SYBASE®
An SAP® Company

新機能ガイド

Adaptive Server Enterprise

15.7 ESD #2

ドキュメント ID：DC01058-01-1572-01
改訂：2012 年 6 月

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

目次

第 1 章　遅延テーブルの作成

create table コマンドの with deferred_allocation パラメータを使用すると、テーブルへのページ割り付けを遅延させることができます。遅延テーブルは、アプリケーションで多数のテーブルを作成してそのうちの少数のテーブルだけを使用する場合に役立ちます。Adaptive Server ® がテーブルのページを割り付けるまで、そのテーブルは " 遅延 " と呼ばれます。

システム・テーブルには遅延テーブルのエントリが含まれます。これらのエントリを使用すると、遅延テーブルに関連付けられたオブジェクト (ビュー、プロシージャ、トリガなど) を作成できます。

Adaptive Server は最初のローを挿入 (テーブルをマテリアライズ) するときに遅延テーブルに対してページを割り付けます。最初の insert が実行される前にテーブルにアクセスすると (選択、削除、更新など)、領域使用量を報告する関数や他のテーブルへの DML 操作で参照整合性制約を適用する関数は、テーブルが空であるかのように動作します。つまり、遅延テーブルに対して select を実行すると空の結果セットが生成されるということです。遅延テーブルにインデックスを作成できますが、作成したインデックスに対するページ割り付けは、Adaptive Server がこのテーブルをマテリアライズするまで遅延されます。

遅延テーブルを作成するためのデータベース・レベルでの設定

以降に作成されるすべてのユーザ・テーブルに対してページ割り付けが遅延されるようにデータベースを設定するには、‘deferred table allocation' データベース・オプションを使用します。

```
sp_dboption database_name, "deferred table allocation", true
```

システム・データベース (master、sybsystemprocs、sybsystemdb など) またはテンポラリ・データベースに対して deferred table allocation を有効にすることはできません。

遅延テーブルの作成

遅延テーブルを作成するには create table . . . with deferred_allocation パラメータを使用します。

create table *table_name* . . . with deferred_allocation

たとえば、im_not_here_yet というテーブルを作成するには、次のように入力します。

```
create table im_not_here_yet (
col_1 int,
col_2 varchar(20)
)
with deferred_allocation
```

遅延テーブルを作成する際に、sp_dboption 'deferred table allocation' を有効にする必要はありません。

sp_dboption 'deferred table allocation' が有効な場合に非遅延テーブルを作成するには、create table . . . with immediate_allocation を使用します。構文は次のとおりです。

create table *table_name* . . . with immediate_allocation

遅延テーブルの明示的なマテリアライズ

遅延テーブルを明示的にマテリアライズするには、alter table . . . immediate_allocation を使用します。構文は次のとおりです。

alter table *table_name* immediate_allocation

テーブルをマテリアライズすると、すべてのデータ・パーティションとインデックス・パーティションにページが割り付けられます。

たとえば、テーブル im_not_here_yet をマテリアライズするには、次のように入力します。

```
alter table im_not_here_yet immediate allocation
```

遅延テーブルの識別

sp_help の object_status カラムには遅延テーブルに関する情報があります。次の例は、im_not_here_yet 遅延テーブルに関する sp_help 出力の一部を示しています。

```
sp_help im_not_here_yet
Name            Owner            Object_type            Object_status
Create_date
---------------  -----  -----------  -------------------  -------------------
im_not_here_yet    dbo   user table  deferred allocation    Apr 9 2012 2:09PM
```

sysobjects の sysstat3 カラムには、そのテーブルが遅延テーブルであることを示す 0x80 ステータス・ビットがあります。

遅延テーブルのロールバック

遅延テーブルのマテリアライズが、ロールバックするトランザクションに含まれている場合、遅延テーブルに対して実行されたページ割り付けはロールバックされません。

次に例を示します。

```
create table im_not_here_yet with deferred_allocation
go
begin tran t1
go
insert into deferred table ...
go
rollback tran t1
```

insert によって im_not_here_yet テーブルがマテリアライズされ、値が挿入されます。rollback tran によってテーブルから値が削除されますが、ページ割り付けはロールバックされません。そのため、テーブルはマテリアライズされたままになり、遅延テーブルではなくなります。

遅延テーブルにおけるコマンドの動作

ほとんどのコマンドは、遅延テーブルに対して空のテーブルと同じように動作します。

コマンド	遅延テーブルに対する動作
insert	テーブルをマテリアライズし、insert を実行する。
select	選択されるローは 0 行。
update	影響を受けるローは 0 行。
delete	影響を受けるローは 0 行。
alter table	テーブルをマテリアライズし、alter table を実行する。
drop table	テーブルを削除する。
create view、trigger または procedure	ビュー、トリガ、またはプロシージャを作成する。
create index	ページ割り付けなしでインデックスを作成する。
drop index	インデックスを削除する。
reorg のサブコマンド	なし
update statistics	なし
truncate table	なし
dbcc checktable	なし
dbcc checkcatalog	遅延テーブルのインデックスをスキップする。

第 2 章　同時実行性の機能強化

Adaptive Server バージョン 15.7 ESD # 2 以降の reorg rebuild コマンドには、online パラメータが含まれています。このパラメータを使用すると、ユーザのデータをブロックせずにデータを再編成し、メンテナンスを実行できます。

reorg rebuild

reorg rebuild ... with online を使用すると、データをオフラインにせずにデータを再編成できます。構文は次のとおりです。

reorg rebuild *table_name*
[with online]

たとえば、titles テーブルのインデックスを再構築する際にデータをオンラインのままにしておくには、次のコマンドを入力します。

```
reorg rebuild titiles with online
```

reorg rebuild ... online には 3 つのフェーズがあります。

- 短時間の排他テーブル・ロックを実行して、新しいメタデータを設定するためのブロック・フェーズ
- データを再編成して、同時処理との同期をとるための非ブロック・フェーズ
- 排他テーブル・ロックを再取得して、残りの同時処理との同期化および新しいメタデータの設定を行うためのブロック・フェーズ

Sybase® では、テーブルのトランザクションの負荷が比較的低いときに、reorg rebuild ...online を実行することをおすすめします。

リカバリ

reorg rebuild ... online の実行は、単一のトランザクション内で行われます。online パラメータによって実行された処理のリカバリは、online パラメータを使用せずに実行された処理のリカバリと同様です。online パラメータを使用して実行された処理をロールバックする場合は、以下の問題について考慮してください。

- ユーティリティのランタイム・ロールバックを使用すると、online パラメータによって割り付けられたページは解除されます。
- クラッシュのリカバリを行うと、online パラメータによって割り付けられたエクステントの割り当てステータスはクリアされます。
- 高可用性環境でノード・フェールオーバ・リカバリ中に reorg rebuild ... online によって物理ロックまたは論理ロックを開始しようとすると、リカバリが完了するまで待機してからロックが取得されます。

制限事項

reorg rebuild ... online には以下の制限事項があります。

- reorg rebuild を実行するテーブルにはユニーク・インデックスが必要です。
- reorg rebuild ... with online の実行中に、すべての DML (つまり、select (select into は対象外)、insert、update、および delete) においてテーブルの操作が可能になります。reorg rebuild ... online の実行中に、すべてのページロック・スキームのテーブルにページ分割が発生する挿入は許可されません。
- 1 つのテーブルに対して複数の reorg rebuild ... online インスタンスを同時に実行することはできません。
- null 入力が不可能なマテリアライズされていないデフォルト・カラムは、reorg rebuild ... online によってマテリアライズされません。

第 3 章　パーティションのマージおよび分割

時間の経過とともに、パーティションのデータ分布に偏りが見られるようになる場合や、最初にデータを分割した方法が現在のビジネス要件に適さなくなる場合があります。alter table を使用してパーティションをマージ、分割、または移動し、データを再分配すると、パーティションの使用によるパフォーマンス上のメリットが回復します。

たとえば、ある企業でデータへのアクセス効率を向上させるために、4 つの地域 (北部、南部、東部および西部) に応じてパーティションを分割するとします。パーティションを分割することにより、顧客担当者は、担当する地域の顧客にアクセスする際に、他の地域とは切り離して高速かつ効率的にアクセスできるようになります。南部で売上が増加し、顧客基盤が大幅に拡大した場合、パーティション・スキャンや保守操作に関連するクエリが頻繁に実行されることによって南部パーティションでのパフォーマンスが低下し、効率が悪くなる恐れがあるため、顧客データを分割することのメリットが損なわれます。この場合、南部パーティションのデータを 2 つのパーティション (南東部および南西部) に分割することによって、他のパーティションのデータに影響を与えずにパフォーマンスを回復させることができます。

また、ある企業で、売上データが年度の四半期 (Q1、Q2、Q3、および Q4) ごとのパーティションに分割されているため、パフォーマンスの向上を図るために、それらのパーティションをマージするとします。この企業では年度末にその年のデータをマージしてアーカイブします。過去の年度の売上データに対してはアクセス頻度が低く、また、古いデータは更新ではなく読み取り対象として使用される可能性が高いため、パーティションをマージすると効果的です。

分割またはマージで使用可能な分割スキーム

alter table を使用すると、以下の分割スキームを分割、マージ、または移動できます。

- 範囲分割
- リスト分割

ラウンドロビン方式で分割されたテーブルに含まれるデータのロケーションは、データ・パーティション間でランダムに分散されるため、ラウンドロビン方式のパーティションを分割またはマージする必要はありません。

ハッシュ・パーティションのテーブルの場合は、再分割ユーティリティを使用してデータを再分配します。

パーティションの分割

2 つ以上のパーティションにデータを再分配するには、alter table ... split partition パラメータを使用します。構文は次のとおりです。

```
alter table table_name
split partition partition_name
into partition_condition_clause
```

各パラメータの意味は次のとおりです。

- *partition_name* ― 分割するパーティション。
- *partition_condition_clause* ―ソース・パーティション・データの分割方法を指定する条件。通常、条件は数値の範囲またはデータの範囲で構成されます。パーティション条件には、ソース・パーティション内のデータだけを含める必要があり、さらにそれらのすべてのデータを含める必要があります。

  partition_condition_clause には、ソース・パーティションと同じセグメントまたは新しいセグメントを指定できます。分割先のパーティション・セグメントを指定しない場合は、ソース・パーティションが置かれているセグメントに新しいパーティションが作成されます。

『リファレンス・マニュアル：コマンド』を参照してください。

alter table ... split partition を発行するには、select intobulkcopy を有効にする必要があります。デフォルトでは、alter table ... split partition を発行すると、分割操作の影響を受ける、分割されたテーブルのローカル・インデックスまたはグローバル・インデックスのセクションが再構築されます。

alter table ... split partition の操作では、インデックスの再構築手順を除き、ログが記録されません。Sybase では、alter table ... split partition コマンドの実行後に、データベース・ダンプを実行することをおすすめします。

次の例では、orders テーブルを作成してから、テーブルのパーティションを分割し、データを再分配します。

```
create table orders (orderid int, amount float,
orderdate datetime)
partition by range (amount)
( P1 values <= (10000) on seg1,
  P2 values <= (50000) on seg2,
  P3 values <= (100000) on seg3,
  P4 values <= (MAX) on seg4)

create clustered index ind_orderid
on orders(orderid) local index (i1 on seg1, i2 on seg2,
i3 on seg3, i4 on seg4)

alter table orders
split partition P2
into
( P5 values <= (25000) on seg2,
  P6 values <= (50000) on seg3)

alter table orders
split partition P3
into
( P7 values <= (50000) on seg2,
  P8 values <= (100000) on seg3)

alter table orders
split partition P4
into
( P9 values <= (200000),
  P10 values <= (MAX))
```

