SYBASE®
An SAP® Company

インメモリ・データベース・ユーザーズ・ガイド

Adaptive Server® Enterprise

15.7

ドキュメント ID：DC01269-01-1570-01
改訂：2011 年 9 月

Sybase, Inc., One Sybase Drive, Dublin, CA 94568.

目次

第 1 章

インメモリ・データベース

インメモリ・データベースは名前付きキャッシュ (つまり Adaptive Server® メモリ領域) 内ですべて実行され、データやログの保存にディスク記憶領域を使用しません。インメモリ・データベースでは I/O が不要であるため、従来のディスク常駐型データベースよりもより優れたパフォーマンスを実現できます。ただし、インメモリ・データベースはリカバリ用には設計されていません。インメモリ・データベースのトランザクション・ログはディスクではなくキャッシュに書き込まれるため、データベースが失敗するとデータの変更内容がすべて失われます。インメモリ・データベースは、実行時ロールバックのため、およびトリガの起動、遅延モードの更新、複写などのその他の操作のためのトランザクション・ロギングを実行します。

ディスク常駐型データベースは、ディスクに書き込むことで、原子性、一貫性、整合性、持続性 (ACID プロパティとも呼ばれます) のトランザクション・プロパティを保証します。持続性とは、トランザクションがコミットされた後の永続性を意味します。ディスク常駐型としても知られている従来の Adaptive Server データベースは、トランザクションがコミットされるとトランザクション・ログをディスクに書き込むことにより、完全な持続性で動作します。この動作に加え、データ・ページのディスクへの定期的な書き込みも行うことで、コミットされたすべてのトランザクションの持続性を保証します。

インメモリ・データベースはデータやログをディスクに書き込まず、トランザクションの持続性と引き換えにパフォーマンスを向上させます。データベースに障害が発生した場合、インメモリ・データベースは回復できません。サーバの障害や通常の停止にともなうデータのリカバリ性が必要なアプリケーションでは、従来の Adaptive Server データベースの使用を考慮してください。

リラックス持続性をサポートすることで、Sybase® はインメモリ・データベースから得られるパフォーマンスのメリットをディスク常駐型データベースにまで拡張します。ディスク常駐型データベースは、サーバの障害からトランザクションを確実にリカバリできるよう、完全な持続性で動作します。リラックス持続性データベースは、コミットされたトランザクションの完全な持続性と引き換えに、トランザクションの負荷の実行時にパフォーマンスを向上させます。

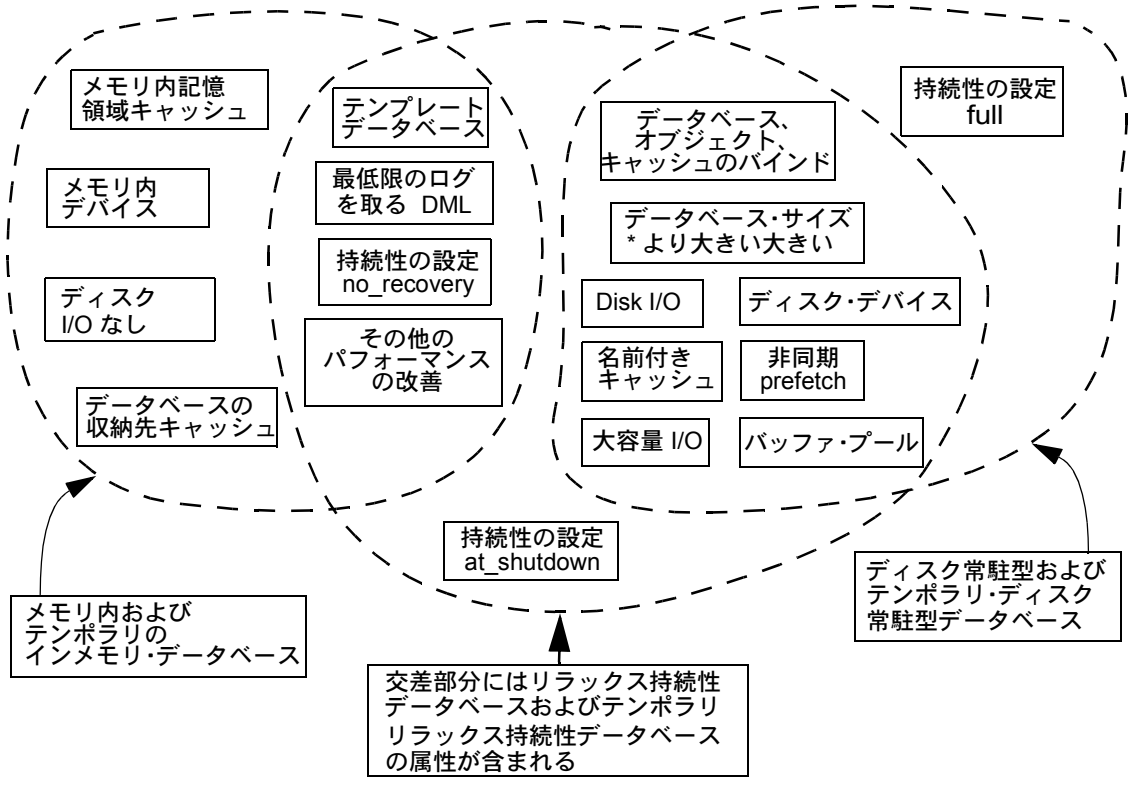
インメモリ・データベースおよびリラックス持続性データベースから得られるパフォーマンスのメリットは次のとおりです。

- インメモリ・データベースは I/O を待機しない。
- バッファおよびユーザ・ログのキャッシュ管理の向上。これにより、Adaptive Server が同じデータに対する更新を同時に実行する場合、ユーザ・ログ・キャッシュの書き込みおよびバッファ管理をオーバーヘッドする必要があります。
- トランザクションのコミットまたはアボート時に、ユーザ・ログ・キャッシュがトランザクション・ログに書き込まれないようにするランタイム方法。これにより、メモリ内ログ・ページの競合が少なくなります。
- メモリ内ロギング手法を使用して大量の DML 操作のパフォーマンスを改善する、最低限のログを取る DML のサポート。

Adaptive Server バージョン 15.5 以降では、次のタイプのデータベース (図 1-1 を参照) を作成できます。

- 持続性レベルが full に設定されているディスク常駐型データベース (デフォルトまたは従来の Adaptive Server)
- ユーザ作成のディスク常駐型テンポラリ・データベース
- 持続性が no_recovery に設定されているメモリ内ユーザ・データベース
- 持続性が no_recovery に設定されているユーザが作成したメモリ内テンポラリ・データベース
- 持続性が no_recovery または at_shutdown に設定されているディスク常駐型リラックス持続性データベース

図 1-1: インメモリ・データベース、リラックス持続性データベース、ディスク常駐型データベースの属性



*キャッシュ・サイズと同じ、またはそれ以上

注意 複写環境でのインメモリ・データベース、リラックス持続性データベース、および DML ロギングの使用の詳細については、お使いの Replication Server® のマニュアルを参照してください。

キャッシュおよびバッファのサポート

インメモリ・データベースをホストするキャッシュは、データベースを十分に格納できる大きさであることが必要です。また、バッファ置換やディスクの I/O を使用せずに、データベースのすべてのページがキャッシュ内に常駐できる必要もあります。次の目的で作成されたキャッシュは使用できません。

- インメモリ・データベースとしてホストされているメモリに、他のデータベースやオブジェクトをバインドすること。
- 他のデータベースや、オブジェクトがバインドされたメモリに、インメモリ・データベースをホストすること。インメモリ・データベースをホストするために使用される名前付きキャッシュは、従来の名前付きキャッシュと異なる構造を使用し、インメモリ・データベース専用になります。

インメモリ・データベースのキャッシュを作成するには sp_cacheconfig を使用します。ディスク・デバイスに似たメモリ内デバイスにキャッシュを分割してセグメントをサポートするには、disk init を使用します。1 つまたは複数の論理セグメントをメモリ内デバイスにバインドして、各セグメントにオブジェクトをバインドできます。

インメモリ・データベースまたはリラックス持続性データベースのキャッシュにオブジェクトをバインドする前に、次のことを考慮してください。

- 名前付きキャッシュを使用して、リラックス持続性データベースまたは完全な持続性データベース全体をバインドします。次のバインドが可能です。
  - リラックス持続性データベース内の個々のオブジェクトを名前付きキャッシュにバインドする。これは、通常のデータベース内のオブジェクトをバインドすることに似ています。
  - リラックス持続性データベースを名前付きキャッシュ (たとえばデフォルトのデータ・キャッシュ) にバインドする。
  - メモリ内記憶域キャッシュは名前付きキャッシュに似ていますが、メモリ内のアクセスを効率よく行えるように設定されています。リラックス持続性データベース内の個々のオブジェクトを別のキャッシュにバインドする。
- メモリ内記憶域キャッシュのキャッシング動作は、通常のキャッシュのキャッシング動作に似ています。
- 名前付きキャッシュのパフォーマンスを上げるために使用するものと同じモニタリング・ツールとチューニング手法を使用して、名前付きキャッシュにバインドされたリラックス持続性データベースのパフォーマンスを改善します。

- 1 つのメモリ内記憶域キャッシュでデータベース全体をホストするため、データベースやオブジェクトは個々のキャッシュにバインドしません。最も一般的なキャッシュ・マネージャ・モニタリングとチューニングがメモリ内キャッシュに適用されます。『パフォーマンス＆チューニング・シリーズ：sp_sysmon による Adaptive Server の監視』の「第 2 章 sp_sysmon を利用したパフォーマンスのモニタリング」を参照してください。

インメモリ・データベースはシングル・キャッシュでホストされる必要がありますが、そのキャッシュ用に作成された複数のメモリ内デバイスに常駐できます。図 1-2 に示されている imdb_cache キャッシュには、単一のメモリ内記憶域デバイス imdb_dev があります。

図 1-2: 単一のデバイスをホストするシングル・キャッシュ

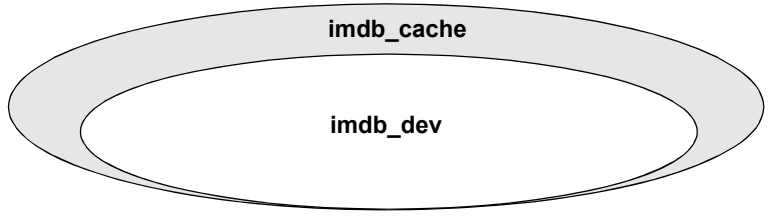
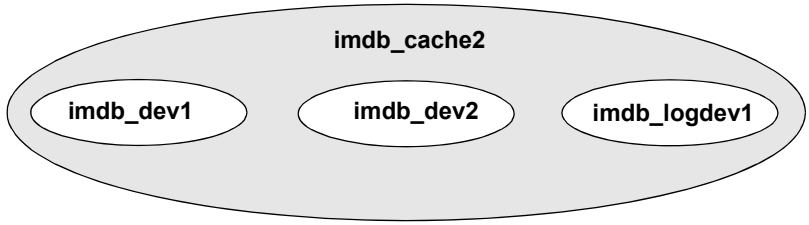


図 1-3 に示されている imdb_cache2 キャッシュには、2 つのメモリ内データ・デバイス imdb_dev1 と imdb_dev2、およびログ・デバイス imdb_logdev1 が含まれています。

図 1-3: 複数のデバイスをホストするシングル・キャッシュ



インメモリ・データベースを論理デバイス上に直接作成するには、create inmemory database を使用します。インストール時にセグメントを使用してオブジェクトの領域を制限していない場合は、メモリ内記憶域キャッシュ全体に使用するメモリ内デバイスにデータベースを作成します。個々のオブジェクトの記憶域を細かく管理し、スレッショルドのプロシージャをサポートするためには、データベースとログを別々のメモリ内デバイスに作成します。

インメモリ・データベースおよびリラックス持続性データベースのデータベース定義とオブジェクト定義を生成するには、ddlgen を使用します。『ユーティリティ・ガイド』を参照してください。

既存のインメモリ・データベースのレイアウトを変更するには、alter database を使用します。『リファレンス・マニュアル：コマンド』を参照してください。

持続性レベル

コミットされた持続性トランザクションで変更したデータは、システム障害または shutdown with nowait の後でサーバを再起動した後も保持されます。従来のディスク常駐型データベースのトランザクションの持続性は、トランザクション・ログおよびデータベース・ページをディスクに書き込むことで提供されます。インメモリ・データベースでは、サーバの障害や強制停止後のトランザクションの持続性は提供されません。

リラックス持続性データベースには 2 つのレベルの持続性があり、ユーザが選択できます。持続性の最初のレベルはインメモリ・データベースに似ています。つまり、サーバに障害が発生すると、データが失われます。持続性の 2 番目のレベルは、ディスク常駐型データベースの持続性とインメモリ・データベースの持続性の中間にあたります。つまり、正常停止の場合のみ、すべてのトランザクションが完了されて、ディスクに永続的に保持されます。これにより、リラックス持続性データベースはインメモリ・データベースで実現されるパフォーマンスの向上の多くを利用できます。

インメモリ・データベースまたはディスク常駐型データベースのどちらの場合も、持続性が no_recovery または at_shutdown に設定されたデータベースは低持続性データベースと呼ばれます。低持続性データベース内のデータはコミットの後も保持されます (サーバを再起動しない場合)。

データベースの持続性レベルを設定するには、create database with durability=*durability_level* を使用します。Adaptive Server でサポートする持続性レベルは、full、no_recovery、および at_shutdown です。『リファレンス・マニュアル：コマンド』を参照してください。

