

Safe Harbor

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, timing, and pricing of any features or functionality described for Oracle's products may change and remains at the sole discretion of Oracle Corporation.

Statements in this presentation relating to Oracle's future plans, expectations, beliefs, intentions and prospects are "forward-looking statements" and are subject to material risks and uncertainties. A detailed discussion of these factors and other risks that affect our business is contained in Oracle's Securities and Exchange Commission (SEC) filings, including our most recent reports on Form 10-K and Form 10-Q under the heading "Risk Factors." These filings are available on the SEC's website or on Oracle's website at <http://www.oracle.com/investor>. All information in this presentation is current as of September 2019 and Oracle undertakes no duty to update any statement in light of new information or future events.



Real-World Performance Techniques for Extreme Data Warehousing

Robert Carlin

Mihajlo Tekic

Real-World Performance
Oracle Database Development



What is Real-World Performance?

- Getting the most out of Software and Hardware
- Achieving Performance Excellence

Real-World Performance Team



Who We Are



- Part of Oracle Database Development
 - Team members in USA, Europe, and Asia
 - Over a hundred years of experience combined
- 

How We Work

- Use the product as designed
- Take a holistic view
- Aim for best performance
- Apply data-driven analysis
- Share what we learn

What we do



Customer Engagements

- Design Review
 - Escalations
 - Performance Projects
- 



Database Development

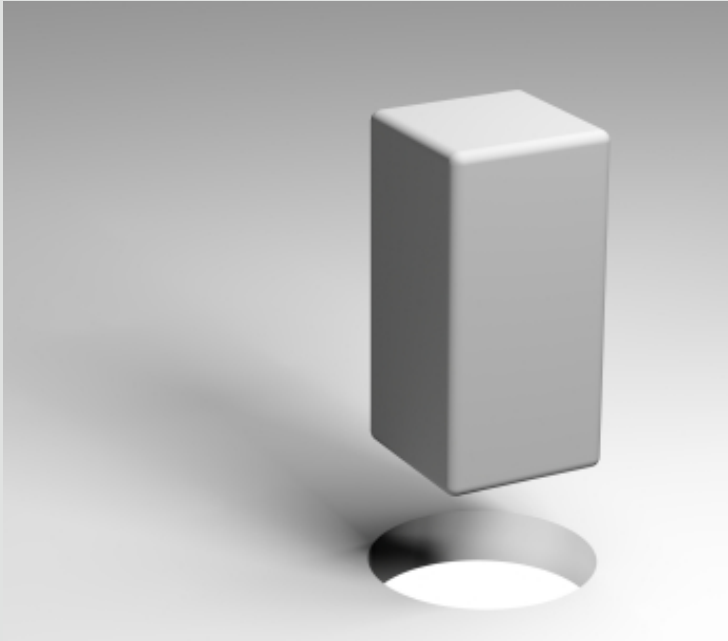
- Tools
- Applications



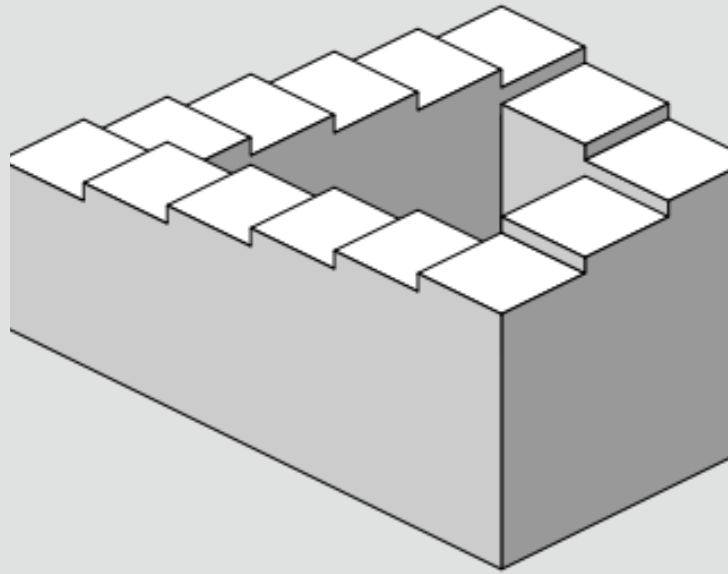
Customer Education

- In Person
- Online

Root Causes of Suboptimal Database Performance



The database is not being used as it was designed to be used



The application architecture/code design is suboptimal



There is a suboptimal algorithm in the database

Data Warehousing Performance

- Oracle has enhanced its data warehousing capabilities dramatically over the last decade

