

```

select /*+ full(dep) full(loc) */
  emp.last_name, job.job_title, loc.city
  from jobs job,
       employees emp,
       departments dep,
       locations loc
 where job.job_id = emp.job_id
       and emp.department_id = dep.department_id
       and dep.location_id = loc.location_id
       and (emp.email = 'HBROWN' or job.job_id = 'HR_REP');

```

LAST_NAME	JOB_TITLE	CITY
Jacobs	Human Resources Representative	London
Brown	Public Relations Representative	Munich

2 rows selected.

Shh! We Have a [SQL] Plan

SQL_ID 6um9z82ayj75t, child number 0
Plan hash value: 1743992561

Id	Operation	Name	Starts	E-Rows	E-Bytes	Cost (%CPU)	E-Time	A-Rows	A-Time	Buffers	OMem	1Mem	Used-Mem
0	SELECT STATEMENT		1			12 (100)		2	00:00:00.01	21			
* 1	HASH JOIN		1	7	511	12 (9)	00:00:01	2	00:00:00.01	21	1106K	1106K	536K (0)
* 2	HASH JOIN		1	7	427	9 (12)	00:00:01	2	00:00:00.01	14	1106K	1106K	553K (0)
3	MERGE JOIN		1	7	378	6 (17)	00:00:01	2	00:00:00.01	8			
4	TABLE ACCESS BY INDEX ROWID	JOBS	1	19	513	2 (0)	00:00:01	19	00:00:00.01	2			
5	INDEX FULL SCAN	JOB_ID_PK	1	19		1 (0)	00:00:01	19	00:00:00.01	1			
* 6	FILTER		19					2	00:00:00.01	6			
* 7	SORT JOIN		19	107	2889	4 (25)	00:00:01	107	00:00:00.01	6	15360	15360	14336 (0)
8	TABLE ACCESS FULL	EMPLOYEES	1	107	2889	3 (0)	00:00:01	107	00:00:00.01	6			
9	TABLE ACCESS FULL	DEPARTMENTS	1	27	189	3 (0)	00:00:01	27	00:00:00.01	6			
10	TABLE ACCESS FULL	LOCATIONS	1	23	276	3 (0)	00:00:01	23	00:00:00.01	7			

Predicate Information (identified by operation id):

- ```

1 - access("DEP"."LOCATION_ID"="LOC"."LOCATION_ID")
2 - access("EMP"."DEPARTMENT_ID"="DEP"."DEPARTMENT_ID")
6 - filter(("EMP"."EMAIL"='HBROWN' OR "JOB"."JOB_ID"='HR_REP'))
7 - access("JOB"."JOB_ID"="EMP"."JOB_ID")
 filter("JOB"."JOB_ID"="EMP"."JOB_ID")

```

| Id   | Operation                           | Name            | Starts | E-Rows | E-Bytes | Cost (%CPU) | E-Time   | A-Rows | A-Time      | Buffers | Reads |
|------|-------------------------------------|-----------------|--------|--------|---------|-------------|----------|--------|-------------|---------|-------|
| 0    | SELECT STATEMENT                    |                 | 1      |        |         | 8 (100)     |          | 2      | 00:00:00.01 | 16      | 1     |
| 1    | VIEW                                | VW_ORE_8CFACDC3 | 1      | 2      | 100     | 8 (0)       | 00:00:01 | 2      | 00:00:00.01 | 16      | 1     |
| 2    | UNION-ALL                           |                 | 1      |        |         |             |          | 2      | 00:00:00.01 | 16      | 1     |
| 3    | NESTED LOOPS                        |                 | 1      | 1      | 73      | 4 (0)       | 00:00:01 | 1      | 00:00:00.01 | 8       | 1     |
| 4    | NESTED LOOPS                        |                 | 1      | 1      | 61      | 3 (0)       | 00:00:01 | 1      | 00:00:00.01 | 6       | 1     |
| 5    | NESTED LOOPS                        |                 | 1      | 1      | 54      | 2 (0)       | 00:00:01 | 1      | 00:00:00.01 | 4       | 0     |
| 6    | TABLE ACCESS BY INDEX ROWID         | EMPLOYEES       | 1      | 1      | 27      | 1 (0)       | 00:00:01 | 1      | 00:00:00.01 | 2       | 0     |
| * 7  | INDEX UNIQUE SCAN                   | EMP_EMAIL_UK    | 1      | 1      |         | 0 (0)       |          | 1      | 00:00:00.01 | 1       | 0     |
| 8    | TABLE ACCESS BY INDEX ROWID         | JOBS            | 1      | 1      | 27      | 1 (0)       | 00:00:01 | 1      | 00:00:00.01 | 2       | 0     |
| * 9  | INDEX UNIQUE SCAN                   | JOB_ID_PK       | 1      | 1      |         | 0 (0)       |          | 1      | 00:00:00.01 | 1       | 0     |
| 10   | TABLE ACCESS BY INDEX ROWID         | DEPARTMENTS     | 1      | 1      | 7       | 1 (0)       | 00:00:01 | 1      | 00:00:00.01 | 2       | 1     |
| * 11 | INDEX UNIQUE SCAN                   | DEPT_ID_PK      | 1      | 1      |         | 0 (0)       |          | 1      | 00:00:00.01 | 1       | 1     |
| 12   | TABLE ACCESS BY INDEX ROWID         | LOCATIONS       | 1      | 1      | 12      | 1 (0)       | 00:00:01 | 1      | 00:00:00.01 | 2       | 0     |
| * 13 | INDEX UNIQUE SCAN                   | LOC_ID_PK       | 1      | 1      |         | 0 (0)       |          | 1      | 00:00:00.01 | 1       | 0     |
| 14   | NESTED LOOPS                        |                 | 1      | 1      | 73      | 4 (0)       | 00:00:01 | 1      | 00:00:00.01 | 8       | 0     |
| 15   | NESTED LOOPS                        |                 | 1      | 1      | 73      | 4 (0)       | 00:00:01 | 1      | 00:00:00.01 | 7       | 0     |
| 16   | NESTED LOOPS                        |                 | 1      | 1      | 61      | 3 (0)       | 00:00:01 | 1      | 00:00:00.01 | 6       | 0     |
| 17   | NESTED LOOPS                        |                 | 1      | 1      | 54      | 2 (0)       | 00:00:01 | 1      | 00:00:00.01 | 4       | 0     |
| 18   | TABLE ACCESS BY INDEX ROWID         | JOBS            | 1      | 1      | 27      | 1 (0)       | 00:00:01 | 1      | 00:00:00.01 | 2       | 0     |
| * 19 | INDEX UNIQUE SCAN                   | JOB_ID_PK       | 1      | 1      |         | 0 (0)       |          | 1      | 00:00:00.01 | 1       | 0     |
| * 20 | TABLE ACCESS BY INDEX ROWID BATCHED | EMPLOYEES       | 1      | 1      | 27      | 1 (0)       | 00:00:01 | 1      | 00:00:00.01 | 2       | 0     |
| * 21 | INDEX RANGE SCAN                    | EMP_JOB_IX      | 1      | 1      |         | 0 (0)       |          | 1      | 00:00:00.01 | 1       | 0     |
| 22   | TABLE ACCESS BY INDEX ROWID         | DEPARTMENTS     | 1      | 1      | 7       | 1 (0)       | 00:00:01 | 1      | 00:00:00.01 | 2       | 0     |
| * 23 | INDEX UNIQUE SCAN                   | DEPT_ID_PK      | 1      | 1      |         | 0 (0)       |          | 1      | 00:00:00.01 | 1       | 0     |
| * 24 | INDEX UNIQUE SCAN                   | LOC_ID_PK       | 1      | 1      |         | 0 (0)       |          | 1      | 00:00:00.01 | 1       | 0     |
| 25   | TABLE ACCESS BY INDEX ROWID         | LOCATIONS       | 1      | 1      | 12      | 1 (0)       | 00:00:01 | 1      | 00:00:00.01 | 1       | 0     |

