IBM solidDB
IBM solidDB Universal Cache

バージョン 6.5

インメモリー・データベース・ユーザー・ガイド

Information Management software

SC88-8165-00
(英文原典：SC23-9875-00)

IBM solidDB
IBM solidDB Universal Cache

バージョン 6.5

インメモリー・データベース・ユーザー・ガイド

Information Management software

SC88-8165-00
(英文原典：SC23-9875-00)

注記

本書および本書で紹介する製品をご使用になる前に、 53 ページの『特記事項』に記載されている情報をお読みください。

本書は、バージョン 6 リリース 5 の IBM solidDB (製品番号 5724-V17) および IBM solidDB Universal Cache (製品番号 5724-W91)、および新しい版で明記されていない限り、以降のすべてのリリースおよびモディフィケーションに適用されます。

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

原典： SC23-9875-00
IBM solidDB
IBM solidDB Universal Cache
Version 6.5
In-Memory Database User Guide

発行： 日本アイ・ビー・エム株式会社

担当： トランスレーション・サービス・センター

第1刷 2009.10

目次

表

本書について

IBM® solidDB® インメモリー・データベースは、特有のデュアル・エンジンを備えたデータベース管理システム (DBMS) アーキテクチャーを採用しています。そのため、最大パフォーマンスと大容量データ処理能力の間で最適なバランスを選択できます。データベース・サーバーには、2 つのエンジンが搭載されています。1 つはメイン・メモリー・エンジン (MME) であり、パフォーマンスが重要なデータに対して可能な最高速度のアクセスを提供します。もう 1 つは従来のディスク・ベース・エンジンであり、ほぼすべてのデータ・ボリュームを効率的に処理できます。

solidDB メイン・メモリー・エンジンは、solidDB ディスク・ベース・エンジンおよび solidDB の機能に基づいて構築されています。したがって、solidDB メイン・メモリー・エンジンは、これらの製品のすべての機能を継承しています。solidDB メイン・メモリー・エンジンは、組み込みシステムで使用することができるため、管理や保守はほとんど必要ありません。solidDB を高可用性構成でデプロイすることにより、solidDB メイン・メモリー・エンジンを高可用性システム用に最適化できます。また、拡張レプリケーション・コンポーネントをデプロイすることもできます。その場合、複数の solidDB メイン・メモリー・エンジンのサーバーと solidDB ディスク・ベース・エンジンのサーバーが、互いにデータを共有し、同期化することができます。

本書では、インメモリー・データベース・テクノロジーを使用することにより、データベース・サーバーのパフォーマンスを最適化できる機能を紹介します。

本書の読者は、リレーショナル・データベース管理システムに関する全般的な知識を備え、また SQL をよく知っているものとします。さらに、solidDB 製品ファミリーに関する基本的な知識も備えていると想定します。本書をお読みになる前に、「*IBM solidDB 管理者ガイド*」をお読みください。まだリレーショナル・データベースに精通していない場合は、「*IBM solidDB スタートアップ・ガイド* 」および「*IBM solidDB SQL ガイド*」も最初にお読みください。

書体の規則

solidDB の資料では、以下の書体の規則を使用します。

表 1. 書体の規則

フォーマット	用途
データベース表	このフォントは、すべての通常テキストに使用します。
NOT NULL	このフォントの大文字は、SQL キーワードおよびマクロ名を示しています。
`solid.ini`	これらのフォントは、ファイル名とパス式を表しています。

表 1. 書体の規則 (続き)

フォーマット	用途
`SET SYNC MASTER YES;` `COMMIT WORK;`	このフォントは、プログラム・コードとプログラム出力に使用します。SQL ステートメントの例にも、このフォントを使用します。
run.sh	このフォントは、サンプル・コマンド行に使用します。
TRIG_COUNT()	このフォントは、関数名に使用します。
java.sql.Connection	このフォントは、インターフェース名に使用します。
LockHashSize	このフォントは、パラメーター名、関数引数、および Windows® レジストリー項目に使用します。
argument	このように強調されたワードは、ユーザーまたはアプリケーションが指定すべき情報を示しています。
管理者ガイド	このスタイルは、他の資料、または同じ資料内の他の章の参照に使用します。新しい用語や強調事項もこのように記述します。
ファイル・パス表示	特に明記していない場合、ファイル・パスは UNIX® フォーマットで示します。スラッシュ (/) 文字は、インストール・ルート・ディレクトリーを表します。
オペレーティング・システム	資料にオペレーティング・システムによる違いがある場合は、最初に UNIX フォーマットで記載します。UNIX フォーマットに続いて、小括弧内に Microsoft® Windows フォーマットで記載します。その他のオペレーティング・システムについては、別途記載します。異なるオペレーティング・システムに対して、別の章を設ける場合があります。

構文表記法の規則

solidDB の資料では、以下の構文表記法の規則を使用します。

表 2. 構文表記法の規則

フォーマット	用途
`INSERT INTO` *`table_name`*	構文の記述には、このフォントを使用します。置き換え可能セクションには、この フォントを使用します。
`solid.ini`	このフォントは、ファイル名とパス式を表しています。
[ ]	大括弧は、オプション項目を示します。太字テキストの場合には、大括弧は構文に組み込む必要があります。
|	垂直バーは、構文行で、互いに排他的な選択項目を分離します。

表 2. 構文表記法の規則 (続き)

フォーマット	用途
{ }	中括弧は、構文行で互いに排他的な選択項目を区切ります。太字テキストの場合には、中括弧は構文に組み込む必要があります。
...	省略符号は、引数が複数回繰り返し可能なことを示します。
. . .	3 つのドットの列は、直前のコード行が継続することを示します。

1 基本フィーチャー

IBM solidDB メイン・メモリー・エンジンは、インメモリー表の特徴であるハイパフォーマンスと、ディスク・ベース表の特徴であるほぼ制限のない容量を兼ね備えています。これは、市場の他のソリューションでは不可能な機能です。純粋なインメモリー・データベースは高速ですが、メモリー・サイズによって厳密に制限されます。純粋なディスク・ベース・データベースでは、ストレージ容量にほとんど制限がありませんが、そのパフォーマンスはディスク・アクセスに支配されます。コンピューターのメモリー・バッファー内に、データベース全体を格納するのに十分なメモリーが存在する場合であっても、ディスク・ベース表のために設計したデータベース・サーバーは低速です。なぜなら、ディスク・ベース表に最適なデータ構造とインメモリー表に最適なデータ構造はまったく異なるためです。

solidDB のソリューションでは、単一のデータベース・サーバーが、最適化された 2 つのサーバーをその内部に持っています。サーバーの 1 つはディスク・ベース・アクセス用に最適化されており、もう 1 つはインメモリー・アクセス用に最適化されています。どちらのサーバーも同じプロセス内に共存し、単一の SQL ステートメントで両方のエンジンのデータにアクセスできます。

インメモリー表

インメモリー表とディスク・ベース表

表をインメモリー表として指定した場合、その表の内容全体がメモリーに格納されるため、データへのアクセスは可能な限り高速になります。一方、表がディスク・ベース表の場合、データは主にディスクに保管され、通常、サーバーからメモリーに毎回ごく少量のデータのみコピーされます。

ほとんどの点で、インメモリー表とディスク・ベース表は類似しています。最も重要な点は、どちらの表も、他に異なる指定をされない限り、データを保持し続けるということです。それぞれの表で同じタイプの照会を実行することが可能です。同じ SQL 照会またはトランザクションの中で、ディスク・ベース表とインメモリー表を組み合わせて使用できます。SA API を使用している場合は、両方のタイプの表を照会できます。また、インメモリー表は、索引、トリガー、ストアード・プロシージャーなどと組み合わせて使用できます。インメモリー表では、主キー制約や外部キー制約などの制約も使用できます。ただし、ノンパーシスタント表での外部キー制約については、いくつかの制限があります。

solidDB では、どの表をインメモリー表にするか、およびどの表をディスク・ベース表にするかをユーザーが決定できます。例えば、使用頻度の高い表をメイン・メモリー内に置くと、より高速にその表にアクセスでき、十分なメモリーがあれば、すべての表をメイン・メモリー内に置くことができます。

インメモリー表のタイプ

solidDB には、パーシスタントとノンパーシスタントの 2 種類の異なるインメモリー表が用意されています。これらについて以下に説明します。

パーシスタント・インメモリー表

名前のパーシスタントが「永続的」を意味することから分かるように、パーシスタント・インメモリー表の存続期間に制限はありません。クライアントの照会はメモリー内のデータのコピーにアクセスしますが、サーバーはシャットダウン時にディスクにインメモリー表を格納するため、そのデータはサーバーが始動するときに常に使用可能になります。インメモリー表は、トランザクション・ロギングも使用します。これは、電源障害などが原因でサーバーが突然シャットダウンした場合に、発生したトランザクションのレコードがサーバーに残り、サーバーが表を更新して、すべてのコミット済みのトランザクションのデータをすべて表に確実に反映できるようにするためです。ディスク・ベース表と同様に、インメモリー表も、チェックポイントの処理中にハード・ディスク・ドライブにデータをコピーします (チェックポイントについて詳しくは「*IBM solidDB 管理者ガイド*」を参照してください)。

また、インメモリー表は、solidDB の HotStandby コンポーネントと組み合わせて使用することもできます。つまり、インメモリー表のデータは、2 次サーバーにコピーされ、1 次サーバーで障害が発生しても 2 次サーバーからそのデータが利用できます。

インメモリー表を使用するアプリケーションを設計する際には、インメモリー表とディスク・ベース表との間にあるいくつかの違いを考慮する必要があります。最も重要な点は、インメモリー表が常に行レベルのペシミスティック並行性制御 (ロック方式) を使用するのに対して、ディスク・ベース表がオプティミスティック並行性制御 (バージョニング方式) をデフォルトで使用することです。したがって、インメモリー表の読み取り操作では、トランザクションの読み取り中に書き込み操作をブロックします。さらに、インメモリー表ではデッドロックが発生する可能がありますが、バージョニング方式のディスク・ベース表では発生する可能性はありません。一方、オプティミスティック並行性制御を使用している場合には、並行性の競合が発生する可能性があります。

アプリケーションのエラー処理を設計する際に、これらの考慮事項に留意することが重要です。エラー処理では、使用する表のタイプによって、異なるエラー・コードを考慮する必要があります。

もう 1 つの重要な違いは、使用されるチェックポイント処理アルゴリズムの違いです。インメモリー表のチェックポイント処理は、ディスク・ベース表で使用されているアルゴリズムとはまったく違います。インメモリー表のチェックポイント処理の大きな利点は、チェックポイント処理中に決して、トランザクションが表へアクセスすることをブロックしないことです。したがって、応答時間の予測可能性の点では、インメモリー表の方がディスク・ベース表よりも優れています。

インメモリー表とディスク・ベース表の 3 番目の大きな違いは、インメモリー表では副次索引がディスクに書き込まれることは決してないということです。その代わりに、それらはメモリー内にのみ維持され、サーバーの始動時に再作成されます。したがって、書き込みのパフォーマンスに対する影響は、ディスク・ベース表に比べ相当に小さくなります。また、インメモリー表では、すべての索引の速度が等しく高速ですが、ディスク・ベース表では、主キーの速度が他の索引よりも大幅に速くなっています。

他のあらゆる点から考えて、インメモリー表の方が全般的に相当速いことを除いて、インメモリー表とディスク・ベース表との間に違いはありません。

ノンパーシスタント・インメモリー表

ノンパーシスタント・インメモリー表は、サーバーのシャットダウン時にディスクに書き込まれません。したがって、正常であっても異常であっても、サーバーがシャットダウンがすれば、ノンパーシスタント表のデータは消失します。これらのデータは、ログに記録されず、チェックポイント処理も行われません。そのためリカバリー不能ですが、パーシスタント表よりも非常に高速です。

ノンパーシスタント表は、「スクラッチパッド」表として主に使用されます。例えば、パーシスタント表からデータをコピーし、一連の解析を実行した後に、非永続的なコピーを廃棄する場合が考えられます。また、一連の長いトランスフォーメーションを実行した後に、最終的な結果をパーシスタント表にコピーして戻す場合も考えられます。

ノンパーシスタント・インメモリー表には、トランジエント表とテンポラリー表という 2 つの異なるタイプの表があります。2 つの違いについては、17 ページの『2 章 テンポラリー表とトランジエント表』に詳しい説明があります。ここでは簡単な概要のみを説明します。

トランジエント表

トランジエント表は、データベース・サーバーがシャットダウンするまで存続します。複数のユーザーが同じトランジエント表を使用することができ、各ユーザーは別のユーザーのデータをすべて見ることができます。

トランジエント表は、パーシスタント・インメモリー表と比べるといくつか制限があります。例えば、トランジエント表には、外部キー (参照制約) を使用する上でいくつか制限があります。また、トランジエント表は、HotStandby 2 次サーバーにはコピーされません。

テンポラリー表

テンポラリー表のデータは、データを挿入した接続にのみ可視で、データはその接続の期間中のみ保持されます。テンポラリー表は、他の人が参照できない個人用のスクラッチパッドに似ています。

ロギングが回避されることに加えて、いずれのタイプの並行性制御メカニズム (レコードのロック方式など) も使用しないため、トランジエント表よりもさらに高速です。

インメモリー表でパフォーマンスが向上する仕組み

ディスク・ベース表に格納されているデータは、使用する前にメモリーに読み取る必要があります。そして、使用後に、ディスクに書き戻す必要があります。インメモリー表では、すべてのデータが常にメイン・メモリーに存在するため、パフォーマンスが向上します。そして、サーバーは、より効率的な技法を使用して、データのアクセスと操作に最大のパフォーマンスを実現できます。

