IBM solidDB
バージョン 7.0

インメモリー・データベース・ユーザー・ガイド

SA88-4537-01
(英文原典：SC27-3845-04)

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、 43 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 7.0 フィックスパック 5 の IBM solidDB (製品番号 5724-V17) 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： SC27-3845-04
IBM solidDB
Version 7.0
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 2013.3

目次

表

変更の要約

改訂 04 での変更点

- 編集上の修正。

改訂 03 での変更点

- 編集上の修正。

改訂 02 での変更点

- 「パーシスタント・インメモリー表」のセクションで、並行性制御に関する情報が更新されました。

改訂 01 での変更点

- 編集上の修正。

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 solidDB のインメモリー機能の概要

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。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・サーバーは低速となる可能性があります。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。

1.1 インメモリー表とディスク・ベース表の対比

表がインメモリー表の場合、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。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一度にごく少量のデータのみがコピーされます。

アプリケーション設計の観点から見ると、インメモリー表およびディスク・ベース表はほとんどの点で同じです。

- 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ます。
- それぞれの表で同じタイプの照会を実行できます。
- 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。
- どちらの表タイプも、索引、トリガー、ストアード・プロシージャー、およびその他の共通データベース・オブジェクトで使用できます。
- どちらの表タイプも、主キー制約、外部キー制約などの制約を使用できます。ただし、ノンパーシスタント・インメモリー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。

インメモリー表とディスク・ベース表との主な違いはパフォーマンスです。インメモリー表では、パフォーマンスが向上します。また、ディスク・ベース表と同じ耐久性および復元可能性を提供できます。例えば、インメモリー表での読み取り操作は、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。

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ます。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。

1.2 インメモリー表のタイプ

インメモリー表には、パーシスタント表とノンパーシスタント表の 2 つの基本タイプがあります。パーシスタント表は、データの復元可能性を提供するのに対し、ノンパーシスタント表は高速アクセスを提供します。

ディスク・ベース表は、常にパーシスタント表です。

1.2.1 パーシスタント・インメモリー表

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パーシスタント・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。パーシスタント・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。(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、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反映させることができます。ディスク・ベース表の場合と同じように、パーシスタント・インメモリー表のデータは、チェックポイント処理中にハード・ディスクにコピーされます。

また、パーシスタント・インメモリー表は、solidDB 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。その場合、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。

パーシスタント・インメモリー表とディスク・ベース表の相違点

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。以下のセクションでは、インメモリー表とディスク・ベース表の相違点を中心に説明します。

並行性制御

インメモリー表は、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) を使用します。ディスク・ベース表は、デフォルトでオプティミスティック並行性制御 (バージョニング方式) を使用します。

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。

チェックポイント処理アルゴリズム

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は、チェックポイント処理中にトランザクションがどのような方法で表にアクセスすることも妨げません。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。

副次索引

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ありません。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。副次索引がインメモリー表の書き込みパフォーマンスに与える影響は、ディスク・ベース表の場合に比べてかなり小さくなります。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。

1.2.2 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・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。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。

ノンパーシスタント表は、スクラッチパッドとして使用されます。例えば、パーシスタント表からデータをコピーし、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。

ノンパーシスタント表のトランザクションはログに記録されないため、ノンパーシスタント表は HotStandby や solidDB Universal Cache では使用できません。

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からのみ可視で、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。テンポラリー表は、ロギングを使用せず、また、どのようなタイプの並行性制御メカニズム (レコード・ロッキングなど) も使用しないため、トランジエント表よりさらに高速です。

可視性の制限

テンポラリー表のデータの可視性には制限があります。これは、データを挿入したセッション (接続) のみがそのデータを参照できるためです。

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した場合、他のユーザー・セッションは、その表に対する特権が付与されていても、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ことができますが、それぞれのセッションは、そのセッションが保有するデータしか認識できません。

各セッションが認識できるのはそのセッションが保有するデータのみであるため、表に固有の制約がある場合にも、固有の値を表に挿入できるようにするために他のセッションと調整する必要はありません。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 の値を 1 に設定したレコードを挿入することが可能です。各セッションは自身が保有するデータしか認識しないため、UPDATE や DELETE などの操作は、そのセッション内のデータにのみ影響を与えます。

