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

Oracle SQL Tuning

BASIC

www.lingaro.com

lingaro

The Value of Business Intelligence

Training Agenda

- SQL Tuning Introduction
- Instrumentation & Rewrite SQl Text
- SQL Plan Reading

Topic Agenda

SQL Tuning Introduction

- Dynamic Performance Views - V$
- Automatic Workload Repository - AWR
- Performance Metrics
- Time Model
- Wait Events
- Active Session History - ASH
- SQL Monitoring
- Optymizer Hints
- Bind Variables

Dynamic Performance Views

V$ Overview

- Based on x$ dynamic tables (based on SGA)
- Listed in v$fixed_table view
- Contains performance statistics (not only)
- Not guarantee read consistency
- Source for almost all performance tools
- Owned by SYS user (real names v_$)
- Public synonyms with v$ names
- SELECT_CATALOG_ROLE needed to access all
- Many are usable during SQL tuning
- Use gv$ in RAC database – union from all instances c$

Dynamic Performance Views

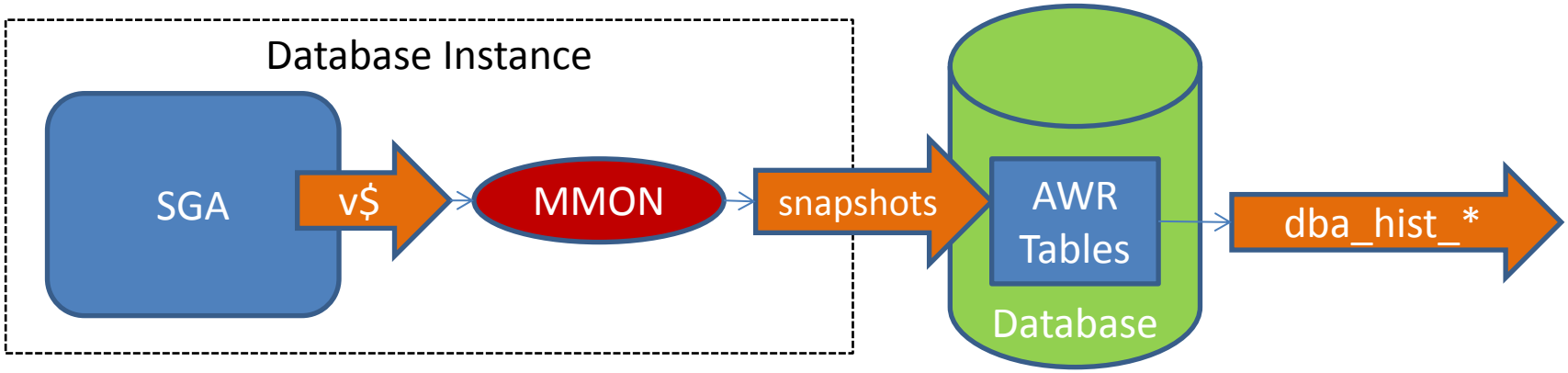
V$ Examples

- v$sqltext, v$sqltext_with_new_lines – sql text
- v$sql_bind_capture – bind variables definition and values
- v$sqlarea, v$sqlstat – cursor (statement) statistics
- v$sql – child cursor (cursor version/plan) statistics
- v$sql_plan – execution plan operations
- v$session – detailed sessions description
- v$sesstat – sessions statistics
- v$mystat – current session statistics
- v$service – active database services
- v$service_stats – services statistics
- v$instance – one row about database instance
- v$sysstat – instance statistics
- v$osstat – operating system statistics

www.lingaro.com

Automatic Workload Repository

AWR Overview




```
SELECT snap_interval, retension, topnsql FROM dba_hist_wr_control;
```

	SNAP_INTERVAL	RETENTION	TOPNSQL
	+00 01:00:00.000000	+08 00:00:00.000000	DEFAULT
Default	Snapshot every 1 hr	Left 8 days of retention	
ADW	+00 00:30:00.000000	+120 00:00:00.000000	DEFAULT

Number of top SQL captured for each criteria
(Elapsed Time, CPU Time, Parse Calls, -- Shareable Memory, Version Count).
DEFAILT = 30 for TYPICAL and 100 for ALL statistics level

Dictionary views

dba_hist_sqltext
dba_hist_sqlbind
dba_hist_sqlstat
dba_hist_sql_plan

dbs_hist_sess_time_stats
dba_hist_sessmetric_history
dba_hist_service_stat
dba_hist_database_instance
dba_hist_sysstat
dba_hist_sysmetric_history
dba_hist_osstat

www.lingaro.com

Performance Metrics

Overview

- Calculated automatically from base stats by MMON
- Calculated as delta in time – rate of change
- E.g. “Physical Reads Per Sec”
- Available in

v$metric – most recent values (only few metrics)
v$metricgroup – calculation intervals
v$metric_history – one hour history (60 and 15 seconds intervals)
v$sessmetric
v$servicemetric
v$servicemetric_history
v$sysmetric - many metrics
v$sysmetric_history
dba_hist_sessmetric_history
dba_hist_sysmetric_history

Time Model

Overview

- Database Time
  - Sum of all sessions time in ACTIVE state
  - Consists of processing (CPU) time and wait time (but no idle time)
- Time model
  - Time hierarchy with DB Time as root
  - Describing where time is used in database
  - Enable to drill down into most time consuming operations
  - Leaf can help to find very detailed performance problem reason
  - Leaf e.g. “hard parse (sc) time”

```
v$sess_time_model
v$sys_time_model
dba_hist_sys_time_model
```

STAT_NAME	VALUE
DB time	14093932161
DB CPU	4895968750
parse time elapsed	2584226426
hard parse elapsed time	1769777748
sql execute elapsed time	14278200204
connection management call elapsed time	92870541
failed parse elapsed time	121601539
failed parse (out of shared memory) elapsed time	0
hard parse (sharing criteria) elapsed time	732782215
hard parse (bind mismatch) elapsed time	3558585

Wait Events

Overview

- Occurred when active session is blocked by another process
  - See dba_waiters, dba_blockers, v$lock, v$locked_object, v$access, v$wait_chains
- v$session view display current (if in progress) or last wait

  status – ACTIVE or INACTIVE
  state – WAITING or WAITED KNOWN TIME
  event – wait event name
  wait_time – last wait time or 0 if currently wailing [1/100 sec]
  seconds_in_wait – If WAIT_TIME = 0, then SECONDS_IN_WAIT is the seconds spent in the current wait
  If WAIT_TIME > 0, then SECONDS_IN_WAIT is the seconds since the start of the last wait
  SECONDS_IN_WAIT - WAIT_TIME / 100 is the active seconds since the last wait ended
  p1text, p1, p1raw, p2text, p2, p2raw, p3text, p3, p3raw – wait parameters (v$event_name)
- When event is finished time is added to aggregates v$ stats in SGA

  v$waitstat
  v$waitclassmetric
  v$waitclassmetric_history
  v$service_wait_class
  v$session_wait
  v$session_wait_class
  v$session_wait_history
  v$system_wait_class

  v$session_event v$service_event
  v$system_event v$eventmetric

  and later written to AWR snapshots
  dba_hist_system_event
  dba_hist_service_wait_class
  dba_hist_waitclassmet_history
  dba_hist_waitstat