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

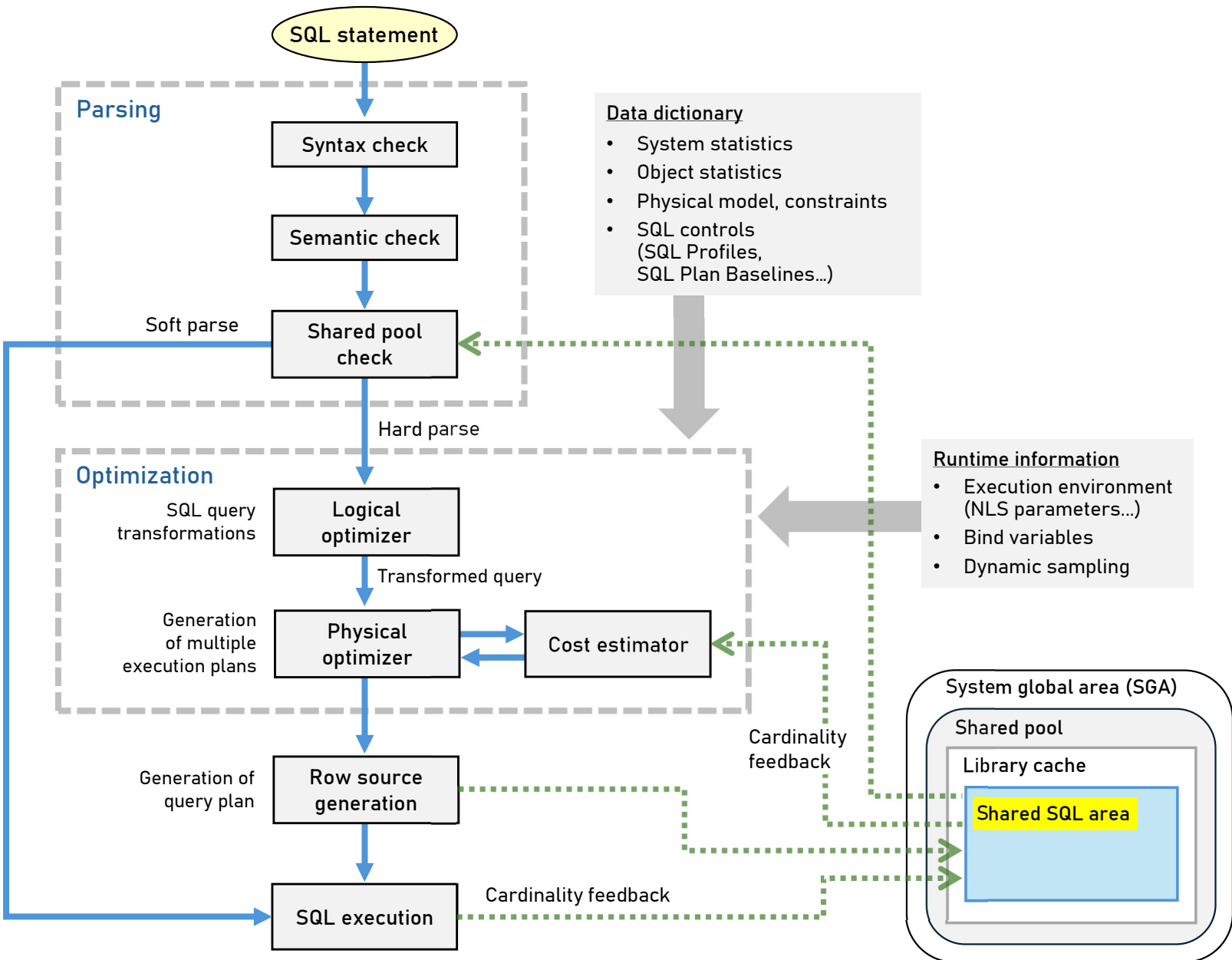
```

1 - SET$2A13AF86 / VW_ORE_8CFACDC3@SEL$8CFACDC3
2 - SET$2A13AF86
3 - SET$2A13AF86_1
6 - SET$2A13AF86_1 / EMP@SET$2A13AF86_1
7 - SET$2A13AF86_1 / EMP@SET$2A13AF86_1
8 - SET$2A13AF86_1 / JOB@SET$2A13AF86_1
9 - SET$2A13AF86_1 / JOB@SET$2A13AF86_1
10 - SET$2A13AF86_1 / DEP@SET$2A13AF86_1
11 - SET$2A13AF86_1 / DEP@SET$2A13AF86_1
12 - SET$2A13AF86_1 / LOC@SET$2A13AF86_1
13 - SET$2A13AF86_1 / LOC@SET$2A13AF86_1
14 - SET$2A13AF86_2
18 - SET$2A13AF86_2 / JOB@SET$2A13AF86_2
19 - SET$2A13AF86_2 / JOB@SET$2A13AF86_2
20 - SET$2A13AF86_2 / EMP@SET$2A13AF86_2
21 - SET$2A13AF86_2 / EMP@SET$2A13AF86_2
22 - SET$2A13AF86_2 / DEP@SET$2A13AF86_2
23 - SET$2A13AF86_2 / DEP@SET$2A13AF86_2
24 - SET$2A13AF86_2 / LOC@SET$2A13AF86_2
25 - SET$2A13AF86_2 / LOC@SET$2A13AF86_2

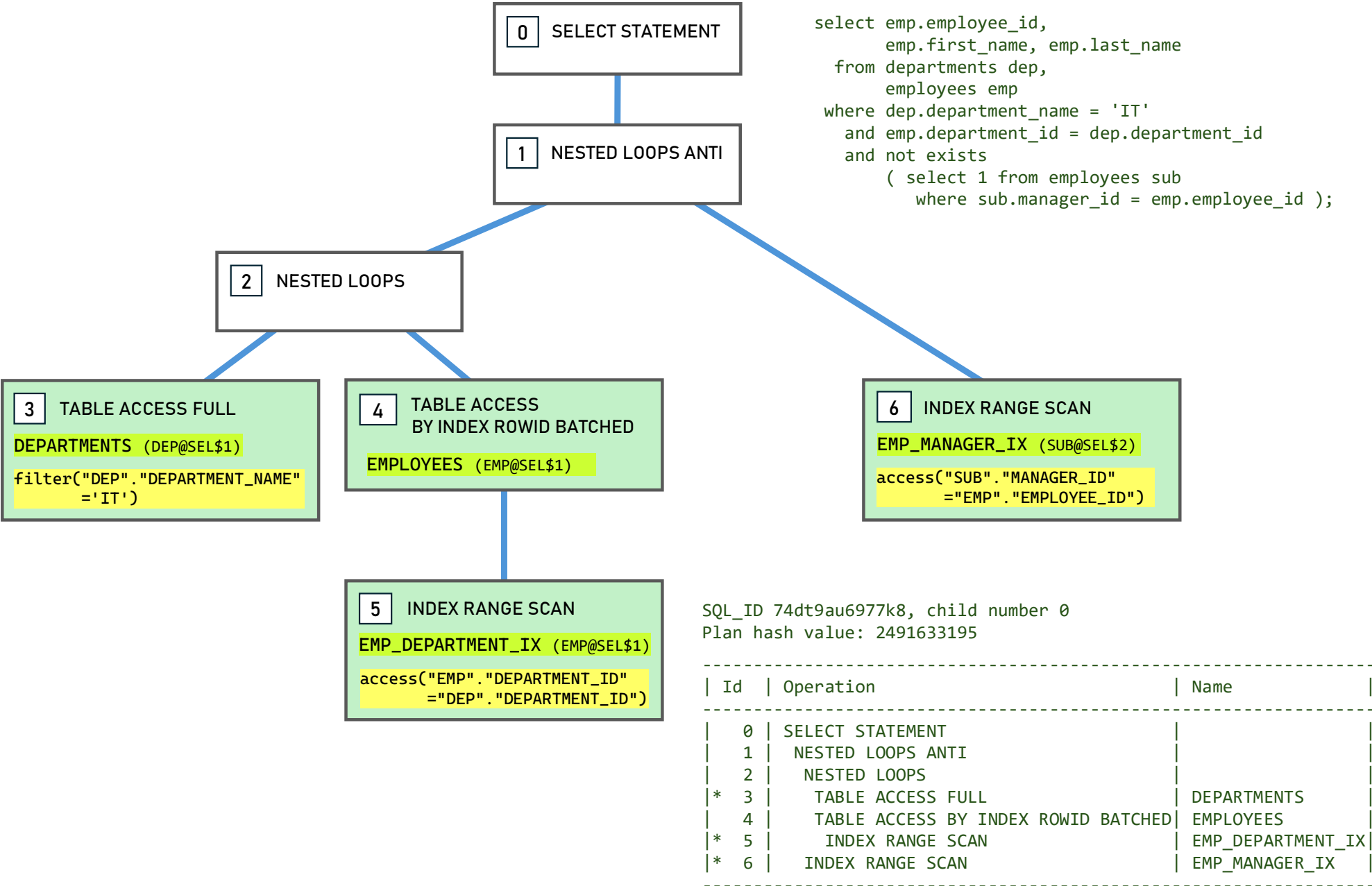
```

