Oracle **Database** Technology Night
～集え！オラクルの力（チカラ）～

パフォーマンス・チューニングの極意
津島博士の
明日から使えるSQLチューニング

ORACLE DATABASE 12c

Plug into the Cloud

日本オラクル株式会社
クラウド・テクノロジー事業統括
Database & Exadata プロダクトマネジメント本部
応用技術部 担当ディレクター
津島 浩樹

ORACLE

Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Agenda

1. AWRからの解析
2. SQLチューニング
3. オプティマイザ統計

AWRからの解析

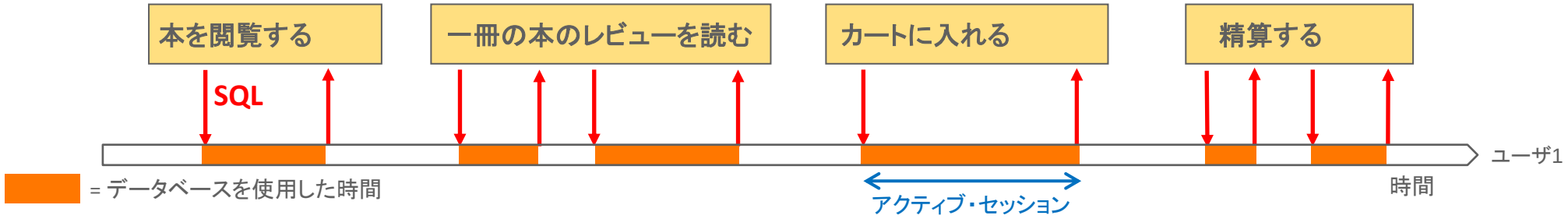
- 解析できないもの
- DB Timeベースのチューニング
- AWRについて
- 待機イベント
- サンプルAWR

AWRからの解析
解析できないもの

- データベースに負荷が掛っていない
  - アプリケーション側の問題
  - データベース処理以外の時間が多い
- どんな処理か
  - 大量のデータを持ってきてアプリケーションで処理する
    - 必要な（処理をした）データだけを持ってきましょう
  - Row by Row処理（繰返し処理でデータベースに1行づつアクセスする）
    - 結合などはデータベースで行いましょう
  - バッチ処理で多い
- OS統計でリソース（CPU、I/Oなど）の使用状況から判断

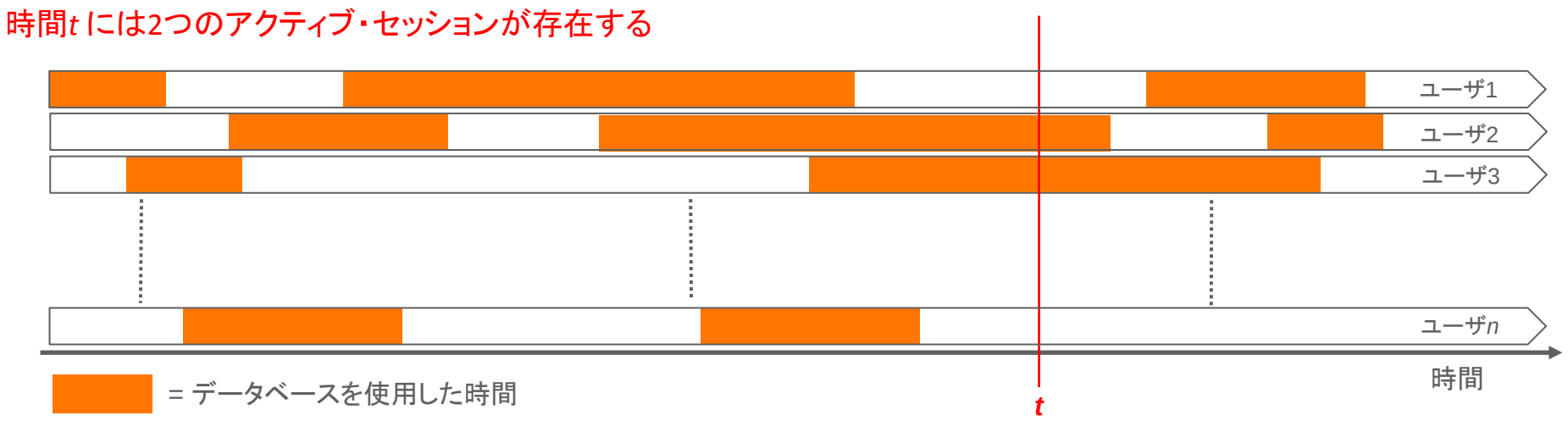
AWRからの解析
DB Timeベース・チューニング（DB Timeとアクティブ・セッション）

- DB Time（データベース時間）
  - すべてのデータベース内処理に要したセッション（フォアグランド）の合計時間（CPU時間、I/O時間、非アイドル待機時間が含む）
- Active Session（アクティブ・セッション）
  - 現在データベース内で処理を行っている（DB Time中の）セッション
- %Activity（平均アクティビティ）
  - 実経過時間とアクティブな（データベースを使用した）時間の割合



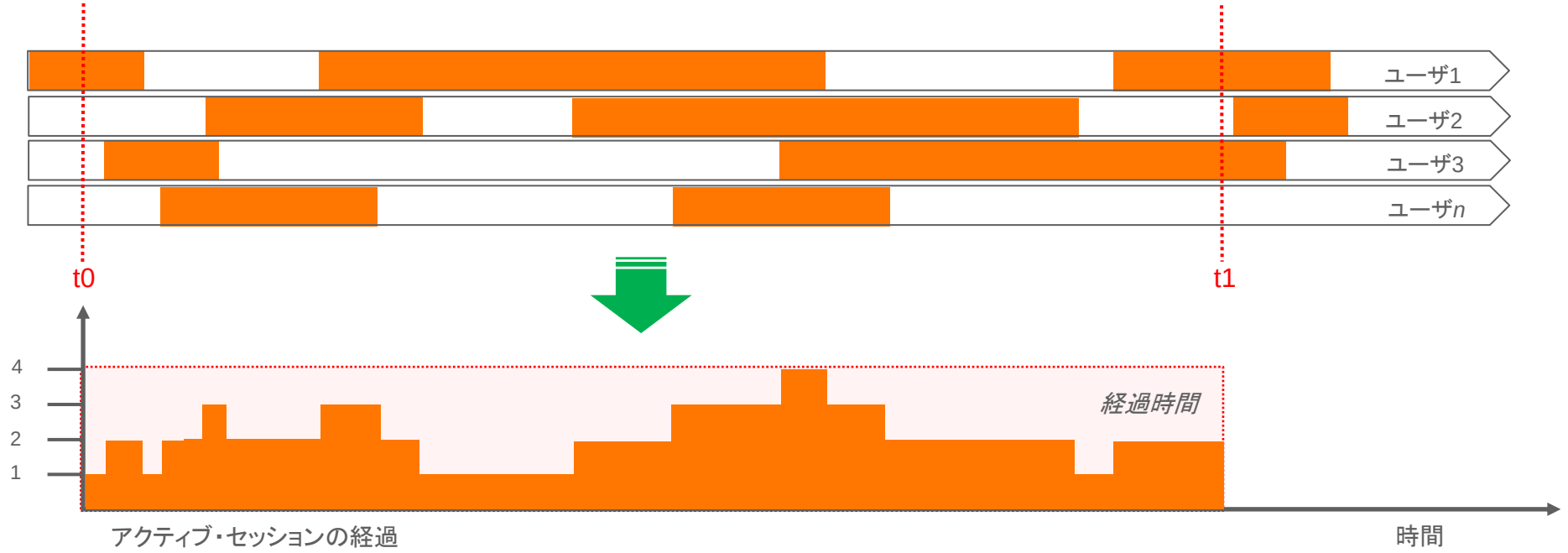
AWRからの解析
DB Timeベース・チューニング（複数セッションのとき）

- DB Time：すべてのセッションのデータベース時間の合計
- Average Active Sessions（平均アクティブ・セッション）
  - すべてのセッションの平均アクティブ数
  - アクティブ・セッションが多い時間帯が負荷が多い