Exadata

Database In-Memory

Smart Scans

Bloom Filtering

Vector Processing

Storage Indexes

Columnar Storage

Data Warehousing Performance

- Many older Data Warehouse projects were built with an OLTP mindset

Over indexing

Home grown parallelism

- Understanding of Data Warehousing concepts

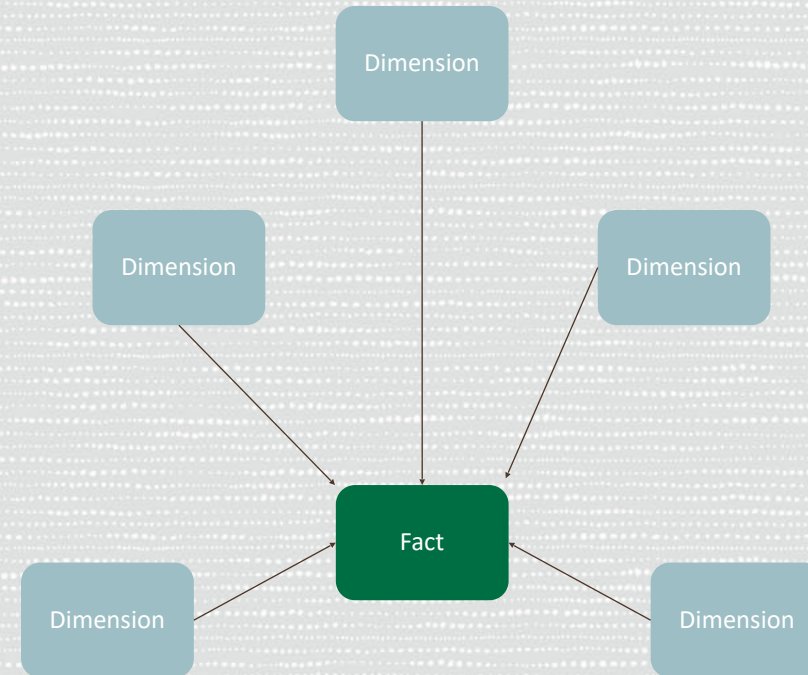
Storage IO Bandwidth

Parallel Processing

Set-based processing techniques

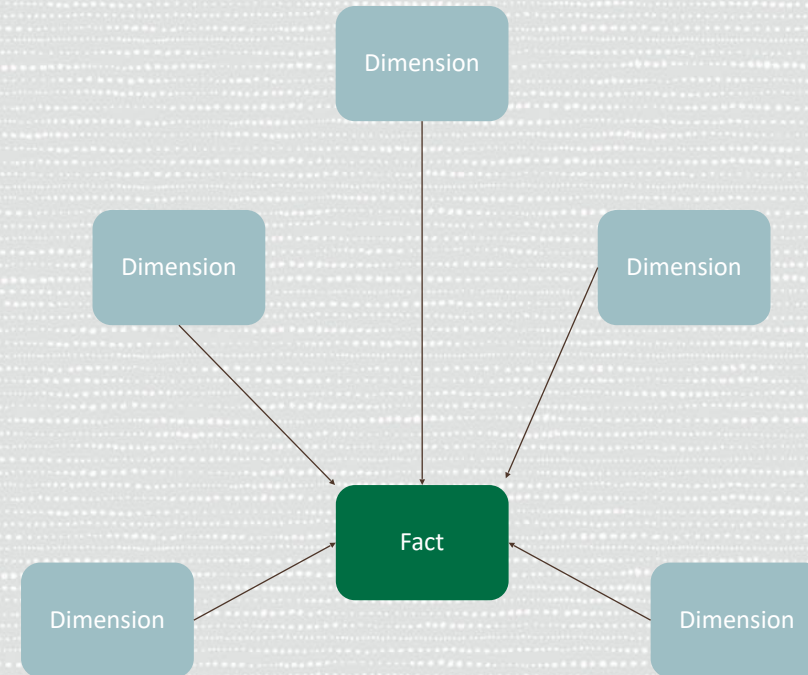
Data Model

- Ideal data model is Star Schema



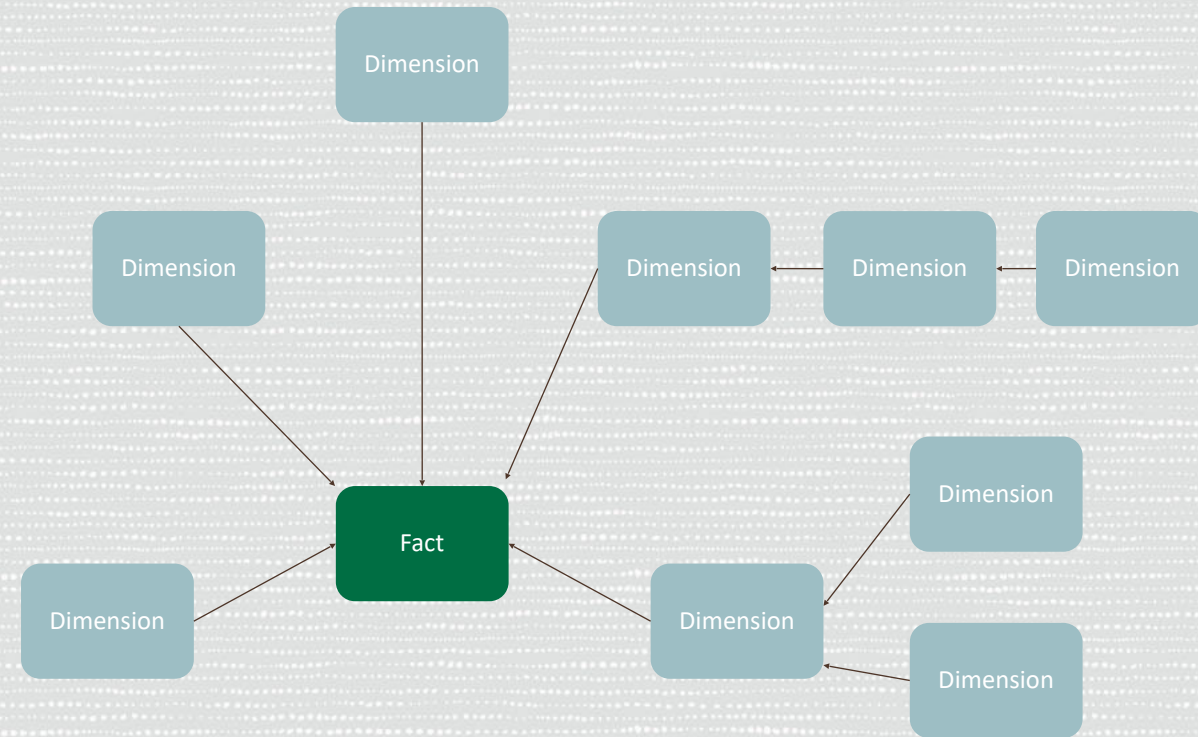
Data Model

- Ideal data model is Star Schema



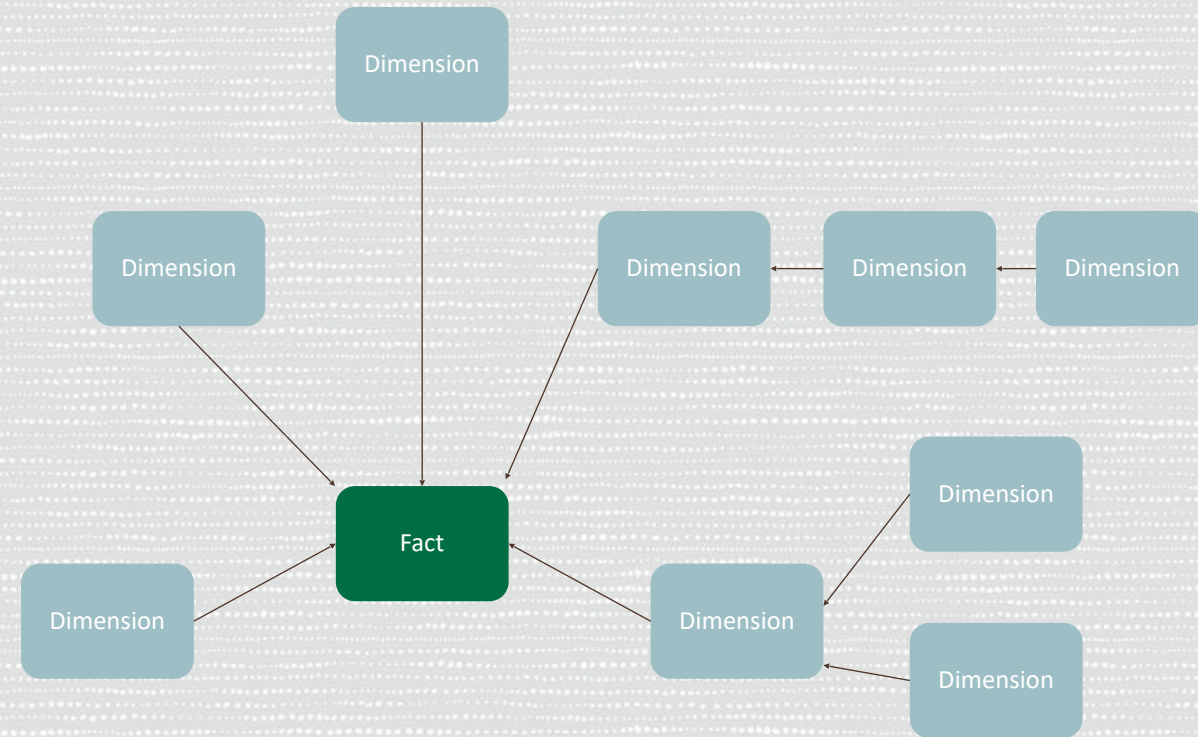
Data Model

- Ideal data model is Star Schema
- Or a derivative Snowflake Schema



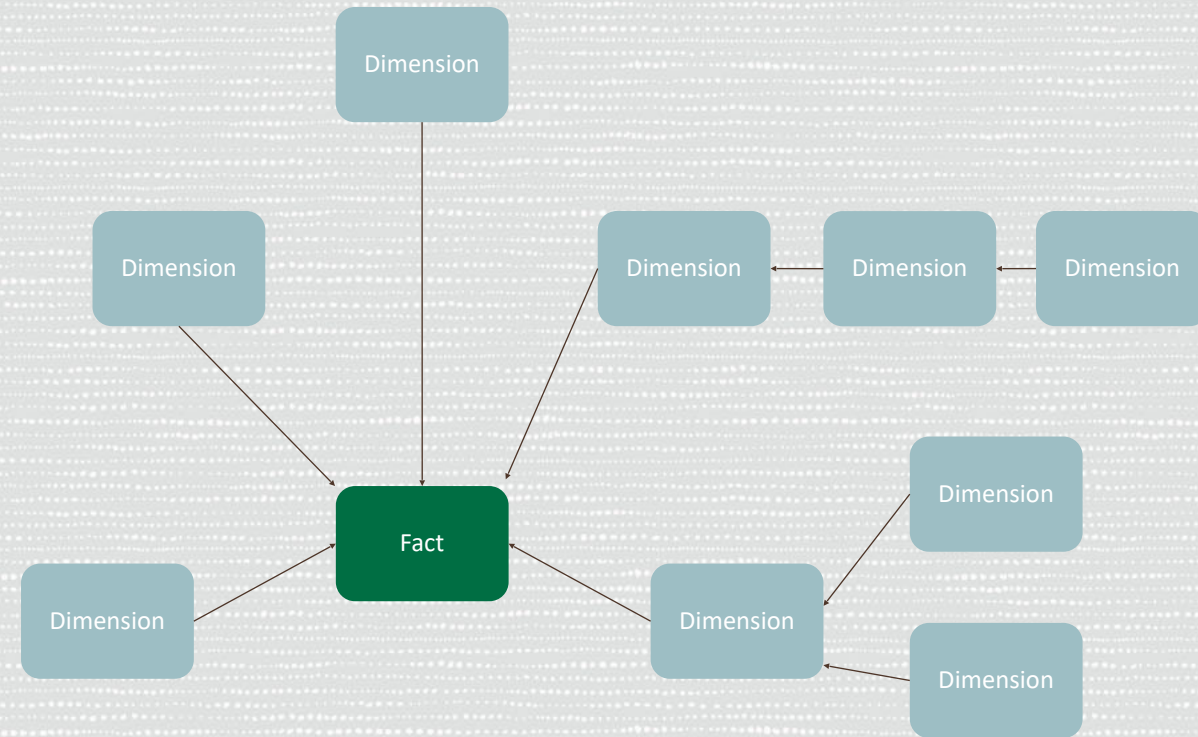
Data Model

- Ideal data model is Star Schema
- Or a derivative Snowflake Schema
- Well understood design pattern



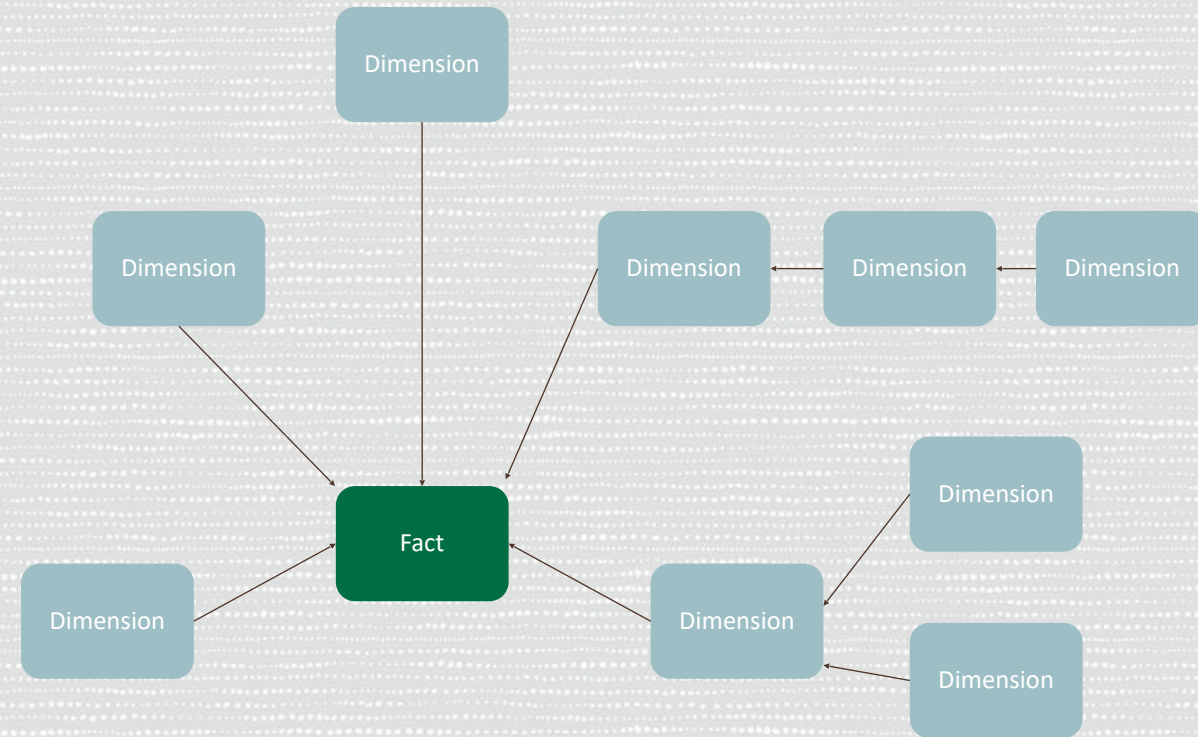
Data Model

- Ideal data model is Star Schema
- Or a derivative Snowflake Schema
- Well understood design pattern
- Proven to scale well



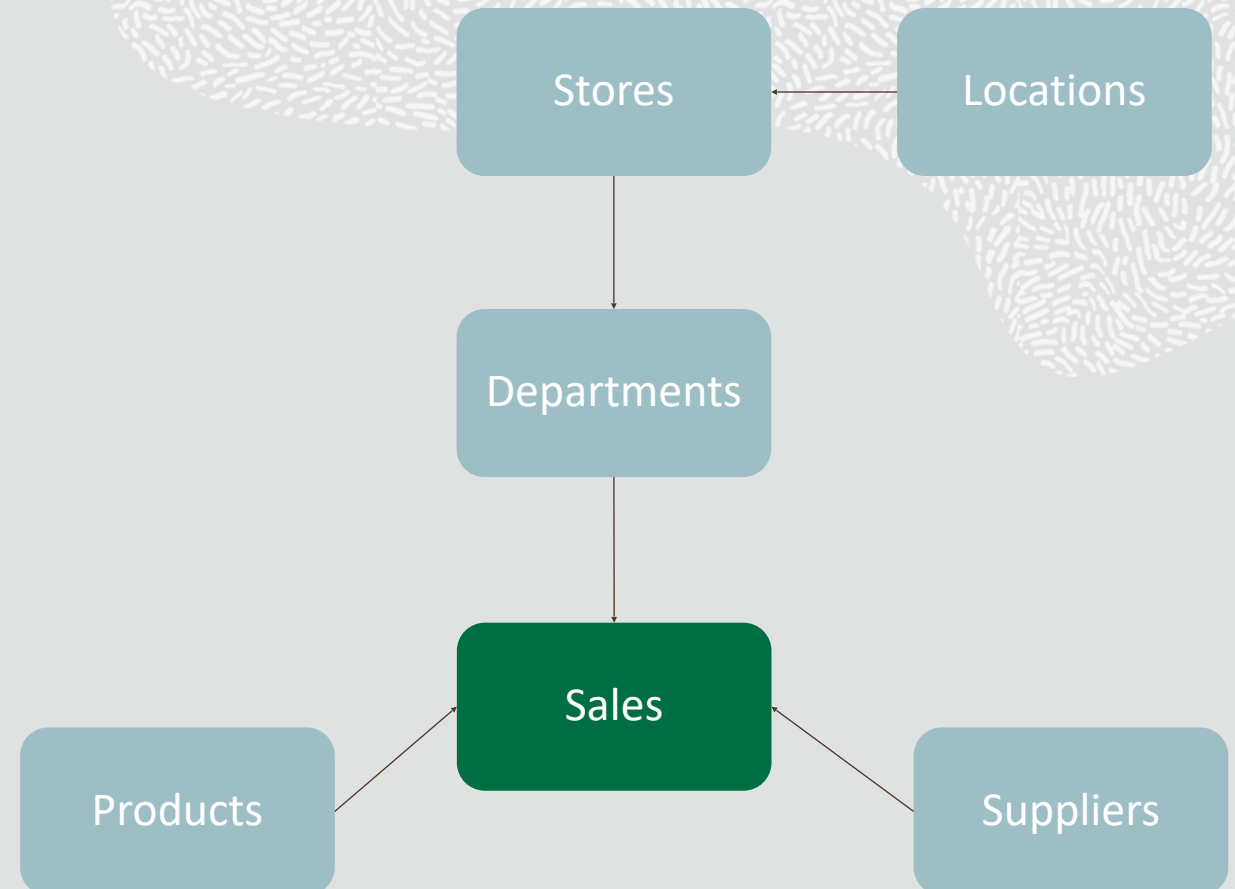
Data Model

- Ideal data model is Star Schema
- Or a derivative Snowflake Schema
- Well understood design pattern
- Proven to scale well
- Works well with query and ETL tools



Schema and SQL Statement

```
SELECT
  CATEGORY_ID
, COUNTRY
, SUM(CTRL) as CTRL
, SUM(QUANTITY) AS S_Q
FROM
  (SELECT
    P.CATEGORY_ID
  , SP.COUNTRY
  , CASE
      WHEN SP.CATEGORY='CAT_' || P.CATEGORY_ID
      THEN 1
      ELSE 0
    END as CTRL
  , QUANTITY
FROM
  SALES SL
  INNER JOIN DEPARTMENTS D ON SL.DEPARTMENT_ID=D.DEPARTMENT_ID
  INNER JOIN STORES S ON D.STORE_ID=S.STORE_ID
  INNER JOIN LOCATIONS L ON S.LOCATION_ID=L.LOCATION_ID
  INNER JOIN PRODUCTS P ON SL.PRODUCT_ID=P.PRODUCT_ID
  INNER JOIN SUPPLIERS SP ON SL.SUPPLIER_ID=SP.SUPPLIER_ID
  WHERE (L.CITY) IN ('CITY_1','CITY_2','CITY_3')
  AND S.STORE_TYPE=0
  )
GROUP BY CATEGORY_ID, COUNTRY
```