# Part #1: Introduction

# Overview of SQL processing



# A plan is a tree of row source operations



# SQL plan operations & options

select distinct operation, options from v\$sql\_plan union  
select distinct operation, options from dba\_hist\_sql\_plan  
order by 1, 2;

Only a limited subset of all operations  
and options can fit on this slide!

| OPERATION       | OPTIONS |
|-----------------|---------|
| FILTER          |         |
| COUNT           | STOPKEY |
| INLIST ITERATOR |         |

| OPERATION             | OPTIONS |
|-----------------------|---------|
| VIEW                  |         |
| VIEW PUSHED PREDICATE |         |

| OPERATION                  | OPTIONS |
|----------------------------|---------|
| CONCATENATION              |         |
| UNION-ALL                  |         |
| UNION ALL PUSHED PREDICATE |         |
| MINUS                      |         |

| OPERATION    | OPTIONS                       |
|--------------|-------------------------------|
| TABLE ACCESS | FULL                          |
| TABLE ACCESS | BY INDEX ROWID                |
| TABLE ACCESS | BY INDEX ROWID BATCHED        |
| TABLE ACCESS | BY LOCAL INDEX ROWID          |
| TABLE ACCESS | BY LOCAL INDEX ROWID BATCHED  |
| TABLE ACCESS | BY GLOBAL INDEX ROWID BATCHED |
| TABLE ACCESS | BY USER ROWID                 |

| OPERATION             | OPTIONS       |
|-----------------------|---------------|
| FAST DUAL             |               |
| XMLTABLE EVALUATION   |               |
| RESULT CACHE          |               |
| SEQUENCE              |               |
| FIXED TABLE           | FULL          |
| EXTERNAL TABLE ACCESS | FULL          |
| COLLECTION ITERATOR   | PICKLER FETCH |

| OPERATION       | OPTIONS  |
|-----------------|----------|
| PARTITION RANGE | SINGLE   |
| PARTITION RANGE | ITERATOR |
| PARTITION RANGE | ALL      |
| PARTITION RANGE | AND      |
| PARTITION RANGE | SUBQUERY |
| PARTITION LIST  | SINGLE   |
| PARTITION LIST  | ITERATOR |
| PARTITION LIST  | ALL      |
| PARTITION LIST  | SUBQUERY |
| PARTITION HASH  | SINGLE   |
| PARTITION HASH  | INLIST   |
| PARTITION HASH  | ALL      |
| PARTITION HASH  | SUBQUERY |

| OPERATION | OPTIONS               |
|-----------|-----------------------|
| INDEX     | UNIQUE SCAN           |
| INDEX     | RANGE SCAN            |
| INDEX     | RANGE SCAN DESCENDING |
| INDEX     | RANGE SCAN (MIN/MAX)  |
| INDEX     | FULL SCAN             |
| INDEX     | FAST FULL SCAN        |
| INDEX     | FULL SCAN (MIN/MAX)   |
| INDEX     | SAMPLE FAST FULL SCAN |
| INDEX     | SKIP SCAN             |

| OPERATION         | OPTIONS        |
|-------------------|----------------|
| BITMAP INDEX      | SINGLE VALUE   |
| BITMAP INDEX      | RANGE SCAN     |
| BITMAP INDEX      | FULL SCAN      |
| BITMAP INDEX      | FAST FULL SCAN |
| BITMAP AND        |                |
| BITMAP OR         |                |
| BITMAP MERGE      |                |
| BITMAP MINUS      |                |
| BITMAP CONVERSION | FROM ROWIDS    |
| BITMAP CONVERSION | TO ROWIDS      |

| OPERATION | OPTIONS           |
|-----------|-------------------|
| HASH      | UNIQUE            |
| HASH      | GROUP BY          |
| HASH      | GROUP BY PIVOT    |
| SORT      | UNIQUE            |
| SORT      | UNIQUE STOPKEY    |
| SORT      | GROUP BY          |
| SORT      | GROUP BY NOSORT   |
| SORT      | GROUP BY ROLLUP   |
| SORT      | ORDER BY          |
| SORT      | ORDER BY STOPKEY  |
| WINDOW    | BUFFER            |
| WINDOW    | SORT              |
| WINDOW    | SORT PUSHED RANK  |
| WINDOW    | CHILD PUSHED RANK |

