SYBASE®
An SAP® Company

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

Adaptive Server® Enterprise

15.7

ドキュメント ID：DC01068-01-1570-01
改訂：2011 年 9 月

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

目次

第 1 章

set statistics コマンドの使用

set コマンドには、パフォーマンス統計の表示オプションがある statistics パラメータが含まれています。すべての統計オプションのデフォルト設定は off です。

set コマンドの構文

set statistics の構文は、次のとおりです。

```
set statistics {io, simulate, subquerycache, time, plancost} [on | off]
```

次のような単一のパラメータを発行できます。

```
set statistics io on
```

カンマで区切って、次のように 1 行内で複数のコマンドを組み合わせることができます。

```
set statistics io, time on
```

シミュレートされた統計値の使用

optdiag ユーティリティでは、シミュレートされた統計値をロードし、これらの統計値を使用してクエリの診断を実行できます。空のテーブルに対してもシミュレートされた統計値をロードできるため、これを使用して、テーブルとインデックスしか含まれない小さいデータベースでチューニング診断を実行できます。シミュレートされた統計値は、ロード時に既存の統計値を上書きしません。

シミュレートされた統計値をロードした後は、次のように入力して、実際の統計値ではなくこの統計値を使用するようにオプティマイザに指示してください。

```
set statistics simulate on
```

「シミュレートされた統計値の使用」(42 ページ) を参照してください。

サブクエリのキャッシュ・パフォーマンスのチェック

サブクエリがフラット化または実体化されていない場合には、サブクエリのキャッシュが作成され、以前のサブクエリの実行結果が格納されます。これによって、コストのかかるサブクエリの実行回数が減少します。

コンパイル時間と実行時間のチェック

set statistics time には、Adaptive Server コマンドの解析と実行に要する時間の情報が示されます。

```
解析およびコンパイル時間 57.
Adaptive Server cpu time: 5700 ms.

実行時間 175
Adaptive Server cpu time: 17500 ms.  Adaptive Server elapsed time: 70973 ms.
```

この出力の意味は次のとおりです。

- Parse and Compile Time (解析およびコンパイル時間) － クエリを解析、最適化、コンパイルするのにかかった CPU チック数。チックのミリ秒への換算については、後述する。
- Adaptive Server® cpu time (Adaptive Server CPU 時間) － CPU 時間 (ミリ秒)。
- Execution Time (実行時間) － クエリを実行するのにかかった CPU チック数。
- Adaptive Server cpu time (Adaptive Server CPU 時間) － クエリを実行するのにかかった CPU チック数をミリ秒に換算した値。
- Adaptive Server elapsed time (Adaptive Server 経過時間) － コマンドが開始した時刻と現在の時刻との差 (ミリ秒)。時刻は、オペレーティング・システムのクロックから取得する。

この出力は、57 クロック・チックでクエリの解析とコンパイルが行われたことを示しています。実行に 175 チック、つまり 17.5 秒 の CPU 時間がかかっています。合計経過時間は 70.973 秒であり、Adaptive Server がほかのタスクを処理するため、またはディスクまたはネットワークの I/O の完了を待つために若干時間を費やしたことを示しています。

注意　クエリが複雑で実行に時間がかかる場合は、プランが最適ではないために Execution Time が高くなっていないか、また最適化パフォーマンスのために Parse and Compile Time が高くなっていないかを確認してください。

チックからミリ秒への換算

チックからミリ秒に換算するには、次の式を使用します。

$$\text{ミリ秒} = \frac{\text{CPU\_ticks*clock\_rate}}{1000}$$

システムの *clock_rate* を確認するには、次のコマンドを実行します。

```
sp_configure "sql server clock tick length"
```

『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。

物理 I/O と論理 I/O の統計値のレポート

set statistics io は、物理 I/O と論理 I/O に関する情報、およびテーブルがアクセスされた回数をレポートします。set statistics io の結果は、クエリの結果に続いて出力され、クエリによって実行された実 I/O の情報を提供します。

statistics io は、クエリの各テーブル (ワーク・テーブルを含む) に対して、クエリによって読み込まれたページに対する複数の値からなる 1 行の情報と、書き込みの合計回数を知らせる 1 つのローをレポートします。システム管理者がリソース制限を有効にした場合は、statistics io にはクエリの実 I/O コストの合計をレポートする行も含まれます。次の例は、リソースの上下限値を有効にした場合のクエリに対する statistics io 出力を示します。

```
    select avg(total_sales)
    from titles
テーブル：titles、スキャン・カウント 1、論理読み込み：(regular=656 apf=0 total=656)、
物理読み込み：(regular=444 apf=212 total=656)、apf IOs used=212
このコマンドの実 I/O コストの合計：13120
このコマンドの書き込み合計：0
```

以降の項では、statistics io 出力の主要な次の 4 つの要素について説明します。

- 実 I/O コスト
- 書き込み回数の合計
- 読み込みの統計値
- テーブル名と「スキャン・カウント」

実 I/O コストの合計値

リソースの制限が有効になっている場合、statistics io は、「Total actual I/O cost」の行を出力します。Adaptive Server は、実 I/O の合計を単位なしの数値としてレポートします。クエリのコストを求める式は、次のとおりです。

コスト = 全物理 IO X 25 + **全論理** IO X 2 + CPU **コスト** X 0.1

この式は、論理 I/O の「コスト」に論理 I/O の数を乗算したものと、物理 I/O の「コスト」に物理 I/O の数を乗算したものを加算しています。

次に例を示します。

```
テーブル：sysmessages スキャン・カウント 1、論理読み込み：
(regular=454 apf=0 total=454)、物理読み込み：(regular=441 apf=0
total=441)、apf IOs used=10
このコマンドの実 I/O コストの合計：11934
```

つまり、441*25 + 454*2 + 10*0.1 = 11934 になります。

表 1-1 は、“regular” と “apf” の読み込みを示します。

書き込みの統計値

statistics io は、コマンドによって書き込まれたバッファ数の合計をレポートします。読み込み専用クエリによって、ページがキャッシュのウォッシュ・マーカを越えて移動し、ダーティ・ページへの書き込みが開始されると、読み込み専用クエリが書き込みをレポートすることがあります。

データを変更するクエリは、1 回の書き込み、つまりログ・ページ書き込みだけをレポートする場合があります。これは、変更されたページがデータ・キャッシュの MRU セクションに残っているためです。

読み込みの統計値

statistics io はクエリ内の、ワーク・テーブルを含む各テーブルおよびインデックスの論理読み込みと物理読み込みの数をレポートします。インデックスの I/O 回数は、テーブルの I/O 回数に含まれます。

表 1-1 は、statistics io が論理読み込みと物理読み込みについてレポートする値を示します。

表 1-1: 読み込み用 statistics io の出力

出力	説明
論理読み込み	
regular	クエリが必要とするページがキャッシュ内で見つかった回数。非同期プリフェッチ (APF) によって取り込まれなかったページだけがここでカウントされる。
apf	APF 要求によって取り込まれた要求がキャッシュ内で見つかった回数。
total	*regular* と *apf* の論理読み込み回数の合計。
物理読み込み	
regular	標準の非同期 I/O によってバッファがキャッシュに取り込まれた回数。
apf	APF によってキャッシュにバッファが入れられた回数。
total	*regular* と *apf* の物理読み込み回数の合計。
apf IOs used	APF によって入れられたバッファの数で、クエリ中に 1 つ以上のページが使用されたもの。

インデックスがある場合とない場合のサンプル出力

statistics io を使用して、インデックスがないテーブルのクエリを実行し、さらにインデックスがある同じテーブルで同じクエリを実行すると、クエリとシステム・パフォーマンスにとって、優れたインデックスがどれほど重要であるかがわかります。次に簡単なクエリの例を示します。

```
select title
from titles
where title_id = "T5652"
```

インデックスがない場合の *statistics io*

title_id のインデックスがない場合、statistics io は 2K I/O を使用して次の値をレポートします。

```
テーブル：titles スキャン・カウント 1、論理読み込み：(regular=624
apf=0 total=624)、物理読み込み：(regular=230 apf=394
total=624)、apf IOs used=394
このコマンドの実 I/O コストの合計：12480
このコマンドの書き込み合計：0
```

この出力の内容は、次のとおりです。

- このクエリは合計 624 回の論理 I/O を実行し、そのすべてが regular の論理 I/O である。
- このクエリは、624 回の物理読み込みを実行した。このうち、230 回が regular の非同期読み込みで、394 回が非同期プリフェッチ読み込みである。
- APF によって読み込まれたページすべてがクエリによって使用された。

インデックスがある場合の *statistics io*

title_id にクラスタード・インデックスがある場合、statistics io は 2K I/O を使用して同じクエリに次の値をレポートします。

```
テーブル：titles  スキャン・カウント 1、論理読み込み：(regular=3 apf=0 total=3)、
物理読み込み：(regular=3 apf=0 total=3)、apf IOs used=0
このコマンドの実 I/O コストの合計：60
このコマンドの書き込み合計：0
```

この出力の内容は、次のとおりです。

- このクエリは、3 回の論理読み込みを実行した。
- このクエリは、3 回の物理読み込みを実行した。うち 2 回はインデックス・ページの読み込みで、1 回はデータ・ページの読み込みである。

カーソルの *statistics io* 出力

カーソルを使用するクエリでは、statistics io はカーソルがオープンされてからの累積 I/O を出力します。

```
                1> open c
テーブル：titles スキャン・カウント 0、論理読み込み：(regular=0 apf=0 total=0)、物理
読み込み：(regular=0 apf=0 total=0)、apf IOs used=0
このコマンドの実 I/O コストの合計：0
このコマンドの書き込み合計：0
                1> fetch c


title_id type         price
 -------- ------------ ------------------------
 T24140   business                      201.95
テーブル：titles スキャン・カウント 1、論理読み込み：(regular=3 apf=0 total=3)、物理
読み込み：(regular=0 apf=0 total=0)、apf IOs used=0
このコマンドの実 I/O コストの合計：6
このコマンドの書き込み合計：0
                1> fetch c

title_id type         price
 -------- ------------ ------------------------
```

```
 T24226   business                        201.95
テーブル：titles スキャン・カウント 1、論理読み込み：(regular=4 apf=0 total=4)、物理
読み込み: (regular=0 apf=0 total=0)、apf IOs used=0
このコマンドの実 I/O コストの合計：8
このコマンドの書き込み合計：0
```

スキャン数

statistics io はクエリが特定のテーブルにアクセスした回数をレポートします。1 回の「スキャン」は、以下のアクセス・メソッドのいずれかが 1 回実行されたことを意味します。

- テーブル・スキャン。
- クラスタード・インデックスを介したアクセス。クエリがインデックスのルート・ページから開始され、データ・ページを示すポインタに従うごとに、1 回のスキャンとしてカウントされる。
- ノンクラスタード・インデックスを介したアクセス。クエリがインデックスのルート・ページから開始され、インデックスのリーフ・レベル (クエリがカバーされている場合) またはデータ・ページを示すポインタに従うごとにカウントされる。
- クエリが並列処理されている場合は、テーブルへの各ワーカー・プロセスのアクセスが 1 回のスキャンとしてカウントされる。

『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 2 章 showplan の使用」を参照してください。

スキャン数 1 がレポートされるクエリの例

スキャン数 1 を返すクエリの例は、次のとおりです。

- 次のようなポイント・クエリ

```
select title_id
from titles
    where title_id = "T55522"
```

- 次のような範囲クエリ

```
select au_lname, au_fname
    from authors
    where au_lname > "Smith"
    and au_lname < "Smythe"
```

これらのクエリの where 句内のカラムにインデックスが設定されている場合は、クエリはインデックスを使ってテーブルをスキャンできます。設定されていない場合は、テーブル・スキャンを実行します。どちらの場合も、テーブルを 1 回だけ調べて必要なローを返します。

