SYBASE®
An SAP® Company

パフォーマンス＆チューニング・シリーズ：
物理データベースのチューニング

Adaptive Server® Enterprise

15.7

ドキュメント ID：DC01069-01-1570-01
改訂：2011 年 9 月

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

目次

第 1 章

データの物理的配置の制御

この章では、テーブルとインデックスの配置を制御してパフォーマンスを向上させる方法について説明します。

物理データベースのチューニングを最大活用するには、論理デバイスと物理デバイスの違いを理解する必要があります。

- 物理ディスクや物理デバイスは、データを格納するハードウェアである。
- データベース・デバイス (論理デバイス) は、Adaptive Server® で使用するために (disk init コマンドで) 初期化された物理ディスクの一部または全体である。データベース・デバイスはオペレーティング・システム・ファイル、ディスク全体、ディスク・パーティションのいずれかになる。

  ディスクとファイルの使用法に関する特定のオペレーティング・システムの制約については、使用しているプラットフォーム用の『インストール・ガイド』と『設定ガイド』を参照してください。
- セグメントは、データベースが使用するデータベース・デバイスの名前付きのコレクションである。セグメントを構成する複数のデータベース・デバイスは、別々の物理デバイスに配置できる。
- パーティションはテーブルのサブセットである。パーティションは、個別に管理できるデータベース・オブジェクトである。分割テーブルは分けることができるので、複数のタスクが同時にアクセスできる。特定のセグメントにパーティションを配置できる。各パーティションが別々のセグメントにあり、各セグメントが独自のデータベース・デバイスを所有している場合、これらのテーブルにアクセスするクエリは、並列処理の向上というメリットが得られる。パーティションの作成と使用の詳細については、『リファレンス・マニュアル：コマンド』と『Transact-SQL ユーザーズ・ガイド』の「create table」を参照。

デバイス、セグメント、パーティションの詳細については、sp_helpdevice、sp_helpsegment、sp_helpartition を使用してください。

オブジェクト配置の制御によるパフォーマンスの向上

Adaptive Server では、物理記憶デバイス全体のデータベース、テーブル、インデックスの配置を制御できるので、多数のデバイスやコントロール間でディスクの読み書きを均等化して、パフォーマンスを向上させることができます。たとえば次のことを実現できます。

- データベースのデータ・セグメントを特定のデバイス (複数可) に配置して、データベースのログを別々の物理デバイスに格納できるので、ログの読み書きがデータ・アクセスの妨げにならない。
- 複数のデバイス間に大容量かつ使用頻度の高いテーブルを分散配置できる。
- 特定のデバイスに特定のテーブルまたはノンクラスタード・インデックスを配置できる。たとえば、複数のデバイスにわたるセグメントにテーブルを配置し、別のセグメントにテーブルのノンクラスタード・インデックスを配置できる。
- あるテーブルの text および image ページ・チェーンを、テーブルとは別のデバイスに配置できる。テーブルは実データ値へのポインタを別のデータベース構造に格納するため、text または image カラムへの 1 回のアクセスに必要な I/O 回数は少なくとも 2 回になる。
- 別々の物理ディスク上にあるパーティション間でテーブルを均等に分配して、並列クエリ・パフォーマンスを最適化し、insert と update のパフォーマンスを向上させる。

多くのディスク I/O を実行するマルチユーザ・システムとマルチ CPU システムでは、物理デバイスと論理デバイスに関する問題および複数のデバイス間での I/O の分散に特に注意してください。

- 論理デバイスと物理デバイス間でオブジェクトをバランスよく配置できるように計画する。
- ディスク・コントローラなどの物理デバイスを十分に用意して、物理的な帯域幅を確保する。
- 論理デバイスを増やすと、内部 I/O キューの競合を削減できる。
- 使用するパーティションの数を決定して、並列スキャンを可能にし、クエリ・パフォーマンスの目標を達成する。

適切でないオブジェクト配置の識別

オブジェクトをより適切に配置することで、以下の場合に使用しているシステムにメリットをもたらすことができる可能性があります。

- シングルユーザ環境のパフォーマンスは良好だが、複数のプロセスを処理するときに応答時間が著しく長くなる。
- ミラーリングされたディスクへのアクセス時間が、ミラーリングされていないディスクへのアクセス時間の 2 倍もかかる。
- 頻繁にアクセスされるオブジェクト (ホット・オブジェクト) は、これらのオブジェクトが含まれているテーブルを使用するクエリのパフォーマンスを低下させる。
- メンテナンス作業に時間がかかる。
- ディスク領域を他のデータベースと共有している場合に、tempdb のパフォーマンスが低下する。ほとんどのシステム・プロシージャとアプリケーションは、その作業領域として tempdb を使用しているため、tempdb が同じディスクを他のデータベースと共有している場合は、悪影響が及ぶ。
- 使用頻度の多いテーブルで insert のパフォーマンスが低下する。
- 分割またはデバイス上のデータ・ページのバランスが悪いため、並列で動作するクエリのパフォーマンスが低下する。または、極端にバランスが悪いため、クエリが逐次動作する。

ディスクの競合による問題など、オブジェクトの配置に関する問題が発生する場合は、以下の点を確認して修正してください。

- ランダムアクセス (データとインデックスに対する I/O) とシリアルアクセス (ログ I/O) のプロセスが同じディスクを使用している。
- データベースのプロセスとオペレーティング・システムのプロセスが同じディスクを使用している。
- シリアル・ディスク・ミラーリング。
- データベースのロギングや監査を、データ保存に使用するディスクと同じディスクで実行している。

データ配置の変更時の *sp_sysmon* の使用

パフォーマンスに関する問題の原因が、物理デバイス間のデータ配置によるものかどうかを調べるには、sp_sysmon を使います。チューニング中に sp_sysmon の出力をすべてチェックし、変更箇所がすべてのパフォーマンス・カテゴリにどのように影響するかを確かめます。

以下に関連する出力に特に注目してください。

- I/O デバイス競合
- 全ページロック・ヒープ・テーブル
- ヒープ上での最終ページ・ロック
- ディスク I/O 管理

『sp_sysmon による Adaptive Server の監視』を参照してください。

I/O パフォーマンスの向上

Adaptive Server の I/O パフォーマンスを向上させるには、以下のことを試してください。

- ディスク間でデータを分散させて、I/O 競合を防ぐ
- サーバワイド I/O をデータベース I/O から隔離する
- 更新頻度の高いデータベースのデータ記憶領域とログ記憶領域を分離する
- ランダム・ディスク I/O をシーケンシャル・ディスク I/O から隔離する
- 分離した物理ディスク上にミラー・デバイスを置く
- パーティションを使用してデバイス間にテーブル・データを分配する

ディスク間でデータを分散させて、I/O 競合を防ぐ

I/O 競合を防ぐデータ記憶領域を複数のディスクと複数のディスク・コントローラ間に分散してボトルネックを防ぎます。

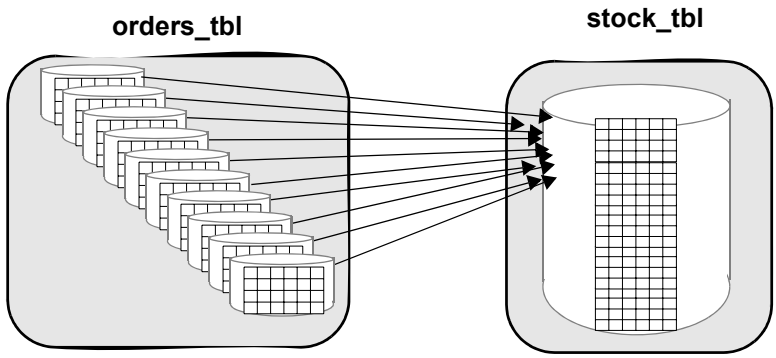
- パフォーマンスの要件が高いデータベースを別々のデバイスに配置する。また、可能であれば、他のデータベースとは別のコントローラを使用する。必要に応じて、重要なテーブルにはセグメント、並列クエリにはパーティションを使用する。
- 使用頻度とジョイン頻度の高いテーブルを別々のディスクに配置する。
- セグメントを使って、テーブルとインデックスを専用のディスクに配置する。

並列ジョイン・キューで物理的な競合を防ぐ

図 1-1 は、2 つのテーブル orders_tbl と stock_tbl のジョインについて説明しています。使用可能なワーカー・プロセスは 10 あります。orders_tbl は、10 個の異なる物理デバイスに 10 の分割を持つ、ジョインの外部テーブルです。stock_tbl は分割されていません。orders_tbl ではワーカー・プロセスにアクセス競合の問題が発生していますが、各ワーカー・プロセスは stock_tbl をスキャンする必要があります。テーブル全体がキャッシュに収まらない場合は、物理 I/O の競合があると考えられます。最悪の場合は、stock_tbl が存在する物理デバイスに 10 のワーカー・プロセスがアクセスしようとします。stock_tbl テーブル全体を含む名前付きキャッシュを作成すると、物理 I/O の競合を防ぐことができます。

物理 I/O 競合を低減または排除する別の方法として、orders_tbl と stock_tbl の両方を分割し、それらの分割を異なる物理デバイスに分配します。

図 *1-1:* 異なる物理デバイス上のテーブルのジョイン



サーバワイド I/O をデータベース I/O から隔離する

頻繁に I/O が発生するシステム・データベース (たとえば tempdb や sybsecurity) を、アプリケーション・データベースとは別の物理ディスクとコントローラに配置します。

tempdb

これは、サーバのすべてのプロセスに影響し、ほとんどのシステム・プロシージャで使用される使用頻度の高いデータベースで、マスタ・デバイスに自動的にインストールされます。さらに領域が必要な場合は、tempdb を他のデバイスに拡張できます。tempdb がアクティブであると予想される場合は、他の重要なデータベース・アクティビティに使用されていない最速のディスクに配置します。

一部の UNIX システムでは、オペレーティング・システム・ファイルへの I/O はロー・デバイスへの I/O よりはるかに高速です。シャットダウン後、tempdb はリカバリされるのではなく常に再作成されるので、tempdb をロー・デバイスではなくオペレーティング・システム・ファイルに移動すると、パフォーマンスを向上できる場合があります。使用しているシステムでこれを試してみてください。

tempdb に推奨される配置の詳細については、「第 6 章 テンポラリ・データベース」を参照してください。

sybsecurity

監査システムを有効にすると、sybsecurity データベース内の sysaudits テーブルに I/O が頻繁に実行されます。アプリケーションが多くの監査を実行する場合は、応答時間が最速でなくてもよいテーブル用に使用されるディスクに sybsecurity を配置します。可能であれば、専用のデバイスに sybsecurity を配置するのが理想的です。

スレッショルド・マネージャを使用して空き領域を監視し、監査データベースが満杯になった場合にもユーザ・トランザクションが中断しないようにします。適切なスレッショルドの判断については、『システム管理ガイド：第 2 巻』の「第 16 章スレッショルドによる空き領域の管理」を参照してください。

トランザクション・ログを独立したディスクに保管する

トランザクション・ログを別のセグメントに配置して、ログが他のオブジェクトとディスク領域を競い合うのを防ぎます。ログを別の物理ディスクに配置すると、次のようなメリットがあります。

- I/O 競合を減らすことで、パフォーマンスが向上する。
- データ・デバイス上でハードディスクがクラッシュしても、完全なリカバリが保証される。
- 非同期プリフェッチ要求を、競合しないでログ・デバイスおよびデータ・デバイスの両方で同時に先読みできるため、リカバリの速度が向上する。

トランザクション・ログをデータと同じデバイスに配置する前に、create database と alter database の両方に with override を使用する必要があります。

ログ・デバイスは、更新アクティビティが大量に発生するシステムに対しては大量の I/O を実行します。トランザクションがコミットし、遅延更新を遅延操作と置き換えるためにメモリにログ・ページを読み込む必要があるときに、Adaptive Server はログ・ページをディスクに書き込みます。

ログとデータが同一のデータベース・デバイスにあると、ログ・ページを保管するために割り付けられるエクステントどうしが連続しなくなり、ログ・エクステントとデータ・エクステントが混在します。ログが専用デバイスに配置されると、エクステントが連続して割り付けられる可能性が高くなるため、ディスク・ヘッドの移動距離とシーク回数が削減され、より高い I/O レートが維持されます。

Adaptive Server は、各ユーザのログ・レコードをユーザ・ログ・キャッシュにバッファして、メモリ上のログ・ページに書き込むときの競合を減らします。ログとデータが同一デバイスにある場合には、ユーザ・ログ・キャッシュのバッファ機能は無効になり、SMP システムのパフォーマンスが大幅に低下します。

『システム管理ガイド：第 1 巻』の「第 6 章ディスク・リソースについての概要」を参照してください。

別のディスク上にデバイスをミラーする

ディスク・ミラーリングは、Adaptive Server がデータベース・デバイス全体の内容を複製する高可用性機能です。

『システム管理ガイド：第 2 巻』の「第 2 章ディスク・ミラーリング」を参照してください。

データをミラーリングする場合は、ミラーによるパフォーマンスの低下が最小限に抑えられるように、ミラーをミラーリング元デバイスとは別の物理ディスクに配置します。ディスク・ハードウェア障害によって物理ディスク全体が損失したり、使用不可能になることがよくあります。

ミラーリングを使用しない場合や、オペレーティング・システムのミラーリングを使用する場合は、設定パラメータ disable disk mirroring を 1 に設定すると、パフォーマンスが若干向上する可能性があります。

ミラーリングを実行すると、書き込みが 2 つのディスクに対して逐次または同時に実行されるため、ディスクの書き込み完了までの時間が長くなります。ディスク・ミラーリングは、データの読み込みに必要な時間には影響しません。

ミラーリングされたデバイスは、ディスク書き込み時に次の 2 つのモードのいずれかを使います。

- 非逐次モード － ミラーリングされていない書き込みよりも、書き込みの完了に時間がかかる場合がある。非逐次モードでは、両方の書き込みが同時に開始し、Adaptive Server は両方が完了するまで待機する。非逐次書き込みの完了に要する時間は、2 つの I/O 時間の長い方になる。
- 逐次モード － 非逐次モードよりデータの書き込みに必要な時間がさらに長くなる。Adaptive Server は最初の書き込みを開始し、それが完了してから 2 番目の書き込みを開始する。所要時間は 2 つの I/O 時間の合計である。

逐次モードの使用

逐次モードはパフォーマンスに影響しますが、書き込み中に発生する障害から保護するので、デフォルトのモードです。

逐次モードは、最初の書き込みが完了してから 2 番目の書き込みを開始するため、1 つの障害が 2 つのディスクに影響することはありません。非逐次モードを使用すると、パフォーマンスは向上しますが、障害が発生して 2 つの書き込みに影響すると、データ損失の恐れがあります。

警告！ ミラーリングしているデータベース・システムに絶対的な信頼が要求される場合は、逐次モードを使用してください。

セグメントの使用法

セグメントとは、1 つまたは複数の論理デバイスを指すラベルです。セグメントを使用してスループットを向上させるには、以下のようにします。

- 並列クエリ・パフォーマンスのために分割されたテーブルを含むサイズの大きいテーブルをディスク間で分割する。
- テーブルとテーブルのノンクラスタード・インデックスをディスク間で分離して保存する。
- テーブルのパーティションとインデックスをディスク間で分離して保存する。
- テキスト・ページ・チェーンとイメージ・ページ・チェーンを、テキスト値を示すポインタが保存されているテーブルとは別のディスクに配置する。

また、セグメントを使用して、領域の使用状況を次のように制御できます。

- テーブルやパーティションは、セグメントに割り付けたサイズを超えることができない。セグメントを使用すると、テーブルやパーティションのサイズを制限できる。
- あるセグメント上のテーブルまたはパーティションは、別のセグメント上のオブジェクトに割り付けられている領域を使用できない。
- スレッショルド・マネージャは領域の使用状況をモニタする。

セグメントにオブジェクトを作成する

各データベースは、データベース作成時にシステムが作成する 3 つのセグメント (system、log segment、default) を含む、最大 32 セグメントを使用できます。

テーブルとインデックスはセグメント上に保管されます。セグメントを指定せずに create table または create index を実行すると、オブジェクトはデータベースの default セグメントに格納されます。これらのコマンドのどちらかでセグメントを指定すると、そのセグメントにオブジェクトが作成されます。sp_placeobject システム・プロシージャを使用すると、以降の領域の割り付けがすべて指定したセグメントで行われて、テーブルが複数のセグメントにわたるように設定できます。

システム管理者がデバイスを disk init で初期化し、デバイスをデータベースに割り付けます。あるいは、データベース所有者が create database または alter database を使用してこれを実行できます。

データベースにデバイスが割り付けられると、データベース所有者またはオブジェクト所有者はデバイスのセグメントの作成およびオブジェクト配置を実行できます。

ユーザ定義セグメントを作成する場合は、create table または create index コマンドを次のように使用して、テーブル、インデックス、パーティションをそのセグメントに配置できます。

```
create table tableA(...) on seg1
create nonclustered index myix on tableB(...)
      on seg2
```

この例では、テーブル fictionsales を作成します。これは date カラムの値に従った範囲で分割されます。

```
create table fictionsales
(store_id int not null,
order_num int not null,
date datetime not null)
partition by range (date)
(q1 values <= ("3/31/2005") on seg1,
q2 values <= ("6/30/2005") on seg2,
q3 values <= ("9/30/2005") on seg3,
q4 values <= ("12/31/2005") on seg4)
```

重要なテーブルのロケーションを制御することによって、ディスク間にこれらのテーブルとインデックスを分散できます。

テーブルとインデックスを分離する

セグメントを使用すると、テーブルを 1 つのディスク・セットに配置し、ノンクラスタード・インデックスを別のディスク・セットに配置できます。クラスタード・インデックスをデータ・ページと異なるセグメントに配置することはできません。on *segment_name* 句を使用してクラスタード・インデックスを作成すると、指定したセグメントにテーブル全体が移動し、そのセグメントにクラスタード・インデックス・ツリーが構築されます。

分離したセグメントにノンクラスタード・インデックスを配置して、パフォーマンスを向上できます。

大きなテーブルをデバイス間で分割する

セグメントは複数のデバイスにわたって存在できるため、これを使用してデータを 1 つまたは複数のディスクに分散できます。使用頻度の高い大きいテーブルの場合に、I/O 負荷の分散に役立ちます。並列クエリの場合は、パーティションベースのスキャン中の I/O 並列処理に備え、複数のデバイスにわたってセグメントを作成することが重要です。

『システム管理ガイド：第 2 巻』の「第 8 章セグメントの作成と使用」を参照してください。

テキスト記憶領域を独立したデバイスに移動する

テーブルに含まれるデータ型が text、image、またはロー外にある Java の場合は、テーブルはデータ値を示すポインタを保管します。実データは、LOB (ラージ・オブジェクト) チェーンという別のリンク先のページのリストに格納されます。

LOB 値の書き込みまたは読み込みには、少なくとも 2 回のディスク・アクセスが必要です。最初のディスク・アクセスはポインタに対する読み込みまたは書き込みのためであり、2 回め以降はデータの読み込みまたは書き込みのためです。アプリケーションが頻繁に LOB 値を読み書きする場合は、LOB チェーンを分離した物理デバイスに配置するとパフォーマンスを向上できます。LOB チェーンは、ほかのアプリケーションに関連するテーブル・アクセスまたはインデックス・アクセスの発生頻度が低いディスクに隔離します。

LOB カラムのあるテーブルを作成すると、LOB データを格納するオブジェクトの sysindexes と syspartitions にローが作成されます。name カラム内の値は、プレフィックスが「t」のテーブル名で、indid は常に 255 です。1 つのテーブルに複数の LOB カラムがある場合は、データの格納に使用される 1 つのオブジェクトのみが存在します。デフォルトでは、このオブジェクトはテーブルと同一のセグメントに配置されます。

LOB カラムの今後の割り付けをすべて別のセグメントに移動するには、sp_placeobject を使用します。

パフォーマンスのためにテーブルを分割する

テーブルを分割すると、いくつかの処理タイプのパフォーマンスを向上できます。

- テーブルの各分割にアクセスする並列クエリ処理が実行できる。パーティションベース・スキャンでは、各ワーカー・プロセスが別々の分割を読み込む。
- バルク・コピーを使ってテーブルを並列にロードできる。

  並列 bcp の詳細については、『ASE ユーティリティ・ガイド』を参照してください。
- テーブルの I/O を複数のデータベース・デバイスに分散できる。
- セマンティック分割 (range-、hash-、list-partitioned の各テーブル) を使用すると、クエリ・プロセッサが一部の分割を排除するため、応答時間が短くなる。
- ヒープ・テーブルに複数の挿入ポイントができる。

分割するテーブルの選定と分割の種類は、発生するパフォーマンス問題と、そのテーブルに対するクエリのパフォーマンス目標によって決まります。

分割の使用と作成の詳細と例については、『Transact-SQL ユーザーズ・ガイド』の「第 10 章テーブルとインデックスの分割」を参照してください。

Adaptive Server がデバイス間で分割を分散する仕組み

Adaptive Server 15.0 の以前のバージョンでは、複数のデータベース・デバイスにマップされたセグメントに複数の分割を作成した場合、分割とデバイス間の結び付きが自動的に維持されていました。Adaptive Server 15.0 以降では、これがなくなり、すべての分割が最初のデバイスに作成されます。分割とデバイスを結び付けるには、以下の手順に従ってください。

1　特定のデバイスのセグメントを作成します。

2　そのセグメントに明示的に分割を配置します

最大 29 のユーザ・セグメントを作成できます。セグメントの作成には、Adaptive Server バージョン 15.0 以降の alter table 構文を使用する必要があります。旧バージョンの構文 (alter table t partition 20) では分割をセグメントに明示的に配置できません。

分割の数とセグメント内のデバイス数が一致すると、並列クエリの I/O パフォーマンスが最大になります。

テーブルで使用するデータ型が text、image、またはロー外にある Java の場合は、テーブルを分割できます。しかし、それらのカラム自体は分割されずに 1 つのページ・チェーンに残ります。

RAID デバイスと分割されたテーブル

ストライプ RAID (Redundant Array of Independent Disks) デバイスは複数の物理ディスクを含むことができますが、Adaptive Server はそのようなデバイスを 1 つの論理デバイスとして扱います。1 つの論理デバイスで複数の分割を使用すると、並列クエリのパフォーマンスを向上できます。

混合アプリケーションに最適な分割の数を決めるには、ストライプ・セットの各デバイスにつき 1 つの分割から始めます。オペレーティング・システムのユーティリティ (UNIX では vmstat、sar、iostat、Windows ではパフォーマンス・モニタ) を使用して、使用率と待ち時間を確認します。

デバイスの最大スループットを確認するには、index *table_name* 句を使用する select count(*) を使用し、ノンクラスタード・インデックスがある場合は、テーブルのスキャンを強制します。このコマンドが必要とする CPU 動作は最小限であり、ほかのリソースとの競合はごくわずかしか発生しません。

分割されたテーブル用の領域を計画する

分割されたテーブルを計画する場合は、以下のメンテナンス方法を考慮してください。

- パーティションベース・スキャンのパフォーマンスと I/O 並列処理に備えたディスク間の負荷分散。
- インデックスの削除と再作成や reorg rebuild の実行用に、テーブルが占有する領域の約 120% を必要とするクラスタード・インデックス。

領域計画の決定は、以下に基づきます。

- テーブルの保存に使用できるディスク・リソース容量
- アプリケーションの混合と受信データの特性 (セマンティック分割のテーブルの場合)

分割テーブルがメンテナンスを必要とする頻度を見積もります。バランスを保つためにインデックスを頻繁に再作成しなければならないアプリケーションもあれば、ほとんど管理する必要がないアプリケーションもあります。

パフォーマンス向上のために頻繁に負荷分散する必要があるアプリケーションの場合は、クラスタード・インデックスを再作成したり reorg rebuild を実行する領域を用意することで、最速かつ最も簡単に結果が得られます。しかし、クラスタード・インデックスの作成にはデータ・ページをコピーする必要があるため、セグメントで使用できる領域は、テーブルが占有する領域の約 120% の大きさがなければなりません。

「メンテナンス作業に使用可能な領域の確認」(116 ページ) を参照してください。

読み込み専用テーブル、ほとんどが読み込みであるテーブル、およびランダムなデータ修正があるテーブルについての以降の説明では、オブジェクトの配置と分割されたテーブルの管理についての一般的な例を示します。

メンテナンス中に必要な特定の作業については、『Transact-SQL ユーザーズ・ガイド』の「第 10 章テーブルとインデックスの分割」を参照してください。

読み込み専用テーブル

読み込み専用のテーブル、またはほとんど変更されないテーブルは、セグメント上の使用可能な領域を完全に満たすことができ、管理する必要がありません。テーブルにクラスタード・インデックスが必要でない場合は、並列バルク・コピー (並列 bcp) を使ってセグメントの領域を完全に満たすことができます。

クラスタード・インデックスが必要な場合、テーブルのデータ・ページは、セグメントの領域の 80% までを占有できます。クラスタード・インデックス・ツリーの場合は、テーブルが使用する領域の約 20% が必要です。

この領域要件は、キーの長さによって異なります。最初は、テーブルへのデータのロードとクラスタード・インデックスの作成に複数の手順が必要ですが、これらの手順を実行しておくと、最小限のメンテナンスで済みます。

ほとんどが読み込みであるテーブル

前述の読み込み専用テーブルのガイドラインは、ごくわずかの挿入がある、ほとんどが読み込みであるテーブルにもあてはまります。例外を次に示します。

- テーブルに挿入があり、クラスタード・インデックス・キーが各分割に新しい領域を均等に割り付けできない場合は、一部の分割が置かれているディスクが満杯になり、新しいエクステントが別の物理ディスクに割り付けられることがある。これをエクステントの横取りという。

  多数のディスクにまたがる大規模なテーブルでは、ほかのデバイスに対する割り付けのパーセンテージが小さくても問題にならない。エクステントの横取りは、sp_helpsegment を使用して、使用可能な領域がないデバイスをチェックし、sp_helpartition を使用して、ページ数のバランスがとれていない分割をチェックすると検出できる。

  分割のサイズのバランスが悪いために並列クエリ応答時間が悪化したり、並列クエリの最適化が阻まれる場合は、『Transact-SQL ユーザーズ・ガイド』の「第 10 章テーブルとインデックスの分割」で説明する方法を使用すると均等に分散できる。

- テーブルがヒープのラウンドロビン方式で分割されたテーブルの場合は、ヒープ・テーブル挿入のランダムな特性によって分割のバランスが保たれている。

大規模なバルク・コピーを実行する場合には注意する。ただし、セマンティック分割されたテーブルの場合は、alter table... partition by を使用して分割条件を変えて適切な負荷分散にすることを検討する。

各分割にまたがるデータのバランスをとるには、並列バルク・コピー (並列 bcp) を使用して、ページ数が最も少ない分割にローを送る。『ユーティリティ・ガイド』の「第 4 章 bcp を使用した Adaptive Server とのデータ転送」を参照。

ランダムなデータ修正があるテーブル

ある期間にわたって多くの挿入、更新、削除が発生する、クラスタード・インデックスを持つテーブルでは、データ・ページの約 70 ～ 75% が埋まってしまう傾向があります。このため、次のようにパフォーマンスが低下する可能性があります。

- 指定数のローにアクセスするために、読み込むページ数を増やす必要があり、I/O の回数が増し、データ・キャッシュ領域が浪費される。
- 全ページ・ロックを使用するテーブルでは、ページ・チェーンがエクステントやアロケーション・ユニットと交差するため、大規模な I/O と非同期プリフェッチのパフォーマンスが低下する。

  大容量 I/O を使用して取り込まれたバッファは、すべてのページが読み込まれる前にキャッシュからフラッシュされることがある。非同期プリフェッチの予備セットのサイズは、ページ・チェーンをたどるときにアロケーション・ユニット間をホップすることにより低減される。

  データオンリー・ロックを使用するテーブルでは、転送ページがエクステントやアロケーション・ユニットをクロスするため、大規模な I/O と非同期プリフェッチのパフォーマンスが低下する。

断片化によってアプリケーションのパフォーマンスが低下し始めると、クラスタード・インデックスの削除と再作成にはテーブルの占有領域の 120% が必要になることに留意して、メンテナンスを実行します。

それだけの領域を使用できない場合は、メンテナンスが複雑化して時間がかかります。ほとんどの場合、必要なだけディスク容量を追加してインデックス作成用の領域を用意することが、最良であり、コストが最低になる解決策です。

デバイスが満杯である場合にディスクを追加する

分割が満杯の場合、ディスクを追加してインデックスを再作成するだけでは、負荷のバランス問題は解決できません。分割のある物理デバイスが完全に満杯になる場合は、インデックスの再作成のデータ・コピー段階で、その物理デバイスにデータをコピーできません。

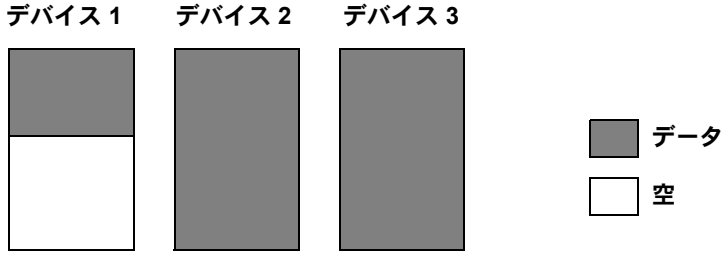
物理デバイスがほぼ完全に満杯の場合は、クラスタード・インデックスの再作成によって必ずしも良い負荷バランスが確立するとは限りません。

デバイスが満杯である場合にディスクを追加する

物理デバイスが満杯のときにクラスタード・インデックスを作成した結果、他の物理デバイスの 1 つに 2 つの分割が作成されます。

デバイス 2 と**デバイス 3** は 図 1-2 に示すように満杯です。

図 *1-2: 3* つのデバイスに *3* つの分割を持つテーブル

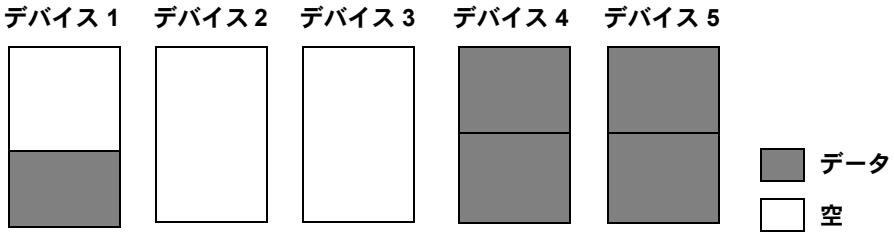


上の例で、2 つのデバイスを追加し、5 つの分割を使うようにテーブルを再分割し、クラスタード・インデックスを削除して再作成すると、次のような結果になります。

デバイス 1	1 つの分割があり、約 40% 満たされている。
デバイス 2 とデバイス 3	空。create index を開始したときには、これらのデバイスに空き領域がなかったので、インデックスのコピー用の分割をデバイスに作成できない。
デバイス 4 とデバイス 5	それぞれに 2 つの分割があり、どれも完全に満杯である。

図 1-3 は、これらの結果を示しています。

図 *1-3: create index* の実行後のデバイスおよび分割

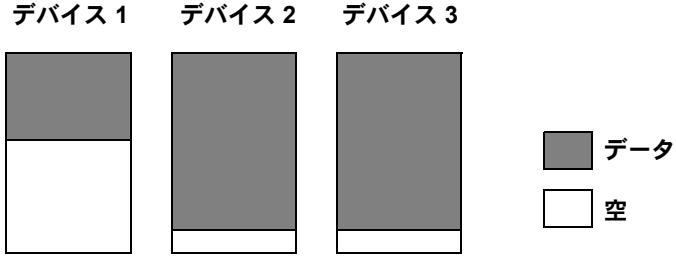


デバイスの 1 つが完全に満杯になったら、データをバルク・コピー・アウトし、テーブルをトランケートし、再びデータをコピー・インすることが唯一の解決策です。

デバイスがほぼ満杯である場合にディスクを追加する

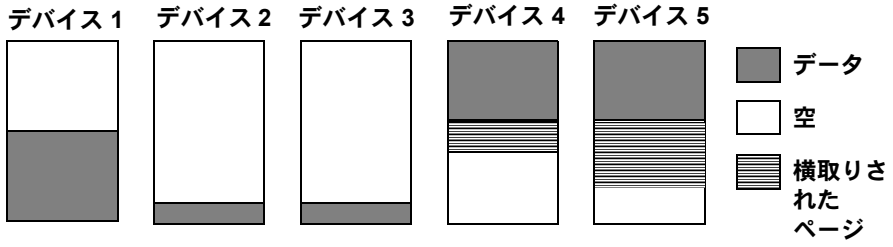
デバイスがほぼ満杯である場合は、クラスタード・インデックスを再作成してもデバイス間でデータのバランスをとることはできません。代わりに、ほぼ満杯のデバイスは分割の小さな一部分を格納し、分割に対するほかの領域の割り付けは、ほかのデバイスのエクステントを横取りします。図 1-4 は、ほぼ満杯のデータ・デバイスを持つテーブルを示しています。

