

ORACLE®

ORACLE®

SQLチューニングに 必要な考え方と最新テクニック

日本オラクル株式会社
テクノロジーコンサルティング統括本部
シニアコンサルタント
柴田 歩



日本オラクル、今年最大の技術トレーニング・イベント

**Oracle DBA &
Developer Day 2013**

以下の事項は、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。

OracleとJavaは、Oracle Corporation 及びその子会社、関連会社の米国及びその他の国における登録商標です。文中の社名、商品名等は各社の商標または登録商標である場合があります。

軽く自己紹介

- 日本オラクル株式会社
テクノロジーコンサルティング統括本部
DBアーキテクト部
シニアコンサルタント
柴田 歩(しばた あゆむ)
- シバタツさん、しばちょうさんに続く3人目の柴田！
- 2007年4月に中途で入社
- 某小売業(DB全般) → 某証券業(DB&GI全般) → 某小売業(OVM x86)
→ 某製造業(DB全般・Exadata・EMGC11g) → 某人材派遣業
(SuperCluster・DB&GI全般・EMCC12c) → 某企業(Exadata PoC・RAT)
… と、主にDB廻りのプロジェクトを歴任

本セミナーの趣旨

- 本セミナーは、SQLチューニングを実施する上で有効となる「テクニック」の紹介を主眼に置いています。
 - SQLチューニングの「戦略」ではなく、「戦術」を紹介していきます。
- そのテクニックを紹介する前に、テクニックを活用する上で必要となる、前提条件としての考え方を説明します。

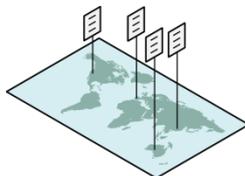
Program Agenda

- 1章. SQLと云う言語の特徴について
- 2章. 「全体最適」と「個別最適」の必要性
- 3章. 「予測」と「実測」の乖離を補正せよ！
 - OracleDBの機能を活用した最新テクニック
- 4章. まとめ

1章. SQLと云う言語の特徴について

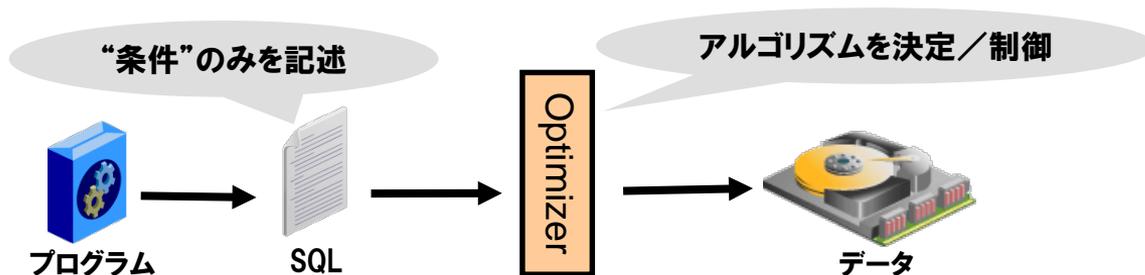
SQLという言葉の特徴(1)

- SQLと云う言語は、過去を振り返ってみても最も成功した言語／標準規格の一つと言えます。
- 何故成功したのか？それはどうやって(How)データを抽出するかを書かずに、何を(What)抽出するかという“条件”のみを記述する事です。
 - 多くの言語では、繰り返し(for～)や条件分岐(if～)などのアルゴリズムを記述する必要があります。
 - アルゴリズムを書かない(※書く必要が無い)のが、SQLと云う言語の最大の特徴です。



SQLという言葉の特徴(2)

- どうやって(How)データを抽出するか、即ちアルゴリズムの決定・制御は、RDBMS のオプティマイザで行われます。
 - オプティマイザが決定したアルゴリズム=実行計画です。
 - オプティマイザはSQLテキストや統計情報などを基に、適切な実行計画を予測して組み立てます。



アルゴリズムを書かないことによるメリット

- アルゴリズムを書かないことによる最大のメリット...

それは言語の習得が「**簡単**」だということです。

- エンジニアではない現場のお客様でも、SQLなら書けるという方は結構いらっしゃいます。
- 習得が簡単ということは、生産性の高さ＝コスト削減にも繋がります。
 - エンジニアの確保が容易
 - アプリケーション作成の工数も少なく済みます。(※一般論ですが)



アルゴリズムを書かないことによるデメリット

- アルゴリズムを書かないことによる最大のデメリット...

それは「**性能**」に関する問題が出やすい事です。

- 適切ではないアルゴリズム(実行計画)による、著しい性能劣化
- アルゴリズム(実行計画)変動に伴う、性能変動(劣化)
- 加えてSQLを使うユーザ(開発者)は、性能が悪くなり易い
良くない実行計画となるSQLを書いてしまいがちです。
 - 多くのユーザは必要なデータが抽出できれば良く、実行計画など気にしない。
 - 一部ユーザが無意識に強烈な実行計画のSQLをリリースすることも、、、



参考：某チューニング案件の超巨大SQL実行計画

- SQLテキストで6700行以上、40表以上を結合したSELECT文の実行計画
 - 実行計画のステップ数換算で、実に500ステップ以上の超巨大SQL

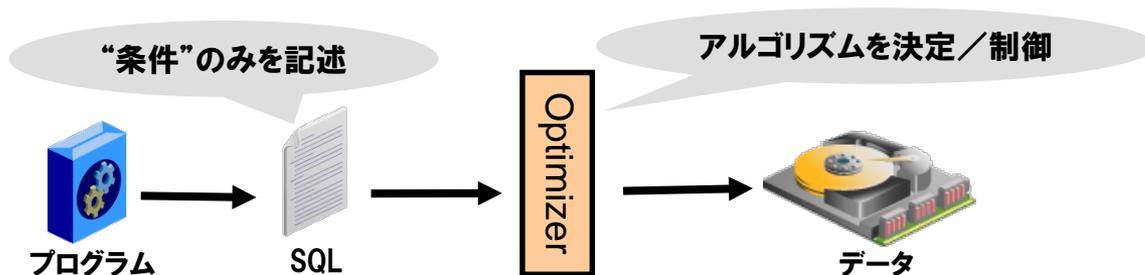
Id	Operation	Name
0	MERGE STATEMENT	
1	MERGE	
2	VIEW	
3	NESTED LOOPS OUTER	
4	VIEW	
5	TEMP TABLE TRANSFORMATION	
6	LOAD AS SELECT	
7	PARTITION LIST SINGLE	
8	TABLE ACCESS STORAGE FULL	
9	PX COORDINATOR	
10	PX SEND QC (RANDOM)	
11	LOAD AS SELECT	
12	HASH JOIN	
13	BUFFER SORT	
14	PX RECEIVE	
15	PX SEND BROADCAST	
16	VIEW	
17	TABLE ACCESS STORAGE FULL	
18	PX BLOCK ITERATOR	
19	TABLE ACCESS STORAGE FULL	
20	PX COORDINATOR	
21	PX SEND QC (RANDOM)	
22	LOAD AS SELECT	
23	HASH JOIN	
24	BUFFER SORT	
25	PX RECEIVE	
26	PX SEND BROADCAST	
27	VIEW	
28	TABLE ACCESS STORAGE FULL	
29	PX BLOCK ITERATOR	
30	TABLE ACCESS STORAGE FULL	
31	SORT ORDER BY	
32	UNION-ALL	
33	FILTER	
34	PX COORDINATOR	
35	PX SEND QC (RANDOM)	

...(中略)...

Id	Operation	Name
508	PARTITION RANGE I	
509	TABLE ACCESS BY	
510	INDEX RANGE SCA	
511	PARTITION RANGE IT	
512	TABLE ACCESS BY L	
513	INDEX RANGE SCAN	
514	PARTITION RANGE ITE	
515	TABLE ACCESS BY LO	
516	INDEX UNIQUE SCAN	
517	BUFFER SORT	
518	PX RECEIVE	
519	PX SEND HASH	
520	TABLE ACCESS STORAGE FU	
521	BUFFER SORT	
522	PX RECEIVE	
523	PX SEND HASH	
524	TABLE ACCESS STORAGE FULL	
525	BUFFER SORT	
526	PX RECEIVE	
527	PX SEND HASH	
528	TABLE ACCESS STORAGE FULL	
529	BUFFER SORT	
530	PX RECEIVE	
531	PX SEND HASH	
532	TABLE ACCESS STORAGE FULL	
533	BUFFER SORT	
534	PX RECEIVE	
535	PX SEND HASH	
536	TABLE ACCESS STORAGE FULL	
537	TABLE ACCESS STORAGE FULL FIRST ROWS	
538	PARTITION RANGE ITERATOR	
539	PARTITION HASH ITERATOR	
540	TABLE ACCESS BY LOCAL INDEX ROWID	
541	INDEX RANGE SCAN	

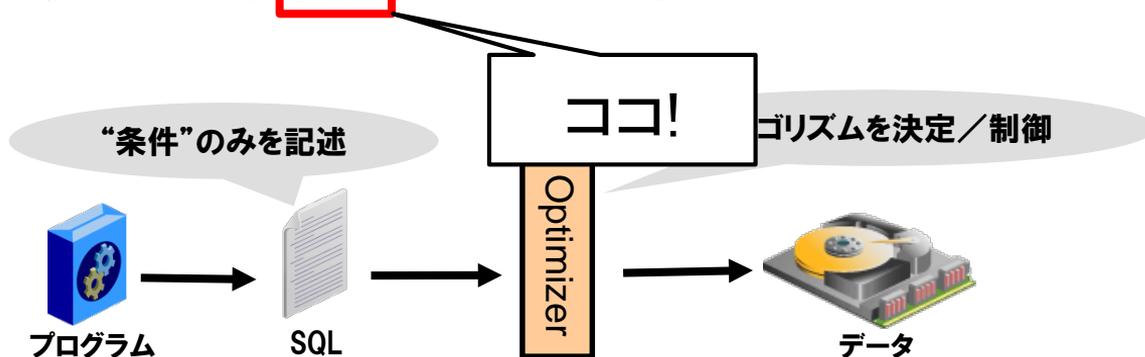
振り返ってこのスライド...実は大切な事を書いています。

- どうやって(How)データを抽出するか、即ちアルゴリズムの決定・制御は、RDBMS のオプティマイザで行われます。
 - オプティマイザが決定したアルゴリズム=実行計画です。
 - オプティマイザはSQLテキストや統計情報などを基に、適切な実行計画を予測して組み立てます。



本セミナーのキーワードの一つ:「予測」

- どうやって(How)データを抽出するか、即ちアルゴリズムの決定・制御は、RDBMS のオプティマイザで行われます。
 - オプティマイザが決定したアルゴリズム=実行計画です。
 - オプティマイザはSQLテキストや統計情報などを基に、適切な実行計画を**予測**して組み立てます。