表 1-1 は、各持続性レベルで実行できる操作を示します。

表 1-1: データベースの持続性レベル

操作	no_recovery	at_shutdown	full
ディスク常駐型データベースの作成	可能	可能	可能
インメモリ・データベースの作成	可能	不可能	不可能
ランタイム・ロールバック	可能	可能	可能
障害リカバリ	不可能	不可能	可能
正常停止して再起動した後のデータベース・リストア	不可能	可能	可能
データベースのアーカイブへのダンプ、アーカイブからのロード	可能	可能	可能
デバイスからアーカイブへのトランザクション・ログのダンプアーカイブからデバイスへのトランザクション・ログのロード	不可能	不可能	可能

持続性が低減してパフォーマンスが向上するのは、at_shutdown および no_recovery を使用するリラックス持続性データベースに限られます。リラックス持続性データベースは名前付きキャッシュにバインドできます。この場合、キャッシュ・サイズをデータベース・サイズよりも小さくできます。

テンポラリ・データベースとメモリ内テンポラリ・データベース

テンポラリ・データベースは no_recovery の持続性を暗黙的に使用します。no_recovery の持続性を明示的に指定してテンポラリ・データベースを作成し、パフォーマンスを向上できます。また、既存のテンポラリ・データベースの持続性をアップグレード後に明示的に no_recovery に設定することもできます。

すべてメモリ内で使用されるテンポラリ・データベースは no_recovery の持続性を明示的に使用します。

Adaptive Server では、デフォルトの tempdb グループだけでなく、ユーザ作成の tempdb グループも、ユーザが作成および管理できます。ユーザ作成の tempdb グループに、他のユーザ作成テンポラリ・データベースを含め、アプリケーションおよびログインのバインドをサポートできます。

システム tempdb は、default テンポラリ・データベース・グループから削除できません。システム tempdb は、他のユーザ作成の tempdb グループに追加できません。

テンポラリ・データベース・グループでは、ディスク常駐型またはメモリ内テンポラリ・データベースを混在できます。

ユーザ作成の tempdb を指定し、ディスク常駐型またはメモリ内テンポラリ・データベースのみを含めるように管理できます。Adaptive Server では明示的に制限されませんが、tempdb グループのメンバシップを制御して、特定のディスク専用またはメモリ内専用の tempdb グループを特定のログインやアプリケーションに割り当てることができます。

複数データベースのトランザクションとデータベースのタイプ

データベースの持続性は、複数のデータベースにまたがるトランザクションに影響します。

低持続性データベース (ユーザ作成のテンポラリ・データベース、インメモリ・データベース、またはディスク常駐型リラックス持続性データベース) は、調整データベースで完全な持続性を使用する場合であっても、複数データベースのトランザクションに関与できます。ただし、調整データベースで低持続性を使用する場合は、他の完全な持続性データベースにまたがるトランザクションを使用できません。

表 1-2: 複数データベースのトランザクションに関与するデータベース

調整データベース	関与するデータベース	複数データベースのトランザクション
完全な持続性データベース	テンポラリ・データベース、インメモリ・データベース、リラックス持続性データベース、テンポラリ・インメモリ・データベース	可能
テンポラリ・データベース、インメモリ・データベース、リラックス持続性データベース、テンポラリ・インメモリ・データベース	テンポラリ・データベース、インメモリ・データベース、リラックス持続性データベース、テンポラリ・インメモリ・データベース	可能
テンポラリ・データベース、インメモリ・データベース、リラックス持続性データベース、テンポラリ・インメモリ・データベース	完全な持続性データベース	不可能

テンプレート・データベース

model 以外のデータベースをインメモリ・データベースのテンプレートとして使用し、テーブルやストアド・プロシージャなどのリファレンス・データをテンプレートで作成したデータベースにロードできます。テンプレート・データベースは、完全な持続性レベルが full の既存のディスク常駐型ユーザ・データベースであることが必要です。テンプレートを使用したデータベースの作成と最低限のログを取る DML は、低持続性データベースでのみサポートされます。

持続性が at_shutdown のリラックス持続性データベースは、テンプレートを使用して作成できません。テンプレートを使用できるのは、持続性が no_recovery のデータベースだけです。

no_recovery パラメータを指定して作成されているデータベースのテンプレートに model 以外のデータベースを指定するには、create database use *database_name* as template コマンドを使用します。

model 以外のデータベースをテンプレートとして使用するデータベースが作成されると、Adaptive Server はサーバの再起動時にテンプレート・データベースのデータを使用して従属データベースを再作成します。

Adaptive Server を再起動すると、データベースがテンプレートとして使用するテンプレート・データベースが最初にリカバリされます。Adaptive Server がテンプレート・データベースをリカバリできない場合、そのテンプレートを使用するデータベースを再作成できません。

Adaptive Server は、従属データベースの作成時にテンプレート・データベースの属性を適用します。create database コマンドに含めて指定する属性は、テンプレート・データベースの属性を上書きします。テンプレートを使用して作成したデータベースのデータベース・オプションと属性は、Adaptive Server を再起動した場合や、Adaptive Server が同じテンプレートからデータベースを再作成した場合でも保持されます。テンプレート・データベースの属性に加えた変更は、Adaptive Server が以降の再起動時に従属データベースを再作成する場合に使用されません。

他のデータベースがテンプレートとして使用しているデータベースは削除できません。このデータベースをテンプレートとして使用している他のすべてのデータベースをまず削除するか、または alter database を使用してすべての従属データベースからテンプレート・データベースを分離する必要があります。

注意 テンプレート・データベースのユーザ定義のセグメントは、テンプレート・データベースからインメモリ・データベースまたはリラックス持続性データベースにコピーすると予想どおりに機能しない場合があります。セグメントでは、syssegments テーブルを master データベースの sysuages テーブルにマッピングすることで、特定のデータベース・デバイス (またはデバイス・フラグメント) に領域が割り付けられます。テンプレート・データベースでは、テンプレートを提供するインメモリ・データベースまたはリラックス持続性データベースとは異なるマッピングが使用されることがあります。これは、これらのデータベースのデバイスのレイアウトが異なるデバイス数やサイズ・フラグメントを使用する場合があるためです。テンプレート・データベースのユーザ定義のセグメントを注意して計画および定義してから、インメモリ・データベースまたはリラックス持続性データベースのこれらのセグメントを使用する必要があります。

---

sp_helpdb を実行して、次のデータベースのテンプレートに関する情報をレポートします。

- ユーザ・データベース – ユーザ・データベースがテンプレートとして使用されているかどうか、また、どのデータベースが使用しているかを特定します。次に示す sp_helpdb 出力のセクションは、pubs2 データベースが pubs3 および pubs5 インメモリ・データベースのテンプレートとして使用されていたことを示します。

  ```
  template_for
  --------------------------------------------------
  pubs3
  pubs5
  ```

- テンプレートから作成されたデータベース – 作成時にテンプレートとして使用されたデータベースがどれであるかを特定します (model 以外の場合)。次に示す sp_helpdb 出力のセクションは、pubs2 データベースが pubs3 データベースのテンプレートとして使用されていたことを示します。

  ```
  template
  --------------------------------------------------
  pubs2
  ```

新しいテンプレートを使用するためのデータベースの変更

alter database を使用してテンプレートを変更した場合、Adaptive Server はデータベースの既存のデータを変更しません。Adaptive Server を再起動した場合、Adaptive Server は新しいテンプレートのデータを使用してデータベースを再作成します。

インメモリ・データベースおよび持続性が no_recovery のデータベースのテンプレートのみ変更可能です。システム・データベースや持続性レベルが full の従来のディスク常駐型データベースのテンプレートは変更できません。

最低限のログを取るコマンド

Adaptive Server バージョン 15.5 では、データベース単位、テーブル単位、セッション固有の単位でデータ操作言語 (DML コマンド) の最低限のロギングを実行できます。これにより、ローとページの変更、ページ割り付け、ページ割り付けの解除の最低限のログが取られます。

DML ロギングは、create database、alter database、create table、select into、set DML logging、alter table などのコマンドを設定して、データベース・レベル、テーブル・レベル、およびセッション・レベルで制御できます。

「第 3 章 最低限のログを取る DML」を参照してください。

インメモリ・データベースおよびリラックス持続性データベースの制限

インメモリ・データベースおよびリラックス持続性データベースには、次の制限があります。

- 現在 Cluster Edition は、インメモリ・データベース、リラックス持続性データベース、テンプレート・データベース、または最低限のログを取る DML をサポートしていません。
- 現在、Replication Server は、インメモリ・データベース、または持続性が no_recovery に設定されたデータベースの複写をサポートしていません。
- インメモリ・データベースはアーカイブ・データベースとして使用できません。さらに、インメモリ・データベースはスクラッチ・データベースとしても使用しないことをおすすめします。
- インメモリ・データベースでは、互換性モードを使用したクエリを使用できません。Adaptive Server の互換性モードは、通常のディスク常駐型データベース・テーブルでのみ使用してください。互換モードの有効化の詳細については、『マイグレーション技術ガイド』を参照してください。

互換性モードが有効で、かつインメモリ・データベースのテーブルでクエリを使用する場合は、「制限付き互換モード」を使用して、Adaptive Server をネイティブ・バージョン 15.0 のクエリ・オプティマイザと実行エンジンに戻してください。一般にこのモードでは、Adaptive Server バージョン 12.5 のプランに似たクエリ・プランが生成されます。パフォーマンスが低下した場合は、このクエリに対して互換性モードを無効にすることをおすすめします。

- データベースのサイズを増やす 1 つのコマンド内で、インメモリ・データベースまたは低持続性データベースの持続性レベルまたはロギング・レベルは変更できません。
- データベースの durability 属性または minimal_logging 属性を変更する場合に alter database を使用すると、データベースは自動的にシングルユーザ・モードになるため、データベースへの排他アクセスを取得できない場合、コマンドが失敗します。これを防ぐために、alter database を実行する前にデータベースを手動でシングルユーザ・モードにします。
- インメモリ・データベースおよびリラックス持続性データベースは、分散トランザクションに関与できません。
- インメモリ・データベースおよびリラックス持続性データベースは、完全な持続性のデータベースが含まれているマルチデータベース・トランザクションを調整できません。インメモリ・データベースまたはリラックス持続性データベースからストアド・プロシージャを実行する場合、トランザクションの更新を実行するシステム・プロシージャを実行すると、次のエラーが出されます。

```
Msg 3952, Level 16, State 2:
Procedure 'sp_XX', Line 258:
Command not allowed. You cannot start a multidatabase
operation in database 'master'
after starting the master transaction in 'imdb1' as it may
render the database 'master' unrecoverable.
```

システム・プロシージャを実行するには、次のようにします。

a 持続性レベルが full のデータベースから実行する。たとえば、sp_XX を master で実行するには、次のように入力します。

```
use master
go
exec sp_XX
```

b 次のフォーマットを使用して、現在のデータベース (メモリ内またはリラックス持続性) からストアド・プロシージャを参照する。
`database_name.owner.`*`sp_name`*

たとえば、sp_XX を imdb_1 インメモリ・データベースで実行するには、次のように入力します。

```
use imdb_1
go
exec master.dbo.sp_XX
```

- インメモリ・データベースでは、プロキシ・テーブルまたはデータベースを使用して、他のインメモリ・データベースやディスク常駐型データベースのオブジェクトにマップできません。
- ディスク常駐型のデータベースでは、プロキシ・テーブルやデータベースを使用して、インメモリ・データベースやテーブルにマップできません。

変更されたシステム・プロシージャ

表 1-3 は、メモリ内記憶域キャッシュ、メモリ内デバイス、インメモリ・データベースをサポートするために変更されたストアド・プロシージャの一覧です。

表 1-3: インメモリ・データベースに対応するために変更されたシステム・プロシージャ

システム・プロシージャ	コメント
sp_addsegment	インメモリ・データベースの領域を管理するために更新されました。
sp_addthreshold	インメモリ・データベースの領域を管理するために更新されました。
sp_bindcache	オブジェクトやデータベースはメモリ内記憶域キャッシュにバインドできません。また、インメモリ・データベースやインメモリ・データベースのオブジェクトはキャッシュにバインドできません。
sp_cacheconfig	メモリ内記憶域キャッシュの作成、サイズの拡大、または削除を行います。
sp_cachestrategy	prefetch パラメータと MRU パラメータは、インメモリ・データベースのテーブルおよびインデックスに適用されません。
sp_dbextend	自動データベース拡張は、現在インメモリ・データベースでサポートされていません。
sp_deviceattr	directio 属性と dsync device 属性は、メモリ内デバイスに適用されません。
sp_diskdefault	sp_diskdefault を使用して、メモリ内デバイスをデフォルトのデバイスに指定できません。
sp_downgrade	インメモリ・データベースやリラックス持続性データベース、またはテンプレートや最低限のロギングを使用するデータベースが含まれている Adaptive Server の旧バージョンへのダウングレードをサポートします。
sp_dropdevice	作成されたメモリ内デバイスをメモリ内記憶キャッシュから削除します。

システム・プロシージャ	コメント
sp_dropsegment	インメモリ・データベースの領域を管理するために更新されました。
sp_dropthreshold	インメモリ・データベースの領域を管理するために更新されました。
sp_extendsegment	インメモリ・データベースの領域を管理するために更新されました。
sp_help	最低限のロギング属性など、テーブルのプロパティを報告します。
sp_helpcache	メモリ内記憶域キャッシュのプロパティ、このキャッシュに作成されたインメモリ・データベースのプロパティ、このキャッシュの空き容量の詳細を表示します。
sp_helpdb	持続性、DML ロギング・レベル、インメモリ・データベースかどうか、テンプレート・データベースがある場合はテンプレート・データベースとしての使用など、データベースのプロパティを報告します。
sp_helpdevice	メモリ内記憶域キャッシュで作成されたメモリ内デバイスのプロパティを報告します。
sp_modifythreshold	インメモリ・データベースの領域を管理するために更新されました。
sp_plan_dbccdb	インメモリ・データベースで checkstorage を実行するための dbccdb を設定します。
sp_poolconfig	インメモリ・データベースでは、サイズの大きな I/O バッファ・プールはサポートされていません。
sp_post_xpload	インメモリ・データベースに関するプラットフォームを問わない操作をサポートします。
sp_tempdb	メモリ内テンポラリ・データベースのログインまたはアプリケーションのバインドをサポートします。
sp_unbindcache、sp_unbindcache_all	インメモリ・データベース自体のオブジェクトのバインドをホストのメモリ内記憶域キャッシュから解除できません。

