Chasing the optimizer, step by step

Mauro Pagano

Mauro Pagano

- Consultant/Developer/Analyst
- Oracle ➔ Enkitec ➔ Accenture
- DBPerf and SQL Tuning
- Training
- Tools (SQLT, SQLd360, Pathfinder)

Agenda

- Optimizer
  - Introduction
  - Query Blocks
  - Physical Optimization
  - Logical Optimization
  - Common Transformations
- Real-life example

Trivia

- How many joins evaluated for this SQL? 12c

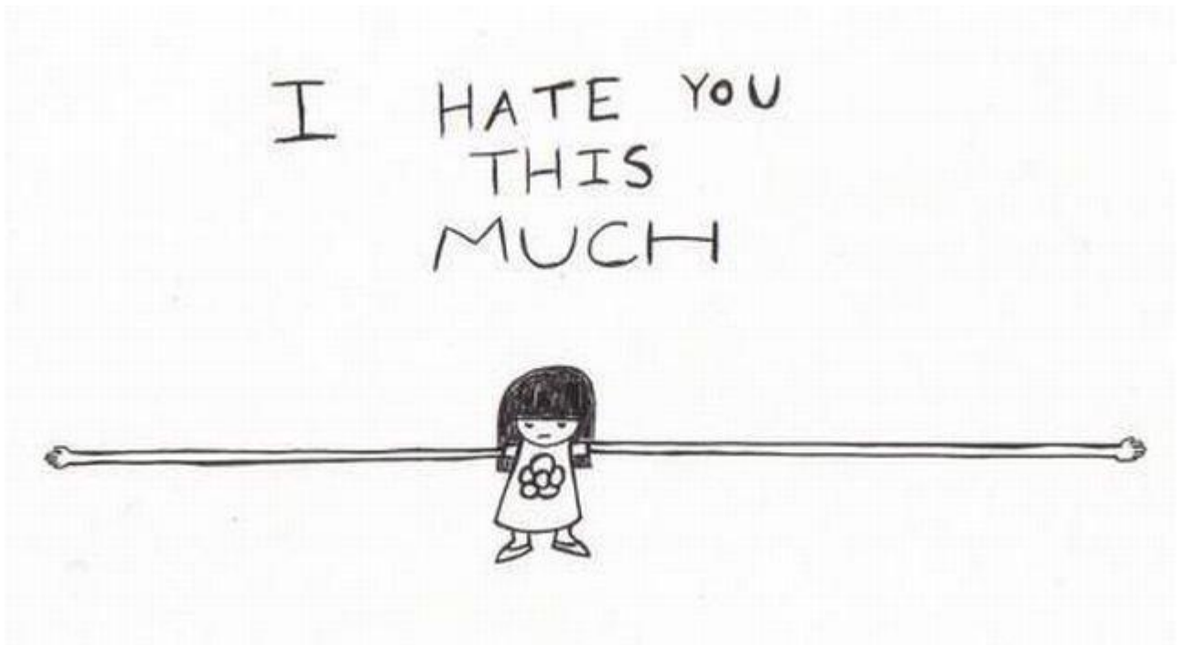
```
SELECT e1.email, jh.job_id
  FROM employees e1,
       job_history jh
 WHERE e1.employee_id = jh.employee_id
   AND jh.start_date > '01-JAN-01'
   AND e1.salary > (SELECT /*+ QB_NAME(SQ1) */ AVG(e2.salary)
                      FROM employees e2
                     WHERE e1.department_id = e2.department_id)
   AND e1.department_id IN (SELECT /*+ QB_NAME(SQ2) */ d.department_id
                              FROM departments d,
                                   locations l
                             WHERE d.location_id = l.location_id
                               AND l.country_id = 'US')
```

```
grep -c 'Join order\[' cdb1_ora_5541.trc
198
```

A day in the life of the optimizer

- Generate an optimal plan
- Multiple challenges
  - Partial knowledge of data and query (stats/binds)
  - Short amount of time (ms)
  - Use as little CPU and memory as possible

Hated and blamed by everybody

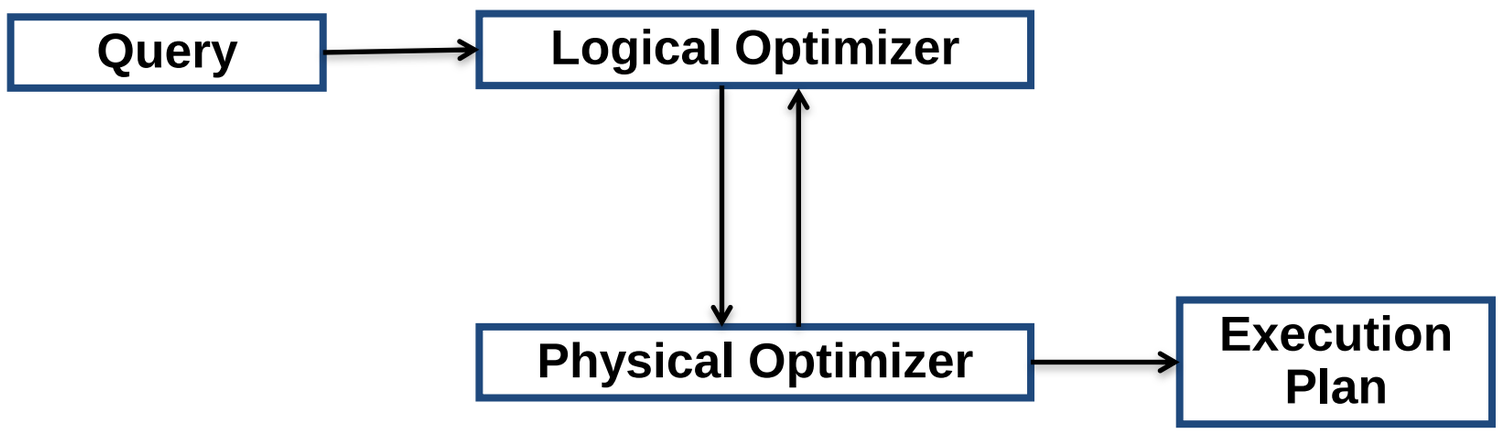


How to make it out alive?

- Play it smart!
- Two phases optimization
  - Logical and Physical
- Logical phase transforms SQL
  - Trying to find potentially better forms
  - Open the door to better plans
- Physical phase
  - Identify optimal access to objects involved

Working together

- Logical opt output input for Physical opt
- Multiple iterations back and forth



Query Block

- SQL can be complex
  - Made of smaller SQLs (each is a QB)
  - QBs correlate to each other
- QB is optimizer unit of optimization
  - Logical opt “reshapes” query blocks
  - Physical opt find optimal way to execute each QB

Before we move on

DISCLAIMER:

DON’T TRUST MAURO

(or anybody else)

Ask for evidence or test it yourself, always!