| OPERATION                  | OPTIONS                      |
|----------------------------|------------------------------|
| CONNECT BY                 | NO FILTERING WITH START-WITH |
| CONNECT BY                 | WITH FILTERING               |
| CONNECT BY                 | WITH FILTERING (UNIQUE)      |
| CONNECT BY                 | WITHOUT FILTERING            |
| CONNECT BY PUMP            |                              |
| UNION ALL (RECURSIVE WITH) | DEPTH FIRST                  |
| UNION ALL (RECURSIVE WITH) | BREADTH FIRST                |
| RECURSIVE WITH PUMP        |                              |

| OPERATION        | OPTIONS              |
|------------------|----------------------|
| NESTED LOOPS     |                      |
| NESTED LOOPS     | ANTI                 |
| NESTED LOOPS     | OUTER                |
| NESTED LOOPS     | SEMI                 |
| HASH JOIN        |                      |
| HASH JOIN        | SEMI                 |
| HASH JOIN        | ANTI                 |
| HASH JOIN        | ANTI NA              |
| HASH JOIN        | OUTER                |
| HASH JOIN        | FULL OUTER           |
| HASH JOIN        | RIGHT SEMI           |
| HASH JOIN        | RIGHT ANTI           |
| HASH JOIN        | RIGHT OUTER          |
| HASH JOIN        | BUFFERED             |
| HASH JOIN        | OUTER BUFFERED       |
| HASH JOIN        | RIGHT OUTER BUFFERED |
| JOIN FILTER      | CREATE               |
| JOIN FILTER      | USE                  |
| PART JOIN FILTER | CREATE               |
| MERGE JOIN       |                      |
| MERGE JOIN       | ANTI                 |
| MERGE JOIN       | CARTESIAN            |
| MERGE JOIN       | OUTER                |
| MERGE JOIN       | SEMI                 |
| SORT             | JOIN                 |
| BUFFER           | SORT                 |
| BUFFER           | SORT (REUSE)         |

| OPERATION      | OPTIONS     |
|----------------|-------------|
| PX COORDINATOR |             |
| PX BLOCK       | ITERATOR    |
| PX RECEIVE     |             |
| PX SEND        | QC (ORDER)  |
| PX SEND        | QC (RANDOM) |
| PX SEND        | BROADCAST   |
| PX SEND        | HASH        |
| PX SEND        | HYBRID HASH |
| PX SEND        | RANGE       |
| PX SEND        | ROUND-ROBIN |

```

select loc.city, dep.department_name
 from (select location_id, city
 from locations
 where city in ('Whitehorse', 'Toronto')
) loc
 full outer join
 (select department_id, location_id, department_name
 from departments
 where department_id in (20, 230)
) dep
 on dep.location_id = loc.location_id;

```

SQL\_ID faskun742dmdj, child number 0  
Plan hash value: 2763787302

|   | Id | Operation                           | Name        | Starts | E-Rows | E-Bytes | Cost (%CPU) | E-Time   | A-Rows | A-Time      | Buffers | Reads |
|---|----|-------------------------------------|-------------|--------|--------|---------|-------------|----------|--------|-------------|---------|-------|
|   | 0  | SELECT STATEMENT                    |             | 1      |        |         | 4 (100)     |          | 3      | 00:00:00.01 | 7       | 1     |
|   | 1  | VIEW                                | VW_FOJ_0    | 1      | 2      | 68      | 4 (0)       | 00:00:01 | 3      | 00:00:00.01 | 7       | 1     |
| * | 2  | HASH JOIN FULL OUTER                |             | 1      | 2      | 120     | 4 (0)       | 00:00:01 | 3      | 00:00:00.01 | 7       | 1     |
|   | 3  | VIEW                                |             | 1      | 2      | 60      | 2 (0)       | 00:00:01 | 2      | 00:00:00.01 | 3       | 1     |
|   | 4  | INLIST ITERATOR                     |             | 1      |        |         |             |          | 2      | 00:00:00.01 | 3       | 1     |
|   | 5  | TABLE ACCESS BY INDEX ROWID BATCHED | LOCATIONS   | 2      | 2      | 24      | 2 (0)       | 00:00:01 | 2      | 00:00:00.01 | 3       | 1     |
| * | 6  | INDEX RANGE SCAN                    | LOC_CITY_IX | 2      | 2      |         | 1 (0)       | 00:00:01 | 2      | 00:00:00.01 | 2       | 1     |
|   | 7  | VIEW                                |             | 1      | 2      | 60      | 2 (0)       | 00:00:01 | 2      | 00:00:00.01 | 4       | 0     |
|   | 8  | INLIST ITERATOR                     |             | 1      |        |         |             |          | 2      | 00:00:00.01 | 4       | 0     |
|   | 9  | TABLE ACCESS BY INDEX ROWID         | DEPARTMENTS | 2      | 2      | 38      | 2 (0)       | 00:00:01 | 2      | 00:00:00.01 | 4       | 0     |
| * | 10 | INDEX UNIQUE SCAN                   | DEPT_ID_PK  | 2      | 2      |         | 1 (0)       | 00:00:01 | 2      | 00:00:00.01 | 2       | 0     |

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

```

1 - SEL$1 / from$_subquery$_005@SEL$4
2 - SEL$1
3 - SEL$2 / LOC@SEL$1
4 - SEL$2
5 - SEL$2 / LOCATIONS@SEL$2
6 - SEL$2 / LOCATIONS@SEL$2
7 - SEL$3 / DEP@SEL$1
8 - SEL$3
9 - SEL$3 / DEPARTMENTS@SEL$3
10 - SEL$3 / DEPARTMENTS@SEL$3

```

Predicate Information (identified by operation id):

```

2 - access("DEP"."LOCATION_ID"="LOC"."LOCATION_ID")
6 - access(("CITY"='Toronto' OR "CITY"='Whitehorse'))
10 - access(("DEPARTMENT_ID"=20 OR "DEPARTMENT_ID"=230))

```

## Part #2: Retrieving and Displaying Plans

## Method #1: EXPLAIN PLAN

Syntax:

```
EXPLAIN PLAN [set statement_id = 'identifier'] [INTO [schema.]plan_table_name]
FOR sql_statement;
```

Semantics: *sql\_statement* is not run; instead EXPLAIN PLAN:

- generates a plan for that statement
- inserts the plan details into SYS.PLAN\_TABLE\$, aka "PUBLIC".PLAN\_TABLE (or into the specified plan table)

```
select * from table(dbms_xplan.display('PLAN_TABLE', 'identifier', 'display_fmt'));
```

Prints a tabular representation of the plan, with details according to *display\_fmt*