パーティションのマージ

マージと互換性のある (つまり、マージで使用できる) 2 つ以上のパーティションのデータを 1 つのパーティションに結合するには、alter table ... merge partition を使用します。パーティションがマージと互換性があるかどうかは、そのパーティションの分割方法によって決まります。

- リスト分割されたテーブルの場合、2 つのパーティションはいずれもマージと互換性があります。
- 範囲分割されたテーブルの場合、パーティションはマージと互換性のあるパーティションと隣接している必要があります。

構文は次のとおりです。

```
alter table table_name
merge partition {partition_name [{, partition_name}…]}
into destination_partition_name [on segment_name]
```

各パラメータの意味は次のとおりです。

- *partition_name* ― マージするソース・パーティション。ソース・パーティションはすべて同じセグメント上になければなりません。
- *destination_partition_name* ―新規または既存のパーティション。*destination_partition_name* が既存のパーティションである場合、マージ対象のソース・パーティションを指定することはできません。

  マージ後のパーティションのパーティション条件は、マージされるすべてのソース・データ・パーティションのパーティション条件から抽出されます。そのため、マージ後のパーティションにはマージされるソース・データ・パーティションに置かれているすべてのデータが含められます。たとえば、リスト分割されたテーブルの場合、マージされるパーティションの新しいパーティション条件は、ソース・データ・パーティションの条件を構成するすべての値を結合したものになります。

『リファレンス・マニュアル：コマンド』を参照してください。

alter table ... merge partition を発行するには、select intobulkcopy を有効にする必要があります。

alter table ... merge partition では、完全にログが記録されます。サーバの障害をリカバリする場合は、トランザクション・ダンプを使用します。

次の例では、sales テーブルを作成してから、テーブルのパーティションをマージし、データを結合します。

```
create table sales(salesmanid int, salesdate datetime,
salesregion varchar(10))
```

```
partition by range(salesdate)
( Q1 values <= ('31 Mar 2007'),
  Q2 values <= ('30 Jun 2007'),
  Q3 values <= ('30 Sep 2007'),
  Q4 values <= ('31 Dec 2007'))
create index ind_region on sales(salesregion)

alter table sales
merge partition Q3
into Q4

alter table sales
merge partition Q1, Q2, Q3, Q4
into Y2007
```

パーティションの移動

パーティション (およびそのパーティションのインデックス) を指定したセグメントに移動するには、alter table ... move partition を使用します。構文は次のとおりです。

alter table *table_name*
move partition *partition_name*
to *destination_segment_name*

各パラメータの意味は次のとおりです。

- *partition_name* ー 移動するパーティション。
- *destination_segment_name* ーパーティションの移動先となる新規または既存のセグメント。*destination_segment_name* に "default" を指定することはできません。

『リファレンス・マニュアル：コマンド』を参照してください。

alter table ... move partition を発行するには、select intobulkcopy を有効にする必要があります。

ロック

パーティションの分割、マージ、または移動が行われている間は、操作が実行されるテーブルとそのテーブルのシステム・テーブル・エントリに排他ロックがかかります。

パーティションの分割またはマージによるインデックスへの影響

split、merge、または move partition をインデックス付きのテーブルに実行すると、影響を受るすべてのインデックスが再構築されます。

表 3-1: インデックス・パーティションの分割およびマージ

コマンド	グローバル・ノンクラスタード・インデックス	ローカル・インデックス	ローカル・クラスタード・インデックス	ローカル・ノンクラスタード・インデックス
split partition	インデックスが再構築される	影響を受けるすべてのインデックス・パーティションが再構築される	すべてのインデックス・パーティションが再構築される	デフォルト・セグメントに再構築される
merge partition	ソースとマージ先のセグメントが同じ場合は影響なし	影響を受けるすべてのインデックス・パーティションが再構築される	すべてのインデックス・パーティションが再構築される	デフォルト・セグメントに再構築される

分割またはマージするテーブルに別のセグメントのインデックスが含まれる場合、新たに再構築されるインデックスは、インデックスのタイプによって異なるセグメントに配置されます。

表 3-2: パーティションの分割によるインデックス・セグメントへの影響

インデックスのタイプ	分割またはマージ操作後
グローバル・ノンクラスタード・インデックス	インデックスは操作前と同じセグメントに残る。
ローカル・ノンクラスタード・インデックス	• (分割またはマージ先のデータ・パーティションに対応する) 新しいインデックス・パーティションが、インデックス・レベルで指定したセグメントに配置される。 • インデックス・セグメントを指定しない場合は、デフォルト・セグメントにインデックスが配置される。 • 影響を受けないインデックス・パーティション (分割またはマージに含まれないその他のデータ・パーティションが該当) はすべて、分割またはマージ操作前と同じセグメントに残る。
ローカル・クラスタード・インデックス	• 新しいインデックス・パーティションは、対応するデータ・パーティションが配置されるセグメントと同じセグメントに配置される。 • 影響を受けないインデックス・パーティション (分割またはマージに含まれないその他のデータ・パーティションが該当) は、分割またはマージ操作前と同じセグメントに残る。

パーティションの分割またはマージ後の正確な統計の維持

パーティションを分割またはマージすると、systabstats および sysstatistics から統計情報が削除されます。Sybase では、パーティションのマージまたは分割後に update statistics を実行することをおすすめします。

第 4 章

ステートメント・キャッシュに保存されるクエリの最大サイズ

バージョン 15.7 ESD #2 より前の Adaptive Server では、statement cache size を大きなサイズに設定していたとしても、ステートメント・キャッシュに保存される個々の文は 16K までに制限されていました。

Adaptive Server バージョン 15.7 ESD #2 以降では、statement cache size および max memory 設定パラメータの値を増やすことにより、個々の SQL 文を 2MB まで (64 ビット・マシンの場合) ステートメント・キャッシュに保存できます。

ステートメント・キャッシュの SQL クエリ・テキストが 16K 未満の場合は、show_cached_text を使用して SQL クエリ・テキストを表示します。これに対し、SQL クエリ・テキストが 16K を超えると、テキスト全体がステートメント・キャッシュで使用できる場合でも、show_cached_text によって SQL クエリ・テキストがトランケートされます。

16K を超える SQL クエリ・テキスト表示するには、show_cached_text_long を使用します。show_cached_text_long では、SQL クエリ・テキストが最大 2MB まで表示されます。

第 5 章　*show_cached_plan_in_xml* の機能強化

Adaptive Server 15.7 ESD #2 では、show_cached_plan_in_xml の出力に、以下に関する新しい情報が含まれます。

- インデックス・スキャンの範囲
- プランで使用されるワーク・テーブル
- 動的パーティション削除の内容
- 合計論理 IO (lio) および合計物理 IO (pio)

スキャンの範囲

<scanCoverage> タグには、クエリ・プランのインデックス・スキャンが "Covered" であるか "NonCovered" であるかが示されます。

カバード・インデックス・スキャンの詳細については、『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「インデックス」を参照してください。

例

```
select show_cached_plan_in_xml(1139220075, 0)

<text>
<![CDATA[
SQL Text:select * from sysobjects group by name]]>
<text>
<IndexScan>
<VA>0<VA>
...
<scanCoverage> NonCovered <scanCoverage>
...
<IndexScan>
```

ワーク・テーブル

<WorkTable> の下の <wtObjName> タグには、クエリ・プランで使用されているワーク・テーブルの名前が示されます。<WorkTable> タグは、ワーク・テーブルが作成される演算子の下に表示されます。

例

```
select show_cached_plan_in_xml(107219961, 0)
go

<text>
<![CDATA[
SQL Text:select distinct c1, c2 from t1, t2 where c1 = d1]]>
<text>
<opTree>
...
    <MergeJoin>
        ...
      <WorkTable>
           <wtObjName>WorkTable2<wtObjName>
    <WorkTable>
    ...
    <MergeJoin>
```

動的パーティション削除

show_cached_plan_in_xml の "partitionInfo" セクションには、<dynamicPartitionElimination> タグによって動的パーティション削除の情報が示されます。これは、実行時にパーティションの削除が行われていることを示します。show_cached_plan_in_xml の <eliminatedPartition> タグには、コンパイル時に削除されたパーティションが示されます。

パーティション削除の詳細については、『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』の並列クエリ処理に関する章を参照してください。

例 2

```
select show_cached_plan_in_xml(435436901,0)
go
<query>
   <statementId>435436901<statementId>
   <text>
   <![CDATA[
```

```
. . .
                    <partitionInfo>
                           <partitionCount>3<partitionCount>
                           <eliminatedPartition>1<eliminatedPartition>
                           <eliminatedPartition>3<eliminatedPartition>
                           <dynamicPartitionElimination>No<dynamicPartition
                            Elimination>
                      <partitionInfo>

. . .
```

合計論理 IO および合計物理 IO

<totalLio> タグと <totalPio> タグには、プランごとの合計論理 IO と合計物理 IO が示されます。2 組の <totalLio> タグと <totalPio> タグは、論理 IO と物理 IO の実測値と推定値を示します。

論理 IO と物理 IO の詳細については、『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「クエリ最適化方式と見積もりの表示」を参照してください。

例

```
select show_cached_plan_in_xml(1123220018, 0)
go
    <text>
           <![CDATA[
                SQL Text:select * from titles]]>
    <text>
<Plan>
<optTree>
. . .


          <est>
                <totalLio>3<totalLio>
                <totalPio>3<totalPio>
            <est>
            <act>
                <totalLio>3<totalLio>
                <totalPio>1<totalPio>
            <act>
. . .
```

第 6 章　高速ログ *bcp*

Adaptive Server バージョン 15.7 ESD #2 以降では、高速モードですべてのログを記録する bcp を実行できます。これにより、完全なデータ・リカバリが可能になります。前のバージョンでは、ページ割り付けのログのみが記録されていました。

『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。

第 7 章

拡張された並列 create index

Adaptive Server バージョン 15.7 ESD #2 以降では、create index を並列形式で発行することで、より効率的にコマンドが実行されるようにクエリ実行エンジンを使用できます。

拡張された並列 *create index* の設定

拡張された並列 create index は、デフォルトでは無効化されています。また、この設定は enable functionality group 設定パラメータに含まれています。Adaptive Server で並列 create index を使用できるようにするには、次の手順に従います。

1 enable functionality group 設定パラメータを有効にします。

```
sp_configure "enable functionality group", 1
```

2 データベース・オプション select intobulkcopypllsort を true に設定します。

```
sp_dboption database_name, "select into", true
```

3 ハードウェア環境に応じて、次の設定パラメータを設定します。

- number of worker processes －すべてのユーザに対する同時実行並列スレッドの最大数を設定します。
- max parallel degree －個々のユーザに対する**最大並列数**を設定します。ただし、number of worker processes 以下にします。
- max online engines －並列スレッドを設定する場合には、ハードウェアの可用性や他の負荷要素に応じて十分な**エンジン数**を設定することをおすすめします。通常、**ワーカー・プロセス数**は**エンジン数**よりも大きく設定します。

拡張された並列 create index の使用

並列 create index を行うように Adaptive Server が設定されると、並列実行の使用が create index の実行に最適であるかどうかが判定されます。逐次クエリ・プランが最も効率的であると判定された場合、並列クエリ・プランは使用されません。並列クエリ・プランが最も効率的であると判定された場合は、以下の条件を満たす場合に、拡張された並列クエリ・プランが選択されます。

