Oracle Direct Seminar

ORACLE®

実践!! パフォーマンス・チューニング
-索引チューニング編-【前編】

日本オラクル株式会社
Oracle Direct

Agenda

- 索引構造の理解
  - Bツリー索引の構造
- 索引を使用した検索
  - 全表走査と索引走査
- オプティマイザによる索引走査/全表走査の判断
  - オプティマイザとは
  - ルールベース・オプティマイザとコストベース・オプティマイザ
- ヒストグラムによる索引利用の効率化

Agenda

- 索引構造の理解
  - Bツリー索引の構造
- 索引を使用した検索
  - 全表走査と索引走査
- オプティマイザによる索引走査/全表走査の判断
  - オプティマイザとは
  - ルールベース・オプティマイザとコストベース・オプティマイザ
- ヒストグラムによる索引利用の効率化

索引の種類

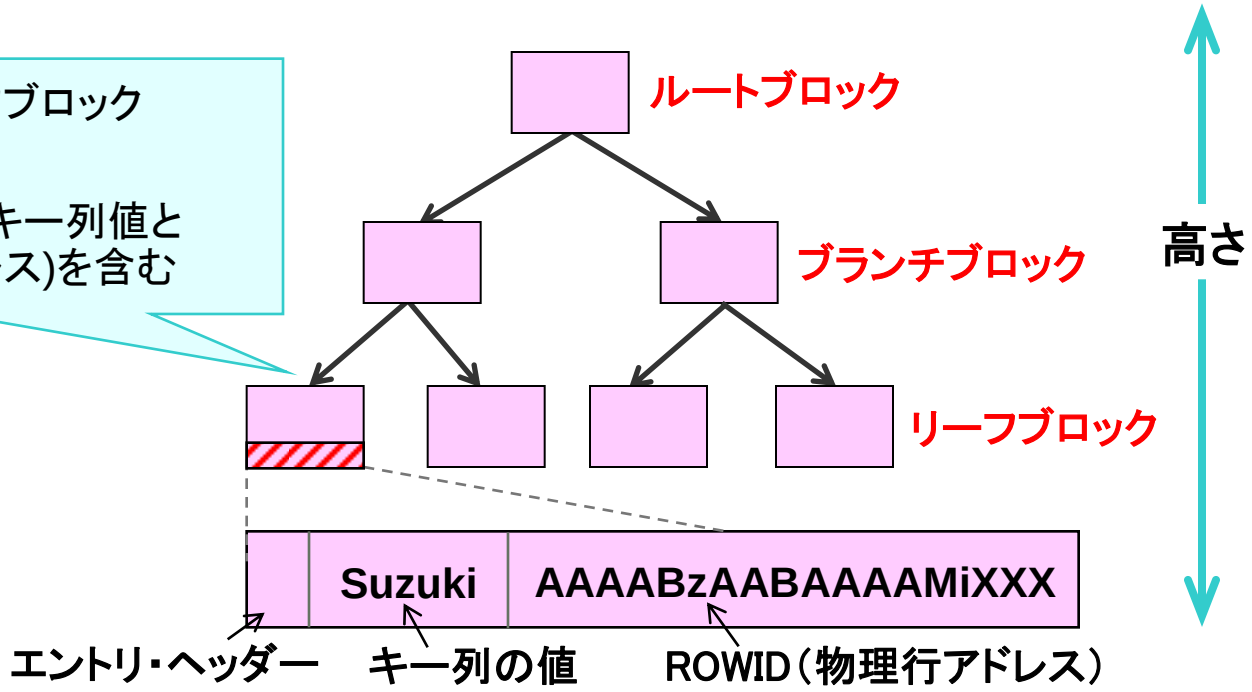
- **索引とは**
  - 百科事典についている索引のように、特定の項目を早く見つけるためのオブジェクト
  - 検索条件で使用する表の列に対して作成
- **索引の種類**
  - Bツリー索引
  - ビットマップ索引
  - 複合索引
  - 逆キー索引
  - 索引構成表

索引チューニング編（後編）

Bツリー索引の構造

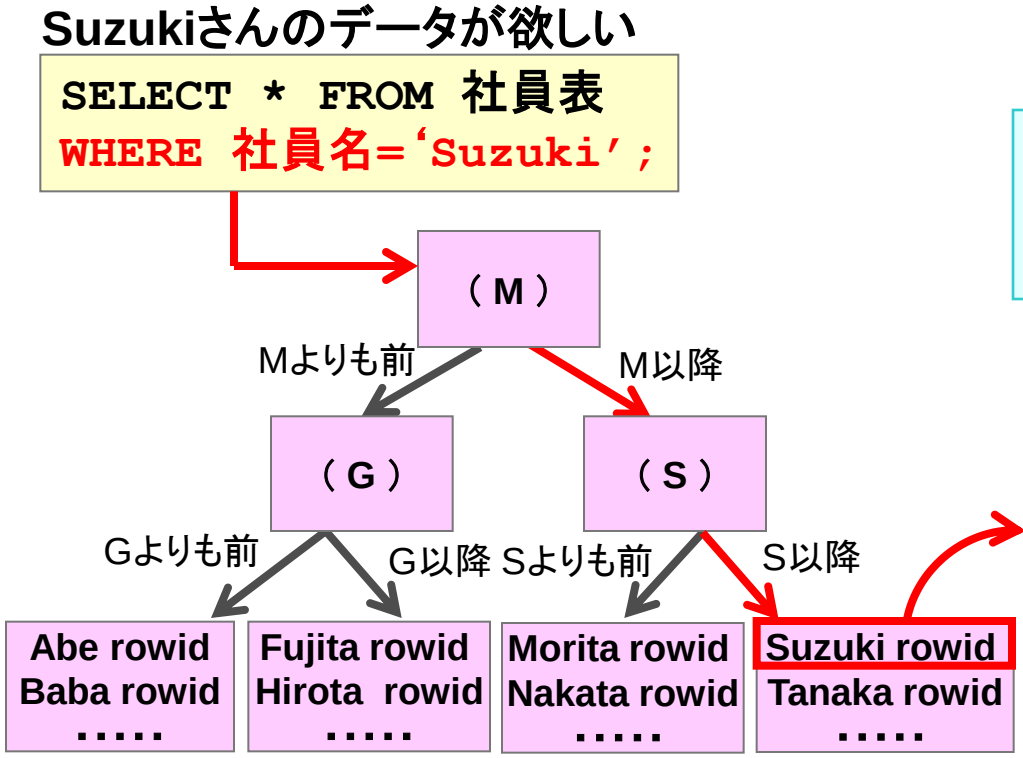
- **ツリー構造で特定のデータにアクセスできるようにした索引**
  - 本の索引の場合、アルファベット順にソートされた索引ならば‘A’のほうが‘X’よりも見つけやすい
  - この問題を解決して、どの値に対しても同じ工数でアクセスできるように「バランス化」したのがB*Tree索引

- ルートブロックからリーフブロックまでの高さが一定
- リーフブロックには列のキー列値とROWID(行の物理アドレス)を含む



Bツリー索引による検索イメージ

- ルートブロックから順にリーフブロックをたどって該当データを検索
  - リーフ・ブロック（最下層）には列のキー列値とROWID（物理アドレス）が含まれる



どのデータにも3ブロックの索引アクセス＋対象データのブロック（計4ブロック）でアクセス可能

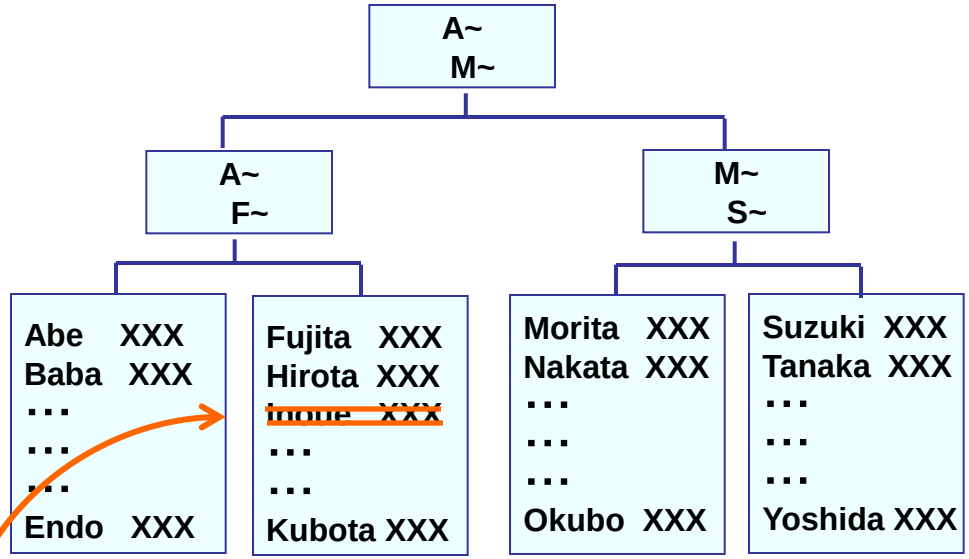
	社員番号	名前	勤務地	性別
ROWID1	1	Tanaka	関東	男
ROWID2	2	Suzuki	関東	女
ROWID3	3	Yoshida	東北	男
ROWID4	4	Abe	関西	女
ROWID5	5	Inoue	関東	男