存続期間の制限

テンポラリー表では、現行セッションを終了する (つまり、サーバーを切断する) とすぐにデータが廃棄されるため、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は、データはもう存在しません。

テンポラリー表 という用語におけるテンポラリーという単語は、表自体ではなくデータのことを表しています。サーバーは、テンポラリー表の定義 (データではなく) をシステム表に保管するため、切断した後も定義は残ります。したがって、後でサーバーに再接続したときは、表自体は残っていますが、その内容は空になっています。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自分や他のユーザーが作成しようとすると、テンポラリー表の所有者はエラー・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。

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。また、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。

テンポラリー表は、ユーザーが切断すると内容が消去されるため、テンポラリー表に大量のデータが含まれるセッションの後は、しばらくプロセッサー使用率が高くなることがあります。

その他の特性

- HotStandby コンポーネントを使用する場合、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。ただし、テンポラリー表の定義自体は、2 次サーバーに複製されます。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に、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、その中のデータは再作成する必要があります。
- Universal Cache で solidDB をソース・データ・ストアとする場合、テンポラリー表はサポートされないため、テンポラリー表をサブスクリプションの一部として使用することはできません。solidDB をターゲット・データ・ストアとするサブスクリプションでは、テンポラリー表を使用できます。
- 拡張レプリケーション・システムでは、テンポラリー表はマスター表としてではなく、レプリカ表としてのみ使用できます。
- テンポラリー表には、参照制約に伴って使用方法に制限があります。テンポラリー表は、他のテンポラリー表を参照できますが、トランジエント表やパーシスタント表は参照できません。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。以下に例を示します。

- テンポラリー表には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。
- テンポラリー表にはトリガーを作成できます。
- テンポラリー表には 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、特定のカタログおよびスキーマの中に存在します。
- 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したり、特権を取り消したりすることができます。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して SELECT 特権を持つ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す。

規格準拠

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* に関する ANSI SQL:1999 の規格に完全に準拠しています。solidDB のテンポラリー表は、すべてグローバル・テンポラリー表です。互換性の理由により、CREATE TABLE 構文ではキーワード GLOBAL がサポートされています。とはいえ、キーワード GLOBAL が指定されていなくても、すべてのテンポラリー表はグローバルです。

solidDB サーバーは、ANSI が定める*ローカル・テンポラリー表* をサポートしていません。

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すべてのデータを見ることができます。

多くの点で、トランジエント表の動作は標準 (パーシスタント) インメモリー表と類似しています。以下に例を示します。

- トランジエント表のデータには、パーシスタント表のデータと同じスコープまたは可視性があります。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は BLOB 列を含めることができます。ただし、BLOB 列の長さはすべてのインメモリー表で数 KB 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。
- solidDB Speed Loader (**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および表の構造がエクスポートされます。

サーバーはトランジエント表の定義 (データではなく) を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、その定義を残します。サーバーを後で再始動すると、表は存在しますが、データはありません。したがって、表は 1 度だけ作成する必要があります。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シャットダウンと同時に消えるもの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。

また、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。

制限

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。

- 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ます。したがって、2 次にフェイルオーバーするために、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、その中のデータは再作成する必要があります。
- Universal Cache で solidDB をソース・データ・ストアとする場合、トランジエント表はサポートされないため、トランジエント表をサブスクリプションの一部として使用することはできません。 solidDB をターゲット・データ・ストアとするサブスクリプションでは、トランジエント表を使用できます。
- 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およびパーシスタント表を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。テンポラリー表およびパーシスタント表は、トランジエント表を参照することはできません。
- 拡張レプリケーション・システムでは、トランジエント表はマスター表としてではなく、レプリカ表としてのみ使用できます。

規格準拠

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

テンポラリー表とトランジエント表の主な違いは、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。ペシミスティック並行性制御 (ロッキング) のみがサポートされます。
- テンポラリー表は、並行性制御を使用しないため、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の間に限られます。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。

1.2.3 表タイプと参照整合性

パーシスタント表と非パーシスタント表とでは、参照整合性が異なります。

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。

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。

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

1.3 インメモリー表を使用するアプリケーション開発時の考慮事項

インメモリー表を使用してアプリケーションの開発を始める前に、以下に示すパフォーマンス、メモリー使用量、トランザクション分離に関する考慮事項、およびインメモリー表を HotStandby または共有メモリー (SMA) とリンク・ライブラリー (LLA) アクセス方式で使用する場合の考慮事項を検討してください。