1 より大きいスキャン数がレポートされるクエリの例

1 より大きいスキャン数を返すクエリの例は、次のとおりです。

- ワーカー・プロセスごとにスキャン数をレポートする並列クエリ。
- or で接続された where 句にインデックスを設定しているクエリは、or 句ごとにスキャンをレポートする。クエリが特殊な or 方式を使用する場合、値ごとに 1 スキャンをレポートする。クエリが or 方式を使用する場合、インデックスごとに 1 スキャン、さらに RID リストへのアクセスに 1 スキャンをレポートする。

  次のクエリは、titles テーブルの title_id と pub_id にインデックスがある場合、スキャン数を 2 とレポートします。

```
select title_id
from titles
    where title_id = "T55522"
    or pub_id  = "P988"
```

```
テーブル：titles スキャン・カウント 2、論理読み込み：(regular=149 apf=0 total=149)、
物理読み込み：(regular=63 apf=80 total=143)、apf IOs used=80
テーブル：Worktable1  スキャン・カウント 1、論理読み込み：(regular=172 apf=0
total=172)、物理読み込み：(regular=0 apf=0 total=0)、apf IOs
```

  ワーク・テーブルの I/O もレポートされます。

- 外部テーブルの条件を満たすローごとに、内部テーブルを 1 回スキャンするネストループ・ジョイン。次の例では、外部テーブル publishers 内に州の値が“NY”である publishers が 3 つあるため、内部テーブル titles はスキャン・カウントとして 3 をレポートする。

```
select title_id
from titles t, publishers p
where t.pub_id = p.pub_id
    and p.state = "NY"
```

```
テーブル：titles スキャン・カウント 3、論理読み込み：(regular=442 apf=0 total=442)、
物理読み込み：(regular=53 apf=289 total=342)、apf IOs used=289
テーブル：publishers スキャン・カウント 1、論理読み込み：(regular=2 apf=0 total=2)、
物理読み込み：(regular=2 apf=0 total=2)、apf IOs used=0
```

  このクエリは、publishers に対してテーブル・スキャンを実行します。publishers は 2 データ・ページしか占有しないため、物理 I/O が 2 回とレポートされています。publishers には 3 つの一致するローがあるため、クエリは pub_id のインデックスを使用して titles を 3 回スキャンします。

- 外部テーブルに重複する値を持つマージ・ジョインは、重複する値ごとにスキャンを再度開始し、そのたびに追加スキャンの回数をレポートする。

スキャン数 0 がレポートされるクエリの例

複数のステップからなるクエリと、その他の特定タイプのクエリは、スキャン数 0 をレポートする場合があります。次に例を示します。

- 遅延更新を実行するクエリ
- select...into クエリ
- ワーク・テーブルを作成するクエリ

物理読み込みと論理読み込みとの関係

1 つのページをディスクから読み込む必要がある場合は、1 回の物理読み込みと 1 回の論理読み込みの両方としてカウントされます。論理 I/O は常に物理 I/O と同じか、それより大きくなります。

論理 I/O は、常に 2K データ・ページをレポートします。物理読み込みと物理書き込みは、バッファ・サイズ単位でレポートされます。1 回の I/O オペレーションで読み込まれる複数ページは、1 つのユニットとして扱われます。つまり、1 つのバッファとして、読み込み、書き込み、キャッシュ内の移動が行われます。

論理読み込み、物理読み込み、2K の I/O

2K の I/O では、あるクエリ用にキャッシュ内でページが見つけられる回数は、論理読み込み回数から物理読み込み回数を減算した結果となります。論理読み込みと物理読み込みの合計数が 1 つのテーブル・スキャンに対して同じである場合は、各ページがディスクから読み込まれ、そのクエリ中に 1 回だけアクセスされたことを意味します。

クエリの対象ページがキャッシュ内で見つかった場合、論理読み込み数は物理読み込み数よりも多くなります。この現象は、インデックスの上位レベルのページで頻繁に発生します。これは、これらが何度も使用され、キャッシュ内に残りやすいためです。

物理読み込みと大容量 I/O

物理読み込みはページ単位ではなく、バッファ単位、つまり Adaptive Server がディスクにアクセスする実回数としてレポートされます。

- クエリが (showplan で確認したとおりに) 16K の I/O を使う場合は、1 回の物理読み込みで 8 データ・ページがキャッシュに読み込まれる。
- クエリが 100 の 16K 物理読み込みをレポートする場合は、クエリが 800 データ・ページ読み込んだことを意味する。
- クエリがこれらのデータ・ページをそれぞれスキャンする必要がある場合、800 の論理読み込みがレポートされる。

- 他の I/O がキャッシュからページをフラッシュしたために、ジョインなどのクエリがページを何度か読み込まなければならない場合は、物理読み込み 1 つ 1 つがカウントされる。

ワーク・テーブルに対する読み込みと書き込み

クエリ用に作成されるワーク・テーブルに対する読み込み回数と書き込み回数もレポートされます。1 つのクエリが複数のワーク・テーブルを作成する場合は、statistics io 出力内でのワーク・テーブル番号は、showplan 出力で使用されるワーク・テーブル番号と対応します。

読み込みにキャッシュを使うことの効果

クエリをテストし、クエリの I/O を調べるときに、同一のクエリを繰り返して実行すると、特にクエリが LRU (最も長い間使用されていない) 置換方式を採用している場合は、物理読み込み値が驚くような数字になることがあります。

最初に実行したクエリは物理読み込みの数が大きく、2 番目に実行したクエリは物理読み込みの数が 0 になるからです。

最初にクエリを実行するときは、すべてのデータ・ページがキャッシュ内に読み込まれ、ほかのサーバの処理がこれらのページをキャッシュからフラッシュするまでキャッシュ内にとどまります。クエリが使うキャッシュ方式に応じて、ページがキャッシュ内に残る期間が次のように変わります。

- クエリが使い捨て (MRU) キャッシュ方式を使用する場合は、ページはキャッシュ内のウォッシュ・マーカの位置に読み込まれる。

  キャッシュの容量が小さかったり、キャッシュへのアクセス頻度が高い場合は、キャッシュ内のウォッシュ・マーカに読み込まれたページはすぐにフラッシュされる。

- クエリが LRU キャッシュ方式を使用して、ページ・チェーンの MRU 側の一番前にページを読み込む場合は、ページがキャッシュにとどまる時間が使い捨て (MRU) キャッシュ方式の場合よりも長くなる。

運用システムで実際に使用しているときは、クエリに必要なページの一部がキャッシュ内で見つかることが予想されます。これは、ほかのユーザの以前のアクセスによって残されたものです。キャッシュ内にないページは、ディスクから読み込む必要があります。インデックスの上位レベルは頻繁に使用されるため、特にキャッシュに残る傾向があります。

すべてのページを保持するのに十分な大きさのキャッシュにバインドされたテーブルまたはインデックスの場合、一度オブジェクトがキャッシュに読み込まれた後は、物理 I/O は実行されません。

一方で、少数のユーザによる開発システムでクエリをチューニングするときに、クエリに必要な全物理 I/O を調べるために、使用されたページをキャッシュからクリアする必要がある場合があります。オブジェクトのページをキャッシュからクリアするには、次のようにします。

- オブジェクトのキャッシュのバインドを変更する。
  - テーブルまたはインデックスがキャッシュにバインドされている場合、バインド解除してから再度バインドする。
  - テーブルまたはインデックスがキャッシュにバインドされていない場合、使用可能な任意のキャッシュにバインドしてからバインド解除する。

  このオプションを使用するには、ユーザ定義キャッシュが少なくとも 1 つ必要です。
- ユーザ定義キャッシュがない場合は、他のテーブルで十分な数のクエリを実行する。これにより、対象のオブジェクトがキャッシュからフラッシュされる。キャッシュが非常に大きい場合は、時間がかかることがある。
- サーバを再起動する。

テスト方法とキャッシュ・パフォーマンスの詳細については、『パフォーマンス&チューニング・シリーズ：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。

statistics io とマージ・ジョイン

statistics io 出力は、マージ・ジョインのソート・コストを含みません。allow resource limits を有効にしている場合、ソートのコストは“Total estimated I/O cost”と“Total actual I/O cost”の統計にはレポートされません。

set statistics plancost を使用したクエリの解析

set statistics plancost を使うと、クエリ解析が容易になります。論理 I/O、物理 I/O、各演算子で評価した実際のロー・カウントと比較したロー・カウントの推定値が表示され、CPU とソート・バッファ・コストがレポートされます。

『移行ガイド』の「第 6 章 安定性とパフォーマンスの確認」の「set statistics plancost」を参照してください。

set statistics plancost の例

set statistics plancost は、論理 I/O、物理 I/O、各演算子で評価した実際のロー・カウントと比較したロー・カウントの推定値を表示し、CPU とソート・バッファ・コストをレポートすることによって、クエリ解析を容易にします。

```
set statistics plancost on
```

plancost を使用して、クエリ・プランの推定コストと実際のコストを比較します。plancost は、クエリ・パフォーマンスに関する問題を診断する際にも役立つことがあります。表 1-2 は、plancost の実際の出力と推定出力を示します。

表 1-2: 推定コストと実際のコスト

演算子ごとに plancost が示す値	操作の対象
el: ー 推定値 l: ー 実際の値	論理 I/O
ep: ー 推定値 p: ー 実際の値	物理 I/O
er: ー 推定値 r: ー 実際の値	ロー・カウント
cpu: ー 実際の値	CPU 数

ソートまたはハッシュベースの操作を実行する実行演算子は、操作に使用されたプライベート・バッファの数をレポートします (“bufct:”、以下の例には示されていません)。すべての数量がすべての演算子に適用されるとは限らないため、plancost は、コストのサブセットを表示することがあります。サブセット・コストを使用して、オプティマイザの推定値が最適ではないクエリ・プランに対して有効かどうかを確認します。

たとえば、authors、titleauthor、titles テーブルでジョイン・クエリを実行すると、plancost は次の出力を発行します。

```
select A.au_fname, A.au_lname, T.title
from authors A, titleauthor TA, titles T
where A.au_id = TA.au_id and T.title_id = TA.title_id
=================== Lava Operator Tree ====================
                                        Emit
                                        (VA = 6)
                                        r:25 er:342
                                        cpu: 0
                                   /
                                  MergeJoin
                                  Inner Join
                                  (VA = 5)
                                  r:25 er:342
                             /                          ¥
                            Sort                        IndexScan
                            (VA = 3)                    titles_indx (T)
                            r:25 er:25                  (VA = 4)
                            l:6 el:6                    r:18 er:18
                            p:0 ep:0                    l:2 el:3
                            cpu: 0 bufct: 24            p:0 ep:3
                       /
                      MergeJoin
                      Inner Join
                      (VA = 2)
                      r:25 er:25
                 /                              ¥
```

```
IndexScan                              IndexScan
auidind (TA)                           authors_indx (A)
(VA = 0)                               (VA = 1)
r:25 er:25                             r:23 er:23
l:1 el:2                               l:1 el:2
p:0 ep:2                               p:0 ep:2
```

クエリ・オプティマイザは推定値を生成します。実際の数値は、クエリ実行の結果を表します。

この出力は、titleauthors テーブルの左下の IndexScan 演算子が推定した値 (`er:`) および実際の値 (`r:`) のロー・カウント 25 を示します。つまり、オプティマイザの推定は正しかったことになります。ただし、上記の MergeJoin (`VA = 5`) のロー・カウント推定は、クエリ・プロセッサの推定値が 342 で実際のロー・カウントは 25 であるため、間違っていることになります。

