# Guesswork and the Optimizer

Iordan K. Iotzov

Director Data and Systems Administration
at News America Marketing (NewsCorp)

Blog: http://iiotzov.wordpress.com/

# About me

- 15+ years of database administration and development experience
- MS in Computer Science, BS in Electrical Engineering
- Presented at NYOUG, Hotsos, RMOUG and Virta-Thon
- Currently Director Data and Systems Administration at News America Marketing (NewsCorp)

# Agenda

- Overview
- Foundations of Estimating Cardinality
- Confidence of Cardinality Estimates
  - The Current State
  - An Attempt to Measure
  - Practical Applications
- Conclusion

# Overview

## Ask Why

Typical SQL tuning thought process:

The optimizer generated suboptimal plan that takes a long time to execute.

*Why? What happened?*

In many cases, the optimizer did not get cardinalities correct– the actual number is very different from the estimated one (Tuning by Cardinality Feedback)

*Why did the optimizer miscalculate the cardinalities?*

It lacked statistics or **it made assumptions/guesses that turned out to be incorrect**

# Overview

## Ask Why

Guesswork in technology is (rightfully) considered bad…

**BAAG (Battle Against Any Guess ) party:**

“The main idea is to eliminate guesswork from our decision making process — once and for all.”

**…**

“Guesswork - just say no!”

So if guesswork is so bad, should we not be aware that the Oracle CBO is guessing in some cases?

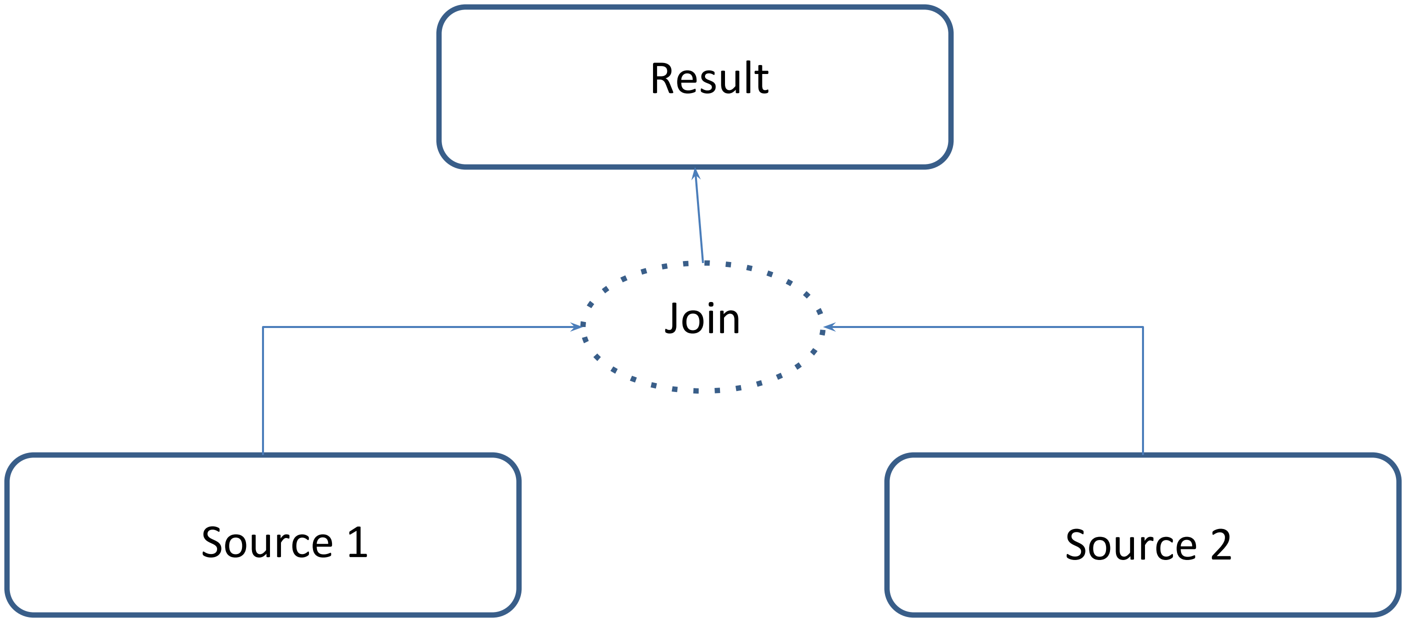
You bet…

# Foundations of Estimating Cardinality

## Joins

Basic formula for join cardinality:

$$Card_{Result} = Card_{Source\ 1} \times Card_{Source\ 2} \times Sel_{Pred}$$

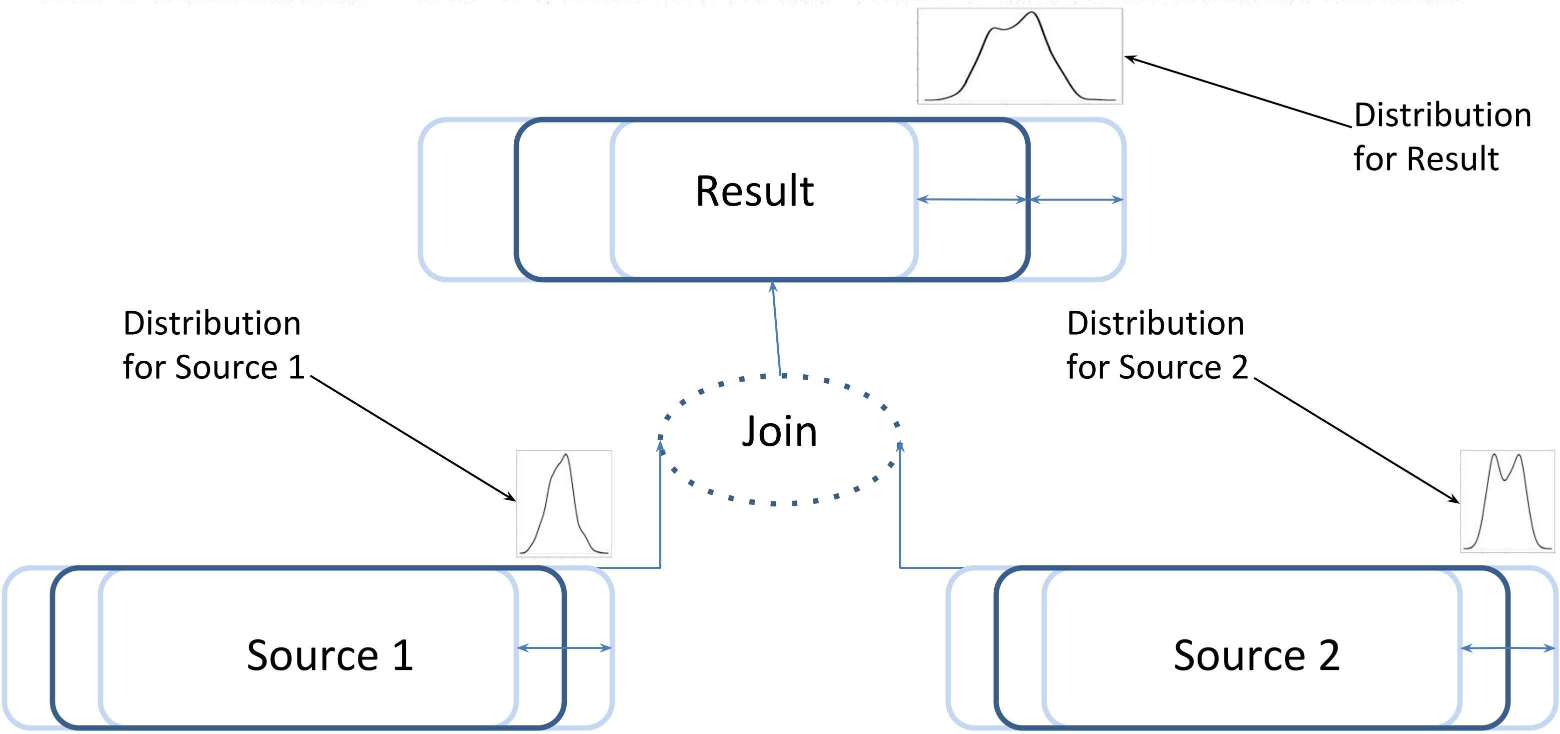


# Foundations of Estimating Cardinality

## Joins - Accounting for Errors

Formula for join cardinality accounting for errors:

$$(1+e^{+}_{R})Card_{Result} = (1+e^{+}_{S1}) \times Card_{Source\,1} \times (1+e^{+}_{S2}) \times Card_{Source\,2} \times Sel_{Pred}$$