SQLのアルゴリズムは「予測」で組み立てられる。

- SQLのアルゴリズム≡実行計画は、オプティマイザが最適と考えられるものを「予測」して組み立てます。
- そして「予測」である以上、必ず**ハズレ**のケースが出てきます。
 - ハズレの実行計画を引くと、性能問題として顕在化！
 - SQLの特徴に由来する、全てのRDBMSに共通した本質的な困難
 - 各ベンダやオープンソースのRDBMSは、このSQLの本質的な困難に立ち向かうべく、日々凌ぎを削っています。(もちろんOracleも！)
- まずこのハズレの実行計画の存在を意識／認識しておくのが、本セミナーの出発点となります。

2章. 「全体最適」と「個別最適」 の必要性

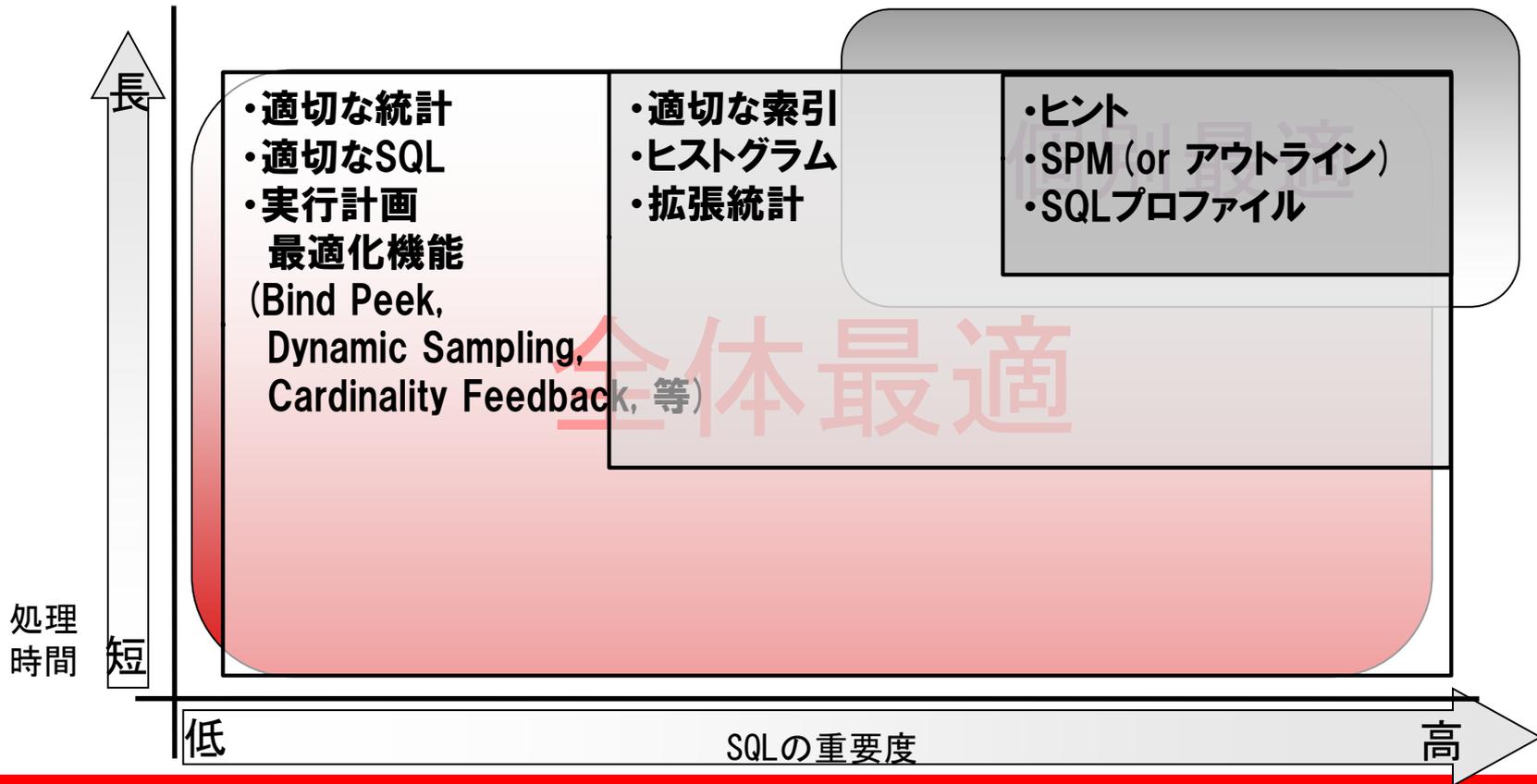
「全体最適」と「個別最適」の使い分け

- SQLチューニングの手法には、全体最適にマッチする手法と、個別最適に適した手法の2つに大別されます。
- 全体最適にマッチする手法としては、以下のようなものが挙げられます。
 - 適切なSQL(SQLコーディング・ガイド、コード・インスペクション、等)
 - 適切な統計(常時最新化されたフレッシュな統計、最大件数想定 of 固定値、等)
 - 適切な索引(一意キー索引、追加索引、等)
 - 実行計画最適化機能(BindPeek, DyanmicSampling, CardinalityFeedback、等)
- 個別最適にマッチする手法としては、以下のようなものが挙げられます。
 - ヒント
 - SPM(11gR1以降)、アウトライン(10gR2以前)
 - SQLプロファイル

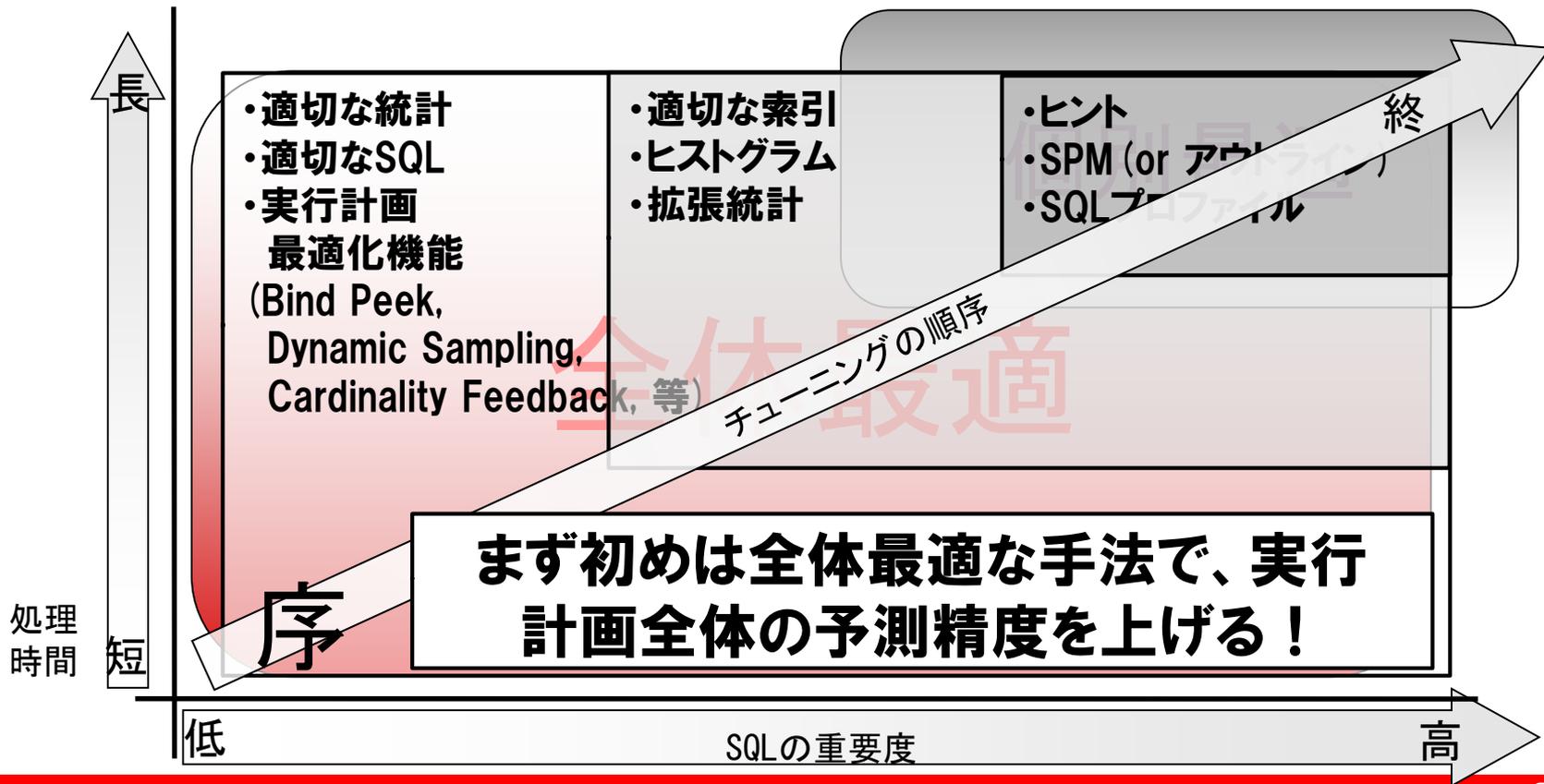
「全体最適」と「個別最適」の適用イメージ(1)