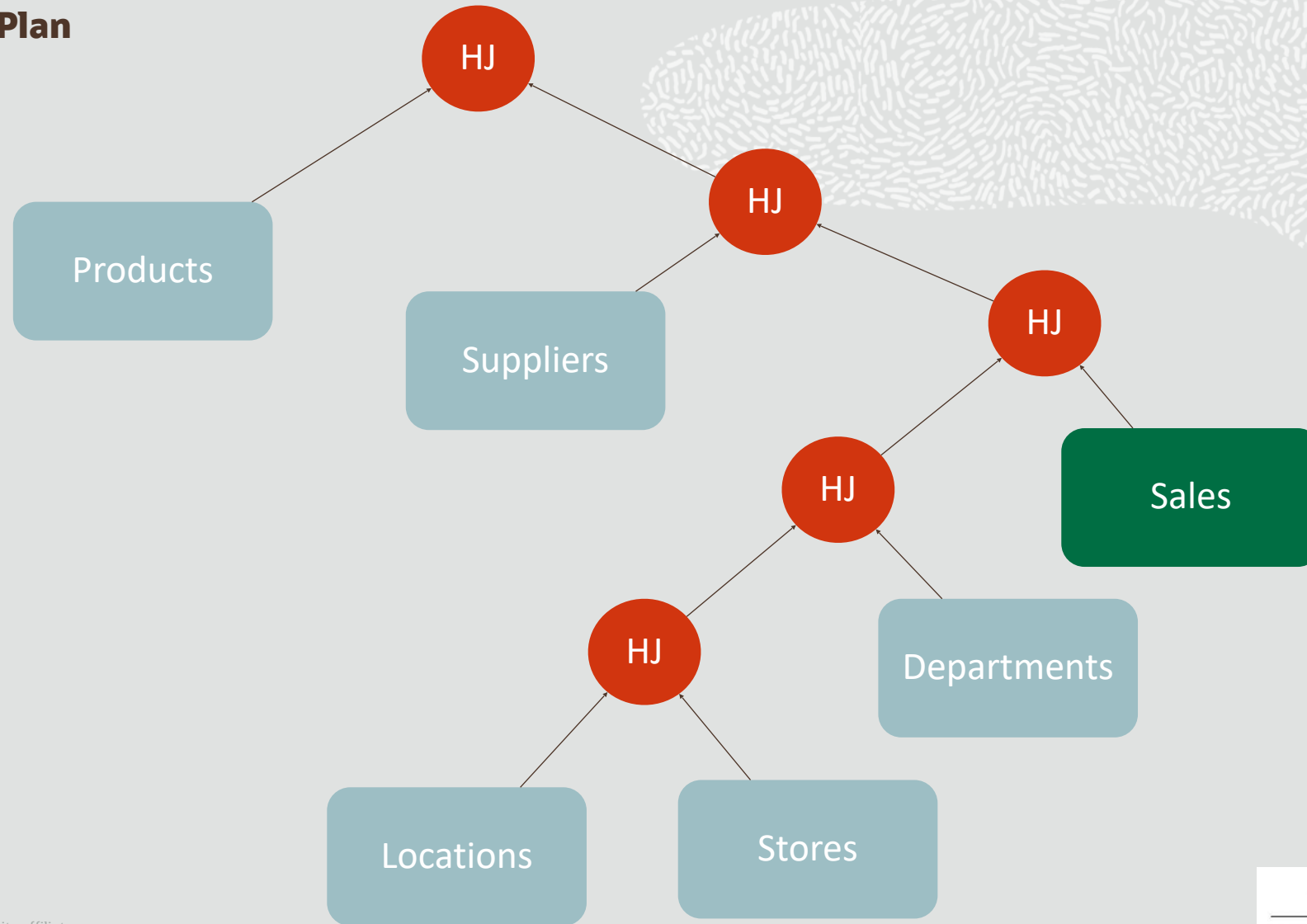


Dimension Model

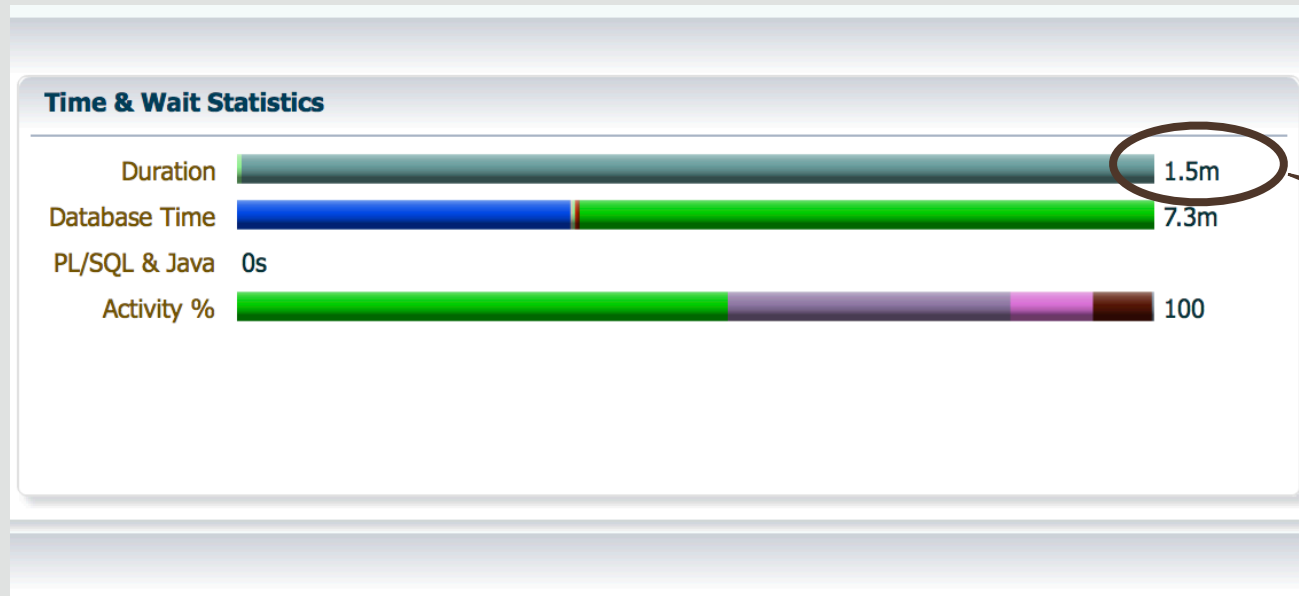
- The advantage of the dimensional data model is that the desired execution plan is predictable
- Knowing the data sizes, we would expect our query to return in 10s or less

We should at least be able to estimate the data acquisition phase

Desired Execution Plan



Query Execution: Running poorly



89 Seconds

What do we do to fix it?



What do we do to fix it?

- Update the statistics?

What do we do to fix it?



- Update the statistics?
- Histograms?

What do we do to fix it?

- Update the statistics?
- Histograms?
- Column Groups?

What do we do to fix it?

- Update the statistics?
- Histograms?
- Column Groups?
- Increase the degree of parallelism?

What do we do to fix it?

- Update the statistics?
- Histograms?
- Column Groups?
- Increase the degree of parallelism?
- Set `optimizer_index_cost_adj`?

What do we do to fix it?

- Update the statistics?
- Histograms?
- Column Groups?
- Increase the degree of parallelism?
- Set `optimizer_index_cost_adj`?
- Set `cursor_sharing = FORCE`?

What do we do to fix it?

- Update the statistics?
- Histograms?
- Column Groups?
- Increase the degree of parallelism?
- Set `optimizer_index_cost_adj`?
- Set `cursor_sharing = FORCE`?
- Increase the block size?

What do we do to fix it?

- Update the statistics?
- Histograms?
- Column Groups?
- Increase the degree of parallelism?
- Set `optimizer_index_cost_adj`?
- Set `cursor_sharing = FORCE`?
- Increase the block size?
- Google for some magic hidden parameter?

What do we do to fix it?

- Update the
- Histogram
- Column
- Increase the degree of parallelis
- Set opt
- Set curs
- Increase the
- Google for some hidden parameter?

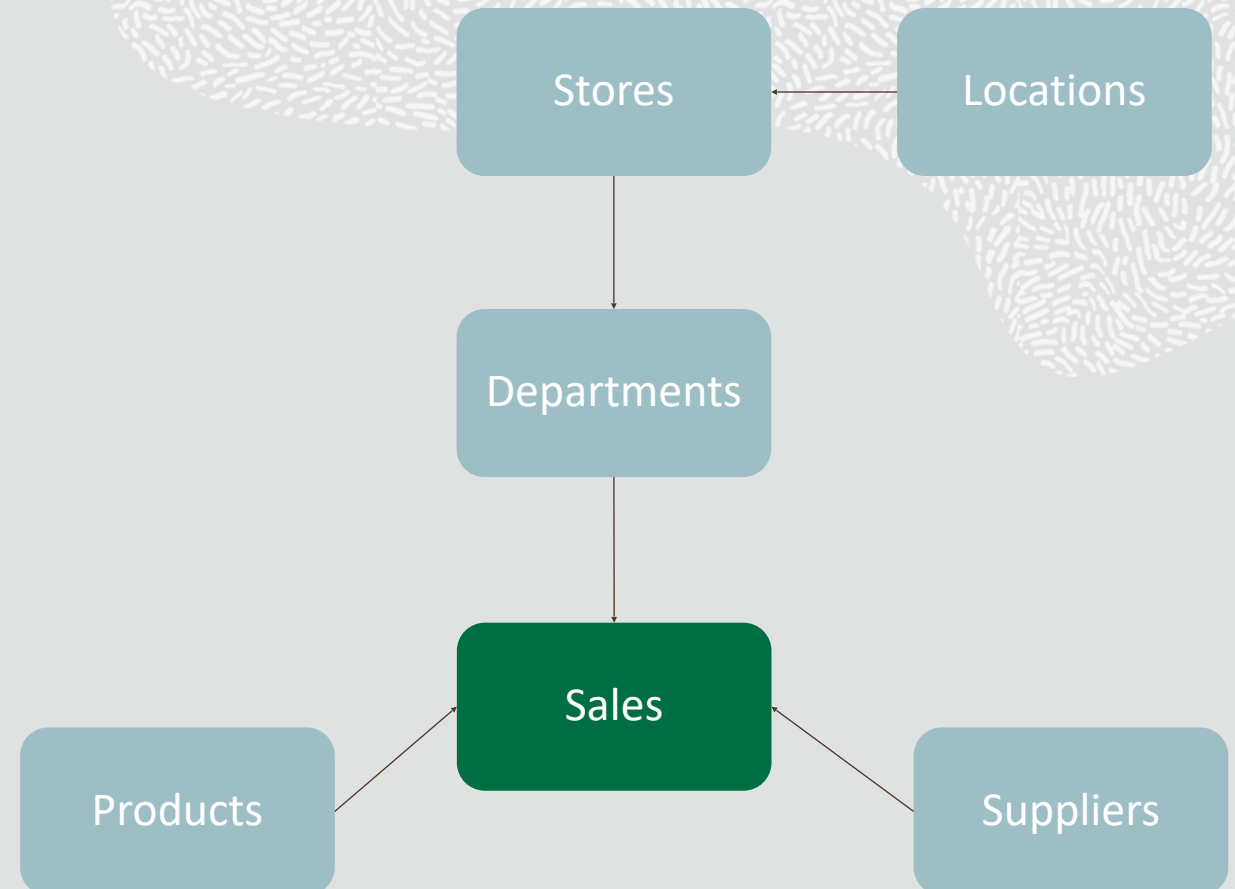
“Let’s work the problem people; let’s not make things worse by guessing”

Gene Kranz
Flight Director Apollo 13



Schema and SQL Statement

```
SELECT
  CATEGORY_ID
, COUNTRY
, SUM(CTRL) as CTRL
, SUM(QUANTITY) AS S_Q
FROM
  (SELECT
    P.CATEGORY_ID
  , SP.COUNTRY
  , CASE
      WHEN SP.CATEGORY='CAT_' || P.CATEGORY_ID
      THEN 1
      ELSE 0
    END as CTRL
  , QUANTITY
FROM
  SALES SL
  INNER JOIN DEPARTMENTS D ON SL.DEPARTMENT_ID=D.DEPARTMENT_ID
  INNER JOIN STORES S ON D.STORE_ID=S.STORE_ID
  INNER JOIN LOCATIONS L ON S.LOCATION_ID=L.LOCATION_ID
  INNER JOIN PRODUCTS P ON SL.PRODUCT_ID=P.PRODUCT_ID
  INNER JOIN SUPPLIERS SP ON SL.SUPPLIER_ID=SP.SUPPLIER_ID
  WHERE (L.CITY) IN ('CITY_1','CITY_2','CITY_3')
  AND S.STORE_TYPE=0
  )
GROUP BY CATEGORY_ID, COUNTRY
```



Demo

Baseline Run

Monitored SQL Execution Details: **azmv8ma1147ff** Navigate to SQL Details Save Page Refreshed **5:15:31 PM GMT-0700**

Overview

General

SQL Text

SELECT /* q15 */ CATEGORY_ID , COUNTRY , SUM(C

...