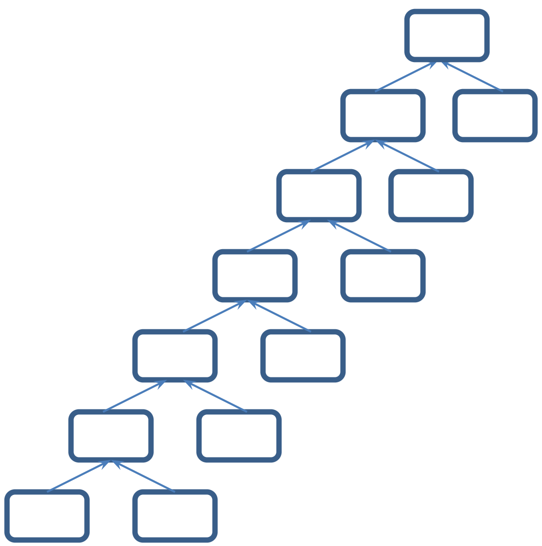
# Foundations of Estimating Cardinality

## Joins - Accounting for Errors

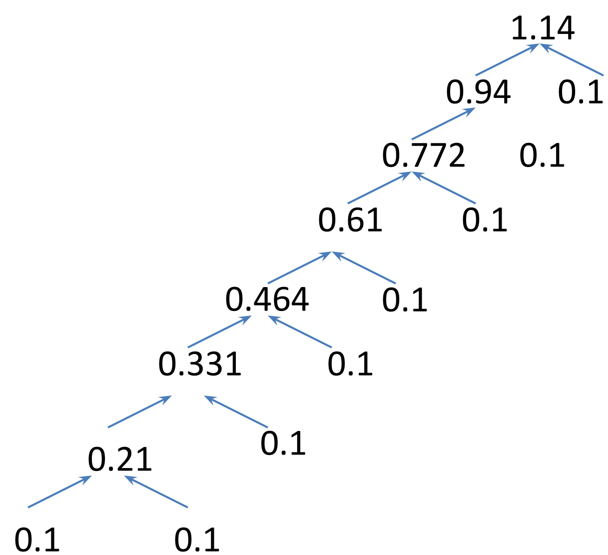
Error propagation for multi-step joins:

Each table cardinality estimate comes with (only!) 10% errors

Join structure



Errors

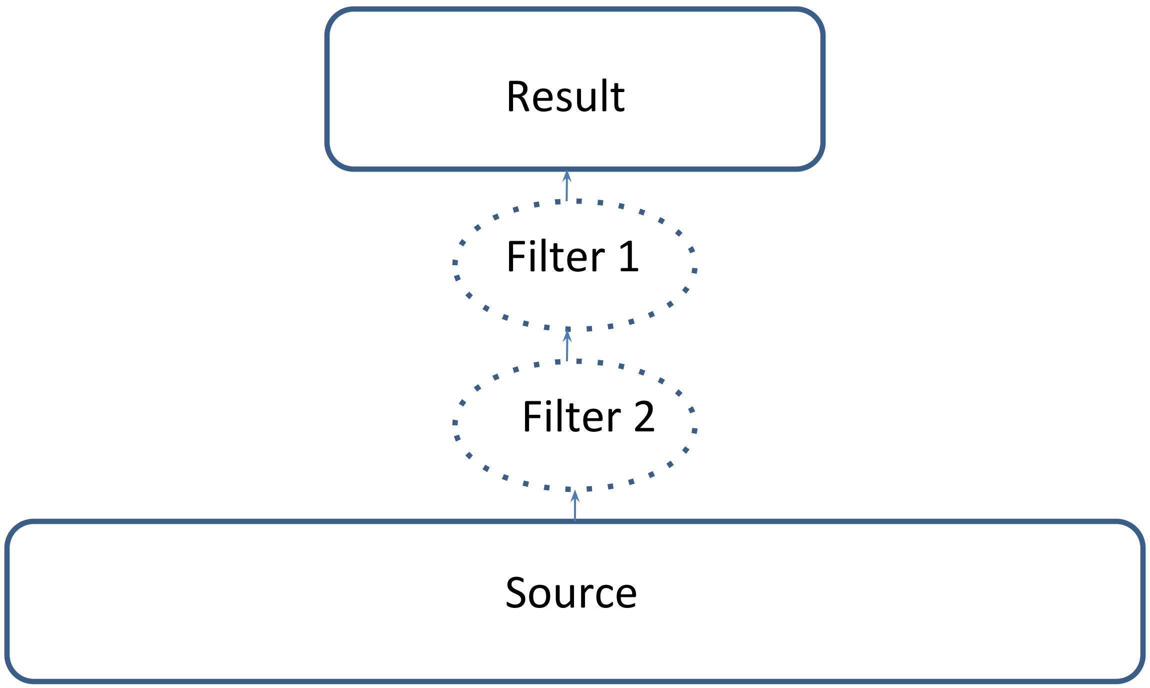


# Foundations of Estimating Cardinality

## Filters

Basic formula for cardinality with two filters (AND)

$$Card_{Result} = Card_{Source} \times Sel_{Filter\ 1} \times Sel_{Filter\ 2}$$

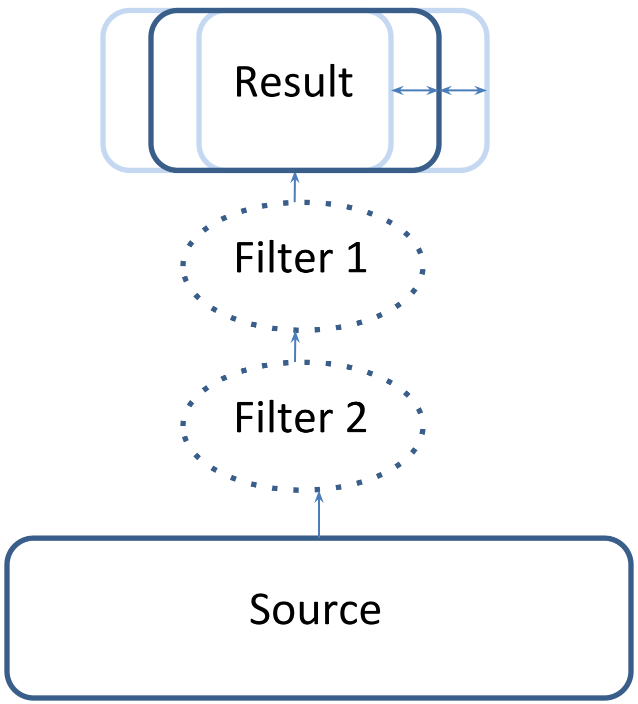


# Foundations of Estimating Cardinality

## Filters – Accounting for Errors

Formula for cardinality with two filters (AND), accounting for errors

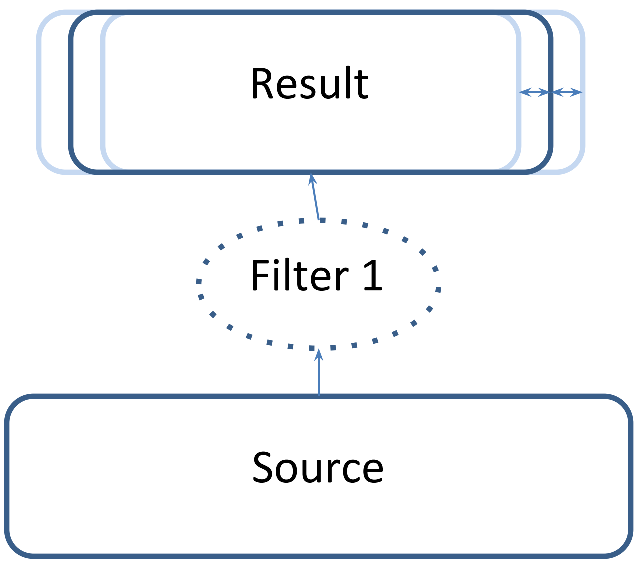
$$(1 + e^{+}_{R})Card_{Result} = Card_{Source} \times (1 + e^{+}_{F1})Sel_{Filter\ 1} \times (1 + e^{+}_{F2})Sel_{Filter\ 2}$$

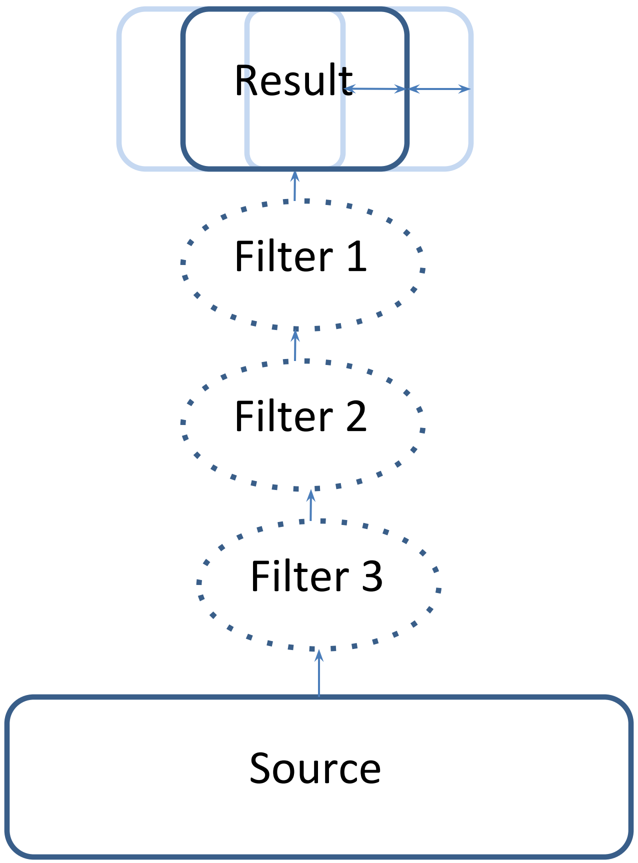


# Foundations of Estimating Cardinality

## Filters – Accounting for Errors

Aggregation of errors for multiple filters



# Confidence of Cardinality Estimates

## Definition

My definition:
Confidence of a cardinality estimate is inversely related to the maximum *relative* cardinality error.

