Oracle SQL Tuning

3 parts (BASIC, INTERMEDIATE, ADVANCED)

www.lingaro.com

lingaro

The Value of Business Intelligence

Training Agenda

- BASIC
  - SQL Tuning Introduction
  - Instrumentation & Rewrite SQL Text
  - SQL Plan Reading
- INTERMEDIATE
  - Optimizer & Statistics
  - SQL Plan Tuning
  - SQL Plan Operations
- ADVANCED
  - SQL Processing & Cursor Sharing
  - Advanced Plan Operations
  - SQL Plan Transformations
  - SQL Plan Management

Training Agenda

- Theory
- Quiz Comp
- Workshop
- Workshop Comp
- Winner Awards On WINS

Topic Agenda

Tuning Techniques (not prepared yet)

- Using Automatic SQL Tuning
- Using SQL Tuning & SQL Access Advisors (dbms_sqltune or OEM)
- Using Oracle Enterprise Manager Console OEM
- Using DBA_HIST and V$ performance views (described here)
- Using Test Executions
- Using SQL Decomposition
  - Test Execution on SQL parts

Oracle SQL Tuning

ADVANCED

www.lingaro.com

lingaro

The Value of Business Intelligence

Topic Agenda

SQL Processing

- Introduction
- Processing Steps
- Workload Characteristics
- SQL Memory Structures
- Cursor Sharing

Introduction

SQL Processing

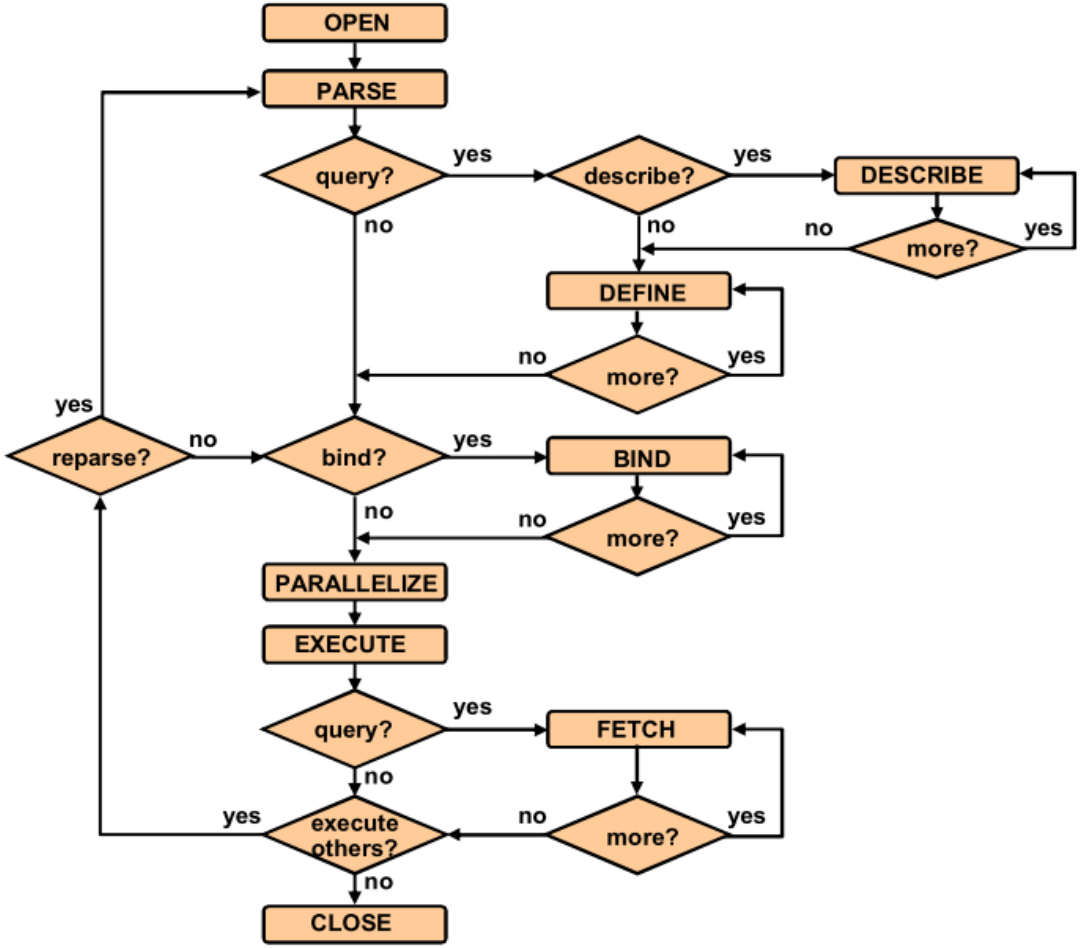
- Is done inside Oracle Database Server SQL engine
- Oracle client send SQL text & receive results from server
  - by network - connected via database service
  - by local connection - connected to local instance using ORACLE_SID
- Server steps:

	Query	DML	DDL
Parse	v	v	v
Bind	v	v	
Execute	v	v	
Fetch	v		

- SQL optimization is done by Oracle Optimizer during parse
  - Has significant influence on execution and fetch phases costs
  - Parse phase cost is more important in OLTP workload

Introduction

SQL Processing Steps



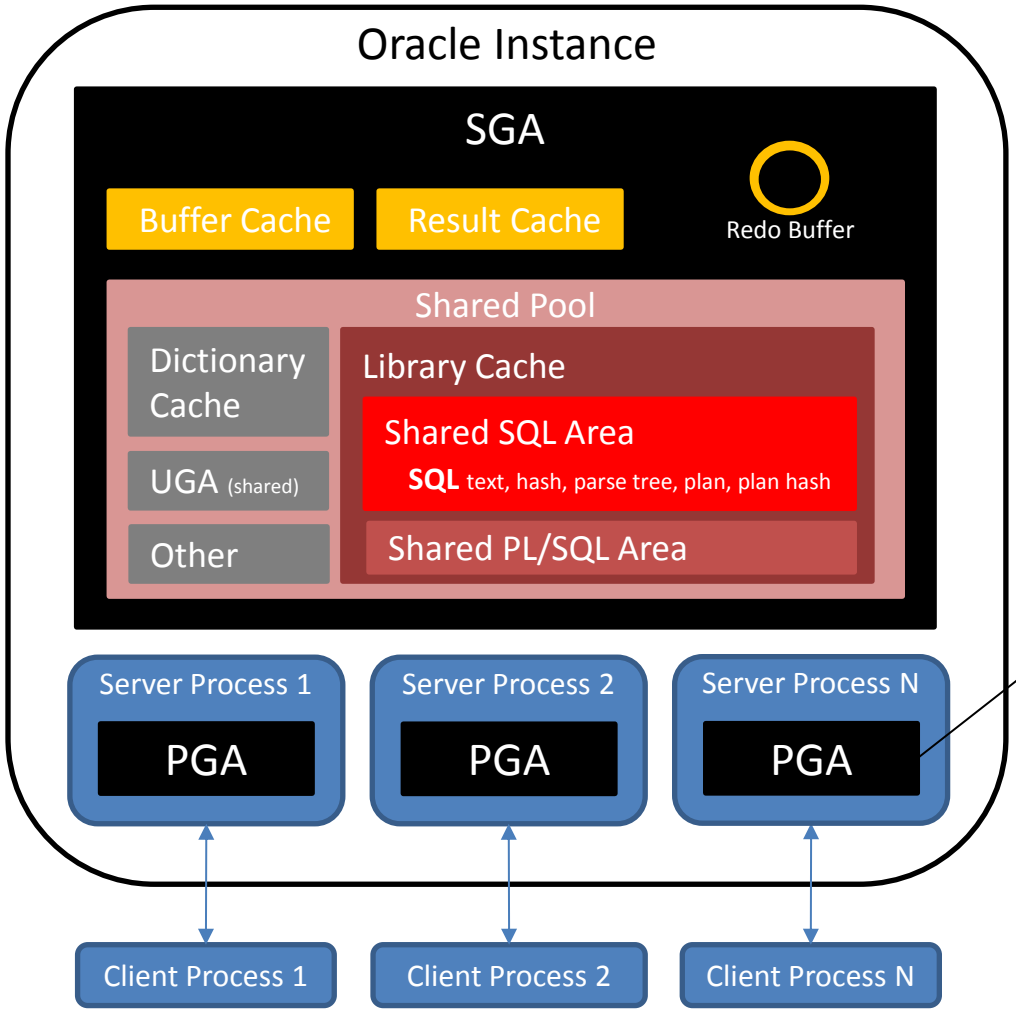
SQL Processing

Workload characteristic

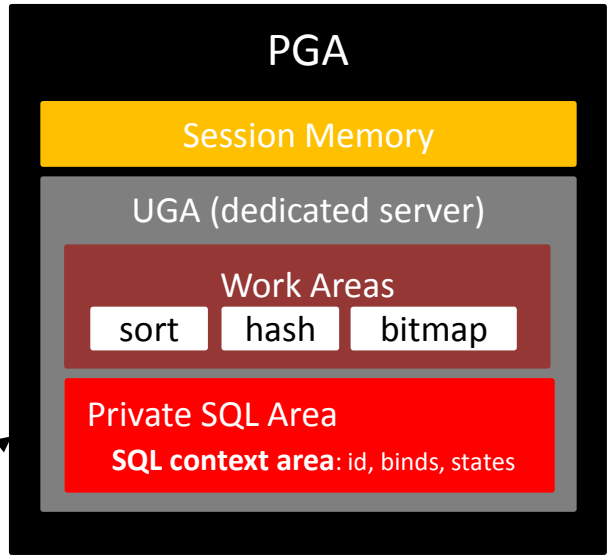
- OLTP workload
  - Large number of short time single/few rows query and DML
  - All SQLs summary parse cost is big and more important
  - SQL optimization is less important
  - Main tuning goal – response time
  - Critical resource – CPU, memory
  - Top blocking event - contention

- DSS/Warehouse workload
  - average number of long running large data volume reports and ETL
  - execution and fetch cost is bigger and more important
  - SQL optimization is very important
  - Main tuning goal – throughput
  - Critical resource – CPU, storage IO
  - Top blocking event – ON CPU - processing

SQL Memory Structures



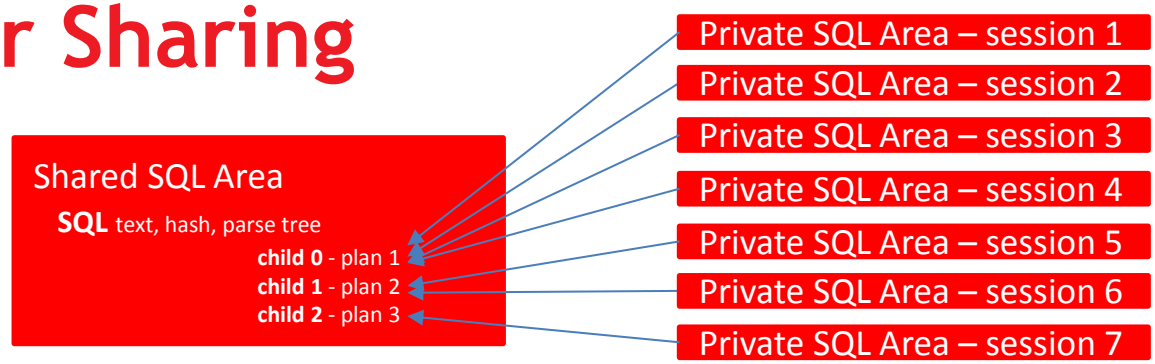
Cursor is
name, pointer or handle
to SQL context area
(linked to child of shared SQL area)



