この設定は、データベース構成の場合にのみ有効です)。

ACCESS

無効な索引は、その索引に対して最初にアクセスが行われるときに再作成されます。フル・ロギングで記録された索引作成は、DB2 ロールフォワードまたは HADR ログ再生の間に再実行されます。HADR が開始され、HADR のテークオーバーが行われると、基礎表が最初にアクセスされるときに、テークオーバーに続いて無効な索引が再作成されます。

ACCESS_NO_REDO

無効な索引は、基礎表が最初にアクセスされるときに再作成されます。フル・ロギングで記録された索引作成は DB2 ロールフォワードまたは HADR ログ再生の際に再実行されず、それら無効な索引は無効のままになります。HADR が開始され、HADR のテークオーバーが行われると、基礎表が最初にアクセスされるときに、テークオーバーに続いて無効な索引が再作成されます。

RESTART

indexrec のデフォルト値です。 RESTART DATABASE コマンドが明示的または暗黙的に発行されるときに、無効な索引は再作成されます。フル・ロギングで記録された索引作成は、DB2 ロールフォワードまたは HADR ログ再生の間に再実行されます。HADR が開始され、HADR のテークオーバーが行われると、テークオーバーの最後に無効な索引が再作成されます。

なお、RESTART DATABASE コマンドが暗黙的に発行されるのは、*autorestart* パラメーターが使用可能になっている場合です。

RESTART_NO_REDO

RESTART DATABASE コマンドが明示的または暗黙的に発行されるときに、無効な索引は再作成されます。(RESTART DATABASE コマンドが暗黙的に発行されるのは、*autorestart* パラメーターが使用可能になっている場合です。)フル・ロギングで記録された索引作成は DB2 ロールフォワードまたは HADR ログ再生の際に再実行されませんが、それらの索引はロールフォワードが完了するとき、または HADR テークオーバーが行われるときに再作成されます。

索引が無効になるのは、致命的なディスク問題が発生したときです。これがデータ自体に生じた場合は、データが失われる恐れがあります。しかし、この問題が発生したのが索引である場合は、その索引を再作成することで、索引はリカバリーできます。ただし、ユーザーがデータベースに接続されているときに、索引が再作成されると、次のような 2 つの問題が生じる可能性があります。

- 索引ファイルの再作成に伴って、予期しない応答時間の低下が発生する場合があります。ユーザーが表にアクセスして、この特定の索引を使用する場合は、索引が再作成されている間、ユーザーは待機することになります。
- 索引の再作成後、予期しないロックが保留される場合があります。特に、その索引の再作成の原因となったユーザー・トランザクションが COMMIT や ROLLBACK をまったく実行していなかった場合は、その可能性があります。

推奨: このオプションに関しては、ユーザー数の多いサーバーであり、しかも再始動のタイミングが重要でない場合は、クラッシュ後にデータベースをオンラインに戻す処理の一環として、DATABASE RESTART 時に索引が再作成されるようにするのが、最善の選択です。

このパラメーターを「ACCESS」または「ACCESS_NO_REDO」に設定すると、索引の再作成の間、データベース・マネージャーのパフォーマンスが低下します。ユーザーがその特定の索引または表にアクセスした場合は、索引が再作成されるまで、ユーザーは待機する必要があります。

このパラメーターが「RESTART」に設定されている場合は、データベースの再始動に要する時間は、索引の再作成のせいで長くなりますが、データベースがオンラインに戻った後は、通常の処理に影響が生じることはありません。

注: データベース・リカバリー時には、リカバリー中のデータベースに属するファイル・システム上にある、すべての SQL プロシージャー実行可能ファイルが除去されます。 *indexrec* が RESTART に設定されている場合は、すべての SQL プロシージャー実行可能ファイルがデータベース・カタログから取り出され、次回データベースに接続するときにファイル・システムに書き戻されます。 *indexrec* が RESTART に設定されていない場合、SQL 実行可能ファイルは、その SQL プロシージャーが最初に実行されるときにだけファイル・システムに取り出されます。

RESTART 値と RESTART_NO_REDO 値の違い、または ACCESS 値と ACCESS_NO_REDO 値の違いは、CREATE INDEX および REORG INDEX 操作のような索引作成操作、または索引再作成に関してフル・ロギングが有効にされている場合にのみ顕著となります。このロギングは、*logindexbuild* データベース構成パラメーターを使用可能にするか、表を変更するときに LOG INDEX BUILD を使用可能にすることによって有効にできます。 *indexrec* を RESTART または ACCESS に設定すると、索引オブジェクトを無効な状態のままにはせずに、ロギングによって記録された索引作成に関係した操作のロールフォワードが行われるため、索引を後で再作成しなければならない場合があります。

instance_memory - インスタンス・メモリー

このパラメーターは、1 つのデータベース・パーティションに割り振ることができるメモリーの最大量を指定します。

構成タイプ
データベース・マネージャー

適用

- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- ローカル・クライアントを持つデータベース・サーバー
- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ
オンラインで構成可能

デフォルト [範囲]

32 ビット・プラットフォーム
Automatic [0 - 1 000 000]

64 ビット・プラットフォーム
Automatic [0 - 68 719 476 736]

単位 ページ (4 KB)

割り振られるタイミング
インスタンスが開始されたとき

解放されるタイミング
インスタンスが停止されたとき

instance_memory のデフォルト値は AUTOMATIC になっているので、データベース・パーティションのアクティベーション時 (db2start) に実際の値が計算されます。使用される実際の値は、システムの物理 RAM の 75% から 95% の間で、インスタンス内の構成されているローカル・データベース・パーティションの数で割ったものです。この値は、専用データベース・サーバー・システムに対して適切である必要があります。

注:

- *instance_memory* で指定した値が、システムの物理メモリー量より大きいと、db2start は失敗し、SQL1220N (データベース・マネージャーの共有メモリーを割り振れません) が出されます。
- *instance_memory* が、物理 RAM の量より少ない値に動的に更新される場合は、要求が処理されて新しい上限が設定されます。動的に *instance_memory* が減らされるのは、新しい設定が使用中の *instance_memory* の現行量より大きい場合のみです。それ以外の場合は、要求は次の db2start まで据え置かれます。
- *instance_memory* が、インスタンスがアクティブであるときの物理 RAM の量より大きな値に動的に更新されると、要求は据え置かれ、次回の db2start は失敗し、SQL1220N (データベース・マネージャー共有メモリーを割り振れません) が出されます。

高速コミュニケーション・マネージャー (FCM) 共有メモリーが割り振られている場合、システムの FCM 共有メモリー・サイズ全体のうちの各ローカル・データベース・パーティションの分が、そのデータベース・パーティションの *instance_memory* 限度になります。

特定のヒープにメモリーが要求され、かつデータベース・パーティション・メモリー限度 (*instance_memory*) にすでに達している場合は、DB2 はまず共有および専用ヒープ全体のメモリー使用量を要求されたメモリー量の分だけ減らそうとします。それでも十分な空き *instance_memory* がない場合は、要求は失敗し、その要求を開始したアプリケーションは、どのヒープがメモリー不足により失敗したかを説明する適切な SQLCODE を受け取ります。

この動作に対する例外は、DB2 オペレーションにとって重大であると認識されたメモリー要求に対するものです (つまり、メモリー要求が失敗すると、データベースが無効であると見なされる、またはインスタンスがシャットダウンする)。この重大な要求は、まずデータベース・パーティションの現行のメモリー使用量を減らそうとします。それでも十分な空き *instance_memory* がない場合は、DB2 がオペレーティング・システムからメモリーを要求します。オペレーティング・システムがそのメモリー要求を許可すると、*instance_memory* の現行値は、構成された上限を超えます。ただし他の重大ではないメモリー要求は、十分なメモリーが解放されるまですべて失敗します。

注: DPF インスタンスの制限事項: *instance_memory* は、1 つの DB2 データベース・パーティションが割り振るメモリーの量を指定しますが、これはインスタン

ス・レベルの構成パラメーターであるため、インスタンスにあるすべてのデータベース・パーティションは同一のメモリー限度を持つようになります。

DB2 メモリーの消費を制御する:

instance_memory が AUTOMATIC に設定されている場合、インスタンスの合計のメモリー消費量の固定された上限は、インスタンスの開始時 (db2start) に設定されます。DB2 の実際のメモリー消費量は、作業負荷により変動します。STMM で *database_memory* 調整の実行が有効になっていると (新規データベースのデフォルト)、ランタイム時に STMM が、ファンクション・メモリー所要量に使用できる十分な空き *instance_memory* を確保しつつ、システムの空き物理メモリーに応じてデータベース共有メモリー・セット内でのパフォーマンスが重要なヒープのサイズを動的に更新します。

DB2 のデフォルトのメモリー構成は、作業負荷に合わせて、全体のインスタンス・メモリーの明示的なセルフチューニングを必要とせずに、インスタンスのメモリー所要量に適応します。例えば、次のようにします。

- 頻繁に使用されるインスタンスでは、STMM はパフォーマンスが重要なヒープのサイズを必要に応じて増やします。アプリケーションにサービスを提供し、ファンクション・メモリーを消費するデータベース・エージェントが増えると、より多くのファンクション・メモリーが消費されます。十分な空き *instance_memory* はあるものの、システムの物理メモリーが非常に少ない場合、STMM は、パフォーマンスが重要なヒープのサイズを削減して、システムがページングを開始しないようにします。ファンクション・メモリー所要量が減ると、システムの空き物理メモリーが必然的に増えるので、STMM は再びパフォーマンスが重要なヒープを増やし始めます。
- あまり使用しないインスタンスの場合、インスタンスによって消費されるファンクション・メモリーは少なくなります。また、十分な空き物理メモリーがシステムに残っていない場合は、STMM はパフォーマンスが重要なヒープを縮小します。

instance_memory が特定の値に設定されており、少なくとも 1 つのアクティブなデータベースで *database_memory* が AUTOMATIC 値になっていて、かつ STMM がそのデータベースに対して有効である場合、STMM は *database_memory* のサイズを増やして、DB2 が *instance_memory* で指定されているメモリーのほとんどすべてを使用して、ファンクション・メモリー要求に使用できる十分な空き *instance_memory* だけを確保するようにします。このシナリオでは、STMM はマシンにある空き物理メモリーのモニターは行いません。そのため、ページングが発生しないように *instance_memory* を適切に構成する必要があります。

新しい admin_get_dbp_mem_usage ユーザー定義関数 (UDF) を使用して、特定のデータベース・パーティションまたはすべてのデータベース・パーティションに対して DB2 インスタンスが消費するメモリーの総消費量を得ることができます。この UDF は、現在の上限値も返します。

一部の Linux カーネルでの制限:

一部の Linux カーネルのオペレーティング・システム制限のため、STMM で *database_memory* を AUTOMATIC に設定することは、現在許可されていません。 ただし、*instance_memory* が AUTOMATIC ではなく特定の値に設定されている場合に限り、この設定はこれらのカーネル上でも許可されま

す。*database_memory* が AUTOMATIC に設定されているときに、*instance_memory* が後から AUTOMATIC に戻されると、*database_memory* 構成パラメーターは、次回のデータベース・アクティベーション時に自動的に COMPUTED に更新されます。一部のデータベースがすでにアクティブになっている場合は、STMM が全体の *database_memory* サイズ調整を停止します。

intra_parallel - パーティション内並列処理機能の使用可能化

このパラメーターは、データベース・マネージャーでパーティション内並列処理を使用できるかどうかを指定します。

構成タイプ

データベース・マネージャー

適用

- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- ローカル・クライアントを持つデータベース・サーバー
- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ

構成可能

デフォルト [範囲]

NO (0) [SYSTEM (-1), NO (0), YES (1)]

値が -1 の場合、データベース・マネージャーが動作しているハードウェアに基づき、パラメーター値を「YES」または「NO」に設定します。

このパラメーターが「YES」の場合、並列操作によってパフォーマンスを改善できる操作の中には、データベース照会と索引作成が含まれます。

注:

- 並列索引作成では、この構成パラメーターは使用されません。
- このパラメーター値を変更すると、パッケージがデータベースに再バインドされることがあり、パフォーマンスが低下する場合もあります。

java_heap_sz - Java インタープリター最大ヒープ・サイズ

このパラメーターでは、Java DB2 ストアード・プロシージャーおよび UDF を使用可能にするために開始された Java インタープリターによって使用される、ヒープの最大サイズを決定します。

構成タイプ

データベース・マネージャー

適用

- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- クライアント
- ローカル・クライアントを持つデータベース・サーバー

- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ
構成可能

デフォルト [範囲]

HP-UX
4096 [0 - 524 288]

その他のすべてのオペレーティング・システム
2048 [0 - 524 288]

単位 ページ (4 KB)

割り振られるタイミング
Java ストアード・プロシージャーまたは UDF が始動するとき

解放されるタイミング
db2fmp プロセス (fenced) または db2agent プロセス (トラステッド) が終了するとき

ヒープは、それぞれの DB2 プロセスごとに 1 つずつ (Linux および UNIX プラットフォームの場合は、それぞれのエージェントまたはサブエージェントごとに 1 つずつ、およびそれ以外のプラットフォームの場合は、それぞれのインスタンスごとに 1 つずつ) あります。ヒープは、それぞれの fenced UDF および fenced ストアード・プロシージャー・プロセスごとに 1 つずつあります。ヒープは、トラステッド・ルーチンのエージェント (サブエージェントは含まない) ごとに 1 つずつあります。ヒープは、Java ストアード・プロシージャーを実行する db2fmp プロセスごとに 1 つずつあります。マルチスレッド db2fmp プロセスの場合は、スレッド・セーフ fenced ルーチンを使用する複数のアプリケーションが単一のヒープからサービスを受けます。いずれの状態においても、このメモリーを割り振ることがあるのは、Java UDF またはストアード・プロシージャーを実行するエージェントまたはプロセスだけです。パーティション・データベース・システムでは、それぞれのデータベース・パーティションで同じ値が使用されます。

XML データは、ストアード・プロシージャーに IN、OUT、または INOUT パラメーターとして受け渡された時、マテリアライズされます。Java ストアード・プロシージャーを使用している場合、XML 引数の数量やサイズおよび並行して実行されている外部ストアード・プロシージャーの数に応じて、ヒープ・サイズを大きくする必要がある可能性があります。

jdk_path - Software Developer's Kit for Java インストール・パス

このパラメーターは、Java 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する場合に使用される、Software Developer's Kit for Java をインストールするディレクトリーを指定します。Java インタープリターで使用される CLASSPATH や他の環境変数は、このパラメーターの値から計算されます。

構成タイプ
データベース・マネージャー

適用

- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- ローカル・クライアントを持つデータベース・サーバー
- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ
構成可能

デフォルト [範囲]
NULL [有効なパス]

ご使用の DB2 製品に SDK for Java がインストールされている場合は、このパラメーターが正しく設定されます。ただし、データベース・マネージャー (dbm cfg) パラメーターをリセットした場合は、SDK for Java のインストール場所を指定する必要があります。

keepfenced - fenced プロセスの維持

このパラメーターは fenced モード・ルーチン呼び出しの完了後に、fenced モード・プロセスが保持されるかどうかを示します。 fenced モード・プロセスは、ユーザー作成 fenced モード・コードをデータベース・マネージャー・エージェント・プロセスから分離するために、独立したシステム・エンティティーとして作成されます。このパラメーターは、データベース・サーバーにのみ適用できます。

構成タイプ
データベース・マネージャー

適用

- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- ローカル・クライアントを持つデータベース・サーバー
- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ
構成可能

デフォルト [範囲]
Yes [Yes; No]

keepfenced が *no* に設定されていて、実行されているルーチンがスレッド・セーフでない場合は、fenced モードの呼び出しごとに、新しい fenced モード・プロセスが作成および破棄されます。 *keepfenced* が *no* に設定されていて、実行されているルーチンがスレッド・セーフである場合は、fenced モード・プロセスは持続しますが、呼び出しのために作成されたスレッドは終了します。 *keepfenced* が *yes* に設定されている場合は、fenced モード・プロセスまたはスレッドが後続の fenced モード呼び出しのために再使用されます。データベース・マネージャーが停止されると、未解決の fenced モード・プロセスおよびスレッドすべてが終了します。

このパラメーターを *yes* に設定すると、追加のシステム・リソースが、活動化されている fenced モード・プロセスごとに、最大で *fenced_pool* パラメーターに含まれている値まで、データベース・マネージャーによって使用される結果になります。新しい処理は、既存の fenced モード・プロセスが使用可能でなく、後続の fenced

ルーチン呼び出しを処理できない場合にのみ作成されます。 *fenced_pool* が 0 に設定されている場合は、このパラメーターは無視されます。

推奨: fenced モード要求の数が fenced でないモード要求に比べて相対的に大きく、しかもシステム・リソースが制約されていない環境では、このパラメーターは *yes* に設定できます。こうすれば、呼び出しの処理には既存の fenced モード・プロセスが使用されるので、初期 fenced モード・プロセス作成オーバーヘッドを回避することで、fenced モード・プロセスのパフォーマンスが向上します。特に、Java ルーチンの場合は、こうすることによって、 Java 仮想マシン (JVM) を始動するコストが節減され、非常に大幅なパフォーマンスの向上がもたらされます。

例えば、銀行用 OLTP の貸借トランザクション・アプリケーションでは、各トランザクションを実行するコードは、分散モード・プロセスで処理されるストアード・プロシージャーで実行することができます。このアプリケーションでは、メイン・ワークロードは分散モード・プロセスの外側で実行されます。このパラメーターが *no* に設定されている場合は、それぞれのトランザクションごとに新しい fenced モード・プロセスを作成するオーバーヘッドが発生して、大幅なパフォーマンスの低下を招きます。ただし、このパラメーターを *yes* に設定すると、各トランザクションが既存の分散モード・プロセスを使用しようとして、このオーバーヘッドを避けることになります。

以前のバージョンの DB2 では、このパラメーターは *keepdari* という名前で呼ばれていました。

local_gssplugin - ローカル・インスタンス・レベル許可に使用する GSS API プラグイン

このパラメーターでは、*authentication* データベース・マネージャー構成パラメーターの値が GSSPLUGIN または GSS_SERVER_ENCRYPT に設定されている場合に、インスタンス・レベル・ローカル許可に使用するデフォルトの GSS API プラグイン・ライブラリーの名前を指定します。

構成タイプ
データベース・マネージャー

適用

- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- クライアント
- ローカル・クライアントを持つデータベース・サーバー
- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ
構成可能

デフォルト [範囲]
Null [任意の有効ストリング]

max_connections - クライアント接続の最大数

このパラメーターは、データベース・パーティション当たりの許容クライアント接続最大数を示します。

構成タイプ
データベース・マネージャー

パラメーター・タイプ
オンラインで構成可能

適用

- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- ローカル・クライアントを持つデータベース・サーバー
- ローカルおよびリモート・クライアントを持つデータベース・サーバーまたは接続サーバー (*max_connections*、*max_coordagents*、*num_initagents*、*num_poolagents* の場合。フェデレーテッド環境を使用している場合は、*federated_async* も該当します。)

デフォルト [範囲]
-1 および AUTOMATIC (*max_coordagents*) [-1 および AUTOMATIC; 1-64 000]

値を -1 に設定すると、自動設定値または動作ではなく、*max_coordagents* に関連付けられた値が使用されます。 AUTOMATIC は、データベース・マネージャーにより、最適に動作する技法が使用されて、このパラメーターの値が選択されることを意味します。 AUTOMATIC とは構成ファイル内での ON/OFF スイッチで値とは関係がないため、-1 と AUTOMATIC の両方をデフォルト設定にできます。

詳細については、 518 ページの『max_coordagents および max_connections を構成する際の制限と動作』 を参照してください。

コンセントレーター

コンセントレーターが OFF なのは、*max_connections* が *max_coordagents* に等しいか、それより小さいときです。*max_connections* が *max_coordagents* より大きいときは、コンセントレーターは ON です。

このパラメーターは、インスタンスのデータベース・パーティションに接続できるクライアント・アプリケーションの最大数をコントロールします。通常、各アプリケーションにはコーディネーター・プログラム・エージェントが割り当てられます。エージェントは、アプリケーションとデータベースの間の操作の手助けをします。このパラメーターにデフォルト値が使用される場合、コンセントレーター・フィーチャーは活動化されません。結果として、各エージェントは専用メモリー内で機能し、データベース・マネージャーと、バッファー・プールなどのデータベース・グローバル・リソースを他のエージェントと共有します。パラメーターがデフォルトよりも大きな値に設定されると、コンセントレーター・フィーチャーが活動化されます。

max_connretries - ノード接続再試行回数

このパラメーターは、2 つのデータベース・パーティション・サーバー間で TCP/IP 接続の確立を試行する最大回数を指定します。

構成タイプ
データベース・マネージャー

適用　ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ
オンラインで構成可能

伝搬クラス
即時

デフォルト [範囲]
5 [0–100]

2 つのデータベース・パーティション・サーバー間に通信を確立する試みが失敗した (例えば、*conn_elapse* パラメーターによって指定された値に達した) 場合は、*max_connretries* で、データベース・パーティション・サーバーに対して実行できる接続再試行回数を指定します。このパラメーターに指定された値を超えた場合は、エラーが戻されます。

max_coordagents - コーディネーター・エージェントの最大数

このパラメーターは、コーディネーター・エージェントの数を制限するために使用されます。

構成タイプ
データベース・マネージャー

適用

- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- ローカル・クライアントを持つデータベース・サーバー
- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ
オンラインで構成可能

デフォルト [範囲]
200, Automatic [-1; 0–64 000]

-1 と設定すると、200 という値に変換されます。

詳細については、 518 ページの『max_coordagents および max_connections を構成する際の制限と動作』 を参照してください。

コンセントレーター

コンセントレーターが OFF のとき、つまり、*max_connections* が *max_coordagents* 以下の場合、このパラメーターは、サーバー・ノード上に同時に存在できるコーディネーター・エージェントの最大数を決定します。

データベースに接続する、またはインスタンスにアタッチするローカルまたはリモートのアプリケーションごとに 1 つのコーディネーター・エージェントが獲得されます。インスタンスへのアタッチを必要とする要求には、CREATE DATABASE、DROP DATABASE、およびデータベース・システム・モニター・コマンドがあります。

コンセントレーターが ON の場合、つまり、*max_connections* が *max_coordagents* より大きいときは、接続の数がそれにサービスするためのコーディネーター・エージェントの数よりも多くなります。アプリケーションがアクティブ状態であるのは、アプリケーションにサービスするコーディネーター・エージェントがある場合だけです。コーディネーター・エージェントがない場合、アプリケーションは非アクティブ状態です。アクティブ・アプリケーションからの要求には、データベース・コーディネーター・エージェント (および SMP または MPP 構成内のサブエージェント) がサービスします。非アクティブ・アプリケーションからの要求はキューに入れられ、そのアプリケーションにサービスするデータベース・コーディネーター・エージェントが割り当てられると、アプリケーションがアクティブになります。したがって、このパラメーターを使用すると、システムに対する負荷をコントロールできます。

max_querydegree - 照会の最大並列処理多重度

このパラメーターは、データベース・マネージャーのこのインスタンスで実行中の SQL ステートメントで使用される、パーティション内並列処理の最大多重度を指定します。 SQL ステートメントの実行中に、そのステートメントによってデータベース・パーティション内でこの数より多くの並列操作が行われることはありません。

構成タイプ
データベース・マネージャー

適用

- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- ローカル・クライアントを持つデータベース・サーバー
- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ
オンラインで構成可能

伝搬クラス
ステートメント境界

デフォルト [範囲]
-1 (ANY) [ANY, 1 - 32 767] (ANY はシステム判別を指す)

データベース・パーティションでSQL ステートメントのパーティション内並列処理を使用できるようにするには、*intra_parallel* 構成パラメーターを「YES」に設定する必要があります。 *intra_parallel* パラメーターは、並列索引作成には必要なくなりました。

この構成パラメーターのデフォルト値は -1 です。この値は、オプティマイザーによって決められた並列処理の度合いがシステムで使用されることを意味します。それ以外の場合は、ユーザー指定の値が使用されます。

注: SQL ステートメントの並列処理の多重度は、CURRENT DEGREE 特殊レジスターまたは DEGREE BIND オプションを使用して、ステートメントのコンパイル時に指定することができます。

アクティブなアプリケーションの照会の最大並列処理多重度は、SET RUNTIME DEGREE コマンドを使用して変更することができます。実際に実行時に使用される多重度は、次の値より低くなります。

- *max_querydegree* 構成パラメーター
- アプリケーション実行時の多重度
- SQL ステートメント・コンパイル時の多重度

この構成パラメーターは、照会にのみ適用されます。

max_time_diff - ノード間最大時差

このパラメーターは、ノード構成ファイルにリストされているデータベース・パーティション・サーバー間に許容されている最大時差を分数で指定します。

構成タイプ
データベース・マネージャー

適用 ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ
構成可能

デフォルト [範囲]
60 [1 - 1 440]

単位 分

各データベース・パーティション・サーバーには、それぞれ固有のシステム・クロックがあります。複数のデータベース・パーティション・サーバーが 1 つのトランザクションに関連付けられていて、それらのクロックがこのパラメーターで指定されている時間内で同期していないと、そのトランザクションはリジェクトされ、SQLCODE が戻されます。 (トランザクションがリジェクトされるのは、データ変更がトランザクションに関連している場合だけです)。

DB2 では *協定世界時* (UTC) を使用しているので、このパラメーターを設定すると、異なる時間帯は考慮事項になりません。協定世界時とは、グリニッジ標準時と同じものです。

maxagents - 最大エージェント数

このパラメーターはバージョン 9.5 では推奨されませんが、バージョン 9.5 より前のデータ・サーバーおよびクライアントでは引き続き使用されています。DB2 バージョン 9.5 のデータベース・マネージャーは、この構成パラメーターに対して指定された値をすべて無視します。

注: 以下の情報は、バージョン 9.5 より前のデータ・サーバーおよびクライアントだけに当てはまります。

このパラメーターは、コーディネーター・エージェントかサブエージェントかにかかわらず、どの特定の時点においても、アプリケーション要求を受け入れるために使用可能なデータベース・マネージャー・エージェントの最大数を示します。

構成タイプ
データベース・マネージャー

適用
- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- ローカル・クライアントを持つデータベース・サーバー
- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ
構成可能

デフォルト [範囲]
200 [1 - 64 000]

400 [1 - 64 000] ローカル・クライアントおよびリモート・クライアントを持つパーティション・データベース・サーバーの場合

単位 カウンター

コーディネーター・エージェントの数を制限する場合は、 *max_coordagents* パラメーターを使用してください。

個々の追加エージェントには付加的なメモリーが必要になるので、このパラメーターはメモリーの制限がある環境でデータベース・マネージャーの合計メモリー使用率を制限するのに役立ちます。

推奨: *maxagents* の値は、少なくとも、同時にアクセスできるそれぞれのデータベース内の *maxappls* の値の合計数であることが必要です。データベースの数が *numdb* パラメーターよりも多い場合は、*numdb* と *maxappls* の最大値の積を使用するのが最も安全な方法です。

追加エージェントごとに、データベース・マネージャーの開始時に割り振られる多少のリソース・オーバーヘッドが必要になります。

データベースに接続しようとしているときにメモリーのエラーが発生した場合は、以下のように構成を調整してみてください。
- 非パーティション・データベース環境で、有効な照会内並列処理がない場合は、*maxagents* データベース構成パラメーターの値を増やす。
- パーティション・データベース環境または照会内並列処理が有効になっている環境では、*maxagents* または *max_coordagents* のいずれかで大きいほうの値を増やす。

maxcagents - 最大並行エージェント数

このパラメーターはバージョン 9.5 では推奨されませんが、バージョン 9.5 より前のデータ・サーバーおよびクライアントでは引き続き使用されています。DB2 バージョン 9.5 のデータベース・マネージャーは、この構成パラメーターに対して指定された値をすべて無視します。

注: 以下の情報は、バージョン 9.5 より前のデータ・サーバーおよびクライアントだけに当てはまります。

このパラメーターは、データベース・マネージャー・トランザクションを並行的に実行できるデータベース・マネージャー・エージェントの最大数を制限することにより、多くのアプリケーション活動が同時に実行されている間のシステムの負荷をコントロールするために使用されます。

構成タイプ
データベース・マネージャー

適用
- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- ローカル・クライアントを持つデータベース・サーバー
- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ
構成可能

デフォルト [範囲]
-1 (*max_coordagents*) [-1; 1 – *max_coordagents*]

単位 カウンター

このパラメーターは、データベースに接続できるアプリケーションの数を制限しません。これは、データベース・マネージャーが一度に並行して処理できるデータベース・マネージャー・エージェント数のみを制限しますが、これにより、処理のピーク時にシステム・リソースの使用率を制限できます。例えば、多数の接続が必要なシステムがあるのに、その接続用のメモリーに制限がある場合などです。このパラメーターの調整は、多くの活動が同時に実行される場合に、極端なオペレーティング・システム・ページングの原因となるような環境で有効です。

値が -1 の場合は、限度が *max_coordagents* であることを示します。

推奨: ほとんどの場合に、このパラメーターのデフォルト値を使用できます。多くのアプリケーションを同時に実行することによって問題が生じている場合は、ベンチマーク・テストを使用してこのパラメーターを調整し、データベースのパフォーマンスを最適化できます。

mon_heap_sz - データベース・システム・モニター・ヒープ・サイズ

このパラメーターは、データベース・システム・モニター・データに割り振られるメモリーの大きさ (ページ数) を決定します。メモリーは、スナップショットの取得、モニター・スイッチのオン、モニターのリセット、あるいはイベント・モニターの活動化といった、データベースのモニター活動を実行するときにモニター・ヒープから割り振られます。

バージョン 9.5 では、このデータベース構成パラメーターのデフォルト値は AUTOMATIC であるため、*instance_memory* 限度に達するまで、モニター・ヒープが必要に応じて増加します。

構成タイプ
データベース・マネージャー

適用

- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- ローカル・クライアントを持つデータベース・サーバー
- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ
オンラインで構成可能

デフォルト [範囲]
Automatic [0 - 60 000]

単位 ページ (4 KB)

割り振られるタイミング
データベース・マネージャーが db2start コマンドで開始されるとき

解放されるタイミング
データベース・マネージャーが db2stop コマンドで停止されたとき

ゼロの値を指定すると、データベース・マネージャーはデータベース・システム・モニター・データを収集しません。

推奨: 活動のモニターに必要なメモリーの量は、スイッチが設定されるモニター・アプリケーション (スナップショットまたはイベント・モニターを取るアプリケーション) の数と、データベース活動のレベルに依存します。

このヒープ内の構成済みメモリーが使い尽くされ、インスタンス共有メモリー領域内に使用可能な予約されていないメモリーがそれ以上ない場合、以下のいずれかが行われます。

- 最初のアプリケーションが、このイベント・モニターが定義されているデータベースに接続すると、エラー・メッセージが管理通知ログに書き込まれます。
- SET EVENT MONITOR ステートメントを使用して動的に開始しているイベント・モニターが失敗した場合は、エラー・コードがアプリケーションに戻されます。
- モニター・コマンドまたは API サブルーチンが失敗した場合は、エラー・コードがアプリケーションに戻されます。

nodetype - マシン・ノード・タイプ

このパラメーターは、マシンにインストールされている DB2 製品についての情報を提供するものであり、したがって、データベース・マネージャー構成のタイプについての情報を提供します。

構成タイプ
データベース・マネージャー

適用

- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- クライアント
- ローカル・クライアントを持つデータベース・サーバー

- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ
通知

このパラメーターによって戻される可能性のある値、および該当のノード・タイプに関連した製品が、以下に示してあります。

- **ローカルおよびリモート・クライアントを持つデータベース・サーバー** - ローカルおよびリモート・データ・サーバー・ランタイム・クライアントをサポートし、他のリモート・データベース・サーバーにアクセスできる DB2 サーバー製品。
- **クライアント** - リモート・データベース・サーバーにアクセスできるデータ・サーバー・ランタイム・クライアント。
- **ローカル・クライアントを持つデータベース・サーバー** - ローカル・データ・サーバー・ランタイム・クライアントをサポートし、他のリモート・データベース・サーバーにアクセスできる DB2 リレーショナル・データベース管理システム。
- **ローカルおよびリモート・クライアントを持つパーティション・データベース・サーバー** - ローカルおよびリモート・データ・サーバー・ランタイム・クライアントをサポートし、他のリモート・データベース・サーバーにアクセスでき、並列処理に対応可能な DB2 サーバー製品。

notifylevel - 通知レベル

このパラメーターは、管理通知ログに書き込まれる管理通知メッセージのタイプを指定します。

構成タイプ
データベース・マネージャー

適用
- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- クライアント
- ローカル・クライアントを持つデータベース・サーバー
- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ
オンラインで構成可能

伝搬クラス
即時

デフォルト [範囲]
3 [0 — 4]

UNIX プラットフォームでは、管理通知ログは、*instance*.nfy と呼ばれるテキスト・ファイルです。Windows では、管理通知メッセージはイベント・ログに書き込まれ

ます。エラーは DB2、ヘルス・モニター、キャプチャー・プログラムとアプライ・プログラム、およびユーザー・アプリケーションによって書き込むことができます。

このパラメーターの有効な値は、次のとおりです。

- **0** - 管理用通知メッセージはキャプチャーされません (この設定は推奨できません)。
- **1** - 致命的エラーまたはリカバリー不能エラー。致命的エラーまたはリカバリー不能エラーだけがログに記録されます。これらの状態の中には、リカバリーのために DB2 サービスの援助が必要なものもあります。
- **2** - 即時アクションが必要です。システム管理者やデータベース管理者による即時アテンションを要する状態がログに記録されます。状態が解決されない場合は、致命的エラーに至る恐れがあります。エラーではないが、非常に大きなアクティビティー (例えば、リカバリー) の通知もこのレベルでログに記録される場合があります。このレベルでは、ヘルス・モニター・アラームをキャプチャーします。
- **3** - 重要な情報であるが、即時アクションの必要はありません。即時アクションが必要とされるほど重大ではないが、最適なシステムではないことを示す状態がログに記録されます。このレベルでは、ヘルス・モニター・アラーム、ヘルス・モニター警告、およびヘルス・モニター・アテンションをキャプチャーします。
- **4** - 通知メッセージ。

管理通知ログには、値 *notifylevel* 以下 (この値を含む) の値をもつメッセージが組み込まれます。例えば、*notifylevel* を 3 に設定すると、レベル 1、2、および 3 に該当するメッセージが管理通知ログに組み込まれることになります。

ユーザー・アプリケーションから通知ファイルまたは Windows イベント・ログに書き込みできるようにするためには、db2AdminMsgWrite API を呼び出す必要があります。

推奨: 問題の解決に役立つ追加の問題判別データを収集する場合は、このパラメーターの値を大きくすることができます。また、ヘルス・モニターの構成で定義されている連絡先に通知を送信するためには、*notifylevel* を 2 以上の値に設定する必要があります。

num_initagents - プール内エージェントの初期数

このパラメーターは、DB2START 時にエージェント・プールで作成されるアイドル状態のエージェントの初期数を決定します。

構成タイプ

データベース・マネージャー

適用

- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- ローカル・クライアントを持つデータベース・サーバー
- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ
オンラインで構成可能

デフォルト [範囲]
0 [0–64 000]