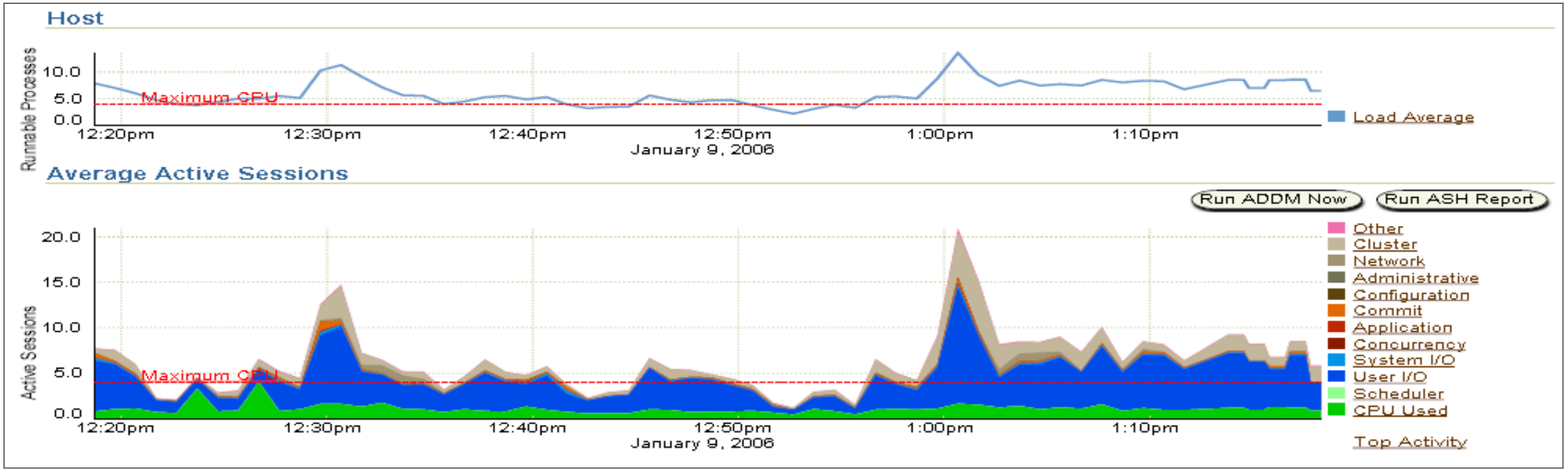
AWRからの解析
DB Timeベース・チューニング（DB Timeの可視化）

$$\text{平均アクティブ・セッション} = \frac{\text{全データベース時間}}{\text{全アクティブ・セッションの経過時間}}$$



- アクティブ・セッション（DB Time中のセッション）数から負荷を見る

AWRからの解析
DB Timeベース・チューニング（EM パフォーマンス・ページ）



- 待機クラスごとのアクティブ・セッションの経過
- カラー領域の合計 ＝ DB Time

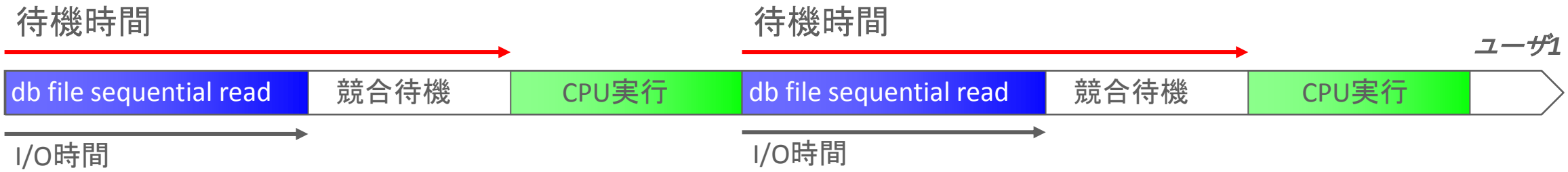
これの待機イベントごとが、
AWRではTop 5 Timed Events

AWRからの解析
DB Timeベース・チューニング（システムのパフォーマンスとDB Time）

- システム負荷の増加（セッション数の増加など）
  - 競合待機時間などの増加（=> DB Timeの増加）
- DBマシンがCPUネック
  - CPU実行時間の増加（=> DB Timeの増加）
- I/Oパフォーマンスの低下
  - I/O時間の増加（=> DB Timeの増加）

バックグランド（BG）プロセスについて

- **BGプロセスが原因でもDB Timeに現れる（例、log parallel writeが多いとlog file syncも増える）**
- **BGプロセスはH/Wリソースが問題なければ影響は少ない**



AWRからの解析
AWRについて（AWR内での時間）

- DB Time（データベース時間）
  - フォアグランドがデータベース内で処理した合計時間 ≠ **応答時間（Elapsed Time）**
- DB CPU（CPU時間）
  - DB Time内でのCPU時間（DB CPU / DB Time ≠ **CPU使用率**）
    - バックグランド・プロセスは（Oracleデータベース以外も）含まれない
    - CPUリソースの限界は判断できない（CPU%の分母は利用可能なCPU時間）
      - 待機時間が存在してもCPU使用率が100%になるときもある
    - CPU使用率はOS統計から（AWRはスナップショット間の平均値なので判断できない）
- SQL ordered by Elapsed Time（SQL経過時間の合計）
  - SQLで使用したDB Time ≠ **応答時間（Elapsed Time）**
    - パラレル実行では全スレーブ・プロセスの合計時間

AWRからの解析
AWRについて（Report Summary）

- Load Profile
  - Oracleデータベースに対する負荷
    - Oracleデータベースに対する負荷の変化などを監視／比較
- Instance Efficiency Percentages (Target 100%)
  - インスタンス内の効率良さ（100%に近いほど効率が良い）
    - **インスタンス・チューニングの判断**（そうでなければSQLチューニング=>SQL統計: by Elapsed Time ）
    - バッファ・キャッシュ・ヒット率など
- Top 5 Timed Foreground Events
  - フォアグランド・イベント時間（DB Timeの内訳）の上位5（または上位10）

ここで問題は何か（どのイベントが多いのか）を特定する

AWRからの解析
待機イベント（体表的なもの）

- I/O
  - 非ダイレクトI/O, ダイレクトI/Oなど<=第26回
- エンキュー
  - HW(HWM), SQ(シーケンス), TX(行ロック)など<=第18回
- ライブラリ・キャッシュ
  - 共有カーソル関係（cursor: xxx, library cache: xxxなど）<=第32回
  - ライブラリ・キャッシュ・オブジェクト（同じSQLやPL/SQLパッケージの実行）<=第50回
- マニュアルにない待機イベント
  - SRで問い合わせてください

AWRからの解析
待機イベント（I/O）

- 代表的なI/O統計
  - db file sequential read（索引スキャン）
  - db file scattered read / direct path read（フル・スキャン）
  - direct path read temp / direct path write temp（一時表へのI/O）
- I/O性能の確認（問題なければSQLチューニングへ）
  - I/O統計のAv Rd(ms)など
  - OS統計（sar, iostat）の%Busyなど
- SQLの特定
  - SQL統計（SQL ordered by User I/O Wait Time）
- 実行計画の出力（AWRのスナップショットから）
  - $ORACLE_HOME/rdbms/admin/awrsqrpt.sql（Actual Rowsが出力されない）