How many QBs?

```
create table t1 as select * from dba_objects;
create table t2 as select * from dba_objects;

select owner, object_name
  from t1
 where last_ddl_time >
    (select median(last_ddl_time)
       from t2
      where t1.owner = t2.owner);
```

```
Registered qb: SEL$1 0x71b6df90 (PARSER)
---------------------
QUERY BLOCK SIGNATURE
---------------------
  signature (): qb_name=SEL$1 nbfros=1 flg=0
    fro(0): flg=4 objn=24765 hint_alias="T1"@"SEL$1"

Registered qb: SEL$2 0x71b5e260 (PARSER)
---------------------
QUERY BLOCK SIGNATURE
---------------------
  signature (): qb_name=SEL$2 nbfros=1 flg=0
    fro(0): flg=4 objn=24766 hint_alias="T2"@"SEL$2"
```

How many QBs?

```
create table t1 as select * from dba_objects;
create table t2 as select * from dba_objects;

select *
  from t1
 union all
select *
  from t2;
```

```
Registered qb: SEL$1 0x719d65b8 (PARSER)
---------------------
QUERY BLOCK SIGNATURE
---------------------
  signature (): qb_name=SEL$1 nbfros=1 flg=0
    fro(0): flg=4 objn=24765 hint_alias="T1"@"SEL$1"

Registered qb: SET$1 0x719c6610 (PARSER)
---------------------
QUERY BLOCK SIGNATURE
---------------------
  signature (): qb_name=SET$1 nbfros=1 flg=0
    fro(0): flg=0 objn=0 hint_alias="NULL_HALIAS"@"SET$1"

Registered qb: SEL$2 0x719c6c68 (PARSER)
---------------------
QUERY BLOCK SIGNATURE
---------------------
  signature (): qb_name=SEL$2 nbfros=1 flg=0
    fro(0): flg=4 objn=24766 hint_alias="T2"@"SEL$2"
```

Physical optimizer

- Been around for a long time
  - The one to always get blamed (*)
  - Lots of literature about it
- Goal: find optimal exec plan for QB
  - The “calculator” part of the optimizer
  - Identify driver table and access method for it
  - Identify optimal join order and join methods

How does it look in 10053?

- Roughly between these two text blocks

```
---------------------
QUERY BLOCK SIGNATURE
---------------------
signature (optimizer): qb_name=SEL$683B0107 nbfros=1 flg=0
  fro(0): flg=0 objn=24766 hint_alias="T2"@"SEL$2”
----------------------------
SYSTEM STATISTICS INFORMATION
.....
.....
.....
Final cost for query block SEL$683B0107 (#2) - All Rows Plan:
  Best join order: 1
  Cost: 80.6652  Degree: 1  Card: 20353.0000  Bytes: 284942
  Resc: 80.6652  Resc_io: 79.0000  Resc_cpu: 52711834
  Resp: 80.6652  Resp_io: 79.0000  Resc_cpu: 52711834
```

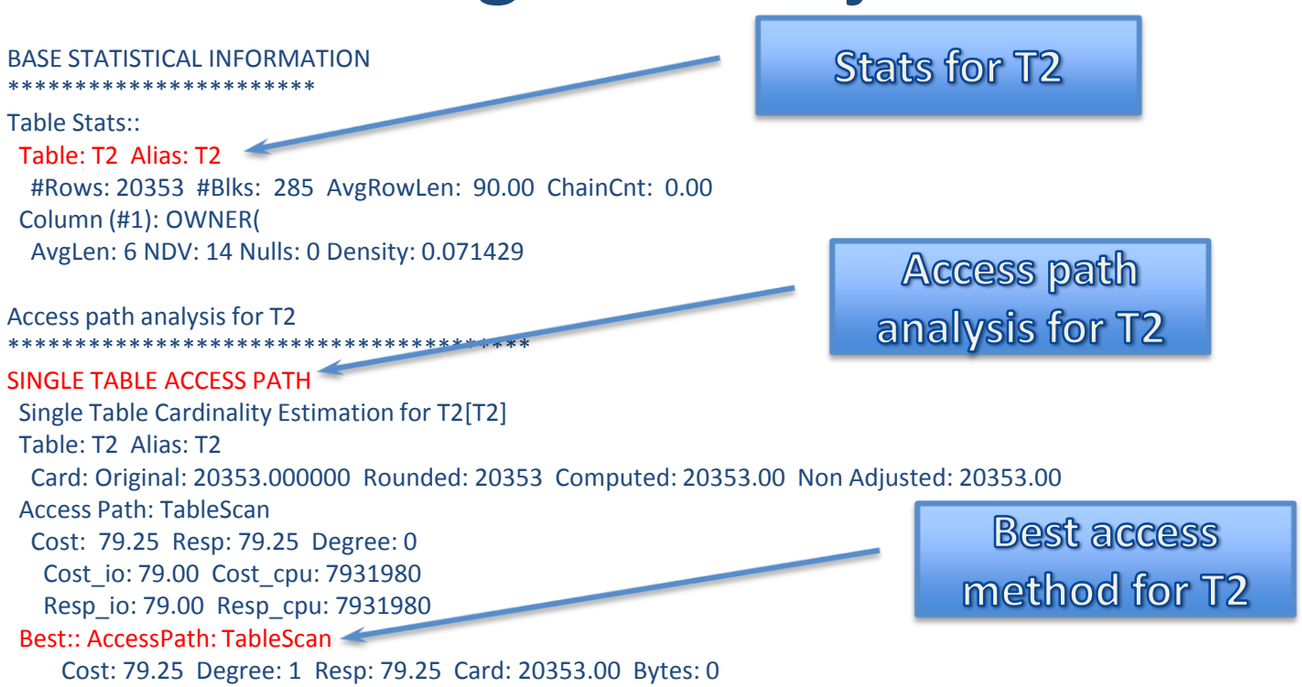
QB to optimize

All the calculations are performed here

Best join order identified and cost

How does it look in 10053? (2)

- Statistics report objects in QB
- Best access path per object identified
  - At this stage the objects are disconnected



How does it look in 10053? (3)

- Joins evaluated and cheapest one selected

```
Permutations for Starting Table :0
Join order[1]:  T1[T1]#0  T2[T2]#1
***************
Now joining: T2[T2]#1
***************
…
************************
Best so far:  Table#: 0  cost: 79.2506  card: 20352.0000  bytes: 936192
        Table#: 1  cost: 164297.6779  card: 29587446.8571  bytes: 1775246820
************************
Join order[2]:  T2[T2]#1  T1[T1]#0
***************
Now joining: T1[T1]#0
***************
….
Join order aborted: cost > best plan cost
```