図 *1-4:* ほぼ完全にデバイスを満たす分割



デバイスを追加し、クラスタード・インデックスを再作成すると、図 1-5 に示すような結果になります。

図 *1-5:* エクステントの横取りとバランスがとれていないデータの分配



デバイス 2 と**デバイス 3** の分割が、使用可能なわずかな領域を使用した後は、**デバイス 4** と**デバイス 5** からエクステントの横取りを始めます。

別の時期にインデックスを再作成する方がバランスの良い分配になる可能性があります。ただし、エクステントの横取りによってデバイスの 1 つがほぼ満杯の場合は、インデックスを再作成しても問題の解決にはなりません。

バルク・コピーを使用してデータをコピー・アウトしてから再びコピー・インするのが、このようなバランスの悪さを解消する最も有効な解決策です。

このような状況を防ぐために、デバイスの領域の使用状況をモニタし、早めに領域を追加します。

分割されたテーブルの管理について

分割されたテーブルのメンテナンス作業の要件は、テーブルに実行される更新の頻度と種類によって異なります。

管理がほとんど不要である、分割されたテーブルは、次のとおりです。

- 読み込み専用テーブルまたはごくわずかの更新が行われるテーブル。更新が少ないテーブルの場合は、バランスを定期的にチェックするだけで十分である。
- 各挿入が分割の間でバランスよく分配されているテーブル。分割されたヒープ・テーブルへのランダムな挿入や、ローをさまざまな分割に配置するクラスタード・インデックスによって均等に分配されている挿入では、ページの分散に偏りが生じることはない。

  データの修正によって領域の断片化やデータ・ページが部分的に満たされる状態が生じる場合は、クラスタード・インデックスの再作成が必要になる可能性がある。

- バルク・コピーによって挿入が実行されるヒープ・テーブル。並列バルク・コピーを使って新しいデータを特定の分割に入れ、負荷のバランスを保つ。

モニタリングと管理を頻繁に行う必要がある、分割されたテーブルには、新しいローを分割のサブセットに入れる傾向があるクラスタード・インデックスを持つテーブルが含まれます。昇順キー・インデックスには、さらに管理を頻繁に行うことが必要になります。

分割されたテーブルのための定期的な管理チェック

分割されたテーブルの定期的なモニタリングには、定期的なデータベースの一貫性検査のほかに、次の種類のチェックが含まれている必要があります。

- sp_helpartition を使って、各分割のバランスをチェックする。一部の分割のサイズが平均よりも大幅に大きいか小さい場合は、クラスタード・インデックスを再作成してデータを再分配する。
- sp_helpsegment を使用して、基盤となるディスクの領域のバランスをチェックする。
- 並列クエリ・パフォーマンスのために、クラスタード・インデックスを再作成してデータを再分配する場合は、ほぼ 50% 満たされているデバイスがあるかどうかチェックする。デバイスがあふれる前に領域を追加すると、この章で前述した複雑な手順を実行しないで済む。
- sp_helpsegment を使用して、各デバイスの空きページとして使用可能な領域をチェックする。または、sp_helpdb を使用して、空き (キロバイト) をチェックする。

以下の理由から、分割されたテーブルのクラスタード・インデックスを再作成しなければならない場合があります。

- インデックス・キーは、分割のサブセットに各挿入を割り当てる傾向がある。
- 削除アクティビティは、分割のサブセットからデータを削除する傾向がある。そのために、I/O およびパーティションベース・スキャンのバランスがくずれる。
- テーブルには、数多くの挿入、更新、削除が実行され、部分的に満たされたデータ・ページが多数発生する。この状態によってディスクとキャッシュの両方で無駄な領域が生じ、多くのクエリで読み取るページが増えるため、I/O が増える。

第 2 章

データの格納

この章では、Adaptive Server® がページにデータ・ローを格納する方法と、インデックスがない場合に select 文とデータ修正文でこれらのページを使用する方法について説明します。この章の内容は、インデックスの作成、クエリのチューニング、オブジェクトの格納での課題への対処によって Adaptive Server のパフォーマンスを高める方法を理解する上での基礎となります。

クエリの最適化

Adaptive Server オプティマイザはクエリ内の各テーブルのデータへの、最も効率的なアクセス・パスを見つけようとします。そのために、データのアクセスに必要な物理 I/O のコストと、データ・キャッシュ内に読み込む必要がある各ページの回数を見積もります。

ほとんどのデータベース・アプリケーションでは、データベースに多くのテーブルが存在し、各テーブルが 1 つまたは複数のインデックスを持ちます。インデックスを作成しているかどうかということと、どのようなインデックスを作成しているかによって、次のようなオプティマイザのアクセス・メソッドが使用されます。

- テーブル・スキャン – テーブルのすべてのデータ・ページを読み込む。場合によっては数百、数千、数万ものページを読み込む。
- インデックス・アクセス – インデックスを使って必要なデータ・ページだけを検索する。全体で 3、4 ページほどしか読み込まない場合もある。
- インデックス・カバーリング – インデックスだけを使ってデータを返す。実データ・ローを読み込まないで、そのテーブルのスキャンに必要な部分のページを読み込むだけで済む。

テーブルで適切なインデックスを使用すると、クエリのほとんどが、最小限のページ数を読み込むだけで必要なデータにアクセスできるようになります。

クエリ処理とページの読み込み

クエリ実行時間のほとんどは、ディスクからデータ・ページを読み込む作業に費やされます。そのため、パフォーマンスの向上の大部分は、クエリごとに Adaptive Server が実行する必要のあるディスクの読み込みの回数を減らすことによって達成されます。

クエリがテーブル・スキャンを実行する場合は、データを検索するのに使用できるインデックスが存在しないため、Adaptive Server はテーブル内のすべてのページを読み込みます。ディスクの読み込みに時間がかかるため、クエリの応答時間が著しく長くなる場合があります。コストのかかるテーブル・スキャンを引き起こすクエリは、サーバのほかのクエリに対するパフォーマンスも低下させます。

テーブル・スキャンによって、CPU 時間、ディスク I/O、ネットワーク容量などのシステム・リソースが使用されるため、ほかのユーザに応答が返されるまでの時間が長くなります。

テーブル・スキャンは、ある特定のクエリに対して多数のディスク読み込み (I/O) を行います。アクセス・メソッド、チューニング・ツール、テーブルのサイズと構造、アプリケーションのクエリを理解でき、使用可能なインデックスがわかれば、あるジョインまたは選択演算で実行される I/O 操作の回数を推測できるようになります。

テーブルのインデックス・カラムの状況とともに、テーブルとインデックスのサイズがわかれば、クエリをたびたび調べて、その動作を予測できます。同じテーブルに対してさまざまなクエリを実行する場合は、次のようなことを判断できるようになります。

- このクエリは、where 句の条件に一致する 1 つのローまたは少数のローを返す。

  where 句内の条件にインデックスが設定され、このインデックスに対して 2 から 4 の I/O が実行されて、正しいデータ・ページを読むためにさらに 1 つの I/O が実行される。

- このクエリの select リストと where 句内のすべてのカラムは、ノンクラスタード・インデックスに含まれる。おそらくこのクエリは、約 600 ページの、インデックスのリーフ・レベルに対するスキャンを実行する。

  インデックス未設定カラムを select リストに追加すると、クエリはテーブルをスキャンし、5,000 回のディスク読み込みが必要になる。

- このクエリに使用できるインデックスがない。クエリはテーブル・スキャンを実行するため、最低でも 5,000 回のディスク読み込みが必要。

この章では、テーブルがどのように格納されるか、およびインデックスが使用されていない場合にデータ・ローへのアクセスがどのように行われるかについて説明します。

インデックスのアクセス方法については、『パフォーマンス＆チューニング・シリーズ：ロックと同時実行制御』の「第 5 章インデックス」を参照してください。クエリに対して使用されるアクセス・メソッド、テーブルとインデックスのサイズ、1 つのクエリが実行する I/O の量の決定の仕方については、「第 3 章記憶領域管理プロパティの設定」および「第 4 章テーブルおよびインデックスのサイズ」で説明します。それらの章では、オプティマイザがクエリのデータ・アクセスのコストをどのようにモデル化するかを理解する上での基本について説明します。

Adaptive Server ページ

以下のページ形式はデータベース・オブジェクトを格納します。

- データ・ページ ― テーブルのデータ・ローを格納する。
- インデックス・ページ ― インデックスのすべてのレベルのインデックス・ローを格納する。
- ラージ・オブジェクト (LOB) ページ ― text カラム、image カラム、ロー外にある Java カラムのデータを格納する。

Adaptive Server バージョン 12.5 以降では、マスタ・デバイスの構築に buildmaster バイナリを使用しません。代わりに、dataserver バイナリに buildmaster の機能が組み込まれました。

dataserver コマンドを使用すれば、論理ページのサイズが 2K、4K、8K、16K のマスタ・デバイスと master データベースを作成できます。論理ページ・サイズを大きくすると、より大きなローを作成でき、1 ページ分の読み込みによってアクセスできるデータの量が増えるので、パフォーマンスが向上します。たとえば、論理ページ・サイズが 16K ならば 2K ページの 8 倍のデータを保持でき、8K ページの場合は 2K ページの 4 倍のデータを保持できます。

論理ページ・サイズはサーバ全体に適用される設定です。そのため、同じサーバ内で論理ページ・サイズの異なるデータベースを設定することはできません。すべてのテーブルのサイズは、ロー・サイズがサーバの現在のページ・サイズより大きくならない範囲で自動的に設定されます。つまり、ローは複数のページにまたがることはできません。

dataserver コマンドを使ってマスタ・デバイスを構築する方法については、『ユーティリティ・ガイド』を参照してください。

Adaptive Server では、単一のクエリ、DML オペレーション、またはコマンドで大量のデータを扱わなければならない場合があります。たとえば、char(2000) カラムを持つデータオンリーロック (DOL) テーブルを使用する場合は、テーブルのスキャン中にカラムのコピーを実行できるように Adaptive Server でメモリを割り付ける必要があります。クエリやコマンドの実行中にメモリ要求が増えると、スループットが低下する可能性があります。

Adaptive Server の論理ページのサイズによって、サーバの領域の割り付けが決まります。各アロケーション・ページ、オブジェクト・アロケーション・マップ (OAM) ページ、データ・ページ、インデックス・ページ、テキスト・ページなどが論理ページ上で構築されます。たとえば、Adaptive Server の論理ページ・サイズが 8K の場合は、これらのページのサイズは 8K となります。論理ページ・サイズで指定されたサイズ全体が、これらのすべてのページによって使用されます。OAM ページは、論理ページ・サイズが大きいほど (たとえば、8K) そのエントリの数が 2K のページ・サイズの場合よりも多くなります。

ページ・ヘッダとページ・サイズ

すべてのページは、ページが属するパーティション ID などの情報とページ上の領域を管理するのに使用されるその他の情報を格納するヘッダを持ちます。表 2-1 は、2K ページに設定されているサーバのデータ・ページとインデックス・ページのオーバヘッドと使用できる領域のバイト数を示します。

表 2-1: データ・ページとインデックス・ページのオーバヘッドとユーザ・データ領域

ロック・スキーム	オーバヘッド	ユーザ・データのバイト数
全ページ	32	2016
データオンリー	46	2002

ページの残りの領域はデータ・ローとインデックス・ローの格納に使用されます。

text、image、Java カラムの格納方法については、「ラージ・オブジェクト (LOB) ページ」(23 ページ) を参照してください。

データ・ページとインデックス・ページ

データオンリーロック・テーブルのデータ・ページとインデックス・ページは、ページ上の各ローの先頭バイトを示すポインタを格納するロー・オフセット・テーブルを持ちます。各ポインタは 2 バイトを使用します。

データ・ローとインデックス・ローは、ページのページ・ヘッダのすぐ後に挿入され、隣接して埋められます。データオンリーロック・テーブル上のすべてのテーブルとインデックスでは、ロー・オフセット・テーブルはページの最終バイトから始まり、上の方へと増えていきます。

各ローは、実カラム・データと、ロー番号、ロー内の可変長カラムと null カラムの数などの情報を格納します。全ページロック・テーブルのインデックス・ページはロー・オフセット・テーブルを持ちません。

ローは、text カラム、image カラム、ロー外にある Java カラムを除いて、ページ境界にまたがることはありません。各データ・ローには少なくとも 4 バイトのオーバヘッドがあります。可変長データを持つローにはさらに多くのオーバヘッドがあります。

データ・サイズとインデックス・ロー・サイズ、オーバヘッドの詳細については、「第 4 章 テーブルおよびインデックスのサイズ」を参照してください。

ラージ・オブジェクト (LOB) ページ

テーブルの text カラム、image カラム、ロー外にある Java カラムは、一連のページから構成される、それぞれ独立したデータ構造として格納されます。このようなカラムはラージ・オブジェクト (LOB) カラムと呼ばれます。text カラムまたは image カラムのあるテーブルには、これらのデータ構造の 1 つが存在します。テーブル内に複数の LOB カラムがある場合でも、テーブルにはこれらの独立したデータ構造が 1 つだけ存在します。

テーブルには、そのローの値の最初のページを示す 16 バイトのポインタが格納されます。値の 2 ページめ以降は、次ページへのポインタ、前ページへのポインタによってリンクされます。各値は、それぞれの独立したページ・チェーンに格納されます。最初のページには、text 値のバイト数が格納されます。ある値のチェーンの最終ページには、null の次ページへのポインタがあり、最終ページであることを示します。

LOB 値の読み込みまたは書き込みには、少なくとも 2 つのページ読み込みかページ書き込みが必要です。この内訳は次のとおりです。

- ポインタに対する 1 ページの読み込みまたは書き込み。
- テキスト・オブジェクト内のテキストの実際の格納位置に対する 1 ページの読み込みまたは書き込み。

各 LOB ページは最大 1,800 バイトまで格納できます。どの非 null 値も少なくとも 1 ページ全体を使用します。

LOB 構造は sysindexes テーブルに独立して格納されています。LOB 構造の ID はテーブルの ID と同じです。インデックス ID カラムである indid は常に 255 であり、name はテーブル名を表し、テーブルの名の先頭には "t" がつきます。

エクステント

Adaptive Server のページは、常にテーブル、インデックス、または LOB 構造に割り付けられます。この 8 ページ構成のブロックを「エクステント」といいます。エクステントのサイズは、サーバが使用するページ・サイズによって異なります。2K サーバ上のエクステント・サイズは 16K であり、8K サーバの場合は 64K となります。テーブルまたはインデックスが占有できる最小領域は 1 エクステントまたは 8 ページです。エクステントの割り付けが解除されるのは、エクステント内のすべてのページが空の場合だけです。

Adaptive Server 内のエクステントの使用は、領域の使用状況についてレポートを調べる場合を除いて、ユーザには見えません。たとえば、sp_spaceused ストアド・プロシージャによるレポートは、割り付けられた領域 (reserved カラム) およびデータとインデックスが使用している領域を表示します。unused カラムは、オブジェクトに割り付けられているが、まだデータを格納していないエクステント内の領域の量を表示します。

```
sp_spaceused titles
name   rowtotal reserved data    index_size unused
------ -------- -------- ------- ---------- ------
titles 5000     1392 KB 1250 KB 94 KB      48 KB
```

このレポートでは、titles テーブルとそのインデックスは複数のエクステントに 1,392KB の領域を予約しており、そのうちの 48KB (24 データ・ページ) にはデータが割り付けられていません。

注意 Adaptive Server は無駄な領域が生じないよう、ターゲットのアロケーション・ページ内の現在割り付けられているエクステントを、それらのエクステントが他のパーティションに割り当てられていても、満杯まで使おうとします。その結果、ターゲットのアロケーション・ページに空きエクステントがなくなったときにだけ、エクステントが使われるようになります。

領域の割り付けを管理するページ

データの格納に使用されるデータ、インデックス、LOB ページに加えて、Adaptive Server はそのほかのページ形式を使用して、格納の管理、領域の割り付けの記録、データベース・オブジェクトの格納位置の特定を行います。sysindexes テーブルは、データ・アクセス中に使用されるポインタも格納します。

領域の割り付けを管理するページと sysindexes のポインタは、次の目的で使用されます。

- データベース内のオブジェクトの検索プロセスを高速化する。
- オブジェクトへの領域の割り付けと割り付け解除プロセスを高速化する。

- オブジェクトがすでに使用している領域の近くに、Adaptive Server がオブジェクト用の領域を追加して割り付けできるようにする。ディスクヘッドの移動距離が減るため、パフォーマンスが向上する。

次のページは、データベース・オブジェクトによるディスク領域使用状況を記録します。

- GAM (グローバル・アロケーション・マップ) ページ。あるデータベース全体の割り付けビットマップが入っている。
- アロケーション・ページ。256 ページ、つまり 0.5MB から成るグループ内の領域使用状況とオブジェクトを記録する。
- OAM (オブジェクト・アロケーション・マップ) ページ。1 つのオブジェクト用に使用されているエクステントについての情報が入っている。テーブルとインデックスの各パーティションには少なくとも 1 つの OAM ページがあって、オブジェクト用のページがデータベース内のどこに格納されているかを記録する。
- OAM ページ。分割されたテーブルに対する領域の割り付けを管理する。

グローバル・アロケーション・マップ・ページ

各データベースには GAM ページがあります。GAM ページは、データベースのすべてのアロケーション・ユニットのビットマップを、アロケーション・ユニットごとに 1 ビットずつ格納します。アロケーション・ユニット内にオブジェクトの格納に使用できる空きエクステントがない場合は、GAM 内のこのアロケーション・ユニットに対応するビットが 1 に設定されます。

このメカニズムによって、オブジェクトへの新しい領域の割り付けが単純化されます。GAM ページはユーザからは見えませんが、システム・カタログ内では sysgams テーブルとして表示されます。

アロケーション・ページ

データベース作成時またはデータベースへの領域追加時に、256 データ・ページから成るアロケーション・ユニットに領域が分割されます。各「アロケーション・ユニット」の最初のページがアロケーション・ページです。ページ 0 とページ番号が 256 の倍数であるすべてのページは、アロケーション・ページです。

アロケーション・ページは、エクステント上に格納されているオブジェクトのパーティション ID、オブジェクト ID、およびインデックス ID、使用中のページ数と使用されていないページ数を記録することによって、アロケーション・ユニット内の各エクステントの領域を記録します。アロケーション・ページには、OAM ページに対応するテーブルまたはインデックスのページ ID も格納されます。

オブジェクト・アロケーション・マップ・ページ

テーブル、インデックス、テキストのチェーンのパーティションごとに 1 つまたは複数の OAM (オブジェクト・アロケーション・マップ) ページが、テーブルまたはインデックスに割り付けられているページ上に格納されます。1 つのテーブル内に複数の OAM ページがある場合は、これらのページはチェーン内にリンクされます。OAM ページは、オブジェクト用のページが入っているアロケーション・ユニットを示すポインタを格納します。

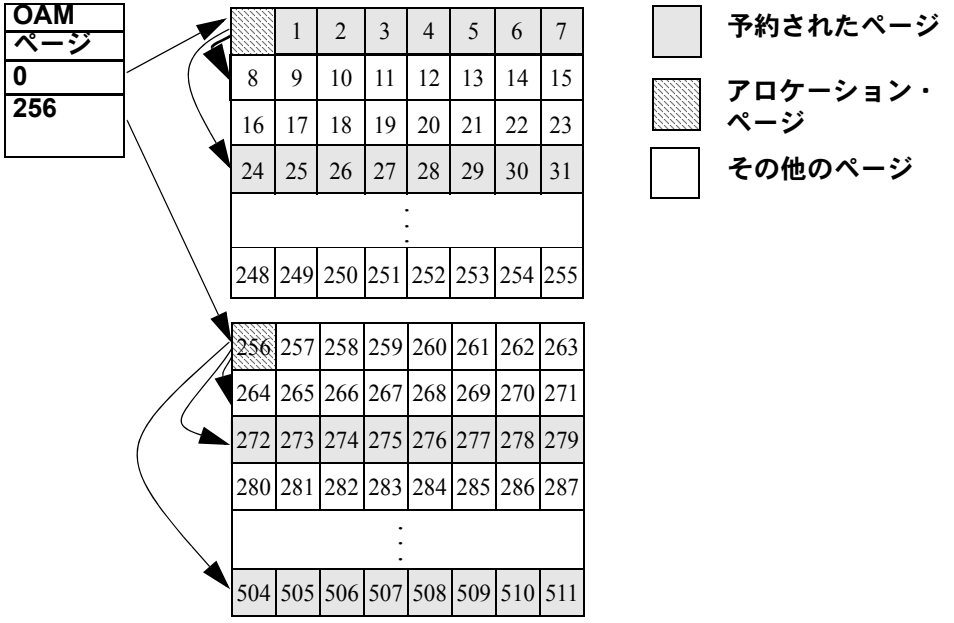
チェーンの最初のページには、割り付けのヒント (アロケーション・ヒント) が入り、チェーン内のどの OAM ページに空き領域のあるアロケーション・ユニットについての情報が格納されているかを示します。これによりオブジェクトに追加領域が高速に割り付けられ、しかもオブジェクトがすでに使用しているページの近くに新しい領域が割り付けられます。

OAM ページとアロケーション・ページによるオブジェクトの格納の管理方法

図 2-1 は、アロケーション・ユニット、エクステント、オブジェクトが OAM ページとアロケーション・ページによって管理される様子を示します。

- 2 つのアロケーション・ユニットを示す。1 つはページ 0 から始まり、もう 1 つはページ 256 から始まる。各ユニットの最初のページはアロケーション・ページ。
- あるテーブルは 4 エクステントを使用して格納され、各エクステントは最初のアロケーション・ユニットではページ 1 と 24 から、2 番めのアロケーション・ユニットではページ 272 と 504 から始まる。
- テーブルの最初のページは、そのテーブルの OAM ページ。OAM ページは、そのオブジェクトによってページの領域が使用されている各アロケーション・ユニットのアロケーション・ページを指すため、この例ではページ 0 と 256 を示す。
- アロケーション・ページ 0 と 256 は、エクステントが属するテーブルのオブジェクト ID、インデックス ID、およびパーティション ID を格納する。アロケーション・ページ 0 はそのテーブルに対してページ 1 と 24 を、アロケーション・ページ 256 はページ 272 と 504 を示す。

図 *2-1: OAM* ページとアロケーション・ページのポインタ



オブジェクトのページをまとめるページの割り付け

Adaptive Server は、各オブジェクト (テーブル・パーティション、テーブル・インデックス、テーブルのテキスト・チェーンやイメージ・チェーンなど) のページの割り付けがなるべく近くにまとまるようにします。

通常、Adaptive Server によって新しいページが要求されると、次のような仕組みで割り付けが行われます。

- オブジェクトの現在のエクステント内に空きページがある場合は、Adaptive Server はこのページを使用する。
- 現在のエクステントに空きページはないが、同一のアロケーション・ユニット上のオブジェクトに割り当てられている別のエクステントに空きページがある場合は、Adaptive Server はこのページを使用する。
- 現在のアロケーション・ユニットに、空きページがオブジェクトに割り当てられているエクステントはないが、空きエクステントがある場合は、Adaptive Sever は、空きエクステントで最初に使用可能なページを使用する。

- 現在のアロケーション・ユニットが満杯の場合は、Adaptive Server は、エクステントに空きページがある別のアロケーション・ユニットについてオブジェクト OAM ページをスキャンし、最初に使用可能なページを使用する。
- OAM エントリによって使用可能な空きページが示されない場合は、Adaptive Server は OAM エントリとグローバル・アロケーション・マップのページを比較して、アロケーション・ユニットに空きエクステントが存在するかどうかを確認する。Adaptive Server は、最初の空きエクステントの最初に使用可能なページを割り付ける。
- すべての OAM エントリでアロケーション・ユニットが満杯であると示された場合は、Adaptive Server は、空きエクステントを 1 つ以上持つアロケーション・ユニットについてグローバル・アロケーション・マップを検索する。Adaptive Server は、そのアロケーション・ユニットについて新しい OAM エントリを追加し、空きエクステントを割り付け、そのエクステント上で最初の空きページを使用する。

注意 大規模な割り付けを使用する bcp や reorg rebuild などのオペレーションでは、割り付け済みのエクステント上で空きページを検索せず、完全な空きエクステントを割り付けます。通常、大規模な割り付けでは、各アロケーション・ユニットの最初のエクステントは使用されません。最初のエクステントは、その最初のページにアロケーション・ページの構造が含まれているため、使用可能なページは 7 ページのみです。

---

sysindexes および *syspartitions* を使用したデータ・アクセス

sysindexes テーブルには、インデックス・テーブルとインデックスなしのテーブルについての情報が格納されます。sysindexes には、次のようなローが 1 つあります。

- 全ページロック・テーブルで、テーブルにクラスタード・インデックスがない場合は、indid カラムは 0。また、クラスタード・インデックスがある場合は、1。
- データオンリーロック・テーブルで、indid は常に 0。
- ノンクラスタード・インデックスおよびデータオンリーロック・テーブルの各クラスタード・インデックス。
- 1 つ以上の LOB カラムを含むテーブルで、LOB 構造に対するインデックス ID は常に 255。

syspartitions は、各テーブルとインデックス・パーティションに関する情報を格納し、パーティションごとに 1 つのローを含めます。

Adaptive Server バージョン 15.0 以降では、syspartitions の各ローは、テーブルまたはインデックスを示すポインタを格納して、オブジェクトへのアクセスを高速化します。表2-2 は、データ・アクセス中のこれらのポインタの使用法を示します。

表 2-2: データ・アクセス時の *syspartitions* ポインタの使用

カラム	テーブル・アクセスでの使用	インデックス・アクセスでの使用
root	indid が 0 で、テーブルが、分割された全ページロック・テーブルの場合、root はヒープの最終ページを指す。	インデックス・ツリーのルート・ページを検索するのに使用される。
first	全ページロック・テーブルのページ・チェーン内の最初のデータ・ページを指す。	データオンリーロック・テーブル上のノンクラスタード・インデックスまたはクラスタード・インデックスの最初のリーフ・レベルのページを指す。
doampg	テーブルの最初の OAM ページを指す。	
ioampg		インデックスの最初の OAM ページを指す。

領域のオーバヘッド

設定されている論理ページ・サイズに関係なく、オブジェクト (テーブル、インデックス、テキスト・ページ・チェーン) の領域はエクステント単位で割り付けられます。1 エクステントは 8 論理ページです。つまり、サーバが 2K の論理ページを使用するように設定されている場合は、これらのオブジェクトに 1 エクステントとして 16K が割り付けられます。16K の論理ページを使用するように設定されている場合は、各オブジェクトに 1 エクステントとして 128K が割り付けられます。

システム・テーブルに対しても、同様の割り付けが行われます。小さいテーブルが多数あるサーバの場合は、使用する論理ページ・サイズが大きいと、領域の消費量が膨大になることがあります。

たとえば、2KB 論理ページ用に設定されているサーバでは、約 31 の短いロー、クラスタード・インデックス、ノンクラスタード・インデックスから成る systypes 用として 3 エクステント、つまり 48KB のメモリが予約されます。サーバをマイグレートして 8KB ページを使用する場合、systypes 用に予約される領域は 3 エクステントのままですが、メモリは 192KB になります。

16KB に設定されているサーバの場合、systypes は 384KB のディスク領域を必要とします。サイズが小さいテーブルの最後のエクステントで未使用となっている領域が、大きい論理ページ・サイズを使用するサーバでは重要になることがあります。

ページ・サイズを大きくすると、データベースもその影響を受けます。各データベースには、システム・カタログとそのインデックスがあります。小さい論理ページ・サイズから大きい論理ページ・サイズにマイグレートする場合は、各データベースが必要とするディスク領域の量を考慮する必要があります。

カラムの数とサイズ

テーブルに作成可能な最大カラム数は次のとおりです。

- 全ページロック (APL) テーブルとデータオンリーロック (DOL) テーブルの固定長カラムは 1024 まで
- APL テーブルの可変長カラムは 254 まで
- DOL テーブルの可変長カラムは 1024 まで

カラムの最大サイズは次の条件によって決定されます。

- テーブルに可変長カラムまたは固定長カラムが含まれるかどうか。
- データベースの論理ページ・サイズ。たとえば、2K の論理ページを持つデータベースでは、APL テーブルのカラムの最大サイズを、単一のローと同じ約 1962 バイトからロー・フォーマット・オーバヘッドを引いた値に設定できる。同じように、4K の論理ページを持つデータベースでは、APL テーブルのカラムの最大サイズを、約 4010 バイトからロー・フォーマット・オーバヘッドを引いた値に設定できる。詳細については、表2-3を参照。
- 最大論理ページ・サイズより大きい固定長カラムを持つテーブルを作成しようとすると、create table の実行時にエラー・メッセージが発行される。

表 2-3: ローとカラムの最大長

ロック・スキーム	ページ・サイズ	ローの最大長	カラムの最大長
	2K (2048 バイト)	1962	1960 バイト
	4K (4096 バイト)	4010	4008 バイト
APL テーブル	8K (8192 バイト)	8106	8104 バイト
	16K (16384 バイト)	16298	16296 バイト
	2K (2048 バイト)	1964	1958 バイト
	4K (4096 バイト)	4012	4006 バイト
DOL テーブル	8K (8192 バイト)	8108	8102 バイト
	16K (16384 バイト)	16300	16294 バイト テーブルに可変長カラムがない場合
	16K (16384 バイト)	16300 (可変長カラムの最大開始オフセット=8191 に従う)	8191-6-2 = 8183 バイト − 1 つ以上の可変長カラムがテーブルにある場合

16K 論理ページ・サイズを持つ DOL テーブルの固定長カラムの最大サイズは、テーブルに可変長カラムがあるかどうかによって異なります。可変長カラムの最大開始オフセット値は 8191 です。テーブルに可変長カラムがある場合は、ローの固定長部分とオーバヘッドの合計は 8191 バイト以下でなければなりません。また、テーブルに可変長カラムが含まれる場合に、すべての固定長カラムの最大サイズは、8183 バイトに制限されます。

表 2-3 では、最大カラム長は、ロー・オーバーヘッドの 6 バイトとローの長さのフィールドの 2 バイトを引いて決定しています。

APL テーブルの可変長カラム

可変長カラム (varchar、varbinary など) が 1 つ含まれている APL テーブルは、ローごとに以下の最小オーバヘッドを伴います。

- 初期ロー・オーバヘッドでは 2 バイト。
- ローの長さでは 2 バイト。
- カラム・オフセット・テーブルのローの終りでは 2 バイト。上の値は、常に *n+1* バイトとして表される。*n* は、ロー内の可変長カラムの数。

単一カラム・テーブルには、少なくとも 6 バイトのオーバヘッドと追加オーバヘッドがあります。オーバヘッドを考慮した後の最大カラム・サイズは、カラムの長さに追加オーバーヘッドと 6 バイトのオーバヘッドを加えた値以下でなければなりません。

表 *2-4: APL* テーブルの可変長カラムの最大サイズ

ページ・サイズ	ローの最大長	カラムの最大長
2K (2048 バイト)	1962	1948
4K (4096 バイト)	4010	3988
8K (8192 バイト)	8096	8068
16K (16384 バイト)	16298	16228

論理ページ・サイズを超えている可変長カラム

テーブルに 2K 論理ページを使用する場合は、そのローの長さの合計が 2K ページ・サイズでのローの最大長を超えている可変長カラムを作成することができます。したがって、一部の可変長カラムに最大サイズが適用されるテーブルを作成できます。ただし、create table を発行すると、ローのサイズが最大ロー・サイズを超えるため、その後に insert または update を実行すると失敗する可能性があることを示す警告メッセージが表示されます。