AWRからの解析
待機イベント（子カーソル）

- 子カーソルとは（**第7回**）
  - 同じSQLで別の実行計画を作成すること（ハード・パースが増える）
- 子カーソルの待機イベント
  - 11.1以上はcursor: mutex関係、10.2以前はlibrary cache pin
  - SQL統計のSQL ordered by Version Count（子カーソルの多いSQL）
- 子カーソルが作成される原因はV$SQL_SHARED_CURSORを参照
  - 代表的なもの
    - カーディナリティ・フィードバック、バイド・ピーク/Adaptive Cursor Sharing（優れたカーソル共有）、バインド変数の属性が異なる、スキーマ・オブジェクトが異なるなど

AWRからの解析
サンプルAWR

- サンプルのAWRを見てみる

Top 5 Timed Foreground Events

Event	Waits	Time(s)	Avg wait (ms)	% DB time	Wait Class
DB CPU		42,936		34.52	
enq: HW - contention	13,247	25,791	1947	20.74	Configuration
log file sync	8,815,233	21,371	2	17.18	Commit
gc current block 2-way	15,379,020	6,745	0	5.42	Cluster
gc current block 3-way	7,377,119	5,755	1	4.63	Cluster

Load Profile

	Per Second	Per Transaction	Per Exec	Per Call
DB Time(s):	34.6	0.0	0.00	0.00
DB CPU(s):	11.9	0.0	0.00	0.00
Redo size:	12,960,945.7	2,168.8		
Logical reads:	341,733.9	57.2		
Block changes:	60,202.1	10.1		
Physical reads:	5,676.7	1.0		
Physical writes:	1,912.1	0.3		
User calls:	11,796.6	2.0		
Parses:	16.0	0.0		
Hard parses:	0.1	0.0		
W/A MB processed:	108.2	0.0		
Logons:	0.5	0.0		
Executes:	10,434.0	1.8		
Rollbacks:	3,513.1	0.6		
Transactions:	5,976.1			

パース時間に対するCPU時間の割合
CPU時間以外（ラッチなど）が多いか

Instance Efficiency Percentages (Target 100%)

Buffer Nowait %:	99.86	Redo NoWait %:	100.00
Buffer Hit %:	99.88	In-memory Sort %:	100.00
Library Hit %:	100.21	Soft Parse %:	99.24
Execute to Parse %:	99.85	Latch Hit %:	99.32
Parse CPU to Parse Elapsd %:	46.91	% Non-Parse CPU:	99.52

Execute to Parse % : パースなし（カーソルキャッシュ）で実行された割合
% non-parse CPU : パース以外で使用されたCPU時間の割合

アジェンダ

1. AWRからの解析
2. SQLチューニング
3. オプティマイザ統計

SQLチューニング

- SQLの基本知識
  - 索引を使用しない条件
  - 結合条件(1=1)、WHERE句条件(1=2)
  - 問合せ変換（Query Transformation）
- 実行計画の見方
  - 必要最低限のもの
- 実行計画のチューニング
  - チューニング手順、SQL、ヒントなど

SQLチューニング
SQLの基礎知識（索引を使用しない条件）

- 演算している（BIツールなどは注意）
- NULL比較
- NOT(!=)
- OR条件（INリスト）
  - OR拡張（UNION ALLに変換）で索引を使用
- 後方一致（中間一致）条件
  - 索引スキップ・スキャン（9iから）
    - 全表スキャンより効果的なとき（先頭の個別値が少ないときなど）に使用

意識して使用する（第9回）

SQLチューニング
SQLの基礎知識（結合条件(1=1)、WHERE句条件(1=2)）

- 異なる結合を同じ構文で行うようなとき（動的SQLで多い）
  - 結合効率が悪いので行わない（第44回）

```
/* すべてを FULL OUTER JOIN で行う */
  …
  IF abc = '01' THEN     -- すべての行にMAX(c1)を入れる
    v_from := '(SELECT MAX(c1) c1 FROM tab2 WHERE c2 = 1) B ON (1=1)';
  ELSIF abc = '02' THEN -- 結合しない
    v_from := '(SELECT c1 FROM tab2 WHERE (1=2)) B ON ON A.c1 = B.c1';
  ELSE                    -- 通常のFULL OUTER JOIN
    v_from := 'tab2 B ON A.c1 = B.c1';
  END IF
  v_stmt_str := 'SELECT … FROM tab1 A FULL OUTER JOIN ' || v_from;
  OPEN v_tab3_cursor FOR v_stmt_str;
  …
```

SQLチューニング
SQLの基礎知識（問合せ変換）

- 無駄な処理の排除
  - 結合（外部キー）、ORDER BY（インライン・ビュー内）、DISTINCT（主キー）<=**第34回**
- ビュー（インライン・ビュー）の最適化（次頁）
- 副問合せのネスト解除（セミ結合、アンチ結合）
  - INの方が制約が多いので**EXISTS**を使用する（**第29回**）
  - OR条件で変換されないときがある（**第44回**、**第52回**）
- UNION ALLに変換（OR拡張、表拡張<=索引が使用可と使用不可のパーティションに）
  - 索引を使用するように（**第9回**、**第34回**）
  - パラレル実行時のOR拡張は一部シリアル処理に（**第52回**）

SQLチューニング
SQLの基礎知識（ビュー、インライン・ビュー）

- ビューとは
  - アクセス制御などで使用（条件は外で指定）
  - 最初に実行される
- 特性（メリット／デメリット）
  - 結合順が調整できない <= ×
    - 例のtab1は最後の結合に
  - ビュー実行後は索引が使用できない <= ×
    - 例のA.c1=B.c1をB.c1の索引を使用したネステッド・ループ結合にできない
  - 結合前に行数を削減できる（DISTINCT、Group Byなど）<=○
    - 例のインライン・ビューにGroup byがなくても結果は同じ（主問合せでGroup byしているから）

- ビューの問合せ変換
  - ビューを使用しない
    - 結合順を変えたい、索引を使用したい
    - View Merging（ビュー・マージ）<=第29回
  - ビューが効果的なとき（ビューを作成）
    - Group by Placement （Group by の配置）<=第42回
  - ビューをマージできない／しない
    - Group by後にネステッド・ループ結合するなど
    - Predicate Pushing（述語のプッシュ）<=第29回

```
SQL> SELECT A.c2, SUM(b2), SUM(b3) FROM tab1 A,
  2    (SELECT C.c1 b1, SUM(C.c2) b2, SUM(D.c2) b3
  3       FROM tab2 C,tab3 D
  4      WHERE C.c1 = D.c1 AND C.c3 < 100
  5      GROUP BY C.c1) B
  6   WHERE A.c1 = B.c1 GROUP BY A.c2 ;
```