第 2 章

インメモリ・データベースおよびリラックス持続性データベースの管理

インメモリ・データベースの名前付きキャッシュの指定

メモリ内記憶域キャッシュには大きなページを使用することをおすすめします。使用しているプラットフォームの『設定ガイド』を参照してください。

インメモリ・データベースを保持するキャッシュは、データベース全体を十分に収容できる大きさであることが必要です。インメモリ・データベースを収容するキャッシュはメモリ内記憶域と呼ばれ、次のものを無効にします。

- ランタイム・オペレーション中の I/O
- バッファ・ウォッシング
- バッファ置換およびウォッシング

メモリ内記憶域キャッシュを作成したら、1 つまたは複数のパーティションに分割します。各パーティションは、データベースまたはログのフラグメントを保持します。「キャッシュおよびバッファのサポート」(4 ページ) を参照してください。

メモリ内記憶域キャッシュを作成するには、inmemory_storage パラメータを指定して sp_cacheconfig を使用します。『リファレンス・マニュアル：プロシージャ』を参照してください。

注意 メモリ内記憶域キャッシュを作成する前に、max memory の値が指定したキャッシュ・サイズに対して十分であるかどうか確認してください。max memory の値が十分でないと、Adaptive Server はエラー・メッセージを返します。

たとえば、imdb_cache という名前のメモリ内記憶域キャッシュを作成するには、次のように入力します。

```
sp_cacheconfig imdb_cache, '2G', inmemory_storage
```

注意 通常の名前付きキャッシュでは、使用可能なメモリ・サイズがキャッシュに対して要求されたメモリ・サイズよりも小さい場合、Adaptive Server はメモリ・サイズを小さくしてキャッシュを作成します。つまり、キャッシュは正常に作成されますがサイズは小さくなります。ただし、インメモリ・データベースでは、キャッシュを作成するだけの十分な容量がない場合、このコマンドは失敗します。

設定ファイルに加えた変更の確認

設定ファイル (*$SYBASE/server_name.cfg*) にメモリ内記憶域キャッシュの情報が正しく指定されていることを確認します。各メモリ内記憶域キャッシュには、("Named Cache: *cache_name*") というラベルの付いた設定ファイルの見出しが含まれています。`Named Cache` のエントリには、次の情報が含まれています。

- `cache size` − キャッシュのサイズは、インメモリ・データベース全体を十分に保持できる大きさであることが必要です。
- `cache status` − "in-memory storage cache" に設定されます。
- `cache replacement policy` − "DEFAULT" または "none" に設定されます。
- `local cache partition number` − ローカル・キャッシュ・パーティションの番号または "DEFAULT"。

imdb_cache というキャッシュのエントリは、次のようになります。

```
[Named Cache:imdb_cache]
cache size = 2G
cache status = in-memory storage cache
cache replacement policy = none
local cache partition number = 8
```

インメモリ・データベースの静的な設定パラメータの変更

Adaptive Server は、再起動時に静的な設定パラメータを変更します。インメモリ・データベースはサーバの再起動時に再作成され、変更内容はすべて失われるため、Adaptive Server を再起動するときは次のいずれかを実行して、インメモリ・データベースのデータが失われないようにしてください。

- インメモリ・データベースが作成される前に、静的な設定変更をすべて加える。または、
- インメモリ・データベースをアーカイブにダンプし、静的な設定変更を加え、サーバを再起動し、アーカイブからインメモリ・データベースをロードする。

メモリ内デバイスの作成

メモリ内記憶域キャッシュを、インメモリ・データベースを作成するために使用するメモリ内デバイスと呼ばれる小さな部分に分割するには、disk init を使用します。ディスク常駐型デバイスで使用する命名規則と同じものをメモリ内デバイスでも使用することをおすすめします。ユーザ定義またはシステム定義のセグメントにバインドされるオブジェクトの領域の使用量を制御するには、これらのセグメントとメモリ内デバイスの論理名をバインドします。

注意　通常の名前付きキャッシュを使用してメモリ内デバイスを作成できません。つまり、メモリ内デバイスを作成するには、disk init の type=inmemory パラメータを使用する必要があります。

メモリ内デバイスを作成する構文は次のようになります。

```
disk init name = device_name
   physname = {"physical_name" | "cache_name"}
. . .
[, type = "inmemory"]
```

ここで、*device_name* はメモリ内デバイスの論理名、*cache_name* は sp_cacheconfig で作成されるメモリ内記憶域キャッシュの名前を表し、inmemory はデバイスがインメモリ・データベース用であることを示します。

次に例を示します。

```
disk init name = imdb_cache_dev,
physname = "imdb_cache" ,
size = "50M",
type = "inmemory"
```

インメモリ・データベースの作成

model または別のユーザ・データベースをテンプレートとして使用し、インメモリ・データベースを作成するには、create inmemory database を使用します。

テンポラリ・データベースは、全体がメモリ内記憶域に常駐するインメモリ・データベースとして作成することもできます。ただし、メモリ内テンポラリ・データベースにはテンプレートを指定できません。

ユーザデータベース・テンプレートを使用してインメモリ・データベースを作成すると、すべてのユーザ、権限、オブジェクト、プロシージャがテンプレート・データベースからインメモリ・データベースにコピーされます。

メモリ内テンポラリ・データベースを作成すると、次のことが行われます。

- guest ユーザがテンポラリ・データベースに追加されます。
- create table 権限が public に付与されます。

システム・データベース (たとえば sybsecurity) は、インメモリ・データベースとして作成できません。システム・データベースは Adaptive Server に障害が発生した場合に更新する必要があるためです。

インメモリ・データベースを作成するときに、insert、update、delete、および一部のバルク・コピー・イン・オペレーションが、データベース単位、オブジェクト単位、またはセッション固有単位で最低限のログを取るか、または完全なログを取るかを指定できます。「第 3 章 最低限のログを取る DML」を参照してください。

次の例では、2GB のメモリ内デバイス上にインメモリ・データベースを作成します。with override を使用して、同じメモリ内デバイスにデータ・セグメントとログ・セグメントを作成できます (インメモリ・データベースでサポートされている持続性レベルは no_recovery のみです。別の持続性レベルを使用しようとするとエラーになります)。

```
create inmemory database imdb1
on imdb_data_dev1 = '1.0g'
log on imdb_data_dev1 = '0.5g'
with override, durability = no_recovery
```

リラックス持続性のディスク常駐型データベースの作成

リラックス持続性データベースの持続性レベルは、no_recovery または at_shutdown に設定します。「持続性レベル」(6 ページ) を参照してください。

リラックス持続性データベースは既存のディスク上に作成できます。リラックス持続性データベースではディスク記憶領域を使用するため、disk init を使用してデータベースが常駐するデバイスを作成する必要があります。

次の例では、pubs6 データベースを at_shutdown の持続性レベルで作成します。

```
create database pubs6
on pubs6_dev
with override, durability=at_shutdown
```

インメモリ・データベースの管理

インメモリ・データベースを作成したら、メモリ内記憶域キャッシュのサイズ変更/削除、インメモリ・データベースのサイズの拡大、ダンプ/ロードの実行などを行ってデータベースを管理します。

メモリ内記憶域キャッシュのサイズ変更

実行時のメモリ内記憶域キャッシュのサイズを増やすには、sp_cacheconfig を使用します。たとえば、imdb_cache のサイズを 3GB に増やすには、次のように入力します。

```
sp_cacheconfig imdb_cache, '3G', inmemory_storage
```

同じプロシージャを使用して、メモリ内記憶域キャッシュのサイズを減らすことができます。メモリ内記憶域キャッシュのサイズは、現在インメモリ・データベースで使用していないキャッシュの記憶域の大きさまで減らすことができます。つまり、メモリ内記憶域キャッシュはインメモリ・データベースよりも小さくできません。

サイズが減ったキャッシュは、サーバの再起動時に作成されます。

メモリ内記憶域キャッシュの削除

メモリ内記憶域キャッシュを削除する前に、すべてのデバイスとインメモリ・データベースを削除する必要があります。メモリ内記憶域キャッシュを削除するには、サイズにゼロを設定します。

```
sp_cacheconfig imdb_cache, '0g'
```

インメモリ・データベースのサイズの増加

インメモリ・データベースのサイズを増やすには、alter database を使用します。サイズを増やすデバイスは、データベースが常駐しているデバイスを現在ホストしているキャッシュの一部であることが必要です (つまり、追加のメモリ内記憶域キャッシュを作成して、この記憶域キャッシュ上の既存のインメモリ・データベースのサイズを増やすことはできません)。disk resize を実行して、インメモリ・データベース・デバイスのサイズを大きくすることはできません (標準のデータベース・デバイスも同様です)。

インメモリ・データベースのサイズを大きくするには、次の手順に従います。

1 メモリ内記憶域キャッシュを大きくするには sp_cacheconfig を使用します。

2 大きくしたメモリ内記憶域キャッシュに 2 番目のインメモリ・デバイスを作成するには、disk init を使用します。

3 alter database を実行し、手順 2 で作成した新しいデータベース・デバイス上にインメモリ・データベースを拡張します。

リラックス持続性データベースのサイズを大きくするには、標準のディスクベースのデータベースのサイズを大きくする手順と同じ手順を行います。

インメモリ・データベースのダンプとロード

インメモリ・データベースおよびリラックス持続性データベースをアーカイブ・デバイスにダンプするには dump database を使用し、アーカイブ・デバイスからロードするには load database を使用します。インメモリ・データベースまたはリラックス持続性データベースをダンプまたはロードする場合、dump コマンドまたは load コマンドに特別なパラメータを指定する必要はありません。

インメモリ・データベースに対する load database では、データはメモリ内記憶域キャッシュに直接ロードされます。

次の操作ができます。

- 複数の持続性レベルにまたがるダンプとロード。たとえば、持続性レベルが full のデータベースを、持続性レベルが常に no_recovery のインメモリ・データベースにダンプできます。
- プラットフォーム間でのダンプとロードの実行。

次の処理はできません。

- インメモリ・データベースまたはリラックス持続性データベースからの dump transaction の実行。load transaction は、順序付けられていないログ・レコードが含まれている可能性のあるトランザクション・ログを使用して、要求されたタスクを実行できないためです。

- 持続性が no_recovery または at_shutdown のデータベースから、インメモリ・データベースまたはリラックス持続性データベースをサポートしないバージョンの Adaptive Server へのダンプのロード。ただし、旧バージョンの Adaptive Server からインメモリ・データベースやリラックス持続性データベースにデータベース・ダンプをロードできます。

注意　使用する Backup Server は、インメモリ・データベースまたはリラックス持続性データベースのダンプ操作およびロード操作で使用するバージョンの Adaptive Server と共に出荷されるバージョンのものであることが必要です。

number of backup connections の設定

インメモリ・データベースをダンプおよびロードするには、Backup Server を Adaptive Server に接続する必要があります。load コマンドは、ストライプと同数の接続を使用します。dump コマンドは、ストライプと同数の接続と 1 つの追加の接続を使用します。Backup Server で使用できる最大ユーザ接続数を設定するには、number of backup connections を使用します。

『システム管理ガイド 第 1 巻』の「第 5 章 設定パラメータ」を参照してください。

Backup Server を Adaptive Server に接続するには次のことが必要です。

- Backup Server インターフェイス・ファイルに Adaptive Server のエントリがあること。
- Backup Server が Adaptive Server に接続するために使用するユーザ名とパスワードが dump コマンドおよび load コマンドを実行しているユーザと同じであること。Kerberos などの安全な外部認証を使用して接続が確立された場合、Backup Server は Adaptive Server から取得されたパスワードを取り出すことができません。sp_remotelogin を使用して SYB_BACKUP の trusted リモート・ログインを定義します。そうしない場合、Backup Server は認証エラーを受け取ります。

インメモリ・データベースの削除

インメモリ・データベースを削除するには、drop database を使用します。次の例では、pubs6 データベースを削除します。

```
drop database pubs6
```

インメモリ・データベースを削除すると、データベースがシステム・テーブルから削除されて、メモリ内記憶域キャッシュが開放されます。ただし、バッファとデータはデバイス内に残ります。インメモリ・データベースを削除しても、その作成先のメモリ内記憶域キャッシュには影響しないため、他のインメモリ・データベースで使用できます。

メモリ内デバイスの削除

メモリ内デバイスを削除するには sp_dropdevice を使用します。sp_dropdevice は、メモリ内デバイスがデータベースによって現在使用されていない場合にのみ、これを削除します。最初にデータベースを削除してから、メモリ内デバイスを削除します。メモリ内デバイスを削除すると、そのエントリが sysdevices から削除されて、メモリは作成先のキャッシュに返されるので、新しいメモリ内デバイスの作成など、他の目的で使用できます。次の例では、pubs6_device というデバイスを削除します。

```
sp_dropdevice 'pubs6_device'
```

第 3 章

最低限のログを取る DML

Adaptive Server はディスクのトランザクション・ログに書き込まれるログ・レコードを最適化するために、insert、update、delete、低速 bcp などの一部のデータ操作言語 (DML: data manipulation language) をあらゆる種類の低持続性データベース (at_shutdown オプションや no_recovery オプションを使用するインメモリ・データベースや低持続性データベースなど) に対して実行する場合は、最低限のロギングを実行するか、ロギングを実行しないのいずれかです。DML の最低限のロギングは、データベース単位、テーブル単位、セッション単位で実行できます。

