

Oracle TimesTen
In-Memory Database 7.0
グッド・プラクティス・ガイド

Oracle ホワイト・ペーパー
2007 年7月

Oracle TimesTen In-Memory Database 7.0 グッド・プラクティス・ガイド

1	はじめに	3
1.1	Oracle TimesTen 直接リンク・アプリケーション	3
1.2	Oracle TimesTen クライアント/サーバー・アプリケーション	3
1.3	インストール・ディレクトリ <code>install_dir</code>	4
1.4	C++ユーザーのための <code>TTCclasses</code> ライブラリ	4
2	パフォーマンスの最大化	4
2.1	応答時間が重要なアプリケーションでの直接リンク接続の使用	5
2.2	すべての SQL 文の事前準備	5
2.3	すべての表における統計情報の更新	6
2.4	接続のチューニングと接続プーリングの使用	6
2.5	カーソルの迅速なクローズ	7
2.6	ディスク書き込み頻度の制御	7
2.7	<code>DurableCommit = 1</code> の代替ソリューション	8
2.8	適切な表索引の作成	8
2.9	問合せプランのレビュー	10
2.10	自動コミットの無効化と定期的なコミット	11
2.11	XLA 確認の効果的な使用	11
2.11.1	更新のプリフェッチ	12
2.11.2	更新の確認	12
2.12	Java アプリケーションにおける考慮事項	12
2.13	大規模な表をロードした後の索引の作成	13
3	同時実行性とスケーラビリティのチューニング	13
3.1	適切なメモリー・ログ・バッファ・サイズの設定	13
3.2	トランザクション・ログとチェックポイント・ファイルの分離	13
3.2.1	ディスク速度の問題	13
3.3	適切な RAM ポリシーの使用	14
3.4	高容量の DELETE 文の回避	14
3.5	長期のトランザクションの短縮	14
4	安定性と高可用性の最大化	14
4.1	定期的なチェックポイントの実行	14
4.2	トランザクション・ログの蓄積回避	15
4.3	データベースのバックアップ	15
4.4	高可用性の計画	15

Oracle TimesTen In-Memory Database 7.0 グッド・プラクティス・ガイド

1 はじめに

このドキュメントでは、Oracle TimesTen In-Memory Database を使用して、最大のパフォーマンスと堅牢性を実現するアプリケーションを開発する方法について説明します。このドキュメントの読者は、Oracle TimesTen In-Memory Database の基本機能に関する知識があり、Oracle TimesTen を使用したアプリケーションの計画、中または開発中であることを前提としています。このドキュメントは、Oracle TimesTen の製品ドキュメント・セットを補完するものです。

以下の各項では、Oracle TimesTen アプリケーションにおけるプログラミングの概念と用語について説明します。

1.1 Oracle TimesTen 直接リンク・アプリケーション

"直接リンク・アプリケーション"という用語は、直接リンク接続モードを使用して、Oracle TimesTen データベースに接続するアプリケーションを指します（アプリケーションに組み込まれた Oracle TimesTen データベースなど）。

直接リンク・モードの場合、C および C++アプリケーションは、以下の共有ライブラリのいずれかに明示的にリンクします。

- `libtten.so` (Solaris, Linux)
- `libtten.sl` (HP-UX)
- `tten70.lib` (Windows)

Java アプリケーションで直接リンク・モードを使用する場合、Oracle TimesTen のデータソース接続文字列は、"`jdbc:TimesTen:direct:...`"という形式になります。

1.2 Oracle TimesTen クライアント/サーバー・アプリケーション

この用語は、Oracle TimesTen クライアント/サーバー・インタフェースを使用した、Oracle TimesTen データベースに接続するアプリケーションを指します。

クライアント/サーバー・モードの場合、C および C++アプリケーションは、以下の共有ライブラリのいずれかに明示的にリンクします。

- `libttclient.so` (Solaris, Linux)
- `libttclient.sl` (HP-UX)
- `ttcl170.lib` (Windows)

Java アプリケーションで直接クライアント/サーバー接続モードを使用する場合、Oracle TimesTen のデータソース接続文字列は、"`jdbc:TimesTen:client:...`"という形式になります。

クライアント/サーバー・アプリケーションは、Oracle TimesTen データベースと同じマシン上、または別のマシン上に配置することができます。クライアント/サーバー接続モードを使用する場合、IPC およびネットワークのオーバーヘッドが発

"直接リンク・アプリケーション"とは、直接リンク接続モードを使用して Oracle TimesTen データベースに接続するアプリケーションを指します（アプリケーションに組み込まれた Oracle TimesTen データベースなど）。

生するため、直接リンク・アプリケーションに比べてアプリケーションの応答時間が長くなり、スループットは低下します。

1.3 インストール・ディレクトリ *install_dir*

このディレクトリに、Oracle TimesTen をインストールします。詳しくは、『Oracle TimesTen In-Memory Database インストレーション・ガイド』を参照してください。

デフォルトでは、このディレクトリは以下のとおりになります。

- `/opt/TimesTen/InstanceName` (UNIX、root ユーザーでインストールされたインスタンスの場合)
- `$HOME/TimesTen/InstanceName` (UNIX、root ユーザー以外でインストールされたインスタンスの場合)
- `C:\¥TimesTen¥ttt70` (Windows の場合)

Oracle TimesTen インスタンスは、それぞれの運用環境に応じて、root ユーザーでもその他のユーザーでもインストールを実行できます。

1.4 C++ユーザーのための *TTClasses* ライブラリ

TTClasses と呼ばれる Oracle TimesTen の C++インタフェース・クラスは、Oracle TimesTen データベースに対して、高パフォーマンスの使いやすいインタフェースを提供します。この C++ライブラリは、以下を含むもっとも一般的な ODBC 機能の"ラッパー"を提供します。