索引処理の考慮点

- 索引は、DML処理においては悪影響を及ぼす可能性
  - 表データと索引データを同時に更新する必要があるため

社員表

社員番号	名前	勤務地	性別
1	Tanaka	関東	男
2	Suzuki	関東	女
3	Yoshida	東北	男
4	Abe	関西	女
~~5~~	~~Inoue~~	~~関東~~	~~男~~

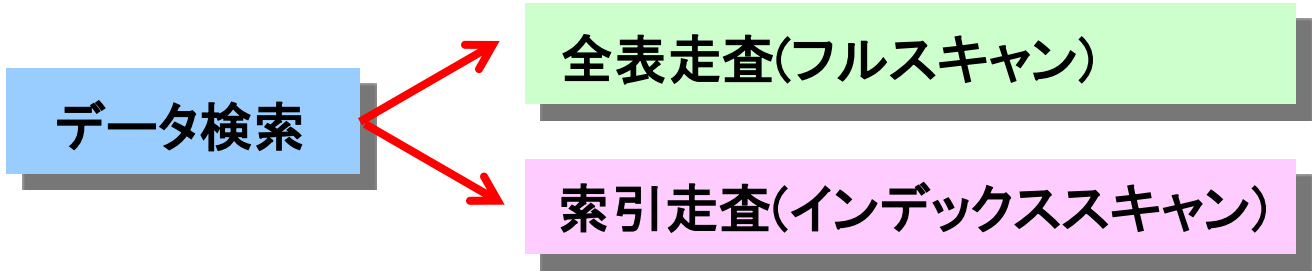


退職（DELETE）

Agenda

- 索引構造の理解
  - Bツリー索引の構造
- 索引を使用した検索
  - 全表走査と索引走査
- オプティマイザによる索引走査/全表走査の判断
  - オプティマイザとは
  - ルールベース・オプティマイザとコストベース・オプティマイザ
- ヒストグラムによる索引利用の効率化

データの検索方法



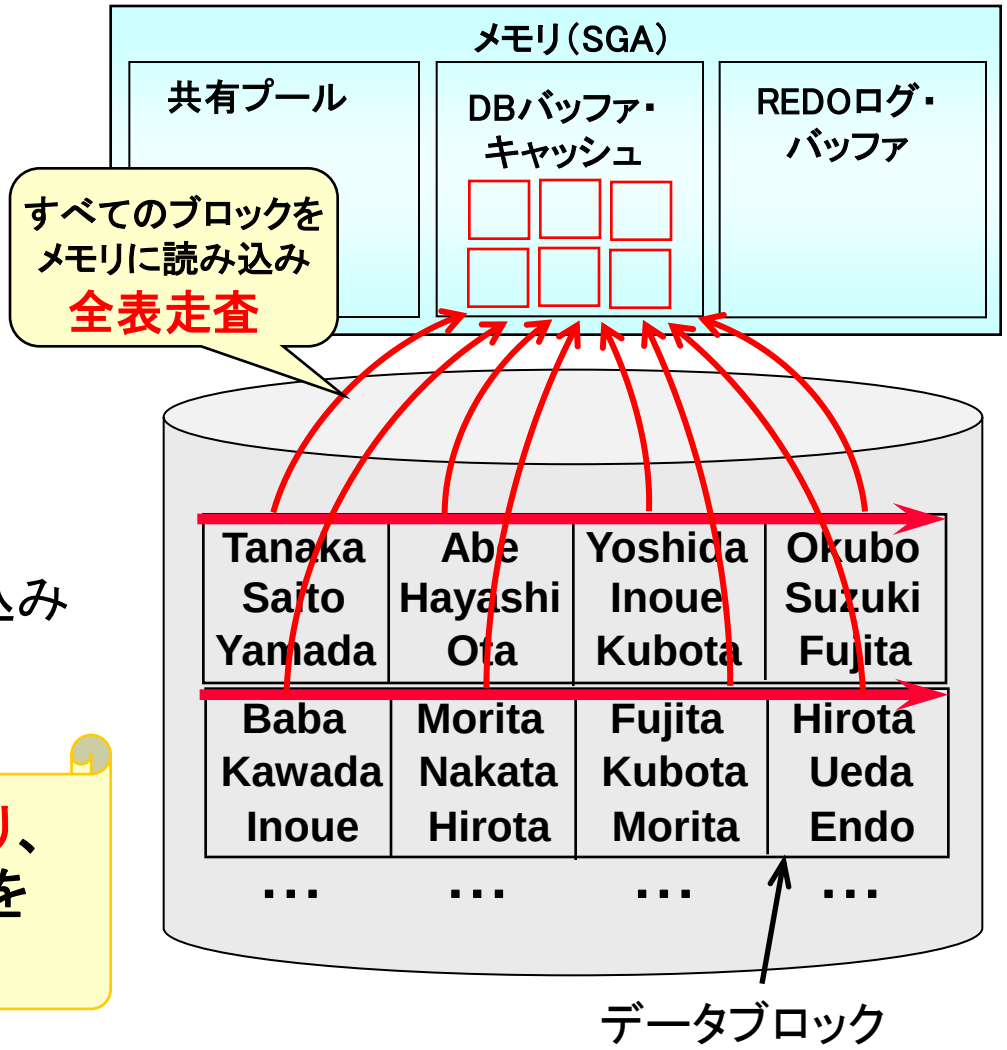
- 全表走査(フルスキャン)
  - 全てのデータを検索、比較して該当データを取得
- 索引走査(インデックススキャン)
  - 索引にアクセスし、索引ブロックから行アドレス(ROWID)を取得
  - ROWIDを使用して直接該当データの入ったブロックにアクセス

全表走査(フルスキャン)