SQL Monitoring

Overview

- Real time statistic for statements during execution
- Active execution plan with current operation indication
- Real time statistics for operations in plan
- By default parallel and longer then 5 sec SQLs are monitored
- Enabled/disabled by hints /*+ MONITOR */ /*+ NOMONITOR */
- Results visible in
  - v$sql_monitor
  - v$sql_plan_monitor
  - In 11r2 are not avalilable in AWR
  - Flash/html/text monitoring report for single SQL

```
SELECT dbms_sqltune.report_sql_monitor( sql_id => '526mvccm5nfy4',
                                        type => 'ACTIVE', -- TEXT, HTML, XML
                                        report_level => 'ALL') AS report
  FROM dual;
```

Active Session History

ASH Overview

- **Every second** all active sessions are sampled
  - Session info, stats and wait event info from v$session are gathered
  - Only here single event parameters values are preserved (not like in agg)
- ASH visible in
  - SGA - **v$active_session_history** – most recent samples
  - AWR - **dba_hist_active_sess_history** - **only 10%** of samples from SGA
  - Same columns, e.g. **session_state** –"ON CPU" or "WAITING"
  - Not contains exact results (only samples)
  - Columns useful during SQL tuning

```
SAMPLE_TIME SESSION_ID SESSION_SERIAL# USER_ID SESSION_STATE
SQL_ID SQL_OPNAME SQL_PLAN_HASH_VALUE SQL_PLAN_LINE_ID
SQL_EXEC_ID SQL_EXEC_START PLSQL_OBJECT_ID PLSQL_SUBPROGRAM_ID
EVENT WAIT_TIME TIME_WAITED BLOCKING_SESSION BLOCKING_SESSION_SERIAL#
CURRENT_OBJ# IN_PARSE PROGRAM MODULE ACTION MACHINE
TM_DELTA_TIME TM_DELTA_CPU_TIME TM_DELTA_DB_TIME  [microsec]
```

SQL Monitoring

Lingaro SQL Monitor

- Java tool used to monitor SQL in Oracle DB
- Available in http://wiki.lingaro.local/display/PG/Oracle+SQL+Monitor
- Lists monitored statements

STATUS	USERNAME	SQL	EXECUTION_...	SQL_ID
xecuting				
	ADWG_IZOOMT2_BI	select /*+ parallel(4)*/'"||sc.node_trim_val||'"	2015-12-25	2xkyp743wz8up
	ADWGP_CBLS_RPT	select /*+ PARALLEL(2) */ T4269346.TIME_NAME as	2015-12-27	bu5jq5k5fu7wn
	ADWG_IZOOMT2_BI	select /*+ parallel(4)*/'"||sc.node_trim_val||'"	2015-12-23	4k59hd49n2w7h
	ADWGP_CBLS_RPT	select /*+ PARALLEL(2) */ T4269278.TIME_NAME as	2015-12-27	aujpwk322h5mp
	ADWGP_CBLS_RPT	select /*+ PARALLEL(2) */ T4269278.TIME_NAME as	2015-12-27	9x4tkus6jspp1

- Generates ACTIVE/HTML/TEXT monitoring report on demand
- Requires access to many v$ / gv$ views – best is select_catalog_role
- Version 2.0 requires:
  - java 7 JRE installed on laptop
  - Adobe Flash Player MSIE
- Example SQL Monitoring report

Example SQL Monitoring Report

Optymizer Hints

Introduction

- Used by developer to change SQL execution method (plan)
- Placed in comment after statement first keyword
- Hint comment have to start from + (plus) character
- Examples

```
SELECT /*+ USE_HASH(f d) FULL(f) SWAP_JOIN_INPUTS(f) */ ...
INSERT /*+ APPEND PARALLEL(4) */ INTO ...
```

- Possible strong influence on performance
- Using hints is kind of hardcoding
- Is not recommended – use as last resort
- Instead use:
  - Fresh optimizer statistics
  - SQL Profile
  - SQL Plan Baseline

Topic Agenda

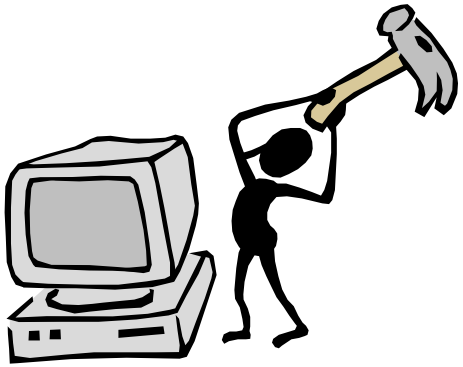
Tuning Techniques (not prepared yet)

- 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

Q & A

+ Quiz

Introduction
Workshop



- Login to Enterprice Manager

  default IP = 192.168.56.101

  https://<virtual host IP>:1158/em/

  User: sys Pass: oracle (as sysdba)

- Enter „Performance” page (and later) „SQL Monitoring” page

- Use script `SQL_Tuning_BASIC.sql` in Sqldeveloper
- Execute workload `sh_workload_pro(TRUE)`, make observations in EM
- Execute query on Dynamic Performance Views

- Execute query on `v$mystat` before and after workload on your session using

```
exec SH_WORKLOAD_PRO(TRUE,1,'prod_categ_pro',false);
exec SH_WORKLOAD_PRO(TRUE,1,'chanl_class_pro',false);
```

Introduction
Workshop

Performance Reporting Examples

```
SELECT /*+ NO_MERGE(s) */ sid, serial#, username,
       plsql_entry_object_id, object_type, object_name,
       sql_id, sql_text
  FROM (SELECT sid, serial#, sql_id, username, plsql_entry_object_id, sql_text
          FROM v$session NATURAL JOIN v$sqltext WHERE piece = 0
       ) s JOIN dba_objects o ON o.object_id = plsql_entry_object_id;
```

```
SELECT COUNT(*) samples, sql_id  FROM v$active_session_history WHERE session_state = 'ON CPU'
    AND user_id = (SELECT user_id FROM all_users WHERE username = 'TRAIN')
  GROUP BY sql_id ORDER BY samples DESC;
```

```
SELECT COUNT(*) samples, session_state, event, in_parse
  FROM v$active_session_history h WHERE h.sql_id = 'fdd01xqmvtkat'
  GROUP BY h.sql_id, session_state, event, in_parse ORDER BY samples DESC;
```

```
SELECT h.*, sql_text FROM (
    SELECT snap_id, begin_interval_time,
           COUNT(*) samples, user_id, sql_id, in_parse
      FROM dba_hist_active_sess_history NATURAL JOIN dba_hist_snapshot
      WHERE sql_id = 'b1thg2wm16t5b'
      GROUP BY sql_id, user_id, in_parse, snap_id, begin_interval_time
  ) h JOIN dba_hist_sqltext t ON h.sql_id = t.sql_id
  ORDER BY samples DESC;
```