## Method #2: retrieve and display actual plans

- Cursors still available in the cursor cache
  - Retrieve the *sql\_id* and *child\_number*
  - Print the plan details using:  

```
select * from table(dbms_xplan.display_cursor('sql_id', child_number, 'display_fmt'));
```
- Special case: latest cursor in *this* session:  

```
select * from table(dbms_xplan.display_cursor(null, null, 'display_fmt'));
```
- Plans stored in the AWR<sup>(\*)</sup>
  - Retrieve the *sql\_id* (and, possibly, the plan *hash\_value*)
  - Print the plan details using:  

```
select * from table(dbms_xplan.display_awr('sql_id', hash_value, null, 'display_fmt'));
```

db\_id; null = current database id

(\*) Requires the Advanced Diagnostics Pack license

## Plan-related tables & views

- *Predicted plans* from EXPLAIN PLAN: `PLAN_TABLE`
- *Actual plans* from the cursor cache: `V$SQL_PLAN_STATISTICS_ALL`
- *Actual plans* from the AWR: `DBA_HIST_SQL_PLAN` (\*)

## Requirements

- EXPLAIN PLAN: **privileges to run the target statement**  
+ **READ or SELECT on all underlying tables** (otherwise ORA-01039 is raised)
- Plans from the cursor cache: **READ / SELECT grants** on the following:
  - `V$SQL`
  - `V$SQL_PLAN_STATISTICS_ALL`
  - `V$SESSION` (columns `sql_id`, `child_number`, `prev_sql_id`, `prev_child_number`)
  - Plus, possibly: `V$ACTIVE_SESSION_HISTORY` (\*), etc.
- Plans from the AWR (\*): **READ / SELECT grants** on the following:
  - `DBA_HIST_SQLTEXT`
  - `DBA_HIST_SQL_PLAN`
  - Plus, possibly: `DBA_HIST_SQLSTAT`, `DBA_HIST_ACTIVE_SESS_HISTORY`, `DBA_HIST_SNAPSHOT`, etc.

EXPLAIN PLAN requires high privileges on the application's data.

Access to actual plans from the cursor cache or the AWR (\*) requires DBA-level (viewing) privileges.

(\*) Requires the Advanced Diagnostics Pack license



## EXPLAIN PLAN vs actual plans—which method should you use?

“**EXPLAIN PLAN *lies***”: the plan generated by EXPLAIN PLAN can be different from actual plans, due to EXPLAIN PLAN limitations:

- EXPLAIN PLAN does not use *bind peeking*  
Therefore, it always assumes VARCHAR data type, possibly using different type conversions than in reality  
And it cannot use column histograms at all, possibly resulting in a wholly different plan shape
- EXPLAIN PLAN requires privileges to run the target statement, plus READ / SELECT privileges on *all* underlying tables, creating opportunities for view merging that might otherwise not happen
- EXPLAIN PLAN always uses the *latest* published statistics,  
as opposed to statistics *at the time* when the actual cursor was created
- *Et caetera...* (adaptive plans?)

Bottom line: you mostly want to *use actual plans, especially in SQL tuning activities.*

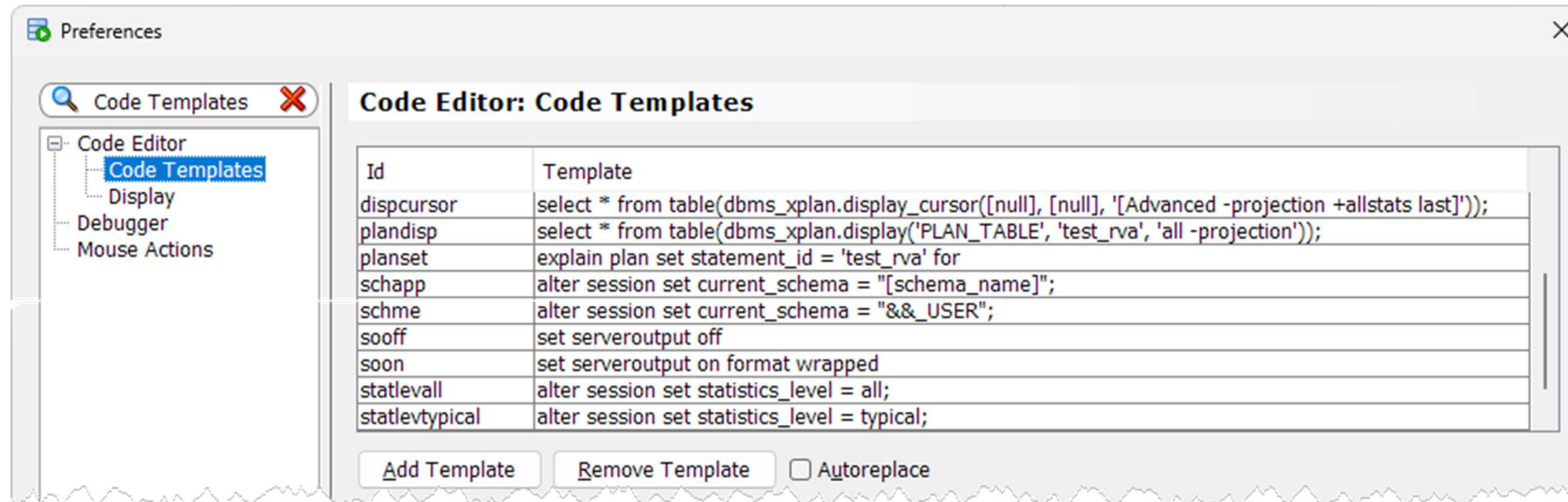
EXPLAIN PLAN still has its uses:

- For quick checking—or demonstration purposes—without actually running a statement
- As a workaround for `dbms_xplan.display_cursor` not being able to correctly render complex predicates—yielding meaningless expressions such as `filter( IS NULL)`

But you should be aware of the limitations.

Remark: the TKPROF utility uses EXPLAIN PLAN in the Execution Plan section, possibly resulting in a mismatch with the Row Source Operation section, which uses actual plan execution statistics from the trace file.

## SQL Dev. tip: use code templates to save typing



(Stored in: %APPDATA%\SQL Developer\CodeTemplate.xml)

Choose Ids which are:

i. easy to remind & type

And:

ii. which work well with auto-completion (so you just type a prefix, then Alt + Space)

| Id             | Template                                                                                                             |
|----------------|----------------------------------------------------------------------------------------------------------------------|
| schapp         | alter session set current_schema = "[schema_name]";                                                                  |
| schme          | alter session set current_schema = "&&_USER";                                                                        |
| sooff          | set serveroutput off                                                                                                 |
| soon           | set serveroutput on format wrapped                                                                                   |
| statlevall     | alter session set statistics_level = all;                                                                            |
| statlevtypical | alter session set statistics_level = typical;                                                                        |
| planset        | explain plan set statement_id = 'test_rva' for                                                                       |
| plandisp       | select * from table(dbms_xplan.display('PLAN_TABLE', 'test_rva', 'all -projection'));                                |
| dispcursor     | select * from table(dbms_xplan.display_cursor([null], [null], '[Advanced -projection -qbregistry +allstats last]')); |

DB ≥ 19c

# Demo: EXPLAIN PLAN

```
set pagesize 50000
```

```
variable JOB_ID varchar2(10)
exec :JOB_ID := 'SA_REP';
```

```
variable
print JOB_ID
```

```
explain plan set statement_id = 'test #1' for
select count(*)
 from hr.employees emp
 where emp.job_id = :JOB_ID
 and emp.hire_date > date '2010-01-01';
```