1.3.1 パフォーマンスおよびインメモリー表

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、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。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。

テンポラリー表とトランジエント表およびパフォーマンス

テンポラリー表とトランジエント表のパフォーマンスは、以下の理由により、パーシスタント表よりも高くなっています。

- テンポラリー表とトランジエント表のデータは、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。
- テンポラリー表とトランジエント表は、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・ストレージ構造を使用します。
- テンポラリー表には、この他にも、トランジエント表と比較した場合の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ありません)。

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。これにより、パフォーマンスが向上しますが、メモリー・スペースの消費量が増えます。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ため、表のデータへのアクセス速度を向上させるために、インメモリー索引を使用するようにしてください。ただし、表と索引をすべて格納するために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では問題の解決に役立たないことがあります。一部の照会の速度は速くなりますが、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により、その他の照会の速度が遅くなるからです。

1.3.2 物理メモリーと仮想メモリー

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ません。

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してください。

表に必要なスペース容量を計算する際に、「BLOB」データ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には、表内のデータ用のスペースのみでなく、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、その表のすべての索引用のスペースも含まれます。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。

1.3.3 インメモリー表を使用する場合のトランザクション分離制限

インメモリー表では、SERIALIZABLE 分離レベルはサポートされません。

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、インメモリー表は使用できません。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 次サーバーのインメモリー表の場合、トランザクション分離レベルは常に READ COMMITTED です。

HotStandby を使用していて HotStandby 2 次サーバーに接続した場合は、インメモリー表からデータを読み取ると、トランザクション分離レベルは、たとえ REPEATABLE READ を指定したとしても自動的に READ COMMITTED に設定されます。

REPEATABLE READ に関連する、インメモリー表とディスク・ベース表の違いがあります。

インメモリー表を使用する場合、トランザクション分離レベルを REPEATABLE READ に設定すると (デフォルトは READ COMMITTED)、読み取りトランザクションの間、読み取り操作は書き込み操作をブロックします。さらに、READ REPEATABLE 分離レベルを使用する場合、インメモリー表ではデッドロックが発生する可能性がありますが、バージョニング方式のディスク・ベース表ではデッドロックは発生しません。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。

1.3.4 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、イン・メモリー表と互換性があります。

詳しくは、『共有メモリー・アクセスおよびリンク・ライブラリー・アクセスの概要』を参照してください。

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。

1.3.5 HotStandby とインメモリー表

インメモリー表は、以下の考慮事項を前提として、solidDB 高可用性で使用できます。

- パーシスタント・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ません。

2 インメモリー表での操作

2.1 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、実際には、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。

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てみてください。原則として、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はディスクに残します。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を使用した場合、表のタイプは表へのアクセス「密度」に対応したものにすべきです。インメモリー表は、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効率的に機能します。

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。基本的なガイドラインを以下に示します。

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。

   「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。
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。

注: テンポラリー表およびトランジエント表には、ユーザーの決定に影響を与える可能性のある制限事項があります。例えば、テンポラリー表は、他のテンポラリー表を参照できますが、トランジエント表やパーシスタント表は参照できません。他のタイプの表は、テンポラリー表を参照できません。

関連情報:

29 ページの『付録 A. メモリー内に格納する表を選択するアルゴリズム』

3 ページの『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からのみ可視で、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。テンポラリー表は、ロギングを使用せず、また、どのようなタイプの並行性制御メカニズム (レコード・ロッキングなど) も使用しないため、トランジエント表よりさらに高速です。