# Confidence of Cardinality Estimates

## Current State

Working assumption: Oracle CBO does not universally compute or use confidence level of its cardinality estimates

It is hard to prove a negative, but

- There is no official documentation about confidence levels of cardinality estimates
- Experts agree in general* – Thanks Mr. Lewis!
- 10053 trace shows no indication that such information is available

*https://community.oracle.com/message/11161714#11161714

# Confidence of Cardinality Estimates

## Current State

Q1 – query that forces
CBO to make wild assumptions

```
select
    tab2.*
from
    tab1 ,
    tab2
where
    tab1.str like '%BAA%'
and
    tab1.id = tab2.id
```

Q2 – query that forces
CBO to make reasonable assumptions
based on the fact the NUM column has
20 distinct values that are uniformly
distributed

```
select
    tab2.*
from
    tab1 ,
    tab2
where
    tab1.NUM = 14
and
    tab1.id = tab2.id
```

# Confidence of Cardinality Estimates

## Current State

The cardinality estimates and the execution plans for Q1 and Q2 are identical

Plan for Q1:

| Id | Operation | Name | Rows | Bytes | Cost | Time |
|---|---|---|---|---|---|---|
| 0 | SELECT STATEMENT | | | | 38K | |
| 1 | HASH JOIN | | 488K | 21M | 38K | 00:08:49 |
| 2 | TABLE ACCESS FULL | TAB1 | 488K | 11M | 11K | 00:02:20 |
| 3 | TABLE ACCESS FULL | TAB2 | 9766K | 210M | 10K | 00:02:06 |

Plan for Q2:

| Id | Operation | Name | Rows | Bytes | Cost | Time |
|---|---|---|---|---|---|---|
| 0 | SELECT STATEMENT | | | | 38K | |
| 1 | HASH JOIN | | 488K | 15M | 38K | 00:08:45 |
| 2 | TABLE ACCESS FULL | TAB1 | 488K | 4395K | 11K | 00:02:20 |
| 3 | TABLE ACCESS FULL | TAB2 | 9766K | 210M | 10K | 00:02:06 |

# Confidence of Cardinality Estimates

## Current State

Oracle CBO (DS 3 and up) is aware of some of the predicates that force it to guess, and is able to dynamically gather statistics on the respective tables

| Dynamic Sampling Level | Sampling Conditions |
|---|---|
| 0 | No dynamic sampling |
| 1 | unanalyzed table is joined to another table<br>unanalyzed table has no indexes;<br>unanalyzed table has "significant" size |
| **2** | **all unanalyzed tables (the default)** |
| 3 | all tables for which standard selectivity estimation used *a guess* for some predicate that is a potential dynamic sampling predicate |
| 4 | tables that have single-table predicates that reference 2 or more columns. |
| 5,6,7,8,9,10 | |

# Confidence of Cardinality Estimates

## Current State, Oracle 12c

Dynamic Sampling in Oracle 12c:

It goes up to 11. *Really!*


| Dynamic Sampling Level | Sampling Conditions |
|---|---|
| 11 | Use dynamic statistics automatically when the optimizer deems it necessary. The resulting statistics are persistent in the statistics repository, making them available to other queries. |

# Confidence of Cardinality Estimates

## Current State, Oracle 12c

When would auto dynamic sampling fire? In too many situations…

Table TAB3:

| Column Name | Not NULL | Type |
|---|---|---|
| **ID (Primary Key)** | **Y** | **NUMBER** |
| STR | | VARCHAR2(100) |

Search by primary key – no guesswork

```
alter session set optimizer_dynamic_sampling = 11 ;

select * from tab3 where id = 123 ;

....
Note
-----
   - dynamic statistics used: dynamic sampling (level=AUTO)
```

# Confidence of Cardinality Estimates

## Current State, Oracle 12c

Not only DS 11 (AUTO) fires more than needed, but it also can use excessive resources in same cases

### Dynamic Sampling : 11

```
select * from tab3
where id = 123 ;
```

…

```
15  recursive calls
 0  db block gets
11  consistent gets
 0  physical reads
 0  redo size
```

### Dynamic Sampling : 2 (Default)

```
select * from tab3
where id = 123 ;
```

…

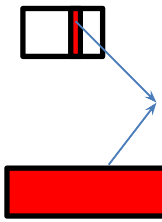
```
1  recursive calls
0  db block gets
3  consistent gets
0  physical reads
0  redo size
```

# Confidence of Cardinality Estimates

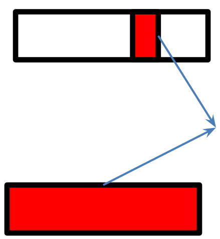
## Current State, Oracle 12c

### Adaptive Execution Plans – desired behavior

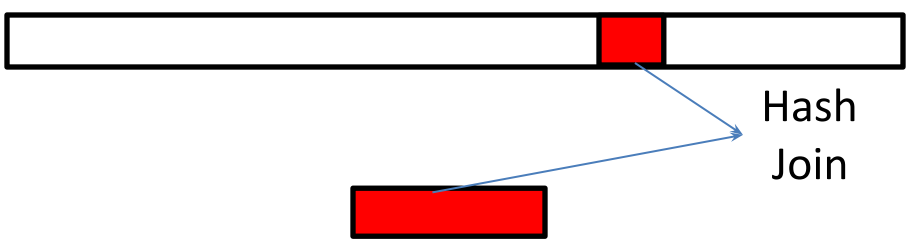
 - Size of table  - Size after filters applied




If one of the sets is “significantly” smaller than the other, and there are appropriate indexes, then Oracle should choose a NL Join




If one of the sets is “slightly” smaller than the other, and there are appropriate indexes, then Oracle should choose an Adaptive Plan




If the size of the two sets is “comparable”, then Oracle should choose a Hash Join.

# Confidence of Cardinality Estimates

## Current State, Oracle 12c

Do adaptive execution plans take into account the confidence of cardinality estimates?

If that was the case, CBO would favor adaptive execution plans when dealing with SQL that force it to make a wild assumption.

I do not see this behavior in the documentation or in practice(at least in 12cR1), so I assume that adaptive plans do not utilize confidence of cardinality estimates.

I have no inside knowledge of adaptive execution plans, so this is merely an opinion.

# Confidence of Cardinality Estimates

## An Attempt to Measure

`XPLAN_CONFIDENCE` package:

- Attempts to measure the maximum error , a proxy for confidence, as a continuous variable
- Absolutely no warranties (I wrote it) – for demonstration purposes only
- Uses a couple of basic factors and has a few limitations:
  - Does not recognize sub-queries, inline views and other “complex” structures
  - Limited ability to parse complex filters
  - Not aware of profiles, dynamic sampling, adaptive cursor sharing
  - Limited ability to parse and handle functions(built-in and PL/SQL)
  - Not aware of query rewrite and materialized views,
  - Very limited ability to handle extended statistics
  - Does not support Oracle 12c
  - And many, many more limitations and restrictions

# Confidence of Cardinality Estimates

## An Attempt to Measure

A simple example:

```
select * from table(xplan_confidence.display('61hfhfk5ts01r'))
```

```
PLAN_TABLE_OUTPUT
SQL_ID  61hfhfk5ts01r, child number 0
-------------------------------------
SELECT RPLW.A_ID FROM TAB1 RPLW ,
TAB2 RMW WHERE RMW.P_ID =
RPLW.P_ID AND RMW.O_ID = :B2 AND RMW.R_ID = :B1

Plan hash value: 3365837995
------------------------------------------------------------------------------------------------------------------------
--
| Id  | Operation                    | Name                       | Rows  | Bytes | Cost (%CPU)| Time     |     Max. Error
|
------------------------------------------------------------------------------------------------------------------------
--
|   0 | SELECT STATEMENT             |                            |       |       |     9 (100)|          |
.16|
|*  1 |  HASH JOIN                   |                            |     1 |    23 |     9  (12)| 00:00:01 |
.16|
|*  2 |   TABLE ACCESS BY INDEX ROWID| TAB2                       |     1 |    15 |     3   (0)| 00:00:01 |
.1|
|*  3 |    INDEX RANGE SCAN          | NUK_TAB2_R_ID              |    83 |       |     1   (0)| 00:00:01 |
.05|
|   4 |   TABLE ACCESS FULL          | TAB1                       |  1179 |  9432 |     5   (0)| 00:00:01    |
0|
------------------------------------------------------------------------------------------------------------------------
--

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

# Confidence of Cardinality Estimates

## An Attempt to Measure

Technical notes:

```
create or replace
package xplan_confidence as
……
 function display(sql_id           varchar2 default  null,
                  cursor_child_no  integer  default  0,
                  format           varchar2 default  'typical')
  return sys.dbms_xplan_type_table
  pipelined;
end xplan_confidence;
```

➢Input and Output similar to DBMS_XPLAN.DISPLAY_CURSOR

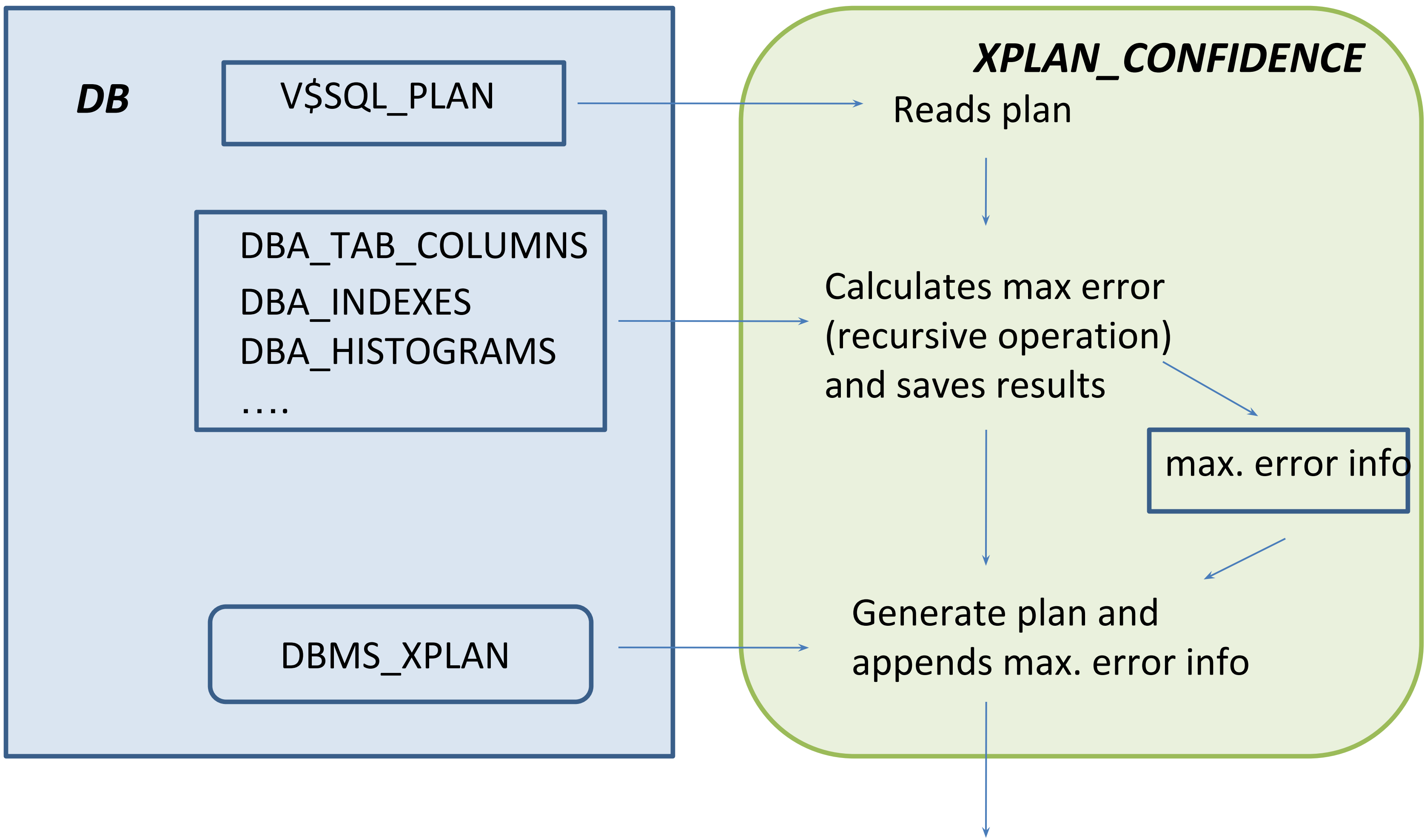
➢Works only for SQL statements that are in the cache – V$SQL_PLAN

➢Works best in SQL Developer

# Confidence of Cardinality Estimates

## An Attempt to Measure

Inside XPLAN_CONFIDENCE package:



# Confidence of Cardinality Estimates

## An Attempt to Measure, Assumptions

| Predicate | Complex | Bind/ Variables | Histograms | Assigned max. error | DS 3 | Opportunities for improving CBO's confidence |
|---|---|---|---|---|---|---|
| = | Y | | | 20% | NA | Substitute with simple predicates |
| | N | Y | | 5% | N | Consider literals (be aware of parsing implications) |
| | | N | Y | 1% | N | |
| | | | N | 5% | N | Consider histograms if appropriate |

# Confidence of Cardinality Estimates

## An Attempt to Measure, Assumptions

| Predicate | Complex | Bind/ Variables | Histograms | Assigned max. error | DS 3 | Opportunities for improving CBO's confidence |
|---|---|---|---|---|---|---|
| > | Y | | | 40% | NA | Substitute with simple predicate(s) |
| | N | Y | | 10% | N | Consider literals (be aware of parsing implications) |
| | | N | Y | 1% | N | |
| | | | N | 10% | N | Consider histograms if appropriate |

# Confidence of Cardinality Estimates

## An Attempt to Measure, Assumptions

| DB structures | Assigned max. error | Opportunities for improving CBO's confidence |
|---|---|---|
| Unique Index | 0% | |
| Extended Statistics Columns | 5% | A very effective way to deal with correlated columns as well as large number of filter conditions |

# Confidence of Cardinality Estimates

## An Attempt to Measure, Assumptions

| Predicate | Assigned max. error | DS 3 | Opportunities for improving CBO's confidence |
|---|---|---|---|
| LIKE | 200% | Y | Force dynamic sampling |
| MEMBER OF* | 200% | N | IN predicate, if number of records is low.<br>Store records in DB table; make sure table stats are available |

* - significantly better in the later versions

# Confidence of Cardinality Estimates

## An Attempt to Measure

## Outside of the model:

| Predicate | Opportunities for improving CBO's confidence |
|---|---|
| PL/SQL functions | Force dynamic sampling<br>Utilize ASSOCIATE STATISTICS |
| Pipeline functions | Force dynamic sampling (DYNAMIC_SAMPLING hint, version 11.1.0.7 +)<br>Utilize ASSOCIATE STATISTICS |
| CONNECT BY LEVEL < | |

# Confidence of Cardinality Estimates

## Practical Applications

Review SQL coding standard and vet all new SQL features and constructs:

- ➢Can CBO reliably figure out the selectivity/cardinality of the new feature under different circumstances?
    - ➢Explain plan
    - ➢10053 traces

- ➢How easy it is to supply the CBO with the needed information?

# Confidence of Cardinality Estimates

## Practical Applications, Normal Confidence Deterioration

| Id | Operation | Name | Rows | Bytes | Cost (%CPU) | Time | Max. Error |
|---|---|---|---|---|---|---|---|
| 0 | SELECT STATEMENT | | | | 72 (100) | | 9.11 |
| 1 | VIEW | | 1 | 13 | 72 (25) | 00:00:01 | 9.11 |
| 2 | SORT UNIQUE | | 1 | 257 | 71 (20) | 00:00:01 | 9.11 |
| 3 | NESTED LOOPS OUTER | | 257 | 66049 | 70 (19) | 00:00:01 | 9.11 |
| 4 | MERGE JOIN OUTER | | 171 | 42921 | 70 (15) | 00:00:01 | 8.53 |
| 5 | SORT JOIN | | 164 | 39852 | 62 (20) | 00:00:01 | 7.64 |
| 6 | NESTED LOOPS OUTER | | 164 | 39852 | 61 (19) | 00:00:01 | 7.64 |
| 7 | MERGE JOIN OUTER | | 120 | 28440 | 61 (19) | 00:00:01 | 6.84 |
| 8 | SORT JOIN | | 119 | 27251 | 51 (20) | 00:00:01 | 6.11 |
| 9 | NESTED LOOPS OUTER | | 119 | 27251 | 50 (18) | 00:00:01 | 6.11 |
| 10 | MERGE JOIN OUTER | | 119 | 26775 | 50 (18) | 00:00:01 | 5.77 |
| 11 | SORT JOIN | | 119 | 25704 | 42 (20) | 00:00:01 | 4.57 |
| 12 | NESTED LOOPS OUTER | | 119 | 25704 | 41 (18) | 00:00:01 | 4.57 |
| 13 | MERGE JOIN OUTER | | 119 | 25228 | 41 (18) | 00:00:01 | 4.31 |
| 14 | SORT JOIN | | 79 | 16037 | 37 (17) | 00:00:01 | 3.81 |
| 15 | MERGE JOIN OUTER | | 79 | 16037 | 36 (14) | 00:00:01 | 3.81 |
| 16 | SORT JOIN | | 79 | 15405 | 19 (16) | 00:00:01 | 3.37 |
| 17 | MERGE JOIN OUTER | | 79 | 15405 | 18 (12) | 00:00:01 | 3.37 |
| 18 | SORT JOIN | | 79 | 15010 | 13 (8) | 00:00:01 | 2.96 |
| 19 | NESTED LOOPS OUTER | | 79 | 15010 | 12 (0) | 00:00:01 | 2.96 |
| 20 | NESTED LOOPS | | 1 | 175 | 9 (0) | 00:00:01 | 2.77 |
| 21 | NESTED LOOPS | | 1 | 166 | 9 (0) | 00:00:01 | 2.59 |
| 22 | NESTED LOOPS | | 1 | 148 | 8 (0) | 00:00:01 | 2.42 |
| 23 | NESTED LOOPS | | 1 | 120 | 6 (0) | 00:00:01 | 2.26 |
| 24 | NESTED LOOPS | | 1 | 117 | 6 (0) | 00:00:01 | 2.1 |
| 25 | NESTED LOOPS OUTER | | 1 | 102 | 5 (0) | 00:00:01 | 1.95 |
| 26 | NESTED LOOPS OUTER | | 1 | 98 | 5 (0) | 00:00:01 | 1.81 |
| 27 | NESTED LOOPS OUTER | | 1 | 79 | 3 (0) | 00:00:01 | 1.68 |
| 28 | NESTED LOOPS OUTER | | 1 | 75 | 3 (0) | 00:00:01 | 1.55 |
| 29 | NESTED LOOPS OUTER | | 1 | 71 | 3 (0) | 00:00:01 | 1.43 |
| 30 | NESTED LOOPS OUTER | | 1 | 67 | 3 (0) | 00:00:01 | 1.32 |
| 31 | NESTED LOOPS OUTER | | 1 | 63 | 3 (0) | 00:00:01 | 1.21 |
| 32 | NESTED LOOPS OUTER | | 1 | 59 | 3 (0) | 00:00:01 | 1.1 |
| * 33 | TABLE ACCESS FULL | RW | 1 | 54 | 3 (0) | 00:00:01 | 1 |
| * 34 | INDEX UNIQUE SCAN | PK_RS_ID | 1 | 5 | 0 (0) | | .05 |
| * 35 | INDEX UNIQUE SCAN | PK_SS_IND | 1 | 4 | 0 (0) | | .05 |
| * 36 | INDEX UNIQUE SCAN | PK_SS_IND | 1 | 4 | 0 (0) | | .05 |
| * 37 | INDEX UNIQUE SCAN | PK_SS_IND | 1 | 4 | 0 (0) | | .05 |
| * 38 | INDEX UNIQUE SCAN | PK_SS_IND | 1 | 4 | 0 (0) | | .05 |
| * 39 | INDEX UNIQUE SCAN | PK_SS_IND | 1 | 4 | 0 (0) | | .05 |
| 40 | TABLE ACCESS BY INDEX ROWID | RS_O | 1 | 19 | 2 (0) | 00:00:01 | .05 |
| * 41 | INDEX UNIQUE SCAN | RSO_PK | 1 | | 1 (0) | 00:00:01 | .05 |
| * 42 | INDEX UNIQUE SCAN | PK_SS_IND | 1 | 4 | 0 (0) | | .05 |
| * 43 | TABLE ACCESS BY INDEX ROWID | CLS | 1 | 15 | 1 (0) | 00:00:01 | .05 |
| * 44 | INDEX UNIQUE SCAN | CL_PK | 1 | | 0 (0) | | .05 |
| * 45 | INDEX UNIQUE SCAN | PK_BT_ID | 1 | 3 | 0 (0) | | .05 |
| 46 | TABLE ACCESS BY INDEX ROWID | RS_OE | 1 | 28 | 2 (0) | 00:00:01 | .05 |
| * 47 | INDEX RANGE SCAN | ROE_IDX3 | 1 | | 1 (0) | 00:00:01 | .05 |
| 48 | TABLE ACCESS BY INDEX ROWID | RS_P | 1 | 18 | 1 (0) | 00:00:01 | .05 |
| * 49 | INDEX UNIQUE SCAN | RSP_PK | 1 | | 0 (0) | | .05 |
| * 50 | INDEX UNIQUE SCAN | RSE_PK | 1 | 9 | 0 (0) | | .05 |
| 51 | TABLE ACCESS BY INDEX ROWID | RW | 70 | 1050 | 3 (0) | 00:00:01 | .05 |
| * 52 | INDEX RANGE SCAN | NUK_RR_ID | 70 | | 1 (0) | 00:00:01 | .05 |
| * 53 | SORT JOIN | | 1532 | 7660 | 5 (20) | 00:00:01 | .1 |
| 54 | INDEX FAST FULL SCAN | NUK_RC_ID | 1532 | 7660 | 4 (0) | 00:00:01 | 0 |
| * 55 | SORT JOIN | | 15179 | 118K | 17 (12) | 00:00:01 | .1 |
| 56 | INDEX FAST FULL SCAN | ALU_IDX | 15179 | 118K | 15 (0) | 00:00:01 | 0 |
| * 57 | SORT JOIN | | 3 | 27 | 4 (25) | 00:00:01 | .1 |
| 58 | TABLE ACCESS FULL | RES_IND | 3 | 27 | 3 (0) | 00:00:01 | 0 |
| * 59 | INDEX UNIQUE SCAN | PK_C_VV_ID | 1 | 4 | 0 (0) | | .05 |
| * 60 | SORT JOIN | | 1538 | 13842 | 8 (13) | 00:00:01 | .22 |
| 61 | TABLE ACCESS FULL | RES_OCW | 1538 | 13842 | 7 (0) | 00:00:01 | 0 |
| * 62 | INDEX UNIQUE SCAN | PK_OSW | 1 | 4 | 0 (0) | | .05 |
| * 63 | SORT JOIN | | 1555 | 12440 | 10 (10) | 00:00:01 | .1 |
| 64 | TABLE ACCESS FULL | RES_PLW | 1555 | 12440 | 9 (0) | 00:00:01 | 0 |
| * 65 | INDEX UNIQUE SCAN | PK_SPP_IS | 1 | 6 | 0 (0) | | .1 |
| * 66 | SORT JOIN | | 1603 | 12824 | 8 (13) | 00:00:01 | .1 |
| 67 | TABLE ACCESS FULL | RES_TCW | 1603 | 12824 | 7 (0) | 00:00:01 | 0 |
| * 68 | INDEX UNIQUE SCAN | PK_STT_ID | 2 | 12 | 0 (0) | | .06 |

# Confidence of Cardinality Estimates

## Practical Applications, Normal Confidence Deterioration

Reasons why the larger the SQL, the higher the chance of suboptimal execution plan:

- The confidence of the cardinality get diminished as the query progresses
  - Most SQL constructs pass on or amplify the cardinality errors
  - Few SQL construct reduce cardinality errors

Example:

```
where
    col in
    (select max(col1) from subq)
```

The cardinality errors in `subq` will not be propagated

# Confidence of Cardinality Estimates

## Practical Applications, Normal Confidence Deterioration

- CBO cannot examine all join permutations

Number of permutation for n tables is (n!). (n!) is growing really fast:

| (n) | (n!) |
|---|---|
| 13 | 6226020800 |
| 14 | 87178291200 |
| 15 | 1307674368000 |

Trends:

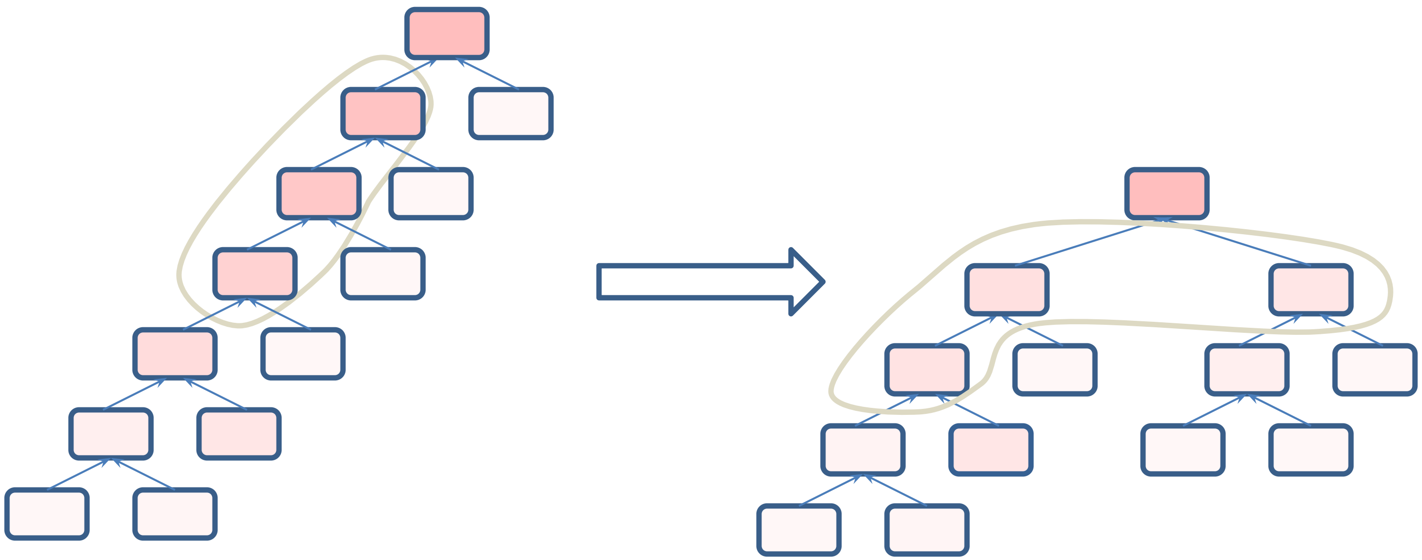
- (+) Faster CPU allow for more permutations
- (+) The optimizer uses better heuristics to reduce the search space
- (-) More parameters are used to measure cost

# Confidence of Cardinality Estimates

## Practical Applications, Normal Confidence Deterioration

Mitigating the effects of Normal Confidence Deterioration for very large queries:

Logically "split" the query



# Confidence of Cardinality Estimates

## Practical Applications, Normal Confidence Deterioration

Logically "splitting" the query using NO_MERGE hint

```
select
  subq1.id , sub12.name, …
from
 (select .. from a,b ..)  subq1 ,
 (select .. from n,m ..)  subq2
where
 subq1.col1 = subq2.sol2
 and…
```