Execution Plan

4

Execution Started

Sun Sep 15, 2019 5:13:48 PM

Last Refresh Time

Sun Sep 15, 2019 5:15:17 PM

Execution ID

33554432

User

MTEKIC@PDB1

Fetch Calls

1

Time & Wait Statistics

Duration

1.5m

Database Time

7.3m

PL/SQL & Java

0s

Activity %

100

IO Statistics

Buffer Gets

50M

IO Requests

6,421K

IO Bytes

136GB

Cell Offload Efficiency

98%

Details

Plan Statistics

Plan

Parallel

Activity

Metrics

Plan Hash Value

1155856492

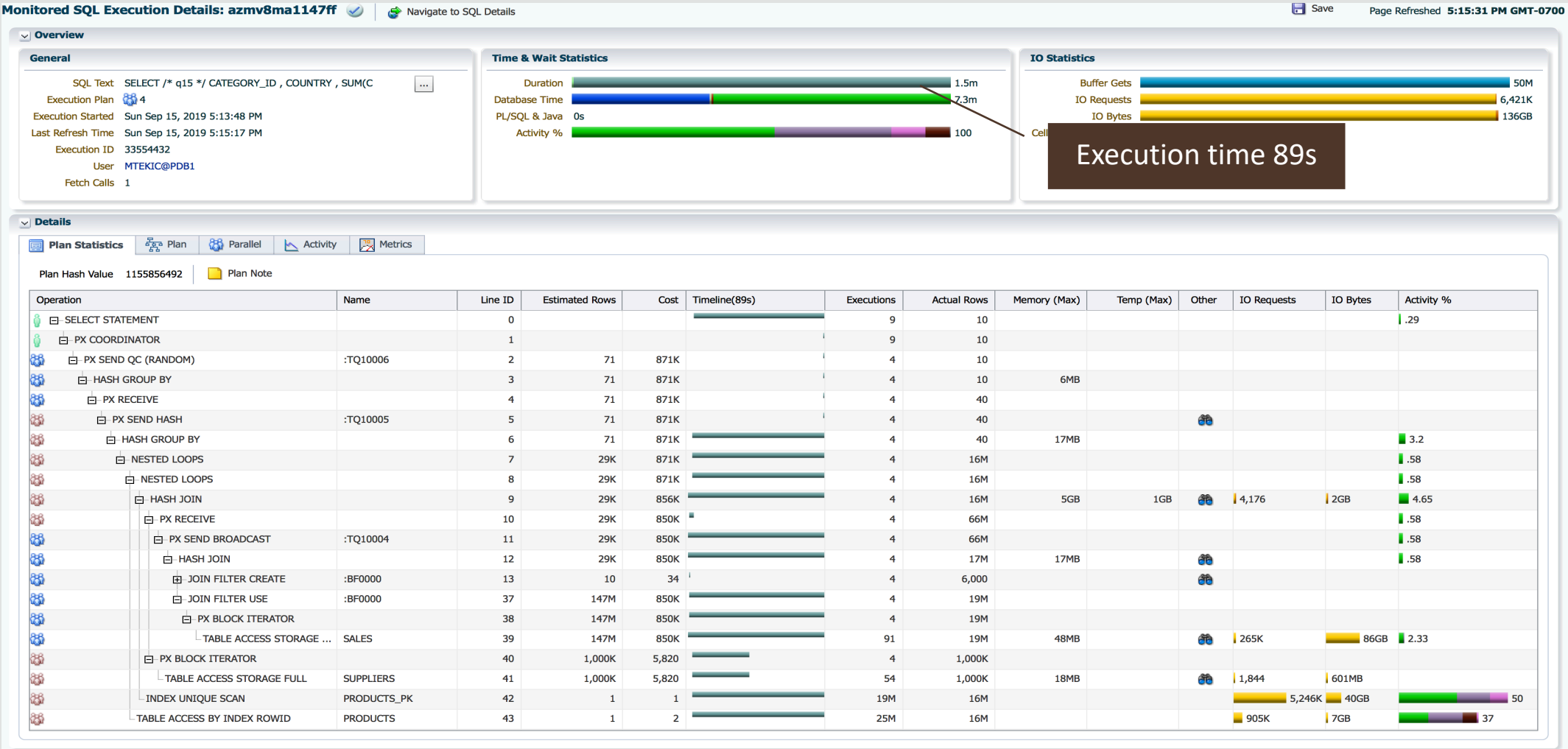
Plan Note

Operation	Name	Line ID	Estimated Rows	Cost	Timeline(89s)	Executions	Actual Rows	Memory (Max)	Temp (Max)	Other	IO Requests	IO Bytes	Activity %
SELECT STATEMENT		0				9	10						.29
PX COORDINATOR		1				9	10						
PX SEND QC (RANDOM)	:TQ10006	2	71	871K		4	10						
HASH GROUP BY		3	71	871K		4	10	6MB					
PX RECEIVE		4	71	871K		4	40						
PX SEND HASH	:TQ10005	5	71	871K		4	40						
HASH GROUP BY		6	71	871K		4	40	17MB					3.2
NESTED LOOPS		7	29K	871K		4	16M						.58
NESTED LOOPS		8	29K	871K		4	16M						.58
HASH JOIN		9	29K	856K		4	16M	5GB	1GB		4,176	2GB	4.65
PX RECEIVE		10	29K	850K		4	66M						.58
PX SEND BROADCAST	:TQ10004	11	29K	850K		4	66M						.58
HASH JOIN		12	29K	850K		4	17M	17MB					.58
JOIN FILTER CREATE	:BF0000	13	10	34		4	6,000						
JOIN FILTER USE	:BF0000	37	147M	850K		4	19M						
PX BLOCK ITERATOR		38	147M	850K		4	19M						
TABLE ACCESS STORAGE ...	SALES	39	147M	850K		91	19M	48MB			265K	86GB	2.33
PX BLOCK ITERATOR		40	1,000K	5,820		4	1,000K						
TABLE ACCESS STORAGE FULL	SUPPLIERS	41	1,000K	5,820		54	1,000K	18MB			1,844	601MB	
INDEX UNIQUE SCAN	PRODUCTS_PK	42	1	1		19M	16M				5,246K	40GB	50
TABLE ACCESS BY INDEX ROWID	PRODUCTS	43	1	2		25M	16M				905K	7GB	37

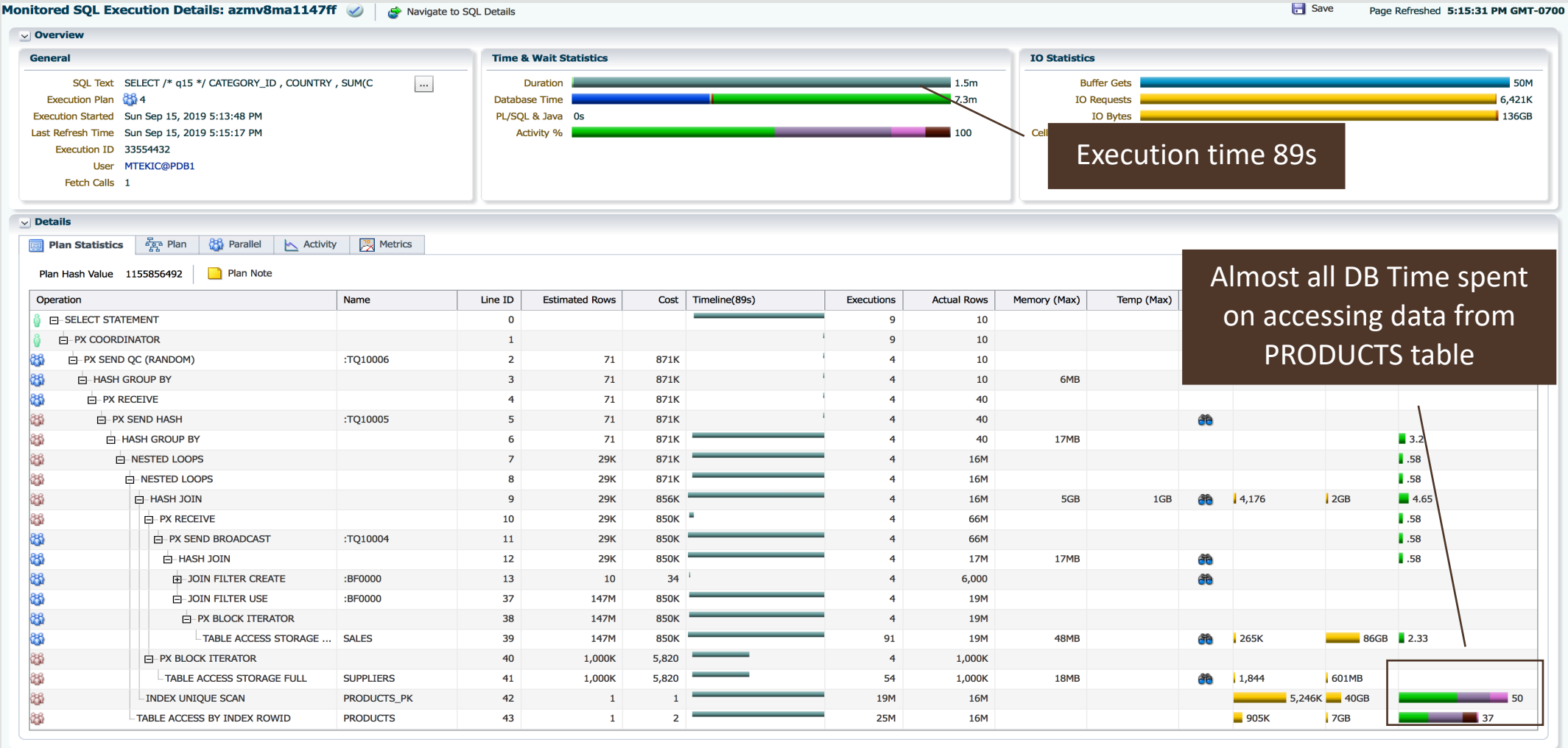
ORACLE®

REAL-WORLD PERFORMANCE

Baseline Run



Baseline Run



Baseline Run

Monitored SQL Execution Details: azmv8ma1147ff

[Navigate to SQL Details](#)

[Save](#)

Page Refreshed 5:15:31 PM GMT-0700

Overview

General

SQL Text: `SELECT /* q15 */ CATEGORY_ID , COUNTRY , SUM(C`

Execution Plan: 4

Execution Started: Sun Sep 15, 2019 5:13:48 PM

Last Refresh Time: Sun Sep 15, 2019 5:15:17 PM

Execution ID: 33554432

User: MTEKIC@PDB1

Fetch Calls: 1

Time & Wait Statistics

Duration: 1.5m

Database Time: 7.3m

PL/SQL & Java: 0s

Activity %: 100

IO Statistics

Buffer Gets: 50M

IO Requests: 6,421K

IO Bytes: 136GB

Execution time 89s

Nested Loops join when
joining PRODUCTS table
Why?

Almost all DB Time spent
on accessing data from
PRODUCTS table

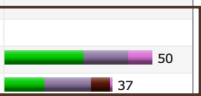
Details

Plan Statistics

Plan Hash Value: 1155856492

Plan Note

Operation	Name	Line ID	Estimated Rows	Cost	Timeline(89s)	Executions	Actual Rows	Memory (Max)	Temp (Max)
SELECT STATEMENT		0				9	10		
PX COORDINATOR		1				9	10		
PX SEND QC (RANDOM)	:TQ10006	2	71	871K		4	10		
HASH GROUP BY		3	71	871K		4	10	6MB	
PX RECEIVE		4	71	871K		4	40		
PX SEND HASH	:TQ10005	5	71	871K		4	40		
HASH GROUP BY		6	71	871K		4	40	17MB	
NESTED LOOPS		7	29K	871K		4	16M		3.2
NESTED LOOPS		8	29K	871K		4	16M		.58
HASH JOIN		9	29K	856K		4	16M	5GB	1GB
PX RECEIVE		10	29K	850K		4	66M		4.65
PX SEND BROADCAST	:TQ10004	11	29K	850K		4	66M		.58
HASH JOIN		12	29K	850K		4	17M	17MB	.58
JOIN FILTER CREATE	:BF0000	13	10	34		4	6,000		.58
JOIN FILTER USE	:BF0000	37	147M	850K		4	19M		
PX BLOCK ITERATOR		38	147M	850K		4	19M		
TABLE ACCESS STORAGE ...	SALES	39	147M	850K		91	19M	48MB	265K
PX BLOCK ITERATOR		40	1,000K	5,820		4	1,000K		86GB
TABLE ACCESS STORAGE FULL	SUPPLIERS	41	1,000K	5,820		54	1,000K	18MB	1,844
INDEX UNIQUE SCAN	PRODUCTS_PK	42	1	1		19M	16M		5,246K
TABLE ACCESS BY INDEX ROWID	PRODUCTS	43	1	2		25M	16M		905K



Baseline Run

Monitored SQL Execution Details: azmv8ma1147ff

Navigate to SQL Details

Save

Page Refreshed 5:15:31 PM GMT-0700

Overview

General

SQL Text: `SELECT /* q15 */ CATEGORY_ID , COUNTRY , SUM(C`

Execution Plan: 4

Execution Started: Sun Sep 15, 2019 5:13:48 PM

Last Refresh Time: Sun Sep 15, 2019 5:15:17 PM

Execution ID: 33554432

User: MTEKIC@PDB1

Fetch Calls: 1

Time & Wait Statistics

Duration: 1.5m

Database Time: 7.3m

PL/SQL & Java: 0s

Activity %: 100

IO Statistics

Buffer Gets: 50M

IO Requests: 6,421K

IO Bytes: 136GB

Execution time 89s

Nested Loops join when
joining PRODUCTS table
Why?

Estimated cardinality for
row source operation 9 is
18K

Almost all DB Time spent
on accessing data from
PRODUCTS table

Details

Plan Statistics

Plan Hash Value: 1155856492

Plan Note

Operation	Name	Line ID	Estimated Rows	Memory (Max)	Temp (Max)
SELECT STATEMENT		0			
PX COORDINATOR		1			
PX SEND QC (RANDOM)	:TQ10006	2	71		
HASH GROUP BY		3	71	6MB	
PX RECEIVE		4	71		
PX SEND HASH	:TQ10005	5	71		
HASH GROUP BY		6	71	17MB	
NESTED LOOPS		7	29K		
NESTED LOOPS		8	29K		
HASH JOIN		9	29K	5GB	1GB
PX RECEIVE		10	29K	66M	
PX SEND BROADCAST	:TQ10004	11	29K	66M	
HASH JOIN		12	29K	17MB	
JOIN FILTER CREATE	:BF0000	13	10	6,000	
JOIN FILTER USE	:BF0000	37	147M	19M	
PX BLOCK ITERATOR		38	147M	19M	
TABLE ACCESS STORAGE ...	SALES	39	147M	48MB	
PX BLOCK ITERATOR		40	1,000K		
TABLE ACCESS STORAGE FULL	SUPPLIERS	41	1,000K	18MB	
INDEX UNIQUE SCAN	PRODUCTS_PK	42	1	16M	
TABLE ACCESS BY INDEX ROWID	PRODUCTS	43	1	16M	

Baseline Run

Monitored SQL Execution Details: azmv8ma1147ff

Navigate to SQL Details

Save

Page Refreshed 5:15:31 PM GMT-0700

Overview

General

SQL Text: `SELECT /* q15 */ CATEGORY_ID , COUNTRY , SUM(C`

Execution Plan: 4

Execution Started: Sun Sep 15, 2019 5:13:48 PM

Last Refresh Time: Sun Sep 15, 2019 5:15:17 PM

Execution ID: 33554432

User: MTEKIC@PDB1

Fetch Calls: 1

Time & Wait Statistics

Duration: 1.5m

Database Time: 7.3m

PL/SQL & Java: 0s

Activity %: 100

IO Statistics

Buffer Gets: 50M

IO Requests: 6,421K

IO Bytes: 136GB

Execution time 89s

Nested Loops join when
joining PRODUCTS table
Why?

Estimated cardinality for
row source operation 9 is
29K, Actual 16M

Almost all DB Time spent
on accessing data from
PRODUCTS table

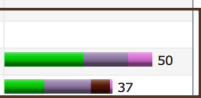
Details

Plan Statistics

Plan Hash Value: 1155856492

Plan Note

Operation	Name	Line ID	Estimated Rows	Memory (Max)	Temp (Max)
SELECT STATEMENT		0			
PX COORDINATOR		1			
PX SEND QC (RANDOM)	:TQ10006	2	71		
HASH GROUP BY		3	71	6MB	
PX RECEIVE		4	71		
PX SEND HASH	:TQ10005	5	71		
HASH GROUP BY		6	71	17MB	
NESTED LOOPS		7	29K	16M	
NESTED LOOPS		8	29K	16M	
HASH JOIN		9	29K	5GB	1GB
PX RECEIVE		10	29K	66M	
PX SEND BROADCAST	:TQ10004	11	29K	66M	
HASH JOIN		12	29K	17MB	
JOIN FILTER CREATE	:BF0000	13	10	6,000	
JOIN FILTER USE	:BF0000	37	147M	19M	
PX BLOCK ITERATOR		38	147M	19M	
TABLE ACCESS STORAGE ...	SALES	39	147M	48MB	
PX BLOCK ITERATOR		40	1,000K	1,000K	
TABLE ACCESS STORAGE FULL	SUPPLIERS	41	1,000K	18MB	
INDEX UNIQUE SCAN	PRODUCTS_PK	42	1	16M	
TABLE ACCESS BY INDEX ROWID	PRODUCTS	43	1	25M	



Baseline Run

Monitored SQL Execution Details: azmv8ma1147ff

Navigate to SQL Details

Save

Page Refreshed 5:15:31 PM GMT-0700

Overview

General

SQL Text: `SELECT /* q15 */ CATEGORY_ID , COUNTRY , SUM(C`

Execution Plan: 4

Execution Started: Sun Sep 15, 2019 5:13:48 PM

Last Refresh Time: Sun Sep 15, 2019 5:15:17 PM

Execution ID: 33554432

User: MTEKIC@PDB1

Fetch Calls: 1

Time & Wait Statistics

Duration: 1.5m

Database Time: 7.3m

PL/SQL & Java: 0s

Activity %: 100

IO Statistics

Buffer Gets: 50M

IO Requests: 6,421K

IO Bytes: 136GB

Execution time 89s

Nested Loops join when
joining PRODUCTS table
Why?