- インデックスが作成されるテーブルが以下に該当する。
  - データオンリーロック形式を使用する。
  - 分割されていない。
  - テーブルが空でない。
- 作成するインデックスがノンクラスタード・インデックスである。
- インデックスの先行カラムに 2 つ以上の個別の値がある。

通常は逐次クエリ・プランが使用されるような場合でも、並列クエリ・プランが強制的に使用されるようにするには、create index ... with consumers = *N* を使用します。たとえば、テーブルに含まれるロー数が少なすぎる場合でも、次の並列クエリ・プランが使用されます。

```
create index i1 on t1(c1, c3) with consumers = 3
```

with consumers を使用して並列 create index を強制的に実行する際に、拡張された並列クエリ・プランが選択されない場合は、Adaptive Server 15.7 ESD #2 より前のバージョンの並列 create index クエリ・プランが使用されます。

showplan による並列 *create index* コマンドの表示

並列 create index を行うように Adaptive Server が設定されており、拡張された並列 create index クエリ・プランが選択された場合は、showplan を使用すると、PLL CREATE INDEX COORDINATOR 演算子と CREATE INDEX 演算子の下に create index コマンドに関する情報が表示されます。次に例を示します。

```
create index i1 on t5(c1) with consumers = 3
. . .

文 1 (1 行目 ) のクエリ・プラン。
調整プロセスと 3 作業プロセスにより並列に実行されました。

STEP 1
  クエリのタイプは CREATE INDEX です。

  5 operator(s) under root

|ROOT:EMIT Operator (VA = 5)
|
|   |PLL CREATE INDEX COORDINATOR Operator
|   |
|   |   |EXCHANGE Operator (VA = 3) (Merged)
|   |   |Executed in parallel by 3 Producer and 1 Consumer processes.

|   |   |
|   |   |   |EXCHANGE:EMIT Operator (VA = 2)
|   |   |   |
|   |   |   |   |CREATE INDEX Operator
|   |   |   |   |
|   |   |   |   |   |SCAN Operator (VA = 0)
|   |   |   |   |   |  FROM TABLE
|   |   |   |   |   |  t5
|   |   |   |   |   |  Table Scan.
|   |   |   |   |   |  Forward Scan.
|   |   |   |   |   |  テーブルの最初に位置付けます。
|   |   |   |   |   |  3 方向範囲再分割スキャンにより並列に実行されました。
|   |   |   |   |   |  データ・ページに対して IO サイズ 16 キロバイトを使用しています。
|   |   |   |   |   |  データ・ページに対する MRU でのバッファ置換方式
```

第 8 章　事前計算済み結果セット

事前計算済み結果セット (PRS) は、結果が計算され、保存されたビューであり、将来的な使用に備えて用意されたものです。事前計算済み結果セットを使用するように Adaptive Server を設定すると、事前にクエリが計算され、後続の繰り返し処理で計算済みの結果の使用が試行されます。事前計算済み結果セットは、マテリアライズされたビューとも呼ばれます。

概念上、事前計算済み結果セットはビューでもあり (システム・テーブルに保存されたクエリ定義を含むため)、テーブルでもあります (永続的なデータを含むため)。インデックスの作成、update statistics の実行など、テーブルに実行するさまざまな操作を、事前計算済み結果セットにも実行できます。

事前計算済み結果セットを使用するように Adaptive Server が設定されている場合、オプティマイザは事前計算済み結果セットを使用して各クエリを自動的に書き換えようとします。ただし、オプティマイザが選択する最終的なプランは主にコスト・ベースになります。

オプティマイザは、事前計算済み結果セットを使用してクエリを書き換える際に、どの事前計算済み結果セットが最も適しているかを判断します。クエリのすべてまたは一部を事前計算済み結果セットで置き換えるようにオプティマイザが選択した場合、必要な補正 (つまり、元のユーザ・クエリと書き換えられたクエリの等価性を保つために必要な述部) も書き換えられたクエリに追加されます。たとえば、ユーザ・クエリに次のジョインが含まれる場合に、

```
c1=c2 and c2=c3 and c3=c4
```

事前計算済み結果セットに次のジョインが含まれる場合は、

```
c1=c2 and c3=c4
```

同等のクエリを形成するために、事前計算済み結果セットを使用して書き換えられたクエリには、`c1=c3` のように補正された述部が必要になります。

インデックスと同様に、insert、update、および delete の各文を同時に実行する場合は、事前計算済み結果セットでも管理コストがかかります。定義に複数のテーブルのジョインが含まれる場合、一般的に、事前計算済み結果セットの管理オーバヘッドは、インデックスを管理するときよりも大きくなります。したがって、事前計算済み結果セットは、OLTP で insert、update、delete 文の同時実行が頻繁に行われ、インデックス・ベースの単純な select が実行される場合には向きません。

事前計算済み結果セットの利点

サイトで事前計算済み結果セットを使用すると効果的であるかどうかは、そのサイトの設計方法によって異なります。できるだけ多くのクエリ (特にジョインの多いクエリ) を事前に計算し、複数のクエリで使用できるようにしておく必要がある場合でも、事前計算済み結果セットを使用すると追加のディスク領域が必要となり、管理コストもより高くなります。追加のインデックスを作成することにより、クエリ・パフォーマンスを向上させることができますが、追加の管理コストも発生します。

事前計算済み結果セットは、負荷の高いクエリ (集合演算やジョイン演算が集中的に行われるクエリなど) が頻繁に実行される場合に最も適しています。クエリを実行すると、基本テーブルの代わりに既存の事前計算済み結果セットを使用するように、オプティマイザがクエリを書き換えようとします。

通常は、アプリケーションの負荷を把握し、その負荷に基づいて事前計算済み結果セットを設計します。まず手始めに、すべてのクエリについて、ジョインとその使用頻度をまとめたグラフを作成し、複数のクエリで同じ事前計算済み結果セットを使用する適当な候補を見つけるとよいでしょう。

事前計算済み結果セットは、テストしてから運用環境に配置してください。読み取り専用のクエリまたはほぼ読み取りで使用されるクエリの場合、そのクエリのパフォーマンス向上と、クエリが使用する追加のディスク領域やクエリによるデータ入力にかかる時間を照らし合わせて評価します。読み取り専用のクエリとほぼ読み取りのみのクエリが混在する場合、事前計算済み結果セットがスループットに与える影響を評価します。

事前計算済み結果セットを使用するための Adaptive Server の設定

事前計算済み結果セットを作成または変更する前に、以下の set セッション・パラメータが正しく設定されていることを確認してください。

- set ansinull － on
- set arithabort － on
- set arithignore － off
- set string_rtruncation － on

事前計算済み結果セットを作成するには、create precomputed result set を使用します。クエリで事前計算済み結果セットを使用するには、そのセッションで set materialized_view_optimization コマンドを発行します。

事前計算済み結果セットの作成

事前計算済み結果セットを定義するには、create コマンドを使用します。構文は次のとおりです。

```
create {precomputed result set | materialized view}
    prs_name [(alternative_column_name
    [ [constraint constraint_name]
        unique (column_name,...)]

    [{immediate | manual } refresh]
    [{populate | nopopulate}]
    [enable | disable]
    [{enable | disable } use in optimization]
    [lock { datarows | datapages | allpages}]
    [on segment_name]
    [partition_clause]

as query_expression
```

事前計算済み結果セットでは、次のものを指定できます。

- パーティション
- セグメント
- インデックス (関数インデックスは指定できません)

- ユニーク・キー (immediate refresh を使用する事前計算済み結果セットを作成する場合は、ユニーク・キー制約を含める必要があります)

基本テーブルを削除すると、事前計算済み結果セットは disabled に変更されます。

『リファレンス・マニュアル：コマンド』を参照してください。

事前計算済み結果セットの識別

sp_help の Object_type カラムと object_status カラムには、事前計算済み結果セットに関する情報が次のように表示されます。

```
sp_help mv1
Name  Owner  Object_type              Object_status
  Create_date
-----  -----  ----------------------  ------------------------------------
  --------------------
mv1   dbo   precomputed result set  immediate, enabled, enabled for QRW
   Apr 10 2012  8:57AM
...
```

sysobjects の type カラムの値が RS の場合は、そのオブジェクトが事前計算済み結果セットであることを示しています。

事前計算済み結果セットのリフレッシュ

事前計算済み結果セットは、構成元の基本テーブルと必ずしも同期されているとはかぎりません。そのため、事前計算済み結果セットを自動または手動でリフレッシュする必要があります。リフレッシュ・ポリシーを設定するには、事前計算済み結果セットの作成時または作成後に alter precomputed result set コマンドを使用します。

- 即時リフレッシュー事前計算済み結果セットは基本テーブルを更新するトランザクション内で更新されます。これは、デフォルトのオプションです。ただし、immediate refresh を使用する事前計算済み結果セットを作成する場合、そのユーザが定義クエリ内のすべてのテーブルを所有している必要があります。

- 手動リフレッシュ－事前計算済み結果セットは明示的な refresh コマンドによって更新されます。手動リフレッシュは維持されないため、その事前計算済み結果セットに含まれるデータは (refresh の発行直後でも) 古いものであるとみなされます。したがって、クエリで古いデータが許容される場合にのみ、この事前計算済み結果セットがクエリの書き換え対象として選択されます。refresh コマンドは、独立性レベル 1 以上で実行されます。

事前計算済み結果セットを手動でリフレッシュする構文は次のとおりです。

```
refresh {precomputed result set | materialized view}
    [owner_name.]prs_name
```

事前計算済み結果セットが抽出された基本テーブルのスキーマが変更されている場合、または削除されて再作成されている (つまり、オブジェクト ID が変更されている) 場合、refresh コマンドは失敗し、事前計算済み結果セットの削除と再作成が必要であることを示すエラーが返されます。

事前計算済み結果セットの所有者のみが refresh コマンドを使用できます。基本テーブルを更新するパーミッションがユーザにある場合、そのユーザは事前計算済み結果セットを管理することもできます。

ほとんどの場合、オプティマイザはクエリを書き換える際に、manual refresh ではなく immediate refresh を指定して事前計算済み結果セットを使用します (materialized_view_optimization が stale に設定されている場合を除く)。

update、insert、delete の各文が発生するタイミングを制御する場合は、手動による事前計算済み結果セットのリフレッシュが最適です。各文が発生したあとで、事前計算済み結果セットの手動の計画リフレッシュを実行してから事前計算済み結果セットを使用すると、読み取り専用アプリケーションに効果的です。ただし、手動リフレッシュの実行と計画には、それに応じた時間と追加のディスク領域が必要となります。

注意　事前計算済み結果セットの作成後、その所有者には基本テーブルに対する select パーミッションがない場合があります。そのような場合に事前計算済み結果セットを手動リフレッシュで管理すると失敗することがあります。また、失敗した場合は、基本テーブルからの新しい変更は事前計算済み結果セットに反映されません。

---

refresh コマンドをバッチの一部として実行することはできません。

次の例では、事前計算済み結果セットをリフレッシュする方法を説明します。

1 テーブル t1 を作成します。

```
create table t1 (
c1 int,
c2 int,
c3 char(5))
```

このテーブルに次のデータを入力します。

```
 c1          c2          c3
----------- ----------- -----
          1           3  Aagg
          2           8  Xyz
```

