分布頻度ヒストグラムの統計情報を、手動作成するための方法
高さ調整ヒストグラムの統計情報を、手動作成するための方法
実行計画の見方
バインドピーク機能とは

ヒストグラム

列の値をバケット（分布条件区分）に分けて、列値レコードの個数をカウント（集計）した列値（レコードの区分け）の分布状態を表すための統計情報
インデックスを作成した列が Where 条件に指定されていた場合に、10%以上の列が合致する場合には、インデックスを使用してテーブルを読み込むよりテーブル全体をフルスキャンした方が短時間で処理が終了する（ブロック単位のまとめ読み込み効果）

オプティマイザが、実行計画を作成（パース）時に、このヒストグラムを活用すれば、Where 条件で指定された列の比較値により、インデックス読み込みとテーブル全体のフルスキャンのどちらが効率的であるかが判断出来、実行計画の内容が決められる

なお、ヒストグラムの活用のためにはインデックスが必要となるが、ヒストグラムを作成時に、インデックスが自動で作成される訳ではない。
インデックスは、別途手動で作成する
ヒストグラム自身にはレコードの保存位置を示すような索引ガイド的な機能はない

インデックスとヒストグラムの違い

ヒストグラムは、Select Where 条件でインデックスを使った読込みを行うか否かの判断をするためだけの指標である

頻度分布ヒストグラム

列の値の種類１個ずつに対してバケット（分布条件区分）を用意して、列の値のレコードをカウントして何個存在しているかを示しているヒストグラムをいう

高さ調整ヒストグラム

列の値の種類を示すバケット（分布条件区分）を、列の値の種類分用意できないヒストグラム　（ 列の値の種類 ＞ バケット（分布条件）区分け個数）

この場合には、列の値をソートして、バケット（分布条件区分）に順番にかつ均等の個数を入れていき、バケット内の最大の列値を記録しているヒストグラムとなる

ヒストグラムを活用して、Where 条件の列に対する検索条件の処理で、的確に列の値分布によって、『インデックス読み込み』と『テーブル・フルスキャン』を使いわけるための Oracle 設定

・列に対してインデックスを作成する
・列に対してヒストグラムを作成する
・処理 SQL 文の Where 条件で、列に対する比較をリテラル値で行う
　（バインド変数を使用しない）
・初期化パラメータ CURSOR_SHARING：SIMILAR に設定する

頻度ヒストグラムの作成方法（手動）

```
exec　dbms_stats.gather_table_stats( ownname=>'＜オーナー名＞' ,　-
tabname　=>　'＜テーブル名＞' ,　estimate_percent　=>　＜調査サンプル%＞ ,　-
method_opt => 'FOR COLUMNS　＜カラム名＞　SIZE　＜作成バケット数＞' ,　-
cascade　=>　true )
```

ownname：	オーナー名
tabname：	テーブル名
estimate_ percent：	分布状況を作成するために、全レコードの何パーセントを調査するのかの指定値
method_ opt：	分布の調査対象となる列名とバケット（分布条件区分）数の指定 FOR COLUMNS　カラム名 SIZE　バケット（分布条件）の区分け数
cascade：	レコードの列値変更を行った場合に、それに伴う分布状態のヒストグラム自動更新

※　『分布頻度ヒストグラム』と『高さ調整ヒストグラム』の作成方法の違いは、列の値の種類数と作成指定するバケット数の関係で決まる

作成例）

```
exec　dbms_stats.gather_table_stats( ownname=>'KOZUE' ,　-
tabname　=>　'EMP' ,　estimate_percent　=>　100 ,　-
method_opt => 'FOR COLUMNS　deptno　SIZE 254' ,　-
cascade　=>　true )
```

ヒストグラムの確認

```
SELECT　column_name ,　endpoint_value ,　endpoint_number ,
        endpoint_actual_value
  FROM　dba_histograms
  WHERE　owner　=　'＜オーナー名＞'
    AND　table_name = '＜テーブル名＞'
    AND　column_name　=　'＜列名＞'
  ORDER　BY　endpoint_value ;
```

確認例）

```
col  column_name  format  a20
col  endpoint_actual_value  a30

SELECT   column_name ,  endpoint_value ,  endpoint_number ,
         endpoint_actual_value
   FROM  dba_histograms
   WHERE  owner  =  'KOZUE'
      AND  table_name = 'EMP'
      AND  column_name  =  'DEPTNO'
   ORDER  BY  endpoint_value ;
```

【分布頻度ヒストグラムの場合】

COLUMN_NAME	ENDPOINT_VALUE	ENDPOINT_NUMBER
DEPTNO	1	1
DEPTNO	2	3
DEPTNO	3	5
DEPTNO	99	50
↑	↑	↑
列名	列の値	行数累計

上から１個目のヒストグラムのバケットの中の状態、２個目の・・・

【高さ調整ヒストグラムの場合】

COLUMN_NAME	ENDPOINT_VALUE	ENDPOINT_NUMBER
DEPTNO	1 ←最小値	0
DEPTNO	8	1
DEPTNO	16	2
DEPTNO	44	5
↑	↑	↑
列名	バケットの中の最大の列値	バケット番号

バケット３～５までのバケットの中の最大の列値は、すべて４４である

実行計画の確認

同一の Select 文の where 句（比較列名は同一、比較する値は異なる）でも、比較する値の分布が、同一値で多数存在する場合は、実行計画にテーブルの Full Scan が採用される。同一値の存在が少ない場合には、インデックス使用の Read が行われる