SQL Processing

Cursor Sharing

One SQL Example



- Multiple executions of the same SQL uses the same shared SQL area
- **Reduces hard parses** - plan generation **memory & CPU costs**
- Can be done by
  - Reusing the same SQL statement many times using code standardization and libraries
  - Changing many similar SQLs to one SQL using Bind Variables instead of literal values in SQL text
    - by developer - PL/SQL source code change - use of PL/SQL variables

      e.g. `chanl_id = 102, chanl_id = 105, ...` -> `chanl_id = l_chanl_id`

      - or SQL script session variables - host variables

      e.g. `chanl_id = 102, chanl_id = 105, ...` -> `chanl_id = :g_chanl_id`

    - by DBA or developer - set CURSOR_SHARING parameter - bind variables automatically generated
      - EXACT - default value - SQL text need to be identical to share cursor
      - - if text is different separate SQL area is allocated in library cache
      - SIMILAR - bind variables used if plan not need to be changed when bind value is changed
      - FORCE - deterministic bind variables usage

        e.g. `ALTER SESSION SET cursor_sharing='SIMILAR';`

Cursor Sharing

Reusing the same SQL statement many times

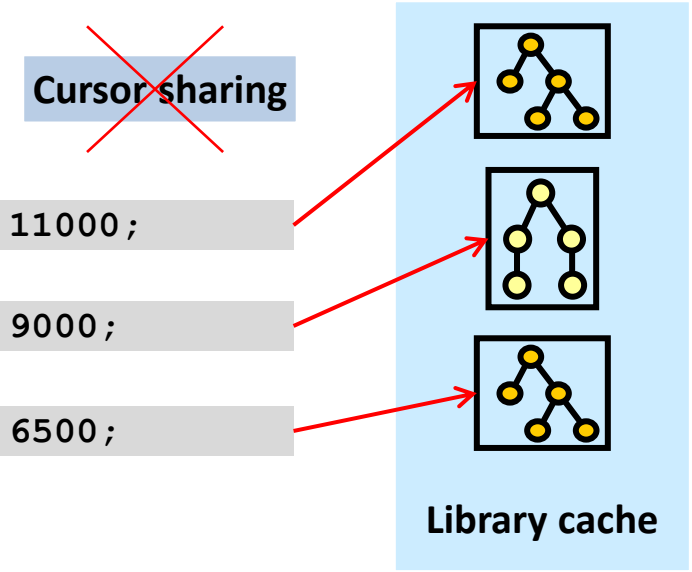
- SQL source code standardization
  - Like: keywords uppercase, identifiers lowercase, single space separator, two space indentation
  - Standards make SQLs to be more likely reused - Example:

```
SELECT /*+PARALLEL(2)*/
    TO_CHAR(f.sales_date,'YY/MM/DD') AS wk_end_date,
    f.gln                            AS store_id,
    SUM(pos_amt)                     AS sales_amt
  FROM gpos_cp_fads PARTITION("3663_CP") f,
       prod_extrn_dads p
  WHERE f.srce_sys_id = 3663
    AND f.prod_extrn_id = p.prod_extrn_id
    AND f.srce_sys_id = p.srce_sys_id
    AND ((f.pos_amt != 0 AND f.pos_amt IS NOT NULL)
      OR (f.pos_qty != 0 AND f.pos_qty IS NOT NULL))
  GROUP BY
    f.sales_date,
    f.gln;
```

- Using shared library modules
  - PLSQL procedures containing static SQL statements
  - Avoid using dynamic SQL
  - If dynamic SQL is necessary then use bind variables inside dynamic SQL code

Bind Variables

Not used

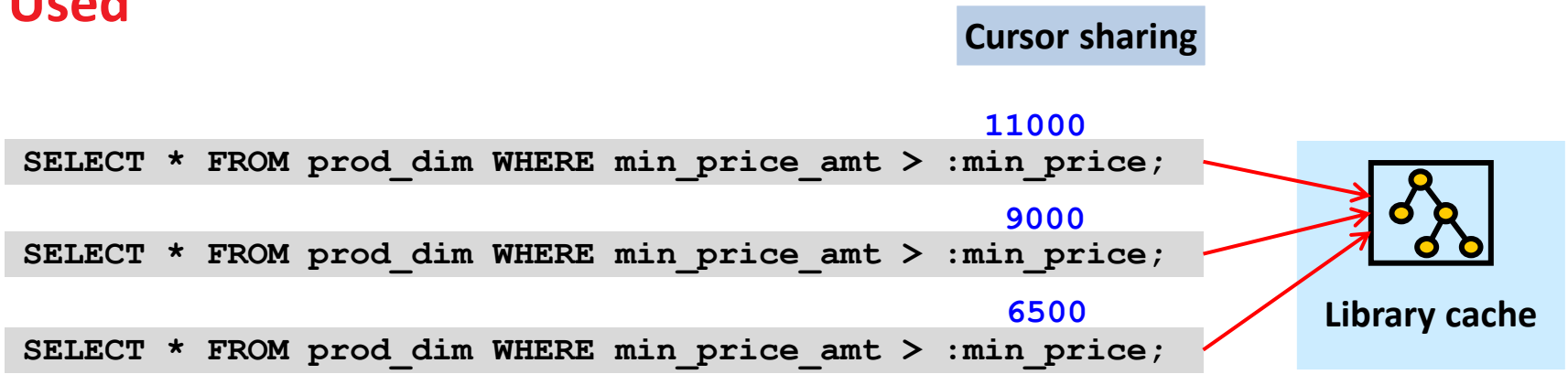



```
SELECT * FROM prod_dim WHERE min_price_amt > 11000;

SELECT * FROM prod_dim WHERE min_price_amt > 9000;

SELECT * FROM prod_dim WHERE min_price_amt > 6500;
```

- Pros
  - Each filter constat value have its own execution plan
- Cons
  - Each constant value produce separate SQL statement with different SQL_ID
  - Many plan tuning techniques needs constant SQL_ID for tuned statement
  - Large number of sql statements
  - Large memory usage and statements compilation CPU costs

Bind Variables

Used



- **Pros**
  - One statement executed many tims
  - One plan for all executions - lower CPU cost for compilation
  - All plan tuning techniques possible
- **Cons**
  - One SQL plan can be not optimal for all bind varaible values

    but Adaptive Cursor Sharing makes many plans for 1 SQL possible

Cursor Sharing

Matching Signatures

```
select 0 from dual where dummy = 'A';
select 0 from dual where dummy = 'B';
select 0 from dual where dummy = 'C';
select 0 from dual where dummy ='A';

SELECT  force_matching_signature, exact_matching_signature, sql_text
  FROM  v$sqlarea
  WHERE sql_text like 'select 0%';
```

FORCE_MATCHING_SIGNATURE	EXACT_MATCHING_SIGNATURE	SQL_TEXT
13154199455204052618	8255937935626560799	select 0 from dual where dummy ='A'
13154199455204052618	8255937935626560799	select 0 from dual where dummy = 'A'
13154199455204052618	4228015713888331644	select 0 from dual where dummy = 'C'
13154199455204052618	16821029007583452703	select 0 from dual where dummy = 'B'

Cursor Sharing

Finding SQL candidates to use

– source code standardization

```
SELECT
  ''''||exact_matching_signature||''',''
  FROM  v$sqlarea
  WHERE exact_matching_signature <> 0
  GROUP BY exact_matching_signature
  HAVING count(*) > 2;

SELECT exact_matching_signature, sql_id,
       count(*) OVER (PARTITION BY
          exact_matching_signature) cnt
  FROM v$sqlarea
  WHERE exact_matching_signature IN (
    '16523011601676190874',
    '29458002838394233321',
    '2672114946588399948',
    '3854345704683617056',
    '14461745472358877479'
  ) ORDER BY cnt DESC,1;
```

– bind variables implementation

```
SELECT
    ''''||force_matching_signature||''',''
  FROM  v$sqlarea
  WHERE force_matching_signature <> 0
  GROUP BY force_matching_signature
  HAVING count(*) > 100;

SELECT force_matching_signature, sql_id,
       count(*) OVER (PARTITION BY
         force_matching_signature) cnt
  FROM v$sqlarea
  WHERE force_matching_signature IN (
      '78814178434052199 78',
      '16841543523900648019',
      '1106612903482690771',
      '12296313255013650141',
      '6556405521552985841',
      '8931422929029301538'
  ) ORDER BY cnt DESC,1;
```

Cursor Sharing

SQL Information from V$SQLSTAT

```
SELECT sql_id, version_count, executions, invalidations, parse_calls,
       rows_processed, cpu_time, elapsed_time, avg_hard_parse_time
  FROM v$sqlstats WHERE sql_text LIKE 'update ENGINE_INSTANCES%'
```

SQL_ID	VERSION_COUNT	EXECUTIONS	INVALIDATIONS	PARSE_CALLS	ROWS_PROCESSED	CPU_TIME	ELAPSED_TIME	AVG_HARD_PARSE_TIME
akbss8gya5a28	5	182558	0	182558	182558	167678549	392783430	19182

- executions = parse calls (soft parse)
  - client doing single SQL execution in one database call
- executions > parse calls
  - single anonymous PL/SQL call run doing many executions of the same SQL
  - only first run of PL/SQL procedure increase PARSE_CALLS of included SQL
- executions < parse calls
  - statement parsed but not executed
  - statement additionally parsed during execution (purged or invalidated)

Cursor Sharing

SQL Cursor Versions (Childs) Information from V$SQL

```
SELECT sql_id, hash_value, address, plan_hash_value,
       child_number, executions, parsing_schema_name
  FROM v$sql WHERE sql_id = 'akbss8gya5a28' ORDER BY child_number;
```

SQL_ID	HASH_VALUE	ADDRESS	PLAN_HASH_VALUE	CHILD_NUMBER	EXECUTIONS	PARSING_SCHEMA_NAME
akbss8gya5a28	4238518344	00000018C23FEBB8	3804546214	0	33843	ADWGP_REA
akbss8gya5a28	4238518344	00000018C23FEBB8	3804546214	1	94318	ADWGP_REA
akbss8gya5a28	4238518344	00000018C23FEBB8	2055260210	2	13945	ADWGP_REA_LOAD_BAL
akbss8gya5a28	4238518344	00000018C23FEBB8	2055260210	3	34772	ADWGP_REA_LOAD_BAL
akbss8gya5a28	4238518344	00000018C23FEBB8	4278141064	4	6037	ADWG_IBA_CRS_ETL

```
SELECT child_number, auth_check_mismatch, bind_mismatch
  FROM v$sql_shared_cursor WHERE sql_id='akbss8gya5a28';
```

CHILD_NUMBER	AUTH_CHECK_MISMATCH	BIND_MISMATCH
0	N	N
1	N	Y
2	Y	N
3	Y	Y
4	Y	N

Cursor Sharing

Adaptive Cursor Sharing: Overview

- Allows for intelligent cursor sharing for statements that use bind variables
- Works for bind variables produced in source code by developers or produced automatically using `CURSOR_SHARING` parameter.
- Is used to trade off between cursor sharing and plan optimization
- Has the following benefits:
  - Automatically detects when different executions would benefit from different execution plans – statements marked as "bind aware"
  - Limits the number of generated child cursors to a minimum
  - Provides an automated mechanism that cannot be turned off

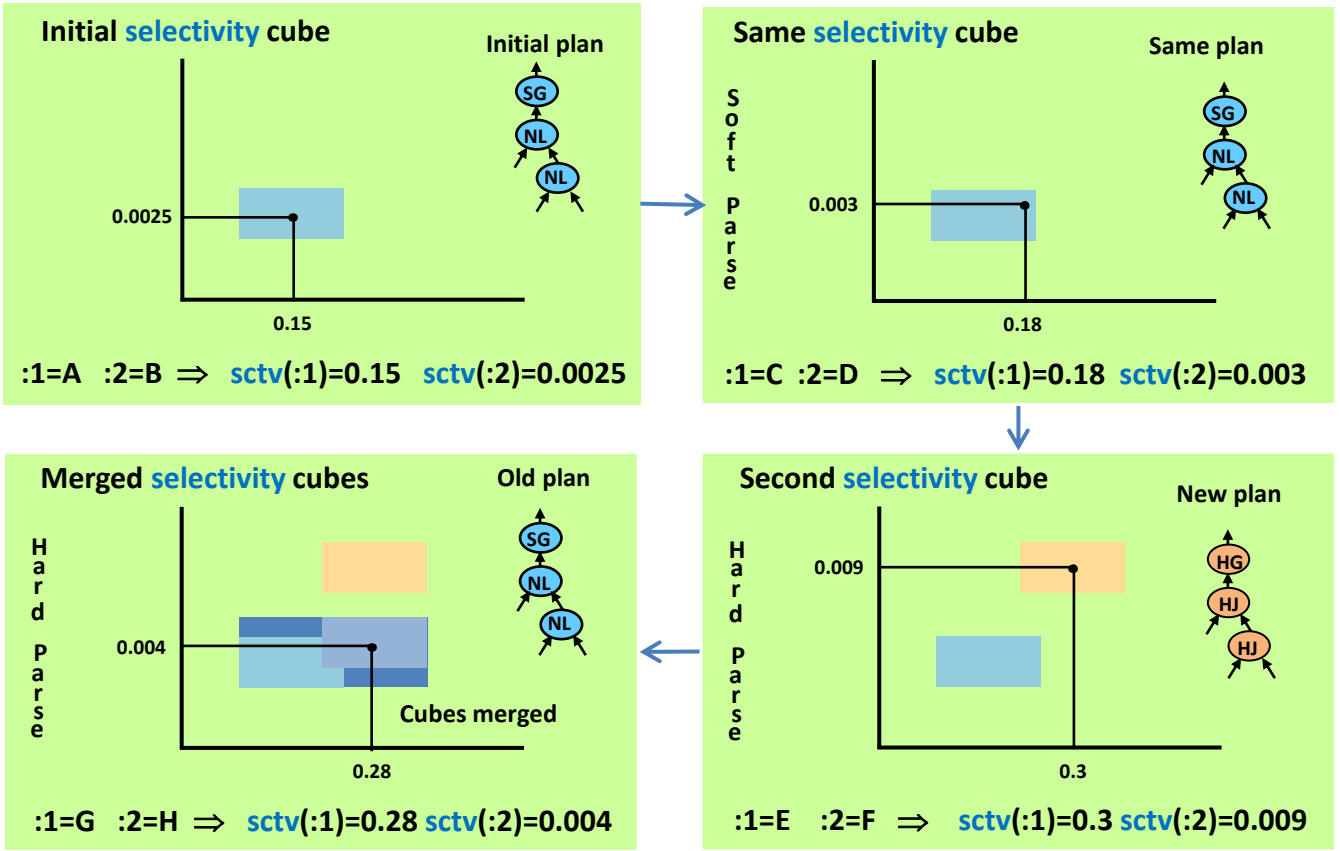
Cursor Sharing

Adaptive Cursor Sharing: Architecture

Bind-sensitive cursor → System observes statement

```
SELECT * FROM emp WHERE sal = :1 and dept = :2
```