5 ページの『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すべてのデータを見ることができます。

2.2 インメモリー表とディスク・ベース表の作成

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。

1. CREATE TABLE コマンドまたは ALTER TABLE コマンドの STORE MEMORY 節または STORE DISK 節を使用します。

   ```
   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. **General.DefaultStoreIsMemory** パラメーターを使用してデフォルトを指定します。

   以下に例を示します。

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

   General.DefaultStoreIsMemory 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。

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。

2.3 テンポラリー表とトランジエント表の作成

デフォルトでは、インメモリー表を作成する場合、表はパーシスタント表になります。テンポラリー表やトランジエント表を作成する場合は、キーワード TEMPORARY または TRANSIENT を使用します。

テンポラリー表の作成

テンポラリー表を作成するには、次のコマンドを使用します。

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

上記の要素について以下に説明します。

GLOBAL は、互換性の理由によりサポートされています。とはいえ、キーワード GLOBAL が指定されていなくても、すべてのテンポラリー表はグローバルです。

<...> 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。

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。

トランジエント表の作成

トランジエント表を作成するには、次のコマンドを使用します。

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

上記の要素について以下に説明します。

<...> 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。

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。

2.4 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表からディスク・ベースの表に、またはその逆に変更できます。これを行うには、以下のコマンドを使用します。

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

表にデータが含まれている場合、データのコピー先として、別の名前を付けて新規の表を作成する必要があります。データを新規の表にコピーした後、古い表を除去し、元の表と同じ名前で新規の表の名前を変更することができます。

3 インメモリー・データベースの構成

General.DefaultStoreIsMemory パラメーターを yes に設定することにより、デフォルトで新規表をインメモリー表として作成するようにsolidDB を構成することができます。その他の多くのインメモリー機能は、solid.ini ファイルの [MME] セクションでパラメーターを使用して構成されます。

メモリー使用量の制御には特別な注意を払う必要があります。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内の使用可能な仮想メモリーをすべて使い切ると、データの追加や更新を行えなくなります。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。

3.1 構成パラメーター

solidDB インメモリー・データベースに関連するパラメーターのほとんどは、solid.ini 構成ファイルの [MME] セクションに格納されています。

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。

3.1.1 General セクション

表 3. 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。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、ファクトリー値をオーバーライドできます。 **注:** 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ます。	yes	RW
MultiprocessingLevel	このパラメーターは、コンピューター・システム内で使用可能な処理装置 (プロセッサー、コア) の数を定義します。一般には、この値をシステム内の物理プロセッサー (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を向上させることができます。 V6.5 フィックスパック 4 では、ファクトリー値は、論理処理装置の数としてシステムから読み取られます。自動検出された値は、サーバーの始動時に solmsg.out に出力されます。一部のプロセッサー・アーキテクチャーでは、論理処理装置の数が物理コアの数と異なる場合があります。このような場合、このパラメーターの最適値は一般的に、物理コアの数と論理処理装置の数の間で変化します。 V6.5 フィックスパック 4 より前のリリースでは、このパラメーターのファクトリー値は 4 です。 **注:** V6.5 フィックスパック 4 では、**MME.RestoreThreads** パラメーターの値は、別の値に明示的に設定されていない限り、デフォルトによりこのパラメーターの値に設定されます。	システムからの読み取り	RW/Startup

3.1.2 MME セクション

表 4. 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、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ます。
	
	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`
	
	値が 0 の場合は「無制限」であることを示します。
	
	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。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は、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。
	重要: 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。使用可能な物理メモリーの容量を超えると、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

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。	90	RW

表 4. 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
ImdbMemoryWarningPercentage	このパラメーターは、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。 **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。ディスク・ベース表には適用されません。	80	RW
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。 このパラメーターを yes に設定した場合、現行トランザクションにおいて、指定した数の行が同一表内でロックされると、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。 可能な値は yes および 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
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

表 4. 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