First join considered T1->T2

Best join so far

Second join considered T2->T1

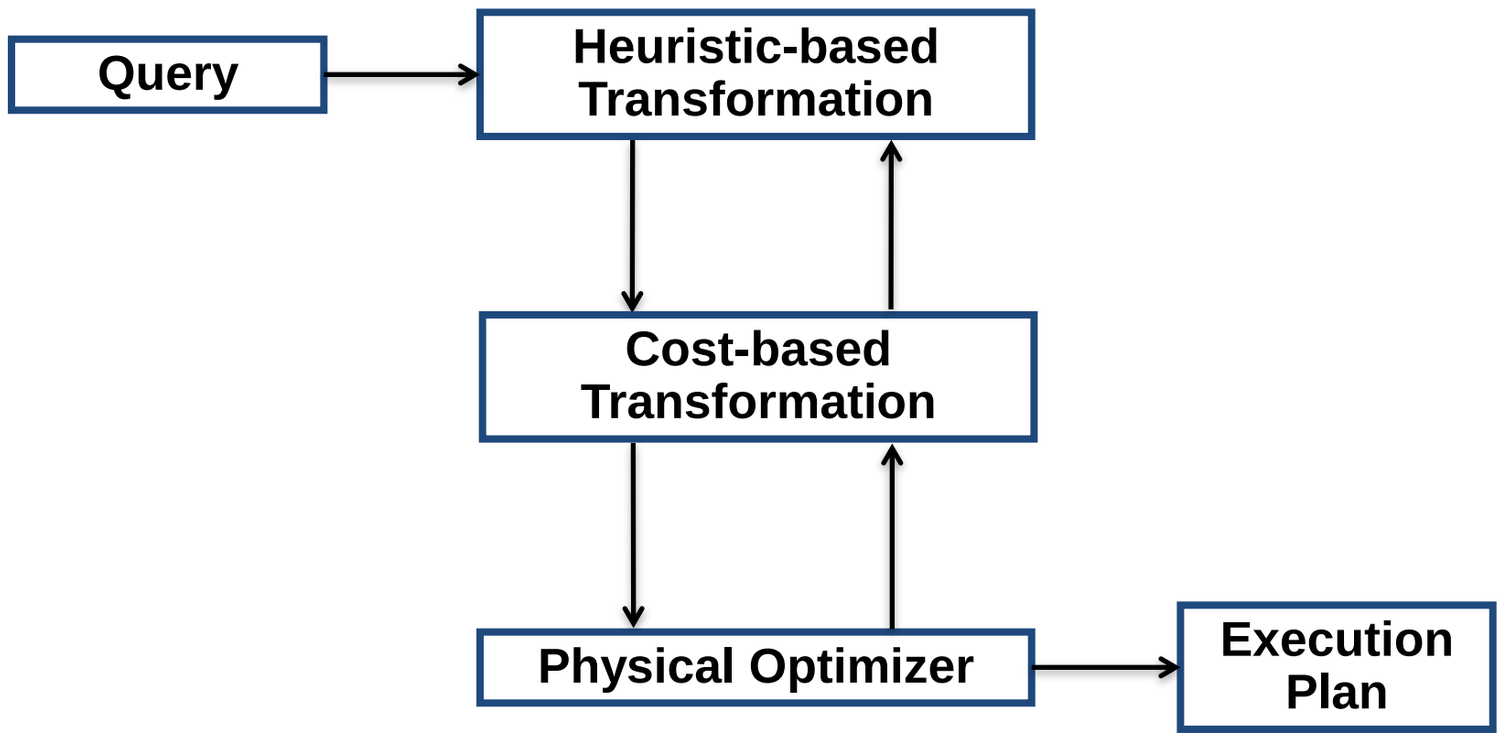
More expensive then best so discarded

Logical Optimizer

- The “brain” of the optimizer
- Applies transformations to SQL
  - Multiple transformations per QB
  - **MUST** keep semantics of query
- Use two approaches
  - Heuristic-based trans -> always beneficial
  - Cost-based trans -> “it depends” beneficial

Working together (2)

- Heuristic-b and cost-b work together
- Physical optimizer works on their output



Heuristic-based transformations

- Apply those changes always beneficial
  - Filter and project as soon as possible
  - Eliminate redundant operations
  - Reduce number of QBs (merge)
    - Less QBs means more permutations per QB
- Applied before cost-based
  - And sometime after cost-based too

Cost-based transformations

- Require costing to determine if beneficial
  - Physical optimizer costs with trans ON/OFF, cheapest wins
- CBQT Framework
  - Orchestrates multiple transformations
    - Interleaved (one on top of another)
    - Juxtaposed (either one or another)
  - Allows new transformations to be plugged in
  - Time savvy (limit search space, annotations, etc)

How many transformations?

- **A LOT!**
- Can change plan very little and/or very much
- Dependent on coding style
- Some are very common
  - Subquery unnesting (heuristic and cost-based)
  - View merging (heuristic and cost-based)
  - Join Predicate Push Down (mostly cost-based today)

Subquery Unnesting

```
select *
  from t1
 where object_id in (
   select object_id
     from t2);
```

```
---------------------------------
| Id  | Operation          | Name |
---------------------------------
|   0 | SELECT STATEMENT   |      |
|*  1 |  FILTER            |      |
|   2 |   TABLE ACCESS FULL| T1   |
|*  3 |   TABLE ACCESS FULL| T2   |
```

Easy to spot
FILTER operation is gone

```
select *
  from t1, t2
 where t1.object_id s=
       t2.object_id;
```

```
---------------------------------
| Id  | Operation          | Name |
---------------------------------
|   0 | SELECT STATEMENT   |      |
|*  1 |  HASH JOIN RIGHT SEMI|    |
|   2 |   TABLE ACCESS FULL  | T2   |
|   3 |   TABLE ACCESS FULL  | T1   |
```

View Merging

```
select t1.*
  from t1,
       (select object_id
          from t2) t2
 where t1.object_id =
       t2.object_id;
```

```
select t1.*
  from t1, t2
 where t1.object_id =
       t2.object_id;
```

Easy to spot
VIEW operation is gone

```
-----------------------------------
| Id  | Operation          | Name |
-----------------------------------
|   0 | SELECT STATEMENT    |      |
|*  1 |  HASH JOIN          |      |
|   2 |   VIEW              |      |
|   3 |    TABLE ACCESS FULL| T2   |
|   4 |   TABLE ACCESS FULL | T1   |
```

```
-------------------------------------+
| Id  | Operation          | Name    |
-------------------------------------+
| 0   | SELECT STATEMENT    |         |
| 1   |  HASH JOIN          |         |
| 2   |   TABLE ACCESS FULL | T2      |
| 3   |   TABLE ACCESS FULL | T1      |
```

Join Predicate Push Down

```
select t1.*
  from t1,
       (select object_id
          from t2) t2
 where t1.object_id =
       t2.object_id(+);
```

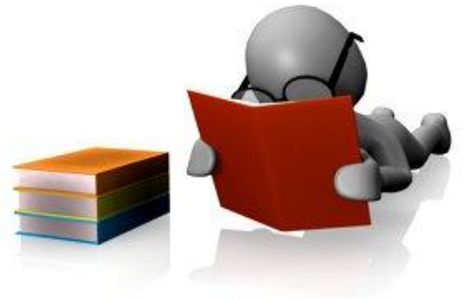
```
-------------------------------------
| Id  | Operation           | Name |
-------------------------------------
|   0 | SELECT STATEMENT    |      |
|*  1 |  HASH JOIN RIGHT OUTER|    |
|   2 |   VIEW              |      |
|   3 |    TABLE ACCESS FULL | T2  |
|   4 |   TABLE ACCESS FULL  | T1  |
```

Easy to spot
VIEW operation shows
PUSH occurred

```
select t1.*
  from t1,
       (select object_id
          from t2
               t1.object_id =
               object_id(+)) t2
```

```
-------------------------------------
| Id  | Operation           | Name |
-------------------------------------
|   0 | SELECT STATEMENT    |      |
|   1 |  NESTED LOOPS OUTER |      |
|   2 |   TABLE ACCESS FULL | T1   |
|   3 |   VIEW PUSHED PREDICATE |  |
|*  4 |    TABLE ACCESS FULL | T2  |
```

So far we learned



- CBO life is tough ☺
- SQL is split and optimized in chunks (QBs)
- Logical Optim modifies QBs (1+)
- Physical Optim costs changed QBs (1)
- Cheapest plan (Logical + Physical) wins
- Some transformations are common
- It all sounds pretty simple, right?