Bind-aware cursor



Cursor Sharing

Adaptive Cursor Sharing: Views

V$SQL	Two columns show whether a cursor is bind sensitive or bind aware.
V$SQL_CS_HISTOGRAM	Shows the distribution of the execution count across the execution history histogram
V$SQL_CS_SELECTIVITY	Shows the selectivity cubes stored for every predicate containing a bind variable and whose selectivity is used in the cursor sharing checks
V$SQL_CS_STATISTICS	Shows execution statistics of a cursor using different bind sets

SQL Processing Steps

Parse Step

Soft Parse

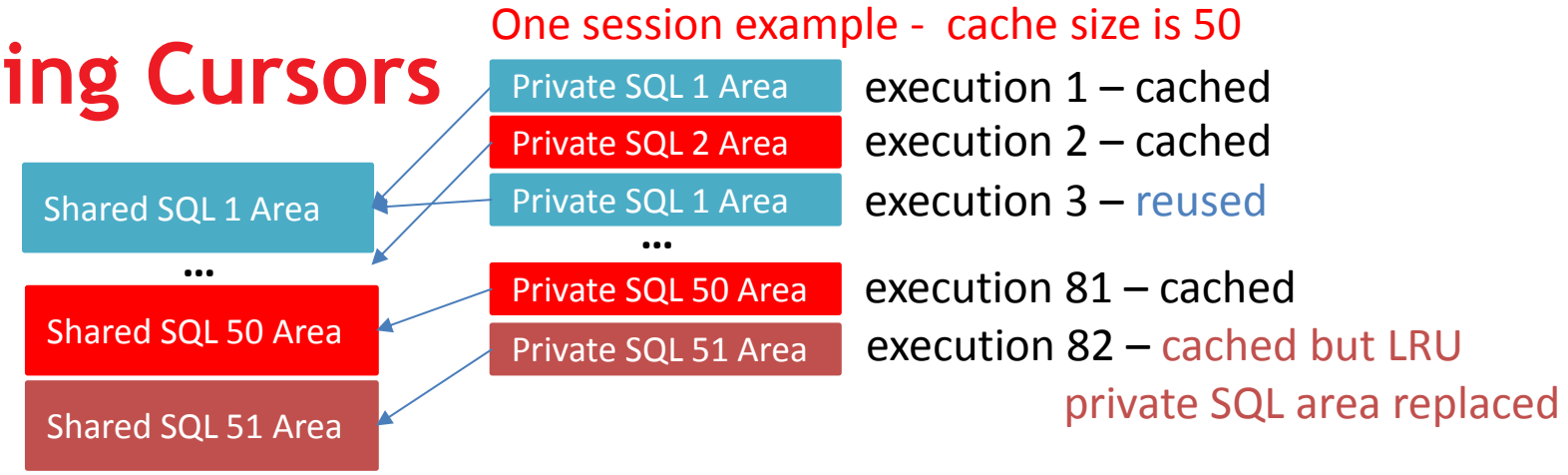
- Determining statement type (query, DML, DDL)
- Syntax Checks: keywords, grammar, rules
- Semantic Analysis Checks:
  - referenced objects exists
  - have access to referenced objects
  - ambiguities
- Finding if same statement was executed (Shared Pool Check)
  - Using hash from SQL text to find same statement was executed
    * on same session (exists in private SQL area if cursor is cached)
    * on different sessions (exists in shared SQL areas)

Hard Parse

- Optimization & Estimation
- Execution Plan Generation

done by Oracle Optimizer (next topic)

Caching Cursors



- Multiple executions of the same SQL in the same session
  - Can use the same session cached cursor
- Persistent part of private area is retained even cursor is closed
  - (after 3-th same SQL execution on the session)
- Can reduce private area generation costs - **soft parse** will be cheaper
- Enable to find shared SQL area handle in private SQL area
  - decrease library cache latch wait events and parsing serialization
  - increase scalability
- Size can be modified by

```
ALTER SESSION SET session_cached_cursors = 100  -- default 50 cursors cached
```

- Diagnostic

```
SELECT name, value FROM v$statname NATURAL JOIN v$mystat
  WHERE LOWER(name) LIKE '%cursor ca%';
```

NAME	VALUE
session cursor cache hits	49
session cursor cache count	37

SQL Processing Steps

Open Step (environment and language dependent)

- Generating cursor identifier before parse phase
- PL/SQL example

```
  c_cur_num INTEGER;
BEGIN
  c_cur_num := dbms_sql.open_cursor;
```

- PL/SQL OPEN statement

```
 c_cur CURSOR IS SELECT ...;
BEGIN
  OPEN c_cur;
```

  - Is doing almost all processing steps

    `open, parse, bind, execution`

  - Is not doing fetch step

SQL Processing Steps

Other (less important) Steps

- Describe results (query only)
  - Datatypes, lengths, and names of columns for query result determination
  - Only necessary if the characteristics of a query's result are not known
- Define output (query only)
  - Location, size, and datatype of variables used to receive fetched value.
  - Oracle performs datatype conversion here if necessary.
- Bind
  - Assign value to bind variable
- Close cursor
  - Release none persistent part of private SQL area

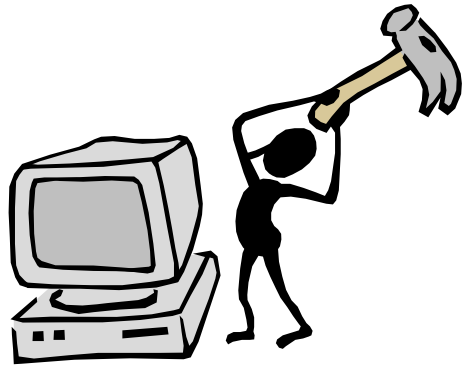
Q & A

+ Quiz

Q & A

+ Workshop Comp

Cursor Sharing
Workshop



- Implement bind variables
  - Use bind variables in wksh_ts_cur_sh_pro procedure
  - See statistics for test execution after reconnect before and after change

```
SELECT name, value FROM v$mystat NATURAL JOIN v$statname
  WHERE name IN ('parse time cpu', 'parse count (hard)');
```

  - before

```
parse time cpu          33
parse count (hard)     504
```

  - after

```
parse time cpu           0
parse count (hard)       2
```

Topic Agenda

Advanced Plan Operations

- Sort & Grouping Operations
- Partition Pruning
- Parallel Plan
  - Introduction
  - Interpretetion
  - Rows Distribution
  - Tuning

Sort Operations

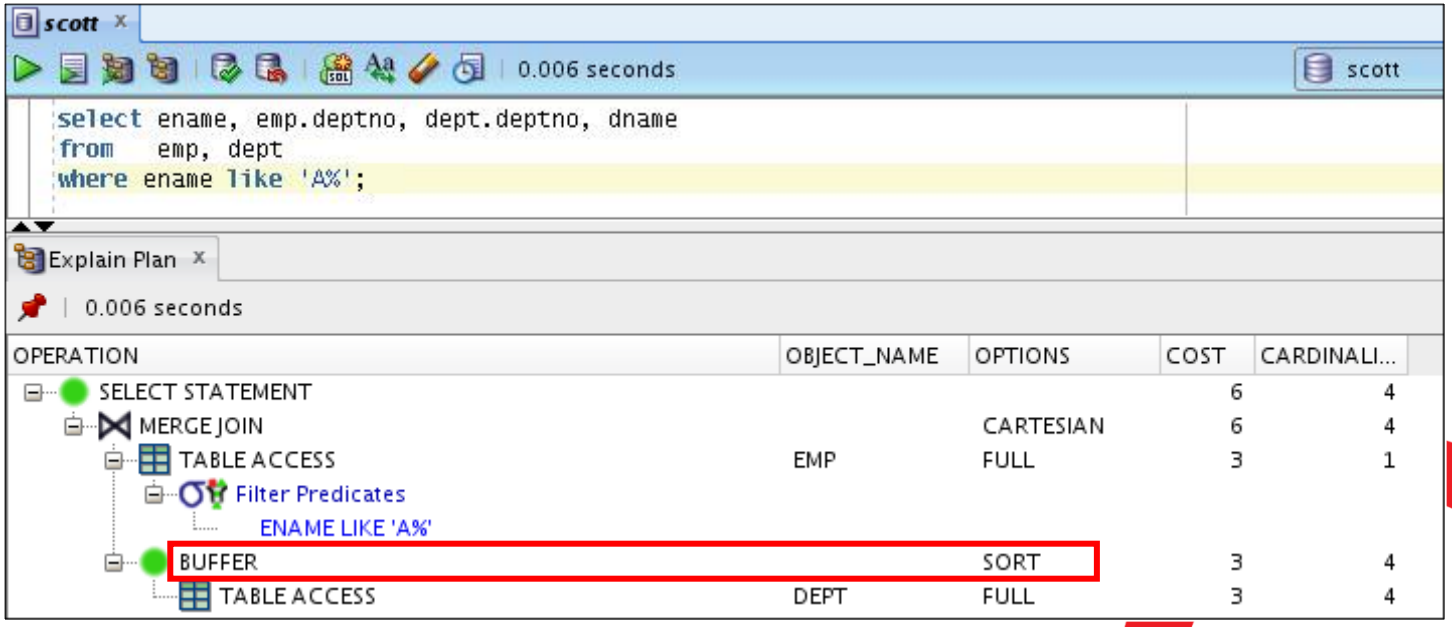
List

- **SORT** operators
  - **AGGREGATE** Single row from group function
  - **UNIQUE** To eliminate duplicates (DISTINC, UNION, MINUS, INTERSECT)
  - **JOIN** Precedes a merge join
  - **GROUP BY**
  - **ORDER BY**
- **HASH** operators
  - **UNIQUE** Equivalent to **SORT UNIQUE**
  - **GROUP BY** Equivalent to **SORT GROUP BY** but not sorted results
  - If you want ordered results, always use **ORDER BY**
- **BUFFER SORT**

Sort Operations

BUFFER SORT

- Used when operation needs all the input data before it can start.
- Used to avoid multiple table scans against real data blocks.
- Uses a temporary table or a UGA sort area to store intermediate data.
- Swaps sort area to temporary tablespace when UGA sort area is full.
- Data is not necessarily sorted - **it is not sort itself**.