Q & A

+ Workshop Comp

Topic Agenda

Instrumentation & Rewrite SQL Text

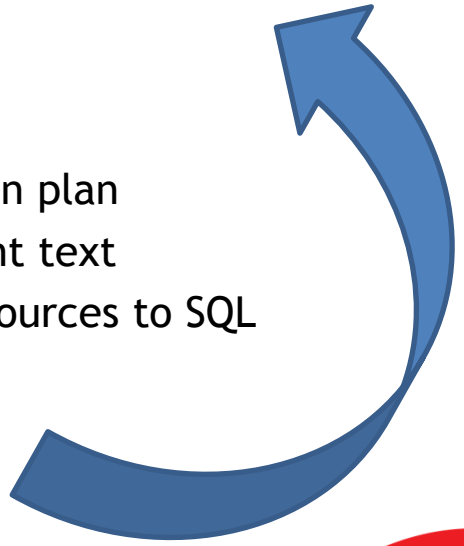
- Statistic Level & Module/Action Info
- Cricital Server Resource
- Resource Consumer SQL
- SQL Performance Issue Reasons
- Rewrite Problematic SQL Workshop

Prerequisites

SQL Tuning Steps

- 1. Set statistic level & module/action info (optional)
- 2. Find most critical (limited) resource in database server
  - CPU, Disk, Wait Time (in performance issue time range)
- 3. Find top SQL statements in workload by critical resource

  (Will be described later)
- 4. Analyze SQL plan operations (start from top SQL)
  - Rank operations in SQL
  - Find top cost reason
- 5. Tune SQL by
  - Influencing on execution plan
  - Rewriting SQL statement text
  - Adding more server resources to SQL



www.lingaro.com

Prerequisites

Optional Settings

- Set on workload session before workload (e.g. logon trigger)
- Statistic Level
  - Gather current statistics about row processed to execution plan by
  - Setting on session

```
ALTER SESSION SET statistic_level = ALL    -- default TYPICAL
```

  - In statement using hint

```
/*+ GATHER_PLAN_STATISTICS */
```

- Module, Action, Client Information
  - Mark session with module/action/client info
  - It will simplify finding interesting statements in statistics
  - It enables module/action/client rankings

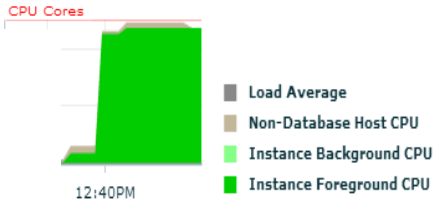
```
dbms_application_info.set_module('Loading FCT','STEP010')
dbms_application_info.set_action('STEP020')
```

Critical Resource

Is CPU Overloaded?

```
SELECT stat_name, value
  FROM v$osstat WHERE stat_name
    IN ('NUM_CPUS', 'NUM_CPU_CORES');
```

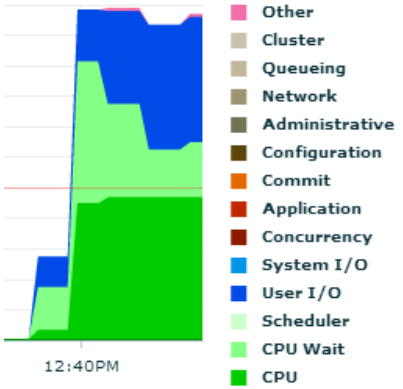
- Near 100% of host CPU utilization ?




```
SELECT value os_cpu, begin_time,
       numtodsinterval(end_time - begin_time, 'DAY') INTERVAL
  FROM v$sysmetric_history
  WHERE metric_name = 'Host CPU Utilization (%)'
  ORDER BY value desc; -- % Busy/(Idle+Busy)

SELECT round(b.VALUE/(i.VALUE+b.VALUE),3)*100 cpu_pct,
       begin_interval_time,
       end_interval_time - begin_interval_time INTERVAL
  FROM dba_hist_osstat i JOIN dba_hist_osstat b USING (snap_id,dbid,instance_number)
  JOIN dba_hist_snapshot USING (snap_id)
  WHERE b.stat_name = 'BUSY_TIME' AND i.stat_name = 'IDLE_TIME'
  ORDER BY cpu_pct DESC;
```

- Number of CPU session near number of cores ?

```
SELECT count(DISTINCT session_id) sess_cnt, sample_time
  FROM v$active_session_history WHERE session_state = 'ON CPU'
  GROUP BY sample_time ORDER BY sess_cnt DESC;
```



- CPU Wait is Over 10% of CPU time ?

SQL Ranks

Critical Resource

Is Storage Overloaded?

- IO Calibration

```
DBMS_RESOURCE_MANAGER.CALIBRATE_IO(num_physical_disks => in_dsk, max_latency => in_max_lat
        max_iops = > out_max_iops,  max_mbps => out_max_mbps, actual_latency => out_actual_lat)

SELECT * FROM dba_rsrc_io_calibrate;
```

START_TIME	END_TIME	MAX_IOPS	MAX_MBPS	MAX_PMBPS	LATENCY	NUM_PHYSICAL_DISKS
15/12/29 14:34:15,995000000	15/12/29 14:41:17,066000000	2019	289	289	0	1

- Check haw far is current workload from calibrated maximum

```
select round(value,2) per_sec, begin_time,
       numtodsinterval(end_time - begin_time, 'DAY') INTERVAL
  from v$sysmetric_history
  where metric_name = 'I/O Megabytes per Second'
  order by value desc;

SELECT round(VALUE,2) req_per_sec, begin_time,
       numtodsinterval(end_time - begin_time, 'DAY') INTERVAL
  FROM dba_hist_sysmetric_history
  WHERE metric_name = 'I/O Requests per Second'
    AND begin_time between sysdate - 1 and sysdate
  ORDER BY VALUE DESC;
```

TOP SQL Ranking

By Critical Resource in Time Range

- CPU rank based on CPU time

```
SELECT h.*, sql_text FROM (SELECT MAX(cpu_time_total) cpu_time,
                                  MAX(disk_reads_total) disk_rds, sql_id
                            FROM dba_hist_sqlstat NATURAL JOIN dba_hist_snapshot
                            WHERE begin_interval_time > sysdate - 1 GROUP BY sql_id
) h JOIN dba_hist_sqltext t ON h.sql_id = t.sql_id ORDER BY cpu_time DESC;
```

- CPU rank based on number of CPU samples in ASH
  - Number of samples = seconds in v$ (but in dbs_hist_ seconds/10)

```
SELECT h.*, sql_text FROM ( SELECT  COUNT(*) samples, user_id, sql_id
                             FROM dba_hist_active_sess_history
                             WHERE sample_time > sysdate - 1
                               AND session_state = 'ON CPU'  GROUP BY sql_id, user_id
) h JOIN dba_hist_sqltext t ON h.sql_id = t.sql_id JOIN all_users u ON h.user_id = u.user_id
  ORDER BY samples desc;
```