How does it look in 10053?

- This is where things get tricky
  - No way out, 10053 is source of truth
- Average SQL produces 100k/250k+ rows trace
  - Trace not designed to be summarized
  - Supposed to be READ by you
  - Very very verbose (and sometimes not enough ☹)
- We need a plan!
- Easier to show with an example

Let the fun begin

Objects from HR sample schema

```
SELECT e1.email, jh.job_id
  FROM employees e1,
       job_history jh
 WHERE e1.employee_id = jh.employee_id
   AND jh.start_date > '01-JAN-01'
   AND e1.salary > (SELECT AVG(e2.salary)
                      FROM employees e2
                     WHERE e1.department_id = e2.department_id)
   AND e1.department_id IN (SELECT d.department_id
                              FROM departments d,
                                   locations l
                             WHERE d.location_id = l.location_id
                               AND l.country_id = 'US’)
```

This is the final plan

```
-------------------------------------------------------------------------
| Id  | Operation                     | Name                 |
-------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |                      |
|*  1 |  FILTER                       |                      |
|*  2 |   HASH JOIN                   |                      |
|*  3 |    TABLE ACCESS BY INDEX ROWID BATCHED| JOB_HISTORY          |
|*  4 |     INDEX SKIP SCAN           | JHIST_EMP_ID_ST_DATE_PK |
|   5 |    TABLE ACCESS FULL          | EMPLOYEES            |
|   6 |   NESTED LOOPS                |                      |
|   7 |    TABLE ACCESS BY INDEX ROWID| DEPARTMENTS          |
|*  8 |     INDEX UNIQUE SCAN         | DEPT_ID_PK           |
|*  9 |    TABLE ACCESS BY INDEX ROWID| LOCATIONS            |
|* 10 |     INDEX UNIQUE SCAN         | LOC_ID_PK            |
|  11 |   SORT AGGREGATE              |                      |
|  12 |    TABLE ACCESS BY INDEX ROWID BATCHED| EMPLOYEES            |
|* 13 |     INDEX RANGE SCAN          | EMP_DEPARTMENT_IX    |
-------------------------------------------------------------------------
```

No, it’s this one

```
-------------------------------------------------------------------------+
| Id  | Operation                          | Name                 |
-------------------------------------------------------------------------+
| 0   | SELECT STATEMENT                   |                      |
| 1   |  NESTED LOOPS                      |                      |
| 2   |   NESTED LOOPS SEMI                |                      |
| 3   |    NESTED LOOPS                    |                      |
| 4   |     TABLE ACCESS BY INDEX ROWID BATCHED  | JOB_HISTORY          |
| 5   |      INDEX SKIP SCAN               | JHIST_EMP_ID_ST_DATE_PK|
| 6   |     TABLE ACCESS BY INDEX ROWID    | EMPLOYEES            |
| 7   |      INDEX UNIQUE SCAN             | EMP_EMP_ID_PK        |
| 8   |    VIEW PUSHED PREDICATE           | VW_NSO_2             |
| 9   |     NESTED LOOPS                   |                      |
| 10  |      TABLE ACCESS BY INDEX ROWID   | DEPARTMENTS          |
| 11  |       INDEX UNIQUE SCAN            | DEPT_ID_PK           |
| 12  |      TABLE ACCESS BY INDEX ROWID   | LOCATIONS            |
| 13  |       INDEX UNIQUE SCAN            | LOC_ID_PK            |
| 14  |    VIEW PUSHED PREDICATE           | VW_SQ_1              |
| 15  |     FILTER                         |                      |
| 16  |      SORT AGGREGATE                |                      |
| 17  |       TABLE ACCESS BY INDEX ROWID BATCHED | EMPLOYEES            |
| 18  |        INDEX RANGE SCAN            | EMP_DEPARTMENT_IX    |
-------------------------------------------------------------------------+
```

Sorry, it’s this one

```
-------------------------------------------------------------------------------
| Id  | Operation                            | Name                 |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                     |                      |
|*  1 |  FILTER                              |                      |
|   2 |   NESTED LOOPS SEMI                  |                      |
|   3 |    NESTED LOOPS                      |                      |
|*  4 |     TABLE ACCESS BY INDEX ROWID BATCHED | JOB_HISTORY          |
|*  5 |      INDEX SKIP SCAN                 | JHIST_EMP_ID_ST_DATE_PK |
|   6 |     TABLE ACCESS BY INDEX ROWID      | EMPLOYEES            |
|*  7 |      INDEX UNIQUE SCAN               | EMP_EMP_ID_PK        |
|   8 |    VIEW PUSHED PREDICATE             | VW_NSO_1             |
|   9 |     NESTED LOOPS                     |                      |
|  10 |      TABLE ACCESS BY INDEX ROWID     | DEPARTMENTS          |
|* 11 |       INDEX UNIQUE SCAN              | DEPT_ID_PK           |
|* 12 |      TABLE ACCESS BY INDEX ROWID BATCHED| LOCATIONS            |
|* 13 |       INDEX RANGE SCAN               | LOC_COUNTRY_IX       |
|  14 |   SORT AGGREGATE                     |                      |
|  15 |    TABLE ACCESS BY INDEX ROWID BATCHED  | EMPLOYEES            |
|* 16 |     INDEX RANGE SCAN                 | EMP_DEPARTMENT_IX    |
-------------------------------------------------------------------------------
```

Ops, it’s this one, I swear!

```
-------------------------------------------------------------------------------
| Id  | Operation                           | Name                  |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |                       |
|*  1 |  FILTER                             |                       |
|   2 |   NESTED LOOPS                      |                       |
|   3 |    NESTED LOOPS                     |                       |
|   4 |     NESTED LOOPS                    |                       |
|   5 |      VIEW                           | VW_NSO_1              |
|   6 |       HASH UNIQUE                   |                       |
|   7 |        NESTED LOOPS SEMI            |                       |
|   8 |         VIEW                        | index$_join$_004      |
|*  9 |          HASH JOIN                  |                       |
|  10 |           INDEX FAST FULL SCAN      | DEPT_ID_PK            |
|  11 |           INDEX FAST FULL SCAN      | DEPT_LOCATION_IX      |
|* 12 |         TABLE ACCESS BY INDEX ROWID BATCHED| LOCATIONS          |
|* 13 |          INDEX RANGE SCAN           | LOC_COUNTRY_IX        |
|  14 |      TABLE ACCESS BY INDEX ROWID BATCHED   | EMPLOYEES          |
|* 15 |       INDEX RANGE SCAN              | EMP_DEPARTMENT_IX     |
|* 16 |     INDEX RANGE SCAN                | JHIST_EMP_ID_ST_DATE_PK |
|* 17 |    TABLE ACCESS BY INDEX ROWID      | JOB_HISTORY           |
|  18 |   SORT AGGREGATE                    |                       |
|  19 |    TABLE ACCESS BY INDEX ROWID BATCHED     | EMPLOYEES          |
|* 20 |     INDEX RANGE SCAN                | EMP_DEPARTMENT_IX     |
-------------------------------------------------------------------------------
```

Recap

What did I say before?