ほとんどすべてのデータベース・サーバーでは、メモリーの量が多いほど動作が高速になり、サーバーのキャッシュ・メモリーに格納できるデータのパーセンテージが大きくなります。しかし、solidDB メイン・メモリー・エンジンのハイパフォーマンス・インメモリー・テクノロジーは、単にデータをメモリーにコピーする以上に多くの機能を備えています。また、solidDB メイン・メモリー・エンジンは、完全にメモリー内に格納されているデータを処理するように最適化された索引構造を使用します。 solidDB メイン・メモリー・エンジンは、表が拡大または縮小する際のメモリーの「フラグメント化」など、インメモリー表で発生する問題も考慮しています。

インメモリー表として指定する表を決定する方法

メモリー内にすべての表を格納するのに十分なメモリーがコンピューターに搭載されており、可能な最高のパフォーマンスをデータベース・トランザクションに提供できれば理想的です。しかし、現実の世界では、大部分のユーザーは、メモリー内に格納する表のサブセットを選択し、残りの表をディスク・ベースにする必要があります。

もちろん、メモリー内にすべての表を格納できない場合は、最も頻繁に使用するデータをメモリーに格納しようと試みるはずです。小さくて頻繁に使用する表をメモリーに入れ、大きくてあまり使用しない表をディスクに残すことは誰でも容易に考え付きます。しかし、大きな表を非常によく使用する場合や、小さな表をあまり使用しない場合など、可能性のある他の組み合わせについては、どうするべきでしょうか。

この点について検討する最も良い方法は、表に対するアクセスの「密度」を考えることです。メガバイト単位当たりの 1 秒間のアクセス数が多いほど、その表をメモリー内に格納すると効果が大きくなります。詳細なアルゴリズムについては、 23 ページの『メモリー内に格納する表を選択するアルゴリズム』を参照してください。

メモリー内に表を格納すると決定したら、そのデータをパーシスタント表、トランジエント表、またはテンポラリー表のいずれで格納するか選択する必要があります。基本的な規則を、以下に示します。これらの規則はガイドラインであり、厳密な規則ではないことに注意してください。最終的な決定を下す前に、トランジエント表とテンポラリー表に関する詳細な説明を確認してください。

以下の説明で使用する「サーバー・セッション」という用語は、サーバーが始動してから、意図的にシャットダウンするか、または予期しない理由 (電源障害など) で停止するまでの、サーバーの「1 回の実行」を意味しています。「接続」は、単一ユーザーがサーバーに接続した後、そのユーザーがその接続を切断する時点まで継続します。(1 人のユーザーは、複数の接続を確立できますが、それらの接続は互いに独立しています。)

最初に「はい」と回答する質問に達するまで以下の質問に答えると、最も適切な表のタイプを決定できます。

1. 次回のサーバー始動時に、同じデータを再び使用できる必要がありますか。答えが「はい」の場合は、パーシスタント表を使用してください。
2. そのデータを 2 次 HotStandby サーバーにコピーする必要がありますか。答えが「はい」の場合は、パーシスタント表を使用してください。

3. 現行サーバー・セッション中にのみそのデータが必要で、そのデータを複数のユーザーから使用できる (または同一ユーザーからの複数の接続で使用できる) 必要がありますか。答えが「はい」の場合は、トランジエント表を使用してください。
4. 上記の規則のいずれにも該当しない場合は、テンポラリー表を使用してください。

重ねて注意しますが、最終的な決定を下す前に、テンポラリー表とトランジエント表に関する詳細な説明を確認してください。テンポラリー表とトランジエント表には、外部キー (参照制約) の制限など、いくつかの制限事項があり、決定に影響する可能性があります。

表をメモリーに格納するよう指定する方法

表をメモリー内に配置するか、またはディスク上に配置するか、明示的に指定する方法が 2 つあります。システムにより、デフォルトでは、ディスク・ベース表が作成されます。

1. CREATE TABLE コマンドまたは ALTER TABLE コマンドの STORE 節を使用します。例えば、以下のように指定します。

   ```
   CREATE TABLE employees (name CHAR(20)) STORE MEMORY;
   CREATE TABLE ... STORE DISK;
   ALTER TABLE network_addresses SET STORE MEMORY;
   ```

   (CREATE TABLE ステートメントおよび ALTER TABLE ステートメントの構文について詳しくは、「*IBM solidDB SQL ガイド*」を参照してください。)
2. solid.ini ファイルにデフォルトを指定します。便宜上、新しく作成する表の大部分またはすべてが同じタイプ (例えば、インメモリー) の場合には、別に指定しない限り、このストレージ・タイプを自動的に使用するようサーバーに指示できます。新しい表にデフォルト・タイプを指定するには、solid.ini 構成ファイルに以下のパラメーターを設定します。

   ```
   [General]
   DefaultStoreIsMemory=yes
   ```

   このパラメーターを「yes」に設定すると、CREATE TABLE ステートメントで別の指定をしない限り、新しい表はインメモリー表として作成されます。このパラメーターを「no」に設定すると、CREATE TABLE ステートメントで別の指定をしない限り、新しい表はディスク・ベース表として作成されます。solid.ini ファイル内の他のパラメーターと同様に、このパラメーターを変更した場合、新しい値は、次回サーバーを始動するまで有効になりません。
   DefaultStoreIsMemory パラメーターについて詳しくは、「*IBM solidDB 管理者ガイド*」を参照してください。

注: これらの説明は、パーシスタント表にのみ適用されます。テンポラリー表またはトランジエント表として宣言された表は、STORE MEMORY 節を使用しなくても、自動的にメモリー内に格納されます。

メモリー使用量

メモリー使用量を理解し、制御することは重要です。インメモリー・データベースまたはサーバー・プロセスがシステムの使用可能な仮想メモリーをすべて使い切る

と、データの追加も更新もできなくなるからです。サーバーが物理メモリーをすべて使い切って、仮想メモリーを使い始めた場合、サーバーの動作は継続しますが、パフォーマンスが大幅に低下します。

インメモリー・データベースがメイン・メモリーを使用する方法は、標準の solidDB の使用法とは異なります。インメモリー・データベースは、そのデータベース専用のメモリー・プールに常駐しています。solidDB のメモリー使用量について詳しくは、「*IBM solidDB 管理者ガイド*」を参照してください。

solidDB メイン・メモリー・エンジンには、インメモリー・データベースおよびサーバー・プロセスのメモリー使用量をモニターおよび制御するためのコマンドと構成パラメーターが用意されています。これらのコマンドとパラメーターの対象は、サーバー全体ではなく、サーバーのインメモリー・データベース機能が中心です。

メモリー使用量のモニター

メモリー使用量をモニターするための管理コマンドは何種類かあります。以下に簡単に説明します。

- ADMIN COMMAND 'info imdbsize';
- ADMIN COMMAND 'info processsize';
- ADMIN COMMAND 'pmon mme';
- ADMIN COMMAND 'memory';

これらのコマンドについて以下に説明します。

下記のコマンドの場合

ADMIN COMMAND 'info imdbsize';

インメモリー・データベースの表と索引用に現在割り振られているメモリーの量を返します。戻り値は VARCHAR で、サーバーの使用サイズをキロバイト単位で示します。返されるのは仮想メモリーの使用量で、物理メモリーの使用量ではないことに注意してください。

時間が経過すると、imdbsize の値が大きくなることがあります。その理由は、データをオペレーティング・システムに戻す操作は、割り振り単位でのみ可能であり、その割り振り単位は、オペレーティング・システムに戻す前に完全に未使用の状態でなければならないためです。

一時的なメモリー割り振り (SQL 実行のグラフなど) は、ADMIN COMMAND 'info imdbsize'; のレポートから除外されます。

下記のコマンドの場合

ADMIN COMMAND 'info processsize';

仮想メモリー・プロセスのサイズを返します。つまり、インメモリー・データベース・プロセスが使用するデータベース・サーバー・アドレス・スペース全体のサイズです。戻り値は VARCHAR で、プロセスの使用サイズをキロバイト単位で示します。返されるのは仮想メモリーの使用量で、物理メモリーの使用量ではないことに注意してください。

また、インメモリー・データベース・サーバーに関連するランタイム情報を含むパフォーマンス・カウンターもいくつか入手できます。下記のコマンドを入力します。

ADMIN COMMAND 'pmon mme';

以下のような現在のカウンター値を示すリストが生成されます。

```
RC TEXT
-- ----
 0 Performance statistics:
 0 Time (sec)                              30    21    Total
 0 MME current number of locks       :     0     0        0
 0 MME maximum number of locks       :     0     0        0
 0 MME current number of lock chains :     0     0        0
 0 MME maximum number of lock chains :     0     0        0
 0 MME longest lock chain path       :     0     0        0
 0 MME memory used by tuples         :     0     0        0
 0 MME memory used by indexes        :     0     0        0
 0 MME memory used by page structures:     0     0        0
10 rows fetched.
```

パフォーマンス統計リストで、タプル、索引、およびページ構造が使用するメモリー容量はキロバイト単位で示されます。

下記のコマンドの場合

ADMIN COMMAND 'memory';

動的に割り振られたヒープ・メモリーの量だけを報告します。ヒープ・ベースのメモリー割り振りでは、メモリーは、ヒープと呼ばれる未使用メモリー領域の大容量プールから割り振られます。ヒープ・メモリー割り振りのサイズは、実行時に決定することができます。一時的なメモリー割り振り (SQL 実行のグラフなど) は、ADMIN COMMAND 'mem'; のレポートの中に含まれます。

メモリー使用量の制御

インメモリー・データベースのメモリー使用量は、solid.ini ファイルの [MME] セクションにある、以下の 3 つの構成パラメーターによって制御されます。

- **ImdbMemoryLimit**
- **ImdbMemoryLowPercentage**
- **ImdbMemoryWarningPercentage**

さらに、プロセス ・メモリーの使用量は、solid.ini ファイルの [SRV] セクションにある、以下の 4 つの構成パラメーターによって制御されます。

- **ProcessMemoryLimit**
- **ProcessMemoryLowPercentage**
- **ProcessMemoryWarningPercentage**
- **ProcessMemoryCheckInterval**

インメモリー・データベースのメモリーおよびプロセスの限度に違反した場合は、それが solmsg.out ログ・ファイルにログとして記録されます。**ImdbMemoryLimit** および **ProcessMemoryLimit** パラメーターで定義されたメモリー限度を超えるたび

に、システム・イベントが通知されます。これらのシステム・イベントについては、「*IBM solidDB SQL ガイド*」に記載があります。

ImdbMemoryLimit

MME.ImdbMemoryLimit パラメーターは、インメモリー表 (テンポラリー表、トランジエント表、および「通常」のインメモリー表を含む) およびそれらのインメモリー表の索引に割り振ることができる仮想メモリーの最大容量を指定します。

ImdbMemoryLimit のデフォルト値は 0 です。これは「限度なし」を意味しています。デフォルト値を使用しないことを強く推奨します。このパラメーターには、インメモリー・データが完全に物理メモリー内に確実に収まるような値を設定してください。必然的に、以下の要因を考慮する必要があります。

- コンピューターの物理メモリー量
- オペレーティング・システムにより使用されているメモリー量
- solidDB (当プログラム自体) のメモリー使用量
- solidDB サーバーのキャッシュ用に確保してあるメモリー量 (**IndexFile.CacheSize** `solid.ini` 構成パラメーター)
- サーバー内で並行動作する接続、トランザクション、およびステートメントで必要となるメモリーの容量。サーバー内の並行接続とアクティブ・ステートメントが多くなるほど、サーバーに必要な作業メモリーが増えます。通常、サーバー内のクライアント接続については、それぞれ 0.5 MB 以上のメモリーを割り振る必要があります。
- コンピューター内で実行中の他のプロセス (プログラムとデータ) が使用するメモリー

限度に到達した (インメモリー表が、 **ImdbMemoryLimit** で指定されたメモリーを 100% 使い切ろうとしている) 場合、サーバーはインメモリー表に対する UPDATE 操作を禁止します。限度に到達する前に、サーバーは新しいインメモリー表の作成を禁止し、それらの表に対する INSERT 操作を禁止します。詳しくは、**ImdbMemoryLowPercentage** パラメーターの説明を参照してください。

以下に例を示します。

```
[MME]
ImdbMemoryLimit=1000MB
```

ImdbMemoryLowPercentage

ImdbMemoryLowPercentage 変数は、メモリーのパーセンテージとして (つまり、**ImdbMemoryLimit** に対するパーセンテージとして) 表される「最低水準点」を設定します。サーバーは、指定されたパーセンテージのメモリーを消費すると、メモリー消費が引き続き増大するのを防止するために、アクティビティーの制限を開始します。例えば、**ImdbMemoryLimit** が 1000 MB で **ImdbMemoryLowPercentage** が 90% の場合、サーバーはインメモリー表に割り振られたメモリーが 900 メガバイトを超えると、アクティビティーの制限を開始します。具体的には、サーバーは以下のように動作します。

- それ以上、インメモリー表 (テンポラリー表とトランジエント表を含む) の作成およびインメモリー表に対する索引の作成を禁止します。
- インメモリー表への INSERT を禁止します。

上限自体 (つまり、**ImdbMemoryLimit**) に到達すると、サーバーは、インメモリー表のレコードに対する UPDATE 操作も禁止します。

ImdbMemoryLowPercentage の有効な値は、60 から 99 (パーセント) までの範囲です。

ImdbMemoryWarningPercentage

ImdbMemoryWarningPercentage パラメーターは、IMDB メモリー・サイズの警告限度を設定します。警告限度は、ImdbMemoryLimit パラメーター値に対するパーセンテージとして表されます。ImdbMemoryWarningPercentage 限度を超えると、システム・イベントが発生します。