統計値を最新の状態に保つか、ヒストグラムのステップ数を増やして、クエリ・プロセッサの推定精度を高めることができます。set option show_missing_stats on を使用して、ジョイン・カラムにヒストグラムがあるかどうかを確認します。ない場合は、ヒストグラムを作成して、クエリ・プロセッサの推定精度を高めることができます。

推定ロー・カウントが 25 なのに実際のロー・カウントが 30 の場合は、クエリ・プロセッサの推定値が間違っているとは限りません。推定値と実際の値を比較する場合は、「桁違いの相違」に注意してください (上記の例の 25 と 342 の違いなど)。

クエリ・プロセッサには、テーブル名ではなく、`IndexScan` 演算子ノードのインデックス名が表示されます。演算子ノードに関連付けられたテーブルを特定するには、次の点に注意してください。

- インデックス名によって、一意にテーブルが識別されることがある。
- 相関名がクエリに含まれる場合、演算子ノード出力には相関名が含まれている。たとえば、上記の出力の `IndexScan` 演算子の“`(TA)`”については、SQL クエリの“`titleauthor TA`”を参照。

クエリ・ツリーおよび showplan 出力には“`(VA=`*n*`)`”が含まれています。*n* = 0、1、2 などで、各演算子ノードを一意に識別します。

第 2 章

統計テーブルおよび *optdiag* を使った統計の表示

この章では、統計の格納と表示のしくみについて説明します。

統計を格納するためのシステム・テーブル

テーブル、インデックス、明示的に統計を作成したインデックスなしのカラムの統計値は、すべて systabstats テーブルと sysstatistics テーブルに格納されます。

- systabstats は、テーブルまたはインデックスに関する情報をオブジェクトとして格納する。これらの情報は、クエリの処理、データ定義言語、update statistics コマンドによって更新される。
- sysstatistics には、特定のカラムの値についての情報が格納される。これらの情報は、データ定義言語と update statistics コマンドによって更新される。

「統計に対する SQL コマンドの影響」(47 ページ) を参照してください。これらを含むすべてのシステム・テーブルの詳細については、『リファレンス・マニュアル：テーブル』を参照してください。

systabstats テーブル

systabstats テーブルには、テーブルとインデックスに関する次のような基本的な統計が格納されます。

- テーブルのデータ・ページ数、あるいはインデックスのリーフ・レベル・ページ数
- テーブルにあるロー数
- インデックスの高さ
- データ・ローとリーフ・ローの平均長
- 転送されたローと削除されたローの数
- 空のページ数
- I/O コスト見積もりの精度を向上させるための統計で、クラスタ率、アロケーション・ページとエクステントを共有するページ数、およびそのオブジェクトに使用されるアロケーション・ページと OAM の数などが含まれる
- reorg コマンドが処理を再開するための停止点

systabstats には、クラスタード・インデックスごとに 1 つのロー、ノンクラスタード・インデックスごとに 1 つのロー、クラスタード・インデックスのないテーブルごとに 1 つのロー、パーティションごとに 1 つのローがあります。クラスタード・インデックス情報の格納場所は、そのテーブルのロック・スキームによって決まります。

- データオンリーロック・テーブルの場合、systabstats にクラスタード・インデックスの別のローが格納される。
- 全ページロック・テーブルの場合、そのデータ・ページはインデックスのリーフ・レベルとして取り扱われ、クラスタード・インデックスの systabstats エントリはテーブル・データと同じローに格納される。

  全ページロック・テーブルのクラスタード・インデックスの indid カラムは常に 1 になる。

sysstatistics テーブル

sysstatistics テーブルには、ユーザ・テーブルにある各インデックス・カラムの 1 つ以上のローが格納されます。また、インデックスのないカラムの統計も格納されます。

- 各カラムの最初のローには、カラムについての基本的な統計、たとえば、ジョインの密度、探索引数、いくつかの演算子の選択性、そのカラムのヒストグラムに格納されるステップ数などが格納される。
- 複数のカラムを持つインデックスの場合、あるいはインデックスのないカラムに統計を作成したときは、カラムの各プレフィクス・サブセットごとにローがある。

プレフィクス・サブセットの詳細については、「カラムの統計」(25 ページ) を参照してください。

追加のローには、先行のカラムのヒストグラム・データが格納されます。テーブルへのデータ挿入前にインデックスが作成された場合は、ヒストグラムは作成されません。ヒストグラムを生成するには、データの挿入後に update statistics を実行します。

「ヒストグラムの内容」(30 ページ) を参照してください。

optdiag ユーティリティを使った統計の表示

optdiag ユーティリティを使うと、systabstats テーブルおよび sysstatistics テーブルの統計を表示できます。optdiag は、sysstatistics 情報の更新にも使用できます。optdiag を実行できるのは、システム管理者だけです。

optdiag の構文

optdiag の構文は次のとおりです。

```
optdiag
[binary] [simulate] statistics {-i input_file | database[.owner[.table[.partition
[.column]]]]
[-o output_file]} [-U user_name] [-P password] [-I interfaces_file]
[-S server] [-v] [-h] [-s] [-T flag_value] [-z language]
[-J client_charset] [-a display_charset]
```

optdiag では、データベース全体、1 つのテーブルとそのインデックスおよびカラム、または特定のカラムの統計を表示できます。

たとえば、pubtune データベースにあるすべてのユーザ・テーブルの統計を表示してそれを *pubtune.opt* ファイルに出力するには、次のコマンドを使用します。

```
optdiag statistics pubtune -Usa -Ppasswd -o pubtune.opt
```

titles テーブル、およびそのテーブルのインデックスの統計を表示するには、次のコマンドを使用します。

```
optdiag statistics pubtune..titles -Usa -Ppasswd -o titles.opt
```

スライスされたテーブルからの optdiag の実行

バージョン 15.0 以前の Adaptive Server® のスライス・テーブルから optdiag を出力すると、データは送信先テーブルのスキーマに基づいてロードされます。

- 送信先テーブルが単一パーティションのテーブルの場合、optdiag は標準の方法を使い、[最大パーティション内のページ] フィールドを無視してテーブルをロードする。
- 送信先テーブルが複数パーティションのテーブルで、[最大パーティション内のページ] フィールドが空白の場合、optdiag は最初のパーティションに統計をロードする。
- [最大パーティション内のページ] フィールドが空白でない場合、optdiag はこのフィールドの値を使用して、[データ・ページ・カウント] フィールドと [データ・ロー・カウント] フィールドの値を均等にすべてのパーティションに分割する。optdiag は常にカラム・レベルの統計をカラムのグローバル統計にロードする。

optdiag コマンドの詳細については、『ユーティリティ・ガイド』を参照してください。以降の項に、optdiag 出力に関する情報を示します。

optdiag ヘッダ情報

optdiag を実行すると、バージョン情報およびサーバ名が出力され、統計の表示に使用されている引数がまとめられます。

optdiag のヘッダには、次のようにオブジェクトのリストがレポートされます。

```
サーバ名：              "test_server"

指定データベース：        "pubtune"
指定テーブル所有者：       指定されていません
指定テーブル：           "titles"
指定カラム：            指定されていません
```

表 2-1 は出力内容を示します。

表 2-1: テーブルとカラムに関する情報

ローの名前	指定する情報
サーバ名	*@@servername* 変数に格納されているサーバ名。この変数にサーバ名を入れるには、sp_addserver を使って、サーバを再起動する必要がある。
指定データベース	optdiag コマンド・ラインで指定したデータベース名。
指定テーブル所有者	optdiag コマンド・ラインで指定したテーブル所有者。
指定テーブル	optdiag コマンド・ラインで指定したテーブル名。
指定カラム	optdiag コマンド・ラインで指定したカラム名。

テーブル統計

これはテーブル統計の optdiag 出力例です。

```
テーブル所有者：                          "dbo"
テーブル名：                              "authors"

テーブルの統計：                          "authors"

      パーティションのカウント：          3

パーティションの統計：                    "authors_1376004902"
      データ・ページ・カウント：          74
      空のデータページカウント：          0
      データ・ロー・カウント：            1666.0000000000000000
      転送済みロー・カウント：            0.0000000000000000
      削除済みロー・カウント：            0.0000000000000000
      データ・ページ CR カウント：        10.0000000000000000
      OAM + アロケーション・ページ・カウント： 3
      最初のエクステント・データ・ページ：  0
      データ・ロー・サイズ：              85.2623049219687914
      並列ジョイン度：                    0.0000000000000000
      未使用ページのカウント：            5
      OAM ページのカウント：              1

   導出統計：
      データ・ページ・クラスタ比率：      1.0000000000000000
      領域利用：                          0.9521597490347491
      大容量 I/O 効率：                   1.0000000000000000
```

表 2-2: テーブル統計

ローの名前	指定する情報
テーブル所有者	テーブル所有者名。コマンド・ラインで *dbname..tablename* を指定すると、所有者名を省略できる。同じ名前だが所有者が異なる複数のテーブルがある場合は、optdiag は指定した名前の各テーブルについての情報を出力する。
テーブル名	テーブルの名前。
テーブルの統計	統計を出力するテーブルの名前。
パーティションのカウント	パーティション数。
パーティションの統計	統計が表示されるパーティションの名前。
データ・ページ・カウント	テーブルにあるデータ・ページ数。
空データ・ページ・カウント	削除されたローだけを持つページのカウント。
データ・ロー・カウント	テーブルにあるロー数。
転送済みロー・カウント	テーブルにある転送されたローの数。全ページロック・テーブルでは、この値は必ず 0。
削除済みロー・カウント	テーブルから削除されたロー数。これはコミットされた削除で、削除されたローをクリアする関数のいずれかによってその領域は要求されていない。 全ページロック・テーブルでは、この値は必ず 0。
データ・ページ CR カウント	データ・ページのクラスタ率を導出するカウンタ。この比率を見ると、テーブル・スキャンおよび範囲スキャンの大容量 I/O の効率を知ることができる。この値は、update statistics を実行した場合にのみ更新される。
OAM + アロケーション・ページ・カウント	テーブルの OAM ページ数と、テーブルが領域を使用するアロケーション・ユニットの数。これらの統計値は、データオンリーロック・テーブル上の OAM スキャンのコストを見積もるために使用される。 値はデータオンリーロック・テーブル上でのみ管理される。
最初のエクステント・データ・ページ	アロケーション・ページを持つアロケーション・ユニットにある最初のエクステントを共有するページ数。これらのページは、大容量 I/O ではなく 2K I/O を使って読み込む必要がある。 情報はデータオンリーロック・テーブル上でのみ管理される。
データ・ロー・サイズ	データ・ロー平均長 (バイト)。サイズにはローのオーバヘッドが含まれる。 update statistics、create index、および alter table...lock のみがこの値を更新する。
並列ジョイン度	ネストループ・ジョインで使用される並列度を示す整数値。
未使用ページのカウント	エクステント内の未使用のページの数。
OAM ページのカウント	OAM ページ数。
抽出統計	optdiag が情報を抽出する統計のグループ。
データ・ページ・クラスタ率	以下の「データ・ページ・クラスタ率」を参照。
領域の使用率	以下の「領域の使用率」を参照。
大容量 I/O の効率	以下の「大容量 I/O の効率」を参照。

テーブル・レベルで抽出される統計

「抽出統計」は、「データ・ページ CR カウント」とデータ・ページ・カウントから抽出された「データ・ページのクラスタ率」、「領域使用率」、「大容量 I/O の効率」の統計をレポートします。

データ・ページ・クラスタ率

全ページロック・テーブルでは、データ・ページ・クラスタ率は、テーブルがページ・チェーン順に読み込まれるときにエクステント上でページがどの程度うまく連続しているかを表します。クラスタ率 1.0 は、完全な連続を示します。クラスタ率が小さくなるほど、ページ・チェーンがより多く断片化していることを意味します。