DON’T TRUST MAURO

(or anybody else)

Let’s look at 10053 and stop guessing!

Starting point

Objects from HR sample schema

```
SELECT e1.email, jh.job_id
  FROM employees e1,
       job_history jh
 WHERE e1.employee_id = jh.employee_id
   AND jh.start_date > '01-JAN-01'
   AND e1.salary > (SELECT AVG(e2.salary)
                      FROM employees e2
                     WHERE e1.department_id = e2.department_id)
   AND e1.department_id IN (SELECT d.department_id
                              FROM departments d,
                                   locations l
                             WHERE d.location_id = l.location_id
                               AND l.country_id = 'US')
```

45k 10053 trace file

How many QBs?

```
Registered qb: SEL$1 0x4cf0e098 (PARSER)
signature (): qb_name=SEL$1 nbfros=2 flg=0
    fro(0): flg=4 objn=96089 hint_alias="E1"@"SEL$1"
    fro(1): flg=4 objn=96093 hint_alias="JH"@"SEL$1"

Registered qb: SEL$2 0x4cf04f00 (PARSER)
signature (): qb_name=SEL$2 nbfros=1 flg=0
    fro(0): flg=4 objn=96089 hint_alias="E2"@"SEL$2"

Registered qb: SEL$3 0x4cf03f60 (PARSER)
signature (): qb_name=SEL$3 nbfros=2 flg=0
    fro(0): flg=4 objn=96084 hint_alias="D"@"SEL$3"
    fro(1): flg=4 objn=96081 hint_alias="L"@"SEL$3”
```

- Think of 3 QBs as 3 pieces of LEGO
- Transformed and cost N times

Which transformations applied?

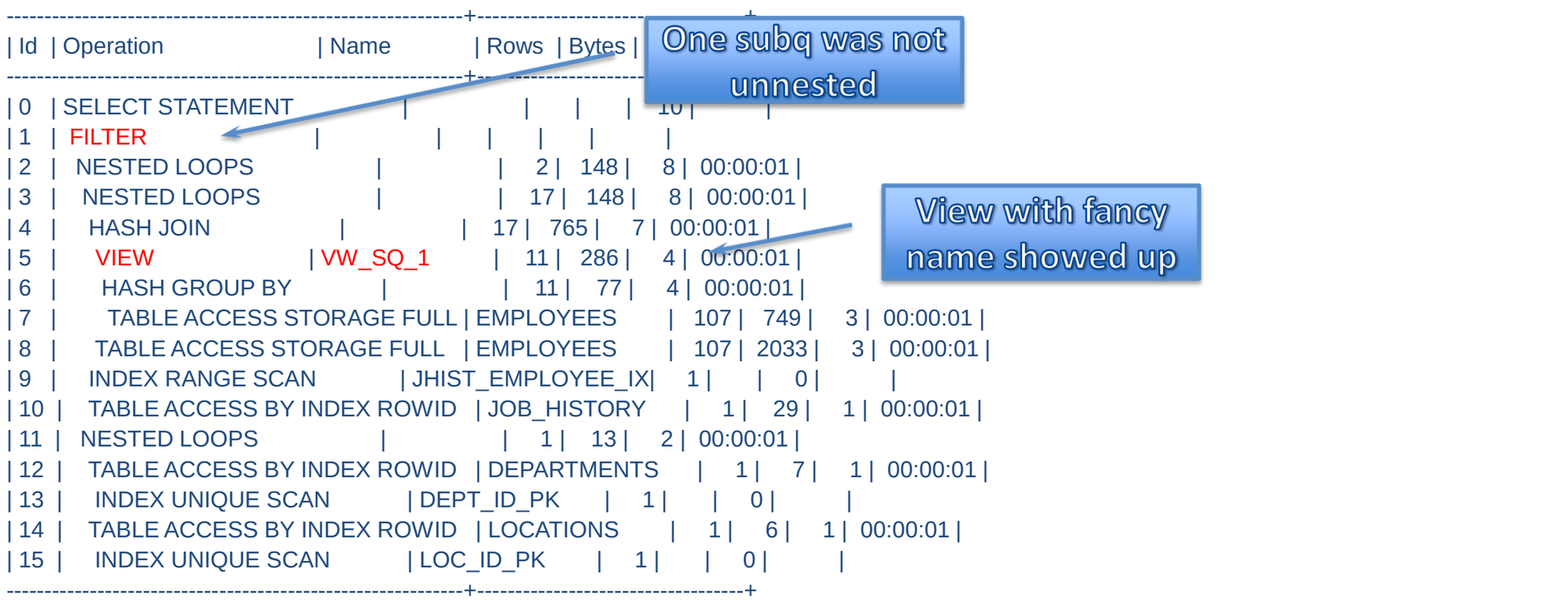
- QB Registry in 10053 tells you the story
- All (most) of the trans for each QB
  - Includes name of new QBs
  - Indentation for parent/child relationship
  - QBs part of best plan have [FINAL]
  - QBs at the same level are crossroads
- Used as a map to chase CBO actions
  - Enable specific searches in 10053