たとえば、2K ページ・サイズを使用するテーブルを作成して、1975 バイトの長さの可変長カラムを設定すると、Adaptive Server によってテーブルは作成されますが、警告メッセージが表示されます。ローの最大長 (1962 バイト) を超えるデータは挿入できません。

DOL テーブルの可変長カラム

DOL テーブルの単一可変長カラムでの各ローの最小オーバヘッドは次のとおりです。

- 初期ロー・オーバヘッドでは 6 バイト。
- ローの長さでは 2 バイト。
- カラム・オフセット・テーブルのローの終りでは 2 バイト。各カラムのオフセット・エントリは 2 バイトである。そのようなエントリは *n* 個あり、この n はローに含まれる可変長カラムの数を表す。

総オーバヘッドは 10 バイトです。DOL ローの場合は、調整テーブルはありません。実際の可変長カラムのサイズは次のようになります。

```
column length + 10 bytes overhead
```

表 *2-5: DOL* テーブルの可変長カラムの最大サイズ

ページ・サイズ	ローの最大長	カラムの最大長
2K (2048 バイト)	1964	1954
4K (4096 バイト)	4012	4002
8K (8192 バイト)	8108	8098
16K (16384 バイト)	16300	16290

可変長カラムを持つ DOL テーブルであらゆる挿入を実行できるようにするには、オフセットを 8191 バイトより小さくする必要があります。たとえば、以下の挿入は、カラム c2、c3、および c4 のオフセットが 9010 で、最大値 8191 バイトを超えているため、失敗します。

```
create table t1(
    c1 int not null,
    c2 varchar(5000) not null
    c3 varchar(4000) not null
    c4 varchar(10) not null
    ... more fixed length columns)
cvarlen varchar(nnn)) lock datarows
```

ワイドな可変長ロー

16K の論理ページ・サイズに設定してある Adaptive Server では、データオンリーロック (DOL) カラムで、長い可変長 DOL ローに最大 32767 バイトのロー・オフセットを使用できます。

以下のオプションを使用して、長い可変長 DOL ローをデータベースごとに有効にします。

```
sp_dboption database_name, 'allow wide dol rows', true
```

注意 allow wide dol rows は、テンポラリ・データベースでは、デフォルトでオンです。マスタ・データベースに allow wide dol rows は設定できません。

sp_dboption 'allow wide dol rows' は、サイズが 16K より小さい論理ページ・サイズのユーザ・データベースには影響しません。Adaptive Server は、サイズが 16384 バイトより小さいページ上には長い可変長 DOL ローを作成できません。

この例では、ページ・サイズを 16K に指定してサーバ上の pubs2 データベースで長い可変長 DOL ローを使用するように設定します。

1　以下のコマンドを使用して、pubs2 データベースで長い可変長ローを有効にします。

```
sp_dboption pubs2, 'allow wide dol rows', true
```

2　book_desc テーブルを作成します。このテーブルには、8192 より後のロー・オフセットで開始する長い可変長 DOL カラムが含まれます。

```
create table book_desc
(title varchar(80) not null,
title_id varchar(6) not null,
author_desc char(8192) not null,
book_desc varchar(5000) not null)
lock datarows
```

ワイド・データのバルク・コピー

長い可変長 DOL ローを含んでいるデータのバルク・コピー・インを行うには、Adaptive Server バージョン 15.7 以降に付属している bcp のバージョンを使用する必要があります。データを受信するデータベースで長い可変長 DOL ローを受け入れるように設定する必要があります (つまり、bcp は、allow wide dol rows が有効でないデータベースに長いローをコピーしません)。

ダウングレードのチェック

長い可変長ローを使用する Adaptive Server のダウングレードの詳細については、使用しているプラットフォーム用の『インストール・ガイド』を参照してください。

長い可変長 DOL カラムのダンプとロード

データベースとトランザクション・ログのダンプには allow wide dol rows の設定が保持されています。この設定は、ダンプをロードするデータベースにインポートされます (このデータベースでこのオプションが設定されていない場合)。

たとえば、my_table_log.dmp という名前のトランザクション・ログのダンプ (allow wide dol rows が true に設定されている) をデータベース big_database (allow wide dol rows が未設定) にロードした場合、big_database へのロードが完了した後も、my_table.log の allow wide dol rows の true の設定が保持されます。

ただし、データベースまたはトランザクション・ログのダンプで allow wide dol rows が設定されず、ダンプをロードするデータベースでは設定されている場合、その allow wide dol rows の設定が維持されます。

15.7 以前のバージョンの Adaptive Server に対して allow wide dol rows が有効にされているデータベースのダンプは、ロードできません。

長い可変長 DOL ローがあるプロキシ・テーブルの使用

長い可変長 DOL ローがあるプロキシ・テーブルを使用できます。

プロキシ・テーブルを作成する場合 (ローの長さは問わない)、制御している Adaptive Server では create table コマンドまたは create proxy table コマンドを実行できます。Adaptive Server は、接続中のサーバに対してこれらのコマンドを実行します。ただし、Adaptive Server は、データが格納されているサーバに対してデータ操作文 (insert、delete) を実行します。ユーザのローカル・サーバは、要求をフォーマットして送信するだけで、何も制御しません。

Adaptive Server は、データがリモート・サーバ上にある場合でも、ローカル・サーバ上で作成されたかのようにプロキシ・テーブルを作成します。長い可変長ローを含んでいる DOL テーブルを作成するには、create proxy table コマンドを実行します。ただし、このコマンドは、プロキシ・テーブルを作成するデータベースで allow wide dol rows が true に設定されている場合にのみ、正常に動作します。

注意 Adaptive Server は、ローカル・サーバ lock scheme の設定を使用してプロキシ・テーブルを作成し、lock scheme が datarows または datapages に設定されている場合は、DOL ローがあるプロキシ・テーブルを作成します。

プロキシ・テーブルでデータを挿入したり、データを更新したりした場合、Adaptive Server は allow wide dol rows のローカル・データベース設定を無視します。insert または update が正常に動作するかどうかは、データがあるサーバによって決まります。

ロック・スキーム変換または select into 使用時の制限

alter table を使ってロック・スキームを変更する場合でも、select into を使ってデータを新しいテーブルにコピーする場合でも以下の制限が適用されます。

16K ページ以外のページ・サイズを使用するサーバでは、APL テーブルの可変長カラムの最大長は、DOL テーブルの場合より小さいので、最大サイズの可変長カラムを持つ APL テーブルのロック・スキームを DOL に変換できます。ただし、少なくとも 1 つの最大サイズ可変長カラムを持つ DOL テーブルを APL テーブルに変換することはできません。

16K ページを使用するサーバでは、APL テーブルに DOL テーブルに格納する場合よりも大きいサイズの可変長カラムを格納できます。テーブルは DOL から APL に変換できますが、APL から DOL に変換することはできません。

ロック・スキーム変換に関するこれらの制限が適用されるのは、送信元テーブルのデータがターゲット・テーブルの制限を超えている場合だけです。制限に違反していると、ロー・フォーマットを 1 つのロック・スキームから別のスキームに変換するときにエラー・メッセージが表示されます。テーブルが空の場合は、そのようなデータ変換が要求されないので、ロック変更オペレーションに成功します。ただし、その後にテーブルに対して insert または update を実行しようとすると、変更したテーブルのターゲット・スキームに関連するカラムまたはロー・サイズの制限によってエラーが発生する可能性があります。

可変長カラムのサイズに基づいた DOL テーブルのカラムの編成

可変長カラムを使用する DOL テーブルでは、テーブル定義の終端寄りに最長カラムが配置されるようにカラムを調整します。このように配置すると、大きいカラムをテーブル定義の先頭に配置する場合よりも大きいローを持つテーブルを作成することができます。たとえば、16K ページ・サーバでは、以下のテーブル定義が有効です。

```
create table t1 (
    c1 int not null,
    c2 varchar(1000) null,
    c3 varchar(4000) null,
    c4 varchar(9000) null) lock datarows
```

しかし、一般に以下のようなテーブル定義は、その後の挿入を想定した場合には不適切です。カラム c2 の開始オフセットは、その前に 9000 バイトの c4 カラムがあるため、8192 バイトの制限を超える可能性があります。

```
create table t2 (
    c1 int not null,
    c4 varchar(9000) null,
    c3 varchar(4000) null,
    c2 varchar(1000) null) lock datarows
```

テーブルは作成されますが、その後の挿入には失敗する可能性があります。

データ・ページあたりのロー数

DOL データ・ページに使用できるロー数は、以下の条件によって決定されます。

- ページ・サイズ
- ロー転送アドレスを指定するロー ID の 10 バイト・オーバヘッド

表2-6 は、DOL データ・ページに挿入できるローの最大数を示しています。

表 *2-6: DOL* データ・ページのデータ・ローの最大数

ページ・サイズ	ローの最大数
2K	166
4K	337
8K	678
16K	1361

APL データ・ページには、最大で 256 までのローを使用できます。各ページには、1 バイトのロー番号指定子が必要なので、短いローを持つ大きなページでは、未使用領域が生じます。たとえば、Adaptive Server が 8K 論理ページと、25 バイトのロー用に設定されている場合は、ロー・オフセット・テーブルとページ・ヘッダを計算に入れた後で 1275 バイトの未使用領域が生じます。

オブジェクトとサイズの追加制限数

ストアド・プロシージャの引数の最大数は 2048 です。『Transact - SQL ユーザーズ・ガイド』の「第 16 章ストアド・プロシージャの使用」を参照してください。

バージョン 12.5 以降の Adaptive Server では、バージョン 12.5 より前の ASE で格納されたデータとは制限の異なるデータを格納できます。クライアントは、新しいデータ数の制限を保存および処理できる必要があります。Open Client と Open Server の古いバージョンを使用している場合は、次の処理を行ったときにデータを処理できません。

- Adaptive Server バージョン 12.5 以降へのアップグレード
- ワイド・カラムを持つテーブルの削除と再作成
- ワイド・データの挿入

『Open Client 設定ガイド』の「第 2 章 Open Client に必要な基本設定」を参照してください。

クラスタード・インデックスのないテーブル

Adaptive Server 上でテーブルを作成する場合に、クラスタード・インデックスを作成しないと、そのテーブルは「ヒープ」として格納されます。データ・ローは特定の順序がつけられずに格納されます。この項では、データ検索に有効な「利用できる」インデックスがない場合に、ヒープ・テーブルに対して select、insert、delete、および update オペレーションがどのように動作するかについて説明します。

ヒープ・テーブルを使った方がよいケースはわずかしかありません。ほとんどのアプリケーションでは、テーブルにクラスタード・インデックスを設定した方がパフォーマンスが向上します。ただし、サイズが小さく、少数のページしか使用しないテーブルと変更があまり発生しないテーブルにはヒープ・テーブルは効果があります。

ヒープ・テーブルは、次のようなテーブルに使うと効果的です。

- 任意の 1 つのローに直接アクセスする必要がないテーブル。
- 結果セットを順序付ける必要がないテーブル

ヒープ・テーブルは、テーブル・ローのサブセットを返す必要のある多くの大きなテーブルに対するクエリには向きません。

クラスタード・インデックスの削除と作成のオーバヘッドが許されないアプリケーションで、大容量のバッチ挿入が頻繁に発生する場合には、分割したヒープ・テーブルを使用すると効果的です。

シーケンシャルなディスク・アクセスは、大容量 I/O と非同期プリフェッチを使用した場合に特に効率的です。しかし、値を見つけるためには常にテーブル全体をスキャンしなければならないので、データ・キャッシュと他のクエリに大きな影響を与える可能性があります。

バッチによる挿入で、効率的にシーケンシャル I/O が実行できますが、複数の処理がデータを同時に挿入しようとすると、最終ページにボトルネックが発生する可能性があります。

where 句内に指定されたカラム上にインデックスが存在していても、オプティマイザが、テーブル・スキャンを実行するよりもこのインデックスを使う方がコストがかかると判断する場合もあります。

テーブル内のすべてのローを選択すると、テーブル・スキャンが常に使用されます。クエリが、ノンクラスタード・インデックス内のキーであるカラムだけを指定する場合にかぎって、テーブル・スキャンは使用されません。

詳細については、『パフォーマンス&チューニング・シリーズ：ロックと同時実行制御』の「第 5 章インデックス」を参照してください。

ロック・スキーム

APLテーブル内のデータ・ページは、各ページ上のポインタによって、ページのリストにリンクされます。データオンリーロック・テーブル内のページはページ・チェーンにリンクされません。

全ページロック・テーブル内の各ページには、チェーン上の次のページを示すポインタと、チェーン上の前のページを示すポインタが格納されます。新しいページを挿入すると、となりあう 2 つのページのポインタが更新されて、新しいページを指します。Adaptive Server は全ページロック・テーブルをスキャンするときに、これらのページ・ポインタに従ってページを順番に読み込みます。

ページは、全ページロック・テーブルの各インデックス・レベルとデータオンリーロック・テーブル上のインデックスのリーフ・レベルにおいても双方向にリンクされています。全ページロック・テーブルが分割されている場合は、パーティションごとにページ・チェーンが 1 つあります。

全ページロック・テーブルと異なり、通常、データオンリーロック・テーブルでは、クラスタード・インデックスを作成した直後を除いて、ページ・チェーンは維持されません。しかし、このページ・チェーンは、テーブルに対して初めてコマンドを発行したときに切れます。

Adaptive Server はデータオンリーロック・テーブルをスキャンするときに、OAM ページに格納されている情報を使用します。「オブジェクト・アロケーション・マップ・ページ」(26 ページ) を参照してください。

全ページロック・テーブルとデータオンリーロック・テーブルのもう 1 つの違いは、データオンリーロック・テーブルが固定ロー ID を使用していたことです。つまり、ロー ID (ページ番号とページ上のロー番号の組み合わせ) は、通常のクエリ処理中にデータオンリーロック・テーブルでは変化しません。

ロー ID は、データ・ローをコピーする必要のある処理の 1 つを実行する場合 (たとえば、reorg rebuild 中またはクラスタード・インデックスの作成中) にだけ変化します。

固定ロー ID がヒープ・オペレーションにどのように影響するかについては、「データオンリーロック・ヒープ・テーブルからの削除」(41 ページ) と 「データオンリーロック・ヒープ・テーブル」(42 ページ) を参照してください。

ヒープ・テーブルに対する選択オペレーション

ヒープを対象に select クエリが発行され、利用できるインデックスがない場合には、Adaptive Server はテーブル内のすべてのデータ・ページをスキャンして、クエリの条件を満たすすべてのローを検出しなければなりません。条件に一致するローは 1 つしかないこともあれば、多数あることもあります。一致するローが存在しないこともあります。

全ページロック・ヒープ・テーブル

全ページロック・テーブルの場合、Adaptive Server は、テーブルの syspartitions 内の firstpage カラムを読み込み、最初のページをキャッシュに読み込んでから次のページのポインタをたどります。テーブルの最終ページを見つけるまでこれを続けます。

データオンリーロック・テーブル

データオンリーロック・テーブルのページはページ・チェーンでリンクされていないので、データオンリーロック・ヒープ・テーブルに対する select クエリはテーブルの OAM ページとアロケーション・ページを使用して、テーブル内のすべてのローの位置を特定します。OAM ページはアロケーション・ページを指し、アロケーション・ページはテーブルのエクステントとページを指します。

全ページロック・ヒープ・テーブルへのデータの挿入

クラスタード・インデックスのない全ページロック・ヒープ・テーブルにデータが挿入されると、データ・ローは常にテーブルの最終ページに追加されます。テーブルにクラスタード・インデックスがなく、テーブルが分割されていない場合は、このヒープ・テーブルに対する syspartitions.root エントリがヒープの最終ページへのポインタを格納し、データを挿入する必要があるページへ位置付けます。

最終ページがいっぱいになると、現在のエクステント内に新しいページが割り付けられ、チェーンにリンクされます。エクステントがいっぱいになると、Adaptive Server はそのテーブルが使用しているほかのエクステント上に空のページを探します。使用できるページがないとテーブルに新しいエクステントが割り付けられます。

全ページ・ロックを使用するヒープ・テーブルでのサーバのパフォーマンス上の厳しい制限の 1 つは、ロー追加時にページがロックされなければならず、かつそのロックがトランザクションが終了するまで保持されることです。多くのユーザが同時に 1 つの全ページロック・ヒープ・テーブルにローを追加しようとした場合には、各挿入は前のトランザクションが完了するまで待たなければなりません。

このようなヒープ・テーブルの最終ページ競合の問題は、次のような場合に発生します。

- 1 つのローが挿入される場合。
- select into や insert...select コマンド、またはバッチによる複数の insert 文を使って複数のローが挿入される場合。
- テーブルにバルク・コピーが実行される場合。

ヒープ・テーブルの最終ページ競合を回避するには次のような方法があります。

- データページ・ロックまたはデータ・ロー・ロックに切り替える。
- 挿入を別のページに変更するクラスタード・インデックスを作成する。
- テーブルを分割する。テーブルに複数の挿入ポイントが作成され、全ページがロックされたテーブルで複数の「最終ページ」が使用できる。

ロック競合が発生する可能性のあるすべてのトランザクションには、次のガイドラインも適用されます。

- トランザクションを短く保つ。
- トランザクションがロックを取得したら、できるだけネットワーク・アクティビティとユーザからの入力を避ける。

データオンリーロック・ヒープ・テーブルへのデータの挿入

ユーザがデータオンリーロック・ヒープ・テーブルにデータを挿入すると、Adaptive Server は、最後に挿入が行われたページ番号を追跡して、領域を必要とする将来のタスクに対する提案としてそのページ番号を保存します。以降のテーブルへの挿入は、これらのページの 1 つに対して行われます。ページがいっぱいになると、Adaptive Server は新しいページを割り付けて、新しいページ番号で古いヒントを置換します。

多くのユーザが同時にデータを挿入するときのブロックの発生は、APL テーブルと比較して、データオンリーロック・ヒープ・テーブルへの挿入中の方が少なくなります。ブロックが発生した場合には、Adaptive Server は少数の空のページを割り付け、新しく割り付けられたこれらのページをヒントとして使用して、これらのページに対して新しい挿入を行います。

データ・ローロック・テーブルの場合、データへの実際の変更が書き込まれる場合だけブロックが発生します。ローのロックはトランザクションの実行期間中保持されますが、ほかのローをページに挿入できます。ローレベル・ロックによって、複数のトランザクションで、ページに対するロックを保持できます。

データオンリーロック・テーブルでわずかなブロックが発生する場合があります。それは、Adaptive Server は、多くのページが割り付けられた直後に少数のブロックを許可するので、追加のページが割り付けられる前に新しく割り付けられたページが満杯になるためです。

データオンリーロック・テーブルでは、ヒープ・テーブルへの挿入中の競合は大幅に減少しますが、それでも発生することがあります。競合によって挿入の速度が低下した場合には、次のような回避方法があります。

- テーブルがデータページ・ロックを使用している場合は、データ・ロー・ロックに切り替える。
- クラスタード・インデックスを使用してデータの挿入を分散する。
- テーブルを分割する。ヒントが追加され、ブロックが発生した場合に各パーティションに新しいページを割り付けることができる。

ヒープ・テーブルからデータを削除する

利用できるインデックスがないヒープ・テーブルからローを削除すると、Adaptive Server はテーブル内のデータ・ローをスキャンして、削除対象のローを見つけだします。クエリの条件に一致するローがどれだけあるかを知るには、すべてのローを調べるしか方法がありません。

全ページロック・ヒープ・テーブルからの削除

全ページロック・テーブルのページからデータ・ローを削除すると、同じページの、削除するローの後のローが前に移動して、ページ上のデータは互いに隣接したまま配置されるため、ページ内の断片化が防止されます。

データオンリーロック・ヒープ・テーブルからの削除

データオンリーロック・ヒープ・テーブルからローを削除するには、使用できるインデックスがない場合、テーブル・スキャンが必要です。OAM ページとアロケーション・ページを使用して、ページの位置を特定します。

ページ上の領域はすぐにはリカバリされません。データオンリーロック・テーブルのローは固定ロー ID を保持しなければならず、トランザクションがロールバックされる場合には、同じ位置に挿入し直す必要があります。

削除トランザクションがコミットされても、次のいずれかのプロセスがページ上のローをシフトして、使用可能な領域を連続させます。

- ハウスキーピング・ガーベジ・コレクション・プロセス
- ページ上の領域を探す必要のある挿入
- reorg reclaim_space コマンド

ページの最後のローの削除

あるページの最後のローが削除されると、このページの割り付けは解除されます。エクステント内に、まだそのテーブルに使用されているページがある場合は、新しいページが必要になったときに、そのテーブルは割り付けを解除されたページを再使用できます。

割り付けを解除されるページ以外の、エクステント内のすべてのページが空である場合は、エクステント全体が割り付けを解除されます。そのエクステントは、データベース内のほかのオブジェクトに割り付けることができます。テーブルまたはインデックスの最初のデータ・ページが割り付けを解除されることはありません。

ヒープ・テーブルでのデータの更新

ヒープ・テーブルに対する他のオペレーションと同じように、where 句内のカラムに利用できるインデックスがないテーブルに対する update コマンドは、テーブル・スキャンを実行して変更の必要があるローの位置を特定します。

全ページロック・ヒープ・テーブル

全ページロック・ヒープ・テーブルに対する更新の実行には、次のようにいくつかの形があります。

- ローの長さが変わらない場合は、更新されたローが既存のローに入れ替わり、ページ上のデータの移動はない。
- ローの長さが変わり、ページ上に十分な空き領域がある場合は、ローはページ上の同じ位置にとどまるが、ほかのローは前後に移動して、ページ上のローの連続性を保つ。

ページの最後のロー・オフセット・ポインタが調整されて、変更されたローの位置を示す。

- ローがページ上に収まらない場合は、ローが現在のページから削除され、テーブルの最終ページに新しいローとして挿入される。この種の更新は、ヒープの最終ページ上で競合を起こす可能性がある。

データオンリーロック・ヒープ・テーブル

データオンリーロック・テーブルの稼働条件の 1 つは、(テーブルの意図的な再構築時を除き) データ・ローのロー ID は決して変化しないことです。したがって、データオンリーロック・テーブルの更新は、ローがページに収まっているかぎり、上記の最初の 2 つの方法によって実行できます。

しかし、データオンリーロック・テーブル内のローが更新され、ページに収まらなくなった場合には、「ローの転送」と呼ばれるプロセスが以下の手順を実行します。

1 ローは別のページに挿入される。

2 新しいページ上のロー ID を示すポインタは、ローの元の位置に保管される。

ローが転送される場合、インデックスを修正する必要はありません。すべてのインデックスは元のロー ID を示します。

ローを再度転送する必要がある場合、元の位置が更新され、新しいページを示します。転送されたローは、元の位置から 2 回以上ホップすることはありません。

インデックスを更新する必要がないので、ローの転送は更新オペレーション中の同時実行性を向上させます。しかし、タスクでは、元の位置にあるページを読み込み、そして転送されたデータの保管されたページを読み込む必要があるので、データ検索速度が低下する場合があります。

転送されたローをテーブルからクリアするには、reorg コマンドを使用します。

『パフォーマンス&チューニング・シリーズ：クエリ処理と抽象プラン』の「第 1 章クエリ処理について」を参照してください。

Adaptive Server によるヒープ・オペレーションの I/O

クエリがあるデータ・ページを必要とすると、まず Adaptive Server はデータ・キャッシュ内にそのページが存在するかどうかを調べます。そのページが見つからない場合は、ディスクから読み込まなければなりません。新しくインストールされた、論理ページ・サイズが 2K の Adaptive Server には、2K I/O に設定されたデータ・キャッシュが 1 つだけ存在します。各 I/O オペレーションは、1 つの Adaptive Server データ・ページを読み込んだり、書き込んだりします。システム管理者は次の作業ができます。

- 複数のキャッシュを設定する。
- テーブル、インデックス、テキスト・チェーンをキャッシュにバインドする。
- データ・キャッシュを、最大 8 データ・ページ (1 エクステント) までの、ページ・サイズの倍数の容量の I/O を実行するように設定する。

こうしたキャッシュを最も効率的に使用し、I/O オペレーションの数を減らすために、Adaptive Server オプティマイザには次の機能があります。

- 一度に最大 8 ページまでをプリフェッチするように選択できる。
- さまざまなキャッシュ方式から選択できる。

順次プリフェッチまたは大容量 I/O

Adaptive Server のデータ・キャッシュは、大容量 I/O を実行するように設定できます。キャッシュで大容量 I/O が実行可能な場合、Adaptive Server はデータ・ページをプリフェッチできます。

キャッシュは、論理ページ・サイズに応じたバッファ・プールを持つことができるため、Adaptive Server は 1 回の I/O オペレーションで最大 1 エクステント (8 データ・ページ) 全体を読み込むことができます。

I/O オペレーションの実行に必要な時間のほとんどは、ディスクのシークとヘッドの位置付けに費やされるため、16K I/O で 8 ページを読み取るのにかかる時間は、2K I/O で 1 ページ読み取るのとほぼ同じ時間になります。8 つの 2K I/O を使用して 8 ページを読み取ると、1 つの 16K I/O を使用して 8 ページを読み取るよりも約 8 倍の費用がかかります。大容量 I/O を使用すると、テーブル・スキャンのパフォーマンスも大幅に向上します。

1 回の I/O で複数のページがキャッシュに読み込まれると、これらのページは 1 つの単位として扱われます。これらのページはキャッシュ内で共に古くなり、バッファがキャッシュにあるときに単位の中のどれかのページが変更されると、すべてのページが 1 つの単位としてディスクに書き込まれます。

『パフォーマンス&チューニング・シリーズ:基本』の「第 5 章メモリの使い方とパフォーマンス」を参照してください。

注意 大容量 I/O は、論理ページ・サイズが 2K のサーバに基づきます。ページ・サイズが 8K のサーバでは、I/O の基本単位は 8K になります。ページ・サイズが 16K のサーバでは、I/O の基本単位は 16K になります。

ヒープ・テーブル管理

時間の経過とともに、記憶領域が断片化されるに従って、ヒープ・テーブルに対する I/O の効率が低下します。削除と更新によって、次のような状態になる可能性があります。

- 部分的に使用されているページが多数できる。
- エクステントに空のページが多くできるため、大容量 I/O の効率が低下する。
- データオンリーロック・テーブルでローが転送される。

ヒープ・テーブル内の領域を再利用するには、次の方法を実行します。

- reorg rebuild コマンドを使用する (データオンリーロック・テーブルのみ)
- クラスタード・インデックスを作成してから削除する
- bcp (バルク・コピー・ユーティリティ) と truncate table を使用する。

reorg rebuild を使用して領域を再利用する

reorg rebuild は、すべてのデータ・ローを新しいページにコピーして、ヒープ・テーブル上にインデックスを再構築します。reorg rebuild は、データオンリーロック・テーブルでのみ使用できます。

クラスタード・インデックスを作成して領域を再利用する

クラスタード・インデックスを作成するには、データベース内に少なくともテーブルのサイズの 1.2 倍の空き領域が必要です。

「メンテナンス作業に使用可能な領域の確認」(116 ページ) を参照してください。

bcp を使って領域を再利用する

bcp を使って領域を再利用するには、次の手順に従います。

1 bcp を使ってテーブルをファイルにコピーします。

2 truncate table を使ってテーブルをトランケートし、未使用領域を再利用します。

3 bcp を使ってテーブルにコピーを戻します。

分割されたテーブルを取り扱う手順については、『Transact-SQL ユーザーズ・ガイド』の「第 10 章テーブルとインデックスの分割」を参照してください。

bcp の詳細については、『ユーティリティ・ガイド』を参照してください。

トランザクション・ログ：特殊なヒープ・テーブル

Adaptive Server のトランザクション・ログは、データベース内のデータ修正に関する情報を格納する特殊なヒープ・テーブルです。トランザクション・ログは、常にヒープ・テーブルであり、新しいトランザクション・レコードはログの最後に追加されます。トランザクション・ログはインデックスを持っていません。

データ・ページとインデックス・ページとは別の物理デバイス上にログを置いてください。ログは連続しているため、ログ・デバイスのディスク・ヘッドはシークをする必要がほとんどなく、ログへの I/O 率を高いまま維持できます。

トランザクション・ログ書き込みは頻繁に発生します。データベース内のほかの I/O と競合させないでください。ほかの I/O は、通常データ・ページのあちこちで発生します。

リカバリに加えて、次のオペレーションもトランザクション・ログを読み込みます。

- 遅延モードで実行されるデータ修正。
- 挿入されたテーブルまたは削除されたテーブルへの参照を含むトリガ。これらのテーブルは、問い合わせ時に、トランザクション・ログ・レコードから構築される。
- トランザクションのロールバック。

ほとんどの場合、このようなクエリに必要なトランザクション・ログ・ページは、Adaptive Server がページを読み込む必要があるときにもまだデータ・キャッシュ内に残っているため、ディスク I/O は必要ありません。

トランザクション・ログのパフォーマンスを改善する方法については、「トランザクション・ログを独立したディスクに保管する」(6 ページ) を参照してください。

ヒープ・テーブルでの非同期プリフェッチと I/O

物理 I/O を実行する必要のあるタスクは、I/O が完了するのを待つ間、サーバのエンジン (CPU) を解放します。テーブル・スキャンで 1,000 ページを読み込む必要があり、1 ページもキャッシュに入っていない場合は、非同期プリフェッチせずに 2K I/O を実行すると、タスクは、エンジンで処理を実行し、I/O が完了するまでスリープ状態になるループを 1,000 回実行します。16K I/O を使用すると、そのようなループが 125 回しか必要ありません。

非同期プリフェッチにより、テーブル・スキャンを実行するクエリのパフォーマンスは向上します。タスクがアロケーション・ユニットから最初のページをフェッチするときに、非同期プリフェッチはテーブルに属するアロケーション・ユニットのすべてのページを要求できます。1,000 ページのテーブルがちょうど 4 つのアロケーション・ユニットに存在している場合は、タスクが実行とスリープのループを繰り返す回数はずっと少なくて済みます。

I/O のタイプ	ループ	各ループの手順
2K I/O プリフェッチしない	1000	1 ページを要求する。 2 ディスクからページが読み込まれるまでスリープする。 3 ページを要求する。 4 Adaptive Server エンジン (CPU) の実行順が来るまで待つ。 5 ページ上のローを読み込む。
16K I/O プリフェッチしない	125	1 エクステントを要求する。 2 ディスクからエクステントが読み込まれるまでスリープする。 3 Adaptive Server エンジン (CPU) の実行順が来るまで待つ。 4 8 ページのローを読み込む。
プリフェッチする	4	1 アロケーション・ユニット内の全ページを要求する。 2 ディスクから最初のページが読み込まれるまでスリープする。 3 Adaptive Server エンジン (CPU) の実行順が来るまで待つ。 4 全ページのすべてのローをキャッシュに読み込む。

実際のパフォーマンスは、キャッシュ・サイズとデータ・キャッシュ内での他のアクティビティによって異なります。

『パフォーマンス＆チューニング・シリーズ：基本』の「第 6 章非同期プリフェッチのチューニング」を参照してください。

キャッシュとオブジェクトのバインド

テーブルは特定のキャッシュにバインドできます。あるテーブルが特定のキャッシュにバインドされていないが、そのテーブルのデータベースがキャッシュにバインドされている場合は、そのテーブルを対象とするすべての I/O はこのキャッシュ内で行われます。

そうでない場合は、テーブルの I/O はデフォルト・データ・キャッシュ内で行われます。デフォルト・データ・キャッシュを大容量 I/O に設定できます。ヒープ・テーブルを使用するアプリケーションでは、16K I/O に設定されたキャッシュを使用するとパフォーマンスが向上します。

『システム管理ガイド 第 2 巻』の「第 4 章データ・キャッシュの設定」を参照してください。

ヒープ・テーブル、I/O、キャッシュ方式

各 Adaptive Server データ・キャッシュは、バッファの MRU/LRU (most recently used/least recently used) チェーンとして管理されます。キャッシュ内のバッファが古くなるにつれ、バッファは MRU 側の終端から LRU 側の終端へと移動します。