データオンリーロック・テーブルでは、データ・ページ・クラスタ率はエクステント上にどの程度ページがうまくパックされているかを示します。クラスタ率 1.0 は、エクステントの完全なパックを意味します。データ・ページ・クラスタ率が小さくなると、テーブルに割り当てられているエクステント内に空のページがあることを意味します。

領域の使用率

領域の使用率は、ローの平均サイズおよびロー数を使って期待されるデータ・ページの最低数を計算し、期待される最低数を現在のページ数と比較します。領域の使用率が低いと、テーブルに対する reorg rebuild の実行、またはクラスタード・インデックスの削除と再作成により、データ・ページにある空の領域が減少し、テーブルに割り当てられているエクステントにある空のページ数が減少します。

fillfactor や reservepagegap などの領域管理プロパティを使っている場合は、クラスタード・インデックスを持つテーブルのデータ・ページにあるローの追加のための空き領域とテーブルのエクステントにある空のページ数が領域使用率の値に影響します。

長く統計の更新を行っていない場合、およびローの平均サイズが変更された場合、あるいはローまたはページの数が不正確な場合は、領域使用率が 1.0 より大きくなることがあります。

大容量 I/O の効率

「大容量 I/O の効率」は、大容量 I/O ごとに取得される有用なページ数を見積もります。たとえば、この値が 0.5 である場合、1 回の 16K I/O は、平均して、クエリに必要な 4 つの 2K ページと、空のページまたはクラスタ化が不十分なためにエクステントを共有しているページを 4 ページ返します。この値が低いときは、クラスタード・インデックスを作成し直すか、テーブルで reorg rebuild を実行すると、I/O パフォーマンスが向上する可能性があります。

インデックスの統計

このサンプル出力は、各ノンクラスタード・インデックスと、データオンリーロック・テーブルにあるクラスタード・インデックス、全ページロック・テーブルのクラスタード・インデックスに対して出力されます。全ページロック・テーブルのクラスタード・インデックスの情報は、テーブル統計のレポートに含まれます。表 2-3 (23 ページ) は出力内容を示します。

```
インデックスの統計：                    "title_id_ix" (ノンクラスタード)
インデックス・カラム・リスト：          "title_id"
    リーフ・カウント：                  45
    空のリーフ・ページ・カウント：      0
    データ・ページ CR カウント：        4952.0000000000000000
    インデックス・ページ CR カウント：  6.0000000000000000
    データ・ロー CR カウント：          4989.0000000000000000
    最初のエクステント・リーフ・ページ：0
    リーフ・ロー・サイズ：              17.8905999999999992
    インデックス・ハイト：              1


    導出統計：
        データ・ページ・クラスタ比率：  0.0075819672131148
        インデックス・ページ・クラスタ比率：1.0000000000000000
        データ・ロー・クラスタ比率      0.0026634382566586
```

注意 このサンプルに含まれていない並列ジョイン度、未使用ページのカウント、OAM ページのカウントは、クラスタード・インデックスの全ページロック・テーブルにのみ表示されます。

表 2-3: インデックスの統計

ローの名前	指定する情報
インデックスの統計	インデックス名とタイプ。
インデックス・カラム・リスト	インデックスにあるカラムのリスト。
リーフ・カウント	インデックスにあるリーフ・レベルのページ数。
空のリーフ・ページ・カウント	インデックスにある空のリーフ・ページ数。
データ・ページ CR カウント	インデックスを使っているテーブルへのアクセスに必要なデータ・ページ・クラスタ率を計算するためのカウンタ。 「インデックス・レベルで抽出される統計」(23 ページ) を参照してください。
インデックス・ページ CR カウント	インデックス・ページのクラスタ率を計算するカウンタ。 「インデックス・レベルで抽出される統計」(23 ページ) を参照してください。
データ・ロー CR カウント	データ・ローのクラスタ率を計算するためのカウンタ。 「インデックス・レベルで抽出される統計」(23 ページ) を参照してください。
最初のエクステント・リーフ・ページ	アロケーション・ユニットの最初のエクステントに格納されているインデックスにあるリーフ・ページ数。これらのページは、大容量 I/O ではなく 2K I/O を使って読み込む必要がある。 この値は、データオンリーロック・テーブルだけに保持される。
リーフ・ロー・サイズ	インデックスにあるリーフ・レベルのローの平均サイズ。update statistics、create index、および alter table...lock を実行したときのみこの値が更新される。
インデックス・ハイト	リーフ・レベルを含まないインデックスの高さ。このローがテーブル・レベルの出力に含まれるのは、全ページロック・テーブルのクラスタード・インデックスだけである。その他のインデックスでは、インデックスの高さはインデックス・レベルの出力に含まれる。 この値は、ヒープ・テーブルには適用されない。

インデックス・レベルで抽出される統計

インデックス・レベル・セクションの抽出統計は、「インデックスの統計」(22 ページ) に示した「CR カウント」の値に基づいて計算されます。

データ・ページ・クラスタ率

データ・ページ・クラスタ率は、データ・ページへのアクセスに使われているインデックスについて、大容量 I/O の効率を計測するために使われます。インデックスから見てテーブルが完全にクラスタ化されている場合は、クラスタ率は 1.0 となります。データ・ページのクラスタ率は、広い範囲で変化します。あるインデックスについては高く、別のインデックスについては非常に低いということがあり得ます。

インデックス・ページ・クラスタ比率

インデックス・ページのクラスタ率は、ノンクラスタード・インデックス、またはデータオンリーロック・テーブルにあるクラスタード・インデックスから大量のリーフ・レベルのページを読み込む必要のあるクエリが使用する大容量 I/O のコストを見積もるために使われます。このようなクエリの例としては、カバード・インデックス・スキャンや多くのローを読み込む範囲クエリが挙げられます。

新規に作成されたインデックスでは、「インデックス・ページ・クラスタ率」は 1.0 または 1.0 の近似値になります。これは、エクステントのインデックス・リーフ・ページのクラスタ化が最適であることを示しています。インデックス・ページが分割され、新しいページが追加のエクステントに割り付けられると、この率が低下します。この比率が非常に低くなった場合は、そのインデックスを削除して再作成するか、reorg rebuild を実行すると、パフォーマンスが改善されます。この作業は、特に多くのクエリがカバード・スキャンを行う場合に有効です。

データ・ロー・クラスタ率

データ・ロー・クラスタ率は、このインデックスを使ってデータ・ページにアクセスしているときに読み込む必要のあるページ数の見積もりに使われます。

インデックスの領域使用率

領域の使用率は、ローの平均サイズおよびロー数を使って期待されるリーフ・レベルのインデックス・ページの最低サイズを計算し、それを現在のリーフ・ページ数と比較します。

領域の使用率が低いと、そのインデックスに対する reorg rebuild の実行、そのインデックスの削除と再作成により、インデックス・ページにある空の領域が減少し、そのインデックスに割り当てられているエクステントにある空のページ数が減少します。

fillfactor や reservepagegap などの領域管理プロパティを使っている場合は、リーフ・ページにあるローの追加のための空き領域とインデックスのエクステントにある空のページ数が領域使用率の値に影響します。

長く統計の更新を行っていない場合、およびローの平均サイズが変更された場合、あるいはローまたはページの数が不正確な場合は、領域使用率が 1.0 より大きくなることがあります。

インデックスの大容量 I/O の効率

大容量 I/O の効率は、大容量 I/O ごとに取得される有用なページ数を見積もります。たとえば、この値が 0.5 である場合、1 回の 16K I/O は、平均して、クエリに必要な 4 つの 2K ページと、空のページまたはクラスタ化が不十分なためにエクステントを共有しているページを 4 ページ返します。

この値が低いときは、インデックスを作成し直すか、reorg rebuild を実行すると、I/O パフォーマンスが向上する可能性があります。

カラムの統計

optdiag のカラム・レベルの統計には、次の内容が含まれます。

- カラムの密度と選択性を示す統計値。インデックスが複数のカラムからなる場合、optdiag を実行すると、インデックス・キーの各プレフィクス・サブセットに関して表 2-4 に示した情報が表示される。カラム名リストに update statistics を実行して統計を作成した場合は、密度の統計はそのカラム・リストのプレフィクス・サブセットごとに格納される。
- ヒストグラム。テーブルに 1 つ以上のデータのローが含まれている場合に、インデックスの作成時または update statistics の実行時に作成される。ヒストグラムは次の要素の先行カラムに作成される。
  - 現在存在する個々のインデックス (インデックスの作成時にカラムに少なくとも 1 つの非 NULL 値があることが条件)
  - 一度作成され削除されたインデックス (delete statistics が実行されていないことが条件)
  - update statistics が実行されたカラム・リスト

  次の要素にもヒストグラムが作成される。
  - update index statistics コマンドが使用されている場合は、インデックスにある各カラム
  - update all statistics コマンドが使用されている場合は、テーブルにある各カラム

optdiag を実行すると、統計のないテーブルのカラムのリストも出力されます。たとえば、authors テーブルのこれらのカラムには統計値がありません。

```
No statistics for column(s):        "address"
(default values used)               "au_fname"
                                    "phone"
                                    "state"
                                     "zipcode"
```

カラム統計の出力例

次の例は、authors テーブルの city カラムの統計値を示します。

```
カラムの統計：                                  "au_lname"
ローカル・パーティションから統計を抽出しました。
カラム統計の最終更新：                          Apr 8 2008 9:29:07:706PM

    範囲セル密度：                              0.0005507993510047
    総密度：                                    0.0005507993510047
    範囲選択性：                                デフォルトが使用されます。(0.33)
    中間選択性：                                デフォルトが使用されます。(0.25)
    ユニークな値の範囲：                        0.0005215561314205
    ユニークな合計値：                          0.0005215561314205
    平均カラム幅：                              6.7988000000000000
```

表 2-4: カラムの統計

ローの名前	指定する情報
カラムの統計	カラム名。このブロックには、複合インデックスまたはカラム・リストにあるプレフィクス・サブセットについての情報が“Statistics for column group”ロー・ラベルの下に出力される。
カラム統計の最終更新	インデックスの作成日、update statistics の前回の実行日、または optdiag を実行して最後に統計を変更した日。
ディストリビューション・ページのアップグレードから導出された統計	11.9 より前の Adaptive Server のディストリビューション・ページのアップグレードから得られる統計。update statistics をすでにテーブルまたはインデックスに対して実行している場合、または統計のアップグレード後にインデックスの削除と再作成を行っている場合には、このメッセージは出力されない。 このメッセージが optdiag の出力に表示される場合は、update statistics の実行が必要である。
optdiag からロードされた統計	optdiag は sysstatistics 情報の変更に使用された。create index コマンドは、編集した統計が上書きされることを示す警告メッセージを出力する。 このローは、統計値が update statistics または create index によって生成された場合は表示されない。
範囲セル密度	カラムにある等価探索引数の密度。 「範囲セルと総密度の値」(28 ページ) を参照してください。
総密度	カラムのジョイン密度。この値は、カラムのジョインについて返されるロー数の推定に使われる。 「範囲セルと総密度の値」(28 ページ) を参照してください。
範囲選択性	デフォルト値の 0.33 を出力。ただし、この値が optdiag 入力モードで更新されていないことが条件。 この値は、探索引数が不明な場合に範囲クエリに使用される。 以下の「範囲および中間選択性の値」を参照。
中間選択性	デフォルト値の 0.25 を出力。ただし、この値が optdiag 入力モードで更新されていないことが条件。 この値は、探索引数が不明な場合に範囲クエリに使用される。 以下の「範囲および中間選択性の値」を参照。
ユニークな値の範囲	頻度セルを除くカラム内のユニーク値の数。
ユニークな合計値	ユニーク値の合計。
平均カラム幅	テーブルのすべてのカラム幅の平均。