2 テーブル t2 を作成します。

```
create table t2
(a1 int,
a2 int,
a3 char(5))
```

このテーブルに次のデータを入力します。

```
 a1          a2          a3
----------- ----------- -----
          1           5  Ghr
          2           1  Gser
          3           6  agfh
```

3 事前計算済み結果セット prs_1 を作成します。

```
create precomputed result set prs_1
unique (t1.c1, t2.a2)
as select t1.c1, t2.a2 from t1, t2 where t1.c1=t2.a1
```

prs_1 が作成され、次の最初のローに格納されます。

```
 c1          a2
----------- -----------
          1           5
          2           1
```

4　t1 に値 3 、7、および "fhi" を挿入すると、prs_1 が 値 3 および 6 でただちに更新されます。

```
 c1          a2
----------- -----------
          1           5
          2           1
          3           6
```

5　a1 = 2 に該当するローを t2 から削除すると、prs_1 がこの変更内容でただちに更新されます。

```
 c1          a2
----------- -----------
          1           5
          3           6
```

基本テーブルを更新しているトランザクションがロールバックされると、その基本テーブルの事前計算済み結果セットに対する update も同一トランザクションの一部としてただちにロールバックされます。

事前計算済み結果セットの変更

事前計算済み結果セットのポリシーまたはプロパティを変更するには、alter コマンドを使用します。構文は次のとおりです。

```
alter {precomputed result set | materialized view}
        prs_name
    {immediate | manual} refresh
    | enable | disable
    | {enable | disable} use in optimization
```

『リファレンス・マニュアル：コマンド』を参照してください。

次の例では、事前計算済み結果セット author_prs を manual から immediate に変更します。

```
alter precomputed result set author_prs
immediate refresh
```

alter を使用して manual リフレッシュから immediate リフレッシュ、または disable から enable に変更すると、事前計算済み結果セットが自動的にリフレッシュされます。事前計算済み結果セットを disable use in optimization に変更すると、その事前計算済み結果セットは以降のクエリ書き換えの対象から除外されます。ただし、事前計算済み結果セットを使用するキャッシュ済みプランは再コンパイルされません。

他の DDL コマンドと同様に、複数文のトランザクションの一部として alter precomputed result set を発行することはできません (データベースの ddl in tran オプションが true に設定されている場合を除く)。alter precomputed result set を発行するには、事前計算済み結果セットの所有者である必要があります。

事前計算済み結果セットのベースとなる基本テーブルまたはビューが削除または変更された場合、その事前計算済み結果セットは自動的に disable に変更されます。事前計算済み結果セットの生成元である基本テーブルに対して bcp in または select into existing を実行すると、その事前計算済み結果セットは disable に設定されます。

(基本テーブルに対して alter precomputed result set コマンドまたは alter table コマンドを使用して) 事前計算済み結果セットを disable に変更すると、その事前計算済み結果セットを使用するキャッシュ済みプランは、次回実行時に再コンパイルされます。

事前計算済み結果セットの削除またはトランケート

事前計算済み結果セットの削除では、そのデータ、システム・テーブル・エントリ、事前計算済み結果セットが削除されます。構文は次のとおりです。

```
drop {precomputed result set | materialized view}
    prs_name
```

drop precomputed result set を発行するには、事前計算済み結果セットの所有者である必要があります。『リファレンス・マニュアル：コマンド』を参照してください。

次の例では、authors_prs を削除します。

```
drop precomputed result set authors_prs
```

事前計算済み結果セットのデータをトランケートするには、truncate コマンドを使用します。truncate では、事前計算済み結果セットの定義がシステム・テーブルで保持されるので、後から refresh コマンドを使用して、その事前計算済み結果セットを再配置できます。

事前計算済み結果セットをトランケートすると、その事前計算済み結果セットの状態は無効になります。refresh prs コマンドを発行すると、事前計算済み結果セットの状態は有効に戻ります。

構文は次のとおりです。

```
truncate {precomputed result set | materialized view}
    prs_name
```

次の例では、author_prs をトランケートします。

```
truncate precomputed result set authors_prs
```

Adaptive Server では、refresh コマンドがまず truncate コマンドとして実装され、続いて事前計算済み結果セットが再計算されます。万一、truncate が成功し、再計算が失敗した場合は、事前計算済み結果セットが無効のままになるので、refresh コマンドを再発行します。

古いデータを許可する設定

事前計算済み結果セットのデータを最新の状態に保つには、基本テーブルからの更新が必要です。事前計算済み結果セットに対して immediate 更新が設定されている場合は、基本テーブルのすべての更新が事前計算済み結果セットにも反映されます。この更新は、基本テーブルへの変更を使用した増分メンテナンスとして行われます。一方、事前計算済み結果セットに対して manual 更新が設定されている場合は、refresh コマンドを実行した場合にのみ更新が行われるため (増分メンテナンスが実行されるのではなく、コマンドの実行中に事前計算済み結果セットが再計算されます) データが古くなる可能性があります。

特に指定のない場合は、クエリの書き換え時に古い事前計算済み結果セットは使用されません。set materialized_view_optimization を使用すると、最適化中にクエリを書き換える場合に古い事前計算済み結果セットを使用しても良いかどうかをセッション・レベルで指定できます。

```
set materialized_view_optimization {disable | fresh | stale}
```

クエリの書き換えで古い事前計算済み結果セットが使用されるようにするには、以下の条件を満たす必要があります。

- ユーザが古い事前計算済み結果セットの所有者であること。
- set materialized_view_optimization に stale が設定されていること。

『リファレンス・マニュアル：コマンド』を参照してください。

事前計算済み結果セットのクエリ

事前計算済み結果セットから情報を選択することはできますが、事前計算済み結果セットに対して情報を挿入、更新、削除することはできません。代わりに、基本テーブルに対して情報の挿入、更新、削除を行ってから、事前計算済み結果セットをリフレッシュします。

クエリの書き換え

クエリ書き換えメカニズムでは、使用可能な事前計算済み結果に基づいて代替プランが生成されます。代替プランは、オプティマイザで他のプランと比較され、その中で見積コストが最も低いプランが選択されます。ただし、クエリ書き換えメカニズムは選択クエリのみで機能します。insert、update、delete、および select into の各クエリは書き換えの対象になりません。

同等の事前計算済み結果セットを作成する上で、クエリ全体が書き換えられる場合、またはクエリの一部が書き換えられる場合があります。これは、クエリのプロパティや使用できる事前計算済み結果セットによって異なります。事前計算済み結果セットには、Adaptive Server が書き換えるクエリの論理データ・セットが完全に含まれている必要があります。

たとえば、次のようなクエリを使用しているとします。このクエリには複数のテーブルから構成されるジョインや多数の述部、グループ、集計が含まれているため、複雑なクエリになっています。

```
select t1.col1,t2.col1,t3.col1,
     sum(t1.col3),sum(t2.col3), sum(t3.col3)
from t1, t2, t3
where t1.col1 = t2.col1
```

```
        and t2.col1 = t3.col1
        and t1.col2 < 60
        and t1.col1 > 5
        and t1.col2 + t2.col2 < 40
    group by t1.col1, t2.col1, t3.col1
```

さらに、次の事前計算済み結果セットを作成します。

```
    create precomputed result set newprs
    as
    select t1.col1 as p11, t1.col2 as p12, t2.col1 as p21,
        t2.col2 as p22, t3.col1 as p31, t3.col2 as p32,
        sum(t1.col3) as agg_s13,sum(t2.col3) as agg_s23,
        sum(t3.col3) as agg_s33
    from t1, t2, t3
    where t1.col1 = t2.col1
    and t1.col2 < 60
    and t1.col2 + t2.col2 < 40
    group by t1.col1, t2.col1, t3.col1, t1.col2,
        t2.col2, t3.col2
```

クエリ書き換えメカニズムによって元のクエリが次のようなクエリに変更されます。このクエリのほうがはるかに簡単であるためコストが低くなります。

```
    select p11,p21,p31,
        sum(agg_s13),sum(agg_s23),sum(agg_s33)
    from newprs
    where p21 = p31
        and p11 > 5
    group by p11, p21, p31
```

統計の更新

事前計算済み結果セットに対して updates statistics を実行できます。

事前計算済み結果セットの複写

事前計算済み結果セットでは、以下のコマンドが複写されます。

- create precomputed result set
- alter precomputed result set
- drop precomputed result set
- truncate precomputed result set
- refresh precomputed result set

事前計算済み結果セットの DDL を複写することはできますが、事前計算済み結果セットを複写の対象としてマークすることはできません。つまり、通常のテーブルとは異なり、事前計算済み結果セットに保存されているデータに対してメンテナンス上の変更が発生してもその変更は複写されません。これは、その変更が事前計算済み結果セットの DDL で開始されたものでも (その DDL 自体は複写されます)、基本テーブルの update トランザクション (immediate refresh ポリシーで開始された) の一部であっても同様です。

事前計算済み結果セットの DDL の複写は、事前計算済み結果セット機能を含む 2 つの Adaptive Server 間でのみサポートされます。

制限事項

事前計算済み結果セットには次のものを含めることはできません。

- 別の事前計算済み結果セットへの参照。ただし、事前計算済み結果セットを手動の refresh ポリシーで作成する場合は、その事前計算済み結果セットでビューを参照できます。
- select リストの式。ただし、group by リストの一部として式を含めることはできます。
- 暗号化カラム、または結果セット自体のカラム・データを暗号化する結果セット。
- 別のデータベース内にあるテーブルまたは関数への参照。
- 基本テーブル内の仮想計算カラムへの参照。ただし、基本テーブル内のマテリアライズされた仮想計算カラムを参照することはできます。

- compute、compute by、group by all、または order by 句。
- nondeterministic 関数 (getdate など)。
- XML。
- サブクエリ。
- 外部ジョインおよびセミジョイン。
- ユーザ定義関数の呼び出し。
- 抽出テーブル。
- システム・テーブル、テンポラリ・テーブル、またはフェイク・テーブルへの参照。
- union 句。
- ユニーク・キー制約以外の制約。
- identity 句、null 句、または not null 句が定義されているカラム。
- デフォルトまたはルール。
- カーソル。
- 統計集合関数。
- text、image、または unitext カラム。

上記の制約事項に加えて、immediate refresh ポリシーを使用する事前計算済み結果セットには次のものを含めることはできません。

- top、min、max、および avg コマンド
- distinct 句
- セルフジョイン
- 関数
- プロキシ・テーブルへの参照
- ビューへの参照
- 集計の結果を参照する having 句
- null 入力可能な式を参照する sum 関数

第 9 章　*dump database* コマンドと *dump transaction* コマンドの同時実行

Adaptive Server バージョン 15.7 ESD #2 以降では、dump transaction コマンドと dump database コマンドを同時に実行できます。これにより、データベースの更新がダンプ・ポリシーで設定された時間よりも長くかかる場合に、その更新を損失する危険性が低下します。

dump database コマンドは、2 つのフェーズで実行されます。第 1 フェーズでは、データベース・ページがダンプ・アーカイブにコピーされ、第 2 フェーズではトランザクション・ログのアクティブな部分がダンプ・アーカイブにコピーされます。dump transaction では、トランザクション・ログのアクティブな部分をダンプ・アーカイブにコピーする際に単一のフェーズが使用されます。