Query Block Registry

- Each QB dumped with transformations

```
Query Block Registry:
SEL$3 0x4c50a700 (PARSER) [FINAL]
  SEL$8771BF6C 0x0 (SUBQUERY UNNEST SEL$1; SEL$3;)
    SEL$7C12A527 0x0 (COMPLEX SUBQUERY UNNEST SEL$8771BF6C)
    SEL$5DB0472E 0x0 (SPLIT/MERGE QUERY BLOCKS SEL$8771BF6C)
    SEL$2AD7F9D9 0x0 (PUSHED PREDICATE SEL$291F8F59; SEL$8771BF6C; "VW_NSO_11"@"SEL$8771BF6C" 4)
  SEL$2E20A9F9 0x0 (SUBQUERY UNNEST SEL$7511BFD2; SEL$3; SEL$2;)
    SEL$CC348667 0x0 (COMPLEX SUBQUERY UNNEST SEL$2E20A9F9)
  SEL$291F8F59 0x0 (SUBQ INTO VIEW FOR COMPLEX UNNEST SEL$3)
    SEL$555A942D 0x0 (VIEW MERGE SEL$C149BB3C; SEL$291F8F59)
    SEL$C149BB3C 0x0 (PROJECTION VIEW FOR CVM SEL$291F8F59)
      SEL$555A942D 0x0 (VIEW MERGE SEL$C149BB3C; SEL$291F8F59)
    SEL$2AD7F9D9 0x0 (PUSHED PREDICATE SEL$291F8F59; SEL$8771BF6C; "VW_NSO_11"@"SEL$8771BF6C" 4)
SEL$2 0x4c50b6a0 (PARSER)
  SEL$C772B8D1 0x4c50ff78 (SUBQUERY UNNEST SEL$7511BFD2; SEL$2) [FINAL]
    SEL$841DDE77 0x0 (VIEW MERGE SEL$C772B8D1; SEL$683B0107)
    SEL$C6423BE4 0x0 (PUSHED PREDICATE SEL$683B0107; SEL$C772B8D1; "VW_SQ_1"@"SEL$7511BFD2" 4)
  SEL$2E20A9F9 0x0 (SUBQUERY UNNEST SEL$7511BFD2; SEL$3; SEL$2;)
    ...
  SEL$683B0107 0x4c50b6a0 (SUBQ INTO VIEW FOR COMPLEX UNNEST SEL$2) [FINAL]
    SEL$841DDE77 0x0 (VIEW MERGE SEL$C772B8D1; SEL$683B0107)
    SEL$C6423BE4 0x0 (PUSHED PREDICATE SEL$683B0107; SEL$C772B8D1; "VW_SQ_1"@"SEL$7511BFD2" 4)
SEL$1 0x4c50ff78 (PARSER)
  SEL$8771BF6C 0x0 (SUBQUERY UNNEST SEL$1; SEL$3;)
    ...
  SEL$7511BFD2 0x4c50ff78 (VIEW ADDED SEL$1)
    SEL$C772B8D1 0x4c50ff78 (SUBQUERY UNNEST SEL$7511BFD2; SEL$2) [FINAL]
      ...
    SEL$2E20A9F9 0x0 (SUBQUERY UNNEST SEL$7511BFD2; SEL$3; SEL$2;)
      ...
```

Final plan is



```
-----------------------------------------------------+-----------------------------------+
| Id  | Operation                    | Name          | Rows  | Bytes |  Cost (%CPU)| Time     |
-----------------------------------------------------+-----------------------------------+
| 0   | SELECT STATEMENT             |               |       |       |    10 |             |
| 1   |  FILTER                      |               |       |     |     |     |         |
| 2   |   NESTED LOOPS               |               |     2 |   148 |     8 |  00:00:01 |
| 3   |    NESTED LOOPS              |               |    17 |   148 |     8 |  00:00:01 |
| 4   |     HASH JOIN                |               |    17 |   765 |     7 |  00:00:01 |
| 5   |      VIEW                    | VW_SQ_1       |    11 |   286 |     4 |  00:00:01 |
| 6   |       HASH GROUP BY          |               |    11 |    77 |     4 |  00:00:01 |
| 7   |        TABLE ACCESS STORAGE FULL | EMPLOYEES     |   107 |   749 |     3 |  00:00:01 |
| 8   |      TABLE ACCESS STORAGE FULL   | EMPLOYEES     |   107 |  2033 |     3 |  00:00:01 |
| 9   |     INDEX RANGE SCAN         | JHIST_EMPLOYEE_IX|     1 |       |     0 |           |
| 10  |    TABLE ACCESS BY INDEX ROWID   | JOB_HISTORY   |     1 |    29 |     1 |  00:00:01 |
| 11  |   NESTED LOOPS               |               |     1 |    13 |     2 |  00:00:01 |
| 12  |    TABLE ACCESS BY INDEX ROWID   | DEPARTMENTS   |     1 |     7 |     1 |  00:00:01 |
| 13  |     INDEX UNIQUE SCAN        | DEPT_ID_PK    |     1 |       |     0 |           |
| 14  |    TABLE ACCESS BY INDEX ROWID   | LOCATIONS     |     1 |     6 |     1 |  00:00:01 |
| 15  |     INDEX UNIQUE SCAN        | LOC_ID_PK     |     1 |       |     0 |           |
-----------------------------------------------------+-----------------------------------+
```


1 - filter(IS NOT NULL)
4 - access("E1"."DEPARTMENT_ID"="ITEM_1")
4 - filter("E1"."SALARY">"AVG(E2.SALARY)")
9 - access("E1"."EMPLOYEE_ID"="JH"."EMPLOYEE_ID")

10 - filter(("JH"."START_DATE">'01-JAN-01' AND "END_DATE">'01-JAN-01'))
13 - access("D"."DEPARTMENT_ID"=:B1)
14 - filter("L"."COUNTRY_ID"='US')
15 - access("D"."LOCATION_ID"="L"."LOCATION_ID")

How to attack 10053?

- Keep it simple
- Remember: multiple internal QB trans from A to B
- Ignore “noise”, focus on what you seek

```
> wc –l MAURRRO1_ora_119219_test.trc
44612 MAURRRO1_ora_119219_test.trc

> grep -nE '^[A-Z]{2,}:|Registe’ MAURRRO1_ora_119219_test.trc | grep -v 'CBRID…' | wc -l
377
```

- Use physical opt input QB as anchor
  - Last one before big jump in 10053
  - QB represent the valid status

Ok one step at a time

1930:SU: Unnesting query blocks in query block SEL$1 (#1) that are valid to unnest.
1935:SU: Considering subquery unnest on query block SEL$1 (#1).
1936:SU: Checking validity of unnesting subquery SEL$2 (#3)
1937:SU: Passed validity checks, but requires costing.
1938:SU: Checking validity of unnesting subquery SEL$3 (#2)
1939:SU: Passed validity checks, but requires costing.
1940:SU: Using search type: exhaustive
1941:SU: Starting iteration 1, state space = (2,3) : (1,1)

1945:Registered qb: SEL$683B0107 0x96361e50 (SUBQ INTO VIEW FOR COMPLEX UNNEST SEL$2)
1952:Registered qb: SEL$7511BFD2 0x962788a8 (VIEW ADDED SEL$1)
1962:Registered qb: SEL$291F8F59 0x96276b58 (SUBQ INTO VIEW FOR COMPLEX UNNEST SEL$3)
1970:Registered qb: SEL$2E20A9F9 0x962788a8 (SUBQUERY UNNEST SEL$7511BFD2; SEL$3; SEL$2;)

2002:SU: Costing transformed query.
2003:CBQT: Looking for cost annotations for query block SEL$683B0107, key = SEL$683B0107_00002002_4
2004:CBQT: Could not find stored cost annotations.
2005:CBQT: Looking for cost annotations for query block SEL$291F8F59, key = SEL$291F8F59_00002002_4
2006:CBQT: Could not find stored cost annotations.
…
5151:CBQT: Saved costed qb# 3 (SEL$683B0107), key = SEL$683B0107_00002002_4
5152:CBQT: Saved costed qb# 2 (SEL$291F8F59), key = SEL$291F8F59_00002002_4
5153:CBQT: Saved costed qb# 1 (SEL$2E20A9F9), key = SEL$2E20A9F9_00000000_0