Suzukiさんのデータが欲しい

表のブロックデータを全て読み込み
Suzukiさんのデータを返す

**表にあるすべての行を読み取り、
選択基準を満たしていない行を
フィルタリング**

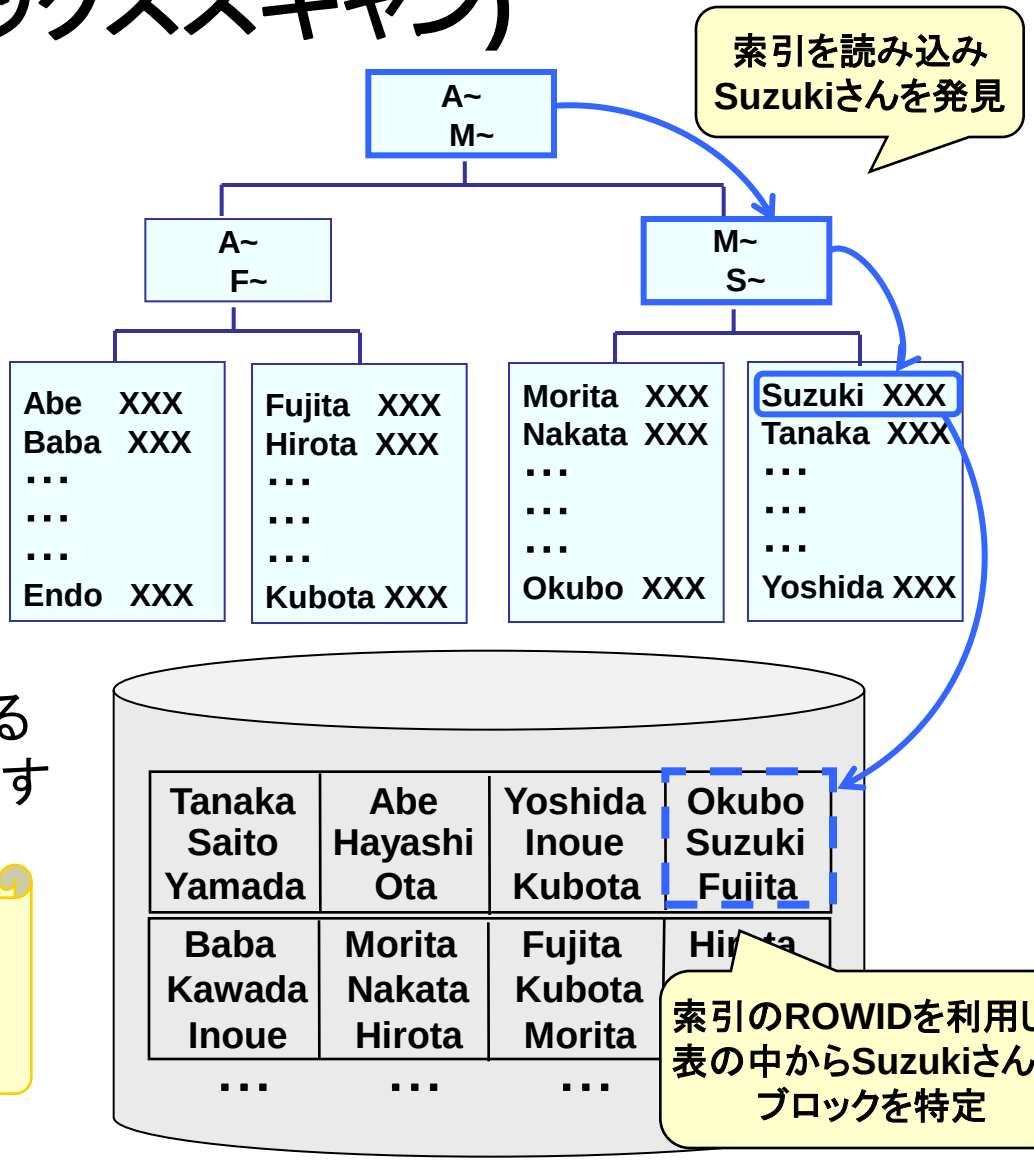


索引走査(インデックススキャン)

Suzukiさんのデータが欲しい

Suzukiさんのデータが入っている
ブロックを読み込み、データを返す

**少量のデータを読む場合、
読まなければならない
ブロックが少なくなる**



全表走査と索引走査

一般的に索引をつけたほうが効率的と言われているが
全表走査より索引走査のほうが常に効率的なのか？

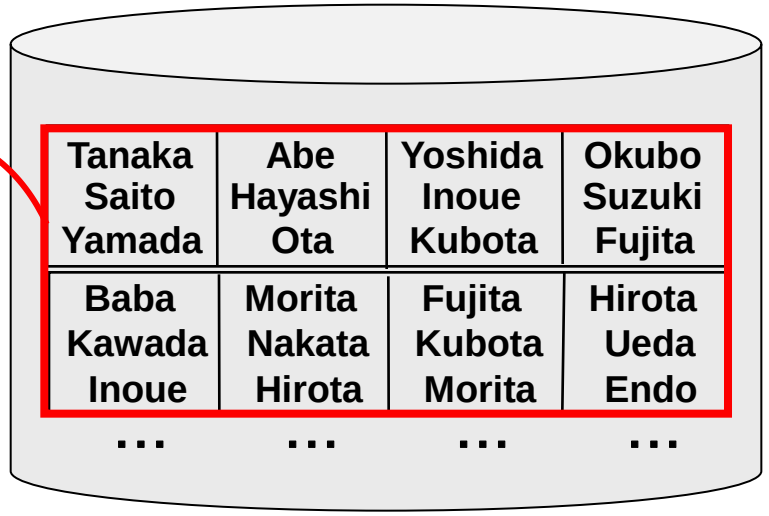
- 全表走査(フルスキャン)
  - 全てのデータを検索、比較する必要がある
  - マルチブロック READをサポート
- 索引走査(インデックススキャン)
  - 索引にアクセスし、索引ブロックから行アドレス(ROWID)を取得
  - ROWIDを使用して直接該当データの入ったブロックにアクセ
  - シングルブロック READ

マルチブロックREAD

- 読み込むブロックは隣接しているため、ブロックより大きいI/O コールを使用可能 ⇒ マルチブロックREAD
- マルチブロックREADを利用することにより、DISKへのI/O回数を減らすことができる

初期化パラメータ db_file_multiblock_read_count = 8

8ブロック単位
でデータを読み込む



マルチブロックREADの効果例

- マルチブロックREADの設定を変えてSQL文を実行
  経過時間を比較

```
SQL> alter system set db_file_multiblock_read_count=1;
SQL> SELECT * FROM employees;
経過: 00:00:04.35
```

```
SQL> alter system set
db_file_multiblock_read_count=10;
SQL> SELECT * FROM employees;
経過: 00:00:01.37
```

＜補足＞

db_file_multiblock_read_countの最大値は？

db_file_multiblock_read_count =< 最大I/Oサイズ / db_block_size

※最大I/Oサイズは、オペレーティング・システムの制限を受けます。

全表走査と索引走査ではどちらが効率的か

- **例：以下のようなEMP表でのI/O回数はどちらが多いか？**
  - データ件数：社員番号1～4000番の4000件
  - サイズ：3200KB
    （ブロックサイズ：8K、400ブロック、1ブロックあたり10件格納）
  - 社員番号（EMPNO）：1～1000番の社員リストが欲しい

```
SELECT * FROM EMP
WHERE EMPNO BETWEEN 1 AND 1000;
```

1/4の絞込み ?

索引走査 (EMPNOに索引ありの場合)
- I/O：100ブロック+索引ブロック = **100+α回**

全表走査 (DB_FILE_MULTIBLOCK_READ_COUNT=8の場合)
- I/O：400ブロック÷8 = **50回**

全表走査と索引走査のまとめ

一般的に索引をつけたほうが効率的と言われているが
全表走査より索引走査のほうが常に効率的なのか？

- 全表走査(フルスキャン)
  - 全てのデータを検索、比較する必要がある
  - マルチブロック READをサポート

→ 選択率が高くなる程、全表走査が有利

- 索引走査(インデックススキャン)
  - 索引ブロックから行アドレス(ROWID)を得て直接行にアクセス
  - シングルブロック READ

→ 単一行へのアクセスは索引走査が有利

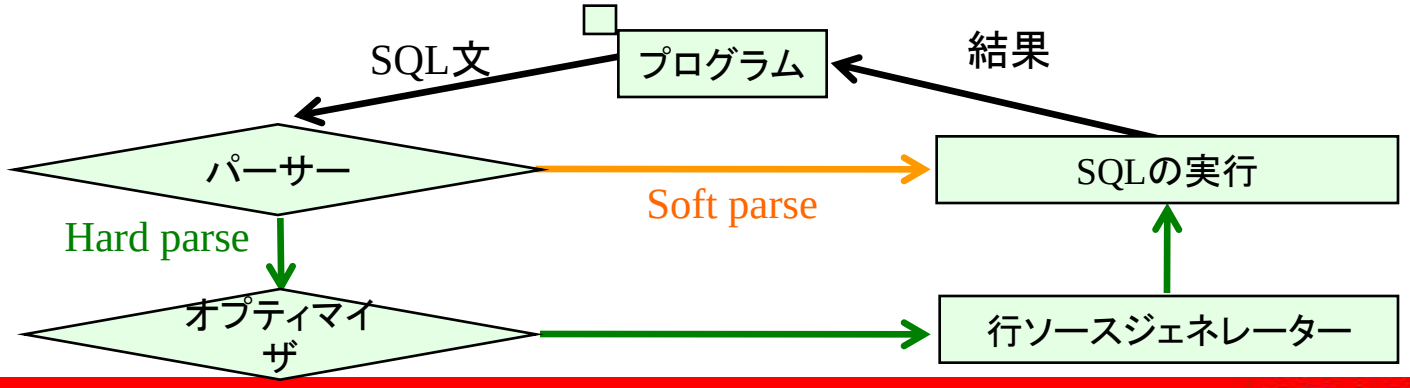
選択行数によっては全表走査の方が速いので、
厳密にチューニングするのであれば、全表走査時と索引走査時の
タイムを測って、パフォーマンスの良い方を選択する