MaxBytesCachedInPrivateMemoryPool	このパラメーターは、MME の専用メモリー・プール (専用メモリー・プールは、メイン・メモリーの索引ごとに個別に用意されています) のフリー・リストに格納される最大バイト数を定義します。専用プールにさらに空きメモリーがある場合は、その余剰メモリーはグローバル・プールにマージされます。 このパラメーターの値が 0 の場合、グローバル・プールへのマージが直ちに行われます。通常、パフォーマンスは低下しますが、メモリーのフットプリントは最小化されます。最大値はありません。デフォルト値の 100000 で、優れたパフォーマンスが得られます。メモリー・オーバーヘッドはほとんどありません。	100000	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
MaxTransactionSize	このパラメーターでは、トランザクションの概算最大サイズをバイト単位で定義します。 MME トランザクション (DELETE FROM <table> など) によっては、solidDB が操作のために大量のメモリーを割り振ることがあります。これによってメモリー不足状態が発生し、solidDB がオペレーティング・システムからそれ以上メモリーを割り振ることができなくなり、非常時終了を実行することがあります。この状態が起きないようにするには、このパラメーターを使用して、MME トランザクションごとに最大概算サイズ (バイト単位) を定義します。トランザクションのサイズがこのパラメーターで設定した値を超えると、そのトランザクションは `SOLID Database error 16509: MME transaction maximum size exceeded` エラーで失敗します。 値が 0 の場合は無制限であることを示します。	0	RW

MemoryPoolScope	このパラメーターではメモリー・プールの有効範囲を設定します。設定可能な値は「`Global`」と「`Table`」です。
	
	「`Table`」に設定すると、同じデータベース表に属するオブジェクトのみが単一のメモリー・セグメントから割り振られます。この機能により、例えば 1 つの表全体を除去した場合、メモリー・セグメントが解放されてオペレーティング・システムに戻されるようになります。未使用のメモリー・セグメントのみがシステムに戻される場合もあります。
	
	「`Global`」に設定すると、すべての MME データ間でメモリー・プールが共有されます。
	
	MME.MemoryPoolScope を `Table` に設定した場合、DESCRIBE <table> ステートメントを使用して、該当の表に関するメモリー消費量を確認することができます。以下に例を示します。
	`DESCRIBE tmemlimit_tab;`
	`RESULT`
	`------`
	`Catalog: DBA`
	`  Schema: DBA`
	`    Table: TMEMLIMIT_TAB`
	`      Table type: in-memory`
	
	`      Memory usage: 7935 KB (total), 7925 KB (active),`
	`                    6192 KB (rows), 1733 KB (indexes).`
	
	`...`
	`1 rows fetched.`
		グローバル	RW/Startup

表 4. 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
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
ReleaseMemoryAtShutdown	yes に設定すると、このプロセスに関連付けられた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・システムに行わせる代わりに、サーバーがシャットダウン時に、インメモリー表によって使用されたメモリーを明示的に解放するようになります。一部のオペレーティング・システムでは、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。 可能な値は yes および 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 no です。	no	RW/Startup
RestoreThreads	このパラメーターは、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。このパラメーターを明示的に設定しない場合、このパラメーターの値は **General.MultiprocessingLevel** と同じ値に設定されます。 有効な値は 1 から 65536 までの間です。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。 値が無効な場合、このパラメーターはデフォルトで **General.MultiprocessingLevel** の値に設定されます。 インメモリー・データベースのリストアでは、表の数がパラメーター値以下であると、個々の表ごとにスレッドが 1 個ずつ割り当てられます。 並行性は、パラメーター値がコア/プロセッサーの数と、データベース内の表の数の 2 つの値より小さいときに、最大になります。	**General. Multiprocessing Level** と同じ	RW/Startup

3.2 メモリー使用量

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 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。

3.2.1 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。

ADMIN COMMAND 'info imdbsize'

ADMIN COMMAND 'info imdbsize'; コマンドは、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。

ADMIN COMMAND 'info processsize'

ADMIN COMMAND 'info processsize'; コマンドは、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。このコマンドが返す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ありません。

ADMIN COMMAND 'pmon mme'

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'; コマンドで、カウンターの現行値の以下の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。

ADMIN COMMAND 'memory'

ADMIN COMMAND 'memory'; コマンドは、動的に割り振られたヒープ・メモリーの容量を報告します。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。

3.2.2 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。

データベース・メモリー

MME.ImdbMemoryLimit: **MME.ImdbMemoryLimit** パラメーターは、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) 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。

MME.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。デフォルト値は使用しないでください。その代わりに、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。以下の要因も考慮してください。

- 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

MME.ImdbMemoryLimit で指定されたメモリーの 100% に到達すると、サーバーは、インメモリー表内のレコードに対する 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、『MME.ImdbMemoryLowPercentage』を参照してください。

以下に例を示します。

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