Estimated cardinality for
row source operation 9 is
29K, Actual 16M

Almost all DB Time spent
on accessing data from
PRODUCTS table

Details

Plan Statistics

Plan Hash Value: 1155856492

Plan Note

Operation	Name	Line ID	Estimated Rows	Memory (Max)	Temp (Max)
SELECT STATEMENT		0			
PX COORDINATOR		1			
PX SEND QC (RANDOM)	:TQ10006	2	71		
HASH GROUP BY		3	71	6MB	
PX RECEIVE		4	71		
PX SEND HASH	:TQ10005	5	71		
HASH GROUP BY		6	71	17MB	
NESTED LOOPS		7	29K	16M	
NESTED LOOPS		8	29K	16M	
HASH JOIN		9	29K	5GB	1GB
PX RECEIVE		10	29K	66M	
PX SEND BROADCAST	:TQ10004	11	29K	66M	
HASH JOIN		12	29K	17MB	
JOIN FILTER CREATE	:BF0000	13	10	6,000	
JOIN FILTER USE	:BF0000	37	147M	19M	
PX BLOCK ITERATOR		38	147M	19M	
TABLE ACCESS STORAGE ...	SALES	39	147M	48MB	
PX BLOCK ITERATOR		40	1,000K	1,000K	
TABLE ACCESS STORAGE FULL	SUPPLIERS	41	1,000K	18MB	
INDEX UNIQUE SCAN	PRODUCTS_PK	42	1	19M	16M
TABLE ACCESS BY INDEX ROWID	PRODUCTS	43			

The underestimate
originates from row source
13

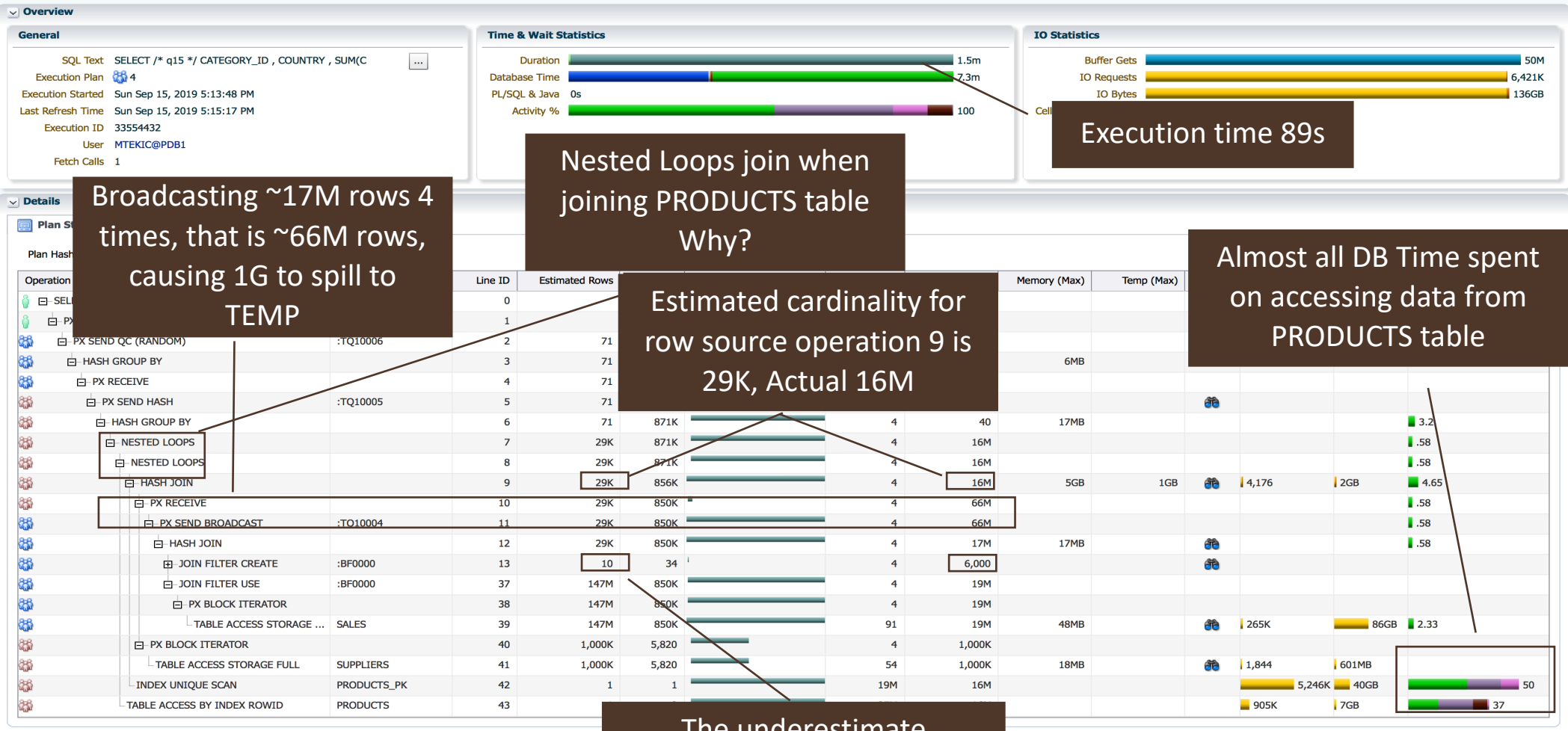
Baseline Run

Monitored SQL Execution Details: azmv8ma1147ff

Navigate to SQL Details

Save

Page Refreshed 5:15:31 PM GMT-0700



Baseline Run

Plan Statistics													
Plan													
Parallel													
Activity													
Metrics													
Plan Hash Value 1155856492													
Plan Note													
Operation	Name	Line ID	Estimated Rows	Cost	Timeline(89s)	Executions	Actual Rows	Memory (Max)	Temp (Max)	Ot...	IO Requests	IO Bytes	Activity %
JOIN FILTER CREATE	:BF0000	13	10	34		4	6,000						
PX RECEIVE		14	10	34		4	6,000						
PX SEND BROADCAST	:TQ10003	15	10	34		4	6,000						
HASH JOIN BUFFERED		16	10	34		4	1,500	33MB					
JOIN FILTER CREATE	:BF0001	17	1	20		4	600						
PX RECEIVE		18	1	20		4	600						
PX SEND HYBRID HASH	:TQ10001	19	1	20		4	600						
STATISTICS COLLECTOR		20				4	150						
VIEW	VW_GBF_16	21	1	20		4	150						
HASH GROUP BY		22	1	20		4	150	19MB					
PX RECEIVE		23	1	20		4	150						
PX SEND HASH	:TQ10000	24	1	20		4	150						
HASH GROUP BY		25	1	20		4	150	14MB					
HASH JOIN		26	1	20		4	150	17MB					
JOIN FILTER ...	:BF0002	27	3	17		4	240						
TABLE ACCE...	LOCATIONS	28	3	17		4	240						
JOIN FILTER ...	:BF0002	29	700	3		4	150						
PX BLOCK I...		30	700	3		4	150						
TABLE AC...	STORES	31	700	3		22	150			22	176KB		
PX RECEIVE		32	50K	14		4	0						
PX SEND HYBRID HASH	:TQ10002	33	50K	14		4	0						
JOIN FILTER USE	:BF0000	37	147M	850K		4	19M						
PX BLOCK ITERATOR		38	147M	850K		4	19M						
TABLE ACCESS STORAGE FULL	SALES	39	147M	850K		91	19M	48MB		265K	86GB	2.33	
PX BLOCK ITERATOR		40	1,000K	5,820		4	1,000K						
TABLE ACCESS STORAGE FULL	SUPPLIERS	41	1,000K	5,820		54	1,000K	18MB		1,844	601MB		
INDEX UNIQUE SCAN	PRODUCTS_PK	42	1	1		19M	16M			5,246K	40GB		50
TABLE ACCESS BY INDEX ROWID	PRODUCTS	43	1	2		25M	16M			905K	7GB		37

Baseline Run

Plan Statistics | Plan | Parallel | Activity | Metrics

Plan Hash Value 1155856492 | Plan Note

Operation	Name	Line ID	Estimated Rows	Cost	Timeline(89s)	Executions	Actual Rows	Memory (Max)
JOIN FILTER CREATE	:BF0000	13	10	34		4	6,000	
PX RECEIVE		14	10	34		4	6,000	
PX SEND BROADCAST	:TQ10003	15	10	34		4	6,000	
HASH JOIN BUFFERED		16	10	34		4	1,500	33MB
JOIN FILTER CREATE	:BF0001	17	1	20		4	600	
PX RECEIVE		18	1	20		4	600	
PX SEND HYBRID HASH	:TQ10001	19	1	20		4	600	
STATISTICS COLLECTOR		20				4	150	
VIEW	VW_GBF_16	21	1	20		4	150	
HASH GROUP BY		22	1	20		4	150	19MB
PX RECEIVE		23	1	20		4	150	
PX SEND HASH	:TQ10000	24	1	20		4	150	
HASH GROUP BY		25	1	20		4	150	14MB
HASH JOIN		26	1	20		4	150	17MB
JOIN FILTER ...	:BF0002	27	3	17		4	240	
TABLE ACCE...	LOCATIONS	28	3	17		4	240	
JOIN FILTER ...	:BF0002	29	700	3		4	150	
PX BLOCK I...		30	700	3		4	150	
TABLE AC...	STORES	31	700	3		22	150	22 176KB
PX RECEIVE		32	50K	14		4	0	
PX SEND HYBRID HASH	:TQ10002	33	50K	14		4	0	
JOIN FILTER USE	:BF0000	37	147M	850K		4	19M	
PX BLOCK ITERATOR		38	147M	850K		4	19M	
TABLE ACCESS STORAGE FULL	SALES	39	147M	850K		91	19M	48MB 265K 86GB 2.33
PX BLOCK ITERATOR		40	1,000K	5,820		4	1,000K	
TABLE ACCESS STORAGE FULL	SUPPLIERS	41	1,000K	5,820		54	1,000K	18MB 1,844 601MB
INDEX UNIQUE SCAN	PRODUCTS_PK	42	1	1		19M	16M	5,246K 40GB
TABLE ACCESS BY INDEX ROWID	PRODUCTS	43	1	2		25M	16M	905K 7GB 37

The first big underestimate is coming from row source 28:

Estimated: 3

Actual: 60

Due to small table replication, actual cardinality is multiplied by the number of PX servers (4), hence 240 reported

Baseline Run

Plan Statistics								
Plan								
Parallel								
Activity								
Metrics								
Display Tabular Plan Hash Value 1155856492 Plan Note								
Operation	Object	Line ID	Predicate	Pruning	Operation Cost	Estimated Rows	Estimated Bytes	
HASH JOIN BUFFERED		16				10	390	
JOIN FILTER CREATE	:BF0001	17				1	30	
PX RECEIVE		18				1	30	
PX SEND HYBRID HASH	:TQ10001	19				1	30	
STATISTICS COLLECTOR		20						
VIEW	VW_GBF_16	21				1	30	
HASH GROUP BY		22				1	27	
PX RECEIVE		23				1	27	
PX SEND HASH	:TQ10000	24				1	27	
HASH GROUP BY		25				1	27	
HASH JOIN		26				1	27	
JOIN FILTER CREATE	:BF0002	27				3	48	
TABLE ACCESS STORAGE FULL	LOCATIONS	28			17	3	48	
JOIN FILTER USE	:BF0002	29	filter: ("L"."CITY"='CITY_1' OR "L"."CITY"='CITY_2' OR "L"."CITY"='CITY_3')			700	7,700	
PX BLOCK ITERATOR		30				700	7,700	
TABLE ACCESS STORAGE FULL	STORES	31			3	700	7,700	
PX RECEIVE		32				50K	450K	
PX SEND HYBRID HASH	:TQ10002	33				50K	450K	
JOIN FILTER USE	:BF0001	34				50K	450K	
PX BLOCK ITERATOR		35				50K	450K	
TABLE ACCESS STORAGE FULL	DEPARTMENTS	36			14	50K	450K	
JOIN FILTER USE	:BF0000	37				147M	2,786M	
PX BLOCK ITERATOR		38				147M	2,786M	
TABLE ACCESS STORAGE FULL	SALES	39			850K	147M	2,786M	
PX BLOCK ITERATOR		40				1,000K	30M	
TABLE ACCESS STORAGE FULL	SUPPLIERS	41			5,820	1,000K	30M	
INDEX UNIQUE SCAN	PRODUCTS_PK	42			1	1		
TABLE ACCESS BY INDEX ROWID	PRODUCTS	43			2	1	9	