範囲セルと総密度の値

Adaptive Server では、カラム値の密度に次の 2 つの値が格納されます。

- 「範囲セル密度」は、範囲セルのみの重複した値を示す。

  そのカラムに頻度セルがあると、その範囲セル密度の計算から除外される。

  そのカラムに頻度セルのみで範囲セルがないと、その範囲セル密度は 0 となる。

  範囲セルおよび頻度セルについては、「ヒストグラムの出力内容」(31 ページ) を参照。

- 「総密度」は、範囲セルと頻度セルで示される、すべてのカラムの重複値を表している。

2 つの異なる値を使うことにより、次の場合、オプティマイザが返すロー数の見積もり精度が向上します。

- 探索引数が頻度セルの値と一致する場合は、返される頻度セルのウェイトで示されるローの割合。
- 探索引数が範囲セルの範囲内であれば、範囲セル密度および範囲セルのウェイトが返されるロー数の見積もりに使用される。

ジョインの場合は、オプティマイザは、カラムにあるすべての値の平均重複値を示す総密度が最善の見積もりとなるように、テーブルのスキャンごとに返される平均ロー数に基づいて見積もります。クエリの最適化時に探索引数の値がわかっていない場合は、総密度は等価引数にも使用されます。

「範囲および中間選択性の値」(29 ページ) を参照してください。

複数のカラムのインデックスでは、範囲セル密度と総密度はプレフィクス・サブセットごとに格納されます。次に示す titles (pub_id, type, pubdate) のインデックスの出力例では、追加カラムが考慮されるごとに密度の値が減少します。

```
Statistics for column group:     "pub_id", "type"
カラム統計の最終更新：               Apr 8 2008 9:22:40:963AM

     範囲セル密度：                0.0000000362887320
     総密度：                     0.0000000362887320
   範囲選択性：                   デフォルトが使用されます。(0.33)
   中間選択性：                   デフォルトが使用されます。(0.25)
   ユニークな値の範囲：             0.0000000160149449
   ユニークな合計値：               0.0000000160149449
   平均カラム幅：                 デフォルトが使用されます。(8.00)

Statistics for column group：    "pub_id", "type", "pubdate"
カラム統計の最終更新：               Apr 8 2008 9:22:40:963AM

     範囲セル密度：                0.0000000358937986
     総密度：                     0.0000000358937986
```

```
範囲選択性：                デフォルトが使用されます。(0.33)
中間選択性：                デフォルトが使用されます。(0.25)
ユニークな値の範囲：        0.0000000158004305
ユニークな合計値：          0.0000000158004305
平均カラム幅：              2.0000000000000000
```

このテーブルのロー数は 5000 です。返されるロー数の見積もりをオプティマイザが行いますが、見積もりの精度は、クエリに使用される探索引数の数により決まります。

- pub_id のみに等価探索引数を使用すると、0.0335391029690461 * 5000 ロー、つまり 168 ローという見積もりが返される。
- 3 つのカラムすべてに等価探索引数を使用すると、0.0002011791956201 * 5000 ロー、つまり 1 ローと見積もりが返される。

より多くの探索引数を評価すると精度レベルが向上するので、多くのクエリの最適化に非常に役立ちます。

範囲および中間選択性の値

optdiag は、範囲選択性および中間選択性の値を出力します。以前の optdiag セッションでこの 2 つの選択性に値が設定されている場合は、その値を出力します。これらの値は、範囲クエリの最適化時に探索引数の値がわかっていない場合に、範囲クエリに使われます。

値が不定の等価探索引数については、総密度がデフォルト値として使用されます。

次の場合、最適化時には探索引数は不明です。

- プロシージャ内に変数を設定するストアド・プロシージャ
- バッチ内で探索引数に変数を設定するバッチ内のクエリ

これらの近似値は、クエリが返すロー数を大きく、または少なく見積もることがあるので、十分に最適と言えない結果となることがあります。

optdiag を使用して選択性の値を取得する方法については、「optdiag 入力モードを使った選択性の更新」(39 ページ) を参照してください。

ヒストグラムの内容

ヒストグラムには、カラムでの値の分布に関する情報が格納されます。表 2-5 は、ヒストグラムを作成および更新するためのコマンドとその対象となるカラムを示します。

表 2-5: ヒストグラムを作成するコマンド

コマンド	ヒストグラムのカラム
create index	先行カラムのみ
update statistics	
table_name または *index_name*	先行カラムのみ
column_list	先行カラムのみ
update index statistics	すべてのインデックス・カラム
update all statistics	すべてのカラム

ヒストグラムの出力例

```
カラムのヒストグラム：                    "city"
カラム・データ型：                        varchar(20)
要求ステップ数：                          20
実際のステップ数：                        20
サンプリング率：                          0
Out of range histogram adjustment：  DEFAULT
```

optdiag は、表 2-6 に示す、ヒストグラムの計算データを最初に出力します。

表 2-6: ヒストグラムの概要統計

ローの名前	指定する情報
カラムのヒストグラム	カラムの名前。
カラム・データ型	カラムのデータ型で、そのデータ型に適していれば長さ、精度、位取りが示される。
要求ステップ数	カラムに必要なステップ数。
実際のステップ数	そのカラムに生成されたステップ数。 カラムにある明確な値の数が要求されているステップ数よりも小さい場合は、この数は要求されたステップ数よりも小さくなることがある。
サンプリング率	ヒストグラムを作成するためにサンプリングしたテーブル・データのパーセンテージ。値の範囲は 0 ～ 100。
Out of range histogram adjustment	カラムのヒストグラムを調整して、カラムの限界外の探索引数に選択性値を割り当てるかどうかを示す。値はオン、オフ、デフォルトのうちの 1 つ。このローは、グローバル・カラム統計のヒストグラム情報と一緒に表示される。

ヒストグラム出力は、表 2-7 に示すようにカラムに出力されます。

表 2-7: *optdiag* のヒストグラム出力のカラム

カラム	指定する情報
ステップ	ステップ数。
ウェイト	ステップのウェイト。
(演算子)	<、<=、または = (値の限度を示す)。セルが範囲セルを示すか頻度セルを示すかにより演算子は異なる。このカラムには見出しは表示されない。
値	範囲セルまたは頻度セルの上限値。

ヒストグラムの出力内容

ヒストグラムとは、1 組のセルで、それぞれのセルにウェイトが割り当てられています。各セルには、上限と下限があり、どちらもそのカラムの明確な値です。セルのウェイトは、0 と 1 の間の浮動小数点値で、次のいずれかを表します。

- 演算子が <= の場合は、値が範囲内にあるテーブルのローの割合
- 演算子が = の場合は、ステップに一致した値の数

オプティマイザは、範囲、ウェイト、および密度の値の組み合わせを使って、カラムにあるクエリ句に返されるテーブルのロー数を計算します。

Adaptive Server では、各セルが表すロー数がほぼ等しくなる等高ヒストグラムが使われます。たとえば、以下に示す pubtune..authors の city カラムのヒストグラムには 20 ステップあります。個々のステップは、テーブルの約 5% を表します。

```
ステップ    ウェイト                値

   1     0.00000000     <=     "APO Miami¥377¥377¥377¥377¥377¥377¥377"
   2     0.05460000     <=     "Atlanta"
   3     0.05280000     <=     "Boston"
   4     0.05400000     <=     "Charlotte"
   5     0.05260000     <=     "Crown"
   6     0.05260000     <=     "Eddy"
   7     0.05260000     <=     "Fort Dodge"
   8     0.05260000     <=     "Groveton"
   9     0.05340000     <=     "Hyattsville"
  10     0.05260000     <=     "Kunkle"
  11     0.05260000     <=     "Luthersburg"
  12     0.05340000     <=     "Milwaukee"
  13     0.05260000     <=     "Newbern"
  14     0.05260000     <=     "Park Hill"
  15     0.05260000     <=     "Quicksburg"
  16     0.05260000     <=     "Saint David"
  17     0.05260000     <=     "Solana Beach"
  18     0.05260000     <=     "Thornwood"
  19     0.05260000     <=     "Washington"
  20     0.04800000      <=     "Zumbrota"
```

ヒストグラムの最初のステップは、テーブルにある null 値の割合を示します。city には null 値がないので、ウェイトは 0 です。null 値を表すステップの値は、最小カラム値より小さい値の中の最大の値で表されます。

文字列の場合は、最初のセルの値は最小カラム値よりも小さい最大の文字列値となります (上記の出力では、“APO Miami”)。定義されたカラム長さに満たない場合は、サーバの文字セットで使われる最大の文字が埋め込まれます。実際の出力内容は、optdiag 出力ファイルを見るときに使用する文字セット、端末の種類、およびソフトウェアによって異なります。

上記のヒストグラムでは、各セルの表す値に上限値は含まれていますが下限値は含まれていません。このヒストグラムのセルは、それぞれが値の範囲を示すので、「範囲セル」と呼ばれます。

範囲セルに含まれる値の範囲は、次のように表されます。

```
lowerbound < (values for cell) <= upper bound
```

optdiag 出力では、前のステップの値が下限値となり、現在のステップの値が上限値となります。

たとえば、上のヒストグラムでは、ステップ 4 に Charlotte (上限値) が含まれますが、Boston (下限値) は除外されます。このステップのウェイトは 0.0540 で、テーブルの 5.4% が次のクエリ句に一致することを示しています。

```
where city > Boston and city <= "Charlotte"
```

optdiag ヒストグラム出力の演算子カラムは **<=** 演算子を示しています。重複値が多いヒストグラムでは、別の演算子を使います。

重複値の多いカラムのヒストグラム

重複値の多いカラムのヒストグラムは、多くの値が別個の値であるカラムのヒストグラムとまったく違います。重複値の多いカラムのヒストグラムでは、「頻度セル」と呼ばれる 1 つのセルが重複値を表します。

頻度セルのウェイトは、一致する値を持つカラムの割合を示します。

頻度セルのヒストグラム出力は、カラム値が次のいずれを表すかによって異なります。

- 密頻度カウント。カラムにある値は、ドメイン内で連続している。たとえば、1、2、3 は連続した整数値である。
- 疎頻度カウント。可能な値のドメインには、テーブルにある個別の値セットが表す値を含まない。
- 疎頻度カウントと密頻度カウントの混合。

一部のカラムのヒストグラム出力には、頻度セルと範囲セルが混在します。

密頻度カウントのヒストグラム

次の出力は、1 ～ 6 までの個別の整数値といくつかの null 値を持つカラムのヒストグラムです。

```
ステップ    ウェイト          値

   1      0.13043478      <=      1
   2      0.04347826      <=      1
   3      0.17391305      <=      2
   4      0.30434781      <=      3
   5      0.13043478      <=      4
   6      0.17391305      <=      5
   7      0.04347826      <=      6
```

上のヒストグラムは「密頻度カウント」を示し、カラムのすべての値が連続した整数値です。

最初のセルは、null 値を表します (以降の項で説明)。null 値があるので、このセルのウェイトはカラムにある null 値の割合を示します。

最初のステップの“Value”カラムには、テーブルにある最小カラム値と < 演算子が表示されます。

疎頻度カウントのヒストグラム

疎頻度カウントを持つカラムのヒストグラムでは、重複度の高い値は = 演算子とセルのウェイトとともに個別の値を示すステップで表されます。

各ステップの前には、0.0 のウェイト、同じ値、および < 演算子を持ったステップがあり、これはテーブルにその中間の値を持ったローがないことを示します。null 値を持ったカラムでは、テーブルに null 値があると最初のステップのウェイトはゼロ以外となります。