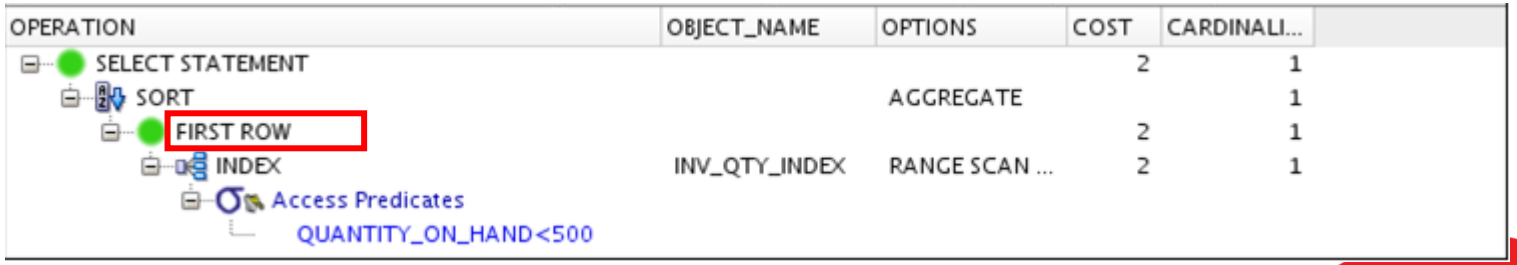


Other Operations

Min/Max FIRST ROW

- Retrieves only the first row selected by a query.
- Stops accessing the data after the first value is returned.
- Works with the index range scan and the index full scan.
- On slide - there is an index on the quantity_on_hand column.

```
SELECT MIN(quantity_on_hand)
FROM INVENTORIES
WHERE quantity_on_hand < 500;
```



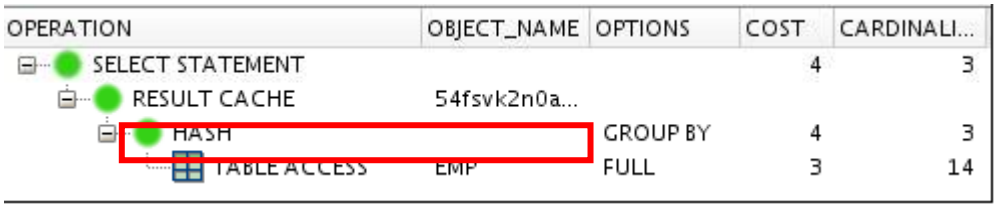
Other Operations

RESULT CACHE

Hint: RESULT_CACHE

- Use for SQL executed frequently on large volume but returning small results
- SQL results are stored in server memory and used on next execution
- Cached results are invalidated after source table modification
- Monitoring
  - V$RESULT_CACHE_STATISTICS - result cache settings and memory usage statistics
  - V$RESULT_CACHE_MEMORY - memory blocks in result cache
  - V$RESULT_CACHE_OBJECTS - objects whose results are in
  - V$RESULT_CACHE_DEPENDENCY - what results are dependent on
  - USER_TABLES - RESULT_CACHE column that shows the result cache mode: DEFAULT/FORCE/MANUAL

```
SELECT /*+ RESULT_CACHE */ deptno, AVG(sal)
FROM emp GROUP BY deptno;
```



OPERATION	OBJECT_NAME	OPTIONS	COST	CARDINALI...
SELECT STATEMENT			4	3
RESULT CACHE	54fsvk2n0a...			
HASH		GROUP BY	4	3
TABLE ACCESS	EMP	FULL	3	14

Partition Pruning

Examples

- **Static**

```
SELECT * FROM sales_fct WHERE time_id = TO_DATE('2001.01.01');
```

In Pstart, Pstop:
partition numbers

Operation for range filter:
PARTITION RANGE ITERATOR
All partitions chosen:
PARTITION RANGE ALL

```
| Id | Operation                | Name  | Pstart| Pstop |
-------------------------------------------------------------
|  0 | SELECT STATEMENT         |       |       |       |
|  1 |  PARTITION RANGE SINGLE  |       | 17    | 17    |
|* 2 |   TABLE ACCESS FULL      | SALES | 17    | 17    |
```

- **Dynamic**

```
SELECT SUM(sold_amt) FROM sales_fct WHERE time_id IN
     (SELECT time_id FROM time_dim WHERE fisc_yr = 2000);
```

In Pstart, Pstop:
KEY – data convert
KEY(SQ) – subquery
KEY(I) – bind variable

```
| Id  | Operation                | Name  | Pstart| Pstop |
-------------------------------------------------------------
|   0 | SELECT STATEMENT         |       |       |       |
|   1 |  SORT AGGREGATE          |       |       |       |
|*  2 |   HASH JOIN              |       |       |       |
|*  3 |    TABLE ACCESS FULL     | TIMES |       |       |
|   4 |    PARTITION RANGE SUBQUERY|     |KEY(SQ)|KEY(SQ)|
|   5 |     TABLE ACCESS FULL    | SALES |KEY(SQ)|KEY(SQ)|
```

Partition Pruning

Bloom Filter

Hint: PX_JOIN_FILTER
Opp.: NO_PX_JOIN_FILTER

```
-------------------------------------------------------------------------------
| Id  | Operation                                | Name      | Pstart| Pstop |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                         |           |       |       |
|   1 |  PX COORDINATOR                          |           |       |       |
|   2 |   PX SEND QC (RANDOM)                    | :TQ10002  |       |       |
|*  3 |    FILTER                                |           |       |       |
|   4 |     HASH GROUP BY                        |           |       |       |
|   5 |      PX RECEIVE                          |           |       |       |
|   6 |       PX SEND HASH                       | :TQ10001  |       |       |
|   7 |        HASH GROUP BY                     |           |       |       |
|*  8 |         HASH JOIN                        |           |       |       |
|   9 |          PART JOIN FILTER CREATE         | :BF0000   |       |       |
|  10 |           PX RECEIVE                     |           |       |       |
|  11 |            PX SEND PARTITION (KEY)       | :TQ10000  |       |       |
|  12 |             PX BLOCK ITERATOR            |           |       |       |
|  13 |              TABLE ACCESS FULL           | CUSTOMERS |       |       |
|  14 |          PX PARTITION HASH JOIN-FILTER   |           |:BF0000|:BF0000|
|* 15 |           TABLE ACCESS FULL              | SALES     |:BF0000|:BF0000|
-------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   3 - filter(COUNT(SYS_OP_CSR(SYS_OP_MSR(COUNT(*)),0))>100)
   8 - access("S"."CUST_ID"="C"."CUST_ID")
  15 - filter("S"."TIME_ID"=TO_DATE(' 1999-07-01 00:00:00',
                                    'syyyy-mm-dd hh24:mi:ss'))
```

Bloom Filters

- Data structure (array)
  - Tell you rapidly whether an element is present in a set
  - Memory efficient
- Contains Bit Vector and uses Hash Functions
  - Modified during each set element addition
  - Used to check - element is in set ?
  - False positive possible (can return more rows then needed but do not lose any)
  - Good for large number of rows pre elimination
- Oracle can use it to:
  - reduce data communication between PX slaves (good for RAC)
  - join-filter in partition pruning
  - support result caches
  - filter data from storage server cells in Exadata

Parallel Plan

```
SELECT /*+ PARALLEL(4) */ t.fisc_yr_num, SUM(f.sold_amt) AS sold_amt
  FROM sh_sales_fct f INNER JOIN sh_time_dim t ON t.time_id  = f.time_id
  GROUP BY t.fisc_yr_num

| Id  | Operation                          | Name          |    TQ  |IN-OUT| PQ Distrib |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                   |               |        |      |            |
|   1 |  PX COORDINATOR                    |               |        |      |            |
|   2 |   PX SEND QC (RANDOM)              | :TQ10004      |  Q1,04 | P->S | QC (RAND)  |
|   3 |    HASH GROUP BY                   |               |  Q1,04 | PCWP |            |
|   4 |     PX RECEIVE                     |               |  Q1,04 | PCWP |            |
|   5 |      PX SEND HASH                  | :TQ10003      |  Q1,03 | P->P | HASH       |
|   6 |       HASH GROUP BY                |               |  Q1,03 | PCWP |            |
|   7 |        HASH JOIN                   |               |  Q1,03 | PCWP |            |
|   8 |         PX RECEIVE                 |               |  Q1,03 | PCWP |            |
|   9 |          PX SEND HASH              | :TQ10001      |  Q1,01 | P->P | HASH       |
|  10 |           VIEW                     | VW_GBC_5      |  Q1,01 | PCWP |            |
|  11 |            HASH GROUP BY           |               |  Q1,01 | PCWP |            |
|  12 |             PX RECEIVE             |               |  Q1,01 | PCWP |            |
|  13 |              PX SEND HASH          | :TQ10000      |  Q1,00 | P->P | HASH       |
|  14 |               HASH GROUP BY        |               |  Q1,00 | PCWP |            |
|  15 |                PX BLOCK ITERATOR   |               |  Q1,00 | PCWC |            |
|  16 |                 TABLE ACCESS FULL  | SH_SALES_FCT  |  Q1,00 | PCWP |            |
|  17 |         PX RECEIVE                 |               |  Q1,03 | PCWP |            |
|  18 |          PX SEND HASH              | :TQ10002      |  Q1,02 | P->P | HASH       |
|  19 |           PX BLOCK ITERATOR        |               |  Q1,02 | PCWC |            |
|  20 |            TABLE ACCESS FULL       | SH_TIME_DIM   |  Q1,02 | PCWP |            |
```

Parallel Plan

Row Distribution

- HASH
  - Hash function applied to value of the join column
  - Distribute to the workload on the corresponding hash value
- Round Robin
  - Randomly but evenly distributes the data among the consumers
- **Broadcast**
  - **The size of one of the result sets is small**
  - Sends a copy of the data to all consumers
- Range
  - Typically used for parallel sort operations
  - Individual parallel servers work on data ranges
  - QC doesn't sort just present the parallel server results in the correct order
- Partitioning Key Distribution – PART (KEY)
  - Assumes that the target table is partitioned
  - Partitions of the target tables are mapped to the parallel servers
  - Producers map each row to consumer based on partitioning column

Parallel Plan & PX

Tuning

```
Hint: STATEMENT_QUEUING
Opp.: NO_STATEMENT_QUEUING
```

- Validate if correct distribution method is used
  - E.g. BROADCAST only for small number of rows
- Check if parallel reduction or serialization is not used
  - for intermediate operations on large volume (e.g. serial PL/SQL fn)
- Find SQL candidates for tuning by comparing
  - SQL elapsed time is very far from DEGREE times smaller from
  - SQL database time
- Check if one SQL slaves are not locking each other
- Set DOP on large tables instead of using hints if possible
- Set `PARALLEL_DEGREE_POLICY=AUTO` on session
  - To enable DEGREE auto-tuning, queuing and in-memory PX

Parallel Plan & PX

Tuning

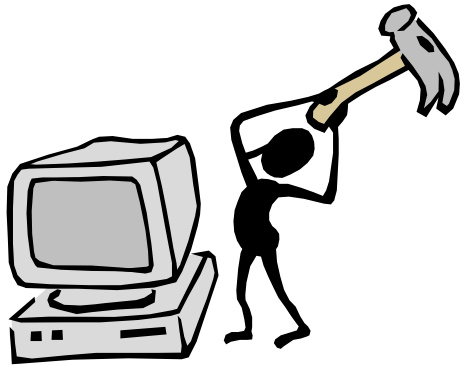
```
Hint: STATEMENT_QUEUING
Opp.: NO_STATEMENT_QUEUING
```

- Validate if correct distribution method is used
  - E.g. BROADCAST only for small number of rows
- Check if parallel reduction or serialization is not used
  - for intermediate operations on large volume (e.g. serial PL/SQL fn)
- Find SQL candidates for tuning by comparing
  - SQL elapsed time is very far from DEGREE times smaller from
  - SQL database time
- Check if one SQL slaves are not locking each other
- Set DOP on large tables instead of using hints if possible
- Set `PARALLEL_DEGREE_POLICY=AUTO` on session
  - To enable DEGREE auto-tuning, queuing and in-memory PX

Q & A

+ Quiz

Advanced Plan Operations
Workshop



- Check if partition pruning is working for SQL

```
SELECT /*+GATHER_PLAN_STATISTICS*/
    t.fisc_mth_num, SUM(f.sold_amt) AS sold_amt
  FROM sh_sales_fct f INNER JOIN sh_time_dim t
    ON t.time_id  = f.time_id
  WHERE t.fisc_yr_num = 2000 GROUP BY t.fisc_mth_num
```

- Try to use parallel(4) hint – variant 1
  and parallel with pq_distribute hints – variant 2
  - Compare SQL plan and execution time

Topic Agenda

Plan Transformations

- Introduction
- Star Transformation
- OR expansion
- Simple / Complex(11g) View Merging
- Filter Push Down (Non-mergable View)
- Subquery Unnesting
- Join Predicate Push Down – 11g
- Cost Based Transformation – 11g
  - Join Factorization & Elimination - 11g
  - COUNT(col) to COUN(*) – 11g
  - Transitive Closure
  - Table Elimination (PK-FK, Outer Join)
  - Predicate Move Around
  - Select, Project, Join
  - Order By Ellimination
  - Set Join Conversion

Plan Transformation

Introduction

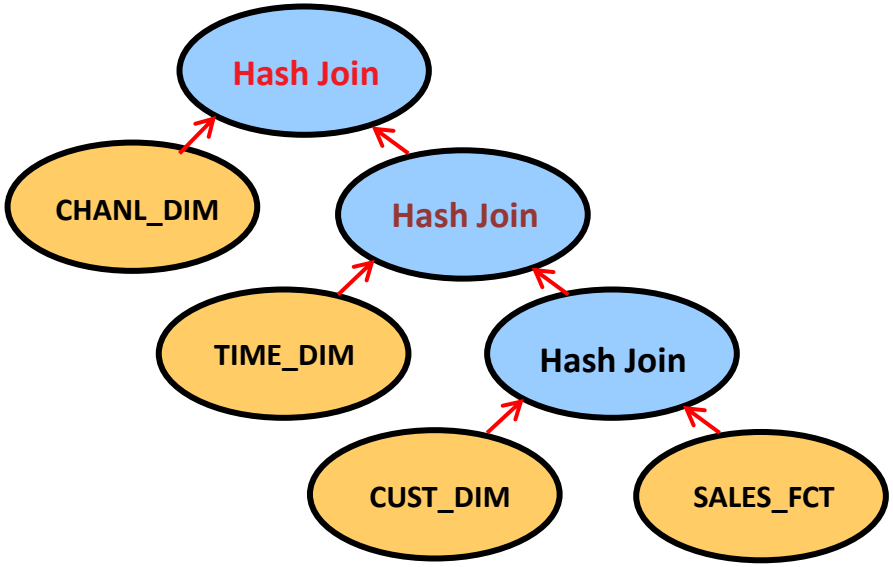
- Oracle can automatically rewrite SQL statements
- e.g. subquery can be modified to join
- New statement performance should be better
- Usage decision is based on estimated cost
- Results are the same
- Very hard to correlate plan operations
  with original SQL text parts