Baseline Run

Plan Statistics									
Plan									
Parallel									
Activity									
Metrics									
Display Tabular Plan Hash Value 1155856492 Plan Note									
Operation	Object	Line ID	Plan	Source	Filter	Estimated Bytes			
HASH JOIN BUFFERED		16							
JOIN FILTER CREATE	:BF0001	17							
PX RECEIVE		18							
PX SEND HYBRID HASH	:TQ10001	19							
STATISTICS COLLECTOR		20							
VIEW	VW_GBF_16	21							
HASH GROUP BY		22							
PX RECEIVE		23							
PX SEND HASH	:TQ10000	24							
HASH GROUP BY		25							
HASH JOIN		26							
JOIN FILTER CREATE	:BF0002	27							
TABLE ACCESS STORAGE FULL	LOCATIONS	28		17					
JOIN FILTER USE	:BF0002	29							
PX BLOCK ITERATOR		30							
TABLE ACCESS STORAGE FULL	STORES	31		3					
PX RECEIVE		32							
PX SEND HYBRID HASH	:TQ10002	33							
JOIN FILTER USE	:BF0001	34							
PX BLOCK ITERATOR		35							
TABLE ACCESS STORAGE FULL	DEPARTMENTS	36		14					
JOIN FILTER USE	:BF0000	37							
PX BLOCK ITERATOR		38							
TABLE ACCESS STORAGE FULL	SALES	39		850K					
PX BLOCK ITERATOR		40							
TABLE ACCESS STORAGE FULL	SUPPLIERS	41		5,820					
INDEX UNIQUE SCAN	PRODUCTS_PK	42		1					
TABLE ACCESS BY INDEX ROWID	PRODUCTS	43		2					

At row source 28 the following filter predicate is evaluated:

CITY in ('CITY_1','CITY_2','CITY_3')

filter: ("L"."CITY"='CITY_1' OR "L"."CITY"='CITY_2' OR "L"."CITY"='CITY_3')

At row source 28 the following filter predicate is evaluated:

CITY in ('CITY_1','CITY_2','CITY_3')

filter: ("L"."CITY"='CITY_1' OR "L"."CITY"='CITY_2' OR "L"."CITY"='CITY_3')

Analysis

Analysis

```
SELECT
    CATEGORY_ID
  , COUNTRY
  , SUM(CTRL) as CTRL
  , SUM(QUANTITY) AS S_Q
FROM
  (SELECT
      P.CATEGORY_ID
    , SP.COUNTRY
    , CASE
        WHEN SP.CATEGORY='CAT_' || P.CATEGORY_ID
        THEN 1
        ELSE 0
      END as CTRL
    , QUANTITY
  FROM
    SALES SL
  INNER JOIN DEPARTMENTS D ON SL.DEPARTMENT_ID=D.DEPARTMENT_ID
  INNER JOIN STORES S ON D.STORE_ID=S.STORE_ID
  INNER JOIN LOCATIONS L ON S.LOCATION_ID=L.LOCATION_ID
  INNER JOIN PRODUCTS P ON SL.PRODUCT_ID=P.PRODUCT_ID
  INNER JOIN SUPPLIERS SP ON SL.SUPPLIER_ID=SP.SUPPLIER_ID
  WHERE (L.CITY) IN ( 'CITY_1' , 'CITY_2' , 'CITY_3' )
  AND S.STORE_TYPE=0
  )
GROUP BY CATEGORY_ID, COUNTRY;
```


Analysis

```
SELECT
    CATEGORY_ID
  , COUNTRY
  , SUM(CTRL) as CTRL
  , SUM(QUANTITY) AS S_Q
FROM
  (SELECT
      P.CATEGORY_ID
    , SP.COUNTRY
    , CASE
        WHEN SP.CATEGORY='CAT_' || P.CATEGORY_ID
        THEN 1
        ELSE 0
      END as CTRL
    , QUANTITY
  FROM
    SALES SL
    INNER JOIN DEPARTMENTS D ON SL.DEPARTMENT_ID=D.DEPARTMENT_ID
    INNER JOIN STORES S ON D.STORE_ID=S.STORE_ID
    INNER JOIN LOCATIONS L ON S.LOCATION_ID=L.LOCATION_ID
    INNER JOIN PRODUCTS P ON SL.PRODUCT_ID=P.PRODUCT_ID
    INNER JOIN SUPPLIERS SP ON SL.SUPPLIER_ID=SP.SUPPLIER_ID
  WHERE (L.CITY) IN ('CITY_1','CITY_2','CITY_3')
  AND S.STORE_TYPE=0
  )
GROUP BY CATEGORY_ID, COUNTRY;
```

Analysis

```
SELECT
  CATEGORY_ID
, COUNTRY
, SUM(CTRL) as CTRL
, SUM(QUANTITY) AS S_Q
FROM
  (SELECT
    P.CATEGORY_ID
  , SP.COUNTRY
  , CASE
      WHEN SP.CATEGORY='CAT_' || P.CATEGORY_ID
      THEN 1
      ELSE 0
    END as CTRL
  , QUANTITY
  FROM
    SALES SL
  INNER JOIN DEPARTMENTS D ON SL.DEPARTMENT_ID=D.DEPARTMENT_ID
  INNER JOIN STORES S ON D.STORE_ID=S.STORE_ID
  INNER JOIN LOCATIONS L ON S.LOCATION_ID=L.LOCATION_ID
  INNER JOIN PRODUCTS P ON SL.PRODUCT_ID=P.PRODUCT_ID
  INNER JOIN SUPPLIERS SP ON SL.SUPPLIER_ID=SP.SUPPLIER_ID
  WHERE (L.CITY) IN ('CITY_1','CITY_2','CITY_3')
  AND S.STORE_TYPE=0
  )
GROUP BY CATEGORY_ID, COUNTRY;
```

```
SELECT num_rows
FROM user_tab_statistics
WHERE table_name = 'LOCATIONS';
```

NUM_ROWS

100000

```
SELECT num_distinct, num_nulls, histogram, num_buckets
FROM user_tab_col_statistics
WHERE table_name = 'LOCATIONS'
AND column_name = 'CITY';
```

NUM_DISTINCT	NUM_NULLS	HISTOGRAM	NUM_BUCKETS
-----	-----	-----	-----
98672	0	HYBRID	254

```
SELECT COUNT(*)
FROM user_tab_histograms
WHERE column_name='CITY'
AND endpoint_actual_value in ('CITY_1','CITY_2','CITY_3');
```

COUNT(*)

0

Analysis: Histogram with 2048 buckets?

```
SELECT
  CATEGORY_ID
, COUNTRY
, SUM(CTRL) as CTRL
, SUM(QUANTITY) AS S_Q
FROM
  (SELECT
    P.CATEGORY_ID
  , SP.COUNTRY
  , CASE
      WHEN SP.CATEGORY='CAT_' || P.CATEGORY_ID
      THEN 1
      ELSE 0
    END as CTRL
  , QUANTITY
FROM
  SALES SL
  INNER JOIN DEPARTMENTS D ON SL.DEPARTMENT_ID=D.DEPARTMENT_ID
  INNER JOIN STORES S ON D.STORE_ID=S.STORE_ID
  INNER JOIN LOCATIONS L ON S.LOCATION_ID=L.LOCATION_ID
  INNER JOIN PRODUCTS P ON SL.PRODUCT_ID=P.PRODUCT_ID
  INNER JOIN SUPPLIERS SP ON SL.SUPPLIER_ID=SP.SUPPLIER_ID
WHERE (L.CITY) IN ('CITY_1','CITY_2','CITY_3')
  AND S.STORE_TYPE=0
)
GROUP BY CATEGORY_ID, COUNTRY;
```

Analysis: Histogram with 2048 buckets?

```
SELECT
  CATEGORY_ID
, COUNTRY
, SUM(CTRL) as CTRL
, SUM(QUANTITY) AS S_Q
FROM
  (SELECT
    P.CATEGORY_ID
  , SP.COUNTRY
  , CASE
      WHEN SP.CATEGORY='CAT_' || P.CATEGORY_ID
      THEN 1
      ELSE 0
    END as CTRL
  , QUANTITY
  FROM
    SALES SL
  INNER JOIN DEPARTMENTS D ON SL.DEPARTMENT_ID=D.DEPARTMENT_ID
  INNER JOIN STORES S ON D.STORE_ID=S.STORE_ID
  INNER JOIN LOCATIONS L ON S.LOCATION_ID=L.LOCATION_ID
  INNER JOIN PRODUCTS P ON SL.PRODUCT_ID=P.PRODUCT_ID
  INNER JOIN SUPPLIERS SP ON SL.SUPPLIER_ID=SP.SUPPLIER_ID
  WHERE (L.CITY) IN ('CITY_1','CITY_2','CITY_3')
  AND S.STORE_TYPE=0
  )
GROUP BY CATEGORY_ID, COUNTRY;
```

```
SQL> exec dbms_stats.gather_table_stats(user,'LOCATIONS',
METHOD_OPT=>'FOR COLUMNS CITY SIZE 2048 FOR ALL COLUMNS SIZE
AUTO');
```

PL/SQL procedure successfully completed.

```
SELECT num_distinct, num_nulls, histogram, num_buckets
FROM user_tab_col_statistics
WHERE table_name = 'LOCATIONS'
AND column_name = 'CITY';
```

NUM_DISTINCT	NUM_NULLS	HISTOGRAM	NUM_BUCKETS
98672	0	HYBRID	2048

```
SELECT COUNT(*)
FROM user_tab_histograms
WHERE column_name='CITY'
AND endpoint_actual_value in ('CITY_1','CITY_2','CITY_3');
```

COUNT(*)
0

Analysis: Histogram with 2048 buckets?

```
SELECT
  CATEGORY_ID
, COUNTRY
, SUM(CTRL) as CTRL
, SUM(QUANTITY) AS S_Q
FROM
  (SELECT
    P.CATEGORY_ID
  , SP.COUNTRY
  , CASE
      WHEN SP.CATEGORY='CAT_' || P.CATEGORY_ID
      THEN 1
      ELSE 0
    END as CTRL
  , QUANTITY
FROM
  SALES SL
  INNER JOIN DEPARTMENTS D ON SL.DEPARTMENT_ID=D.DEPARTMENT_ID
  INNER JOIN STORES S ON D.STORE_ID=S.STORE_ID
  INNER JOIN LOCATIONS L ON S.LOCATION_ID=L.LOCATION_ID
  INNER JOIN PRODUCTS P ON SL.PRODUCT_ID=P.PRODUCT_ID
  INNER JOIN SUPPLIERS SP ON SL.SUPPLIER_ID=SP.SUPPLIER_ID
WHERE (L.CITY) IN ('CITY_1','CITY_2','CITY_3')
  AND S.STORE_TYPE=0
)
GROUP BY CATEGORY_ID, COUNTRY;
```

```
SQL> exec dbms_stats.gather_table_stats(user,'LOCATIONS',
METHOD_OPT=>'FOR COLUMNS CITY SIZE 2048 FOR ALL COLUMNS SIZE
AUTO');
```

PL/SQL procedure successfully completed.

```
SELECT num_distinct, num_nulls, histogram, num_buckets
FROM user_tab_col_statistics
WHERE table_name = 'LOCATIONS'
  AND column_name = 'CITY';
```

NUM_DISTINCT	NUM_NULLS	HISTOGRAM	NUM_BUCKETS
98672	0	HYBRID	2048

```
SELECT COUNT(*)
FROM user_tab_histograms
WHERE column_name='CITY'
AND endpoint_actual_value in ('CITY_1','CITY_2','CITY_3');
```