```
select /*+ NO_MERGE(subq1)
NO_MERGE(subq2) */
  subq1.id , sub12.name, …
from
 (select .. from a,b ..)  subq1 ,
 (select .. from n,m ..)  subq2
where
 subq1.col1 = subq2.sol2
 and…
```

# Confidence of Cardinality Estimates

## Practical Applications, Rapid Confidence Deterioration

Plan hash value: 390455572

| Id | Operation | Name | Rows | Bytes | Cost (%CPU) | Time | Max. Error |
|---|---|---|---|---|---|---|---|
| 0 | SELECT STATEMENT | | | | 2373 (100) | | 26.02 |
| 1 | VIEW | | 911 | 11843 | 2373 (18) | 00:00:29 | 26.02 |
| 2 | SORT UNIQUE | | 911 | 243K | 2371 (4) | 00:00:29 | 26.02 |
| 3 | NESTED LOOPS OUTER | | 3389 | 906K | 2168 (4) | 00:00:27 | 26.02 |
| * 4 | HASH JOIN OUTER | | 2259 | 591K | 2168 (4) | 00:00:27 | 24.48 |
| 5 | NESTED LOOPS | | 1304 | 331K | 1842 (4) | 00:00:23 | 23.26 |
| * 6 | HASH JOIN | | 1304 | 326K | 1842 (4) | 00:00:23 | 22.11 |
| 7 | NESTED LOOPS | | 826 | 199K | 1515 (3) | 00:00:19 | 21.01 |
| 8 | NESTED LOOPS | | 826 | 195K | 1515 (3) | 00:00:19 | 19.96 |
| * 9 | HASH JOIN | | 934 | 215K | 1515 (3) | 00:00:19 | 18.01 |
| * 10 | HASH JOIN | | 729 | 162K | 1371 (3) | 00:00:17 | 17.11 |
| 11 | NESTED LOOPS OUTER | | 237 | 50481 | 976 (1) | 00:00:12 | 16.24 |
| 12 | NESTED LOOPS OUTER | | 237 | 49533 | 976 (1) | 00:00:12 | 15.42 |
| 13 | NESTED LOOPS | | 131 | 26200 | 845 (1) | 00:00:11 | 14.64 |
| * 14 | HASH JOIN | | 131 | 25021 | 845 (1) | 00:00:11 | 13.9 |
| * 15 | HASH JOIN | | 131 | 22663 | 826 (1) | 00:00:10 | 13.19 |
| * 16 | HASH JOIN | | 127 | 18415 | 650 (1) | 00:00:08 | 12.51 |
| * 17 | HASH JOIN OUTER | | 127 | 16510 | 569 (1) | 00:00:07 | 11.87 |
| * 18 | FILTER | | | | | | 11.25 |
| * 19 | HASH JOIN RIGHT OUTER | | 127 | 15494 | 553 (1) | 00:00:07 | 4.91 |
| 20 | VIEW | index$_join$_021 | 81 | 1701 | 3 (34) | 00:00:01 | 0 |
| * 21 | HASH JOIN | | | | | | 0 |
| 22 | INDEX FAST FULL SCAN | PK_SS_IND | 81 | 1701 | 1 (0) | 00:00:01 | 0 |
| 23 | INDEX FAST FULL SCAN | UK_SD | 81 | 1701 | 1 (0) | 00:00:01 | 0 |
| 24 | NESTED LOOPS OUTER | | 127 | 12827 | 550 (1) | 00:00:07 | 4.63 |
| 25 | NESTED LOOPS OUTER | | 127 | 12319 | 550 (1) | 00:00:07 | 4.36 |
| 26 | NESTED LOOPS OUTER | | 127 | 9906 | 310 (1) | 00:00:04 | 4.11 |
| 27 | NESTED LOOPS OUTER | | 127 | 9398 | 310 (1) | 00:00:04 | 3.86 |
| 28 | NESTED LOOPS OUTER | | 127 | 8890 | 310 (1) | 00:00:04 | 3.63 |
| 29 | NESTED LOOPS OUTER | | 127 | 8382 | 310 (1) | 00:00:04 | 3.41 |
| 30 | NESTED LOOPS OUTER | | 127 | 7747 | 310 (1) | 00:00:04 | 3.2 |
| * 31 | TABLE ACCESS FULL | RS | 127 | 7239 | 310 (1) | 00:00:04 | 3 |
| * 32 | INDEX UNIQUE SCAN | PK_SS_IND | 1 | 4 | 0 (0) | | .05 |
| * 33 | INDEX UNIQUE SCAN | PK_RR_ID | 1 | 5 | 0 (0) | | .05 |
| * 34 | INDEX UNIQUE SCAN | PK_SS_IND | 1 | 4 | 0 (0) | | .05 |
| * 35 | INDEX UNIQUE SCAN | PK_SS_IND | 1 | 4 | 0 (0) | | .05 |
| * 36 | INDEX UNIQUE SCAN | PK_SS_IND | 1 | 4 | 0 (0) | | .05 |
| 37 | TABLE ACCESS BY INDEX ROWID | RS_OPTY | 1 | 19 | 2 (0) | 00:00:01 | .05 |
| * 38 | INDEX UNIQUE SCAN | RSO_PK | 1 | | 1 (0) | 00:00:01 | .05 |
| * 39 | INDEX UNIQUE SCAN | PK_SS_IND | 1 | 4 | 0 (0) | | .05 |
| 40 | INDEX FAST FULL SCAN | A_UL_IDX | 15179 | 118K | 15 (0) | 00:00:01 | 0 |
| 41 | VIEW | index$_join$_003 | 9506 | 139K | 81 (2) | 00:00:01 | 0 |
| * 42 | HASH JOIN | | | | | | 0 |
| 43 | INDEX FAST FULL SCAN | CL_PK | 9506 | 139K | 40 (0) | 00:00:01 | 0 |
| * 44 | INDEX FAST FULL SCAN | CL_SL_IDX | 9506 | 139K | 60 (0) | 00:00:01 | 0 |
| 45 | TABLE ACCESS FULL | RS_OE | 48774 | 1333K | 175 (1) | 00:00:03 | 0 |
| 46 | TABLE ACCESS FULL | RS_P | 1020 | 18360 | 19 (0) | 00:00:01 | 0 |
| * 47 | INDEX UNIQUE SCAN | RSE_PK | 1 | 9 | 0 (0) | | .05 |
| * 48 | INDEX RANGE SCAN | PK_RI_ID | 2 | 18 | 1 (0) | 00:00:01 | .05 |
| * 49 | INDEX UNIQUE SCAN | PK_CV_ID | 1 | 4 | 0 (0) | | .05 |
| 50 | TABLE ACCESS FULL | R_M | 152K | 2227K | 378 (1) | 00:00:05 | 0 |
| 51 | INDEX FAST FULL SCAN | UK_RPP_ID | 82708 | 646K | 128 (1) | 00:00:02 | 0 |
| * 52 | INDEX UNIQUE SCAN | PK_S_ID | 1 | 6 | 0 (0) | | .1 |
| * 53 | INDEX UNIQUE SCAN | PK_R_ID | 1 | 5 | 0 (0) | | .05 |
| 54 | TABLE ACCESS FULL | RES_OC | 168K | 1480K | 310 (1) | 00:00:04 | 0 |
| * 55 | INDEX UNIQUE SCAN | PK_SCS_ID | 1 | 4 | 0 (0) | | .05 |
| 56 | TABLE ACCESS FULL | R_T_C | 187K | 1464K | 309 (1) | 00:00:04 | 0 |
| * 57 | INDEX UNIQUE SCAN | PK_ST_ID | 2 | 12 | 0 (0) | | .06 |

# Confidence of Cardinality Estimates

## Practical Applications, Rapid Confidence Deterioration

Certain predicates contribute disproportionately to confidence deterioration

Step 31:

```
filter(""RS"".""RS_ID""MEMBER OF:1
       AND "RS"".""ST"" LIKE "%C" )
```

Step 18:

```
filter((""S"".""D"" IS NULL
       OR LOWER(""S"".""D"")='fs1'
       OR LOWER(""S"".""D"")='fs2'
       OR (LOWER(""S"".""D"")='cmb'
           AND LOWER(""S"".""R"")='f1' )
       OR ( LOWER(""S"".""D"")='cnld'
           AND LOWER(""S"".""R"")='f2')
       OR LOWER(""S"".""D"")='err'))"
```

In many cases, performance optimization is nothing more than finding the predicates that confuse the optimizer the most, and dealing with them

# Confidence of Cardinality Estimates

## Practical Applications, Rapid Confidence Deterioration

Ways to deal with “problem“ predicates:

- Dynamic sampling

  ->In some cases (Oracle 11Rr2 and up) Oracle decides to run dynamic sampling without explicit instructions

- Utilize strategies to supply relevant information to the optimizer

  ->Extended Statistics

  ->Virtual columns

  ->ASSOCIATE STATISTICS

- Rewrite to a less “confusing” predicate

  for example, this clause