- Wait events rank for particular SQL

```
SELECT COUNT(*) samples, session_state, event, in_parse
  FROM dba_hist_active_sess_history h
  WHERE h.sql_id = '3f35td3yb01m6'
  GROUP BY h.sql_id, session_state,
           event, in_parse
  ORDER BY samples DESC;
```

SAMPLES	SESSION_STATE	EVENT	IN_PARSE
32	ON CPU	(null)	N
25	WAITING	db file scattered read	N
6	WAITING	db file sequential read	N

Issue Reasons

Introduction

- Stale or missing optimizer statistics
- Missing access structures
  - Indexes, Partitions, Materialized Views ...
- Suboptimal execution plan selection
  - Even statistics are correct
- Poorly constructed SQL
  - Logical, architectural and design mistakes
  - Poopy mistakes like unwanted cross join, missed filters
  - Not optimal processing method e.g. correlated subquery on large volume
  - Costly PL/SQL functions used in SQL

Issue Reasons

Inefficient SQL: Examples

correlation on large volume

```
SELECT COUNT(*) FROM prod_dim p
  WHERE prod_list_price <
    1.15 * (SELECT avg(unit_cost) FROM cost_fct c
              WHERE c.prod_id = p.prod_id)
```

expression not column on left side

```
SELECT * FROM job_history jh, employees e
  WHERE substr(to_char(e.employee_id),2) =
  substr(to_char(jh.employee_id),2)
```

conversion

```
SELECT * FROM orders WHERE order_id_char = 1205
```

function moved left

```
SELECT * FROM employees
    WHERE to_char(salary) = :sal
```

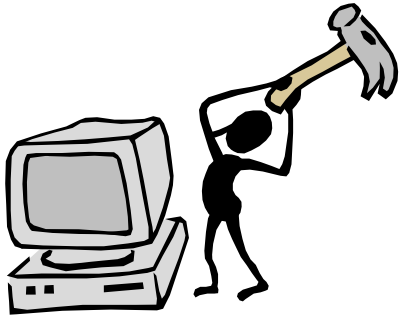
should be UNION ALL

```
SELECT * FROM parts_old
UNION
SELECT * FROM parts_new
```

Q & A

+ Quiz

Instrumentation
Workshop



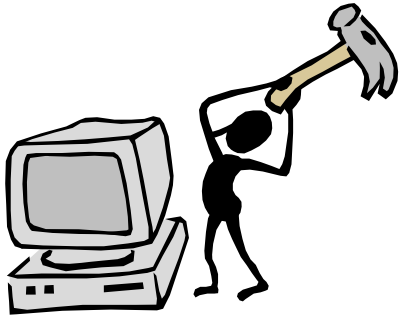
- Test `GATHER_PLAN_STATISTICS` hint
- Use module and action SQL indication

- Workshop statemets included in `SQL_Tuning_BASIC.sql`

Q & A

+ Workshop Comp

Rewrite SQL Text
Workshop Comp



- Modify SQL text to archive better performance
- Workshop statemets included in `SQL_Tuning_BASIC.sql`

Topic Agenda

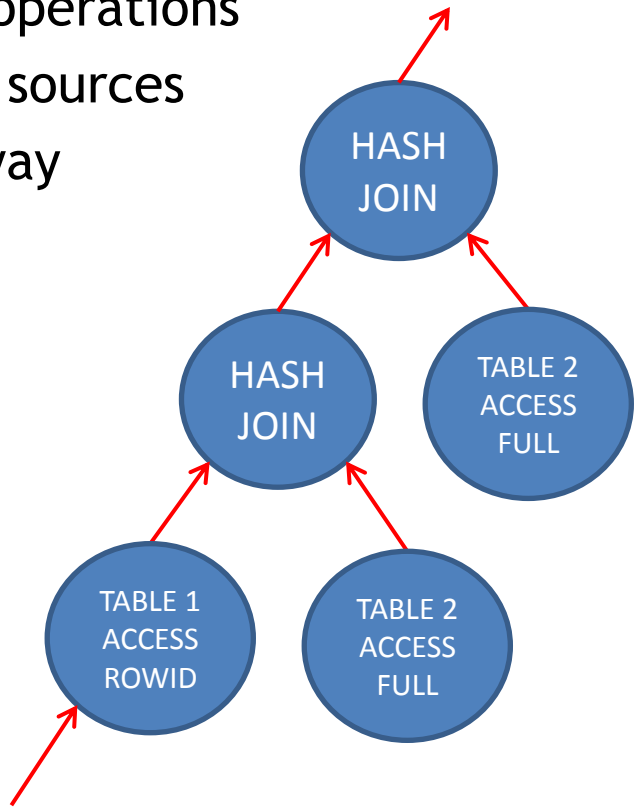
Execution Plan Reading

- Introduction
- Methods to See Plan
- Plan Interpretation

SQL Execution Plan

Inroduction

- Consists of plan Row Source Operations related in hierarchical form
- Child operation is Row Source (set of rows producer)
- Parent operations consumes rows from child operations
- JOIN parent operation receives data two row sources
- Is specification of SQL statement execution way
  - Needed because SQL code not contains all “how to do” info
  - One way from many possible ways
  - Way selected by optimizer during SQL hard parse phase
- Plan describes
  - Order of operations
  - Object access methods
  - Join methods
  - Join order
  - And more
- PL/SQL do not have execution plans
  - PL/SQL code describes how to do work itself



www.lingaro.com

See Execution Plan

Tools & Sources

- **PLAN_TABLE** (not really used plan)
  - EXPLAIN PLAN command and
    SELECT from PLAN_TABLE
    or DBMS_XPLAN.DISPLAY()
  - SQL Developer - F10
  - Autotrace
- **V$SQL_PLAN_MONITOR (11g)**
  DBMS_SQLTUNE.REPORT_SQL_MONITOR()
- **V$SQL_PLAN (Library Cache)**
  DBMS_XPLAN.DISPLAY_CURSOR()
- **DBA_HIST_SQL_PLAN (AWR)**
  DBMS_XPLAN.DISPLAY_AWR()
- SQL management base (SQL plan baselines)
  DBMS_XPLAN.DISPLAY_SQL_PLAN_BASELINE()
- SQL tuning set (STS)
  DBMS_XPLAN.DISPLAY_SQLSET()