Include most useful information,  
without column projections

```
select * from table(dbms_xplan.display('PLAN_TABLE', 'test #1', 'All -projection'));
```

Plan hash value: 2830499944

Estimated by the SQL Optimizer

| Id  | Operation                   | Name       | Rows | Bytes | Cost (%CPU) | Time     |
|-----|-----------------------------|------------|------|-------|-------------|----------|
| 0   | SELECT STATEMENT            |            | 1    | 17    | 2 (0)       | 00:00:01 |
| 1   | SORT AGGREGATE              |            | 1    | 17    |             |          |
| * 2 | TABLE ACCESS BY INDEX ROWID | EMPLOYEES  | 6    | 102   | 2 (0)       | 00:00:01 |
| * 3 | INDEX RANGE SCAN            | EMP_JOB_IX | 6    |       | 1 (0)       | 00:00:01 |

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

1 - SEL\$1  
2 - SEL\$1 / EMP@SEL\$1  
3 - SEL\$1 / EMP@SEL\$1

Predicate Information (identified by operation id):

2 - filter("EMP"."HIRE\_DATE">TO\_DATE(' 2010-01-01 00:00:00',  
'yyyy-mm-dd hh24:mi:ss'))  
3 - access("EMP"."JOB\_ID"=:JOB\_ID)

**Note:** there *is* a pending transaction in the session at this stage, because EXPLAIN PLAN has inserted rows into the PLAN\_TABLE.

## Demo: dbms\_xplan.display\_cursor

```
set pagesize 50000
```

```
alter session set statistics_level = all;
```

Statistics level must be “all”, in order to collect *actual* plan statistics  
(Note: inherent overhead due to per-row source counting & timing)

```
set serveroutput off
```

Prevents SQL Dev from reading the dbms\_output buffer after each statement  
(thereby ruining prev\_sql\_id, prev\_child\_number in v\$sqlsession)

```
variable JOB_ID varchar2(10)
exec :JOB_ID := 'SA_REP';
```

```
select count(*)
 from hr.employees emp
 where emp.job_id = :JOB_ID
 and emp.hire_date > date '2010-01-01';
```

Use prev\_sql\_id,  
prev\_child\_number  
from v\$sqlsession

Include *actual* plan statistics  
(if available) in the readout

```
select * from table(dbms_xplan.display_cursor(null, null, 'All -projection +peeked_binds +allstats last'));
```

```
SQL_ID b0x08w3bzxdv, child number 0
```

```

Plan hash value: 1756381138
```

|                             |                   |           |            |               |         |             |          |            |             |         |
|-----------------------------|-------------------|-----------|------------|---------------|---------|-------------|----------|------------|-------------|---------|
| Plan hash value: 1756381138 |                   |           | Actual (A) | Estimated (E) |         |             |          | Actual (A) |             |         |
| Id                          | Operation         | Name      | Starts     | E-Rows        | E-Bytes | Cost (%CPU) | E-Time   | A-Rows     | A-Time      | Buffers |
| 0                           | SELECT STATEMENT  |           | 1          |               |         | 3 (100)     |          | 1          | 00:00:00.01 | 6       |
| 1                           | SORT AGGREGATE    |           | 1          | 1             | 17      |             |          | 1          | 00:00:00.01 | 6       |
| * 2                         | TABLE ACCESS FULL | EMPLOYEES | 1          | 30            | 510     | 3 (0)       | 00:00:01 | 30         | 00:00:00.01 | 6       |

```

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

```

1 - SEL$1
2 - SEL$1 / EMP@SEL$1
```

```

Peeked Binds (identified by position):
```

```

1 - :1 (VARCHAR2(30), CSID=873): 'SA_REP'
```

```

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

```

2 - filter(("EMP"."JOB_ID"=:JOB_ID AND "EMP"."HIRE_DATE"
 >TO_DATE(' 2010-01-01 00:00:00', 'syyy-mm-dd hh24:mi:ss')))
```

**Important:** always pay attention to the  
“Notes” section, if there is one.

SQL\_ID dcmc91w8z6s9d, child number 0  
Plan hash value: 2945430922

| Id   | Operation                           | Name          | Starts | E-Rows | E-Bytes | Cost (%CPU) | E-Time   | A-Rows | A-Time      | Buffers | Reads |
|------|-------------------------------------|---------------|--------|--------|---------|-------------|----------|--------|-------------|---------|-------|
| 0    | SELECT STATEMENT                    |               | 1      |        |         | 9 (100)     |          | 2      | 00:00:00.01 | 20      | 2     |
| 1    | NESTED LOOPS                        |               | 1      | 1      | 48      | 6 (0)       | 00:00:01 | 2      | 00:00:00.01 | 20      | 2     |
| 2    | NESTED LOOPS                        |               | 1      | 2      | 48      | 6 (0)       | 00:00:01 | 2      | 00:00:00.01 | 18      | 2     |
| 3    | NESTED LOOPS                        |               | 1      | 2      | 74      | 4 (0)       | 00:00:01 | 2      | 00:00:00.01 | 8       | 2     |
| 4    | TABLE ACCESS BY INDEX ROWID BATCHED | EMPLOYEES     | 1      | 2      | 44      | 2 (0)       | 00:00:01 | 2      | 00:00:00.01 | 4       | 1     |
| * 5  | INDEX SKIP SCAN                     | EMP_NAME_IX   | 1      | 2      |         | 1 (0)       | 00:00:01 | 2      | 00:00:00.01 | 2       | 1     |
| 6    | TABLE ACCESS BY INDEX ROWID         | EMPLOYEES     | 2      | 1      | 15      | 1 (0)       | 00:00:01 | 2      | 00:00:00.01 | 4       | 1     |
| * 7  | INDEX UNIQUE SCAN                   | EMP_EMP_ID_PK | 2      | 1      |         | 0 (0)       |          | 2      | 00:00:00.01 | 2       | 1     |
| * 8  | INDEX UNIQUE SCAN                   | EMP_EMP_ID_PK | 2      | 1      |         | 0 (0)       |          | 2      | 00:00:00.01 | 10      | 0     |
| 9    | NESTED LOOPS SEMI                   |               | 2      | 1      | 23      | 3 (0)       | 00:00:01 | 2      | 00:00:00.01 | 8       | 0     |
| * 10 | TABLE ACCESS BY INDEX ROWID BATCHED | EMPLOYEES     | 2      | 1      | 15      | 2 (0)       | 00:00:01 | 2      | 00:00:00.01 | 4       | 0     |
| * 11 | INDEX SKIP SCAN                     | EMP_NAME_IX   | 2      | 1      |         | 1 (0)       | 00:00:01 | 3      | 00:00:00.01 | 2       | 0     |
| * 12 | TABLE ACCESS BY INDEX ROWID         | EMPLOYEES     | 2      | 6      | 48      | 1 (0)       | 00:00:01 | 2      | 00:00:00.01 | 4       | 0     |
| * 13 | INDEX UNIQUE SCAN                   | EMP_EMP_ID_PK | 2      | 1      |         | 0 (0)       |          | 2      | 00:00:00.01 | 2       | 0     |
| 14   | TABLE ACCESS BY INDEX ROWID         | EMPLOYEES     | 2      | 1      | 11      | 1 (0)       | 00:00:01 | 2      | 00:00:00.01 | 2       | 0     |