Star Join

Hint: NO_STAR_TRANSFORMATION

No transformation

```
SELECT ch.chanl_class_name, c.cust_city_name, t.cal_qrt_code,
    SUM(f.sold_amt) sales_amt
  FROM sales_fct f, time_dim t, cust_dim c, chanl_dim ch
  WHERE f.time_id = t.time_id
    AND f.cust_id = c.cust_id
    AND f.chanl_id = ch.chan_id
    AND c.cust_st_pro_code = 'CA'
    AND ch.chanl_name IN ('Internet','Catalog')
    AND t.cal_qrt_code IN ('1999-Q1','1999-Q2')
  GROUP BY ch.chanl_class_name, c.cust_city_name, t.cal_qrt_code;
```




```
-----------------------------------------
| Id  | Operation              | Name     |
-----------------------------------------
|   0 | SELECT STATEMENT       |          |
|   1 |  HASH GROUP BY         |          |
|*  2 |   HASH JOIN            |          |
|*  3 |    TABLE ACCESS FULL   | CHANL_DIM |
|*  4 |    HASH JOIN           |          |
|*  5 |     TABLE ACCESS FULL  | TIME_DIM  |
|*  6 |     HASH JOIN          |          |
|*  7 |      TABLE ACCESS FULL| CUST_DIM  |
|   8 |      TABLE ACCESS FULL| SALES_FCT |
-----------------------------------------
```

Plan Transformations

Star Transformation

Overview

```
Hint: STAR_TRANSFORMATION
      FACT(tab)   NO_FACT(tab)
```

- Carried out in two phases:
  - First, identify interesting fact rows using bitmap indexes based on dimensional filters (phase 1). 
  - Join them to the dimension tables (phase 2). 
- Requirements
  - Create bitmap indexes on fact tables foreign keys.
  - Set STAR_TRANSFORMATION_ENABLED to TRUE
  - At least two dimensions and one fact table
  - Gather statistics on all corresponding objects.

Star Transformation

`Hint: STAR_TRANSFORMATION`

Restrictions

- Queries are not transformed if containing:
  - bind variables
  - references to remote fact tables
  - antijoined tables
  - references to unmerged views
  - NO_STAR_TRANSFORMATION hint used

Star Transformation

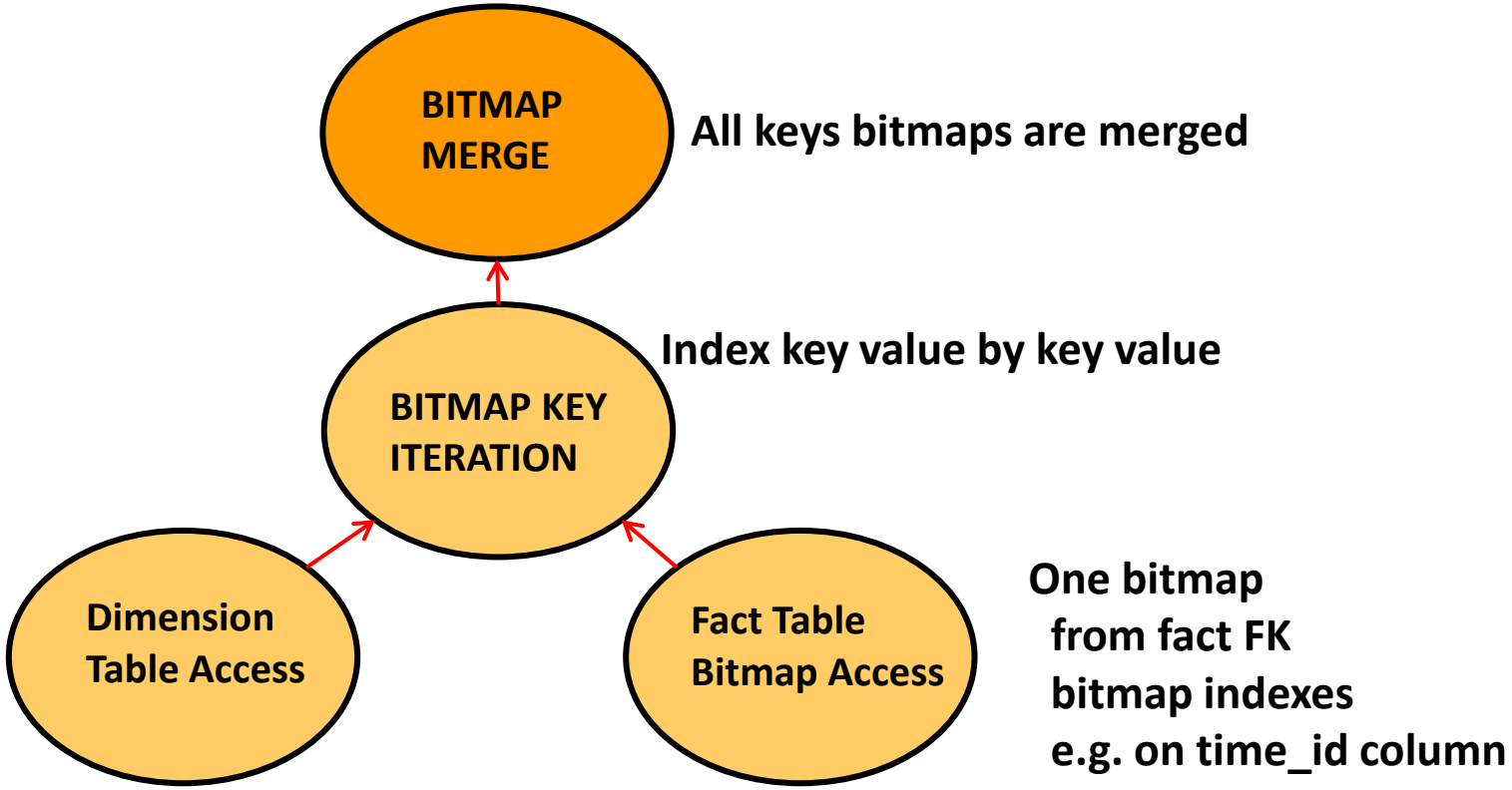
Phase 1

```
SELECT f.sold_amt, f.time_id, f.cust_id, f.chanl_id
  FROM sales_fct f
  WHERE time_id IN (SELECT time_id
                      FROM time_dim
                      WHERE cal_qrt_code IN
                        ('1999-Q1','1999-Q2'))
    AND cust_id IN (SELECT cust_id
                      FROM cust_dim
                      WHERE cust_st_prov = 'CA')
    AND chanl_id IN (SELECT chanl_id
                       FROM chanl_dim
                       WHERE chanl_name IN
                         ('Internet','Catalog'));
```

Star Transformation

Phase 1

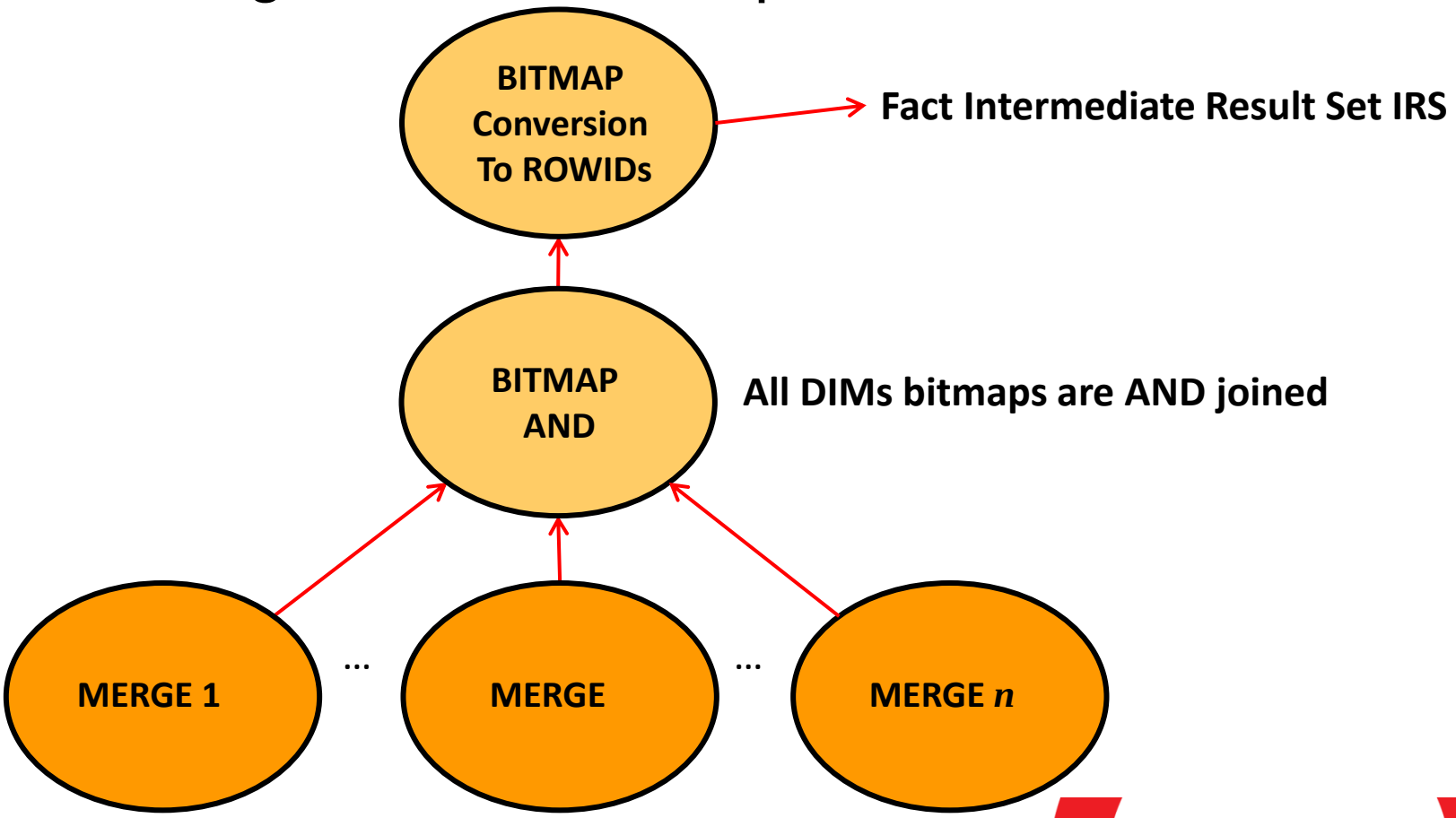
- Retrieving fact rows bitmap - one dimension