- SQL 問合せの実行
- イベント通知 (XLA)
- システム・カタログ情報

また、TTClasses の設計には、このドキュメントに記載されている推奨プラクティスの多くが組み込まれています。TTClasses は、Oracle TimesTen In-Memory Database に同梱されています。TTClasses インタフェースの説明については、『Oracle TimesTen In-Memory Database TTClasses ガイド』を参照してください。

Oracle TimesTen 製品のインストールには、TTClasses の使用法を示す便利なサンプル・プログラムが多く含まれます。サンプル・プログラムには、以下の使用法を示すプログラムが含まれます。

- Oracle TimesTen データソースの監視方法と管理方法
- Oracle TimesTen XLA を使用したイベント通知プログラムの実装方法
- データソース・サイズの見積り方法
- XLA ブックマークの管理方法

2 パフォーマンスの最大化

この項では、Oracle TimesTenアプリケーションのコーディング・プラクティスを推奨することで、最適なパフォーマンスを達成します。この項に含まれる情報は、『[Oracle TimesTen In-Memory Databaseオペレーション・ガイド](#)』の"データ・ストアのパフォーマンス・チューニング"の章と組み合わせて使用してください。

TTClasses と呼ばれる Oracle TimesTen の C++インタフェース・クラスは、Oracle TimesTen データベースに対して、高パフォーマンスの使いやすいインタフェースを提供します。

2.1 応答時間が重要なアプリケーションでの直接リンク接続の使用

直接リンク・アプリケーションでは、データベース操作がアプリケーション・プロセスのアドレス空間から直接実行されるため、プロセス間通信やネットワーク・ラウンドトリップによるオーバーヘッドが発生しません。このため、クライアント/サーバー・アプリケーションと比べると大幅なパフォーマンスの向上を実現できます。

Oracle Database からキャッシュされたデータへアクセスするアプリケーションで直接リンク接続を使用すると、Oracle Database からキャッシュされていないデータの場合と同様に、アプリケーション・パフォーマンスが向上します。

アプリケーションを Oracle TimesTen データベースと同じマシン上で実行しており、Oracle TimesTen データベースに対するアプリケーションのアクセス方法を制御できる場合、直接リンク接続モードを使用することを推奨します。直接リンク・アプリケーションでは、データベース操作がアプリケーション・プロセスのアドレス空間から直接実行されるため、プロセス間通信やネットワーク・ラウンドトリップによるオーバーヘッドが発生しません。このため、クライアント/サーバー・アプリケーションと比べると大幅なパフォーマンスの向上を実現できます。

Oracle TimesTen はマルチプロセス・アプリケーションおよびマルチスレッド・アプリケーションをサポートしているため、直接リンク接続モードおよびクライアント/サーバー接続モードを使用している複数のプロセス間で、Oracle TimesTen データベースを共有することができます。

JDBC を使用して Oracle TimesTen データベースにアクセスする Java アプリケーションの場合、直接リンク接続モードを使用することは簡単です。使用方法は、JDBC 接続文字列を"jdbc:TimesTen:direct:..."の形式に変更するだけです。

Oracle Database からキャッシュされたデータへアクセスするアプリケーションで直接リンク接続を使用すると、Oracle Database からキャッシュされていない Oracle TimesTen データの場合と同様に、アプリケーション・パフォーマンスが向上します。直接リンク接続を使用できないアプリケーションの場合も、クライアント/サーバー接続を使用することで、従来のディスクベースのリレーショナル・データベースへのアクセスと比べて大幅なパフォーマンスの向上が実現できます。これは、おもに、Oracle TimesTen 製品がインメモリー・データの構造とレイアウト用に設計および最適化されているためです。

以下のような場合、一般的なガイドラインとして、アプリケーションはクライアント/サーバー・モードで Oracle TimesTen データ・ストアに接続することが推奨されます。

- Oracle TimesTen データベースを配置したホストとは異なるホスト上で、アプリケーションを実行する必要がある場合。
- 32 ビットのクライアント・アプリケーションから 64 ビットの Oracle TimesTen データベースに接続する必要があり、32 ビットのクライアント・アプリケーションを 64 ビットで再コンパイルできない場合。アプリケーションが Oracle TimesTen データ・ストアと同じホスト上に配置されているか、またはそのように再配置できるものの 64 ビット・モードで再コンパイルできない場合は、Oracle TimesTen の共有メモリー・プロセス間通信 (SHMIPC) を使用することが推奨されます。SHMIPC を利用すると、TCP/IP 経由のクライアント/サーバー・データベース接続で、大幅なパフォーマンス向上が実現されます。

多数の Oracle TimesTen クライアント/サーバー接続が必要となるシステムを設計する場合、それぞれのデータ・ストア接続によりサーバー・ホスト上のオペレーティング・システム・リソースが消費されることに注意する必要があります。サーバー・ホストのサイジングやオペレーティング・システム・チューニングを行う場合、この要素を考慮に入れてください。

2.2 すべての SQL 文の事前準備

最適なパフォーマンスを実現するには、複数回実行する SQL 文を事前に準備しておく必要があります。これはすべてのリレーショナル・データベースに当てはまることですが、Oracle TimesTen とそのきわめて高速なトランザクション速度において、SQL 文のコンパイル時間が実行時間の何倍にもなる場合があります。

最適なパフォーマンスを実現するには、複数回実行する SQL 文を事前に準備しておく必要があります。

SQL 文の事前準備に加えて、これらの SQL 文の入力パラメータや出力列を事前にバインドしておく必要があります。

- ODBC を使用している場合、*SQLPrepare* 機能を使用して SQL 文を事前に準備します。
- *TTClasses* を使用している場合、*TTCmd::Prepare* メソッドを使用して SQL 文を事前に準備します。
- JDBC を使用している場合、*PreparedStatement* クラスを使用して SQL 文を事前に準備します。

もう 1 つのグッド・プラクティスとして、表の統計情報を更新した後に、問合せを準備することが推奨されます。

2.3 すべての表における統計情報の更新

通常、Oracle TimesTen の問合せオブティマイザは、適切な問合せプランを選択します。しかし、複雑な問合せに対して適切なプランを選択するには、その問合せに含まれる表に関する追加情報が必要になります。