MME.ImdbMemoryLowPercentage: **MME.ImdbMemoryLowPercentage** パラメーターでは、インメモリー表に割り振ることができる仮想メモリー量の「下限基準点」を設定します。限度は、**MME.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。

サーバーは、**MME.ImdbMemoryLowPercentage** で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ぐために、アクティビティーの制限を開始します。例えば、**MME.ImdbMemoryLimit** が 1000 メガバイトで、**MME.ImdbMemoryLowPercentage** が 90% である場合、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、サーバーが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、(テンポラリー表とトランジエント表を含む) インメモリー表の作成およびインメモリー表に対する索引が作成されないようにします。
- インメモリー表への INSERT を禁止します。

MME.ImdbMemoryLimit で設定された上限自体に到達すると、サーバーは、インメモリー表内のレコードに対する UPDATE 操作も禁止します。

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

MME.ImdbMemoryWarningPercentage: **MME.ImdbMemoryWarningPercentage** パラメーターで設定した限度に到達すると、インメモリー表に割り振ることができる仮想メモリーの最大量に達したことを警告するためのシステム・イベントが発生します。

警告限度は、**MME.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。 **MME.ImdbMemoryWarningPercentage** 限度を超えると、システム・イベントが発生します。

MME.ImdbMemoryLimit のトラブルシューティング:

MME.ImdbMemoryLimit で設定された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、すぐに対処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ご使用のシステムがメモリー不足にならないようにすることです。長期的には、将来、表が拡張されてもメモリー不足にならないようにする必要があります。

当面の問題の解決

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。また、切断したユーザーがテンポラリー表を使用していた場合には、切断によって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. メモリーを解放するために十分なテンポラリー表がない場合には、いくつかのトランジエント表の索引、またはトランジエント表自体を一部ドロップします (それらがある場合)。

メモリーを解放するために十分なテンポラリー表およびトランジエント表がない場合には、以下のようにします。

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、そのいずれの表にも索引がなく、またその表すべてに重要データが格納されているなど) 廃棄可能なものがまったくメモリーに存在しない場合は、**MME.ImdbMemoryLimit** の値を少し大きくしてから、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、ユーザーがテンポラリー表またはトランジエント表を作成することがないようにしてください。

長期的な問題の解決

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空き容量を確保した後で、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してください。そのための方法としては、(テンポラリー表、トランジエント表を含む) インメモリー表の数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のみ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ることがないようにします。
- 通常のインメモリー表で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に、サーバーが使用できるメモリー容量を増やすことができないときには、メイン・メモリーから 1 つ以上の表をディスクに移動します。

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中間的なディスク・ベース表にコピーします。

   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、より小さな (例えば、トランザクションごとの行数が数千程度の) トランザクションで行う必要があります。

   注: 中間的な表には索引は不要です。データのコピーが正常に完了したら、新規の表に索引を再作成する必要があります。
3. インメモリー表をドロップします。
4. 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。

ヒント:

- 実際に使用可能な最大容量よりもやや低めの値を **MME.ImdbMemoryLimit** に設定する必要があります。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、**MME.ImdbMemoryLimit** の値を少し大きくし、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して、サーバーを再始動することができます。
- メモリー使用量が増加していることを警告するために、**MME.ImdbMemoryWarningPercentage** を使用します。
- 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。

プロセス・メモリー

Srv.ProcessMemoryLimit:

Srv.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。

Srv.ProcessMemoryLimit のファクトリー値は 0 です。この場合、プロセス・メモリーに限度はありません。このパラメーターを使用する場合には、インメモリー・データベース・プロセスが完全に物理メモリー内に確実に収まるような値に、このパラメーターを設定してください。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 (テンポラリー表およびトランジエント表を含みます) およびインメモリー表の索引によって使用されるメモリーの量
- solidDB サーバーのキャッシュ用に確保されたメモリーの量 (**IndexFile.CacheSize** 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

限度に達したとき、つまりインメモリー・データベース・プロセスが **Srv.ProcessMemoryLimit** で指定されたメモリーを 100% 使い切ってしまったときには、サーバーは ADMIN コマンドしか受け付けなくなります。
Srv.ProcessMemoryWarningPercentage および **Srv.ProcessMemoryLowPercentage** パラメーターを使用して、プロセス・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。

注:

- **Srv.ProcessMemoryLimit** パラメーターと **Srv.ProcessMemoryCheckInterval** パラメーターは相互にリンクし合っています。**ProcessMemoryCheckInterval** パラメーターを 0 に設定すると、**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。
- SMA を使用している場合は、**Srv.ProcessMemoryLimit** パラメーターを設定しないでください。SMA サーバーが使用するメモリーを制限する必要がある場合は、**SharedMemoryAccess.MaxSharedMemorySize** パラメーターを使用してください。

Srv.ProcessMemoryLowPercentage:

Srv.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。限度は、**Srv.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。