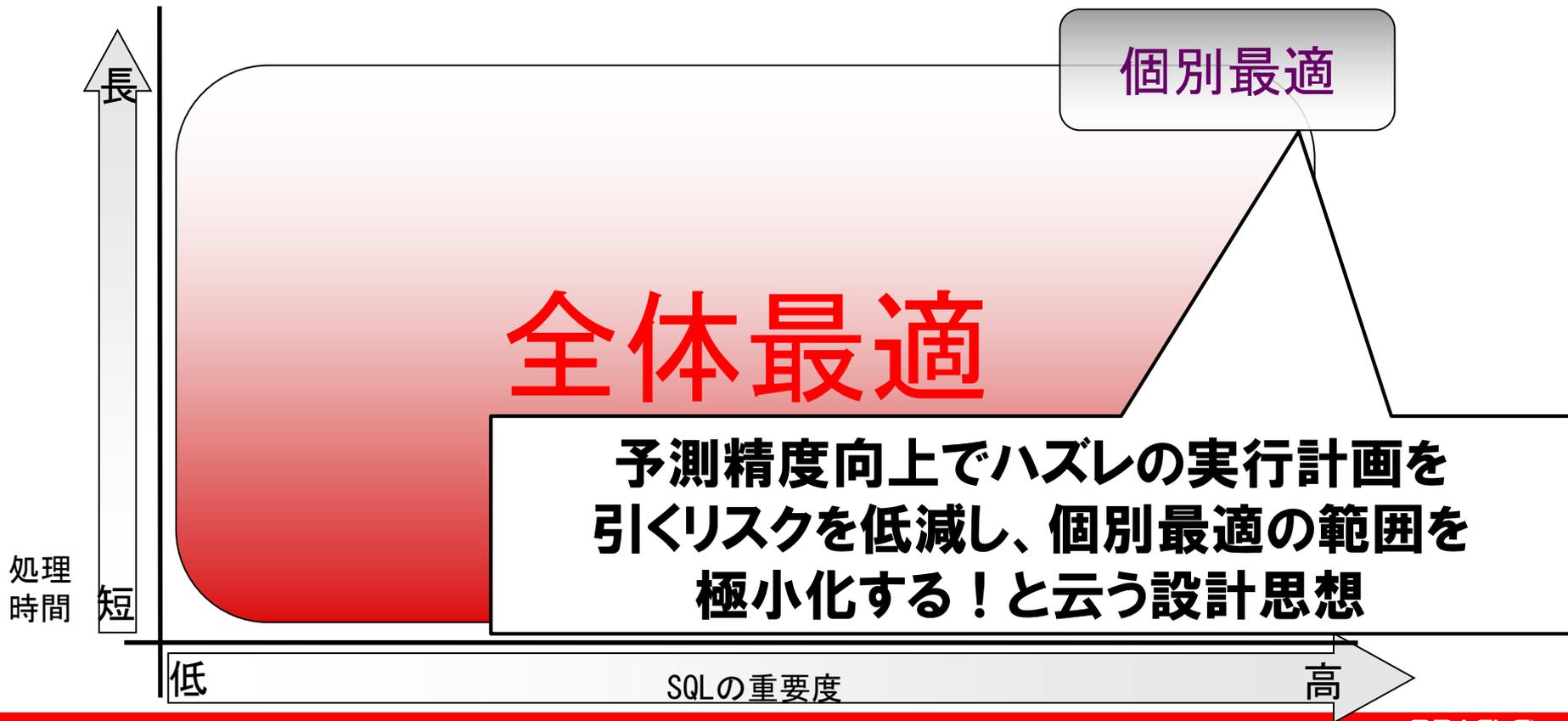
「全体最適」と「個別最適」の適用イメージ(2)



「全体最適」と「個別最適」の適用イメージ(3)



「全体最適」と「個別最適」の適用イメージ(4)



「全体最適」のうち「個別最適」、、、の考え方

- まずは「全体最適」の設計思想で、実行計画全体の予測精度を向上させることが必要だと考えます。
- でも「個別最適」の必要性は無くなりません。何故か？
- それはこれまで述べてきた**ハズレ**の実行計画の存在です。
 - SQLの仕組み上、ハズレの実行計画は不可避であると云う認識が必要
 - ハズレの実行計画は有りうべきものとして、「個別最適」でチューニングする。
- 本セミナーでは、この「個別最適」のチューニングで有効なテクニックを紹介していきたいと思います。
 - 「全体最適」の話については、これはこれで長くなるので、またいずれ、、、、

3章. 「予測」と「実測」の乖離を 補正せよ！ OracleDBの機能を活用した 最新テクニック

SQLチューニングの流れ(1)

遅いSQLの特定(sql_id特定)

実行計画のボトルネック特定

Tuning案①

Tuning案②

Tuning案③

Tuning案④



目標達成!!

SQLチューニングの流れ(2)

遅いSQLの特定(sql_id特定)

実行計画のボトルネック特定

•OracleDB の機能を利用した、ボトルネック特定方法を紹介

Tuning案①

Tuning案②

Tuning案③

Tuning案④



目標達成!!

SQLチューニングの流れ(3)

遅いSQLの特定(sql_id特定)

実行計画のボトルネック特定

・複数のチューニング案を
疑似ワークショップ形式
で紹介

Tuning案①

Tuning案②

Tuning案③

Tuning案④



目標達成!!



本セミナーで解説する範囲

- OracleDatabaseの機能を利用した、効率の良い「実行計画のボトルネック特定」方法を紹介します。
 - DBMS_XPLAN.DISPLAY_CURSOR で実行統計を出力する方法
(参考Document:KROWN#141531)
 - DBMS_SQLTUNE.REPORT_SQL_MONITOR によるリアルタイム監視SQLのレポートニング
- 加えて、あるSQLをチューニングすると云う疑似ワークショップで、複数のチューニング案を紹介していきます。
 - 複数のチューニング案を覚えて、引き出しを増やす。
 - チューニングの引き出しを増やすことが、上級者への近道！

DBMS_XPLAN と DBMS_SQLTUNE による 実行計画のボトルネック特定

DBMS_XPLAN.DISPLAY_CURSOR の概要

- 実行計画を出力するためのPL/SQLパッケージ(標準機能)
- ボトルネック特定を行いたいSQLの sql_id を調べます。
- 対象SQLの完了を待つか、Ctrl+C等 で強制終了させます。
- 対象SQLの完了後、下記SQLを実行して
対象SQL の実行計画／実行統計を出力します。
 - formatパラメータに ALLSTATSオプションを付与します。

```
SELECT * FROM TABLE (  
  DBMS_XPLAN.DISPLAY_CURSOR (  
    '対象SQL の sql_id', NULL, 'ALL ALLSTATS LAST'));
```

DBMS_XPLAN.DISPLAY_CURSOR の制限事項

- 10gR1以降の機能となります。
- ALLSTATS書式を有効化するには下記の「どちらか」を満たして、実行統計が採取される状態で SQL を実行する必要があります。
 - 初期化パラメータ「STATISTICS_LEVEL = ALL」を設定
※セッション単位(ALTER SESSION～)でも設定可能
 - SQL に /*+ gather_plan_statistics */ ヒントを付与
- 対象SQLが終了すると、実行統計が共有プールのカーソルに反映されます。
 - SQLが完全に終了するか、Ctrl+C等で強制終了させる必要があります。
 - 強制終了させた場合は、強制終了時点までの実行統計が反映されます。
- SQL終了前に実行計画を出力しても、実行統計は出てきません。

DBMS_XPLAN.DISPLAY_CURSORの実行例

- 実行例を下記に示します。(※一部の出力行／出力項目を省略)

```
SQL> SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY_CURSOR('80w7gaz4dywud', NULL, 'ALL ALLSTATS LAST'));
```

```
PLAN_TABLE_OUTPUT
```

```
SQL_ID 80w7gaz4dywud, child number 0
```

```
SELECT /*+ gather_plan_statistics */ C.DATBI, C.HI_PR, C.LOW_PR,
```

```
:
```

```
Plan hash value: 3232858554
```

Id	Operation	Name	E-Rows	E-Bytes	E-Time	A-Rows	A-Time
0	SELECT STATEMENT					5924	00:00:53.45
1	SORT ORDER BY		414	28566	00:02:43	5924	00:00:53.45
2	VIEW		414	28566	00:02:43	5924	00:00:53.43
3	SORT UNIQUE		414	47610	00:02:43	5924	00:00:53.42
4	WINDOW SORT		414	47610	00:02:43	53460	00:00:53.33
* 5	HASH JOIN		2722	305K	00:02:43	53460	00:00:52.69
6	TABLE ACCESS BY INDEX ROWID	DBN_FTBP900	3640	145K	00:02:43	54338	00:00:52.49
* 7	INDEX SKIP SCAN	DBN_FTBP900PK	2		00:02:43	54338	00:00:51.98
* 8	TABLE ACCESS FULL	DBN_FTBA045	11477	829K	00:00:01	22310	00:00:00.01

```
:
```

DBMS_XPLAN.DISPLAY_CURSORによる解析(1)

■ 注目ポイント1

- SQLの実行統計(「実測」の処理件数(A-Rows)/処理時間(A-Time))に注目

```
SQL> SELECT * FROM TABLE (DBMS_XPLAN.DISPLAY_CURSOR('80w7gaz4dywud', NULL, 'ALL ALLSTATS LAST'))
:
```

Plan hash value: 3232858554

Id	Operation	Name	...	E-Rows	E-Bytes	...	E-Time	A-Rows	A-Time	...
0	SELECT STATEMENT			5924	00:00:53.45	...
1	SORT ORDER BY		...	414	28566	...	00:02:43	5924	00:00:53.45	...
2	VIEW		...	414	28566	...	00:02:43	5924	00:00:53.43	...
3	SORT UNIQUE		...	414	47610	...	00:02:43	5924	00:00:53.42	...
4	WINDOW SORT		...	414	47610	...	00:02:43	53460	00:00:53.33	...
* 5	HASH JOIN		...	2722	305K	...	00:02:43	53460	00:00:52.69	...
6	TABLE ACCESS BY INDEX ROWID	DBN_FTBP900	...	3640	145K	...	00:02:43	54338	00:00:52.49	...
* 7	INDEX SKIP SCAN	DBN_FTBP900PK	...	2		...	00:02:43	54338	00:00:51.98	...
* 8	TABLE ACCESS FULL	DBN_FTBAT045	...	11477	829K	...	00:00:01	22310	00:00:01.01	...

:

実行統計

ここが遅そう...

DBMS_XPLAN.DISPLAY_CURSORによる解析(2)

■ 注目ポイント2

- 実行統計(Actual)と CBO予測(Estimate)の乖離に注目

```
SQL> SELECT * FROM TABLE (DBMS_XPLAN.DISPLAY_CURSOR('80w',  
:  
Plan hash value: 3232858554
```

Id	Operation	Name	E-Rows	E-Bytes	E-Time	A-Rows	A-Time
0	SELECT STATEMENT					5924	00:00:53.45
1	SORT ORDER BY		414	28566	00:02:43	5924	00:00:53.45
2	VIEW		414	28566	00:02:43	5924	00:00:53.43
3	SORT UNIQUE		414	47610	00:02:43	5924	00:00:53.42
4	WINDOW SORT		414	47610	00:02:43	53460	00:00:53.33
* 5	HASH JOIN		2722	305K	00:02:43	53460	00:00:52.69
6	TABLE ACCESS BY INDEX ROWID	DBN_FTBP900	3640	145K	00:02:43	54338	00:00:52.49
* 7	INDEX SKIP SCAN	DBN_FTBP900PK	2		00:02:43	54338	00:00:51.98
* 8	TABLE ACCESS FULL	DBN_FTBAT045	11477	829K	00:00:01	22310	00:00:00.01

CBO予測 (E-Rows) と **実行統計** (A-Rows) の乖離に注目。

CBOは2件アクセスと予測しているが、実際は54338件にアクセス

DBMS_XPLAN.DISPLAY_CURSORによる解析(3)

- 前2ページの解析より、実行計画の Id 7 がボトルネックになっていると判断される。

```
SQL> SELECT * FROM TABLE (DBMS_XPLAN.DISPLAY_CURSOR('80w7gaz4dywud', NULL, 'ALL ALLSTATS LAST'));
```

```
:
```

```
Plan hash value: 3232858554
```