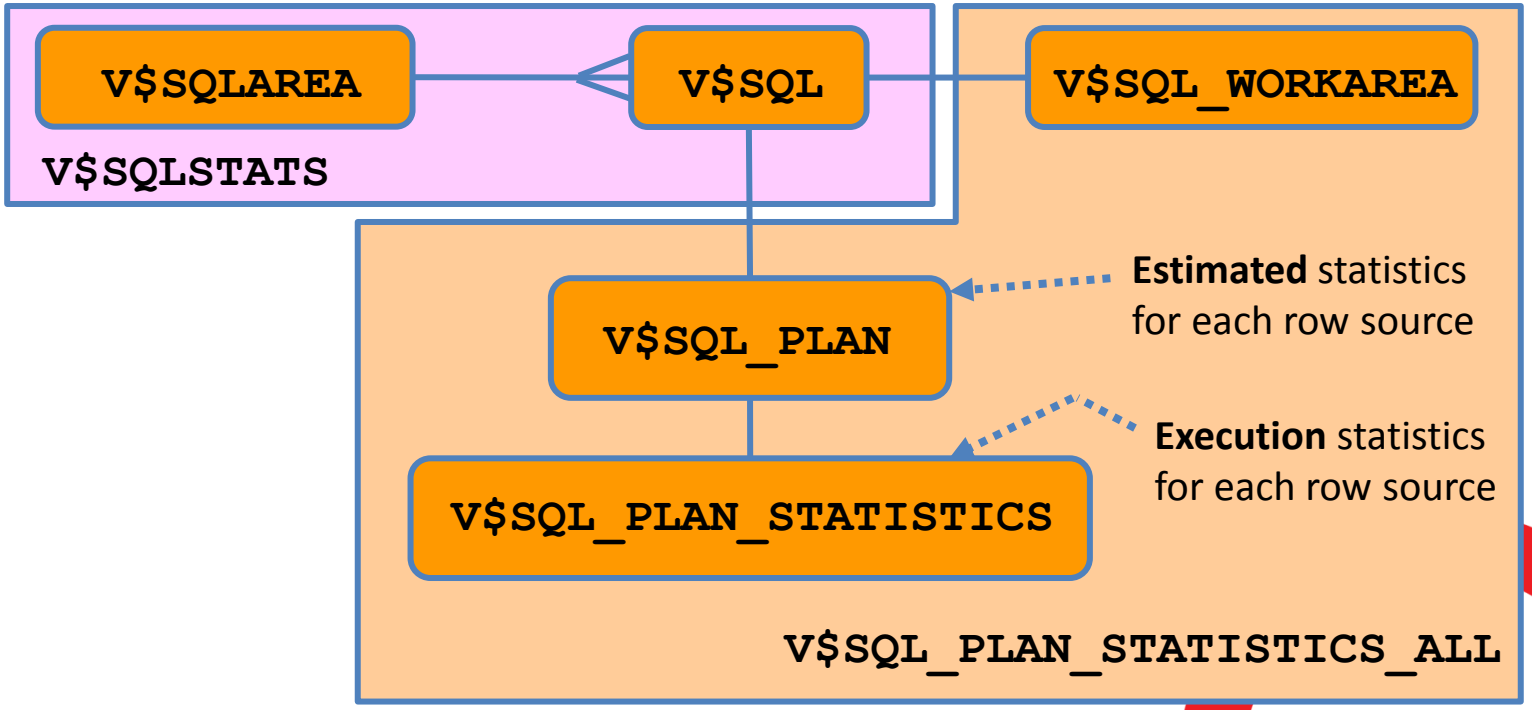
OPERATION	OBJECT_NAME	CARDINALITY	COST	PARTITION_START	PARTITION
SELECT STATEMENT		532664	21319		
HASH (GROUP BY)		532664	21319		
HASH JOIN		918843	3728		
Access Predicates					
T.TIME_ID=S.TIME_ID					
PART JOIN FILTER (CREATE)	:BF0000	1826	17		
TABLE ACCESS (FULL)	SH_TIME_DIM	1826	17		
HASH JOIN		918843	3706		
Access Predicates					
TABLE ACCESS (FULL)	SH_PROMO_DIM	503	5		
HASH JOIN		918843	3696		
Access Predicates					
VIEW	index$_join$_004	72	3		
HASH JOIN		918843	3688		
Access Predicates					
TABLE ACCESS (FULL)	SH_CNTRY_UKP	23	3		
HASH JOIN		918843	3681		
Access Predicates					
TABLE ACCESS (FULL)	SH_CHANL_DIM	5	3		
HASH JOIN		918843	3673		
Access Predicates					
TABLE ACCESS (FULLSH_CUST_DIM		55500	413		
PARTITION RANGE (		918843	1385	:BF0000	:BF0000

www.lingaro.com

See Execution Plan

Actual Execution Statistics

- V$SQL_PLAN_STATISTICS
  - STATISTICS_LEVEL set to ALL
  - The GATHER_PLAN_STATISTICS hint
- V$SQL_PLAN_STATISTICS_ALL
  - Enables side-by-side comparisons of estimates with the actual statistics.



PLAN_TABLE

- Automatically created but you can create - utlxplan.sql
- 👈 SQL is not executed. 👎 May not be the actual execution plan
- Row = one plan operation, hierarchy by `ID` and `PARENT_ID` columns.
- Many columns are the same like in v$sql_plan

Statement and plan columns:

ADDRESS	Address of the handle to the parent for this cursor
HASH_VALUE	Hash value of the parent statement
SQL_ID	SQL identifier of the parent cursor in the library cache
PLAN_HASH_VALUE	Numerical representation of the SQL plan for the cursor.
CHILD_ADDRESS	Address of the child cursor
CHILD_NUMBER	Number of the child cursor that uses this execution plan.
STATEMENT_ID	Parameter specified in the EXPLAIN PLAN statement
PLAN_ID	Unique identifier of a plan in the database
TIMESTAMP	Date and time when the execution plan was generated
REMARKS	Indicate whether an outline or SQL Profile was used

Operations columns:

ID	
PARENT_ID	
DEPTH	Depth (or level) of the operation in the tree.
POSITION	Order of processing for have the same PARENT_ID
OPERATION	
OPTIONS	e.g. object access method
OBJECT_NODE	
OBJECT_OWNER	
OBJECT_NAME	
OBJECT_ALIAS	
OBJECT_INSTANCE	
OBJECT_TYPE	

Estimates columns:

CARDINALITY
COST
BYTES
PARTITION_ID
OTHER
OTHER_XML
DISTRIBUTION
CPU_COST
IO_COST
TEMP_SPACE
TIME

Other columns:

OTHER_TAG
PARTITION_START
PARTITION_STOP
OPTIMIZER
SEARCH_COLUMNS
ACCESS_PREDICATES
FILTER_PREDICATES
PROJECTION
QBLOCK_NAME

PLAN_TABLE

Inserting Plan Rows

```
EXPLAIN PLAN
    [ SET STATEMENT_ID = 'text' ]
    [ INTO your plan table ]
    FOR statement
```