DML ロギング設定の種類

DML ロギング設定の制御は、次のような階層になっています。

- データベース・レベルのロギング – デフォルトでは、すべてのテーブルに対してデータベース・レベルで DML のロギングが可能です。DML ロギング設定が影響するのは、ユーザ・テーブルとテンポラリ・テーブルだけです。
- テーブル・レベルのロギング – テーブルの作成方法や変更方法に応じて、データベース・レベルでのロギング設定のレベルを上書きします。
- セッション・レベルのロギング – テーブル・レベルおよびデータベース・レベルでのロギング設定のレベルを上書きします。

データベース・レベルのロギング

データベース・レベルでロギングを有効/無効にする機能は、主にテンポラリ・データベースのためのものです。テンポラリ・データベースでは、ロギングに依存しないすべてのアプリケーションをより効率的に実行でき、アプリケーションのコード変更やテンポラリ・テーブルを作成するプロシージャの変更は不要です。

model データベースを含め、システム・データベースのロギング・モードは変更できません。ただし、システム tempdb およびユーザ・テンポラリ・データベースの DML ロギング・モードは、最低限のロギングに変更できます。モードを変更する前に、最低限のロギングへの変更が、テンポラリ・テーブルを使用するアプリケーションのロールバック・セマンティックにどのような影響を与えるかを調べることをおすすめします。「トランザクションのセマンティック」(28 ページ) を参照してください。

最低限のログを取る DML を使用できるのは、持続性レベルが no_recovery または at_shutdown に設定されたデータベース内に限られます。最低限のロギングを有効にするためには、データベースの select into オプションを on に設定する必要があります。

データベースのデフォルトのロギング・モードは、master データベースからのみ変更できます。また、変更対象のデータベースで次の条件が満たされていることも必要です。

- ユーザ・データベースに対してシングルユーザ・モードになっていること
- データベース所有者だけがデータベースを使用できるように、テンポラリ・データベースに対して dbo-use only が true (on) に設定されていること

データベースが要求されたモードになっていない場合、サーバはデータベースを該当するモードにします。これが失敗した場合、サーバはエラーを出して、データベースを明示的に正しいモードにするようにユーザに要求します。

コマンド

データベース・レベルで DML ロギング・モードを変更する構文は次のようになります。

```
create [temporary] database database_name
    [on {default | database_device [= size]
                [, {database_device [= size]...]
    [log on {database_device [= size]
            [, {database_device [= size]]...]
    [with {override | default_location = "pathname"
    | [[,]durability = { no_recovery | at_shutdown | full} ]
    | [[,]dml_logging = {full | minimal} ]
    }...
]
    [for {load | proxy_update}]
```

既存のデータベースの DML ロギングの設定をデータベース・レベルで変更するには、次の構文を使用します。

```
alter database dbname
set dml_logging = {full | minimal}
```

『リファレンス・マニュアル：コマンド』を参照してください。

テーブル・レベルのロギング

DML ロギング・オプションをテーブル・レベルで設定すると、テーブルの作成方法や変更方法に応じてデータベース・レベルの設定が上書きされます。デフォルト・モードでは、データベースと同じ設定を使用します。

最低限のロギングがデータベース・レベルで有効になっていない場合は、次のようになります。

- 最低限の DML ロギングは、create table または alter table を使用して、dml_logging が明示的に minimal に設定されているテーブルに対して行われます。
- 他のすべてのテーブルに関しては、DML は完全にログされます。

最低限のロギングがデータベース・レベルで有効になっている場合は、次のようになります。

- 完全な DML ロギングは、dml_logging が明示的に full に設定されているテーブルに対して行われます。
- 他のすべてのテーブルに関しては、最低限の DML ロギングが行われます。

テーブルの default ロギング設定では、テーブルは DML の実行時に、ロギングの現在のデータベース・レベルを継承できます。データベース管理者は、データベース・レベルで定期的にロギングをオフにしてから、もう一度オンにすることがあります。この場合、明示的なテーブル・レベル設定を使用して制御する必要があるのは、full や minimal の DML ロギングなどの特定の要件を持つテーブルだけです。

データベースで select into データベース・オプションがオンになっている場合は、テーブルに対して最低限のログを取る DML コマンドを実行できます。それ以外の場合は、すべての DML コマンドで完全にログされます。

DML 文に対応するトリガが有効になっている (たとえば、insert 文を実行すると insert トリガが有効になる) テーブルに対する DML 文の場合、Adaptive Server は完全なロギングを実行します。つまり、トリガの実行中にログ・レコードを必要とするビジネス・ルールやセキュリティ・メカニズムが、トリガによって実装されなくなります。最低限のロギングを実行するには、DML 文を実行する前にテーブル所有者が特定のトリガを無効にする必要があります。

更新可能なビューに対して DML 文が実行されると、この DML 文は最低限のログを取る DML 操作に対応できるベース・テーブルの DML 文を最終的に解決するため、ビューに対して実行される DML 文が結果的にベース・テーブルの最低限のログを取る DML になります。ビューによって更新されるベース・テーブルのロギング・モードを制御するには、alter table または set dml_logging を使用して、基本となるテーブルにロギング・モードを設定します。

alter table では、ユーザ・テーブルのロギング・モードのみ変更できます。ビューや他のオブジェクトは変更できません。

コマンド

完全な DML ロギングまたは最低限のロギングが設定されたテーブルを作成するには、次の構文を使用します。

```
create table tablename (
        <rest of the column list specifications>
)
lock lock_scheme
with { max_rows_per_page = num_rows
        , exp_row_size = num_bytes
        , reservepagegap = num_pages
        , identity_gap = value
        , dml_logging = {full | minimal}
        }
on segment_name
```

select into を使用してテーブルを作成し、ターゲット・テーブルの DML ロギングを有効または無効にするには、次の構文を使用します。

```
select <column list>
into table_name
        [ <external table specifications> ]
        on segment_name
        [ partition_clause ]
        lock lock_scheme
        with { max_rows_per_page = num_rows
                , exp_row_size = num_bytes
                , reservepagegap = num_pages
                , identity_gap = value
                        , dml_logging = {full | minimal}
                }
[ from_clause ]
[ where_clause ]
...
```

注意 バージョン 15.5 では、テーブル所有者がロギングを明示的にオフにしてテーブルを作成した場合、無条件にテーブルのロギングをオンにできません。これにより、最低限のロギングを使用するように作成されたテーブルに対して、大きな DML が大量のロギングを生成できなくなります。

セッション・レベルのロギング

ロギング・オプションのセッション固有の設定は、ロギング・オプションのテーブル・レベルおよびデータベース・レベルの設定を上書きします。

データベース・レベルまたはテーブル固有で完全な DML ロギングが設定されている場合であっても、現在のセッションの DML に対してロギングを有効または無効にするには、次の構文を使用します。

```
set dml_logging { minimal | default}
```

『リファレンス・マニュアル：コマンド』を参照してください。

DML ロギングを minimal に設定した場合、現在のセッション・ユーザが所有するオブジェクトのロギング・モードにのみ影響します。セッション・ユーザに sa_role がある場合は、すべてのユーザ・オブジェクトのロギング・モードが minimal になります。

セッション固有の DML ロギングを minimal に設定すると、set dml_logging default は、テーブル・レベルおよびデータベース・レベルの設定に基づき、影響を受けるテーブルで現在有効なロギング・モードをデフォルトのロギング・モードに戻します。

set dml_logging コマンドを使用して、テーブルに対して最低限のロギングを実行できます。ただし、データベース所有者またはテーブル所有者が、テーブルが最低限のロギングで実行されるようにすでに設定している場合は、このコマンドを使用して完全なログを取る DML を実行できません。

実際に特定のオブジェクトのログが取られるかどうかを決定するのは、ロギング・モードのセッション固有の設定です。これは、前述および以降の項で説明する、特定のテーブルのロギング・モードの選択に関する各種ルールがあるためです。

- set コマンドの呼び出しが成功した場合は、パーミッションと権限に従って、現在のセッションのすべてのテーブルがロギング・モードの選択対象候補になります。
- set コマンドでは、ユーザ・テーブルのロギング・モードのみ変更できます。システム・テーブルやビューなどの他のオブジェクトは変更できません。
- 最低限の DML ロギングでは、select into データベース・オプションがオンである必要があります。このオプションをオンにするためには、データベース所有者であるか、または sa_role 権限が必要です。

最低限のロギングに関するその他のルール

データベース、テーブル、セッションに関連した基本的なルール以外に、次のルールも最低限のロギングに影響します。

- 最低限のログを取る DML を使用できるのは、インメモリ・データベースまたはリラックス持続性データベースだけです。持続性レベルが full のデータベースでは使用できません。
- 複数文のトランザクション内のテーブルに対するロギング・モードは、トランザクション全体にわたって変更されません。
- 宣言参照整合性または論理参照整合性の制約を受けるテーブルに対しては、すべての DML コマンドが完全にログされます。

- set dml_login オプションは、ログイン・トリガからクライアント・セッションにエクスポートできます。ただし、ほとんどの set オプションとは異なり、set export_options を有効にした場合であっても、set dml_logging をストアド・プロシージャからエクスポートしたり、execute immediate をオプションの呼び出し側に対して出すことはできません。
- セーブポイントの後ろにあるすべての DML コマンドは、テーブルで最低限のロギングが有効であっても、**完全な**ロギングで実行されるように設定されます。
- アクティブなトリガがテーブルに存在している場合は、完全なロギングが実行されます。最低限のロギング・モードで実行する DML では、DML 文を実行する前に、すべてのトリガを無効にしてください。
- オプティマイザで遅延モードの更新処理が選択されると、最低限の DML ロギング設定が上書きされて、DML は **完全な**ロギングで実行されます。
- ログベースの複写をサポートするために、複写されたテーブルの DML は常に**完全な**ロギングを実行します。

トランザクションのセマンティック

最低限のロギング・モードでの操作時は、ランタイム・ロールバック後のトランザクションの原子性が保証されません。

ランタイム中はロギングが不完全であるため、失敗したコマンドで加えられた変更を完全にロールバックできません。すべての変更は、コミット・モードでデータベースに適用されます。たとえば、多くのローを削除したトランザクションはロールバックできません。削除されたローに対する変更はすでにコミットされているためです。トランザクションでページのローを削除してから、ページの割り付けを解除した場合、ページの割り付けは解除されたままです。

ただし、ロギングのない変更によって、影響を受けるローやページのロックが解除されることはありません。使用可能なロールバックのない最低限のロギング・モードで DML コマンドを実行する場合であっても、影響を受けるローおよびページのロックが取得されて、トランザクションが終わるまで保持されます。ロックはトランザクションの終了時にのみ解除されます。コマンドは rollback または commit です。

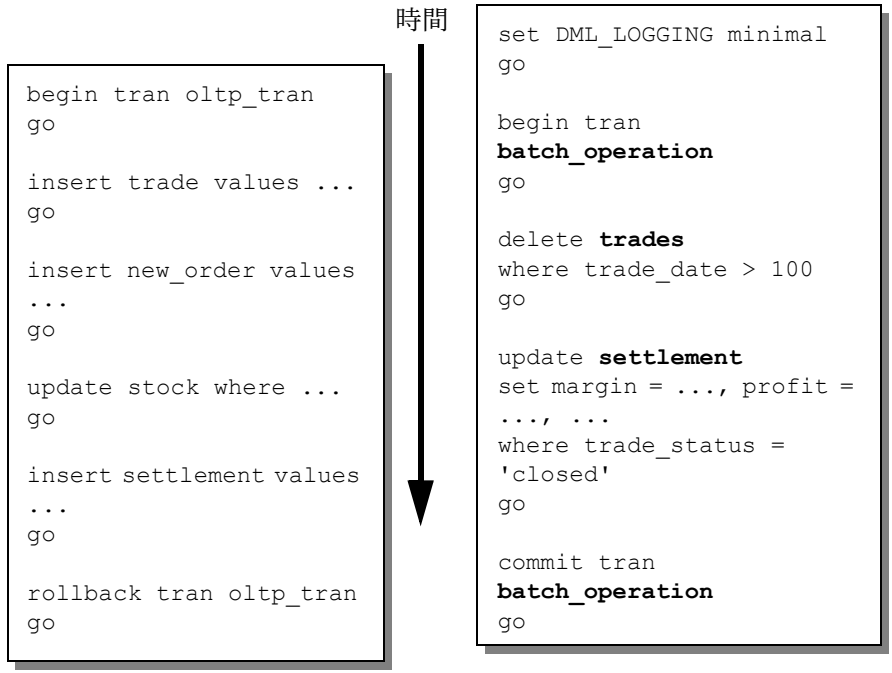
1 つまたは複数のテーブルで最低限のロギングを指定してトランザクションを実行した場合に rollback コマンドを出すと、ロールバックの発生ポイントで Adaptive Server がトランザクションをコミットしていることを警告するメッセージが出されます。

同時トランザクションのロギング

set dml_logging コマンドが影響するのは、現在のセッションのロギング・モードと、現在のセッションのユーザが所有するテーブルの DML 文だけです。このため、DML コマンドは複数のトランザクションから同時に実行できます。たとえば、full ロギングが適用された 1 つのセッションと、同時に実行している minimal ロギングの別のセッションから実行できます。full ロギングで実行しているセッションのロールバックはすべて取り消されますが、別のセッションのロールバックの変更は取り消されません。

このような使い方の一般的な例として、小さなトランザクションと大量のバッチ更新または削除を minimal のロギング・モードで同時に実行するトランザクション・システムが挙げられます。どちらかのセッションでエラーが発生しても、もう一方のセッションのトランザクションの一貫性には影響しません。次の例では、OLTP トランザクションを full ロギングで実行して、完全なリカバリ性を有効にする一方で、minimal のロギングで別のセッションからバッチ操作を実行します。