Id	Operation	Name	E-Rows	E-Bytes	E-Time	A-Rows	A-Time
0	SELECT STATEMENT					5924	00:00:53.45
1	SORT ORDER BY		414	28566	00:02:43	5924	00:00:53.45
2	VIEW		414	28566	00:02:43	5924	00:00:53.43
3	SORT UNIQUE		414	47610	00:02:43	5924	00:00:53.42
4	WINDOW SORT		414	47610	00:02:43	53460	00:00:53.33
* 5	HASH JOIN		2722	305K	00:02:43	53460	00:00:52.69
6	TABLE ACCESS BY INDEX ROWID	DBN_FTBPR900	3640	145K	00:02:43	54338	00:00:52.49
* 7	INDEX SKIP SCAN	DBN_FTBP900PK	2		00:02:43	54338	00:00:51.98
* 8	TABLE ACCESS FULL	DBN_FTBA1045	11477	829K	00:00:01	22310	00:00:00.01

- 処理時間の実行統計 (A-time) が多い。
- 処理件数のCBO予測 (E-Rows) と実行統計 (A-Rows) が乖離している。

DBMS_SQLTUNE.REPORT_SQL_MONITORの概要

- SQLチューニングのための機能を提供するPL/SQLパッケージ(オプション)
- ボトルネック特定を行いたいSQLの sql_id を調べます。
- 下記SQLを実行して 対象SQL の リアルタイムSQL監視レポートを出力します。(※対象SQLが実行中でも構いません。)

```
SET LONG 1000000
SET LONGC 1000000
VAR c_rep CLOB;
EXEC :c_rep := DBMS_SQLTUNE.REPORT_SQL_MONITOR (sql_id => '対象SQLのsql_id');
PRINT c_rep;
```

DBMS_SQLTUNE.REPORT_SQL_MONITORの制限事項

- 11gR1以降の機能となります。
- DBMS_SQLTUNEパッケージを使用するため、Enterprise Edition の Tuning Packオプションライセンスが必要となります。
- リアルタイムSQL監視の対象となる(※V\$SQL_MONITORビューに載る)必要があります。下記の「どれか」が満たされる必要があります。
 - SQL の処理時間が5秒以上
 - SQL に MONITORヒントを付与
 - パラレル・クエリ

DBMS_SQLTUNE.REPORT_SQL_MONITORの実行例

- 実行例を下記に示します。(※一部の出力行／出力項目を省略)

```
SQL> EXEC :C_REP := DBMS_SQLTUNE.REPORT_SQL_MONITOR(SQL_ID => '9ksyk16au4zzf');
SQL> PRINT C_REP;
SQL Text
```

```
-----
SELECT B762.R_DIST_CODE || B762.R_CUSTOMER_CODE ...
```

Global Stats

```
=====
| Elapsed | Cpu | IO | Other | Buffer | Read | Read |
| Time(s) | Time(s) | Waits(s) | Waits(s) | Gets | Reqs | Bytes |
=====
| 1268 | 1267 | 0.03 | 0.29 | 15317 | 1 | 1MB |
=====
```

SQL Plan Monitoring Details (Plan Hash Value=546406420)

```
=====
| Id | Operation | Name | Rows | Excs | Rows | Activity | Activity Detail |
| | | | (Estim) | | (Actual) | (%) | (# samples) |
=====
| 0 | SELECT STATEMENT | | | 1 | | | |
| 1 | SORT UNIQUE | | 6011 | 1 | 0 | | |
| 2 | UNION-ALL | | | 1 | 1105 | | |
| : | : | : | : | : | : | : | :
| 7 | VIEW | VM_NWVW_1 | 7 | 1 | | | |
-> 8 | HASH GROUP BY | | 7 | 1 | 0 | 92.11 | Cpu (1168)
-> 9 | HASH JOIN | | 2M | 1 | 980M | 7.57 | Cpu (96)
| 10 | INDEX RANGE SCAN | KUAB11101 | 6021 | 1 | 2635 | | |
| : | : | : | : | : | : | : | :
| 20 | TABLE ACCESS FULL | CUAB111 | 2635 | 1 | 2635 | | |
=====
```

DBMS_SQLTUNE.REPORT_SQL_MONITORの解析(1)

- 注目ポイント1…実行統計(SQLの「実際」の処理時間／処理件数)に注目

```
SQL> EXEC :C_REP := DBMS_SQLTUNE.REPORT_SQL_MONITOR(SQL_ID => '9ksyk16au4zzf');
SQL> PRINT C_REP;
SQL Text
```

```
SELECT B762. R_DIST_CODE || B762. R_CUSTOMER_CODE ...
```

Global Stats

Elapsed Time(s)	Cpu Time(s)	IO Waits(s)	Other Waits(s)	Buffer Gets	Read Reqs	Read Bytes
1268	1267	0.03	0.29	15317	1	1MB

実行統計 (トータル)

SQL Plan Monitoring Details (Plan Hash Value=546406420)

Id	Operation	Name	Rows (Estim)	Execs	Rows (Actual)	Activity (%)	Activity Detail (# samples)
0	SELECT STATEMENT			1			
1	SORT UNIQUE		6011	1	0		
2	UNION-ALL			1	1105		
:	:	:	:	:	:	:	:
7	VIEW	VM_NVVW_1	7	1			
-> 8	HASH GROUP BY		7	1	0	92.11	Cpu (1168)
-> 9	HASH JOIN		2M	1	980M	7.57	Cpu (96)
10	INDEX RANGE SCAN	KUAB11101	6021	1	2635		
:	:	:	:	:	:	:	:
20	TABLE ACCESS FULL	CUAB111	2635	1	2635		

実行統計

ここが遅そう…

DBMS_SQLTUNE.REPORT_SQL_MONITORの解析(2)

- 注目ポイント2…実行統計(Actual)と CBO予測(Estimate)の乖離に注目

```
SQL> EXEC :C_REP := DBMS_SQLTUNE.REPORT_SQL_MONITOR(SQL_ID => '9ksyk16au4zzf');
SQL> PRINT C_REP;
SQL Text
```

```
SELECT B762. R_DIST_CODE || B762. R_CUSTOMER_CODE ...
```

Global Stats

Elapsed Time(s)	Cpu Time(s)	IO Waits(s)	Other Waits(s)	Buffer Gets	Read Reqs	Read Bytes
1268	1267	0.03	0			

実行統計 (トータル)

CBO予測

実行統計 (待機イベント付き)

SQL Plan Monitoring Details (Plan Hash Value=546406420)

Id	Operation	Name	Rows (Estim)	Execs	Rows (Actual)	Activity (%)	Activity Detail (# samples)
0	SELECT STATEMENT			1			
1	SORT UNIQUE		6011	1	0		
2	UNION-ALL			1	1105		
:	:		:	:	:	:	:
7	VIEW	VM_NWVW_1	7	1			
-> 8	HASH GROUP BY		7	1	0	92.11	Cpu (1168)
-> 9	HASH JOIN		2M	1	980M	7.57	Cpu (96)
10	INDEX RANGE SCAN	KUAB11101		1	2635		
:	:		:	:	:	:	:
20	TAB						

CBOは2M件アクセスと予測しているが、実際は980M件にアクセス

DBMS_SQLTUNE.REPORT_SQL_MONITORの解析(3)

- 前2ページの解析より、実行計画の Id 8,9 がボトルネックと判断

```
SQL> EXEC :C_REP := DBMS_SQLTUNE.REPORT_SQL_MONITOR(SQL_ID => '9ksyk16au4zzf');
SQL> PRINT C_REP;
SQL Text
```

```
-----
SELECT B762.R_DIST_CODE || B762.R_CUSTOMER_CODE ...
```

Global Stats

Elapsed Time(s)	Cpu Time(s)	IO Waits(s)	Other Waits(s)	Buffer Gets	Read Reqs	Read Bytes
1268	1267	0.03	0.29	15317	1	1MB

SQL Plan Monitoring Details (Plan Hash Value=546406420)

Id	Operation	Name	Rows (Estim)	Execs	Rows (Actual)	Activity (%)	Activity Detail (# samples)
0	SELECT STATEMENT						
1							
2							
3							
4							
5							
6							
7	VIEW	VM_NWVW	7	1	0	92.11	Cpu (1168)
8	HASH GROUP BY		7	1	0	92.11	Cpu (1168)
9	HASH JOIN		2M	1	980M	7.57	Cpu (96)
10	INDEX RANGE SCAN	KUAB11101	6021	1	2635		
11							
12							
13							
14							
15							
16							
17							
18							
19							
20	TABLE ACCESS FULL	CUAB111	2635	1	2635		

- 処理時間の実行統計 (Activity Detail) が多い。
- 処理件数のCBO予測 (Rows Estim) と実行統計 (Rows Actual) が乖離している。

(参考)EM による REPORT_SQL_MONITOR

- テキスト形式と同等の情報を、GUIのオペレーションで参照可能です。



両者の比較

	DBMS_XPLAN.DISPLAY_CURSOR (ALLSTATS書式)	DBMS_SQLTUNE. REPORT_SQL_MONITOR
バージョン	10gR1以降	11gR1以降
有償オプション	不要	要Tuning Pack
事前仕込	どちらかを事前に仕込む必要アリ <ul style="list-style-type: none">▪ STATISTICS_LEVEL=ALL▪ gather_plan_statistics ヒント付与	SQLが5秒以上、又はパラレル・クエリなら不要 <ul style="list-style-type: none">▪ 非パラレル かつ 5秒未満の場合はMONITORヒント付与
採取可能な情報	実行統計(行数/処理時間)	実行統計(行数/処理時間/ トータル統計/待機イベント)
SQL終了要否	対象SQLが終了しているか、Ctrl+C 等で強制終了させる必要がある。	対象SQLが実行中でも 情報採取可能

- 
- ・有償オプションが必要な DBMS_SQLTUNE の方が総じて優秀
 - ・但し DBMS_XPLAN でしか採取できない情報もあるため、両方採取することを推奨

疑似ワークショップによる SQLチューニング案の紹介

疑似ワークショップの目的

- ワークショップの目的は SQLチューニングのネタ／引き出し(選択肢)を増やすことにあります。
 - 上級者への入り口です。
- チューニングのネタ／引き出しが増えれば、目標到達への可能性が広がります。
- 複数あるチューニング案の一部を紹介していきます。
 - SQLチューニングを疑似体験して、引き出しを増やしましょう！

疑似ワークショップの課題SQL

- チューニング対象のSQL は以下の通りです。
 - TEST_TABLE_A表 と TBL_B表 を結合する SQL

```
-- g9gnrhjwajfnn
SELECT /*+ MONITOR */
      A.*
FROM TEST_TABLE_A A
      , TBL_B B
WHERE A.P_NO2 = B.P_NO
      AND A.P_CHAR = B.P_CHAR
      AND TO_CHAR(B.P_DATE, 'YYYYMMDD') = '20120801';
```

SQL実行時間とAUTOTRACE統計

- チューニング前のSQL実行時間／AUTOTRACE統計
 - 下記の実行時間(Elapsed)と負荷(gets/reads)を減らすのが、SQLチューニングのセオリーとなります。

```
10:39:35 SQL> SELECT /*+ MONITOR */
10:39:35 2      A.*
10:39:35 3      FROM TEST_TABLE_A A
10:39:35 4           , TBL_B B
10:39:35 5      WHERE A.P_NO2      = B.P_NO
10:39:35 6           AND A.P_CHAR = B.P_CHAR
10:39:35 7           AND TO_CHAR(B.P_DATE, 'YYYYMMDD') = '20120801';
```

1102 rows selected.