```
and  col1 <= nvl(col2,to_timestamp( '12-31-9999','mm-dd-yyyy'))
```

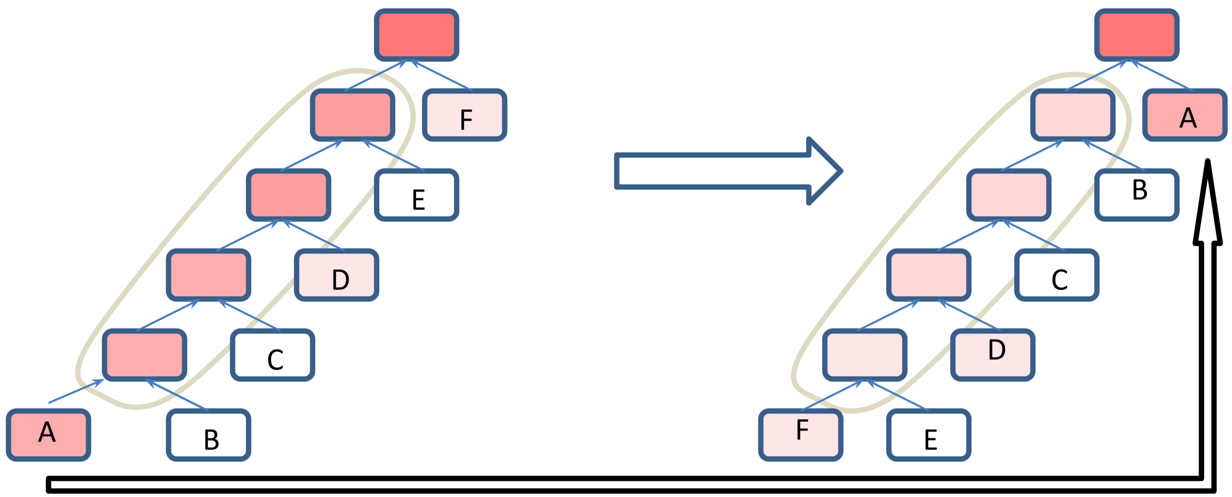
  can be simplified to

```
and (col1 <= col2 or col2 is NULL)
```

# Confidence of Cardinality Estimates

## Practical Applications, Rapid Confidence Deterioration

➢ If there is a suitable selective predicate, push the “problem” predicate towards the end of the execution plan, so the damage it does is minimized




Predicates:

A: str like '%A%' and str like '%B%' and str like '%C%’

D: flag""<>2

F : sec_id between 100  and 200

# Confidence of Cardinality Estimates

## Practical Applications, Rapid Confidence Deterioration

```
select
/*+ LEADING ( f e d c b a)  */
sum( length(a.str) + length( b.str ) + length(  c.str
) + length( d.str ) + length ( e.str ) + length
(f.str) )
where  a.str like '%A%'  ) d , E e , F f
and    a.str like '%B%'
and    a.str like '%C%'
and    a.id = b.id
and    c.id = b.id
and    c.id = d.id
and    d.flag not in (2)
and    e.id = d.id
and    f.id = e.id
and    f.sec_id between 100 and 200
```

<- not selective (next to `and d.flag not in (2)`)

<- selective (next to `and f.sec_id between 100 and 200`)

# Confidence of Cardinality Estimates

## Practical Applications, Rapid Confidence Deterioration

Original Query:

```
"Plan hash value: 850278183"
"-----------------------------------------------------------------------------------------------
--------------"
"| Id  | Operation                          | Name | Starts | E-Rows | A-Rows |   A-Time    |,,,|  Max. Error|"
"-----------------------------------------------------------------------------------------------
--------------"
"|   0 | SELECT STATEMENT                   |      |      1 |        |      1 |00:01:13.40 |,,,|      11.38|"
"|   1 |  SORT AGGREGATE                    |      |      1 |      1 |      1 |00:01:13.40 |,,,|      11.38|"
"|   2 |   NESTED LOOPS                     |      |      1 |      1 |   1214 |00:01:36.32 |,,,|      11.38|"
"|   3 |    NESTED LOOPS                    |      |      1 |    200 |  1230K|00:01:01.52 |,,,|       9.7|"
"|   4 |     NESTED LOOPS                   |      |      1 |    200 |  1230K|00:00:50.52 |.,,|      9.19|"
"|   5 |      NESTED LOOPS                  |      |      1 |    250 |  1538K|00:00:38.20 |,,.|      7.82|"
"|   6 |       NESTED LOOPS                 |      |      1 |    250 |  1538K|00:00:20.64 |,,,|       7.4|"
"|*  7 |        TABLE ACCESS FULL           | A    |      1 |    250 |  1538K|00:00:04.84 |,,,|         7|"
"|   8 |        TABLE ACCESS BY INDEX ROWID| B    |  1538K|      1 |  1538K|00:00:16.79 |,,,|       .05|"
"|*  9 |         INDEX UNIQUE SCAN          | B PK |  1538K|      1 |  1538K|00:00:01.97 |,,,|       .05|"
"|  10 |       TABLE ACCESS BY INDEX ROWID | C    |  1538K|      1 |  1538K|00:00:17.60 |,,,|       .05|"
"|* 11 |        INDEX UNIQUE SCAN           | C PK |  1538K|      1 |  1538K|00:00:01.92 |,,,|       .05|"
"|* 12 |      TABLE ACCESS BY INDEX ROWID  | D    |  1538K|      1 |  1230K|00:00:07.65 |,,,|       .16|"
"|* 13 |       INDEX UNIQUE SCAN            | D PK |  1538K|      1 |  1538K|00:00:01.88 |,,,|       .05|"
"|  14 |     TABLE ACCESS BY INDEX ROWID   | E    |  1230K|      1 |  1230K|00:00:11.71 |,,,|       .05|"
"|* 15 |      INDEX UNIQUE SCAN             | E PK |  1230K|      1 |  1230K|00:00:01.50 |,,,|       .05|"
"|* 16 |    TABLE ACCESS BY INDEX ROWID    | F    |  1230K|      1 |   1214 |00:00:11.46 |,,,|       .16|"
"|* 17 |     INDEX UNIQUE SCAN              | F PK |  1230K|      1 |  1230K|00:00:01.56 |,,,|       .05|"
"-----------------------------------------------------------------------------------------------
-"
```

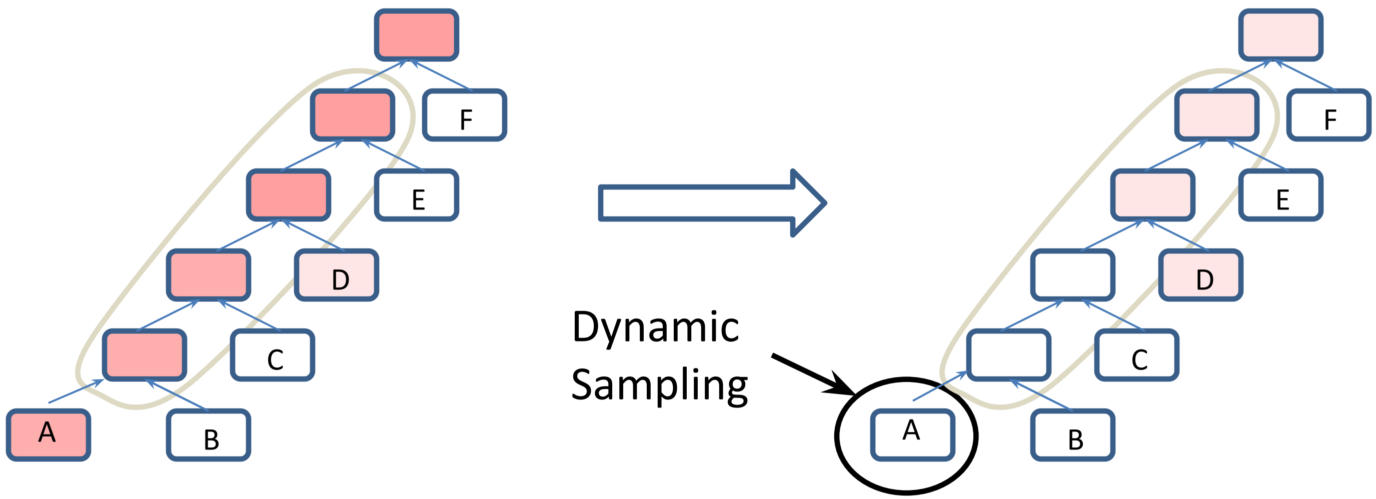
Query with Reordered Execution `/*+ LEADING ( f e d c b a)  */`:

```
"Plan hash value: 929876539"
"-----------------------------------------------------------------------------------------------
------"
"| Id  | Operation                          | Name   | Starts | E-Rows | A-Rows |   A-Time    |_|  Max. Error  |"
"-----------------------------------------------------------------------------------------------
------"
"|   0 | SELECT STATEMENT                   |        |      1 |        |      1 |00:00:10.72 |_|       11.38|"
"|   1 |  SORT AGGREGATE                    |        |      1 |      1 |      1 |00:00:10.72 |_|       11.38|"
"|   2 |   NESTED LOOPS                     |        |      1 |      1 |   1214 |00:00:11.65 |_|       11.38|"
"|   3 |    NESTED LOOPS                    |        |      1 |   1632 |   1582 |00:00:09.04 |_|         .47|"
"|   4 |     NESTED LOOPS                   |        |      1 |   1632 |   1582 |00:00:06.60 |_|          .4|"
"|*  5 |      HASH JOIN                     |        |      1 |   1632 |   1582 |00:00:06.58 |_|         .34|"
"|   6 |       NESTED LOOPS                 |        |      1 |        |   1960 |00:00:05.90 |_|         .16|"
"|   7 |        NESTED LOOPS                |        |      1 |   2040 |   1960 |00:00:00.02 |_|         .16|"
"|   8 |         TABLE ACCESS BY INDEX ROWID| F      |      1 |   2040 |   1960 |00:00:00.01 |_|          .1|"
"|*  9 |          INDEX RANGE SCAN          | F IDX  |      1 |   2040 |   1960 |00:00:00.01 |_|          .1|"
"|* 10 |         INDEX UNIQUE SCAN          | E PK   |   1960 |      1 |   1960 |00:00:00.01 |_|         .05|"
"|  11 |        TABLE ACCESS BY INDEX ROWID | E      |   1960 |      1 |   1960 |00:00:06.21 |_|           0|"
"|* 12 |       TABLE ACCESS FULL            | D      |      1 |  1599K|  1600K|00:00:00.34 |_|          .1|"
"|  13 |      TABLE ACCESS BY INDEX ROWID   | C      |   1582 |      1 |   1582 |00:00:00.02 |_|         .05|"
"|* 14 |       INDEX UNIQUE SCAN            | C PK   |   1582 |      1 |   1582 |00:00:00.01 |_|         .05|"
"|  15 |     TABLE ACCESS BY INDEX ROWID    | B      |   1582 |      1 |   1582 |00:00:03.07 |_|         .05|"
"|* 16 |      INDEX UNIQUE SCAN             | B PK   |   1582 |      1 |   1582 |00:00:00.01 |_|         .05|"
"|* 17 |    TABLE ACCESS BY INDEX ROWID     | A      |   1582 |      1 |   1214 |00:00:00.89 |_|         7.4|"
"|* 18 |     INDEX UNIQUE SCAN              | A PK   |   1582 |      1 |   1582 |00:00:00.88 |_|         .05|"
"-----------------------------------------------------------------------------------------------"
```

# Confidence of Cardinality Estimates

## Practical Applications, Rapid Confidence Deterioration

➢If there is not a suitable selective predicate, force dynamic sampling




Predicates:

A: `a.str like '%AVBGG%'`
`or a.str like '%BDDRF%'`
`or a.str like '%CFFTT%')`

D: flag""<>2

# Confidence of Cardinality Estimates

## Practical Applications, Rapid Confidence Deterioration