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

```
1 - SEL$1
4 - SEL$1 / JAM1@SEL$1
5 - SEL$1 / JAM1@SEL$1
6 - SEL$1 / MGR1@SEL$1
7 - SEL$1 / MGR1@SEL$1
8 - SEL$1 / MGR2@SEL$1
9 - SEL$BE5C8E5F
10 - SEL$BE5C8E5F / JAM2@SEL$2
11 - SEL$BE5C8E5F / JAM2@SEL$2
12 - SEL$BE5C8E5F / MID@SEL$3
13 - SEL$BE5C8E5F / MID@SEL$3
14 - SEL$1 / MGR2@SEL$1
```

Peeked Binds (identified by position):

```
1 - (VARCHAR2(30), CSID=873): 'Julia'
```

Predicate Information (identified by operation id):

```
5 - access("JAM1"."FIRST_NAME"=:EMP_FIRST_NAME)
 filter("JAM1"."FIRST_NAME"=:EMP_FIRST_NAME)
7 - access("MGR1"."EMPLOYEE_ID"=:JAM1"."MANAGER_ID")
8 - access("MGR2"."EMPLOYEE_ID"=:MGR1"."MANAGER_ID")
 filter(IS NOT NULL)
10 - filter("JAM2"."EMPLOYEE_ID">=:B1)
11 - access("JAM2"."FIRST_NAME"=:B1)
 filter("JAM2"."FIRST_NAME"=:B1)
12 - filter("MID"."MANAGER_ID"=:B1)
13 - access("JAM2"."MANAGER_ID"=:MID"."EMPLOYEE_ID")
```

## Part #3: Understanding SQL Plans

## Christian Antognini's classification of plan operations

In Troubleshooting Oracle Performance<sup>(\*)</sup>, Antognini defined 4 Categories of Plan operations.

| Category of plan operations | Definition                                                                                                                                                                                                              | Examples                                                                                                     |
|-----------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------|
| Stand-alone                 | Single child operations, which start their child operation only once. Many operations belong in that category.                                                                                                          | VIEW<br>COUNT STOPKEY<br>SORT UNIQUE/ORDER BY/GROUP BY<br>HASH UNIQUE/GROUP BY<br>...<br>Single-child FILTER |
| Iterative                   | Single child operations, which may start their child operation repeatedly (or not at all)                                                                                                                               | INLIST ITERATOR<br>PARTITION LIST/RANGE/HASH ITERATOR                                                        |
| Unrelated-combine           | Operations with 2 (or more) child operations, which run their child operations only once, in turn, independantly of one another                                                                                         | HASH JOIN<br>MERGE JOIN<br>UNION ALL                                                                         |
| Related-combine             | Operations with 2 (or more) child operations, in which processing is driven by rows from one of the children, and the other child operations are called repeatedly, using the current row of the driving child as input | NESTED LOOPS<br>FILTER with multiple children<br>CONNECT BY WITH FILTERING<br>UNION ALL (RECURSIVE WITH)     |

This is a model—there are exceptions, and special cases—but a most helpful one.

(\*) Troubleshooting Oracle Performance, 2nd Edition [[link](#)]  
Christian Antognini, Apress, 2014  
ISBN-13 (softcover): 978-1-4302-5758-5 / ISBN-13 (electronic): 978-1-4302-5759-2

## HASH JOIN pseudo-code *(high-level, simplified perspective)*

## HASH JOIN

|                    |                                             |              |                                   |
|--------------------|---------------------------------------------|--------------|-----------------------------------|
| CHILD_ROW_SOURCE_1 | ← driving/build row source, or “left” input | alias: $r_1$ | columns: $(c_1, c_2, \dots, c_n)$ |
| CHILD_ROW_SOURCE_2 | ← probe row source, or “right” input        | alias: $r_2$ | columns: $(c_1, c_2, \dots, c_m)$ |

with join conditions as follows:

$$\left. \begin{array}{l} r_1 \cdot c_{h_1} = r_2 \cdot c_{j_1} \\ \text{and } r_1 \cdot c_{h_2} = r_2 \cdot c_{j_2} \\ \dots \\ \text{and } r_1 \cdot c_{h_k} = r_2 \cdot c_{j_k} \end{array} \right\} \text{equality conditions}$$
  

$$\text{and expression}(r_1 \cdot c_{h_{k+1}}, \dots, r_1 \cdot c_{h_p},$$

$$r_2 \cdot c_{j_{k+1}}, \dots, r_2 \cdot c_{j_q}) \left. \vphantom{\text{expression}} \right\} \text{non-equality conditions}$$

Start CHILD\_ROW\_SOURCE\_1

```

For each row $r_1 = (c_1, c_2, \dots, c_n)$ from CHILD_ROW_SOURCE_1 Loop -- build loop
 insert r_1 into the hash table using $(r_1.c_{h_1}, \dots, r_1.c_{h_k})$ as the hash key

```

```
End loop -- CHILD_ROW_SOURCE_1 has been fully processed
```

If CHILD\_ROW\_SOURCE\_1 returned at least 1 row Then

Start CHILD\_ROW\_SOURCE\_2

```
For each row $r_2 = (c_1, c_2, \dots, c_m)$ from CHILD_ROW_SOURCE_2 Loop -- probe loop
```

For each row  $r_1$  matching  $(r_2.c_{j_1}, \dots, r_2.c_{j_k})$  in the hash table /\* access conditions \*/ Loop

```
/* evaluate non-equality conditions: filter conditions */
```

**If**  $\text{expression}(r_1.c_{h_{k+1}}, \dots, r_1.c_{h_n}, r_2.c_{j_{k+1}}, \dots, r_2.c_{j_n})$  **is true** **Then**

Yield the combined row  $rj = (r_1.c_1, \dots, r_1.c_n, r_2.c_1, \dots, r_2.c_m)$  to the parent operation  $(*)$

**End If**

**End Loop**

**End Loop**

**End If**

(\*) Actually, only projected columns are passed to the parent operation

Key points:

- CHILD\_ROW\_SOURCE\_1 and \_2 are started only once (per start of the parent), and processed independently, in turn
- The hash table (in workarea) is built from CHILD\_ROW\_SOURCE\_1: rows from CHILD\_ROW\_SOURCE\_2 are not buffered (*iff* the hash join can be processed fully in memory)
- The hash key is formed of equi-joined columns; non-equality join conditions are always used as *filter* conditions, and evaluated by *iterating* on rows matching the probe key in the hash table—if there are too many such rows, a lot of CPU time could go into that
- The optimizer may swap join inputs, depending on (estimated) memory requirements of using either as the build row source

## NESTED LOOPS pseudo-code *(high-level, simplified perspective)*

### NESTED LOOPS

CHILD\_ROW\_SOURCE\_1  $\leftarrow$  driving row source (or “outer” row source)    alias:  $r_1$     columns:  $(c_1, c_2, \dots, c_n)$   
CHILD\_ROW\_SOURCE\_2  $\leftarrow$  inner row source (or “probe” row source)    alias:  $r_2$     columns:  $(c_1, c_2, \dots, c_m)$

with join conditions defined on columns  $(c_{h_1}, c_{h_2}, \dots, c_{h_p})$  of  $r_1$ , and  $(c_{j_1}, c_{j_2}, \dots, c_{j_q})$  of  $r_2$