Star Transformation

Phase 1

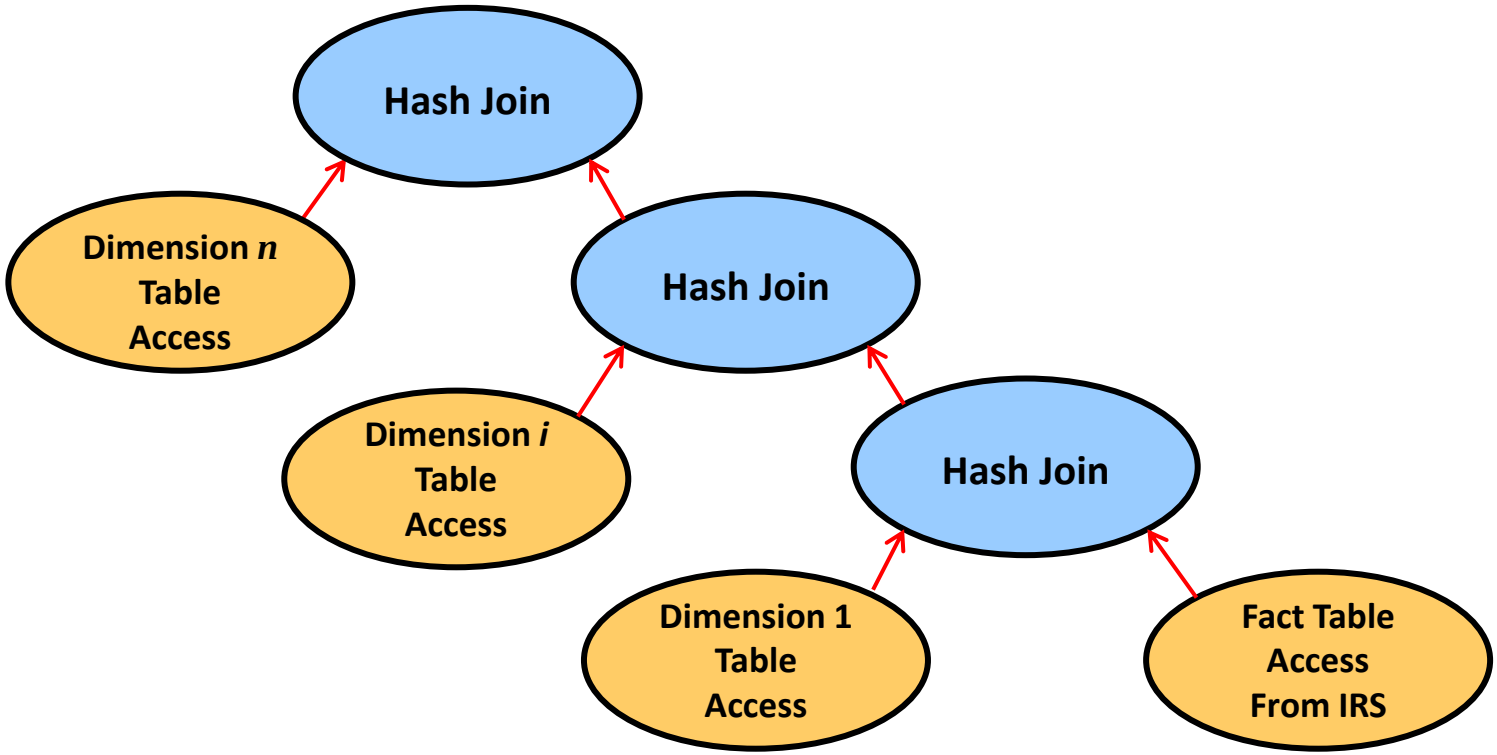
- Retrieving fact rows bitmap - all dimensions



Star Transformation

Phase 2

- Join selected fact rows with dimension data



Star Transformation

Plan

```
SORT GROUP BY
 HASH JOIN
    HASH JOIN
      TABLE ACCESS BY INDEX ROWID SALES_FCT
       BITMAP CONVERSION TO ROWIDS
        BITMAP AND
         BITMAP MERGE
          BITMAP KEY ITERATION
           BUFFER SORT
            TABLE ACCESS FULL CHANL_DIM
           BITMAP INDEX RANGE SCAN SALES_CHANL_BX
         BITMAP MERGE
          BITMAP KEY ITERATION
           BUFFER SORT
            TABLE ACCESS FULL TIME_DIM
           BITMAP INDEX RANGE SCAN SALES_TIME_BX

         ...
      TABLE ACCESS FULL CHANL_DIM
    TABLE ACCESS FULL TIME_DIM
```

Star Transformation

Large Dimension Table

- Dimension tables are accessed twice
  - once for each phase.
  - can performance issue for big dimension tables (low selectivity)
- Oracle might decide to use temporary table
  - instead of accessing the same dimension table twice

```
LOAD AS SELECT          SYS_TEMP_0FD9D6720_BEBDC
  TABLE ACCESS FULL     CUSTOMERS
…
  filter("C"."CUST_STATE_PROVINCE"='CA')
```

  - Can be disabled on session

    ALTER SESSION SET star_transformation_enabled = **temp_disable**;

OR Expansion
CONCATENATION

```
Hint: USE_CONCAT
Opp.: NO_EXPAND
```

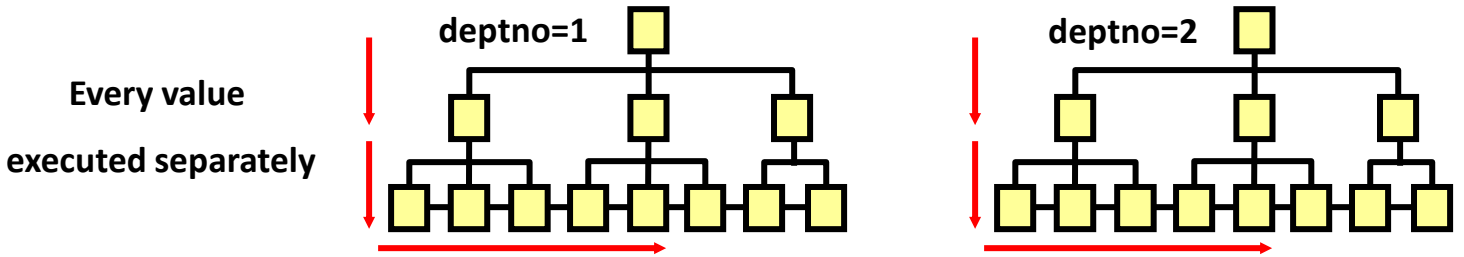
- Concatenates the rows returned by two or more row sets.
- Works like UNION ALL but removes duplicate rows.
  - So appends a negation of the previous components using LNNVL function
  - filter (LNNVL(SAL=2)) returns all rows for which SAL != 2 or SAL is NULL.

```
SELECT /*+ USE_CONCAT */ * FROM emp WHERE deptno = 1 OR sal = 2;

-------------------------------------------------------------------------
|   0 | SELECT STATEMENT                |           |     8 |   696 |
|   1 |  CONCATENATION                  |           |       |       |
|   2 |   TABLE ACCESS BY INDEX ROWID| EMP        |     4 |   348 |
|   3 |    INDEX RANGE SCAN             | I_SAL     |     2 |       |
|   4 |   TABLE ACCESS BY INDEX ROWID| EMP        |     4 |   348 |
|   5 |    INDEX RANGE SCAN             | I_DEPTNO  |     2 |       |
-------------------------------------------------------------------------
   3 - access("SAL"=2)
   4 - filter(LNNVL("SAL"=2))
   5 - access("DEPTNO"=1)
```

Preventing OR Expansion

INLIST ITERATOR

- When many values are used inside IN clause or between OR operators
- Works as a FOR LOOP statement in PL/SQL and similar to NESTED LOOPS
- Alternative for UNION ALL of each value or a FILTER against all the rows
- Optimizer use it when selective index exists for that column rather than CONCATENATION or UNION ALL because it is more efficient.




```
SELECT * FROM emp WHERE empno IN (7876, 7900, 7902);
```

OPERATION	OBJECT_NAME	OPTIONS	COST	CARDINALI...
SELECT STATEMENT			2	3
INLIST ITERATOR				
TABLE ACCESS	EMP	BY INDEX RO...	2	3
INDEX	PK_EMP	UNIQUE SCAN	1	3
Access Predicates				
OR				
EMPNO=7876				
EMPNO=7900				
EMPNO=7902				

View Merge

Simple

```
Hint: MERGE[(vw)]
Opp.: NO_MERGE[(vw)]
```

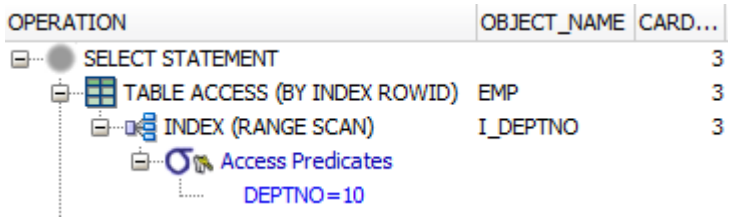
- View merging
  - View SQL text and query using view is transformed to one query (view disappear)
  - New query have one more effective plan then two separate plans
- Simple view merging is possible if view not contain

  GROUP BY, Agg. Functions, DISTINCT, Outer join, MODEL, CONNECT BY, Set operations

  NO_MERGE hint
- Simple merging is still possible even view contain
  - view appears on the right side of a semijoin or antijoin.
  - view contains subqueries in the SELECT list.
  - outer query block contains PL/SQL functions.
  - view participates in an outer join,

View Merge

Simple



- Created view can be merged into query.

```
CREATE VIEW vw AS SELECT /*+ MERGE */ deptno, sal FROM emp;
SELECT * FROM vw WHERE deptno = 10;
```

- Inline view also can be merged

```
SELECT /*+ NO_MERGE(i_vw) or MERGE(…) */ e.first_name, locs_v.street_address FROM employees e,
     (SELECT d.department_id, d.department_name, l.street_address
        FROM departments d, locations l WHERE d.location_id = l.location_id) i_vw
  WHERE dept_locs_v.department_id = e.department_id
  AND e.last_name = 'Smith';
```

- Not merged – VIEW Operation

```
-------------------------------------------------------------
| Id  | Operation                     | Name         | Cost (
-------------------------------------------------------------
|   0 | SELECT STATEMENT              |              |     7
|*  1 |  HASH JOIN                    |              |     7
|   2 |   TABLE ACCESS BY INDEX ROWID| EMPLOYEES    |     2
|*  3 |    INDEX RANGE SCAN           | EMP_NAME_IX  |     1
|   4 |   VIEW                        |              |     5
|*  5 |    HASH JOIN                  |              |     5
|   6 |     TABLE ACCESS FULL         | LOCATIONS    |     2
|   7 |     TABLE ACCESS FULL         | DEPARTMENTS  |     2
-------------------------------------------------------------
```

- Merged view – 3 tables join

```
-----------------------------------------------------------------------
| Id  | Operation                     | Name         | Cost (%CPU)|
-----------------------------------------------------------------------
|   0 | SELECT STATEMENT              |              |     4   (0)|
|   1 |  NESTED LOOPS                 |              |            |
|   2 |   NESTED LOOPS                |              |     4   (0)|
|   3 |    NESTED LOOPS               |              |     3   (0)|
|   4 |     TABLE ACCESS BY INDEX ROWID| EMPLOYEES    |     2   (0)|
|*  5 |      INDEX RANGE SCAN         | EMP_NAME_IX  |     1   (0)|
|   6 |     TABLE ACCESS BY INDEX ROWID| DEPARTMENTS  |     1   (0)|
|*  7 |      INDEX UNIQUE SCAN        | DEPT_ID_PK   |     0   (0)|
|*  8 |    INDEX UNIQUE SCAN          | LOC_ID_PK    |     0   (0)|
|   9 |   TABLE ACCESS BY INDEX ROWID | LOCATIONS    |     1   (0)|
-----------------------------------------------------------------------
```

Plan Transformations

View Merge

Complex

- Optimizer merges complex views
  - containing GROUP BY and DISTINCT
- Unable to perform when:
  - **outer query tables do not have a ROWID or unique column**
  - view appears in a CONNECT BY query block
  - view contains GROUPING SETS, ROLLUP, or PIVOT clauses
  - view or outer query block contains the MODEL clause

```
SELECT c.cust_id, c.cust_first_name, c.cust_email
  FROM cust_dim c, prod_dim p,
       (SELECT DISTINCT s.cust_id, s.prod_id
          FROM sales_fct f) cust_prod_ivw
  WHERE c.cntry_id = 52790
    AND c.cust_id = cust_prod_v.cust_id
    AND cust_prod_ivw.prod_id = p.prod_id
    AND p.prod_name = 'T3 Faux Fur-Trimmed
Sweater';
```

```
-----------------------------------------
| Id  | Operation             | Name      |
-----------------------------------------
|   0 | SELECT STATEMENT      |           |
|   1 |  VIEW                 | VM_NWVW_1 |
|   2 |   HASH UNIQUE         |           |
|*  3 |    HASH JOIN          |           |
|*  4 |     HASH JOIN         |           |
|*  5 |      TABLE ACCESS FULL| PROD_DIM  |
|   6 |      TABLE ACCESS FULL| SALES_FCT |
|*  7 |     TABLE ACCESS FULL | CUST_DIM  |
-----------------------------------------
```

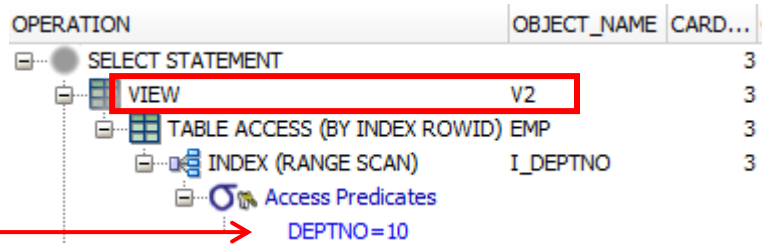
Not Mergable View

Filter Push Down

```
CREATE VIEW v2 AS SELECT /*+ NO_MERGE */ deptno, sal FROM emp;
```

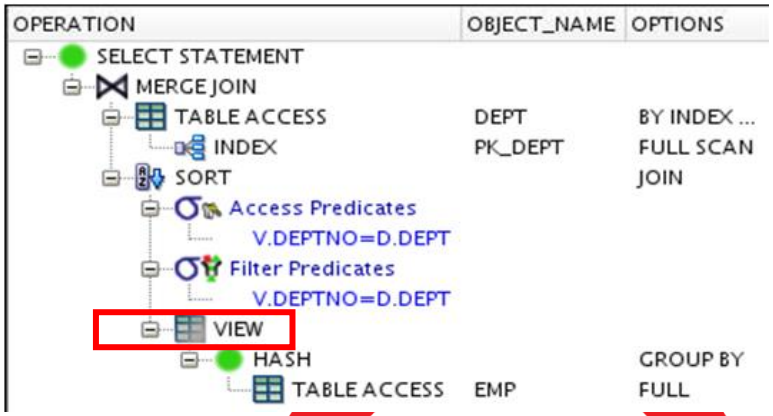
```
SELECT * FROM v2 WHERE deptno = 10;
```

- If not merged appear as VIEW operator
  - executed separately
  - all rows from the view are returned
  - but predicates can be pushed or pulled



Nonmergeable inline view

```
SELECT v.*, d.dname
  FROM (SELECT deptno, SUM(sal) sum_sal
          FROM emp GROUP BY deptno) v,
       dept d
  WHERE v.deptno = d.deptno;
```



Why this inline view is not mergable?