```
select
/*+ DYNAMIC_SAMPLING ( a 3 )  */
sum( length(a.str) + length( b.str ) + length(  c.str
) + length( d.str ) + length ( e.str ) + length
(f.str) )
from A a , B b , C c , D d , E e , F f
where (a.str like '%AVBGG%'
    or a.str like '%BDDRF%'
    or a.str like '%CFFTT%')
and    a.id = b.id
and    c.id = b.id
and    c.id = d.id
and    d.flag not in (2)          <- not selective
and    e.id = d.id
and    f.id = e.id
```

# Confidence of Cardinality Estimates

## Practical Applications, Rapid Confidence Deterioration

Original Query:

```
"Plan hash value: 292659014"
" "
"-------------------------------------------------------------------------------------------
-------"
"| Id  | Operation                    | Name | Starts | E-Rows | A-Rows |   A-Time    |_"
"-------------------------------------------------------------------------------------------
-------"
"|   0 | SELECT STATEMENT             |      |      2 |        |      2 |00:00:14.33 |_"
"|   1 |  SORT AGGREGATE              |      |      2 |      1 |      2 |00:00:14.33 |_"
"|*  2 |   HASH JOIN                  |      |      2 |   228K|     62 |00:00:14.    |_"
"|*  3 |    HASH JOIN                 |      |      2 |   228K|     62 |00:00:12.09 |_"
"|*  4 |     HASH JOIN                |      |      2 |   228K|     62 |00:00:09.86 |_"
"|*  5 |      TABLE ACCESS FULL       | D    |      2 |  1599K|  3200K|00:00:00.59 |_"
"|*  6 |      HASH JOIN               |      |      2 |   285K|     74 |00:00:07.25 |_"
"|*  7 |       HASH JOIN              |      |      2 |   285K|     74 |00:00:04.92 |_"
"|*  8 |        TABLE ACCESS FULL     | A    |      2 |   285K|     74 |00:00:02.61 |_"
"|   9 |        TABLE ACCESS FULL     | B    |      2 |  1999K|  3999K|00:00:00.76 |_"
"|  10 |       TABLE ACCESS FULL      | C    |      2 |  1999K|  3999K|00:00:00.76 |_"
"|  11 |     TABLE ACCESS FULL        | E    |      2 |  1999K|  3999K|00:00:00.72 |_"
"|  12 |    TABLE ACCESS FULL         | F    |      2 |  1999K|  3999K|00:00:00.74 |_"
"-------------------------------------------------------------------------------------------
-------"
```

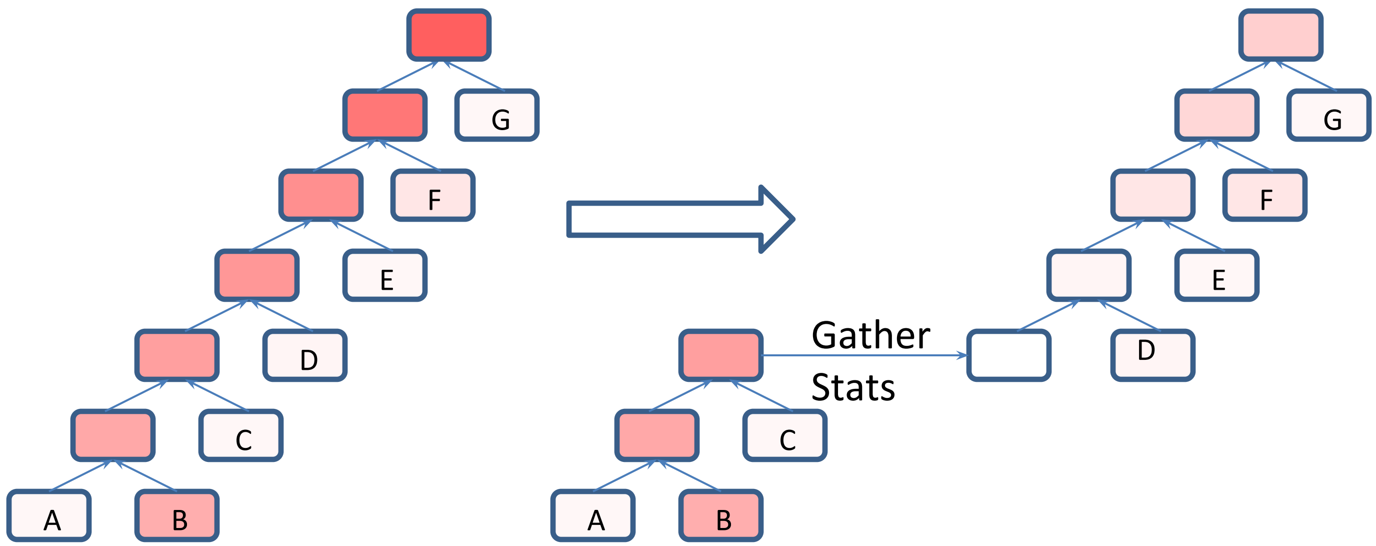
Query with Dynamic Sampling `DYNAMIC_SAMPLING ( a 3 ):`

```
"Plan hash value: 850278183"
" "
"-----------------------------------------------------------------------------------------------
"| Id  | Operation                          | Name | Starts | E-Rows | A-Rows |   A-Time    |..
"-----------------------------------------------------------------------------------------------
"|   0 | SELECT STATEMENT                   |      |      1 |        |      1 |00:00:01.33 |..
"|   1 |  SORT AGGREGATE                    |      |      1 |      1 |      1 |00:00:01.33 |..
"|   2 |   NESTED LOOPS                     |      |      1 |    137 |     31 |00:00:01.33 |..
"|   3 |    NESTED LOOPS                    |      |      1 |    137 |     31 |00:00:01.33 |..
"|   4 |     NESTED LOOPS                   |      |      1 |    137 |     31 |00:00:01.33 |..
"|   5 |      NESTED LOOPS                  |      |      1 |    171 |     37 |00:00:01.33 |..
"|   6 |       NESTED LOOPS                 |      |      1 |    171 |     37 |00:00:01.33 |..
"|*  7 |        TABLE ACCESS FULL           | A    |      1 |    171 |     37 |00:00:01.33 |..
"|   8 |        TABLE ACCESS BY INDEX ROWID | B    |     37 |      1 |     37 |00:00:00.01 |..
"|*  9 |         INDEX UNIQUE SCAN          | B PK |     37 |      1 |     37 |00:00:00.01 |..
"|  10 |       TABLE ACCESS BY INDEX ROWID  | C    |     37 |      1 |     37 |00:00:00.01 |..
"|* 11 |        INDEX UNIQUE SCAN           | C PK |     37 |      1 |     37 |00:00:00.01 |..
"|* 12 |      TABLE ACCESS BY INDEX ROWID   | D    |     37 |      1 |     31 |00:00:00.01 |..
"|* 13 |       INDEX UNIQUE SCAN            | D PK |     37 |      1 |     37 |00:00:00.01 |..
"|  14 |     TABLE ACCESS BY INDEX ROWID    | E    |     31 |      1 |     31 |00:00:00.01 |..
"|* 15 |      INDEX UNIQUE SCAN             | E PK |     31 |      1 |     31 |00:00:00.01 |..
"|  16 |    TABLE ACCESS BY INDEX ROWID     | F    |     31 |      1 |     31 |00:00:00.01 |..
"|* 17 |     INDEX UNIQUE SCAN              | F PK |     31 |      1 |     31 |00:00:00.01 |..
"-----------------------------------------------------------------------------------------------
```

# Confidence of Cardinality Estimates

## Practical Applications, Rapid Confidence Deterioration

- In rare cases, when the above methods are not appropriate, split the single SQL into multiple SQL



# Conclusion

Ask not what the optimizer can do for you - ask what you can do for the optimizer…

Huge SQL statements are not the solution to our problem, huge SQL statements are the problem...

Cool new features – trust, but verify…