Example

```
SQL> EXPLAIN PLAN
  2  SET STATEMENT_ID = 'demo01' FOR
  3  SELECT * FROM emp;
```


www.lingaro.com

Show Plan

from PLAN_TABLE using DISPLAY function

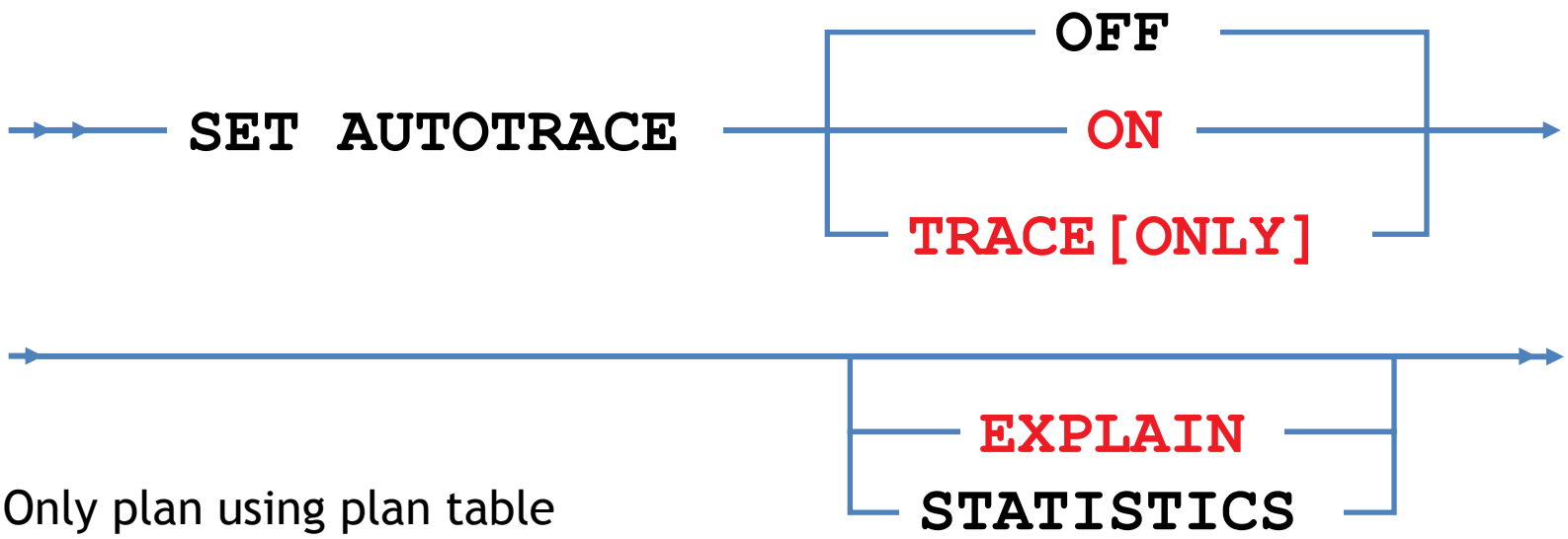
```
SELECT plan_table_output FROM TABLE(
  dbms_xplan.display(table_name => NULL, statement_id => 'demo01',
                     format => 'ADVANCED -Projection -Predicate -Aliases, filter_preds => NULL));

Plan hash value: 3956160932
-----------------------------------------------------------------------------
| Id  | Operation         | Name | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |      |     1 |    37 |     3   (0)| 00:00:01 |
|   1 |  TABLE ACCESS FULL| EMP  |     1 |    37 |     3   (0)| 00:00:01 |
-----------------------------------------------------------------------------
Outline Data
-------------
  /*+
      BEGIN_OUTLINE_DATA
      FULL(@"SEL$1" "EMP"@"SEL$1")
      OUTLINE_LEAF(@"SEL$1")
      ALL_ROWS
      DB_VERSION('11.1.0.6')
      OPTIMIZER_FEATURES_ENABLE('11.1.0.6')
      IGNORE_OPTIM_EMBEDDED_HINTS
      END_OUTLINE_DATA
  */
```


www.lingaro.com

Show Plan

Autotrace

- Autotrace Is a SQL*Plus and SQL Developer facility
- Needs a `PLAN_TABLE`
- Needs the `PLUSTRACE` role to retrieve statistics from some `V$` views
- By default, execution plan and statistics after running the query



- Only plan using plan table
  - `SET AUTOTRACE TRACE EXPLAIN`

Autotrace

SQL Execution Statistics

```
SQL> show autotrace
autotrace OFF
SQL> set autotrace traceonly statistics
SQL> SELECT * FROM prod_dim;

288 rows selected.

Statistics
----------------------------------------------------------
       1334  recursive calls
          0  db block gets
        686  consistent gets
        394  physical reads
          0  redo size
     103919  bytes sent via SQL*Net to client
        629  bytes received via SQL*Net from client
         21  SQL*Net roundtrips to/from client
         22  sorts (memory)
          0  sorts (disk)
        288  rows processed
```

DBMS_XPLAN

DISPLAY_CURSOR Function Example

```
SQL> SELECT plan_table_output
       FROM TABLE(dbms_xplan.display_cursor(sql_id => '3vjxpmhhzngu4',
                                            child_number => NULL
                                            format = 'ALL +Projection'));

SQL_ID  3vjxpmhhzngu4, child number 0
-------------------------------------
SELECT * FROM dual

Plan hash value: 272002086

---------------------------------------------------------------------------------------
| Id  | Operation                  | Name | Rows  | Bytes | Cost (%CPU)| Time     |
---------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT           |      |       |       |     2 (100)|          |
|   1 |  TABLE ACCESS STORAGE FULL| DUAL |     1 |     2 |     2   (0)| 00:00:01 |
---------------------------------------------------------------------------------------

Query Block Name / Object Alias (identified by operation id):
-------------------------------------------------------------

   1 - SEL$1 / DUAL@SEL$1

Column Projection Information (identified by operation id):
-----------------------------------------------------------

   1 - "DUAL"."DUMMY"[VARCHAR2,1]
```