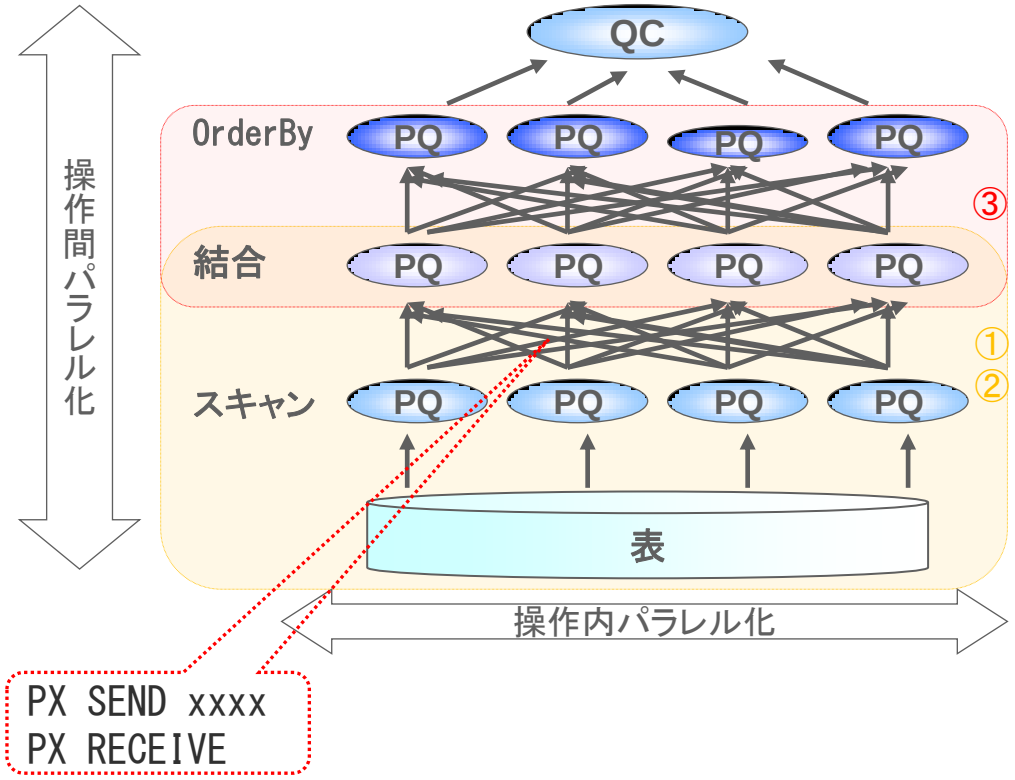
SQLチューニング
実行計画の見方（注目する項目）

- カーディナリティ
  - 述語を適用した行数（Rows、E-Rows、A-Rows）
- アクセス方法
  - 索引スキャン、全表スキャン、ビュー・アクセス
- 結合方法
  - ネステッド・ループ結合、ハッシュ結合、ソート・マージ結合、直積結合
- 結合タイプ
  - 内部結合、外部結合、セミ結合、アンチ結合
- 結合順序
  - SQLによって決まる場合も

- パーティション
  - パーティション・プルーニング（第22回、第46回）
    - 静的（アクセスする番号）
    - 動的（KEY、KEY(OR)など）
    - 11gR2からKEY(AP)が追加
      - AND Pruning（静的＋動的）
- パラレル実行
  - スキャンのデータ分散（第20回、第39回）
    - ブロック単位で分割（PX BLOCK ITERATOR）
    - パーティション単位（PX PARTITION RANGE ALL など）
      - パーティション・ワイズ結合など

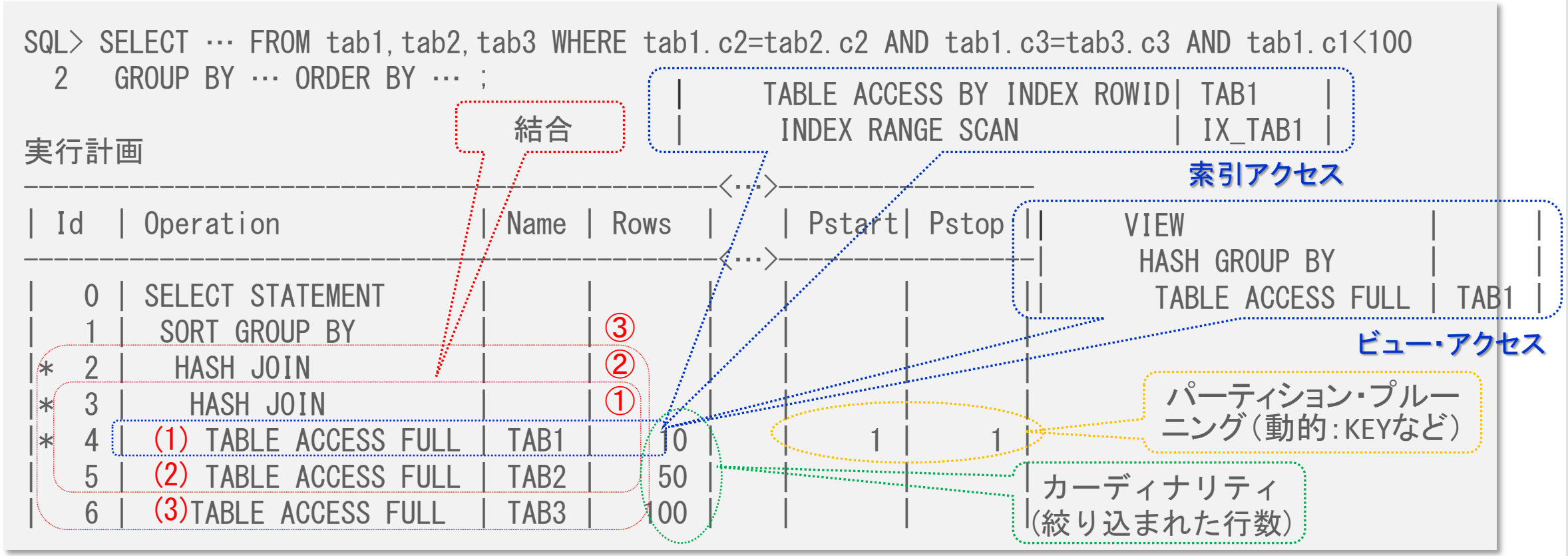
SQLチューニング
実行計画の見方（注目する項目）

- パラレル実行（続き）
  - スキャン以外のデータ再分散（第20回、第39回）
    - HASH（基本はこれを使用する）
    - RANGE（ソートなど）<=偏りやすい
    - BROADCAST（結合時の片方が小さい）
      - 同じデータをすべてのプロセスに
    - PARTITION（パーティション・ワイズ結合）
      - パーティション分割（片方がパーティション表のとき）
    - NONE（データ分散しない／データ通信しない）
      - PX SEND xxxxとPX RECEIVEがない
      - PARTITIONやBROADCASTなどと使用
    - 偏りが発生すると効果が低下する
      - PQ_DISTRIBUTEヒントで調整



SQLチューニング
実行計画の見方

- リーフ・ステップ(インデントの一番深いステップ)から実行して、結合(同一インデント)は上位に表示されたものが最初になる



SQLチューニング
実行計画の見方(パラレル実行)

```
SQL> SELECT * FROM tab01,tab02 WHERE tab01.c1 = tab02.c1 ORDER BY c0;

実行計画                    結合プロセス(同じ)
----------------------------------------------------------<...>-------------------------------
| Id  | Operation                      | Name     |      |    TQ  |IN-OUT| PQ Distrib |
----------------------------------------------------------<...>-------------------------------
|   0 | SELECT STATEMENT               |          |      |        |      |            |
|   1 |  PX COORDINATOR                |          |      |        |      |            |
|   2 |   PX SEND QC (ORDER)           | :TQ10003 |      | Q1,03  | P->S | QC (ORDER) |
|   3 |    SORT ORDER BY               |          |      | Q1,03  | PCWP |            |
|   4 |     PX RECEIVE                 |          |      | Q1,03  | PCWP |            |
|   5 |      PX SEND RANGE             | :TQ10002 |      | Q1,02  | P->P | RANGE      |
|   6 |       HASH JOIN BUFFERED       |          |      | Q1,02  | PCWP |            |
|   7 |        PX RECEIVE              |          |      | Q1,02  | PCWP |            |
|   8 |         PX SEND HASH           | :TQ10000 |      | Q1,00  | P->P | HASH       |
|   9 |          PX BLOCK ITERATOR     |          |      | Q1,00  | PCWC |            |
|  10 |           TABLE ACCESS FULL    | TAB01    |      | Q1,00  | PCWP |            |
|  11 |         PX RECEIVE             |          |      | Q1,02  | PCWP |            |
|  12 |          PX SEND HASH          | :TQ10001 |      | Q1,01  | P->P | HASH       |
|  13 |           PX BLOCK ITERATOR    |          |      | Q1,01  | PCWC |            |
|  14 |            TABLE ACCESS FULL   | TAB02    |      | Q1,01  | PCWP |            |
```