キャッシュ内の更新されたページが、「ウォッシュ・マーカ」という、MRU/LRU チェーン上のあるポイントを越えると、Adaptive Server は、キャッシュに入っていたときに更新されたページに対して非同期書き込みを開始します。これにより、ページがキャッシュの LRU 側の終端に到達したときに、ページがクリーンであり、再使用することができます。

Adaptive Server には、データ・キャッシュを効率的に使用するために、2 つの主な方式があります。

- LRU 置換方式
- MRU、つまり使い捨て置換方式

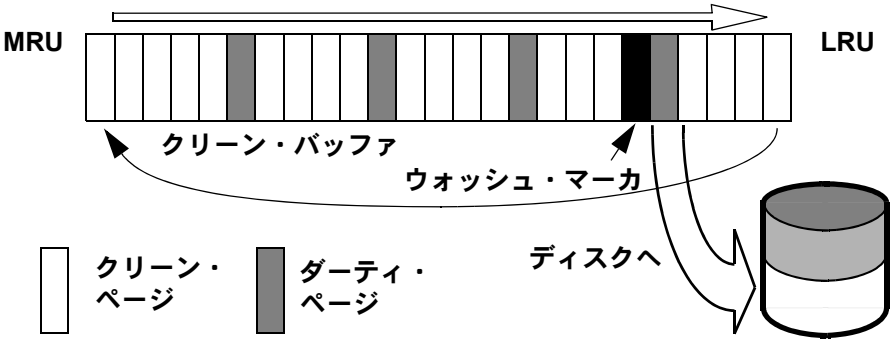
LRU 置換方式

Adaptive Server は、次のような場合にLRU 方式を使用します。

- ページ上のデータを更新する文。
- 1 つのクエリが複数回必要とするページ。
- OAM ページ
- 多数のインデックス・ページ。
- クエリ内に LRU 方式が指定されているクエリ。

LRU 置換方式は、「最も長い間使用されていない」バッファを置き換えて、データ・ページをキャッシュ内に順番に読み込みます。バッファは、データ・バッファ・チェーンの MRU 側の終端に置かれます。キャッシュにページが読み込まれるにつれて、バッファは LRU 側の終端に向かって移動します。

図 2-2: キャッシュの *LRU* 側の終端からクリーン・ページを取り出す *LRU* 方式



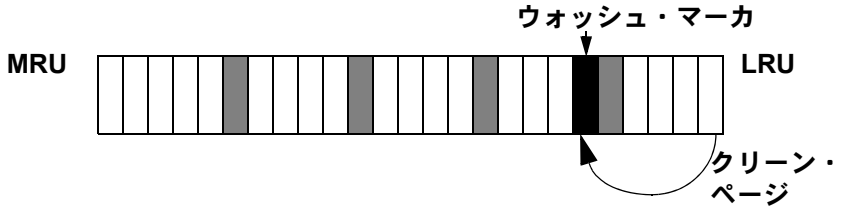
MRU 置換方式

MRU (使い捨て方式) は、クエリがあるページを 1 回しか必要としない場合によく使用されます。MRU では、次のものが対象になります。

- ジョインを使用しないクエリでの、ほとんどのテーブル・スキャン。
- ジョイン・クエリでの 1 つまたは複数のテーブル。

MRU 置換方式は、ヒープ・テーブルでのテーブル・スキャンによく使用されます。この方式は、図 2-3 に示すように、ページをキャッシュ内のウォッシュ・マーカの直前に読み込みます。

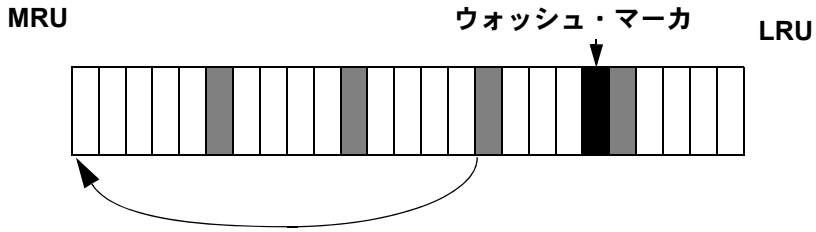
図 2-3: ウォッシュ・マーカの直前にページを配置する *MRU* 方式



ウォッシュ・マーカに 1 回だけ必要なページを配置しても、ほかのページはキャッシュから追い出されません。

使い捨て方式は、そのクエリでディスクから実際に読み込まれるページだけに対して使用されます。テーブルを対象としたそれ以前の操作によってあるページがすでにキャッシュ内に存在する場合は、このページはキャッシュの MRU 側の終端に配置されます。

図 *2-4:* キャッシュ内で必要なページが見つかった場合



選択オペレーションとキャッシング

ほとんどの状況では、ヒープを対象とする単一テーブルの select オペレーションは次の方法を使用します。

- テーブルが利用できる最大サイズの I/O。
- 使い捨て (MRU) 置換方式。

ヒープ・テーブルでは、大容量 I/O を実行する選択オペレーションが非常に効率的になります。Adaptive Server はあるテーブル内のすべてのエクステントを順番に読み取れます。

『パフォーマンス&チューニング・シリーズ：クエリ処理と抽象プラン』の「第 1 章クエリ処理について」を参照してください。

ヒープがネストループ・ジョインの内部テーブルとしてスキャンされていないかぎり、クエリはデータ・ページを 1 回しか必要としないため、MRU 置換方式はページを読み込んだ後にキャッシュから破棄します。

注意　ページ・チェーンが分割されていない場合にかぎり、全ページがロックされたヒープ・テーブルに対して大容量 I/O を使用すると効率的です。「ヒープ・テーブル管理」(44 ページ) を参照してください。

データ修正とキャッシング

Adaptive Server は、変更されたページをキャッシュ内に保持することで、ディスク書き込みの回数を最小限に抑えようとします。あるデータ・ページがキャッシュ内にあるかぎり、多くのユーザがそのデータ・ページを変更できます。これらの変更内容は、トランザクション・ログ内に記録されますが、変更されたデータ・ページとインデックス・ページはすぐにはディスクに書き込まれません。

キャッシングとヒープ・テーブルへの挿入

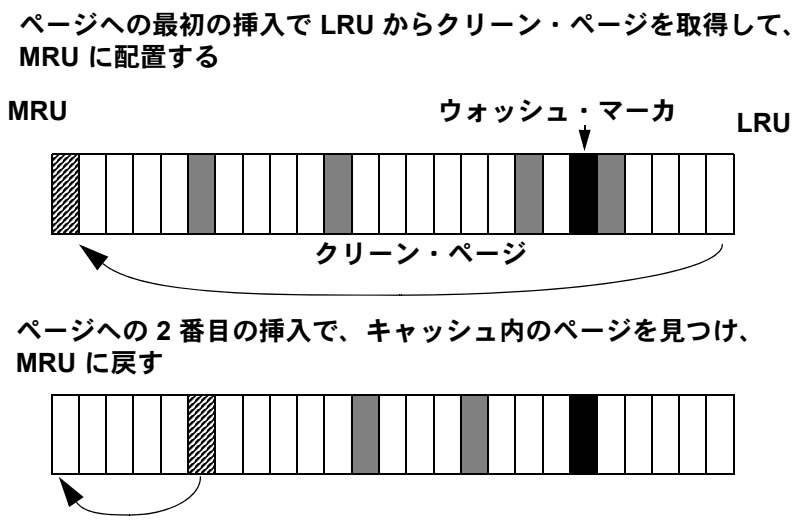
ヒープ・テーブルへの挿入は次のページ上で行われます。

- 全ページ・ロックを使用するテーブルの最終ページ。
- データオンリー・ロックを使用するテーブル上にある、成功した挿入に最近使用されたページ。

ある挿入が、テーブルの新しいページの最初のローである場合は、図 2-5 に示すように、クリーン・データ・バッファが割り付けられてデータ・ページを格納します。このページは、他の処理によってメモリ内にページが読み込まれると、データ・キャッシュ内の MRU/LRU チェーン内を後ろの方へと移動し始めます。

ページがメモリ内にあるうちに、このページへの次の挿入が発生すると、キャッシュ内でこのページが探し出され、MRU/LRU チェーンの先頭に戻ります。

図 2-5: データ・キャッシュでのヒープ・ページへの挿入



変更されたデータ・ページは、ページ・チェーンの LRU 側の終端に到達しないかぎり、キャッシュ内にとどまります。ページは、キャッシュに入っている間は何度でも変更または参照できます。しかし、ディスクに書き込まれるのは、次のどちらかの場合だけです。

- ページがウォッシュ・マーカを越えて移動する。
- チェックポイント・タスクまたはハウスキーピング・ウォッシュ・タスクがディスクに書き込む。

『パフォーマンス&チューニング・シリーズ:基本』の「第 5 章メモリの使い方とパフォーマンス」を参照してください。

キャッシング、およびヒープ・テーブルに対する更新オペレーションと削除オペレーション

ヒープ・テーブルのローを更新または削除する場合には、ヒープ・テーブルにローを挿入する場合と同じような影響がデータ・キャッシュにおよびます。ページがすでにデータ・キャッシュ内にある場合は、ローが変更され、バッファ全体 (I/O サイズに応じて、1 ページから最大 8 ページ) がチェーンの MRU 側の終端に置かれます。

ページがキャッシュ内にない場合は、ディスクからキャッシュに読み込まれ、ページ上のローがクエリ句と一致するかどうかがチェックされます。データ・ページ上のデータを変更する必要があるかどうかによって、MRU/LRU チェーン上に置かれる位置が決まります。具体的には、次のとおりです。

- ページ上のデータを変更する必要がある場合は、バッファは MRU 側の終端に置かれる。バッファはキャッシュ内にとどまり、そこで他のユーザに繰り返し更新されたり、読み取られたりしてから、ディスクにフラッシュされる。
- ページ上のデータを変更する必要がない場合は、バッファはキャッシュ内でウォッシュ・マーカの直前に置かれる。

第 3 章

記憶領域管理プロパティの設定

記憶領域管理プロパティを設定して、テーブルとインデックスの高いパフォーマンスを維持するのに必要な管理作業量を減らすことができます。

この章では、領域の使用量を管理するための主要な記憶領域管理プロパティ、これらのプロパティが領域の使用量に与える影響、異なるテーブルおよびインデックスにこれらのプロパティを適用する方法について説明します。

インデックス管理作業量を減らす

create index コマンドの fillfactor オプションでは、インデックス・ページとクラスタード・インデックスのデータ・ページにどのくらいの割合で格納するかを指定できます。100 パーセント以外の任意の値の fillfactor を指定すると、データ・ローとインデックス・ローは、デフォルトの設定で必要とされるよりも、データベースのディスク領域を多く使用します。

サイズが大きくなるテーブルにインデックスを作成する場合は、fillfactor オプションを指定して、ページ分割がテーブルとインデックスに与える影響を小さくできます。

fillfactor は、インデックスを作成し、テーブルの再編成オペレーションの一環として再び reorg rebuild を使用してインデックスを再構築する場合 (テーブルでクラスタード・インデックスを再構築したり reorg rebuild を実行したりする場合など) に使用します。fillfactor の値は sysindexes に保存されず、データ・ページやインデックス・ページが満杯である状態は維持されません。それ以降のテーブルへの挿入や更新時に fillfactor は維持されません。

fillfactor の値のために当初はインデックスのリーフ・レベルのページの一部のみが満杯状態の場合でも、それ以降の挿入によってこの空き領域が使用され、その後、リーフ・レベルのページが分割される傾向があります。今後行われる挿入によってこのような分割が行われないように、指定された fillfactor の値で、reorg rebuild...index を使用してリーフ・レベルのページを構築し、作成します。fillfactor の値がインデックス全体のリーフ・レベルで別の領域を使用できるように、インデックス・レベル全体で reorg rebuild を実行します。ローカル・インデックスがある場合は、ローカル・インデックス・パーティションのリーフ・ページのみが調整されてリーフ・レベルでの今後の挿入のために別の領域はそのままの状態にするために、パーティション・レベルで reorg rebuild index を実行します。

注意 Adaptive Server 15.0 以降では、ローカル・インデックス・パーティションで reorg rebuild...index を実行できます。

---

create index コマンドを発行するときに fillfactor 値をその一部として指定すると、値は次のように適用されます。

- クラスタード・インデックスの場合 −
  - 全ページロック・テーブルでは、fillfactor はデータ・ページに適用される。
  - データオンリーロック・テーブルでは、fillfactor はインデックスのリーフ・ページに適用され、データ・ページは満杯になる (sp_chgattribute を使用してテーブルの fillfactor を保存していない場合)。
- ノンクラスタード・インデックスの場合 − fillfactor 値は、インデックスのリーフ・ページに適用される。

sp_chgattribute を使用して、reorg rebuild がテーブルで実行されるときに使用される fillfactor の値を保存することもできます。

「fillfactor 値の設定」(56 ページ) を参照してください。

fillfactor を使用するメリット

fillfactor を低い値に設定すると一時的にパフォーマンスが向上します。データベースへの挿入によってデータ・ページまたはインデックス・ページ上で使われる領域の量が増えるため、パフォーマンスの向上が低下します。

低い fillfactor 値を使用すると次のようになります。

- インデックスのリーフ・レベル、および全ページロック・テーブルのデータ・ページでのページ分割が減る。
- 挿入が実行されるクラスタード・インデックスを持つ、データオンリーロック・テーブルにおいて、データ・ローのクラスタリングが向上する。

- 2 つの処理が同一のデータ・ページまたはインデックス・ページを同時に必要とする可能性が減るため、ページ・レベルのロックを使用するテーブルにおけるロック競合を減らすことができる。
- ページ分割の発生頻度が低くなるため、データ・ページとノンクラスタード・インデックスのリーフ・レベルを対象とする大容量 I/O を効率的なまま維持できる。つまり、1 つのエクステント上の 8 ページが連続する確率が高くなる。

fillfactor を使用するデメリット

fillfactor を使用すると (特に非常に低い値で使用する場合)、クエリと管理アクティビティに次のような影響が生じます。

- テーブル・スキャンまたはノンクラスタード・インデックスでのリーフ・レベル・スキャンを実行するクエリに対して読み込まなければならないページ数が増える。

  データ・レベルのページ数が増え、各インデックス・レベルのページ数も増える可能性が高いため、場合によってはインデックスの B ツリー構造にレベルが追加される。
- インデックス・サイズが増加し、効率的にインデックス領域が減少する。ページ・レベルでの fillfactor 値の調整は不可能であるため、使用可能な予約領域が存在していてもデータの分散が偏っているページ分割が頻繁に発生する。
- dbcc コマンドでチェックする必要のあるページ数が増えるため、コマンドが完了するまでの時間が長くなる。
- ダンプする必要があるページ数が増えるため、dump database の実行に要する時間が長くなる。dump database は、データを格納するすべてのページをコピーするが、未使用のページはダンプしない。ダンプとロードではテープも大量に使用される可能性がある。
- fillfactor の値が、時間がたつにつれて小さくなる。fillfactor を使ってページ分割によるパフォーマンス低下を抑える場合は、システムをモニタして、ページ分割がパフォーマンスを低下させ始めた時点でインデックスを再作成する必要がある。

fillfactor 値の設定

sp_chgattribute を使用して、各インデックスとテーブルの fillfactor の割合を保存します。sp_chgattribute で設定する fillfactor は、次の時に適用されます。

- ロック・スキームを使用してテーブルに reorg rebuild を実行する時。
- alter table...partition by を使用してテーブルを再分割する時。
- alter table...lock を使用してテーブルのロック・スキームを変更する時、またはテーブルのコピーを必要とする alter table...add/modify コマンドを使用する時。
- create clustered index の実行時で、テーブルの値が保存されている時。

これらの各コマンドの詳細については、『リファレンス・マニュアル：コマンド』を参照してください。

デフォルトで fillfactor が 0 に設定されていると、新しいインデックスが作成されたときには、インデックス管理処理によって各ページに 2 つのローを追加できるだけの空き領域が確保されます。fillfactor を 100 パーセントに設定すると、追加ローのための空き領域は確保されません。fillfactor がサイズの計算に影響するのは、クラスタード・インデックス・ページ数の計算とリーフ・ページ以外のページ数の計算だけです。この 2 つの計算では、ページあたりのロー数から 2 が引かれています。この -2 の部分を計算式から削除してください。

fillfactor をその他の値にすると、データ・ページとリーフ・インデックス・ページについて、ページあたりのロー数が減ります。fillfactor を指定する場合に正しい値を計算するには、データ・ページの使用可能サイズ (2,016) に fillfactor の値を乗算してください。たとえば、fillfactor の値を 75 パーセントにすると、1 データ・ページの使用可能サイズは 1,512 バイトになります。ページあたりのロー数を計算するときは、この値を 2,016 の代わりに使います。これらの計算については、「データ・ページの数を計算する」(89 ページ) と 「インデックス内のリーフ・ページの数を計算する」(93 ページ) を参照してください。

create clustered index コマンドを実行した結果としてノンクラスタード・インデックスが再構築される場合は、保存された fillfactor 値は適用されません。

- create clustered index で fillfactor 値を指定した場合、その値は各ノンクラスタード・インデックスに適用される。
- create clustered index で fillfactor 値を指定しなかった場合、サーバワイドなデフォルト値 (default fillfactor percent 設定パラメータを使用して設定) がすべてのインデックスに適用される。

fillfactor の例

次の例は、fillfactor 値の適用例を示します。

保存されていない *fillfactor* 値

sysindexes に fillfactor 値が保存されてない場合、create index で fillfactor を指定すると、表3-1 で示すように適用されます。

```
create clustered index title_id_ix
on titles (title_id)
with fillfactor = 80
```

表 *3-1:* テーブルレベルで値が保存されていない場合の *fillfactor* 値の適用

コマンド	全ページロック・テーブル	データオンリーロック・テーブル
create clustered index	データ・ページ：80	データ・ページ：完全なパック リーフ・ページ：80
ノンクラスタード・インデックスの再構築	リーフ・ページ：80	リーフ・ページ：80

ノンクラスタード・インデックスは、create clustered index コマンドで指定した fillfactor を使用します。

create clustered index で fillfactor を指定しない場合、ノンクラスタード・インデックスはサーバワイドなデフォルト値を常に使用します。sysindexes の値は使用しません。

alter table...lock と *reorg rebuild* に使用する値

fillfactor 値が保存されていない場合、alter table...lock と reorg rebuild ではサーバワイドなデフォルト値が適用されます。このデフォルト値は、default fillfactor percentage で設定します。デフォルトの fillfactor は、表3-2 に示すように適用されます。

表 *3-2:* 再構築時に適用される *fillfactor* 値

コマンド	全ページロック・テーブル	データオンリーロック・テーブル
クラスタード・インデックスの再構築	データ・ページ：デフォルト値	データ・ページ：完全なパック リーフ・ページ：デフォルト値
ノンクラスタード・インデックスの再構築	リーフ・ページ：デフォルト	リーフ・ページ：デフォルト

テーブルレベルまたはクラスタード・インデックスの保存されている *fillfactor* 値

次のコマンドは、テーブルの fillfactor 値として 50 を保存します。

```
sp_chgattribute titles, "fillfactor", 50
```

テーブルレベルの値として fillfactor を 50 に設定して保存した場合、この create clustered index コマンドは表3-3 に示すように fillfactor 値を適用します。

```
create clustered index title_id_ix
on titles (title_id)
with fillfactor = 80
```

表 *3-3:* 保存されているクラスタード・インデックスの *fillfactor* 値の使用

コマンド	全ページロック・テーブル	データオンリーロック・テーブル
create clustered index	データ・ページ：80	データ・ページ：50 リーフ・ページ：80
ノンクラスタード・インデックスの再構築	リーフ・ページ：80	リーフ・ページ：80

注意 create clustered index を実行すると、sysindexes に保存されているテーブルレベルの fillfactor 値がすべて 0 にリセットされます。

create clustered index コマンドまたは reorg コマンド実行時に、データオンリーロック・データ・ページを埋めるように指定するには、最初に sp_chgattribute を発行する必要があります。

値が保存されている場合の *alter table...lock* の影響

保存されている fillfactor 値は、alter table...lock コマンドによってテーブルがコピーされてインデックスが再構築されるときに使用されます。

クラスタード・インデックスを持つテーブル

全ページロック・テーブルでは、テーブルとクラスタード・インデックスは sysindexes ローを共有します。したがって、保存できる fillfactor の値は 1 つだけで、その値をテーブルとクラスタード・インデックスに使用できます。テーブル名またはクラスタード・インデックス名を与えると、データ・ページの fillfactor 値を設定できます。次のコマンドは、値 50 を保存します。

```
sp_chgattribute titles, "fillfactor", 50
```

次のコマンドは、値 80 を保存します。前回使用したコマンドで設定された値 50 は上書きされます。

```
sp_chgattribute "titles.clust_ix", "fillfactor", 80
```

上記の sp_chgattribute コマンドの発行後に titles テーブルを変更してデータオンリー・ロックを使用する場合、保存されている fillfactor 値 80 は、データ・ページとクラスタード・インデックスのリーフ・ページ両方に使用されます。

データオンリーロック・テーブルでは、クラスタード・インデックスに関する情報は sysindexes の別のローに格納されます。テーブルに指定した fillfactor 値はデータ・ページに適用され、クラスタード・インデックスに指定した fillfactor 値はクラスタード・インデックスのリーフ・レベルに適用されます。

DOL テーブルを変更して全ページ・ロックを使用する場合、そのテーブルの保存されている fillfactor はデータ・ページに使用されます。クラスタード・インデックスの保存されている fillfactor は無視されます。

表3-4 は、上記の sp_chgattribute コマンド実行後に実行された alter table...lock コマンドを使用して、データ・ページとインデックス・ページに設定される fillfactor 値を示します。

表 *3-4: alter table* 実行時における、保存されている *fillfactor* 値の影響

alter table...lock	クラスタード・インデックスなし	クラスタード・インデックス。
全ページ・ロックからデータオンリー・ロックへの変更	データ・ページ：80	データ・ページ：80 リーフ・ページ：80
データオンリー・ロックから全ページ・ロックへの変更	データ・ページ：80	データ・ページ：80

注意　alter table...lock を使用すると、テーブルの保存されている fillfactor 値はすべて 0 に設定されます。

ノンクラスタード・インデックスの保存されている *fillfactor* 値

各ノンクラスタード・インデックスは、別の sysindexes ローによって表されます。次のコマンドは、2 つのノンクラスタード・インデックス用に異なる値を保存します。

```
sp_chgattribute "titles.ncl_ix", "fillfactor", 90
sp_chgattribute "titles.pubid_ix", "fillfactor", 75
```

表3-5 は、上記の sp_chgattribute コマンドが fillfactor 値の保存に使用される場合の、データオンリーロック・テーブルに対する reorg rebuild コマンドの影響を示します。

表 *3-5: reorg rebuild* 実行時における、保存されている *fillfactor* 値の影響

reorg rebuild	クラスタード・インデックスなし	クラスタード・インデックス。	ノンクラスタード・インデックス
データオンリーロック・テーブル	データ・ページ：80	データ・ページ：50 リーフ・ページ：80	ncl_ix リーフ・ページ：90 pubid_ix リーフ・ページ：75

sorted_data オプションと *fillfactor* オプションの使用

create index の sorted_data オプションは、ソート対象のデータがすでにインデックス・キーで指定された順番に並べられている場合に使用します。これによって、create clustered index はデータのソート、再割り付け、テーブルのデータ・ページの再構築を省略できます。

たとえば、テーブルにバルク・コピーされたデータがすでにクラスタード・インデックス・キーを基準として順番に並べられている場合、sorted_data オプションを使用すると、ソートが実行されずにインデックスが作成されます。データを新しいページにコピーする必要がない場合は、fillfactor は適用されません。ただし、他の create index オプションを使用した場合はコピーが必要になる場合があります。

「ソートされたデータのインデックス作成」(105 ページ) を参照してください。

ローの転送の削減

アプリケーションによって null 値または短い可変長文字フィールドが含まれるローが挿入可能な場合、データオンリーロック・テーブルの予測ロー・サイズを指定できます。以降発生する更新に応じて、このローのサイズは大きくなります。予測ロー・サイズを設定してローの転送を減らします。

たとえば、pubs2 データベースの titles テーブルに、*varchar* カラムと null 値が許されるカラムが数多く存在するとします。このテーブルの最大ロー・サイズは 331 バイトで、平均ロー・サイズ (optdiag によってレポートされます) は 184 バイトですが、多くのカラムで null 値が許されるため、40 バイトより小さいローを挿入できます。データオンリーロック・テーブルでは、サイズの小さいローを挿入して更新すると、ローの転送が発生する可能性があります。

「データオンリーロック・ヒープ・テーブル」(42 ページ) を参照してください。

次のいずれかの方法で、可変長カラムを持つテーブルに予測ロー・サイズを設定します。

- create table 文に exp_row_size パラメータを使用する。
- 既存のテーブルに対しては、sp_chgattribute を使用する。
- default exp_row_size percent 設定パラメータで、サーバワイドなデフォルト値を使用する。この値は可変長カラムを持つテーブルすべてに適用されるが、create table または sp_chgattribute を使用して明示的にロー・サイズを設定した場合、またはデータ・ページ上のローが満杯になるように指定した場合は適用されない。

全ページロック・テーブルに予測ロー・サイズを指定すると、その値は sysindexes に保存されますが、挿入時と更新時には適用されません。後でテーブルをデータオンリー・ロックに変換すると、exp_row_size は変換処理時に適用され、それ以降に発生するすべての挿入と更新にも適用されます。exp_row_size の値はテーブル全体に適用されます。

exp_row_size のデフォルト値、最小値、最大値

表3-6 は、予測ロー・サイズの最小値と最大値と、特殊値である 0 と 1 が何を表すかを示します。

表 *3-6:* 予測ロー・サイズの有効な値

exp_row_size 値	最小値、最大値、特殊値
最小	次の 2 つの値のうち大きい方 • 2 バイト • すべての固定長カラムの合計
最大	データ・ローの最大長
0	サーバワイドなデフォルト値を使用する
1	全ページを満杯にし、ロー拡張用の領域を予約しない

固定長カラムだけを持つテーブルには予測ロー・サイズを指定できません。定義としては、null 値が許されるカラムは可変長です。null 値の場合、カラムの長さは 0 になるためです。

デフォルト値

可変長カラムを持つ、データオンリーロック・テーブルの作成時に、予測ロー・サイズまたは値 0 を指定しない場合、Adaptive Server は default exp_row_size percent 設定パラメータで指定した領域の量を可変長カラムを持つテーブルに使用します。

default exp_row_size のデータ・ページの領域に対する影響については、「サーバワイドなデフォルト予測ロー・サイズの設定」(62 ページ) を参照してください。テーブルのカラムに定義されている長さを参照するには、sp_help を使用します。

create table による予測ロー・サイズの指定

次の create table 文によって、予測ロー・サイズ 200 バイトを指定します。

```
create table new_titles (
      title_id    tid,
      title       varchar(80) not null,
      type        char(12),
      pub_id      char(4) null,
      price       money null,
      advance     money null,
      total_sales int null,
      notes       varchar(200) null,
      pubdate     datetime,
      contract    bit                              )
lock datapages
with exp_row_size = 200
```

予測ロー・サイズの追加または変更

テーブルの予測ロー・サイズを追加または変更するには、sp_chgattribute を使用します。たとえば、new_titles テーブルの予測ロー・サイズを 190 に設定するには、次のように入力します。

```
sp_chgattribute new_titles, "exp_row_size", 190
```

テーブルのロー・サイズの値を、現在の明示的に指定されている値から default exp_row_size percent 値に切り替えるには、次のように入力します。

```
sp_chgattribute new_titles, "exp_row_size", 0
```

ロー拡張用に領域を保存せず、ページを満杯にする場合は、値を 1 に設定します。

sp_chgattribute によって予測ロー・サイズを変更しても、すぐには既存のデータの記憶領域には影響しません。新しい値は、次の時点で適用されます。

- テーブルでクラスタード・インデックスを作成したとき、または reorg rebuild を実行したとき。ローが新しいデータ・ページにコピーされるときに、予測ロー・サイズが適用される。

  exp_row_size の値を増やして、クラスタード・インデックスを再作成するか reorg rebuild を実行すると、テーブルの新しいコピーにはより多くの記憶領域が必要になる場合がある。

- 次にページがデータ修正による影響を受けたとき。

サーバワイドなデフォルト予測ロー・サイズの設定

default exp_row_size percent によって、拡張更新用に予約するページ・サイズの割合を指定します。このデフォルト値を 5 にすると、可変長カラムを持つデータオンリーロック・テーブルすべてについて、データ・ページごとに 5% の空き領域が予約されます。データオンリーロック・テーブルのデータ・ページでは 2002 バイトが使用可能なので、デフォルト値を使用すると ロー拡張用に 100 バイトが予約されます。次のコマンドは、デフォルト値を 10% に設定します。

```
sp_configure "default exp_row_size percent", 10
```

default exp_row_size percent を 0 に設定すると、create table または sp_chgattribute によって予測ロー・サイズが明示的に設定されていないテーブルでは、拡張更新用の領域が予約されません。

テーブルの予測ロー・サイズを create table または sp_chgattribute によって指定した場合、その値はサーバワイドな設定よりも優先します。

テーブルの予測ロー・サイズの表示

テーブルの予測ロー・サイズを表示するには、次のように sp_help を使用します。

```
sp_help titles
```

この値が 0 で、テーブルに null が許されるカラムまたは可変長カラムがある場合は、次のように sp_configure を使用してサーバワイドなデフォルト値を表示します。

```
sp_configure "default exp_row_size percent"
```

次のクエリによって、データベース内の全ユーザ・テーブルの exp_rowsize カラムの値を表示します。

```
select object_name(id), exp_rowsize
from sysindexes
where id > 100 and (indid = 0 or indid = 1)
```

テーブルの予測ロー・サイズの選択

予測ロー・サイズを設定すると、転送されるローの数が減ります。ただし、ローがテーブルに挿入された後に拡張する場合に限ります。予測ロー・サイズを正しく設定すると、次のような効果があります。

- アプリケーションにおいて、転送されるローの割合が小さくなる。
- 予測ロー・サイズの値に必要以上に多くの領域を割り当てたためにデータ・ページで無駄な領域が発生することがない。

optdiag の使用による転送されたローのチェック

データがすでに存在するテーブルについては、optdiag を使用してテーブルの統計を表示します。“Data row size” には、ロー・オーバヘッドを含むデータ・ローの平均長が表示されます。次の例は、optdiag を使用して titles テーブルについて出力した統計で、転送されたローが 12 あり、データ・ローの平均サイズが 184 バイトであることを示しています。

```
Statistics for table:               "titles"

     Data page count:               655
     Empty data page count:         5
     Data row count:                4959.000000000
     Forwarded row count:           12.000000000
     Deleted row count:             84.000000000
     Data page CR count:            0.000000000
     OAM + allocation page count:   6
     Pages in allocation extent:    1
     Data row size:                 184.000000000
```

optdiag を使用すると、テーブルの転送されたローの数をチェックして、アプリケーションにおいて転送されるローの数が減るように exp_row_size が設定されているかどうかを確認することもできます。

『パフォーマンス＆チューニング・シリーズ：統計的分析によるパフォーマンスの向上』の「第 2 章統計テーブルおよび optdiag を使った統計の表示」を参照してください。

転送されたローの *systabstats* のクエリ

systabstats テーブルの forwrowcnt カラムには、テーブルの転送されたローの数が格納されます。オブジェクト ID が 101 以上のすべてのユーザ・テーブルで、転送されたローの数とロー・サイズの平均値を表示するには、次のクエリを使用します。

```
select objectname = object_name(id),
    partitionname = (select name from syspartitions p
      where p.id = t.id and p.indid = t.indid)
  , forwrowcnt, datarowsize
  , exprowsize = (select i.exp_rowsize from sysindexes i
     where i.id = t.id and i.indid = t.indid)
into #temptable
from systabstats t
where id > 100 and indid IN (0,1)

exec sp_autoformat #temptable
```

注意 転送されたローの数はメモリ内で更新され、ハウスキーピング・タスクによって定期的にディスクにフラッシュされます。

SQL を使用して systabstats テーブルを問い合わせ、まず sp_flushstats を使用して統計を最新のものに更新してください。optdiag によって、値の表示前に統計がディスクにフラッシュされます。