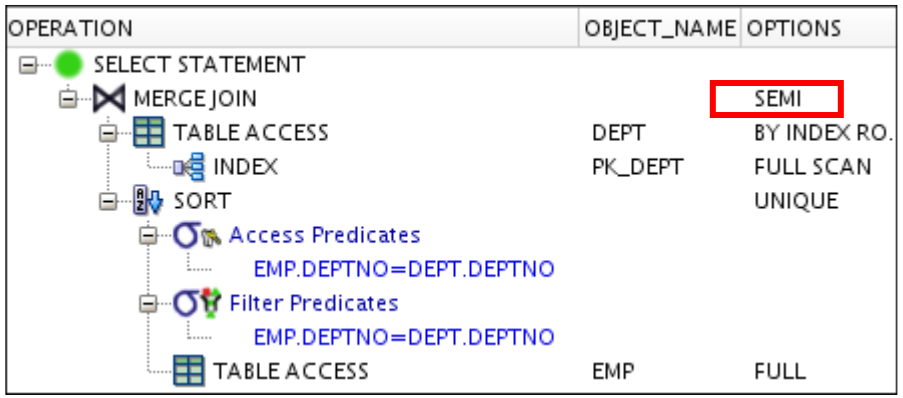
Subquery Unnesting

Semi-Join

```
Hints: UNNEST MERGE_SJ
       HASH_SJ   NL_SJ
 Opp.: NO_UNNEST
```

- Transforming an EXISTS or IN subquery into a join
- Look only for the first match - return rows without duplicating rows
  - even subquery or normal join returns duplicates
- For each DEPT record, only the first matching EMP record is returned
- Merge, nested loops or hash semi joins can be used.

```
SELECT deptno, dname
  FROM dept
  WHERE EXISTS (SELECT 1 FROM emp WHERE emp.deptno = dept.deptno);
```



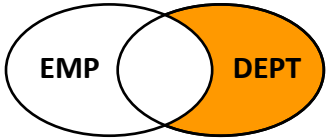
Subquery Unnesting

Anti-Join

```
Hints: UNNEST MERGE_AJ
       HASH_AJ   NL_AJ
 Opp.: NO_UNNEST
```

- Transforming an NOT EXISTS or NOT IN subquery into a join
- Similar to Semi-Join but
  - Reverse of what would have been returned by a join

```
SELECT deptno, dname FROM dept
  WHERE deptno IS NOT NULL
    AND deptno NOT IN
 (SELECT /*+ HASH_AJ */ deptno
    FROM emp WHERE deptno IS NOT NULL);
```



OPERATION	OBJECT_NAME	OPTIONS
SELECT STATEMENT		
HASH JOIN		ANTI
Access Predicates		
DEPTNO=DEPTNO		
TABLE ACCESS	DEPT	FULL
TABLE ACCESS	EMP	FULL
Filter Predicates		
DEPTNO IS NOT NULL		

Cost Based Transform

Group by Placement

```
Hints: PLACE_GROUP_BY(@qb)
Opp.: NO_PLACE_GROUP_BY(@qb)
```

- Aggregate table before joining if volume from this table is large

```
SELECT /*+ qb_name(main)
           no_place_group_by(@main)
       */
  t.fisc_yr_num, SUM(f.sold_amt)
  FROM sh_sales_fct f
    INNER JOIN sh_time_dim t
       ON t.time_id  = f.time_id
    GROUP BY t.fisc_yr_num

-----------------------------------------------------
| Id  | Operation                     | Name         |
-----------------------------------------------------
|   0 | SELECT STATEMENT              |              |
|   1 |  HASH GROUP BY                |              |
|   2 |   HASH JOIN                   |              |
|   3 |    PART JOIN FILTER CREATE    | :BF0000      |
|   4 |     TABLE ACCESS FULL         | SH_TIME_DIM  |
|   5 |    PARTITION RANGE JOIN-FILTER|              |
|   6 |     TABLE ACCESS FULL         | SH_SALES_FCT |
-----------------------------------------------------
```

```
/*+ qb_name(main)
    place_group_by(@main)
*/
-----------------------------------------------
| Id  | Operation               | Name        |
-----------------------------------------------
|   0 | SELECT STATEMENT        |             |
|   1 |  HASH GROUP BY          |             |
|   2 |   HASH JOIN             |             |
|   3 |    VIEW                 | VW_GBC_5    |
|   4 |     HASH GROUP BY       |             |
|   5 |      PARTITION RANGE ALL|             |
|   6 |       TABLE ACCESS FULL | SH_SALES_FCT |
|   7 |    TABLE ACCESS FULL    | SH_TIME_DIM |
-----------------------------------------------

Query Block Name / Object Alias
-----------------------------------------------
   1 - SEL$B359EB6E
   3 - SEL$8FF313FB / VW_GBC_5@SEL$CBB12FA2
   4 - SEL$8FF313FB
   6 - SEL$8FF313FB / F@SEL$1
   7 - SEL$B359EB6E / T@SEL$1
```

Plan Transformations

QB_NAME hint

```
SELECT /*+ QB_NAME(QB_MAIN) */
    prod_name, SUM(sold_amt) AS chanl2_amt, chanl3_amt
  FROM sh_sales_fct f2
  JOIN sh_prod_dim p ON f2.prod_id = p.prod_id
  JOIN (SELECT /*+ QB_NAME(QB_SUB) */
           prod_id, SUM(sold_amt) AS chanl3_amt
         FROM sh_sales_fct WHERE chanl_id = 3
         GROUP BY prod_id) f3
    ON f2.prod_id = f3.prod_id WHERE chanl_id = 2
  GROUP BY p.prod_name, chanl3_amt
```

```
dbms_xplan.display_cursor ('976cmcndpj9pd', format => 'basic +alias +outline'));
```

```
| Id  | Operation                    | Name         |
-----------------------------------------------------
|   0 | SELECT STATEMENT             |              |
|   1 |  HASH GROUP BY               |              |
|   2 |   HASH JOIN                  |              |
|   3 |    VIEW                      | VW_GBC_9     |
|   4 |     HASH GROUP BY            |              |
|   5 |      HASH JOIN               |              |
|   6 |       VIEW                   |              |
|   7 |        HASH GROUP BY         |              |
|   8 |         PARTITION RANGE ALL  |              |
|   9 |          TABLE ACCESS FULL   | SH_SALES_FCT |
|  10 |       PARTITION RANGE ALL    |              |
|  11 |        TABLE ACCESS FULL     | SH_SALES_FCT |
|  12 |    TABLE ACCESS FULL         | SH_PROD_DIM  |
```

```
Query Block Name / Object Alias
-------------------------------------------
 1 - SEL$2F9F79D7
 3 - SEL$C6F03CC3 / VW_GBC_9@SEL$CA133882
 4 - SEL$C6F03CC3
 6 - QB_SUB       / F3@SEL$2
 7 - QB_SUB
 9 - QB_SUB       / SH_SALES_FCT@QB_SUB
11 - SEL$C6F03CC3 / F2@SEL$1
12 - SEL$2F9F79D7 / P@SEL$1

Outline Data
-------------
   /*+
      BEGIN_OUTLINE_DATA
      ...
      MERGE(@"SEL$58A6D7F6")
      OUTLINE(@"QB_MAIN")
      OUTLINE(@"SEL$58A6D7F6")
      MERGE(@"SEL$1")
      OUTLINE(@"SEL$2")
      OUTLINE(@"SEL$1")
      FULL(@"SEL$2F9F79D7" "P"@"SEL$1")
      LEADING(@"SEL$2F9F79D7"
      ...
      END_OUTLINE_DATA
  */
```

Join Predicate Pushdown

```
Hints: PUSH_PRED(vw)
Opp.: NO_PUSH_PRED(vw)
```

- View to be joined with nested-loop and pushed filter from outside

```
SELECT /*+ push_pred(ivw) */
    c.cust_last_name, c.cust_city_name
  FROM sh_cust_dim c,
     (SELECT DISTINCT f2.cust_id
        FROM sh_sales_fct f2
        WHERE f2.chanl_id = 2) ivw
  WHERE c.cust_st_prov_name = 'CA'
    AND c.cust_id = ivw.cust_id
```

OPERATION	OBJECT_NAME	CARDINALITY
SELECT STATEMENT		402
NESTED LOOPS (SEMI)		402
TABLE ACCESS (FULL)	SH_CUST_DIM	3162
Filter Predicates		
C.CUST_ST_PROV_NAME='CA'		
PARTITION RANGE (ALL)		5
VIEW PUSHED PREDICATE		5
TABLE ACCESS (BY LOCAL INDEX ROWID)	SH_SALES_FCT	37
Filter Predicates		
F2.CHANL_ID=2		
BITMAP CONVERSION (TO ROWIDS)		
BITMAP INDEX (SINGLE VALUE)	SH_SALES_C...	
Access Predicates		
F2.CUST_ID=C.CUST_ID		

```
/*+ no_query_transform */
```

OPERATION	OBJECT_NAME	CARDINALITY
SELECT STATEMENT		3162
HASH JOIN		3162
Access Predicates		
C.CUST_ID=IVW.CUST_ID		
JOIN FILTER (CREATE)	:BF0000	3162
TABLE ACCESS (FULL)	SH_CUST_DIM	3162
Filter Predicates		
C.CUST_ST_PROV_NAME='CA'		
VIEW		7059
HASH (UNIQUE)		7059
JOIN FILTER (USE)	:BF0000	260254
PARTITION RANGE (ALL)		260254
TABLE ACCESS (FULL)	SH_SALES_FCT	260254
Filter Predicates		
AND		
F2.CHANL_ID=2		
SYS_OP_BLOOM_FILTER		

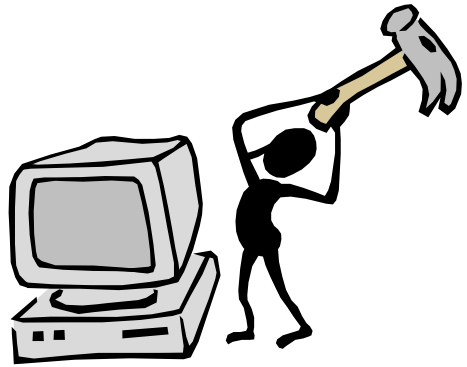
Bloom Filter

Plan Transformations

Q & A

+ Quiz

Plan Transformations
Workshop



- Check execution time and plan for star query case
  - Compare second variant by uncomment second –SELECT
- Compare SQl plans same way for next few cases

Topic Agenda

SQL Plan Management

- Overview
- Using SQL Profiles
- SQL Plan Baselines
- Documented 11.2 Hints

Plan Management

Overview

- Use hints embedded in SQL source code only if
  - Each statement execution uses different sql_id
  - Examples: dynamic SQL used, reports with substitution variables
  - To avoid this you can use
    * bind variables in source code
    * CURSOR_SHARING parameter
- Use SQL Profile where you need only SQL plan correction
- Use SQL Plan Baseline if
  - Need execution plan stability
  - Many good validated plans exist for SQL
  - To avoid surprise with new wrong plan
- SQL Profile and SQL Plan Baseline
  - Requires access from DBA (to select_catalog_role, ADMINISTER SQL MANAGEMENT OBJECT, dbms_sqltune. ADVISOR)
  - Need same sql_id for all executions of one SQL
  - **Can apply hints without source code modification**