SEL$2/3 valid to cost-based unnest

CBQT will use exhaustive approach First unnest both

Unnested SEL$2

Internal trans

Unnested SEL$3

SEL$1 with unnested

Physical Optimizer

Digging into 10053

- Use the QB name to search for costs

```
2160 Final cost for query block SEL$683B0107 (#3) - All Rows Plan:
2161   Best join order: 1
2162   Cost: 4.002661  Degree: 1  Card: 107.000000  Bytes: 749.000000

2646 Final cost for query block SEL$291F8F59 (#2) - All Rows Plan:
2647   Best join order: 1
2648   Cost: 3.218206  Degree: 1  Card: 14.000000  Bytes: 182.000000

2683 signature (optimizer): qb_name=SEL$2E20A9F9 nbfros=4 flg=0
2684   fro(0): flg=0 objn=91753 hint_alias="E1"@"SEL$1"
2685   fro(1): flg=0 objn=91757 hint_alias="JH"@"SEL$1"
2686   fro(2): flg=1 objn=0 hint_alias="VW_NSO_2"@"SEL$2E20A9F9"
2687   fro(3): flg=1 objn=0 hint_alias="VW_SQ_1"@"SEL$7511BFD2"
…
5140 Final cost for query block SEL$2E20A9F9 (#1) - All Rows Plan:
5141   Best join order: 4
5142   Cost: 11.366818  Degree: 1  Card: 2.000000  Bytes: 174.000000

5151:CBQT: Saved costed qb# 3 (SEL$683B0107), key = SEL$683B0107_00002002_4
5152:CBQT: Saved costed qb# 2 (SEL$291F8F59), key = SEL$291F8F59_00002002_4
5153:CBQT: Saved costed qb# 1 (SEL$2E20A9F9), key = SEL$2E20A9F9_00000000_0
```

Unnested SEL$2

Unnested SEL$3

SEL$1 with unnested

First interleaved transformation

<<skipped interleaved DP>>
8254:SU: Considering interleaved join pred push down
8255:SU: Unnesting subquery query block SEL$2 (#3)Subquery removal for query block SEL$2 (#3)
8258:SU: Transform an ANY subquery to semi-join or distinct.
8259:JPPD: Checking validity of push-down in query block SEL$2E20A9F9 (#1)
8260:JPPD: Checking validity of push-down from query block SEL$2E20A9F9 (#1) to query block SEL$291F8F59 (#2)
8262:JPPD: Passed validity checks
8263:JPPD: Checking validity of push-down from query block SEL$2E20A9F9 (#1) to query block SEL$683B0107 (#3)
8265:JPPD: Passed validity checks
8266:JPPD: JPPD: Pushdown from query block SEL$2E20A9F9 (#1) passed validity checks.
8268:JPPD: Using search type: linear
8269:JPPD: Considering join predicate push-down
8270:JPPD: Starting iteration 1, state space = (2,3) : (0,0)
<<copy QB here>>
8286:JPPD: Performing join predicate push-down (no transformation phase) from query block SEL$2E20A9F9 (#1) to query block SEL$291F8F59 (#2)
8288:JPPD: Performing join predicate push-down (no transformation phase) from query block SEL$2E20A9F9 (#1) to query block SEL$683B0107 (#3)
8312:JPPD: Costing transformed query.
…
11461:CBQT: Saved costed qb# 3 (SEL$683B0107), key = SEL$683B0107_00042002_4
11462:CBQT: Saved costed qb# 2 (SEL$291F8F59), key = SEL$291F8F59_00042002_4
11463:CBQT: Saved costed qb# 1 (SEL$2E20A9F9), key = SEL$2E20A9F9_00000000_0

SEL$2/3 valid to cost-based JPPD

CBQT will use linear approach
First no JPPD both

Physical Optimizer

Digging into 10053

- Use the QB name to search for costs

```
8470 Final cost for query block SEL$683B0107 (#3) - All Rows Plan:
8471   Best join order: 1
8472   Cost: 4.002661  Degree: 1  Card: 107.000000  Bytes: 749.000000

8956 Final cost for query block SEL$291F8F59 (#2) - All Rows Plan:
8957   Best join order: 1
8958   Cost: 3.218206  Degree: 1  Card: 14.000000  Bytes: 182.000000

8993 signature (optimizer): qb_name=SEL$2E20A9F9 nbfros=4 flg=0
8994   fro(0): flg=0 objn=91753 hint_alias="E1"@"SEL$1"
8995   fro(1): flg=0 objn=91757 hint_alias="JH"@"SEL$1"
8996   fro(2): flg=1 objn=0 hint_alias="VW_NSO_6"@"SEL$2E20A9F9"
8997   fro(3): flg=1 objn=0 hint_alias="VW_SQ_5"@"SEL$7511BFD2"
…
11450 Final cost for query block SEL$2E20A9F9 (#1) - All Rows Plan:
11451   Best join order: 4
11452   Cost: 11.366818  Degree: 1  Card: 2.000000  Bytes: 174.000000

11461:CBQT: Saved costed qb# 3 (SEL$683B0107), key = SEL$683B0107_00042002_4
11462:CBQT: Saved costed qb# 2 (SEL$291F8F59), key = SEL$291F8F59_00042002_4
11463:CBQT: Saved costed qb# 1 (SEL$2E20A9F9), key = SEL$2E20A9F9_00000000_0
11464:JPPD: Updated best state, Cost = 11.366818
```

SU + NO JPPD SEL$2

SU+ NO JPPD SEL$3

SEL$1 with SU + NO JPPD
Same cost as just SU

Keeping track of cost

More JPPD iterations

```
11465:JPPD: Starting iteration 2, state space = (2,3) : (1,0)
11466:JPPD: Performing join predicate push-down (candidate phase) from query block SEL$2E20A9F9 (#1) to query block SEL$291F8F
11467:JPPD:   Pushing predicate "E1"."DEPARTMENT_ID"="VW_NSO_6"."DEPARTMENT_ID"
11469:JPPD: Push dest of pred 0x7fd396030c68 is qb 0x7fd396033fb8:query block SEL$291F8F59 (#2)
11471:JPPD: Performing join predicate push-down (no transformation phase) from query block SEL$2E20A9F9 (#1) to query block SEL$683B0107 (#3)
11503:JPPD: Costing transformed query.
12363:CBQT: Looking for cost annotations for query block SEL$683B0107, key = SEL$683B0107_00002002_4
12364:CBQT: Replaced cost annotations in query block SEL$683B0107.
12365:CBQT: Looking for cost annotations for query block SEL$2AD7F9D9, key = SEL$2AD7F9D9_00082212_4
12366:CBQT: Could not find stored cost annotations.
…
13907:CBQT: Saved costed qb# 3 (SEL$683B0107), key = SEL$683B0107_00002002_4
13908:CBQT: Saved costed qb# 1 (SEL$2E20A9F9), key = SEL$2E20A9F9_00100200_0
13909:JPPD: Not update best state, Cost = 13.036576

…
13910:JPPD: Starting iteration 3, state space = (2,3) : (0,1)
13952:JPPD: Costing transformed query.
13953:CBQT: Looking for cost annotations for query block SEL$C6423BE4, key = SEL$C6423BE4_00082212_4
13954:CBQT: Could not find stored cost annotations.
13955:CBQT: Looking for cost annotations for query block SEL$291F8F59, key = SEL$291F8F59_00002002_4
13956:CBQT: Replaced cost annotations in query block SEL$291F8F59.
…
15338:JPPD: Not update best state, Cost = 28.256163
```