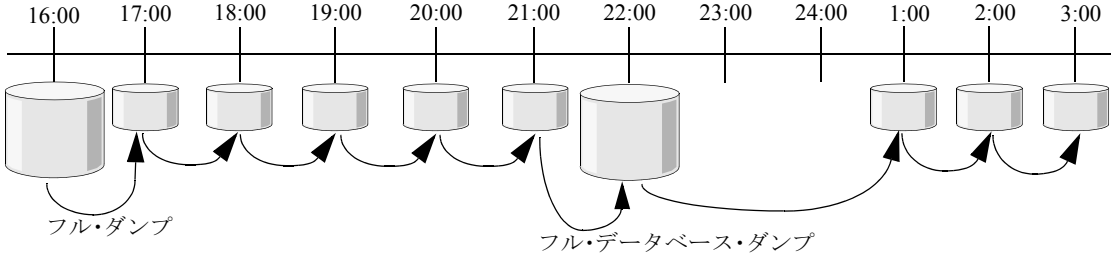
dump database では、データベース・ページをコピーする際 (最も時間のかかるフェーズ) に dump transaction を同時に実行できますが、トランザクション・ログのアクティブな部分をコピーする際には dump transaction を同時に実行することはできません。トランザクション・ログのアクティブな部分がコピーされている間は、dump database が終了するまで待機してから dump transaction が開始されます。または反対に dump transaction が終了するまで待機してから dump database が開始されます。

dump database と同時に行われる (つまり、dump database でアクティブ・ログのコピーが開始される前に完了する) dump transaction を、そのデータベース・ダンプの先頭でロードすることはできません。トランザクション・ログのダンプは前のロード・シーケンスに含まれます。

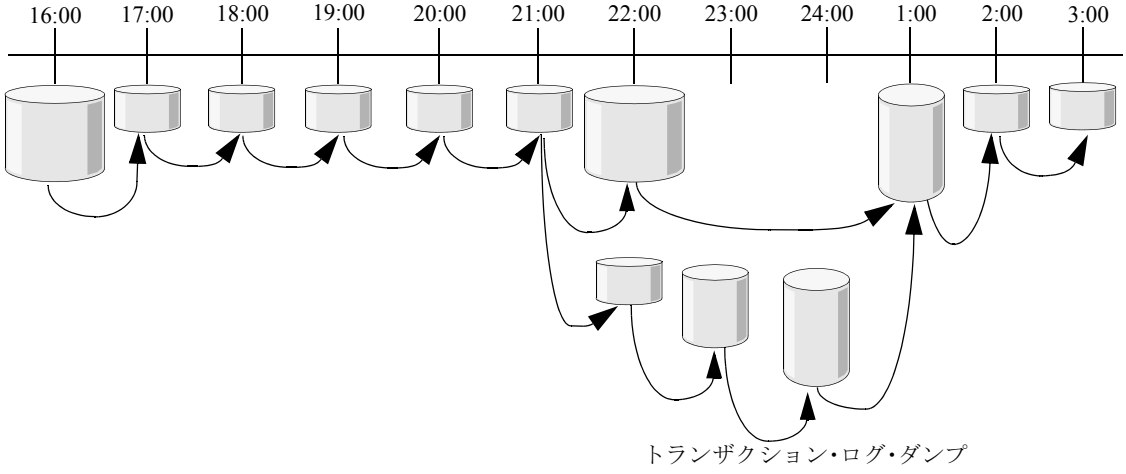
dump tran でダンプしたトランザクション・ログがデータベース・ダンプよりも前のものなのか (その場合、トランザクション・ログのロードは不要)、あるいはデータベース・ダンプの後に発生したものなのか (その場合、トランザクション・ログのロードが必要) を判断するのが困難な場合があります。ダンプ履歴ファイルを使用して順序の問題を解決し、必要に応じてトランザクション・ログを含めます (または除外します)。「ダンプ履歴ファイル」(58 ページ) を参照してください。

次に、一般的な日次バックアップ方針の例を示します。この例では、直列化されたダンプを使用しており、dump database の実行中、23:15 にクラッシュが発生しましたが、21:00 の時点までしかデータベースをリストアできないため、2 時間以上ものデータベースの更新が失われることになります。

直列化ダンプ



一方、同時ダンプでは、dump database の実行中にトランザクション・ログ・ダンプを継続できるため、23:15 にクラッシュが発生しても、22:00 と 23:00 にトランザクション・ログ・ダンプが実行されているため、23:00 の時点までデータベースをリストアすることが可能です。その結果、失われるデータベースの更新はわずか 15 分になります。



注意　dump transaction が dump database と同時に実行されている間、トランザクション・ログはトランケートされません。

同時ダンプを使用するための Adaptive Server の設定

enable concurrent dump tran 設定パラメータによって、Adaptive Server で同時ダンプを実行できるようにします。

enable concurrent dump tran

要約	
デフォルト値	0 (オフ)
有効な値	0 (オフ)、1 (オン)
ステータス	動的
表示レベル	包括
必要な役割	システム管理者
設定グループ	アプリケーション機能

enable concurrent dump tran は、Adaptive Server で同時ダンプを使用できるようにします。

enable concurrent dump tran は enable functionality 設定パラメータ・グループに含まれます。このグループのパラメータのデフォルト値は、enable functionality group に設定されている値によって変わります。enable functionality group 以外のこのグループの個々の設定パラメータの DEFAULT の値は、enable functionality group と同じ値に設定されていることを意味します。たとえば、enable functionality group を 1 に設定すると、そのグループのその他の設定パラメータの DEFAULT 値は 1 に設定されます。

enable functionality group の値を除けば、アプリケーション機能グループ内の個々の設定パラメータに対する sp_configure と sp_helpconfig からの出力の DEFAULT 値は無視できます。

詳細については、『システム管理ガイド 第 1 巻』を参照してください。

制限事項

同時ダンプを使用する場合、次の操作はできません。

- dump database と別の dump database の同時実行
- dump transaction と別の dump transaction の同時実行
- 同時に実行されているデータベース・ダンプがない状態で dump transaction を開始した場合に、dump transaction の終了前に dump database コマンドを開始すること

第 10 章　ハッシュベースの update statistics

Adaptive Server バージョン 15.7 ESD #2 以降では、マイナー・インデックスの属性およびインデックスのないカラムに対してソートベースの統計を使用する代わりにハッシュベースの統計を収集できます。ソートベースの統計の代わりにハッシュベースの統計を使用すると、必要なスキャン回数が減少し、ディスクベースのソートが回避されるため、パフォーマンスが向上します。

ハッシュベースの統計はソートベースの統計よりも、以下の点で柔軟性に優れています。

- ハッシュベースの統計はソートベースの統計よりも実行時間が短く済むため、メンテナンス時間中により多くの作業を実行できます。
- ハッシュベースの統計では必要なプロシージャ・キャッシュが少なく済むため、メンテナンス時間外にデータオンリーロック・テーブルに対して update statistics を実行できます。tempdb バッファ・キャッシュ (通常はデフォルト・データ・キャッシュを使用) は、一般的にプロシージャ・キャッシュよりも大容量になります。
- 通常、ハッシュベースの統計では、tempdb に対して大容量のディスク割り付けは必要ありません。update statistics からの大規模なソートに対応するために tempdb のサイズを事前に増やしている場合は、この領域を再配備できます。
- ハッシュを使用した update index statistics と update all statistics は、with sampling のソートよりも実行速度が速くなる場合があります。ただし、update statistics *table_name*(*col_name*) は例外です。

ハッシュベースの統計では、ユニークなカラム値が 65536 よりも少ない場合は低い値の範囲のアルゴリズムが、65536 以上の場合は高い値の範囲のアルゴリズムがカラムに対して使用されます。2 つのアルゴリズムのうち低い値の範囲のハッシュでは、すべての範囲の値の実際の数を使用してヒストグラムが作成されるため、より正確なヒストグラムが作成されます。高い値の範囲のハッシュでは、65536 個のユニーク値の各ブロックを更新するためにメモリ内に中間ヒストグラムが生成されるため、精度がさほど高くないヒストグラムが作成されることがあります。

ハッシュベースの統計の収集は CPU への負荷が大きいため、EC3 属性を持つ実行クラスを作成して、そのクラスに update statistics ログインを割り当てることをおすすめします update statistics メンテナンス・セッションには低い優先度が設定されるので、メンテナンス時間が少ない場合や、メンテナンスできない場合の影響が軽減されます。

update statistics の実行時には次の実施をおすすめします。

- update statistics メンテナンスが必要なパーティションがアクティブなパーティションのみになるように、分割されたテーブルを使用します。
- update statistics を実行するタイミングを判断するには datachange を使用します。
- 同時実行性の問題が懸念される場合は、全ページロック・テーブルに対して update statistics を実行しないようにします (update statistics では全ページロック・テーブルでレベル 1 のページ・ロックが使用されます。その場合、データオンリーロック・テーブルでのダーティ・リードよりも同時実行性が低くなります)。

ハッシュベースの統計の有効化

Adaptive Server でハッシュベースの統計を収集できるようにするには、update statistics hashing 設定パラメータを使用します。

update statistics hashing

要約情報	
デフォルト値	partial
値の範囲	次のいずれか • off ー ハッシュを行わない • on ー すべてのカラムに対してハッシュを行う • partial ー ユニークな件数が低いカラムに対してのみハッシュを行う • default ー off
ステータス	動的
表示レベル	包括
必要な役割	システム管理者
設定グループ	一般情報

update statistics hashing を使用すると、Adaptive Server でハッシュベースの統計を収集できるようになります。

ハッシュベースの統計の収集

ハッシュベースの統計を作成するには、次の構文を使用します。

```
update index statistics
      table_name [[partition data_partition_name] |
      [ [index_name [partition index_partition_name]]]
      [using step values]
      [with consumers = consumers] [, sampling=N [percent]]
[, no_hashing | partial_hashing | hashing]
            [, max_resource_granularity = N [percent]]
            [, histogram_tuning_factor = int ]

update all statistics
      table_name [partition data_partition_name ]
      [using step values]
      [with consumers = consumers] [, sampling=N [percent]]
[, no_hashing | partial_hashing | hashing]
            [, max_resource_granularity = N [percent]]
            [, histogram_tuning_factor = int ]

update statistics
      table_name [[partition data_partition_name]
            [ (col1, col2, ⋯) | (col1), (col2), ⋯] |
```

```
    [ [index_name [partition index_partition_name]]]
    [using step values]
    [with consumers = consumers] [, sampling=N [percent]]
[, no_hashing | partial_hashing | hashing]
        [, max_resource_granularity = N [percent]]
        [, histogram_tuning_factor = int ]
```

ハッシュのレベルは [no_hashing | partial_hashing | hashing] によって決定します。

- no_hashing －バージョン 15.7 ESD #2 よりも前の Adaptive Server のソート・アルゴリズムを使用します。
- partial_hashing －ユニークな件数が低い範囲に対してハッシュを使用します。ユニークな件数が 65536 のスレッショルドを超えるカラムが見つかった場合は、ソートによる追加のスキャンが実行されます。ソートは、そのカラムで生成された前回のヒストグラムにおいて、ユニーク値の数が 65536 以上であった場合にも使用されます。

  これらのパラメータのデフォルトは、update statistics hashing で設定された値です。

『リファレンス・マニュアル：コマンド』を参照してください。

次の例では、authors テーブルでハッシュベースの統計を収集します。

```
update index statistics authors with hashing
```

update statistics コマンドでハッシュベースの統計を明示的に指定した場合は、update statistics hashing 設定パラメータの値よりも優先されます。上記の例では、update statistics ... with hashing がサーバ・レベルの update statistics hashing パラメータよりも優先されます。

ハッシュベースの統計を収集する場合、consumer および sampling パラメータは使用できません。ソートベースの統計 (consumer および sampling パラメータがサポートされます) がデフォルトで使用される前に、partial_hashing による低い値の範囲のハッシュベースの統計が試行されます。

ハッシュベースの update statistics では、カッコで囲んだカラム名のカンマ区切りリストによってカラム・セットを指定できます。

update statistics *table_name* (*column1*), (*column2), (column3*), ...