Start CHILD\_ROW\_SOURCE\_1

**For** each row  $r_1 = (c_1, c_2, \dots, c_n)$  from CHILD\_ROW\_SOURCE\_1 **Loop** -- outer loop

    Start CHILD\_ROW\_SOURCE\_2, given  $(r_1.c_{h_1}, r_1.c_{h_2}, \dots, r_1.c_{h_p})$

    /\*  
    CHILD\_ROW\_SOURCE\_2 uses the values of columns from the  
    current row  $r_1$  in join access/filter conditions in order  
    to find all rows  $r_2$  joining with  $r_1$   
    \*/

**For** each row  $r_2 = (c_1, c_2, \dots, c_m)$  from CHILD\_ROW\_SOURCE\_2 **Loop** -- inner loop

        /\*  
        Rows from CHILD\_ROW\_SOURCE\_2 are joined to the  
        current row from CHILD\_ROW\_SOURCE\_1  
        \*/

        Yield the combined row  $r_j = (r_1.c_1, \dots, r_1.c_n, r_2.c_1, \dots, r_2.c_m)$  to the parent operation (\*)

**End Loop**

**End loop**

(\*) Actually, only projected columns  
are passed to the parent operation

Key points:

- CHILD\_ROW\_SOURCE\_1 is started once per start of its parent
- CHILD\_ROW\_SOURCE\_2 is started as many times as CHILD\_ROW\_SOURCE\_1 supplies a row to be joined with
- CHILD\_ROW\_SOURCE\_2 uses join columns from the “outer row” as input
- Join access/filter conditions are processed by CHILD\_ROW\_SOURCE\_2



```
merge into
(select emp.employee_id, emp.job_id, emp.salary
 from employees emp
) tgt
using (select sal.employee_id, sal.salary_incr_pct
 from &_USER..salary_raises sal
) src
on (tgt.employee_id = src.employee_id)
when matched then update
 set tgt.salary = tgt.salary * (1 + src.salary_incr_pct / 100)
 where (select job.max_salary from jobs job
 where job.job_id = tgt.job_id) >=
 tgt.salary * (1 + src.salary_incr_pct / 100);
```

1 row merged.

SQL\_ID 44srjbpw278ra, child number 0  
Plan hash value: 3955867600

| Id  | Operation                   | Name          | Starts | E-Rows | E-Bytes | Cost (%CPU) | E-Time   | IN-OUT | A-Rows | A-Time      | Buffers |
|-----|-----------------------------|---------------|--------|--------|---------|-------------|----------|--------|--------|-------------|---------|
| 0   | MERGE STATEMENT             |               | 1      |        |         | 5 (100)     |          |        | 0      | 00:00:00.01 | 12      |
| 1   | MERGE                       | EMPLOYEES     | 1      |        |         |             |          |        | 0      | 00:00:00.01 | 12      |
| 2   | VIEW                        |               | 1      |        |         |             |          |        | 3      | 00:00:00.01 | 7       |
| 3   | NESTED LOOPS                |               | 1      | 3      | 111     | 5 (0)       | 00:00:01 |        | 3      | 00:00:00.01 | 7       |
| 4   | NESTED LOOPS                |               | 1      | 3      | 111     | 5 (0)       | 00:00:01 |        | 3      | 00:00:00.01 | 4       |
| 5   | TABLE ACCESS FULL           | SALARY_RAISES | 1      | 3      | 24      | 2 (0)       | 00:00:01 |        | 3      | 00:00:00.01 | 2       |
| * 6 | INDEX UNIQUE SCAN           | EMP_EMP_ID_PK | 3      | 1      |         | 0 (0)       |          |        | 3      | 00:00:00.01 | 2       |
| 7   | TABLE ACCESS BY INDEX ROWID | EMPLOYEES     | 3      | 1      | 29      | 1 (0)       | 00:00:01 |        | 3      | 00:00:00.01 | 3       |
| 8   | TABLE ACCESS BY INDEX ROWID | JOBS          | 2      | 1      | 12      | 1 (0)       | 00:00:01 | PCWP   | 2      | 00:00:00.01 | 4       |
| * 9 | INDEX UNIQUE SCAN           | JOB_ID_PK     | 2      | 1      |         | 0 (0)       |          | PCWP   | 2      | 00:00:00.01 | 2       |

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

```
1 - SEL$76AA3327
3 - SEL$8984BF49
5 - SEL$8984BF49 / SAL@SEL$4
6 - SEL$8984BF49 / EMP@SEL$3
7 - SEL$8984BF49 / EMP@SEL$3
8 - SEL$6 / JOB@SEL$6
9 - SEL$6 / JOB@SEL$6
```

Predicate Information (identified by operation id):

```
6 - access("EMP"."EMPLOYEE_ID"="SAL"."EMPLOYEE_ID")
9 - access("JOB"."JOB_ID"=:B1)
```

## Part #4: Demos

# Demo SQL scripts

```
Oracle Instant Client - sqlplus /nolog

===== [Demo #4] =====

-- Employees with ids between 100 and 105, having prior assignment before 2010
-- Note: the subquery is hinted to demonstrate this particular plan shape, with
-- (expectedly) a child operation below the index range scan at line id 2.

select emp.employee_id,
 emp.first_name,
 emp.last_name
 from employees emp
 where emp.employee_id between 100 and 105
 and exists (select /*+ no_unnest push_subq */ 1
 from job_history jh
 where jh.employee_id = emp.employee_id
 and jh.start_date <= date '2010-01-01');

EMPLOYEE_ID FIRST_NAME LAST_NAME

101 Neena Yang

1 row selected.

SQL_ID f218773b3h29j, child number 0
Plan hash value: 641866015

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

0 | SELECT STATEMENT | | 1 | | 18 | 3 (100) | | |
1 | TABLE ACCESS BY INDEX ROWID BATCHED | EMPLOYEES | 1 | | 18 | 2 (0) | 00:00:01 | |
* 2 | INDEX RANGE SCAN | EMP_EMP_ID_PK | 1 | | | 1 (0) | 00:00:01 | |
* 3 | INDEX SKIP SCAN | JHIST_EMP_ID_ST_DATE_PK | 6 | | 12 | 1 (0) | 00:00:01 | |

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

1 - SEL$1 / EMP@SEL$1
2 - SEL$1 / EMP@SEL$1
3 - SEL$2 / JH@SEL$2
```

Source code : [link](#)

## Requirements

- HR schema, from the Oracle Database Sample Schemas 23c [[download link](#); [installation instructions](#)]
- A user with DML rights on HR's tables, plus READ/SELECT on a few v\$ views

## Principles

The demo consists in 4 SQL\*Plus scripts intended for a live demo:

- A simple query is run
- The corresponding plan (from the cursor cache) is shown, possibly with a comment or two
- The script pauses before continuing with the next example
- Repeat...

See the README [[link](#)] for details.