---

max_rows_per_page から *exp_row_size* への変換

max_rows_per_page の値を全ページロック・テーブルに設定すると、その値は alter table...lock コマンド実行時に予測ロー・サイズの計算に使用されます。表3-7 は、値の変換方式を示します。

表 *3-7: max_rows_per_page* から *exp_row_size* への変換

max_rows_per_page の値	exp_row_size の値
0	default exp_row_size percent で設定された割合値
1 ～ 254	次のいずれか小さい方 • 最大ロー・サイズ • (論理ページサイズ) − (ページ・ヘッダ・オーバーヘッド) / max_rows_per_page

たとえば、最大定義ロー・サイズが 300 バイトである 2K ページに設定されたサーバ上の全ページロック・テーブルに対して max_rows_per_page を 10 に設定すると、データオンリー・ロックにテーブルを変換した後は、exp_row_size 値は 200 (2002/10) になります。

max_rows_per_page を 10 に設定したが最大定義ロー・サイズが 150 である場合は、予測ロー・サイズの値は 150 に設定されます。

予測ロー・サイズを使用するテーブルのモニタリングと管理

テーブルの予測ロー・サイズの設定後に、アプリケーションにおいて転送されたローの数を調べるには、optdiag を実行するか、systabstats に対して問い合わせます。reorg forwarded_rows を実行すると、転送されるローの数が多く、アプリケーションのパフォーマンスに影響を及ぼします。reorg forwarded_rows は短いトランザクションが使用される非介入型のコマンドであるため、アプリケーションがアクティブなときに実行できます。

『システム管理ガイド 第 2 巻』の「第 9 章 reorg コマンドの使用方法」を参照してください。

転送されるローをパーティション単位で監視し、大量の転送対象ローがあるそれらのパーティションで reorg forwarded rows を実行することができます。『リファレンス・マニュアル：コマンド』を参照してください。

それでもアプリケーションで大量のローが転送される場合は、sp_chgattribute を使用してテーブルの予測ロー・サイズを増やすことを検討してください。

転送されるローの割合がそれほどでなければ、そのままにしておいてもかまいません。reorg を実行して転送されたローをクリアしてもアプリケーションの同時実行性の問題が発生しない場合や、ピーク時外に reorg を実行できる場合、転送されるローが少々存在しても、パフォーマンスに関する重大な問題は発生しません。

テーブルに予測ロー・サイズを設定すると、記憶領域の量と、一連のローを読み込むのに必要な I/O の数が増えます。記憶領域が増えたために I/O の数が多くなった場合、ローの転送はそのままにしておいて、時々 reorg を実行すると、パフォーマンス全体に対する影響が少なくなる場合があります。

転送されるローと挿入用に領域を残す

reservepagegap 値を設定して記憶領域の断片化を減らすと、reorg rebuild の実行やテーブルのインデックス再作成など、管理作業の実行頻度も減少します。データオンリーロック・テーブルで良好なパフォーマンスを得るには、テーブルによって使用されるページ、エクステント、アロケーション・ユニットのデータのクラスタリング状態が良好である必要があります。

転送されたローとインデックス・キー順に挿入されたローを格納するための領域が近くにある場合は、物理記憶領域内のデータ・ページとインデックス・ページのクラスタリングは良好な状態にあります。reservepagegap 記憶領域管理プロパティは、追加のページの割り付けが必要な場合に、拡張用に空のページを予約するために使用します。

ローとページのクラスタ率は通常 1.0 です。クラスタード・インデックスをテーブルに作成した直後や、reorg rebuild を実行した直後は、1.0 の近似値となります。しかしデータを修正すると、ローの転送が発生して、挿入されたローを格納するためにデータ・ページとインデックス・ページをさらに割り付ける必要がある場合があります。

全ページとデータオンリーロック・テーブルのデータおよびインデックス・レイヤ・ページに対して、ページ・ギャップの予約値を設定できます。

エクステント割り付けコマンドと *reservepagegap*

エクステントの割り付けとは、ページが一度に 1 ページずつではなく 8 の倍数単位で割り付けられるということです。これによって、ログ・レコードが 8 つではなく 1 つのみが書き込まれてロギング・アクティビティが減少します。

エクステント割り付けを実行するコマンドは、select into、create index、reorg rebuild、bcp、alter table...lock、alter table...unique です。primary key 制約オプションもエクステント割り付けを実行します。制約によってインデックスが作成されるためです。カラムの追加、削除、修正、またはテーブル分割スキームの変更を実行する alter table コマンドではテーブルコピー・オペレーションも必要になる場合があります。デフォルトでは、これらすべてのコマンドでエクステント割り付けが使用されます。

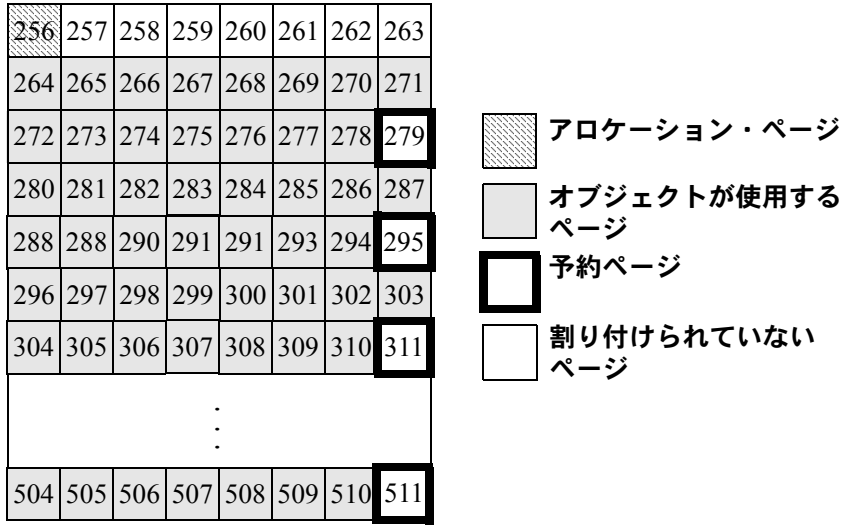
reservepagegap はページ単位で指定し、満杯のページに対する、空のページの比率を示します。たとえば、reservepagegap に 8 を指定した場合、エクステント割り付けオペレーションによって 7 ページが使用され、8 番目のページは空になります。

各アロケーション・ユニットの最初のページはアロケーション・ページを格納するため、エクステント割り付けオペレーションでは使用されません。たとえば、大きいテーブルにクラスタード・インデックスを作成して、ページ・ギャップの予約値を指定しない場合、各アロケーション・ユニットには空のページが 7 つ、使用済みのページが 248、アロケーション・ページが 1 つ存在します。Adaptive Server ではこの 7 つの空のページをローの転送とテーブルへの挿入に使用できます。これにより、転送されたローとクラスタード・インデックスによる挿入を同じアロケーション・ユニットに格納できます。reservepagegap を使用すると、各アロケーション・ユニットの空のページが増えます。

reservepagegap の使用方法の詳細については、『Transact-SQL ユーザーズ・ガイド』の「第 12 章テーブルのインデックスの作成」を参照してください。

図 3-1 は、テーブルで reservepagegap の値を 16 に指定してクラスタード・インデックスを作成した後の、アロケーション・ユニットの状態を示します。最初のエクステントをアロケーション・ユニットと共有しているページは使用されず、テーブルに割り付けられません。ページ 279、295、311 は、テーブルに割り付けられたエクステントの未使用ページです。

図 *3-1:* クラスタード・インデックス作成後に予約されたページ



create table によるページ・ギャップの予約値の指定

次の create table コマンドによって、reservepagegap 値を 16 に指定します。

```
create table more_titles (
      title_id    tid,
      title       varchar(80) not null,
      type        char(12),
      pub_id      char(4) null,
      price       money null,
      advance     money null,
      total_sales int null,
      notes       varchar(200) null,
      pubdate     datetime,
      contract    bit
    )
lock datarows
with reservepagegap = 16
```

more_titles テーブルにエクステント割り付けを実行するオペレーションによって、15 の満杯のページにつき、空のページが 1 つできます。分割されたテーブルでは、すべてのパーティションに reservepagegap 値が適用されます。

reservepagegap のデフォルト値は 0 です。つまり、領域は予約されません。

create index によるページ・ギャップの予約値の指定

次のコマンドは、ノンクラスタード・インデックス・ページに reservepagegap 値 10 を指定します。

```
create index type_price_ix
on more_titles(type, price)
with reservepagegap = 10
```

reservepagegap の値を、alter table...constraint オプション、primary key、unique を使用して指定することもできます。これらによって、インデックスが作成されます。分割されたテーブルのローカル・インデックスの reservepagegap 値は、すべてのローカル・インデックス・パーティションに適用されます。

次の例で、ユニークな制約を作成します。

```
alter table more_titles
add constraint uniq_id unique (title_id)
with reservepagegap = 20
```

reservepagegap の変更

titles テーブルのページ・ギャップの予約値を 20 に変更するには、次のように入力します。

```
sp_chgattribute more_titles, "reservepagegap", 20
```

次のコマンドは、インデックス title_ix のページ・ギャップの予約値を 10 に設定します。

```
sp_chgattribute "titles.title_ix",
    "reservepagegap", 10
```

sp_chgattribute はシステム・テーブルの値だけを変更します。このプロシージャの実行により、データがデータ・ページに移動することはありません。テーブルの reservepagegap を変更すると、その後のデータの格納に次の影響があります。

- データがテーブルにバルク・コピーされた場合、ページ・ギャップの予約値は新たに割り付けられた領域すべてに適用されるが、既存のページの記憶領域は影響を受けない。
- テーブル・データをコピーして新しいバージョンのテーブルを作成するコマンドは、オペレーションのデータ・コピー・フェーズでページ・ギャップの予約値を適用する。たとえば、reorg rebuild または alter table を使用するテーブルのロック・スキームや分割スキームの変更や、データ・コピーを必要とするスキーマの変更は、どちらもページ・ギャップの予約値に適用される。
- クラスタード・インデックス作成時に、そのテーブル用に保存されているページ・ギャップの予約値がデータ・ページに適用される。

ページ・ギャップの予約値は次の時にインデックス・ページに適用されます。

- alter table...lock の実行時。インデックスが再構築される。
- alter table を使用するロック・スキームや分割スキームの変更時、またはデータ・コピーを必要とするスキーマの変更時の reorg rebuild のインデックス再構築フェーズ。
- クラスタード・インデックスを作成する、create clustered index コマンドと alter table コマンドの実行時。ノンクラスタード・インデックスが再構築される。

reservepagegap の例

この項で示すいくつかの例は、alter table コマンドと reorg rebuild コマンドの実行時に reservepagegap がどのように適用されるかを示します。

reservepagegap をテーブルだけに指定する場合

次のコマンドは、テーブルに reservepagegap を指定しますが、create index コマンドの値は指定しません。

```
sp_chgattribute titles, "reservepagegap", 16
create clustered index title_ix on titles(title_id)
create index type_price on titles(type, price)
```

表3-8 は、reorg rebuild の実行時またはクラスタード・インデックスの削除および作成時に適用される値を示します。

表 *3-8:* テーブルレベルで値が保存されている場合の *reservepagegap* 値の適用

コマンド	全ページロック・テーブル	データオンリーロック・テーブル
create clustered index、または reorg rebuild によるクラスタード・インデックスの再構築	データ・ページとインデックス・ページ：16	データ・ページ：16 インデックス・ページ：0 (満杯のエクステント)
ノンクラスタード・インデックスの再構築	インデックス・ページ：0 (満杯のエクステント)	インデックス・ページ：0 (満杯のエクステント)

クラスタード・インデックスを持つ全ページロック・テーブルでは、データ・ページとインデックス・ページの両方に reservepagegap が適用されます。データオンリーロック・テーブルでは、reservepagegap はデータ・ページには適用されますが、クラスタード・インデックス・ページには適用されません。

reservepagegap をクラスタード・インデックスに指定する場合

次のコマンドは、テーブルとクラスタード・インデックスに異なる reservepagegap 値と、ノンクラスタード・インデックスの type_price に値を指定します。

```
sp_chgattribute titles, "reservepagegap", 16
create clustered index title_ix on titles(title)
    with reservepagegap = 20
create index type_price on titles(type, price)
    with reservepagegap = 24
```

表3-9 は、この一連のコマンドの影響を示します。

表 *3-9:* インデックス・ページに適用された *reservepagegap* 値

コマンド	全ページロック・テーブル	データオンリーロック・テーブル
create clustered index、または reorg rebuild によるクラスタード・インデックスの再構築	データ・ページとインデックス・ページ：20	データ・ページ：16 インデックス・ページ：20
ノンクラスタード・インデックスの再構築	インデックス・ページ：24	インデックス・ページ：24

全ページロック・テーブルでは、create clustered index で指定した reservepagegap は、データ・ページとインデックス・ページ両方に適用されます。データオンリーロック・テーブルでは、create clustered index で指定した reservepagegap は、インデックス・ページだけに適用されます。テーブルの reservepagegap の値が保存されている場合は、その値がデータ・ページに適用されます。

reservepagegap の値の選択

reservepagegap の値は、次の点を考慮して選択します。

- テーブルがクラスタード・インデックスを持っているか。
- テーブルへの挿入の割合。
- テーブルにおいて転送されたローの数。
- クラスタード・インデックスを再構築したり、reorg rebuild コマンドを使用したりする頻度。

reservepagegap の値を適切に設定すると、インデックスの定期管理作業の合間でもテーブル、クラスタード・インデックス、ノンクラスタード・リーフレベル・ページのクラスタ率が良好な状態に保たれるように、新しいページのテーブルとインデックスへの割り付けに十分な数のページを確保できます。

reservepagegap 設定のモニタリング

optdiag を使用すると、テーブルのクラスタ率と転送されたローの数をチェックできます。クラスタ率が下がっている場合、reorg コマンドを実行するとパフォーマンスが向上することもあります。

- クラスタード・インデックスのデータ・ページのクラスタ率が低い場合は、reorg rebuild を実行するか、クラスタード・インデックスを削除して再作成する。
- インデックス・ページのクラスタ率が低い場合は、ノンクラスタード・インデックスを削除して再作成する。

クラスタ率維持のために reorg コマンドを実行する頻度を減らすには、reorg rebuild の実行前に reservepagegap を多少増やしてください。

『パフォーマンス&チューニング・シリーズ：統計的分析によるパフォーマンスの向上』の「第 2 章統計テーブルおよび optdiag を使った統計の表示」を参照してください。

reservepagegap オプションと *sorted_data* オプション

すでにインデックス・キー順でデータ・ページに格納されているテーブルにクラスタード・インデックスを作成する場合、sorted_data オプションを使用すると、非分割テーブルでキー順でデータ・ページをコピーする手順を省くことができます。create clustered index コマンドに reservepagegap オプションを指定すると、テーブルで使用されるエクステント上に、拡張用の領域として空のページを確保できます。どちらのオプションが適用されるかについて、ルールがあります。sp_chgattribute によって reservepagegap 値を変更しても、両方のオプションのメリットを得ることはできません。

create clustered index で両方のオプションを指定すると、次のようになります。

- 分割されていない全ページロック・テーブルでは、create clustered index で指定した reservepagegap 値が sysindexes にすでに保存されている値と一致する場合、sorted_data オプションが優先される。データ・ページはコピーされず、reservepagegap は適用されない。create clustered index コマンドで指定した reservepagegap 値が sysindexes に保存されている値と異なる場合、データ・ページがコピーされ、コマンドで指定した reservepagegap 値がそのコピーされたページに適用される。
- データオンリーロック・テーブルでは、create clustered index で指定した reservepagegap 値は、インデックス・ページだけに適用される。データ・ページはコピーされない。

reservepagegap のほかにも、create clustered index のオプションにはソートが必要なものがあります。このようなオプションが存在すると、sorted_data オプションは無視されます。詳細については、「ソートされたデータのインデックス作成」(105 ページ)を参照してください。

reservepagegap の使用については、特に次の点が関連します。

- 分割されたテーブルでは、データ・ページのコピーが必要な create clustered index コマンドによって並列ソートが実行され、データ・ページがソート順にコピーされる。このとき、ページが新しいエクステントにコピーされるときに reservepagegap 値が適用される。
- sorted_data オプションが create clustered index の他のオプションによって無効にされなかった場合は、テーブルがスキャンされて、データがキー順に格納されているかどうかが確認される。スキャン時にインデックスが構築されるが、ソートは実行されない。

表3-10 は、これらのルールがどのように適用されるかを示します。

表 *3-10: reservepagegap* オプションと *sorted_data* オプション

	分割されたテーブル	分割されていないテーブル
全ページロック・テーブル		
create index with sorted_data と、保存されている値と一致する reservepagegap 値	データ・ページはコピーされない。ページがスキャンされるときにインデックスが構築される。	データ・ページはコピーされない。ページがスキャンされるときにインデックスが構築される。
create index with sorted_data と、保存されている値と異なる reservepagegap 値	並列ソートが実行される。ページが新しいソート順で新しいロケーションに格納されるときに reservepagegap が適用される。	データ・ページがコピーされ、reservepagegap が適用される。ページがコピーされるときにインデックスが構築されるが、ソートは実行されない。
データオンリーロック・テーブル		
create index with sorted_data とすべての reservepagegap 値	reservepagegap はインデックス・ページだけに適用される。データ・ページはコピーされない。	reservepagegap はインデックス・ページだけに適用される。データ・ページはコピーされない。

オプションの使用方法

テーブルのデータ・ページを再分配して拡張用の領域を確保するには、次のようにします。

- 全ページロック・テーブルでは、sorted_data オプションを使用しないで、クラスタード・インデックスを削除して再作成する。sysindexes に保存されている値が希望する値ではない場合は、create clustered index を使用して希望の reservepagegap 値を指定する。
- データオンリーロック・テーブルでは、sp_chgattribute を使用してテーブルの reservepagegap 値を希望の値に設定し、次にクラスタード・インデックスを削除して再作成する。このとき、sorted_data オプションは使用しない。テーブル用に保存された reservepagegap は、データ・ページに適用される。create clustered index コマンドで reservepagegap を指定した場合、インデックス・ページだけに適用される。

データ・ページをコピーしないでクラスタード・インデックスを作成するには、次のようにします。

- 全ページロック・テーブルでは、sorted_data オプションを使用する。ただし、create clustered index コマンドで reservepagegap を指定してはいけない。別の方法として、sysindexes に保存されている値と一致する値を指定する。
- データオンリーロック・テーブルでは、sorted_data オプションを使用する。create clustered index コマンドで reservepagegap 値を指定した場合、その値はインデックス・ページだけに適用され、データ・ページのコピーは実行されない。

バルク・コピー・オペレーション、select into コマンド、またはエクステント割り付けを使用する別のコマンドを実行した後に sorted_data オプションを使用するには、データ・ページに希望の reservepagegap 値を設定してから、データをコピーしたり select into コマンドにその値を指定したりしてください。データ・ページが割り付けられて満杯になったら、次のコマンドによって reservepagegap をインデックス・ページだけに適用します。これは、データ・ページのコピーが必要ないためです。

```
create clustered index title_ix
on titles(title_id)
with sorted_data, reservepagegap = 32
```

全ページロック・テーブルでの *max_rows_per_page* の使用

ページあたりのローの最大数を設定すると、全ページロック・テーブルとインデックスにおける競合を減らすことができます。ほとんどの場合、テーブルを変換してデータオンリー・ロック・スキームを使用することをおすすめします。ロック・スキームを変更できない事情があり、全ページロック・テーブルまたはインデックスで競合の問題が発生する場合、max_rows_per_page 値を設定するとパフォーマンスが向上することがあります。

インデックス・ページとデータ・ページのローの数が少ないほど、ロックの競合が発生する可能性は低くなります。キーがより多くのページに分散して設定されているほど、自分が必要とするページと他のユーザが必要とするページが異なる可能性が高くなります。ページあたりのロー数を変更するには、テーブルとインデックスの fillfactor 値または max_rows_per_page 値を調整します。

fillfactor (sp_configure または create index で定義されます) は、Adaptive Server が既存データに新しいインデックスを作成するときに各データ・ページをどのくらいの割合で埋めるかを決定します。fillfactor はページ分割を減らすのに役立つため、インデックスに対する排他ロックの数も最小限に抑えられ、パフォーマンスが向上します。ただし、fillfactor の値はデータへの今後の変更には維持されません。max_rows_per_page (sp_chgattribute、create index、create table、または alter table で定義されます) は fillfactor と似ていますが、データが変更されても Adaptive Server が max_rows_per_page 値を維持するという点が異なります。

fillfactor または max_rows_per_page を使用してページあたりのロー数を減らすと、同じ個数のデータ・ページを読むのに I/O が増え、データ・キャッシュから同じパフォーマンスを得るためにメモリやロックも増えます。さらに、テーブルの max_rows_per_page 値を小さくすると、テーブルにデータを挿入するときのページ分割の回数が増えます。

ロック競合の低減

create table、create index、または alter table コマンドに指定される max_rows_per_page 値は、1 つのデータ・ページ、クラスタード・インデックス・リーフ・ページ、またはノンクラスタード・インデックス・リーフ・ページに設定可能なロー数を制限します。これにより、ロック競合が減少して、頻繁にアクセスされるテーブルの同時実行性が改善されます。

max_rows_per_page は、ヒープ・テーブルのデータ・ページか、インデックスのリーフ・ページに適用されます。テーブルまたはインデックスの作成後は保持されない fillfactor とは異なり、ローの追加または削除にも max_rows_per_page 値は Adaptive Server で保持されます。

次のコマンドでは、sales テーブルを作成し、1 ページあたりのローの最大数を 4 に制限します。

```
create table sales
      (stor_id          char(4)       not null,
      ord_num           varchar(20)   not null,
      date              datetime      not null)
      with max_rows_per_page = 4
```

max_rows_per_page 値を指定してテーブルを作成してから、max_rows_per_page を指定しないでテーブルにクラスタード・インデックスを作成すると、クラスタード・インデックスは create table 文の max_rows_per_page 値を継承します。また、max_rows_per_page を指定してクラスタード・インデックスを作成すると、テーブルのデータ・ページの値が変更されます。

インデックスと *max_rows_per_page*

max_rows_per_page のデフォルト値は、0 です。この設定では、データ・ページが満杯になるようにクラスタード・インデックスを作成し、リーフ・ページが満杯になるようにノンクラスタード・インデックスを作成し、クラスタード・インデックスとノンクラスタード・インデックスの両方のインデックス B ツリー内に、良好に動作できるだけの量の領域が残されます。

ヒープ・テーブルとクラスタード・インデックスについては、max_rows_per_page の範囲は 0 ～ 256 です。

ノンクラスタード・インデックスについては、max_rows_per_page の最大値は、リーフ・ページに入るインデックス・ローの数ですが、256 を超えることはありません。最大値を算出するには、ページ・サイズから 32 (ページ・ヘッダのサイズ) を減算し、差をインデックス・キーのサイズで除算します。次の文では、ノンクラスタード・インデックスの max_rows_per_page の最大値を求めます。

```
select (@@pagesize - 32)/minlen
    from sysindexes
    where name = "indexname"
```

select into と max_rows_per_page

デフォルトでは、select into はベース・テーブルの max_rows_per_page 値を継承しませんが、max_rows_per_page の値が 0 の新しいテーブルを作成します。ただし、select into に with max_rows_per_page オプションを追加して 0 以外の値を指定することができます。

既存データに対する *max_rows_per_page* の適用

既存のデータに max_rows_per_page 値を適用するには、次に示すようにいくつかの方法があります。

- テーブルにクラスタード・インデックスがある場合は、このクラスタード・インデックスを削除して、異なる max_rows_per_page 値を使用してインデックスを再作成する。
- sp_chgattribute を使用して max_rows_per_page の値を変更したら、reorg rebuild を使用してテーブル全体とそのインデックスを再構築する。たとえば、authors テーブルの max_rows_per_page 値を 1 に変更するには、次のように入力します。

  ```
  sp_chgattribute authors, "max_rows_per_page", 1
  go
  reorg rebuild authors
  go
  ```

- bcp を使用してテーブルを再配置し、次を実行する。
  a テーブル・データをコピー・アウトする。
  b テーブルをトランケートする。
  c sp_chgattribute を使用して、max_rows_per_page 値を設定する。
  d データをコピー・インする。

第 4 章

テーブルおよびインデックスのサイズ

この章では、テーブルとインデックスの現在のサイズを決定する方法、および領域の計画を立てるためにテーブルのサイズを見積もる方法について説明します。

クエリとシステムの動作を理解するためには、テーブルとインデックスのサイズを知ることが重要です。チューニング作業のいくつかの段階の中では、次の目的でサイズについての情報が必要になります。

- 特定のクエリ・プランに対応する statistics io のレポートについて理解する。『パフォーマンス&チューニング・シリーズ：統計的分析によるパフォーマンスの向上』の「第 1 章 set statistics コマンドの使用」には、statistics io を使用して実行対象の I/O を調査する方法が記載されている。
- オプティマイザによるクエリ・プランの選択について理解する。Adaptive Server のコストベースのオプティマイザは、使う可能性がある各アクセス・メソッドについて必要な物理 I/O と論理 I/O を見積もり、最もコストの低い方法を選択する。特定のクエリ・プランが通常と異なっていると思える場合は、dbcc traceon(302) を使うと、オプティマイザがそのプランを選択した理由を確認できる。この出力には、ページ数の見積もりも含まれる。
- データベース・オブジェクトのサイズと、そのオブジェクトについて予想される I/O パターンに基づいて、オブジェクトの配置を決定する。データベース・オブジェクトを複数のデータベース・デバイスに分散し、ディスクの読み込みと書き込みが均等に分散することによって、パフォーマンスを向上させる。オブジェクトの配置については、「第 1 章データの物理的配置の制御」を参照。

- パフォーマンスの変化について理解する。オブジェクト・サイズが大きくなると、パフォーマンス特性が変化する可能性がある。たとえば、あるテーブルが頻繁に使用され、常にキャッシュに格納されているとする。このテーブルが大規模になって、キャッシュに収まらなくなると、このテーブルにアクセスするクエリのパフォーマンスが突然低下する可能性がある。複数回のスキャンを必要とするジョインでは、特にその可能性が高い。
- キャパシティについて計画を立てる。新しいシステムを設計する場合、または従来のシステムの拡張を計画する場合は、物理ディスクとメモリのプランニングのために、必要となる領域について知っておく必要がある。
- Adaptive Server Monitor と sp_sysmon の物理 I/O に関するレポートを理解する。

テーブルとインデックスのサイズの決定

Adaptive Server には、現在のテーブルまたはインデックスのサイズに関する情報を提供するツール、サイズを見積もるためのツールなどがあります。

- optdiag は、テーブルとインデックスについて、サイズなどの多くの統計情報を表示する。『パフォーマンス&チューニング・シリーズ：統計的分析によるパフォーマンスの向上』の「第 2 章統計テーブルおよび optdiag を使った統計の表示」を参照。
- sp_spaceused は、既存のテーブルとインデックスの現在のサイズに関する情報を表示する。
- sp_estspace は、ロー数をパラメータとして指定すると、テーブルのインデックスのサイズを見積もる。

この章では、テーブルとインデックスのサイズ計算に使用できる式についても説明します。sp_spaceused と optdiag は、実際の領域の使用状況をレポートします。この章で説明するほかの方法を使うと、サイズの見積もりができます。

sp_helpartition は、分割されたテーブルについて、分割後のテーブルの各部分に格納されているページ数をレポートします。『Transact-SQL ユーザーズ・ガイド』の「第 10 章テーブルとインデックスの分割」を参照してください。

データ更新がオブジェクト・サイズに及ぼす影響

時間がたつと、個々のテーブルのランダムに分散されたデータ更新の影響により、平均で約 75 パーセント満たされたデータ・ページとインデックス・ページが生成される傾向があります。主な要因は次のとおりです。

- クラスタード・インデックスのある、全ページロック・テーブルのページ上に置かれる必要のあるローが挿入され、そのローを入れるための空きページがないと、ページが分割され、約 50 パーセント満たされたページが 2 つ生成される。
- ヒープ・テーブルまたはクラスタード・インデックスのあるテーブルからローが削除されると、ページで使用されている領域が減る。ローが少ししかない、または 1 つしかないページができる場合もある。
- 削除とページ分割が何回か発生したあとでクラスタード・インデックスのあるテーブルにローが挿入されると、分割されたページまたはローが削除されたページが満杯になる確率が高くなる。

満杯になっているインデックス・ページにローを挿入する必要がある場合にもページ分割は発生するため、結果的にインデックス・ページも平均して約 75 パーセント満たされる傾向にあります。ただし、インデックスの削除と再作成を定期的に行う場合を除きます。

optdiag を使ってオブジェクト・サイズを表示する

optdiag コマンドは、テーブルとインデックスのサイズを含む、テーブル、インデックス、カラムの統計を表示します。クエリのチューニングを行う場合、必要なすべての統計を表示するためには optdiag を使用するのが最適です。次に、pubtune データベース内の titles テーブルのサンプル・レポートを示します。

```
Table owner:                          "dbo"
Statistics for table:                 "titles"
     Data page count:                 662
     Empty data page count:           10
     Data row count:                  4986.0000000000000000
     Forwarded row count:             18.0000000000000000
     Deleted row count:               87.0000000000000000
     Data page CR count:              86.0000000000000000
     OAM + allocation page count:     5
     First extent data pages:         3
Data row size:                  238.8634175691937287
```

『パフォーマンス＆チューニング・シリーズ：統計的分析によるパフォーマンスの向上』の「第 2 章統計テーブルおよび optdiag を使った統計の表示」を参照してください。

optdiag のメリット

optdiag のメリットは次のとおりです。

- データベース内のすべてのテーブル、または単一のテーブルの統計を表示できる。
- optdiag の出力には、インデックスの高さやローの平均長などのクエリのコストを判断するのに便利な追加情報がある。
- 他のチューニング作業でも頻繁に optdiag を使うので、このコマンドによるレポートを手元に置いておくとよい。

optdiag のデメリット

optdiag の基本的なデメリットは出力量が多いことです。テーブル内のページ数など、わずかな情報しか必要としない場合には、ほかの方法を使う方が速く、システムのオーバヘッドが少なくなります。

sp_spaceused を使ったオブジェクト・サイズの表示

sp_spaceused システム・プロシージャは、オブジェクトの OAM ページに格納されている値を読み込んで、オブジェクトが使用している領域に関する情報をすばやくレポートします。

```
                sp_spaceused titles
name         rowtotal reserved   data      index_size  unused
------------ -------- ---------- --------- ----------- --------
titles       5000        1756 KB   1242 KB     440 KB     74 KB
```

rowtotal 値は正確でない場合もあります。OAM ページ上でこの値を更新しない Adaptive Server の処理もあるからです。update statistics コマンド、dbcc checktable コマンド、dbcc checkdb コマンドは、OAM ページの rowtotal 値を修正します。表 4-1 では、sp_spaceused の出力にある見出しについて説明します。

表 *4-1: sp_spaceused* の出力

カラム	意味
rowtotal	ロー数の見積もりをレポートする。この値は、OAM ページから読み込まれる。正確でない場合もあるが、select count(*) よりもすばやく、わずかな競合で見積もりができる。
reserved	テーブルとテーブルのインデックス用に予約されているページ数をレポートする。オブジェクトに割り付けられているエクステントの使用されているページも、使用されていないページもレポートする。これは data、index_size、unused の合計。
data	テーブルが使用するページの容量をキロバイト単位でレポートする。
index_size	インデックスが使用するページの合計容量をキロバイト単位でレポートする。
unused	オブジェクトのインデックス用に割り付けられている未使用のページの容量も含み、オブジェクトに割り付けられているエクステントの未使用のページの容量をキロバイト単位でレポートする。

インデックスのサイズを個別にレポートするには、次のコマンドを使用します。

```
                sp_spaceused titles, 1
 index_name           size       reserved   unused
 -------------------- ---------- ---------- ----------
 title_id_cix         14 KB      1294 KB    38 KB
 title_ix             256 KB     272 KB     16 KB
 type_price_ix        170 KB     190 KB     20 KB

name         rowtotal reserved   data       index_size  unused
------------ -------- ---------- ---------- ----------- --------
titles       5000        1756 KB   1242 KB     440 KB     74 KB
```