データベース・マネージャーは、*num_initagents* 個のアイドル・エージェントを常に db2start コマンドの一部として開始します。ただし、このパラメーターの値が始動時に *num_poolagents* よりも大きく、*num_poolagents* が AUTOMATIC に設定されていない場合を除きます。この場合、データベース・マネージャーはプール可能な数を超えるアイドル・エージェントを開始する理由がないため、num_poolagents 個のアイドル・エージェントのみを開始します。

num_initfenced - fenced プロセスの初期数

このパラメーターは、DB2START 時に db2fmp プールに作成される、非スレッド化アイドル db2fmp プロセスの初期数を示します。

構成タイプ
データベース・マネージャー

適用

- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- ローカル・クライアントを持つデータベース・サーバー
- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ
オンラインで構成可能

デフォルト [範囲]
0 [0–64 000]

このパラメーターを設定すると、非スレッド・セーフ C ルーチンおよび COBOL ルーチンを実行する場合に初期起動時間が短縮されます。 *keepfenced* が指定されていない場合、このパラメーターは無視されます。

DB2START 時に多くの db2fmp プロセスを開始するよりも、*fenced_pool* をシステムにとって適切なサイズに設定することの方がはるかに重要です。

以前のバージョンでは、このパラメーターは *num_initdaris* という名前で呼ばれていました。

num_poolagents - エージェント・プール・サイズ

このパラメーターにより、アイドル・エージェント・プールの最大サイズが設定されます。

構成タイプ
データベース・マネージャー

適用

- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- ローカル・クライアントを持つデータベース・サーバー

- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ
オンラインで構成可能

デフォルト
100, Automatic [-1, 0–64 000]

この構成パラメーターはデフォルトで AUTOMATIC (値 100) に設定されます。-1 を設定することも依然としてサポートされており、値 100 に変換されます。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。通常このことは、エージェントが作業を完了すると、終了するのではなく、ある期間アイドル状態になることを意味します。エージェントのワークロードとタイプに応じて、一定時間の後に終了する場合があります。

AUTOMATIC を使用する場合、さらに num_poolagents 構成パラメーターに値を指定することができます。プールされたアイドル・エージェントの現行数が、指定した値以下である場合、追加のアイドル・エージェントは常にプールされます。

例:

num_poolagents **を 100 および AUTOMATIC に設定する**
エージェントは解放されると、アイドル・エージェント・プールに追加されます。このプールでは、ある時点でデータベース・マネージャーがエージェントを終了させるかどうかを評価します。データベース・マネージャーがエージェントの終了を検討しているときに、プールされているアイドル・エージェントの総数が 100 を超えているなら、このエージェントは終了されます。アイドル・エージェントが 100 未満の場合、アイドル・エージェントは作業待機中のままになります。 AUTOMATIC という設定を使用することにより、100 を超えた分の追加のアイドル・エージェントもプールできます。これは、システム・アクティビティーが増大して作業の頻度がより大きな規模で変動する可能性のある期間に便利です。常にアイドル・エージェントが 100 未満であると見込まれる場合、エージェントは必ずプールされます。これにより、システム・アクティビティーが少ない期間は新しい作業の始動コストが少なくなるという利点があります。

num_poolagents **を動的に構成する**
プールされたエージェントの数よりもパラメーター値が大きい値になると、その効果は即時に現れます。新しいエージェントはアイドル状態になるとプールされます。パラメーター値が小さくなっても、データベース・マネージャーはプール内のエージェントの数を直ちに減らすことはしません。むしろプール・サイズはそのままで、エージェントは、使用されて再びアイドル状態になると終了します。このようにして、プール内のエージェントの数は、次第に新しい制限まで減少します。

推奨: ほとんどの環境では、デフォルトの 0 および AUTOMATIC で十分です。特定のワークロードがあって、その中であまりにも多くのエージェントが作成され終了されていると考えられる場合は、パラメーターを AUTOMATIC に設定しておいたまま、*num_poolagents* の値を大きくすることを検討できます。

numdb - ホストおよび System i データベースを含めた並行アクティブ・データベースの最大数

このパラメーターは、並行してアクティブにできる (つまり、アプリケーションを接続できる) ローカル・データベースの数、または DB2 Connect サーバー上にカタログできる異なるデータベース別名の最大数を指定します。

構成タイプ
データベース・マネージャー

適用

- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- ローカル・クライアントを持つデータベース・サーバー
- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ
構成可能

デフォルト [範囲]

UNIX 8 [1 — 256]

ローカル・クライアントおよびリモート・クライアントを持つ Windows データベース・サーバー
8 [1 — 256]

ローカル・クライアントを持つ Windows データベース・サーバー
3 [1 — 256]

単位 カウンター

各データベースはストレージを使い果たすと、アクティブなデータベースは新しい共用メモリー・セグメントを使用します。

推奨: 通常、この値の最良の設定は、増加を見込んで、データベース・マネージャーにすでに定義されている実際のデータベース数に 10% を加えた値にすることです。

numdb パラメーターを変更すると、割り振られたメモリーの合計量に影響を与えます。したがって、このパラメーターは頻繁に変更しないことをお勧めします。このパラメーターを更新するときは、データベースやそのデータベースに接続されているアプリケーションにメモリーを割り振ることができる、他の構成パラメーターについて考慮する必要があります。

query_heap_sz - 照会ヒープ・サイズ

このパラメーターはバージョン 9.5 では推奨されませんが、バージョン 9.5 より前のデータ・サーバーおよびクライアントでは引き続き使用されています。DB2 バージョン 9.5 のデータベース・マネージャーは、この構成パラメーターに対して指定された値をすべて無視します。

注: 以下の情報は、バージョン 9.5 より前のデータ・サーバーおよびクライアントだけに当てはまります。

構成タイプ
データベース・マネージャー

適用
- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- ローカル・クライアントを持つデータベース・サーバー
- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ
構成可能

デフォルト [範囲]
1 000 [2 - 524 288]

単位 ページ (4 KB)

割り振られるタイミング
アプリケーション (ローカルまたはリモートの) がデータベースに接続するとき

解放されるタイミング
アプリケーションがデータベースから切断されたか、またはインスタンスから切断されたとき

このパラメーターは、照会ヒープに割り振ることができるメモリーの**最大**数を指定して、アプリケーションに、エージェント内で不必要に大きな仮想メモリーを消費させないようにします。

照会ヒープは、各照会をエージェントの専用メモリーに格納するために使用されます。各照会の情報は、入力と出力 SQLDA、ステートメント・テキスト、SQLCA、パッケージ名、作成者、セクション番号、および整合性トークンで構成されます。

照会ヒープは、ブロック・カーソルに割り振られるメモリー用としても使用されます。このメモリーは、カーソル・コントロール・ブロックとその内容を表現する出力 SQLDA から構成されます。

割り振られる初期照会ヒープは、*aslheapsz* パラメーターによって指定されたアプリケーション・サポート層ヒープと同じサイズです。照会ヒープのサイズは 2 以上でなければならず、*aslheapsz* パラメーター以上でなければなりません。この照会ヒープの大きさが不十分なために指定された要求を処理できない場合は、要求に必要なサイズまで (*query_heap_sz* を超えない範囲で) 再割り振りが行われます。この新しい照会ヒープが *aslheapsz* の 1.5 倍を超える大きさの場合、照会が終了したときに照会ヒープは *aslheapsz* のサイズまで再度割り振られます。

推奨: ほとんどの場合、デフォルト値が効率的です。最小値として、*query_heap_sz* を少なくとも *aslheapsz* の 5 倍より大きい値に設定してください。これにより、*aslheapsz* より大きな照会が使用可能になり、指定時にオープンされる 3 つまたは 4 つのブロック・カーソルに追加メモリーが提供されます。

とても大きな LOB を持っている場合、このパラメーターの値を増やして、照会ヒープをそれらの LOB を収容する大きさにすることが必要になる場合があります。

release - 構成ファイル・リリース・レベル

このパラメーターは、構成ファイルのリリース・レベルを指定します。

構成タイプ
データベース・マネージャー、データベース

適用

- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- ローカル・クライアントを持つデータベース・サーバー
- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ
通知

resync_interval - トランザクション再同期インターバル

このパラメーターは、トランザクション・マネージャー (TM)、リソース・マネージャー (RM)、または同期点マネージャー (SPM) が、TM、RM、または SPM で検出された未解決の未確定トランザクションのリカバリーを再試行する時間インターバルを秒数で指定します。このパラメーターは、分散作業単位 (DUOW) 環境でトランザクションを実行している場合にのみ使用できます。このパラメーターは、フェデレーテッド・データベース・システムのリカバリーにも適用されます。

構成タイプ
データベース・マネージャー

適用

- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- ローカル・クライアントを持つデータベース・サーバー
- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ
構成可能

デフォルト [範囲]
180 [1 - 60 000]

単位 秒

推奨: 現在の環境内で、未確定トランザクションがデータベースに対して実行される他のトランザクションを妨害しなければ、このパラメーターの値を増加してもかまいません。 DB2 Connect ゲートウェイを使用して DRDA2 アプリケーション・サーバーにアクセスしている場合は、たとえローカル・データ・アクセスを妨害することがないとしても、アプリケーション・サーバーで未確定トランザクションがもたらす影響を考慮する必要があります。未確定トランザクションがない場合、パフォーマンスの低下は最小限で済みます。

rqrioblk - クライアント入出力ブロック・サイズ

このパラメーターは、リモート・アプリケーションと、データベース・サーバー上のデータベース・エージェントの間の通信バッファーのサイズを指定します。ブロ

ック・カーソルがオープンされているとき、データ・サーバー・ランタイム・クライアントの入出力ブロック・サイズを決定するのにも使用されます。

構成タイプ
データベース・マネージャー

適用

- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- クライアント
- ローカル・クライアントを持つデータベース・サーバー
- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ
構成可能

デフォルト [範囲]
32 767 [4 096 - 65 535]

単位 バイト

割り振られるタイミング

- リモート・クライアント・アプリケーションが、サーバー・データベースに接続要求を発行するとき
- ブロック・カーソルがオープンされ、追加のブロックがクライアントでオープンされたとき

解放されるタイミング

- リモート・アプリケーションがサーバー・データベースから切断されたとき
- ブロック・カーソルがクローズされるとき

データ・サーバー・ランタイム・クライアントがリモート・データベースへの接続を要求すると、この通信バッファーがクライアントに割り振られます。データベース・サーバーでは、接続が確立されて、サーバーでクライアントの *rqrioblk* の値を判別できるようになるまでは、最初に 32 767 バイトの通信バッファーが割り振られます。サーバーにこの値が分かった後で、クライアントのバッファーが 32 767 バイトでない場合、サーバーは、その通信バッファーを再割り振りします。

ブロック・カーソル用のメモリーは、アプリケーションの専用アドレス・スペースから割り振られるので、それぞれのアプリケーション・プログラムごとに割り振る専用メモリーの最適量を決定する必要があります。データ・サーバー・ランタイム・クライアントがアプリケーションの専用メモリーからブロック・カーソル用のスペースを割り振れない場合は、非ブロッキング・カーソルがオープンされます。

推奨: 非ブロッキング・カーソルの場合、単一の照会ステートメントによって送信されるデータ (例えばラージ・オブジェクト・データ) が大きすぎてデフォルト値では不十分な場合などには、このパラメーターの値を増やしてください。

ブロック・カーソルの数および潜在的なサイズに対するこのパラメーターの影響についても考慮する必要があります。転送される行の数が多い、またはそのサイズが大きい場合 (例えば、データの量が 4 096 バイトより大の場合) は、行ブロックが

大きければ、パフォーマンスの向上がもたらされる可能性もあります。ただし、レコード・ブロックを大きくすると、それぞれの接続ごとの作業セット・メモリーのサイズも増大するので、トレードオフが必要になります。

また、レコード・ブロックが大きくなれば、フェッチ要求も実際にアプリケーションで必要とされる数よりも多くなる可能性もあります。フェッチ要求の数は、アプリケーションの SELECT ステートメントの OPTIMIZE FOR 節を使ってコントロールできます。

sheapthres - ソート・ヒープしきい値

このパラメーターは、専用ソートがどの時点でも使用できるメモリーの合計量についての、インスタンス単位のソフト・リミットです。インスタンスの専用ソートのメモリー使用量の合計がこの制限に達すると、追加の入力専用ソート要求に割り当てられたメモリーが相当量削減されます。

構成タイプ
データベース・マネージャー

適用

- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- ローカル・クライアントを持つデータベース・サーバー
- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー
- OLAP 関数

パラメーター・タイプ
オンラインで構成可能

伝搬クラス
即時

デフォルト [範囲]

UNIX 32 ビット・プラットフォーム
0 [0 - 2 097 152]

Windows 32 ビット・プラットフォーム
0 [0 - 2 097 152]

64 ビット・プラットフォーム
0 [0 - 2 147 483 647]

単位 ページ (4 KB)

ソート・ヒープを使用する操作の例としては、ソート、ハッシュ結合、動的ビットマップ (索引 ANDing およびスター型結合で使用される)、およびメモリー内の表の操作などがあります。

しきい値を明示的に定義すると、データベース・マネージャーが大規模なソートに対して過剰なメモリー量を使用することを防げます。

非パーティション・データベース環境からパーティション・データベース環境に移行するにあたって、このパラメーターの値を大きくする理由はありません。単一データベース・パーティション環境でデータベースおよびデータベース・マネージャ

ー構成パラメーターを調整してあれば、ほとんどの場合、同じ値がパーティション・データベース環境にも適合します。このパラメーターをノードやデータベース・パーティションごとに異なる値に設定するには、複数の DB2 インスタンスを作成するしかありません。このためには、異なるデータベース・パーティション・グループにまたがる異なる DB2 データベースの管理が必要になります。こうした方法では、パーティション・データベース環境の利点の多くの目的が達成されないことになります。

インスタンス・レベルの *sheapthres* を 0 に設定すると、ソートのメモリー使用量の追跡はデータベース・レベルでのみ行われ、ソートのメモリー割り振りはデータベース・レベルの *sheapthres_shr* 構成パラメーターの値で制限されます。

sheapthres_shr 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。

以下のいずれかの設定が該当する場合、このパラメーターは動的更新可能ではなくなります。

- *sheapthres* の開始値が 0 であり、ターゲット値が 0 以外の値である。
- *sheapthres* の開始値が 0 以外の値であり、ターゲット値が 0 である。

推奨: このパラメーターをデータベース・マネージャー・インスタンスの最大の *sortheap* パラメーターの適切な倍数に設定することが理想です。このパラメーターは、**少なくとも**インスタンス内のデータベースに定義された最大の *sortheap* の 2 倍の大きさにする必要があります。

専用ソートを実行しており、システムにメモリーの制約がない場合は、このパラメーターの理想値を以下の手順で計算することができます。

1. 以下のように各データベースの通常のソート・ヒープ使用率を計算します。

   (データベースに対して実行される並行エージェントの標準的な数)
   × (そのデータベースのために定義されるソート・ヒープ数)

2. 上記の結果の合計を計算すると、インスタンス内のすべてのデータベースに対する典型的な状況において、使用可能なソート合計ヒープが提供されます。

ソート・パフォーマンスとメモリー使用率を適切にバランスするためには、このパラメーターを調整するためにベンチマーク技法を使用する必要があります。

データベース・システム・モニターを使用すると、ポストしきい値ソート (*post_threshold_sorts*) モニター・エレメントを使用して、ソート・アクティビティーを追跡できます。

spm_log_file_sz - 同期点マネージャー・ログ・ファイル・サイズ

このパラメーターは、同期点マネージャー (SPM) ログ・ファイル・サイズを 4 KB ページ単位で識別します。

構成タイプ

データベース・マネージャー

適用

- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- ローカル・クライアントを持つデータベース・サーバー

- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ
構成可能

デフォルト [範囲]
256 [4 - 1000]

単位 ページ (4 KB)

ログ・ファイルは、sqllib の下の spmlog サブディレクトリーに含まれ、最初に SPM が始動したときに作成されます。

推奨: 同期点マネージャー・ログ・ファイル・サイズにはパフォーマンスを維持できるだけの十分な大きさが必要ですが、スペースの浪費を防げる程度の大きさにとどめておく必要もあります。必要なサイズは、同期点マネージャーを使用するトランザクションの数、および COMMIT または ROLLBACK の発行頻度によって異なります。

SPM ログ・ファイルのサイズを変更するには、以下を行ってください。

1. LIST DRDA INDOUBT TRANSACTIONS コマンドを使用して、未確定トランザクションが存在しないことを判別します。
2. 存在しない場合は、データベース・マネージャーを停止します。
3. 新しい SPM ログ・ファイル・サイズでデータベース・マネージャー構成を更新します。
4. $HOME/sqllib ディレクトリーに移動し、 rm -fr spmlog を発行して、現在の SPM ログを削除します。(注 : これは AIX コマンドを表しています。その他のシステムでは、別の除去コマンドまたは削除コマンドが必要になります。)
5. データベース・マネージャーを始動します。指定されたサイズの新しい SPM ログがデータベース・マネージャーの始動時に作成されます。

spm_log_path - 同期点マネージャー・ログ・ファイル・パス

このパラメーターは、同期点マネージャー (SPM) ログが書き込まれるディレクトリーを指定します。

構成タイプ
データベース・マネージャー

適用

- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- ローカル・クライアントを持つデータベース・サーバー
- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ
構成可能

デフォルト [範囲]
sqllib/spmlog [有効なパスまたは装置]

デフォルトでは、ログは sqllib/spmlog ディレクトリーに書き込まれます。このディレクトリーは、大量のトランザクションを持つ環境では、入出力障害の原因になる可能性があります。このパラメーターを使用して、SPM ログ・ファイルが現在の sqllib/spmlog ディレクトリーではなく高速ディスクに置かれるようにします。これにより、SPM エージェント間の並列処理が高くなります。

spm_max_resync - 同期点マネージャー再同期エージェント数の限度

このパラメーターは、再同期操作を同時に実行できるエージェントの数を指定します。

構成タイプ
データベース・マネージャー

適用

- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- ローカル・クライアントを持つデータベース・サーバー
- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ
構成可能

デフォルト [範囲]
20 [10 — 256]

spm_name - 同期点マネージャー名

このパラメーターは、データベース・マネージャーに対して同期点マネージャー (SPM) インスタンスの名前を識別します。

構成タイプ
データベース・マネージャー

適用

- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- ローカル・クライアントを持つデータベース・サーバー
- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ
構成可能

デフォルト
TCP/IP ホスト名から派生

srvcon_auth - サーバーでの着信接続の認証タイプ

このパラメーターでは、サーバーで着信接続を処理する際に、ユーザー認証を実行する方法と場所を指定します。このパラメーターは現行の認証タイプをオーバーライドします。

構成タイプ
データベース・マネージャー

適用
- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- ローカル・クライアントを持つデータベース・サーバー
- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ
構成可能

デフォルト [範囲]
Null [CLIENT; SERVER; SERVER_ENCRYPT; KERBEROS; KRB_SERVER_ENCRYPT; GSSPLUGIN; GSS_SERVER_ENCRYPT]

値を指定しない場合、DB2 は *authentication* データベース・マネージャー構成パラメーターの値を使用します。

各認証タイプについては、 527 ページの『authentication - 認証タイプ』を参照してください。

srvcon_gssplugin_list - サーバーでの着信接続用の GSS API プラグインのリスト

このパラメーターでは、データベース・サーバーがサポートする GSS API プラグイン・ライブラリーを指定します。これは、*srvcon_auth* パラメーターに KERBEROS、KRB_SERVER_ENCRYPT、GSSPLUGIN、または GSS_SERVER_ENCRYPT が指定されている場合、または *srvcon_auth* が指定されておらず、authentication に KERBEROS、KRB_SERVER_ENCRYPT、GSSPLUGINまたは GSS_SERVER_ENCRYPT が指定されている場合に、サーバーでの着信接続を処理します。

構成タイプ
データベース・マネージャー

適用
- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- ローカル・クライアントを持つデータベース・サーバー
- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ
構成可能

デフォルト [範囲]
Null [任意の有効ストリング]

デフォルトでは、この値は Null です。認証タイプが GSSPLUGIN で、このパラメーターが NULL であると、エラーが戻されます。認証タイプが KERBEROS で、

このパラメーターが NULL であると、 DB2 提供の Kerberos モジュールまたはライブラリーが使用されます。別の認証タイプが使用されている場合には、このパラメーターは使用されません。

認証タイプが KERBEROS であり、このパラメーターの値が NULL ではない場合、リストに含まれる Kerberos プラグインは 1 つでなければならず、このプラグインが認証に使用されます (リスト内の他のすべての GSS プラグインは無視されます)。複数の Kerberos プラグインがある場合は、エラーが戻されます。

GSS API プラグイン名どうしはコンマ (,) で区切ります。コンマの前後にスペースは不要です。プラグイン名は優先する順序にリストします。

srvcon_pw_plugin - サーバーでの着信接続用のユーザー ID-パスワード・プラグイン

このパラメーターでは、サーバー側の認証に使用するデフォルトのユーザー ID-パスワード・プラグイン・ライブラリーの名前を指定します。このパラメーターは、*srvcon_auth* パラメーターに CLIENT、SERVER、SERVER_ENCRYPT、または DATA_ENCRYPT が指定されている場合、または *srvcon_auth* が指定されておらず *authentication* に CLIENT、SERVER、SERVER_ENCRYPT、または、DATA_ENCRYPT が指定されている場合に、サーバーでの着信接続を処理します。

構成タイプ
データベース・マネージャー

適用

- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- ローカル・クライアントを持つデータベース・サーバー
- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ
構成可能

デフォルト [範囲]
Null [任意の有効ストリング]

デフォルトの値は NULL で、DB2 提供のユーザー ID-パスワード・プラグイン・ライブラリーが使用されます。このプラグインはすべてのグループ参照数に使用されます。非 root インストールについては、DB2 のユーザー ID およびパスワード・プラグイン・ライブラリーが使用される場合は、DB2 製品を使用する前に、db2rfe コマンドを実行する必要があります。

srv_plugin_mode - サーバー・プラグイン・モード

このパラメーターでは、fenced モードまたは unfenced モードでプラグインを実行するかどうかを指定します。 unfenced モードのみがサポートされています。

構成タイプ
データベース・マネージャー

適用

- ローカルおよびリモート・クライアントを持つデータベース・サーバー

- ローカル・クライアントを持つデータベース・サーバー
- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ
構成可能

デフォルト [範囲]
UNFENCED

start_stop_time - タイムアウトの開始および停止

このパラメーターは分単位で指定します。この範囲内にすべてのデータベース・パーティション・サーバーが DB2START または DB2STOP コマンドに応答する必要があります。また、ADD DBPARTITIONNUM 操作時にタイムアウト値としても使用されます。

構成タイプ
データベース・マネージャー

適用 ローカルおよびリモート・クライアントを持つデータベース・サーバー

パラメーター・タイプ
オンラインで構成可能

伝搬クラス
即時

デフォルト [範囲]
10 [1 - 1 440]

単位 分

指定された時間内に DB2START コマンドに応答しないデータベース・パーティション・サーバーは、インスタンスのホーム・ディレクトリーの `sqllib` サブディレクトリーの `log` サブディレクトリーにある `db2start` エラー・ログにメッセージを送信します。それらを再始動する前に、これらのノードで DB2STOP を出す必要があります。

指定された時間内に DB2STOP コマンドに応答しないデータベース・パーティション・サーバーは、インスタンスのホーム・ディレクトリーの `sqllib` サブディレクトリーの `log` サブディレクトリーにある `db2stop` エラー・ログにメッセージを送信します。応答しないそれぞれのデータベース・パーティション・サーバーごと、またはそのすべてに対して、`db2stop` を出すことができます。 (すでに停止しているサーバーは、停止していることを示す記述を返します。)

複数パーティション・データベースにおける `db2start` または `db2stop` 操作が、*start_stop_time* データベース・マネージャー構成パラメーターにより指定された値以内で完了しなかった場合、タイムアウトとなったデータベース・パーティションは内部で強制終了します。 *start_stop_time* の値が低いデータベース・パーティションが多数ある環境では、この動作が発生する可能性があります。この動作による問題を解決するには、*start_stop_time* にもっと大きい値を指定してください。

DB2START、START DATABASE MANAGER、ADD DBPARTITIONNUM のいずれかのコマンドを使用して新規データベース・パーティションを追加する場合、デー

タベース・パーティション追加操作で、インスタンス内の各データベースの自動ストレージが使用可能かどうかを判別する必要があります。これは、各データベースのカタログ・パーティションと通信することで行います。自動ストレージが使用可能な場合、ストレージ・パスの定義はその通信の中で取得されます。同様に、データベース・パーティションを使用して SYSTEM TEMPORARY 表スペースを作成する操作でも、別のデータベース・パーティション・サーバーと通信して、そのサーバーにあるデータベース・パーティションの表スペース定義を取得する必要がある場合があります。これらの要素は、*start_stop_time* パラメーターの値を決定するときに考慮する必要があります。

svcename - TCP/IP サービス名

このパラメーターには、データベース・サーバーがリモート・クライアント・ノードからの通信を待つために使用する、 TCP/IP ポートの名前が含まれています。この名前は、データベース・マネージャーが使用するために予約されています。

構成タイプ
データベース・マネージャー

適用

- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- ローカル・クライアントを持つデータベース・サーバー
- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ
構成可能

デフォルト
Null

TCP/IP を使用して、データベース・クライアントからの接続要求を受け入れるためには、データベース・サーバーがそのサーバーに指定されたポートで待機中でなければなりません。データベース・サーバーのシステム管理者は、ポートを予約して (番号 *n*)、サーバーのサービス・ファイルにその関連する TCP/IP サービス名を定義する必要があります。

データベース・サーバー・ポート (番号 *n*) とその TCP/IP サービス名は、データベース・クライアントのサービス・ファイルに定義する必要があります。

Linux および UNIX システムでは、このサービス・ファイルは `/etc/services` にあります。

svcename パラメーターは、データベース・サーバーの始動時に、着信接続要求を listen するポートを判別できるように、主接続ポートに関連したサービス名に設定する必要があります。

sysadm_group - システム管理権限グループ名

このパラメーターは、データベース・マネージャー・インスタンスの SYSADM 権限を持つグループ名を定義します。

構成タイプ
データベース・マネージャー

適用
- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- クライアント
- ローカル・クライアントを持つデータベース・サーバー
- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ
構成可能

デフォルト
Null

システム管理 (SYSADM) 権限はデータベース・マネージャー内の最高レベルの権限であり、すべてのデータベース・オブジェクトをコントロールします。

SYSADM 権限は、特定の運用環境で使用されるセキュリティー機能によって決定されます。

- Windows オペレーティング・システムでは、このパラメーターは、任意のローカル・グループに設定することができ、Windows セキュリティー・データベースで定義されます。グループ名は、SQL および XML の制限値で指定された長さの限度に従う必要があります。このパラメーターに「NULL」を指定する場合、管理者グループのすべてのメンバーは SYSADM 権限を持っています。
- Linux および UNIX システムでは、このパラメーターの値として「NULL」を指定する場合、SYSADM グループのデフォルトは、インスタンス所有者の 1 次グループです。

  値が「NULL」でない場合、SYSADM グループは有効な UNIX グループ名です。

パラメーターをそのデフォルト (NULL) 値にリストアするため UPDATE DBM CFG USING SYSADM_GROUP NULL を使用してください。 キーワード「NULL」は大文字で指定する必要があります。

sysctrl_group - システム制御権限グループ名

このパラメーターは、システム制御権限 (SYSCTRL) を持つグループ名を定義します。 SYSCTRL には、システム・リソースに影響を与える操作を許可する特権がありますが、データへの直接アクセスはできません。

構成タイプ
データベース・マネージャー

適用
- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- クライアント
- ローカル・クライアントを持つデータベース・サーバー

- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ
構成可能

デフォルト
Null

すべてのプラットフォームで、グループ名は、SQL および XML の制限値で指定された長さの限度に従っている限り、受け入れられます。

警告: システム・セキュリティーが使用される (すなわち、認証が CLIENT、SERVER、DCS、またはほかの有効な認証である) 場合、 Windows クライアントでは、このパラメーターは Null にする必要があります。これは、Windows オペレーティング・システムはグループ情報を保管しないため、ユーザーが、指定された SYSCTRL グループのメンバーであるかどうかを判別する方法がないからです。グループ名が指定される場合、ユーザーはそのメンバーにはなれません。

パラメーターをそのデフォルト (NULL) 値にリストアするため UPDATE DBM CFG USING SYSCTRL_GROUP NULL を使用してください。キーワード「NULL」は大文字で指定する必要があります。

sysmaint_group - システム保守権限グループ名

このパラメーターは、システム保守 (SYSMAINT) 権限を持つグループ名を定義します。

構成タイプ
データベース・マネージャー

適用

- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- クライアント
- ローカル・クライアントを持つデータベース・サーバー
- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ
構成可能

デフォルト
Null

SYSMAINT は、データに直接アクセスすることなしに、インスタンスに関連するすべてのデータベースでの保守操作を実行する特権を持っています。

すべてのプラットフォームで、グループ名は、SQL および XML の制限値で指定された長さの限度に従っている限り、受け入れられます。

警告: システム・セキュリティーが使用される (すなわち、認証が CLIENT、SERVER、DCS、またはほかの有効な認証である) 場合、 Windows クライアントでは、このパラメーターは Null にする必要があります。これは、Windows

オペレーティング・システムはグループ情報を保管しないため、ユーザーが、指定された SYSMAINT グループのメンバーであるかどうかを判別する方法がないからです。グループ名が指定される場合、ユーザーはそのメンバーにはなれません。

パラメーターをそのデフォルト (NULL) 値にリストアするため UPDATE DBM CFG USING SYSMAINT_GROUP NULL を使用してください。キーワード「NULL」は大文字で指定する必要があります。

sysmon_group - システム・モニター権限グループ名

このパラメーターは、システム・モニター (SYSMON) 権限を持つグループ名を定義します。

構成タイプ

データベース・マネージャー

適用

- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- クライアント
- ローカル・クライアントを持つデータベース・サーバー
- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ

構成可能

デフォルト

Null

インスタンス・レベルで SYSMON 権限を持つユーザーは、データベース・マネージャー・インスタンスまたはそのデータベースのデータベース・システム・モニター・スナップショットを取ることができます。 SYSMON 権限では、以下のコマンドを使用できます。

- GET DATABASE MANAGER MONITOR SWITCHES
- GET MONITOR SWITCHES
- GET SNAPSHOT
- LIST ACTIVE DATABASES
- LIST APPLICATIONS
- LIST DCS APPLICATIONS
- RESET MONITOR
- UPDATE MONITOR SWITCHES

ユーザーは SYSADM、SYSCTRL、または SYSMAINT 権限を持つと自動的に、データベース・システム・モニター・スナップショットを取ることも、上記のコマンドを使用することもできます。

すべてのプラットフォームで、グループ名は、SQL および XML の制限値で指定された長さの限度に従っている限り、受け入れられます。

パラメーターをそのデフォルト (NULL) 値にリストアするため UPDATE DBM CFG USING SYSMON_GROUP NULL を使用してください。キーワード「NULL」は大文字で指定する必要があります。

tm_database - トランザクション・マネージャー・データベース名

このパラメーターは、各 DB2 インスタンスのトランザクション・マネージャー (TM) データベースの名前を識別します。

構成タイプ
データベース・マネージャー

適用

- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- クライアント
- ローカル・クライアントを持つデータベース・サーバー
- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ
構成可能

デフォルト [範囲]
1ST_CONN [任意の有効なデータベース名]

TM データベースは、次のいずれかです。

- ローカル DB2 データベース
- ホストまたは AS/400 システムにはないリモート DB2 データベース
- DB2 for OS/390 V5 データベース (TCP/IP を介してアクセスされ、同期点マネージャー (SPM) が使用されない場合)

TM データベースとは、ロガーおよびコーディネーターとして使用され、未確定トランザクションのリカバリーを行うために使用されるデータベースのことです。

このパラメーターを **1ST_CONN** に設定すると、TM データベースを、ユーザーが最初に接続する最初のデータベースにすることができます。

推奨: 管理および運用を単純化するために、多数のインスタンスに対して少数のデータベースを作成し、これらのデータベースを TM データベースとして排他的に使用することができます。

tp_mon_name - トランザクション・プロセッサー・モニター名

このパラメーターでは、使用されているトランザクション処理 (TP) モニター製品の名前を識別します。

構成タイプ
データベース・マネージャー

適用

- ローカルおよびリモート・クライアントを持つデータベース・サーバー

- クライアント
- ローカル・クライアントを持つデータベース・サーバー
- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ
構成可能

デフォルト
デフォルトなし

有効な値

- CICS®
- MQ
- ENCINA
- CB
- SF
- TUXEDO
- TOPEND
- ブランクまたは何か別の値 (ただし、UNIX および Windows の場合であり、Solaris または SINIX の場合は、他に指定できる値はない)

- アプリケーションが WebSphere Enterprise Server Edition CICS 環境で実行される場合は、このパラメーターは「CICS」に設定する必要があります。
- アプリケーションが WebSphere Enterprise Server Edition Encina® 環境で実行される場合は、このパラメーターは「ENCINA」に設定する必要があります。
- アプリケーションが WebSphere Enterprise Server Edition Component Broker 環境で実行される場合は、このパラメーターは「CB」に設定する必要があります。
- アプリケーションが IBM MQSeries® 環境で実行される場合は、このパラメーターは「MQ」に設定する必要があります。
- アプリケーションが BEA Tuxedo 環境で実行される場合は、このパラメーターは「TUXEDO」に設定する必要があります。
- アプリケーションが IBM San Francisco 環境で実行される場合は、このパラメーターは「SF」に設定する必要があります。

IBM WebSphere EJB および **Microsoft Transaction Server** のユーザーは、このパラメーターの値を構成する必要はありません。

上記の製品のいずれも使用されていない場合は、このパラメーターは構成しないで、ブランクのままにしておく必要があります。

Windows 上の以前のバージョンの IBM DB2 では、XA トランザクション・マネージャーの関数 *ax_reg* および *ax_unreg* が入っていた DLL のパスおよび名前が、このパラメーターに含まれていました。このフォーマットは、まだサポートされています。このパラメーターの値が上記の TP モニター名のいずれにも一致しない場合、値は、*ax_reg* および *ax_unreg* 関数が入っているライブラリー名であると見なされます。UNIX および Windows 環境には、これが該当します。

TXSeries® CICS および Encina のユーザー: 以前のバージョンのこの製品の Windows 版では、このパラメーターを「libEncServer:C」または「libEncServer:E」として構成する必要がありました。これはまだサポートされてはいますが、必須ではなくなりました。パラメーターを「CICS」または「ENCINA」として構成すれば十分です。

MQSeries ユーザー: 以前のバージョンのこの製品の Windows 版では、このパラメーターを「mqmax」として構成する必要がありました。これはまだサポートされてはいますが、必須ではなくなりました。パラメーターを「MQ」として構成すれば十分です。

Component Broker ユーザー: 以前のバージョンのこの製品の Windows 版では、このパラメーターを「somtrx1i」として構成する必要がありました。これはまだサポートされてはいますが、必須ではなくなりました。パラメーターを「CB」として構成すれば十分です。

San Francisco ユーザー: 以前のバージョンのこの製品の Windows 版では、このパラメーターを「ibmsfDB2」として構成する必要がありました。これはまだサポートされてはいますが、必須ではなくなりました。パラメーターを「SF」として構成すれば十分です。

このパラメーターに指定できるストリングの最大長は 19 文字です。

このパラメーターは、IBM DB2 バージョン 9.1 の XA OPEN ストリングで構成することも可能です。複数のトランザクション処理モニターで単一の DB2 インスタンスを使用している場合は、この機能を使用する必要があります。

trust_allclnts - 全クライアントのトラステッド化

このパラメーターと *trust_clntauth* は、データベース環境に対するユーザーの妥当性が検証される場所を決定する場合に使用されます。

構成タイプ

データベース・マネージャー

適用

- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- ローカル・クライアントを持つデータベース・サーバー
- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ

構成可能

デフォルト [範囲]

YES [NO, YES, DRDAONLY]

このパラメーターは、*authentication* パラメーターが CLIENT に設定されている場合のみ、アクティブになります。

このパラメーターのデフォルト「YES」を受け入れると、すべてのクライアントがトラステッド・クライアントとして扱われます。これは、セキュリティーのレベル

をクライアントで利用できることをサーバーが前提にしていること、およびクライアントでユーザーを有効にできる可能性を示しています。

このパラメーターは、*authentication* パラメーターが CLIENT に設定されている場合、「NO」にしか変更できません。このパラメーターが「NO」に設定されている場合、非トラステッド・クライアントは、サーバーに接続するときにユーザー ID とパスワードの組み合わせを指定する必要があります。非トラステッド・クライアントは、認証ユーザーのセキュリティー・サブシステムを持たないオペレーティング・システム・プラットフォームです。

このパラメーターを「DRDAONLY」に設定すると、DB2 for OS/390 and z/OS、DB2 for VM and VSE、および DB2 for OS/400® からのクライアントを除く、すべてのクライアントに対して保護されます。上記のクライアントだけを、クライアント側の認証を行うよう承認することができます。他のすべてのクライアントには、サーバーによって認証されているユーザー ID とパスワードが必要です。

trust_allclnts が「DRDAONLY」に設定されていると、*trust_clntauth* パラメーターは、クライアントが認証される場所を判別するために使用されます。 *trust_clntauth* が「CLIENT」に設定されている場合、認証がクライアントで発生します。*trust_clntauth* が「SERVER」に設定されている場合、認証がクライアントで発生するのは、パスワードが指定されていない場合であり、パスワードが指定されている場合は、サーバーで発生します。

trust_clntauth - トラステッド・クライアント認証

このパラメーターでは、クライアントが接続用のユーザー ID とパスワードの組み合わせを指定したときにトラステッド・クライアントがサーバーまたはクライアントで認証されているかどうかを指定します。このパラメーター (および *trust_allclnts*) は、*authentication* パラメーターが CLIENT に設定されている場合にのみアクティブです。ユーザー ID とパスワードが指定されていない場合、ユーザーは有効であるとクライアントは仮定し、サーバーでそれ以上の妥当性検査は行われません。

構成タイプ
データベース・マネージャー

適用

- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- ローカル・クライアントを持つデータベース・サーバー
- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ
構成可能

デフォルト [範囲]
CLIENT [CLIENT, SERVER]

このパラメーターが CLIENT (デフォルト) に設定されている場合は、トラステッド・クライアントは、ユーザー ID とパスワードの組み合わせを示さなくても接続でき、オペレーティング・システムがユーザーをすでに認証していると想定されま

す。このパラメーターが SERVER に設定されている場合は、ユーザー ID およびパスワードの妥当性がサーバーで検査されます。

CLIENT の数値は 0 で、SERVER の数値は 1 です。

util_impact_lim - インスタンス影響ポリシー

データベース管理者 (DBA) は、このパラメーターを使用して、ワークロードに対するスロットル・ユーティリティーの性能の低下を制限することができます。

構成タイプ
データベース・マネージャー

適用

- ローカル・クライアントを持つデータベース・サーバー
- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ
オンラインで構成可能

伝搬クラス
即時

デフォルト [範囲]
10 [1 - 100]

単位 許可するワークロードへの影響のパーセンテージ

パフォーマンスの低下が制限されている場合、DBA は、重要な実動期間中にオンライン・ユーティリティーを実行し、実動作業のパフォーマンスへの影響が許容限度を超えないようにすることができます。

例えば、DBA で *util_impact_lim* (影響ポリシー) の値を 10 に設定した場合、スロットルされたバックアップの起動は、ワークロードに 10% を超える影響を与えることはありません。

util_impact_lim が 100 になっていると、ユーティリティーの起動はスロットルされません。この場合、ユーティリティーは、ワークロードに対して任意の影響を与えることがあります (好ましくない)。 *util_impact_lim* を 100 以下の値に設定すると、ユーティリティーをスロットル・モードで起動できます。また、スロットル・モードで実行する場合は、ユーティリティーをゼロ以外の優先順位で起動する必要があります。

推奨事項: 多くの場合、*util_impact_lim* は低い値 (例えば、1 から 10 程度) に設定したほうが益があります。

スロットルされたユーティリティーは、通常、スロットルされていないユーティリティーよりも完了に時間がかかります。ユーティリティーの実行に極端に長い時間がかかっている場合は、*util_impact_lim* の値を増やすか、*util_impact_lim* を 100 にして、スロットルを完全に無効にしてください。

wlm_collect_int - ワークロード管理収集間隔構成パラメーター

このパラメーターは、ワークロード管理 (WLM) 統計の収集およびリセットの間隔を分単位で指定します。

x *wlm_collect_int* 分ごと (x は *wlm_collect_int* パラメーターの値) に、すべてのワークロード管理統計が収集されて、任意のアクティブな統計イベント・モニターに送信され、その後で統計がリセットされます。アクティブなイベント・モニターが存在する場合は、それが作成された方法によって、統計はファイルか表のいずれかに書き込まれます。アクティブなイベント・モニターが存在しない場合は、統計はリセットのみされて、収集はされません。

収集とリセットのプロセスは、カタログ・パーティションから開始されます。カタログ・パーティションでは、*wlm_collect_int* パラメーターを指定する必要があります。これは、他のパーティションでは使用されません。

構成タイプ
データベース

パラメーター・タイプ
オンラインで構成可能

デフォルト [範囲]
0 [0 (収集は実行されない), 5 - 32 767]

統計イベント・モニターにより収集されるワークロード管理統計は、システムの短期および長期の動作をモニターするために使用できます。短い間隔を使用して、システムの短期および長期の動作をモニターすることができます。これは、その結果をマージすると、長期の動作を取得できるためです。ただし、異なる間隔で得られた結果を手動でマージすると、分析が複雑になります。短い間隔の統計が必要なければ、オーバーヘッドが無用に増大するだけです。したがって、長期の動作の分析だけで十分な場合は、短期の動作をキャプチャーする間隔を下げ、オーバーヘッドを削減する間隔を上げます。

この間隔は SQL 要求、コマンドの呼び出し、またはアプリケーションごとではなく、データベースごとにカスタマイズする必要があります。他の構成パラメーターを考慮する必要はありません。

注: すべての WLM 統計表関数は、統計が前回リセットされてから累積した統計を戻します。この統計は、この構成パラメーターで指定された間隔で定期的にリセットされます。

データベース 構成パラメーター

alt_collate - 代替照合シーケンス

このパラメーターでは、非ユニコード・データベース内のユニコード表に使用する照合シーケンスを指定します。

構成タイプ
データベース

適用

- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- クライアント
- ローカル・クライアントを持つデータベース・サーバー
- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ
構成可能

デフォルト [範囲]
Null [IDENTITY_16BIT]

このパラメーターを設定しないうちは、ユニコードの表とルーチンを非ユニコード・データベース内に作成できません。このパラメーターを一度設定すると、変更もリセットもできません。

このパラメーターはユニコード・データベースには設定できません。

app_ctl_heap_sz - アプリケーション制御ヒープ・サイズ

このパラメーターはバージョン 9.5 では推奨されませんが、バージョン 9.5 より前のデータ・サーバーおよびクライアントでは引き続き使用されています。DB2 バージョン 9.5 のデータベース・マネージャーは、この構成パラメーターに対して指定された値をすべて無視します。バージョン 9.5 では、このパラメーターは *appl_memory* 構成パラメーターに置き換えられています。

注: 以下の情報は、バージョン 9.5 より前のデータ・サーバーおよびクライアントだけに当てはまります。

パーティション・データベースの場合、およびパーティション内並列処理が使用可能 (intra_parallel=ON) な非パーティション・データベースの場合は、このパラメーターは、アプリケーション用として割り振られる共有メモリー領域の平均サイズを指定します。パーティション内並列処理が使用不可 (intra_parallel=OFF) の非パーティション・データベースの場合は、これはヒープとして割り振られる最大専用メモリー・サイズです。それぞれのデータベース・パーティションごとに 1 つの接続に 1 つずつアプリケーション・コントロール・ヒープがあります。

構成タイプ
データベース

パラメーター・タイプ
構成可能

デフォルト [範囲]

ローカルおよびリモート・クライアントを持つデータベース・サーバー

- 128 [1 - 64 000] INTRA_PARALLEL が無効のとき
- 512 [1 - 64 000] INTRA_PARALLEL が有効のとき

ローカル・クライアントを持つデータベース・サーバー

- 64 [1 - 64 000] (UNIX 以外のプラットフォームの場合) INTRA_PARALLEL が無効のとき

- 512 [1 - 64 000] (UNIX 以外のプラットフォームの場合) INTRA_PARALLEL が有効のとき
- 128 [1 - 64 000] (Linux および UNIX プラットフォームの場合) INTRA_PARALLEL が無効のとき
- 512 [1 - 64 000] (Linux および UNIX プラットフォームの場合) INTRA_PARALLEL が有効のとき

ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

512 [1 - 64 000]

単位 ページ (4 KB)

割り振られるタイミング

アプリケーション開始時期

解放されるタイミング

アプリケーション完了時期

アプリケーション・コントロール・ヒープが必要なのは、主として、同一要求のために作動するエージェント間で情報を共用するためです。このヒープの使用量が最小になるのは、非パーティション・データベースで、並列処理の多重度が 1 で照会を実行しているときです。

このヒープは、宣言済み一時表の記述子情報を保管することにも使用されます。明示的にドロップされていないすべての宣言済み一時表の記述子情報はこのヒープのメモリーに保持され、宣言済み一時表がドロップされるまでドロップすることはできません。

推奨: 最初はデフォルト値で開始してください。複雑なアプリケーションを実行する場合、多数のデータベース・パーティションを含むシステムである場合、または宣言済み一時表を使用する場合は、値をより高く設定しなければならないことがあります。必要とされるメモリーの量は、同時にアクティブである宣言済み一時表の数とともに増加します。多くの列を持つ宣言済み一時表の表記述子サイズは、少ない列を持つ表のものよりも大きくなります。そのため、アプリケーションの宣言済み一時表が多数の列を持つと、要求されるアプリケーション・コントロール・ヒープも増えます。

appgroup_mem_sz - アプリケーション・グループ・メモリー・セットの最大サイズ

このパラメーターはバージョン 9.5 では推奨されませんが、バージョン 9.5 より前のデータ・サーバーおよびクライアントでは引き続き使用されています。DB2 バージョン 9.5 のデータベース・マネージャーは、この構成パラメーターに対して指定された値をすべて無視します。バージョン 9.5 では、このパラメーターは *appl_memory* 構成パラメーターに置き換えられています。

注: 以下の情報は、バージョン 9.5 より前のデータ・サーバーおよびクライアントだけに当てはまります。

このパラメーターでは、アプリケーション・グループ共用メモリー・セグメントのサイズを決定します。

構成タイプ
データベース

パラメーター・タイプ
構成可能

デフォルト [範囲]

ローカル・クライアントを持つ UNIX データベース・サーバー (32 ビット HP-UX を除く)
20 000 [1 - 1 000 000]

32 ビット HP-UX

- ローカル・クライアントを持つデータベース・サーバー
- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

10 000 [1 - 1 000 000]

ローカル・クライアントを持つ Windows データベース・サーバー
10 000 [1 - 1 000 000]

ローカル・クライアントおよびリモート・クライアントを持つデータベース・サーバー (32 ビット HP-UX を除く)
30 000 [1 - 1 000 000]

ローカル・クライアントおよびリモート・クライアントを持つパーティション化されたデータベース・サーバー (32 ビット HP-UX を除く)
40 000 [1 - 1 000 000]

単位 ページ (4 KB)

同じアプリケーションを使用するエージェント間で共有する必要がある情報は、アプリケーション・グループ共用メモリー・セグメントに保管されます。

パーティション・データベース、またはパーティション内並列処理が使用可能か、コンセントレーターが使用可能の非パーティション・データベースでは、複数のアプリケーションが 1 つのアプリケーション・グループを共有します。アプリケーション・グループ共用メモリー・セグメントが 1 つ、アプリケーション・グループに割り振られます。アプリケーション・グループ共用メモリー・セグメント内では、各アプリケーションにそれぞれ固有のアプリケーション・コントロール・ヒープがあり、すべてのアプリケーションで 1 つのアプリケーション・グループ共有ヒープを共有します。

1 つのアプリケーション・グループ内のアプリケーションの数は、次のようにして計算されます。

```
appgroup_mem_sz / app_ctl_heap_sz
```

アプリケーション・グループ共有ヒープ・サイズは、次のようにして計算されます。

```
appgroup_mem_sz * groupheap_ratio / 100
```

各アプリケーション・コントロール・ヒープのサイズは、次のようにして計算されます。

```
app_ctl_heap_sz * (100 - groupheap_ratio) / 100
```

推奨: パフォーマンス上の問題が検出されない限り、このパラメーターのデフォルト値を変更しないでください。

appl_memory - アプリケーション・メモリー構成パラメーター

このパラメーターにより、DBA および ISV がアプリケーション要求を保守するため、DB2 データベース・エージェントにより割り振られるアプリケーション・メモリーの最大量を制御できるようになります。デフォルトでは、この値は AUTOMATIC に設定されており、データベース・パーティションにより割り振られたメモリーの総量が、*instance_memory* 限度内である限り、すべてのアプリケーション・メモリー要求が許可されます。

構成タイプ
データベース

適用

- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- ローカル・クライアントを持つデータベース・サーバー
- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ
オンラインで構成可能

デフォルト [範囲]
Automatic [128 - 4 294 967 295]

単位 ページ (4 KB)

割り振られるタイミング
データベース・アクティベーション時

解放されるタイミング
データベース非アクティベーション時

注: *appl_memory* が AUTOMATIC に設定されている場合、データベース・アクティベーション時の初期アプリケーション・メモリー割り振りは最小ですが、必要に応じて増加 (もしくは減少) します。 *appl_memory* が特定の値に設定されている場合、要求されたメモリーの量はデータベース・アクティベーション時に最初に割り振られ、アプリケーション・メモリーのサイズは変わりません。初期アプリケーション・メモリー量がオペレーティング・システムから割り振られない場合や、*instance_memory* 限度を超える場合は、データベース・アクティベーションは失敗し、SQL1084C エラー (共有メモリー・セグメントを割り振れない) が戻されます。

applheapsz - アプリケーション・ヒープ・サイズ

以前のリリースでは、*applheapsz* データベース構成パラメーターは、アプリケーションに対して動作する各データベース・エージェントが個別に消費するアプリケーション・メモリーの量を参照していました。バージョン 9.5 では、*applheapsz* はアプリケーション全体で消費されるアプリケーション・メモリーの総量を参照しま

す。DPF、コンセントレーター、または SMP 構成では、以前のリリースで使用されていた *applheapsz* の値は、AUTOMATIC 設定が使用されているのでない限り、同様の作業負荷においては増やす必要が生じます。

バージョン 9.5 では、このデータベース構成パラメーターのデフォルト値は AUTOMATIC であるため、*appl_memory* 限度または *instance_memory* 限度のいずれかに達するまで、必要に応じて増加します。

構成タイプ
データベース

パラメーター・タイプ
オンラインで構成可能

デフォルト [範囲]
Automatic [16 - 60 000]

単位 ページ (4 KB)

割り振られるタイミング
アプリケーションがデータベースに関連付けられたとき、またはデータベースに接続したとき。

解放されるタイミング
アプリケーションとデータベースの関連性が解除されたとき、またはデータベースから切断されたとき。

注: このパラメーターにより、アプリケーション・ヒープの最大サイズが定義されます。アプリケーションがデータベースに最初に接続した時点で、データベース・アプリケーションごとにアプリケーション・ヒープが 1 つ割り振られます。そのヒープは、そのアプリケーションに対して動作するすべてのデータベース・エージェントで共有されます。(以前のリリースでは、各データベース・エージェントにそれぞれのアプリケーション・ヒープが割り振られていました。)メモリーは、このパラメーターで指定した限度まで、アプリケーションを処理する必要に応じて、アプリケーション・ヒープから割り振られます。AUTOMATIC に設定しているときは、データベースの *appl_memory* 限度またはデータベース・パーティションの *instance_memory* 限度のいずれかまで、アプリケーション・ヒープが必要に応じて増量して割り振られます。アプリケーションがデータベースから切断されると、アプリケーション・ヒープ全体が解放されます。

オンラインで変更された値は、アプリケーション接続境界で有効になります。つまりこの値が動的に変更された後、現在接続しているアプリケーションは古い値を使用しますが、新しく接続されたアプリケーションはすべて新しい値を使用します。

archretrydelay - エラー時のアーカイブ再試行遅延

このパラメーターでは、ログ・ファイルのアーカイブを試行して失敗してから、再びアーカイブを試行するまでに待機する秒数を指定します。

構成タイプ
データベース

適用

- ローカルおよびリモート・クライアントを持つデータベース・サーバー

- クライアント
- ローカル・クライアントを持つデータベース・サーバー
- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ
オンラインで構成可能

デフォルト [範囲]
20 [0 - 65 535]

再試行が有効であるためには、 *numarchretry* データベース構成パラメーターの値が少なくとも 1 でなければなりません。

auto_del_rec_obj - リカバリー・オブジェクトの自動削除構成パラメーター

このパラメーターは、データベース・ログ・ファイル、バックアップ・イメージ、およびロード・コピー・イメージを、それらに関連するリカバリー履歴ファイルの項目が整理されるときに、削除するかどうかを指定します。

構成タイプ
データベース

パラメーター・タイプ
オンラインで構成可能

伝搬クラス
即時

デフォルト [範囲]
OFF [ON; OFF]

リカバリー履歴ファイル内の項目は、PRUNE HISTORY コマンドまたは db2Prune API を使用して整理できます。また、データベース全体をバックアップするたびに、リカバリー履歴ファイルを自動的に整理するように IBM Data Server データベース・マネージャーを構成することもできます。 *auto_del_rec_obj* データベース構成パラメーターを ON に設定した場合、データベース・マネージャーは、履歴ファイルを整理するときに、対応する物理ログ・ファイル、バックアップ・イメージ、およびロード・コピー・イメージも削除します。ストレージ・メディアがディスクの場合、または Tivoli Storage Manager などのストレージ・マネージャーを使用している場合、データベース・ログ、バックアップ・イメージ、およびロード・コピー・イメージなどのリカバリー・オブジェクトだけをデータベース・マネージャーは削除できます。

auto_maint - 自動保守

このパラメーターは、他のすべての自動保守データベース構成パラメーター (*auto_db_backup*、*auto_tbl_maint*、 *auto_runstats*、*auto_stats_prof*、*auto_stmt_stats*、*auto_prof_upd*、および *auto_reorg*) の親パラメーターです。

構成タイプ
データベース

適用

- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- クライアント
- ローカル・クライアントを持つデータベース・サーバー
- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ

オンラインで構成可能

伝搬クラス

即時

デフォルト [範囲]

ON [ON; OFF]

このパラメーターを使用不可にすると、データベース構成ファイルに記録されている子パラメーターの設定を変更しなくても、すべての子パラメーターが使用不可になります。この親パラメーターを使用可能にすると、記録されている子パラメーターの値が有効になります。このようにして、自動保守を一括で使用不可または使用不可にすることができます。

デフォルトでは、このパラメーターは ON に設定されます。

以下のパラメーターを設定することにより、それぞれの自動保守機能を個別に使用可能または使用不可にできます。

auto_db_backup

この自動保守パラメーターは、データベースの自動バックアップ操作を使用可能または使用不可にします。自動化された動作を指定するために、バックアップ・ポリシー (ルールまたはガイドラインの定義済みセット) を使用できます。バックアップ・ポリシーの目的は、必ず定期的にデータベースがバックアップされるようにすることです。データベースのバックアップ・ポリシーは、DB2 ヘルス・モニターが最初に実行するときに自動的に作成されます。デフォルトでは、このパラメーターは OFF に設定されます。このパラメーターを使用可能にするには、このパラメーターを ON に設定し、親パラメーターも使用可能にすることが必要です。

auto_tbl_maint

このパラメーターは、すべての表保守パラメーター (*auto_runstats*、*auto_stats_prof*、 *auto_prof_upd*、and *auto_reorg*) の親パラメーターです。このパラメーターを使用不可にすると、データベース構成ファイルに記録されている子パラメーターの設定を変更しなくても、すべての子パラメーターが使用不可になります。この親パラメーターを使用可能にすると、記録されている子パラメーターの値が有効になります。このようにして、表保守を一括で使用可能または使用不可にすることができます。

デフォルトでは、このパラメーターは ON に設定されます。

auto_runstats

この自動表保守パラメーターは、データベースの自動表 RUNSTATS 操作を使用可能または使用不可にします。自動化された動作を指定するために、RUNSTATS ポリシー (ルールまたはガイドラインの定義済みセット) を使

用できます。 RUNSTATS ユーティリティーによって収集された統計は、物理データにアクセスするための最も効率的なプランを判別するために、オプティマイザーによって使用されます。このパラメーターを使用可能にするには、このパラメーターを ON に設定し、親パラメーターも使用可能にすることが必要です。

デフォルトでは、このパラメーターは ON に設定されます。

auto_stats_prof

この自動表保守パラメーターを使用可能にすると、統計プロファイル生成がオンになります。統計プロファイル生成は、複数の表に対する複雑な照会、多数の述部、結合、およびグループ化操作を含むワークロードを持つアプリケーションを改善するために設計されたものです。このパラメーターを使用可能にするには、このパラメーターを ON に設定し、親パラメーターも使用可能にすることが必要です。

デフォルトでは、このパラメーターは OFF に設定されます。

auto_stmt_stats

このパラメーターは、リアルタイム統計の収集を有効/無効にします。このパラメーターは、*auto_runstats* 構成パラメーターの子です。このフィーチャーは、親である *auto_runstats* 構成パラメーターも有効である場合にのみ有効です。例えば、*auto_stmt_stats* を有効にするには、*auto_maint*、*auto_tbl_maint*、および *auto_runstats* を ON に設定します。子の値を保持するために、*auto_maint* 構成パラメーターが OFF になっているときに *auto_runstats* 構成パラメーターを ON にすることができます。その場合でも、対応する Auto Runstats フィーチャーは OFF のままです。

Auto Runstats と Auto Reorg の両方が有効であると仮定すると、設定は次のようになります。

```
自動保守                                    (AUTO_MAINT) = ON
  データベース自動バックアップ            (AUTO_DB_BACKUP) = OFF
    表自動保守                              (AUTO_TBL_MAINT) = ON
      自動 RUNSTATS                         (AUTO_RUNSTATS) = ON
        自動ステートメント統計    (AUTO_STMT_STATS) = OFF
      自動統計プロファイル作成                (AUTO_STATS_PROF) = OFF
        自動プロファイル更新          (AUTO_PROF_UPD) = OFF
        自動再編成                            (AUTO_REORG) = ON
```

auto_tbl_maint を OFF に設定すると、Auto Runstats と Auto Reorg の両方のフィーチャーを一時的に無効にすることができます。どちらのフィーチャーも、*auto_tbl_maint* の設定を ON に戻すことによって、後で有効にすることができます。変更を有効にするために、db2stop コマンドまたは db2start コマンドを発行する必要はありません。

デフォルトでは、このパラメーターは OFF に設定されます。

auto_prof_upd

この自動表保守パラメーター (*auto_stats_prof* の子パラメーター) を使用可能にすると、推奨値にしたがった RUNSTATS プロファイルの更新が指定されます。このパラメーターを使用不可にすると、推奨値は opt_feedback_ranking 表に保管されます。 RUNSTATS プロファイルを手動で更新するときに、この推奨値を調べることができます。このパラメーターを使用可能にするには、このパラメーターを ON に設定し、親パラメーターも使用可能にすることが必要です。

デフォルトでは、このパラメーターは OFF に設定されます。

auto_reorg
この自動表保守パラメーターは、データベースの表および索引の自動再編成を使用可能または使用不可にします。自動化された動作を指定するために、再編成ポリシー (ルールまたはガイドラインの定義済みセット) を使用できます。このパラメーターを使用可能にするには、このパラメーターを ON に設定し、親パラメーターも使用可能にすることが必要です。

デフォルトでは、このパラメーターは OFF に設定されます。

autorestart - 自動再始動使用可能

このパラメーターは、データベースが異常終了した場合に、アプリケーションがデータベースに接続するときにデータベース・マネージャーが自動的にデータベース再始動ユーティリティーを呼び出せるかどうかを決定します。

構成タイプ
データベース

パラメーター・タイプ
オンラインで構成可能

伝搬クラス
即時

デフォルト [範囲]
On [On; Off]

データベース再始動ユーティリティーは、アプリケーションがデータベースに接続しているときに (電源障害やシステム・ソフトウェアの障害などで) データベースが異常終了した場合にクラッシュ・リカバリー を実行します。データベースのバッファー・プールに入っていても、障害の発生時にディスクに書き込まれていなかったコミット済みトランザクションには、すべてクラッシュ・リカバリーが適用されます。また、非コミット・トランザクションは、ディスクに書き込まれていたとしても、この操作によってすべてロールバックされます。

autorestart が使用可能になっていない場合は、クラッシュ・リカバリーが実行される必要がある (再始動される必要がある) データベースに接続を試みたアプリケーションは、SQL1015N エラーを受け取ることになります。この場合は、そのアプリケーションでデータベース再始動ユーティリティーを呼び出すか、あるいはユーザーがリカバリー・ツールの再始動操作を選択することで、データベースを再始動することができます。

avg_appls - アクティブ・アプリケーションの平均数

このパラメーターを照会オプティマイザーで使用すると、選択されたアクセス・プラン用として実行時に使用できる、バッファー・プールの量を見積もるのに役立ちます。

構成タイプ
データベース

パラメーター・タイプ
オンラインで構成可能

伝搬クラス
ステートメント境界

デフォルト [範囲]
Automatic [1 – maxappls]

単位 カウンター

推奨: DB2 をマルチユーザー環境で、特に複雑な照会と大容量バッファー・プールを使用して実行する場合は、照会オプティマイザーに複数の照会ユーザーがシステムを使用していることを知らせて、照会オプティマイザーがバッファー・プールをどれだけ使えるかをより少なく見積もるようにする必要があります。

このパラメーターを設定するときは、このデータベースを通常使用する複雑な照会を実行するアプリケーションの数を見積もる必要があります。軽い OLTP アプリケーションはすべて、この見積もりから除外する必要があります。この数の見積もりが難しい場合は、以下の値を掛けて計算できます。

- データベースに対して実行中のアプリケーションの平均数。特定の時点におけるアプリケーションの数についての情報は、データベース・システム・モニターによって得ることができ、サンプリング技法を使用すれば、ある期間についての平均を計算できます。データベース・システム・モニターから得られる情報には、OLTP アプリケーションと非 OLTP アプリケーションの両方が含まれます。
- 複合照会を実行するアプリケーションのパーセンテージの見積もり

オプティマイザーに影響する他の構成パラメーターの調整の場合と同じように、このパラメーターも増分を小さくして調整する必要があります。こうすることによって、アクセス・パスの変化を小さくすることができます。

このパラメーターを変更した場合は、アプリケーションの再バインド (REBIND PACKAGE コマンドを使用) を考慮してください。

backup_pending - バックアップ・ペンディング標識

このパラメーターは、データベースにアクセスする前にデータベース全体のバックアップを行う必要があるかどうかを示します。

構成タイプ
データベース

パラメーター・タイプ
通知

このパラメーターはデータベースをリカバリー不能からリカバリー可能にするために、データベース構成を変更した場合にのみオンになります。 (つまり、最初に *logretain* および *userexit* パラメーターの両方が NO に設定されてから、これらのパラメーターのいずれかあるいは両方が YES に設定される、または、LOGARCHMETH1、LOGARCHMETH2のいずれかがOFF以外の値に設定され、データベース構成への更新が受け入れられた場合です)。

blk_log_dsk_ful - ログ・ディスク・フルによるアプリケーション中断

このパラメーターを設定して、DB2 で新しいログ・ファイルをアクティブ・ログ・パス内に作成できない場合に、ディスク・フル・エラーが発生しないようにすることができます。

構成タイプ
データベース

パラメーター・タイプ
オンラインで構成可能

伝搬クラス
即時

デフォルト [範囲]
No [Yes; No]

ディスク・フル・エラーを生成する代わりに、正常実行されるまで DB2 は 5 分おきにログ・ファイルの作成を試行します。試行するたびに、DB2 はメッセージを管理通知ログに書き込みます。ログ・ディスクが満杯になったためにアプリケーションが停止したことを確認するには、管理通知ログをモニターするしかありません。ログ・ファイルが正常に作成されるまでは、表データの更新を試行するユーザー・アプリケーションにより、トランザクションをコミットすることができません。読み取り専用照会が直接影響を受ける可能性はありませんが、照会が更新要求によってロックされているデータ、または更新アプリケーションによってバッファー・プール内に固定されているデータにアクセスする必要がある場合は、読み取り専用照会もハングするように見えます。

blk_log_dsk_ful を *yes* に設定すると、DB2 がログ・ディスク・フル・エラーを検出したときにアプリケーションがハングするので、ユーザーがエラーを解決し、トランザクションを完了することができます。ディスク・フル状態は、古いログ・ファイルを別のファイル・システムに移動するか、ファイル・システムを拡張して、ハングしているアプリケーションが完了できるようにすることで解決できます。

blk_log_dsk_ful が *no* に設定されている場合は、ログ・ディスク・フル・エラーを受け取ったトランザクションは失敗し、ロールバックされます。状態によっては、トランザクションがログ・ディスク・フル・エラーの原因である場合は、データベースがダウンする場合があります。

catalogcache_sz - カタログ・キャッシュ・サイズ

このパラメーターでは、カタログ・キャッシュがデータベース・ヒープから使用できる最大スペースをページ単位で指定します。

構成タイプ
データベース

パラメーター・タイプ
オンラインで構成可能

伝搬クラス
即時

デフォルト [範囲]
-1 [MAXAPPLS*5]

単位 ページ (4 KB)

割り振られるタイミング
データベースの初期化時

解放されるタイミング
データベースがシャットダウンされるとき

このパラメーターは、データベース共用メモリーから割り振られ、システム・カタログ情報をキャッシュに入れる場合に使用されます。パーティション・データベース・システムでは、それぞれのデータベース・パーティションごとにカタログ・キャッシュが 1 つずつあります。

個々のデータベース・パーティションでカタログ情報をキャッシュに入れると、データベース・マネージャーは、以前検索された情報を入手するためにシステム・カタログ (またはパーティション・データベース環境におけるカタログ・ノード、あるいはその両方) にアクセスする必要がなくなるので、その内部オーバーヘッドを低減できます。カタログ・キャッシュを使用することにより、次の操作の総合的なパフォーマンスを向上することができます。

- パッケージのバインド、および SQL と XQuery ステートメントのコンパイル
- データベース・レベル特権のチェック、ルーチン特権、グローバル変数特権、およびロール許可を伴う操作
- パーティション・データベース環境で非カタログ・ノードに接続されるアプリケーション

サーバーまたはパーティション・データベース環境でデフォルト (-1) を取ることによって、ページ割り振りの計算に使用される値は、*maxappls* 構成パラメーターに指定されている値の 5 倍になります。これに対する例外が生じるのは、*maxappls* の 5 倍が 8 より小さい場合です。この状態では、デフォルト値 -1 で、*catalogcache_sz* は 8 に設定されます。

推奨: デフォルト値で開始し、データベース・システム・モニターを使用して調整してください。このパラメーターを調整するときは、カタログ・キャッシュ用として予約されている余分のメモリーについて、たとえば、バッファー・プールやパッケージ・キャッシュなどといった別の目的に割り振った方が、その有効性が増すかどうか考慮する必要があります。

短時間に SQL または XQuery コンパイルが集中し、その後はほとんど発生しないようなケースでは、このパラメーターの調整が特に重要です。キャッシュが大き過ぎる場合は、使用されなくなった情報のコピーの保留にメモリーが浪費される可能性があります。

パーティション・データベース環境では、非カタログ・ノードで必要とされるカタログ情報は、必ず最初にカタログ・ノードでキャッシュに入れられるので、カタログ・ノードの *catalogcache_sz* は、設定値を大きくする必要があるかどうか考慮してください。

cat_cache_lookups (カタログ・キャッシュ参照数)、*cat_cache_inserts* (カタログ・キャッシュ挿入)、*cat_cache_overflows* (カタログ・キャッシュ・オーバーフロー)、および *cat_cache_size_top* (カタログ・キャッシュ最高水準点) モニター・エレメントは、この構成パラメーターを調整する必要があるかどうか判別する場合に役立ちます。

注: カタログ・キャッシュは、パーティション・データベース環境のすべてのノードに存在します。それぞれのノードごとにローカル・データベース構成ファイルがあるので、それぞれのノードの *catalogcache_sz* 値によって、ローカル・カタログ・キャッシュのサイズが定義されます。キャッシングが効率的に行われ、オーバーフローが発生しないようにするために、それぞれのノードで *catalogcache_sz* 値を明示的に設定し、非カタログ・ノードの *catalogcache_sz* をカタログ・ノードの値よりも小さい値に設定できる可能性を考慮する必要があります。非カタログ・ノードでキャッシュに入れる必要のある情報は、カタログ・ノードのキャッシュから検索されるということを念頭に置いておいてください。したがって、非カタログ・ノードのカタログ・キャッシュは、カタログ・ノードのカタログ・キャッシュにある情報のサブセットのようなものです。

一般的に、キャッシュ・スペースが多く必要になるのは、作業単位に幾つもの動的 SQL または XQuery ステートメントが含まれる場合、または多数の静的 SQL または XQuery ステートメントが含まれるパッケージをバインドする場合です。

chngpgs_thresh - 変更済みページしきい値

このパラメーターで、非同期ページ・クリーナーが現在アクティブでない場合に、非同期ページ・クリーナーが始動する変更済みページ数のレベル (パーセント) を指定します。

構成タイプ
データベース

パラメーター・タイプ
構成可能

デフォルト [範囲]
60 [5 – 99]

単位 パーセント

非同期ページ・クリーナーは、バッファー・プール内のスペースがデータベース・エージェントによって必要とされる前に、変更済みページをバッファー・プール (複数の場合もある) からディスクに書き込みます。その結果として、データベース・エージェントは、変更済みページが書き出されて、バッファー・プール内のスペースを使用できるようになるのを待機する必要がなくなります。こうして、データベース・アプリケーション全体のパフォーマンスが向上します。

ページ・クリーナーを始動すると、ディスクに書き込むページのリストが作成されます。これらのページのディスクへの書き込みが完了すると、ページ・クリーナーは再度非アクティブになり、次のトリガーの開始を待ちます。

DB2_USE_ALTERNATE_PAGE_CLEANING レジストリー変数が設定されると (つまり、ページ・クリーニングの代替方法を使用すると)、*chngpgs_thresh* パラメーター

は無効になり、データベース・マネージャーはバッファー・プールに保持するダーティー・ページの数を自動的に判別します。

推奨: 更新トランザクション・ワークロードが大きいデータベースの場合は、パラメーター値をデフォルト値以下に設定することによって、一般的には、バッファー・プール内に十分なクリーン・ページを確保できます。データベースに非常に大きな表が少数しかない場合は、パーセンテージをデフォルトよりも大きくすると、パフォーマンスを上げることができます。

codepage - データベースのコード・ページ

このパラメーターは、データベースを作成するために使用されたコード・ページを示します。 *codepage* パラメーターは、*codeset* パラメーターに基づいて派生されます。