表の統計情報は、オブティマイザが適切なプランを選ぶ上で重要な手助けとなります。表について、行数や列の値のデータ分布が分かっている場合、オブティマイザがその表にアクセスするための効率的な問合せプランを選ぶ可能性が大幅に高まります。

表にアクセスする問合せを準備する前に、データベース内すべての表の統計情報を更新することが推奨されます。

表にデータを移入する前に統計情報を更新すると、その表には行がまったく（または、ほとんど）含まれないという前提に基づいて問合せが最適化されます。後でこの表に何百万行も移入して問合せを実行すると、ほとんど行を含まない表では有効であったプランのパフォーマンスが著しく低下する場合があります。これは、その表の現在のデータ・セットに対して最適化が実行されていないためです。

表に対して多数の行を追加または削除したら、パフォーマンスを最適化するために、この表の統計情報を更新して現在の行数を反映させる必要があります。表の統計情報を更新したら、問合せの準備を行ってください。

統計情報の更新についての詳細は、[『Oracle TimesTen In-Memory Database API リファレンス・ガイド』](#)の"[組込みプロシージャ](#)"の章に記載されている組込みプロシージャ `ttOptUpdateStats` および `ttOptEstimateStats` を参照してください。

2.4 接続のチューニングと接続プーリングの使用

マルチスレッド・アプリケーションを使用しており、同じ Oracle TimesTen データベースに対して複数の接続を開いている場合、接続に留意し、慎重な管理を行う必要があります。インメモリー・データベースである Oracle TimesTen は、使用可能なプロセッサ・リソースを効率的に使用しますが、通常は CPU の制約を受けません（ディスクベースの RDBMS 上で実行されるアプリケーションの多くはディスクの制約を受けます）。一般的に、マシン上で処理を実行する CPU より多くの同時接続がアクティブな状態にある場合、パフォーマンスを最適化することは難しく、同時接続によりアプリケーション・スループットが低下する可能性があります。

多数の接続を必要とするアプリケーションの場合、トランザクションに対して接続が効率的に保持されるようにする必要があります。この問題を回避する方法の 1 つに、接続プーリングの使用があります。接続プーリングの良い使用例としては、*TTClasses* の *TTConnectionPoolclass* があげられます。また、市販の J2EE アプリケーション・サーバーは、通常、デフォルトで接続プーリングを使用します。

表にアクセスする問合せを準備する前に、データベース内すべての表の統計情報を更新することが推奨されます。

Oracle TimesTen の読取り専用トランザクションは、コミットする必要がありません。ただし、読取り専用の SQL 問合せが保持しているすべてのリソースを解放するために、読取り専用カーソルを直ちにクローズすることが重要です。

DurableCommit = 0 を設定した場合、トランザクションがインメモリー・ログ・バッファにコミットされるとすぐにアプリケーションに制御が戻されるため、トランザクション応答時間が最適化されます。

DurableCommit = 1 を設定した場合、この接続でトランザクションがコミットされるたびに、トランザクション・ログがディスクに書き込まれます。結果として、アプリケーションには完全な永続性がもたらされます。

2.5 カーソルの迅速なクローズ

Oracle Database と同様に、Oracle TimesTen の読取り専用トランザクションはコミットする必要がありません。ただし、読取り専用の SQL 問合せが保持しているすべてのリソース（ソート処理に使用される一時領域など）を解放するために、読取り専用カーソルを直ちにクローズすることが重要です。

SQL カーソルをクローズするには、以下のメソッドを使用します。

- ODBD の場合、`SQLFreeStmt (SQL_CLOSE)` を使用します。
- TTClasses の場合、`TTCmd::Close` を使用します。
- JDBC の場合、`PreparedStatement.close` を使用します。

2.6 ディスク書込み頻度の制御

ほとんどのアプリケーションは、リカバリとシステムの再起動を実現するために、いくらかのデータ永続性とトランザクション永続性を必要とします。たとえば、ハードウェア障害またはソフトウェア障害によるシステムのクラッシュに備え、アプリケーションはデータベースをリカバリ可能にするため、最新の更新を適用する必要があります。以下に、**DurableCommit** 属性を使用して Oracle TimesTen データベースを設定する方法を示します。

- 最初のオプションを利用すると、Oracle TimesTen トランザクション・ログ・マネージャは、アプリケーションのトランザクション・コミット・コールとは非同期に、インメモリー・トランザクション・ログ・バッファからディスクへの書込みを行います。トランザクション・ログ・マネージャのバックグラウンド・ログ・フラッシュは、トランザクション・ログ・データを効率的な方法でディスクへ書き込みます。この永続性保留オプションは、接続属性 **DurableCommit = 0** を設定することで利用できます。このコミット設定により、トランザクションがインメモリー・ログ・バッファにコミットされるとすぐにアプリケーションに制御が戻されるため、トランザクション応答時間が最適化されます。ただし、この機能を利用するアプリケーションには、システム障害の発生時にいくらかの脆弱性が生じます。この脆弱性により、最大でインメモリー・トランザクション・ログ・バッファのサイズと同じトランザクション量を失う可能性があります。
- 2 番目のオプションでは、接続属性 **DurableCommit = 1** を設定して、Oracle TimesTen データベースに接続します。この場合、この接続でトランザクションがコミットされるたびに、トランザクション・ログがディスクに書き込まれます。結果として、アプリケーションには完全な永続性がもたらされます。ただし、この設定によりアプリケーションの書込みスループットが大幅に低下する場合があります。これは、トランザクションがコミットされるたびに、トランザクション・データがディスクに書き込まれるのを待つ間、制御が戻されるまでアプリケーションがブロックされるためです。アプリケーションに多数の接続を設定して、データベースに同時書込みを行うようにすると、スループットを向上させることができます。