PX SEND PARTITION
PX SEND BROADCAST
データ分散

PX PARTITION RANGE ALL
スキャンの分散

SQLチューニング
実行計画の見方

- 後半には、述語の情報とNote部が出力される

```
実行計画
----------------------------------------
| Id  | Operation             | Name |
----------------------------------------
|   0 | SELECT STATEMENT      |      |
|   1 |  SORT GROUP BY        |      |
|*  2 |   HASH JOIN           |      |
|*  3 |    HASH JOIN          |      |
|*  4 |     TABLE ACCESS FULL | TAB1 |
|   5 |     TABLE ACCESS FULL | TAB2 |
...
Predicate Information (identified by operation id):
---------------------------------------------------

   2 - access("TAB1"."C2"="TAB2"."C2")
   3 - access("TAB1"."C3"="TAB3"."C3")
   4 - filter("TAB1"."C1"<100)

Note
-----
   - dynamic sampling used for this statement (level=2)
```

述語の情報

Note部

SQLチューニング
実行計画の見方（ネステッド・ループ結合）

- 通常のネステッド・ループ結合
  - 基本は内部表の索引を使用して結合
    - 駆動表の1行に対して索引アクセス
- Multi Join Key Pre-fetching（9*i*から）**第34回**
  - 索引範囲スキャンのデータ・ブロックの先読み
- Nested Loops Join Batching（11gから）**第34回**
  - 索引で結合後にROWIDを並べ替えてテーブルにアクセス
  - 索引一意スキャンでも可

```
SQL> SELECT … FROM tab1,tab2 WHERE tab1.c1 = tab2.c1 GROUP BY … ;

実行計画
----------------------------------------------------
| Id  | Operation                     | Name     |
----------------------------------------------------
|   0 | SELECT STATEMENT              |          |
|   1 |  HASH GROUP BY                |          |
|   2 |   NESTED LOOPS                |          |
|   3 |    TABLE ACCESS FULL          | TAB2     | <- 駆動表
|   4 |    TABLE ACCESS BY INDEX ROWID| TAB1     | <- 内部表
|*  5 |     INDEX RANGE SCAN          | IX_TAB1  |
----------------------------------------------------

----------------------------------------------------
|   0 | SELECT STATEMENT              |          |
|   1 |  HASH GROUP BY                |          |
|   2 |   TABLE ACCESS BY INDEX ROWID| TAB1     | 内部表をPer-fetchする
|   3 |    NESTED LOOPS               |          |
|   4 |     TABLE ACCESS FULL         | TAB2     | 駆動表
|*  5 |     INDEX RANGE SCAN          | IX_TAB1  | 内部表
----------------------------------------------------

----------------------------------------------------
|   0 | SELECT STATEMENT              |          |
|   1 |  HASH GROUP BY                |          |
|   2 |   NESTED LOOPS                |          | Nested Loops Join(2)
|   3 |    NESTED LOOPS               |          | Nested Loops Join(1)
|   4 |     TABLE ACCESS FULL         | TAB2     |  駆動表(1)
|*  5 |     INDEX RANGE SCAN          | IX_TAB1  |  内部表(1)
|   6 |    TABLE ACCESS BY INDEX ROWID| TAB1     | 内部表(2)(ここを改善)
----------------------------------------------------
                                                  結果を駆動表(2)として
```

SQLチューニング
実行計画の見方（ソート・マージ結合、直積結合）

- ソート・マージ結合
  - 索引の代わりにソートして結合
  - 等価結合以外など

```
SQL> SELECT … FROM tab1,tab2 WHERE tab1.c1 > tab2.c1
  2   GROUP BY … ;

実行計画
----------------------------------------------
| Id  | Operation            | Name | Rows  |
----------------------------------------------
|   0 | SELECT STATEMENT     |      |       |
|   1 |  HASH GROUP BY       |      |       |
|   2 |   MERGE JOIN         |      |   xxx |
|   3 |    SORT JOIN         |      |   100 |
|   4 |     TABLE ACCESS FULL| TAB2 |   100 |
|*  5 |    SORT JOIN         |      |   100K|
|   6 |     TABLE ACCESS FULL| TAB1 |   100K|
```

- 直積結合
  - できるだけ行わない
  - 結合条件がないので効率が悪い

```
SQL> SELECT … FROM tab1,tab2 GROUP BY … ;

実行計画
----------------------------------------------
| Id  | Operation            | Name | Rows  |
----------------------------------------------
|   0 | SELECT STATEMENT     |      |       |
|   1 |  HASH GROUP BY       |      |       |
|   2 |   MERGE JOIN CARTESIAN|      |       |
|   3 |    TABLE ACCESS FULL | TAB2 |   100 |
|   4 |    BUFFER SORT       |      |   100K|
|   5 |     TABLE ACCESS FULL| TAB1 |   100K|
```

SQLチューニング
実行計画の見方（ハッシュ結合、外部結合）

- ハッシュ結合（等価結合のみ）
  - 索引の代わりにメモリ上にハッシュ・テーブルを作成（最初にアクセスした表に）
  - スター・スキーマはRight-deep Joinが効果的（SWAP_JOIN_INPUTSヒント）<=第46回

```
SQL> SELECT … FROM tab1,tab2,tab3
  2   WHERE tab1.c1=tab2.c1 AND tab1.c2=tab3.c2
  3     AND tab2.c3=xxx AND tab3.c2=xxx GROUP BY … ;
```

実行計画（Left-deep Join）

Id	Operation	Name
0	SELECT STATEMENT	
1	HASH GROUP BY	
* 2	HASH JOIN	
* 3	HASH JOIN	
* 4	TABLE ACCESS FULL	TAB2
5	TABLE ACCESS FULL	TAB1
* 6	TABLE ACCESS FULL	TAB3

実行計画（Right-deep Join）

Id	Operation	Name
0	SELECT STATEMENT	
1	HASH GROUP BY	
* 2	HASH JOIN	
* 3	TABLE ACCESS FULL	TAB3
* 4	HASH JOIN	
* 5	TABLE ACCESS FULL	TAB2
6	TABLE ACCESS FULL	TAB1

- 外部結合
  - LEFT OUTER JOIN, RIGHT OUTER JOIN
  - ハッシュ結合で核でないテーブルを先にアクセス可能に（10gから）

```
SQL> SELECT … FROM tab1 LEFT OUTER JOIN tab2 USING (c1) GROUP BY … ;
```

実行計画

Id	Operation	Name
0	SELECT STATEMENT	
1	HASH GROUP BY	
2	HASH JOIN OUTER	
3	TABLE ACCESS FULL	TAB1
4	TABLE ACCESS FULL	TAB2

実行計画（Oracle Database 10gから）

Id	Operation	Name
0	SELECT STATEMENT	
1	HASH GROUP BY	
2	HASH JOIN RIGHT OUTER	
3	TABLE ACCESS FULL	TAB2
4	TABLE ACCESS FULL	TAB1