図 3-1: ロギング・モードの異なるトランザクションの同時実行



ddl in tran が true に設定された最低限のロギング

ddl in tran データベース・オプションが true に設定されている場合、同じトランザクション内で DDL 操作がすでに実行されたテーブルに対して DML 文を minimal のロギング・モードで実行できません。このようなトランザクションでは、DML コマンドは full ロギングで実行されます。

次のトランザクションでは、sp_dboption　データベース・オプションの ddl in tran が true に設定されていて、テーブル t1 は最低限の DML ロギングで設定されています。ただし、Adaptive Server は実行時に、drop index コマンドの後にくる DML コマンドを完全なロギングで実行します。これは、drop index コマンドを完全なロギングで実行したためです。

```
sp_dboption pubs1, 'ddl in tran', on
go

begin tran
go

update t1 set ...
go

drop index t1.ind1
go

insert t1 values ...
go

insert t1 values ...
go

delete t1 where ...
go

update t1 where ...
go

rollback tran
go
```

create index コマンドと drop index コマンドが同じトランザクションで実行されると、最低限のロギングが選択される方法に影響します (つまり、上のスクリプトでは、このどちらかのコマンドによって、Adaptive Server が最低限のロギングではなく完全なロギングを実行するようになりました)。

参照整合性制約の影響

参照整合性制約が関係するテーブルの場合、最低限のロギングに関するルールは次のようになります。

- 参照整合性制約のあるテーブルに対するすべての DML 文は、常に完全にログされ、データベース・レベルの設定またはセッション固有の設定に基づいてテーブルに適用されるロギング設定をすべて上書きします。
- 参照整合性制約を含むテーブルの場合、alter table を使用してテーブルのロギング・モードを minimal に変更できます。
- minimal のロギングが定義されているテーブルの間に参照整合性制約を作成できません。たとえば、プライマリ・テーブルに minimal のロギングが明示的に設定されている場合、テーブルからプライマリ・テーブルへの外部参照整合性制約を作成すると、エラーが発生します。

最低限のロギング・モードでの複数文トランザクション

複数のバッチの DML 文が後に続く場合に、明示的な begin transaction で最低限のロギング・モードの DML を使用するには、次のルールに従います。

- 複数文のトランザクション内で、ロギング・モードが完全または最低限の複数のテーブルで操作する DML 文を使用できます。set dml_logging コマンドを使用して、トランザクションの途中でロギング・モードを変換できますが、後続の DML 文に対する影響は同じトランザクションですでに実行された DML 操作の内容に応じて変わります。トランザクションの開始時にロギング・モードが full である場合は、次の SQL トランザクションが可能です。

```
begin tran
go
delete t1 where ...
go
set dml_logging minimal
go
insert t1 values ...
go
update t2 where ...
go
commit tran
go
```

`delete t1` は完全なロギングで実行されますが、`update t2` は最低限のロギング・モードで実行されます。ロールバックの場合、更新は取り消されませんが、削除はロールバックされます。

- 複数文トランザクションのテーブルで full ロギングまたは minimal ロギングを指定してコマンドを実行すると、同じテーブルに対する後続のコマンドでは、セッションの dml_logging 設定に関係なく同じロギング (full または minimal) を使用する必要があります。上記の例では、insert t1 は full ロギングで実行されています。これは、先行する delete on t1 が full ロギングで実行されたためです。
- 逆に言うと、セッションのロギング・モードをデフォルトのモードにリセットすると、テーブルが同じトランザクションで最低限のロギング・モードで動作していた場合は、DML 文のロギングは再開されません。
- 同じトランザクション内のテーブルごとにロギング・モードが異なると、トランザクションがロールバックされた場合に関連するテーブルの結果に差異がでます。完全なロギングが指定されたテーブルの変更はロールバックされますが、最低限のロギングが指定されたテーブルはコミットされたままです。
- ロギング・モードの選択、およびストアド・プロシージャ内部の set dml_logging コマンドの使用に関するルールは、複数文バッチのルールと同じです。次のようになります。
  - プロシージャが明示的なトランザクションの外側で実行される場合は、各文は個々のトランザクションとして実行されます。
  - プロシージャがトランザクションの内側で実行される場合は、上述の同じルールが適用されます。
- トランザクションの内側でロギング・モードを変更した後で、同じトランザクション内で別のロギング・モードで動作していたテーブルから select を実行する場合は、特に制約はありません。delete t1 は完全なロギング・モードで実行されますが、update t2 は最低限のロギング・モードで実行されます。完全なロギング・モードで動作していたテーブル t1 のロギング・モードが minimal になりましたが、この同じテーブルを読み込むために参照してもエラーになりません。

```
begin tran
go

delete t1 where ...
go

set dml_logging minimal
go

update t2
where t2.c2 = (select c1 from t1 where ...)
go

commit tran
go
```

ストアド・プロシージャと最低限のログを取る DML

セッション内の DML ロギング設定は、呼び出されたシステム・プロシージャから継承されます。また、プロシージャ内で実行される set dml_logging コマンドの動作設定はサブプロシージャに継承されます。プロシージャのスコープを終了した後は、set export_options が on である否かにかかわらず、親セッションまたは親プロシージャ内の dml_logging 設定がリストアされます。

例

次の例は、これらのルールをどのように適用し、プロシージャ内のロギング・モードにどのように影響するかを示します。

例 1　次の例では、プロシージャを実行するユーザはプロシージャ所有者でもあり、かつこのプロシージャの影響を受けるすべてのテーブルの所有者でもあります。

```
create procedure p1 as
begin
    delete t1 where...

    set dml_logging minimal

    update t2 where...
end
go

set dml_logging default
go

exec p1
go
/*
** Exiting from the procedure restores the
** session's setting to what it was before
** calling the procedure, in this case, the
** logging mode will be back to DEFAULT
** (i.e. FULL).
*/
-- This will operate in logged mode now.
delete t2
go
```

1　プロシージャ p1 での実行開始時のロギング・モードは full です。

2　`delete t1` は完全なロギング・モードで実行され、その後でプロシージャのセッション・レベルのロギング・モードは minimal に変更されます。

3　`update t2` は最低限のロギング・モードで実行されます。

4　p1 を終了する際、コントロールが外部 SQL バッチを返すと、ロギング・モード設定が full に戻ります。次の delete t2 は、完全なロギング・モードで実行されます。

例 2 次の例では、セッション・レベルのロギング・モードを minimal に設定してプロシージャ p1 を実行します。delete t1 と update t2 は最低限のロギング・モードで動作します。p1 が終了すると、次の delete t2 も最低限のロギング・モードで動作します。ロギング・モードは p1 が呼び出される前のモードにリストアされるためです。

```
set dml_logging minimal
go

exec p1
go

-- This will operate in minimally logged mode.
delete t2
go
```

プロシージャ内の DML 文のロギング・モードは呼び出し側のセッションまたはプロシージャのセッション・レベル設定に左右されるため、プロシージャ本文の先頭で目的のロギング・モードを明示的に選択することをおすすめします。プロシージャの終了時にモードを default に戻すことができます。

```
create procedure p1 as
begin
    set dml_logging minimal

    delete t1 where ...
    update t2 where ...

    -- Optionally, reset upon exit
    set dml_logging default
end
go
```

ただし、プロシージャ内の DML 文を、ほとんどの場合シングルロギング・モード (たとえば full) で実行するが、ときどき別のモード (最低限のロギング) で実行する必要がある場合は、プロシージャの本文にロギング・モードを含めずに、呼び出し側のプロシージャのセッション・レベル設定または isql セッションでロギング DML 文を制御することをおすすめします。

例 3 DML ロギング設定が影響するのは、セッションのユーザが所有するテーブルだけです。これは、特定のテーブルについて最低限のログを取る DML を実行する必要のあるプロシージャに影響しますが、これらのプロシージャは、テーブルを所有しないユーザによって exec proc 特権を使用して実行されます。

次の例では、Joe は、Mary が所有するテーブルに対して delete を実行するプロシージャ mary.delete_proc を実行します。Joe は set コマンドを使用して最低限のロギングを要求しますが、これを行うことで、プロシージャ内で Mary が所有するテーブルのロギング・モードが影響を受けます。

```
isql -Sservername -Umary -Pmaryspwd
```

```
create procedure mary.delete_proc as
begin
    delete mary.large_table where ...
end
go

grant exec on mary.delete_proc to joe
go

isql -Sservername -Ujoe -Pjoespwd

-- User 'joe' executes the following SQL:
--
set dml_logging MINIMAL
go

exec mary.delete_proc
go
```

例 4　Adaptive Server は、テーブルを所有しないユーザによってプロシージャが実行されているとき、プロシージャ所有者である Mary が特定の文に対して最低限のロギングを使用することを許可しません。プロシージャ内の set dml_logging コマンドは、セッション所有者が所有するテーブルにのみ適用されます。

次の例では、最低限のロギングは `delete mary.large_table` に適用されませんが、ユーザの Joe がデフォルトのロギング設定でプロシージャを実行するときに `update joe.very_large_table` に適用されます。

```
isql -Sservername -Umary -Pmaryspwd

create procedure mary.delete_proc2 as
begin
    set dml_logging MINIMAL

    delete mary.large_table where ...

    update joe.very_large_table where ...
end
go

grant exec on mary.delete_proc2 to joe
go

isql -Sservername -Ujoe -Pjoespwd

exec mary.delete_proc2
go
```

複数の所有者が所有しているテーブルに対して DML 文を実行するプロシージャでは、最低限のログを取る DML の実行対象テーブルは、プロシージャを実行するユーザで決まります。特定のテーブルに対して最低限のログを取る DML を実行するプロシージャを実行できるのは、テーブル所有者または sa_role のあるユーザだけです。

実行中のプロシージャや以前実行したプロシージャのキャッシュされたプランを再コンパイルしても、プロシージャの本文に示される個々の DML コマンドに対して実行時に選択されるロギング・モードには影響しません。テーブルのロギング・モードを変更する可能性のある set コマンドは、DML 文の実行開始時に考慮されます。

トリガでの *set dml_logging* の指定

テーブルに対する DML 操作にアクティブなトリガがある場合は、DML 文は完全にログされます。トリガが作成され、テーブルのロギングが無効な場合は、DML 文が完全なロギング・モードで動作することを示す警告メッセージが表示されます。ただし、トリガは正常に作成されます。

トリガ内の DML 文のロギング・モードは、トリガを起動したユーザによって異なります。トリガを起動したユーザが所有するテーブルは、最低限のロギング・モードで動作できます。トリガを起動したユーザが所有しないテーブルに対して実行される DML 文は、これらのテーブルに対して最低限のロギングを明示的に設定しないかぎり、完全なロギング・モードで実行されます。

ユーザの Mary が所有するオブジェクト m1 に対する削除トリガ delete_trig_m1 があるとします。このトリガは、Joe.j2 や Paul.p3 などの他のテーブルに対しては最低限のロギング・モードで DML 文を実行します。Mary.m1 の delete 特権がユーザの Sally に与えられています。

```
Create trigger Mary.delete_trig_m1 FOR DELETE
On Mary.m1
as
Begin
    Delete Mary.min_logged_table_t3
    WHERE ...

    SET DML_LOGGING MINIMAL

    DELETE Paul.p3 WHERE ...
End
```

ユーザの Sally が Mary.m1 に対して delete 文を実行すると、トリガ Mary.delete_trig_m1 が起動して、Sally のために適切なテーブルの権限チェックが実行されます。Mary がトリガを所有しているため、Adaptive Server は `delete Mary.min_logged_table_t3` の権限チェックを実行しません。このテーブルでは DML 文は最低限のモードで実行されます (このテーブルは、他の方法で最低限のロギングに定義されています)。Sally には Paul.p3 の dml_logging をオフにする権限がないため、set コマンドは暗黙的に有効になりません。次の文 `delete Paul.p3 where...` は完全なロギング・モードで実行します。ユーザの Mary がトリガを実行した場合、DML ロギングの動作は同じ (完全なロギング) です。

ただし、ユーザの Paul が Mary.m1 に対して外部 delete を実行すると、トリガが起動して、トリガ本文の両方の文が最低限のロギング・モードで実行されます。

トリガが起動したら、外部 DML 文の完全なログを取る必要があります。つまり、トリガ本文の同じ外部テーブルに対して最低限のログを取る DML を実行しようとしても、無視されて、DML 文は完全にログされます。

ロギング・モードはビューに対して無効にできません。このため、現在ビューに対して唯一サポートされている instead of トリガは、instead of トリガの指定された DML 文をビューに対して実行する場合に、ビューが参照するベース・テーブルのロギング・モードやセッション・レベルの設定に影響されません。ただし、複数のテーブルに対する完全または最低限のロギング・モードの選択に関するルール、トリガ本文内での set コマンドの使用に関するルール、およびトランザクション・セマンティックのすべてが、instead of トリガ内の DML 文に適用されます。

遅延更新の使用

最低限のロギング操作に対応するテーブルに対してクエリ・オプティマイザが遅延モード更新を選択した場合、該当する文の最低限のロギング設定は無効になり、文は full ロギングの遅延モードで動作します。大量のトランザクション・ロギングを使用して DML 文を遅延モードで生成するアプリケーションにとっては、最低限のロギングのメリットがありません。テーブルの複数文トランザクションに対して遅延モードの操作を実行すると、このテーブルの後続のすべての DML 文は完全にログされます。

診断情報の取得

テーブルのロギング・モードの決定には多くのルールが関係しています。アプリケーション開発者は、Adaptive Server が特定のテーブルに対して現在最低限のログを取る DML コマンドを生成しているかどうか、また、スキーマのタイプ、制約、セッションの設定などについて認識していることが必要です。

object_attr は、セッション・レベル、テーブル・レベル、データベース・レベルの設定に応じて、テーブルの現在のロギング・モードを報告します。『リファレンス・マニュアル：ビルディング・ブロック』を参照してください。