Elapsed: 00:00:12.44

:

Statistics

8923 consistent gets
5985 physical reads

疑似ワークショップの関連情報(TEST_TABLE_A表)

■ テーブル定義

```
SQL> DESC TEST_TABLE_A;
名前          NULL?     型
-----
P_NO          NUMBER
P_NO2        NUMBER    ※結合条件列
P_CHAR       VARCHAR2(1) ※結合条件列
P_DATE       DATE
```

■ 索引

- P_NO2列 と P_CHAR列
(※結合条件列)の複合索引
- 一意索引無し

```
CREATE INDEX TEST_TABLE_A_I1
ON TEST_TABLE_A(P_NO2, P_CHAR);
```

■ レコード件数

- レコード件数は約260万件

```
SQL> SELECT COUNT(*) FROM TEST_TABLE_A;
COUNT(*)
-----
2600026
```

■ 各列のDISTINCT値

```
SQL> SELECT COUNT(DISTINCT P_NO) AS P_NO
2      , COUNT(DISTINCT P_NO2) AS P_NO2
3      , COUNT(DISTINCT P_CHAR) AS P_CHAR
4      , COUNT(DISTINCT P_DATE) AS P_DATE
5      FROM TEST_TABLE_A;
```

	P_NO	P_NO2	P_CHAR	P_DATE
	2600000	1000	26	546

疑似ワークショップの関連情報(TBL_B表)

■ テーブル定義

```
SQL> DESC TBL_B;
名前          NULL?      型
-----
P_NO          NUMBER     ※結合条件列
P_CHAR        VARCHAR2(1) ※結合条件列
P_DATE        DATE       ※抽出条件
```

■ 索引

- 索引無し

■ レコード件数

- レコード件数は約3万件

```
SQL> SELECT COUNT(*) FROM TBL_B;
COUNT(*)
-----
30012
```

■ 各列のDISTINCT値

```
SQL> SELECT COUNT(DISTINCT P_NO) AS P_NO
2      , COUNT(DISTINCT P_CHAR) AS P_CHAR
3      , COUNT(DISTINCT P_DATE) AS P_DATE
4      FROM TBL_B;

P_NO    P_CHAR    P_DATE
-----
10      10        32
```

DBMS_XPLAN.DISPLAY_CURSOR結果

```
SQL> SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY_CURSOR('g9gnrhjwajfnn', NULL, 'ADVANCED ALLSTATS LAST'));
```

```
PLAN_TABLE_OUTPUT
```

```
-----
SQL_ID g9gnrhjwajfnn, child number 0
-----
```

```
SELECT /*+ MONITOR */      A.* FROM TEST_TABLE_A A      , TBL_B B
WHERE A.P_NO2 = B.P_NO      AND A.P_CHAR = B.P_CHAR      AND
TO_CHAR(B.P_DATE, 'YYYYMMDD') = '20120801'
```

```
Plan hash value: 960095112
```

Id	Operation	Name	E-Rows	E-Bytes	E-Time	A-Rows	A-Time
0	SELECT STATEMENT					1102	00:00:10.41
* 1	HASH JOIN		81	2349	00:00:01	1102	00:00:10.41
2	TABLE ACCESS FULL	TEST_TABLE_A	26	416	00:00:01	2600K	00:00:01.49
* 3	TABLE ACCESS FULL	TBL_B	300	3900	00:00:01	11	00:00:00.09

DBMS_SQLTUNE.REPORT_SQL_MONITOR結果

SQL Text

```
SELECT /*+ MONITOR */ A.* FROM TEST_TABLE_A A , TBL_B B WHERE A.P_NO2 = B.P_NO AND
A.P_CHAR = B.P_CHAR AND TO_CHAR(B.P_DATE, 'YYYYMMDD') = '20120801 '
```

Global Stats

Elapsed Time(s)	Cpu Time(s)	IO Waits(s)	Fetch Calls	Buffer Gets	Read Reqs	Read Bytes	Write Reqs	Write Bytes
11	8.06	2.90	75	8936	108	26MB	305	74MB

SQL Plan Monitoring Details (Plan Hash Value=960095112)

Id	Operation	Name	Rows (Estim)	Rows (Actual)	Activity (%)	Activity Detail (# samples)
0	SELECT STATEMENT			1102		
1	HASH JOIN		81	1102	90.91	Cpu (9) direct path write temp (1)
2	TABLE ACCESS FULL	TEST_TABLE_A	26	3M	9.09	Cpu (1)
3	TABLE ACCESS FULL	TBL_B	300	11		

チューニング案の一覧

- 今回のSQLでは下記に挙げるチューニングで性能が改善しました。

- 案1 … DBMS_STATSによる統計情報採取
- 案2 … 拡張統計の採取
- 案3 … SQLプロファイル適用(by DBMS_SQLTUNE)
- 案4 … CardinalityFeedbackの使用
- 案5 … ヒント や SPM による実行計画操作
- 案6 … パラレル・クエリ化
- 案7 … SQL修正(WHERE句書き換え)
- 案8 … SQL修正(WITH句によるサブクエリ切り出し)
- 案9 … Dynamic Sampling適用
- 案10…新規索引付与
- 案11…リザルト・セットの適用

**今回紹介する
チューニング案**

案1. DBMS_STATSによる 統計情報採取

案1. DBMS_STATSによる統計情報採取(1)

- まず確認するのは、統計情報が実態と合っているか？

```
SQL> SELECT TABLE_NAME, NUM_ROWS FROM USER_TABLES WHERE TABLE_NAME IN ('TEST_TABLE_A', 'TBL_B');
```

TABLE_NAME	NUM_ROWS
TEST_TABLE_A	26
TBL_B	30012

•統計は 26件

```
SQL> SELECT COUNT(*) FROM TEST_TABLE_A;
```

COUNT(*)
2600026

•実態は 約2,600,000件

```
SQL> SELECT COUNT(*) FROM TBL_B;
```

COUNT(*)
30012

案1. DBMS_STATSによる統計情報採取(2)

- DBMS_STATSパッケージで統計情報を採取する。
 - 採取した統計を即座に使用(NO_INVALIDATE => FALSE)
 - 4パラで採取(DEGREE => 4)

```
EXEC DBMS_STATS.GATHER_TABLE_STATS(  
    USER  
    , 'TEST_TABLE_A'  
    , NO_INVALIDATE => FALSE  
    , DEGREE => 4);
```

案1. DBMS_STATSによる統計情報採取(3)

■ 統計情報採取後の実行統計

```
10:39:35 SQL> SELECT /** MONITOR */  
10:39:35 2      A.*  
10:39:35 3      FROM TEST_TABLE_A A  
10:39:35 4           , TBL_B B  
10:39:35 5      WHERE A.P_NO2    = B.P_NO  
10:39:35 6           AND A.P_CHAR = B.P_CHAR  
10:39:35 7           AND TO_CHAR(B.P_DATE, 'YYYYMMDD') = '20120801';
```

1102 rows selected.

Elapsed: 00:00:12.44

Statistics

8923 consistent gets
5985 physical reads

```
10:48:08 SQL> SELECT /** MONITOR */  
10:48:08 2      A.*  
10:48:08 3      FROM TEST_TABLE_A A  
10:48:08 4           , TBL_B B  
10:48:08 5      WHERE A.P_NO2    = B.P_NO  
10:48:08 6           AND A.P_CHAR = B.P_CHAR  
10:48:08 7           AND TO_CHAR(B.P_DATE, 'YYYYMMDD') = '20120801';
```

1102 rows selected.

Elapsed: 00:00:04.71

Statistics

8994 consistent gets
59 physical reads

性能改善!

DBMS_XPLAN.DISPLAY_CURSOR結果の比較

■ Before

Id	Operation	Name	E-Rows	E-Bytes	E-Time	A-Rows	A-Time
0	SELECT STATEMENT					1102	00:00:12.32
* 1	HASH JOIN		81	2349	00:00:01	1102	00:00:12.32
2	TABLE ACCESS FULL	TEST_TABLE_A	26	416	00:00:01	2600K	00:00:01.46
* 3	TABLE ACCESS FULL	TBL_B	300	3900	00:00:01	11	00:00:00.09

■ After

Id	Operation	Name	E-Rows	E-Bytes	E-Time	A-Rows	A-Time
0	SELECT STATEMENT					1102	00:00:04.65
* 1	HASH JOIN		30012	967K	00:00:32	1102	00:00:04.65
* 2	TABLE ACCESS FULL	TBL_B	300	3900	00:00:01	11	00:00:00.08
3	TABLE ACCESS FULL	TEST_TABLE_A	2600K	49M	00:00:31	2600K	00:00:01.48

DBMS_SQLTUNE.REPORT_SQL_MONITOR結果の比較

Before

Id	Operation	Name	Rows (Estim)	Rows (Actual)	Activity Detail (# samples)
0	SELECT STATEMENT			1102	
1	HASH JOIN		81	1102	Cpu (8) direct path write temp (4)
2	TABLE ACCESS FULL	TEST_TABLE_A	26	3M	
3	TABLE ACCESS FULL	TBL_B	300	11	

After

Id	Operation	Name	Rows (Estim)	Rows (Actual)	Activity Detail (# samples)
0	SELECT STATEMENT			1102	
1	HASH JOIN		30012	1102	Cpu (5)
2	TABLE ACCESS FULL	TBL_B	300	11	
3	TABLE ACCESS FULL	TEST_TABLE_A	3M	3M	

案1. DBMS_STATSによる統計情報採取(6)

- Oracle の CBO は統計情報を元に、最適な実行計画を予測して組み立てます。
- テーブルの統計情報と実態(実件数)が合致しているかどうかは、真っ先に確認すべきです！
 - 0件統計(←典型的なアンチパターン！)
 - 0件では無いが、実態と乖離した統計
- 本チューニング案では統計情報最新化で HASH JOIN の結合順序が入れ替わり、一時表領域へのI/Oが無くなって性能が改善しています。

案2. 拡張統計の採取

案2. 拡張統計の採取(1)

- 表の列(カラム)に関数を適用しているのは、SQLのアンチパターンの一つです。

```
SELECT /*+ MONITOR */  
  A.*  
FROM TEST_TABLE_A A  
  , TBL_B B  
WHERE A.P_NO2 = B.P_NO  
  AND A.P_CHAR = B.P_CHAR  
  AND TO_CHAR(B.P_DATE, 'YYYYMMDD') = '20120801';
```