※ 全レコード件数の 10%が、検索方法（インデックス使用 Read とテーブルの Full Scan）を変更する境界値の目安
但し、高さ調整ヒストグラムの場合には、一致バケットの分布件数を分布幅で割って概算で比較数を決定するので、正確なレコード分布件数の 10%とは一致しない

分布頻度ヒストグラムを col3 に対して作成してある場合のレコードの偏りが、
col3 に対し、> 10000 の条件で分布数が、10%以上の実行計画

```
Select col1 From TEST1 Where col3 > 10000 ;
```

テーブルの Full Scan を採用

ID	Operation	Name	Rows	Bytes	Cost
0	SELECT STATEMENT		1	8	218
1	SORT AGGREGATE		1	8	
2	TABLE ACCESS FULL	TEST1	999K	7916K	218

col3 の> 500000 の条件で分布数が、10%未満の実行計画

```
Select col1 From TEST1 Where col3 > 500000 ;
```

テーブルのインデックス使用の Read を採用

ID	Operation	Name	Rows	Bytes	Cost
0	SELECT STATEMENT		1	8	218
1	SORT AGGREGATE		1	8	
2	TABLE ACCESS BY INDEX ROWID				
		TEST1	999K	7916K	218
3	INDEX RANGE SCAN	TEST1	999K	7916K	218

バインドピーク機能とは

バインドピークとは、ハードパースでオプティマイザが実行計画を作成する際に、バインド変数にセットしてある実際の値を**実行前に覗く**（peek）機能です

バインド変数を使用した SELECT 文の場合、

Select　col1　From　TEST1　Where　col3 > B1　;

では、B1 はバインド変数であり、実行時に都度値を変更セットして実行します

バインドピーク機能を使うと、初回実行時にオプティマイザが B1 にセットされる値に応じて実行計画を作成し、この時の値でインデックス使用 Read とテーブルの Full Scan の選択が決定される

この実行計画が、共有プールに保存され、次回以降にバインド変数の値が変更された場合でも、異なったバインド値の SQL で共有された同一実行計画が実行される

（バインド変数を使用した時の実行計画の共有度は、次ページの CURSOR_SHARING 初期化パラメータの値によって異なってくる）

このことから、バインド変数にセットした値によっては（列の値の分布が異なるので)、効率が悪いテーブルへのアクセス方法の実行計画で SQL 文が処理される場合がある

バインドピーク機能を OFF にすると

バインドピーク機能を OFF にすると、オプティマイザはバインド変数にセットしてある実際の値を覗く（peek）ことはなくなります

Oracle システムが持っている内部デフォルト値を基準に、インデックスの使用可否を決定し、実行計画を作成することになります

バインドピーク機能が使用されていない場合の内部デフォルト値

Where 条件	基準値	使用アクセス方法
Sel(COL ＝ :x)	1/NDV(列値の種類数)	NDV ＜ 11 フルアクセス読込み NDV ＞ 10 インデックス使用読込み
Sel(COL ＞ :x)	0.05	インデックス使用読込み
Sel(COL ＞＝ :x)	0.05	インデックス使用読込み
Sel(COL Like :x)	0.05	インデックス使用読込み

バインドピーク機能を OFF にする方法

初期化パラメータ _OPTIM_PEEK_USER_BINDS を False に設定する

バインドピーク機能についての自分の私的な見解

バインドピーク機能を OFF にしても、不適切な実行計画が選択される可能性がある

よって、不適切な実行計画が選択されない様にするためには、バインド変数を使わず、Where 条件をリテラル値（定数）で記述して、リテラル値が異なった SQL 文に対しては毎回 実行計画をオプティマイザに作成（解析）させるようにする方が良い

その反面、デメリットとしては、実行計画解析回数が増え CPU コストが増大することと、ライブラリー・キャッシュに保存される実行計画が多くなってしまうことである

【オプティマイザが実行計画を作成するときに、類似した SQL 文を同一と判断し、実行計画を共有する基準】

（バインド変数使用時、リテラル値が異なっても実行計画を共有化する方法）
（ヒストグラム統計情報を判断基準にしたオプティマイザの実行計画作成法）

オプティマイザが実行計画を作成するときに、類似した SQL 文をどんな基準で同一と認識させ、実行計画を共有させるかを示す初期化パラメータ

初期化パラメータ：CURSOR_SHARING

指定値：
- EXACT　SQL 文記述が不一致の場合に、新たに実行計画を作成する、ブランクの個数違いでも不一致（デフォルト）
- FORCE　リテラルの値によらず、実行計画は１つで共有する
- SIMILAR　異なるリテラル値に対しては、それぞれ SQL 文に子カーソルを作成して、異なる実行計画を作成する

→リテラル値で指定された Where 条件付き SQL 文が発行されたとき、オラクル・システムは、**リテラル値をバインド変数に置き換え**て、オプティマイザへ受け渡します

EXACT、FORCE 指定

メリット

リテラル値が異なっていても同一の実行計画として扱われるので、共有プール（ライブラリー・キャッシュ）の使用量が削減できる

デメリット

列の値の分布に偏りがある場合、異なったリテラル値の実行時には、非効率な実行計画を使った SQL 文の処理が行われることがある

SIMILAR 指定

メリット

列の値の分布に偏りがある場合、異なったリテラル値に対してそれぞれの実行計画が作成（子カーソル）されるので、恒に最適な実行計画を使った SQL 文処理が行われる

デメリット

実行計画が SQL 文での指定列値に対して別々に作成されるので、共有プール（ライブラリー・キャッシュ）の使用量が多くなる