Agenda

- 索引構造の理解
  - Bツリー索引の構造
- 索引を使用した検索
  - 全表走査と索引走査
- オプティマイザによる索引走査/全表走査の判断
  - オプティマイザとは
  - ルールベース・オプティマイザとコストベース・オプティマイザ
- ヒストグラムによる索引利用の効率化

SQL文の処理ステップ

1. 発行されたSQL文はパーサーによってパース（解析）
   - SQL文の構文チェック、意味チェック（表、列が存在するか等）
   - 「同一SQL文」が共有プールにキャッシュされているかチェック
     - キャッシュに存在すれば、後続の処理は必要なくすぐに実行する（soft parse）
     - キャッシュに存在しなければ、オプティマイザによる処理を行う（hard parse）
2. オプティマイザにより最適な実行計画を検討
   - 索引を利用するか、全表走査するか
   - 複数表を結合する場合にどの順番で、どの結合方法を使うか　etc
3. ジェネレータがオプティマイザが生成した実行計画を受け取り、実行



SQL文の実行計画

実行計画の確認方法については、
『実践!! パフォーマンス・チューニング – モニタリング手法編 -』をご受講ください！

- 実行計画の調べ方
  - SQL*PLUSのAUTOTRACEコマンド
  - Explain plan for <SQL>
  - SQLトレース
  - V$SQL及びV$SQL_PLAN(9*i*～)
  - Enterprise Manager（10g～）

参考　実行計画の調べ方（SQL*PlusのAUTOTRACE機能）

1. SYSユーザでPLUSTRACEロールを作成し、SQLを実行するユーザに付与する。

```
SQL> @%ORACLE_HOME%¥sqlplus¥admin¥plustrce.sql
SQL> GRANT plustrace TO scott;
```

2. SQLを実行するユーザで実行計画を保存するための表(PLAN_TABLE)を作成する。

```
SQL> connect scott/tiger
SQL> @%ORACLE_HOME%¥rdbms¥admin¥utlxplan.sql
```

3. AUTOTRACE 機能を ON にし、SQL文を実行する。

```
SQL> SET AUTOTRACE ON
SQL> SELECT ...
```

実行計画の例

結合方法

DEPARTMENTS表へのアクセス方法

EMPLOYEES表へのアクセス方法

```
SELECT last_name,department_name  FROM employees JOIN departments
USING  (department_id);

-------------------------------------------------------------------------------
| Id  | Operation                    | Name         | Rows  | Bytes | Cost (%CPU)|···
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |              |   106 |  2862 |     6  (17)|
|   1 |  MERGE JOIN                  |              |   106 |  2862 |     6  (17)|
|   2 |   TABLE ACCESS BY INDEX ROWID| DEPARTMENTS  |    27 |   432 |     2   (0)|
|   3 |    INDEX FULL SCAN           | DEPT_ID_PK   |    27 |       |     1   (0)|
|*  4 |   SORT JOIN                  |              |   107 |  1177 |     4  (25)|
|   5 |    TABLE ACCESS FULL         | EMPLOYEES    |   107 |  1177 |     3   (0)|
```

```
SELECT last_name,department_name  FROM employees JOIN departments
USING  (department_id)
WHERE  department_id=10;

-----------------------------------------------------------------------------------
| Id  | Operation                    | Name               | Rows  | Bytes | Cost (%CPU)|···
-----------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |                    |     1 |    27 |     2   (0)|
|   1 |  NESTED LOOPS                |                    |     1 |    27 |     2   (0)|
|   2 |   TABLE ACCESS BY INDEX ROWID| DEPARTMENTS        |     1 |    16 |     1   (0)|
|*  3 |    INDEX UNIQUE SCAN         | DEPT_ID_PK         |     1 |       |     0   (0)|
|   4 |   TABLE ACCESS BY INDEX ROWID| EMPLOYEES          |     1 |    11 |     1   (0)|
|*  5 |    INDEX RANGE SCAN          | EMP_DEPARTMENT_IX  |     1 |       |     0   (0)|
```

オプティマイザとは

- **オプティマイザ：問合せの結果を生成する最も効率的な方法（物理的なアクセス手順）を決定し、実行計画を作成する機能**
  - 索引を利用するか
  - 全表スキャンを利用するか
  - 複数の表を結合するときに、結合順序/結合方法はどうするか　など
- **ルールベースオプティマイザ（RBO）**
  - あらかじめ定義されたルール、ランキングに基づいて実行計画を生成
  - SQL文の書き方のみでアクセスパスが決まり、データの量/特性に依存しない
  - Oracle 10g 以降ではサポートされない
- **コストベースオプティマイザ（CBO）**
  - オプティマイザ統計に基づきコストを算出し、最もコストの低い実行計画を生成
  - データの量/特性によってアクセスパスが決まる

ルールベースオプティマイザ（RBO）

下記のランクを利用し、実行計画を作成する

1	ROWIDによる単一行
2	クラスタ結合による単一行
3	一意／主キーをもつハッシュ・クラスタ・キーによる単一行
4	一意／主キーによる単一行
5	クラスタ結合
6	ハッシュ・クラスタ・キー
7	索引付きのクラスタ・キー
8	複合索引
9	単一列索引
10	索引列の境界付きの範囲検索
11	索引列の境界なしの範囲検索
12	ソート／マージ結合
13	索引付きの列のMAXまたはMIN
14	索引付きの列のORDER BY
15	全表スキャン

RBOのルールのアクセスパス

例:

```
SELECT empno FROM emp
WHERE  name = 'Suzuki'
AND    salary > 20000;
```

対象列	EMP表の索引名	索引の種類
empno	EMPNO_IDX	主キー
name	NAME_IDX	非一意索引
salary	SALARY_IDX	非一意索引

① EMP表へのすべてのアクセスパスの洗い出し

考えられるアクセスパス
パス1:(索引検索)NAME_IDXを使用した単一列索引検索
パス2:(索引検索) SALARY_IDXを使用した範囲検索
パス3:(全表スキャン)索引を使用しないアクセスパス

RBOのルールのアクセスパス

② ランクと照らし合わせて最もランクの高いパスを選択

1	ROWIDによる単一行
2	クラスタ結合による単一行
3	一意／主キーをもつハッシュ・クラスタ・キーによる単一行
4	一意／主キーによる単一行
5	クラスタ結合
6	ハッシュ・クラスタ・キー
7	索引付きのクラスタ・キー
8	複合索引
9	**単一列索引**
10	索引列の境界付きの範囲検索
11	**索引列の境界なしの範囲検索**
12	ソート／マージ結合
13	索引付きの列のMAXまたはMIN
14	索引付きの列のORDER BY
15	**全表スキャン**

一番ランクの高い
<ランク9>を採用

パス1：NAME_IDXを使用した単一列索引検索<ランク9>
パス2：SALARY_IDXを使用した範囲検索 <ランク11>
パス3：全表スキャン <ランク15>

RBOの問題点1

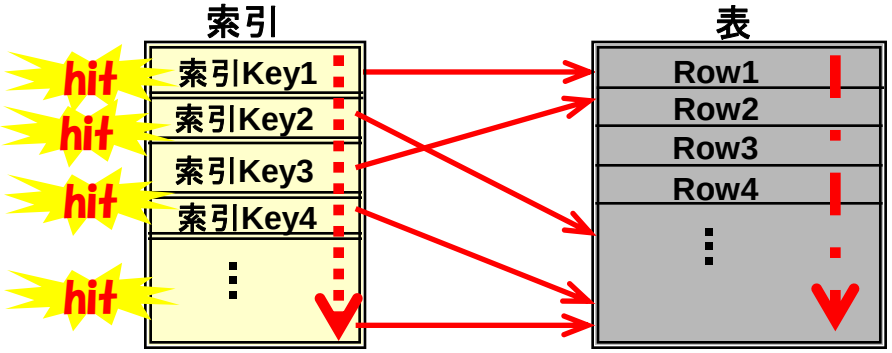
- SQL文の内容（構文）だけで実行計画が決まる
  - データの中身により、より高速なアクセスパスが存在する
  - 結合する表の数が多くなると、開発者は最適なSQLを作成するのが難しい