P_DATE列にTO_CHAR関数を適用

- 本パターンが性能劣化し易いのは、下記2つの理由に依ります。
 - (1). 列に作成された索引が使用されない。
 - (2). CBO が列統計を使用できず、実行計画の予測精度が落ちる。
- (1)の理由は有名ですが、今回は (2) に着目します。

案2. 拡張統計の採取(2)

- DBMS_STATSパッケージで拡張統計情報を採取する。
 - METHOD_OPT に 拡張統計(※今回のSQLでは式統計)の明示的採取を設定する。
 - ESTIMATE_PERCENT に 100 を設定して、全サンプリング。

```
DBMS_STATS.GATHER_TABLE_STATS(  
  USER, 'TBL_B',  
  NO_INVALIDATE => FALSE,  
  METHOD_OPT => 'FOR ALL COLUMNS SIZE AUTO ' ||  
               'FOR COLUMNS (TO_CHAR(P_DATE, 'YYYYMMDD')) SIZE 254',  
  DEGREE => 4,  
  ESTIMATE_PERCENT => 100);
```

案2. 拡張統計の採取(3)

■ 拡張統計(式統計)採取後の実行統計

```
10:48:08 SQL> SELECT /** MONITOR */
10:48:08 2      A.*
10:48:08 3      FROM TEST_TABLE_A A
10:48:08 4      , TBL_B B
10:48:08 5      WHERE A.P_NO2 = B.P_NO
10:48:08 6      AND A.P_CHAR = B.P_CHAR
10:48:08 7      AND TO_CHAR(B.P_DATE, 'YYYYMMDD') = '20120801';
```

1102 rows selected.

Elapsed: 00:00:04.71

Statistics

8994 consistent gets
59 physical reads

```
13:47:45 SQL> SELECT /** MONITOR */
13:47:45 2      A.*
13:47:45 3      FROM TEST_TABLE_A A
13:47:45 4      , TBL_B B
13:47:45 5      WHERE A.P_NO2 = B.P_NO
13:47:45 6      AND A.P_CHAR = B.P_CHAR
13:47:45 7      AND TO_CHAR(B.P_DATE, 'YYYYMMDD') = '20120801';
```

1102 rows selected.

Elapsed: 00:00:00.16

Statistics

1301 consistent gets
0 physical reads

性能改善!

DBMS_XPLAN.DISPLAY_CURSOR結果の比較

■ Before

Id	Operation	Name	E-Rows	E-Bytes	E-Time	A-Rows	A-Time
0	SELECT STATEMENT					1102	00:00:04.65
* 1	HASH JOIN		30012	967K	00:00:32	1102	00:00:04.65
* 2	TABLE ACCESS FULL	TBL_B	300	3900	00:00:01	11	00:00:00.08
3	TABLE ACCESS FULL	TEST_TABLE_A	2600K	49M	00:00:31	2600K	00:00:01.48

■ After

Id	Operation	Name	E-Rows	E-Bytes	E-Time	A-Rows	A-Time
0	SELECT STATEMENT					1102	00:00:00.11
1	NESTED LOOPS					1102	00:00:00.11
2	NESTED LOOPS		1106	37604	00:00:14	1102	00:00:00.09
* 3	TABLE ACCESS FULL	TBL_B	11	154	00:00:01	11	00:00:00.09
* 4	INDEX RANGE SCAN	TEST_TABLE_A_I1	100		00:00:01	1102	00:00:00.01
5	TABLE ACCESS BY INDEX ROWID	TEST_TABLE_A	101	2020	00:00:02	1102	00:00:00.01

DBMS_SQLTUNE.REPORT_SQL_MONITOR結果の比較

■ Before

Id	Operation	Name	Rows (Estim)	Rows (Actual)	Activity Detail (# samples)
0	SELECT STATEMENT			1102	
1	HASH JOIN		30012	1102	Cpu (5)
2	TABLE ACCESS FULL	TBL_B	300	11	
3	TABLE ACCESS FULL	TEST_TABLE_A	3M	3M	

■ After

Id	Operation	Name	Rows (Estim)	Rows (Actual)	Activity Detail (# samples)
0	SELECT STATEMENT			1102	
1	NESTED LOOPS			1102	
2	NESTED LOOPS		1106	1102	
3	TABLE ACCESS FULL	TBL_B	11	11	
4	INDEX RANGE SCAN	TEST_TABLE_A_I1	100	1102	
5	TABLE ACCESS BY INDEX ROWID	TEST_TABLE_A	101	1102	

案2. 拡張統計の採取(6)

- 今回のケースでは「式統計」を採取することで、CBO予測の誤りを補正しています。
 - CBO予測の誤りが補正されたことで、より良い実行計画が選択された。

```
SELECT /*+ MONITOR */  
  A.*  
FROM TEST_TABLE_A A  
  , TBL_B B  
WHERE A.P_NO2 = B.P_NO  
  AND A.P_CHAR = B.P_CHAR  
  AND TO_CHAR(B.P_DATE, 'YYYYMMDD') = '20120801';
```

この式統計を採取

- 拡張統計には、式統計の他に複数列統計があります。
 - 複数列統計はサブクエリ内で DISTINCT / GROUP BY を使用するケースで、グルーピングしている列値同士に相関関係が有る場合に有効です。

案3. SQLプロファイル適用 (by DBMS_SQLTUNE)

案3. SQLプロファイル適用(by DBMS_SQLTUNE)(1)

- SQLプロファイルは、DBMS_SQLTUNE のチューニング・タスク実行によって作成／提案されます。
 - リアルタイムSQL監視(REPORT_SQL_MONITOR) に並ぶ、DBMS_SQLTUNEパッケージ の有効な機能
 - DBMS_SQLTUNEパッケージの機能であるため、Enterprise Edition の Tuning Packオプションライセンスが必要です。
- 個別SQLのCBO予測(実行計画作成)を精緻化する働きを持ちます。
 - 使いこなせれば、強力なSQLチューニング手段の一つ

案3. SQLプロファイル適用(by DBMS_SQLTUNE)(2)

- 下記手順で SQLプロファイルを作成／適用できます。
 - ①チューニング・タスク作成
DBMS_SQLTUNE.CREATE_TUNING_TASK
 - ②チューニング・タスク実行
DBMS_SQLTUNE.EXECUTE_TUNING_TASK
 - ③チューニング・タスクのレポートイング
DBMS_SQLTUNE.REPORT_TUNING_TASK
 - ④SQLプロファイルの承認(※③でSQLプロファイルが提案された場合)
DBMS_SQLTUNE.ACCEPT_SQL_PROFILE

案3. SQLプロファイル適用(by DBMS_SQLTUNE)(3)

■ SQLプロファイルの作成／適用サンプル

①チューニング・タスク作成

```
SQL> VARIABLE STMT_TASK VARCHAR2(64);  
SQL> EXEC :STMT_TASK := DBMS_SQLTUNE.CREATE_TUNING_TASK(sql_id => 'g9gnrhjwajfnn', time_limit => 10800);  
Elapsed: 00:00:00.46
```

②チューニング・タスク実行

```
SQL> EXEC DBMS_SQLTUNE.EXECUTE_TUNING_TASK(:STMT_TASK);  
Elapsed: 00:12:12.02
```

③チューニング・タスクのレポートニング

```
SQL> SET LONGCHUNKSIZE 2000000 LONG 2000000 LINESIZE 300 PAGESIZE 1000;  
SQL> COLUMN REPORT FORMAT A300  
SQL> SELECT DBMS_SQLTUNE.REPORT_TUNING_TASK(:STMT_TASK) AS REPORT FROM DUAL;  
:  
1- SQL Profile Finding (see explain plans section below)
```

A potentially better execution plan was found for this statement.

Recommendation (estimated benefit: 87.03%)

- Consider accepting the recommended SQL profile.
 execute dbms_sqltune.accept_sql_profile(task_name => 'TASK_20343',
 task_owner => 'AYSHIBAT', replace => TRUE);

```
SQL> execute dbms_sqltune.accept_sql_profile(task_name => 'TASK_20343', task_owner => 'AYSHIBAT', replace => TRUE);  
Elapsed: 00:00:00.44
```

④SQLプロファイルの承認

```
SQL>
```

案3. SQLプロファイル適用(by DBMS_SQLTUNE)(4)

■ SQLプロファイルのサンプル(チューニング・タスク・レポート)

```
19:00:16 SQL> SELECT DBMS_SQLTUNE.REPORT_TUNING_TASK(:STMT_TASK)
19:00:16 2> AS REPORT FROM DUAL;
```

1- SQL Profile Finding (see explain plans section below)

A potentially better execution plan was found for this statement.

Recommendation (estimated benefit: 87.03%)

- Consider accepting the recommended SQL profile.

```
execute dbms_sqltune.accept_sql_profile(task_name => 'TASK_20343',
task_owner => 'AYSHIBAT', replace => TRUE);
```

Validation results

The SQL profile was tested by executing both its plan and the original plan and measuring their respective execution statistics. The original plan was only partially executed if the other could be run to completion in less time.

SQLプロファイルが提案されている。

	Original Plan	With SQL Profile	% Improved
Completion Status:	COMPLETE	COMPLETE	
Elapsed Time (s):	4.673067	.098527	97.89 %
CPU Time (s):	4.68	.099	97.88 %
User I/O Time (s):	0	0	
Buffer Gets:	8919	1207	86.46 %
Physical Read Requests:	0	0	
Physical Write Requests:	0	0	
Physical Read Bytes:	0	0	
Physical Write Bytes:	0	0	
Rows Processed:	1102	1102	
Fetches:	1102	1102	
Executions:	1	1	

Notes

1. Statistics for the original plan were averaged over 10 executions.
2. Statistics for the SQL profile plan were averaged over 10 executions.

SQLプロファイル
適用前の統計

SQLプロファイル
適用後の統計

案3. SQLプロファイル適用(by DBMS_SQLTUNE)(5)

■ SQLプロファイル適用後の実行統計

```
10:48:08 SQL> SELECT /** MONITOR */
10:48:08 2      A.*
10:48:08 3      FROM TEST_TABLE_A A
10:48:08 4           , TBL_B B
10:48:08 5      WHERE A.P_NO2    = B.P_NO
10:48:08 6           AND A.P_CHAR = B.P_CHAR
10:48:08 7           AND TO_CHAR(B.P_DATE, 'YYYYMMDD') = '20120801';
```

1102 rows selected.

Elapsed: 00:00:04.71

Statistics

8994 consistent gets
59 physical reads

```
19:00:18 SQL> SELECT /** MONITOR */
19:00:18 2      A.*
19:00:18 3      FROM TEST_TABLE_A A
19:00:18 4           , TBL_B B
19:00:18 5      WHERE A.P_NO2    = B.P_NO
19:00:18 6           AND A.P_CHAR = B.P_CHAR
19:00:18 7           AND TO_CHAR(B.P_DATE, 'YYYYMMDD') = '20120801';
```

1102 rows selected.