COUNT(*)
0

CITY is a column with > 2048 popular values

Histogram on column CITY may not always be helpful

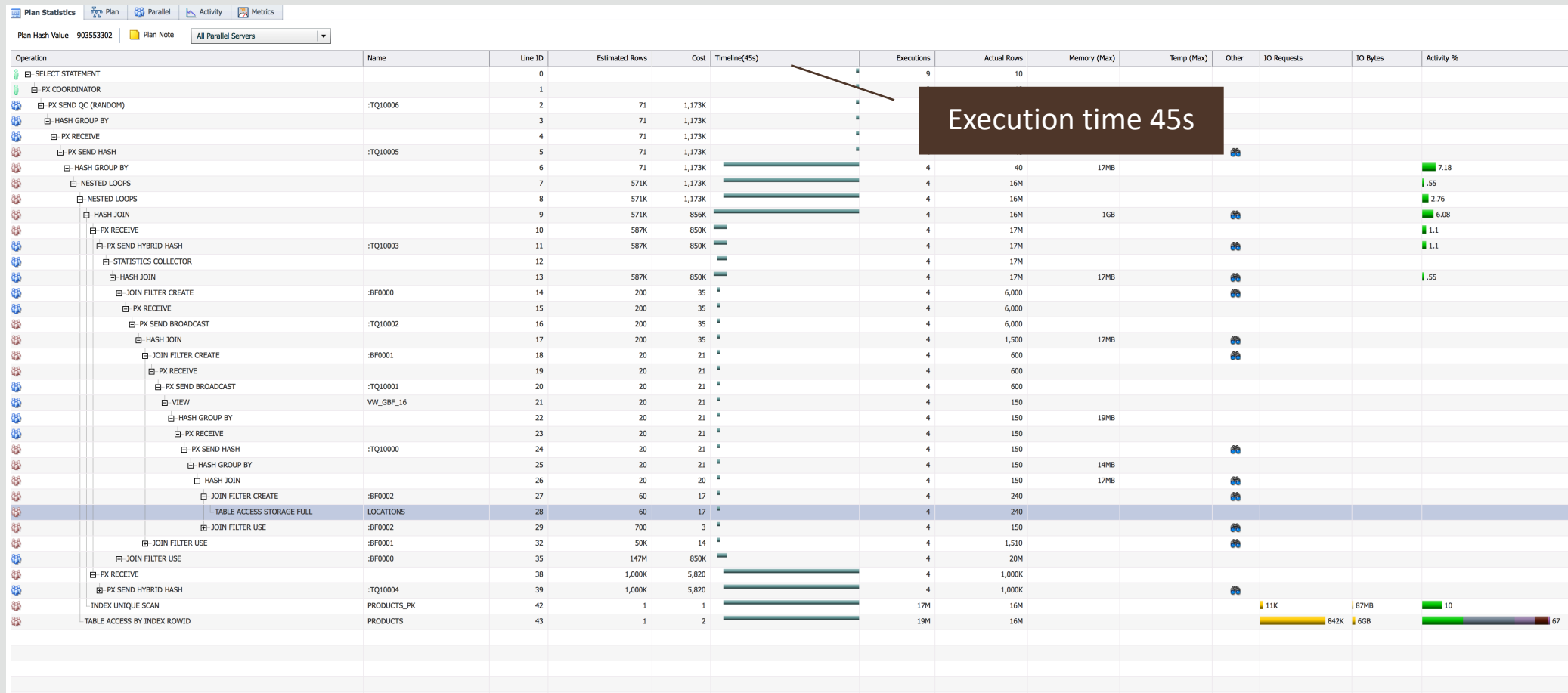
Analysis

Dynamic statistics with higher sampling level

Plan Statistics													
Plan Hash Value 903553302 Plan Note All Parallel Servers													
Operation	Name	Line ID	Estimated Rows	Cost	Timeline(45s)	Executions	Actual Rows	Memory (Max)	Temp (Max)	Other	IO Requests	IO Bytes	Activity %
SELECT STATEMENT		0				9	10						
PX COORDINATOR		1				9	10						
PX SEND QC (RANDOM)	:TQ10006	2	71	1,173K		4	10						
HASH GROUP BY		3	71	1,173K		4	10	6MB					
PX RECEIVE		4	71	1,173K		4	40						
PX SEND HASH	:TQ10005	5	71	1,173K		4	40						
HASH GROUP BY		6	71	1,173K		4	40	17MB					7.18
NESTED LOOPS		7	571K	1,173K		4	16M						.55
NESTED LOOPS		8	571K	1,173K		4	16M						2.76
HASH JOIN		9	571K	856K		4	16M	1GB					6.08
PX RECEIVE		10	587K	850K		4	17M						1.1
PX SEND HYBRID HASH	:TQ10003	11	587K	850K		4	17M						1.1
STATISTICS COLLECTOR		12				4	17M						
HASH JOIN		13	587K	850K		4	17M	17MB					.55
JOIN FILTER CREATE	:BF0000	14	200	35		4	6,000						
PX RECEIVE		15	200	35		4	6,000						
PX SEND BROADCAST	:TQ10002	16	200	35		4	6,000						
HASH JOIN		17	200	35		4	1,500	17MB					
JOIN FILTER CREATE	:BF0001	18	20	21		4	600						
PX RECEIVE		19	20	21		4	600						
PX SEND BROADCAST	:TQ10001	20	20	21		4	600						
VIEW	VW_GBF_16	21	20	21		4	150						
HASH GROUP BY		22	20	21		4	150	19MB					
PX RECEIVE		23	20	21		4	150						
PX SEND HASH	:TQ10000	24	20	21		4	150						
HASH GROUP BY		25	20	21		4	150	14MB					
HASH JOIN		26	20	20		4	150	17MB					
JOIN FILTER CREATE	:BF0002	27	60	17		4	240						
TABLE ACCESS STORAGE FULL	LOCATIONS	28	60	17		4	240						
JOIN FILTER USE	:BF0002	29	700	3		4	150						
JOIN FILTER USE	:BF0001	32	50K	14		4	1,510						
JOIN FILTER USE	:BF0000	35	147M	850K		4	20M						
PX RECEIVE		38	1,000K	5,820		4	1,000K						
PX SEND HYBRID HASH	:TQ10004	39	1,000K	5,820		4	1,000K						
INDEX UNIQUE SCAN	PRODUCTS_PK	42	1	1		17M	16M				11K	87MB	10
TABLE ACCESS BY INDEX ROWID	PRODUCTS	43	1	2		19M	16M				842K	6GB	67

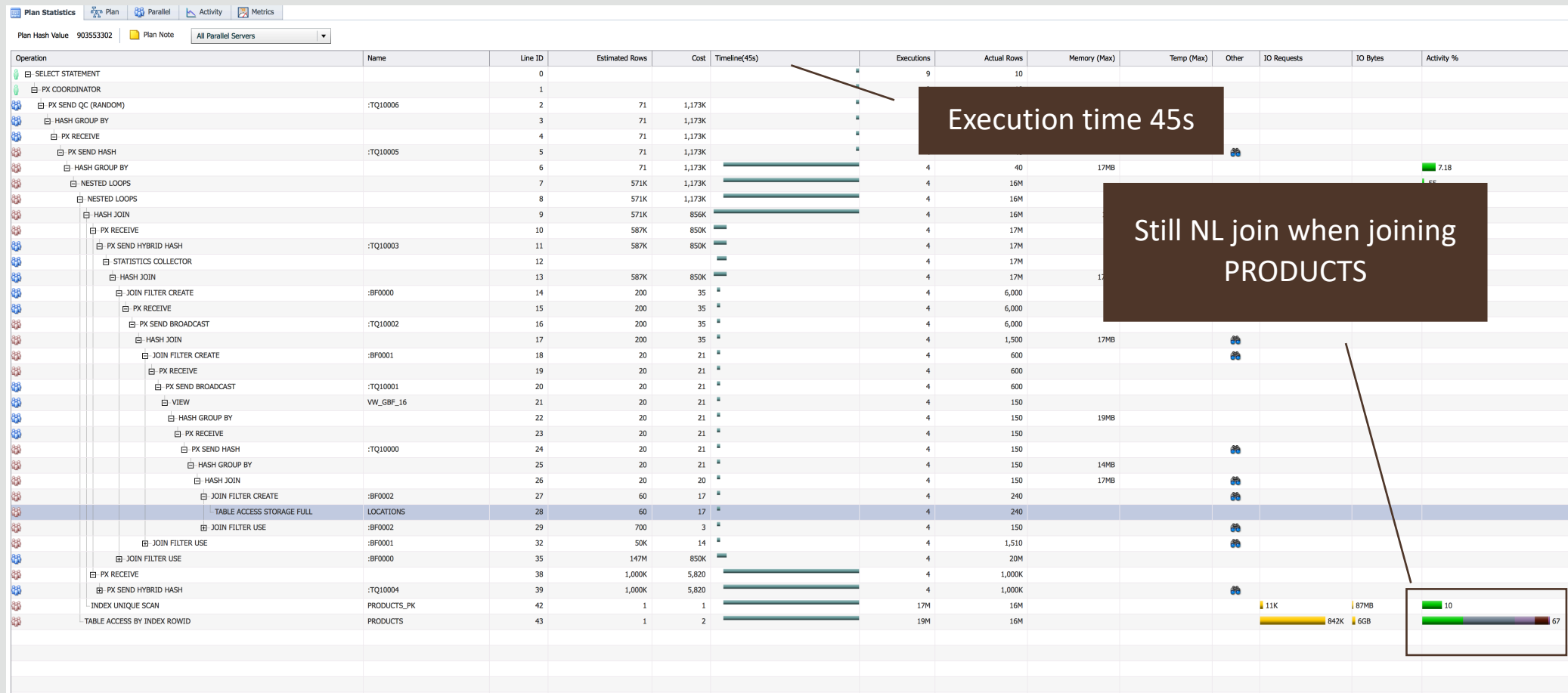
Analysis

Dynamic statistics with higher sampling level



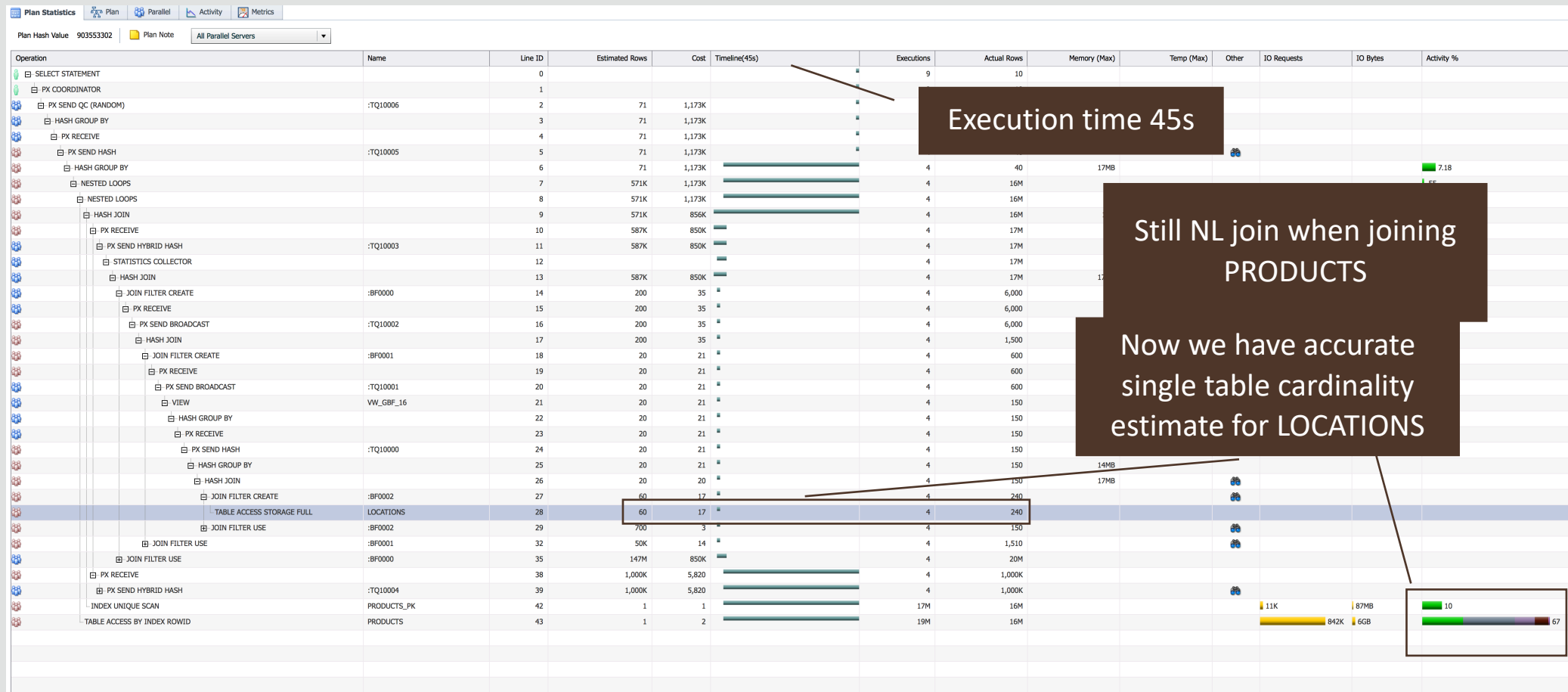
Analysis

Dynamic statistics with higher sampling level



Analysis

Dynamic statistics with higher sampling level



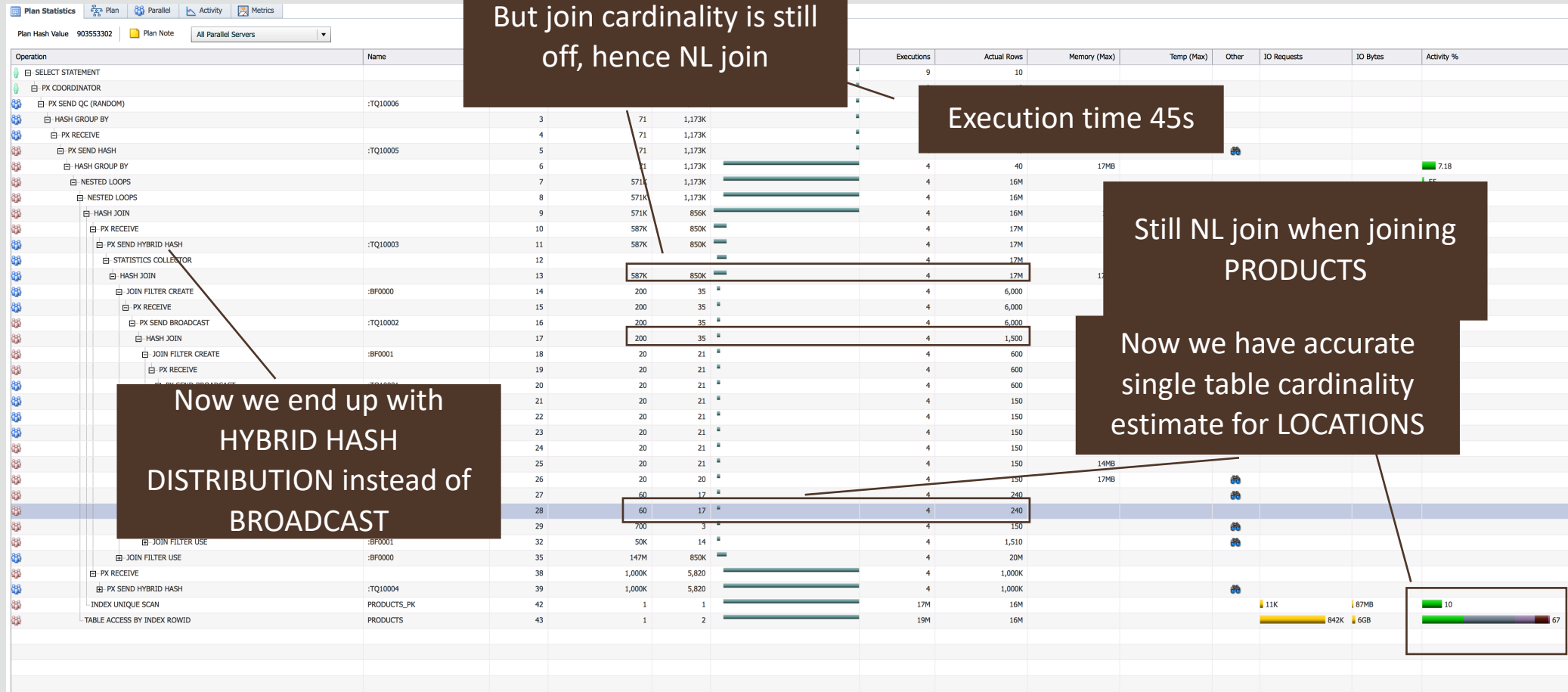
Analysis

Dynamic statistics with higher sampling level



Analysis

Dynamic statistics with higher sampling level



Analysis – Reason for join cardinality misestimate between STORES and LOCATIONS

Analysis – Reason for join cardinality misestimate between STORES and LOCATIONS

```
SELECT
  CATEGORY_ID
, COUNTRY
, SUM(CTRL) as CTRL
, SUM(QUANTITY) AS S_Q
FROM
  (SELECT
    P.CATEGORY_ID
  , SP.COUNTRY
  , CASE
      WHEN SP.CATEGORY='CAT_' || P.CATEGORY_ID
      THEN 1
      ELSE 0
    END as CTRL
  , QUANTITY
FROM
  SALES SL
  INNER JOIN DEPARTMENTS D ON SL.DEPARTMENT_ID=D.DEPARTMENT_ID
  INNER JOIN STORES S ON D.STORE_ID=S.STORE_ID
  INNER JOIN LOCATIONS L ON S.LOCATION_ID=L.LOCATION_ID
  INNER JOIN PRODUCTS P ON SL.PRODUCT_ID=P.PRODUCT_ID
  INNER JOIN SUPPLIERS SP ON SL.SUPPLIER_ID=SP.SUPPLIER_ID
  WHERE (L.CITY) IN ( 'CITY_1' , 'CITY_2' , 'CITY_3' )
  AND S.STORE_TYPE=0
  )
GROUP BY CATEGORY_ID, COUNTRY;
```