ImdbMemoryLimit に到達したときの対処方法

この限度に到達したことを示すエラー・メッセージが表示された場合、すぐに有効な対処を実施する必要があります。当面の問題と長期的な問題の両方に対処する必要があります。当面の問題とは、ユーザーが重大なエラーに遭遇しないようにすること、さらにサーバーのシャットダウンまでの間にある程度、メモリーの空き容量を確保するようにして、サーバーを再始動したときに同じ状態 (メモリー不足) にならないようにすることです。長期的な問題とは、今後、表が拡大しても、このような状態にならないようにすることです。

当面の問題に対処するには、一般に以下の作業を行う必要があります。

1. サーバーから切断するようにユーザーに通知します。これには 2 つの効果があります。1 つは、状態が悪化した場合に、その影響を受けるユーザーの数が最小になります。もう 1 つは、テンポラリー表を使用していたユーザーが切断すれば、メモリーの空き容量が増えます。このエラーが発生したときに、ユーザーまたはプログラム、またはその両方が必ず安全な切断を試みるようにするためのポリシーまたはエラー検査コードを決めておくと便利です。
2. メモリー使用量を大幅に低減させるほどのテンポラリー表がなかった場合は、トランジエント表の索引、またはトランジエント表自体 (あれば) を一部ドロップします。

メモリー使用量に大きな差が生じるほどのテンポラリー表とトランジエント表がなかった場合は、さらに徹底した処理を行う必要があります。

1. インメモリー表の 1 つ以上の索引をドロップします。
2. サーバーをシャットダウンします。
3. 「通常の」インメモリー表のみが存在し、そのすべての表に索引がなく、また必要なデータのみを格納しているなど、メモリーに廃棄可能なものがまったくなかった場合は、**ImdbMemoryLimit** の値を少しだけ大きくしてから、サーバーを再始動します。こうすることで、サーバーは仮想メモリーのページングを強制的に開始するため、パフォーマンスは大幅に低下しますが、サーバーを引き続き使用して、長期的な問題に対処できます。以前から **ImdbMemoryLimit** を最大よりもやや低めの値に設定してある場合は、システムに仮想メモリーのページングを強制的に開始させることなく、この時点でやや高めの値に設定できます。
4. サーバーを再始動します。

5. 長期的な問題に対処する時間ができるまでは、システムを使用するユーザーの数をできるだけ減らします。長期的な問題への対処が完了するまでは、必ずテンポラリー表もトランジエント表もユーザーが作成できないようにしてください。

当面の問題を解決し、サーバーのメモリーに最低限必要な少量の空き容量を確保したら、今度は長期的な問題に対処する段階に入ります。

長期的な問題に対処するには、インメモリー表に格納されるデータの量を減らす必要があります。この方法としては、インメモリー表 (テンポラリー表、トランジエント表を含む) の数を少なくするか、またはサイズを小さくする方法と、インメモリー表の索引の数を減らす方法があります。

テンポラリー表 (またはトランジエント表) の使用頻度が高いことだけが問題の原因になっていた場合には、多すぎる数のセッションが大きいテンポラリー表 (またはトランジエント表) を同時に数多く作成しすぎないようにする以外に必要な対処はないことがあります。

「通常の」インメモリー表がメモリーを多量に使いすぎているために問題が発生していた場合と、サーバーの使用可能なメモリー容量を増やすことができない場合には、メイン・メモリーから 1 つ以上の表をディスクに移動する必要があります。幸いにも、これは難しい作業ではありません。表をメモリーからディスクに移動するには、以下の手順を実行します。

1. メモリー内の表の 1 つと同じ構造 (名前は異なる) を持つ空のディスク・ベース表を 1 つ作成します。
2. インメモリー表内の情報をディスク・ベース表にコピーします。[1]
3. インメモリー表をドロップします。
4. 今ドロップしたインメモリー表の元の名前にディスク・ベース表を名前変更します。

予防措置として、実際に使用可能な最大容量よりもやや低めの値を **ImdbMemoryLimit** に意図的に設定するようにしてください。こうしておくと、メモリー不足になったときに、メモリーから除外可能で不要なインメモリー表または索引がない場合に、**ImdbMemoryLimit** を少しだけ大きくして、サーバーを再始動することで、長期的な必要性に対処できる十分な空きメモリー容量を確保できます。

メモリー使用量が増加していることを警告するために、**ImdbMemoryWarningPercentage** を使用できることにも留意してください。

1.

SQL ステートメント (INSERT INTO ...VALUES SELECT FROM) を 1 つ使用して、大きな表のレコードを別の表にコピーしようとする場合には、1 つのトランザクションの中ですべての操作が行われることに留意してください。このような操作は、データの全体量がサーバーのキャッシュ・メモリーに収まる場合にのみ効率的に行われます。トランザクションのサイズがキャッシュ・サイズよりも大きくなると、パフォーマンスは大幅に低下します。したがって、大きな表のデータを別の表にコピーする作業は、簡単なストアード・プロシージャーまたはアプリケーションを使用して、もっと小さなトランザクション (例えば、トランザクションごとの行数が数千件程度) で行うことを推奨します。

また、中間的な表には索引が必要ないことにも注意してください。データのコピーが正常に完了したら、新規の「実際の表」に索引を再作成する必要があります。

どのような状態でも、インメモリー表の数を減らせばよいというわけではありません。場合によっては、単にコンピューターのメモリーを増設する方が現実的な解決策になることもあります。

また、問題を解決しようとするよりは、問題を回避するようにした方がよいことにも留意してください。インメモリー表に使用可能なメモリー容量をすべて使い切る前に、信頼できる警告を通知するように、**ImdbMemoryWarningPercentage** パラメーターに適切な値を設定することを強く推奨します。

ProcessMemoryLimit

ProcessMemoryLimit パラメーターでは、インメモリー・データベース・プロセスに割り振り可能な仮想メモリーの最大量を指定します。**ProcessMemoryLimit** パラメーターは、**ProcessMemoryCheckInterval** パラメーターで制御されます。

ProcessMemoryCheckInterval パラメーター値が 0 (ファクトリー値) の場合は、**ProcessMemoryLimit** パラメーターが無効になります。つまり、プロセス・メモリーの限度は設定されません。

ProcessMemoryLimit のファクトリー値は 1 G (1 ギガバイト) です。インメモリー・データベース・プロセスが必ず物理メモリー内に完全に収まるような値をパラメーターに設定してください。必要なメモリー量に影響する要因は以下のとおりです。

- コンピューターの物理メモリー量
- オペレーティング・システムにより使用されているメモリー量
- インメモリー表 (テンポラリー表、トランジエント表、および「通常の」インメモリー表を含む) およびこうしたインメモリー表の索引により使用されているメモリー量
- solidDB サーバーのキャッシュ用に確保してあるメモリー量 (**CacheSize** `solid.ini` 構成パラメーター)
- サーバー内で並行動作する接続、トランザクション、およびステートメントで必要となるメモリーの容量。サーバー内の並行接続とアクティブ・ステートメントが多くなるほど、サーバーに必要な作業メモリーが増えます。通常、サーバー内のクライアント接続については、それぞれ 0.5 MB 以上のメモリーを割り振る必要があります。
- コンピューター内で実行中の他のプロセス (プログラムとデータ) が使用するメモリー

限度に達する、つまりインメモリー・データベース・プロセスが **ProcessMemoryLimit** で指定されたメモリーを 100% 使い切ってしまった場合には、サーバーは admin コマンドしか受け付けなくなります。**ProcessMemoryWarningPercentage** および **ProcessMemoryLowPercentage** パラメーターを使用して、メモリー使用量が増えたときに警告を出すようにすることができます。

ProcessMemoryLowPercentage

ProcessMemoryLowPercentage パラメーターはプロセス・サイズの合計の警告限度を設定します。警告限度は **ProcessMemoryLimit** パラメーター値のパーセントとして表します。この限度を超える前に、**ProcessMemoryWarningPercentage** パラメー

ターを使用して定義された警告限度を超え、警告を受け取っています。**ProcessMemoryLowPercentage** 限度を超えるとシステム・イベントが発生します。

ProcessMemoryLowPercentage で設定する限度は、**ProcessMemoryWarningPercentage** の限度よりも大きな値でなければなりません。例えば、**ProcessMemoryWarningPercentage** が 82 に設定されている場合は、**ProcessMemoryLowPercentage** の値を 83 以上にする必要があります。

ProcessMemoryWarningPercentage

ProcessMemoryWarningPercentage パラメーターはプロセス・サイズの合計の最初の警告限度を設定します。警告限度は、**ProcessMemoryLimit** パラメーター値のパーセントとして表します。**ProcessMemoryWarningPercentage** 限度を超えると、システム・イベントが発生します。

ProcessMemoryWarningPercentage で設定する限度は、**ProcessMemoryLowPercentage** の限度よりも小さな値でなければなりません。

ProcessMemoryCheckInterval

プロセス・サイズの限度を定期的に検査します。検査間隔は、**ProcessMemoryCheckInterval** パラメーターを使用して設定します。間隔はミリ秒で指定します。

ゼロ以外の最小値は、1000 (ms)です。0 または 1000、あるいは 1000 (1 秒) を超える値だけが許可されます。指定した値が 0 より大きく、1000 未満の場合は、エラー・メッセージが表示されます。

ファクトリー値は 0 (プロセス・サイズの検査が無効) です。

ProcessMemoryCheckInterval は、さらに **ProcessMemoryLimit** パラメーターを制御します。**ProcessMemoryCheckInterval** パラメーター値が 0 の場合は、**ProcessMemoryLimit** パラメーターが無効になります。つまり、プロセス・メモリーの限度は設定されません。

ディスク・スペースの必要量の計算

データベースの格納に必要なディスク・スペースを計算する際には、ディスク・ベース表だけでなく、インメモリー表を格納するのに必要なスペース容量を考慮する必要があります。その理由は、サーバーのシャットダウン時に、ディスク・ドライブにインメモリー・データを格納するためです。(サーバーは、この情報を再始動時にメモリーに読み戻します。)

また、シャットダウン時にサーバーはディスク・ドライブにインメモリー・データを書き込む必要があるため、サーバーにインメモリー表があると、通常、ディスク・ベース表だけを持つサーバーに比べてシャットダウンにかかる時間が長くなります。同様に、始動時にサーバーはディスク・ドライブからメモリーにデータを再ロードする必要があるため、サーバーにインメモリー表があると、通常、始動に余分な時間がかかります。

規格準拠

インメモリー表の機能は、SQL-99 対応の ANSI 規格の一部ではありません。

インメモリー表の制限

物理メモリーと仮想メモリー

最も明らかな制限は、インメモリー・データベース表の合計サイズは使用可能な仮想メモリーの容量を超えることができない点です。

重要:

仮想メモリーは頻繁にディスクにスワップされるため、仮想メモリーを使用すると、インメモリー表の利点が部分的に損なわれます。インメモリー表のサイズを使用可能な仮想メモリーのサイズではなく、使用可能な物理メモリーのサイズよりも小さくすることを推奨します。

表に必要なスペース容量を計算する際に、「BLOB」データを考慮するのを忘れないでください。一般的に、メイン・メモリーの表では BLOB 列の最大サイズは大幅に小さくなるため、BLOB データは、ディスク・ベース表で保持する必要があります。

表の格納に必要なスペース容量としては、表内のデータ用のスペースだけでなく、主キー制約および外部キー制約のサポート用に作成される索引を含む、表のすべての索引用のスペースも考慮に入れてください。また、表が占有するスペースは、ディスクよりもメモリーでの方が大幅に大きくなります。

INSERT 操作または ALTER TABLE 操作の実行中に表を拡張するときなど、サーバーがメモリーを割り振ろうとしたときに仮想メモリーを使い尽くすと、エラー・メッセージが表示されます。

インメモリー表からディスク・ベース表へ、またはその逆への変更

表が空の場合、表のタイプをインメモリーからディスク・ベースに、またはその逆に変更できます。そのためには、以下のコマンドを使用します。

```
ALTER TABLE table_name SET STORE MEMORY | DISK
```

表にデータが入っている場合は、データをコピーするための新しい表を (別の名前で) 作成する必要があります。新しい表にデータをコピーした後で、古い表をドロップし、以下のコマンドを使用して、新しい表の名前を変更できます。

```
ALTER TABLE current_table_name SET TABLE NAME new_table_name
```

トランザクション分離

SERIALIZABLE 分離レベルはサポートされていません。

トランザクション分離レベルが SERIALIZABLE になっているトランザクションでは、インメモリー表を使用できません。インメモリー表にサポートされているトランザクション分離レベルは、REPEATABLE READ と READ COMMITTED です。

HotStandby 2 次サーバーでは、トランザクション分離は常に READ COMMITTED です。

HotStandby を使用しており、HotStandby 2 次サーバーに接続した場合は、REPEATABLE READ を指定した場合であっても、インメモリー表からデータを読み取ったときのトランザクション分離レベルは自動的に READ COMMITTED に設

定されます。(インメモリー表は、1 次サーバーでも 2 次サーバーでも SERIALIZABLE をサポートしていません。)