SQLチューニング
実行計画の見方（セミ結合、アンチ結合）

- セミ結合
  - EXISTS, IN条件の副問合せ
  - セミ・ハッシュ結合で副問合せのテーブルを先にアクセス可能に（10gから）

```
SQL> SELECT … FROM tab1 WHERE EXISTS
  2   (SELECT 0 FROM tab2 WHERE tab1.c1 = tab2.c1) ;

実行計画
----------------------------------
| Id  | Operation          | Name |
----------------------------------
|   0 | SELECT STATEMENT   |      |
|   1 |  HASH JOIN SEMI    |      |
|   2 |   TABLE ACCESS FULL| TAB1 |
|   3 |   TABLE ACCESS FULL| TAB2 |
```

```
実行計画（Oracle Database 10gから）
----------------------------------
| Id  | Operation          | Name |
----------------------------------
|   0 | SELECT STATEMENT   |      |
|   1 |  HASH JOIN RIGHT SEMI|    |
|   2 |   TABLE ACCESS FULL| TAB2 |
|   3 |   TABLE ACCESS FULL| TAB1 |
```

- アンチ結合
  - NOT EXISTS, NOT IN条件の副問合せ
  - アンチ・ハッシュ結合で副問合せのテーブルを先にアクセス可能に（10gから）

```
SQL> SELECT … FROM tab1 WHERE NOT EXISTS
  2   (SELECT 0 FROM tab2 WHERE tab1.c1 = tab2.c1) ;

実行計画
----------------------------------
| Id  | Operation          | Name |
----------------------------------
|   0 | SELECT STATEMENT   |      |
|   1 |  HASH JOIN ANTI    |      |
|   2 |   TABLE ACCESS FULL| TAB1 |
|   3 |   TABLE ACCESS FULL| TAB2 |
```

```
実行計画（Oracle Database 10gから）
----------------------------------
| Id  | Operation          | Name |
----------------------------------
|   0 | SELECT STATEMENT   |      |
|   1 |  HASH JOIN RIGHT ANTI|    |
|   2 |   TABLE ACCESS FULL| TAB2 |
|   3 |   TABLE ACCESS FULL| TAB1 |
```

SQLチューニング
実行計画のチューニング

- チューニングの手順
  1. オプティマイザ統計の再収集
  2. 索引の作成（索引のチューニング）
  3. SQLの変更（ヒントの追加）
- SQLの変更ができない
  - SPM（SQL Plan Management）のSQL計画手動ロード（第38回）
    - ヒントを入れた実行計画をベースラインとして登録する
  - SQL翻訳フレームワーク（第51回）
    - SQLを変更したいとき（SQLの置き換え方法を登録）
- 問題を特定できない
  - SQLチューニング・アドバイザ（第38回）

SQLチューニング
問題となる実行計画（主なSQL）と対処

問題点	統計	ヒント	SQLなどの変更
結合方法や結合順が最適でない	○	○	副問合せの追加/削除など
問合せ変換が最適でない	○	○	明示的にSQLを変更
索引が使用できていない	○	○	索引を使用できるように変更、索引を作成
同じような副問合せがある	×	×	SQLを分割する（WITH句を使用する）第11回
同じ表に異なる条件でSELECTしている	×	×	SQLをCASE式で1つに第24回
同じ表に異なる条件でINSERTしている	×	×	SQLをマルチ・テーブル・インサートに（同じ表でも可）第24回
同じ表をUPDATE文とINSERT文でアクセス	×	×	SQLをMERGE文で1つに第30回
結合したUPDATE文	×	×	SQL（複雑な副問合せを使用する）をMERGE文に第30回
Redoログ出力がネック（特にパラレルDML）	×	×	ダイレクト・パス・インサートに（UPDATE、DELETEも）第15回
TEMP領域を使用	×	×	• パラレル度を上げる、プログラムを分割／並列化 • 索引を使用する（結合、ソートなど）　第45回

SQLチューニング
オプティマイザ統計の再収集

- 行数の見積もりが正しくない（実行時と大きく異なるとき）
  - 実行計画の見積もり行数（E-Rows）と実行行数（A-Rows）を比較

```
SQL> SELECT /*+ GATHER_PLAN_STATISTICS */ c1,c2,c3 FROM tab01 WHERE c1 = 11 ;

レコードが選択されませんでした。

SQL> SELECT * FROM TABLE(dbms_xplan.display_cursor(format=>'typical allstats last'));

実行計画
----------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name     | Starts | E-Rows |E-Bytes| ... | A-Rows | ... |
----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |          |      1 |        |       | ... |      0 | ... |
|   1 |  TABLE ACCESS BY INDEX ROWID| TAB01    |      1 |     21 |    189| ... |      0 | ... |
|*  2 |   INDEX RANGE SCAN           | IX_TAB01 |      1 |     21 |       | ... |      0 | ... |
```

SQLチューニング
実行計画の確認(DBMS_XPLAN.DISPLAY_CURSOR関数)

- 見積もり以外に実行時の統計も出力できる(STATISTICS_LEVEL=ALLまたはヒント)

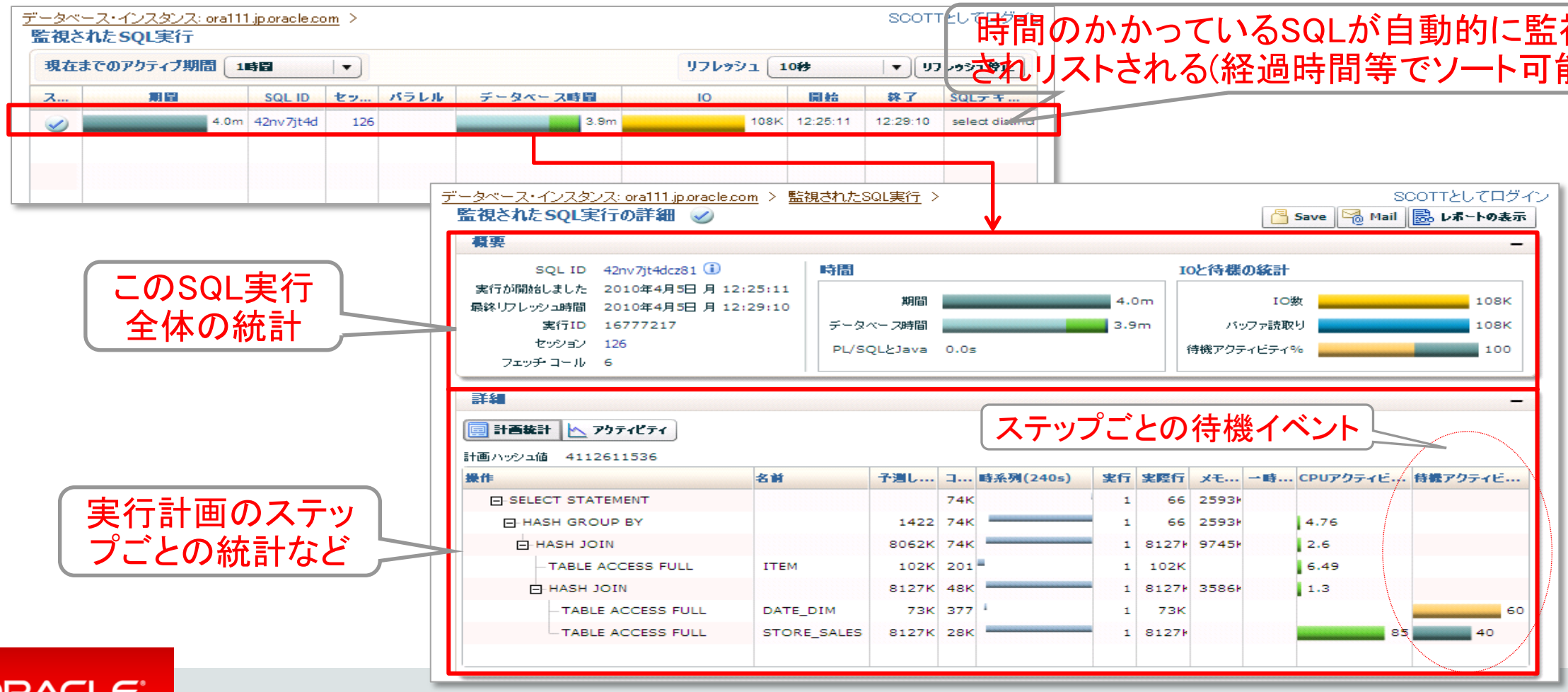
```
SQL> SELECET /*+ GATHER_PLAN_STATISTICS */ … ;
SQL> SELECT * FROM TABLE(dbms_xplan.display_cursor(sql_id=>'<sql_id>', format=>'typical allstats last'));

実行計画
----------------------------------------------------------------------------------------------------
| Id | Operation        | Name | Starts| E-Rows|E-Bytes| Cost (%CPU)| E-Time   | A-Rows|   A-Time |
----------------------------------------------------------------------------------------------------

-----------------------------------------
 Buffers|  OMem|  1Mem| Used-Mem| Used-Tmp|
-----------------------------------------
```