例：検索対象となる該当件数が多い場合も、インデックスがあれば使う

```
SELECT emp_no FROM emp
WHERE  salary > 20000 ;
```

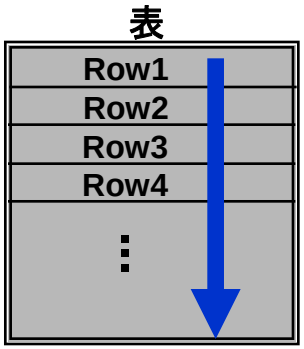
→ salary>20000の人が社員の半分以上いた場合

ランク9 索引走査 (INDEX SCAN)



“索引へのSingle Block Read” +
“ROWIDによる表のSingle Block Read”

ランク15 全表走査 (FULL SCAN)

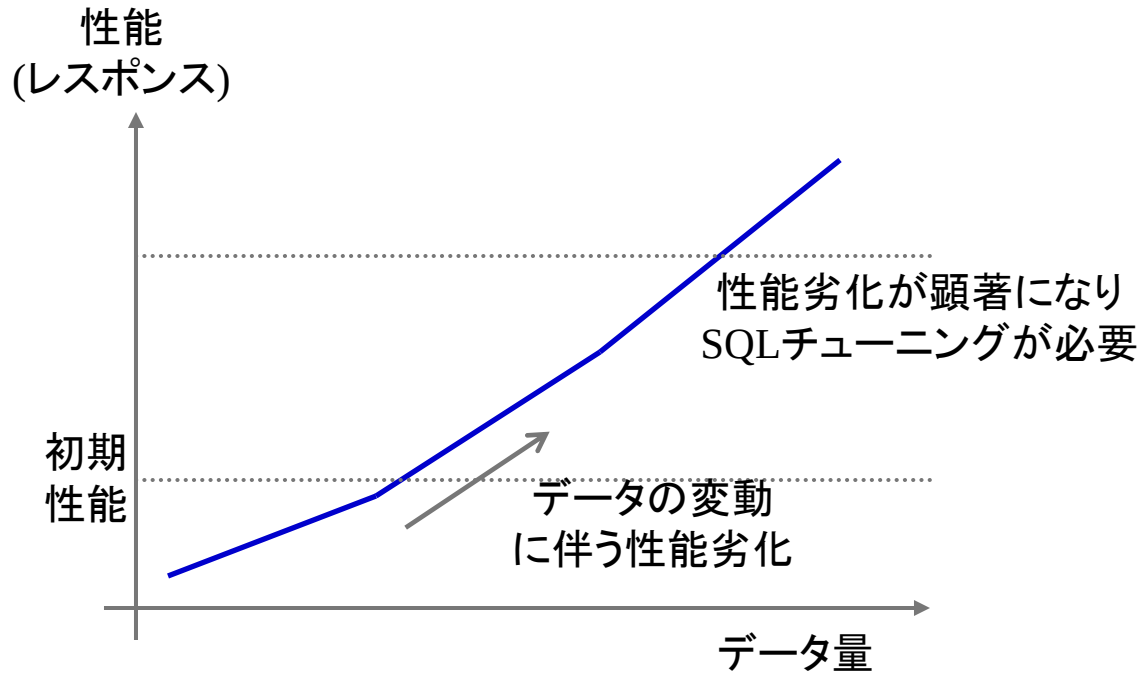


“表へのMulti Block Read”

高速！

RBOの問題点2

- システムの成長にともなう以下の変化への対応が難しい
  - 検索SQLの変化
  - データ量の変化



RBOの問題点3

- Oracle7.3以降の新機能には対応していない
  - パーティション表，パーティション索引
  - パラレル問合せ，パラレルDML
  - BITMAP索引，逆キー索、ファンクション・ベース索引 等の索引
  - スター・ジョイン，ハッシュ・ジョイン
  - 索引構成表
  - Select文のSample句 (9*i*～)
  - 索引スキップ・スキャン(9*i*～)
  - 索引結合

  等々
- **Oracle10g 以降サポートされない**
  - **問題があった場合にも、サポートを受けることができない**
  - **Ruleヒントも不可**

Oracle7 R7.3
Oracle8
Oracle8*i*
Oracle9*i*
Oracle10*g*
Oracle11*g*

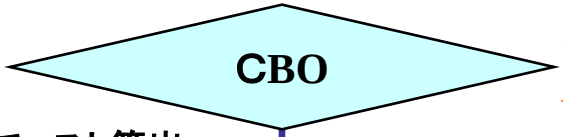
コストベースオプティマイザ (CBO)を使用

コストベースオプティマイザ（CBO）

- コスト：DISK I/O、CPU使用量、メモリー使用量から算出される
  『使用リソース』
- コストベースオプティマイザは以下の情報に基づいてアクセスコストを見積もり、最もコストの低い実行計画を作成する
  - 統計情報
    - 表統計（行数、ブロック長、平均行長）
    - 列統計（列内のデータ種類数、列内のNULL数）
    - 索引統計（リーフブロック数、レベル（ツリーの高さ））
    - システム統計（I/Oパフォーマンス、CPUパフォーマンス）
  - 初期化パラメータの情報
    - DB_FILE_MULTIBLOCK_READ_COUNT
    - OPTIMIZER_MODE

CBOのアクセスパス

```
SELECT emp_no FROM emp
WHERE  name = 'Suzuki'
AND    salary > 20000;
```



次の情報を用いてコスト算出
- SQL文の条件句
- 初期化パラメータ
- EMP表の統計情報
- DEPTNO列の索引
- DEPTNO列の統計情報
- システム統計

初期化パラメータ

optimizer_features_enable
db_file_multiblock_read_count
pga_aggregate_target
optimizer_mode
cursor_sharing
optimizer_index_cost_adj
optimizer_index_caching
etc…

オプティマイザ統計

表統計
- -ブロック数
- -行数
- -平均行長

索引統計
- -ブロック数
- -ツリーの高さ

etc…

列統計
- -列の平均長
- -列の種類数
- -ヒストグラム

システム統計
- -IO性能
- -CPU性能

etc…

コストの低いアクセスパスを選択

```
-----------------------------------------------------------------------------
| Id  | Operation          | Name       | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |            |    45 |  3060 |     3   (0)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL | EMP        |    45 |  3060 |     3   (0)| 00:00:01 |
-----------------------------------------------------------------------------
```

CBOの注意点

- 正確な情報を収集することにより、最適な実行計画を選択できる
- DBMS_STATSパッケージを利用して統計情報収集

  例) 表ごとの統計を収集

```
EXECUTE DBMS_STATS.GATHER_TABLE_STATS('SCOTT','EMP');
```

  例) スキーマ内のすべてのオブジェクトの統計を収集

```
EXECUTE DBMS_STATS.GATHER_SCHEMA_STATS('SCOTT');
```

  例)データベース内のすべてのオブジェクトの統計を収集

```
EXECUTE DBMS_STATS.GATHER_DATABASE_STATS();
```

- 統計が正しく取られていない場合、最適な実行計画が立てられない
  - 統計を取得していない場合
    - 大量削除等を行った場合
  - 表や索引のメンテナンスを行っていない場合
    - 索引が断片化され格納効率が低下している場合

オプティマイザ統計の自動収集

- 9iR2：統計情報が存在しなくても、動的サンプリングによって収集
  - 初期化パラメータOPIMIZER_DYNAMIC_SAMPLINGで指定
  - 頻繁に動的サンプリングが発生するとデータベース全体のパフォーマンスが低下

- **10g:GATHER_STATS_JOBにより統計を自動で収集**
  - スケジュールして任意の時間に統計を取得することが可能
  - 以下のオブジェクトに対して、定期的に統計収集
    - 統計情報をまだ収集していないオブジェクトやデータ
    - 前回の統計取得から10%以上更新されたオブジェクト