Oracle TimesTen は、永続性の制御に対して柔軟なアプローチを提供しています。アプリケーションは、接続レベルおよび/またはトランザクション・レベルで永続性設定を指定できます。また、アプリケーションから組み込みプロシージャ **ttDurableCommit** を呼び出すことで、いつでもトランザクション・データをディスクに"書き込む"ことができます。アプリケーションによっては、データの脆弱性を一定時間内に留めるために、特定の時間間隔で **ttDurableCommit** を呼び出します。シングル・ポイント障害の発生時にもデータ損失ゼロを保証する必要があるアプリケーションの場合、**DurableCommit = 1** の代替ソリューションを検討します。詳しくは、次の項の説明を参照してください。

Oracle TimesTen Replication 製品オプションと Cache Connect to Oracle 製品オプションを使用すると、**DurableCommit** 設定に関係なく、トランザクションの永続性と可用性を向上できます。

2.7 DurableCommit = 1 の代替ソリューション

シングル・ポイント障害の発生時にも、データ損失ゼロを保証する必要があるアプリケーションもあります。しかし、すべてのトランザクションを永続的にコミットした場合、トランザクションが発生するたびにディスクが I/O を完了する速度の影響を受けるため、スループットと応答時間の目標を達成できない可能性があります。トランザクション・パスにディスク I/O が含まれる場合、トランザクション負荷のピーク時に応答時間の一貫性を維持できない可能性があります。品質保証契約 (SLA) により、事前定義されたアプリケーション応答時間を達成する必要がある場合、**DurableCommit = 1** に対する代替ソリューションが必要になります。

永続コミットを使用する代わりに、2 つの Oracle TimesTen ノード間で 2-SAFE Replication を使用すると、データ損失ゼロが保証されます。

永続コミットを使用する代わりに、2 つの Oracle TimesTen ノード間で 2-SAFE Replication を使用することができます。2-SAFE を使用すると、アプリケーションに制御が戻される前に、プライマリ・ノードとスタンバイ・ノードの両方のメモリー上にトランザクションがコミットされます。このため、シングル・ポイント障害の発生時にもデータ損失ゼロが保証されます。データ整合性が保証されることに加えて、トランザクション・パスにディスク I/O が含まれる場合は、達成不可能であった一貫性のある応答時間を実現できます。

2-SAFE Replication の設定および使用方法については、『Oracle TimesTen Replication - TimesTen to TimesTen 開発者および管理者ガイド』を参照してください。

2.8 適切な表索引の作成

いくつかの索引を作成すると優れたデータベース・パフォーマンスを実現できるのかについて判断することは、少し複雑な問題です。索引が少なすぎると、頻繁に実行されるデータ操作のパフォーマンスが通常より低下する場合があります。索引が多すぎると、索引の更新に余分な時間がかかるため、INSERT/UPDATE/DELETE 操作が遅延する可能性があります。

Oracle TimesTen の表および索引スキーマを設計する際、設計上の考慮事項がいくつかあります。

Oracle TimesTen がサポートする索引には、ハッシュ索引と T ツリー索引の 2 種類があります。

- Oracle TimesTen がサポートする索引には、ハッシュ索引と T ツリー索引の 2 種類があります。完全一致に関して言えば、適切にチューニングされたハッシュ索引は対応する T ツリー索引よりも高速です。しかし、ハッシュ索引は範囲検索には使用できません (T ツリー索引は、完全一致と範囲検索の両方に使用でき、ORDER BY、GROUP BY、DISTINCT などの SQL 問合せのソートにも使用できます)。
- 主キーに対して一意のハッシュ索引を指定していない場合、T ツリー索引が自動的に作成されます (Oracle TimesTen 7.0 以降のリリースが該当します)。
- 一意のハッシュ索引を指定する場合、適切な索引サイズを設定してください。CREATE TABLE 文で "PAGES=" オプションを使用して、予想される表サイズを指定します。指定するページ数を算出するには、その表に予想される行数を 256 で割ります。以下に例を示します。

```
CREATE TABLE EMP (  
    ID          NUMBER NOT NULL PRIMARY KEY,  
    NAME        VARCHAR2(100)  
) UNIQUE HASH ON (ID) PAGES=500;
```

どちらの索引でも使用できる場合、等価条件に関しては、ハッシュ索引のパフォーマンスが T ツリー索引を上回ります。

- どちらの索引でも使用できる場合、等価条件に関しては、ハッシュ索引のパフォーマンスが T ツリー索引を上回ります。ただし、ハッシュ索引は T ツリー索引よりも多くの領域を必要とします。索引に指定するページが多すぎると、領域の浪費につながります。ページが少なすぎると、ハッシュ・バケットのオーバーフローによりパフォーマンスが低下します。不適切なサイズが指定されたハッシュ索引を使用すると、ツリー索引よりも大幅にパフォーマンスが低下する場合があります。
- 表のサイズが大幅に増加した場合、ハッシュ索引のサイズを変更する必要があります。

あります。実行するには、ALTER TABLE 文を使用します。

- T ツリー索引を含まない表に対して全表スキャンを実行する場合、T ツリー索引（任意の T ツリー索引）を作成するとパフォーマンスが向上します。これは、表スキャンがこの索引に含まれる列を参照しない場合にも当てはまります。この効果は直感的には理解しにくいものですが、簡単に実証できます。したがって、アプリケーションの全表スキャンで参照されるすべての表に対して、少なくとも 1 つの T ツリー索引を作成することが推奨されます。

簡単な問合せ例を使用して、いくつかの索引シナリオを確認します。

```
SELECT ...FROM T1 WHERE COL1 = ?AND COL2 = ?
```

シナリオ 1

T1 上には、2 つの索引が作成されています。

- ハッシュ索引：(COL1、COL2)
- T-ツリー索引：(COL1、COL2、COL3)

この場合、どちらの索引を使用しても、この問合せに回答できます。

ハッシュ索引が使用できるのは、WHERE 句に含まれる列とハッシュ索引に含まれる列が完全に一致するためです。T ツリー索引が使用できるのは、WHERE 句に含まれる列がこの索引の先頭部分（最初の 2 つ）の列と一致するためです。