次のヒストグラムは、titles テーブルの type カラムを表します。わずか 9 種類のタイプしかないため、18 ステップで表されます。

```
ステップ    ウェイト                    値

   1      0.00000000      <       "UNDECIDED   "
   2      0.11500000      =       "UNDECIDED   "
   3      0.00000000      <       "adventure   "
   4      0.11000000      =       "adventure   "
   5      0.00000000      <       "business    "
   6      0.11040000      =       "business    "
   7      0.00000000      <       "computer    "
   8      0.11640000      =       "computer    "
   9      0.00000000      <       "cooking     "
  10      0.11080000      =       "cooking     "
  11      0.00000000      <       "news        "
  12      0.10660000      =       "news        "
  13      0.00000000      <       "psychology  "
  14      0.11180000      =       "psychology  "
```

```
    15      0.00000000      <       "romance     "
    16      0.10800000      =       "romance     "
    17      0.00000000      <       "travel      "
    18      0.11100000      =       "travel      "
```

たとえば、この type カラムの値の 10.66% は“news”であるため、オプティマイザは、ロー数が 5000 のテーブルでは 533 のローが返されると見積もります。

疎の値と密な値が混在するカラムのヒストグラム

重複性の高いいくつかの値とそれ以外の分散された値を持つテーブルでは、ヒストグラムの出力には、演算子と頻度セルと範囲セルが混在します。

下のヒストグラムで表されるカラムには、値が 30.0、50.0、および 100.0 の大きな割合を占めるローがあります。

ヒストグラムには、これらの値のそれぞれに 2 つのステップがあります。1 つは重複性の高い値を表すステップで、= 演算子とその値に一致するカラムの割合を示すウェイトを持ちます。もうひとつの重複度の高い値のステップは、演算子が < でウェイトは 0.0 です。このカラムのデータ型は numeric(5,1) です。

```
ステップ    ウェイト        値
    1     0.00000000    <=    0.9
    2     0.04456094    <=    20.0
    3     0.00000000    <     30.0
    4     0.29488859    =     30.0
    5     0.05996068    <=    37.0
    6     0.04292267    <=    49.0
    7     0.00000000    <     50.0
    8     0.19659241    =     50.0
    9     0.06028834    <=    75.0
   10     0.05570118    <=    95.0
   11     0.01572739    <=    99.0
   12     0.00000000    <     100.0
   13     0.22935779    =     100.0
```

このカラムにある最小の値は 1.0 なので、null 値のステップは 0.9 で表されます。

重複度が高い値のステップ数の選択

このセクションで示した頻度セルのヒストグラム例では、重複度の高い値の数が比較的少なく、取得されるヒストグラムに必要なステップ数は、create index と update statistics のデフォルト・ステップ数の 20 より少なくてすみます。

テーブルに多くの重複度の高い値を持つカラムがあり、しかもそのカラムのキーの分布が一様ではない場合は、そのヒストグラムでのステップ数を増やすと、オプティマイザはそのカラムに対する探索引数を持ったクエリに関してより正確なコスト見積もりができるようになります。

密頻度カウントを持ったカラムについては、ステップ数を少なくとも値の数より 1 つ多くして null 値を表すセルのステップを確保すべきです。

疎頻度カウントを持つカラムについては、個別の値の数の 2 倍のステップ数が必要です。これによりゼロ・ウェイトと干渉するセルと null 値を表すセルのステップを確保できます。たとえば、pubtune データベースにある titles テーブルに 30 の異なる価格があると、次の update statistics コマンドは 60 ステップのヒストグラムを作成します。

```
update statistics titles
using 60 values
```

次の create index コマンドでは 60 ステップが指定されます。

```
create index price_ix on titles(price)
with statistics using 60 values
```

ごく少数のローにしか一致しない値がカラムにいくつかある場合でも、これらの値は範囲セルとして表され、その結果得られるヒストグラムのステップ数は必要な数よりも少なくなってしまいます。たとえば、state カラムに 100 ステップが要求されていると、少ない割合のロー数で表される州の範囲セルがいくつか作成される可能性があります。

histogram tuning factor の設定

histogram tuning factor は、Adaptive Server が update statistics、update index statistics、update all statistics、create index について 1 つのヒストグラムで分析するステップ数を制御します。

histogram tuning factor を使用すると、ヒストグラムで使用されるリソースを最小限に抑えられます。リソース使用量を増やすのは、最適化のために適切な場合のみです。たとえば、重複する値を持つカラムがある場合や、データ分布が一様ではない場合などです。このような場合には、最大 400 のヒストグラム・ステップが使用されます。ただし、ほとんどの場合は Adaptive Server でデフォルト値 (上記の例では 20) が使用されます。

15.0.2 ESD #2 以前のバージョンの Adaptive Server の場合、この設定パラメータのデフォルト値は 1 (無効) です。15.0.2 ESD #2 以降のバージョンでは、この設定パラメータのデフォルト値は 20 (有効) です。

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

ヒストグラムのステップ数

number of histogram steps のデフォルト値 (20) は、明確な値の数が小さい (ロー・カーディナリティ) 場合や、ローの数が少ないテーブルでは十分です。しかし、ローの数が多いテーブルや、明確な値が多いカラムを持つテーブルでは、特にカラムの値がロー間で均一に分布されていない場合は不十分です。

クエリ・プロセッサによって生成されるクエリ・プランが最適でないと思われる場合は、ステップ数を増やしてヒストグラムの細分性を増やしてみてください。ヒストグラムのステップ数を増やすと、クエリ・プロセッサがクエリ・プランを処理するまでのリソース消費 (特にプロシージャ・キャッシュの使用量) が増え、最適化に時間がかかります。

optimization timeout limit 設定パラメータが原因で、ヒストグラムのステップ数を増やしたときにクエリ・プロセッサで生成されるクエリ・プランが最適でなくなることがあります。

ヒストグラムのステップ数を増やすことで、クエリ・プランと全体のパフォーマンスが最適化されるかどうかを判断してください。number of histogram steps の値を 200 に増やしてみるのも 1 つの方法です。これでもクエリ・プランとパフォーマンスが改良されない場合は、500 などの数値で試してみてください。

または、10,000 データ・ページごとに 10,000 のヒストグラムのステップを使うこともできます。ただし一般的には、ステップ数を 1000 ～ 2000 に増やしても改善は見られません。ヒストグラムのステップ数を変更してもクエリ・プランまたはパフォーマンスに改善が見られない場合は、別のボトルネックを探してください。

Adaptive Server では、update index statistics が実行されたときにヒストグラムのステップ数を判断します。

1 number of histogram steps は、新しいヒストグラムのすべての create index コマンドと update index statistics コマンドのデフォルトを設定する。number of histogram steps のデフォルト値は 20。

2 update index statistics の場合、次のコマンドを実行して、ヒストグラムのステップ数を *nnn* の値に明示的に設定できる。

```
update index statistics table_name
using nnn values
```

3 update index statistics は、新しいヒストグラムの作成時に既存のヒストグラムの現在のステップ数を使用する。number of histogram steps 設定パラメータは、既存のヒストグラムに適用されない。update [index] statistics は、`using nnn values` で明示的にステップ数を指定した場合にのみ、既存のヒストグラムを上書きする。

このステップを完了すると、number of histogram steps 設定パラメータの値はヒストグラムのターゲット・ステップ数になる。これは optdiag 出力には「要求ステップ数」として表示される。

4　Adaptive Server は、*nnn* (ステップ 2 で決定) の値に histogram tuning factor の値を乗算して、内部の中間ヒストグラムを生成する。たとえば、*nnn* が 100 で histogram tuning factor が 20 の場合、中間ヒストグラムのステップ数は最大 2000 ステップ (20 * 100 = 2000) になる。このような内部のヒストグラムを生成すると、Adaptive Server で重複したデータ値を持つ「頻度セル」を特定しやすくなる。頻度セルが見つからないと、ヒストグラムは元のステップ数に戻る。見つかった頻度セルはすべて Adaptive Server が保管する。

元のステップ数は optdiag 出力に「要求ステップ数」として表示される。optdiag の出力には、ヒストグラムの実際のステップ数が「実際のステップ数」として表示される。

5　15.0.1 ESD #1 以前のバージョンの Adaptive Server では、histogram tuning factor がデフォルトで 1 に設定される。15.0.1 ESD#1 以降のバージョンでは、histogram tuning factor にはデフォルト値 20 が使用される。Sybase® では、Sybase 製品の保守契約を結んでいるサポートから特別な指示がない限り、デフォルトの 20 を使用することをおすすめします。

optdiag を使った統計の変更

システム管理者は、optdiag を使ってカラム・レベルの統計を変更できます。

警告！ optdiag を使って統計を変更すると、一部のクエリのパフォーマンスが向上します。ただし、optdiag の実行によりシステム・テーブルにある既存の情報が上書きされ、そのテーブルのすべてのクエリに影響を及ぼす可能性があります。

十分に注意を払い、そのテーブルを使用するすべてのクエリに与える変更を徹底的にテストする必要があります。できれば、統計を運用サーバにロードする前に、開発サーバ上で optdiag simulate を実行して変更のテストを行うことをおすすめします。

シミュレート・モードを使わないで統計をロードする場合は、必要に応じて、何の変更も加えていない optdiag の出力コピーを使うか、update statistics を再実行して元の統計をリストアできるように準備を行ってください。

どの統計も update コマンド、delete コマンド、または insert コマンドを使用しての変更は決してしないでください。

32 ビット版 Adaptive Server の optdiag の出力を使って、別の 32 ビット版 Adaptive Server の統計を変更することはできますが、64 ビット版 Adaptive Server の統計を変更することはできません。同様に、64 ビット版 Adaptive Server の optdiag の出力は、32 ビット版 Adaptive Server への入力としては使用できません。

---

optdiag を使って統計を変更した後で、create index または update statistics を実行すると変更が上書きされてしまいます。コマンドは正常に実行されますが、警告メッセージが表示されます。

```
警告：編集した統計が上書きされます。テーブル：'titles' (objectid 208003772)、
カラム：'type'。
```

optdiag binary の使用

注意 出力はバイナリではなく、16 進数で表示されます。

浮動小数点数を使用すると精度が低下することがあるので、optdiag には binary オプションが用意されています。次のコマンドを実行すると、ユーザが読める統計のほかにバイナリの統計も出力されます。

```
optdiag binary statistics pubtune..titles.price
        -Usa -Ppasswd -o price.opt
```

binary では、optdiag で編集できる統計はすべて、バイナリ値で 1 回、浮動小数点値で 1 回、合計 2 回出力されます。浮動小数点値を表す行の先頭には、optdiag でコメント文字として使われているシャープ記号 (#) が置かれます。

次の例は、authors テーブルにある city カラムのヒストグラムの最初の数行です。

```
ステップ  ウェイト       値

  1   0x3d2810ce    <=    0x41504f204d69616d68ffffffffffffffffffffff
# 1   0.04103165    <=    "APO Miami¥377¥377¥377¥377¥377¥377¥377¥377"
  2   0x3d5748ba    <=    0x41746c616e7461
# 2   0.05255959    <=    "Atlanta"
  3   0x3d5748ba    <=    0x426f79657273
# 3   0.05255959    <=    "Boyers"
  4   0x3d58e27d    <=    0x4368617474616e6f6f6761
# 4   0.05295037    <=    "Chattanooga"
```

optdiag がこのファイルをロードするときには、コメント行以外のすべての行が読み込まれ、シャープ記号で始まる行はすべて無視されます。バイナリ値の代わりに浮動小数点値を編集するには、浮動小数点値を示す行からシャープ記号を取り除き、それに対応するバイナリ値を表す行の先頭にシャープ記号を付けます。

binary の使用が適する場合

optdiag 出力のヒストグラムの 2 つのステップでは、バイナリ値が異なる場合でも精度が落ちるため同じ値として表示されることがあります。たとえば、1.999999999 と 2.000000000 は、バイナリ値が違っていても十進値では 2.000000000 と表示される可能性があります。このような場合は、入力に binary を使用してください。