構成タイプ
データベース

パラメーター・タイプ
通知

codeset - データベース用コード・セット

このパラメーターは、データベースを作成するために使用されたコード・セットを示します。コード・セットは、*codepage* パラメーターの値を決定する場合に、データベース・マネージャーで使用されます。

構成タイプ
データベース

パラメーター・タイプ
通知

collate_info - 照合情報

このパラメーターは、データベースの照合シーケンスを決定します。言語認識照合の場合、先頭の 256 バイトには、ストリング表記による照合名が入ります (例えば、″SYSTEM_819_US″)。

このパラメーターを表示できるのは、db2CfgGet API を使用した場合だけです。コマンド行プロセッサーやコントロール・センターでは表示**できません**。

構成タイプ
データベース

パラメーター・タイプ
通知

このパラメーターは、260 バイトのデータベース照合情報を提供します。最初の 256 バイトでデータベース照合シーケンスを指定するのに対して、バイト「n」には、データベースのコード・ページで基本 10 進表記が「n」になっている、コード・ポイントのソートに対する重みづけが入ります。

最後の 4 バイトには、照合シーケンスのタイプについての内部情報が入ります。このパラメーターの最後の 4 バイトは整数です。整数は、プラットフォームのエンディアン順序に依存しています。使用できる値は次のとおりです。

- **0** - シーケンスに非固有の重みが含まれる
- **1** - シーケンスに固有の重みすべてが含まれる
- **2** - シーケンスは ID シーケンスで、ストリングがバイトごとに比較される
- **3** - シーケンスは NLSCHAR (TIS620-1 (コード・ページ 874) タイ語データベースの文字のソートに使用される)
- **4** - シーケンスは IDENTITY_16BIT で、「CESU-8 Compatibility Encoding Scheme for UTF-16: 8-bit (UTF-16 互換の 8–bit ビット・エンコード・スキーム)」のアルゴリズムをインプリメントします。これは Unicode Technical Consortium Web サイト (http://www.unicode.org) で入手可能な Unicode Technical Report #26 で指定されているものです。
- **X'8001'** - シーケンスは UCA400_NO であり、これは Unicode 規格バージョン 4.00 に基づく Unicode 照合アルゴリズム (UCA) で、正規化が暗黙的に ON に設定されたものをインプリメントします。
- **X'8002'** - シーケンスは UCA400_LTH であり、これは Unicode 規格バージョン 4.00 に基づいて Unicode 照合アルゴリズム (UCA) をインプリメントし、すべてのタイ語文字を Royal Thai Dictionary の順序でソートします。
- **X'8003'** - シーケンスは UCA400_LSK であり、これは Unicode 規格バージョン 4.00 に基づく Unicode 照合アルゴリズム (UCA) をインプリメントし、すべてのスロバキア語文字を適正にソートします。

注: 言語認識照合の場合、先頭の 256 バイトには、ストリング表記による照合名が入ります。

この内部タイプ情報を使用する場合は、別のプラットフォームにあるデータベースに関する情報を検索するときに、バイト反転を考慮する必要があります。

照合シーケンスは、データベース作成時に指定できます。

country/region - データベース・テリトリー・コード

このパラメーターは、データベースを作成するために使用されたテリトリー・コードを示します。

構成タイプ
データベース

パラメーター・タイプ
通知

database_consistent - データベースの整合性

このパラメーターは、データベースが整合状態にあるかどうかを示します。

構成タイプ
データベース

パラメーター・タイプ
通知

YES では、すべてのトランザクションがコミットまたはロールバックされているので、データは整合性がとれていることを示します。データベースの整合性がとれているときに、システムが「クラッシュ」した場合は、データベースを使用可能にするために、特殊なアクションを行う必要はありません。

NO では、トランザクションがペンディング中であるか、データベース上でペンディング中のタスクが他にあって、この時点でデータに整合性がないことを示します。データベースに整合性がないときに、システムが「クラッシュ」した場合は、RESTART DATABASE コマンドを使用してデータベースを再始動し、データベースを使用可能にする必要があります。

database_level - データベース・リリース・レベル

このパラメーターは、データベースを使用できるデータベース・マネージャーのリリース・レベルを示します。

構成タイプ
データベース

パラメーター・タイプ
通知

マイグレーションが不完全であったり、失敗したりした場合は、このパラメーターに反映されるのはマイグレーションされなかったデータベースのリリース・レベルであり、 *release* パラメーター (データベース構成ファイルのリリース・レベル) とは異なる可能性があります。それ以外の場合は、*database_level* の値は、*release* パラメーターの値と同一になります。

database_memory - データベース共用メモリー・サイズ

このパラメーターは、データベース共用メモリー領域として予約されている共用メモリーの量を指定します。この量が個々のメモリー・パラメーター (たとえば、locklist、ユーティリティー・ヒープ、バッファー・プールなど) から計算した量よりも少ない場合には、その大きい方の量が使用されます。

構成タイプ
データベース

適用

- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- ローカル・クライアントを持つデータベース・サーバー
- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ
オンラインで構成可能

デフォルト [範囲]
Automatic [Computed, 0 - 4 294 967 295]

単位 ページ (4 KB)

割り振られるタイミング
データベースが活動化された時点

解放されるタイミング
データベースが非活動化された時点

このパラメーターを AUTOMATIC に設定するとセルフチューニングが使用可能になります。使用可能になると、メモリー・チューナーはデータベースの全体のメモリー要件を決定し、データベース共用メモリーに割り振られるメモリー量を現行データベース要件に応じて増加または減少します。たとえば、現行データベース要求量が高く、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・メモリー要求量が下げられるか、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。

メモリー・チューナーは、インスタンスに追加メモリーを追加することのメリットを計算し、常に最小の空きメモリー量を保持します。インスタンスに追加メモリーを提供することに大きなメリットがある場合、メモリー・チューナーはその分、保持空きメモリー量を減らします。メリットが少ない場合はその分、保持空きメモリーを増やします。これによってデータベースはシステム・メモリーの分配に協力することができます。

メモリー・チューナーは異なるメモリー・コンシューマーの間でメモリー・リソースをやりとりするので、セルフチューニングをアクティブにするには、少なくとも 2 つのメモリー・コンシューマーが使用可能になっていなければなりません。

この構成パラメーターの自動チューニングは、セルフチューニング・メモリーがデータベースに対して使用可能なとき (*self_tuning_mem* 構成パラメーターが ON に設定されているとき) にのみ行われます。

このパラメーターの管理を単純化するために、COMPUTED 設定によって、必要なメモリーの量の計算、およびデータベース活動化時点におけるその割り振りを、データベース・マネージャーに指示します。ヒープが構成サイズを超える場合は常に、データベース共用メモリー領域内のヒープのピーク・メモリー要件を満たすためにデータベース・マネージャーは追加メモリーの割り振りも行います。動的構成更新などその他の操作も、この追加メモリーにアクセスできます。 db2pd コマンドに -memsets オプションを指定して使用すると、データベース共用メモリー領域に残されている未使用のメモリーの量をモニターできます。

推奨: この値は、通常は AUTOMATIC のままになります。 AUTOMATIC 設定をサポートしない環境の場合、これを COMPUTED に設定する必要があります。例えば、追加のメモリーは、新規バッファー・プールを作成する場合や、既存のバッファー・プールのサイズを大きくする場合に使用できます。

注: バージョン 9.5 では、*database_memory* 構成パラメーターを AUTOMATIC に設定すると、初期データベース共有メモリーの割り振りは、そのデータベースに定義されているすべてのヒープとバッファー・プールの構成サイズになり、メモリーは必要に応じて増えます。 *database_memory* が特定の値に設定されている場合は、データベース・アクティベーション時に、要求されたメモリーの量が最初に割り振られます。初期メモリー量がオペレーティング・システムから割り振られない場合や、*instance_memory* 限度を超える場合は、データベース・アクティベーションは失敗し、SQL1084C エラー (共有メモリー・セグメントを割り振れない) が戻されます。

DB2 メモリーの消費を制御する:

instance_memory が AUTOMATIC に設定されている場合、インスタンスの合計のメモリー消費量の固定された上限は、インスタンスの開始時 (db2start) に設定されます。データベース・マネージャーの実際のメモリー消費量は、作業負荷により変動します。セルフチューニング・メモリー・マネージャーで *database_memory* 調整の実行が有効になっていると (新規データベースのデフォルト)、ランタイム時にセルフチューニング・メモリー・マネージャーが、ファンクション・メモリー所要量に使用できる十分な空き *instance_memory* を確保しつつ、システムの空き物理メモリーに応じてデータベース共有メモリー・セット内でのパフォーマンスが重要なヒープのサイズを動的に更新します。詳しくは、*instance_memory* 構成パラメーターを参照してください。

一部の Linux[1] カーネルでの制限:

一部の Linux カーネルのオペレーティング・システム制限のため、セルフチューニング・メモリー・マネージャーで *database_memory* を AUTOMATIC に設定することは、現在許可されていません。ただし、*instance_memory* が AUTOMATIC ではなく特定の値に設定されている場合に限り、この設定はこれらのカーネル上でも許可されます。*database_memory* が AUTOMATIC に設定されているときに、*instance_memory* が後から AUTOMATIC に戻されると、*database_memory* 構成パラメーターは、次回のデータベース・アクティベーション時に自動的に COMPUTED に更新されます。一部のデータベースがすでにアクティブになっている場合は、セルフチューニング・メモリー・マネージャーが全体の *database_memory* サイズ調整を停止します。

[1] Linux の場合、このパラメーターの AUTOMATIC 設定は RHEL5 および SUSE 10 SP1 以降でサポートされています。他のすべての検証済みの Linux 配布版は、カーネルがこのフィーチャーをサポートしていない場合 COMPUTED に戻します。

db_mem_thresh - データベース・メモリーしきい値

このパラメーターは、コミット済みのメモリー・ページを解放してオペレーティング・システムに戻そうとするまでに、コミット済みであっても現在未使用という状態をデータベース・マネージャーがどの量まで許可するかを、データベース共用メモリーの最大パーセンテージで表すものです。

構成タイプ

データベース

パラメーター・タイプ

オンラインで構成可能

伝搬クラス

即時

デフォルト [範囲]

10 [0–100]

単位 パーセント

このデータベース構成パラメーターは、データベース・マネージャーが余分な未使用データベース共用メモリーをどのように扱うかに関係します。メモリーのページ

は通常、プロセスからアクセスがあるたびにコミットされます。つまり、メモリーのページはオペレーティング・システムによって割り振られており、物理メモリーまたはディスク上のページ・ファイル内のスペースを占有します。データベース・ワークロードによっては、一日の特定の時間にデータベース共用メモリーのピーク時要件が存在する場合があります。オペレーティング・システムがそうしたピーク時要件を満たせるメモリーをいったんコミットすると、メモリーのピーク時要件が収まった後でも、そのメモリーはコミットされたままです。

許容値は、0 (未使用データベース共用メモリーを即時に解放する) から 100 (未使用データベース共用メモリーをまったく解放しない) までの範囲の整数です。デフォルトは 10 (データベース共用メモリーの 10% を超える量が現在未使用のときだけ未使用メモリーを解放する) です。これは、たいていのワークロードに適合します。

この構成パラメーターは、動的に更新することができます。このパラメーターを更新するときは、注意が必要です。設定値を低くしすぎるとマシンに過剰のメモリー・スラッシングが発生する可能性があり (メモリー・ページのコミットと解放が絶えず行われる)、また設定値を高くしすぎるとデータベース・マネージャーが、他のプロセスが使用できるように、データベース共用メモリーをオペレーティング・システムに戻せなくなる可能性があります。

データベース共用メモリー領域が DB2_PINNED_BP レジストリー変数によって滞留されている場合や DB2_LARGE_PAGE_MEM レジストリー変数によってラージ・ページに構成されている場合、あるいは DB2MEMDISCLAIM レジストリー変数によってメモリーの解放が明示的に使用不可になっている場合は、この構成パラメーターは無視されます (未使用データベース共用メモリー・ページはコミットされたままになります)。

Linux のバージョンによっては、共用メモリー・セグメントの部分範囲を解放してオペレーティング・システムに戻すことをサポートしていないものがあります。こうしたプラットフォームでは、このパラメーターは無視されます。

dbheap - データベース・ヒープ

このパラメーターにより、データベース・ヒープが使用する最大メモリーを決定します。

バージョン 9.5 では、このデータベース構成パラメーターのデフォルト値は AUTOMATIC であるため、*database_memory* 限度または *instance_memory* 限度のいずれかに達するまで、データベース・ヒープが必要に応じて増加します。

構成タイプ
データベース

パラメーター・タイプ
オンラインで構成可能

伝搬クラス
即時

デフォルト [範囲]
Automatic [32 - 524 288]

単位　ページ (4 KB)

割り振られるタイミング
データベースが活動化された時点

解放されるタイミング
データベースが非活動化された時点

データベースごとに 1 つのデータベース・ヒープが存在し、データベース・マネージャーは、データベースに接続されているすべてのアプリケーションの代わりに、データベース・ヒープを使用します。これには表、索引、表スペース、およびバッファー・プールのコントロール・ブロック情報が含まれます。また、ログ・バッファー (*logbufsz*)、およびユーティリティーによって使用される一時メモリー用のスペースも含まれます。したがって、ヒープのサイズは多数の変数によって決まることになります。コントロール・ブロック情報は、すべてのアプリケーションがデータベースから切断されるまで、ヒープ内に保持されます。

データベース・マネージャーが始動のために取得する必要がある最小量は、最初の接続時に割り振られます。構成された上限に達するまで、または AUTOMATIC に設定されている場合は *database_memory* か *instance_memory*、またはその両方のメモリーがすべて消費されるまで、データ域は必要に応じて拡張されます。

dbheap 構成パラメーターに割り当てる概算値を決定するガイドラインとして、以下の公式を使用することができます。

```
表スペースごとに 10K + 表ごとに 4K + 範囲がクラスター化された表 (RCT) ごとに
(1K + 4 * 使用エクステントの数)
```

構成する *dbheap* の値は、割り振られるデータベース・ヒープの一部しか表していません。このデータベース・ヒープは、グローバルなデータベース・メモリー要件を満たすために使用されるメイン・メモリー領域です。 *dbheap* の値に加え、データベースの始動に必要な基本割り振りを含めたサイズになります。メモリー・トラッカー、スナップショット・モニター、および db2pd などメモリーの使用量をレポートするツールが、この大きめのデータベース・ヒープの統計を報告します。*dbheap* 構成パラメーターによって表される割り振りに対して、独立したトラッキングが行われることはありません。そのため、これらのツールからレポートされるデータベース・ヒープのメモリー使用量の統計が、*dbheap* パラメーターで構成した値を超過しても、それは正常なことです。

データベース・システム・モニターを使用すれば、*db_heap_top* (割り振られる最大データベース・ヒープ) エレメントからデータベース・ヒープ用として使用されたメモリーの最大量をトラックすることができます。

注:

- Workload Management (WLM) の作業クラス・セットと作業アクション・セットは、データベース・ヒープ内に保管されます。ただしこのために、メモリーがわずかに消費されます。
- トラステッド・コンテキスト、Workload Management、および監査ポリシーの情報は、高速処理のためにメモリー内にキャッシュされます。このメモリーは、データベース・ヒープから割り振られます。したがって、ユーザー定義のトラステッド・コンテキスト、ワークロード管理、および監査ポリシーの各オブジェクトでは、より多くのメモリー所要量をデータベース・ヒープに課すことになりま

す。この場合、データベース・マネージャーによりデータベース・ヒープ・サイズが自動的に管理されるように、ご使用のデータベース・ヒープ構成を AUTOMATIC に設定することをお勧めします。

decflt_rounding - 10 進浮動小数点丸め構成パラメーター

このパラメーターを使用して、10 進浮動小数点 (DECFLOAT) の丸めモードを指定できます。丸めモードは、サーバー、および LOAD における 10 進浮動小数点の操作に影響します。

構成タイプ
データベース

パラメーター・タイプ
構成可能

下記の 609 ページの『decflt_rounding を変更した場合の影響』を参照してください。

デフォルト [範囲]
ROUND_HALF_EVEN [ROUND_CEILING、 ROUND_FLOOR、 ROUND_HALF_UP、 ROUND_DOWN]

DB2 では、IEEE に準拠した 5 つの 10 進浮動小数点丸めモードがサポートされています。丸めモードにより、計算結果が精度を超えたときの、結果の丸め方法が指定されます。すべての丸めモードの定義は、以下のとおりです。

ROUND_CEILING
正の無限大へ丸めます。廃棄される数字がすべてゼロであるか、または符号が負の場合、結果は未変更のままになります。それ以外の場合、結果の係数は 1 だけ増大する (切り上げられる) はずです。

ROUND_FLOOR
負の無限大の方向へ丸めます。廃棄される数字がすべてゼロであるか、または符号が正の場合、結果は未変更のままになります。それ以外の場合、符号は負になり、結果の係数は 1 だけ増大するはずです。

ROUND_HALF_UP
最も近い値に丸めます。等距離の場合は、1 に切り上げます。廃棄される数字が、左側の次の桁内の 1 の半分 (0.5) 以上の場合、結果の係数は 1 だけ増大する (切り上げられる) はずです。それ以外の場合、廃棄された数字 (0.5 以下) は無視されます。

ROUND_HALF_EVEN
最も近い値に丸めます。等距離の場合は、結果の数字が偶数になるように丸めます。廃棄される数字が、左側の次の桁内の 1 の半分 (0.5) より大きい場合、結果の係数は 1 だけ増大する (切り上げられる) はずです。半分より小さい場合、結果の係数は調整されません。つまり、廃棄された数字は無視されます。それ以外の場合 (正確に半分の場合)、結果の係数の右端の数字が偶数であれば、その係数は変更されませんが、右端の数字が奇数であれば、その桁が偶数になるように 1 だけ増大 (切り上げ) します。この丸めモードは IEEE の 10 進浮動小数点仕様によるデフォルトの丸めモードであり、DB2 製品のデフォルトの丸めモードでもあります。

ROUND_DOWN
0 の方向へ丸めます (切り捨て)。廃棄された数字は無視されます。

表 71 に、12.341、12.345、12.349、12.355、および -12.345 をそれぞれ異なる丸めモードで、4 桁に丸めた結果を示します。

表 71. 10 進浮動小数点の丸めモード

丸めモード	12.341	12.345	12.349	12.355	-12.345
ROUND_DOWN	12.34	12.34	12.34	12.35	-12.34
ROUND_HALF_UP	12.34	12.35	12.35	12.36	-12.35
ROUND_HALF_EVEN	12.34	12.34	12.35	12.36	-12.34
ROUND_FLOOR	12.34	12.34	12.34	12.35	-12.35
ROUND_CEILING	12.35	12.35	12.35	12.36	-12.34

decflt_rounding を変更した場合の影響

- 以前作成されたマテリアライズ照会表 (MQT) の内容が、新しい丸めモードで作成されるものとは異なる結果になる可能性があります。この問題を訂正するには、影響を受ける可能性のある MQT をリフレッシュします。
- トリガーの結果は、新しい丸めモードに影響されることがあります。丸めモードを変更しても、既に書き込まれたデータには影響しません。
- 表にデータの挿入を許可した制約が、再評価すると、同じデータを拒否することがあります。同様に、表にデータの挿入を許可しなかった制約が、再評価すると、同じデータを受け入れることがあります。このような問題を確認して修正するには、SET INTEGRITY ステートメントを使用します。計算が decflt_rounding に依存している生成済みの列の値は、その生成済みの列の値以外は同一である 2 つの行で、片方の行が decflt_rounding の変更前に挿入され、もう片方の行が decflt_rounding の変更後に挿入された場合に、異なることがあります。
- 丸めモードは、セクションにコンパイルされません。したがって、静的 SQL は *decflt_rounding* の変更後に再コンパイルする必要はありません。

注: この構成パラメーターの値は動的には変更されませんが、データベースからすべてのアプリケーションが切断された後でのみ、有効になります。データベースは活動化している場合、非活動化する必要があります。

dft_degree - デフォルト多重度

このパラメーターは、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指定します。

構成タイプ
データベース

パラメーター・タイプ
オンラインで構成可能

伝搬クラス
接続

デフォルト [範囲]
1 [-1(ANY), 1 - 32 767]

デフォルト値は 1 です。

値が 1 では、パーティション内並列処理がないことを意味します。値が -1 (または ANY) では、オプティマイザーがプロセッサーの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決定することを示します。

SQL ステートメントの並列処理の多重度は、CURRENT DEGREE 特殊レジスターまたは DEGREE BIND オプションを使用して、ステートメントのコンパイル時に指定されます。アクティブ・アプリケーションのパーティション内並列処理の最大実行時の多重度は、SET RUNTIME DEGREE コマンドを使用して指定します。「照会の最大並列処理多重度」 (*max_querydegree*) 構成パラメーターでは、すべての SQL 照会のパーティション内の照会最大並列処理多重度を指定します。

実行時に使用される実際の多重度は、次のうちで最も低い値になります。

- *max_querydegree* 構成パラメーター
- アプリケーション実行時の多重度
- SQL ステートメント・コンパイル時の多重度

dft_extent_sz - 表スペースのデフォルトのエクステント・サイズ

このパラメーターは、表スペースのデフォルト・エクステント・サイズを設定します。

構成タイプ
データベース

パラメーター・タイプ
オンラインで構成可能

伝搬クラス
即時

デフォルト [範囲]
32 [2 – 256]

単位 ページ

表スペースの作成時に、オプションで `EXTENTSIZE n` を指定できます。 `n` はエクステント・サイズです。 CREATE TABLESPACE ステートメントでエクステント・サイズを指定しなかった場合、データベース・マネージャーはこのパラメーターで与えられた値を使用します。

推奨 : 多くの場合、表スペースの作成時に、エクステント・サイズを明示的に指定します。このパラメーターの値を選択する前に、CREATE TABLESPACE ステートメントのエクステント・サイズを明示的に指定する方法を理解しておくべきです。

dft_loadrec_ses - ロード・リカバリー・セッションのデフォルト数

このパラメーターは、表ロードのリカバリー中に使用されるデフォルトのセッション数を指定します。

構成タイプ
データベース

パラメーター・タイプ
オンラインで構成可能

伝搬クラス
即時

デフォルト [範囲]
1 [1 - 30 000]

単位 カウンター

この値は、ロード・コピーの検索に使用される、適正な入出力セッション数に設定してください。ロード・コピーの検索は、リストアに似た操作です。環境変数 DB2LOADREC で指定されたコピー・ロケーション・ファイル内の項目によって、このパラメーターをオーバーライドすることができます。

ロード検索に使用されるデフォルトのバッファー数は、このパラメーターの値に 2 を加えた数になります。コピー・ロケーション・ファイル内のバッファー数もオーバーライドが可能です。

このパラメーターは、ロールフォワードができる場合にのみ適用されます。

dft_mttb_types - 最適化用デフォルト保守表タイプ

このパラメーターは、CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION 特殊レジスターのデフォルト値を指定します。このレジスターの値は、照会の最適化中に使用される REFRESH DEFERRED マテリアライズ照会表のタイプを決定します。

構成タイプ
データベース

パラメーター・タイプ
構成可能

デフォルト [範囲]
SYSTEM [ALL、NONE、FEDERATED_TOOL、 SYSTEM、USER、または値のリスト]

値のリストをコンマで区切って指定できます。例えば、'USER,FEDERATED_TOOL' などです。 ALL または NONE を他の値を一緒にリストすることはできません。同じ値を複数回指定することもできません。 db2CfgSet および db2CfgGet API とともに使用する場合、許容されるパラメーター値は 8 (ALL)、4 (NONE)、16 (FEDERATED_TOOL)、1 (SYSTEM)、および 2 (USER) です。ビット単位 OR を使用して、複数の値を一緒に指定できます。例えば、18 は USER,FEDERATED_TOOL に相当します。従来と同様、4 と 8 の値を他の値と一緒に使用することはできません。

dft_prefetch_sz - デフォルト・プリフェッチ・サイズ

このパラメーターは、表スペースのデフォルト・プリフェッチ・サイズを設定します。

構成タイプ
データベース

パラメーター・タイプ
オンラインで構成可能

伝搬クラス
即時

デフォルト [範囲]
Automatic [0 - 32 767]

単位 ページ

表スペースの作成時に、オプションで PREFETCHSIZE *n* を指定できます (*n* は、プリフェッチが行われる場合に、データベース・マネージャーが読み取るページ数です)。 CREATE TABLESPACE ステートメントの呼び出し時にエクステント・サイズを指定しなかった場合、データベース・マネージャーは *dft_prefetch_sz* パラメーターの現行値を使用します。

表スペースが AUTOMATIC DFT_PREFETCH_SZ を使用して作成される場合、表スペースのプリフェッチ・サイズは AUTOMATIC になります。 AUTOMATIC は、DB2 が次の式を使用して表スペースのプリフェッチ・サイズを自動的に計算および更新することを意味します。

```
prefetch size = (# containers) * (# physical spindles) * extent size
```

physical spindle の数はデフォルト設定が 1 であり、 DB2 レジストリー変数 DB2_PARALLEL_IO で指定できます。この計算は以下のタイミングで実行されます。

- データベース始動時
- AUTOMATIC プリフェッチ・サイズの指定で、初めて表スペースが作成されるとき
- ALTER TABLESPACE ステートメントの実行によって、表スペースのコンテナーの数が変更されるとき
- ALTER TABLESPACE ステートメントの実行によって、表スペースのプリフェッチ・サイズが AUTOMATIC に更新されるとき

ALTER TABLESPACE ステートメントを呼び出してプリフェッチ・サイズを手動で更新すると、直ちにプリフェッチ・サイズの AUTOMATIC 状態はオンまたはオフに切り替わります。

推奨: システム・モニター・ツールを使用して、システムが入出力を待機しているときに CPU がアイドルになっているかどうかを判別できます。このパラメーターの値を増やすと、使用される表スペースのプリフェッチ・サイズが定義されていない場合に役立ちます。

このパラメーターは、データベース全体にデフォルト値を適用しますが、データベース内のすべての表スペースに適しているとは限りません。例えば、値 32 はエクステント・サイズが 32 ページの表スペースには適しているかもしれませんが、エクステント・サイズが 25 ページの表スペースには適していません。一番よい方法は、それぞれの表スペースにプリフェッチ・サイズを明示的に設定することです。

デフォルトの表スペース・エクステント・サイズ (*dft_extent_sz*) で定義された表スペースの入出力を最小化するには、このパラメーターを *dft_extent_sz* パラメーター

の値の因数または倍数として指定してください。例えば、*dft_extent_sz* パラメーターが 32 の場合は、*dft_prefetch_sz* を 16 (32 の因数) または 64 (32 の倍数) に設定します。プリフェッチ・サイズがエクステント・サイズの倍数である場合は、以下の条件が満たされていればデータベース・マネージャーは入出力を並列して行うことができます。

- プリフェッチ中のエクステントが異なる物理装置上にある
- 複数の入出力サーバーが構成されている (*num_ioservers*)

dft_queryopt - デフォルト照会最適化クラス

照会最適化クラスは、SQL および XQuery 照会のコンパイルに別々の段階の最適化を使用するように、オプティマイザーに指示するために使用します。このパラメーターでは、SET CURRENT QUERY OPTIMIZATION ステートメントも BIND コマンドの QUERYOPT オプションも使用されていないときに使用される、デフォルトの照会最適化クラスを設定することによって、さらに柔軟性が得られます。

構成タイプ
データベース

パラメーター・タイプ
オンラインで構成可能

伝搬クラス
接続

デフォルト [範囲]
5 [0 — 9]

単位 照会最適化クラス (以下を参照)

照会最適化クラスは、現在のところ次のように定義されています。

- 0 - 最小照会最適化。
- 1 - DB2 バージョン 1 とほぼ同等。
- 2 - 低レベルの最適化。
- 3 - 中レベルの照会最適化。
- 5 - アクセス・プランの選択に費やされる労力を制限する、試行錯誤による重要な照会最適化。これはデフォルトです。
- 7 - かなりの照会最適化。
- 9 - 最大照会最適化。

dft_refresh_age - デフォルトの REFRESH AGE

このパラメーターは、REFRESH TABLE ステートメントが特定の REFRESH DEFERRED マテリアライズ照会表に対して処理されてからの最大期間を表します。この時間制限を超えると、マテリアライズ照会表がリフレッシュされるまで、マテリアライズ照会表を使用した照会は行われません。

構成タイプ
データベース

パラメーター・タイプ
構成可能

デフォルト [範囲]

0 [0, 99999999999999 (ANY)]

このパラメーターでは、CURRENT REFRESH AGE 特殊レジスターが指定されていない場合は、REFRESH AGE にデフォルト値が使用されます。このパラメーターは DECIMAL(20,6) というデータ・タイプのタイム・スタンプ期間値を指定します。CURRENT REFRESH AGE の値が 99999999999999 (ANY) で、QUERY OPTIMIZATION クラスの値が 2、または 5 以上である場合は、REFRESH DEFERRED マテリアライズ照会表が、動的照会の処理を最適化すると見なされます。

dft_sqlmathwarn - 演算例外時継続

このパラメーターは、演算エラーおよび検索変換エラーを SQL ステートメントのコンパイル時のエラーまたは警告として処理することを決定する、デフォルト値を設定します。

構成タイプ

データベース

パラメーター・タイプ

構成可能

デフォルト [範囲]

No [No, Yes]

静的 SQL ステートメントの場合は、このパラメーターの値は、バインド時にパッケージに関連付けられます。動的 SQL DML ステートメントの場合は、このパラメーターの値は、ステートメントの準備時に使用されます。

重要: データベースの *dft_sqlmathwarn* 値を変更した場合は、算術式が組み込まれているチェック制約、トリガー、およびビューの振る舞いが変更されることがあります。この結果、データベースのデータ整合性に影響を及ぼす場合があります。したがって、データベースの *dft_sqlmathwarn* の設定を変更するのは、新しい演算例外処理の振る舞いがチェック制約、トリガー、およびビューにどのように影響するかを入念に評価した後のみに限る必要があります。一度変更すると、その後の変更でも同じく入念な評価が必要です。

例として、割り算の算術演算が組み込まれている、次のチェック制約を考察してみましょう。

```
A/B > 0
```

dft_sqlmathwarn が「No」であるとき、B=0 で INSERT を試みると、ゼロによる割り算は演算エラーとして処理されます。DB2 が制約をチェックできないため、挿入操作は失敗します。*dft_sqlmathwarn* が「Yes」に変更された場合は、ゼロによる割り算は、演算警告として処理され、NULL という結果になります。NULL という結果では、述部が UNKNOWN に評価され、挿入操作は正常に行われます。*dft_sqlmathwarn* が変更されて元の「No」に戻った場合は、ゼロによる割り算のために DB2 が制約を評価できないため、同じ行を挿入する試みは失敗します。*dft_sqlmathwarn* が「Yes」であったときに B=0 で挿入された行は、表内にとどまり、選択できます。行に対する更新は、制約を評価させる更新の場合は失敗するが、制約の再評価を必要としない更新の場合は、正常に行われます。

dft_sqlmathwarn を「No」から「Yes」に変更する場合は、算術式から NULL を明示的に処理するために、あらかじめ制約の書き直しを考慮しておく必要があります。例:

```
( A/B > 0 ) AND ( CASE
                    WHEN A IS NULL THEN 1
                    WHEN B IS NULL THEN 1
                    WHEN A/B IS NULL THEN 0
                    ELSE 1
                    END
                  = 1 )
```

が使用できるのは、A と B が両方とも NULL 可能な場合です。そして、A または B が NULL 可能でない場合は、対応する IS NULL WHEN 節は除去できます。

dft_sqlmathwarn を「Yes」から「No」に変更するときは、まずその前に、例えば、次のような述部を使用して、不整合になる可能性のあるデータがないかどうかチェックしておく必要があります。

```
WHERE A IS NOT NULL AND B IS NOT NULL AND A/B IS NULL
```

不整合の行の切り分けができたら、*dft_sqlmathwarn* を変更する前に、適切なアクションを行って不整合を訂正しておく必要があります。また、変更後の算術式を使用して、制約を手動で再チェックすることもできます。これを実行するには、まず最初に、影響を受けた表を (SET CONSTRAINTS ステートメントの OFF 節によって) チェック・ペンディング状態に置き、次に表が (SET CONSTRAINTS ステートメントの IMMEDIATE CHECKED 節によって) チェックされるよう要求します。不整合データが演算エラーによって示されるので、制約は評価されません。

推奨: 演算例外が組み込まれている照会の処理が特に必要でない限り、デフォルト設定の No を使用してください。その上で、値 Yes を使用します。この状態が生じる可能性があるのは、他のデータベース・マネージャーで、演算例外が発生するかどうかに関係なく結果が得られる、SQL ステートメントを処理している場合です。

discover_db - データベース・ディスカバリー

このパラメーターは、サーバーでディスカバリー要求が受け取られたときに、データベースについての情報がクライアントに戻されないようにするために使用します。

構成タイプ
データベース

パラメーター・タイプ
オンラインで構成可能

伝搬クラス
即時

デフォルト [範囲]
Enable [Disable, Enable]

このパラメーターがデフォルトの状態では、このデータベースに対するディスカバリーを使用できます。

このパラメーター値を「Disable」に変更することによって、機密データを持つデータベースをディスカバリー処理から隠すことができます。これは、データベースに対する他のセキュリティー・コントロールに追加して実行することができます。

dlchktime - デッドロック・チェック・インターバル

このパラメーターは、1 つのデータベースに接続されているすべてのアプリケーションの間にデッドロックがないか、データベース・マネージャーがチェックする頻度を定義します。

構成タイプ
データベース

パラメーター・タイプ
オンラインで構成可能

伝搬クラス
即時

デフォルト [範囲]
10 000 (10 秒) [1 000 - 600 000]

単位 ミリ秒

デッドロックが発生するのは、同じデータベースに接続されている複数のアプリケーションが 1 つのリソースを際限なく待っているときです。この待機が解決されることはありません。それぞれのアプリケーションが、他のアプリケーションで継続する必要のあるリソースを保留しているからです。

注:

1. パーティション・データベース環境では、このパラメーターが適用されるのは、カタログ・ノードの場合だけです。
2. パーティション・データベース環境では、2 回目の反復の後でないと、デッドロックにフラグが立てられることはありません。

推奨: このパラメーターの値を大きくすると、デッドロックをチェックする頻度が低くなり、したがって、アプリケーション・プログラムでデッドロックの解決を待つために必要な時間が長くなります。

このパラメーター値を減らすとデッドロック・チェックの頻度が増え、その結果デッドロックが解決されるのをアプリケーション・プログラムが待たなければならない時間が減りますが、データベース・マネージャーがデッドロックをチェックするための時間が増えます。デッドロック・チェック・インターバルが小さすぎると、データベース・マネージャーは頻繁にデッドロック検出を実行するため、ランタイム・パフォーマンスが落ちる可能性があります。並行性を高めるために、このパラメーターが低めに設定されている場合は、*maxlocks* と *locklist* を適切に設定することにより、ロック競合の増加を招き、その結果として、デッドロック状態が増えることになりかねない、不必要なロック・エスカレーションを避けることができます。

dyn_query_mgmt - 動的 SQL および XQuery 照会管理

このパラメーターは、Query Patroller が、サブミットされた照会についての情報をキャプチャーするかどうかを決定します。

構成タイプ
データベース

パラメーター・タイプ
オンラインで構成可能

デフォルト [範囲]
0 (DISABLE) [1(ENABLE), 0 (DISABLE)]