※オプティマイザについては、Oracle Direct Seminar **「Optimizer120%活用」**にて
より詳細にご紹介しております。あわせてご受講ください。

<まとめ>オプティマイザ
- RBOとCBOの比較 -

	ルールベース オプティマイザ (RBO)	コストベース オプティマイザ (CBO)
概要	使用可能なアクセスパスを順序づけるランキングに基づいて実行計画を作成（OLTP向き）	統計情報に基づきコストを見積もり、最もコストの低い実行計画を作成（OLTP、DSS共に有効）
メリット	• 開発者にとってRBOの考え方は理解しやすい • SQL実行計画の変動がほとんど起こらない	• データの変動に追随できる • 機能強化の恩恵を受けられる • データの偏りや量に基づいて実行計画を作成できる
デメリット	• データの変動に追随できない • データの偏りや量は考慮されない • 機能強化の恩恵を全く受けられない • Oracle10gからはサポートされない	• 統計情報の取得が必要です • 統計情報の再収集によって性能が変化(*1)するリスクがある **(*1) Oracle11g（EE）では「SQL計画管理」機能により性能変化を抑えることができる**

Agenda

- 索引構造の理解
  - Bツリー索引の構造
- 索引を使用した検索
  - 全表走査と索引走査
- オプティマイザによる索引走査/全表走査の判断
  - オプティマイザとは
  - ルールベース・オプティマイザとコストベース・オプティマイザ
- ヒストグラムによる索引利用の効率化

最適でないアクセスパスの選択 例1

- 10000のうち5000件がヒットしているにもかかわらず索引スキャン

```
SQL> SELECT * FROM emp WHERE deptno=1;

実行計画
----------------------------------------------------------------------------------------
| Id  | Operation                    | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
----------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |          |  4953 |  2437K|  2378   (1)| 00:00:29 |
|   1 |  TABLE ACCESS BY INDEX ROWID| EMP      |  4953 |  2437K|  2378   (1)| 00:00:29 |
|*  2 |   INDEX RANGE SCAN           | DEPT_IDX |  4953 |       |    11   (0)| 00:00:01 |
----------------------------------------------------------------------------------------


SQL> SELECT COUNT(*) FROM emp WHERE deptno=1;

  COUNT(*)
----------
      5000
```

⇒10000件のうち5000件がHit

最適でないアクセスパスの選択　例2

- 10000件のうち1件しかヒットしてないにもかかわらず全表スキャン

```
SQL> SELECT * FROM emp WHERE deptno>500;

実行計画
----------------------------------------------------------------------------
| Id  | Operation          | Name | Rows  | Bytes | Cost (%CPU)| Time     |
----------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |     8 |  4120 |   204   (0)| 00:00:03 |
|*  1 |  TABLE ACCESS FULL| EMP  |     8 |  4120 |   204   (0)| 00:00:03 |
----------------------------------------------------------------------------


SQL> SELECT COUNT(*) FROM emp WHERE deptno>500;

  COUNT(*)
----------
         1
```

⇒10000件のうち1件がHit

EMP表の状態

- データの分布に偏りがある

```
SQL> SELECT deptno,count(*) FROM emp
  2  GROUP BY deptno;

    DEPTNO   COUNT(*)
----------  ----------
         1        5000
         2          25
         3          25
         4          25
         5          25
         6          25

       198          25
       199          25
       200          25
      1000           1
```

↓deptno=1を5000件

↑特異値としてdeptno=10000を1件

合計：10000件

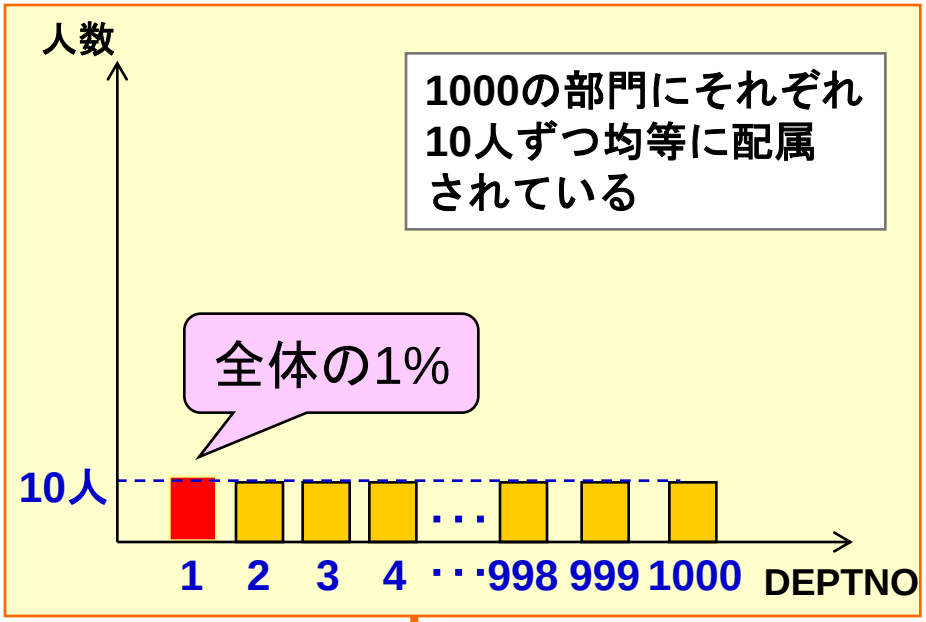
* deptno= 1～200にデータが分布
  特異値としてdeptno=1000を1件

* DEPTNOに対してDEPT_IDX
  という索引を作成

なぜ最適でないアクセスパスが選択されるのか？

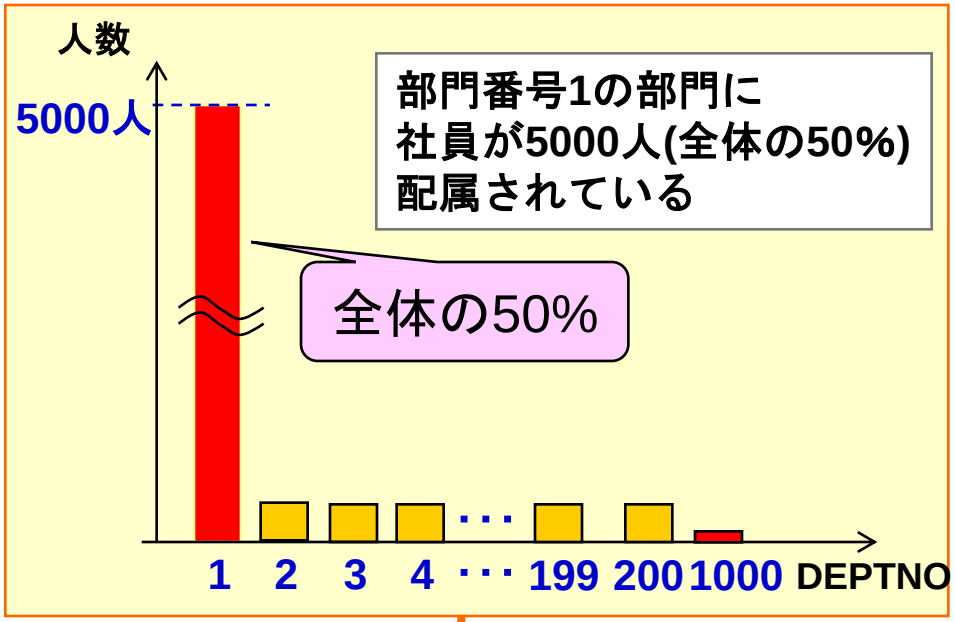
- オプティマイザは、**列値は最小値と最大値間に均一に分散している**と考える
  - 実際の値の分布に偏りがある場合、最適でないアクセスパスを選択する場合がある

オプティマイザの判断



索引走査

実際



全表走査

データの分布に偏りがある場合の解決法

- ヒストグラムを作成することで改善する可能性
  - データの分布状況を統計情報として取得できる
  - データの実際の分布状況に応じて最適なアクセスパスを選択できる
    - 検索対象データ数が少ない場合は索引スキャン
    - 検索対象データ数が多い場合は全表スキャン