Iteration #2 JPPD in SEL$3 but not SEL$2

Cost for SU + NO JPPD for SEL$2 is know so skip it

Physical Optimizer

Higher cost, trashed

Iteration #3 JPPD in SEL$2 but not SEL$3

Higher cost, trashed

Done with JPPD, moving on

15339:JPPD: Will not use JPPD from query block SEL$2E20A9F9 (#1)
15341:SU: Rejected interleaved query.
15342:SU: Finished interleaved join pred push down
15343:SU: Updated best state, Cost = 11.366818
15344:Registered qb: SEL$CC348667 0x962788a8 (COMPLEX SUBQUERY UNNEST SEL$2E20A9F9)
15354:SU: Starting iteration 2, state space = (2,3) : (1,0)
15356:Registered qb: SEL$8771BF6C 0x9607ed38 (SUBQUERY UNNEST SEL$1; SEL$3;)
15390:SU: Costing transformed query.
…
16988:SU: Considering interleaved complex view merging
16990:CVM: Considering view merge (candidate phase) in query block SEL$8771BF6C (#1)
16993:CVM: Considering view merge (candidate phase) in query block SEL$291F8F59 (#2)
16996:CVM: CBQT Marking query block SEL$291F8F59 (#2) as valid for CVM.
16997:CVM: Merging complex view SEL$291F8F59 (#2) into SEL$8771BF6C (#1).
17002:Registered qb: SEL$8771BF6C 0x96096420 (COPY SEL$8771BF6C)
17007:Registered qb: SEL$5DB0472E 0x96075e60 (SPLIT/MERGE QUERY BLOCKS SEL$8771BF6C)
17014:Registered qb: SEL$C149BB3C 0x96096420 (PROJECTION VIEW FOR CVM SEL$291F8F59)
17041:Registered qb: SEL$555A942D 0x96096420 (VIEW MERGE SEL$C149BB3C; SEL$291F8F59)
17075:SU: Costing transformed query.
…
20976:SU: Interleaved cost better than best so far.

JPPD not a good idea

Second iteration of parent SU

Physical Optimizer for just SU

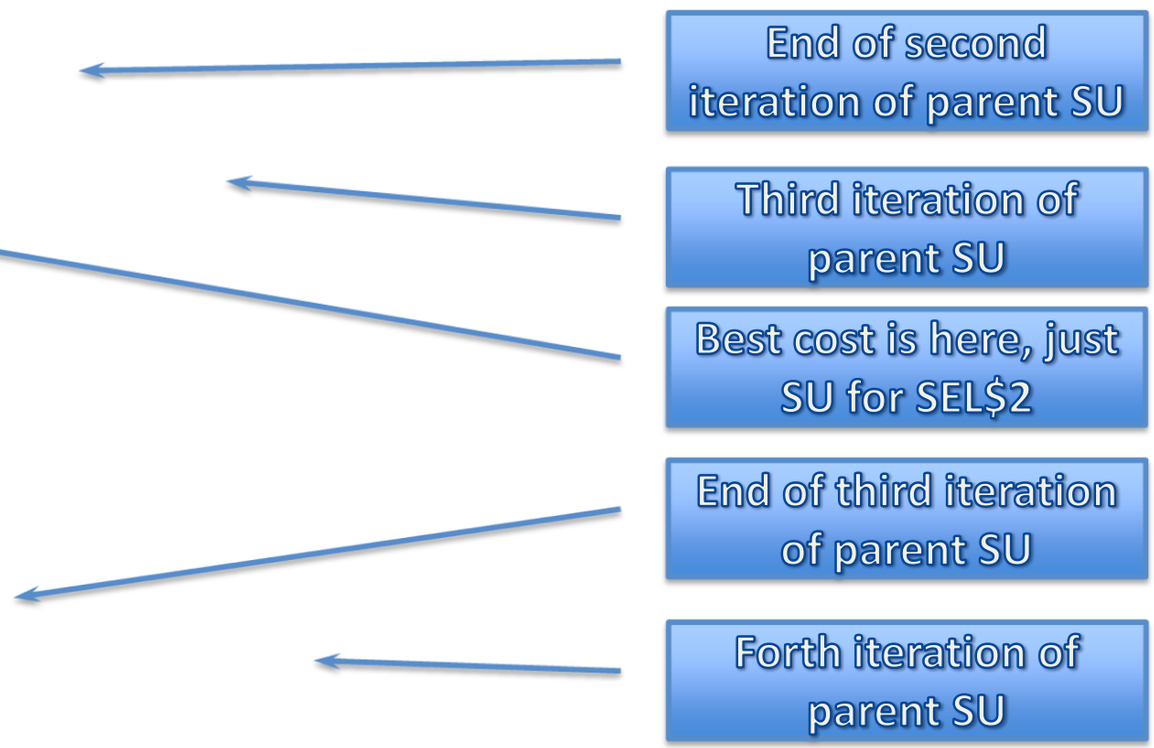
Considering CVM

Physical Optimizer

Better cost within this parent iteration

Flash forward

20977:SU: Finished interleaved complex view merging
20978:SU: Considering interleaved distinct placement
…
22396:SU: Rejected interleaved distinct placement.
22397:SU: Finished interleaved distinct placement
22398:SU: Considering interleaved join pred push down
…
25631:SU: Rejected interleaved query.
25632:SU: Finished interleaved join pred push down
25633:SU: Not update best state, Cost = 14.206022
…
25643:SU: Starting iteration 3, state space = (2,3) : (0,1)
…
27951:SU: Considering interleaved complex view merging
…
30186:SU: Considering interleaved distinct placement
…
30188:SU: Considering interleaved join pred push down
30200:JPPD: Starting iteration 1, state space = (3) : (0)
32118:JPPD: Updated best state, Cost = 10.133928
32119:JPPD: Starting iteration 2, state space = (3) : (1)
33561:JPPD: Not update best state, Cost = 27.023273
33562:JPPD: Will not use JPPD from query block SEL$C772B8D1 (#1)
…
33566:SU: Updated best state, Cost = 10.133928
33567:SU: Starting iteration 4, state space = (2,3) : (0,0)



Finally the end ☺

34775:SU: Not update best state, Cost = 17.020791
34776:SU: Will not unnest subquery SEL$3 (#2)
34777:SU: Will unnest subquery SEL$2 (#3)

34778:SU: Reconstructing original query from best state.
…

End of iterations, the winner is…

Summary

- Logical optimizer changes QBs
- Physical optimizer used to cost changes
- QB Registry tracks all those changes
  - Can be used to find the cost of each trans
  - Find if trans expected was even considered
  - Find cost of trans expected vs selected
- It’s not hard but it requires practice


Q&A

References


court to
on, a matter
SYN.
ref·er·ence
referred, esp.
person, committee
connection; regard
directing of attention
4. a)
work

Contact Information

- mauro.pagano@gmail.com
- http://mauro-pagano.com
- @mautro