その他のインメモリー・エンジンの機能拡張

インメモリー表に加えて、solidDB メイン・メモリー・エンジンには、パフォーマンスを向上する特徴が 2 つあります。第一に、読み取り操作は、チェックポイント処理、トランザクションのロギングなどのアクティビティーをシステムが実行しているときでも、ディスク・アクセスを待機しません。次に、リラックス・トランザクション・ロギングを使用すると、書き込み操作はディスク・アクセスを待機しなくなります。リラックス・トランザクション・ロギングとストリクト・トランザクション・ロギングについて詳しくは、「*IBM solidDB 管理者ガイド*」を参照してください。

solidDB インメモリー・エンジンでの共有メモリー・アクセス、リンク・ライブラリー・アクセス、および HotStandby の使用

共有メモリー・アクセスとリンク・ライブラリー・アクセス

共有メモリー・アクセス (SMA) とリンク・ライブラリー・アクセス (LLA) は、リンク可能なライブラリーの形式で solidDB サーバーを提供します。ユーザーは SMA または LLA ライブラリーにアプリケーションを直接リンクして、ネットワーク通信プロトコルを通さずに関数呼び出しでライブラリーにアクセスできます。

SMA と LLA は、solidDB メイン・メモリー・エンジンのインメモリー表機能と互換性があります。インメモリー表を SMA または LLA サーバー内に作成することもできます。

SMA と LLA の詳細については、「*IBM solidDB 共有メモリー・アクセスおよびリンク・ライブラリー・アクセス・ユーザー・ガイド*」を参照してください。

HotStandby

solidDB 高可用性コンポーネントは、「ホット・スタンバイ」機能を提供します。このことは、データベース・サーバーと 2 次サーバーがペアになり、この 2 つのサーバー上のデータが自動的に同期されること意味しています。一方のサーバーに障害が発生しても、引き続きもう一方を使用できます。

solidDB メイン・メモリー・エンジンのインメモリー機能は、solidDB HotStandby と互換性があります。

パーシスタント・インメモリー表 (つまり、明確に TEMPORARY または TRANSIENT として指定されていないインメモリー表) は、HotStandby 1 次サーバーから 2 次サーバーに複製されます。

テンポラリー表とトランジエント表は、2 次サーバーに複製されないことに注意してください。

旧バージョンの solidDB 製品と互換性のない事項

solidDB メイン・メモリー・エンジンは、旧バージョンの solidDB 製品とほぼ 100% の互換性があります。例外については、リリース・ノートに説明があります。

2 テンポラリー表とトランジエント表

テンポラリー表とトランジエント表は、標準のインメモリー表よりもパフォーマンスが高くなっています。テンポラリー表とトランジエント表のデータは永続的ではありません。したがって、データを永続的に保管する場合には、テンポラリー表またはトランジエント表から別の表にそのデータをコピーする必要があります。

インメモリー表を作成すると、その表はデフォルトでは、「パーシスタント」つまり永続的になります。表は、サーバーのシャットダウン時にディスクに書き込まれ、サーバーの再始動時にディスクから読み戻されます。しかし、solidDB メイン・メモリー・エンジンには、テンポラリー表とトランジエント表という、2 タイプの永続的でないインメモリー表が用意されています。このタイプの表は両方とも、「一時的な」データに使用することが目的です。

テンポラリー表とトランジエント表のパフォーマンスがより高い理由は以下のとおりです。

- テンポラリー表とトランジエント表のデータは、メモリーのみに格納され、ディスクには書き込まれません。(サーバーをシャットダウンし再始動した場合、またはサーバーが異常終了した場合に、データは消失します。テンポラリー表の場合、データはユーザー・セッションの終了時に廃棄されます。つまり、サーバーのシャットダウンまで維持されることがありません。)
- テンポラリー表とトランジエント表は、トランザクション・データをディスクに記録しません。(サーバーが異常終了すると、データはリカバリー不能になります。)
- サーバーの定期的な「チェックポイント」処理の実行により、データベースのデータはディスク・ドライブに書き込まれますが、テンポラリー表とトランジエント表のデータはディスクに書き込まれません。(チェックポイントの詳細な説明については、「*IBM solidDB 管理者ガイド*」を参照してください。)
- テンポラリー表とトランジエント表は、solidDB のインメモリー表にハイパフォーマンス・テクノロジーを使用するだけでなく、通常のインメモリー表が使用するよりも効率的なデータ・ストレージ構造も使用します。
- テンポラリー表には、この他にもパフォーマンス上の利点があります。セッション (接続) 間でテンポラリー表内の相互のレコードを参照することがないため、テンポラリー表には高度な並行性制御は必要ありません (例えば、表内のレコードにロックの競合がないか検査する必要がないなどです)。

テンポラリー表とトランジエント表は、「スクラッチパッド」として使用すると特に便利です。例えば、パーシスタント表からデータをコピーし、そのデータがテンポラリー表にある間に、データに対して一連の操作を集中的に行ってから、結果をパーシスタント表に戻して格納することができます。こうすれば、パフォーマンスを最大化しながら、処理の完了後のデータの一部またはすべてを残すことができます。何らかの理由により、処理が中断されても、元のデータはパーシスタント表に安全に保管されているので、処理を再開できます。

テンポラリー表とトランジエント表の主な違いは、テンポラリー表のデータは単一の接続にのみ可視であるのに対して、トランジエント表のデータはすべてのユーザーに可視であることです。

以下では、これらの 2 つのタイプの表のそれぞれについて具体的に説明します。テンポラリー表とトランジエント表には共通点がいくつもあるため、以下の 2 つのセクションでは部分的に重複している個所があります。2 つのタイプの表についてそれぞれ完全に説明した上で、両者の違いを明確にします。 24 ページの『テンポラリー表とトランジエント表のパフォーマンス・チューニング情報』で、その違いについて詳しく説明します。

テンポラリー表

テンポラリー表のデータの可視性と存続期間は、強い制限を受けます。

可視性の制限

データを挿入したセッション (接続) でのみ、そのデータを参照できるため、データの可視性には制限があります。セッションでテンポラリー表を作成し、その表にデータを挿入する場合、表に対する特権を付与したとしても、他のユーザーのセッションからは、そのデータを参照できません。複数のセッションが同じ表を同時に使用する可能性がありますが、それぞれのセッションには、そのセッション独自のデータしか見えないことに注意してください。(各セッションに見えるのは独自のデータのみであるため、表に固有の制約があったとしても、固有の値を表に確実に挿入するために他のセッションと調整する必要がないことに注意してください。例えば、ID 列にユニーク制約があるテンポラリー表を作成した場合、別々のセッション双方で、ID を値 1 に設定したレコードを挿入することが可能です。) 各セッションからは、そのセクション独自のデータしか参照しないため、UPDATE、DELETE などの操作は、そのセッション独自のデータのみに影響します。

存続期間の制限

現行セッションを終了する (つまり、サーバーの接続を切断する) とすぐにデータは廃棄されるため、データの存続期間には制限があります。再度、接続したときに、データはもう存在しません。

理解すべき重要な点は、「テンポラリー表」という名前の「テンポラリー」という言葉が、表自体を指すのではなく、データを指すということです。サーバーは、テンポラリー表の定義 (データではなく) をサーバーのシステム表に実際に保管するため、切断した後にも定義は残ります。したがって、後でサーバーに再接続したときに、表自体は存在しているが、内容は空になっています。つまり、いったん表を作成したら、それ以降のセッションで同じ表を再作成する必要はありません。実際、既存のテンポラリー表と同じ名前のテンポラリー表をいずれかのユーザーが作成しようとすると、エラー・メッセージを受け取ります。「テンポラリー表」の意味を切断と同時に (単にデータではなく) 表そのものが消えると考えているのであれば、これは予期しない動作かもしれません。

もちろん、データはなくても表は存続するため、その表が不要になったら、DROP TABLE コマンドを使用して表定義をドロップする必要があります。

表は存続するので、データベース・スキーマ定義をエクスポートした場合、テンポラリー表を再作成するためのコマンドが出力に含まれます。

ユーザーがセッションを切断すると、そのセッションのテンポラリー表の内容が消去されるので、セッションでテンポラリー表のデータを大量に処理していた場合には、セッション切断後しばらくの間は、サーバーの CPU 使用率が高い場合があります。

追加制限

以下に、テンポラリー表の追加制限をいくつか示します。

- HotStandby コンポーネントの使用時に、テンポラリー表のデータは、2 次サーバーに複製されません。テンポラリー表の定義自体は、HotStandby 2 次サーバーに複製されることに注意してください。したがって、2 次サーバーへのフェイルオーバーが必要な場合、既に作成したテンポラリー表を再作成する必要はありません。ただし、その中にデータを再作成する必要はあります。
- テンポラリー表は、拡張レプリケーション・システムで「マスター」表としては使用できません。(ただし、拡張レプリケーション・システム内でレプリカ表としては使用できます。)
- テンポラリー表には、参照制約があることによる使用方法に関する制限事項があります。テンポラリー表は、別のテンポラリー表を参照できますが、他のタイプ (つまり、トランジエントまたはパーシスタント) の表を参照することはできません。他のタイプの表は、テンポラリー表を参照できません。この章の参照制約の表を参照してください。

このセクションにリストした制限を除いて、テンポラリー表は、通常の (パーシスタント) インメモリー表のように動作します。例えば、

- テンポラリー表に索引を付けることができます。
- テンポラリー表はビューで使用できます。
- テンポラリー表にトリガーを作成することができます。
- テンポラリー表に BLOB 列を格納できます (ただし、それらの列の長さは、数キロバイトに制限されています)。
- テンポラリー表は、特定のカタログ内およびスキーマ内に存在します。
- テンポラリー表には、特権が適用されます。言い換えると、テンポラリー表の作成者は、その表に特権を付与すること、および特権を取り消すことができます。また、DBA も、表に対する特権の付与と取り消しを行うことができます。ただし、あるセッションでテンポラリー表にデータを入れた場合、そのデータを他のセッションから見ることはできません。たとえ DBA によるセッションや、そのテンポラリー表に対する SELECT 特権を付与されたユーザーのセッションであっても、それは不可能です。したがって、表に対する特権を付与することは、単にその表を使用する権利を他のユーザーに付与することであり、データを使用する権利を付与することではありません。テンポラリー表に対するデフォルトの特権は、パーシスタント表に対するデフォルトの特権と同じであることに注意してください。

テンポラリー表を作成するには、以下に示す構文を使用します。ここで、「<...>」は、他のいずれの表タイプの場合も同じ構文であることを表しています。

```
CREATE [GLOBAL] TEMPORARY TABLE <...>;
```

CREATE TABLE コマンドの完全な構文については、「*IBM solidDB SQL ガイド*」を参照してください。

テンポラリー表は、常にインメモリー表です。 STORE DISK 節を使用すると、サーバーはエラーを返します。STORE MEMORY を使用するか、または STORE 節全体を省略した場合、サーバーはテンポラリー表をインメモリー表として作成します。

solidDB のテンポラリー表の実装は、「グローバル・テンポラリー表」に対する ANSI SQL:1999 規格に完全に準拠しています。 solidDB のすべてのテンポラリー表は、キーワード GLOBAL が指定されているかどうかに関係なくグローバル表です。solidDB は、ANSI の定義にあるように、「ローカル・テンポラリー表」をサポートしません。

トランジエント表

トランジエント表のデータは存続期間が制限されています。サーバーがデータを維持する期間は、シャットダウンするまでの間だけです。

トランジエント表のデータの「スコープ」(可視性) は、パーシスタント表のデータと同じです。トランジエント表に挿入されたデータは、該当する特権があれば、他のユーザーのセッションからも見ることができます。

トランジエント表には以下のいくつかの制限があります。

- トランジエント表は、HotStandby コンポーネントを使用している場合に、2 次サーバーには複製されません。トランジエント表自体 (トランジエント表のデータではなく) は HotStandby 2 次サーバーに複製されることに注意してください。したがって、2 次サーバーにフェイルオーバーする必要がある場合は、既に作成したトランジエント表を再作成する必要はありません。ただし、その中にデータを再作成する必要はあります。
- トランジエント表には、参照制約でどのように使用できるかに関する制限事項があります。トランジエント表は、他のトランジエント表もパーシスタント表も参照することができます。テンポラリー表を参照することはできません。テンポラリー表もパーシスタント表もトランジエント表を参照することはできません。この章の参照制約の表を参照してください。
- トランジエント表は、拡張レプリケーション・システムで「マスター」表としては使用できません。(ただし、拡張レプリケーション・システム内でレプリカ表としては使用できます。)

このセクションにリストした制限を除いて、トランジエント表は、「通常の」(パーシスタント) インメモリー表のように動作します。以下に例を示します。

- トランジエント表はビューで使用できます。
- トランジエント表に索引を付けることができます。
- トランジエント表にトリガーを作成することができます。
- トランジエント表に BLOB 列を格納できます (ただし、それらの列の長さはインメモリー表の場合と同じように、数キロバイトに制限されています)。

- トランジエント表には、特権が適用されます。
- トランジエント表は、特定のカタログ内とスキーマ内に存在します。

トランジエント表が入っているデータベースをエクスポートすると、トランジエント表のデータもエクスポートされます (表の構造もそのままエクスポートされます)。

トランジエント表を作成するには、以下に示す構文を使用します。ここで、「<...>」は、他のいずれの表タイプの場合も同じ構文であることを表しています。

```
CREATE TRANSIENT TABLE <...>;
```

CREATE TABLE コマンドの完全な構文については、「*IBM solidDB SQL ガイド*」を参照してください。

トランジエント表は、常にインメモリー表です。STORE DISK 節を使用すると、サーバーはエラーを返します。STORE MEMORY を使用するか、または STORE 節全体を省略した場合、サーバーはトランジエント表をインメモリー表として作成します。