省略すると最後のSQL

＜実行統計＞
Starts：実行された回数
Buffers：バッファのアクセス数
A-Rows：処理行数
A-Time：処理時間
Used-Mem：使用されたメモリサイズと処理方法 (0:Optimal,1:1-pass,N:Multi-pass)
Used-Tmp：使用されたTEMPサイズ

SQLチューニング
実行計画の確認(リアルタイムSQL監視)

- 実行中の実行計画を表示(デフォルトで5秒以上のSQL、MONITORヒントで強制的に)



SQLチューニング
オプティマイザ統計の再収集

- サンプル・サイズを大きくしても精度が上がらないとき
  - ヒストグラムと拡張時計を確認
- 拡張統計は自動的に作成されない<=12cで改善
  - 列グループ（ないときは列値の組合せは均等となる）
    - WHERE c1 = xxx AND c2 = xxx（フィルター条件、結合）
    - GROUP BY c1,c2
  - 式（BIツールを使用しているときなど）
    - WHERE UPPER(c1) = xxx
- 一意値が多い列のヒストグラムには限界がある
  - バケット数の最大が254（12cから2048に拡張）
  - 列サイズも12cから拡張（32バイトから64バイト）

SQLチューニング
索引の作成（索引のチューニング）

- 全表スキャンを行っているテーブルは索引スキャンした方が良くないか
  - 実行計画のRowsを確認
- 索引スキャンの効率が悪くないか
  - 索引の列の組合せ、索引の列の順番などを確認
    - 例えば、以下のようなSQL

```
SQL> SELECT … WHERE c1=xx AND c2=xx AND c3=xx;
SQL> SELECT … WHERE c1=xx AND c3=xx;
SQL> SELECT … WHERE c1=xx AND c2=xx AND c4=xx;
```

    - 優先順位を明確にして作成する
      - これが難しい（そのため、フル・スキャンでも高速なExadataが効果的）
- 作り過ぎないように使用しない索引は削除する

SQLチューニング
SQLの変更（主なSQLの変更）

- 結合のUPDATE文をMERGE文に

```
SQL> UPDATE t01 A
  2     SET A.c3 = A.c3 + (SELECT c2 FROM t02 B WHERE A.c1 = B.c1)
  3   WHERE EXISTS (SELECT 0 FROM t02 B WHERE A.c1 = B.c1);
```

⇩

```
SQL> MERGE INTO t01 A
  2  USING (SELECT c1,c2 FROM t02) B
  3  ON (A.c1 = B.c1)
  4  WHEN MATCHED THEN UPDATE SET A.c3 = A.c3 + B.c2;
```

- 繰り返し副問合せはWITH句で

```
SQL> SELECT * FROM
  2   (SELECT 部門,sum(売上) 部門売上 FROM 売上表 GROUP BY 部門) w_A
  3     WHERE 部門売上 < (SELECT avg(部門売上) FROM
  4       (SELECT 部門,sum(売上) 部門売上 FROM 売上表 GROUP BY 部門));
```

⇩

```
SQL> WITH w_A AS (SELECT 部門, sum(売上) FROM 売上表 GROUP BY 部門)
  2    SELECT * FROM w_A
  3     WHERE 部門売上 < (SELECT avg(部門売上) FROM w_A);
```

- UPDATE,DELETEもダイレクト・インサートに

```
SQL> DELETE FROM tab000
  2   WHERE 日付 < TO_DATE( '20101001' ,' YYYYMMDD' ) ;
```

⇩

```
SQL> CREATE TABLE tab001 NOLOGGING PARALLEL AS
  2    SELECT * FROM tab000
  3     WHERE 日付 >= TO_DATE( '20101001' ,' YYYYMMDD' ) ;
SQL> DROP TABLE tab000 ;
SQL> RENAME tab001 TO tab000 ;
```

SQLチューニング
SQLの変更(ヒント)

- 表の別名があるときは別名を

```
SQL> SELECT /*+ INDEX(A ix_tab1) */ * FROM tab1 A WHERE … ;
```

- ビュー(副問合せ)内の表は参照しない

```
SQL> SELECT /*+ INDEX(tab1 ix_tab1) */ *
  2    FROM (SELECT * FROM tab1 WHERE … ) A ;
```

```
SQL> SELECT /*+ INDEX(A.tab1 ix_tab1) */ *
  2    FROM (SELECT * FROM tab1 WHERE … ) A ;
```

- 副問合せ内でも指定できる(ビュー・マージされて使用される)

```
SQL> SELECT * FROM (SELECT /*+ INDEX(tab1 ix_tab1) */ *
  2                   FROM tab1 WHERE … ) A ;
```

- 主に使用するヒント
  - 結合順を変える(ORDERED, LEADING)
  - ビュー・マージを止める(NO_MERGE)
  - 索引を使用する(INDEX, INDEX_FFS)
  - 索引を使用しない(FULL, NO_INDEX)
  - 結合方法を変える(USE_HASH, USE_MERGE, USE_NL)
  - パラレル実行関係(PARALLEL, PQ_DISTRIBUTE)
  - 問合せ変換関連のヒント

アジェンダ

1. AWRからの解析
2. SQLチューニング
3. オプティマイザ統計

オプティマイザ統計

- 統計の種類
  - どのように使用されるか
- 収集方法
  - 収集方法とデフォルトで収集されるもの
- 統計の補正
  - どこまで行えるか

オプティマイザ統計
統計の種類

- 表統計 ⇨ **表アクセス・コスト**
  - 行数、データ・ブロック数、行連鎖・行移行の数、平均行長
- 索引統計 ⇨ **索引アクセス・コスト**
  - 索引内の個別値数、索引の深さ(BLEVEL)、リーフ・ブロック数、クラスタ化係数
- 列統計 ⇨ **カーディナリティ、サイズ**
  - 個別値数、NULL数、平均列データ長、データ分布(最小値と最大値、ヒストグラム)
- システム統計(I/O+CPUコスト・モデル)
  - CPU性能、I/O性能(単一ブロック・リード、マルチ・ブロック・リード)

パーティション表には、パーティション、サブ・パーティションごとにも同様の情報を収集する

アクセスする行数やブロック数を求め、CPU性能やI/O性能でコストを計算する

オプティマイザ統計
統計の種類

- 列統計（ヒストグラム）
  - ヒストグラムは完全には設定できないものも（12cで拡張）
    - 異なる値が255以上の高さ調整済ヒストグラム
      - バケット数が最大254なので、各値を別バケットに入れられない
  - 拡張統計（自動作成されない）
- システム統計（CPU性能、I/O性能）
  - Exadataは固有のシステム統計を取得する（DB稼働後一度だけで良い）
    - DBMS_STATS.GATHER_SYSTEM_STATS('EXADATA');
  - Exadata以外
    - 初回起動時のデフォルト値（単一ブロック・リード、マルチ・ブロック・リード時間などがない）で良いが、索引スキャンとフル・スキャンを使用する場合は一度収集した方が良い