選択したロギング・モードが Adaptive Server によって上書きされたかどうかを判別するには、set print_minlogged_mode_override を使用します。『リファレンス・マニュアル：コマンド』を参照してください。
print_minlogged_mode_override はトレース情報をセッション出力に生成します。参照整合性制約の存在、遅延モードの選択、影響を受けるテーブルの名前、影響を及ぼす規則の説明などの規則によって、テーブルの最低限のログを取るモードが上書きされる文についてレポートします。このスイッチをサーバ全体で有効にして、アプリケーション全体から診断出力を取得します。この出力は print_output_to_log スイッチを使用してエラー・ログにリダイレクトします。

第 4 章

インメモリ・データベースのパフォーマンスとチューニング

この章では、sp_sysmon の結果、モニタ・カウンタ、Backup Server など、インメモリ・データベースおよびリラックス持続性データベースのパフォーマンスとチューニングの特徴について説明します。

メモリ内記憶域キャッシュの設定

Adaptive Server は、名前付きキャッシュおよびバインドを使用するあらゆるタイプのデータベースについて、さまざまなデータベース設定およびキャッシュの設定をサポートします。キャッシュをサイズ変更またはバインドするときは、次の点を考慮してください。

- 大容量のディスク常駐型データベースを小さな名前付きキャッシュにバインドする。これは標準の Adaptive Server 設定で、データベースの内容の一部のみがキャッシュにバインドされます。作業負荷によっては、通常のキャッシュ置換方式によって、かなりのデータベース・ページがリサイクルされます。
- データベース・サイズと同じくらいのキャッシュ・サイズを作成し、キャッシュされるページのリサイクルを減らす。ただし作業負荷によっては、ディスクへの書き込みが増える場合があります。
- テンポラリ・データベース全体をホストできる十分な大きさの名前付きキャッシュにテンポラリ・データベース全体をバインドする。サーバがテンポラリ・データベースに適用するデフォルトの最適化 (および delayed commit などの方法) を使用すると、完全にキャッシュされるテンポラリ・データベースのパフォーマンスを飛躍的に向上できます。

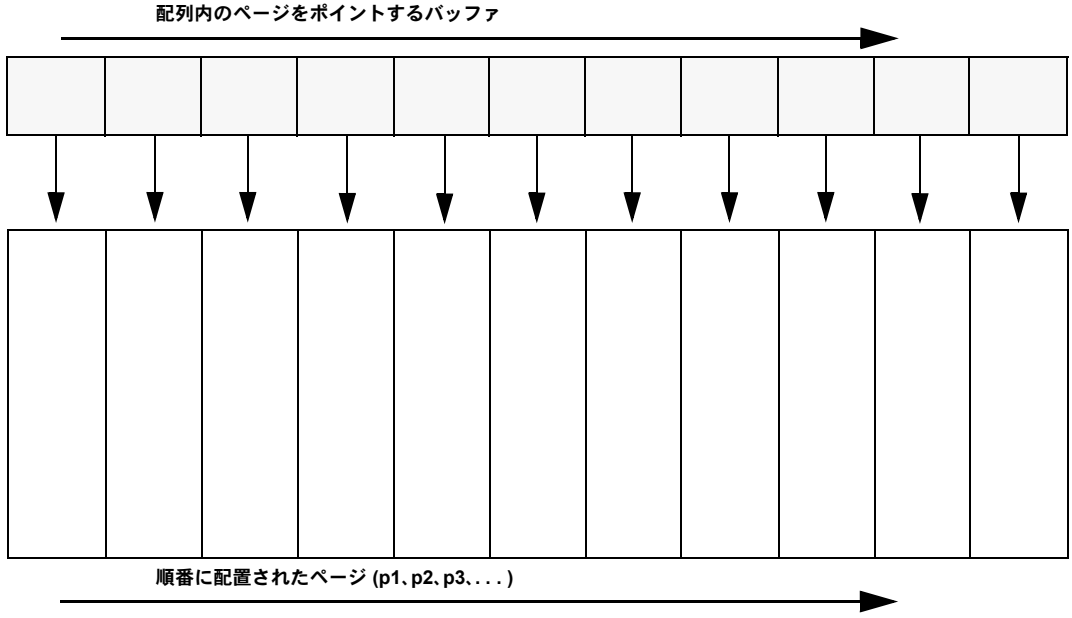
キャッシュ・レイアウト

名前付きキャッシュにバインドされたリラックス持続性データベースとそのオブジェクトでは、通常のキャッシュと同じレイアウトが使用されます。

キャッシュにバインドされたオブジェクトのアクセス・パターンは、通常のキャッシュのレイアウトを示します。Adaptive Server は、厳密な LRU キャッシュ置換方法を使用して、最も長い間使用されていない (LRU) バッファとページのペアを置換用に選択します。Adaptive Server は、ディスクに常駐している名前付きキャッシュにバインドされたオブジェクトのページ (データベース、テーブル、およびインデックス ID で識別) が要求されると、これらをキャッシュにロードします。これらのオブジェクトは、置換用として Adaptive Server によって選択されるまでキャッシュに残ります。Adaptive Server は、LRU 方法または MRU (最も最近に使用された) 方法が使用されているかに応じて、空きバッファのアベイラビリティに準拠するか、または置換可能なバッファに準拠して、このページが読み込まれるキャッシュ内の場所を特定します。Adaptive Server がページをメモリに保持するために使用するバッファは一定ではなく、バッファ・キャッシュ内のバッファの場所もこのキャッシュ設定で変わることがあります。

メモリ内記憶域キャッシュを使用するデータベースはキャッシュにバインドされ、このデータベース内のすべてのオブジェクトとそのインデックスは同じキャッシュを使用します。データベース全体がキャッシュでホストされます。データベース内のすべてのページ (割り付け済みおよび未割り付けの両方を含む) がハッシュされるため、キャッシュ内のページ検索では要求されたページが必ず検出されます。ページは順番に配置され (図 4-1 を参照)、すべてのページがキャッシュ内に常駐するため、バッファとページのペアの位置は変わりません。

図 4-1: インメモリ・データベースに順番に配置されるページ



メモリ内記憶域キャッシュはデフォルト・プールのみをサポートします。デフォルト・プールは、ページの論理読み込み / 書き込みにサーバ・ページ・サイズを使用します。メモリ内記憶域キャッシュでは、ディスクから大容量 I/O を実行する場合に使用する、大容量のバッファ・プールは不要です。実行時の I/O パフォーマンスを上げてページ検索中のキャッシュ・ミスを減らす非同期プリフェッチは、メモリ内記憶域キャッシュでサポートされていません。

データをディスクに格納しないため、メモリ内記憶域キャッシュでは、次のことは不要です。

- 置換方式 (置換方式は各キャッシュで none に定義されています)。キャッシュ内のページ位置は変わりません。
- I/O サポート。つまり、ウォッシュ領域および大容量のプールはサポートされていません。
- バッファ置換の実行。プライベート・バッファでは、デフォルトのデータ・キャッシュが使用される場合があります。
- バッファの動的なハッシュ。データベースの作成時に、すべてのページはハッシュ・テーブルにロードされてハッシュされます。alter database を使用してデータベースのサイズを増やす場合は、実行時にインメモリ・データベースでバッファのハッシュが必要になる場合があります。
- 非同期プリフェッチの実行。すべてのページがキャッシュ・メモリに常駐するためです。

キャッシュ・パーティションはメモリ内記憶域キャッシュでサポートされており、インメモリ・データベース内のページのサブセットをパーティションに分配することでシングル・スピンロックの競合を減らします。パーティションでは、各サブセットは別々のスピンロックにより制御されます。

インメモリ・データベースに対する sp_sysmon の出力

インメモリ・データベースのパフォーマンスをモニタするには、インメモリ・データベースに対して sp_sysmon を実行します。

インメモリ・データベースの Cache Hits の値は常に 100%、Cash Misses の値は常に 0 であることが必要です。

```
Cache: imdb
                              per sec      per xact       count   % of total
-----------------------   ------------  ------------  ----------   ----------
Spinlock Contention                n/a           n/a         n/a      50.9 %

Utilization                        n/a           n/a         n/a      99.9 %

Cache Searches
Cache Hits                     16954.4        5735.8     1152897     100.0 %
Found in Wash                      0.0           0.0           0       0.0 %
Cache Misses                       0.0           0.0           0       0.0 %
-----------------------   ------------  ------------  ----------   ----------
Total Cache Searches            16954.4         5735.8     1152897
```

メモリ内キャッシュには、次に示すようなウォッシュ領域はありません。

```
Buffer Wash Behavior
   Statistics Not Available - No Buffers Entered Wash Section Yet

Cache Strategy
Cached (LRU) Buffers           21225.3         7180.7     1443321      100.0 %
Discarded (MRU) Buffers            0.0            0.0           0        0.0 %
```

『パフォーマンス&チューニング・シリーズ：sp_sysmon による Adaptive Server の監視』を参照してください。

次に示す sp_sysmon のセクションは、メモリ内記憶域キャッシュに適用されません。これはデータベース全体がキャッシュにバインドされ、すべてのデータベース・ページがキャッシュ内に常駐するためです。

- Utilization (使用率)
- Cache Hits (キャッシュのヒット)
- Found in Wash (ウォッシュ内での発見)
- Cache Misses (キャッシュのミス)

- Large I/O Usage (大容量 I/O の使用率)
- プールのターンオーバー

インメモリ・データベースはバッファ・グラブを実行しないため、次のセクションはインメモリ・データベースに適用されません。

- LRU Buffer Grab (LRU バッファの取り込み)
- Grabbed Dirty (取り込まれたダーティ・バッファ)
- キャッシュのターンオーバの総数

インメモリ・データベースはウォッシュ領域を定義しないため、次のセクションはインメモリ・データベースに適用されません。ディスクの読み取りと書き込みは安定状態では発生しません。

- Buffer Wash Behavior (バッファ・ウォッシュの動作)
- キャッシュ方式

インメモリ・データベースは大容量のプールを使用しないため、次のセクションはインメモリ・データベースに適用されません。

- Large I/Os Performed (実行された大容量 I/O)
- Large I/Os Denied (拒否された大容量 I/O)
- Total Large I/O Requests (大容量 I/O 要求の総数)
- 大容量 I/O の詳細

デフォルト・データ・キャッシュのパフォーマンスのモニタ

すべてのページはメモリ内記憶域キャッシュにハッシュされるため、ソート中などに中間ページを保管するために少量のプライベート・バッファを必要とするタスクは、デフォルトのデータ・キャッシュを使用します。バッファをソートするために Adaptive Server でデフォルトのデータ・キャッシュを一時的に使用しているかどうかを判別するには、buf_imdb_privatebuffer_grab モニタ・カウンタを使用します。

```
buf_imdb_privatebuffer_grab  buffer_0          244             510525
```

次のスクリプトは、すべてのモニタリング・グループからモニタ・カウンタを収集します。最初にすべてのモニタ・カウントをクリアしてから、1 分間モニタ・カウントのサンプルを収集して、そのモニタ間隔中に更新されたすべてのモニタ・カウントの値をレポートします。

buf_imdb_privatebuffer_grab の現在の値を確認するには、以下を実行します。

```
dbcc monitor ('clear','all','on')
go
dbcc monitor ('sample','all','on')
go
waitfor delay "00:01:00"
go
dbcc monitor ('sample','all','off')
go
dbcc monitor ('select','all','on')
go
select * from master.dbo.sysmonitors
where field_name = 'buf_imdb_privatebuffer_grab'
go
```

buf_imdb_privatebuffer_grab の値は、中間ソート用のテンポラリ・バッファを必要とし、インメモリ・データベースのテーブルに対して実行されるクエリのデフォルト・データ・キャッシュのバッファ使用率を示します。デフォルト・データ・キャッシュのサイズに準拠して buf_imdb_privatebuffer_grab の値を評価します。

- 取り込まれるバッファの数がデフォルト・データ・キャッシュのサイズと比べて非常に少ない場合、インメモリ・データベースのテーブルに対して実行されるクエリでのテンポラリ・バッファの使用率は大きくありません。
- 取り込まれるバッファの数がデフォルト・データ・キャッシュ内のかなりの数のバッファに相当する場合、デフォルト・データ・キャッシュでインメモリ・データベースに対して実行されるクエリで使用するバッファの負荷が大きいことを示します。一般にこのような現象が発生するのは、デフォルト・データ・キャッシュが非常に小さい場合 (デフォルト・サイズが 8MB など) だけです。

  少ない容量のデフォルト・データ・キャッシュを使用すると、デフォルト・データ・キャッシュを使用する他のアプリケーションのパフォーマンスに影響することがあります。インメモリ・データベースを使用するテンポラリ・バッファの要求と、同じキャッシュを使用する他の同時実行性アプリケーションの要求の両方を満たすように、デフォルト・データ・キャッシュのサイズを増やしてください。

『パフォーマンス&チューニング・シリーズ：sp_sysmon による Adaptive Server の監視』の「第 2 章 sp_sysmon を利用したパフォーマンスのモニタリング」を参照してください。

メモリ内デバイスの物理データの編成

インメモリ・データベースは I/O を使用しないため、物理データの配置を編成する際にデバイス I/O の特性 (I/O デバイスの速度など) を考慮する必要がありません。頻繁にアクセスされるインデックス・ページを高速デバイスに配置したり、使用度の低いオブジェクトのページ (text ページや image ページなど) を低速度のデバイスに配置するといった問題について考慮する必要がありません。ただし、インメモリ・データベースではディスク I/O に起因するその他のボトルネックがなくなることで、スピンロックやラッチの競合などの別のボトルネックの発生率が高くなる可能性があります。