サーバーは実際にはトランジエント表の定義 (データではなく) をサーバーのシステム表に格納し、サーバーがシャットダウンした後も、その定義を残すことに注意してください。サーバーを後で再始動すると、表はそこに残っていますが、データは残っていません。つまり、いったん表を作成したら、再作成する必要はありません。実際、いずれかのユーザーが既存のトランジエント表と同じ名前のトランジエント表を作成しようとすると、エラー・メッセージを受け取ります。これは、その名前の表を最初に作成した後に、サーバーをシャットダウンして再始動した場合であっても、結果は同じです。「トランジエント表」の意味をサーバーの切断と同時に消える表であると考えているのであれば、これは予期しない動作かもしれません。

もちろん、データはなくてもトランジエント表は存続するため、その表が不要になったら、DROP TABLE コマンドを使用して表定義をドロップすることができます。

solload ユーティリティーを使用して、トランジエント表にデータをインポートすることができます。

トランジエント表は SQL 対応の ANSI 規格には規定されていません。トランジエント表は、SQL 規格に対する solidDB の拡張機能です。

テンポラリー表とトランジエント表の相違点

トランジエント表とテンポラリー表の主な違いは、以下のとおりです。

- トランジエント表では、システム内のすべてのセッション (接続) で同じデータを表示できます。テンポラリー表では、データを作成したユーザーのみが、そのデータを表示できます。

  複数のユーザーが同じデータにアクセスする可能性があるので、トランジエント表では、並行性制御が使用されます。現在、トランジエント表は、ペシミスティック並行性制御 (「ロック方式」) のみサポートしています。

テンポラリー表は並行性制御を使用しないので、トランジエント表よりも高速です。

- トランジエント表内のデータは、サーバーがシャットダウンするまで存続しますが、テンポラリー表内のデータが存続するのは、ユーザーがセッションからログアウトするまでだけです。このことは、あるセッションでトランジエント表に挿入したデータは、そのデータの作成者が切断した後も、他のセッションで表示できることを意味しています。
- トランジエント表のデータは、solexp ツールを使用してエクスポートできますが、テンポラリー表のデータはできません。
- 2 つの表タイプに対する参照整合性規則は異なります。

  以下の表に、どの表タイプが他のタイプを参照できるかを示します。例えば、トランジエント表が、パーシスタント表を参照する外部キーを持つことができる場合、「トランジエント子」の行と「パーシスタント親」の列の交点のセルに「YES」とあります。外部キー制約が許可されない場合には、ダッシュ (-) があります。

表 3. 参照制約

参照される側の表 **参照する側の表**	**パーシスタント・ディスク・ベース表**	**パーシスタント・インメモリー表**	**トランジエント表**	**テンポラリー表**
パーシスタント ディスク・ベース表	YES	YES	-	-
パーシスタント インメモリー表	YES	YES	-	-
トランジエント 表	YES	YES	YES	-
テンポラリー 表	-	-	-	YES

どのタイプの表も、自分自身を参照できます。加えて、トランジエント表は、パーシスタント表を参照できます (逆の参照はできません)。その他のすべての組み合わせは無効です。

3 サーバーの最適化とチューニング

この章では、solidDB メイン・メモリー・エンジンに備わっている機能を使用して、サーバーの最適化とチューニングを行う方法について詳しく説明します。

メモリー内に格納する表を選択するアルゴリズム

いくつかの表を格納するのに十分なメモリーが存在するが、メモリー内にすべての表を格納することはできない場合のために、以下に、メモリー内に格納する表を選択する戦略を示します。

その原則は、アクセスの「密度」を考慮することです。明らかに、アクセス頻度が高いと、アクセス「密度」も高くなります。同様に、1 秒間のアクセス回数が一定であれば、表が大きいほどアクセス「密度」は低くなります。

アクセス密度は、1 秒間の 1 メガバイト当たりのアクセス単位で測定し、行/MB/秒と記述します。(簡単にするために、行ごとに 1 アクセスとします。) 例えば、1 メガバイトの表があり、10 秒間で 300 行にアクセスする場合、密度は以下のようになります。

30 行/MB/秒 = 300 行 / 1MB / 10 秒

2 番目の例として、500 KB の表が存在し、1 秒間で 300 行にアクセスする場合を考えます。アクセス密度は、以下のようになります。

600 行/MB/秒 = 300 行 / 0.5 MB / 秒

明らかに、最初の表より 2 番目の表の方がアクセス密度が高く、格納できるのがこれらの表のどちらか 1 つのみの場合には、2 番目の表をメモリーに格納すべきです。

この公式は、いくらか簡略化されています。

1. 毎回のアクセスでアクセスされるバイト数を考慮した方がいい場合があります。一般に、これは平均行サイズですが、バイナリー・ラージ・オブジェクト (BLOB) を使用する場合、または表全体ではなく索引のみを読み取ることで必要なすべての情報をサーバーが検出できる場合には平均行サイズにならない可能性があります。

   サーバーは、通常「ブロック」(1 ブロックは一般に 8 KB) の倍数でディスクからデータを読み取るため、アクセスごとのバイト数または行ごとのバイト数は、これらを考慮しない公式に比べ若干正確な数値にすぎないことに注意してください。読み取るのが 10 バイトの行でも、また 2000 バイトの行でも、サーバーは、ほぼ同じ量の作業を行います。
2. 表のサイズを考慮する場合は、その表に対する索引のサイズも考慮しなければならないことに注意してください。索引を追加するたびに、その表に関して、さらに多くのデータを追加して格納することになります。その上、表に外部キー制約を追加すると、サーバーは、適切な索引 (まだ存在しない場合) を作成して、そ

の表に対する特定のタイプの参照操作を高速化します。メモリー内の表のサイズを計算する場合は、その表、その表のすべての索引、およびその表のすべての BLOB を考慮する必要があります。

すべての表のアクセス密度を計算した後で、それらの表に最上位から最下位まで順序を付けます。次に、使用可能な物理メモリーをすべて使い切るまで、最高密度の表から始めて、リストの下位の表へ順に、表をインメモリー表として指定していきます。

残念ながら、このプロセスは、このように単純ではありません。その理由は、この説明では、完全に情報が揃っていると想定し、かつ任意の時点でディスク・ベースからインメモリー (またはその逆) に表を変更できると想定しているためです。実際には、コンピューター内の空きメモリーの合計容量が分からない場合があります。誤って、コンピューターの物理メモリーにある空き以上に多くのインメモリー表を指定することがあります。その結果として、表がディスクにスワップされる場合があります。このため、パフォーマンスが大幅に低下することがあります。また、表に相当の量のデータを入れるまで、その表の実際のアクセス頻度が分からない場合があります。しかし、現行バージョンの solidDB サーバーでは、表にデータを入れる前の表の作成時にインメモリー表かディスク・ベース表かを指定する必要があります。そのため、各表の使用量の推定値、各表のサイズの推定値、および空きメモリー容量の推定値に基づいて計算を行う必要があります。さらに、平均アクセス密度が時間とともに変化しないと想定しています。

また、このアプローチでは、今後さらに表を追加する計画はないものと想定し、表のサイズは大きくならないと想定しています。一般に、利用できるすべてのメモリーを使い果たしてはいけません。表のサイズが大きくなる見込みを考慮して十分なスペースを残し、またエラーの場合の余裕を多少残して、メモリー不足にならないようにする必要があります。

それでも、このアルゴリズムは、メモリー内に格納する表を選択するための基本的なガイドとなります。

重要: 仮想メモリーは頻繁にディスクにスワップされる可能性があるため、仮想メモリーを使用すると、インメモリー表の利点が相殺されます。常に、DBMS プロセス全体がコンピューターの物理メモリーに確実に収まるようにしてください。

テンポラリー表とトランジエント表のパフォーマンス・チューニング情報

注意:
テンポラリー表とトランジエント表のいずれを使用するかを決める際に、テンポラリー表とトランジエント表のデータが一時的なデータであり、永続的なデータではないことに留意してください。データはディスクには保管されません。HotStandby を使用している場合、データは HotStandby 2 次サーバーにはコピーされません。データはトランザクション・ログにも書き込まれないため、サーバーが異常終了した場合には、リカバリーできません。データを失い、作業を最初からやり直すわけにいかない場合には、テンポラリー表とトランジエント表を使用しないでください。

テンポラリー表とトランジエント表のいずれにするのかを決める場合、念頭におく主要項目は以下のとおりです。

- テンポラリー表のデータはセッションが終了すると消失しますが、トランジエント表のデータはサーバーがシャットダウンするまで存続します。
- テンポラリー表のデータは、他のセッション/接続から見ることはできません。
- テンポラリー表は、並行性の競合検査をほとんど行わないため、トランジエント表よりも高速です。

データを「共有」する必要がなく、また現行セッションから切断した後にデータが不要であれば、テンポラリー表を選択してください。テンポラリー表の方がオーバーヘッドが少なく、パフォーマンスが高いからです。

索引

表がメモリー内に格納される場合、その表の索引もすべてメモリーに格納されます。当然、パフォーマンスはアップしますが、同時にメモリー・スペースの消費も増えます。

一般に、インメモリー索引によって飛躍的な高速化が可能なので、表のデータへのアクセス速度を確実に速くするために、インメモリー索引を使用することを推奨します。

表と索引をすべて格納できる十分なメモリーがない場合は、特定の索引を追加することが最善策ではないこともあります。一部の照会の速度が速くなったとしても、その照会が、別の表の格納に使用されるはずのメモリーを使用することで、その他の照会の速度は遅くなるからです。

4 インメモリー・データベースの構成

この章では、ご使用の環境、パフォーマンス、および操作要件に合わせて、solidDB インメモリー・データベースを構成する方法を説明します。この説明は、「*IBM solidDB 管理者ガイド*」の『*solidDB の構成*』セクションの一部です。

solidDB メイン・メモリー・エンジンに関連する使用可能なすべてのパラメーター・リストの概要については、43 ページの『付録 C. 構成パラメーター』を参照してください。

構成ファイルとパラメーター設定

solidDB は、ほとんどの構成情報を `solid.ini` ファイルから取得します。より具体的には、2 つの異なる `solid.ini` 構成ファイルが、1 つはサーバー上、もう 1 つはクライアント上に存在します。どちらの構成ファイルも、必須ではありません。構成ファイルが存在しない場合は、ファクトリー値が使用されます。`solid.ini` 構成ファイルには、クライアント用とサーバー用の構成パラメーターがそれぞれ格納されています。クライアント・サイド構成ファイルが使用されるのは、ODBC ドライバーが使用され、構成ファイルがアプリケーションの作業ディレクトリーに存在しなければならない場合です。

注: solidDB の資料では、`solid.ini` ファイルについて言及するときは、通常、サーバー・サイドの `solid.ini` ファイルを指します。

solidDB は、始動時に、まず SOLIDDIR 環境変数で設定されているディレクトリーから `solid.ini` ファイルを開こうとします。この変数で指定されたパスからこのファイルが見つからない場合、またはこの変数が設定されていない場合には、サーバーまたはクライアントは、現行作業ディレクトリーでこのファイルを開こうと試みます。現行作業ディレクトリーは、通常、solidDB サーバーを始動、またはクライアント・アプリケーションを開始したディレクトリーと同じです。 -c サーバー・コマンド行オプションを使用して、別の作業ディレクトリーを指定することもできます。コマンド行オプションについて詳しくは、「*IBM solidDB 管理者ガイド*」の『*付録 B solidDB コマンド行オプション*』を参照してください。

構成ファイルには、solidDB パラメーター用の設定が入っています。特定のパラメーターの値が `solid.ini` ファイルに設定されていない場合、solidDB はそのパラメーターのファクトリー値を使用します。ファクトリー値は、使用するオペレーティング・システムによって異なることがあります。

一般に、ファクトリー値のままでパフォーマンスにも操作容易性にも問題はありませんが、状況によっては、一部のパラメーター値を変更することで、パフォーマンスが向上することもあります。

`solid.ini` ファイルにパラメーターの名前と値のペアを設定することで、構成を変更することができます。例えば、サーバーのネットワーク・アドレスを指定するには、パラメーター名 Listen および適切な値を使用して、以下のように指定します。

```
Listen=tcp 192.168.255.1 1315
```

これは、サーバーがクライアント要求を listen する際に、TCP/IP プロトコル、ネットワーク・アドレス 192.168.255.1、およびポート番号 1315 を使用して listen する必要があることを指定します。

パラメーターは、構成ファイルのセクション・カテゴリーに従ってグループ化されています。セクション・カテゴリーおよび使用可能なすべてのパラメーターの概要については、「*IBM solidDB 管理者ガイド*」の『*付録 A サーバー・サイド構成パラメーター*』および『*付録 B クライアント・サイド構成パラメーター*』を参照してください。

各セクション・カテゴリーは、以下の例のように、大括弧で囲まれたセクション名で始まります。

```
[com]
```

[com] セクションには、通信情報がリストされます。セクション名は、大/小文字を区別しないことに注意してください。セクション名 [COM]、[Com]、および [com] はすべて同じものです。

サーバー・サイド `solid.ini` 構成ファイルのセクションの例を以下に示します。

```
[IndexFile]
FileSpec_1=C:¥soldb¥solid1.db 1000M
CacheSize=64M
```

サーバー・サイド・パラメーターの管理

solidDB パラメーターとその値の表示と変更を行う方法としては、以下の方法があります。

- 以下のコマンドを入力します。

  ADMIN COMMAND 'parameter'

  および

  ADMIN COMMAND 'describe parameter'

  solidDB SQL エディター (テレタイプ) で入力してください。
- solidDB ディレクトリーにある `solid.ini` ファイルを直接、編集する。