Oracle TimesTen オプティマイザは、より高速であるハッシュ索引を選択します。

シナリオ 2

T1 上には、2 つの索引が作成されています。

- ハッシュ索引：(COL1、COL2、COL3)
- T-ツリー索引：(COL1、COL2)

この場合、この問合せの回答に使用できるのは T ツリー索引のみです。

ハッシュ索引に含まれる列のうちの 1 つ (COL3) が問合せの WHERE 句に指定されていないため、この索引を使用することはできません。

T ツリー索引が付けられた列は、問合せの WHERE 句と完全に一致します。

Oracle TimesTen オプティマイザは、T ツリー索引を選択します。

シナリオ 3

T1 上には、2 つの索引が作成されています。

- ハッシュ索引：(COL1)
- T-ツリー索引：(COL3、COL1、COL2)

この場合、この問合せの回答に使用できるのはハッシュ索引のみです。

ハッシュ索引に含まれるすべての列が問合せの WHERE 句に含まれるため、この索引を使用できます。問合せの WHERE 句には別の列も含まれているため、問合せの結果が返される前に、ハッシュ索引から読み取られたすべての行に対して、"COL2 = ?"条件が適用されます。

T ツリー索引は最初に COL3 でソートされており、この問合せは COL3 を参照しないため、この索引は使用できません。

Oracle TimesTen オプティマイザは、ハッシュ索引を選択します。

シナリオ 4

T1 上には、2 つの索引が作成されています。

ハッシュ索引：(COL1、COL2、COL3)

T ツリー索引：(COL3、COL1、COL2)

この場合、どちらの索引もこの問合せの回答に効果的ではありませんが、T ツリー索引を使用して全表スキャンを実行することはできます。

ハッシュ索引は、シナリオ 2 と同じ理由により使用できません。また、問合せに含まれる列 (COL1、COL2) が T ツリー索引の先頭部分と一致しないため、T ツリー索引は使用できません。

どちらの索引も使用できないので、問合せに回答するために、SQL エンジンでは表スキャンを実行する必要があります。この結果、問合せのパフォーマンスは不十分なものになります。必要に応じて一時索引が作成される場合もありますが、この場合もパフォーマンスは不十分なままです。

これらの例から分かるように、実行する問合せの頻度に応じて、どの索引を作成するかを慎重に選択する必要があります。

適切な索引の作成について詳しくは、『Oracle TimesTen In-Memory Database オペレーション・ガイド』の、「文のチューニングと索引の使用」および「ハッシュ索引または T ツリー索引の適切な選択」の項を参照してください。

実行する問合せの頻度に応じて、どの索引を作成するかを慎重に選択する必要があります。

2.9 問合せプランのレビュー

特定の問合せのパフォーマンスが予想よりも悪い場合、Oracle TimesTen オプティマイザは問合せに回答するための最適な問合せプランを選択していない可能性があります。問合せプランを生成し、レビューを行う必要があります。問合せプランの生成方法とプランのレビュー方法について詳しくは、『Oracle TimesTen In-Memory Database オペレーション・ガイド』を参照してください。

特定の問合せのパフォーマンスが予想よりも悪い場合、Oracle TimesTen オプティマイザが、問合せに回答するための最適な問合せプランを選択していない可能性があります。問合せプランを生成し、レビューを行う必要があります。

ttlsql コマンドライン・ユーティリティでオプティマイザ・プランを表示するには、次のコマンドを使用します。

```
autocommit 0;
showplan 1;
```

問合せプランをレビューする際、問合せの評価に関与していて索引付けされていない条件に注目します。索引付けされていない条件に対して索引を作成できる場合、問合せパフォーマンスが向上します。簡単な変更により、いかに問合せパフォーマンスが大きく向上するかについて、次の例を確認します。

以下の例では、メインの表 (SHIPMENT) が比較的大きい問合せのテストから、一部のステップを抽出しています。この表には 1000 万を超える行が含まれます。最初に問合せを実行した際、完了までに約 800 秒かかりました。これは期待通りのパフォーマンスではなく、過度に遅いものでした。次に、問合せプランを生成すると、あるステップに次のプランが含まれることが分かりました。

```
STEP:          10
LEVEL:         8
OPERATION:     TblLkTtreeScan
TBLNAME:       SHIPMENT
IXNAME:        SHIPMENT_IDX1
INDEXED:       <NULL>
NOT INDEXED:   TBL1.OB_FLG <> 'Y'
               AND TBL1.SHIPMENT_ID = CLIENT.SHIPMENT_ID
               AND TBL1.SHIPMENT_QUAL =
                   CLIENT.SHIPMENT_QUAL
               AND TBL1.CARRIER = CLIENT.CARRIER
               AND TBL1.ROLE = 'Y'
```

"NOT INDEXED" と示された個所に複数の条件が記載されており、実際、"INDEXED" の行に記載されている条件はありませんでした。SHIPMENT_ID、SHIPMENT_QUAL、および CARRIER 列には索引が作成されていなかったため、オプティマイザはこれらの条件を評価するために全表スキャンを行う必要がありました。これら 3 つの列に索引を作成することで問題を解決した結果、以下に示すとおり、ステップ 10 のプランが大幅に改善されました。

```

STEP:          10
LEVEL:         8
OPERATION:     RowLkTtreeScan
TBLNAME:       SHIPMENT
IXNAME:        SHIPMENT_IDX0
INDEXED:       TBL1.SHIPMENT_ID = CLIENT.SHIPMENT_ID
                AND TBL1.SHIPMENT_QUAL =
CLIENT.SHIPMENT_QUAL
                AND TBL1.CARRIER = CLIENT.CARRIER
NOT-INDEXED:   TBL1.OB_FLG <> 'Y' AND TBL1.ROLE = 'Y'