この構文によって、すべてのカラムの統計を更新する単一のテーブル・スキャンが実行されます。ただし、1 つのカッコに複数のカラムを指定してハッシュ統計を収集した場合は、エラー・メッセージが発行されます。

分布密度の設定

サーバのヒストグラムの分布密度 (ステップ数) を決定するには、histogram tuning factor 設定パラメータを使用します。

update statistics ... histogram_tuning_factor パラメータには update statistics によるヒストグラムの分布密度を指定します。これにより、偏った値を特定することができます。また、同じ重み付けの範囲セルの動作が向上します。update statistics では、範囲セルに設定されたステップ数に従うように最終的なヒストグラムが変換されます。ただし、頻度セルはそのまま保持されます。update statistics で作成される最終的なヒストグラムで重み付けに偏りが見られる場合は、同じ重み付けの範囲セルが多く作成されるようにチューニング係数を増やしてみてください。

バッファ・マネージャ・メモリの設定

ハッシュ処理によって tempdb バッファ・キャッシュ・メモリが大量に使用されることがあります。デフォルトでは、max resource granularity 設定パラメータに設定されている値が update statistics で使用されます。このパラメータには使用可能な tempdb バッファ・キャッシュの割合を設定します。

『システム管理ガイド 第 1 巻』の「設定パラメータ」を参照してください。

update statistics ... max_resource_granularity によって、使用されるバッファ・メモリの量を制限します。この値に達した場合は、メモリがリサイクルされるカラムが選択されるため、残りのカラムのハッシュを完了することができます。リソースがリサイクルされるカラムのヒストグラムは、後続のスキャンでハッシュを使用して収集されます。追加のスキャンを回避するには、必要に応じて max resource granularity の値を増やします。

第 11 章

update statistics での進行メッセージの表示

Adaptive Server バージョン 15.7 ESD #2 以降の update index statistics、update statistics、および update all statistics には、print_progress パラメータがあります。このパラメータを使用すると、これらのコマンドで進行メッセージを表示できます。

print_progress パラメータの使用

print_progress の構文は次のとおりです。

- update index statistics

  ```
  update index statistics
  table_name [[partition data_partition_name] |
  . . .
  [, print_progress = int]
  ```

- update all statistics

  ```
  update all statistics
  table_name [partition data_partition_name]
  . . .
  [, print_progress = int]
  ```

- update statistics

  ```
  update statistics
  table_name [[partition data_partition_name]
  . . .
  [, print_progress = int]
  ```

各パラメータの意味は次のとおりです。

- 0 – (デフォルト) print_progress を無効にし、進行メッセージを表示しません。
- 1 – print_progress を有効にし、進行メッセージを表示します。

次の例では、テーブル bigtable に対して update statistics を実行した場合の進行メッセージを示します。

```
update statistics bigtable with print_progress=1
Update Statistics STARTED.
Update Statistics index scan started on index 'bigtable_NC1'.
Update Statistics table scan started on table 'bigtable' for summary statistics.
Update Statistics FINISHED.
```

次の例では、テーブル bigtable に対して update index statistics を実行した場合の進行メッセージを示します。

```
update index statistics bigtable with partial_hashing, print_progress=1
Update Statistics STARTED.
Update Statistics index scan started on index 'bigtable_NC1'.
...It is using existing index scan to hash minor column 'a2' (column id = 2).
...Column 'a2' (column id = 2) is moved from hashing to sorting.
Update Statistics table scan started on table 'bigtable' for summary statistics.
Update Statistics table scan started on table 'bigtable'.
...Sorting started for column 'a2' (column id = 2).
Update Statistics FINISHED.
```

次の例では、テーブル bigtable に対して update statistics ... with hashing を実行した場合の進行メッセージを示します。

```
update statistics bigtable (a1), (a2), (a3) with hashing, print_progress=1
Update Statistics STARTED.
Update Statistics table scan started on table 'bigtable'.
...Column 'a3' (column id = 3) is picked as hash victim due to limited resource.
Update Statistics table scan started on table 'bigtable'.
```

第 12 章　dump および load の機能強化

Adaptive Server 15.7 ESD #2 では、dump および load コマンドの機能が強化されています。

- dump configuration コマンドを使用すると、Adaptive Server の設定ファイル、ダンプ履歴ファイル、およびクラスタ設定ファイルをバックアップできます。『リファレンス・マニュアル：コマンド』を参照してください。
- Adaptive Server 15.7 ESD #2 では、データベース・ダンプを作成するためのオプションを定義するダンプ設定が導入されています。その後、Backup Server がこの設定を使用してデータベース・ダンプを実行します。次の設定を使用できます。
  - ダンプ設定を作成、変更、リストするためのダンプ設定。この設定は dump database または dump transaction のダンプ実行時に使用します。『リファレンス・マニュアル：コマンド』を参照してください。sp_config_dump 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。
  - enforce dump configuration 設定パラメータ。ダンプ操作でダンプ設定を使用できるようにします。「設定パラメータ」(55 ページ) を参照してください。
  - 設定グループ "dump configuration"。ユーザが作成したダンプ設定を示します。「ダンプ設定の使用」(56 ページ) を参照してください。
- ダンプ履歴ー Adaptive Server 15.7 ESD #2 は、以下の機能を提供します。
  - dump database および dump transaction コマンドの履歴をダンプ履歴ファイルに保持します。このファイルは、あとで特定の時点までデータベースをリストアする際に使用できます。「ダンプ履歴ファイル」(58 ページ) を参照してください。
  - ダンプ履歴ファイルを読み込み、データベースのリストア時に必要な SQL 文のロード・シーケンスを再生成します。次のコマンドを使用します。

    load database with listonly=load_sql until_time = *datetime*

load database の拡張機能の詳細については、『リファレンス・マニュアル：コマンド』を参照してください。

- sp_dump_history を使用して、履歴レコードを消去します。

  sp_dump_history 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。

- enable dump history 設定パラメータを使用して、各ダンプ操作の最後でダンプ履歴ファイルに対してデフォルトで行われる更新を無効にします。
- dump history filename 設定パラメータを使用して、ダンプ履歴ファイルの名前を指定します。

- Adaptive Server 15.7 ESD #2 の dump with listonly コマンドは、2 つのオプションを使用できるように強化されています。次の処理を実行できます。

  - create_sql オプションを使用して、disk init、sp_cacheconfig、create database、および alter database コマンドのシーケンスをリストします。このリストはソース・データベースと同じレイアウトでターゲット・データベースを作成する場合に必要になります。
  - load_sql オプションを使用することにより、ダンプ履歴ファイルを使用して load database および load transaction コマンドのリストを生成します。このリストは特定の時点までデータベースを再移植する場合に必要になります。

  load database および load transaction の拡張機能の詳細については、『リファレンス・マニュアル：コマンド』を参照してください。

  拡張されたダンプ・ヘッダの詳細については、「ダンプ・ヘッダの機能強化」(60 ページ) を参照してください。

設定パラメータ

Adaptive Server のダンプ設定用の設定パラメータは次のとおりです。

enforce dump configuration

要約情報	
デフォルト値	0 (無効)
値の範囲	0 (無効)、1 (有効)
ステータス	動的
表示レベル	基本
必要な役割	システム管理者
設定グループ	バックアップとリカバリ

enforce dump configuration では、データベース・ダンプを実行する際にダンプ設定を使用するかどうかを指定します。

有効にした場合は、ダンプ設定を使用したダンプ操作のみが許可されます。dump コマンドでパラメータ (blocksize、compression など) を指定した場合、ダンプ設定で定義されている値は指定した値で上書きされません。

無効にした場合は、コマンド・ラインで指定したパラメータ値が使用され、ダンプ設定で定義されている値が上書きされます。

enable dump history

要約情報	
デフォルト値	0 (無効)
値の範囲	0 (無効)、1 (有効)
ステータス	動的
表示レベル	基本
必要な役割	システム管理者
設定グループ	バックアップとリカバリ

dump history update では、データベース・ダンプ操作の最後にダンプ履歴ファイルを更新するかどうかを指定します。

デフォルトでは、ダンプ履歴ファイルはデータベース・ダンプの終了後に毎回更新されます。

dump history filename

要約情報	
デフォルト値	dumphist
値の範囲	
ステータス	動的
表示レベル	基本
必要な役割	システム管理者
設定グループ	バックアップとリカバリ

dump history filename にはダンプ履歴ファイルのパスを指定します。

ダンプ設定の使用

dump configuration 設定パラメータ・グループは、ユーザが作成した次のダンプ設定を示します。

- stripe directory －ダンプ操作中にファイルがアーカイブされるディレクトリです。通常、アーカイブ・ファイルの名前には次の命名規則が使用されます。

  database_name.nump_type.date-timestamp.stripeID

- external api name －ダンプ操作に使用する外部 API (バイト・ストリーム・デバイス) の名前です。次の形式に従う必要があります。

  ```
  External API Name::Options
  ```

- number of stripes －ダンプ操作で使用するストライプ・デバイスの数です。デフォルトでは、単一のストライプ・デバイスが使用されます。

- number of retries －致命的でないエラーの場合にサーバがダンプ操作を試行する回数です (最大 5 回)。デフォルトは 0 です。

- block size －ダンプ・デバイスのブロック・サイズで、デバイスのデフォルトのブロック・サイズを上書きします。blocksize は、データベース・ページ 1 ページ分以上で、データベース・ページ・サイズの整数倍でなければなりません。
- compression level －圧縮ダンプの圧縮レベルです。デフォルトでは、圧縮は無効になっています。
- retain days －ダンプを上書きできない日数です。期限切れ前のボリュームに上書きしようとすると、Backup Server が確認を要求します。retaindays のデフォルト値は 0 で、これはダンプが上書き可能であること意味します。
- init －ボリュームの再初期化が必要かどうかを指定します。デフォルトは "noinit" です。
- verify － Backup Server でデータ・ページがアーカイブにコピーされる際に、データ・ページに対して最小限のページ・ヘッダー検査または完全なロー構造検査を実行するかどうかを指定します。グローバル・アロケーション・マップ (GAM)、オブジェクト・アロケーション・マップ (OAM)、アロケーション・ページ、インデックス、テキスト、ログ・ページについては、構造検査が実行されません。デフォルトでは、アーカイブ時にデータ・ページの検査は行われません。
- notify － Backup Server のデフォルトのメッセージ送信先です。次のいずれかを指定します。
  - client - dump コマンドを開始した端末にメッセージを送信します。
  - operator_console - Backup Server が稼動している端末にメッセージを送信します。
- remote backup server name －ダンプ操作に使用するリモート Backup Server を指定します。デフォルトは SYB_BACKUP です。