www.lingaro.com

DBMS_XPLAN

DISPLAY_AWR Function Example

```
SQL> SELECT plan_table_output
       FROM TABLE(dbms_xplan.display_awr(sql_id => ' 78zm0s46bbm5h',
                                         plan_hash_value => NULL
                                         db_id => NULL
                                         format => 'TYPICAL +Note'));

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
SQL_ID 78zm0s46bbm5h
...
Plan hash value: 376465169
-------------------------------------------------------------------------------
| Id  | Operation          | Name        | Rows  | Bytes | Cost (%CPU)| Time    |
-------------------------------------------------------------------------------
...
-------------------------------------------------------------------------------

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


www.lingaro.com

DBMS_XPLAN

Output Format

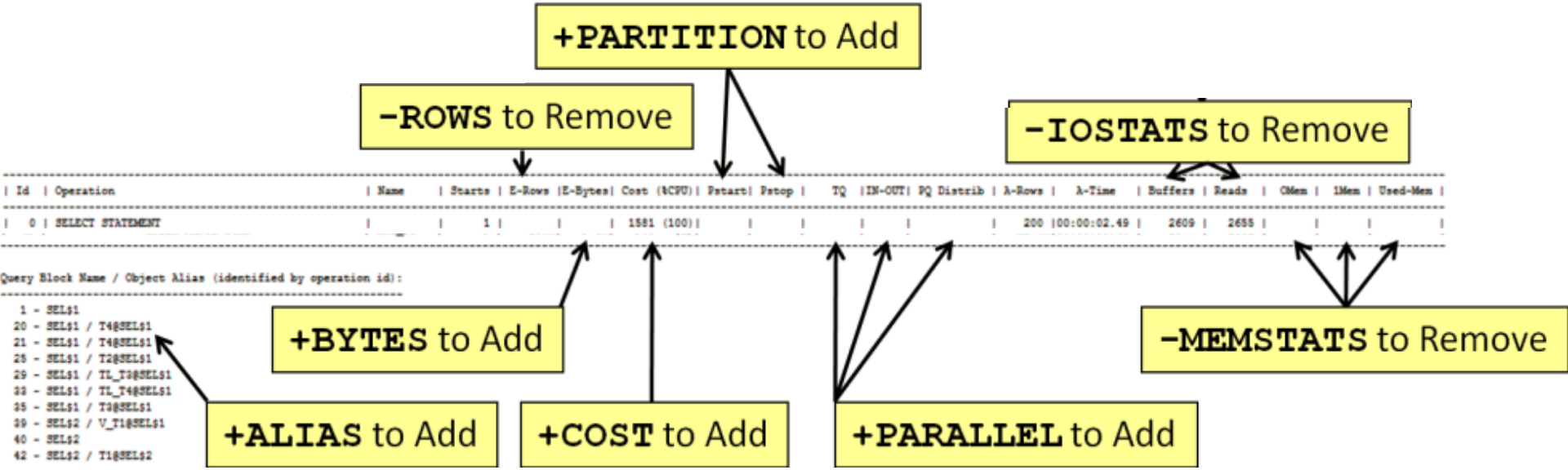
dbms_xplan.display(NULL, NULL, format => '<FORMAT +-flag +-flag ...>')

FORMAT: BASIC, TYPICAL, ALL, ADVANCED = ALL+Outline

flags: columns: iostat, memstat, allstats=iostat & memstat, bytes, parallel, cost, rows, partition

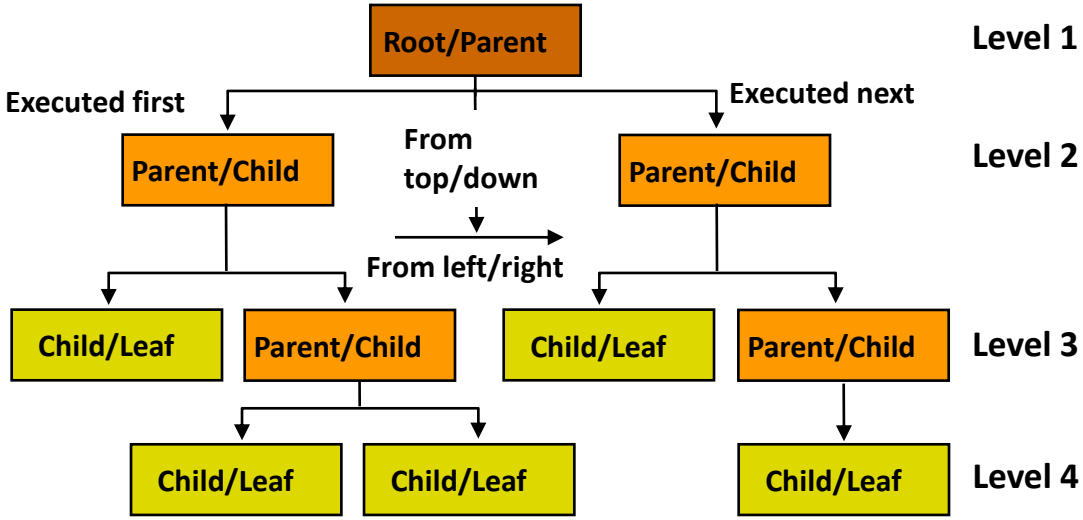
sections: Alias, Outline, Predicate, Projection, Note

executions: last, all (default)



Interpretation

```
id= 1  (pid= )                root/parent
id= 2  (pid=1)  (pos=1)            parent/child
id= 3  (pid=2)  (pos=1)              child/leaf
id= 4  (pid=2)  (pos=2)             parent/child
id= 5  (pid=4)  (pos=1)              child/leaf
id= 6  (pid=4)  (pos=2)              child/leaf
id= 7  (pid=1)  (pos=2)             parent/child
id= 8  (pid=7)  (pos=1)              child/leaf
id= 9  (pid=7)  (pos=2)             parent/child
id=10  (pid=9)  (pos=1)              child/leaf
```

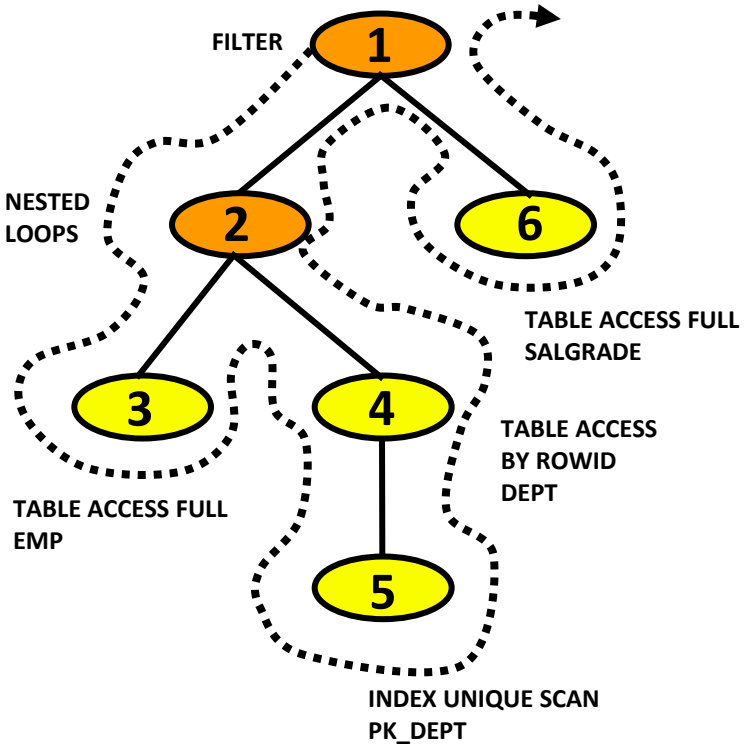


www.lingaro.com

Interpretation: Example 1

```
SELECT ename, job, sal, dname
  FROM emp, dept
  WHERE dept.deptno = emp.deptno AND NOT EXISTS(SELECT *
                                                FROM salgrade
                                                WHERE emp.sal BETWEEN losal AND hisal);
```