以下のセクションでは、ADMIN COMMAND と `solid.ini` を使用して、パラメーターを管理する手順について説明します。

注: サーバーの通信プロトコル・パラメーターの表示および設定について詳しくは、「*IBM solidDB 管理者ガイド*」の『*ネットワーク接続の管理*』のセクションを参照してください。

ADMIN COMMAND によるパラメーターの表示および設定

ADMIN COMMAND を使用すれば、solidDB サーバーを介してサーバーを再始動することなくパラメーターをリモートで変更することができます。すべてのパラメーターが、たとえ `solid.ini` 構成ファイルになくても、アクセス可能です。パラメーターが存在しなければ、ファクトリー値が使用されます。

パラメーターの表示

以下のコマンドを使用して、1 つのパラメーターまたは数多くのパラメーターのサマリーを表示できます。

```
ADMIN COMMAND 'parameter [-r] [section_name[.parameter_name]]';
```

上記の要素について以下に説明します。

- -r オプションは、現行値のみが必要であることを指定します。
- *section_name* は、solid.ini 内でパラメーターが置かれているカテゴリーの名前です。

パラメーターをすべて表示するには、solidDB SQL エディター (テレタイプ) で以下のコマンドを入力します。

ADMIN COMMAND 'parameter';

現行値、*開始値*、および*ファクトリー値* を示した全パラメーターのリストが返されます。以下のようにセクション名を追加することにより、表示パラメーターを特定のセクションに限定することができます。

```
ADMIN COMMAND 'parameter logging';
```

以下に示すように、あるパラメーターの名前を完全に指定すると、その単一パラメーターに関連する値をすべて表示することができます。

```
admin command 'parameter logging.durabilitylevel';
    RC TEXT
    -- ----
    0  Logging DurabilityLevel 3 2 2
1 rows fetched.
```

3 つの値が表示されます (以下の順序で表示される)。

- *現行値*
- サーバー始動時に使用された*開始値*
- 製品に事前設定されている*ファクトリー値*

必要に応じて、このコマンドを -r オプションで修飾することにより、現行値のみを表示することも可能です。以下に例を示します。

ADMIN COMMAND 'parameter -r';

特定のパラメーター説明の表示

特定のパラメーターに関するより詳細な説明を表示することもできます。この説明の中には、有効なパラメーター・タイプおよびアクセス・モードが含まれます。特にパラメーターは動的な処理が必要になることがあるため、この情報は役立ちます。パラメーターのサポートは、製品、プラットフォーム、またはリリースによって異なる場合があります。

パラメーターの説明を表示するには、solidDB SQL エディター (テレタイプ) で以下のコマンドを入力します。

```
ADMIN COMMAND 'describe parameter [section_name[.parameter_name]] ';
```

単一パラメーターの結果セットが以下のように表示されます。

```
admin command 'describe parameter logging.durabilitylevel';
    RC TEXT
    -- ----
    0 DurabilityLevel
    0 Default transaction durability level
    0 LONG
    0 RW
    0 2
    0 3
    0 2
7 rows fetched.
```

結果セットの行内容は以下のとおりです。

- *パラメーター名*。これは、CacheSize のようなパラメーターの名前です。
- *パラメーターの説明*。
- *データ型*
- アクセス・モード。以下のうちの 1 つです。
  - RO: 読み取り専用。値は動的には変更できません
  - RW: 読み取り/書き込み。値は動的に変更でき、変更後すぐに有効になります。
  - RW/STARTUP: 値は動的に変更できますが、次回のサーバー始動時まで変更内容は反映されません。
  - RW/CREATE: 値を動的に変更できますが、変更は新しいデータベースの作成時に有効になります。
- *開始値* には、パラメーターの開始値を表示します。
- *現行値* には、パラメーターの現行値を表示します。
- *ファクトリー値* は製品に事前設定されている値を表示します。

パラメーター値の設定

特定のパラメーターに値を設定するには、solidDB SQL エディター (teletype) を使用して以下のコマンドを入力します。

```
ADMIN COMMAND 'parameter section_name.parameter_name=value [temporary]';
```

上記の要素について以下に説明します。

value は、有効なパラメーター値です。

注:

値を指定しないと、パラメーターにファクトリー値 (または設定解除値) が設定されます。また、アスタリスク (*) の付いたパラメーター値を割り当てた場合は、そのパラメーターにはファクトリー値が設定されます。

temporary を設定すると、変更値は solid.ini ファイルに格納されません。

オプションで、等号の前後にブランクを使用できることに注意してください。

以下に例を示します。

```
-- 通信トレースをオンに設定
ADMIN COMMAND 'parameter com.trace = yes';
```

注:

パラメーター管理の操作はトランザクションの一部ではないため、ロールバックすることはできません。

このコマンドは、結果セットとして新しい値を返します。パラメーターのアクセス・モードが RO (読み取り専用) か、または入力した値が無効な場合、ADMIN COMMAND ステートメントはエラーを返します。

パラメーター変更の永続性

アクセス・モードが RW* のパラメーターに対する変更は、次のチェックポイントにおいて solid.ini ファイルにすべて格納されます。これは、temporary オプションで設定されている値には適用されません。

以下のコマンドを使用すれば、変更した値をすぐに格納するように要求することもできます。

```
ADMIN COMMAND 'save parameters [ini_file_name]';
```

ini_file_name が指定されていない場合、現行の solid.ini ファイルが再書き込みされます。そうでない場合は、構成ファイル全体が新しい場所に書き込まれます。これは、後で使用できるように、構成ファイルのチェックポイントを保存する便利な方法です。

solid.ini のパラメーターの表示および設定

1. solidDB プロセスの作業ディレクトリーにある solid.ini ファイルを開きます。
2. パラメーターの値を表示します。

   表示されるパラメーターは、サーバーで現在アクティブなパラメーターです。パラメーター値を設定していない場合は、始動時にファクトリー値が使用されます。ファクトリー値は、solidDB が稼働しているオペレーティング・システムにより異なる場合があります。
3. 必要であれば、セクション、パラメーター、およびパラメーター値を追加します。
4. 変更を保存します。

変更をアクティブにするには、サーバーを再始動する必要があります。

パラメーターの定数値

構成ファイルの IndexFile セクションの中にある Blocksize パラメーターのアクセス・モードは RO です。このパラメーターはデータベースの作成時に設定され、それ以降は変更できません。

別の定数値を使用する場合には、データベースを新規作成する必要があります。データベースを新規作成する前に、solidDB ディレクトリーの solid.ini ファイルを編集して、パラメーターの新しい定数値を設定します。

以下の例では、solid.ini ファイルに以下の行を追加して、索引ファイルに新しいブロック・サイズを設定しています。

```
[IndexFile]
Blocksize = 4096
```

solid.ini ファイルの編集と保存が終了したら、古いデータベース・ファイルとログ・ファイルを移動または削除してから、solidDB を始動します。

注:

ログのブロック・サイズは、次にサーバーを始動するまでの間に変更できます。

付録 A. 最大 BLOB サイズの計算

目的

インメモリー表とディスク・ベース表の重要な違いの 1 つは、インメモリー表の列の値が単一「ページ」(solid.ini 構成ファイルに指定されているページ・サイズ、最大サイズは 32 KB) に収まる必要があるということです。したがって、インメモリー表は、ページ・サイズよりも大きい文字ファイルやバイナリー・ファイルを格納することができません。しかし、小さめのバイナリー・ファイルはサポートされます。

この付録では、インメモリー表に収まるように、文字列またはバイナリー列の値の最大サイズを計算する方法について示します。

バックグラウンド

今日、INT や CHAR などの標準的なデータ型で簡単には格納できないデータを使用するアプリケーションが数多くあります。代わりに、long 文字やバイナリー・フォーマットがより適している場合があります。そのようなときのデータは、*CLOB* (*文字ラージ・オブジェクト*) および *BLOB* (*バイナリー・ラージ・オブジェクト*) で格納できます。 CLOB は、最大 20 億文字の解釈可能文字を格納します。BLOB データ型には、一連のバイナリー数 (8 ビットのバイト群) として、ほぼすべてのデータを格納できます。一般に、BLOB は、簡単に数値または文字として解釈できない大きな可変長データの格納に使用されます。例えば、BLOB はデジタル化された音 (コンパクト・ディスク上の音楽など)、マルチメディア・ファイル、またはセンサーから読み取った時系列データを保持できます。

solidDB では、BLOB は幅広くサポートされており、BINARY、VARBINARY および LONG VARBINARY など、さまざまなデータ型から選択します。その中で最後の LONG VARBINARY は標準データ型 BLOB にマップされます。

CLOB は、6 つのデータ型、CHAR、WCHAR、VARCHAR、WVARCHAR、LONG VARCHAR、および LONG WVARCHAR で実装されています。最後の 2 つのデータ型は、標準データ型 CLOB および NCLOB にマップされます。データ型 CLOB と BLOB について詳しくは、「*IBM solidDB SQL ガイド*」の*付録 A* の『*文字データ型*』および『*バイナリー・データ型*』の各セクションを参照してください。

ディスク・ベース表に関しては、solidDB で BLOB ストレージが実装されているため、アクセス速度と大量データを格納できることの必要性との間でバランスが取れます。データ型 (VARCHAR、VARBINARY など) に関係なく、一般に、短い値は表に格納されますが、より長い値は、そのデータの一部または全体がデータベース・ストレージ・ツリーの個別領域に格納されます。この点がユーザーに意識されることはまったくありません。ユーザーは、単にデータ型を決定し、solidDB が残りの処理を担当します。ユーザー・データは、データの実際の物理的な位置に関係

なく、常に同じ方法でアクセスされ、また表に格納されるように見えます。ディスク・ベース表では、VARCHAR フィールドと VARBINARY フィールドの最大長は 2 ギガバイトです。

インメモリー表では、BLOB データは、完全に表自体に格納されます。また、BLOB の最大長は、「ブロック・サイズ」で制限されます (インメモリー表の行は、ページ長または「ブロック」長を超えることはできません)。この付録には、インメモリー表に格納できる VARCHAR データまたは VARBINARY データの最大サイズを見積もるのに役立つ情報があります。

計算

BLOB で使用可能なスペースを計算するためのアルゴリズムでは、概算値が計算されることに注意してください。以下の書式をコピーして、ご使用の表に該当する値を記入してください。記入されている手順に従って、BLOB データに使用可能な残りスペースを計算します。

表 4. BLOB データに使用可能なスペースの計算

	値	値に入力する内容	値の意味
1		左のスペースに、ご使用のブロック・サイズまたは 32767 (どちらか小さい方) を入力します。ブロック・サイズは、`solid.ini` 構成ファイルの [IndexFile] の BlockSize に設定した値、または「*IBM solidDB 管理者ガイド*」に記載のデフォルトのどちらかです。	ブロック・サイズ (ページ・サイズ) は、「ブロック」内のバイト数であり、ディスク・ブロックに似ています。各行は 1 つのブロック内に収まる必要があるため、ブロック・サイズは行の最大サイズを表しています。
2	17	左記のハードコード値を使用します。	ページごとのオーバーヘッドのバイト数です。
3	10	左記のハードコード値を使用します。	行ごとのオーバーヘッドのバイト数です。大きな BLOB の場合には、各ページ内に 1 行のみ存在すると想定します。
4		表に対して明示的に主キーを宣言した場合には、値として 10 を入力します。それ以外の場合には、20 を入力します。	サーバーが各表に自動的に追加する列で使用するバイト数を表します。
5		表内の列数を 2 倍した値を入力します。	列に関するオーバーヘッドのバイト数です。
6		表内のデータの固定サイズ列のサイズの合計を入力します。(各固定サイズ・データ型のサイズについては、以下の表 5 を参照してください。)	固定サイズ列が占有するスペースを表します。
7		BLOB 列数を入力します。	BLOB 値の終端に使用するバイト数です (値ごとに 1 バイト)。
8		2 行目から 7 行目までの値を合計します。	BLOB 値以外のすべてで使用される合計スペースです。

表 4. BLOB データに使用可能なスペースの計算 (続き)

	値	値に入力する内容	値の意味
9		8 行目から 1 行目を減算します。	BLOB データ用に使用可能な概算バイト数です。表内に単一の BLOB 列が存在する場合は、その BLOB 値の概算最大サイズです。

注: 最大ブロック・サイズは 64 K です。ただし、最大行サイズ (つまり、最大 BLOB サイズ) は 32 K (実際には 32 K - 1、つまり 32767) にすぎません。ブロック・サイズが 64K または 32K の場合は、表の 1 行目にブロック・サイズの代わりに 32767 を入力してください。

以下の表に、各固定サイズ・データ型の値を格納するのに必要なバイト数を示します。例えば、SQL FLOAT 型の値の格納には 8 バイトが必要です。

表 5. 値の格納に必要なバイト数

データ型	ストレージ・サイズ (バイト単位)
TINYINT	1
SMALLINT	2
INT	4
BIGINT	8
DATE/TIME/TIMESTAMP	11
FLOAT / DOUBLE PRECISION	8
REAL	4
NUMERIC / DECIMAL	11
CHAR / VARCHAR / LONG VARCHAR	char_length(column_value) + 1
WCHAR / WVARCHAR / LONG WVARCHAR	char_length(column_value) * 2 + 1
BINARY / VARBINARY / LONG VARBINARY	octet_length(column_value) + 1

付録 B. ストレージ要件の計算

この付録では、メモリーまたはディスクに表およびその索引を格納するのに必要なメモリーまたはディスク・スペースを見積もるための情報を記載します。

ここに示す公式は、以下を含めて、いくつかの理由で厳密ではないことに注意してください。

- solidDB は、一部のデータを圧縮します。
- 可変長データ (例えば VARCHAR) では、格納される値の実際の長さに応じて、必要なスペース容量は異なります。
- インメモリー・データ構造では、すべてのレコードに対して、必ずしも同じ数の「ポインター」が格納されるわけではありません。