例

例 1. Adaptive Server 設定ファイルに作成された複数のダンプ設定を示します。

```
[dump configuration :dmp_cfg1]
   stripe_dir = workdmp_cfg1_dir
   ext_api = DEFAULT
   num_stripes = 5
   retry = 0
   blocksize = DEFAULT
   compression = 9
   retaindays = DEFAULT
```

```
   init = DEFAULT
   verify = DEFAULT
   backup_srv_name = DEFAULT

[dump configuration : dmp_cfg2]
   stripe_dir = workdmp_cfg2_dir
   ext_api = syb_tsm
   num_stripes = DEFAULT
   retry = 3
   blocksize = DEFAULT
   compression = DEFAULT
   retaindays = DEFAULT
   init = DEFAULT
   verify = DEFAULT
   backup_srv_name = SYB_REMOTE
```

ダンプ履歴ファイル

Adaptive Server では、dump database および dump transaction コマンドで成功および失敗したバックアップの履歴がダンプ履歴ファイルで管理されます。Adaptive Server は、ダンプ履歴ファイルを読み込んでデータベースをリストアし、続いて load database と load transaction のシーケンスを再生成します。このシーケンスはデータベースを特定の時点までリストアする際に必要になります。

各 Adaptive Server インスタンスには、すべてのデータベース・ダンプとサーバ設定ダンプに関する情報、およびダンプの成功または失敗に関する情報を含むダンプ履歴ファイルがあります。このファイルのデフォルト・ロケーションは、-m 起動パラメータで指定したロケーションです。または、-m を指定しない場合は、$SYBASE ディレクトリになります。

ダンプ履歴ファイルをバックアップするための構文は次のとおりです。*file_name* にはダンプ履歴ファイルの名前を指定します。

dump configuration with file = *dump_hist*

デフォルトのダンプ履歴ファイル名は、*dumphist* です。

ダンプ履歴ファイルの各行はダンプ・レコードを示します。データベースを複数のストライプ・デバイスにダンプすると、各ストライプ・デバイスにダンプ・レコードが作成されます。ダンプ・レコードのフィールドはタブで区切られています。

ダンプ・レコードには次の情報が含まれます。

- レコード・タイプ
- データベース ID
- データベース名
- ダンプのタイプ
- ダンプ操作で使用したストライプの総数
- リモート Backup Server 名
- 最新のダンプのシーケンス番号 (最新のダンプのタイムスタンプ)
- 前回のダンプのシーケンス番号 (前回のダンプのタイムスタンプ)
- ダンプ作成時刻
- ストライプ名
- ダンプ・サーバ名
- Adaptive Server のエラー数
- パスワードで保護された情報 (バックアップがパスワードで保護されているかどうかを示すブール値)
- 圧縮レベル
- 最大論理ページ番号 (ダンプされたデータベースの最大論理ページ番号)
- ステータス

ダンプ履歴ファイルの読み込みと書き込みは Adaptive Server が行います。Adaptive Server を起動するユーザは、ダンプ履歴ファイルに対して適切な読み取りパーミッションおよび書き込みパーミッションが必要です。

ダンプ・ヘッダの機能強化

Adaptive Server バージョン 15.7 ESD #2 以降では、データベース・デバイスに関する情報がダンプ・ヘッダに保存されます。この情報は、ダンプ・イメージの作成時に、disk init コマンド、sp_cacheconfig コマンドのシーケンスおよび disk init コマンドと、create database コマンド、および alter database コマンドのシーケンスをターゲット・データベース用に生成するためにセグメント・マップとともに使用されます。

ダンプ・ヘッダ・ブロックには、各データベース・デバイスに関する次の情報が含まれます。

- デバイス番号
- 論理名
- 実際のデバイス・サイズ
- 物理パス
- デバイス・タイプ
- データベース・デバイスのサイズ
- デバイス・オプション

インメモリ・データベース・デバイスはキャッシュを使用して設定します。このキャッシュに関する情報は、イメージを作成する際に disk init コマンドと sp_cacheconfig コマンドを生成するために必要になります。そのため、キャッシュ固有の情報はダンプ・ヘッダに保存されます。キャッシュ固有の情報には次のものがあります。

- キャッシュ ID
- キャッシュ名
- データベース・キャッシュ・サイズ
- 実際のキャッシュ・サイズ
- デバイス・タイプ
- キャッシュ・オプション

第 13 章　データ・コピーを実行しない場合のテーブルからのカラムの削除

alter table drop column に対して no datacopy パラメータを指定すると、データ・コピーを実行せずにテーブルからカラムを削除できるので、alter table drop column の実行に必要な時間が短縮されます。

構文は次のとおりです。

```
alter table [[database.][owner].table_name
      {add column_name datatype}
. . . }

      modify column_name
      drop {column_name [, column_name]...
            with exp_row_size=num_bytes
                  | transfer table [on | off]}
                  | no datacopy
```

no datacopy では、テーブルからただちにカラムが削除される代わりに、システム・テーブルが更新されます。これは、次に reog rebuild が実行されるか、別のデータ・コピー操作が実行されたときに、影響を受けたローが再フォーマットされることを示します (削除したカラム (ラージ・オブジェクトを含む) に割り付けられている領域は、reorg rebuild の次回実行時まで解放されません)。

次の例は、データ・コピーを実行せずに titles テーブルから total_sales カラムを削除します。

```
alter table titles
drop total_sales
with no datacopy
```

制限事項

次のカラムには no datacopy パラメータを使用できません。

- マテリアライズされたカラムまたは仮想計算カラム
- 暗号化カラム
- XML カラム
- IDENTITY カラム
- Java カラム
- プロキシ・テーブル
- 次のデータ型を使用したカラム
  - timestamp
  - bit
- 次のテーブルのロック・スキームは変更できません。
  - no datacopy 操作の影響を受けたテーブル
  - 前回の drop column with no datacopy 以降、reorg rebuild またはデータ・コピー操作がまだ実行されていないテーブル

  reorg rebuild を実行してからテーブルのロック・スキームを変更する必要があります。

第 14 章　最大データベース・サイズの拡張

Adaptive Server バージョン 15.7 ESD #2 以降では、論理ページ番号が符号付き整数から符号なし整数に変換されたため、データベースの最大サイズが約 64 テラバイトまで拡張されます。

バージョン 15.7 ESD #2 より前の Adaptive Server では、使用可能な最大データベース・サイズは約 32 テラバイトでした。

データベースの最大サイズは論理ページ・サイズによって異なります。

- 2K ページ・サーバ － 8TB
- 4K ページ・サーバ － 16TB
- 8K ページ・サーバ － 32TB
- 16K ページ・サーバ － 64TB

注意　Adaptive Server は、論理ページ範囲の上限で 256 個の論理ページ ID (割り付けることも使用することもできない) を予約するため、上記のサイズは、実際の使用可能領域の量よりも若干大きくなっています。このオーバヘッドにより、実際の使用可能領域の量は、記載されている各ページ・サイズの論理ページ・サイズに 256 をかけた数値分少なくなる (たとえば、2K サーバの実際の使用可能サイズは 8TB － (256 x 2K))。

データベースの最大サイズを拡張するには、次のカラムのデータ型を int から unsigned int に変更する必要があります。

- sysusages － lstart、size、および unreservedpgs
- sysaltusages － lstart および size
- syspartitions － firstpage、rootpage、dataoampage、および indoampage
- systabstats － leafcnt、pagecnt、emptypgcnt、warmcachepgcnt、unusedcnt、および oampgct
- syslocks － page

- syslogshold — page
- systhresholds — free_space

注意　上記のカラムに基づくクエリがある場合は、予想される結果を int から unsigned int に変更する必要があります。

以下の関数は、int の代わりに unsigned int の結果を返すようになりました。

- curunreservedpgs
- used_pages
- data_pages
- reserved_pages
- lct_admin

第 15 章　ユーザ定義の最適化目標

Adaptive Server バージョン 15.7 ESD #2 以降では、ユーザ定義の最適化目標 (すべてのアクティブなオプティマイザ基準) を作成できます。ユーザ定義の最適化目標は、以下の機能を提供します。

- オプティマイザの新しい目標の作成
- 目標に含める一連のアクティブな基準の定義
- サーバ・レベル、セッション・レベル、プロシージャ・レベル、およびクエリ・レベルでの目標のアクティブ化
- 目標内容の動的な変更 (クライアント・セッションの切断および再接続は不要)

ユーザ定義の最適化目標を作成したら、その目標をサーバ・レベルで、またはユーザ・セッションに対して呼び出すことができます。

ユーザ定義の最適化目標の作成

ユーザ定義の最適化目標を作成するには、sp_optgoal を使用します。構文は次のとおりです。

```
sp_optgoal "goal_name", "save"
```

各パラメータの意味は次のとおりです。

- *goal_name* －作成する目標の名前を 12 文字以内で指定します。
- save －その目標がまだ存在しない場合は作成します。

『リファレンス・マニュアル：プロシージャ』を参照してください。

例

次の例では、goal_1571 という目標を作成します。内容は次のとおりです。

1　最適化レベルを ase157ga に設定します。

2　最適化目標を allrows_mix に設定します。

3　ハッシュ・ジョインを有効にします。

4 CR # 123456 の最適化基準を有効にします。

5 CR # 234234 の最適化基準を無効にします。

```
set plan optlevel ase157ga
set plan optgoal allrows_mix
set hash_join 1
set CR123456 1
set CR234234 0
go
execute sp_optgoal "goal_1571", "save"
go
```

サーバ全体およびセッションに対する目標の設定

目標をサーバ全体に適用するには、sp_configure 'optimization goal' パラメータを使用します。構文は次のとおりです。

```
sp_configure 'optimization goal',1,'goal_name'
```

たとえば、サーバに goal_1571 を設定するには、次のように入力します。

```
sp_configure 'goal',1,'goal_1571'
```

現在のセッションまたはサーバ全体に目標を設定するには、set を使用します。構文は次のとおりです。

```
set plan optgoal goal_name
```

たとえば、現在のセッションに goal_1571 を設定するには、次のように入力します。

```
set plan optgoal goal_1571
```

次の例では、抽象プランによってクエリ・レベルで *goal_name* を使用します。

```
select count(*) from tab1,tab2
PLAN '(use optgoal goal_1571)'
go
```

目標の表示

sp_optgoal 'show','goal_name' は、*goal_name* という名前の目標によってアクティブ化されている個々の基準をすべて表示します。次に例を示します。

```
sp_optgoal 'goal_1571', 'show'
```

sp_optgoal @@optgoal, 'show' は、現在の目標の設定を表示します。

```
sp_optgoal @@optgoal, 'show'
name
------------------------------------------------------------------------------
distinct_sorted
distinct_sorting
distinct_hashing
group_sorted
group_hashing
nl_join
merge_join
append_union_all
merge_union_all
merge_union_distinct
hash_union_distinct
opportunistic_distinct_view
parallel_query
order_sorting
store_index
replicated_partition
ndex_union
streaming_sort
nary_nl_join
alternative_greedy_search
cr562947:OPTLEVEL EXCEPTION SEE CR - allow cursor table scans
data_page_prefetch_costing:clustered row bias added
mru_buffer_costing:wash size buffer limit for MRU
cr546125:implicitly updatable cursor non-unique index scan
cr545771:improves multi-table outer-join and semi-join costing
cr545653:avoid inner table buffer estimate starvation
cr545585:covered iscan CPU costing too expensive
cr545379:disallow reformatting on user forced index scan
cr545180:avoid reformat with no sargs if useful index exists
cr545059:reduce usage of buffer manager optimization sorts
cr544485:mark subquery join predicates with distinct view as sargs
cr534175:compute GROUP BY worktables in nested subqueries only once when possi
ble
cr531199:increases the number of useful nested loop join plans considered
```

```
cr500736:supports nocase sortorder columns in mergejoin and hashjoin keys
cr487450:improves DISTINCT costing of multi-table outer joins andor semi-joins
cr467566:allow abstract plans and statement cache to work together
cr497066:infer the nullability of isnull() by looking at its parameters
cr421607:support NULL=NULL merge and hash join keys
cr552795:eliminate duplicate rows during reformatting when they're not needed
imdb_costing:0 PIO costing for scans for in-memory database
allow_minmax:allow local session to consider MINMAX optimization
cr646220:enable better store index key generation with correlated predicate
```