このパラメーターは DB2 Query Patroller がインストールされている場合に有効です。このパラメーターを「ENABLE」に設定すると、Query Patroller は、サブミッター ID や実行の見積もりコスト (オプティマイザーが計算) など、照会の情報を収集します。こうした値は、照会を Query Patroller で、ユーザー・レベルおよびシステム・レベルのしきい値を基にして管理すべきかどうかを判断するのに使用します。

このパラメーターを「DISABLE」に設定すると、Query Patroller はサブミットされた照会に関する情報を収集せず、照会管理は行われません。

enable_xmlchar - XML への変換の有効化構成パラメーター

このパラメーターは SQL ステートメントで非 BIT DATA CHAR (つまり CHAR タイプ) の式に XMLPARSE 操作を実行できるかどうかを決定します。

構成タイプ
データベース

パラメーター・タイプ
構成可能

デフォルト [範囲]
Yes [Yes; No]

pureXML™ フィーチャーが非 Unicode データベースで使用されるとき、XMLPARSE 関数は、SQL ストリング・データがクライアント・コード・ページからデータベース・コード・ページに、次いで内部保管のために Unicode 変換されるときに、文字置換を引き起こす可能性があります。 *enable_xmlchar* を NO に設定すると、XML 解析中に文字データ・タイプの使用がブロックされ、文字タイプを非 Unicode データベースに挿入しようとすると、エラーが生成されます。 BLOB データ・タイプと FOR BIT DATA データ・タイプは、*enable_xmlchar* を NO に設定した場合は、これらのデータ・タイプが XML データのデータベースへの受け渡しに使用される際に、コード・ページ変換が実行されないため引き続き許可されます。

デフォルトで、*enable_xmlchar* は YES に設定され、文字データ・タイプの解析は許可されます。この場合、XML データの挿入中に置換文字が使用されないようにするために、挿入される XML データにはデータベース・コード・ページの一部であるコード・ポイントしか含まれないようにする必要があります。

注: この変更を反映させるには、クライアントがエージェントから切断して、再接続する必要があります。

failarchpath - フェイルオーバー・ログ・アーカイブ・パス

このパラメーターでは、メディアの問題の影響で 1 次と 2 次 (設定されている場合) のいずれのアーカイブ宛先にもログ・ファイルをアーカイブできない場合に、DB2 がログ・ファイルのアーカイブを試行する宛先のパスを指定します。ここに指定するパスは、ディスクを参照している必要があります。

構成タイプ
データベース

適用

- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- クライアント
- ローカル・クライアントを持つデータベース・サーバー
- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ
オンラインで構成可能

デフォルト [範囲]
Null [ ]

failarchpath の現行値で指定されたパスにログ・ファイルがある場合、*failarchpath*を更新しても即時に有効にはなりません。代わりに、すべてのアプリケーションが切断されたときに更新が有効になります。

groupheap_ratio - アプリケーション・グループ・ヒープ用メモリーのパーセント

このパラメーターはバージョン 9.5 では推奨されませんが、バージョン 9.5 より前のデータ・サーバーおよびクライアントでは引き続き使用されています。DB2 バージョン 9.5 のデータベース・マネージャーは、この構成パラメーターに対して指定された値をすべて無視します。バージョン 9.5 では、このパラメーターは *appl_memory* 構成パラメーターに置き換えられています。

注: 以下の情報は、バージョン 9.5 より前のデータ・サーバーおよびクライアントだけに当てはまります。

このパラメーターは、アプリケーション・グループ共有ヒープ専用メモリーが、アプリケーション・コントロール共用メモリー・セットに占めるパーセンテージを指定します。

構成タイプ
データベース

パラメーター・タイプ
構成可能

デフォルト [範囲]
70 [1 – 99]

単位 パーセント

このパラメーターは、コンセントレーターがオフに設定され、パーティション内並列処理が使用不可になっている非パーティション・データベースには、まったく影響がありません。

推奨: パフォーマンス上の問題が検出されない限り、このパラメーターのデフォルト値を変更しないでください。

hadr_db_role - HADR データベース役割

このパラメーターでは、データベースがオンラインかオフラインかにかかわらず、データベースの現在の役割を示します。

構成タイプ
データベース

適用

- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- ローカル・クライアントを持つデータベース・サーバー

パラメーター・タイプ
通知

有効な値は STANDARD、PRIMARY、または STANDBY です。

注: データベースがアクティブの場合、データベースの HADR 役割は GET SNAPSHOT FOR DATABASE コマンドを使用して判別することもできます。

hadr_local_host - HADR ローカル・ホスト名

このパラメーターでは、高可用性災害時リカバリー (HADR) TCP 通信のローカル・ホストを指定します。

構成タイプ
データベース

適用

- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- ローカル・クライアントを持つデータベース・サーバー

パラメーター・タイプ
構成可能

デフォルト
Null

ホスト名または IP アドレスのいずれかを使用できます。ホスト名が指定されてホスト名が複数の IP アドレスにマップされると、エラーが戻され、HADR は開始しません。ホスト名が複数の IP アドレスにマップされる場合には (1 次およびスタンバイに同じホスト名を指定したとしても)、1 次およびスタンバイは結局このホスト名を異なる複数の IP アドレスにマップします。それは、一部の DNS サーバーが IP アドレス・リストを非 deterministic 順で戻すためです。

ホスト名の形式は myserver.ibm.com です。 IP アドレスの形式は ″12.34.56.78″ です。

hadr_local_svc - HADR ローカル・サービス名

このパラメーターでは、ローカルの高可用性災害時リカバリー (HADR) プロセスが接続を受け入れる TCP サービス名またはポート番号を指定します。

構成タイプ
データベース

適用

- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- ローカル・クライアントを持つデータベース・サーバー

パラメーター・タイプ
構成可能

デフォルト
Null

1 次またはスタンバイ・データベース・システム上の hadr_local_svc の値は、それぞれのノードの値 svcename または svcename +1 と同じであってはなりません。

hadr_peer_window - HADR ピア・ウィンドウ構成パラメーター

hadr_peer_window をゼロ以外の時刻値に設定すると、1 次データベースでスタンバイ・データベースとの接続が失われた場合に、HADR の 1 次とスタンバイのデータベース・ペアは、依然ピア状態にあるかのように構成された時間の間動作し続けます。これは、データの整合性の保証に役立ちます。

構成タイプ
データベース

パラメーター・タイプ
構成可能

デフォルト [範囲]
0 [0 – 4 294 967 295]

単位 秒

使用上の注意:

- この値は、1 次データベースとスタンバイ・データベースの両方で同じ値である必要があります。
- 推奨の最小値は 120 秒です。
- *hadr_syncmode* 値が ASYNC に設定されている場合は、*hadr_peer_window* 値は無視されます。つまり、この値は ASYNC モードでは意味を持たないため、ゼロ (0) に設定されているかのように扱われます。
- スタンバイ・データベースが意図的にシャットダウンされた場合 (例えば保守のために) に、1 次データベースの可用性に影響を与えないようにするために、HADR ペアをピア状態のままで、スタンバイ・データベースを明示的に非活動化した場合、ピア・ウィンドウは呼び出されません。

hadr_remote_host - HADR リモート・ホスト名

このパラメーターでは、リモートの高可用性災害時リカバリー (HADR) データベース・サーバーの TCP/IP ホスト名または IP アドレスを指定します。

構成タイプ
データベース

適用
- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- ローカル・クライアントを持つデータベース・サーバー

パラメーター・タイプ
構成可能

デフォルト
Null

hadr_local_host と同様に、このパラメーターは必ず 1 つの IP アドレスにのみマップされます。

hadr_remote_inst - リモート・サーバーの HADR インスタンス名

このパラメーターでは、リモート・サーバーのインスタンス名を指定します。 DB2 コントロール・センターなどの管理ツールは、このパラメーターを使用してリモート・サーバーと接続します。高可用性災害時リカバリー (HADR) も、接続を要求しているリモート・データベースが宣言済みのリモート・インスタンスに属しているかどうかをチェックします。

構成タイプ
データベース

適用
- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- ローカル・クライアントを持つデータベース・サーバー

パラメーター・タイプ
構成可能

デフォルト
Null

hadr_remote_svc - HADR リモート・サービス名

このパラメーターでは、リモートの高可用性災害時リカバリー (HADR) データベース・サーバーが使用する TCP サービス名またはポート番号を指定します。

構成タイプ
データベース

適用
- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- ローカル・クライアントを持つデータベース・サーバー

パラメーター・タイプ
構成可能

デフォルト
Null

hadr_syncmode - 対等状態にあるログ書き込みのための HADR 同期モード

このパラメーターでは同期モードを指定します。システムどうしが対等状態にある場合に、 1 次ログ書き込みをスタンバイと同期する方法を指示します。

構成タイプ

データベース

適用

- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- ローカル・クライアントを持つデータベース・サーバー

パラメーター・タイプ

構成可能

デフォルト [範囲]

NEARSYNC [ASYNC; SYNC]

このパラメーターの有効な値は、次のとおりです。

SYNC このモードはトランザクション損失に対する最大限の保護を提供しますが、トランザクション応答時間のコストも最大です。

このモードでは、ログ書き込みは、ログが 1 次データベースのログ・ファイルに書き込まれ、ログがスタンバイ・データベース上のログ・ファイルにも書き込まれたことの確認通知をスタンバイ・データベースから 1 次データベースが受信した場合にのみ、成功したものと考えられます。ログ・データは、確実に両方のサイトに保管されます。

NEARSYNC

このモードは、トランザクション損失に対する保護が幾分緩くなる代わりに、 SYNC モードよりもトランザクション応答時間が短くなります。

このモードでは、ログ書き込みは、ログ・レコードが 1 次データベースのログ・ファイルに書き込まれ、ログがスタンバイ・システム上のメイン・メモリーにも書き込まれたことの確認通知をスタンバイ・システムから 1 次データベースが受信した場合にのみ、成功したものと考えられます。データの消失は、両方のサイトに同時に障害が発生し、ターゲット・サイトが、受信したすべてのログ・データを不揮発性ストレージに転送していない場合にのみ生じます。

ASYNC

このモードは 1 次障害時にトランザクション損失の確率が最も高くなる代わりに、 3 種類のモードの中ではトランザクション応答時間が最も短くなります。

このモードでは、ログ書き込みは、ログ・レコードが 1 次データベース上のログ・ファイルに書き込まれ、 1 次システムのホスト・マシンの TCP レイヤーに送信される場合にのみ、成功したものと考えられます。 1 次システムはスタンバイ・システムからの確認通知を待たないため、トランザクションがスタンバイ・システムへの送信途上であっても、コミット済みであると見なされる場合があります。

hadr_timeout - HADR タイムアウト値

このパラメーターでは、高可用性災害時リカバリー (HADR) プロセスが通信の試行が失敗したと判断するまでに待機する時間 (秒数) を指定します。

構成タイプ
データベース

適用

- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- ローカル・クライアントを持つデータベース・サーバー

パラメーター・タイプ
構成可能

デフォルト [範囲]
120 [1 - 4 294 967 295]

jdk_64_path - 64 ビット Software Developer's Kit for Java インストール・パス DAS

このパラメーターは、DB2 Administration Server 関数を実行するために使用する 64 ビット Software Developer's Kit (SDK) for Java がインストールされているディレクトリーを指定します。

構成タイプ
DB2 Administration Server

適用 DB2 Administration Server

パラメーター・タイプ
オンラインで構成可能

伝搬クラス
即時

デフォルト [範囲]
NULL [任意の有効なパス]

注: これは、32 ビットの SDK for Java を指定する **jdk_path** 構成パラメーターとは異なります。

Java インタープリターで使用される環境変数は、このパラメーターの値から計算されます。このパラメーターは、32 ビットと 64 ビットの両方のインスタンスをサポートするプラットフォームでのみ使用されます。

それらのプラットフォームは次のとおりです。

- AIX、HP-UX、および Solaris オペレーティング・システムの 64 ビット・カーネル
- X64 および IPF 上の 64 ビット Windows
- AMD64 および Intel® EM64T システム (x64)、POWER、および zSeries 上の 64 ビット Linux カーネル

それ以外のすべてのプラットフォームでは、**jdk_path** だけが使用されます。

このパラメーターにはデフォルト値がないため、 SDK for Java のインストール時にこのパラメーターの値を指定する必要があります。

このパラメーターは、バージョン 8 コマンド行プロセッサー (CLP)からのみ更新可能です。

locklist - ロック・リスト用最大ストレージ

このパラメーターは、ロック・リストに割り振られているメモリー量を示します。データベースごとに 1 つのロック・リストが存在し、ロック・リストには、データベースに同時に接続しているすべてのアプリケーションが保持しているロックが含まれています。

構成タイプ
データベース

パラメーター・タイプ
オンラインで構成可能

伝搬クラス
即時

デフォルト [範囲]

UNIX Automatic [4 - 524 288]

ローカル・クライアントおよびリモート・クライアントを持つ Windows データベース・サーバー
Automatic [4 - 524 288]

ローカル・クライアントを持つ Windows 64 ビット・データベース・サーバー Automatic [4 - 524 288]

ローカル・クライアントを持つ Windows 32 ビット・データベース・サーバー Automatic [4 - 524 288]

単位 ページ (4 KB)

割り振られるタイミング
最初のアプリケーションがデータベースに接続する時

解放されるタイミング
最後のアプリケーションがデータベースから切断される時

ロッキングは、複数のアプリケーションがデータベース内にあるデータに並行アクセスするのをコントロールするために、データベース・マネージャーが使用するメカニズムです。行と表の両方がロックの対象です。データベース・マネージャーは Internal Lock を獲得する場合もあります。

このパラメーターは、AUTOMATIC に設定されると、セルフチューニングが使用可能になります。これによってメモリー・チューナーは、このパラメーターによって制御されるメモリー領域のサイズを、ワークロード要件の変化に応じて動的に変更できるようになります。メモリー・チューナーは異なるメモリー・コンシューマーの間でメモリー・リソースをやりとりするので、セルフチューニングをアクティブにするためには、少なくとも 2 つのメモリー・コンシューマーのセルフチューニングが使用可能になっていなければなりません。

locklist の値のチューニングは *maxlocks* パラメーターと一緒に行われるので、*locklist* 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、*maxlocks* パラメーターのセルフチューニングも自動的に使用不可になります。*locklist* パラメーターのセルフチューニングを使用可能にすると、*maxlocks* パラメーターのセルフチューニングも自動的に使用可能になります。

この構成パラメーターの自動チューニングは、データベースでセルフチューニング・メモリーが有効になっている (*self_tuning_mem* 構成パラメーターが「ON」に設定されている) 場合にのみ実行されます。

32 ビットのプラットフォームの場合は、それぞれのロックには、そのオブジェクトで他のロックが保持されているかどうかによって、48 バイトまたは 96 バイトのロック・リストが必要です。

- 他にロックが保留されていないオブジェクトのロックを保留するには、96 バイトが必要です。
- 既存のロックが保持されているオブジェクトのロックを記録するには、48 バイトが必要です。

64 ビットのプラットフォーム (HP-UX/PA-RISC を除く) の場合は、それぞれのロックには、そのオブジェクトで他のロックが保持されているかどうかによって、64 バイトまたは 128 バイトのロック・リストが必要です。

- 他にロックが保留されていないオブジェクトのロックを保留するには、128 バイトが必要です。
- 既存のロックが保持されているオブジェクトのロックを記録するには、64 バイトが必要です。

64 ビット HP-UX/PA-RISC の場合は、それぞれのロックには、そのオブジェクトで他のロックが保持されているかどうかによって、80 バイトまたは 160 バイトのロック・リストが必要です。

1 つのアプリケーションが使用するロック・リストのパーセント (%) が *maxlocks* に達すると、データベース・マネージャーは、そのアプリケーションが保持するロックに対して、行から表にロック・エスカレーションを行います。エスカレーション処理そのものにはそれほど時間がかかりませんが、(個別の行に対するロッキングに対して) 表全体のロッキングは並行性を減少させ、影響を受けた表に対する後続のアクセスのために、データベース・パフォーマンス全体が低下する可能性があります。推奨されるロック・リスト・サイズのコントロール方法は、以下のとおりです。

- ロックを解放するために頻繁に COMMIT を実行します。
- 多くの更新を実行するときには、(SQL LOCK TABLE ステートメントを使用して) 更新前に表全体をロックします。これは 1 つのロックのみを使用し、他のロックが更新に干渉しないようにしますが、データの並行性は減少します。

  また、ALTER TABLE ステートメントの LOCKSIZE オプションを使用して、特定の表のロッキング方法をコントロールすることもできます。

  反復可能読み取り分離レベルを使用すると、結果として自動表ロックになってしまう場合があります。

- 保持された共有ロック数の減少が可能な場合は、カーソル固定分離レベルを使用します。 アプリケーション整合性要件と折り合わない場合は、ロッキングの量をさらに減らすために、カーソル固定ではなく非コミット読み取りを使用してください。
- *locklist* を AUTOMATIC に設定します。ロック・リストが同期的に大きくなっていくので、ロック・エスカレーションの発生やロック・リストがいっぱいになる状態を回避できます。

ロック・リストがいっぱいになると、ロック・エスカレーションが行のロックよりも表のロックを多く行うので、パフォーマンスが低下する場合があり、これにより、データベースの共有オブジェクトにおける並行性が低下します。さらに、アプリケーション間のデッドロックが増える可能性があり (すべてのアプリケーションが、限られた数の表ロックを待つため)、これによりトランザクションがロールバックされることになります。データベースに対するロック要求の最大数に達すると、アプリケーションが SQLCODE -912 を受け取ります。

推奨 : ロック・エスカレーションによってパフォーマンスの問題が生じている場合、このパラメーターか *maxlocks* パラメーターの値を増やす必要があります。 データベース・システム・モニターを使用すると、ロック・エスカレーションが起きているかどうかを判別できます。 *lock_escals (ロック・エスカレーション)* モニター・エレメントを参照してください。

以下のステップは、ロック・リストに必要なページ数を決定するのに役立ちます。

1. ロック・リストのサイズの下限を計算します。ユーザー環境により、以下の計算式の中から *1* つを使用して計算します。

   a.
   ```
   (512 * x * maxappls) / 4096
   ```
   b. コンセントレーターが使用可能になっている場合。
   ```
   (512 * x * max_coordagents) / 4096
   ```
   c. コンセントレーターが使用可能になっているパーティション・データベースの場合。
   ```
   (512 * x * max_coordagents * データベース・パーティションの数) / 4096
   ```

   ただし、512 はアプリケーション当たりの平均ロック数の見積もりであり、x は、既存のロックがあるオブジェクトに対するそれぞれのロックに必要なバイト数 (32 ビット・プラットフォームでは 40 バイト、64 ビット・プラットフォームでは 64 バイト) です。
2. ロック・リスト・サイズの上限を計算します。
   ```
   (512 * y * maxappls) / 4096
   ```

   y はオブジェクトに対する最初のロックに必要なバイト数です。 (32 ビットのプラットフォームの場合は 80 バイトで、64 ビットのプラットフォームの場合は 128 バイトです。)
3. データに対する並列量を見積もり、また計算した上限と下限の間になるように、予測に基づいて *locklist* の初期値を選択します。
4. 以下に記述されているように、データベース・システム・モニターを使用してこのパラメーターの値を調整します。

ご使用のアプリケーション・シナリオで *maxappls* または *max_coordagents* が AUTOMATIC に設定されている場合は、*locklist* も AUTOMATIC に設定する必要があります。

データベース・システム・モニターを使用すると、指定したトランザクションによって保持される最大ロック数を判別することができます。*locks_held_top (保留されているロックの最大数)* モニター・エレメントを参照してください。

この情報は、アプリケーションあたりのロックの見積数の検査または調整に役立ちます。この妥当性検査を実行するには、モニター情報がアプリケーション・レベルではなく、トランザクション・レベルで提供されていることに注意して、複数のアプリケーションをサンプルとする必要があります。

また、*maxappls* を増やしたか、または実行中のアプリケーションが頻繁にコミットを実行していない場合は、*locklist* の増加が必要になる場合もあります。

このパラメーターを変更した場合は、アプリケーションの再バインド (REBIND コマンドを使用) を考慮してください。

locktimeout - ロック・タイムアウト

このパラメーターは、アプリケーションがロックを獲得するために待機する秒数を指定して、アプリケーションのグローバル・デッドロックを防ぎます。

構成タイプ
データベース

パラメーター・タイプ
構成可能

デフォルト [範囲]
-1 [-1; 0 - 32 767]

単位 秒

このパラメーターを `0` に設定した場合、ロックは待機されません。この状態では、要求時にロックが使用可能でない場合は、アプリケーションは即時に -911 を受け取ります。

このパラメーターを `-1` に設定する場合、ロック・タイムアウト検出はオフにされます。この状態では、次のいずれかになるまで、ロックを待ちます (ただし、要求時にロックが使用可能でない場合)。

- ロックが GRANT される
- デッドロックが発生する

推奨: トランザクション処理 (OLTP) 環境では、30 秒が初期値として適当です。照会のみの環境では、より高い値で開始できます。両方の場合とも、ベンチマーク技法を使用して、このパラメーターを調整してください。

ユーザーがワークステーションを離れたためにトランザクションが停止した場合のように、異常な状況が原因で発生している待ち状態をすみやかに検出できるような

値を設定してください。ワークロードがピークであるときに多数の待機状態のロックがあっても、有効なロック要求がタイムアウトにならないような、十分に高い値を設定すべきです。

データベース・システム・モニターを使用すると、アプリケーション (接続) がロック・タイムアウトを経験した回数、またはデータベースがすべての接続しているアプリケーションについて、タイムアウト状態を検出した回数を追跡することができます。

次のような場合は、*lock_timeout* (ロック・タイムアウトの数) モニター・エレメントの値が高くなる原因になり得ます。

- この構成パラメーターの値が低すぎます。
- ロックを保留している時間が伸びているアプリケーション (トランザクション)。データベース・システム・モニターを使用すると、これらのアプリケーションの詳細を調べることができます。
- ロック・エスカレーション (行レベル・ロックから表レベル・ロックへの) によって生じる可能性のある並行性の問題。

log_retain_status - ログ保持状況表示

設定された場合 (*logretain* パラメーターの設定が Recovery に等しい場合)、このパラメーターは、ログ・ファイルがロールフォワード・リカバリーで使用するために保持されていることを示します。

構成タイプ
データベース

パラメーター・タイプ
通知

logarchmeth1 - 1 次ログ・アーカイブ方式

このパラメーターでは、アーカイブ済みログの 1 次宛先のメディア・タイプを指定します。

構成タイプ
データベース

適用

- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- クライアント
- ローカル・クライアントを持つデータベース・サーバー
- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ
オンラインで構成可能

デフォルト [範囲]
OFF [LOGRETAIN、USEREXIT、DISK、TSM、VENDOR]

OFF ログ・アーカイブ方式が使用されないことを指定します。
logarchmeth1 と *logarchmeth2* の両方が OFF に設定される場合、デ

ータベースは、循環ロギングを使用していると見なされ、ロールフォワード・リカバリー可能ではなくなります。これはデフォルトです。

LOGRETAIN
この値は、*logarchmeth1* にのみ使用できるもので、 *logretain* 構成パラメーターを RECOVERY に設定することと同じことです。この値を指定する場合、*logretain* 構成パラメーターが自動的に更新されます。

USEREXIT
この値は、*logarchmeth1* の場合にのみ有効で、 *userexit* 構成パラメーターを ON に設定することと同じことです。この値を指定する場合、*userexit* 構成パラメーターが自動的に更新されます。

DISK この値の後に、コロン (:) を付け、その後に、ログ・ファイルがアーカイブされる完全修飾された既存のパス名を続ける必要があります。たとえば、*logarchmeth1* を DISK:/u/dbuser/archived_logs に設定する場合、アーカイブ・ログ・ファイルは /u/dbuser/archived_logs というディレクトリーに置かれます。

注: テープにアーカイブする予定の場合、db2tapemgr ユーティリティーを使用して、ログ・ファイルを保管および検索できます。

TSM 追加の構成パラメーターなしで指定される場合、この値は、ログ・ファイルはデフォルト管理クラスを使用してローカル TSM サーバーにアーカイブされることを示します。その後にコロン (:) と TSM 管理クラスが続けられる場合、ログ・ファイルは、指定された管理クラスを使用してアーカイブされます。

VENDOR
ベンダー・ライブラリーを使用してログ・ファイルをアーカイブすることを指定します。この値の後に、コロン (:) とライブラリーの名前を続ける必要があります。ライブラリーで提供される API は、ベンダー製品のバックアップおよびリストア API を使用する必要があります。

注:

1. *logarchmeth1* または *logarchmeth2* のいずれかが OFF 以外の値に設定される場合、データベースはロールフォワード・リカバリー用に構成されます。
2. *userexit* または *logretain* 構成パラメーターを更新すると、*logarchmeth1* は自動的に更新されます (その逆もあります)。しかし、*userexit* または *logretain* のいずれかを使用する場合、*logarchmeth2* は OFF に設定する必要があります。

logarchmeth2 - 2 次ログ・アーカイブ方式

このパラメーターでは、アーカイブ済みログの 2 次宛先のメディア・タイプを指定します。

構成タイプ
データベース

適用

- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- クライアント
- ローカル・クライアントを持つデータベース・サーバー
- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ

オンラインで構成可能

デフォルト [範囲]

OFF [LOGRETAIN、USEREXIT、DISK、TSM、VENDOR]

OFF ログ・アーカイブ方式が使用されないことを指定します。*logarchmeth1* と *logarchmeth2* の両方が OFF に設定される場合、データベースは、循環ロギングを使用していると見なされ、ロールフォワード・リカバリー可能ではなくなります。これはデフォルトです。

LOGRETAIN

この値は、*logarchmeth1* にのみ使用できるもので、 *logretain* 構成パラメーターを RECOVERY に設定することと同じことです。この値を指定する場合、*logretain* 構成パラメーターが自動的に更新されます。

USEREXIT

この値は、*logarchmeth1* の場合にのみ有効で、 *userexit* 構成パラメーターを ON に設定することと同じことです。この値を指定する場合、*userexit* 構成パラメーターが自動的に更新されます。

DISK この値の後に、コロン (:) を付け、その後に、ログ・ファイルがアーカイブされる完全修飾された既存のパス名を続ける必要があります。たとえば、*logarchmeth1* を DISK:/u/dbuser/archived_logs に設定する場合、アーカイブ・ログ・ファイルは /u/dbuser/archived_logs というディレクトリーに置かれます。

注: テープにアーカイブする予定の場合、db2tapemgr ユーティリティーを使用して、ログ・ファイルを保管および検索できます。

TSM 追加の構成パラメーターなしで指定される場合、この値は、ログ・ファイルはデフォルト管理クラスを使用してローカル TSM サーバーにアーカイブされることを示します。その後にコロン (:) と TSM 管理クラスが続けられる場合、ログ・ファイルは、指定された管理クラスを使用してアーカイブされます。

VENDOR

ベンダー・ライブラリーを使用してログ・ファイルをアーカイブすることを指定します。この値の後に、コロン (:) とライブラリーの名前を続ける必要があります。ライブラリーで提供される API は、ベンダー製品のバックアップおよびリストア API を使用する必要があります。

注:

1. *logarchmeth1* または *logarchmeth2* のいずれかが OFF 以外の値に設定される場合、データベースはロールフォワード・リカバリー用に構成されます。
2. *userexit* または *logretain* 構成パラメーターを更新すると、*logarchmeth1* は自動的に更新されます (その逆もあります)。しかし、*userexit* または *logretain* のいずれかを使用する場合、*logarchmeth2* は OFF に設定する必要があります。

このパスが指定されている場合、ログ・ファイルはこの宛先と *logarchmeth1* データベース構成パラメーターに指定されている宛先の両方にアーカイブされます。

logarchopt1 - 1 次ログ・アーカイブ・オプション

このパラメーターでは、アーカイブ済みログの 1 次宛先のオプション・フィールドを指定します (必要な場合)。

構成タイプ
データベース

適用
- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- クライアント
- ローカル・クライアントを持つデータベース・サーバー
- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ
オンラインで構成可能

デフォルト [範囲]
NULL [適用外]

logarchopt2 - 2 次ログ・アーカイブ・オプション

このパラメーターでは、アーカイブ済みログの 2 次宛先のオプション・フィールドを指定します (必要な場合)。

構成タイプ
データベース

適用
- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- クライアント
- ローカル・クライアントを持つデータベース・サーバー
- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ
オンラインで構成可能

デフォルト [範囲]
NULL [適用外]

logbufsz - ログ・バッファー・サイズ

このパラメーターによって、*dbheap* パラメーターによって定義されるデータベース・ヒープの中から、ログ・バッファーのサイズを指定します。ログ・バッファーはログ・レコードをディスクに書き込む前に、ログ・レコードのバッファーとして使用されます。

構成タイプ
データベース

パラメーター・タイプ
構成可能

デフォルト [範囲]

32 ビット・プラットフォーム
8 [4 - 4 096]

64 ビット・プラットフォーム
8 [4 - 65 535]

単位 ページ (4 KB)

以下のいずれかのときに、ログ・レコードがディスクに書き込まれます。

- *mincommit* 構成パラメーターによって定義されたように、トランザクションのコミット、またはトランザクションのグループのコミットが行われたとき。
- ログ・バッファーがいっぱいになったとき。
- その他のいくつかの内部データベース・マネージャー・イベントの結果として。

また、このパラメーターは *dbheap* パラメーター以下でなければなりません。ログ・レコードのバッファリングは、ログ・レコードのディスクへの書き込み頻度が少なくなり、一度により多くのログ・レコードが書き込まれるため、より効率的なロギング・ファイル入出力になります。

推奨: 専用ログ・ディスクに相当数の読み取り活動がある場合、またはディスクの使用率が高い場合、バッファー域のサイズを大きくしてください。 このパラメーターの値を増やす場合は、ログ・バッファー域が、 *dbheap* パラメーターでコントロールされるスペースを使用するので、*dbheap* パラメーターも考慮してください。

データベース・システム・モニターを使用すると、特定トランザクション (または作業単位) で使用されたログ・バッファー・スペースの量を判別できます。*log_space_used* (使用される作業単位ログ・スペース) モニター・エレメントを参照してください。

logfilsiz - ログ・ファイルのサイズ

このパラメーターは、1 次および 2 次ログ・ファイルの各サイズを定義します。これらのログ・ファイルのサイズは、ファイルがいっぱいになり、新しいログ・ファイルが必要になる前に書き込めるログ・レコード数を制限します。

構成タイプ
データベース

パラメーター・タイプ
構成可能

デフォルト [範囲]

UNIX 1000 [4 - 524 286]

Windows

1000 [4 - 524 286]

単位 ページ (4 KB)

1 次および 2 次ログ・ファイルの使用は、ログ・ファイルがいっぱいになったときに実行されるアクションと同様、実行されているロギングのタイプによって次のように異なります。

- 循環ロギング

  1 次ログ・ファイルは、そのファイルに記録された変更がコミットされたときに再利用可能になります。ログ・ファイルのサイズが小さく、しかもアプリケーションが変更をコミットせずにデータベースへの変更を頻繁に処理すると、1 次ログ・ファイルがすぐにいっぱいになる可能性があります。 すべての 1 次ログ・ファイルがいっぱいになると、データベース・マネージャーは新しいログ・レコードを保留するために 2 次ログ・ファイルを割り振ります。

- ログ保持ロギング

  1 次ログ・ファイルがいっぱいになると、ログがアーカイブされ、新しい 1 次ログ・ファイルが割り振られます。

推奨: ログ・ファイルのサイズについては、次のように 1 次ログ・ファイルの数とのバランスを取る必要があります。

- データベースに対して行われる、ログ・ファイルを早急にいっぱいにする更新、削除または挿入トランザクションの数が多い場合は、 *logfilsiz* の値を増やす必要があります。

  注: ログ・ファイル・サイズの最大値とログ・ファイルの数 (*logprimary* + *logsecond*) の最大値まで使用すると、全体のアクティブ・ログ・スペースも 512 GB という最大値になります。

  ログ・ファイルが小さすぎると、古いログ・ファイルのアーカイブ、新しいログ・ファイルの割り振り、および使用可能ログ・ファイルの待機などのオーバーヘッドのために、システム・パフォーマンスに影響を与える場合があります。

- ディスク・スペースが不足している場合、1 次ログはこのサイズで事前割り振りされるので、*logfilsiz* の値を減らす必要があります。

  ログ・ファイルが大きすぎると、いくつかのメディアではログ・ファイル全体を保存できない可能性があるので、アーカイブ済みログ・ファイルおよびログ・ファイルのコピーの管理上、柔軟性が失われる可能性があります。

ログ保持を使用している場合、最後のアプリケーションがデータベースから切断されると、現在のアクティブ・ログ・ファイルはクローズされ、切り捨てられます。データベースへの接続が行われると、次のログ・ファイルが使用されます。したがって、並行アプリケーションのロギング要件がはっきりしている場合は、余分なスペースを割り振らないログ・ファイル・サイズを判別できる可能性があります。

loghead - 最初のアクティブ・ログ・ファイル

このパラメーターには、現在アクティブなログ・ファイルの名前が含まれます。

構成タイプ
データベース

パラメーター・タイプ
通知

logindexbuild - 作成されるログ索引ページ

このパラメーターでは、索引の作成、再作成、または再編成の操作をログに記録しておき、 DB2 ロールフォワード操作中または高可用性災害時リカバリー (HADR) ログ再生プロシージャー中に索引を再構成できるようにするかどうかを指定します。

構成タイプ
データベース

適用

- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- クライアント
- ローカル・クライアントを持つデータベース・サーバー
- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ
オンラインで構成可能

デフォルト [範囲]
Off [On; Off]

logpath - ログ・ファイルのロケーション

このパラメーターには、ロギング目的で使用されている現行パスが含まれます。

構成タイプ
データベース

パラメーター・タイプ
通知

このパラメーターは、*newlogpath* パラメーターに対する変更が有効になった後で、データベース・マネージャーによって設定されているので、直接変更することはできません。

データベースが作成されると、データベースのリカバリー・ログ・ファイルが、データベースが入っているディレクトリーのサブディレクトリーに作成されます。デフォルトでは、データベースを入れるために作成されたディレクトリーの下にある SQLOGDIR という名前のサブディレクトリーです。

logprimary - 1 次ログ・ファイル数

このパラメーターを使用すると、事前割り振りされる 1 次ログ・ファイルの数を指定できます。 1 次ログ・ファイル数によって、リカバリー・ログ・ファイルに割り振られる固定量のストレージが設定されます。

構成タイプ
データベース

パラメーター・タイプ
構成可能

デフォルト [範囲]
3 [2 - 256]

単位 カウンター

割り振られるタイミング

- データベースが作成されるとき
- ログが別のロケーションに移動されるとき (これが発生するのは、*logpath* パラメーターの更新時)
- すべてのユーザーが切断した後の次回データベース接続中に、このパラメーター (*logprimary*) の値を大きくした後
- ログ・ファイルがアーカイブされて、新規ログ・ファイルが割り振られるとき (*logretain* または *userexit* パラメーターが使用可能になっている必要があります)
- *logfilsiz* パラメーターが変更されている場合は、アクティブ・ログ・ファイルは、すべてのユーザーが切断された後の次回データベース接続中にサイズ変更されます。