ヒストグラムの種類

- ヒストグラムの種類
  - 頻度分布ヒストグラム(Frequency)
    - それぞれの値が何行あるか正確に記録できる
    - 値の種類数が255以下である場合に作成できる
  - 高さ調整ヒストグラム(Hight Balanced)
    - 頻度分布ほど正確ではないがデータの偏りを検出できる

```
SQL> exec DBMS_STATS.GATHER_TABLE_STATS (
                ownname=>スキーマ名,
                tabname=>表名,
                estimate_percent =>100(サンプリング率),
                method_opt=>'FOR COLUMNS SIZE n deptno',
                cascade =>TRUE(索引の情報も取得するか));
```

FOR ALL INDEXED COLUMNS:すべての索引列について取得
FOR ALL COLUMNS:すべての列について取得
FOR COLUMN 列名:特定の列について取得
SIZE AUTO:ヒストグラム・バケット数を自動設定
SiZE n:ヒストグラム・バケット数を指定

頻度分布ヒストグラム

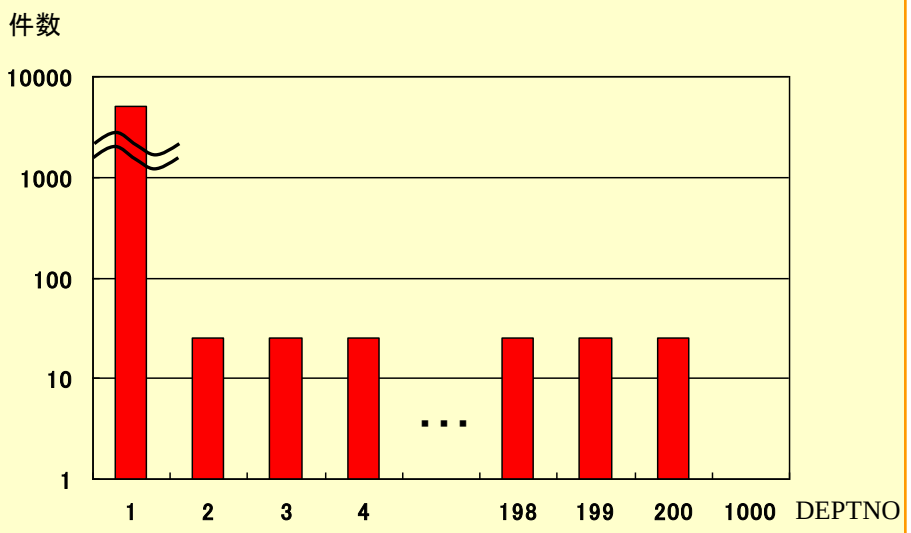
- method_opt引数で列の種類数よりも大きな値を指定する

```
SQL> exec DBMS_STATS.GATHER_TABLE_STATS (
      ownname=>'sh',
      tabname=>'EMP',
      estimate_percent => 100,
      method_opt=>'FOR COLUMNS SIZE 254 deptno',
      cascade =>TRUE);

SQL> SELECT column_name,num_distinct,num_buckets,histogram
 2 > FROM   user_tab_col_statistics
 3 > WHERE  table_name = 'EMP';

COLUMN_NAME        NUM_DISTINCT NUM_BUCKETS HISTOGRAM
---------------- ------------ ----------- ------------
DEPTNO                    201         201 FREQUENCY
```

頻度分布ヒストグラム

- それぞれの値が何行あるか正確に記録される

```
SQL> SELECT endpoint_value, endpoint_number
     2 > FROM user_tab_histograms
     3 > WHERE table_name='EMP'
     4 > column_name='DEPTNO'

ENDPOINT_VALUE ENDPOINT_NUMBER
-------------- ---------------
             1            5000
             2            5025
             3            5050
           ...             ...
           199            9975
           200            9999
          1000           10000
```



高さ調整ヒストグラム

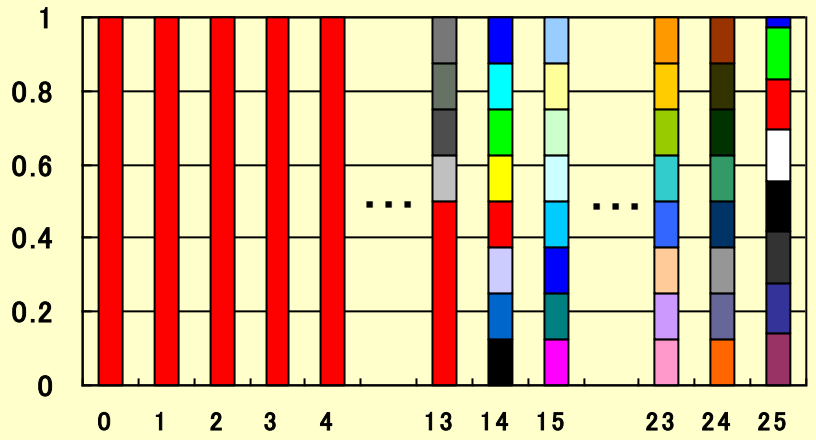
- method_opt引数で列の種類数よりも小さな値を指定する

```
SQL> exec DBMS_STATS.GATHER_TABLE_STATS (
      ownname=>'sh',
      tabname=>'EMP',
      estimate_percent => 100,
      method_opt=>'FOR COLUMNS SIZE 25 deptno',
      cascade =>TRUE);

SQL> SELECT column_name,num_distinct,num_buckets,histogram
 2 > FROM   user_tab_col_statistics
 3 > WHERE  table_name = 'EMP';

COLUMN_NAME          NUM_DISTINCT NUM_BUCKETS HISTOGRAM
-------------------- ------------ ----------- ---------------
DEPTNO                        201          25 HEIGHT BALANCED
```

高さ調整ヒストグラム

- **頻度分布ヒストグラムほど正確ではないが、ヒストグラムなしよりも精度が高い**
  - 指定したバケット数で分割し、ソートした上で同じ件数ずつバケットに格納
  - 各バケットの最後の値（endpoint_value）を記録し、データの偏りを検出するために使う
  - 複数バケットのendpoint_valueが等しければ、他の値よりも多く存在する（ポピュラー値）
  - 領域使用率向上のためポピュラー値がある場合には、最後のバケット情報のみ記録

```
SQL> SELECT endpoint_value, endpoint_number
 2 > FROM user_tab_histograms
 3 > WHERE table_name='EMP'
 4 > column_name='DEPTNO'

ENDPOINT_VALUE ENDPOINT_NUMBER
-------------- ---------------
             1              12
             9              13
            25              14
            41              15
           ...             ...
           169              23
           185              24
          1000              25
```



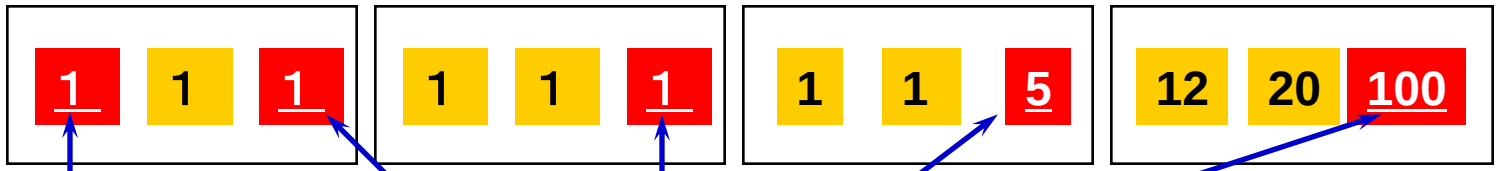
高さ調整ヒストグラムのイメージ

12件のデータ
(1が3分の2)