メモリ内デバイスの物理データを編成するときは、次のことを考慮します。

- ラッチ競合を減らすために、独立したデータとログ・デバイスを使用する (ログとデータが混在しないようにインメモリ・データベースを設定する)。
- ログによる領域消費を制御するために、別個のログ・デバイスを使用する。
- オブジェクトによる領域の使用を制限するために、デバイスはセグメント単位で編成する。
- ページ割り付けの競合を減らすために、異なるデバイスにバインドされたセグメントを使用する。

低持続性データベースのパフォーマンスの最適化

低持続性データベースの持続性は no_recovery と at_shutdown に設定されています。低持続性データベースはリカバリされないため、次のようなパフォーマンス上のメリットがあります。

- 低持続性データベースは、位置決めの解除によるユーザ・ログ・キャッシュ (ULC) をフラッシュしないため、トランザクションの多いシステムではトランザクション・スループットが増加します。

  2 つのトランザクションが同じデータロー・ロックを更新しようとすると ULC の位置決めの解除が発生します。これにより、2 番目のトランザクションはデータ・ページを使用する前にログ・レコードを最初のトランザクションの ULC から syslogs にフラッシュします。syslogs への UCL フラッシュが増えるとログの競合も増えるため、トランザクションの多いシステムでのトランザクション・スループットに悪影響を及ぼします。

sp_sysmon の出力の Transaction Management (トランザクション管理) セクションは、ULC Flushes to Xact Log、by Unipin、および by Single Log Record (すべてのアクションは位置決めの解除に関連付けられている) の値が、低持続性データベースでは完全な持続性データベースよりも際立って低いことを示します。これにより、トランザクションの多いシステムでのトランザクション・スループットが増加します。次に示す低持続性データベースの sp_sysmon の出力は、ULC が位置決めの解除からフラッシュされなかったことを示します。

```
ULC Flushes to Xact Log    per sec       per xact    count  % of total
-----------------------  ----------   ------------  -------  ---------
Any Logging Mode DMLs
by End Transaction             40.3            0.0     2416   100.0 %
by Change of Database           0.0            0.0        1     0.0 %
by Unpin                        0.0            0.0        0     0.0 %
by Other                         0.0            0.0        0      0.0 %
```

- 低持続性データベースではトランザクションがディスクに書き込まれないため、commit の実行中にトランザクション・ログのフラッシュが実行されません。

  通常、トランザクション・ログのフラッシュはリラックス持続性データベースには不要です。リラックス持続性データベースは再起動時にリカバリされないため、コミットされたトランザクションの確実なレコードを必要としません。通常、Adaptive Server はトランザクションの完了時にトランザクション・ログをフラッシュしますが、リラックス持続性データベースの場合はバッファ・ウォッシュ時にこれを行います。

  sp_sysmon の出力の Device Activity Detail (デバイス・アクティビティの詳細) セクションは、リラックス持続性データベースのログ・デバイスに対する書き込み数を示します。次に示すのは、ディスク常駐型バージョン 15.0.3 のデータベースに対する sp_sysmon の出力です。

```
Device:
/disk_resident/device/1503esd2/drdb_device.dat
drdb_device               per sec      per xact       count  % of total
------------------   -----------  ------------  ---------  ----------
Reads
  APF                         0.0           0.0          0      0.0 %
  Non-APF                    14.3           0.0       2380      0.2 %
Writes                     9247.3           1.5    1544304     99.8 %
------------------   -----------  ------------  ---------   ----------
Total I/Os                 9261.6           1.5    1546684     100.0 %
```

次に、Adaptive Server バージョン 15.5 のリラックス持続性データベースに対する同じ出力を示します。これはリラックス持続性データベースであるため、書き込みはディスク常駐型データベースの書き込みの約 23% です。

```
Device:
/relaxed_durability/IMDB/devices/ariesSMP/rddb_device.dat
rddb_device                per sec      per xact      count   % of total
-------------------   -----------  ------------  ---------   ----------
Reads
  APF                         0.0           0.0          0      0.0 %
  Non-APF                    14.7           0.0       1031      0.7 %
Writes                     2106.4           0.1     147446     99.3 %
-------------------   -----------  ------------  ---------   ----------
Total I/Os                 2121.1           0.1     148477      99.9 %
```

- 低持続性データベースは、UCL に含まれるログ・トランザクションのログを取りません。各トランザクションのすべてのログ・レコードが UCL に含まれており、コミット後の処理でトランザクションにログ・レコードが不要な場合、Adaptive Server はトランザクションのコミット時に低持続性データベースのトランザクションに関するログ・レコードを破棄します。

  ログ・レコードを転送する必要がないため、ログの競合が減り、これによりトランザクションの多いシステムでのスループットが増えます。一般に、大量のトランザクションではコミット時にすべてのログ・レコードを ULC に含めることができないため、ULC へのトランザクションのロギングは小さなトランザクションが適しています。ULC に格納しきれないレコードのログが取られる大量のトランザクションでは、ULC が満杯になった時点で ULC を syslogs にフラッシュする必要があります。

  コミット時に ULC に小さなトランザクションが含まれている場合も ULC のフラッシュが発生します。これは、コミット後の作業を実行するために、これらのログ・レコードが Adaptive Server に必要なためです。たとえば、コミット後の作業が必要なトランザクションの場合、データベース内の領域の割り付けを解除するトランザクションで ULC のフラッシュが発生します。

  『パフォーマンス＆チューニング・シリーズ：sp_sysmon による Adaptive Server の監視』の「第 2 章 sp_sysmon を利用したパフォーマンスのモニタリング」の「トランザクション・ログへの ULC のフラッシュ」を参照してください。

- 低持続性データベースは、一部のログを取る変更をディスクにフラッシュしません。つまり、対応するログ・レコードは実行された変更を完全に反映しないため、Adaptive Server は完全な持続性のデータ・ページをディスクに時々フラッシュする必要があります。以下は、その例です。

  - クラスタード・インデックスが指定されたテーブル用に分割されたインデックス・ページまたはデータ・ページ
  - ソート

- writetext コマンド
- 高速 bcp
- テーブルのロック・スキームを変更するための alter table
- テーブルの分割スキームまたはテーブルのスキームを変更するための alter table
- 完全なテーブルの reorg rebuild
- alter table...unparition

インメモリ・データベースはディスクを使用しないため、データ・ページをディスクにフラッシュする必要がありません。リラックス持続性データベースはデータベース・リカバリが不要なため、データ・ページをディスクにフラッシュしません。データ・ページをディスクにフラッシュしないため、ディスク I/O の回数が多い場合 (たとえば、sort オペレーションの最後に変更ページがディスクにフラッシュされる場合など) のパフォーマンスを飛躍的に改善できます。

低持続性データベースのその他の改善として、持続性に直接関係しない内部機能の向上があります。これらの管理は不要です。

- データベース・タイムスタンプの更新 (トランザクションの多いデータベースで頻繁に実行されるオペレーション) の改善
- データオンリー・ロック・テーブルのインデックス・スキャンを使用した削除の改善
- 非ユニーク・インデックスを持つデータオンリー・ロック・テーブルへのバルク挿入の改善

これらの改善は、create database または alter database を使用して、持続性が明示的に no_recovery に設定されたテンポラリ・データベースにも当てはまります。ただし、持続性が暗黙的に no_recovery に設定されたテンポラリ・データベースには当てはまりません。

チェックポイント間隔のチューニング

recovery interval in minutes 設定パラメータは、リカバリ時間の長さを決定するだけでなく、Adaptive Server がデータベースに対してチェックポイントを実行する間隔も決定します。Adaptive Server はインメモリ・データベースに対してチェックポイントを実行しませんが、リラックス持続性データベースに対してはチェックポイントを実行して、変更されたすべてのバッファを recovery interval in minutes に従ってディスクからフラッシュします。

バッファ・ウォッシングのプレッシャーを軽減して、置換可能なバッファを維持するには、recovery interval in minutes を使用します。recovery interval in minutes の値が小さいほど、Adaptive Server は頻繁にチェックポイントを実行して、変更されたすべてのバッファをウォッシュします。ただし、バッファ・ウォッシュの実行によりもたらされるメリットと Adaptive Server がチェックポイントを実行する場合に発生するディスク I/O の数を考慮して、双方のバランスを取ることが必要です。

recovery interval in minutes に高い値を設定する場合は、チェックポイントの回数が減ることでディスク I/O の数が少なくなるというメリットと、バッファの要求が「ダーティ」になり、バッファ・ウォッシュの実行時の使用が遅れる可能性の間のバランスを考慮する必要があります。

recovery interval in minutes は、すべてのデータベースでサーバワイドに適用されるため、変更が完全な持続性データベースに与える影響を評価した後でパラメータを変更してください。サーバがクラッシュした後、重大なリカバリの必要な完全な持続性データベースが少なくとも 1 つある場合は、このデータベースをタイミングよくリカバリできるために必要な値 (一般にはデフォルト値の 5) をそのまま使用してください。完全な持続性データベースにほとんど影響しない変更 (サーバに障害が発生した後でリカバリがほとんど不要な変更) では、デフォルト値の 5 分よりも長いリカバリ間隔 (たとえば 30 分) から始めます。変更後、sp_sysmon の Data Cache Management (データ・キャッシュ管理) セクションで Buffers Grabbed Dirt の値またはキャッシュごとの情報を確認します。Buffer Grabbed Dirty の値が大きい場合は、recovery interval in minutes の値を小さくします。

完全な持続性データベースの動作に影響を与えずに、チェックポイントがリラックス持続性データベースに対して実行するディスクへの I/O の回数を減らすには (リカバリ間隔の値に関わらず)、リラックス持続性データベースの no chkpt on recovery データベース・オプションを true に設定します。上記の方法を使用して Buffers Grabbed Dirty の値を評価し、チェックポイントを無効にしても再利用可能なバッファのアベイラビリティに悪影響を及ぼさないことを確認します。

最低限のログを取る DML

最低限のログを取る DML の間、insert、update、delete、および低速 bcp-in コマンドは最低限のロギングで実行されるか、またはログを取られません。内部エラーやユーザ発行のロールバックなどにより文が失敗した場合、すでに作業が完了した部分はコミットされたまま残りますが、ロールバックされません。ただし、最低限のログを取る DML が設定されている場合、Adaptive Server は変更されたテーブルの論理的な一貫性を維持する必要があります (たとえば、データ・ローの挿入をロールバックした場合は、関連するインデックス・ローをロールバックする必要があります)。この一貫性を確保するために、最低限のログを取る DML のコマンドは、基本単位オペレーションの名前付きサブコマンドに分割されます。これらのサブコマンドはすべて、インデックスとテキスト、イメージ、ロー外のカラムなど、単一ローの変更を実行するために必要なデータ変更です。サブコマンド・レベルでの論理的な一貫性を維持するために、Adaptive Server は最低限のログを取る DML のコマンドのログをユーザ・ログ・キャッシュ (ULC) に取ります。サブコマンドが完了すると、Adaptive Server はサブコマンドのログ・レコードを UCL から破棄します。ULC を syslogs にフラッシュする必要はありません。

すべてのサブコマンドのログ・レコードを ULC に格納しきれない場合、Adaptive Server はログ・レコードを破棄できます。一般にこのような現象が発生するのは、DML の影響を受けるデータ・ローが非常に大きい場合、またはテーブルに多くのインデックスが含まれている場合です。Adaptive Server は ULC を破棄できない場合、ULC のログ・レコードを syslogs にフラッシュします。これにより、次のようなことが起こります。

- 処理量の多いシステムのログ競合が増える
- トランザクション・スループットが妨げられる
- 必要なログ領域が増えるため、最低限のログを取る DML を使用することで得られるメリットが相殺される

ULC がほとんどのサブコマンドのログ・レコードを十分に格納できる大きさであることを確認します。ULC はデフォルトのサーバ・ページ・サイズの 2 倍の大きさにすることをおすすめします。

```
declare @ulc_size int
select @ulc_size = @@maxpagesize * 2
exec sp_configure "user log cache size", @ulc_size
```

持続性が明示的に設定されているメモリ内テンポラリ・データベースまたはリラックス持続性テンポラリ・データベースでは、サーバの論理ページ・サイズの 2 倍の大きさでセッション tempdb 固有の ULC を作成することにより、最低限のロギングの効率を上げ、ULC から syslogs へのフラッシュを回避します。

```
declare @ulc_size int
select @ulc_size = @@maxpagesize * 2
exec sp_configure "session tempdb log cache size", @ulc_size
```

一般にこれらのスクリプトは、1 つのサブコマンドのすべての変更をメモリ内全体に格納するようにサイズ指定された ULC を設定するため、同時 DML のパフォーマンスが飛躍的に改善されます。

ULC のサイズを変更するには、Adaptive Server を再起動する必要があります。

最低限のロギングが効率的であるかどうかを判断するには、sp_sysmon の出力の Transaction Management (トランザクション管理) セクションで確認します。

次の例では、Adaptive Server はサブコマンドで再生されたほとんどのログ・レコードを破棄しているため、最低限のログを取る DML が効率的であることを示します。

```
ML-DMLs ULC Efficiency    per sec      per xact       count  % of total
------------------------  -------  ------------  ----------  ----------
Discarded Sub-Commands    33383.9       11087.8     3071323       100.0
Logged Sub-Commands           0.4           0.1          37         0.0
```

Transaction Detail (トランザクションの詳細) セクションは、指定されたサンプルについて、完全なロギング・モードで実行した DML コマンドと最低限のロギング・モードで実行した DML コマンドの概要を示します。次の出力では、Adaptive Server はデータオンリー・ロック・テーブルに対して、ほとんどすべての挿入を最低限のロギング・モードで実行しました。

```
Transaction Detail             per sec      per xact     count  % of total
-----------------------   ------------  ------------  -------   ----------
Inserts
  Fully Logged
   APL Heap Table                57.8         173.5       694        0.7 %
   APL Clustered Table            0.0           0.0         0        0.0 %
   Data Only Lock Table           0.7           2.0         8        0.0 %
   Fast Bulk Insert               0.0           0.0         0        0.0 %
  Minimally Logged
   APL Heap Table                 0.0           0.0         0        0.0 %
   APL Clustered Table            0.0           0.0         0        0.0 %
   Data Only Lock Table        7775.8       23327.5     93310       99.3 %
```