解放されるタイミング
このパラメーターの値が小さくならない限り、解放されることはありません。パラメーターの値を小さくした場合は、データベースへの次回接続時に、不要のログ・ファイルが削除されます。

循環ロギング下では、1 次ロギングが順次繰り返して使用されます。つまり、あるログがいっぱいになると、そのシーケンス内の次の 1 次ログが使用可能であれば、それが使用されることになります。ログが使用可能と見なされるのは、ログの中にログ・レコードがある作業単位がすべてコミットおよびロールバックされている場合です。シーケンス内の次の 1 次ログが使用不可の場合は、2 次ログが割り振られて使用されます。シーケンス内の次の 1 次ログが使用可能になるか、*logsecond* パラメーターによって設けられた限度に達するまで、追加の 2 次ログが割り振られて使用されます。これらの 2 次ログ・ファイルは、データベース・マネージャーで必要とされなくなると、動的に割り振り解除されます。

1 次ログ・ファイルと 2 次ログ・ファイルの数は、次の式に適合する必要があります。

- *logsecond* の値が -1 の場合、*logprimary* <= 256。
- *logsecond* の値が -1 以外の場合、 (*logprimary* + *logsecond*) <= 256。

推奨: このパラメーターの値として選択される値は、使用されるロギングのタイプ、ログ・ファイルのサイズ、および処理環境のタイプ (例えば、トランザクションの長さやコミットの頻度など) を含む、数多くの要因に応じて決まります。

この値を大きくすると、ログ用のディスク所要量が増えます。 1 次ログ・ファイルは、データベースへの最初の接続時に事前割り振りされるからです。

2 次ログ・ファイルが割り振られる頻度が高いと思われる場合は、ログ・ファイル・サイズ (*logfilsiz*) を大きくするか、1 次ログ・ファイルの数を増やすことで、システム・パフォーマンスを上げることもできます。

データベースにアクセスする頻度が高くない場合は、ディスク・ストレージを節約するために、このパラメーターを 2 に設定します。ロールフォワード・リカバリーが使用可能になっているデータベースの場合は、ほとんど即時に新しいログを割り振るオーバーヘッドを回避するために、このパラメーターはもっと大きい値に設定します。

1 次ログ・ファイルをサイズ変更するには、データベース・システム・モニターを使用できます。ある期間にわたって次のモニター値を監視すると、進行中の要件は平均値に表れる場合が多いので、優れたチューニングの決定に役立ちます。

- *sec_log_used_top* (使用される最大 2 次ログ・スペース)
- *tot_log_used_top* (使用される最大合計ログ・スペース)
- *sec_logs_allocated* (現在割り振られている 2 次ログ数)

logretain - ログ保持使用可能

このパラメーターはバージョン 9.5 では推奨されませんが、バージョン 9.5 より前のデータ・サーバーおよびクライアントでは引き続き使用されています。DB2 バージョン 9.5 のデータベース・マネージャーは、この構成パラメーターに対して指定された値をすべて無視します。

注: 以下の情報は、バージョン 9.5 より前のデータ・サーバーおよびクライアントだけに当てはまります。

このパラメーターにより、アクティブ・ログ・ファイルが保持され、ロールフォワード・リカバリーで使用できるかどうかを決定します。

構成タイプ
データベース

パラメーター・タイプ
構成可能

デフォルト [範囲]
No [Recovery; No]

値は以下のとおりです。

- No は、ログが保持されないことを示します。
- Recovery は、ログが保持され、順方向リカバリーに使用できることを示します。

logretain が Recovery に設定されているか、または *userexit* が Yes に設定されている場合は、アクティブ・ログ・ファイルは保持され、ロールフォワード・リカバリーで使用するためのオンライン・アーカイブ・ログ・ファイルになります。これは、ログ保持ロギングと呼ばれます。

logretain が Recovery に設定された後、または *userexit* が Yes に設定された後 (あるいはその両方) は、データベースの全バックアップを行う必要があります。この状態は、*backup_pending* フラグ・パラメーターによって示されます。

注:

logarchmeth1 または *logretain* のどちらもロールフォワード・リカバリーを有効にします。しかし、データベースに対して一度に有効にするメソッドは 1 つだけにしてください。

logarchmeth1 を使用する場合には、*logretain* と *userexit* は設定しないでください。*logretain* を使用すると、*logarchmeth1* の値は自動的に *logretain* に設定されます。

アーカイブ・ロギングおよびロールフォワード・リカバリーを活動化するには、*logretain* および USEREXIT を使用するのではなく、*logarchmeth1* (および *logarchmeth2*) を使用するようお勧めします。*logretain* および *userexit* オプションは、*logarchmeth1* にまだマイグレーションしていないユーザーに対して引き続きサポートされています。

logsecond - 2 次ログ・ファイル数

このパラメーターは、リカバリー・ログ・ファイルとして作成および使用される (必要な場合のみ) 2 次ログ・ファイルの数を指定します。

構成タイプ
データベース

パラメーター・タイプ
オンラインで構成可能

伝搬クラス
即時

デフォルト [範囲]
2 [-1; 0 – 254]

単位 カウンター

割り振られるタイミング
logprimary が不十分なとき、必要に応じて割り振り (下記の詳細を参照してください)

解放されるタイミング
データベース・マネージャーが不要だと決定した際に、徐々に解放

1 次ログ・ファイルがいっぱいになると、2 次ログ・ファイル (サイズ *logfilsiz*) が、必要に応じて一度に 1 つずつ、このパラメーターがコントロールする最大数まで割り振られます。 2 次ログ・ファイルがこのパラメーターによって許可されている数を超えて必要とされた場合は、アプリケーションにエラー・コードが戻され、データベースはシャットダウンされます。

logsecond を -1 に設定した場合は、データベースは、アクティブ・ログ・スペースが無限として構成されます。したがって、データベース上で実行中の未完了トランザクションのサイズや数に制限はありません。 *logsecond* を -1 に設定した場合でも、データベース・マネージャーがアクティブ・ログ・パスに維持する必要があるログ・ファイルの数を指定する場合は、やはり構成パラメーター *logprimary* および *logfilsiz* を使用します。データベース・マネージャーはログ・ファイルからログ・データを読み取る必要がありますが、そのログ・ファイルがアクティブ・ログ・パスにない場合は、データベース・マネージャーはそのログ・ファイルをアーカイブからアクティブ・ログ・パスに取り出します。(データベース・マネージャーがファイルをオーバーフロー・ログ・パスに取り出すのは、オーバーフロー・ログ・パスが構成してある場合です) ログ・ファイルが取り出されると、データベース・マネージャーはこのファイルをアクティブ・ログ・パス内でキャッシュに入れるので、他に同じファイルからログ・データを読み取る場合は、読み取りが高速化されます。データベース・マネージャーは、必要に応じて、これらのログ・ファイルの検索、キャッシング、および除去を管理します。

ログ・パスがロー・デバイスである場合は、*logsecond* を -1 に設定するためには、*overflowlogpath* 構成パラメーターを構成する必要があります。

logsecond を -1 に設定することによって、作業単位のサイズにも並行作業単位の数にも制限がなくなります。ただし、アーカイブからログ・ファイルを取り出す必要があるため、ロールバック (セーブポイント・レベルと作業単位レベルの両方での) が非常に遅くなる可能性があります。クラッシュ・リカバリーも同じ理由で非常に遅くなる恐れがあります。データベース・マネージャーは、管理通知ログにメッセージを書き込んで、現行セットのアクティブ作業単位が 1 次ログ・ファイル数を超えていることを警告します。これは、ロールバックやクラッシュ・リカバリーが極端に遅くなる恐れがあることを示します。

logsecond を -1 に設定するには、*logarchmeth1* 構成パラメーターを OFF または LOGRETAIN 以外の値に設定する必要があります。

推奨: データベースが周期的に大量のログ・スペースを必要とする場合は、2 次ログ・ファイルを使用してください。例えば、1 カ月に一度実行されるアプリケーションの場合は、1 次ログ・ファイルで用意されているログ・スペースを超えるスペースが必要になる可能性があります。2 次ログ・ファイルでは永続ファイル・スペースを必要としないので、こうした状況では利点になります。

max_log - トランザクション当たりの最大ログ

このパラメーターは、トランザクションが消費できるログ・スペースのパーセントに制限を設けるかどうか、およびその制限が何パーセントであるかを指定します。

構成タイプ
データベース

パラメーター・タイプ
オンラインで構成可能

伝搬クラス
即時

デフォルト [範囲]
0 [0 — 100]

単位 パーセント

この値が 0 ではない場合、このパラメーターは、 1 つのトランザクションで消費できる 1 次ログ・スペースのパーセントを示します。

値を 0 に設定すると、 1 つのトランザクションで消費できるスペース (1 次ログ・ログ・スペース総量のパーセントとして) に制限がなくなります。これは、バージョン 8 より前のトランザクションの動作です。

maxappls - アクティブ・アプリケーションの最大数

このパラメーターは、データベースに (ローカルとリモートの両方で) 接続できる並行アプリケーションの最大数を指定します。データベースにアタッチしている各アプリケーションが、いくつかの専用メモリーを割り振るので、同時アプリケーションの数を多くすると、多くのメモリーが使用される可能性があります。

構成タイプ
データベース

パラメーター・タイプ
オンラインで構成可能

伝搬クラス
即時

デフォルト [範囲]
Automatic [1 - 60 000]

単位 カウンター

maxappls を *Automatic* に設定すると、接続されたアプリケーションを幾つでも任意の数だけ使用できるという効果が生じます。 データベース・マネージャーでは、新しいアプリケーションをサポートするために必要なリソースを動的に割り振ります。

このパラメーターを *Automatic* に設定したくない場合は、このパラメーターの値は、接続されたアプリケーションの数に、これらと同じアプリケーションで 2 フェーズ・コミットおよびロールバックを完了する処理で同時に実行される数を加えた合計に等しいか、それよりも大きくなければなりません。次に、どのようなときでも発生する可能性がある未確定トランザクション数をこの合計に追加します。

アプリケーションがデータベースへの接続を試みたときに、*maxappls* の値にすでに達していた場合は、すでに最大数のアプリケーションがデータベースに接続されていることを示すエラーが、アプリケーションに返されます。

パーティション・データベース環境下では、これはデータベース・パーティションに対して並行してアクティブであるアプリケーションの最大数です。このパラメーターは、サーバーがアプリケーションのコーディネーター・プログラム・ノードであるかどうかにかかわらず、データベース・パーティション・サーバー上のデータベース・パーティションに対するアクティブなアプリケーションの数を制限します。パーティション・データベース環境下のカタログ・ノードでは、この環境下の

すべてのアプリケーションがカタログ・ノードへの接続を要求するため、他のタイプの環境の場合よりも高い *maxappls* の値が必要です。

推奨: *maxlocks* パラメーターを低くしない、もしくは *locklist* パラメーターを増やさずに、このパラメーターの値を増やすと、アプリケーション限界ではなくロック (*locklist*) のデータベース限界に達し、その結果ロック・エスカレーションの問題が広がることになります。

ある程度は、アプリケーションの最大数も *max_coordagents* によって決まります。使用可能な接続 (*maxappls*) に加えて、使用可能なコーディネーター・エージェント (*max_coordagents*) が存在する場合のみ、アプリケーションはデータベースに接続できます。

maxfilop - アプリケーション単位の最大データベース・ファイル・オープン数

このパラメーターは、それぞれのデータベースごとにオープンしておけるファイル・ハンドルの最大数を指定します。

構成タイプ
データベース

パラメーター・タイプ
オンラインで構成可能

伝搬クラス
トランザクション境界

デフォルト [範囲]

AIX、Sun、HP、および Linux 64 ビット
61 440 [64 - 61 440]

Linux 32 ビット
30 720 [64 - 30 720]

Windows 32 ビット
32 768 [64 - 32 768]

Windows 64 ビット
65 335 [64 - 65 335]

単位 カウンター

ファイルを 1 つオープンしただけでこの値を超える場合は、このデータベースで使用中の一部のファイルがクローズされます。*maxfilop* が小さ過ぎる場合は、ファイルをオープンしたり、クローズしたりするオーバーヘッドが過剰になり、パフォーマンスを低下させる可能性があります。

オペレーティング・システムとデータベース・マネージャーの間では、 SMS 表スペースと DMS 表スペース・ファイル・コンテナーが両方ともファイルとして処理されるので、ファイル・ハンドルが必須です。SMS 表スペースでは、DMS ファイル表スペースの場合に使用されるコンテナーの数に比べて、一般的には、多くのフ

ファイルが使用されます。したがって、SMS 表スペースを使用している場合は、DMS ファイル表スペースの場合に必要とする値に比べて大きな値が、このパラメーターに必要になります。

また、このパラメーターを使用して、データベース単位のファイル・ハンドル数を一定に制限することによって、データベース・マネージャーで使用されるファイル・ハンドルの総計がオペレーティング・システム限度を超えることのないようにすることもできます。なお、実際の数は、同時に稼働するデータベースの数に応じて異なります。

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

このパラメーターは、アプリケーションが保持するロック・リストのパーセンテージを定義します。これに達するとデータベース・マネージャーがロック・エスカレーションを実行します。

構成タイプ
データベース

パラメーター・タイプ
オンラインで構成可能

伝搬クラス
即時

デフォルト [範囲]

UNIX Automatic [1 - 100]

Windows
Automatic [1 - 100]

単位 パーセント

ロック・エスカレーションとは、行ロックを表ロックで置き換えて、リスト内のロック数を減らす処理のことです。ある 1 つのアプリケーションによって保持されているロックの数が、合計ロック・リスト・サイズに対してこのパーセントに達すると、そのアプリケーションによって保持されているロックに関してロック・エスカレーションが行われます。ロック・リストがスペースを使い尽くした場合も、ロック・エスカレーションが発生します。

データベース・マネージャーは、アプリケーションのロック・リストを調べ、行ロック数が最も多い表を検索して、ロック・エスカレーションの対象となるロックを判別します。行ロックを単一の表ロックで置き換えた後、*maxlocks* 値を超えることがなくなっていれば、ロック・エスカレーションは停止します。そうならない場合は、保持されているロック・リストのパーセンテージが *maxlocks* の値より低くなるまで、ロック・エスカレーションは続きます。*maxlocks* パラメーターに *maxappls* パラメーターを掛けた値が 100 より小さな値であってはなりません。

このパラメーターは、AUTOMATIC に設定されると、セルフチューニングが使用可能になります。これによってメモリー・チューナーは、このパラメーターによって制御されるメモリー領域のサイズを、ワークロード要件の変化に応じて動的に変更できるようになります。メモリー・チューナーは異なるメモリー・コンシューマー

の間でメモリー・リソースをやりとりするので、セルフチューニングをアクティブにするためには、少なくとも 2 つのメモリー・コンシューマーのセルフチューニングが使用可能になっていなければなりません。

locklist の値のチューニングは *maxlocks* パラメーターと一緒に行われるので、*locklist* 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、*maxlocks* パラメーターのセルフチューニングも自動的に使用不可になります。*locklist* パラメーターのセルフチューニングを使用可能にすると、*maxlocks* パラメーターのセルフチューニングも自動的に使用可能になります。

この構成パラメーターの自動チューニングは、データベースでセルフチューニング・メモリーが有効になっている (*self_tuning_mem* 構成パラメーターが「ON」に設定されている) 場合にのみ実行されます。

32 ビットのプラットフォームの場合は、それぞれのロックには、そのオブジェクトで他のロックが保持されているかどうかによって、48 バイトまたは 96 バイトのロック・リストが必要です。

- 他にロックが保留されていないオブジェクトのロックを保留するには、96 バイトが必要です。
- 既存のロックが保持されているオブジェクトのロックを記録するには、48 バイトが必要です。

64 ビットのプラットフォーム (HP-UX/PA-RISC を除く) の場合は、それぞれのロックには、そのオブジェクトで他のロックが保持されているかどうかによって、64 バイトまたは 128 バイトのロック・リストが必要です。

- 他にロックが保留されていないオブジェクトのロックを保留するには、128 バイトが必要です。
- 既存のロックが保持されているオブジェクトのロックを記録するには、64 バイトが必要です。

64 ビット HP-UX/PA-RISC の場合は、それぞれのロックには、そのオブジェクトで他のロックが保持されているかどうかによって、80 バイトまたは 160 バイトのロック・リストが必要です。

推奨: 次の公式を使用すると、アプリケーションが平均ロック数の 2 倍のロックを保留できるように、*maxlocks* を設定できます。

```
maxlocks = 2 * 100 / maxappls
```

ただし、平均の 2 倍を達成するために 2 が使用され、100 は許容最大パーセント値を表します。同時に実行するアプリケーションの数が少ない場合は、上記の公式に代えて次の公式を使用できます。

```
maxlocks = 2 * 100 / (同時に実行しているアプリケーションの平均数)
```

maxlocks の設定時に考慮する必要のある事項の 1 つに、ロック・リスト (*locklist*) のサイズとの併用があります。ロック・エスカレーションが行われる前にアプリケーションが保持するロックの数の実際の限度は、次のとおりです。

- `maxlocks * locklist * 4096 / (100 * 48)` (32 ビット・システム)
- `maxlocks * locklist * 4096 / (100 * 80)` (64 ビット・システム HP-UX/PA-RISC 環境)

- `maxlocks * locklist * 4096 / (100 * 64)` (64 ビット・システム)

ただし、`4096` は 1 ページのバイト数、`100` は *maxlocks* に許容される最大のパーセント値、`48` は 32 ビット・システムの場合の 1 つのロック当たりのバイト数、`80` は HP-UX/PA-RISC 64 ビット・システムの場合の 1 つのロック当たりのバイト数、`64` はその他の 64 ビット・システムの場合の 1 つのロック当たりのバイト数です。 アプリケーションの 1 つで 1000 ロックが必要であることが分かっていて、しかもロック・エスカレーションが行われないようにする場合は、この公式の *maxlocks* および *locklist* の値を選択すれば、結果は 1000 より大になります。(*maxlocks* に 10 を、*locklist* に 100 を使用すると、この公式によって、必要な 1000 ロックより大という結果が得られます。)

maxlocks の設定が低過ぎると、他の同時アプリケーションにまだ十分のロック・スペースがあるときに、ロック・エスカレーションが行われます。*maxlocks* の設定が高過ぎると、一部のわずかなアプリケーションだけでロック・スペースのほとんどを使用してしまう可能性があり、それ以外のアプリケーションではロック・エスカレーションを実行する必要が生じます。この場合にロック・エスカレーションが必要になるということは、並行性が低下する結果になります。

データベース・システム・モニターを使用すると、この構成パラメーターの追跡および調整ができます。

min_dec_div_3 - 10 進数除算の位取り 3

このパラメーターは、SQL での 10 進数除算の位取りの計算への変更を可能にする簡易な手段として用意されています。

構成タイプ
データベース

パラメーター・タイプ
構成可能

デフォルト [範囲]
No [Yes, No]

min_dec_div_3 は、10 進数除算の位取りを変更するデータベース構成パラメーターです。これは「Yes」または「No」に設定可能です。*min_dec_div_3* のデフォルト値は「No」です。値が「No」の場合は、位取りは 31-p+s-s' として計算されます。値が「Yes」に設定されている場合、位取りは MAX(3, 31-p+s-s') として計算されます。したがって、10 進数の除算の結果は、常に位取りが少なくとも 3 になります。精度は常に 31 です。

このデータベース構成パラメーターを変更すると、既存のデータベースで使用するアプリケーションに変更がもたらされる場合があります。これが起こる可能性があるのは、このデータベース構成パラメーターを変更することによって、10 進数除算の結果の位取りが影響を受けるような場合です。アプリケーションに影響する可能性があるシナリオをいくつか下にリストしてあります。これらのシナリオについては、既存のデータベースがあるデータベース・サーバー上の *min_dec_div_3* を変更する前に考慮する必要があります。

- ビュー列の結果の位取りの 1 つが変更された場合、設定が 1 つしかない環境内で定義されているビューは、データベース構成パラメーターの変更後に参照され

ると、SQLCODE -344 を出して失敗する可能性があります。メッセージ SQL0344N は再帰的共通表式を識別しますが、オブジェクト名 (最初のトークン) がビューの場合は、そのビューをドロップした上で、再度作成してこのエラーを回避する必要があります。

- 静的パッケージの振る舞いは、暗黙的あるいは明示的に、パッケージが再バインドされるまで変わりません。例えば、値を NO から YES に変更した後は、再バインドが行われるまでは、追加の位取りの数字は結果に組み込まれない可能性があります。変更後の静的パッケージの場合は、明示的 REBIND コマンドを使用して、バインドを強制できます。
- 10 進数除算に関係するチェック制約によって、以前は受け入れられていた一部の値が制約される場合があります。そのような行は現在では制約に違反しますが、チェック制約行に関係する列の 1 つが更新されるか、IMMEDIATE CHECKED オプションを指定した SET INTEGRITY ステートメントが処理されるまでは検出されません。そのような制約のチェックを強制的に行うためには、チェック制約をドロップするために ALTER TABLE ステートメントを実行してから、ALTER TABLE ステートメントを実行して再度制約を追加します。

注: *min_dec_div_3* には、次の制限もあります。

1. コマンド `GET DB CFG FOR DBNAME` では *min_dec_div_3* の設定は表示されません。10 進数除算の結果の副次作用を監視するのが、現行設定を判別する最良の方法です。例えば、以下のステートメントを検討してみましょう。

   ```
   VALUES (DEC(1,31,0)/DEC(1,31,5))
   ```

   このステートメントが sqlcode SQL0419N を戻した場合は、データベースには *min_dec_div_3* のサポートがないか、「No」に設定されています。ステートメントが `1.000` を戻した場合は、*min_dec_div_3* は「Yes」に設定されています。
2. *min_dec_div_3* は、次のコマンドを実行したときは構成キーワードのリストに示されません: `? UPDATE DB CFG`

mincommit - グループへのコミット数

このパラメーターを使うと、最低限の回数のコミットが実行されるまで、ログ・レコードのディスク書き込みを遅らせることができるようになり、ログ・レコードへの書き込みに関連するデータベース・マネージャーのオーバーヘッドを減らすことができます。

構成タイプ
データベース

パラメーター・タイプ
オンラインで構成可能

伝搬クラス
即時

デフォルト [範囲]
1 [1 – 25]

単位 カウンター

この遅延により、データベースに対して実行中のアプリケーションが複数あり、非常に短い時間内に数多くのコミットがアプリケーションによって要求されるときは、パフォーマンスが上がります。

こうしたコミットのグループ化が行われるのは、このパラメーターの値が 1 より大きい場合で、かつデータベースに接続されているアプリケーションの数がこのパラメーターの値より大か等しい場合だけです。コミットのグループ化が実行されているときは、1 秒が経過するか、コミット要求の数がこのパラメーターの値に等しくなるか、いずれかが生じるまで、アプリケーションのコミット要求は保留される可能性があります。

このパラメーターは、少しずつ (1 ずつなど) 増分すべきです。また、マルチユーザー・テストも使用して、このパラメーターの値を増やして期待される結果が得られるかどうかを検証してください。

このパラメーターに指定された値に対する変更は、即時に有効になります。したがって、すべてのアプリケーションがデータベースから切断されるまで待つ必要はありません。

推奨: 複数の読み取り/書き込みアプリケーションで一般的に並行データベース・コミットが要求される場合は、このパラメーターをデフォルト値から大きくしてください。そうすれば、ロギングが行われる頻度が低くなり、ロギングが行われる度に書き込まれるログ・レコード数が増えるので、ファイル入出力のロギング効率が上がります。

1 秒当たりのトランザクション数をサンプリングして、1 秒当たりのピーク・トランザクション数 (または、そのうちのかなり大きなパーセンテージ) に対処できるように、このパラメーターを調整することもできます。ピーク・アクティビティーに対処できれば、トランザクション集中期間のログ・レコード書き込みのオーバーヘッドが最小化されます。

mincommit を大きくする場合は、*logbufsz* パラメーターも値を大きくして、トランザクション集中期間にログ・バッファーがいっぱいになって、強制的に書き込みが行われることになるのを避ける必要もあります。この場合は、*logbufsz* が次の値に等しいことが必要です。

```
mincommit * (平均して 1 つのトランザクションで使用されるログ・スペース)
```

データベース・システム・モニターを使用すると、次の方法でこのパラメーターを調整できます。

- 1 秒当たりのピーク・トランザクション数を計算します。

  普通の 1 日を通してモニター・サンプルを取り、トランザクション集中期間を判別することができます。次のモニター・エレメントを加えることによって、合計トランザクション数を計算できます。

  – *commit_sql_stmts (試みられたコミット・ステートメント数)*
  – *rollback_sql_stmts (試みられたロールバック・ステートメント数)*

  この情報と使用可能なタイム・スタンプを使用して、1 秒当たりのトランザクション数を計算できます。
- 1 トランザクション当たりで使用されるログ・スペースを計算します。

ある期間にわたって、トランザクション数をサンプリングする技法を使用して、次のモニター・エレメントによって、使用されるログ・スペースの平均を計算できます。

– *log_space_used (使用される作業単位ログ・スペース)*

mirrorlogpath - ミラー・ログ・パス

このパラメーターによって、ストリングを 242 バイトまでミラー・ログ・パスに指定することができます。ストリングがパス名を示す必要があり、これは相対パス名ではなく、絶対パス名でなければなりません。

構成タイプ
データベース

パラメーター・タイプ
構成可能

デフォルト [範囲]
NULL [有効なパスまたは装置]

注: 単一または複数パーティションの DB2 ESE 環境では、ノード番号は自動的にパスに追加されます。このことは、複数論理ノード構成のパスの固有性を維持するために行われます。

mirrorlogpath が構成されている場合は、DB2 は、ログ・パスとミラー・ログ・パスの両方にアクティブ・ログ・ファイルを作成します。ログ・データはすべて両方のパスに書き込まれます。ミラー・ログ・パスには、アクティブ・ログ・ファイルのセットの複製があるので、ディスク・エラーや人間によるエラーで一方のパスにあるアクティブ・ログ・ファイルが破棄された場合でも、データベースはそのまま機能し続けることができます。

ミラー・ログ・パスが変更された場合は、ログ・ファイルが古いログ・パスにある可能性があります。こうしたログ・ファイルはアーカイブされていない場合があるので、これらのログ・ファイルは、手動でアーカイブしておく必要があります。また、このデータベース上でレプリケーションを実行する場合、ログ・パスが変更される前からのログ・ファイルを必要とします。 Yes に設定された「ユーザー出口使用可能 (*userexit*)」データベース構成パラメーターでデータベースが構成されている場合、およびすべてのログ・ファイルが DB2 によって自動的に、または手操作によってアーカイブされている場合、DB2 はログ・ファイルを検索してレプリケーション処理を完了することができるようになります。それ以外の場合、ファイルは、古いミラー・ログ・パスから新しいミラー・ログ・パスにコピーできます。

logpath または *newlogpath* で、ログ・ファイルの保管されているロケーションとしてロー・デバイスを指定する場合、ミラー・ロギング (*mirrorlogpath* によって示される) を行うことはできません。 *logpath* または *newlogpath* で、ログ・ファイルの保管されているロケーションとしてファイル・パスを指定する場合、ミラー・ロギングは可能であり、 *mirrorlogpath* の指定もまたファイル・パスとする必要があります。

推奨: ログ・ファイルの場合とまったく同様に、ミラー・ログ・ファイルの場合も、入出力の多くない物理ディスクに置く必要があります。

このパスは、1 次ログ・パスとは別の装置に置くことをお勧めします。

データベース・システム・モニターを使用すると、データベース・ロギングに関連する入出力の数を追跡できます。

以下のデータ・エレメントは、データベース・ロギングに関連した入出力活動の量を戻します。 他のディスク入出力アクティビティーに関する情報を収集し、入出力アクティビティーの 2 つのタイプを比較するには、オペレーティング・システムのモニター・ツールを使用できます。

- *log_reads* (読み取られたログ・ページの数)。
- *log_writes* (書き込まれたログ・ページの数)。

multipage_alloc - マルチページ・ファイル割り振り使用可能

マルチページ・ファイル割り振りは、挿入パフォーマンスを上げる場合に使用します。これは、SMS 表スペースにしか適用されません。使用可能にすると、すべての SMS 表スペースに影響します。つまり、個々の SMS 表スペースごとに選択することはできません。

構成タイプ
データベース

パラメーター・タイプ
通知

このパラメーターは、デフォルトでは「Yes」です。つまり、マルチページ・ファイル割り振りが使用可能です。

データベース作成後にこのパラメーターを「No」に設定することはできません。マルチページ・ファイル割り振りを一度使用可能にすると、使用不可にはできません。 db2empfa ツールを使用すると、マルチページ・ファイル割り振りが現在使用不可になっているデータベースに対して、マルチページ・ファイル割り振りが使用できます。

newlogpath - データベース・ログ・パスの変更

このパラメーターによって、242 バイト以下のストリングを 指定して、ログ・ファイルが保管されるロケーションを変更することができます。

構成タイプ
データベース

パラメーター・タイプ
構成可能

デフォルト [範囲]
NULL [有効なパスまたは装置]

ストリングによってパス名またはロー・デバイスを示すことができます。DB2 バージョン 9 からは、データベース・ロギングでのロー・デバイスの使用は推奨されていないことに注意してください。ロー・ログの使用に代わる方法として、直接入出力 (DIO) または並行入出力 (CIO) のいずれかを使用できます。

ストリングがパス名を示す場合は、相対パス名ではなく、完全修飾パス名でなければなりません。

単一または複数パーティションの DB2 ESE 環境では、ノード番号は自動的にパスに追加されます。このことは、複数論理ノード構成のパスの固有性を維持するために行われます。

レプリケーションを使用する場合で、ログ・パスがロー・デバイスであるときは、*overflowlogpath* 構成パラメーターを構成する必要があります。

デバイスを指定するには、オペレーティング・システムがデバイスとして認識するストリングを指定します。例:

- Windows 上では、¥¥.¥d: または ¥¥.¥PhysicalDisk5

  注: ログをデバイスに書き込めるようにするには、Windows バージョン 4.0 サービス・パック 3 またはそれ以降のリリースがインストールされていなければなりません。
- Linux および UNIX プラットフォームでは、/dev/rdblog8

注: AIX、Windows 2000、Windows、 Solaris、HP-UX、および Linux プラットフォームのデバイスのみを指定できます。

新しい設定は、以下の両方が発生するまで *logpath* の値にはなりません:

- データベースが、*database_consistent* パラメーターによって示される整合状態にある。
- すべてのアプリケーションがデータベースから切断された。

データベースへの最初の新しい接続が行われると、データベース・マネージャーはログを *logpath* によって指定されたロケーションに移動します。

古いログ・パスにログ・ファイルがある可能性があります。これらのログ・ファイルは、アーカイブされていないことがあります。これらのログ・ファイルは手操作でアーカイブする必要があります。また、このデータベース上でレプリケーションを実行する場合、ログ・パスが変更される前からのログ・ファイルを必要とします。 Yes に設定された「ユーザー出口使用可能 (*userexit*)」データベース構成パラメーターでデータベースが構成されている場合、およびすべてのログ・ファイルが DB2 によって自動的に、または手操作によってアーカイブされている場合、DB2 はログ・ファイルを検索してレプリケーション処理を完了することができるようになります。それ以外の場合は、ファイルを古いログ・パスから新しいログ・パスにコピーすることができます。

logpath または *newlogpath* で、ログ・ファイルの保管されているロケーションとしてロー・デバイスを指定する場合、ミラー・ロギング (*mirrorlogpath* によって示される) を行うことはできません。 *logpath* または *newlogpath* で、ログ・ファイルの保管されているロケーションとしてファイル・パスを指定する場合、ミラー・ロギングは可能であり、 *mirrorlogpath* の指定もまたファイル・パスとする必要があります。

推奨: 理想としては、ログ・ファイルは他のディスク I/O が**多くない**物理ディスクに置かれることが望まれます。例えば、ログをオペレーティング・システムまたは

大きなボリュームのデータベースと同じディスクに置くことは避けてください。これにより、入出力待ちなどのオーバーヘッドを最小にし、効率的なロギング活動が得られます。

データベース・システム・モニターを使用すると、データベース・ロギングに関連する入出力の数を追跡できます。

モニター・エレメント *log_reads* (読み取られたログ・ページの数) および *log_writes* (書き込まれたログ・ページの数) では、データベース・ロギングに関連する入出力アクティビティーの量を戻します。他のディスク入出力アクティビティーに関する情報を収集し、入出力アクティビティーの 2 つのタイプを比較するには、オペレーティング・システムのモニター・ツールを使用できます。

DB2 High Availability Disaster Recovery (HADR) データベースのペア内の 1 次データベースとスタンバイ・データベース両方のログ・パスとして共用されるネットワーク・ファイル・システムやローカル・ファイル・システムを使用しないでください。 1 次データベースとスタンバイ・データベースには、それぞれトランザクション・ログのコピーが存在し、1 次データベースによりスタンバイ・データベースにログが送信されます。 1 次データベースとスタンバイ・データベースの両方のログ・パスで、同じ物理ロケーションが指示された場合は、1 次データベースとスタンバイ・データベースではそれらの各ログのコピーに同じ物理ファイルが使用されます。データベース・マネージャーにより共用ログ・パスが検出された場合は、エラーが返されます。

num_db_backups - データベース・バックアップの数

このパラメーターは、データベースのために保持するデータベース・バックアップの数を指定します。

構成タイプ
データベース

パラメーター・タイプ
オンラインで構成可能

伝搬クラス
トランザクション境界

デフォルト [範囲]
12 [1 - 32 767]

指定したバックアップ数に達すると、リカバリー履歴ファイルで古いバックアップが有効期限切れとマークされます。有効期限が切れたデータベース・バックアップに関連する表スペース・バックアップおよびロード・コピー・バックアップのリカバリー履歴ファイル項目も有効期限切れとマークされます。バックアップに有効期限切れのマークが付けられると、物理バックアップは、その保管場所 (例えば、ディスク、テープ、TSM など) から除去できます。次のデータベース・バックアップが行われると、有効期限が切れた項目がリカバリー履歴ファイルから除去されます。

rec_his_retentn 構成パラメーターは、*num_db_backups* の値と互換性のある値に設定してください。例えば、*num_db_backup* が大きい値に設定されている場合、*rec_his_retentn* は、バックアップ数をサポートするのに十分な大きさの値が設定されていなければなりません。

num_freqvalues - 保存される高頻度値の数

このパラメーターを使用すると、 WITH DISTRIBUTION オプションが RUNSTATS コマンドで指定されたときに収集される、「最も頻度の高い値」の数を指定できます。

構成タイプ
データベース

パラメーター・タイプ
オンラインで構成可能

伝搬クラス
即時

デフォルト [範囲]
10 [0 - 32 767]

単位 カウンター

このパラメーターの値を大きくすると、統計の収集時に使用される統計ヒープの量 (*stat_heap_sz*) が増大します。

「最も頻度の高い値」統計は、オプティマイザーに列内でのデータ値の分散を理解させるのに役立ちます。値を大きくすれば、照会オプティマイザーで使用できる情報が増える結果になりますが、追加のカタログ・スペースが必要になります。 0 が指定されていると、たとえ分散統計の収集を要求しても、高頻度値統計が保存されることはありません。

また、表レベルまたは列レベルでの RUNSTATS コマンドの一部として、保持される高頻度値の数を指定することもできます。 NUM_FREQVALUES オプションを使用して。何も指定されなかった場合は、*num_freqvalues* 構成パラメーター値が使用されます。保存される高頻度値の数を RUNSTATS コマンドによって変更する方が、 *num_freqvalues* データベース構成パラメーターを使用して変更を行うよりも簡単です。