全ページロック・テーブル上のクラスタード・インデックスの場合は、size 値はルート・インデックス・ページと中間インデックス・ページ用に使用されている領域の容量を示します。reserved 値は、インデックスのサイズと予約データ・ページと使用中データ・ページの数を含みます。

sp_spaceused 構文の "1" は、詳細なインデックス情報が出力対象であることを示します。インデックス ID またはほかの情報とは関係ありません。

sp_spaceused のメリット

sp_spaceused のメリットは次のとおりです。

- テーブルとインデックスの OAM ページ内の値だけを使って結果を返すため、過剰な I/O とロックを実行しないで、すばやくレポートする。
- オブジェクトの拡張に備えて予約されているが、現在はデータの格納に使用されていない領域の合計を表示する。
- インデックスのサイズと、text、image、およびロー外にある Java カラムの記憶領域のサイズの詳細なレポートを表示する。

注意 各パーティションのページ数をレポートするには sp_helpartition を使用します。sp_helpartition は sp_spaceused と同レベルに詳細なレポートはしませんが、パーティションが使用する領域の量についての概要を示します。Adaptive Server バージョン 15.0.2 以降では、sp_spaceusage はインデックス・レベルやパーティション・レベルで使用される領域、断片化など、さまざまなテーマについての詳細な情報を示します。

これらすべてのシステム・プロシージャの詳細については、『ASE 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。

sp_spaceused のデメリット

sp_spaceused のデメリットは次のとおりです。

- ローの合計数と領域の使用状況を示す値が不正確な場合がある。
- ほとんどのクエリ・チューニング作業では計測単位としてページが使用されているのに対し、出力の単位としてキロバイトのみが使用されている。ただし、sp_spaceusage を使用すると指定した単位で情報を出力できる。

sp_estspace を使ってオブジェクト・サイズを見積もる

sp_spaceused と optdiag は、領域の実際の使用状況についてレポートします。sp_estspace を使用すると、テーブルとインデックスの今後の増大に対応する計画を立てることができます。このシステム・プロシージャは、システム・テーブル (sysobjects、syscolumns、sysindexes) 内の情報を使ってデータ・ローとインデックス・ローの長さを決定します。テーブル名とテーブルの予測ロー数を指定すると、sp_estspace は、既存のテーブルとインデックスのサイズを見積もります。テーブル内のデータの実サイズは参照しません。

sp_estspace を使うには、次の作業を実行します。

- テーブルが存在しない場合には、テーブルを作成する。
- テーブルのインデックスを作成する。
- テーブルが保持する予定のロー数を見積もって、このプロシージャを実行する。

sp_estspace の出力は、テーブルとインデックスの各レベルのページ数とバイト数をレポートします。

たとえば、500,000 のロー、1 つのクラスタード・インデックス、2 つのノンクラスタード・インデックスが設定されている titles テーブルのサイズを見積もると、次のように表示されます。

```
    sp_estspace titles, 500000
name                 type         idx_level Pages    Kbytes
-------------------- ------------ --------- -------- --------
titles               data                 0    50002   100004
title_id_cix         clustered            0      302      604
title_id_cix         clustered            1        3        6
title_id_cix         clustered            2        1        2
title_ix             nonclustered         0    13890    27780
title_ix             nonclustered         1      410      819
title_ix             nonclustered         2       13       26
title_ix             nonclustered         3        1        2
type_price_ix        nonclustered         0     6099    12197
type_price_ix        nonclustered         1       88      176
type_price_ix        nonclustered         2        2        5
type_price_ix        nonclustered         3        1        2

Total_Mbytes
-----------------
  138.30

name                 type         total_pages  time_mins
-------------------- ------------ ------------ ------------
title_id_cix         clustered          50308          250
title_ix             nonclustered       14314           91
type_price_ix        nonclustered        6190           55
```

sp_estspace では、フィルファクタ、可変長フィールドとテキスト・フィールドの平均長、および I/O の速度も指定できます。詳細については、『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　sp_estspace によって出力されるインデックス作成時間は、並列ソートの効果は考慮されていません。

sp_estspace のメリット

sp_estspace のメリットは次のとおりです。

- 初期の容量計画の実行や、テーブルとインデックスの増大に備えた計画の作成が効率的にできる。
- インデックス・レベルの数を見積もりやすくなる。
- 今後のディスク領域、キャッシュ領域、メモリの必要量を予測できる。

sp_estspace のデメリット

sp_estspace のデメリットは次のとおりです。

- プロシージャが返すサイズは見積もりでしかなく、フィルファクタ、ページ分割、可変長フィールドの実サイズなどの要因により、実サイズとは異なる場合がある。
- ディスクの速度、エクステント I/O バッファの使用、およびシステムのロードによって、インデックスの作成時間が大幅に異なることがある。

式を使ってオブジェクト・サイズを見積もる

この項で説明する式を使用すると、データベース内にあるテーブルとインデックスの今後のサイズを見積もることができます。可変長フィールドを持つテーブルとインデックスの各ローのオーバヘッドの量は、固定長フィールドだけを持つテーブルのオーバヘッドの量よりも大きいため、2 組の式が必要です。

この処理では、各ローのオーバヘッドを加えたデータのバイト数が計算され、この値を 1 データ・ページあたりで使用できるバイト数に分割します。ページごとに一定のオーバヘッドが必要になるため、データとして使用できるバイト数が制限されます。

- 全ページロック・テーブルの場合、ページのオーバヘッドに 32 バイトが必要なため、2K のページでデータに使用できるのは 2016 バイト。
- データオンリーロック・テーブルの場合、ページのオーバヘッドに 46 バイトが必要なため、データに使用できるのは 2002 バイト。

より正確に見積もるには、ページあたりのロー数を計算した結果は切り下げ (ローは分割されて 2 ページにわたることはありません)、ページ数を計算した結果は切り上げてください。

記憶領域のサイズを変更する要因

領域管理プロパティを使用して、テーブルまたはインデックスに必要な領域を増やすことができます。「領域管理プロパティの効果」(99 ページ) と「max_rows_per_page」(100 ページ) を参照してください。

テーブル内に text または image データ型がある場合、またはロー外にある Java カラムがある場合は、後述の計算式に 16 (ローに格納されているテキスト・ポインタのサイズ) を代入してください。その後、text または image データが実際に必要とする記憶領域を計算する方法については、「LOB ページ」(101 ページ) を参照してください。

データオンリーロック・テーブルのインデックスは、次の 2 つの要因により、式によって算出された値よりも小さくなる可能性があります。

- 重複キーは、一度だけ保存される (そのキーのロー ID のリストが付加される)。
- リーフ・レベル以外のキーの圧縮。隣接するキーとの区別のために必要なキーだけが保存される。特に、キーが長い場合に有効で、キーのサイズを小さくする。

page utilization percent 設定パラメータが 100 未満に設定されている場合は、割り付けられたエクステントのすべてのページが満杯になる前に、Adaptive Server は新しいエクステントを割り付けます。このため、オブジェクトが使用するページ数は変わりませんが、オブジェクトに割り付けられているエクステント内に空のページができます。

各データ型の記憶領域サイズ

表4-2 に、各データ型の記憶領域サイズを示します。

表 *4-2: Adaptive Server* の各データ型の記憶領域サイズ

データ型	サイズ
char	定義サイズ
nchar	定義サイズ * *@@ncharsize*
unichar	n*@@*unicharsize* (*@@unicharsize* は *2* です)
univarchar	実際の文字数 * *@@ncharsize*
varchar	実際の文字数
nvarchar	実際の文字数 * *@@ncharsize*
binary	定義サイズ
varbinary	データ・サイズ
int	4
smallint	2
tinyint	1
float	精度に応じて 4 か 8
double precision	8
real	4
numeric	精度と位取りに応じて 2 ～ 17
decimal	精度と位取りに応じて 2 ～ 17
money	8
smallmoney	4
datetime	8
smalldatetime	4
bit	1
text	16 バイト + 2K * 使用したページ数
image	16 バイト + 2K * 使用したページ数
timestamp	8

numeric または decimal カラムの記憶サイズは、その精度によって異なります。1 桁または 2 桁のカラムの場合、格納領域は 2 バイト必要です。精度が 2 桁増えるごとに、記憶領域サイズは 1 バイトずつ増えます。最大記憶領域サイズは 17 バイトです。

NULL として定義されているカラムは、可変長カラムに関連するオーバヘッドを含むため、可変長カラムとみなされます。

後述の例の計算はすべて、varchar、univarchar、nvarchar、varbinary データの最大サイズ、すなわち各カラムの定義サイズに基づいています。また、これらのカラムは NOT NULL として定義されているものとしています。

式で使用されるテーブルとインデックス

ロー数が 9,000,000 のテーブルについての計算例を示します。

- 固定長カラムのサイズの合計 = 100 バイト
- 2 つの可変長カラムのサイズの合計 = 50 バイト

テーブルには次の 2 つのインデックスがあります。

- 固定長 (4 バイト) カラムのクラスタード・インデックス
- 次のカラムを持つ複合ノンクラスタード・インデックス
  - 固定長 (4 バイト) のカラム
  - 可変長 (20 バイト) のカラム

全ページロック・テーブルとデータオンリーロック・テーブルでは、ページとローあたりのオーバヘッド量が異なるため、異なる式が必要です。

- 全ページ・ロックを使用するテーブルについては、「全ページロック・テーブルのテーブルとクラスタード・インデックスのサイズの計算」(87 ページ) を参照してください。
- データオンリー・ロックを使用するテーブルについては、「データオンリーロック・テーブルのサイズの計算」(94 ページ) を参照してください。

全ページロック・テーブルのテーブルとクラスタード・インデックスのサイズの計算

全ページロック・テーブルの式およびその例を、次の一連の手順に示します。手順 1 ～ 6 では、クラスタード・インデックスのある、全ページロック・テーブルの計算を説明し、テーブル・サイズとインデックス・ツリーのサイズを計算します。手順 7 ～ 12 では、ノンクラスタード・インデックスが必要とする領域の計算について説明します。前述のすべての式では、可変長フィールドの最大サイズが使用されています。手順は次のとおりです。

1 「データ・ロー・サイズを計算する」(88 ページ)

2 「データ・ページの数を計算する」(89 ページ)

3 「クラスタード・インデックス・ローのサイズを計算する」(89 ページ)

4 「クラスタード・インデックス・ページの数を計算する」(90 ページ)

5 「インデックス・ページ数の合計を計算する」(91 ページ)

6 「割り付けのオーバヘッドと合計ページ数を計算する」(91 ページ)

7 「リーフ・インデックス・ローのサイズを計算する」(92 ページ)

8 「インデックス内のリーフ・ページの数を計算する」(93 ページ)

9 「リーフ・ロー以外のローのサイズを計算する」(93 ページ)

10 「リーフ・ページ以外のページ数を計算する」(93 ページ)

11 「リーフ・インデックス・ページ以外のページ数の合計を計算する」(94 ページ)

12 「割り付けのオーバヘッドと合計ページ数を計算する」(94 ページ)

この項で説明する式は、テーブルとクラスタード・インデックスのサイズの計算方法を示します。テーブルにクラスタード・インデックスがない場合は、手順 3、4、5 を省略してください。手順 2 でデータ・ページの数を計算したら、手順 6 に進んで、OAM ページの数を加算してください。

optdiag の出力には、データ・ローとインデックス・ローの平均長が含まれます。平均長を使用する場合は、これらの値をデータ・ローとインデックス・ローの長さとして使用できます。

データ・ロー・サイズを計算する

可変長データを格納するローは、固定長データだけを格納するローよりも多くのオーバヘッドを必要とするため、データ・ローのサイズを計算する式は 2 つあります。

固定長カラムのみのテーブル

固定長カラムだけがテーブルに含まれ、すべてが NOT NULL として定義される場合は、次の式を使います。

式

4	(オーバヘッド)
+	すべての固定長カラムのバイト数の合計
	= データ・ロー・サイズ

可変長カラムを含むテーブル

可変長カラムまたは NULL 値が許されるカラムがテーブルに含まれる場合は、次の式を使います。

ここでは、テーブルに可変長カラムを含む場合の計算例を右側に示します。

式		例
4	(オーバヘッド)	4
+	すべての固定長カラムのバイト数の合計	+ 100
+	すべての可変長カラムのバイト数の合計	+ 50
	= 小計	154
+	(小計 / 256) + 1 (オーバヘッド)	1
+	可変長カラムの数 + 1	3
+ 2	(オーバヘッド)	2
	= データ・ロー・サイズ	160

データ・ページの数を計算する

式

2016 / データ・ロー・サイズ = ページあたりのデータ・ローの数
ローの数 / ページあたりのロー数 = 必要なデータ・ページの数

例

2016 / 160	=	12 (ページあたりのデータ・ローの数)
9,000,000 / 12	=	750,000 (データ・ページ数)

クラスタード・インデックス・ローのサイズを計算する

可変長カラムを含むインデックス・ローは、固定長の値だけを含むインデックス・ローよりも多くのオーバヘッドを必要とします。すべてのキーが固定長である場合は、1 つ目の式を使います。キーに可変長カラムが含まれる場合またはキーが NULL 値を許す場合は、2 つ目の式を使います。

固定長カラムのみのテーブル

この例では、クラスタード・インデックスに固定長キーだけが含まれます。

式		例	
5	(オーバヘッド)		5
+	固定長インデックス・キーのバイト数の合計	+	4
	= クラスタード・ロー・サイズ		9

可変長カラムを含むテーブル

	5	(オーバヘッド)
+		固定長インデックス・キーのバイト数の合計
+		可変長インデックス・キーのバイト数の合計
		= 小計
+		(小計 / 256) + 1 (オーバヘッド)
+		可変長カラムの数 + 1
+	2	(オーバヘッド)
		= クラスタード・インデックス・ロー・サイズ

除算の結果 (小計 / 256) は切り捨てられることに注意してください。

クラスタード・インデックス・ページの数を計算する

式			例		
(2016 / クラスタード・ロー・サイズ) - 2	=	ページあたりのクラスタード・インデックス・ローの数	(2016 / 9) - 2	=	222
データ・ページ数 / ページあたりのクラスタード・インデックス・ローの数	=	次のレベルのインデックス・ページの数	750,000 / 222	=	3379

「次のレベルのインデックス・ページの数」を計算した結果が 1 より大きい場合は、次の除算処理を実行し、計算した商をまた除算して、商が 1 になるまで繰り返します。商が 1 になると、インデックスのルート・レベルに到達したことになります。

式				
最終レベルのインデックス・ページの数	/	ページあたりのクラスタード・インデックス・ローの数	=	次のレベルのインデックス・ページの数

例

3379 / 222	=	16 インデックス・ページ (レベル 1)
16 / 222	=	1 インデックス・ページ (レベル 2)

インデックス・ページ数の合計を計算する

各レベルのページ数を加算して、インデックスのページ数の合計を算出します。

式			例	
インデックス・レベル	ページ数		ページ数	ロー数
2			1	16
1	+		+ 16	3379
0	+		+ 3379	750000
		インデックス・ページ数の合計	3396	

割り付けのオーバヘッドと合計ページ数を計算する

各テーブルとテーブルの各インデックスには、OAM (オブジェクト・アロケーション・マップ) があります。1 OAM ページは、2,000 ～ 63,750 のデータ・ページまたはインデックス・ページのアロケーション・マップを格納します。ほとんどの場合、必要な OAM ページ数は最小値に近くなります。テーブルの OAM ページ数を計算するには、次の式を使います。

式			例		
予約データ・ページ数 / 63,750	=	最小 OAM ページ数	750,000 / 63,750	=	12
予約データ・ページ数 / 2000	=	最大 OAM ページ数	750,000 / 2000	=	376

インデックスの OAM ページ数を計算するには、次の式を使います。

式			例		
予約インデックス・ページ数 / 63,750	=	最小 OAM ページ数	3396/ 63,750	=	1
予約インデックス・ページ数 / 2000	=	最大 OAM ページ数	3396 / 2000	=	2

必要ページ数の合計

最後に、OAM ページ数をこれまで計算した合計に加算して、必要なページ数の合計を算出します。

式			例	
	最小	最大	最小	最大
クラスタード・インデックス・ページ数			3396	3396
OAM ページ	+	+	1	2
データ・ページ	+	+	750000	750000
OAM ページ	+	+	12	376
合計			753409	753773

リーフ・インデックス・ローのサイズを計算する

可変長カラムを含むインデックス・ローは、固定長の値だけを含むインデックス・ローよりも多くのオーバヘッドを必要とします。

固定長キーのみのインデックス

NOT NULL として定義された固定長キーだけがインデックスに含まれる場合は、次の式を使います。

式	
7	(オーバヘッド)
+	固定長キーの長さの合計
	= リーフ・インデックス・ローのサイズ

可変長キーを含むインデックス

インデックスに可変長キー、または NULL として定義されたカラムが含まれる場合は、次の式を使います。

式		例	
9	(オーバヘッド)		9
+	固定長キーの長さの合計	+	4
+	可変長キーの長さの合計	+	20
+	可変長キーの数 + 1	+	2
	= 小計		35
+	(小計 / 256) + 1 (オーバヘッド)	+	1
	= リーフ・インデックス・ローのサイズ		36

インデックス内のリーフ・ページの数を計算する

式			例		
(2016 / リーフ・ローのサイズ)	=	ページあたりのリーフ・インデックス・ローの数	2016 / 36	=	56
テーブル内のローの数 / ページあたりのリーフ・ローの数	=	次のレベルのインデックス・ページの数	9,000,000 / 56	=	160,715

リーフ・ロー以外のローのサイズを計算する

式		例	
	リーフ・インデックス・ローのサイズ		36
+ 4	オーバヘッド	+	4
	= リーフ・ロー以外のローのサイズ		40

リーフ・ページ以外のページ数を計算する

式			例
(2016 / リーフ・ロー以外のローのサイズ) - 2	=	ページあたりのリーフ・インデックス・ロー以外のロー数	(2016 / 40) - 2 = 48

手順 8 のリーフ・ページの数が 1 より大きい場合は、次の除算処理を実行し、計算した商をまた除算して、商が 1 になるまで繰り返します。商が 1 になると、インデックスのルート・レベルに到達したことになります。

式				
前のレベルのインデックス・ページの数	/	ページあたりのリーフ・インデックス・ロー以外のロー数	=	次のレベルのインデックス・ページの数

例	
160715 / 48 = 3349	インデックス・ページ、レベル 1
3349 / 48 = 70	インデックス・ページ、レベル 2
70 / 48 = 2	インデックス・ページ、レベル 3
2 / 48 = 1	インデックス・ページ、レベル 4 (ルート・レベル)

リーフ・インデックス・ページ以外のページ数の合計を計算する

各レベルのページ数を加算して、インデックスのページ数の合計を算出します。

インデックス・レベル		ページ数			ページ数	ロー数
4					1	2
3	+			+	2	70
2	+			+	70	3348
1	+			+	3349	160715
0	+			+	160715	9000000
		使用される 2K データ・ページ数の合計			164137	

割り付けのオーバヘッドと合計ページ数を計算する

式			例		
インデックス・ページ数 / 63,750	=	最小 OAM ページ数	164137 / 63,750	=	3
インデックス・ページ数 / 2000	=	最大 OAM ページ数	164137 / 2000	=	83

必要ページ数の合計

OAM ページ数を手順 11 で計算した合計に加算して、インデックス・ページ数の合計を算出します。

式			例	
	最小	最大	最小	最大
ノンクラスタード・インデックス・ページ数			164137	164137
OAM ページ	+	+	3	83
合計			164140	164220

データオンリーロック・テーブルのサイズの計算

後述の式と例は、テーブルとインデックスのサイズの計算方法を示します。この例では、前の例と同じカラム・サイズとインデックスを使用します。前述のすべての式では、可変長フィールドの最大サイズが使用されています。指定内容については、「式で使用されるテーブルとインデックス」(87 ページ) を参照してください。

データオンリーロック・テーブルの式は、次の 2 組の手順に分かれます。

- 手順 1 ～ 3 では、データオンリーロック・テーブルの計算について説明する。手順 3 に続く例は、ロー数が 9,000,000 のテーブルに対する計算方法を示す。

- 手順 4 ～ 8 は、インデックスが必要とする領域を計算する式について説明する。手順 8 に続く例は、ロー数が 9,000,000 のテーブルに対する計算例を示す。

optdiag の出力には、データ・ローとインデックス・ローの平均長が含まれます。平均長を使用する場合は、これらの値をデータ・ローとインデックス・ローの長さとして使用できます。

手順は次のとおりです

1 「データ・ロー・サイズを計算する」(95 ページ)

2 「データ・ページの数を計算する」(96 ページ)

3 「割り付けのオーバヘッドと合計ページ数を計算する」(96 ページ)

4 「インデックス・ローのサイズを計算する」(97 ページ)

5 「インデックス内のリーフ・ページの数を計算する」(97 ページ)

6 「インデックス内のリーフ・ページ以外のページ数を計算する」(98 ページ)

7 「リーフ・インデックス・ページ以外のページ数の合計を計算する」(98 ページ)

8 「割り付けのオーバヘッドと合計ページ数を計算する」(99 ページ)

データ・ロー・サイズを計算する

可変長データを格納するローは、固定長データだけを格納するローよりも多くのオーバヘッドを必要とするため、データ・ローのサイズを計算する式は 2 つあります。

固定長カラムのみのテーブル

NOT NULL として定義された固定長カラムだけがテーブルに含まれる場合は、次の式を使います。

```
        6    ( オーバヘッド )
+            すべての固定長カラムのバイト数の合計
-------------
             データ・ロー・サイズ
```

注意 データオンリーロック・テーブルの各ローには、転送される 6 バイトのロー ID を保存するための領域が必要です。データオンリーロック・テーブルに 10 バイトよりも短いローが含まれる場合、各ローは挿入時に 10 バイトになるように桁埋めされます。これはデータ・ページにのみ反映され、インデックスには影響しません。また、全ページロック・テーブルにも影響しません。

可変長カラムを含むテーブル

可変長カラムまたは NULL 値が許されるカラムがテーブルに含まれる場合は、次の式を使います。

式		例	
8	(オーバヘッド)		8
+	すべての固定長カラムのバイト数の合計	+	100
+	すべての可変長カラムのバイト数の合計	+	50
+	可変長カラムの数 * 2	+	4
	データ・ロー・サイズ		162

データ・ページの数を計算する

式

2002 / データ・ロー・サイズ = ページあたりのデータ・ローの数

ローの数 / ページあたりのロー数 = 必要なデータ・ページの数

この手順の最初の計算では、ページあたりのロー数は切り捨てます。

例

2002 / 162	=	12 (ページあたりのデータ・ローの数)
9,000,000 / 12	=	750,000 (データ・ページ数)

割り付けのオーバヘッドと合計ページ数を計算する

アロケーション・オーバヘッド

各テーブルとテーブルの各インデックスには、OAM (オブジェクト・アロケーション・マップ) があります。OAM は、テーブルまたはインデックスに割り付けられているページに格納されます。1 OAM ページは、2,000 ～ 63,750 のデータ・ページまたはインデックス・ページのアロケーション・マップを格納します。ほとんどの場合、必要な OAM ページ数は最小値に近くなります。テーブルの OAM ページ数を計算するには、次の式を使います。

式			例		
予約データ・ページ数 / 63,750	=	最小 OAM ページ数	750,000 / 63,750	=	12
予約データ・ページ数 / 2000	=	最大 OAM ページ数	750,000 / 2000	=	375

必要ページ数の合計

OAM ページ数をこれまで計算した合計に加算して、必要なページ数の合計を算出します。

式			例	
	最小	最大	最小	最大
データ・ページ	+	+	750000	750000
OAM ページ	+	+	12	375
合計			750012	750375

インデックス・ローのサイズを計算する

データオンリーロック・テーブルのクラスタード・インデックスとノンクラスタード・インデックスに対しては、次の式を使います。

可変長カラムを含むインデックス・ローは、固定長の値だけを含むインデックス・ローよりも多くのオーバヘッドを必要とします。

固定長キーのみのインデックス

NOT NULL として定義された固定長キーだけがインデックスに含まれる場合は、次の式を使います。

9	(オーバヘッド)
+	固定長キーの長さの合計
	インデックス・ローのサイズ

可変長キーを含むインデックス

可変長キーまたは NULL 値が許されるカラムがインデックスに含まれる場合は、次の式を使います。

式		例	
9	(オーバヘッド)		9
+	固定長キーの長さの合計	+	4
+	可変長キーの長さの合計	+	20
+	可変長キーの数 * 2	+	2
	インデックス・ローのサイズ		35

インデックス内のリーフ・ページの数を計算する

式

2002 / インデックス・ローのサイズ = ページあたりのローの数

テーブル内のローの数 / ページあたりのローの数 = リーフ・ページの数

例

2002 / 35 = 57 (ページあたりのノンクラスタード・インデックス・ローの数)

9,000,000 / 57 = 157,895 (リーフ・ページの数)

インデックス内のリーフ・ページ以外のページ数を計算する

式

リーフ・ページの数	/	ページあたりのインデックス・ローの数	=	次のレベルのページの数

次のレベルのインデックス・ページの数が 1 より大きい場合は、次の除算処理を実行し、計算した商をまた除算して、商が 1 になるまで繰り返します。商が 1 になると、インデックスのルート・レベルに到達したことになります。

式

前のレベルのインデックス・ページの数	/	ページあたりのリーフ・インデックス・ロー以外のロー数	=	次のレベルのインデックス・ページの数

例

157895/57 = 2771	インデックス・ページ、レベル 1
2770 / 57 = 49	インデックス・ページ、レベル 2
48 / 57 =1	インデックス・ページ、レベル 3

リーフ・インデックス・ページ以外のページ数の合計を計算する

各レベルのページ数を加算して、インデックスのページ数の合計を算出します。

式			例	
インデックス・レベル	ページ数		ページ数	ロー数
3			1	49
2	+		49	2771
1	+		2771	157895
0	+		157895	9000000
		使用される 2K ページ数の合計	160716	

割り付けのオーバヘッドと合計ページ数を計算する

式

インデックス・ページ数 / 63,750 = 最小 OAM ページ数
インデックス・ページ数 / 2,000 = 最大 OAM ページ数

例

160713 / 63,750 = 3 (最小)
160713 / 2,000 = 81 (最大)

必要ページ数の合計

OAM ページ数を手順 8 で計算した合計に加算して、インデックス・ページ数の合計を算出します。

式			例	
	最小	最大	最小	最大
ノンクラスタード・インデックス・ページ数			160716	160716
OAM ページ	+	+	3	81
合計			160719	160797

オブジェクト・サイズに影響するそのほかの要因

ある期間にわたって発生するデータ更新が及ぼす影響だけでなく、次の要因もオブジェクト・サイズとサイズの見積もりに影響します。

- 記憶領域管理プロパティ
- 平均ロー・サイズと最大ロー・サイズのどちらが計算に使用されるか
- 非常にサイズの小さいテキスト・ロー
- text データと image データの使用

領域管理プロパティの効果

fillfactor、exp_row_size、reservepagegap、max_rows_per_page の値は、オブジェクト・サイズに影響を与えます。

fillfactor

create index に指定する fillfactor は、インデックスの作成時に適用されます。fillfactor は、テーブルへの挿入時には維持されません。sp_chgattribute を使用して fillfactor をインデックス用に保存した場合、この値は、alter table...lock コマンドと reorg rebuild によってインデックスが再作成されるときに使用されます。fillfactor の主な機能は、インデックス・ページ上に領域を確保し、ページ分割を減らすことです。fillfactor の値を非常に小さくすると、テーブルまたはインデックスに必要な記憶領域が非常に大きくなることがあります。

fillfactor の値の設定の詳細については、「インデックス管理作業量を減らす」(53 ページ) を参照してください。

exp_row_size

テーブルの予測ロー・サイズを設定すると、必要な記憶領域が大きくなる場合があります。予測ロー・サイズよりも小さいサイズのローがテーブルに多く含まれる場合に、この値を設定して reorg rebuild を実行するか、またはロック・スキームを変更すると、テーブルに必要な記憶領域が大きくなります。ただし、それまで max_rows_per_page を使用していたテーブルの領域の使用量は、あまり変わりません。

exp_row_size の値の設定の詳細については、「ローの転送の削減」(60 ページ) を参照してください。

reservepagegap

テーブルまたはインデックスに reservepagegap を設定すると、エクステントの割り付けを行うコマンドが実行されたとき、そのオブジェクトに割り付けられたエクステント上に空のページが残されます。reservepagegap を小さい値に設定すると、空のページ数が増え、データが多くのエクステントに分散するため、create index や reorg rebuild コマンドを実行した直後に、必要な追加領域が最大になります。テーブルに転送または挿入されるローは、予約されたページ内に埋められます。

「転送されるローと挿入用に領域を残す」(66 ページ) を参照してください。

max_rows_per_page

max_rows_per_page の値 (create index、create table、alter table、sp_chgattribute で指定) は、データ・ページあたりのロー数を制限します。

max_rows_per_page を使用する場合に正しい値を計算するには、max_rows_per_page の値か、「データ・ページの数を計算する」(89 ページ) と 「インデックス内のリーフ・ページの数を計算する」(93 ページ) にある 1 つめの式で計算されるページあたりのロー数の、どちらか小さい方の値を使います。

「全ページロック・テーブルでの max_rows_per_page の使用」(74 ページ) を参照してください。

非常にサイズの小さいロー

全ページロック・テーブルの場合、Adaptive Server では 1 ページにつき 256 のデータまたはインデックス・ローを超えて格納することはできません。ローのサイズが非常に小さい場合でも、データ・ページの最小数は次の式で計算します。

ロー数 / 256 =　必要データ・ページ数

LOB ページ

text カラム、image カラム、またはロー外にある Java カラムはそれぞれ、データ・ローにデータ型が varbinary(16) の 16 バイト長のポインタを 1 つ格納します。初期化された各カラムは、最低 2K (1 データ・ページ) の記憶領域を必要とします。

カラムは、暗黙的な NULL 値を格納します。つまり、データ・ロー内のテキスト・ポインタが NULL のままであり、この値に対して初期化されたテキスト・ページがないことを意味し、2K の記憶領域を節約します。

LOB カラムに NULL 値を許す定義がされていて、insert 文を使用してカラムに NULL を組み込んでローを作成した場合、カラムは初期化されず、記憶領域は割り付けられません。

update を使って LOB カラムが変更されると、テキスト・ページが割り付けられます。カラム内に実データを配置する挿入または更新によって、このページは初期化されます。この後にカラムが NULL に設定されると、1 ページが割り付けられたままになります。

各 LOB ページには、約 1800 バイトのデータが保存されます。特定のエントリが使用するページ数を見積もるには、次の式を使います。

データ長 / 1,800 = 2K ページ の数

計算結果は常に切り上げなければなりません。たとえば、データ長が 1,801 バイトの場合は 2K ページが 2 ページ必要です。

実際に必要なデータの合計領域は、計算された値よりもやや大きい場合があります。これは、一部の LOB ページがほかのページ・チェーンのポインタ情報をカラム内に保存するためです。Adaptive Server はこのポインタ情報を使用し、LOB カラムに対してランダム・アクセスを実行し、データをプリフェッチします。ポインタ情報を保存するために必要な追加の領域は、カラム内に保存されるデータの合計サイズと型によって異なります。表4-3 を参照して、データのポインタ情報を LOB カラムに保存するために必要な追加ページ数を見積もってください。

表 *4-3:* ポインタ情報用の *LOB* カラムの追加ページ数の見積もり

データ・サイズとデータ型	ポインタ情報用に必要な追加ページ数
400K の image	0 ～ 1 ページ
700K の image	0 ～ 2 ページ
5MB の image	1 ～ 11 ページ
400K のマルチバイト text	1 ～ 2 ページ
700K のマルチバイト text	1 ～ 3 ページ
5MB のマルチバイト text	2 ～ 22 ページ

オブジェクト・サイズを見積もるために式を使うことのメリット

式を使うことのメリットは、次のとおりです。

- データとインデックスの記憶領域の内部について、より詳しい知識を身に付けられる。
- 式を使うと、文字カラムまたはバイナリ・カラムの平均サイズを柔軟に指定できる。
- インデックス・サイズの計算を通して、各インデックスにいくつのレベルがあるかを確認できるため、パフォーマンスの予測に役立つ。

オブジェクト・サイズを見積もるために式を使うことのデメリット

式を使うことのデメリットは、次のとおりです。

- 見積もりの正確さは、可変長カラムの平均サイズの見積もりと同じ程度でしかない。
- 複数の手順から計算が複雑に構成されているため、手順をとばすと間違いが発生する。
- 使用状況によって、オブジェクトの実サイズと計算した値が異なることがある。