Transaction management (トランザクション管理) セクションは、Adaptive Server がサブコマンドのログを破棄するのではなく、それらのログをどのような方法で取るかに関する詳細情報を示します。次の出力は、ULC がトランザクション・ログをフラッシュする原因となるイベントを、完全なロギングと最低限のログを取る DML に分けて ULC のフラッシュを示します。Minimally Logged DMLs (最低限のログを取る DML) セクションでは、ほとんど同量のフラッシュが Full ULC と By End of Sub-Command により発生しました。

```
ULC Flushes to Xact Log         per sec      per xact     count  % of total
-----------------------   ------------  ------------  -------   ----------
Any Logging Mode DMLs
by End Transaction                0.3           1.0         4       11.1 %
by Change of Database             0.0           0.0         0        0.0 %
by Unpin                          0.0           0.0         0        0.0 %
by Other                          0.0           0.0         0        0.0 %
```

```
Fully Logged DMLs
  by Full ULC                       0.2           0.5        2          5.6 %
  by Single Log Record              0.0           0.0        0          0.0 %

Minimally Logged DMLs
  by Full ULC                         1.3           4.0       16         44.4 %
  by Single Log Record                0.0           0.0        0          0.0 %
  by Start of Sub-Command             0.0           0.0        0          0.0 %
  by End of Sub-Command               1.2           3.5       14         38.9 %
------------------------     ------------  ------------  -------   ----------
 Total ULC Flushes                   3.0           9.0       36
```

Minimally Logged DMLs (最低限のログを取る DML) で、ULC Flushes to Xact Log の by Full ULC の `count` カラムの値が、影響を受けるローの数と比較して高い場合は、user log cache 設定パラメータまたは session tempdb log cache size 設定パラメータの値を大きくします。

次の出力は、最低限のログを取るコマンドの ULC オペレーションの効率を示します。ほとんどすべてのロギング・アクティビティが ULC に格納されて、syslogs へのフラッシュはほとんどないため、最低限のロギングで発生するロギング・オーバーヘッドはわずかです。

```
ML-DMLs ULC Efficiency         per sec      per xact     count  % of total
------------------------   ------------  ------------  -------   ----------
Discarded Sub-Commands          7774.7       23324.0    93296        100.0
Logged Sub-Commands                1.2           3.5       14          0.0
------------------------   ------------  ------------  -------
Total ML-DML Sub-Commands       7775.8       23327.5     93310
```

インメモリ・データベースのダンプとロード

リラックス持続性データベースをダンプおよびロードする方法は、完全な持続性データベースと同じです。Backup Server は、リラックス持続性データベースのダンプ中はディスクベースのデータベース・デバイスから直接読み取りを行い、ロード中はダンプ・アーカイブからページをコピーして、これらのデバイスに直接書き込みを行います。リラックス持続性データベースのダンプおよびロードのパフォーマンスは、ストライプ・デバイスを使用して改善できます。

インメモリ・データベースはディスク・デバイスを使用しないため、Backup Server はダンプ中に Adaptive Server の共有メモリから直接ページを読み取り、アーカイブ・メディアに書き込みます。ロード中、Backup Server はアーカイブ・メディア (テープなど) からページを読み取り、メモリ内記憶域キャッシュのページにデータを直接書き込みます。インメモリ・データベースはサーバのページの読み込みおよび書き込みにディスク I/O を使用しないため、一般に、ダンプおよびロードのパフォーマンスは、同じ仕様のディスクベースのデータベースよりインメモリ・データベースの方が優れています。

dump コマンドおよび load コマンドの実行中、Backup Server はストライプごとに 1 つの CT ライブラリ接続を Adaptive Server に対して開き、Adaptive Server の共有メモリからデータベース・ページを直接読み取るための通信チャネルを作成します。

Backup Server は Adaptive Server の共有メモリからページを直接読み取っている間も、Adaptive Server で同時に行われているアクティブなタスクと直接同期します。load database リカバリをサポートするために、dump オペレーションの実行中、Adaptive Server は「低持続性データベースのパフォーマンスの最適化」(45 ページ) で説明した方法を無効にします。これにより、dump database コマンド実行中のトランザクション・アクティビティのパフォーマンスが低減する場合があります。

スピンロック競合とネットワーク接続のチューニング

インメモリ・データベースはディスク I/O を使用しないため、遅延時間および競合の問題はディスク常駐型データベースに比べて少なくなります。ただし、作業負荷の大きな環境では、インメモリ・データベースはスピンロック競合とメモリ内アクセスの問題が発生する場合があります。

- キャッシュのスピンロック競合 ― 作業負荷の大きな環境では、メモリ内キャッシュのボトルネックになる可能性があります。キャッシュ・パーティションの数を 64 以上に増やすことを検討してください。キャッシュ・パーティションの数を増やすために必要な追加のメモリ・リソースはごくわずかで、キャッシュ・マネージャのスピンロック競合を減らすことでパフォーマンスが改善されます。
- オブジェクト・マネージャのスピンロック競合 ― アプリケーションが少数のオブジェクトに頻繁にアクセスする場合は、メタデータ構造のスピンロック競合が見られる場合があります。このスピンロック競合は sp_sysmon でレポートされます。

  dbcc tune 'des_bind' を使用して、頻繁にアクセスされるオブジェクトの記述子をバインドすると、これらの記述子が取り除かれることはありません。頻繁に使用されるいくつかのオブジェクトの記述子をバインドすることで、全体のメタデータ・スピンロック競合を大幅に減らし、パフォーマンスを改善できます。

ロック・マネージャ・ハッシュテーブルのスピンロック率の競合の改善

インメモリ・データベースのスループットにより、ロック・マネージャ・ハッシュテーブルのスピンロック率の競合が発生する場合があります。テーブル・ロック・ハッシュテーブル、およびページ／ロー・ロック・ハッシュテーブルのスピンロックが、競合に大きく影響することがあります。

sp_sysmon の Lock Management (ロック管理) セクションは、ハッシュテーブルのハッシュ・バケットを制御するスピンロックの競合率を示します。

```
Lock Management
---------------

Lock Summary                    per sec     per xact     count     % of total
------------------------  ------------  ------------  --------    ------------
Total Lock Request           285063.3          43.0  17103795           n/a
Avg Lock Contention            1857.3           0.3    111435         0.7 %
Cluster Locks Retained            0.0           0.0         0         0.0 %
Deadlock Percentage               0.1           0.0         8         0.0 %

Lock Detail                     per sec     per xact     count     % of total
------------------------  ------------  ------------  --------    ------------
Table Lock Hashtable
  Lookups                    130160.4          19.6   7809622           n/a
  Avg Chain Length                n/a           n/a   0.00000           n/a
  Spinlock Contention             n/a           n/a       n/a          4.6 %
[...]
Page & Row Lock HashTable
  Lookups                    268968.1          40.6  16138085           n/a
  Avg Chain Length                n/a           n/a   1.03330           n/a
  Spinlock Contention             n/a           n/a       n/a          4.8 %
```

一般に、この競合が 4% を上回る場合は、ハッシュ・バケットに対するスピンロックの割合を減らします。1 つのスピンロックで制御するハッシュ・バケット数を管理する設定オプションのデフォルト値は、次のようになります。

- lock table spinlock ratio (デフォルト値 20) は、Table Lock Hashtable (テーブル・ロック・ハッシュテーブル) セクションの出力に影響します。
- lock spinlock ratio (デフォルト値 85) は、Page & Row Lock Hashtable (ページ/ロー・ロック・ハッシュテーブル) セクションの出力のスピンロック競合に影響します。

スピンロック競合が著しい場合は、lock table spinlock ratio および lock spinlock ratio の値を減らすと、実行時パフォーマンスを改善できます。少ないハッシュ・バケットを制御しているスピンロックのメモリ・オーバーヘッドを追加しても大きな差はありません。まず、設定パラメータの値を半分に減らします。スピンロックの競合が極端な場合 (10% を超える場合) は、該当する設定オプションを非常に小さな値 (たとえば 3 ～ 5) にまで減らすと、スピンロック・オーバーヘッドのパフォーマンスのボトルネックを取り除くことができる場合があります。

ネットワーク接続数の決定

デフォルトでは、Adaptive Server が使用するネットワーク・リスナは 1 つで、任意のエンジンで実行できます。Adaptive Server が接続要求を受け取ると、接続を受け入れるエンジンがその接続のネットワーク・エンジンになります。また、対応するタスクがネットワーク I/O を実行する場合は、このエンジンにマイグレートする必要があります。

多くのクライアント接続が同時に行われると、Adaptive Server は接続要求の間でリスナをスケジュールできずに (タイムスライスの不足によりスケジュールできない場合を除く), すべての接続を 1 つのエンジンに受け入れることがあります。この場合、対応するあらゆるタスクはこのエンジン上で実行してネットワーク I/O を実行する必要があるため、このエンジンがボトルネックになります。Adaptive Server がネットワーク I/O をどのように分配しているかを特定するには、Network I/O Management (ネットワーク I/O 管理) セクションの sp_sysmon を使用します。次の例では、Adaptive Server はネットワーク I/O を均等に分配していません。Engine 2 がネットワーク I/O の 88% 以上を使用しているのに対して、他のエンジンはほとんど 0% に近い使用率です。

```
Network I/O Management
----------------------

Total Network I/O Requests       7301.5           1.4      438092
n/a
Network I/Os Delayed              0.0           0.0           0
0.0 %

Total TDS Packets Received      per sec     per xact     count      % of total
--------------------------  ------------  ------------  --------    ------------
Engine 0                          308.8           0.1    18528          7.7 %
Engine 1                          163.6           0.0     9818          4.1 %
Engine 2                         3558.8           0.7   213527         88.3 %
Engine 3                            0.0           0.0        2          0.0 %
Engine 4                            0.0           0.0        0          0.0 %
Engine 5                            0.0           0.0        0          0.0 %
--------------------------  ------------      --------  -------
Total TDS Packets Rec'd          4031.3           0.8   241875

Avg Bytes Rec'd per Packet          n/            n/a      136
n/a
-------------------------------------------------------------------------------



Total TDS Packets Sent          per sec     per xact     count     % of total
--------------------------  ------------  ------------  --------    ------------
Engine 0                          308.8           0.1    18529          7.7 %
Engine 1                          163.6           0.0     9818          4.1 %
Engine 2                         3558.9           0.7   213531         88.3 %
Engine 3                            0.0           0.0        2          0.0 %
Engine 4                            0.0           0.0        0          0.0 %
```

```
Engine 5                          0.0           0.0         0          0.0 %
------------------------  -----------  ------------  --------
Total TDS Packets Sent            4031.3            0.8    241880
```

不均衡なネットワーク I/O 使用率を解決するには、複数のネットワーク・リスナを使用して、異なるエンジン (通常は 1 つのエンジンに 1 つのリスナ) をバインドします。各ネットワーク・リスナにバインドするクライアントの数を決定するには、各リスナがほぼ同じ数の接続を受け入れるようにクライアント接続を分けます。たとえば、ネットワーク・リスナが 6 つでクライアントが 60 ある場合は、クライアントを 10 台ずつのグループに分けて、各グループを 1 つのリスナに接続します。

ネットワーク・リスナを上記のように均等化した後の sp_sysmon の出力は、次のようになります。

```
Network I/O Management
----------------------

Total Network I/O Requests       8666.5         1.3     519991            n/a
n/a
Network I/Os Delayed                0.0         0.0          0          0.0 %

Total TDS Packets Received      per sec     per xact     count      % of total
------------------------  ------------  ------------  --------    ------------
Engine 0                         893.4           0.1     53602         17.8 %
Engine 1                         924.5           0.1     55468         18.5 %
Engine 2                         701.9           0.1     42113         14.0 %
Engine 3                         906.0           0.1     54358         18.1 %
Engine 4                         896.1           0.1     53763         17.9 %
Engine 5                         683.8           0.1     41028         13.7 %
------------------------  ------------       --------  -------
Total TDS Packets Rec'd          5005.5           0.8   300332

Avg Bytes Rec'd per Packet          n/            n/a      136
n/a
-------------------------------------------------------------------------------

Total TDS Packets Sent          per sec     per xact     count     % of total
------------------------  ------------  ------------  --------    ------------
Engine 0                         893.3           0.1     53595         17.8 %
Engine 1                         924.5           0.1     55467         18.5 %
Engine 2                         701.9           0.1     42113         14.0 %
Engine 3                         905.9           0.1     54355         18.1 %
Engine 4                         896.1           0.1     53763         17.9 %
Engine 5                         683.8           0.1     41026         13.7 %
------------------------   -----------  ------------  --------
Total TDS Packets Sent            4031.3         0.8    241880
```

不均衡なネットワーク・リスナはインメモリ・データベースだけに限らず、ディスク常駐型データベースでも発生します。ただし、メモリ内常駐型データベースはディスク I/O を使用しないため、一般にディスク常駐型データベースよりもスループットが向上します。スループットが向上し、ディスク I/O 遅延時間がない場合、インメモリ・データベースではディスク常駐型データベースよりも多くの作業が実行されます。つまり、より多くのネットワーク I/O が実行されるため、不均衡なネットワーク・リスナに負荷がかかることでボトルネックの重大度が増すことになります。

索引

お

き

こ

さ

し

す

せ

そ

た

ち

て

と

な

ね

は

ひ

ふ

め

も

ら

り

ろ