バイナリ・モードを使用しないと、optdiag は、ステップ値が増えていないことと、バイナリ・モードを使う必要があることを示すエラー・メッセージを表示します。optdiag は、sysstatistics で精度が低下しないように、エラーが発生したヒストグラムのロードをスキップします。

optdiag 入力モードを使った選択性の更新

オプティマイザはクエリの最適化時に探索引数の値が不定の場合には、範囲選択性と中間選択性の値を使用します。optdiag を使ってサーバ全体のデフォルトの選択性値をカスタマイズして、特定のカラムのデータをユーザ自身のアプリケーションに一致させることができます。

サーバワイドなデフォルト値には次のものがあります。

- 範囲選択性 − 0.33
- 中間選択性 − 0.25

次の例は、optdiag を使用するとデフォルト値がどのように表示されるかを示します。

```
カラムの統計：                  "city"
カラム統計の最終更新：          Feb  4 2008  8:42PM

   範囲セル密度：               0x3f634d23b702f715
#     範囲セル密度：            0.0023561189228464
   総密度：                     0x3f46fae98583763d
#     総密度：                  0.0007012977830773
   範囲選択性：                 デフォルトが使用されます。(0.33)
#     範囲選択性：              デフォルトが使用されます。(0.33)
   中間選択性：                 デフォルトが使用されます。(0.25)
#     中間選択性：              デフォルトが使用されます。(0.25)
```

これらの選択性値を編集するには、「デフォルトが使用されます。(0.33)」または「デフォルトが使用されます。(0.25)」の文字列全体を浮動小数点値に置換します。次の例では、範囲選択性が 0.25、中間選択性が 0.05 と十進値に変更されています。

```
   範囲選択性：       0.250000000
   中間選択性：       0.050000000
```

ヒストグラムの編集

ヒストグラムは、次のように編集できます。

- ウェイトを上下いずれかの行に移し、ステップを削除する。
- セルのウェイトを追加の行に拡張し、新しいステップが表すカラム値の上限値とともに 1 つ以上のステップを追加する。

ヒストグラムへの頻度カウント・セルの追加

ステップ数を大きく増やさないで頻度カウント・セルを追加するために、ヒストグラムを編集することはよくあることです。

密頻度カウントを持つヒストグラムの編集

ヒストグラムにどのような変更を加える必要があるかは、その値が疎頻度カウントと密頻度カウントのいずれを表すかによって異なります。特定のカラム値の頻度セルを追加するには、新しいセルの値より小さいカラム値をチェックします。次の小さな値が追加する値に非常に近い場合には、頻度カウントを特定できます。

変更する次の小さなカラム値が頻度カウント値に非常に近い場合は、頻度カウント・セルを簡単に抽出できます。

たとえば、カラムに最低 1 つの 19 と多くの 20 があって、そのヒストグラムが 17 より大きく 22 以下の値を表すのに 1 つのセルを使用している場合、optdiag の出力には次のようなセル情報が含まれます。

```
ステップ    ウェイト          値
...
4         0.100000000     <=      17
5         0.400000000     <=      22
...
```

このヒストグラムを変更して 値 20 をそれだけのステップに置くには、次に示すように 2 つのステップを追加する必要があります。

```
...
4      0.100000000     <=      17
5      0.050000000     <=      19
6      0.300000000     <=      20
7      0.050000000     <=      22
...
```

上の変更されたヒストグラムでは、ステップ 5 は 17 より大きく 19 以下のすべての値を表します。変更されたヒストグラムでのステップ 5、6、7 のウェイトの合計は、ステップ 5 の元のウェイト値と同じです。

密頻度カウントを持つヒストグラムの編集

カラムに 17 より大きく 20 より小さい値がない場合は、疎頻度カウントの表現方法を使用します。まず元のヒストグラムのステップを示します。

```
ステップ    ウェイト          値
...
4         0.100000000     <=      17
5         0.400000000     <=      22
...
```

次の例では、疎頻度ステップに必要なステップ 5 のゼロ・ウェイト・ステップを示します。

```
...
4     0.100000000    <=    17
5     0.000000000    <     20
6     0.350000000    =     20
7     0.050000000    <=    22
...
```

ステップ 5 の演算子は < でなければなりません。ステップ 6 では、値 20 のウェイトを指定する必要があり、その演算子は = でなければなりません。

ロード時のステップ番号チェックの省略

optdiag の入力モードのデフォルトでは、ヒストグラムのステップ番号が 1 ずつ増加しているかどうかがチェックされます。ヒストグラムのステップを編集した後でこのチェックを省略するには、コマンド・ライン・フラグ -T4 を使用します。

```
optdiag statistics pubtune..titles -Usa -Ppassword -T4 -i titles.opt
```

ヒストグラムのロード中にチェックされるルール

ヒストグラムの入力中は、次のルールがチェックされ、違反が見つかるとエラー・メッセージが出力されます。

- 各ステップ番号は、-T4 がオンでない限り、前のステップ番号より 1 つ多くなければならない。各ステップのカラム値は、前のステップのカラム値以上でなければならない。
- ステップのカラム値は単調に増加しなければならない。
- 各セルのウェイトは 0.0 ～ 1.0 の範囲になければならない。
- 1 カラムの総ウェイトは 1.0 に近い値でなければならない。
- 最初のセル値は null 値を表し、null 値が許可されていないカラムであっても必要である。null 値を表すセルは 1 つだけでなければならない。
- 2 つの隣接するセルで < 演算子を使うことはできない。

統計の更新を失わないインデックスの再作成

ヒストグラムの更新後にインデックスを削除して再作成する必要があり、変更された値を保持する場合は、create index コマンドでステップの数に 0 を指定します。このコマンドは、ヒストグラムを変更せずにインデックスを再作成します。

```
create index title_id_ix on titles(title_id)
with statistics using 0 values
```

シミュレートされた統計値の使用

optdiag を使うと、テーブル・データのコピーを使わずにユーザ環境をシミュレートするための統計を生成できます。これにより、非常に小さいデータベースを使ってクエリ最適化の分析を行えます。たとえば、シミュレートした統計は次の目的に使用できます。

- Sybase 製品の保守契約を結んでいるサポート・センタでのオプティマイザに関する問題の再現
- “what-if” 分析を実行した設定変更の計画
- 開発サーバ上での診断

「シミュレートされた統計をロードして使用するための必要条件」(44 ページ) を参照してください。

また、シミュレートされた統計をデータベースにロードし、それをコピーすることもできます。シミュレートされた統計は、それを実際のテーブル・データと区別するための ID とともにシステム・テーブルにロードされます。set statistics simulate on コマンドは、サーバに対して実際の統計の代わりにシミュレートされた統計を使ってクエリの最適化を行うように指示します。

シミュレートされた統計の *optdiag* 構文

pubtune データベースのシミュレート・モード統計を表示するには、次のコマンドを使用します。

```
optdiag simulate statistics pubtune -o pubtune.sim
```

シミュレートした結果をバイナリ出力するには、次のコマンドを使用します。

```
optdiag binary simulate statistics pubtune -o pubtune.sim
```

これらの統計をロードするには、次のコマンドを使用します。

```
optdiag simulate statistics -i pubtune.sim
```

シミュレートされた統計の出力

optdiag の simulate オプションを使って出力すると、ヒストグラムを除き、統計の各ローが“simulated”(疑似値) ローに出力されます。シミュレートされた値は、実際の値のレコードとしてファイルを保持している間に、変更し、ロードできます。

- binary モードを指定した場合は、次の 3 つのローが出力される。
  - バイナリの「シミュレートされた」ロー。
  - 十進法の「シミュレートされた」ロー。コメント行になっている。
  - 十進法の「実際の」ロー。コメント行になっている。

- binary モードを指定しない場合は、次の 2 つのローが出力される。
  - 「シミュレートされた」ロー。
  - 「実際の」ロー。コメント行になっている。

次の例は、pubtune データベースにある authors テーブルのテーブル・レベルの統計を示します。

```
テーブル所有者：                                "dbo"
テーブル名：                                    "authors"

テーブルの統計：                                "authors"

   パーティションのカウント：                   3

パーティションの統計：                          "authors_1376004902"
   データ・ページ・カウント                     74
   空のデータ・ページ・カウント                 0
   データ・ロー・カウント：                     1666.0000000000000000
   転送済みロー・カウント：                     0.0000000000000000
   削除済みロー・カウント：                     0.0000000000000000
   データ・ページ CR カウント：                 10.0000000000000000
     OAM + アロケーション・ページ・カウント：   3
   最初のエクステント・データ・ページ：         0
   データ・ロー・サイズ：                       85.2623049219687914
   並列ジョイン度：                             0.0000000000000000
   未使用ページのカウント：                     5
OAM ページのカウント：                          1

抽出統計：
   データ・ページ・クラスタ比率：               1.0000000000000000
   領域利用：                                   0.9521597490347491
   大容量 I/O 効率：                            1.0000000000000000
```

simulate optdiag 出力には、テーブル統計とインデックス統計の他に次の出力が行われます。

- 分割されたテーブルの分割情報。テーブルが分割されている場合、分割テーブルごとにシミュレートされた情報と実際の情報が出力される。「Pages in the largest partition」(最大パーティション内のページ) 行は適切ではない。

```
   最大パーティション内のページ：   390.0000000000000000 (疑似値)
#     最大パーティション内のページ：   390.0000000000000000 (実際の値)
```

- 並列処理設定パラメータの設定値。

```
設定パラメータ：
   ワーカー・プロセス数：          20 (疑似値)
#     ワーカー・プロセス数：       20 (実際の値)
   最大並列度：                    10 (疑似値)
#     最大並列度：                 10 (実際の値)
   最大スキャン並列度：            3 (疑似値)
```

```
#     最大スキャン並列度：          3 (実際の値)
```

- デフォルトのデータ・キャッシュの設定情報と、指定したデータベースまたは指定したテーブルとそのインデックスで使用されるキャッシュの設定情報。tempdb がキャッシュにバインドされている場合は、そのキャッシュの設定も含まれる。次の出力例は、pubtune データベースが使用するキャッシュの設定情報を示す。

```
キャッシュ設定：                    "pubtune_cache"

     2K プールのサイズ (Kb)：       15360 (疑似値)
#    2K プールのサイズ (KB)：       15360 (実際の値)
     4K プールのサイズ (Kb)：       0 (疑似値)
#    4K プールのサイズ (Kb)：       0 (実際の値)
     8K プールのサイズ (Kb)：       0 (疑似値)
#    8K プールのサイズ (Kb)：       0 (実際の値)
     16K プールのサイズ (Kb)：      0 (疑似値)
#    16K プールのサイズ (KB)：      0 (実際の値)
```

クエリがどのように 16K プールを使用するかをテストするには、上記のシミュレートされた統計値を次のように変更します。

```
キャッシュ設定：                    "pubtune_cache"

     2K プールのサイズ (Kb)：       10240 (疑似値)
#    2K プールのサイズ (Kb)：       15360 (実際の値)
     4K プールのサイズ (Kb)：       0 (疑似値)
#    4K プールのサイズ (Kb)：       0 (実際の値)
     8K プールのサイズ (Kb)：       0 (疑似値)
#    8K プールのサイズ (Kb)：       0 (実際の値)
     16K プールのサイズ (Kb)：      5120 (疑似値)
#    16K プールのサイズ (Kb)：      0 (実際の値)
```