SQL Profile - Accepted - SQL Tuning Advisor - Example

```
DECLARE
 l_task_name VARCHAR2(30);
 l_sql_ftext   VARCHAR2(32000) :=
  'SELECT /*+ ORDERED USE_NL(c) FULL(c) FULL(s) */ COUNT(*)
     FROM sh_sales_fct s, sh_cust_dim c
     WHERE c.cust_id = s.cust_id
     AND c.cust_first_name = :cname
     AND time_id > :tid
     ORDER BY time_id';
 l_binds sql_binds := sql_binds(
   anydata.ConvertVARCHAR2('Dina'),
   anydata.ConvertDATE(SYSDATE-100000)
   );
BEGIN
  l_task_name := dbms_sqltune.create_tuning_task(
         sql_text    => l_sql_ftext,
         bind_list   => l_binds,
         user_name   => 'SH',
         scope       => 'COMPREHENSIVE',
         time_limit  => 60,
         task_name   => 'SH SQL-tune task 1');
END;
/
```

```
SELECT task_id, task_name
  FROM dba_advisor_log
  WHERE owner = 'SH';
```

TASK_ID	TASK_NAME
5744	SH SQL-tune task 1

```
FINDINGS SECTION (2 findings)
1- SQL Profile Finding
 Recommendation (estimated benefit: 99.87%)
                          Original Plan  With SQL Profile  % Improved
                          -------------  ----------------  ----------
  Completion Status:            PARTIAL          COMPLETE
  Elapsed Time(us):            14370351           1938775       86.5 %
  CPU Time(us):                14343750            171875       98.8 %
  User I/O Time(us):                  0            175859
  Buffer Gets:                  4778546              6159      99.87 %

2- Index Finding
 Recommendation (estimated benefit: 78.42%)
  create index … on SH_SALES_FCT("CUST_ID");
```

```
dbms_sqltune.set_tuning_task_parameter (task_name => 'SH SQL-tune task 1', parameter => 'TIME_LIMIT', value => 300)
dbms_sqltune.execute_tuning_task(task_name => 'SH SQL-tune task 1')
SELECT dbms_sqltune.report_tuning_task('SH SQL-tune task 1') FROM dual;
dbms_sqltune.accept_sql_profile (task_name => 'SH SQL-tune task 1', task_owner => 'SH', replace => TRUE)
dbms_sqltune.drop_tuning_task (task_name => 'SH SQL-tune task 1')
SELECT name, sql_text, category, status FROM dba_sql_profiles;
```

NAME	SQL_TEXT	CATEGORY	STATUS
SYS_SQLPROF_015213a88b700000	SELECT /*+ ORDERED USE_NL(c)...	DEFAULT	ENABLED

Plan Management

SQL Profile - Create Manually - Example

```
set serveroutput on
exec profile_test('SUM(sal)', 10);

SELECT sql_id, plan_hash_value, sql_text
 FROM v$sqlarea WHERE upper(sql_text) LIKE '%DEPTNO = :%'
                  AND sql_text NOT LIKE '%v$%';
```

SQL_ID	PLAN_HASH_VALUE	CHILD_NUMBER	SQL_TEXT
b12dj8f8ftmdk	1364575507	0	SELECT SUM(sal) FROM emp where deptno = :deptno

```
SELECT * FROM TABLE(dbms_xplan.display_cursor ('b12dj8f8ftmdk',0, format => 'basic +alias +outline')) ;
```

```
-----------------------------------------------
| Id  | Operation                     | Name     |
-----------------------------------------------
|   0 | SELECT STATEMENT              |          |
|   1 |  SORT AGGREGATE               |          |
|   2 |   TABLE ACCESS BY INDEX ROWID| EMP      |
|   3 |    INDEX RANGE SCAN           | I_DEPTNO |
-----------------------------------------------
```

```
-------------------------------------
| Id  | Operation           | Name  |
-------------------------------------
|   0 | SELECT STATEMENT    |       |
|   1 |  SORT AGGREGATE     |       |
|   2 |   TABLE ACCESS FULL| EMP   |
-------------------------------------
```

SQL_ID	PLAN_HASH_VALUE	CHILD_NUMBER
b12dj8f8ftmdk	2083865914	0

```
DECLARE
  in_hints VARCHAR2(100)
     := 'FULL(@"SEL$1" "EMP"@"SEL$1")';
  in_sql_id VARCHAR2(100) := 'b12dj8f8ftmdk';
  l_sql_ftext CLOB;
BEGIN
  SELECT SQL_FULLTEXT INTO l_sql_ftext
    FROM V$SQLAREA
    WHERE SQL_ID = in_sql_id AND ROWNUM <= 1;
  DBMS_SQLTUNE.IMPORT_SQL_PROFILE(
    SQL_TEXT => l_sql_ftext,
    PROFILE => SQLPROF_ATTR(in_hints),
    NAME => 'PROFILE_'||in_sql_id,
    REPLACE => TRUE,
    FORCE_MATCH => TRUE
  );
END;
/
```

Plan Management

SQL Profile

Management

- Viewing

  SELECT name, sql_text, category, status FROM dba_sql_profiles;

NAME	SQL_TEXT	CATEGORY	STATUS
PROFILE_b12dj8f8ftmdk	SELECT SUM(sal) FROM emp where deptno = :deptno	DEFAULT	ENABLED

- Dropping

  exec dbms_sqltune.drop_sql_profile('PROFILE_b12dj8f8ftmdk');

- Edit in stage table

  exec dbms_sqltune.**create_stgtab_sqlprof**('my_sql_profiles_stage');

  exec dbms_sqltune**.pack_stgtab_sqlprof**

  ('PROFILE_b12dj8f8ftmdk',staging_table_name => 'my_sql_profiles_stage')

  select obj_name, comp_data from my_sql_profiles_stage;

OBJ_NAME	COMP_DATA
PROFILE_b12dj8f8ftmdk	<outline_data><hint><![CDATA[FULL(@"SEL$1" "EMP"")]]></hint></outline_data>

  Now xml column comp_data can be updated.

  exec dbms_sqltune.**unpack_stgtab_sqlprof**

  (replace => TRUE, staging_table_name => 'my_sql_profiles_stage');

SQL Plan Baseline

Overview

- This is optional set of one SQL plans
  - Can be create, captured, exported, imported
- Optimizer can choose only from baseline plans in if
  - parameter `OPTIMIZER_USE_SQL_PLAN_BASELINES` must be TRUE (on session)
  - baseline exists with at least one enabled plan
- Used to guarantee known optimizer behavior
- Examples of usage:
  - changing database version,
  - code migration DEV -> QA -> UAT -> PROD
  - problems with deep varying statistics and cardinality
- If SQL profile is accepted then is added to baseline
- (Not like SQL profile) SQL Plan Baseline consists of outlines
  - Outline is set off all hints needed to reproduce particular plan

SQL Plan Baseline

- Create

```
DROP INDEX i_deptno;
exec profile_test('SUM(sal)', 10);

SELECT sql_id, plan_hash_value, operation, options sql_text
 FROM v$sql s JOIN v$sql_plan p using(sql_id,plan_hash_value)
 WHERE upper(sql_text) LIKE '%DEPTNO = :%'
   AND sql_text NOT LIKE '%v$%' AND object_name = 'EMP';
```

SQL_ID	PLAN_HASH_VALUE	OPERATION	OPTIONS	SQL_TEXT
b12dj8f8ftmdk	2083865914	TABLE ACCESS	FULL	SELECT SUM(sal) FROM emp where deptno = :deptno

```
BEGIN
  dbms_output.put_line('Loaded: ' || dbms_spm.load_plans_from_cursor_cache(
      sql_id => 'b12dj8f8ftmdk', plan_hash_value => '2083865914'));
END;
/
```

```
SELECT sql_handle, origin, enabled,
       accepted, fixed, autopurge
  FROM dba_sql_plan_baselines;
```

SQL_HANDLE	ORIGIN	ENABLED	ACCEPTED	FIXED	AUTOPURGE
SYS_SQL_f437939c5404f8eb	MANUAL-LOAD	YES	YES	NO	YES

- Use

```
CREATE INDEX i_deptno ON emp(deptno);
ALTER SESSION SET optimizer_use_sql_plan_baselines = TRUE;
```

SQL_ID	PLAN_HASH_VALUE	OPERATION	OPTIONS	SQL_TEXT
b12dj8f8ftmdk	2083865914	TABLE ACCESS	FULL	SELECT SUM(sal) FROM emp where deptno = :deptno

```
ALTER SESSION SET optimizer_use_sql_plan_baselines = FALSE;
```

SQL_ID	PLAN_HASH_VALUE	OPERATION	OPTIONS	SQL_TEXT
b12dj8f8ftmdk	2083865914	TABLE ACCESS	FULL	SELECT SUM(sal) FROM emp where deptno = :deptno
b12dj8f8ftmdk	1364575507	TABLE ACCESS	BY INDEX ROWID	SELECT SUM(sal) FROM emp where deptno = :deptno

Documented 11.2 Hints

Access Paths

FULL
CLUSTER
HASH
INDEX and NO_INDEX
INDEX_ASC and INDEX_DESC
INDEX_COMBINE and INDEX_JOIN
INDEX_JOIN
INDEX_FFS and NO_INDEX_FFS
INDEX_SS and NO_INDEX_SS
INDEX_SS_ASC and INDEX_SS_DESC

Join Orders

LEADING
ORDERED

Join Operations

USE_NL and NO_USE_NL
USE_NL_WITH_INDEX
USE_MERGE and NO_USE_MERGE
USE_HASH and NO_USE_HASH
NO_USE_HASH

Application Upgrade

CHANGE_DUPKEY_ERROR_INDEX
IGNORE_ROW_ON_DUPKEY_INDEX
RETRY_ON_ROW_CHANGE

Parallel Execution

PARALLEL and NO_PARALLEL
PARALLEL_INDEX and NO_PARALLEL_INDEX
PQ_DISTRIBUTE

Query Transformations

NO_QUERY_TRANSFORMATION
USE_CONCAT
NO_EXPAND
REWRITE and NO_REWRITE
MERGE and NO_MERGE
STAR_TRANSFORMATION and NO_STAR_TRANSFORMATION
FACT and NO_FACT
UNNEST and NO_UNNEST

Additional Hints

APPEND, APPEND_VALUES, and NOAPPEND
CACHE and NOCACHE
PUSH_PRED and NO_PUSH_PRED
PUSH_SUBQ and NO_PUSH_SUBQ
QB_NAME
CURSOR_SHARING_EXACT
DRIVING_SITE
DYNAMIC_SAMPLING
MODEL_MIN_ANALYSIS

Undocumented 11.2 Hints

CARDINALITY

OPT_ESTIMATE

SWAP_JOIN_INPUTS
NO_SWAP_JOIN_INPUTS

CONNECT_BY_COST_BASED
NO_CONNECT_BY_COST_BASED

NATIVE_FULL_OUTER_JOIN
NO_NATIVE_FULL_OUTER_JOIN

USE_HASH_AGGREGATION
NO_USE_HASH_AGGREGATION

…

Check v$sql_hint

Nested Views Hints

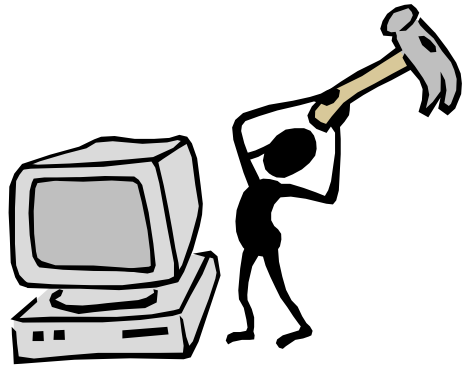
Example

```
SELECT /*+ FULL(vw1.vw2.f1) USE_HASH(vw1.vw2, vw1.vw3) */
    prod_name, SUM(sold_amt), chanl3_amt
  FROM sh_sales_fct f2
  JOIN sh_prod_dim p ON f2.prod_id = p.prod_id
  JOIN (SELECT prod_id, SUM(sold_amt) AS chanl3_amt
          FROM (SELECT prod_id, SUM(sold_amt) AS chanl3_amt
                  FROM sh_sales_fct f1 WHERE chanl_id = 3
                  WHERE cal_yr_num > 2000
                  GROUP BY prod_id, chanl_id) vw2
          JOIN (SELECT prod_id, SUM(sold_amt) AS chanl3_amt
                  FROM sh_sales_fct WHERE chanl_id = 3
                  WHERE cal_yr_num <= 2000
                  GROUP BY prod_id, chanl_id) vw3
          GROUP BY prod_id) vw1,
    ON f2.prod_id = f3.prod_id WHERE chanl_id = 2
  GROUP BY p.prod_name, chanl3_amt
```

Q & A

+ Quiz

Plan Management
Workshop



- Create SQL profile using manual and advisor method.
- Test Profile.
- Create SQL plan baseline and test it.
- All statements available in workshop script.

Topic Agenda

Other Performance Tips (not prepared yet)

- Exadata Tips
  - Smart Scan
  - Bloom Filtering
  - Storage Index
  - Flash Cache