第 16 章　共有クエリ・プラン

Adaptive Server バージョン 15.7 ESD #2 以降では、クエリ・プランを共有することができます。これにより、Adaptive Server で既存のプランと同じクエリ・プランを作成したり再コンパイルしたりする必要がなくなります。共有クエリ・プランは、同時実行システムのプライマリ・クエリ・プランから作成されたクローンです。

Adaptive Server で共有クエリ・プランを有効にするには、sp_configure を使用します。構文は次のとおりです。

```
sp_configure 'enable plan sharing', 1
```

enable plan sharing は enable functionality group に含まれます。このグループに含まれるパラメータの設定方法については、『システム管理ガイド 第 1 巻』を参照してください。

Adaptive Server はクエリ・プランを再使用したり再コンパイルする代わりに共有するため、パフォーマンスが向上します。Adaptive Server が共有クエリを使用している間はプライマリ・クエリ・プランがキャッシュに固定されるため、プロシージャ・キャッシュ・メモリの使用量に若干の変化が見られる場合があります。

以下の条件を満たすクエリ・プランの共有が可能です。

- 次のいずれかに該当するライトウェイト・プロシージャ (LWP) であること。
  - 動的 SQL (ODBCJDBC の prepared 文として使用されることが多い)
  - ステートメント・キャッシュにおける処理 (一致する文は再利用のために LWP に変換される) の結果、生成されたもの
- Lava クエリ・プランであること。
- instead-of-tiggers を含まないこと。

- 含めることができるのは以下のものに限られます。
  - declare、insert、delete、update、merge、または select 文
  - 単一の文。ただし、最初の文が declare 文でなければ、クエリ・プランに 2 つの文を含めることができます。
  - 逐次クエリ・プラン
  - Java オブジェクトを参照するプランにはアクセスできません。

show_cached_plan_in_xml 関数によって、<planSharing> 要素の下にプランの共有に関する情報が出力されます。

- `shareable` ー プランを共有できます。
- `notShareable` ー プランを共有できません。
- `primary` ープライマリ・プラン (このプランのクローンが共有されています)。
- `shared` ー共有プラン (プライマリ・プランのクローン)。

第 17 章　データベースの非同期的な初期化

alter database コマンドおよび create database コマンドで async_init パラメータを使用すると、データベースの使用中にデータベースを非同期に初期化できます。つまり、データベースは、初期化が完了した時点ではなく、作成または変更された時点ですぐに使用可能になります。初期化はユーザに対して透過的に行われます。

初期化されていないデータベースのページを使用するタスクは、そのページが存在するアロケーション・ユニットを初期化します。

非同期的な初期化は、create database または alter database コマンドで起動されるサービス・タスクによって実行されます。Adaptive Server を再起動すると、自動的に新しいサービス・タスクが起動され、初期化が完了します。クラスタ環境では、サービス・タスクを実行しているインスタンスが障害を起こしたり、停止したりした場合は、コーディネーティング・インスタンスによって新しいサービス・タスクが起動され、初期化が完了します。

Adaptive Server でデータベースの非同期的な作成または変更を有効にするための設定

Adaptive Server でデータベースの非同期的な作成または変更を行うかどうかは、enable async database init 設定パラメータで指定します。

enable async database init

要約	
デフォルト値	0 (オフ)
有効な値	0 (オフ)、1 (オン)
ステータス	動的
表示レベル	包括

要約	
必要な役割	システム管理者
設定グループ	SQL Server 管理

enable async database init によって、すべての create database コマンドと alter database コマンドでデータベースの非同期的な初期化がデフォルトで行われるようになります。

データベースの非同期的な作成または変更

データベースを非同期に作成する構文は次のとおりです。

```
create [temporary] database database_name
    [on {default | database_device} [= size]
. . .
    [with {override
    | default_location = "pathname" [,[no]async_init] }
    [for {load | proxy_update}]
```

noasync_init は、データベースが同期的に初期化されることを示します。

データベースを非同期に変更する構文は次のとおりです。

```
alter database database_name
    [on {default | database_device } [= size]
. . .
    [with override [,[no]async_init]]
    [for load]
    [for proxy_update]
```

noasync_init は、データベースを拡張し、拡張された領域が同期的に初期化されることを示します。

create または alter database で [no]async_init パラメータを使用すると、enable async database init の設定が上書きされます。

初期化されていない領域の確認

Adaptive Server では、データ・セグメントおよびログ・セグメントの一部が同時に初期化されてからデータベースが使用可能になるため、イニシャライザが、データベースの領域を必要とするコマンドよりも先に処理可能になります。ただし、領域の初期化中に Adaptive Server がビジー状態になると、データベースに対して通常どおりコマンドを実行してもパフォーマンスが低下する可能性があります。これは、コマンドがまだ初期化されていない領域を使用する場合に、処理を続行するためにその領域を初期化する必要があるために発生します。

初期化された領域の情報は、sysattributes に保存されます。

まだ初期化されていない領域がデータベースに存在する (たとえば、イニシャライザの処理が中断され、データベースに初期化されていない部分が残っている場合など) かどうかを確認するには、次のようなクエリを発行します。

```
select lstart=object_info1, size=object_info2, segmap=object_info3
from master..sysattributes where class=42 and object=db_id("mydb")
lstart           size             segmap
 -----------     -----------  -----------
        1536         3584000            3
        5120           51200            4
```

このクエリで 1 行以上のローが返る場合、データベース (このクエリでは、mydb データベース) にはまだ初期化されていない領域が存在します。このクエリでは、非同期的な初期化のサービス・タスクが実行中であるかどうかはわかりません。該当するタスクが終了していないことのみを示します (終了している場合は、結果セットに含まれるロー数が 0 になります)。

イニシャライザが特定のデータベース (このクエリでは、test データベース) で実行中であるかどうかを確認するには、次のようなクエリを使用します。

```
select spid from sysprocesses
where dbid=db_id("test") and cmd="CRDB AUINIT"
 spid
------
    22
```

非同期的な初期化のサービス・タスクが実行中の場合は、次のメッセージがエラー・ログに書き込まれます。

データベース '*database_name*' の非同期的な初期化は完了しています。

非同期的な初期化のサービス・タスクが中断された場合は、次のメッセージがエラー・ログに書き込まれます。

```
データベース '%.*s' でデータベースの非同期的な初期化が
中断されました。初期化されていないページでは、
初回参照時にパフォーマンスが若干低下するため、
必要に応じて DBCC DBREPAIR(%.*s, async_database_init,
start) を使用して、データベースを再起動してください。
```

制限事項

- async_init パラメータを明示的に使用した場合でも、次のデータベースを非同期的に初期化することはできません。
  - すべてのシステム・データベース
  - すべてのテンポラリ・データベース、システム・データベースまたはユーザ・データベース
  - アーカイブ・データベース
  - プロキシ・データベース
  - for load オプションを指定して作成されたデータベース
- 次のコマンドは、初期化中のデータベースでは実行できません。
  - unmount database
  - alter database ... log off
- 初期化時にデータベースをシングル・ユーザ・モードに設定することができます。ただし、データベースがシングル・ユーザ・モードに設定されている間はイニシャライザが実行されません。データベースのシングル・ユーザ・モードを解除すると、データベースが自動的に再起動され、初期化が継続されます。

注意　非同期的な初期化のサービス・タスクが実行されている際に、初期化中のデータベースの領域を使用する DML を実行すると、パフォーマンスが若干低下する可能性があります。

第 18 章　ロー内ラージ・オブジェクトの圧縮

Adaptive Server バージョン 15.7 ESD #2 以降では、ロー内ラージ・オブジェクト (LOB) の圧縮がサポートされます。『圧縮ユーザーズ・ガイド』を参照してください。

Adaptive Server では、以下の場合にロー内 LOB の圧縮が使用されます。

- テーブルが暗黙的または明示的にロー圧縮またはページ圧縮されている。
- テーブルのすべてのロー内ラージ・オブジェクト・カラムが暗黙的または明示的に LOB 圧縮されている。

第 19 章

共有メモリ・ダンプの設定

memory dump compression level 設定パラメータを使用して、共有メモリ・ダンプ・ファイルを圧縮するように Adaptive Server を設定します。

共有メモリ・ダンプの圧縮を使用するための Adaptive Server の設定

memory dump compression level を有効にすると、Adaptive Server で生成される共有メモリ・ダンプ・ファイルのサイズを大幅に削減できます。

memory dump compression level

要約情報	
デフォルト値	0
有効な値	1 － 9
ステータス	動的
表示レベル	包括
必要な役割	システム管理者
設定グループ	診断

memory dump compression level は、共有メモリ・ダンプの圧縮レベルを制御します。圧縮レベルの範囲は、0 (圧縮なし) から 9 (最高レベルの圧縮率) です。圧縮速度はダンプの圧縮量に反比例します。圧縮レベルが低いほどダンプの圧縮速度は速くなりますが、圧縮ファイルのサイズが大きくなる可能性があります。

共有メモリ・ダンプの設定

共有メモリ・ダンプを設定するには、sp_shmdumpconfig を使用します。構文は次のとおりです。

sp_shmdumpconfig "*action*", *type*, *value*, *max_dumps*, *dump_dir*, *dump_file*, *option1*, *option2*, *option3*, *option4*, *option5*

action パラメータには、ダンプの処理方法を指定します。『リファレンス・マニュアル：コマンド』を参照してください。

注意 共有メモリ・ダンプ・ファイルは、Sybase Customer Support での Adaptive Server の問題分析を支援するために作成されます。sp_shmdumpconfig は、Sybase Customer Support から指示があった場合にのみ使用してください。

次の例では、パラメータを指定せずに sp_shmdumpconfig を発行し、共有メモリ・ダンプの現在の設定を表示します。

```
sp_shmdumpconfig
Configured Shared Memory Dump Conditions
----------------------------------------

   Defaults   ---
     Maximum Dumps:          1
     Halt Engines:           Halt
     Cluster:                Local
     Page Cache:             Omit
     Procedure Cache:        Include
     Unused Space:           Omit
     Dump Directory:         $SYBASE
     Dump File Name:         Generated File Name
     Estimated File Size:    100 MB

Current number of conditions:0
Maximum number of conditions:10

Configurable Shared Memory Dump Configuration Settings
------------------------------------------------------
Dump on conditions:1
Number of dump threads:1
Include errorlog in dump file:1
Merge parallel files after dump:1
```

```
Server Memory Allocation
Procedure Cache  Data Caches  Server  Memory  Total Memory
---------------  -----------  -------------  ------------
          16 MB         9 MB           85 MB        108 MB
```

次の例では、シグナル 11 (セグメンテーション・フォールト) が発生した際には共有メモリ・ダンプを実行するように Adaptive Server を設定します。

```
sp_shmdumpconfig "add", signal, 11,1,"dump_dir"
```