Elapsed: 00:00:00.17

Note

- SQL profile "SYS_SQLPROF_013ae9999999999999" used for this...

Statistics

1312 consistent gets
1 physical reads

性能改善!

**SQLプロファイル
が使用されている**

DBMS_XPLAN.DISPLAY_CURSOR結果の比較

Before

Id	Operation	Name	E-Rows	E-Bytes	E-Time	A-Rows	A-Time
0	SELECT STATEMENT					1102	00:00:04.65
* 1	HASH JOIN		30012	967K	00:00:32	1102	00:00:04.65
* 2	TABLE ACCESS FULL	TBL_B	300	3900	00:00:01	11	00:00:00.08
3	TABLE ACCESS FULL	TEST_TABLE_A	2600K	49M	00:00:31	2600K	00:00:01.48

After

Id	Operation	Name	E-Rows	E-Bytes	E-Time	A-Rows	A-Time
0	SELECT STATEMENT					1102	00:00:00.10
1	NESTED LOOPS					1102	00:00:00.10
2	NESTED LOOPS		1100	36300	00:00:14	1102	00:00:00.09
* 3	TABLE ACCESS FULL	TBL_B	11	143	00:00:01	11	00:00:00.08
* 4	INDEX RANGE SCAN	TEST_TABLE_A_I1	100		00:00:01	1102	00:00:00.01
5	TABLE ACCESS BY INDEX ROWID	TEST_TABLE_A	100	2000	00:00:02	1102	00:00:00.01

DBMS_SQLTUNE.REPORT_SQL_MONITOR結果の比較

Before

Id	Operation	Name	Rows (Estim)	Rows (Actual)	Activity Detail (# samples)
0	SELECT STATEMENT			1102	
1	HASH JOIN		30012	1102	Cpu (5)
2	TABLE ACCESS FULL	TBL_B	300	11	
3	TABLE ACCESS FULL	TEST_TABLE_A	3M	3M	

After

Id	Operation	Name	Rows (Estim)	Rows (Actual)	Activity (%)	Activity Detail (# samples)
0	SELECT STATEMENT			1102		
1	NESTED LOOPS			1102		
2	NESTED LOOPS		1100	1102		
3	TABLE ACCESS FULL	TBL_B	11	11		
4	INDEX RANGE SCAN	TEST_TABLE_A_I1	100	1102		
5	TABLE ACCESS BY INDEX ROWID	TEST_TABLE_A	100	1102		

案3. SQLプロファイル適用(by DBMS_SQLTUNE)(8)

- SQLプロファイルの仕組みについて、マニュアル(※)の記述は下記の通りです。
 - 17.5 SQLプロファイルの管理
SQLプロファイルには、個別の実行計画に関する情報は含まれません。オプティマイザには、計画を選択する際に、次の情報ソースがあります。
 - データベース構成、バインド変数値、オプティマイザ統計、データ・セットなどを含む環境。
 - **SQLプロファイルの補足的な統計情報。**
- ※マニュアル … Oracle Databaseパフォーマンス・チューニング・ガイド 11gリリース2(11.2)B56312-01 より

案3. SQLプロファイル適用(by DBMS_SQLTUNE)(9)

- 下記は某プロジェクトで採取した、SQLプロファイル有効時の10053トレース抜粋となります。
 - SQL内のカーディナリティやセレクトイビティに関する補足情報と考えられます。

```
:  
atom_hint=(~txt=OPT_ESTIMATE (INDEX_FILTER "B111" "xxxx11101" ROWS=2635.000000 ) )  
atom_hint=(~txt=OPT_ESTIMATE (INDEX_SCAN "B111" "xxxx11101" MIN=2635.000000 ) )  
atom_hint=(~txt=OPT_ESTIMATE (TABLE "x111" ROWS=2635.000000 ) )  
atom_hint=(~txt=OPT_ESTIMATE (GROUP_BY ROWS=7.000000 ) )  
atom_hint=(~txt=OPT_ESTIMATE (INDEX_FILTER "A002" "yyyy00201" MIN=1105.000000 ) )  
atom_hint=(~txt=OPT_ESTIMATE (INDEX_SCAN "A002" "yyyy00201" MIN=1105.000000 ) )  
atom_hint=(~txt=OPT_ESTIMATE (TABLE "y002" MIN=1105.000000 ) )  
:
```

案3. SQLプロファイル適用(by DBMS_SQLTUNE)(10)

- SQLプロファイルの正体は、「個別のSQLに特化した補助的なオプティマイザ統計」と言えます。
 - 個別SQLに対する詳細なダイナミック・サンプリングのようなもの
 - そのサンプリング結果を実体として保持している。
 - そのサンプリング結果が適用されると、CBO予測が精緻化されて実行計画が改善する。
 - SPM や アウトライン のように、実行計画そのものを保持している訳ではない。

案3. SQLプロファイル適用(by DBMS_SQLTUNE)(11)

- 前述の通りの仕組みであるため、SQLプロファイルを作成／適用する環境は、「**本番環境、又はそれに準じた量／質のデータが保持されている**」必要があります。
 - この点を理解していれば、複雑なSQLを機械的にチューニングできる、強力な武器となり得ます。
 - 機械的なチューニングによる、時間短縮／必要スキル低減／コスト削減
 - 本番環境⇔開発環境間の相互移行も可能
- 性能改善が100%保証される訳ではないので、適用後の検証／評価は必要です。

案3. SQLプロファイル適用(by DBMS_SQLTUNE)(12)

- SQLプロファイル生成時のチューニング・タスクでは、最終ハード・パースのバインド変数で、対象SQLを実際に実行して統計を採取しています。
- 「Completion Status」の値が双方「COMPLETE」であれば、SQLプロファイル適用による性能改善の確度は高いと言えます。

	Original Plan	With SQL Profile	% Improved
Completion Status:	COMPLETE	COMPLETE	
Elapsed Time (s):	4.673067	.098527	97.89 %
CPU Time (s):	4.68	.099	97.88 %
User I/O Time (s):	0	0	
Buffer Gets:	8919	1207	86.46 %
Physical Read Requests:	0	0	
Physical Write Requests:	0	0	
Physical Read Bytes:	0	0	
Physical Write Bytes:	0	0	
Rows Processed:	1102	1102	
Fetches:	1102	1102	
Executions:	1	1	
:			

SQLプロファイル適用前後の検証が完了している。

案3. SQLプロファイル適用(by DBMS_SQLTUNE)(13)

- EnterpriseManagerからも作成／適用できます。



案4. CardinalityFeedbackの使用

案4. CardinalityFeedbackの使用(1)

- CardinalityFeedback は、CBOの
実行計画予測精度を上げる機能の一つです。
 - 11gR2 で導入された新機能
- CBO予測と実行統計が乖離している際に、
2回目以降のSQL実行時に実行結果を Feedback して、
実行計画を作成し直す機能です。
 - Feedback結果によりCBO予測が精緻化されます。

案4. CardinalityFeedbackの使用(2)

- CardinalityFeedback はデフォルトで有効となっています。
- 隠しパラメータ”_optimizer_use_feedback”で制御します。

```
-- Cardinality Feedback を無効化  
ALTER SYSTEM SET "_optimizer_use_feedback" = FALSE SCOPE = BOTH;  
  
-- Cardinality Feedback を有効化  
ALTER SYSTEM SET "_optimizer_use_feedback" = TRUE SCOPE = BOTH;
```

案4. CardinalityFeedbackの使用(3)

■ CardinalityFeedback有効時の実行統計(2回目)

```
10:48:08 SQL> SELECT /** MONITOR */  
10:48:08 2      A.*  
10:48:08 3      FROM TEST_TABLE_A A  
10:48:08 4           , TBL_B B  
10:48:08 5      WHERE A.P_NO2    = B.P_NO  
10:48:08 6           AND A.P_CHAR = B.P_CHAR  
10:48:08 7           AND TO_CHAR(B.P_DATE, 'YYYYMMDD') = '20120801';
```

1102 rows selected.

Elapsed: 00:00:04.71

:

Statistics

8994 consistent gets
59 physical reads

```
21:12:15 SQL> SELECT /** MONITOR */  
21:12:15 2      A.*  
21:12:15 3      FROM TEST_TABLE_A A  
21:12:15 4           , TBL_B B  
21:12:15 5      WHERE A.P_NO2    = B.P_NO  
21:12:15 6           AND A.P_CHAR = B.P_CHAR  
21:12:15 7           AND TO_CHAR(B.P_DATE, 'YYYYMMDD') = '20120801';
```

1102 rows selected.

Elapsed: 00:00:00.14

Statistics

1301 consistent gets
1 physical reads

性能改善!

DBMS_XPLAN.DISPLAY_CURSOR結果の比較

■ Before

Id	Operation	Name	E-Rows	E-Bytes	E-Time	A-Rows	A-Time
0	SELECT STATEMENT					1102	00:00:04.65
* 1	HASH JOIN		30012	967K	00:00:32	1102	00:00:04.65
* 2	TABLE ACCESS FULL	TBL_B	300	3900	00:00:01	11	00:00:00.08
3	TABLE ACCESS FULL	TEST_TABLE_A	2600K	49M	00:00:31	2600K	00:00:01.48

■ After(2回目の実行計画)

Id	Operation	Name	E-Rows	E-Bytes	E-Time	A-Rows	A-Time
0	SELECT STATEMENT					1102	00:00:00.10
1	NESTED LOOPS					1102	00:00:00.10
2	NESTED LOOPS		1100	36300	00:00:14	1102	00:00:00.09
* 3	TABLE ACCESS FULL	TBL_B	11	143	00:00:01	11	00:00:00.08
* 4	INDEX RANGE SCAN	TEST_TABLE_A_I1	100		00:00:01	1102	00:00:00.01
5	TABLE ACCESS BY INDEX ROWID	TEST_TABLE_A	100	2000	00:00:02	1102	00:00:00.01

DBMS_SQLTUNE.REPORT_SQL_MONITOR結果の比較

▪ Before



Id	Operation	Name	Rows (Estim)	Rows (Actual)	Activity Detail (# samples)
0	SELECT STATEMENT			1102	
1	HASH JOIN		30012	1102	Cpu (5)
2	TABLE ACCESS FULL	TBL_B	300	11	
3	TABLE ACCESS FULL	TEST_TABLE_A	3M	3M	

▪ After(2回目の実行計画)

Id	Operation	Name	Rows (Estim)	Rows (Actual)	Activity Detail (# samples)
0	SELECT STATEMENT			1102	
1	NESTED LOOPS			1102	
2	NESTED LOOPS		1106	1102	
3	TABLE ACCESS FULL	TBL_B	11	11	
4	INDEX RANGE SCAN	TEST_TABLE_A_I1	100	1102	
5	TABLE ACCESS BY INDEX ROWID	TEST_TABLE_A	101	1102	