オプティマイザ統計
統計の種類（拡張統計）

- 列グループの統計（Column Groups）
  - 同一表内の複数列に跨る統計を保持することで、列データ間の相関関係を考慮したカーディナリティの計算を可能とする
  - 例：顧客表の住所列と年代列など
  - 12cからのSQL計画ディレクティブ（**第52回**）と自動列グループ検出（**第49回**）
- 式の統計（Expression Statistics）
  - 関数を含めた統計を保持することで、関数に組み込まれた列を持つWHERE句におけるカーディナリティの計算を可能する
  - 例：Where UPPER（氏名) = :B1

オプティマイザ統計
統計収集

- 自動オプティマイザ統計収集
  - 22:00から26:00（土日は6:00から26:00）に自動実行する
  - 独自の方式で収集しているシステム以外はこれを使用する
  - ディクショナリだけの自動収集も可能
    - DBMS_STATS.SET_GLOBAL_PREFS('AUTOSTATS_TARGET','ORACLE')
- オプティマイザ統計の手動収集
  - 必要な（大量に変更があった）ときを判断して実行する
  - ディクショナリの手動収集
    - DDLを多く発行された後などに実行
      - DBMS_STATS.GATHER_DICTIONARY_STATS

オプティマイザ統計
統計収集（自動オプティマイザ統計収集）

- 自動的に収集するものを決める
  - 10%以上変更されたオブジェクト
- デフォルト値
  - 収集する適切なサイズをOracleが決定（AUTO_SAMPLE_SIAZE）
    - Oracle11gで拡張されたハッシュ・アルゴリズム（サンプリングより高速、Computeモード統計と同等精度）
  - 索引統計の収集を自動判断（AUTO_CASCADE）
  - ヒストグラムの収集を自動判断（FOR ALL COLUMNS SIZE AUTO）
  - カーソルの無効化の時期をOracleが決定（AUTO_INVALIDATE）
    - 収集後に共有プール上の実行計画が正しくなくなる
- ディクショナリもデフォルトで収集される

オプティマイザ統計
統計収集（自動オプティマイザ統計収集）

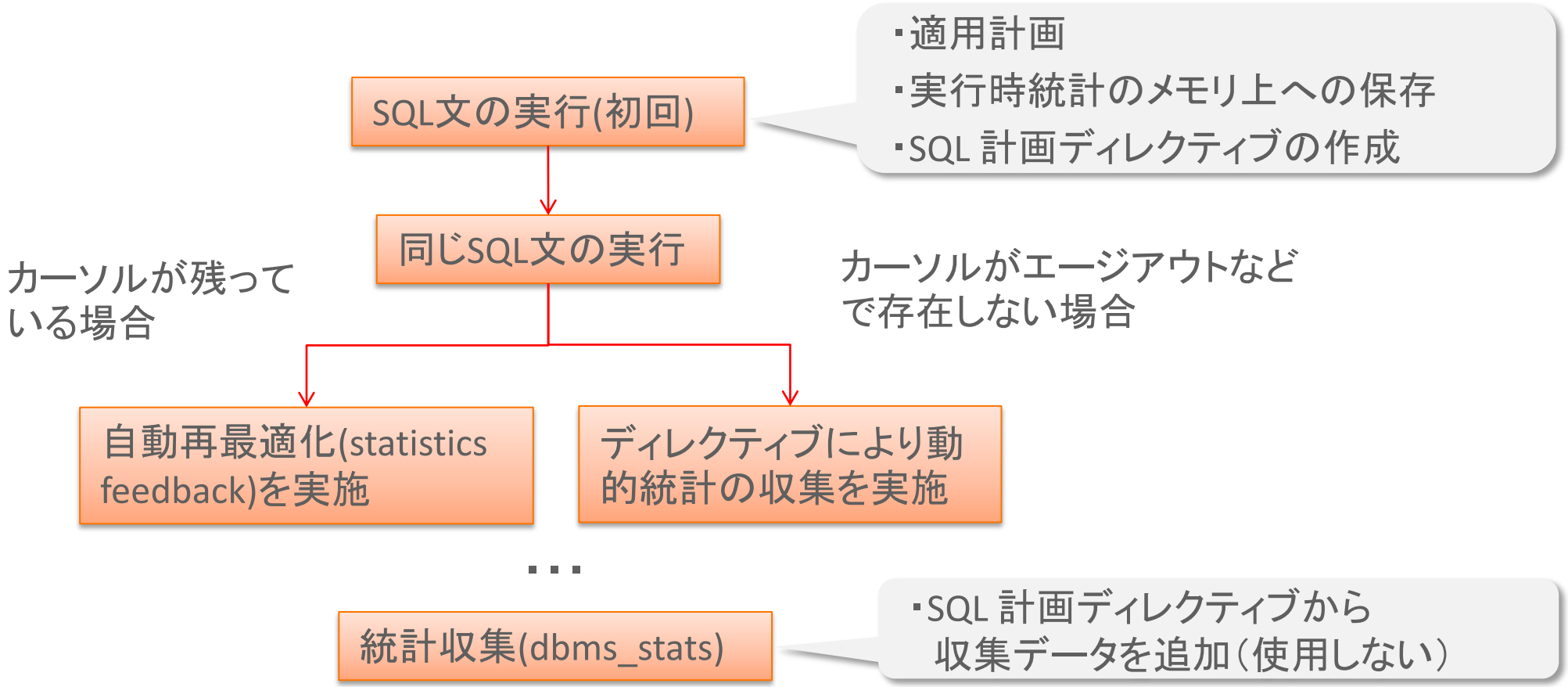
- デフォルトで収集しないものがあるので注意
  - 拡張統計
    - 列グループ統計、式の統計
  - 固定オブジェクト
    - 動的パフォーマンス・ビュー（v$ビュー）の実表（x$表）
    - アプリケーションの変更やデータベース構成の変更の時に手動で再収集する
      - DBMS_STATS.GATHER_FIXED_OBJECTS_STATS;
    - AWRやStatspackでも使用されているので、取得に時間が掛るような時も
  - 一時表（Global Temporary Table）
    - 正しく収集できないので動的サンプリングを使用する
      - 12cからセッション固有統計が提供されたので手動でも収集しやすくなった<=第35回

オプティマイザ統計
統計の補正

- 動的サンプリング（11gまで）
  - 11gR2からパラレル実行時にレベルを自動決定（大きな表、複雑なWHERE句）
- カーディナリティ・フィードバック（11gまで）
  - 見積りと実行時の統計が大きく異なると記録して2回目から使用する
- 適応問合せ最適化（Adaptive Query Optimization）<=12cから（第33回）
  - 適応計画（Adaptive Plans）・・・実行時の最適化
    - 結合方法（Join Methods）
    - パラレル分散方法（Parallel Distribution Methods）
  - 適応統計（Adaptive Statistics）・・・次回実行時以降の最適化
    - 動的統計（Dynamic Statistics）
    - 自動再最適化（Automatic Reoptimization）
    - SQL計画ディレクティブ（SQL Plan Directives）<=第52回

オプティマイザ統計
統計の補正（適応問合せ最適化）

- SQL文実行時に統計が不十分な場合の動作



最後に

- AWRはDB Timeベースで解析、
  まずは「Report Summary」から
- SQLチューニングは基本的なレベルから、
  BIツールなどのSQL自動生成ツールに注意
- オプティマイザ統計は必要なものを正しく設定する、
  限界があることも忘れずに

- よろしければ「津島博士のパフォーマンス講座」を読んでみてください。

Safe Harbor Statement

The preceding is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Integrated Cloud

Applications & Platform Services


ORACLE®