第 5 章

データベースのメンテナンス

この章では、メンテナンス作業が Adaptive Server のほかのアクティビティのパフォーマンスに及ぼす影響と、メンテナンス作業のパフォーマンスを向上させる方法の両方について説明します。

メンテナンス作業には、インデックスの削除と再作成、dbcc チェックの実行、テーブル統計とインデックス統計の更新などの作業が含まれます。これらの作業はすべて、サーバ上のほかの作業処理と競合することがあります。

できるだけ Adaptive Server の使用率が低いときにメンテナンス作業を実行することをおすすめします。この章では、これらのメンテナンス作業が、個々のアプリケーションのパフォーマンスと全体的な Adaptive Server パフォーマンスに対してどのように影響するかについて説明します。

テーブルおよびインデックスでの *reorg* の実行

reorg コマンドを使用してテーブルとインデックスの領域の使用率を改善することによって、データオンリーロック・テーブルのパフォーマンスを向上できます。reorg サブコマンドとその機能は次のとおりです。

- reclaim_space － コミットされた削除やデータ・ローの長さが短くなるような更新でできたスペースをクリアする。
- forwarded_rows － 転送ローをホーム・ページに戻す。
- compact － 上記の両方の処理を実行する。

- rebuild − テーブルまたはインデックス全体を再構築する。全ページロック・テーブルとデータオンリーロック・テーブルの両方に対して reorg rebuild を実行できる。

あるテーブルで reorg rebuild を実行する場合、このテーブルはテーブルとそのインデックスの再構築中ロックされます。ユーザがそのテーブルにアクセスする必要がない場合、テーブル上に reorg rebuild コマンドをスケジュールします。

reorg のそのほかのコマンドは、インデックス上で reorg rebuild を実行する場合も含めて、少数のページを一度にロックし、短期の独立したトランザクションを使用して作業を実行します。これらのコマンドはいつでも実行できます。ただし、I/O バウンドが非常に多いシステムには、望ましくない影響が及ぶ場合があります。

reorg コマンドの実行の詳細については、『システム管理ガイド 第 2 巻』の「第 9 章 reorg コマンドの使用方法」を参照してください。

インデックスの作成とメンテナンス

インデックスを作成するとき、ほかのすべてのユーザはテーブルにアクセスできなくなります。ロックの種類はインデックスの種類によって異なります。

- クラスタード・インデックスを作成するには排他テーブル・ロックが必要であるため、すべてのテーブル・アクティビティができなくなる。クラスタード・インデックス内のローはインデックス・キーの順番に並べられるため、create clustered index はデータ・ページを並べ替える。
- ノンクラスタード・インデックスを作成するには共有テーブル・ロックが必要であるため、更新アクティビティができなくなる。

ソートを高速にする Adaptive Server の設定

入力テーブルからのページを保持するためにキャッシュ内で使用できるバッファ数を設定するには、number of sort buffers 設定パラメータを使用します。さらに、並列ソートでは、ソートを実行するためにキャッシュ内の大容量 I/O を利用します。

『パフォーマンス&チューニング・シリーズ:クエリ処理と抽象プラン』の「第 5 章並列クエリ処理」を参照してください。

インデックス作成後にデータベースをダンプする

インデックス作成時に、Adaptive Server は create index トランザクションとページ割り付けをトランザクション・ログに書き込みますが、データ・ページとインデックス・ページへの実際の変更内容のログは取りません。インデックスを作成した時点以降ダンプしていないデータベースをリカバリするには、トランザクション・ログ・ダンプがロードされると同時に create index プロセス全体が再実行されます。

定期的にインデックスを再作成する (たとえばインデックスの fillfactor をメンテナンスするため) 場合は、前述のオペレーションを定期的なデータベース・ダンプの直前にスケジューリングすることをおすすめします。

ソートされたデータのインデックス作成

クラスタード・インデックスを再作成したり、インデックス・キーの順に従ってサーバにバルク・コピーされたデータにインデックスを作成するには、create index に sorted_data オプションを指定して、インデックスの作成時間を短くします。

データ・ローは、クラスタード・インデックスのキー順に並べられなければならないため、sorted_data オプションを指定しないでクラスタード・インデックスを作成するには、データ・ローをデータ・ページの新しい完全なセットに再度書き込む必要があります。Adaptive Server はテーブルのデータ・ローのソートおよびコピーを省略する場合があります。たとえば、テーブルが分割されているかどうかと、on 句が create index 文で使用されている場合です。

非分割テーブルのインデックスを作成する場合、sorted_data と次の句のいずれかを使用すると、データをコピーする必要がありますが、ソートする必要はありません。

- ignore_dup_row
- fillfactor
- テーブル・データが配置されているセグメントとは別のセグメントを指定する on *segment_name* 句。
- テーブルに対応する値とは別の値を指定する max_rows_per_page 句。

上記のオプションと sorted_data が、分割されたテーブルに対する create index に含まれる場合は、ソート手順が実行され、データがコピーされ、テーブルの各分割にデータ・ページが均一に分配されます。

表 *5-1:* クラスタード・インデックスの作成時のオプションの使用

オプション	分割されたテーブル	分割されていないテーブル
どのオプションも指定しない場合	並列ソートは、データをコピーし、分割に均等に分配し、インデックス・ツリーを作成する。	並列ソートまたは非並列ソートがデータをコピーし、インデックス・ツリーを作成する。
with sorted_data だけ、 または with sorted_data on *same_segment*	インデックス・ツリーのみ作成する。データのソートまたはコピーは実行しない。また、並列実行は行わない。	インデックス・ツリーのみ作成する。データのソートまたはコピーは実行しない。また、並列実行は行わない。
with sorted_data と ignore_dup_row または fillfactor または on *other_segment* または max_rows_per_page	並列ソートは、データをコピーし、分割に均等に分配し、インデックス・ツリーを作成する。	データをコピーし、インデックス・ツリーを作成する。ソートは実行しない。また、並列実行は行わない。

最も単純なのは、非分割テーブルで sorted_data を使用してほかのオプションは使用しない場合です。この場合、テーブル・ローの順序はチェックされ、インデックス・ツリーは、この 1 回のテーブル・スキャンの間に構築されます。

データ・ローはコピーされなければなりませんが、ソートを実行する必要がない場合は、1 回のテーブル・スキャンで、ローの順序がチェックされ、インデックス・ツリーが構築され、データ・ページがこの 1 回のテーブル・スキャン中に新しいロケーションにコピーされます。

インデックスを構築するために、非常に多くのパスを必要とするサイズの大きいテーブルでは、ソート時間を節約することによって、I/O と CPU の使用率が大幅に低減されます。

クラスタード・インデックスを作成する場合は、常にデータ・ローがコピーされます。データをコピーしてインデックス・ページを格納するため、使用可能な領域がテーブル・サイズの約 120% 存在している必要があります。

インデックス統計値とカラム統計値のメンテナンス

インデックスのヒストグラムと密度値は、データ・ローが追加または削除されるときにメンテナンスされません。データベース所有者は、update statistics コマンドを実行して、統計値を最新の値にしてください。次の場合に、update statistics を実行します。

- ローの削除または挿入によって、インデックスのキー値の分布状態が変更された場合。
- truncate table を使ってすでにローが削除されているテーブルに、ローを追加した場合。
- インデックス・カラムの値を変更した場合。

- IDENTITY カラムまたは増加するキー値を含むインデックスに挿入を行った場合。日付カラムは、定期的に増加するキーを持っていることが多くある。

これらの種類のインデックスに update statistics を実行することは、IDENTITY カラムまたはその他の増加キーがインデックスの先行カラムである場合に、特に重要です。インデックスが作成されたときにテーブルの最終キーを越えて多数のローが挿入されると、オプティマイザが認識できるのは、検索値がディストリビューション・ページの最終ローより下にあるということだけです。与えられた値に一致するローの数を調べることはできません。

注意　update statistics の実行に失敗すると、パフォーマンスが著しく低下します。

『パフォーマンス&チューニング・シリーズ：統計的分析によるパフォーマンスの向上』を参照してください。

インデックスの再構築

インデックスを再構築すると、バイナリ・ツリー (すべてのリーフ・ページがインデックスのルート・ページから等距離に配置されたツリー) 内の領域を再使用できます。ページが分割され、ローが削除されると、インデックスにはわずかのローしかないページがたくさんできてしまいます。また、アプリケーションが、カバーリング・ノンクラスタード・インデックスと大容量 I/O でスキャンを実行する場合は、ノンクラスタード・インデックスを再構築すると、断片化が減るので大容量 I/O の効力を維持できます。

インデックスの再構築は、インデックスを削除し、再作成することによってできます。

次の条件の場合にインデックスを再構築します。

- データや使用状況パターンが著しく変更された場合。
- 負荷の高い挿入が、ある期間にわたって続くと予想される場合や、それが完了した直後。
- ソート順が変更された場合。
- 大容量 I/O を使うクエリが、予想よりも多くのディスク読み込みを必要とする場合。または optdiag によって通常よりも低いクラスタ率が報告された場合。
- データ修正が頻繁に行われる多くのデータ・ページとインデックス・ページに空きができたために、領域の使用が見積もりを上回った場合。

- 領域管理プロパティ (フィルファクタ、予測ロー・サイズ、ページ・ギャップの予約値) によって割り付けられた拡張用の空き領域が挿入や削除によっていっぱいになったために、ページ分割、転送されたロー、断片化が起こった場合。
- dbcc がインデックス内でエラーを検出した場合。

クラスタード・インデックスを再作成した場合や、データオンリーロック・テーブルまたは全ページ・ロック・テーブルで reorg rebuild を実行した場合、すべてのノンクラスタード・インデックスは再作成されます。これは、クラスタード・インデックスを再作成すると、ローがさまざまなページに移動するからです。

システム・アクティビティが少ない場合：

- すべてのインデックスを削除して、より効率的なバルク挿入を実行できる。
- インデックスの新しいグループを作成して、一連のレポートを生成できる。

データベースの作成または変更

データベースを作成または変更する作業は I/O を大量に使用し、I/O を大量に使用するほかのオペレーションのパフォーマンスを低下させます。データベース作成時には、Adaptive Server は model データベースを新しいデータベースにコピーしてから、すべてのアロケーション・ページを初期化し、データベース・ページをクリアします。

データベース作成の速度を高め、ほかのプロセスに対する影響を最小限に抑えるには、次の手順を実行します。

- データベースをリストアする場合 (load database コマンドを発行する準備が整った場合) は、create database...for load オプションを指定する。

  for load を指定せずにデータベースが作成されると、データベースは model をコピーし、次にすべてのアロケーション・ユニットを初期化します。

  for load が指定されると、ロードが完了するまでアロケーション・ユニットが初期化されます。ロード完了後に、データベースは、ゼロにされていないアロケーション・ユニットだけを初期化します。非常にサイズの大きいデータベース・ダンプをロードする場合には、この方法で多くの時間を節約できます。
- できる限り、オフピーク時にデータベースを作成する。

create database と alter database は、データ・ページのクリア時に同時並列 I/O を実行します。デバイス数は、number of large i/o buffers 設定パラメータによって制限されます。このパラメータのデフォルト値は 6 で、一度に 6 つのデバイスを使う並列 I/O が可能になります。

単一の create database コマンドと alter database コマンドでは、これらのバッファを一度に最大で 32 個使用できます。これらのバッファは、load database、ディスク・ミラーリング、および一部の dbcc コマンドによっても使用されます。

デフォルト値 6 を使用し、6 つ以上のデバイスを指定した場合は、始めの 6 回の書き込みはただちに開始されます。各デバイスの I/O が終わるたびに、コマンドにリストされたデバイスに 16K バッファが使用されます。次の例では、10 個の別々のデバイスに割り当てています。

```
create database hugedb
      on dev1 = 100,
      dev2 = 100,
      dev3 = 100,
      dev4 = 100,
      dev5 = 100,
      dev6 = 100,
      dev7 = 100,
      dev8 = 100
log on logdev1 = 100,
      logdev2 = 100
```

これらのバッファを使用するオペレーションの実行中は、バッファ数を超えたときに、メッセージがログに送られます。上記の create database コマンドの情報は、create database が、大容量 I/O バッファをすべて使用して、最初の 6 つのディスクのデバイスのクリアを開始し、これらが完了してから、ほかのデバイス上のページをクリアすることを示しています。

```
CREATE DATABASE: allocating 51200 pages on disk 'dev1'
CREATE DATABASE: allocating 51200 pages on disk 'dev2'
CREATE DATABASE: allocating 51200 pages on disk 'dev3'
CREATE DATABASE: allocating 51200 pages on disk 'dev4'
CREATE DATABASE: allocating 51200 pages on disk 'dev5'
CREATE DATABASE: allocating 51200 pages on disk 'dev6'
01:00000:00013:1999/07/26 15:36:17.54 server  No disk i/o buffers are
available for this operation. The total number of buffers is controlled by
the configuration parameter 'number of large i/o buffers'.
CREATE DATABASE: allocating 51200 pages on disk 'dev7'
CREATE DATABASE: allocating 51200 pages on disk 'dev8'
CREATE DATABASE: allocating 51200  pages on disk 'logdev1'
CREATE DATABASE: allocating 51200  pages on disk 'logdev2'
```

注意 Adaptive Server バージョン 12.5.0.3 以降では、create database、alter database、load database、dbcc checkalloc で使用される大容量 I/O バッファのサイズは、以前のバージョンと同様に、1 エクステント (8 ページ) ではなく、1 アロケーション (256 ページ) となります。このため、大容量バッファに対してこれまでより大きなメモリ割り付けが必要です。たとえば、以前のバージョンでディスク・バッファとして 8 ページのメモリが必要だった場合に、今回のバージョンでは 256 ページのメモリが必要となります。

バックアップとリカバリ

Adaptive Server におけるすべてのバックアップ作業は Backup Server が実行します。バックアップ・アーキテクチャは、Adaptive Server を Backup Server のクライアントとするクライアント・サーバ・アーキテクチャを使用します。

ローカル・バックアップ

Adaptive Server は、ローカル Backup Server 命令を、リモート・プロシージャ・コールを使って送信し、ダンプまたはロード対象のページ、使用するバックアップ・デバイス、そのほかのオプションを Backup Server に伝えます。Backup Server は、すべてのディスク I/O を実行します。

Adaptive Server は、ダンプの読み込み、ダンプの送信、データのロードを実行せず、命令だけを送信します。

リモート・バックアップ

Backup Server は、リモート・マシンに対するバックアップもサポートしています。リモート・ダンプとリモート・ロードについては、ローカル Backup Server が、データベース・デバイスに関連するディスク I/O を実行し、ネットワークを介してデータをリモート・バックアップ・サーバに送信します。リモート Backup Server サーバはダンプ・デバイス上でデータを格納します。

オンライン・バックアップ

バックアップは、データベースがアクティブである間も実行できます。こうした処理はほかのトランザクションに影響しますが、システムの信頼性を維持するために必要な頻度で重要なデータベースをバックアップしてください。

バックアップ方式とリカバリ方式の詳細については、『システム管理ガイド 第 2 巻』を参照してください。

スレッショルドを使ってログ領域の不足を防ぐ

データベースのログ領域が少なく、「ラストチャンス・スレッショルド」に達することもある場合には、トランザクション・ログ・ダンプを実行できるだけの時間を確保する別のスレッショルドをインストールします。ログ領域が足りなくなると、パフォーマンスに対して重大な影響があります。たとえば、ログ領域が解放されないと、データ変更コマンドを実行できなくなります。

リカバリ時間を最小限に抑える

recovery interval 設定パラメータを変更することにより、リカバリ時間を最小限に抑えることができます。デフォルト値はデータベースあたり 5 分であり、ほとんどのインストール環境に適しています。機能上の稼働条件が、より短いリカバリ時間を要求する場合だけ、recovery interval の値を減らします。値を減らすと、必要な I/O の量が増えます。

『パフォーマンス&チューニング・シリーズ:基本』の「第 5 章メモリの使い方とパフォーマンス」を参照してください。

リカバリ時間は、housekeeper free write percent 設定パラメータを使っても制御できます。housekeeper free write percent 設定パラメータのデフォルト値を使用すると、サーバのハウスキーピング・ウォッシュ・タスクはディスク I/O の増加率が 20 パーセント以下のアイドル・サイクル中に、ダーティ・バッファをディスクへ書き込みます。

リカバリ順序

リカバリ時には、システム・データベースが最初にリカバリされます。次に、ユーザ・データベースがデータベース ID 順にリカバリされます。

バルク・コピー

Adaptive Server 上のテーブルへのバルク・コピーは、テーブル上にクラスタード・インデックスが存在せず、select into/ bulkcopy が有効な場合、最も速く実行されます。このオプションが有効になっていない場合、**低速** bcp は、インデックスもアクティブなトリガもないテーブルに使用されます。

高速 bcp を指定すると、インデックスのないテーブルについてのみ、ページ割り付けのログを取ります。**高速** bcp を使用すると、データ挿入ごとにインデックスが更新されず、またインデックス・ページの変更内容のログも取らないため、時間の節約になります。ただし、インデックスを持つテーブルで **高速** bcp を使用すると、インデックスの更新のログが記録されます。

高速 bcp は、トリガのあるテーブルでは自動的に使用されます。**低速** bcp を使用するには、コピーの実行中に select into/bulk copy データベース・オプションを無効にします。

高速バルク・コピーを使う方法は、次のとおりです。

1 sp_dboption を使用して、select into/bulkcopy/pllsort オプションを設定します。バルク・コピー・オペレーション完了後は、select into/bulkcopy/pllsort オプションを無効にします。

2 クラスタード・インデックスをすべて削除します。クラスタード・インデックスは、バルク・コピーの完了時に再作成します。

注意 トリガは、コピー中に非アクティブ化する必要はありません。

高速バルク・コピーの処理中には、ルールが実行されず、デフォルトが実行されます。

データへの変更内容のログが取られないため、高速バルク・コピー・オペレーションを実行したあとはすぐに dump database を実行します。ログが取られていないデータ変更内容はトランザクション・ログ・ダンプからリカバリできないため、データベース内で高速バルク・コピーを実行すると dump transaction が使用できなくなります。

並列バルク・コピー

最高のパフォーマンスを得るため、高速バルク・コピーを使用して、データを分割されたテーブルにコピーします。各バルク・コピー・セッションについて、データを格納すべきパーティションを指定します。

入力ファイルがすでにソート順に入っている場合は、その順序に従ってパーティションに対してのデータのバルク・コピーを実行し、クラスタード・インデックス作成中のソート手順を回避できます。

詳細な手順については、『Transact-SQL ユーザーズ・ガイド』の「第 10 章テーブルとインデックスの分割」を参照してください。

バッチとバルク・コピー

高速バルク・コピー時にバッチ・サイズが指定されると、高速バルク・コピー時にはデータ変更内容ではなくページ割り付けについてだけログが取られるため、新しい各バッチは新しいデータ・ページから始まります。バッチ・サイズ 1 が指定されているときに 1,000 ローをコピーするには、トランザクション・ログ内に 1,000 データ・ページと 1,000 割り付けレコードが必要です。

小さいバッチ・サイズを使って入力ファイル内のエラーを検出しやすくする場合は、1 データ・ページに収まるロー数に対応するバッチ・サイズを選択できます。

低速バルク・コピー

Adaptive Server は、select into/bulk copy が有効でテーブルにクラスタード・インデックス、インデックス、またはトリガがある場合、デフォルトで **低速** bcp を使用します。

低速バルク・コピーは、次のように動作します。

- select into/bulkcopy の設定も必要ありません。
- ルールとトリガは起動されないが、デフォルトが実行される。
- ページ割り付けだけでなく、すべてのデータ変更内容のログが取られる。
- ローがテーブル内にコピーされるときにインデックスが更新され、インデックス変更内容のログが取られる。

バルク・コピーのパフォーマンスを高める

バルク・コピーのパフォーマンスを高めるには、次の方法も使用できます。

- trunc log on chkpt オプションを設定して、トランザクション・ログが満杯になるのを防ぐ。データベース内に、ログが満杯になるとログを自動的にダンプするスレッショルド・プロシージャがある場合は、トランザクション・ダンプ時間を節約できる。

  各バッチは別々のトランザクションになることに注意する。これにより、バッチ・サイズを指定していない場合は trunc log on chkpt を設定してもパフォーマンスは改善しない。

- 大量のバルク・コピーを実行する場合、number of pre-allocated extents 設定パラメータを高い値に設定する。

  『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。

- 最適なネットワーク・パケット・サイズを探す。

  『パフォーマンス&チューニング・シリーズ：基本』の「第 2 章ネットワークとパフォーマンス」を参照してください。

サイズの大きいテーブル内でデータを置換する

サイズの大きいテーブル内ですべてのデータを置換する場合は、delete ではなく、truncate table を使います。この場合、少量のログしか取られません。つまり、ページ割り付けの解除についてだけログが取られます。

1 テーブルをトランケートします。

2 テーブルのすべてのインデックスを削除します。

3 データをロードします。

4 インデックスを再構築します。

『リファレンス・マニュアル：コマンド』を参照してください。

大量のデータをテーブルに追加する

サイズの大きいテーブルに、テーブル・サイズの 10% または 20% を超えるローを追加するときは、ノンクラスタード・インデックスを削除し、データをロードしてからノンクラスタード・インデックスを再作成します。

サイズが非常に大きいテーブルの場合は領域に制約があるため、クラスタード・インデックスを現在のロケーションに設定したままにする必要があります。Adaptive Server は、クラスタード・インデックスを作成するときにテーブルのコピーを作成する必要があります。多くの場合、テーブルのサイズが非常に大きくなると、インデックスを現在のロケーションに設定したまま低速バルク・コピーを実行するのに要する時間は、高速バルク・コピーを実行し、クラスタード・インデックスを再作成するのに要する時間より短くなります。

分割と複数のバルク・コピー・プロセスを使う

インデックスがないテーブルにデータをロードする場合は、テーブルにパーティションを作成し、パーティションごとに 1 つの bcp セッションを使用できます。

『ユーティリティ・ガイド』の「第 4 章 bcp を使用した Adaptive Server とのデータの転送」を参照してください。

ほかのユーザに対する影響

バルク・コピーを使用して、サイズの大きいテーブルをコピー・インまたはコピー・アウトすると、ほかのユーザの応答時間が長くなります。できる限り、次のガイドラインに従います。

- バルク・コピー・オペレーションはオフピーク時にスケジューリングする。
- ログと I/O を減らすため、高速バルク・コピーを使用する。

データベース一貫性チェッカ

データベース一貫性チェッカは、dbcc を使用して、定期的に実行します。壊れたデータベースをバックアップしても、バックアップには役に立たないからです。ただし、dbcc はチェック対象オブジェクトにロックを設定するため、dbcc を使用するとパフォーマンスを低下させます。

dbcc とロックの詳細、およびユーザ・アプリケーションに対する dbcc の影響を最小限に抑える方法については、『システム管理ガイド 第 2 巻』の「第 10 章 データベースの一貫性の検査」を参照してください。

dbcc tune (cleanup) の使用

Adaptive Server は、各タスクを処理したあと、最後の整合性チェックとして冗長メモリ・クリーンアップを実行します。スループットが非常に高い環境では、このクリーンアップ・エラー・チェックを省略することによって、わずかにパフォーマンスの向上を実現できます。エラー・チェックをオフにするには、次のように入力します。

```
dbcc tune(cleanup,1)
```

最終のクリーンアップによって、タスクが保持していたメモリがいくつか解放されます。エラー・チェックをオフにすると、メモリ・エラーが発生する場合は、次のように入力して、チェックを再び有効にします。

```
dbcc tune(cleanup,0)
```

スピンロック時の *dbcc tune* の使用

スピンロック競合によるスケーリングの問題が発生している場合には、des_bind を使用して競合が発生しているオブジェクトのオブジェクト記述子が予約されているサーバのスケーラビリティを改善できます。バインドされたオブジェクトの記述子は解放されません。したがって、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減らし、パフォーマンスを改善できます。

```
dbcc tune(des_bind, <dbid>, <objname>)
```

バインドを削除するには、次のコマンドを使用します。

```
dbcc tune(des_unbind, <dbid>, <objname>)
```

注意 データベースからオブジェクトのバインドを解除するには、データベースをシングル・ユーザ・モードにする必要があります。

以下の場合、des_bind を使用しないでください。

- master や tempdb などのシステム・データベース内のオブジェクトの場合
- システム・テーブルの場合

des_bind は永続的でないため、サーバを再起動するたびにすべてのバインド処理コマンドを再発行する必要があります。

メンテナンス作業に使用可能な領域の確認

次のようなメンテナンス作業では、テーブルのデータ・ページのコピーを作成するための空き領域が必要です。

- create clustered index
- alter table...lock
- カラムの追加や修正を行う、いくつかの alter table コマンド
- alter table...partition by
- テーブル上で実行する reorg rebuild

ほとんどの場合、上記のコマンドはインデックスを再作成する領域も必要とするため、次のことを確認する必要があります。

- テーブルとそのインデックスのサイズ。
- テーブルが格納されるセグメント上で使用可能な領域の量。
- テーブルとそのインデックスに設定する領域管理プロパティ。

領域の要件の概要

テーブルのローをコピーするすべてのコマンドは、そのテーブルのすべてのインデックスも再作成します。テーブル全体のコピーとすべてのインデックスのコピー用の領域が必要です。

これらのコマンドは必要な領域の量を見積もりません。コマンドの実行によって、テーブルまたはそのインデックスが使用するセグメントで領域が不足すると、コマンドは停止し、エラー・メッセージが発行されますサイズの大きいテーブルでは、コマンドが開始してから数分後または数時間後にこのエラーが発生する場合もあります。

テーブルやそのインデックスが使用するセグメント上に、次の条件を満たす空き領域が必要です。

- テーブルのセグメントの空き領域は、次のサイズ以上にする。
  - そのテーブルのサイズ。

    - テーブルにクラスタード・インデックスがあり、全ページ・ロックからデータ・オンリーロックに変更している場合、上記のテーブルのサイズにその約 20% を足したサイズ。
- ノンクラスタード・インデックスが使用するセグメントの空き領域は、インデックスのサイズ以上にする。

データオンリー・ロック・テーブルのクラスタード・インデックスには、データ・ページの上にリーフ・レベルのページがあります。クラスタード・インデックスのあるテーブルを、全ページ・ロックからデータオンリー・ロックに変更する場合、結果としてできたクラスタード・インデックスにはもっと領域が必要となります。必要な領域は、インデックス・キーのサイズによって異なります。

領域の使用率と使用可能な領域のチェック

簡単なガイドラインとして、テーブルとそのインデックスのコピーには、テーブルとそのインデックスが使用している現在の領域のサイズと、その約 20% を足したサイズの領域が必要です。ただし、次の点に注意してください。

- データの修正によって部分的に満杯のページが多く作成されている場合、そのテーブルのコピーのための領域要件は現在のサイズより小さい場合がある。
- テーブルの領域管理プロパティに変更があったか、fillfactor または reservepagegap が必要とする領域がデータ修正で満杯になっている場合、そのテーブルのコピーに必要なサイズはもっと大きくなる場合がある。
- カラムを追加したり、より大きいデータ型へカラムを修正したりすると、コピーにもっと大きな領域が必要となる。

ログ領域も必要です。Adaptive Server では reorg rebuild を単一のトランザクションとして処理するため、特に再構築されるテーブルに複数のノンクラスタード・インデックスが含まれる場合、必要とされるログ領域が大きくなる可能性があります。ノンクラスタード・インデックスではそれぞれログ領域が必要になるため、すべてのインデックスを作成するために十分なログ領域が必要になります。

テーブルとインデックスが使用している領域の確認

テーブルとそのインデックスのサイズを調べるには、次のコマンドを使用します。

```
sp_spaceused titles, 1
```

クラスタード・インデックスのサイズを見積もる詳細については 「データオンリーロック・テーブルのサイズの計算」(94 ページ) を参照してください。

セグメント上の領域の確認

テーブルは常に、現在そのテーブルを格納しているセグメント上の空き領域にコピーされ、インデックスは、現在そのインデックスを格納しているセグメント上に再作成されます。クラスタード・インデックスを作成するコマンドはセグメントを指定できます。テーブルのコピーとクラスタード・インデックスは、ターゲット・セグメント上に作成されます、

1 つのセグメントで使用できるページ数を確認するには、sp_helpsegment を使用します。sp_helpsegment の結果の最後の行が、1 つのセグメント上で使用できる空きページの合計数を示します。

次のコマンドは、default セグメントのセグメント情報を出力します。default セグメントには、セグメントが明示的に指定されなかった場合にオブジェクトが格納されます。

```
sp_helpsegment "default"
```

sp_helpsegment は、セグメント上のインデックス名をレポートします。テーブルのセグメント名が分からない場合は、sp_help をそのテーブル名を指定して使用してください。sp_help はインデックスのセグメント名もレポートします。

領域管理プロパティの領域要件の確認

領域管理プロパティの値に重大な変更を行う場合、テーブル・コピーは元のテーブルより大幅に大きくなったり小さくなったりする場合があります。領域管理プロパティの設定は sysindexes テーブルに保管され、sp_help、sp_helpindex を使って表示されます。次の出力は、titles テーブルの領域管理プロパティを示します。

```
exp_row_size reservepagegap fillfactor max_rows_per_page
------------ -------------- ---------- -----------------
         190             16         90                 0
```

次は、sp_helpindex が生成するレポートです。

```
index_name           index_description
    index_keys
    index_max_rows_per_page index_fillfactor index_reservepagegap
    ----------------------- ---------------- --------------------
title_id_ix          nonclustered located on default
    title_id
                         0               75                    0
title_ix             nonclustered located on default
    title
                         0               80                   16
type_price           nonclustered located on default
    type, price
                         0               90                    0
```

テーブルの領域管理プロパティ

コピー手順の間、そのテーブルの領域管理プロパティが次のように使用されます。

- テーブルに予測ロー・サイズが指定され、ロック・スキームが全ページ・ロックからデータオンリー・ロックに変更される場合、予測ロー・サイズは、データ・ローがコピーされるときにデータ・ローに適用される。

  予測ロー・サイズが設定されていなくても、テーブルに max_rows_per_page 値が設定されている場合、予測ロー・サイズが計算され、その値が使用される。

  それ以外の場合、設定パラメータ default exp_row_size percent で指定されたデフォルト値が、そのテーブルに割り付けられた各ページに対して使用される。

- reservepagegap は、そのテーブルにエクステントが割り付けられたときに適用される。

- sp_chgattribute はテーブルの fillfactor 値を保存するのに使用された場合、ローがコピーされるとき新しいデータ・ページに適用される。

インデックスの領域管理プロパティ

インデックスが再構築される場合、そのインデックスの領域管理プロパティが次のように適用されます。

- sp_chgattribute がインデックスの fillfactor 値を保存するのに使用された場合、この値がインデックスの再作成時に適用される。

- reservepagegap 値がインデックスに設定されると、この値がインデックスの再作成時に適用される。

領域管理プロパティの影響の見積もり

表 5-2 は、領域管理プロパティを設定したときの影響の見積もり方法を示します。

表 5-2: 領域の使用に対する領域管理プロパティの影響

プロパティ	式	例
fillfactor	現在ページが満杯の場合、(100/fillfactor) * *num_pages* 必要。	fillfactor が 75 の場合、現在のページ数の 1.33 倍必要。1,000 ページのテーブルは 1,333 ページに増加する。
reservepagegap	現在エクステントが満杯の場合、1/reservepagegap 領域を増加。	reservepagegap が 10 の場合、使用領域は 10% 増加。1,000 ページのテーブルは 1,100 ページに増加する。
max_rows_per_page	データオンリー・ロックへの変換時に、exp_row_size に変換される。	表 5-3 (120 ページ) を参照。
exp_row_size	exp_row_size より小さいローの数と、それらのローの平均長に応じて増加。	exp_row_size が 100 で、1,000 個のローが 60 の長さの場合、領域の増加は次のとおり。 (100 - 60) * 1000 つまり 40,000 バイト、約 20 ページの追加。