```

プランは、大幅に改善されました。オプティマイザが、条件の評価に適した索引を使用して行ロックを選択した結果、400 倍の問合せパフォーマンスを達成しました。変更済みの問合せの実行にかかった時間は 2 秒でした (変更前は 800 秒)。

すべてのパフォーマンス・チューニングと同様に、その効果もさまざまです。ここで重要なのは、問合せパフォーマンスを改善するために問合せプランを確認し、必要な変更を行う時間を確保することです。

2.10 自動コミットの無効化と定期的なコミット

自動コミットを無効化し、複数の SQL 文を 1 つのトランザクション内で明示的にコミットすることで、アプリケーション・パフォーマンスを大幅に向上できます。

ODBC および JDBC では、デフォルトですべてのデータベース接続に自動コミットが有効化されています。これは、それぞれの SQL 文が個別のトランザクション内でコミットされることを意味します。自動コミットを無効化し、複数の SQL 文を 1 つのトランザクション内で明示的にコミットすることで、アプリケーション・パフォーマンスを大幅に向上できます。これは、バルク挿入やバルク削除などの大量処理で特に有効です (TTClasses では、デフォルトで自動コミットが無効化されています)。

しかし、過度に多くの処理を 1 つのトランザクションにまとめると、それぞれのトランザクションでロックが維持される時間が長くなるため、システムの同時実行性が大幅に低下する可能性もあります。一般的に、挿入、更新、および削除のバルク処理は、数千行ごとにコミットするとともに効果を発揮します。

2.11 XLA 確認の効果的な使用

XLA 確認インタフェースは、アプリケーションがメッセージを受信するだけでなく、正しく処理できるように設計されています。更新を確認すると、アプリケーションの XLA ブックマークは、最後に読み取られたレコードにリセットされます。これにより、以前に返されたレコードの再読取りが回避されるため、アプリケーションが XLA に再接続したときにブックマークが再利用されても、以前に確認したレコードを受信することはなくなります。

JMS/XLA では、XLA 更新メッセージを自動的に確認することも、明示的にメッセージを確認するようにアプリケーションで選択することもできます。セッションを作成する際に、更新の確認方法を指定します。JMS/XLA は、次の 3 つの確認モードをサポートしています。

- AUTO_ACKNOWLEDGE - 更新を受信すると、自動的に確認します。それぞれのメッセージは一度だけ配信されます。AUTO_ACKNOWLEDGE モードでは、JMS/XLA は複数レコードのプリフェッチは行いません (トピック内の `xlaprefetch` 属性は無視されます)。
- DUPS_OK_ACKNOWLEDGE - 更新を自動的に確認しますが、アプリケーションの障害時に重複メッセージが配信される場合があります。JMS/XLA は、トピックに指定された `xlaprefetch` 属性に従ってレコードをプリフェッチし、プリフェッチされたブロック内の最終レコードが読み取られると確認を送信します。プリフェッチされたすべてのレコードを読み取る前にアプリケー

ションに障害が発生すると、アプリケーションの再起動時にブロック内のすべてのレコードが渡されます。

CLIENT_ACKNOWLEDGE-アプリケーションは、MapMessage に対して *acknowledge* を呼び出すことにより、更新メッセージの受信を確認します。JMS/XLA は、トピックに指定された *xlapbrefetch* 属性に従ってレコードをプリフェッチします。

2.11.1 更新のプリフェッチ

複数の更新レコードを同時にプリフェッチすると、各更新レコードを XLA から個別に取得するよりも効率的です。可能であれば、アプリケーションで重複更新が容認されるように設計すると、DUPS_OK_ACKNOWLEDGE を使用するか、または明示的に更新を確認できます。

2.11.2 更新の確認

XLA 更新の確認を適切に制御するには、更新メッセージに対して明示的に *acknowledge* を呼び出します。メッセージを暗黙的に確認すると、以前のメッセージがすべて確認されます。CLIENT_ACKNOWLEDGE モードを使用しており、将来、永続サブスクリプションを再利用する予定がある場合は、終了する前に、*acknowledge* を呼び出してブックマークを最終読み取り位置にリセットすることが推奨されます。

2.12 Java アプリケーションにおける考慮事項

Oracle TimesTenのネイティブAPIは、ODBCです。Oracle TimesTenのODBCドライバを使用するCまたはC++アプリケーションでは、Oracle TimesTenのJDBCドライバを使用するJavaアプリケーションよりも、若干優れたパフォーマンスが得られます。ただし、Oracle TimesTen直接リンクJDBCドライバを使用するアプリケーションでは、プロセス間通信やネットワーク・ラウンドトリップによるオーバーヘッドが発生しないため、クライアント/サーバー接続モードで使用されるそのほかほとんどのJDBCドライバよりも大幅な高速化が実現されます。

JDBC パフォーマンスは、実行される問合せの種類（SELECT または UPDATE/INSERT/DELETE）と、トランザクションの組合せ（読み取りまたは書き込み）の影響を受けます。以下に、パフォーマンスに影響を与える要素をあげます。

- 準備された文のバインド・パラメータの数：バインド・パラメータの数を増やすと、Java アプリケーションのパフォーマンスが低下する傾向にあります。
- SELECT 文により返される列の数：SELECT の対象となる列数を増やすと、Java アプリケーションのパフォーマンスが低下する傾向にあります。
- パラメータと列の種類：バイト数の多いデータ型は、一般的に、JDBC のパフォーマンスを低下させます。

基本的に、アプリケーションとデータベース間で転送されるデータ量が増えると、Java アプリケーションのパフォーマンスは低下します。

次に、Java アプリケーションのパフォーマンスに影響を与えるその他の Java 固有要素をあげます。

- JVM ガベージ・コレクション：JVM ガベージ・コレクションの実行中、Java アプリケーションの応答時間が大幅に悪化する場合があります。このガベージ・コレクションにより、高パフォーマンスの"リアルタイム"アプリケーションで応答時間を予測することが難しくなります。JVM ガベージ・コレクションを慎重にチューニングすることが、潜在的なスループットの低下と応答時間の悪化を最小限に抑えるために不可欠です。
- Java の実行時メモリー管理：Java の実行時メモリー管理サブシステムの設計により、隠しデータのコピーが発生させる場合があります。また、アプリケーションに十分な Java ヒープ領域を確保する必要があります。