1 1 1 1 1 1 1 1 5 12 20 100

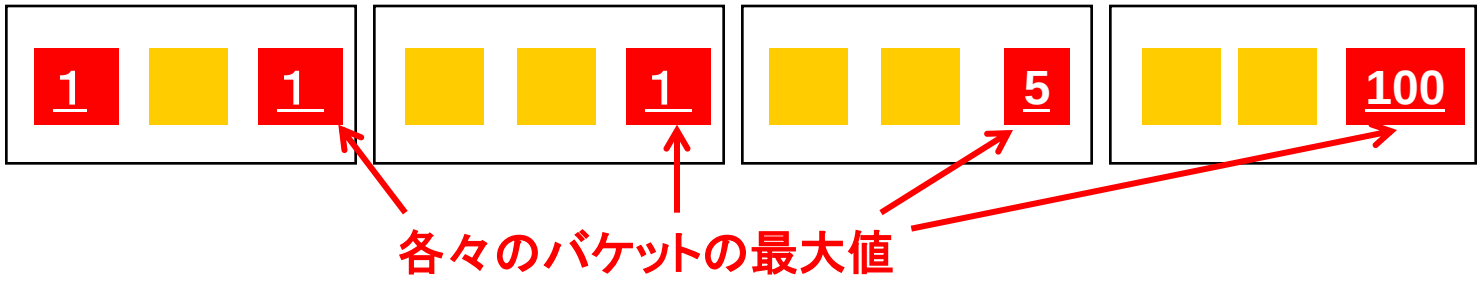
```
DBMS_STATS.GATHER_TABLE_STATS (
      ownname              =>スキーマ名,
      tabname              =>  表名,
      estimate_percent     => 100,
      block_sample         => FALSE,
      method_opt           =>'FOR ALL INDEXED COLUMNS SIZE 4',
      cascade                 =>TRUE);
```

全体をつ4のバケットに分割



データの最小値 と 各々のバケットの最大値 を統計情報に加える

高さ調整ヒストグラムのイメージ




```
SELECT * FROM 表
WHERE 列 = 1;
```

ヒストグラムを使わない
→ 索引操作

ヒストグラムで偏りを調べると
1以下のバケットは2つなので
全体の50%→ 全表走査

ヒストグラム利用方法のまとめ

ヒストグラムが存在しない場合 → CBOは値の最小値～最大値の間にデータが均一に分散されていると考える

↓

ヒストグラムが存在する場合 → CBOはデータの分布状況を見て最適なアクセスパスを選択できる

＜補足＞

- バケット数
  - デフォルトのバケット数(75)がほとんどの場合に有効
  - よりよい結果を得るために、別の値を試す事も必要
- ヒストグラムを使っても意味がないケース
  - where 句で使用されない列
  - データが均一に分布している表
  - 列が一意で等価検索しかされない

ヒストグラム統計収集の自動化

- 9i以前のデータベースでは、データ分布が均一でないために
  パフォーマンスが劣化している部分についいは、手動でヒストグラム
  情報を取得する必要
- 10gからの自動オプティマイザ統計収集では、ヒストグラム関連の
  統計情報がデフォルトで取得される

```
DBMS_STATS.GATHER_TABLE_STATS (
ownname          =>スキーマ名,
tabname          =>表名,
estimate_percent =>100,
block_sample     =>FALSE,
method_opt       =>'FOR ALL COLUMNS SIZE AUTO',
degree           =>NULL,
cascade          =>TRUE);
```

10g～
SIZE AUTO：ヒストグラム・バケット数を自動設定

※DBMS_STATSの詳細については、マニュアル
「**Oracle Database PL/SQLパッケージ・プロシージャおよびタイプ・リファレンス 10*g* リリース2（10.2）**」
をご覧ください。

まとめ

- 索引構造の理解
  - Bツリー索引の構造
- 索引を使用した検索
  - 全表走査と索引走査
- オプティマイザによる索引走査/全表走査の判断
  - オプティマイザとは
  - ルールベース・オプティマイザとコストベース・オプティマイザ
- ヒストグラムによる索引利用の効率化

OTN×ダイセミ　でスキルアップ!!

・一般的な技術問題解決方法などを知りたい！
・セミナ資料など技術コンテンツがほしい！

Oracle Technology Network(OTN)を御活用下さい。

http://otn.oracle.co.jp/forum/index.jspa?categoryID=2

一般的技術問題解決にはOTN掲示版の
「データベース一般」をご活用ください

※OTN掲示版は、基本的にOracleユーザー有志からの回答となるため100%回答があるとは限りません。
ただ、過去の履歴を見ると、質問の大多数に関してなんらかの回答が書き込まれております。

http://www.oracle.com/technology/global/jp/ondemand/otn-seminar/index.html

過去のセミナ資料、動画コンテンツはOTNの
「OTNセミナー　オンデマンド コンテンツ」へ

※ダイセミ事務局にダイセミ資料を請求頂いても、お受けできない可能性がございますので予めご了承ください。
ダイセミ資料はOTNコンテンツ　オン　デマンドか、セミナ実施時間内にダウンロード頂くようお願い致します。

ORACLE

OTNセミナー　オンデマンド　コンテンツ

期間限定にて、ダイセミの人気セミナーを動画配信中!!
ダイセミのライブ感はそのままに、お好きな時間で受講頂けます。

OTN　オンデマンド　　検索

※掲載のコンテンツ内容は予告なく変更になる可能性があります。
期間限定での配信コンテンツも含まれております。お早めにダウンロード頂くことをお勧めいたします。

オラクル クルクルキャンペーン

あのOracle Database Enterprise Editionが超おトク!!

おトクな買い方
オラクル5年分

- ライセンス使用期間　を5年間に設定
- 初期のライセンスコストがなんと67%OFF　！
- テクニカル・サポート価格も53%OFF　！

Oracle Databaseの
ライセンス価格を大幅に抑えて
ご導入いただけます

多くのお客様でサーバー使用期間とされる
5年間にライセンス期間を限定
・期間途中で永久ライセンスへ差額移行
・5年後に新規ライセンスを購入し継続利用
・5年後に新システムへデータを移行

Enterprise Editionはここが違う!!
・圧倒的な**パフォーマンス**!
・データベース**管理がカンタン**!
・データベースを**止めなくていい**!
・もちろん**障害対策**も万全!

この機能でこの価格
ライセンスパック

- Oracle Databaseの機能を存分に使える！
- 2ノードRAC構成も可能！
- サーバー構成によって計4種類のパックから選べる！

詳しくはコチラ

http://www.oracle.co.jp/campaign/kurukuru/index.html

Oracle Direct 0120-155-096

お問い合わせフォーム
http://www.oracle.co.jp/inq_pl/INQUIRY/quest?rid=28

ORACLE

あなたにいちばん近いオラクル

Oracle Direct

まずはお問合せください

Oracle Direct　検索

システムの検討・構築から運用まで、ITプロジェクト全般の相談窓口としてご支援いたします。
システム構成やライセンス/購入方法などお気軽にお問い合わせ下さい。

Web問い合わせフォーム

専用お問い合わせフォームにてご相談内容を承ります。
http://www.oracle.co.jp/inq_pl/INQUIRY/quest?rid=28

※フォームの入力には、Oracle Direct Seminar申込時と同じ
ログインが必要となります。
※こちらから詳細確認のお電話を差し上げる場合がありますので、ご登録さ
れている連絡先が最新のものになっているか、ご確認下さい。

フリーダイヤル

0120－155－096
※月曜~金曜　9:00~12:00、13:00~18:00
（祝日および年末年始除く）

以上の事項は、弊社の一般的な製品の方向性に関する概要を説明するものです。また、情報提供を唯一の目的とするものであり、いかなる契約にも組み込むことはできません。以下の事項は、マテリアルやコード、機能を提供することをコミットメント（確約）するものではないため、購買決定を行う際の判断材料になさらないで下さい。オラクル製品に関して記載されている機能の開発、リリースおよび時期については、弊社の裁量により決定されます。

＜参考＞統計を取得しない場合

- 統計情報が収集されていない場合には、内部的に持っている
  デフォルト値で計算

表のデフォルト値

カーディナリティ	ブロック数*(block_size-24)/100
行の平均の長さ	100バイト
ブロック数	HWM下の実際のブロック数
リモート・カーディナリティ	2000行
リモートの行の平均の長さ	100バイト

索引のデフォルト値

レベル	リーフブロックにたどり着くまでのコスト計算用	1
リーフブロック	索引内のリーフブロックの数	25
平均リーフブロック	索引内の各固有値を持つリーフ・ブロックの平均数。 一意制約及びPKの場合には常に1。	1
平均データブロック	索引内の固有値によって示される表内のデータ ブロックの平均数。	1
個別キー	キーの種類	100
クラスタ係数	索引の値を基に、表の行の並びがどれだけ効率的か	800