Srv.ProcessMemoryLowPercentage 限度を超えると、システム・イベントが発生します。

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

Srv.ProcessMemoryWarningPercentage:

Srv.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、**Srv.ProcessMemoryLimit** パラメーター値のパーセントとして表します。

Srv.ProcessMemoryWarningPercentage 限度を超えると、システム・イベントが発生します。

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

Srv.ProcessMemoryCheckInterval:

Srv.ProcessMemoryCheckInterval パラメーターでは、プロセス・サイズ限度を検査する間隔を定義します。間隔はミリ秒単位で指定します。

Srv.ProcessMemoryCheckInterval の非ゼロの最小値は、1000 (ミリ秒) です。可能な値は、0、1000、または 1000 (1 秒) を超える値のいずれかです。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。

Srv.ProcessMemoryLimit パラメーターと **Srv.ProcessMemoryCheckInterval** パラメーターは相互にリンクし合っています。**ProcessMemoryCheckInterval** パラメーターを 0 に設定すると、**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。

付録 A. メモリー内に格納する表を選択するアルゴリズム

このセクションでは、メモリー内に格納する表を選択する戦略について説明します。

主要な原則は、表へのアクセス密度を考慮することです。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、ここでは行/MB/秒と記述します。(簡単にするために、行当たり 1 アクセスと想定します。)

例 1:

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 30 行/MB/秒 = 300 行 / 1 MB / 10 秒 です。

例 2:

500 KB の表が存在し、1 秒間で 300 行にアクセスする場合、アクセス密度は 600 行/MB/秒 = 300 行 / 0.5 MB / 秒 です。

最初の表より 2 番目の例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。

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ません。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りません。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。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。

この説明では、簡略化のために、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ます。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、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。

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。

付録 B. 最大 BLOB サイズの計算

B.1 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。そのため、インメモリー表は、ページ・サイズより大きい文字ファイルまたはバイナリー・ファイルを格納できません。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。

B.2 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) 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。

B.3 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、概算値が計算されます。以下の表をコピーして、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 5. 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。
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 6. 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 C. ストレージ要件の計算

この付録では、メモリー内またはディスク上に表とその索引を格納する際、どれほどのメモリーまたはディスク・スペースが必要になるかを見積もるための情報を記載しています。

ここに示す公式は厳密なものではありません。例えば、以下のような理由からそういえます。

- 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。

ここに示す公式は、データが圧縮されておらず、かつ最大数のポインターがある場合を想定したものです。したがって、これらの公式を使って得られる結果は余裕のある数値です。つまり、公式では通常、必要なスペース容量を多めに見積もっています。

以下の公式では、sum_of(x) という表記は、各 x のサイズの合計を求めることを意味します。例えば、次のようになります。

- sum_of(col_size) は、表または索引の各列のサイズの合計を求めることを意味します。
- sum_of(index_sizes) は、表の全索引のサイズの合計を求めることを意味します。

C.1 ディスク・ベース表のストレージ要件の計算

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・ベース索引で、*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の 3 倍のサイズが必要になる可能性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は、既にディスク上に存在するページ数と同数のページ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 部 (データベース内のページ用に 1 部、成功した最新のチェックポイント用に 1 部、および実行中の現在のチェックポイント用に 1 部) 存在する可能性があります。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* を大きくすることが必要になる可能性があります。

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、サーバーはサーバー生成の「行番号」を主キーとして使用します。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。

背景情報

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、「 40 ページの『C.3, 列のタイプと列のサイズ』」に示されている、列タイプごとの列のサイズを使用して計算できます。

加えて、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、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 7. 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・ベース表に主キー以外の索引が含まれる場合、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。

以下に例を示します。

まず、次のようにしてディスク・ベース表を作成します。

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

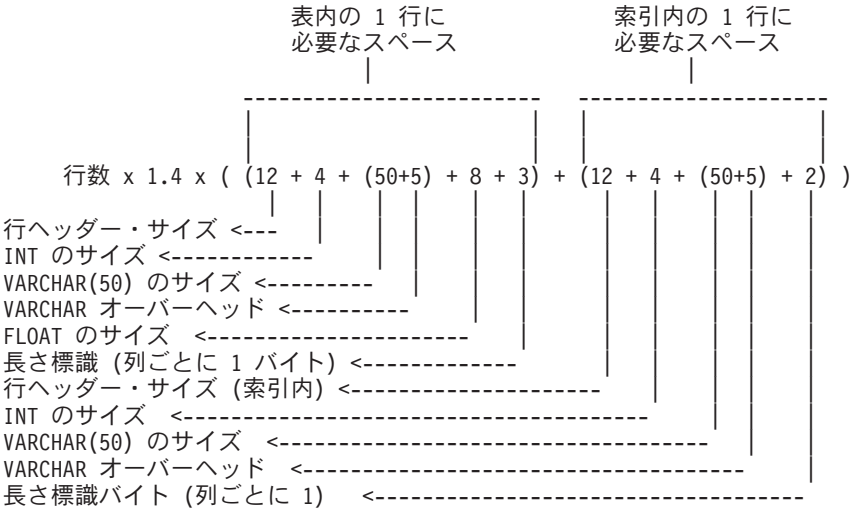
次いで、次のようにして副次索引を作成します。