この説明では、ほとんどの場合、データは圧縮されないと想定し、またポインターの最大数を想定しています。したがって、以下の公式を使用して得られる結果は、普通、ある程度の余裕がある値です。つまり、公式では、必要なスペース容量を多めに見積もります。

下の公式では、以下の表記

```
sum_of(x)
```

は、各「x」のサイズの合計を求めることを意味しています。例えば、

```
sum_of(col_size)
```

は、表または索引内の各列のサイズの合計を求めることを意味しており、また

```
sum_of(index_sizes)
```

は、表に対するすべての索引のサイズの合計を求めることを意味しています。

ディスク・ベース表

ディスク・ベース表で必要なスペースに関する一般公式は、以下のようになります。

```
chkpt_factor x (table_size + sum_of(index_sizes))
上記の要素について以下に説明します。
        chkpt_factor は、 1.0 と 3.0 の間 (以下に説明) です。
        table_size =
            1.4 x rows x (sum_of(col_size + 1) + 12)
上記の要素について以下に説明します。
        rows は、行数です。
        sum_of(col_size + 1) は、列のサイズの合計に、
            列ごとに 1 バイトを加算した値です。
列のサイズは、後の表に示してあります。
```

各ディスク・ベース索引について、index_size は以下のようになります。

```
1.4 x rows x (pkey_size + idx_size)
```

ここで、pkey_size は主キー内の列のサイズの合計、および idx_size は索引内の列のサイズの合計です。

chkpt_factor は、「チェックポイント」処理に一時的に最大でデータベース・サイズの 3 倍が必要になる場合があることを考慮するために必要です。チェックポイント処理中は、データベース内の変更された各ページがメモリーからディスクにコピーされます。データベース内のすべてのページが更新された場合、既にディスク上に存在するのと同じだけのページ数をメモリーからコピーする可能性があります。その上、現行のチェックポイント処理が正常に完了するまで、成功した最新のチェックポイントは削除されません。したがって、チェックポイント処理中は、ディスクに各ページのコピーが最大で同時に 3 つ (データベース内のページ用に 1 つ、成功した最新のチェックポイント用に 1 つ、および実行中の現行チェックポイント用に 1 つ) 存在することがあります。したがって、チェックポイント係数は、1.0 と 3.0 の間になります。多くのデータベースでは、3.0 に近い値を指定することはほとんどありません。高水準のアクティビティーを持つ小規模のデータベースの場合でも、一般に値は 1.5 で十分です。チェックポイント処理の頻度が低いほど、chkpt_factor を大きくする必要があることに注意してください。

注: ディスク・ベース索引では、主キーを明示的に定義しないと、サーバーで生成された「行番号」が主キーとして使用されます。これにより、主キー索引は、挿入されたのと同じ順序でレコードを格納することになります。

インメモリー表

インメモリー表の一般公式は、以下のとおりです。

table_size + sum_of(*index_sizes*)

table_size =

1.3 x rows x (sum_of(*col_sizes*) + (3 x *word_size*) + (2 * *num_cols*) + 2)

ここで、rows は行数です。

word_size は、マシンのワード・サイズ (例えば 32 ビット OS では4 バイト、64 ビット OS では 8 バイト) です。

num_cols は、列数です。

sum_of(*col_sizes*) は、列のサイズの合計です。

各インメモリー索引で、索引サイズは以下のようになります。

1.3 x rows x ((*dist_factor* x sum_of(*col_sizes* + 1)) + (8 x *word_size*) + 4)

ここで、「dist_factor」は、1.0 と 2.0 の間の値で、キー値の分布に左右されます。キー値が大きく異なれば、使用する値は 2.0 により近づけてください。キー値がほとんど異ならなければ、使用する値は 1.0 により近づけてください。

列サイズの表

TINYINT: 2 バイト

SMALLINT: 2 バイト

INT: 4 バイト

BIGINT: 8 バイト

DATE/TIME/TIMESTAMP: 11 バイト

FLOAT / DOUBLE PRECISION: 8 バイト

REAL: 4 バイト

NUMERIC / DECIMAL: 12 バイト

CHAR / VARCHAR / LONG VARCHAR: char_length(column_value) + 5

WCHAR / WVARCHAR / LONG WVARCHAR: char_length(column_value) * 2 + 5

BINARY / VARBINARY / LONG VARBINARY: octet_length(column_value) + 5

メモリー使用量の測定

表と索引を作成した後に、以下のコマンドを使用して、メモリーの実際の使用量を測定することができます。

ADMIN COMMAND 'info imdbsize';

このコマンドは、インメモリー表とインメモリー索引のメモリー総使用量を出力します。単位はキロバイトです。

詳細

この章では、異なるストレージ・ツリーにデータを格納する方法についての詳細な情報を記載します。この情報は、前出の公式について、基本をさらに詳しく理解したい場合に役立ちます。

ディスク・ベース表

ディスク・ベース表では、データと索引は B ツリーに格納されます。ツリー内の各エントリーは、ヘッダーとデータ用にスペースを消費します。

実際のデータが使用するスペースは、前述の列サイズの表を使用して計算できます。その表内の値は最大長です。可変長データ (例えば VARCHAR) や圧縮可能なデータでは、必要なバイト数が少なくなる可能性があります。

さらに、ディスク・ベース表では、列ごとにサーバー用に 1 バイト余分に必要です。このバイトは、長さ標識の一部として使用されます (NULL 標識の役割も果たします)。

各行のヘッダーは、12 バイトを使用します。

表 6. ヘッダー・バイト

バイト数	使用目的
3 バイト	行ヘッダー
3 バイト	表 ID
6 バイト	行バージョン

ディスク・ベース表に索引 (主キー以外) が含まれる場合、それらの索引内のエントリーのサイズは、同じガイドラインを使用して別に見積もる必要があります。索引エントリーには、以下の要素が含まれます。

- 索引内に定義された列
- 表の主キーの列
- 行ヘッダー (12 バイト)

加えて、データベース・ページには、通常、いくらか空のスペース (例えば 20% から 40%) が存在します。このため、公式には、表と索引の両方に乗数 1.4 が含まれています。

例: 次のディスク・ベース表がある場合

```
CREATE TABLE subscriber (
             id INTEGER NOT NULL PRIMARY KEY,
             name VARCHAR(50),
             salary FLOAT) ;
```

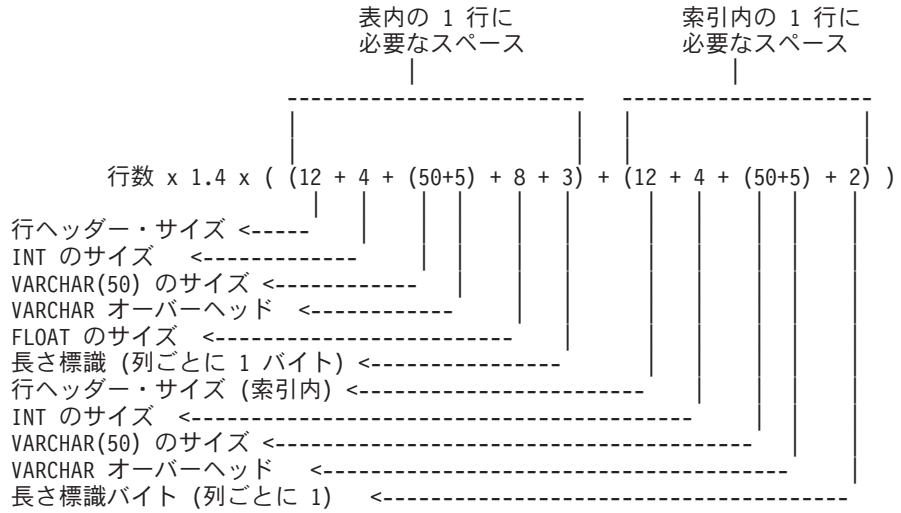
これに、副次索引を作成しました。

```
CREATE INDEX subscriber_idx_name ON subscriber (name);
```

索引エントリーには、NAME 列が含まれます。また、主キー列も含まれており、この場合には ID です。その索引に必要なスペースは、別に見積もる必要があります。「空スペース係数」を 1.4 と想定した、ディスク・ベース表の合計サイズは、次のようになります。

```
行数 x 1.4                    // 1.4 = 空スペースの見積もり。
   x ( (12 + 4 + (50+5) + 8 + 3)   // 表エントリーのサイズ、
       + (12 + 4 + (50+5) + 2) )   // 副次索引エントリーのサイズ
```

これは、別の方法で次のように表現できます。



インメモリー表

インメモリー表で必要なスペースは、別の方法で見積もります。

各エントリーのサイズは、表のデータのサイズと、行ごとのオーバーヘッドの 3 つのメモリー・ポインター (32 ビット OS では各 4 バイト、64 ビット OS では各 8 バイト) を組み合わせたものです。その上、行ごとに 2 バイトのオーバーヘッド、および行の列ごとに 2 バイトのオーバーヘッドが存在します。長さ標識を考慮するために、列ごとに 1 バイトを追加する必要はないことに注意してください。それは、行ごとの 2 バイトに含まれています。

さらに、メイン・メモリー表に索引がある場合は、サーバーの始動時にそこにデータが追加されます。各索引エントリーには、索引で定義された列のデータが含まれています。加えて、各索引エントリーには、最大 8 個のメモリー・ポインターが含まれます。(インメモリー索引には、主キーのコピーが必要ないことに注意してください。)

その他にも、索引の実際のデータ値によって決まるオーバーヘッドが存在します。これは、索引のデータ・サイズのパーセンテージです。この値はキー値の分布に左右されるため、正確な値を示すことはできませんが、乗数は 1.0 と 2.0 の間になります。

また、索引構造自体は、索引エントリーごと (つまり、行ごと) に平均 4 バイトが必要です。

上記の表と索引の例では、32 ビット・オペレーティング・システムのメモリー使用量は、以下のように見積もることができます。

```
行数 * 1.3 x (
  ((3 x 4) + 2 + (4 + 2) + (50+5+2) + (8+2)) // 表内のデータ・サイズ。
+ ((8 x 4) + 4 + 1.2 x 4)        // 主キー索引のサイズ。
+ ((8 x 4) + 4 + 1.2 x (50+5)))  // 副次索引のサイズ。
)
```

```
((3 x 4) + 2 + (4 + 2)+(50+5+2)+(8+2)) // 表内のデータ・サイズ。
  |   |   |   |   |   | | |   | |
  |   |   |   |   |   | | |   | -> 列ごとに 2 バイト
  |   |   |   |   |   | | |   ---> FLOAT (給与) 用に 8 バイト
  |   |   |   |   |   | | -------> 列ごとに 2 バイト
  |   |   |   |   |   | ---------> VARCHAR ごとに 5 バイトのオーバーヘッド
  |   |   |   |   |    ------------> VARCHAR(50) 用に 50 バイト
  |   |   |   |    ----------------> 列ごとに 2 バイト
  |   |   |    --------------------> INT (ID) 用に 4 バイト
  |   |    ------------------------> 行ごとに 2 バイト
  |    ------------------------------> ポインター・サイズ
  |                                     (32 ビット OS 上では 4 バイト)
   ----------------------------------> 3 個のポインター


((8 x 4) + 4 + 1.2 x 4)     // 主キー索引のサイズ。
  |   |   |   |   |
  |   |   |   |    ---------------> INT 用に 4 バイト
  |   |   |    --------------------> 索引値分布係数 1.2
  |   |    ------------------------> 索引エントリーごとに 4 バイト
  |    ------------------------------> ポインター・サイズ
  |                                     (32 ビット OS 上では 4 バイト)
   ----------------------------------> 8 (最大 8 個のポインター)


((8 x 4) + 4 + 1.2 x (50+5))  // 副次索引のサイズ。
  |   |   |   |   | |
  |   |   |   |    ------------> VARCHAR オーバーヘッド用に 4 バイト
  |   |   |   |     -------------> VARCHAR(50) 用に 50 バイト
  |   |   |    --------------------> 索引値分布係数 1.2
  |   |    ------------------------> 索引エントリーごとに 4 バイト
  |    ------------------------------> ポインター・サイズ
  |                                     (32 ビット OS 上では 4 バイト)
   ----------------------------------> 8 (最大 8 個のポインター)
```

64 ビット・オペレーティング・システムでは、メモリー・ポインター・サイズとして、4 バイトではなく 8 バイトを使用します。

上記の見積もりでの係数 1.2 は、「TRIE 索引値分布係数」であり、正確な値は索引列の実際の値によって決まります。その値は、一般に 1 と 2 の間です。ランダム値分布では、この値は 2.0 に近くなり、順次値分布では 1.0 に近くなります。4 バイトは、索引エントリーで平均して必要となるデータ・オーバーヘッドです。

係数 1.3 は、メモリー・アロケーターの内部オーバーヘッドを考慮しています。

注:

メイン・メモリー表の索引は、サーバーが始動するたびに動的に作成されます。ディスクに書き込まれることはなく、したがって、ディスク・スペースを占有しません。しかし、表自体は、サーバーのシャットダウン時 (およびチェックポイント処理中) にディスクに書き込まれます。そのため、ディスク・スペースの合計容量として、ディスク・ベース表とインメモリー表を両方とも格納するのに十分なスペースが必要です。