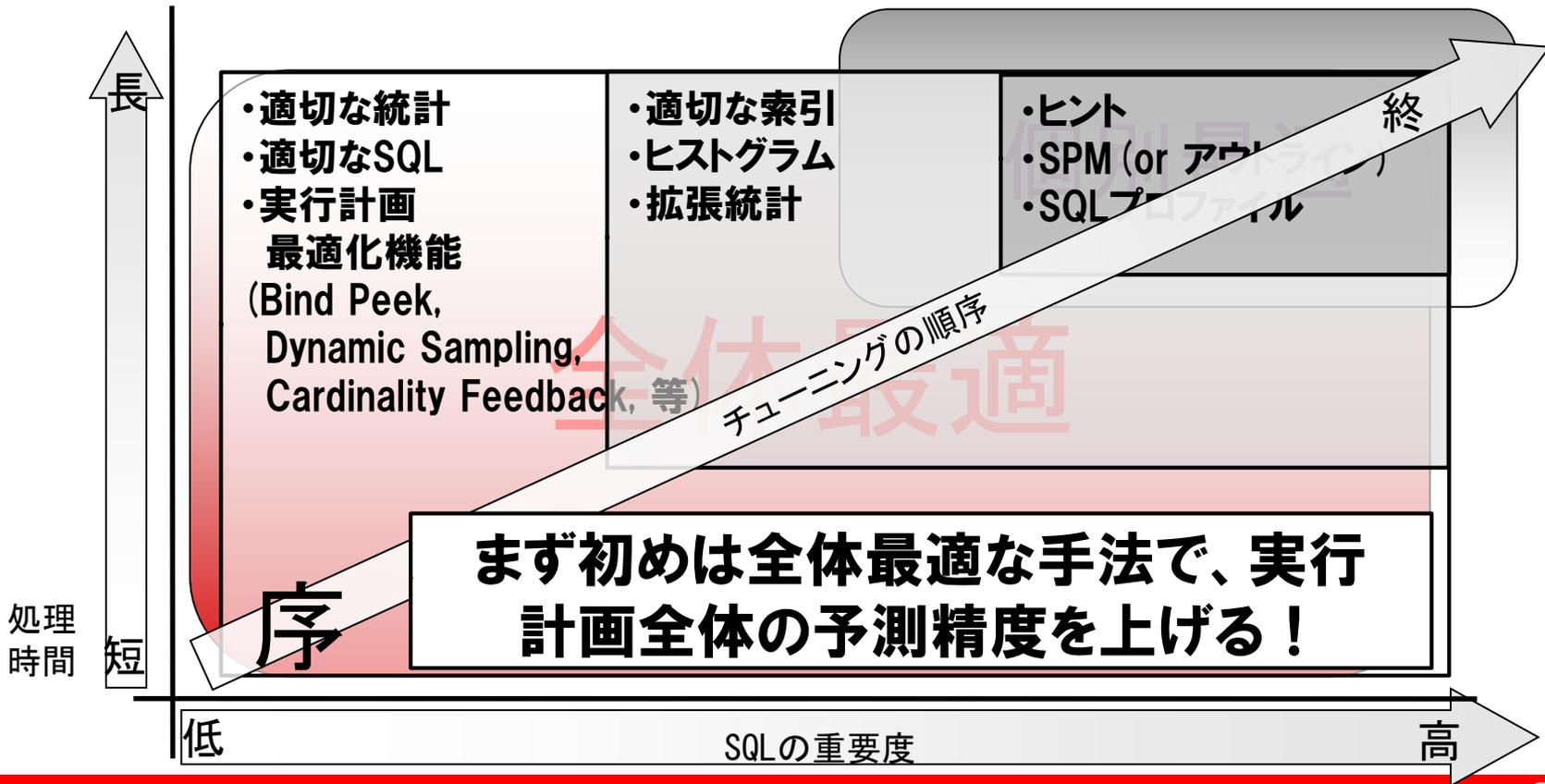
案4. CardinalityFeedbackの使用(6)

- 今回のケースでは「CardinalityFeedback」により、2回目のSQLでCBO予測の誤りが補正されています。
 - CBO予測の誤りが補正されたことで、より良い実行計画が選択された。

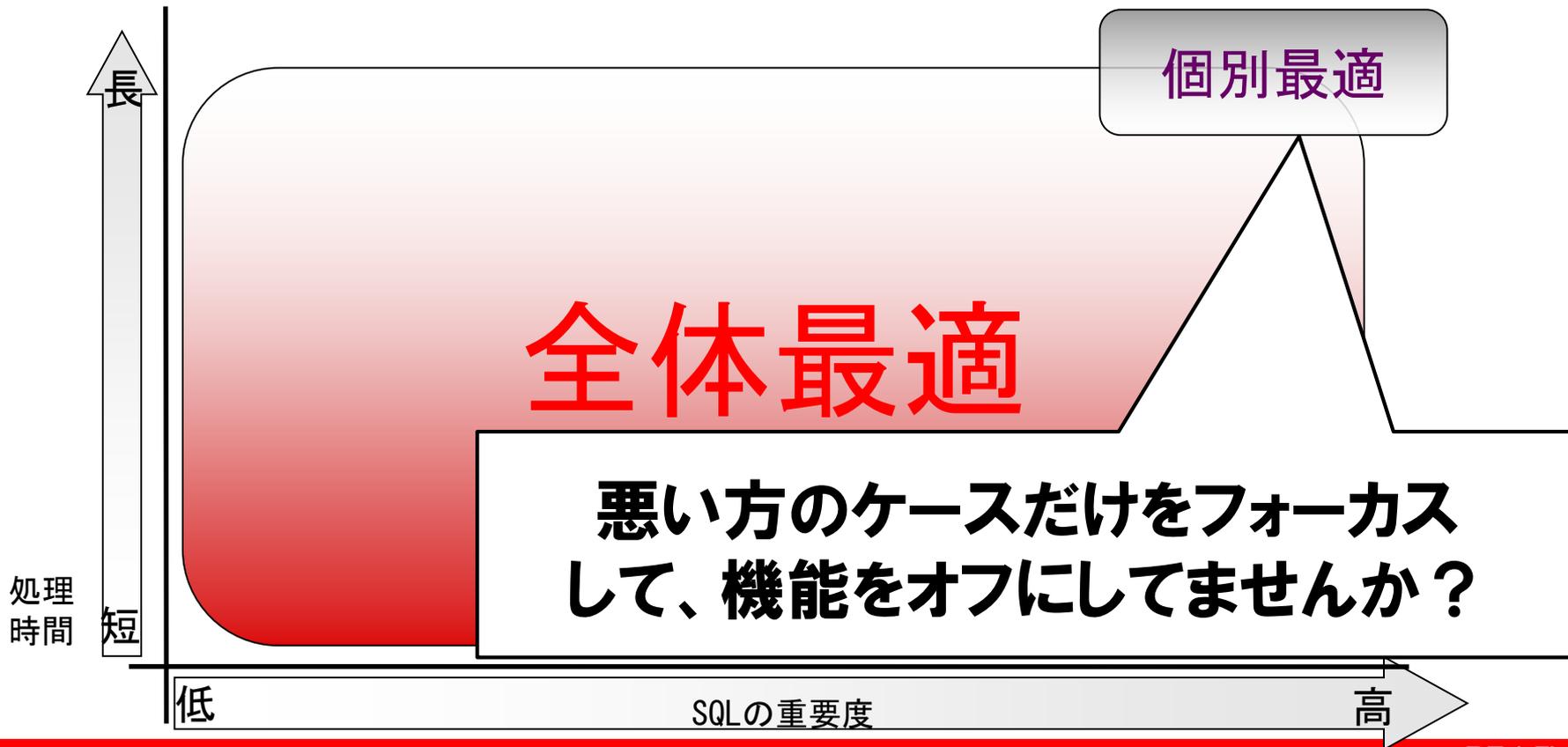
案4. CardinalityFeedbackの使用(7)

- 下記に挙げる機能は、CBOによる実行計画の予測精度を上げる働きを持っています。
 - BindPeek , CardinalityFeedback , DynamicSampling
- これらの機能は「実行計画の安定化」と云うリスクヘッジ目的で、無効化されているシステムも多いと認識しています。
- しかしこれらの機能を無効化することは、CBOの機能／性能をスポイルしている、と云う事実も忘れないで下さい。
 - ハズレの実行計画を引く確率が高まる、、、と云うリスクの増加
 - 10gR2 までの BindPeek の欠点(※初回PARSE時のプランで固定される)は、11gR1導入の「優れたカーソル共有」で改善されています。
 - 現在はBindPeek使用も有力な選択肢で、false一択ではありません。

「全体最適」と「個別最適」の適用イメージ(※再掲)



案4. CardinalityFeedbackの使用(8)



紹介したチューニング案の方向性について

- 今回紹介したのは、全て「予測(Estimate)」と「実測(Actual)」の乖離を補正する方向性でチューニングを実施しています。
- 既に述べた通り、下記のようなチューニングも可能でした。
 - 案5 … ヒント や SPM による実行計画操作
 - 案6 … パラレル・クエリ化
 - 案7 … SQL修正(WHERE句書き換え)
 - 案8 … SQL修正(WITH句によるサブクエリ切り出し)
 - 案9 … Dynamic Sampling適用
 - 案10…新規索引付与
 - 案11…リザルト・セットの適用
- チューニングの正解は1つではありません。併せて利用していきましょう。

4章. まとめ

SQL と オプティマイザ が抱える弱点

- SQL と RDBMSのオプティマイザ は、その言語の仕組みに由来する本質的な弱点を抱えています。
 - SQLはアルゴリズムを記述しないと云う特徴があるため、それ(アルゴリズム)がRDBMS(Optimizer)任せになる。
 - SQLのアルゴリズム≡実行計画は、コストと云う基準の「予測」で導出されている。
 - 「予測」である以上、**ハズレ**のケースが出てくる。

SQLチューニングを上手くやるには。。。

- SQL や オプティマイザ の弱点／アンチパターンを知っておく。
 - 実行計画は「予測」であり、ハズレが有り得ることを認識する。
 - 予測がハズれる、ハズレ易いケースとは？
- 弱点／アンチパターンの対抗手段を見つけておく。
 - 予測精度を向上させる設計技法や機能の活用
 - 「予測(Estimate)」と「実測(Actual)」を補正するというアプローチ
 - アンチパターンをダイレクトに潰す手法も有効
- Oracle Database の機能を利用して、「予測」と「実測」の乖離を捉える。
 - DBMS_XPLAN.DISPLAY_CURSOR(※ALLSTATS書式)
 - DBMS_SQLTUNE.REPORT_SQL_MONITOR(リアルタイムSQL監視)

Hardware and Software

ORACLE®

Engineered to Work Together

ORACLE®