このパラメーターを更新すると、一様に分布していないデータについて、オプティマイザーが一部の述部 (=、<、>) の選択度をより正確に見積もることができます。選択度計算の正確性が増せば、より効率的なアクセス・プランが選択できる可能性があります。

このパラメーターの値を変更した場合は、その後で次のことを行う必要があります。

- 再度 RUNSTATS コマンドを実行して、高頻度値の変更された数で統計を収集します。
- 静的 SQL または XQuery ステートメントが含まれているパッケージがあれば、すべて再バインドします。

RUNSTATS を使用すると、表レベルと列レベルの両方で収集される高頻度値の数を制限できます。したがって、カタログが活用できなかった列に関する分散統計を削減し、しかもなお、重要な列に関する情報は使用して、カタログ内で占有されているスペースについて最適化ができます。

推奨: このパラメーターを更新するためには、一般的に選択述部がある (最も重要な表内の) 最も重要な列における非均等度を決定する必要があります。これには、列内におけるそれぞれの値の出現回数の順序付きランキングを提供する、SQL SELECT ステートメントを使用することができます。均等に分散した列、固有な列、LONG 列、または LOB 列は考慮しないでください。このパラメーターの妥当な実用値は、10 から 100 の範囲にあります。

高頻度値統計を収集する処理には、かなりの量の CPU およびメモリー (*stat_heap_sz*) リソースが必要です。

num_iocleaners - 非同期ページ・クリーナーの数

このパラメーターを使用すると、データベースの非同期ページ・クリーナーの数を指定できます。

構成タイプ
データベース

パラメーター・タイプ
構成可能

デフォルト [範囲]
Automatic [0 – 255]

単位 カウンター

これらのページ・クリーナーは、バッファー・プール内のスペースがデータベース・エージェントによって必要とされる前に、変更済みページをバッファー・プールからディスクに書き込みます。その結果として、データベース・エージェントは、変更済みページが書き出されて、バッファー・プール内のスペースを使用できるようになるのを待機する必要がなくなります。こうして、データベース・アプリケーション全体のパフォーマンスが向上します。

このパラメーターをゼロ (0) に設定した場合は、開始されるページ・クリーナーがないので、その結果として、バッファー・プールからディスクへのページ書き込みのすべてを、データベース・エージェントが実行することになります。データベースが多くの物理ストレージ・デバイスにまたがって保管されている場合は、物理ストレージ・デバイスの 1 つがアイドル状態になる可能性が高いので、このパラメーターがデータベースに重大なパフォーマンス上の影響を及ぼす可能性があります。ページ・クリーナーが構成されていない場合は、周期的にログがいっぱいになる状態がアプリケーションで検出される可能性があります。

このパラメーターを AUTOMATIC に設定すると、開始されるページ・クリーナーの数は、現行マシンに構成された CPU の数と、パーティション・データベース環境内のローカル論理データベース・パーティションの数に基づいて決まります。このパラメーターを AUTOMATIC に設定すると、必ず少なくとも 1 つのページ・クリーナーが開始されます。

このパラメーターを AUTOMATIC に設定したときに開始されるページ・クリーナーの数は、次の公式を使用して計算されます。

```
ページ・クリーナーの数 = max( ceil( CPU 数 / ローカル論理 DP 数) - 1, 1 )
```

この公式により、各論理データベース・パーティションに配布されるページ・クリーナーの数がほぼ均等になるようにし、かつページ・クリーナーの数が CPU の数を超えないようにしています。

データベースを使用するアプリケーションが、主として、データを更新するトランザクションで構成されている場合は、クリーナーの数を増やすと、パフォーマンスが向上します。ページ・クリーナーの数を増やすと、どの時点においても、ディスク上のデータベースの内容の最新性が増すため、停電などのソフト障害からのリカバリー時間も短縮されます。

推奨: このパラメーターの値を設定する際は、次の要因を考慮してください。

- アプリケーションのタイプ
  - 更新が行われない照会専用データベースの場合は、このパラメーターはゼロ (0) になるように設定します。例外は、照会ワークロードによって TEMP 表が多数作成される結果になる (これについては、EXPLAIN ユーティリティーを使用して判別できます) 場合です。
  - データベースに対してトランザクションが実行される場合は、このパラメーターは、1 とデータベース用として使用されている物理ストレージ・デバイスの台数の間になるように設定します。
- ワークロード

  更新トランザクションの率が高い環境では、ページ・クリーナーの数を増やして構成する必要があります。
- バッファー・プール・サイズ

  バッファー・プールの容量が大きい環境でも、ページ・クリーナーの数を増やして構成する必要があります。

データベース・システム・モニターを使用すると、バッファー・プールからの書き込みアクティビティーについてのイベント・モニターから得られる情報を使用して、この構成パラメーターを調整する場合に役立ちます。

- 次の 2 つの条件が真の場合は、パラメーターの値を小さくすることができます。
  - *pool_data_writes* がおよそ *pool_async_data_writes* に等しい。
  - *pool_index_writes* がおよそ *pool_async_index_writes* に等しい。
- 次の 2 つの条件のいずれかが真の場合は、パラメーターの値を大きくする必要があります。
  - *pool_data_writes* が *pool_async_data_writes* よりもはるかに大きい。
  - *pool_index_writes* が *pool_async_index_writes* よりもはるかに大きい。

num_ioservers - 入出力サーバーの数

このパラメーターは、データベースのための入出力サーバー数を指定します。あるデータベースに対して、この入出力数を超えるプリフェッチおよびユーティリティーを行うことはできません。

構成タイプ
データベース

パラメーター・タイプ
構成可能

デフォルト [範囲]
Automatic [1 – 255]

単位 カウンター

割り振られるタイミング
アプリケーションがデータベースに接続するとき

解放されるタイミング
アプリケーションがデータベースから切断されたとき

入出力サーバー (プリフェッチャーとも呼ばれる) は、データベース・エージェントの代わりに、バックアップおよびリストアなどのユーティリティーによるプリフェッチ入出力および非同期入出力を実行するために使用されます。入出力サーバーは、開始された入出力操作が進行中の間、待ち状態になります。非プリフェッチ入出力は、データベース・エージェントから直接スケジュールされ、その結果 *num_ioservers* によって制限されません。

このパラメーターを AUTOMATIC に設定すると、開始されるプリフェッチャーの数は、現行データベース・パーティション内の表スペースの並列処理の設定に基づいて決まります。 (並列処理の設定は、DB2_PARALLEL_IO 環境変数によって制御されます。) DMS 表スペースごとに、この並列処理設定の値に、表スペース・ストライプ・セットに含まれるコンテナーの最大数が掛けられます。 SMS 表スペースごとに、この並列処理設定の値に、表スペースに含まれるコンテナーの数が掛けられます。現行データベース・パーティション内のすべての表スペースの結果の中で最大のものが、開始されるプリフェッチャーの数として使用されます。このパラメーターを AUTOMATIC に設定すると、必ず少なくとも 3 つのプリフェッチャーが開始されます。

このパラメーターを AUTOMATIC に設定すると、開始されるプリフェッチャーの数は、データベース活動化の際に、次の公式に基づいて計算されます。

```
number of prefetchers = max( max over all table spaces
( parallelism setting * [SMS: # containers;
  DMS: max # containers in stripe set] ), 3 )
```

推奨: システム内のすべての入出力装置を完全に利用するために適正な値は、一般にデータベースが置かれている物理装置数に 1 または 2 を加えたものです。各入出力サーバーにはそれぞれほんの少しのオーバーヘッドが発生するのみで、未使用の入出力サーバーはアイドル状態のままになるため、追加の入出力サーバーを構成することをお勧めします。

num_log_span - ログ・スパンの数

このパラメーターは、1 つのトランザクションがいくつのログ・ファイルにまたがることができるかについての制限を設けるかどうか、およびその制限がいくつであるかを指定します。

構成タイプ
データベース

パラメーター・タイプ
オンラインで構成可能

伝搬クラス
即時

デフォルト [範囲]
0 [0 - 65 535]

単位 カウンター

この値が 0 ではない場合、このパラメーターは、 1 つのアクティブ・トランザクションで展開できるアクティブ・ログ・ファイルの数を示します。

値が 0 に設定される場合、1 つのトランザクションで扱えるログ・ファイルの数に制限はありません。これは、バージョン 8 より前のトランザクションの動作です。

num_quantiles - 列の変位値の数

このパラメーターでは、WITH DISTRIBUTION オプションが RUNSTATS コマンドで指定されたときに収集される変位値の数をコントロールします。

構成タイプ
データベース

パラメーター・タイプ
オンラインで構成可能

伝搬クラス
即時

デフォルト [範囲]
20 [0 - 32 767]

単位 カウンター

このパラメーターの値を大きくすると、統計の収集時に使用される統計ヒープの量 (*stat_heap_sz*) が増大します。

「変位値」統計は、オプティマイザーに列内でのデータ値の分散を理解させるのに役立ちます。値を大きくすれば、照会オプティマイザーで使用できる情報が増える結果になりますが、追加のカタログ・スペースが必要になります。 0 または 1 が指定されていると、たとえ分散統計の収集を要求しても、変位値統計が保存されることはありません。

また、NUM_QUANTILES オプションを使用して、表レベルまたは列レベルでの RUNSTATS コマンドの一部として、収集される変位値の数を指定することもできます。何も指定されなかった場合は、*num_quantiles* 構成パラメーター値が使用されます。収集される変位値の数を RUNSTATS コマンドによって変更する方が、*num_quantiles* データベース構成パラメーターを使用して変更を行うよりも簡単です。

このパラメーターを更新すると、一様に分布していないデータについて、範囲述部の選択度をより正確に見積もることができます。オプティマイザーの決定の中でも特に、この情報は、索引スキャンと表スキャンのどちらが選択されるかについて、強い影響を及ぼします (頻繁に発生する範囲の値にアクセスするには、表スキャンを使用する方が効率的であり、発生頻度の低い範囲の値の場合は、索引スキャンを使用する方が効率的です)。

このパラメーターの値を変更した場合は、その後で次のことを行う必要があります。

- 再度 RUNSTATS コマンドを実行して、高頻度値の変更された数で統計を収集します。
- 静的 SQL または XQuery ステートメントが含まれているパッケージがあれば、すべて再バインドします。

RUNSTATS を使用すると、表レベルと列レベルの両方で収集される変位値の数を制限できます。したがって、カタログが活用できなかった列に関する分散統計を削減し、しかもなお、重要な列に関する情報は使用して、カタログ内で占有されているスペースについて最適化ができます。

推奨: このパラメーターのこのデフォルト値では、片側述部 (>、>=、<、または <=) の場合は、最大で約 2.5% の見積エラー、BETWEEN 述部の場合は、最大で 5% のエラーです。変位値の数を概算する簡単な方法は、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。
- 変位値の数は、述部のほとんどが BETWEEN 述部の場合は、およそ 100/P で、述部のほとんどがそれ以外のタイプの範囲述部 (<、<=、>、または >=) の場合は、50/P であることが必要です。

たとえば、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。このパラメーターの妥当な実用値は、10 から 50 の範囲にあります。

numarchretry - エラー時の再試行数

このパラメーターでは、DB2 が、ログ・ファイルをフェイルオーバー・ディレクトリーにアーカイブする前に、ログ・ファイルを 1 次または 2 次アーカイブ・ディレクトリーにアーカイブしようと試行する回数を指定します。

構成タイプ
データベース

適用

- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- クライアント
- ローカル・クライアントを持つデータベース・サーバー
- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ
オンラインで構成可能

デフォルト [範囲]
5 [0 - 65 535]

このパラメーターは、*failarchpath* データベース構成パラメーターが設定されている場合にのみ使用できます。 *numarchretry* が設定されていない場合、 DB2 は 1 次または 2 次ログ・パスへのアーカイブの試行を継続します。

numsegs - SMS コンテナーのデフォルト数

このパラメーターはバージョン 9.5 では推奨されませんが、バージョン 9.5 より前のデータ・サーバーおよびクライアントでは引き続き使用されています。DB2 バージョン 9.5 のデータベース・マネージャーは、この構成パラメーターに対して指定された値をすべて無視します。

注: 以下の情報は、バージョン 9.5 より前のデータ・サーバーおよびクライアントだけに当てはまります。

構成タイプ
データベース

パラメーター・タイプ
通知

単位 カウンター

このパラメーターは、デフォルトの表スペース内で作成されるコンテナーの数を示します。このパラメーターは、CREATE DATABASE コマンドで明示的に指定されているか否かにかかわらず、データベースの作成時に使用された情報も示します。

このパラメーターは SMS 表スペースにのみ適用されます。CREATE TABLESPACE ステートメントは、このパラメーターを使用**しません**。

overflowlogpath - オーバーフロー・ログ・パス

このパラメーターは、ロールフォワード操作に必要なログ・ファイルを、DB2 が検索するロケーションを指定するとともに、アーカイブから取り出したアクティブ・ログ・ファイルを保管する場所を指定します。また、db2ReadLog API を使用するために必要なログ・ファイルを検索して保管する場所も指定します。

構成タイプ
データベース

パラメーター・タイプ
オンラインで構成可能

伝搬クラス
即時

デフォルト [範囲]
NULL [任意の有効なパス]

このパラメーターは、ロギング要件に応じていくつかの関数で使用できます。

- このパラメーターを使用すると、ロールフォワード操作に必要なログ・ファイルを、DB2 が検索するロケーションを指定できます。これは、ROLLFORWARD コマンドの OVERFLOW LOG PATH コマンドに似ています。OVERFLOW LOG

PATH の場合は、ROLLFORWARD コマンドごとにいつも指定するのに対して、この構成パラメーターは一度設定するだけです。ただし、両方使用すると、その特定のロールフォワード操作に関しては、OVERFLOW LOG PATH オプションが *overflowlogpath* 構成パラメーターを上書きします。

- *logsecond* が -1 に設定されている場合は、*overflowlogpath* を使用すると、アーカイブから取り出されたアクティブ・ログ・ファイルを、DB2 が保管するディレクトリーを指定できます。(アクティブ・ログ・ファイルがアクティブ・ログ・パスになくなっている場合は、ロールバック操作のために、アクティブ・ログ・ファイルを取り出す必要があります)。overflowlogpath が指定されていない場合は、DB2 はログ・ファイルを取り出してアクティブ・ログ・パスに入れます。*overflowlogpath* を使用すると、取り出されたログ・ファイルを、DB2 が保管する追加のリソースを用意できます。利点としては、入出力コストが異なるディスクに拡散することや、アクティブ・ログ・パスに保管できるログ・ファイルが増えることなどがあります。
- db2ReadLog API (DB2 V8 より前では、db2ReadLog は sqlurlog と呼ばれていた) を、例えば、レプリケーションを使用する場合で、*overflowlogpath* を使用すると、この API 用として必要なログ・ファイルを、DB2 が検索するロケーションを指定できます。ログ・ファイルが (アクティブ・ログ・パスにもオーバーフロー・ログ・パスにも) 見つからない場合、データベースが *userexit* を使用可能にして構成されていると、DB2 はログ・ファイルを検索します。*overflowlogpath* を使用すると、DB2 が取り出されたログ・ファイルを保管するためのディレクトリーも指定できます。利点としては、アクティブ・ログ・パスの入出力コストの削減、およびアクティブ・ログ・パスに保管できるログ・ファイル数の増加などがあります。
- アクティブ・ログ・パス用としてロー・デバイスを構成してある場合は、*logsecond* を -1 に設定するとき、または db2ReadLog API を使用するときに *overflowlogpath* を構成する必要があります。

overflowlogpath を設定するには、最大 242 バイトのストリングを指定します。ストリングがパス名を示す必要があり、これは相対パス名ではなく、絶対パス名でなければなりません。パス名はディレクトリーでなければならず、ロー・デバイスではありません。

注: 単一または複数パーティションの DB2 ESE 環境では、ノード番号は自動的にパスに追加されます。このことは、複数論理ノード構成のパスの固有性を維持するために行われます。

pagesize - データベース・デフォルト・ページ・サイズ

このパラメーターには、データベースの作成時にデフォルトのページ・サイズとして使用された値が含まれます。可能な値は、4 096、8 192、16 384、および 32 768 です。そのデータベース内でバッファー・プールまたは表スペースが作成された場合には、同じデフォルト・ページ・サイズが適用されます。

構成タイプ
データベース

パラメーター・タイプ
通知

pckcachesz - パッケージ・キャッシュ・サイズ

このパラメーターは、データベース共用メモリーから割り振られ、データベース上の静的および動的 SQL および XQuery ステートメントのセクションをキャッシュするために使用されます。

構成タイプ
データベース

パラメーター・タイプ
オンラインで構成可能

伝搬クラス
即時

デフォルト [範囲]

32 ビット・プラットフォーム
Automatic [-1, 32 - 128 000]

64 ビット・プラットフォーム
Automatic [-1, 32 - 524 288]

単位 ページ (4 KB)

割り振られるタイミング
データベースの初期化時

解放されるタイミング
データベースがシャットダウンされるとき

パーティション・データベース・システムでは、データベース・パーティションごとに 1 つのパッケージ・キャッシュがあります。

パッケージをキャッシュすると、データベース・マネージャーの場合はパッケージを再ロードするときにシステム・カタログにアクセスする必要がなくなるために内部オーバーヘッドが減少し、また動的 SQL または XQuery ステートメントの場合はコンパイルする必要がなくなることによって、内部オーバーヘッドが減少します。セクションは、以下のいずれかの状況になるまで、パッケージ・キャッシュ内に保持されます。

- データベースがシャットダウンされるとき
- パッケージまたは動的 SQL または XQuery ステートメントが無効にされる時
- キャッシュがスペースを使い果たした時

静的または動的 SQL または XQuery ステートメントのセクションのこのキャッシュは、データベースに接続されたアプリケーションによって同じステートメントが複数回使用される場合に、特にパフォーマンスを改善することができます。これは、トランザクション処理環境では特に重要です。

このパラメーターは、AUTOMATIC に設定されると、セルフチューニングが使用可能になります。 *self_tuning_mem* を ON に設定すると、メモリー・チューナーは、*pckcachesz* によって制御されるメモリー領域のサイズを、ワークロード要件の変化に応じて動的に変更するようになります。メモリー・チューナーは異なるメモリー・コンシューマーの間でメモリー・リソースをやりとりするので、セルフチュー

ニングをアクティブにするためには、少なくとも 2 つのメモリー・コンシューマーのセルフチューニングが使用可能になっていなければなりません。

この構成パラメーターの自動チューニングは、データベースでセルフチューニング・メモリーが有効になっている (*self_tuning_mem* 構成パラメーターが「ON」に設定されている) 場合にのみ実行されます。

このパラメーターを -1 に設定すると、ページ割り振りの計算に使用される値は、*maxappls* 構成パラメーターに指定された値の 8 倍になります。 *maxappls* の 8 倍が 32 よりも小さい場合は、例外となります。この場合、デフォルト値 -1 を指定すると *pckcachesz* が 32 に設定されます。

推奨: このパラメーターを調整するときは、パッケージ・キャッシュ用に予約されている余分なメモリーが、バッファー・プールまたはカタログ・キャッシュなどの別の目的のために割り振られた場合にさらに有効に使用できるかどうかを考慮する必要があります。 上記の理由により、このパラメーターの調整時には、ベンチマーク技法を使用してください。

このパラメーターの調整は、いくつかのセクションが最初に使用され、その後は 2、3 のセクションだけが繰り返し実行される場合に特に重要です。 キャッシュが大きすぎると、最初のセクションのコピーを保留している分だけメモリーが無駄になっています。

これらのモニター・エレメントは、この構成パラメーターを調整すべきかどうかの判別に役立ちます。

- *pkg_cache_lookups* (パッケージ・キャッシュ参照)
- *pkg_cache_inserts* (パッケージ・キャッシュ挿入)
- *pkg_cache_size_top* (パッケージ・キャッシュ最高水準点)
- *pkg_cache_num_overflows* (パッケージ・キャッシュ・オーバーフロー)

注: パッケージ・キャッシュは作業キャッシュであるため、このパラメーターをゼロに設定することはできません。 現在実行されている SQL または XQuery ステートメントのすべてのセクションを保留するには、このキャッシュに十分なメモリーが割り振られていなければなりません。現行の必要スペース以上のスペースが割り振られている場合、セクションがキャッシュされます。これらのセクションは、次に必要とされるときにはロードやコンパイルをせずに実行できます。

pckcachesz パラメーターによって指定された制限は緩やかな制限です。メモリーがデータベース共有セットでまだ使用可能な場合、必要であればこの制限を超えることが可能です。パッケージ・キャッシュが一番大きくなったサイズを判別するには、*pkg_cache_size_top* モニター・エレメントを使用し、*pckcachesz* パラメーターによって指定された制限を超えた回数を判別するには、*pkg_cache_num_overflows* モニター・エレメントを使用することができます。

priv_mem_thresh - 専用メモリーしきい値

このパラメーターはバージョン 9.5 では推奨されませんが、バージョン 9.5 より前のデータ・サーバーおよびクライアントでは引き続き使用されています。DB2 バージョン 9.5 のデータベース・マネージャーは、この構成パラメーターに対して指定された値をすべて無視します。

注: 以下の情報は、バージョン 9.5 より前のデータ・サーバーおよびクライアントだけに当てはまります。

構成タイプ
データベース・マネージャー

適用
- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- ローカル・クライアントを持つデータベース・サーバー
- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ
構成可能

デフォルト [範囲]
20 000 [-1; 32 - 112 000]

単位 ページ (4 KB)

このパラメーターは、開始される新しいエージェントが使用できる状態にある、未使用のエージェント専用メモリー量の判別に使用されます。これは、Linux および UNIX プラットフォームには適用されません。

-1 の値を指定すると、このパラメーターは、*min_priv_mem* パラメーターの値を使用します。

推奨: このパラメーターを設定する場合、クライアント接続/切断パターンと、同じマシン上の他の複数のプロセスのメモリー要件を考慮してください。

多くのクライアントがデータベースに同時に接続する期間が短い場合、しきい値を高くすると未使用メモリーがコミット解除されなくなり、他のプロセスで使用できなくなります。この場合、十分なメモリー管理が行われず、メモリーを必要とする他のプロセスに影響を与えます。

並行クライアントの数があまり変わらず、その値の範囲内で頻繁に変動する場合、このしきい値を高くすることによってクライアント・プロセスの為にメモリーを確保し、メモリーの割り振りと割り振り解除のオーバーヘッドを低減させることができます。

rec_his_retentn - リカバリー履歴保持期間

このパラメーターは、バックアップの履歴情報を保持する日数を指定します。

構成タイプ
データベース

パラメーター・タイプ
構成可能

デフォルト [範囲]
366 [-1; 0 - 30 000]

単位 日数

リカバリー履歴ファイルがバックアップ、リストア、ロードの追跡に必要でない場合、このパラメーターの値を小さく設定することができます。

このパラメーターの値が -1 の場合、フル・データベース・バックアップ (およびそのデータベース・バックアップに関連した表スペース・バックアップ) を示す項目数が、*num_db_backups* パラメーターによって指定された値に対応します。リカバリー履歴ファイル内の他の項目は、使用可能なコマンドまたは API を明示的に使用することによってのみ、枝取りすることができます。

保持期間がどんなに短くても、PRUNE ユーティリティーに FORCE オプションを指定していなければ、ほとんどのフル・データベース・バックアップとそのリストア・セットは必ず保持されます。

restore_pending - リストア・ペンディング

このパラメーターは、RESTORE PENDING 状況がデータベース内に存在するかどうかを示します。

構成タイプ
データベース

パラメーター・タイプ
通知

restrict_access - データベース制限付きアクセス構成パラメーター

このパラメーターは、データベースが作成されたときに、デフォルトのアクションの制限セットが使用されたかどうかを示します。言い換えれば、CREATE DATABASE コマンドの中で RESTRICTIVE 節を使用してデータベースが作成されたかどうかを示します。

構成タイプ
データベース

パラメーター・タイプ
通知

YES: このデータベースが作成されたときに、CREATE DATABASE コマンドの中で RESTRICTIVE 節が使用されています。

NO: このデータベースが作成されたときに、CREATE DATABASE コマンドの中で RESTRICTIVE 節は使用されていません。

rollfwd_pending - ロールフォワード・ペンディング標識

このパラメーターは、ロールフォワード・リカバリーが必要かどうか、およびどこで必要なのかを通知します。

構成タイプ
データベース

パラメーター・タイプ
通知

このパラメーターでは、次の状態のいずれか 1 つを指定できます。

- **DATABASE**。ロールフォワード・リカバリー手順がこのデータベースに必要であることを示します。
- **TABLESPACE**。1 つ以上の表スペースがロールフォワードする必要があることを示します。
- **NO**。データベースが使用可能であり、ロールフォワード・リカバリーは必要ないことを意味します。

リカバリー (ROLLFORWARD DATABASE の使用による) が完了しないうちは、データベースにも表スペースにもアクセスできません。

self_tuning_mem- セルフチューニング・メモリー

このパラメーターは、メモリー・チューナーが、セルフチューニングに使用可能なメモリー・コンシューマーに、要求に応じて使用可能なメモリー・リソースを動的に配分するかどうかを決定します。

構成タイプ
データベース

パラメーター・タイプ
オンラインで構成可能

伝搬クラス
即時

デフォルト [範囲]

単一データベース・パーティション環境
ON [ON; OFF]

複数データベース・パーティション環境
OFF [ON; OFF]

以前のバージョンからマイグレーションしたデータベースでは、*self_tuning_mem* は OFF に設定されます。

メモリーは複数のメモリー・コンシューマーの間でやりとりされるので、メモリー・チューナーをアクティブにするためには、少なくとも 2 つのメモリー・コンシューマーのセルフチューニングが使用可能になっていなければなりません。*self_tuning_mem* を ON に設定しても、セルフチューニングが使用可能なメモリー・コンシューマーが 1 つ以下であれば、メモリー・チューナーは非アクティブ状態です。 (ソート・ヒープ・メモリー領域は例外です。この領域は、他のメモリー・コンシューマーのセルフチューニングが使用可能かどうかにかかわらず、チューニング可能です。) *database_memory* が数値に設定されている場合は、セルフチューニングが使用可能であるとみなされます。

このパラメーターは、単一データベース・パーティション環境ではデフォルトで ON です。複数データベース・パーティション環境では、デフォルトで OFF です。

セルフチューニングを使用可能にできるメモリー・コンシューマーとして、以下のものがあります。

- バッファー・プール (ALTER BUFFERPOOL および CREATE BUFFERPOOL ステートメントのサイズ・パラメーターで制御)
- パッケージ・キャッシュ (*pckcachesz* 構成パラメーターで制御)
- ロック・リスト (*locklist* および *maxlocks* 構成パラメーターで制御)
- ソート・ヒープ (*sheapthres_shr* および *sortheap* 構成パラメーターで制御)
- データベース・共用メモリー (*database_memory* 構成パラメーターで制御)

このパラメーターの現在の設定を表示するには、SHOW DETAIL パラメーターを指定した GET DATABASE CONFIGURATION コマンドを使用します。このパラメーターで戻される設定として、以下のものが考えられます。

```
Self Tuning Memory          (SELF_TUNING_MEM) = OFF
Self Tuning Memory          (SELF_TUNING_MEM) = ON (Active)
Self Tuning Memory          (SELF_TUNING_MEM) = ON (Inactive)
Self Tuning Memory          (SELF_TUNING_MEM) = ON
```

値とその意味を、次に示します。

- `ON (Active)` - メモリー・チューナーがアクティブにシステムのメモリーを調整しています。
- `ON (Inactive)` - パラメーターは ON に設定されているものの、セルフチューニングが使用可能なメモリー・コンシューマーが 1 つ以下であるため、セルフチューニングが行われていません。
- (Active) または (Inactive) なしの `ON` - SHOW DETAIL オプションなし、またはデータベース接続のない照会からの場合。

パーティション環境では、*self_tuning_mem* 構成パラメーターは、チューナーが稼働しているデータベース・パーティションでのみ ON (Active) を表示します。他のすべてのノードでは、*self_tuning_mem* は ON (Inactive) を表示します。そのため、メモリー・チューナーがパーティション・データベースにおいてアクティブかどうかを判別するには、すべてのデータベース・パーティション上の *self_tuning_mem* パラメーターを確認する必要があります。

以前のバージョンの DB2 からマイグレーションした DB2 バージョン 9 でセルフチューニング・メモリー・フィーチャーを使用する場合は、以下のヘルス・インディケーターを、しきい値または状態のチェックを使用不可に構成する必要があります。

- 共有ソート・メモリー使用率 - db.sort_shrmem_util
- オーバーフローしたソートのパーセント - db.spilled_sorts
- 長期共有ソート・メモリー使用率 - db.max_sort_shrmem_util
- ロック・リスト使用率 - db.locklist_util
- ロック・エスカレーション率 - db.lock_escal_rate
- パッケージ・キャッシュ・ヒット率 - db.pkgcache_hitratio

セルフチューニング・メモリー・フィーチャーの目的の 1 つは、メモリーをすぐには必要としていないメモリー・コンシューマーにメモリーが割り振られないようにすることです。したがって、より多くのメモリーが割り振られれる前に、メモリー・コンシューマーに割り振られたメモリーの使用率が 100% に近づく可能性があ

ります。これらのヘルス・インディケーターを使用不可にすることによって、メモリー・コンシューマーのメモリー使用率が高いために不必要なアラートが発生することを回避できます。

DB2 バージョン 9 で作成されるインスタンスでは、これらのヘルス・インディケーターはデフォルトで使用不可に設定されます。

seqdetect - 順次検出フラグ

このパラメーターは、データベース・マネージャーに、入出力アクティビティー中の順次ページ読み取りの検出を許可するかどうかを制御します。

構成タイプ
データベース

パラメーター・タイプ
オンラインで構成可能

伝搬クラス
即時

デフォルト [範囲]
Yes [Yes; No]

データベース・マネージャーは、入出力をモニターすることができ、順次ページ読み取りが行われている場合は、データベース・マネージャーは入出力プリフェッチを活動化できます。このタイプの順次プリフェッチは、*順次検出* という名前で呼ばれています。

このパラメーターが No に設定されている場合、プリフェッチは、例えば、表ソート、表スキャン、またはリスト・プリフェッチなど、有用であることがデータベース・マネージャーに分かっている場合にのみ行われます。

推奨: ほとんどの場合、このパラメーターにはデフォルト値を使用する必要があります。順次検出をオフにするのは、それ以外のチューニングによってでは重大な照会パフォーマンスが解決できなかった場合だけにしてください。

sheapthres_shr - 共用ソートのソート・ヒープのしきい値

このパラメーターは、任意の一時点にソート・メモリー・コンシューマーによって使用できるデータベース共用メモリーの合計量に関する緩やかな制限を表します。

構成タイプ
データベース

適用 OLAP 関数

パラメーター・タイプ
オンラインで構成可能

伝搬クラス
即時

デフォルト [範囲]

32 ビット・プラットフォーム
Automatic [250 - 524 288]

64 ビット・プラットフォーム
Automatic [250 - 2 147 483 647]

単位 ページ (4 KB)

ソートの他にも、ハッシュ結合、索引 ANDing、ブロック索引 ANDing、マージ結合、およびメモリー内の表などのソート・メモリー・コンシューマーがあります。共用ソート・メモリー・コンシューマーの共用メモリーの合計が限界値 *sheapthres_shr* に近づくと、メモリー・スロット機構が活動化され、将来の共用ソート・メモリー・コンシューマー要求に対して、その要求よりも少ない量のメモリーが付与される場合があります。しかし、タスクの完了に必要な最低限の量より多くのメモリーは常に付与されます。いったん限界値 *sheapthres_shr* を超過すると、ソート・メモリー・コンシューマーからの共用ソート・メモリーのすべての要求に対して、タスクの完了に必要な最低限のメモリーが付与されます。アクティブ共有ソート・メモリー・コンシューマー用の共用メモリーの合計量がこの限度に達すると、後続のソートは失敗する可能性があります (SQL0955C)。

データベース・マネージャー構成パラメーター *sheapthres* の値が 0 の場合、データベースのソート・メモリー・コンシューマーはすべて、専用ソート・メモリーではなく、*sheapthres_shr* で制限されたデータベース共用メモリーを使用します。

sheapthres_shr が AUTOMATIC に設定されると、セルフチューニングが使用可能になります。これによってメモリー・チューナーは、このパラメーターによって制御されるメモリー領域のサイズを、ワークロード要件の変化に応じて動的に変更できるようになります。メモリー・チューナーは異なるメモリー・コンシューマーの間でメモリー・リソースをやりとりするので、セルフチューニングをアクティブにするためには、少なくとも 2 つのメモリー・コンシューマーのセルフチューニングが使用可能になっていなければなりません。メモリー・コンシューマーには、SHEAPTHRES_SHR、PCKCACHESZ、BUFFER POOL (各バッファー・プールを 1 つと数える)、 LOCKLIST、および DATABASE_MEMORY が含まれます。

sheapthres_shr 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。

sortheap の値をチューニングすると *sheapthres_shr* パラメーターもともにチューニングされ、*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、*sheapthres_shr* パラメーターのセルフチューニングも自動的に使用不可にされます。*sheapthres_shr* パラメーターのセルフチューニングを使用可能にすると、*sortheap* パラメーターのセルフチューニングが自動的に使用可能になります。

この構成パラメーターの自動チューニングは、データベースでセルフチューニング・メモリーが有効になっている (*self_tuning_mem* 構成パラメーターが「ON」に設定されている) 場合にのみ実行されます。

このパラメーターの値がオンラインで更新されると、更新後に行われた共用ソート・メモリーの新規要求のみが新規の値を使用します。 *sheapthres_shr* の値を減らす前に *sortheap* の値を減らし、*sortheap* の値を増やす前に *sheapthres_shr* の値を増やすことをお勧めします。

データベース・マネージャー構成パラメーター *sheapthres* が 0 より大きい場合、*sheapthres_shr* は次の 2 つの場合にのみ意味を成します。

- *intra_parallel* データベース・マネージャー構成パラメーターが *yes* に設定されている場合。*intra_parallel* が *no* に設定されているときは、共有ソートはないからです。
- コンセントレーターがオンの場合 (つまり、*max_connections* が *max_coordagents* より大のとき)。 WITH HOLD オプションを指定して宣言されたカーソルを使用するソートは、共用メモリーから割り振られるからです。

softmax - リカバリー範囲およびソフト・チェックポイント・インターバル

このパラメーターは、ソフト・チェックポイントの頻度およびリカバリー範囲を決定します。これは、クラッシュ・リカバリー・プロセスに役立ちます。

構成タイプ
データベース

パラメーター・タイプ
構成可能

デフォルト [範囲]
100 [1 – 100 * *logprimary*]

単位 1 つの 1 次ログ・ファイルのサイズに占めるパーセンテージ

このパラメーターは、次の場合に使用します。

- クラッシュ (例えば、電源障害など) の後でリカバリーする必要のあるログの数に影響を与える場合。例えば、デフォルト値が使用されている場合、データベース・マネージャーは、リカバリーする必要のあるログの数を 1 に維持しようとします。このパラメーターの値として 300 を指定した場合、データベース・マネージャーは、リカバリーする必要のあるログの数を 3 に維持しようとします。

  クラッシュ・リカバリーに必要なログの数に影響を与えるために、データベース・マネージャーは、このパラメーターを使用してページ・クリーナーを起動して、指定されたリカバリー・ウィンドウよりも古いページについては、確実にディスクに書き込まれているようにします。
- ソフト・チェックポイントの頻度を決定する場合。