付録 C. 構成パラメーター

パラメーターは、solid.ini 構成ファイル内でセクション・カテゴリーに従ってグループ化されています。solidDB インメモリー・データベースに関連するパラメーターは、構成ファイルの [MME] セクションに格納されています。さらに、[General] セクションに、DefaultStoreIsMemory というパラメーターが 1 つ格納されています。

構成パラメーターは、solid.ini 構成ファイルを手動で編集するか、または solidDB SQL エディターに以下のコマンドを入力して変更できます。

```
ADMIN COMMAND 'parameter section_name.param_name=value'
```

以下に例を示します。

ADMIN COMMAND 'parameter mme.imdbmemorylimit=1gb';

注:

サーバーは、始動時にのみ構成ファイルを読み取ります。したがって、構成ファイルの変更は、サーバーの次回の始動時まで有効になりません。

IMDB 関連のすべての構成パラメーターのリストを以下に示します。

General セクション

表 7. MME に関連する [General] セクション内のパラメーター

[General]	説明	ファクトリー値	アクセス・モード
DefaultStoreIsMemory	Yes に設定した場合、CREATE TABLE ステートメントに明示的な STORE 節を指定しないで新しい表を作成すると、それはインメモリー表として作成されます。No に設定した場合、デフォルトでは、新しい表はディスクに格納されます。CREATE TABLE ステートメントに STORE 節を指定して、デフォルト値をオーバーライドできます。 このパラメーターは、solidDB インメモリー・データベースをサポートする製品にのみ適用されます。 このパラメーターを Yes に設定した場合でも、システム表はディスクに格納されることに注意してください。	YES	RW

表 7. MME に関連する [General] セクション内のパラメーター (続き)

[General]	説明	ファクトリー値	アクセス・モード
MultipropcessingLevel	コンピューター・システム内で使用可能な物理処理装置 (プロセッサー、コア) の数を定義します。 この値をシステム内のプロセッサー (コア) 数に一致するように設定すると、データベース内の書き込み操作の並行性が向上します。	4	RW/Startup

MME セクション

注:

DefaultStoreIsMemory パラメーター (`solid.ini` ファイルの [General] セクション内のパラメーター) は、solidDB インメモリー・データベースにも関連しています。

表 8. MME パラメーター

[MME]	説明	ファクトリー値	アクセス・モード

ImdbMemoryLimit	これは、サーバーがインメモリー表とその索引に割り振るメモリー (仮想メモリー) の容量の上限を設定します。「インメモリー表」には、「通常の」(パーシスタント) インメモリー表だけでなく、テンポラリー表とトランジエント表も含まれることに注意してください。
	
	この限度は、バイト、キロバイト (KB)、メガバイト(MB)、ギガバイト (GB) の単位で指定できます。以下に例を示します。
	`ImdbMemoryLimit=1073741824`
	`ImdbMemoryLimit=1048576kb`
	`ImdbMemoryLimit=1024MB`
	`ImdbMemoryLimit=1GB`
	
	値 0 を指定すると、「限度なし」になります。
	
	概して、搭載メモリーが 1 GB 未満のサーバーでは、インメモリー表に割り振る必要がある最大容量は、通常、システムの物理メモリーの 30% から 70% 程度です。システムの搭載メモリーが増えるほど、インメモリー表に使用可能なメモリーのパーセンテージも大きくなります。
	注: このパラメーターが適用されるのは、solidDB メイン・メモリー・エンジン表のみです。その他のバージョンの solidDB またはディスク・ベース表には適用されません。
	
	このパラメーターは、以下のコマンドで変更できます。
	`ADMIN COMMAND 'parameter MME.ImdbMemoryLimit=n[kb|mb|gb]';`
	
	「n」は正の整数です。サーバーが稼働しているときに、この値を大きくすることはできますが、小さくすることはできません。コマンドはすぐに有効になります。新しい値は、シャットダウン時に `solid.ini` ファイルに書き戻されます。
	
	注意:
	必ず、使用するインメモリー表が物理メモリーの空き容量内に収まるようにしてください。物理メモリーの空き容量を超えると、パフォーマンスが著しく低下します。仮想メモリーの空き容量を使い切ると、サーバーは挿入、更新などを突然、制限し始め、エラー・コードを返すようになります。
		0
		
		単位: バイト、キロバイト (KB)、メガバイト (MB)、ギガバイト (GB)
			RW

表 8. MME パラメーター (続き)

[MME]	説明	ファクトリー値	アクセス・モード
ImdbMemoryLowPercentage	**ImdbMemoryLimit** を設定している場合、このパラメーターを追加で設定すれば、メモリーをすべて使い切る前に警告を出すことができます。この **ImdbMemoryLowPercentage** パラメーターを使用すると、使用可能メモリーの容量が何パーセントになったら、インメモリー表に対する行の挿入などの操作を制限し始めるのかを設定することができます。例えば、**ImdbMemoryLimit** が 1000 MB で **ImdbMemoryLowPercentage** が 90 (パーセント) の場合、インメモリー表のメモリーのうち 900 メガバイトを使い切ると、サーバーは挿入を受け入れなくなります。 有効な値は 60 から 99 (パーセント) までの間です。 **注:** このパラメーターが適用される対象は、solidDB メイン・メモリー・エンジン表のみです。その他のバージョンの solidDB またはディスク・ベース表には適用されません。	90	RW/Startup
ImdbMemoryWarningPercentage	このパラメーターは、IMDB メモリー・サイズに対する警告限度を設定します。警告限度は、**ImdbMemoryLimit** パラメーター値に対するパーセンテージとして表されます。**ImdbMemoryWarningPercentage** 限度を超えると、システム・イベントが発生します。 **ImdbMemoryWarningPercentage** パラメーター値の整合性は自動的に検査されます。このパラメーター値は、**ImdbMemoryLimit** パラメーター値よりも小さな値でなければなりません。 **注:** このパラメーターが適用される対象は、solidDB メイン・メモリー・エンジン表のみです。その他のバージョンの solidDB またはディスク・ベース表には適用されません。	80	RW/Startup

表 8. MME パラメーター (続き)

[MME]	説明	ファクトリー値	アクセス・モード
LockEscalationEnabled	並行性競合が発生しないようにサーバーがロックをかける必要がある場合、通常、サーバーは行単位でロックをかけます。つまり、同じ行を使用しようとする他のユーザーがいる場合にのみ影響が生じます。しかし、ロックされる行の数が増えると、それだけサーバーがロックの競合検査に費やさなければならない時間も長くなります。 場合によっては、表内の多数の行をロックするよりも、その表全体をロックする方が効率的です。 **LockEscalationEnabled** を yes に設定した場合、現行トランザクションで (同一表内の) 行が指定の数だけロックされると、ロック・レベルが行レベルから表レベルにエスカレートされます。 ロック・エスカレーションが発生するとパフォーマンスは向上しますが、他のユーザーが同じ表の別の行を使用したくても、その表を一時的に使用できなくなるため、並行性は低下します。 パラメーター **LockEscalationLimit** も参照してください。 値に指定できるのは「yes」または「no」です。 **注:** このパラメーターは、インメモリー表のみに適用されます。	no	RW/Startup
LockEscalationLimit	**LockEscalationEnabled** を yes に設定した場合、このパラメーターは、(同一表内で) 何行までロックされて初めてサーバーがロック・レベルを行レベルから表レベルにエスカレートするかを示します。詳しくは、**LockEscalationEnabled** を参照してください。 値は、1 から 2,147,483,647 (2^32-1) までの任意の数です。 **注:** このパラメーターは、インメモリー表のみに適用されます。	1000	RW/Startup

表 8. MME パラメーター (続き)

[MME]	説明	ファクトリー値	アクセス・モード
LockHashSize	サーバーはロック情報の格納にハッシュ表 (配列) を使用します。配列のサイズを著しく低く見積もっていると、パフォーマンスが低下します。ハッシュ表のサイズが大きすぎても、パフォーマンスに直接は影響しませんが、メモリー・オーバーヘッドの原因になります。**LockHashSize** はハッシュ表の要素数を決定します。 この情報は、サーバーがペシミスティック並行性制御 (ロック方式) を使用している場合に必要です。サーバーは、インメモリー表とディスク・ベース表にそれぞれ別の配列を使用します。このパラメーターは、インメモリー表に適用されます。 概して、必要なロックの数が多いほど、この配列も大きくする必要があります。しかし、必要なロックの数を計算で求めることは難しいため、アプリケーションに最適な値を見つけるために、実際に試してみることが必要になる場合もあります。 入力する値は、ハッシュ表の項目数です。表の各項目のサイズは、ポインター 1 つ分 (32 ビット・アーキテクチャーの場合 4 バイト) です。したがって、例えば、選択したハッシュ表のサイズが 1,000,000 である場合には、必要なメモリーの容量は 4,000,000 バイトです (32 ビット・ポインターを前提とした場合)。	1000000	RW/Startup
MaxCacheUsage	**MaxCacheUsage** の値は、インメモリー表のチェックポイント処理時に使用される、ディスク・ベース表のキャッシュの容量を制限します。値は、バイト単位で指定されたものと認識されます。**MaxCacheUsage** の値に関係なく、最大でディスク・ベース表のキャッシュ (**IndexFile.CacheSize**) の半分がインメモリー表のチェックポイント処理に使用されます。値を **MaxCacheUsage=0** に設定すると、このパラメーター値による限度はなくなります。つまり、キャッシュの使用量は **IndexFile.CacheSize/2** になります。	8 MB	RW/Startup
NumberOfMemoryPools	このパラメーターは、グローバル・メモリー・プール数を定義します。値が大きいほど特定のロード・シナリオでマルチコア・システムのパフォーマンスが向上しますが、メモリーの遊びも増大し、結果的にサーバー・プロセス・サイズが大きくなります。 最小値は 1 です。最大値はありません。ただし、システム内のコア数を超えてはなりません。	1	RW/Startup

表 8. MME パラメーター (続き)

[MME]	説明	ファクトリー値	アクセス・モード
ReleaseMemoryAtShutdown	「yes」に設定した場合、サーバーがシャットダウンするときに、このプロセスに関連するすべてのメモリーのクリーンアップをオペレーティング・システムに任せるのではなく、インメモリー表が使用しているメモリーの解放をサーバーが明示的に実行するようになります。一部のオペレーティング・システム (VxWorks など) では、必ず、すべてのメモリーが解放されるように、これを「yes」に設定することが必要になる場合があります。 指定できる値は、yes と no です。 サーバーをシャットダウンすれば済むので、ファクトリー値はありません。	No	RW/Startup
RestoreThreads	データベース開始処理中にインメモリー・データベースのリストアに使用されるスレッドの最大数を定義します。 値が 1 の場合、ロードが単一スレッドで実行されることを意味します。 インメモリー・データベースのリストアは、表の数がパラメーター値以下であると、個々の表に 1 個のスレッドを割り当てます。 パラメーター値が、コアまたは処理装置の数と、データベース内の表の数という 2 つの値より小さい場合、並行性が最大になります。 有効な値は 1 から 2^31 - 1 です。	4	RW/Startup

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]

[ハ行]

［マ行］

［ラ行］

A

B

C

D

H

I

L

M

N

P

R

S

［特殊文字］

特記事項

本製品は、米国特許 6144941、 7136912、 6970876、 7139775、 6978396、 7266702、 7406489、および 7502796 により保護されています。

本製品は、米国輸出規制品目分類番号 ECCN=5D992b に指定されています。

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。

〒242-8502
神奈川県大和市下鶴間1623番14号
日本アイ・ビー・エム株式会社
法務・知的財産
知的財産権ライセンス渉外

以下の保証は、国または地域の法律に沿わない場合は、適用されません。 IBM およびその直接または間接の子会社は、本書を特定物として現存するままの状態で提供し、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。国または地域によっては、法律の強行規定により、保証責任の制限が禁じられる場合、強行規定の制限を受けるものとします。

この情報には、技術的に不適切な記述や誤植を含む場合があります。本書は定期的に見直され、必要な変更は本書の次版に組み込まれます。 IBM は予告なしに、随時、この文書に記載されている製品またはプログラムに対して、改良または変更を行うことがあります。

本書において IBM 以外の Web サイトに言及している場合がありますが、便宜のため記載しただけであり、決してそれらの Web サイトを推奨するものではありません。それらの Web サイトにある資料は、この IBM 製品の資料の一部ではありません。それらの Web サイトは、お客様の責任でご使用ください。

IBM は、お客様が提供するいかなる情報も、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で、使用もしくは配布することができるものとします。

本プログラムのライセンス保持者で、(i) 独自に作成したプログラムとその他のプログラム (本プログラムを含む) との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。

IBM 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的創作物にも、次のように、著作権表示を入れていただく必要があります。

© (お客様の会社名) (西暦年)。このコードの一部は、IBM Corp. のサンプル・プログラムから取られています。

商標

IBM、IBM ロゴ、ibm.com®、Solid、solidDB、InfoSphere™、DB2®、Informix®、および WebSphere® は、International Business Machines Corporation の米国およびその他の国における商標です。これらおよび他の IBM 商標に、この情報の最初に現れる個所で商標表示 (® または ™) が付されている場合、これらの表示は、この情報が公開された時点で、米国において、IBM が所有する登録商標またはコモン・ロー上の商標であることを示しています。このような商標は、その他の国においても登録商標またはコモン・ロー上の商標である可能性があります。現時点での IBM の商標リストについては、「Copyright and trademark information」(www.ibm.com/legal/copytrade.shtml) をご覧下さい。

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。

Linux® は、Linus Torvalds の米国およびその他の国における商標です。

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。

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。

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。

Printed in Japan

SC88-8165-00

日本アイ・ビー・エム株式会社
〒103-8510 東京都中央区日本橋箱崎町19-21