シミュレートされた統計をロードして使用するための必要条件

シミュレートされた統計を使用するには、set statistics simulate on コマンドを実行してからクエリを実行しなければなりません。

クエリを正確にシミュレートするには、次の点に注意する必要があります。

- テーブルに同じロック・スキームと分割を使う。
- テーブル上にトリガがあればそれを再作成し、同じ参照整合性制約を使用する。
- デフォルトとは別の、キャッシュ方式および同時実行性最適化値を設定する。
- データベースとオブジェクトをシミュレートを行っている環境で使用しているキャッシュにバインドする。

- テストしているバッチに含まれているクエリ最適化に影響する set オプション (set parallel_degree など) を含める。
- クエリで使用されるビューを作成する。
- クエリにカーソルが使われていれば、カーソルを使用する。
- プロシージャの実行をシミュレートしている場合は、ストアド・プロシージャを使用する。

シミュレートされた統計は、元のデータベース、またはクエリに対する“what-if”分析を行う目的のためだけに作成されたデータベースにロードできます。

元のデータベースでのシミュレートされた統計の使用

元のデータベースに統計をロードすると、システム・テーブルにある別のローに置かれ、既存のシミュレートされたものではない統計を上書きすることはありません。シミュレートされた統計は、set statistics simulate コマンドが有効なセッションでのみ使用されます。

シミュレートされた統計は他のセッションのクエリの最適化には使われませんが、シミュレートされた統計を使ってクエリを実行すると実際のテーブルやインデックスに対して最適とは言えないクエリ・プランができあがり、そのようなクエリを実行するとシステム上の他のクエリにも悪影響を与えます。

別のデータベースでのシミュレートされた統計の使用

クエリに対する“what-if”分析を行う目的のためだけに作成されたデータベースに統計をロードするときには、最初に次の点を確認してください。

- 入力ファイルで指定されているデータベースが存在していること。サイズは 2MB で十分である。データベース名は入力ファイルで一度しか現れないので、production から test_db へというようにデータベース名を変更できる。
- 入力ファイルで指定されているすべてのテーブルとインデックスが存在していること。テーブルにテータがある必要はない。
- 入力ファイルで指定されているすべてのキャッシュが存在していること。キャッシュのサイズは最小 512K で 2K のプールが 1 つあれば十分である。シミュレートされた統計は、プールの設定に関する情報を提供する。

シミュレートされた統計の削除

シミュレートされた統計をロードすると、master　データベースにある sysstatistics テーブルにキャッシュ設定を記述したローが追加されます。この統計を削除するには、delete shared statistics を使用します。このコマンドは、シミュレートされた統計がロードされているデータベースにある統計には影響を与えません。

シミュレートされた統計を、実際のテーブルおよびインデックスの統計が入っているデータベースにロードした場合は、そのシミュレートされた統計を次のいずれかの方法で削除できます。

- テーブルに対して delete statistics を実行してすべての統計を削除し、次に update statistics を実行してシミュレートされたものではない統計のみを再作成する。
- optdiag を実行して (simulate モードは使わない)、統計を別の位置にコピーする。次に、そのテーブルに対して delete statistics を実行する。最後に、optdiag を使用して (simulate モードは使わない)、別の位置に移した統計をデータベースにコピーする。

シミュレートされた統計を使ったクエリの実行

set statistics simulate on を使用すると、シミュレートされた統計を使ってクエリを最適化できます。

```
set statistics simulate on
```

ほとんどの場合、set showplan on も使用します。

シミュレートされた統計を運用データベースにロードした場合に、シミュレートされた統計を使ってクエリを実行するときに set noexec on を使うと、クエリが実際のテーブルおよびインデックスに一致しない統計に基づいて実行されることを防げます。これにより、運用システムのパフォーマンスに影響を与えずに showplan の出力を検査できます。

シミュレートされた統計の *showplan* メッセージ

set statistics simulate が有効になっていてシミュレートされた統計を使用できる場合は、showplan を実行すると次のメッセージが出力されます。

```
疑似統計を使用して最適化されました。
```

シミュレーション・テストを行ったサーバで並列クエリ・オプションがシミュレートされた値よりも小さい値に設定されていると、showplan は最初にシミュレートされた統計を使ったプランを表示し、次に調整済みのクエリ・プランを表示します。ただし、set noexec が有効に設定されていると、調整済みのプランは表示されません。

引用符を含む文字データ

文字カラムと datetime カラムのヒストグラムでは、すべてのカラム・データは二重引用符で囲まれます。カラム自体に二重引用符が含まれていると、optdiag は 2 つの引用符を表示します。たとえば、次のカラム値があるとします。

```
a quote "mark"
```

optdiag は次のように出力します。

```
"a quote" "mark"
```

これ以外に、optdiag の出力で使われる特殊文字は、シャープ記号 (#) だけです。入力モードでは、シャープ記号で始まる行のすべての文字は無視されます。ただし、シャープ記号が、カラム名またはカラム値の一部として引用符に囲まれている場合は例外です。

統計に対する SQL コマンドの影響

systabstats および sysstatistics に格納されている情報は、DDL (データ定義言語) の影響を受けます。一部のデータ修正言語 (DML) も systabstats に影響します。表 2-8 は、systabstats テーブルと sysstatistics テーブルに対する DDL の影響をまとめたものです。

表 2-8: systabstats と sysstatistics に対する DDL の影響

コマンド	systabstats に対する影響	sysstatistics に対する影響
alter table...lock	テーブルとインデックスの構造とサイズに対する変更を反映するように値を変更する。 テーブルとインデックスを全ページ・ロックからデータオンリー・ロックに変更するときには、そのテーブルのクラスタード・インデックスの indid は 0 に設定され、新しいローがそのインデックスに挿入される。	全ページ・ロックからデータオンリー・ロックへの変更またはその逆の変更の場合は、create index と同様に、データオンリーロック・スキーム間の変更には影響しない。
alter table によるカラム定義の追加、削除、修正	一部の alter table パラメータ (たとえば、drop column を使用して、add に NULL 以外のカラムを追加するか、modify を使用して可変長カラムの長さを減らすなど) では、テーブルのデータ・コピーが必要である。他の alter table 操作は、システム・カタログ情報を更新して行われる。 変更がローの長さに影響し、alter table でテーブルをコピーする必要がある場合、alter table パラメータは、テーブル構造とインデックス構造、およびサイズに値が反映されるように、値を変更する。 alter table パラメータでデータ・コピーが実行されない場合は、変更は行われない。	変更にデータ・コピーが必要な場合、alter table はすべてのインデックスを再構築し、create clustered または create non-clustered index と同じ効果を持つ。alter table はインデックスのないカラムで管理されている統計ローには影響を与えないが、削除中のカラムの統計ローは削除する。 変更によってデータ・コピーが実行されない場合は、変更は行われない。

コマンド	systabstats に対する影響	sysstatistics に対する影響
create table	そのテーブルのローが追加される。制約がインデックスを作成する場合は、下の create index コマンドを参照。	そのテーブルのローが追加される。制約がインデックスを作成する場合は、下の create index コマンドを参照。
create clustered index	全ページロック・テーブルの場合は、indid が 1 に変更され、インデックス内の対象カラムが更新される。データオンリーロック・テーブルの場合は、新しいローが追加される。	まだ含まれていないカラムのローが追加される。すでに含まれているカラムのローは更新される。
create nonclustered index	非クラスタード・インデックスのローが追加される。	まだ含まれていないカラムのローが追加される。すでに含まれているカラムのローは更新される。
282 = delete statistics	影響なし。	テーブルのすべてのロー、あるいは指定したカラムのローが削除される。
drop index	非クラスタード・インデックスのロー、およびデータオンリーロック・テーブルのクラスタード・インデックスのローを削除する。全ページロック・テーブルのクラスタード・インデックスの場合は、indid が 0 に設定され、カラム値が更新される。	インデックス・カラムの実際の統計は削除されない。したがって、オプティマイザはこの情報を継続して使用できる。 非クラスタード・インデックスのシミュレートされた統計を削除する。全ページロック・テーブルのクラスタード・インデックスの場合は、シミュレートされたテーブル・データを含むローにあるインデックス ID の値が変更される。
drop table	テーブルのすべてのローを削除する。	テーブルのすべてのローを削除する。
reorg	時間制限付きで使用した場合は、再開始点を更新する。ページ・カウントが変更された場合はページ数とクラスタ率を更新する。どのようなオプションを使用したかにより、空のページ数、転送されたローまたは削除されたローの数などの値にも影響する。	rebuild オプションはインデックスを再作成する。
320 = truncate table	空のテーブルを反映するように値をリセットする。ローの長さなど、一部の値は変更されない。	影響なし。update statistics を再実行しなくてもトランケートされたテーブルを再ロードできる。
update statistics (空のテーブルに update statistics を実行しても、システム・テーブルに影響しない。)		
table_name	テーブルの値、および指定したテーブルにあるすべてのインデックスの値を更新する。	テーブルにある個々のインデックスの先行カラムのヒストグラムを更新する。すべてのインデックス、およびインデックスのプレフィクス・サブセットの密度を更新する。
index_name	指定したインデックスの値を更新する。	指定したインデックスの先行カラムのヒストグラムを更新する。そのインデックスのプレフィクス・サブセットの密度を更新する。

コマンド	systabstats に対する影響	sysstatistics に対する影響
column_name(s)	影響なし。	カラムのヒストグラムを更新または作成し、指定したカラムのプレフィクス・サブセットの密度を更新または作成する。
update index statistics		
table_name	テーブルの値、および指定したテーブルにあるすべてのインデックスのすべてのカラムの値を更新する。	テーブルにある個々のインデックスのすべてのカラムのヒストグラムを更新する。すべてのインデックス、およびインデックスのプレフィクス・サブセットの密度を更新する。
index_name	指定したインデックスの値を更新する。	指定したインデックスのすべてのカラムのヒストグラムを更新する。そのインデックスのプレフィクス・サブセットの密度を更新する。
update all statistics		
table_name	テーブルの値、および指定したテーブルにあるすべてのカラムの値を更新する。	テーブルにあるすべてのカラムのヒストグラムを更新する。すべてのインデックス、およびインデックスのプレフィクス・サブセットの密度を更新する。

クエリ処理が *systabstats* に与える影響

データの修正は、systabstats テーブルにある多くの値に影響を与える可能性があります。パフォーマンスを向上させるために、これらの値はメモリ内で変更され、ハウスキーピング・タスクによって定期的に systabstats にフラッシュされます。

systabstats に直接問い合わせるには、sp_flushstats を使ってメモリ内の統計を systabstats にフラッシュしてください。たとえば、titles テーブルとそのテーブルのインデックスの統計をフラッシュするには、次のコマンドを使用します。

```
sp_flushstats titles
```

テーブル名を指定しないと、sp_flushstats はすべてのテーブルの統計を現在のデータベースにフラッシュします。

注意　すべてのページ割り付けおよび割り付けの解除は、データ修正クエリによって変更が行われている間に記録されるので、一部の統計 (特にクラスタ率に関する統計) は多少不正確となることがあります。この結果として生ずる矛盾を修正するには、update statistics または create index を実行してください。

sp_showoptstats を使用した統計とヒストグラムの表示

Adaptive Server には、optdiag スタンドアロン・ユーティリティのように機能する sp_showoptstats があり、systabstats や sysstatistics などのシステム・テーブルからさまざまなタイプのデータ・オブジェクトの統計およびヒストグラムを XML ドキュメントで抽出して表示します。構文は次のとおりです。

```
sp_showoptstats [dbname[.owner[.[table_name] ] ] ], [.column],
    [option]
```

『リファレンス・マニュアル：プロシージャ』の「sp_showoptstats」を参照してください。

索引

記号

A

B

C

D

I

L

O

R

S

T

あ

い

え

か

き

く

こ

さ

し

す

せ

そ

た

ち

て

と

は

ひ

ふ

へ

み

ろ

わ