詳細については、「第 3 章 記憶領域管理プロパティの設定」を参照してください。

テーブルに max_rows_per_page が設定されている場合、そのテーブルを全ページ・ロックからデータオンリー・ロックに変換すると、値が exp_row_size 値に変換されてから alter table...lock コマンドによってテーブルがその新しいロケーションにコピーされます。

exp_row_size は、コピー中に適用されます。表5-3 は、値の変換方法を示します。

表 5-3: *max_rows_per_page* から *exp_row_size* への変換

max_rows_per_page の値	*exp_row_size* が設定される値
0	default exp_row_size percent で設定された割合値
1 ～ 254	次のいずれか小さい方 • 最大ロー・サイズ • 2K の論理ページ － 2002/max_rows_per_page 値 4K の論理ページ － 4050/max_rows_per_page 値 8K の論理ページ － 8146/max_rows_per_page 値 16K の論理ページ － 16338/max_rows_per_page 値

十分な領域がない場合

テーブルをコピーしてすべてのインデックスを再作成するのに十分な領域がない場合、そのテーブルのノンクラスタード・インデックスを削除するとテーブルのコピーの作成に十分な空きができるかどうかを確認します。ノンクラスタード・インデックスがない場合、コピー・オペレーションが必要とするのはそのテーブルとクラスタード・インデックス用の領域だけです。

クラスタード・インデックスは削除しないでください。クラスタード・インデックスは、コピーされたローの順序付けに使用されます。また、後でクラスタード・インデックスを再作成しようとすると、そのテーブルのコピーを作成するための領域が必要になる場合があります。コマンドの完了時、ノンクラスタード・インデックスを再作成します。

第 6 章

テンポラリ・データベース

この章では、テンポラリ・データベースに関連したパフォーマンスの問題について説明します。テンポラリ・データベースはサーバワイドなリソースです。主に、ソート処理、ワーク・テーブル作成、再フォーマット、および、ユーザが作成するテンポラリ・テーブルとインデックスの保管などの目的に使用されます。テンポラリ・データベースにはどのユーザでもオブジェクトを作成できます。テンポラリ・データベースは多くのプロセスによって暗黙的に使われます。

多くのアプリケーションが、テンポラリ・データベース内にテーブルを作成するストアド・プロシージャを使って、複雑なジョインを効率的に作成したり、1 つのステップでは実行しにくい、ジョイン以外の複雑なデータ分析を実行します。

テンポラリ・データベースの管理がパフォーマンスに及ぼす影響

テンポラリ・データベースを適切に管理することは、Adaptive Server の全体的なパフォーマンスにとって非常に大切です。しかし、テンポラリ・テーブルを使用するには、tempdb のサイズを大きくしなければなりません。テンポラリ・テーブルが適切に使われているかどうかは、Adaptive Server とアプリケーションのパフォーマンスに大きく影響します。テンポラリ・データベースの管理を見過ごしたり、テンポラリ・データベースをデフォルトのまま放置することもできません。多くのサーバ上で、tempdb は最も動的なデータベースです。

テンポラリ・データベースに関するパフォーマンス問題を考慮に入れておき、予め計画を立てておくことで問題の大部分を回避できます。

- テンポラリ・データベースが頻繁に満杯になるため、エラー・メッセージがユーザに対して発行される。エラー・メッセージを受け取ったユーザは、領域が使用可能になったときに、必要なクエリをもう一度実行しなければならない。
- テンポラリ・データベースでは低速でソートが行われ、クエリを行うとパフォーマンスの低下が示される。
- システム・テーブルにロックが設定されるため、しばしばユーザのクエリが一時的にテンポラリ・テーブルを作成できなくなる。
- テンポラリ・データベースのオブジェクトの使用頻度の高いため、ほかのページがデータ・キャッシュからフラッシュされる。

これらの問題は、次のように解決します。

- 十分な数のユーザ・テンポラリ・データベースを設定する。
- すべての Adaptive Server アクティビティに対して適切なサイズになるようにテンポラリ・データベースを設定する。
- テンポラリ・データベースの配置を最適化して、競合を最小限に抑える。
- テンポラリ・データベース内で設定するリソースのロックを最小限に抑える。
- テンポラリ・データベースを独自のデータ・キャッシュにバインドする。
- テンポラリ・データベースのグループを適切に設定する。
- 該当するテンポラリ・データベースまたはグループに、ログインまたはアプリケーションをバインドする。

テンポラリ・テーブルの使用

テンポラリ・データベース内に作成されるテーブルは、テンポラリ・テーブルと呼ばれます。さまざまな種類のテンポラリ・テーブルは、テンポラリ・データベースを使用して作成します。テンポラリ・テーブルの種類は次のとおりです。

- ハッシュ (#) テンポラリ・テーブル
- 正規のユーザ・テーブル
- ワーク・テーブル

ハッシュ (#) テンポラリ・テーブル

ハッシュ (#) テンポラリ・テーブルには以下の特性があります：

- ユーザ・セッションの間のみ、またはテンポラリ・テーブルを作成するプロシージャのスコープに対してのみ存在し、セッションまたはプロシージャの終わりに自動的に削除される (または手動で削除できる)。
- 複数のユーザ接続間では共有できない。
- セッションに割り当てられるテンポラリ・データベース内に作成される。

次のように、テーブル名の最初の文字にシャープ記号 (“#”) を使うと、ハッシュ・テンポラリ・テーブルを作成できます。

```
create table #temptable (...)
```

または

```
select select_list
into #temptable ...
```

テンポラリ・テーブルに対してインデックスを作成すると、そのインデックスは、ハッシュ・テーブルが存在するテンポラリ・データベースに割り当てられた同一のセッションに保存されます。

```
create index littletableix on #littletable(col1)
```

正規のユーザ・テーブル

テンポラリ・テーブルで正規のユーザ・テーブルを作成するには、create table コマンドでデータベース名を指定します。

```
create table tempdb..temptable (...)
```

テンポラリ・データベースの正規のユーザ・テーブルには以下の特性があります：

- 複数のセッションにわたって存在できる。
- バルク・コピー (bcp) オペレーションによって使用することができる。
- パーミッションを付与すると、共有できる。
- 所有者が意識的に削除する必要がある。そうしないと、Adaptive Server の再起動時に自動的に削除される。

または

```
select select_list
into tempdb..temptable
```

次のように、テンポラリ・データベースに作成した正規のユーザ・テーブルにインデックスを作成できます。

```
create index tempix on tempdb..temptable(col1)
```

ワーク・テーブル

Adaptive Server は、マージ、ソート、ジョインなどのために、セッションが割り当てられた tempdb に内部テンポラリ・テーブルを作成します。作成されるテンポラリ・テーブルはワーク・テーブルと呼ばれ、次のような特徴があります。

- 共有することはできない。
- コマンド完了時に消失する。

テンポラリ・データベース

テンポラリ・データベースを 1 つしか使用しないことが原因でパフォーマンスが低下するのを避けるために、複数のテンポラリ・データベースを作成できます。

Adaptive Server にはシステムが作成する tempdb と呼ばれるテンポラリ・データベースが 1 つあり、これは、Adaptive Server のインストール時にマスタ・デバイス上に作成されます。

tempdb に加え、Adaptive Server ではユーザが複数のテンポラリ・データベースを作成できます。ユーザが作成するテンポラリ・データベースは tempdb システムと似ており、主にテンポラリ・オブジェクトを作成するのに使用されます。起動時には、リカバリされずに再度作成されます。tempdb とは異なり、ユーザ作成のテンポラリ・データベースは削除できます。

複数のテンポラリ・データベース:

- システム・カタログでの競合と、システム tempdb のログ・ファイルを減らす。
- 高速アクセスデバイス上に作成できる。
- 必要に応じて作成または削除できる。

セッションを割り当てられたテンポラリ・データベース

クライアントが接続されると、Adaptive Server はテンポラリ・データベースをそのセッションに割り当てます。Adaptive Server はこのテンポラリ・データベースをデフォルト領域として使用し、ここで、クライアントが実行する作業用のテンポラリ・オブジェクト (ハッシュ・テンポラリ・テーブル、ワーク・テーブルなど) を作成します。セッションが割り当てられたテンポラリ・データベースでは、セッションがクライアントに接続されるまでセッションへの割り当てが維持されます。

Adaptive Server は、次のルールに従ってセッションのテンポラリ・データベースを選択します。

- ログインにバインドがすでに存在する場合、そのバインドが使用される。
- アプリケーション名が指定されており、バインドがある場合は、そのバインドが使用される。
- Adaptive Server がバインドを見つけられない場合は、ラウンドロビン・スキームでデフォルトのグループからテンポラリ・データベースを割り当てる。

特定のテンポラリ・データベースで Adaptive Server がオブジェクトを作成するように指定します。次に例を示します。

```
create procedure inv_amounts as
     select stor_id, "Total Due" = sum(amount)
     from #tempstores
     group by stor_id
```

複数のテンポラリ・データベースの使用

この項では、テンポラリ・データベースの作成、設定、バインド、および選択を行うする方法について説明します。

ユーザ・テンポラリ・データベースの作成

次のように、create database 構文で temporary database キーワードを使用して複数のテンポラリ・データベースを作成します。

create temporary database *temporary_database_name* on *device_name*=size log on *device_name*=size

たとえば、tempdb_device 上に tempdb_1 という名前のユーザ・テンポラリ・データベースを作成するには、次のように入力します。

```
create temporary database tempdb_1 on tempdb_device = 3
log on log_device = 1
```

デフォルトの tempdb グループの設定

Adaptive Server には、デフォルト・グループと呼ばれるテンポラリ・データベースのグループがあります。Adaptive Server がセッションを開始すると、すべてのテンポラリ・データベースのアクティビティが実行されるデフォルト・グループから (ラウンドロビン法を使用して) テンポラリ・データベースを選択します。Adaptive Server はこのテンポラリ・データベースをセッションに割り当てます。sp_who は、このテンポラリ・データベースを tempdbname カラムに表示します。ラウンドロビン・スキームを使用すると、グループの単一のテンポラリ・データベースがすべてのアクティビティを実行することがなくなるため、Adaptive Server がデフォルト・グループのすべてのテンポラリ・データベースで負荷を均等に分散させることができます。

デフォルトのグループは、最初は tempdb のみで構成されています。ただし、ユーザが複数のユーザ・データベースをデフォルトのグループに追加することができます。デフォルトのグループにユーザ・データベースを追加するには、sp_tempdb を使用します。たとえば、デフォルトのグループに tempdb_1 を追加するには、次のコマンドを使用します。

```
sp_temodb "add'", "tempdb_1" , "default"
```

デフォルトのグループから tempdb_1 を削除するには、次のコマンドを使用します。

```
sp_tempdb "drop", "tempdb_1" , "default"
```

sp_tempdb 構文の詳細については、『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。

グループおよび tempdb へのバインド

sp_tempdb. . . 'bind'...'unbind' システム・プロシージャを使用すると、アプリケーションまたはログインを指定のテンポラリ・データベースや tempdb グループにバインドしたりバインドを解除したりすることができます。バインドの作成後、アプリケーションまたはログインをサーバに接続すると、Adaptive Server は指定されたテンポラリ・データベースまたはテンポラリ・データベース・グループをバインド先に割り当てます。バインドを行うと、特定のアプリケーションまたはログインのテンポラリ・データベースへの割り当てを管理できます。

次の例では、sa のログをデフォルトのグループにバインドします。

```
sp_tempdb 'bind', 'lg', 'sa', 'GR', 'default'
```

次の例では、ログイン sa のバインドを解除します。

```
sp_tempdb 'unbind', 'lg', 'sa'
```

sp_tempdb 構文の詳細については、『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。

テンポラリ・データベースへのアプリケーションとログインのバインド

テンポラリ・データベースのアプリケーションとログインの要件を識別します。これらのアプリケーションまたはログインを異なるデータベースやデフォルトのグループにバインドし、使用可能なテンポラリ・データベース全体で負荷を均等に分散してカタログの競合を回避します。バインドが不適切であると、十分な数のテンポラリ・データベースがあってもカタログの競合は解決せず、Adaptive Server がテンポラリ・データベースで負荷を均等に分散することができません。「グループおよび tempdb へのバインド」(128 ページ) を参照してください。

パフォーマンスに適したシステム・テンポラリ・データベースの調整

この項では、テンポラリ・データベースに関連した設定の問題について説明します。

システム *tempdb* の配置

tempdb の配置場所を決定するには、次の項目を実行します。

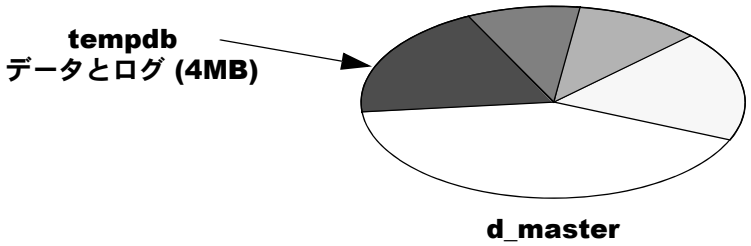
- tempdb は、重要なアプリケーション・データベースの配置先とは別の物理ディスクに保存する。
- 使用可能なディスクの中で最速のディスクを使用する。プラットフォームが固体回路デバイスをサポートしており、アプリケーションにとって tempdb を使うことがボトルネックである場合は、固体回路デバイスを使用する。
- tempdb を他のデバイス上で拡張したら、マスタ・デバイスを system セグメント、default セグメント、logsegment セグメントから削除する。

master データベースと同じデバイス上で tempdb を拡張できても、別のデバイスを使うことをおすすめします。また、Adaptive Server のミラーリングを使用してミラーリングされるのは、データベースではなく論理デバイスであることに注意します。マスタ・デバイスをミラーリングする場合は、マスタ・デバイス内に常駐するデータベースのすべての部分のミラーが作成されます。ミラーに「**逐次**」書き込みが使用されている場合、tempdb データベースの使用頻度が高いと、パフォーマンスが著しく低下します。

システム *tempdb* の初期割り付け

Adaptive Server のインストール時には tempdb のサイズは 4MB であり、図 6-1 に示すように、tempdb 全体がマスタ・デバイス上に置かれます。一般的には、システム管理者が最初に大きくする必要を感じるのが tempdb データベースです。サーバ上のユーザが増えるにつれ、大きくする必要性が高くなります。変更の必要性によっては、tempdb を複数のデバイス間に分散保存する方法があります。

図 *6-1: tempdb* のデフォルトの割り当て



sp_helpdb を使って tempdb のサイズとステータスを表示します。次に、インストール時の tempdb のデフォルト設定を表示する例を示します。

```
                    sp_helpdb tempdb
name        db_size  owner   dbid     created                 status
---------- -------- ------ ------ -----------  --------------------
tempdb       2.0 MB      sa      2 May 22, 1999    select into/bulkcopy

device_frag      size        usage     free kbytes
------------ -------- ------------        --------
master          2.0 MB   data and log             1248
```

tempdb セグメントからのマスタ・デバイスの削除

デフォルトでは、tempdb の system セグメント、default セグメント、logsegment セグメントは、マスタ・デバイス上の tempdb の 4MB の割り付けを含みます。tempdb に新しいデバイスを割り当てる場合、その新しいデバイスを専用のデータまたはログとして追加しないと自動的に 3 つすべてのセグメントの一部になります。次のデバイスが tempdb に割り付けられると、default セグメント、system セグメント、および logsegment セグメントからマスタ・デバイスを削除できます。このようにして、tempdb 内のワーク・テーブルとほかのテンポラリ・テーブルは、マスタ・デバイス上で使用しているほかのテーブルとは競合しないことが確認できます。

マスタ・デバイスをセグメントから削除するには、次の手順に従います。

1 tempdb を別のデバイスに移動し終えていない場合は、移動します。次に例を示します。

```
alter database tempdb on tune3 = 20
```

2　use tempdb コマンドを発行してから、セグメントからマスタ・デバイスを削除します。

```
sp_dropsegment "default", tempdb, master
sp_dropsegment "system", tempdb, master
sp_dropsegment "logsegment", tempdb, master
```

3　今後セグメントがマスタ・デバイスを持たないことを確認するには、マスタ・デバイスに対して次のコマンドを発行します。

```
select dbid, name, segmap
from sysusages, sysdevices
where sysdevices.vdevno= sysusages.vdevno
and dbid = 2
and (status&2=2 or status&3=3))
```

segmap カラムは、次のようにマスタ・デバイス上にある割り付けには“0”をレポートし、セグメントの割り付けが存在しないことを示します。

```
 dbid   name            segmap
 ------ --------------- -----------
      2          master           0
      2           tune3           7
```

または、次のコマンドを発行します。

```
use tempdb
sp_helpdb 'tempdb'
device_fragments     size      usage             created   free kbytes
-----------------   ------   ----------  -----------------  ----------
master              4.0 MB   data only     Feb 7 2008 2:18AM      2376
tune3               20.0 MB  data and log May 16 2008 1:55PM     16212

device    segment
--------- ------------------------------
master    -- unused by any segments --
tune3                        default
tune3                     logsegment
tune3                          system
```

ユーザ作成のテンポラリ・データベースの設定

アプリケーションには、テンポラリ・データベースに対するリソースと領域の個別の要件があります。アプリケーションの要件を把握し、これらのデータベースの要件を満たすアプリケーションとデータベースまたはグループとのバインドを維持しない場合、すべてのテンポラリ・データベースを同じサイズにしてください。すべてのテンポラリ・データベースのサイズが同じになると、アプリケーションまたはセクションがどのデータベースに割り当てられるのかに関係なく、アプリケーションでリソースまたは領域が不足することはないはずです。

ユーザ・テンポラリ・データベースのキャッシュ

通常、キャッシュはグループ内のテンポラリ・データベース全体で同じように設定されます。クエリ・プロセッサはこれらのキャッシュ特性に基づいてクエリ・プランを選択します。設定が異なるキャッシュを使用してプランを実行すると、パフォーマンスが低下する可能性があります。

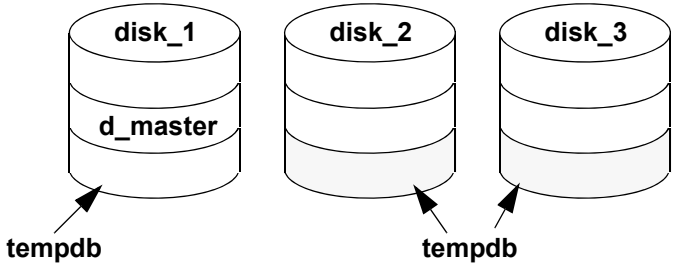
一般的なガイドライン

この項では、システム・データベースとユーザ・テンポラリ・データベースの両方に適用される、テンポラリ・データベースを設定するための一般的なガイドラインについて説明します。

複数のディスクの場合の並列クエリのパフォーマンスの使用

図 6-2 に示すように、テンポラリ・データベースが複数のデバイスにまたがっている場合は、一部のテンポラリ・テーブルまたはワーク・テーブルに、並列クエリのパフォーマンスを利用できます。

図 *6-2:* 複数のディスクにまたがる *tempdb*



tempdb の専用キャッシュへのバインド

Adaptive Server の通常の使用状況では、テンポラリ・テーブルが作成され、データが挿入されて、削除されると、テンポラリ・データベースはデータ・キャッシュを頻繁に使用します。

テンポラリ・データベースをテンポラリ・データベース独自のデータ・キャッシュにバインドする場合は、次のことに留意してください。

- テンポラリ・オブジェクトに対するアクティビティが、デフォルト・データ・キャッシュからほかのオブジェクトをフラッシュしないようにする。
- 複数のキャッシュ間で I/O を分散しやすくする。

キャッシュ・バインド用コマンド

sp_cacheconfig コマンドと sp_poolconfig コマンドを使って、名前付きデータ・キャッシュを作成し、大容量 I/O に対応する指定サイズのプールを設定します。キャッシュとプールを設定できるのはシステム管理者だけです。

注意 大容量 I/O は、論理ページ・サイズが 2K のサーバに基づきます。ページ・サイズが 8K のサーバでは、I/O の基本単位は 8K になります。ページ・サイズが 16K のサーバでは、I/O の基本単位は 16K になります。

名前付きキャッシュおよびプールの設定方法については、『システム管理ガイド 第 2 巻』の「第 4 章データ・キャッシュの設定」を参照してください。

キャッシュが設定され、サーバが再起動されたら、次のように指定して tempdb を新しいキャッシュにバインドできます。

```
sp_bindcache "tempdb_cache", tempdb
```

テンポラリ・データベースのサイズの決定

すべての Adaptive Server ユーザが次のプロセスを同時に処理するように、テンポラリ・データベースに十分な領域を割り当てます。

- マージ・ジョイン用のワーク・テーブル
- distinct、group by、order by の実行、再フォーマット、or 方式、およびビューとサブクエリの実体化のために作成されるワーク・テーブル
- ハッシュ・テンポラリ・テーブル (テーブル名の最初の文字に “#” が指定されているテーブル)
- テンポラリ・テーブルに設定されるインデックス
- テンポラリ・データベースの正規のユーザ・テーブル
- 動的 SQL が構築するプロシージャ

アプリケーションによっては、テンポラリ・テーブルを使って複数のテーブルのジョインを分割する方がパフォーマンスが向上する場合があります。こうした分割方式は、次の場合に使われます。

- 5 つ以上のテーブルをジョインするクエリのとき、適切なクエリ・プランをオプティマイザが選択しない場合
- 非常に大量のテーブルをジョインするクエリ
- 非常に複雑なクエリ
- 中間ステップとしてデータをフィルタする必要があるアプリケーション

次の目的にもテンポラリ・データベースが使われます。

- いくつかのテーブルを非正規化していくつかのテンポラリ・テーブルにする。
- 集合処理を実行するために、非正規化されたテーブルを正規化する。

使用シナリオに基づいてテンポラリ・データベースのサイズを決定します。ほとんどのアプリケーションでは、テンポラリ・データベースをユーザ・データベースの 20 ～ 25% のサイズで作成することで十分な領域を確保できます。

テンポラリ・データベースでのロギングの最小化

trunc log on checkpoint データベース・オプションがテンポラリ・データベース内でオンになっていても、テンポラリ・データベースに対する変更はトランザクション・ログに書き込まれます。次を実行することによって、テンポラリ・データベースでのログ・アクティビティを抑制できます。

- create table と insert の代わりに select into を使う。
- テンポラリ・テーブルに保管する必要のあるテーブルだけを選択する。

select into の使用

テンポラリ・データベース内でテンポラリ・テーブルを作成して移植するときは、できる限り、create table や insert...select ではなく select into コマンドを使います。select into/bulkcopy データベース・オプションはテンポラリ・データベース内ではデフォルトでオンであるため、select into を使えます。

select into オペレーションは、最小限のログしか取られないため、create table や insert...select よりも速く動作します。記録されるのはデータ・ページの割り付けだけであり、各データ・ローの実際の変更内容ではありません。insert...select クエリでは各データ挿入が完全にログされるため、オーバヘッドが多くなります。

短いローの使用

テンポラリ・データベース内にテーブルを作成するアプリケーションがテーブルのごくわずかなカラムしか使わない場合は、次の方法でログ・レコードの数とサイズを最小限に抑えることができます。

- テーブルにデータを挿入する際に、select * を使用するのではなく、アプリケーションに必要なカラムだけを選択する。
- 選択するローを、アプリケーションが必要とするローだけに限定する。

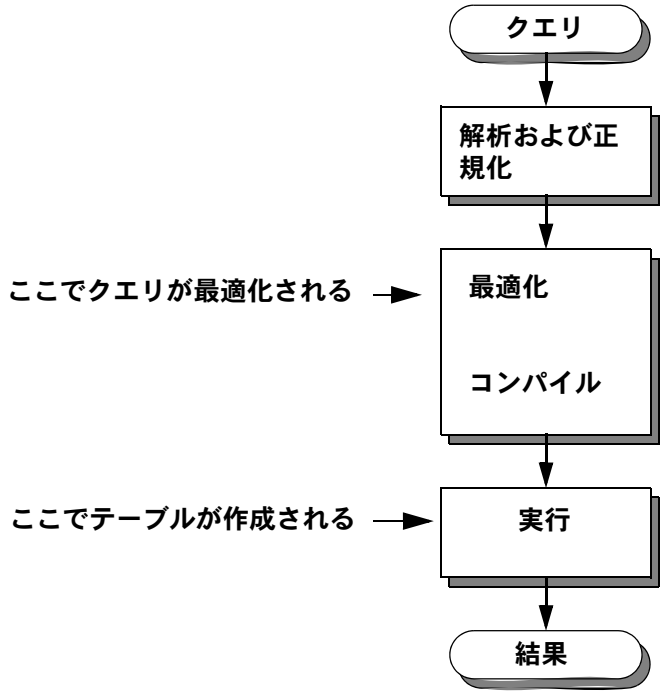
どちらの方法も、テーブルのサイズをより小さく抑えることができます。

テンポラリ・テーブルの最適化

多くの場合、テンポラリ・テーブルはごく短い間だけ単純な仕組みで使われるため、最適化の必要はほとんどありません。ただし、テンポラリ・データベース内のテーブルに複数回アクセスする必要のあるアプリケーションでは、最適化の方式を検討する必要があります。最適化には通常、テーブルの作成やインデックスの作成と、テーブルへのアクセスをプロシージャやバッチを 1 つではなく複数にすることで分離することが必要となります。

図 6-3 に示すように、テーブルが使われるのと同一のストアド・プロシージャまたはバッチでテーブルが作成される場合は、クエリ最適化の時点でテーブルが作成されていないため、クエリ・オプティマイザはテーブルのサイズを判断できません。このことは、テンポラリ・テーブルと正規のユーザ・テーブルにもあてはまります。

図 *6-3:* テンポラリ・テーブルの最適化と作成



この場合オプティマイザは、テーブルのデータ・ページ数を 10、ロー数を 100 であるとみなします。テーブルのサイズが実際に大きい場合は、この前提に基づいてオプティマイザは最良のクエリ・プランの次に良いと思われるクエリ・プランを選択します。

テンポラリ・テーブルの最適化を向上させるには、次の 2 つの方法があります。

- テンポラリ・テーブルにインデックスを作成する
- テンポラリ・テーブルを複雑に使用する場合は、複数のバッチまたはプロシージャに分割して、オプティマイザがテーブルについての情報を収集できるようにする

テンポラリ・テーブルにインデックスを作成する。

テンポラリ・テーブルにインデックスを定義できます。多くの場合、テンポラリ・テーブルにインデックスを定義すると、テンポラリ・データベースを使うクエリのパフォーマンスが向上します。オプティマイザはこのインデックスを通常のユーザ・テーブルのインデックスと同じように使います。テンポラリ・テーブルのインデックスの動作条件は次のとおりです。

- テーブルには、インデックスが作成される時点でデータが存在していなければならない。テンポラリ・テーブルが作成され、空のテーブルにインデックスが設定された場合、Adaptive Server はヒストグラムや密度といった、カラム統計値を作成しない。インデックスを作成した後でデータ・ローが挿入されても、オプティマイザには不完全な統計値しか存在しない。
- インデックスは、インデックスを使うクエリが最適化される間存在していなければならない。インデックスを作成してから、同一のバッチまたはプロシージャ内のクエリにこのインデックスを使用することはできない。クエリ・プロセッサは、ストアド・プロシージャの内部で実行されるクエリで、ストアド・プロシージャに作成されたインデックスを使用する。
- インデックスを作成したり、update statistics を実行した後にローを追加したり、削除した場合、オプティマイザには、最適なプランを選択できない可能性がある。

テンポラリ・テーブルを作成してから、テンポラリ・テーブルに対して大量のオペレーションを実行する複雑なプロシージャでは特に、オプティマイザにインデックスを与えることによってパフォーマンスが著しく向上します。

テンポラリ・テーブルを使ってネスト・プロシージャを作成する

前述のプロシージャを作成するには、実行する手順を増やす必要があります。select_proc が存在しない限り base_proc を作成できず、テンポラリ・テーブルが存在しない限り select_proc を作成できません。

1 プロシージャの外にテンポラリ・テーブルを作成します。テンポラリ・テーブルは空でもかまいません。テンポラリ・テーブルは、ただ存在し、select_proc と互換性のあるカラムを持っている必要があります。次のように指定します。

```
select * into #huge_result from ... where 1 = 2
```

2 前述のように select_proc プロシージャを作成します。

3　#huge_result を削除します。

4　base_proc プロシージャを作成します。

tempdb を使用する処理を複数のプロシージャに分割する

たとえば、次のクエリは #huge_result に関して最適化問題を発生させます。

```
create proc base_proc
as
    select *
        into #huge_result
        from ...
    select *
        from tab,
        #huge_result where ...
```

2 つのプロシージャを使うとパフォーマンスが向上します。base_proc プロシージャが select_proc プロシージャを呼び出すときに、オプティマイザはテーブルのサイズを判断できます。次に例を示します。

```
create proc select_proc
as
    select *
        from tab, #huge_result where ...
create proc base_proc
as
    select *
        into #huge_result
        from ...
    exec select_proc
```

#huge_result の処理に、複数のアクセス、ジョイン、または while が指定されたループなどの処理が必要とされる場合は、#huge_result にインデックスを作成するとパフォーマンスが向上します。base_proc 内でインデックスを作成して、select_proc が最適化されるときにインデックスを使用可能にします。

テンポラリ・データベースのロギングの最適化

テンポラリ・データベースを停止したり、テンポラリ・データベースでエラーが発生したりした場合 Adaptive Server はテンポラリ・データベースをリカバリしませんが、サーバの再起動時にテンポラリ・データベースを作成します。テンポラリ・データベースはリカバリの必要がないため、Adaptive Server はテンポラリ・データベースのロギング・メカニズムを最適化し、次のようにしてパフォーマンスを向上させます。

- 単一のログ・レコード − Adaptive Server がレコードをログに記録した後、ただちに syslogs をディスクにフラッシュするようにする。Adaptive Server は OAM ページまたはアロケーション・ページの修正時に単一のログ・レコードを作成する (同一のデバイスで混合ログおよびデータを使用するように設定されているデータベースにおいて)。バッファの位置決めの際に作成される隠れたデッドロックを回避するために、Adaptive Server が syslogs をフラッシュする必要がある。Adaptive Server はテンポラリ・データベースのバッファの位置決めを行わないため、単一ログ・レコードの書き込み時にテンポラリ・データベースの syslogs データをフラッシュする必要がある。これによって、ログ・セマフォ競合が減少する。
- ダーティ・ページをディスクにフラッシュする − リカバリが必要なデータベースでは、Adaptive Server はチェックポイントのときにダーティ・ページをディスクにフラッシュする。Adaptive Server でエラーが発生した場合は、コミットされたすべてのデータがディスクに保存される。テンポラリ・データベースの場合、Adaptive Server は実行時のロールバックはサポートするが障害リカバリはサポートしないため、チェックポイントでのダーティ・データ・ページのフラッシュを回避できる。
- 先書きログを回避する − 先書きログによって、コミットされたトランザクションのデータは、ログの「再実行」(ログにリストされたトランザクションの再実行) およびアボートまたはロールバックされたトランザクションによって行われた変更の「取り消し」によってリカバリできる。Adaptive Server では、リカバリが不要なデータベースの先書きログをサポートしていない。Adaptive Server はテンポラリ・データベースをリカバリしないためテンポラリ・データベースのバッファの位置決めが行わない。これによって、テンポラリ・データベースを使用してトランザクションがコミットされたときに Adaptive Server はテンポラリ・データベースのログのフラッシュを省略できる。

ULC (ユーザ・ログ・キャッシュ)

Adaptive Server には、セッションに割り当てられるテンポラリ・データベース用の個別のユーザ・ログ・キャッシュ (ULC) が含まれています。ULC を使用すると、ユーザ・データベースとセッションのテンポラリ・データベース間で切り替えを行ったり、次のすべての条件が満たされたりしたときに、Adaptive Server がログのフラッシュを回避することができます。

- Adaptive Server が現在トランザクションをコミットしている。
- すべてのログ・レコードが ULC にある。
- コミット後のログ・レコードがない。

session tempdb log cache size 設定オプションを使用すると ULC のサイズを設定でき、必要なフラッシュの回数を決定できます。『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。

索引

I

L

M

N

O

P

R

S

T

U

W

あ

い

う

え

お

か

き

く

け

こ

さ

し

す

せ

そ

た

ち

つ

て

と

な

ね

の

は

ひ

ふ

へ

ほ

ま

み

め

ゆ

よ

り

れ

ろ