```
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 と想定した場合のディスク・ベース表の合計サイズは、次のようにして計算できます。

```
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
```

ヒント: 上の計算は、次のように表すこともできます。



C.2 インメモリー表のストレージ要件の計算

インメモリー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。

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)

上記の要素について以下に説明します。

- *rows* は行数です。

- *word_size* はマシンのワード・サイズです (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、この値はキー値の分布によって異なります。キー値が大きく異なるほど、2.0 に近い値を使用してください。キー値がほとんど異ならなければ、1.0 に近い値を使用してください。

背景情報

インメモリー表のストレージ要件を計算する場合、各エントリーのサイズは、表のデータのサイズに、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・オペレーティング・システムでは各 4 バイト、64 ビット・オペレーティング・システムでは各 8 バイト) を合計したサイズになります。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ありません。それは、行ごとの 2 バイトに含まれています。

加えて、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。主キーのコピーは、インメモリー索引には必要ありません。

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。

上の例の表と索引の場合

以下に例を示します。

まず、次のようにしてインメモリー表を作成します。

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

次いで、次のようにして副次索引を作成します。

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

それから、32 ビット・オペレーティング・システムの場合のメモリー使用量を次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。この値は通常、1 と 2 の間です。ランダム値分布では、この値は 2.0 に近くなります。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。インメモリー索引はディスクに書き込まれることはないため、ディスク・スペースを使用しません。ただし、表そのものは、チェックポイント処理中およびサーバーのシャットダウン時にディスクに書き込まれるため、ディスク・スペースの合計容量には、ディスク・ベース表とインメモリー表の両方を格納するための十分なスペースが必要です。

C.3 列のタイプと列のサイズ

表 8. 列のタイプと列のサイズ

列のタイプ	サイズ
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

注: 上の表の値は最大長です。可変長データ (VARCHAR) や圧縮可能なデータの場合は、必要なバイト数がこれよりも少なくなる場合があります。

C.4 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。

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

特記事項

IBM の書面による明示的な許可がある場合を除き、本製品のいかなる部分も、いかなる方法においても使用することはできません。

本製品は、米国特許 6144941、7136912、6970876、7139775、6978396、7266702、7406489、7502796、および 7587429 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。

〒103-8510
東京都中央区日本橋箱崎町19番21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。

本書はプランニング目的としてのみ記述されています。記述内容は製品が使用可能になる前に変更になる場合があります。

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。これらのサンプル・プログラムは特定物として現存するままの状態で提供されるものであり、いかなる保証も提供されません。 IBM は、お客様の当該サンプル・プログラムの使用から生ずるいかなる損害に対しても一切の責任を負いません。

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。

© Copyright IBM Corp. _年を入れる_. All rights reserved.

この情報をソフトコピーでご覧になっている場合は、写真やカラーの図表は表示されない場合があります。

商標

IBM、IBM ロゴおよび ibm.com® は、世界の多くの国で登録された International Business Machines Corp. の商標です。他の製品名およびサービス名等は、それぞれ IBM または各社の商標である場合があります。現時点での IBM の商標リストについては、 http://www.ibm.com/legal/copytrade.shtml をご覧ください。

Java™ およびすべての Java 関連の商標およびロゴは Oracle やその関連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。

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。

SA88-4537-01

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