```
-----------------------------------------------------
| Id  | Operation                      | Name     |
-----------------------------------------------------
|   0 | SELECT STATEMENT               |          |
|*  1 |  FILTER                        |          |
|   2 |   NESTED LOOPS                 |          |
|   3 |    TABLE ACCESS FULL           | EMP      |
|   4 |    TABLE ACCESS BY INDEX ROWID| DEPT     |
|*  5 |     INDEX UNIQUE SCAN          | PK_DEPT  |
|*  6 |   TABLE ACCESS FULL            | SALGRADE |
-----------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter( NOT EXISTS
    (SELECT 0 FROM "SALGRADE" "SALGRADE" WHERE
     "HISAL">=:B1 AND "LOSAL"<=:B2))
   5 - access("DEPT"."DEPTNO"="EMP"."DEPTNO")
   6 - filter("HISAL">=:B1 AND "LOSAL"<=:B2)
```



www.lingaro.com

Interpretation: Example 1

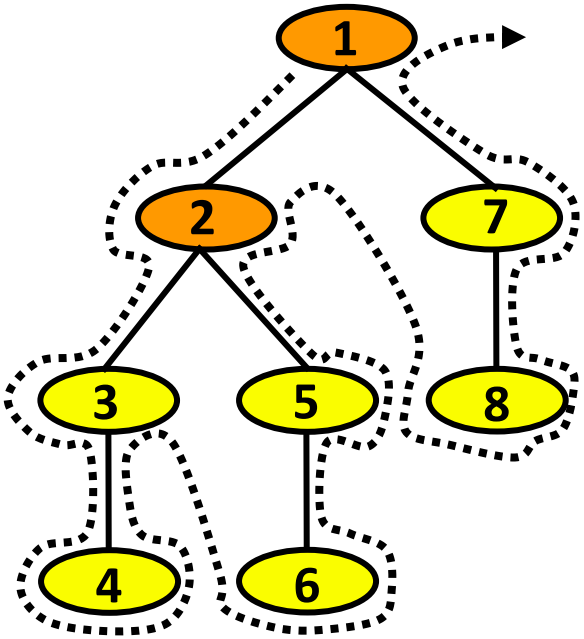
```
SQL> ALTER SESSION SET statistics_level=ALL;

Session altered.

<execute query>

no rows selected

SQL> SELECT *
       FROM TABLE(dbms_xplan.display_cursor(null,null,'TYPICAL IOSTATS LAST'));

SQL_ID  274019myw3vuf, child number 0
-------------------------------------
…
Plan hash value: 1175760222
-----------------------------------------------------------------------------------------
| Id  | Operation                      | Name     | Starts | A-Rows | Buffers |
-----------------------------------------------------------------------------------------
|*  1 |  FILTER                        |          |      1 |      0 |      61 |
|   2 |   NESTED LOOPS                 |          |      1 |     14 |      25 |
|   3 |    TABLE ACCESS FULL           | EMP      |      1 |     14 |       7 |
|   4 |    TABLE ACCESS BY INDEX ROWID| DEPT     |     14 |     14 |      18 |
|*  5 |     INDEX UNIQUE SCAN          | PK_DEPT  |     14 |     14 |       4 |
|*  6 |   TABLE ACCESS FULL            | SALGRADE |     12 |     12 |      36 |
-----------------------------------------------------------------------------------------
…
```

SQL Monitoring

	Executions	Actual Rows

Interpretation: Example 2

```
SQL> SELECT /*+ USE_NL(d) USE_NL(m) */
            m.last_name AS dept_manager,
  2         d.department_name,
  3         l.street_address
  4    FROM hr.employees m
  5    JOIN hr.departments d ON (d.manager_id = m.employee_id)
  6    NATURAL JOIN hr.locations l
  7    WHERE l.city = 'Seattle';
```



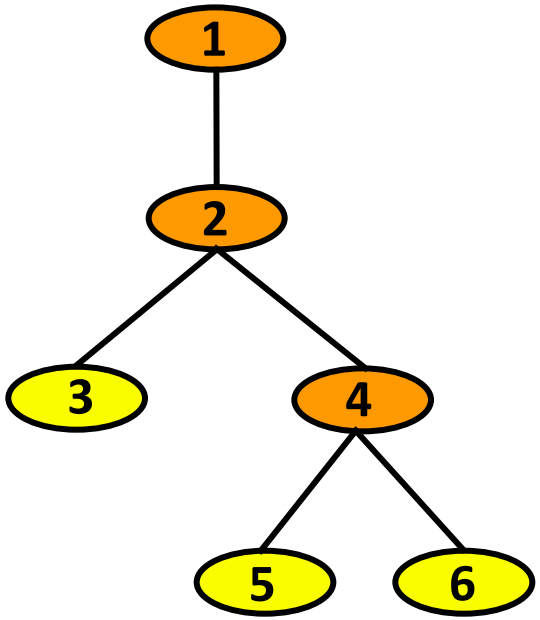

```
0    SELECT STATEMENT
1 0    NESTED LOOPS
2 1     NESTED LOOPS
3 2      TABLE ACCESS BY INDEX ROWID LOCATIONS
4 3       INDEX RANGE SCAN           LOC_CITY_IX
5 2      TABLE ACCESS BY INDEX ROWID DEPARTMENTS
6 5       INDEX RANGE SCAN           DEPT_LOCATION_IX
7 1     TABLE ACCESS BY INDEX ROWID  EMPLOYEES
8 7      INDEX UNIQUE SCAN           EMP_EMP_ID_PK
```


www.lingaro.com

Interpretation: Example 3

```
SELECT /*+ ORDERED USE_HASH(b) SWAP_JOIN_INPUTS(c) */
       MAX(a.i)
  FROM t1 a, t2 b, t3 c
  WHERE a.i = b.i AND a.i = c.i;
```

```
0    SELECT STATEMENT
1     SORT AGGREGATE
2 1    HASH JOIN
3 2     TABLE ACCESS FULL T3
4 2     HASH JOIN
5 4      TABLE ACCESS FULL T1
6 4      TABLE ACCESS FULL T2
```



Expand All | Collapse All

Operation	Object	Order
SELECT STATEMENT		7
SORT AGGREGATE		6
HASH JOIN		5
TABLE ACCESS FULL	T3	1
HASH JOIN		4
TABLE ACCESS FULL	T1	2
TABLE ACCESS FULL	T2	3

Join order is: T1 - T2 - T3

www.lingaro.com

Q & A

+ Quiz

Plan Reading
Workshop

- Interpret plan for top dbtime SQLs
  - using EM SQL Monitoring
  - during worklod jobs executed
- Display top CPU SQL plan using dbms_xplan
  - Minimize displayed information and later display all information
  - Analyze content of plan full report
- Use Autotrace to see statistics only (plan only later)
  - by execution labs/autotrace.sh script in virtual linux desktop

```
SELECT t.chanl_desc, p.prod_name, f.sold_amt
  FROM sh_sales_fct f
    INNER JOIN sh_chanl_dim t ON t.chanl_id  = f.chanl_id
    INNER JOIN sh_prod_dim p ON p.prod_id  = f.prod_id
    WHERE p.prod_id > 10;
```

- Change tables order in SQL text – is plan changed?
- Test following hints: ORDERED, FULL(p), LEADING(p)

Q & A

+ Workshop Comp

SQL Tuning Resources

- Oracle Database Documentation Library
  http://docs.oracle.com/cd/E11882_01/index.htm

- Database Performance Tuning Guide
  http://docs.oracle.com/cd/E11882_01/server.112/e41573/toc.htm

- Oracle SQL Tuning
  http://docs.oracle.com/cd/E28271_01/server.1111/e16638/sql_overview.htm