複数の更新レコードを同時にプリフェッチすると、各更新レコードを XLA から個別に取得するよりも効果的です。

JDBC パフォーマンスは、実行される問合せの種類（SELECT または UPDATE/INSERT/DELETE）と、トランザクションの組合せ（読み取りまたは書き込み）の影響を受けます。

- アプリケーションのスループット要件やスケーラビリティ要件によっては、複雑なマルチスレッドの Java アプリケーションのパフォーマンス・チューニングや、複数の JVM の使用を試してみる必要があります。

特定のプラットフォームにおいては、一部の JVM のパフォーマンスが上回ることもあります。使用するプラットフォーム上で各種 JVM を実行して、各オプションを評価してください。

大量の結果セットが返される場合は、SELECT 文に MAXROWS オプションを使用します。

2.13 大規模な表をロードした後の索引の作成

アプリケーション設計で容認される場合は、データのロード後に T ツリー索引を作成することで、データ・ロードにかかる時間を最小化できます。この場合、操作の順番は以下のとおりになります。

1. 表にデータをロードする (必要に応じて、ロギングを無効化し、データベース・レベルのロックを使用すると、表ロード操作のパフォーマンスが向上します)。
2. T ツリー索引を作成する
3. 統計情報を更新する
4. 問合せを準備する

これは、大量データ・ロードの処理時にのみ適切です。

3 同時実行性とスケーラビリティのチューニング

ここでは、SMP マシン上で実行されるアプリケーションが、Oracle TimesTen を使用して、最適なアプリケーション応答時間とスループットを達成できるようにするためのチューニングのヒントについて説明します。

3.1 適切なメモリー・ログ・バッファ・サイズの設定

Oracle TimesTen トランザクション・ログのバッファ・サイズは、デフォルトで 64MB です。このサイズは、アプリケーション要件に合わせて調整できます。書込みの多いアプリケーションの場合、メモリー・ログ・バッファのサイズを大きくすると、ログ競合が減り、ログ・バッファの待機を回避できます。システム表 MONITOR の LOG_BUFFER_WAIT の値を監視し、必要に応じてログ・バッファ・サイズを増加します。詳しくは、『Oracle TimesTen In-Memory Database オペレーション・ガイド』の「システムとデータ・ストアのチューニング」の章を参照してください。

3.2 トランザクション・ログとチェックポイント・ファイルの分離

書込みの多いアプリケーションの場合、トランザクション・ログとチェックポイント・ファイルを別々のディスクに配置することで、I/O 競合を減らすことができます。アプリケーションのリクエストに対して、一定の応答時間とスループットを確保する必要がある場合、これは特に重要です。チェックポイント・ファイルが同じディスク・スピンドル上に配置されている場合、チェックポイント処理は、同じディスク上で行われるトランザクション・ロギング処理と競合します。

3.2.1 ディスク速度の問題

高速ディスクは、アプリケーションの更新スループットの向上に役立ちます。アプリケーションが INSERT、UPDATE、および DELETE 操作を実行すると、ログ・レコードが生成されることに注意してください。また、ログ・レコードは、AutoRefresh を使用して Oracle Database から読取り専用データをキャッシュしている場合にも生成されます。Cache Connect エージェントが Oracle Database からの更新データで Oracle TimesTen データベースをリフレッシュする際、Oracle TimesTen のキャッシュ表に更新が適用されるため、リフレッシュ処理によりログ・レコー

データのロード後に T ツリー索引を作成することで、データ・ロードにかかる時間を最小化できます。

書込みの多いアプリケーションの場合、トランザクション・ログとチェックポイント・ファイルを別々のディスクに配置することで、I/O 競合を減らすことができます。

ドが生成されます。高速ディスクを使用すると、Oracle TimesTen のログ・マネージャがディスク書込みを迅速に完了できるため、ログ・バッファが使用可能になるまでアプリケーションが待機する可能性が少なくなります。I/O 競合を確認するには、システム表 MONITOR の LOG_BUFFER_WAIT を監視します。

3.3 適切な RAM ポリシーの使用

デフォルトでは、Oracle TimesTen データベースは、最後のデータベース接続が切断されるとメモリーからアンロードされます。データベースのアンロードとリロードを頻繁に行うと、不要な接続オーバーヘッドが増加します。データベースのアンロードとリロードを頻繁に実行しないようにするために、最後の接続が終了した後もデータベースがロードされたままになるように、ttAdmin ユーティリティを使用して RAM ポリシーを設定することができます。詳しくは、『Oracle TimesTen In-Memory Database オペレーション・ガイド』の"接続オーバーヘッドの回避"の章を参照してください。

3.4 高容量の DELETE 文の回避

非常に多数のレコードを1つのトランザクション（または1つの SQL 文）で削除すると、システム全体のパフォーマンスに悪影響を与える場合があります。

非常に多数のレコードを1つのトランザクション（または1つの SQL 文）で削除すると、システム全体のパフォーマンスに悪影響を与える場合があります。

- 大量データを削除すると、多数のログ・レコードが生成されるため、多量の I/O が発生します。トランザクションをロールバックする必要が生じた場合のために、すべての削除レコードはログに残されます。
- 大量削除トランザクションは、システム内のその他の同時操作のパフォーマンスを低下させます。
- また、ターゲット表がレプリケーションの対象になっている場合、トランザクション・ログ・データをサブスクライバ・ノードに転送する必要もあります。これは、Oracle TimesTen Replication が、コミットされたトランザクションをサブスクライバ・ノードに転送する物理レプリケーションを使用しているためです。