電源障害などのようなイベントの結果、データベース障害が発生した時点では、データベースに次のような変更がもたらされている可能性があります。

- コミットされてはいないが、バッファー・プールのデータを更新してしまった変更
- コミットされているが、バッファー・プールからディスクに書き込まれてはいない変更
- コミットされ、しかもバッファー・プールからディスクに書き込まれてしまった変更

データベースが再始動されると、ログ・ファイルを使用して、データベースのクラッシュ・リカバリーが実行されます。これにより、データベースが整合状態のままに置かれる (つまり、コミット済みトランザクションはすべてがデータベースに適用され、非コミット・トランザクションはすべてがデータベースに適用されていない) ことが保証されます。

ログ・ファイルからデータベースに適用される必要のあるレコードを判別するために、データベース・マネージャーはログ・コントロール・ファイルに記録された情報を使用します。 (データベース・マネージャーは、実際にはログ・コントロール・ファイルのコピーを 2 つ (SQLOGCTL.LFH.1 および SQLOGCTL.LFH.2) 維持しており、片方のコピーが損傷を受けた場合でも、もう一方のコピーを使用できます。) これらのログ・コントロール・ファイルは、定期的にディスクに書き込まれ、このイベントの頻度に応じて、データベース・マネージャーがコミット済みトランザクションのログ・レコードを適用したり、すでにバッファー・プールからディスクに書き込まれてしまった変更を記述するログ・レコードを適用したりします。これらのログ・レコードがデータベースに影響を及ぼすことはありませんが、ログ・レコードを適用することによって、データベース再始動処理に多少のオーバーヘッドが発生します。

ログ・コントロール・ファイルは、ログ・ファイルがいっぱいになったとき、およびソフト・チェックポイント時には、必ずディスクに書き込まれます。この構成パラメーターを使用して、追加のソフト・チェックポイントを起動することができます。

ソフト・チェックポイントのタイミングは、*logfilsiz* に占めるパーセンテージとして与えられる、「現行の状態」と「記録された状態」の間の差に基づいて決まります。「記録された状態」が、ディスク上のログ・コントロール・ファイルに示されている最も古い有効ログ・レコードによって決まるのに対して、「現行の状態」は、メモリー内のログ・コントロール情報によって決まります (最も古い有効ログ・レコードが、リカバリー処理で最初に読み込まれるログ・レコードになります)。ソフト・チェックポイントが取られるのは、次の公式による計算値がこのパラメーターの値より大か等しい場合です。

```
( (記録された状態と現行の状態の間のスペース) / logfilsiz ) * 100
```

推奨: このパラメーターの値は、許容リカバリー・ウィンドウが 1 つのログ・ファイルより大きいか小さいかによって、大きくしたり小さくしたりすることができます。このパラメーターの値を小さくすると、データベース・マネージャーがページ・クリーナーを起動する回数が増え、ソフト・チェックポイントを取る頻度も高くなります。これらのアクションによって、処理する必要のあるログ・レコードの数と、クラッシュ・リカバリー時に処理される重複ログ・レコードの数が、両方とも減らせます。

ただし、ページ・クリーナーの起動回数が増え、ソフト・チェックポイントの頻度が増すと、データベース・ロギングに関連したオーバーヘッドが増加し、データベース・マネージャーのパフォーマンスに影響する可能性があることに注意してください。また、ソフト・チェックポイントの頻度が高くなっても、次のような場合には、データベースの再始動に必要な時間が短縮できない可能性もあります。

- 非常に長いトランザクションで、コミット・ポイントが少ない場合。
- 非常に大きいバッファー・プールがあり、コミット済みトランザクションが入っているページがディスクに元どおり書き込まれる頻度があまり高くない場合 (非同期ページ・クリーナーを使用すると、この状態を回避できます)。

上記のケースのどちらでも、メモリーに保持されているコントロール情報は頻繁には変更されず、ログ・コントロール情報は、変更されていない限り、ディスクに書き込んでも利点はありません。

sortheap - ソート・ヒープ・サイズ

このパラメーターでは、専用ソートで使用される専用メモリー・ページの最大数、または共有ソートで使用される共用メモリー・ページの最大数を定義します。

構成タイプ
データベース

適用 OLAP 関数

パラメーター・タイプ
オンラインで構成可能

伝搬クラス
即時

デフォルト [範囲]

32 ビット・プラットフォーム
Automatic [16 - 524 288]

64 ビット・プラットフォーム
Automatic [16 - 4 194 303]

単位 ページ (4 KB)

割り振られるタイミング
ソートを実行する必要に応じて

解放されるタイミング
ソートの完了時

ソートが専用ソートの場合は、このパラメーターによってエージェント専用メモリーに影響が生じます。ソートが共有ソートの場合は、このパラメーターによってデータベース共用メモリーに影響が生じます。それぞれのソートには、必要に応じてデータベース・マネージャーによって割り振られる別々のソート・ヒープがあります。このソート・ヒープは、データがソートされる領域です。オプティマイザーによる指示があった場合は、オプティマイザーが提供する情報を使用して、このパラメーターによって指定されたソート・ヒープよりも小さいソート・ヒープが割り振られます。

このパラメーターは、AUTOMATIC に設定されると、セルフチューニングが使用可能になります。これによってメモリー・チューナーは、このパラメーターによって制御されるメモリー領域のサイズを、ワークロード要件の変化に応じて動的に変更できるようになります。

sortheap の値のチューニングは *sheapthres_shr* パラメーターと一緒に行われるので、*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には、*sheapthres_shr* パラメーターのセルフチューニングも使用不可にしないと実行できません。 *sheapthres_shr* パラメーターのセルフチューニングを使用可能にすると、*sortheap* パラメーターのセルフチューニングが自動的に使用可能になります。ただし *sortheap* パラメーターのセルフチューニングの使用可能化は、*sheapthres_shr* パラメーターを AUTOMATIC にしなくても行うことができます。

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。

この構成パラメーターの自動チューニングは、データベースでセルフチューニング・メモリーが有効になっている (*self_tuning_mem* 構成パラメーターが「ON」に設定されている) 場合にのみ実行されます。

推奨: ソート・ヒープを使用して作業する場合は、次の事項を考慮する必要があります。

- 適切な索引によってソート・ヒープの使用を最小化できます。
- ハッシュ結合バッファー、ブロック索引 ANDing、マージ結合、メモリー内の表、および動的ビットマップ (索引 ANDing および Star Join で使用される) では、ソート・ヒープ・メモリーを使用します。したがって、これらの技法が使用されるときは、このパラメーターのサイズを大きくします。
- ラージ・ソートが頻繁に必要なときは、このパラメーターのサイズを大きくします。
- このパラメーターの値を大きくするときは、データベース・マネージャー構成ファイルにある *sheapthres* および *sheapthres_shr* パラメーターも調整する必要があるかどうか調べる必要があります。
- ソート・ヒープ・サイズは、オプティマイザーがアクセス・パスを決定する際に使用します。このパラメーターを変更した場合は、アプリケーションの再バインド (REBIND コマンドを使用) を考慮してください。

ソート・ヒープ値が更新されると、データベース・マネージャーは現行または新規のソートがあれば、それに対して直ちにこの新しい値を使い始めます。

stat_heap_sz - 統計ヒープ・サイズ

このパラメーターは、RUNSTATS コマンドによる統計の収集の際に使用される、ヒープの最大 サイズを示します。

バージョン 9.5 では、このデータベース構成パラメーターのデフォルト値は AUTOMATIC であるため、*appl_memory* 限度または *instance_memory* 限度のいずれかに達するまで、必要に応じて増加します。

構成タイプ
データベース

パラメーター・タイプ
オンラインで構成可能

デフォルト [範囲]
Automatic [1 096 - 524 288]

単位 ページ (4 KB)

割り振られるタイミング
RUNSTATS ユーティリティーが開始したとき

解放されるタイミング
RUNSTATS ユーティリティーが完了したとき

推奨: デフォルトの設定の AUTOMATIC が推奨値です。

stmtheap - ステートメント・ヒープ・サイズ

このパラメーターで、ステートメント・ヒープのサイズを指定します。これは SQL または XQuery ステートメントのコンパイル時に SQL または XQuery コンパイラー用のワークスペースとして使用されます。

バージョン 9.5 では、このデータベース構成パラメーターのデフォルト値は AUTOMATIC であるため、*appl_memory* 限度または *instance_memory* 限度のいずれかに達するまで、必要に応じて増加します。

構成タイプ
データベース

パラメーター・タイプ
オンラインで構成可能

伝搬クラス
ステートメント境界

デフォルト [範囲]

32 ビットおよび 64 ビット・プラットフォームの場合
Automatic [128 - 524 288]

単位 ページ (4 KB)

割り振られるタイミング
それぞれのステートメントごとに、プリコンパイル時またはバインド時

解放されるタイミング
それぞれのステートメントのプリコンパイルまたはバインドが完了した時点

この作業域は、永続的に割り振られた状態のままになっているのではなく、SQL または XQuery ステートメントが処理されるごとに、その割り振りと解放が行われます。この作業域は、動的 SQL または XQuery ステートメントの場合は、プログラムの実行時に使用されるのに対して、静的 SQL または XQuery ステートメントの場合は、バインド処理時に使用され、プログラム実行時には使用されないことに注意してください。

推奨: ほとんどの場合、このパラメーターのデフォルト値 AUTOMATIC を使用できます。AUTOMATIC に設定すると、コンパイルの動的プログラミング結合列挙フェーズにおいて割り振られるメモリーの総量に内部制限が発生します。この制限を超過すると、欲張り型の結合列挙でステートメントがコンパイルされ、残っている *appl_memory* か *instance_memory* あるいはその両方の量によってのみ制限されます。使用中のアプリケーションで SQL0437W 警告を受け取り、照会に対する実行時のパフォーマンスが許容範囲に達しない場合、必ず動的結合列挙が常に使用されるように、*stmtheap* に手動で十分な大きさの値を設定することを考慮することもできます。

注: 動的結合列挙は、最適化クラス 3 以上でのみ発生します (デフォルトは 5 です)。

territory - データベース・テリトリー

このパラメーターは、データベースを作成するために使用されたテリトリーを示します。 *territory* は、テリトリーを区別するデータの処理時にデータベース・マネージャーによって使用されます。

構成タイプ
データベース

パラメーター・タイプ
通知

trackmod - 変更ページの追跡使用可能化

このパラメーターは、データベース・マネージャーがデータベース変更を追跡するかどうかを指定します。追跡するなら、バックアップ・ユーティリティーは、データベース・ページのどのサブセットが増分バックアップによって検査され、バックアップ・イメージに組み込まれる可能性があるかを検出することができます。

構成タイプ
データベース

パラメーター・タイプ
構成可能

デフォルト [範囲]
No [Yes, No]

このパラメーターを「Yes」に設定した後は、増分バックアップを取ることができるベースラインを決めるために、全データベース・バックアップを取る必要があります。また、このパラメーターが使用可能になっている場合、または表スペースが作成されている場合は、その表スペースを含むバックアップを取る必要があります。このバックアップは、データベース・バックアップと表スペース・バックアップのいずれかになります。このバックアップの後では、増分バックアップに表スペースを含めることができます。

tsm_mgmtclass - Tivoli Storage Manager 管理クラス

Tivoli Storage Manager 管理クラスは、バックアップするオブジェクトのバックアップ・バージョンを TSM がどのように管理するかを判断します。

構成タイプ
データベース

パラメーター・タイプ
構成可能

デフォルト [範囲]
NULL [任意のストリング]

デフォルトでは、DB2 指定の管理クラスはありません。

TSM バックアップを実行すると、データベース構成パラメーターで指定した管理クラスを使用する前に、 TSM はまず TSM クライアント・オプション・ファイル内にある INCLUDE-EXCLUDE リストで指定した管理クラスに、バックアップ・オブジェクトをバインドしようとします。一致するものが見付からない場合には、 TSM

サーバーで指定されたデフォルトの TSM 管理クラスが使用されます。次いで TSM はデータベース構成パラメーターによって指定された管理クラスに、バックアップ・オブジェクトを再バインドします。

このため、デフォルトの管理クラスとデータベース構成パラメーターによって指定された管理クラスには、バックアップ・コピー・グループが含まれていなければなりません。含まれていないと、バックアップ操作は失敗します。

tsm_nodename - Tivoli Storage Manager ノード名

このパラメーターは、Tivoli Storage Manager (TSM) 製品に関連するノード名のデフォルト設定をオーバーライドするために使用されます。

構成タイプ
データベース

パラメーター・タイプ
オンラインで構成可能

伝搬クラス
ステートメント境界

デフォルト [範囲]
NULL [任意のストリング]

ノード名は、別のノードから TSM にバックアップされたデータベースのリストアを可能にするために必要です。

デフォルトでは、バックアップを行ったノード上に TSM からデータベースをリストアすることのみ可能になります。 DB2 でバックアップが行われている (例えば、BACKUP DATABASE コマンドを使用して) 最中に、*tsm_nodename* がオーバーライドされることは可能です。

tsm_owner - Tivoli Storage Manager 所有者名

このパラメーターは、Tivoli Storage Manager (TSM) 製品に関連する所有者のデフォルト設定をオーバーライドするために使用されます。

構成タイプ
データベース

パラメーター・タイプ
オンラインで構成可能

伝搬クラス
ステートメント境界

デフォルト [範囲]
NULL [任意のストリング]

所有者名は、別のノードから TSM にバックアップされたデータベースをリストアできるようにする場合に必要です。DB2 を介した (例えば BACKUP DATABASE コマンドによる) バックアップ時に、*tsm_owner* をオーバーライドすることが可能です。

注: 所有者名には、大文字と小文字の区別があります。

デフォルトでは、バックアップを行ったノード上に TSM からデータベースをリストアすることのみ可能になります。

tsm_password - Tivoli Storage Manager パスワード

このパラメーターは、Tivoli Storage Manager (TSM) 製品に関連するパスワードのデフォルト設定をオーバーライドするために使用されます。

構成タイプ
データベース

パラメーター・タイプ
オンラインで構成可能

伝搬クラス
ステートメント境界

デフォルト [範囲]
NULL [任意のストリング]

パスワードは、別のノードから TSM にバックアップされたデータベースのリストアを可能にするために必要です。

注: *tsm_nodename* が DB2 での (例えば BACKUP DATABASE コマンドによる) バックアップ時にオーバーライドされる場合、*tsm_password* も設定する必要があります。

デフォルトでは、バックアップを行ったノード上に TSM からデータベースをリストアすることのみ可能になります。 DB2 でバックアップが行われている最中に、*tsm_nodename* がオーバーライドされることは可能です。

user_exit_status - ユーザー出口状況標識

このパラメーターが On に設定されている場合は、データベース・マネージャーがロールフォワード・リカバリーのために使用可能状態であり、ユーザー出口プログラムは、データベース・マネージャーによって呼び出されると、ログ・ファイルのアーカイブおよび検索に使用されることを示します。

構成タイプ
データベース

パラメーター・タイプ
通知

userexit - ユーザー出口使用可能

このパラメーターはバージョン 9.5 では推奨されませんが、バージョン 9.5 より前のデータ・サーバーおよびクライアントでは引き続き使用されています。DB2 バージョン 9.5 のデータベース・マネージャーは、この構成パラメーターに対して指定された値をすべて無視します。

注: 以下の情報は、バージョン 9.5 より前のデータ・サーバーおよびクライアントだけに当てはまります。

このパラメーターが使用可能な場合は、*logretain* パラメーターの設定にかかわらず、ログ保持ロギングが実行されます。このパラメーターは、ユーザー出口プログラムをログ・ファイルのアーカイブと検索のために使用する必要があることも示します。

構成タイプ
データベース

パラメーター・タイプ
構成可能

デフォルト [範囲]
Off [On; Off]

ログ・ファイルは、いっぱいになるとアーカイブされます。アーカイブされたログ・ファイルは、ROLLFORWARD ユーティリティーがデータベースを格納するためにログ・ファイルの使用が必要になるときにリトリーブされます。

logretain、*userexit*、またはこれらのパラメーターの両方が使用可能になった後は、データベースのフル・バックアップを作成する必要があります。この状態は、*backup_pending* フラグ・パラメーターによって示されます。

これらのパラメーターの両方が選択解除されている場合は、ログが保持されないので、そのデータベースではロールフォワード・リカバリーが使用できなくなります。この場合は、データベース・マネージャーはオンライン・アーカイブ・ログ・ファイルを含めて、 *logpath* ディレクトリーのすべてのログ・ファイルを削除し、新しいアクティブ・ログ・ファイルを割り振り、循環ロギングに戻ります。

util_heap_sz - ユーティリティー・ヒープ・サイズ

このパラメーターは、BACKUP、RESTORE、および LOAD (ロード・リカバリーを含む) ユーティリティーによって同時に使用できるメモリーの最大量を示します。

構成タイプ
データベース

パラメーター・タイプ
オンラインで構成可能

伝搬クラス
即時

デフォルト [範囲]
5000 [16 - 524 288]

単位 ページ (4 KB)

割り振られるタイミング
データベース・マネージャー・ユーティリティーにより必要とされたとき

解放されるタイミング
ユーティリティーがメモリーを必要としなくなったとき

推奨: ユーティリティーがスペースを使い尽くした場合は、この値を大きくする必要がありますが、そうならない限り、デフォルト値を使用してください。システム上のメモリーが制約されている場合は、このパラメーターの値を小さくして、デー

タベース・ユーティリティーによって使用されるメモリーを制限できます。このパラメーターを低すぎる値に設定した場合、複数のユーティリティーを同時に実行できない可能性があります。必要に応じて、このパラメーターを動的に更新してください。ユーティリティーの数が少ない場合は、このパラメーターを小さい値に設定します。ユーティリティーの数が多い場合、またはメモリーの使用量が大きいユーティリティーの場合は、このパラメーターを大きい値に設定します。

vendoropt - ベンダー・オプション

このパラメーターでは、バックアップ、リストア、またはロード・コピー操作中にストレージ・システムと通信するために DB2 が使用することのできる追加パラメーターを指定します。

構成タイプ
データベース

適用

- ローカルおよびリモート・クライアントを持つデータベース・サーバー
- クライアント
- ローカル・クライアントを持つデータベース・サーバー
- ローカルとリモート・クライアントを持つパーティション・データベース・サーバー

パラメーター・タイプ
オンラインで構成可能

デフォルト [範囲]
Null [ ]

制約事項

vendoropt 構成パラメーターを使用して、ベンダー固有のオプションをスナップショット・バックアップやリストアの操作に指定することはできません。代わりに、バックアップまたはリストアのユーティリティーの OPTIONS パラメーターを使用する必要があります。

DB2 Administration Server (DAS) 構成パラメーター

authentication - 認証タイプ DAS

このパラメーターは、ユーザーの認証が行われる方法とその場所を決定します。

構成タイプ
DB2 Administration Server

適用 DB2 Administration Server

パラメーター・タイプ
構成可能

デフォルト [範囲]
SERVER_ENCRYPT [SERVER_ENCRYPT; KERBEROS_ENCRYPT]

認証が SERVER_ENCRYPT の場合は、ユーザー ID とパスワードがクライアントからサーバーに送信されるので、認証はサーバー上で行うことができます。ネットワークを通して送信されるパスワードは、暗号化されています。

値が KERBEROS_ENCRYPT の場合は、認証のための Kerberos セキュリティー・プロトコルを使用して、認証が Kerberos サーバーで実行されることを示します。

注: KERBEROS_ENCRYPT 認証タイプは、Windows が稼働しているサーバーでのみサポートされます。

このパラメーターは、バージョン 9 コマンド行プロセッサー (CLP)からのみ更新可能です。

contact_host - 連絡先リストのロケーション

このパラメーターは、スケジューラーおよびヘルス・モニターによる通知に使用される連絡先情報が保管されるロケーションを指定します。

構成タイプ
DB2 Administration Server

適用 DB2 Administration Server

パラメーター・タイプ
オンラインで構成可能

伝搬クラス
即時

デフォルト [範囲]
Null [任意の有効な DB2 Administration Server TCP/IP ホスト名]

この場所は、DB2 Administration Server の TCP/IP ホスト名として定義されます。*contact_host* がリモート DAS に置くようにすると、複数の DB2 Administration Server 間での連絡先リストの共用がサポートされます。*contact_host* が指定されていない場合、DAS は、連絡先情報がローカルに指定されているものと見なします。

このパラメーターは、バージョン 8 コマンド行プロセッサー (CLP)からのみ更新可能です。

das_codepage - DAS コード・ページ

このパラメーターは、DB2 Administration Server によって使用されるコード・ページを示します。

構成タイプ
DB2 Administration Server

適用 DB2 Administration Server

パラメーター・タイプ
オンラインで構成可能

伝搬クラス
即時

デフォルト [範囲]
Null [任意の有効な DB2 コード・ページ]

このパラメーターが NULL の場合は、システムのデフォルト・コード・ページが使用されます。このパラメーターは、ローカル DB2 インスタンスのロケールと互換性のあることが必要です。 互換性がない場合、DB2 Administration Server は DB2 インスタンスと通信できません。

このパラメーターは、バージョン 8 コマンド行プロセッサー (CLP)からのみ更新可能です。

das_territory - DAS テリトリー

このパラメーターは、DB2 Administration Server によって使用される territory を示します。

構成タイプ
DB2 Administration Server

適用 DB2 Administration Server

パラメーター・タイプ
オンラインで構成可能

伝搬クラス
即時

デフォルト [範囲]
Null [任意の有効な DB2 テリトリー]

このパラメーターが NULL の場合は、システムのデフォルト territory が使用されます。

このパラメーターは、バージョン 8 コマンド行プロセッサー (CLP)からのみ更新可能です。

dasadm_group - DAS 管理者権限グループ名

このパラメーターは、DAS の DAS 管理者 (DASADM) 権限を持つグループ名を定義します。

構成タイプ
DB2 Administration Server

適用 DB2 Administration Server

パラメーター・タイプ
構成可能

デフォルト [範囲]
NULL [任意の有効なグループ名]

DASADM 権限は、DAS 内の最高レベルの権限です。

DASADM 権限は、特定の運用環境で使用されるセキュリティー機能によって決定されます。

- Windows オペレーティング・システムでは、このパラメーターは、Windows セキュリティー・データベースで定義される任意のローカル・グループに設定することができます。グループ名は長さが 30 バイト以下である限り、受け入れられます。このパラメーターに「NULL」を指定する場合、管理者グループのすべてのメンバーは DASADM 権限を持っています。
- Linux および UNIX システムでは、このパラメーターの値として「NULL」を指定する場合、DASADM グループのデフォルトは、インスタンス所有者の 1 次グループです。

  値が「NULL」でない場合、DASADM グループは有効な UNIX グループ名です。

このパラメーターは、バージョン 8 コマンド行プロセッサー (CLP)からのみ更新可能です。

db2system - DB2 サーバー・システムの名前

このパラメーターは、ユーザーおよびデータベース管理者が DB2 サーバー・システムの識別に使用する名前を指定します。

構成タイプ
DB2 Administration Server

適用 DB2 Administration Server

パラメーター・タイプ
オンラインで構成可能

デフォルト [範囲]
TCP/IP ホスト名 [任意の有効なシステム名]

可能であれば、この名前はご使用のネットワーク内で固有である必要があります。

この名前は、コントロール・センターのオブジェクト・ツリーのシステム・レベルで表示されるので、管理者がコントロール・センターから管理できるサーバー・システムを識別する場合に役立ちます。

構成アシスタントの「ネットワークの検索」機能を使用すると、DB2 ディスカバリーがこの名前を戻し、その結果がオブジェクト・ツリーのシステム・レベルで表示されます。この名前は、ユーザーがアクセスするデータベースが入っているシステムを識別するのに役立ちます。*db2system* の値は、インストール時に次のように設定されます。

- Windows では、セットアップ・プログラムによって Windows システム用に指定されているコンピューター名に等しく設定されます。
- UNIX システムでは、 UNIX システムの TCP/IP ホスト名に等しい名前に設定されます。

discover - DAS ディスカバリー・モード

このパラメーターは、DB2 Administration Server が始動したときに開始されるディスカバリー・モードのタイプを決定します。

構成タイプ
DB2 Administration Server

適用 DB2 Administration Server

パラメーター・タイプ
オンラインで構成可能

伝搬クラス
即時

デフォルト [範囲]
SEARCH [DISABLE; KNOWN; SEARCH]

- discover = SEARCH の場合、Administration Server はクライアントからの SEARCH ディスカバリー要求を処理します。SEARCH は、KNOWN ディスカバリーが提供する機能のスーパーセットを提供します。 discover = SEARCH の場合、Administration Server はクライアントからの SEARCH ディスカバリー要求と KNOWN ディスカバリー要求の両方を処理します。
- discover = KNOWN の場合、Administration Server はクライアントからの KNOWN ディスカバリー要求だけを処理します。
- discover = DISABLE の場合、Administration Server はどのタイプのディスカバリー要求も処理しません。このサーバー・システムの情報は基本的にクライアントから隠されます。

デフォルトのディスカバリー・モードは SEARCH です。

このパラメーターは、バージョン 8 コマンド行プロセッサー (CLP)からのみ更新可能です。

exec_exp_task - 有効期限切れタスクの実行

このパラメーターは、過去にスケジュールされていたが、まだ実行されていないタスクを、スケジューラーが実行するかどうかを指定します。

構成タイプ
DB2 Administration Server

適用 DB2 Administration Server

パラメーター・タイプ
構成可能

デフォルト [範囲]
No [Yes; No]

スケジューラーは、始動時にのみ、有効期限切れタスクを検出します。例えば、毎週土曜日に実行するようにスケジュールされたジョブがあり、スケジューラーが金曜日にオフにされ、月曜日に再始動された場合、土曜日にスケジュールされていたジョブは、過去にスケジュールされていたジョブになります。*exec_exp_task* が Yes に設定されている場合、土曜日のジョブは、スケジューラーが再始動した時点で実行されます。

このパラメーターは、バージョン 8 コマンド行プロセッサー (CLP)からのみ更新可能です。

jdk_path - Software Developer's Kit for Java インストール・パス DAS

このパラメーターは、DB2 Administration Server 関数を実行するために使用する Software Developer's Kit (SDK) for Java がインストールされているディレクトリーを指定します。

構成タイプ
DB2 Administration Server

適用 DB2 Administration Server

パラメーター・タイプ
オンラインで構成可能

伝搬クラス
即時

デフォルト [範囲]
デフォルトの Java インストール・パス [任意の有効なパス]

Java インタープリターで使用される環境変数は、このパラメーターの値から計算されます。

Windows オペレーティング・システムでは、 Java ファイル (必要な場合) は DB2 インストール時に sqllib ディレクトリー (java¥jdk 内にある) の下に置かれています。このとき、*jdk_path* 構成パラメーターは sqllib¥java¥jdk に設定されます。実際は、Java が Windows プラットフォームで DB2 によってインストールされることはなく、Java ファイルはただ sqllib ディレクトリーに置かれるだけで、これは Java がすでにインストールされているかどうかにかかわりなく行われます。

このパラメーターは、バージョン 8 コマンド行プロセッサー (CLP)からのみ更新可能です。

sched_enable - スケジューラー・モード

このパラメーターは、スケジューラーが Administration Server によって開始されるか、されないかを示します。

構成タイプ
DB2 Administration Server

適用 DB2 Administration Server

パラメーター・タイプ
構成可能

デフォルト [範囲]
Off [On; Off]

スケジューラーを使用すると、タスク・センターなどのツールがタスクをスケジュールし、Administration Server で実行できます。

このパラメーターは、バージョン 8 コマンド行プロセッサー (CLP)からのみ更新可能です。

sched_userid - スケジューラー・ユーザー ID

このパラメーターは、スケジューラーがツール・カタログ・データベースに接続するのに使用するユーザー ID を指定します。このパラメーターは、ツール・カタログ・データベースが DB2 Administration Server からリモートにある場合にのみ有効です。

構成タイプ
DB2 Administration Server

適用 DB2 Administration Server

パラメーター・タイプ
通知

デフォルト [範囲]
NULL [任意の有効なユーザー ID]

スケジューラーがリモート・ツール・カタログ・データベースに接続するのに使用するユーザー ID およびパスワードは、db2admin コマンドを使用して指定します。

smtp_server - SMTP サーバー

スケジューラーがオンの場合、このパラメーターは、スケジューラーから E メールまたはページャー通知を送信する場合に使用する、SMTP サーバーを示します。

構成タイプ
DB2 Administration Server

適用 DB2 Administration Server

パラメーター・タイプ
オンラインで構成可能

伝搬クラス
即時

デフォルト [範囲]
Null [任意の有効な SMTP サーバー TCP/IP ホスト名]

このパラメーターは、スケジューラーおよびヘルス・モニターによって使用されます。

このパラメーターは、バージョン 8 コマンド行プロセッサー (CLP)からのみ更新可能です。

toolscat_db - ツール・カタログ・データベース

このパラメーターは、スケジューラーで使用されるツール・カタログ・データベースを示します。

構成タイプ
DB2 Administration Server

適用 DB2 Administration Server

パラメーター・タイプ
構成可能

デフォルト [範囲]
NULL [任意の有効なデータベース別名]

データベースは、*toolscat_inst* で指定されるインスタンスのデータベース・ディレクトリーにあることが必要です。

このパラメーターは、バージョン 8 コマンド行プロセッサー (CLP)からのみ更新可能です。

toolscat_inst - ツール・カタログ・データベース・インスタンス

このパラメーターは、スケジューラーが toolscat_db および toolscat_schema とともに、ツール・カタログ・データベースを識別するために使用するインスタンス名を示します。

構成タイプ
DB2 Administration Server

適用 DB2 Administration Server

パラメーター・タイプ
構成可能

デフォルト [範囲]
NULL [任意の有効なインスタンス]

ツール・カタログ・データベースには、タスク・センターおよびコントロール・センターによって作成されたタスク情報が含まれます。ツール・カタログ・データベースは、この構成パラメーターで指定されたインスタンスのデータベース・ディレクトリーにあることが必要です。データベースは、ローカルとリモートのいずれかです。ツール・カタログ・データベースがローカルの場合は、インスタンスは、TCP/IP 用に構成する必要があります。データベースがリモートの場合は、データベース・ディレクトリーにカタログされるデータベース・パーティションは TCP/IP ノードであることが必要です。

このパラメーターは、バージョン 8 コマンド行プロセッサー (CLP)からのみ更新可能です。

toolscat_schema - ツール・カタログ・データベース・スキーマ

このパラメーターは、スケジューラーで使用されるツール・カタログ・データベースのスキーマを示します。

構成タイプ
DB2 Administration Server

適用 DB2 Administration Server

パラメーター・タイプ
構成可能

デフォルト [範囲]
NULL [任意の有効なスキーマ]

スキーマは、データベース内のツール・カタログ表およびビューのセットを固有に識別する場合に使用します。

このパラメーターは、バージョン 8 コマンド行プロセッサー (CLP)からのみ更新可能です。

第 5 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。

資料番号は、資料が更新される度に大きくなります。資料を参照する際は、以下にリストされている最新版であることを確認してください。

注: DB2 インフォメーション・センターは、PDF やハードコピー資料よりも頻繁に更新されます。

表 72. 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	SC88-4431-01	入手可能
管理ルーチンおよびビュー	SC88-4435-01	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	SC88-4433-01	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	SC88-4434-01	入手可能
コマンド・リファレンス	SC88-4432-01	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	SC88-4421-01	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	SC88-4423-01	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	SC88-4259-01	入手可能
データベース・セキュリティー・ガイド	SC88-4418-01	入手可能
ADO.NET および OLE DB アプリケーションの開発	SC88-4425-01	入手可能
組み込み SQL アプリケーションの開発	SC88-4426-01	入手可能
Java アプリケーションの開発	SC88-4427-01	入手可能
Perl および PHP アプリケーションの開発	SC88-4428-01	入手不可
SQL および外部ルーチンの開発	SC88-4429-01	入手可能
データベース・アプリケーション開発の基礎	GC88-4430-01	入手可能

表 72. 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
DB2 インストールおよび管理概説 (Linux および Windows 版)	GC88-4439-01	入手可能
国際化対応ガイド	SC88-4420-01	入手可能
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	GC88-4438-01	入手可能
Net Search Extender 管理およびユーザーズ・ガイド	SC88-4630-01	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	SC88-4419-01	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	GC88-4441-01	入手不可
DB2 サーバー機能 概説およびインストール	GC88-4440-01	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	SC88-4629-01	入手可能
SQL リファレンス 第 1 巻	SC88-4436-01	入手可能
SQL リファレンス 第 2 巻	SC88-4437-01	入手可能
システム・モニター ガイドおよびリファレンス	SC88-4422-01	入手可能
問題判別ガイド	GI88-4108-01	入手不可
データベース・パフォーマンスのチューニング	SC88-4417-01	入手可能
Visual Explain チュートリアル	SC88-4449-00	入手不可
新機能	SC88-4445-01	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	SC88-4446-01	入手可能
pureXML ガイド	SC88-4447-01	入手可能
XQuery リファレンス	SC88-4448-01	入手不可

表 73. 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	GC88-4443-01	入手可能

表 73. 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	GC88-4444-01	入手可能
DB2 Connect ユーザーズ・ガイド	SC88-4442-01	入手可能

表 74. 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、688 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・センターでの希望する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、ページを最新表示します。希望する言語で DB2 インフォメーション・センターが表示されます。
- Firefox または Mozilla Web ブラウザーの場合に、希望する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、ページを最新表示します。希望する言語で DB2 インフォメーション・センターが表示されます。

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 から資料の更新を入手して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、更新を適用できるようになります。非管理者および非 root の DB2 インフォメーション・センターは常にスタンドアロン・モードで実行されます。を参照してください。
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入手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入手します。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・ディレクトリーから 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・ディレクトリーから 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. インストール・プロセスを開始するには、インストールする更新をチェックして選択し、「**更新のインストール**」をクリックします。
5. 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. 次のようにして、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。help_start.bat は、Ctrl-C や他の方法を使用して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。

商標

以下は、International Business Machines Corporation の米国およびその他の国における商標です。

pureXML	MQSeries
Informix	DB2
400	POWER6
AIX	POWER4
照合	System z
AIX 5L	POWER
Encina	Lotus Notes
WebSphere	OS/390
DB2 Connect	SP
TXSeries	Redbooks
z/OS	System i
CICS	IBM
zSeries	Lotus
Tivoli	DRDA
OS/400	eServer
RS/6000	ibm.com
pSeries	DS8000

以下は、それぞれ各社の商標または登録商標です。

- Linux は、Linus Torvalds の米国およびその他の国における商標です。
- 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。
- Intel、Itanium は、Intel Corporation または子会社の米国およびその他の国における商標または登録商標です。
- Microsoft および Windows は、Microsoft Corporation の米国およびその他の国における商標です。

他の会社名、製品名およびサービス名等はそれぞれ各社の商標です。

索引

日本語, 数字, 英字, 特殊文字の順に配列されています。なお, 濁音と半濁音は清音と同等に扱われています。

[ア行]

[カ行]

[サ行]

[タ行]

[ナ行]

[ハ行]

[マ行]

[ヤ行]

[ラ行]

[数字]

A

B

C

D

M

N

O

P

Q

R

S

T

U

V

W

X

Printed in Japan

SC88-4259-01

日本アイ・ビー・エム株式会社
〒106-8711　東京都港区六本木3-2-12

Spine information:

DB2 Version 9.5 for Linux, UNIX, and Windows

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