DELETE FROM を使用する代わりに、TRUNCATE TABLE または DELETE FIRST の使用を検討してください。

リソース消費を削減するために、DELETE FROM 文の代わりに以下の代替策を使用することを検討してください。

- 表内のすべてのレコードを削除する場合、TRUNCATE TABLE を使用します。
- DELETE FIRST を使用して、トランザクション内のレコード数を分割すると、システムの同時実行性を高めることができます。

3.5 長期のトランザクションの短縮

長時間実行トランザクションは、ロック・タイムアウト・エラーによりその他の処理を失敗させる場合があります。これは、長期のトランザクションで使用するリソースが長時間ロックされたままになるためです。一般的に、長期のトランザクションはシステム全体の同時実行性とアプリケーション・パフォーマンスを低下させます。

4 安定性と高可用性の最大化

すべてのデータベース管理システムにとって、運用計画とシステム計画は重要であり、Oracle TimesTen の場合も同様です。ここでは、Oracle TimesTen データベースの管理を向上させるために実行すべき重要な処理について説明します。

4.1 定期的なチェックポイントの実行

チェックポイント処理が定期的に行われないと、データベースのリカバリにかかる時間が長くなります。

チェックポイント・ファイルには、Oracle TimesTen In-Memory Database のディスク・イメージのスナップショットが含まれています。チェックポイント・ファイルは、既存の Oracle TimesTen データベースをメモリーにリロードする際に使用されます。チェックポイント処理が定期的に行われないと、チェックポイント・ファイルに含まれていないトランザクションをトランザクション・ログ・ファイ

ルから適用しなければならぬため、データベース・リカバリにかかる時間が長くなります。ログから適用する必要のあるトランザクションが多ければ多いほど、メモリーへのデータベースのリロードに時間がかかります。通常のシステム停止やデータベースのアンロード処理において、Oracle TimesTen は"最終"チェックポイント処理を実行し、現在のデータベース・イメージをディスクに書き込みます。既存の Oracle TimesTen データベースをもっとも速くリロードする方法は、"最終"チェックポイント・ファイルからデータベースを再起動することです。

デフォルトで設定されているチェックポイント頻度は、10分(600秒)ごとです。チェックポイント頻度を変更するには、組込みプロシージャ **ttCkptConfig**、または接続属性 **CkptFrequency** を使用します。時間による頻度の設定に加えて、ログ容量を使用したチェックポイント設定が行えます。アクティビティが頻繁に急増するアプリケーションの場合、ログ容量を使用してチェックポイント頻度を設定することが理想的です。アプリケーションのワークロードとトランザクション速度に精通している場合、個別の要件に合わせてチェックポイント頻度を調整することができます。そうでない場合は、デフォルトで設定されたチェックポイント頻度を利用することから始めます。

4.2 トランザクション・ログの蓄積回避

トランザクション・ログを監視し、ログ・ファイルを定期的に消去することで、大量のログ・ファイルが蓄積しないようにする必要があります。以下のいずれかの状況に当てはまる場合、トランザクション・ログ・ファイルが蓄積します。

- 長期間にわたって、チェックポイント処理が実行されていない場合。両方のチェックポイント・ファイルに書き込まれない限り、トランザクション・ログは消去されません。
- トランザクションがレプリケーションによって保持されている場合。レプリケーションによるサブスクリバ・ノードへのトランザクション転送が滞っている場合、トランザクションがリモート・ノードに複製されるまで、トランザクション・ログは消去されません。
- トランザクションが XLA アプリケーションによって保持されている場合。アプリケーションが XLA ブックマークを移動しない限り、トランザクション・ログは消去されません。

定期的なチェックポイント処理を行ってもログ・ファイルが消去されない場合、組込みプロシージャ **ttLogHolds** を実行すると、どの処理がログを保持しているかが確認できます。詳しくは、『Oracle TimesTen In-Memory Database API リファレンス・ガイド』を参照してください。

4.3 データベースのバックアップ

その他すべてのデータベース管理と同様に、Oracle TimesTen データベースを定期的にバックアップすることは、予期しない障害の発生や人為的エラーによりデータベースをリカバリする際に役立ちます。Oracle TimesTen データベースをバックアップするには、Oracle TimesTen の **ttBackup** ユーティリティを使用します。バックアップ・ファイルから Oracle TimesTen データベースをリストアする場合は、**ttRestore** ユーティリティを使用します。

4.4 高可用性の計画

データの可用性がビジネス要件に含まれるアプリケーションの場合、Oracle TimesTen Replication製品オプションの使用を検討してください。この製品オプションは、本番データベースのスタンバイ・レプリカを使用して、シングル・ポイント障害に対処する機能を提供します。詳しくは、『[Oracle TimesTen Replication - TimesTen to TimesTen開発者および管理者ガイド](#)』を参照してください。

データの可用性がビジネス要件に含まれるアプリケーションの場合、Oracle TimesTen Replication の使用を検討してください。



Oracle TimesTen 7.0 グッド・プラクティス・ガイド

2007年7月

著者：Oracle TimesTen 開発チーム

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

海外からのお問合せ窓口：

電話：+1.650.506.7000

ファクシミリ：+1.650.506.7200

www.oracle.com

Copyright © 2007, Oracle. All rights reserved.

本文書は情報提供のみを目的として作成されたものであり、その内容は予告なく変更されることがあります。本文書は、その内容に誤りがないことを保証するものではなく、また、口頭による明示的保証や法律による黙示的保証を含め、商品性ないし特定目的適合性に関する黙示的保証および条件などのいかなる保証および条件も提供するものではありません。オラクル社は本文書に関するいかなる法的責任も明確に否認し、本文書によって直接的または間接的に確立される契約義務はないものとします。本文書はオラクル社の書面による許可を前もって得ることなく、いかなる目的のためにも、電子または印刷を含むいかなる形式や手段によっても再作成または送信することはできません。Oracle、JD Edwards、PeopleSoft、および Siebel は、米国 Oracle Corporation およびその子会社、関連会社の登録商標です。その他の名称はそれぞれの会社の商標です。