Analysis – Reason for join cardinality misestimate between STORES and LOCATIONS

```
SELECT
  CATEGORY_ID
, COUNTRY
, SUM(CTRL) as CTRL
, SUM(QUANTITY) AS S_Q
FROM
  (SELECT
    P.CATEGORY_ID
  , SP.COUNTRY
  , CASE
      WHEN SP.CATEGORY='CAT_' || P.CATEGORY_ID
      THEN 1
      ELSE 0
    END as CTRL
  , QUANTITY
FROM
  SALES SL
  INNER JOIN DEPARTMENTS D ON SL.DEPARTMENT_ID=D.DEPARTMENT_ID
  INNER JOIN STORES S ON D.STORE_ID=S.STORE_ID
  INNER JOIN LOCATIONS L ON S.LOCATION_ID=L.LOCATION_ID
  INNER JOIN PRODUCTS P ON SL.PRODUCT_ID=P.PRODUCT_ID
  INNER JOIN SUPPLIERS SP ON SL.SUPPLIER_ID=SP.SUPPLIER_ID
WHERE (L.CITY) IN ('CITY_1','CITY_2','CITY_3')
AND S.STORE_TYPE=0
  )
GROUP BY CATEGORY_ID, COUNTRY;
```

Analysis – Reason for join cardinality misestimate between STORES and LOCATIONS

```
SELECT
  CATEGORY_ID
, COUNTRY
, SUM(CTRL) as CTRL
, SUM(QUANTITY) AS S_Q
FROM
  (SELECT
    P.CATEGORY_ID
  , SP.COUNTRY
  , CASE
      WHEN SP.CATEGORY='CAT_' || P.CATEGORY_ID
      THEN 1
      ELSE 0
    END as CTRL
  , QUANTITY
FROM
  SALES SL
  INNER JOIN DEPARTMENTS D ON SL.DEPARTMENT_ID=D.DEPARTMENT_ID
  INNER JOIN STORES S ON D.STORE_ID=S.STORE_ID
  INNER JOIN LOCATIONS L ON S.LOCATION_ID=L.LOCATION_ID
  INNER JOIN PRODUCTS P ON SL.PRODUCT_ID=P.PRODUCT_ID
  INNER JOIN SUPPLIERS SP ON SL.SUPPLIER_ID=SP.SUPPLIER_ID
  WHERE (L.CITY) IN ('CITY_1','CITY_2','CITY_3')
  AND S.STORE_TYPE=0
)
GROUP BY CATEGORY_ID, COUNTRY;
```

```
SELECT num_rows
FROM user_tab_statistics
WHERE table_name = 'STORES';
```

NUM_ROWS
5000

```
SELECT num_distinct, num_nulls, histogram, num_buckets
FROM user_tab_col_statistics
WHERE table_name = 'STORES'
  AND column_name = 'LOCATION_ID';
```

NUM_DISTINCT	NUM_NULLS	HISTOGRAM	NUM_BUCKETS
2080	0	HYBRID	254

Analysis – Reason for join cardinality misestimate between STORES and LOCATIONS

```
SELECT
  CATEGORY_ID
, COUNTRY
, SUM(CTRL) as CTRL
, SUM(QUANTITY) AS S_Q
FROM
  (SELECT
    P.CATEGORY_ID
  , SP.COUNTRY
  , CASE
      WHEN SP.CATEGORY='CAT_' || P.CATEGORY_ID
      THEN 1
      ELSE 0
    END as CTRL
  , QUANTITY
FROM
  SALES SL
  INNER JOIN DEPARTMENTS D ON SL.DEPARTMENT_ID=D.DEPARTMENT_ID
  INNER JOIN STORES S ON D.STORE_ID=S.STORE_ID
  INNER JOIN LOCATIONS L ON S.LOCATION_ID=L.LOCATION_ID
  INNER JOIN PRODUCTS P ON SL.PRODUCT_ID=P.PRODUCT_ID
  INNER JOIN SUPPLIERS SP ON SL.SUPPLIER_ID=SP.SUPPLIER_ID
  WHERE (L.CITY) IN ('CITY_1','CITY_2','CITY_3')
  AND S.STORE_TYPE=0
)
GROUP BY CATEGORY_ID, COUNTRY;
```

```
SELECT num_rows
FROM user_tab_statistics
WHERE table_name = 'STORES';
```

NUM_ROWS
5000

```
SELECT num_distinct, num_nulls, histogram, num_buckets
FROM user_tab_col_statistics
WHERE table_name = 'STORES'
  AND column_name = 'LOCATION_ID';
```

NUM_DISTINCT	NUM_NULLS	HISTOGRAM	NUM_BUCKETS
2080	0	HYBRID	254

About than 2 % of the LOCATIONS
used in STORES

Analysis – Reason for join cardinality misestimate between STORES and LOCATIONS

```
SELECT
  CATEGORY_ID
, COUNTRY
, SUM(CTRL) as CTRL
, SUM(QUANTITY) AS S_Q
FROM
  (SELECT
    P.CATEGORY_ID
  , SP.COUNTRY
  , CASE
      WHEN SP.CATEGORY='CAT_' || P.CATEGORY_ID
      THEN 1
      ELSE 0
    END as CTRL
  , QUANTITY
FROM
  SALES SL
  INNER JOIN DEPARTMENTS D ON SL.DEPARTMENT_ID=D.DEPARTMENT_ID
  INNER JOIN STORES S ON D.STORE_ID=S.STORE_ID
  INNER JOIN LOCATIONS L ON S.LOCATION_ID=L.LOCATION_ID
  INNER JOIN PRODUCTS P ON SL.PRODUCT_ID=P.PRODUCT_ID
  INNER JOIN SUPPLIERS SP ON SL.SUPPLIER_ID=SP.SUPPLIER_ID
  WHERE (L.CITY) IN ('CITY_1','CITY_2','CITY_3')
  AND S.STORE_TYPE=0
  )
GROUP BY CATEGORY_ID, COUNTRY;
```

```
SELECT num_rows
FROM user_tab_statistics
WHERE table_name = 'STORES';
```

NUM_ROWS
5000

```
SELECT num_distinct, num_nulls, histogram, num_buckets
FROM user_tab_col_statistics
WHERE table_name = 'STORES'
  AND column_name = 'LOCATION_ID';
```

NUM_DISTINCT	NUM_NULLS	HISTOGRAM	NUM_BUCKETS
2080	0	HYBRID	254

Row filtering on STORES
and LOCATIONS

About than 2 % of the LOCATIONS
used in STORES

Analysis – Reason for join cardinality misestimate between STORES and LOCATIONS

```
SELECT
  CATEGORY_ID
, COUNTRY
, SUM(CTRL) as CTRL
, SUM(QUANTITY) AS S_Q
FROM
  (SELECT
    P.CATEGORY_ID
  , SP.COUNTRY
  , CASE
      WHEN SP.CATEGORY='CAT_' || P.CATEGORY_ID
      THEN 1
      ELSE 0
    END as CTRL
  , QUANTITY
FROM
  SALES SL
  INNER JOIN DEPARTMENTS D ON SL.DEPARTMENT_ID=D.DEPARTMENT_ID
  INNER JOIN STORES S ON D.STORE_ID=S.STORE_ID
  INNER JOIN LOCATIONS L ON S.LOCATION_ID=L.LOCATION_ID
  INNER JOIN PRODUCTS P ON SL.PRODUCT_ID=P.PRODUCT_ID
  INNER JOIN SUPPLIERS SP ON SL.SUPPLIER_ID=SP.SUPPLIER_ID
  WHERE (L.CITY) IN ('CITY_1','CITY_2','CITY_3')
  AND S.STORE_TYPE=0
)
GROUP BY CATEGORY_ID, COUNTRY;
```

Possible correlation between L.CITY
AND S.STORE_TYPE

Row filtering on STORES
and LOCATIONS

```
SELECT num_rows
FROM user_tab_statistics
WHERE table_name = 'STORES';
```

NUM_ROWS
5000

```
SELECT num_distinct, num_nulls, histogram, num_buckets
FROM user_tab_col_statistics
WHERE table_name = 'STORES'
AND column_name = 'LOCATION_ID';
```

NUM_DISTINCT	NUM_NULLS	HISTOGRAM	NUM_BUCKETS
2080	0	HYBRID	254

About than 2 % of the LOCATIONS
used in STORES

Analysis – Reason for join cardinality misestimate between STORES and LOCATIONS

```
SELECT
  CATEGORY_ID
, COUNTRY
, SUM(CTRL) as CTRL
, SUM(QUANTITY) AS S_Q
FROM
  (SELECT
    P.CATEGORY_ID
  , SP.COUNTRY
  , CASE
      WHEN SP.CATEGORY='CAT_' || P.CATEGORY_ID
      THEN 1
      ELSE 0
    END as CTRL
  , QUANTITY
FROM
  SALES SL
  INNER JOIN DEPARTMENTS D ON SL.DEPARTMENT_ID=D.DEPARTMENT_ID
  INNER JOIN STORES S ON D.STORE_ID=S.STORE_ID
  INNER JOIN LOCATIONS L ON S.LOCATION_ID=L.LOCATION_ID
  INNER JOIN PRODUCTS P ON SL.PRODUCT_ID=P.PRODUCT_ID
  INNER JOIN SUPPLIERS SP ON SL.SUPPLIER_ID=SP.SUPPLIER_ID
  WHERE (L.CITY) IN ('CITY_1','CITY_2','CITY_3')
  AND S.STORE_TYPE=0
)
GROUP BY CATEGORY_ID, COUNTRY;
```

Could we create a
correlated column group
between CITY and STORE?

```
SELECT num_rows
FROM user_tab_statistics
WHERE table_name = 'STORES';
```

```
NUM_ROWS
-----
5000
```

```
SELECT num_distinct, num_nulls, histogram, num_buckets
FROM user_tab_col_statistics
WHERE table_name = 'STORES'
      AND column_name = 'LOCATION_ID';
```

NUM_DISTINCT	NUM_NULLS	HISTOGRAM	NUM_BUCKETS
2080	0	HYBRID	254

About than 2 % of the LOCATIONS
used in STORES

Row filtering on STORES
and LOCATIONS

Possible correlation between L.CITY
AND S.STORE_TYPE

End Demo1



Analysis: For the query in question

- Standard statistics are not sufficient
- Due to data distribution, histograms are not effective either
- Dynamic statistics (sampling)

Could fix single table cardinality estimates

Due to data model, data characteristics and data distribution, join cardinality is still inaccurate

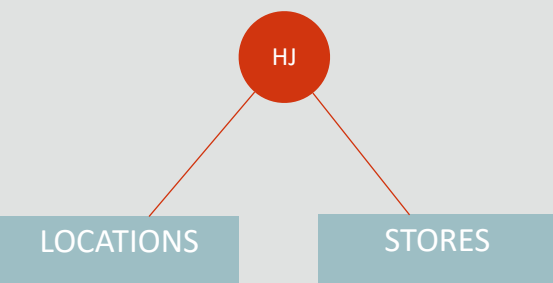
Baseline: Join order and join method



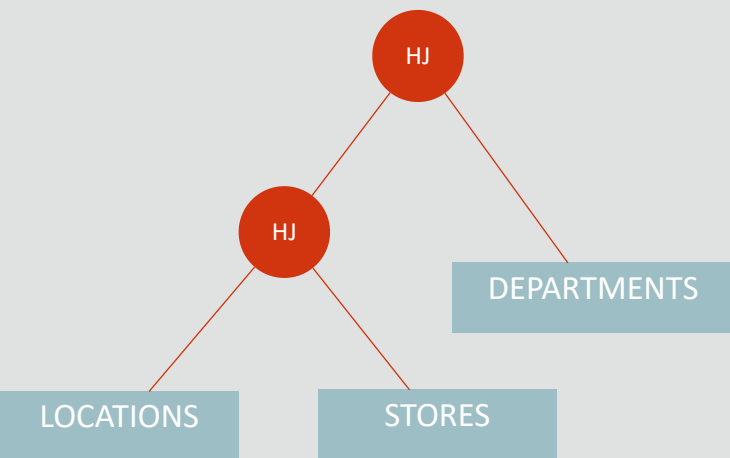
Baseline: Join order and join method



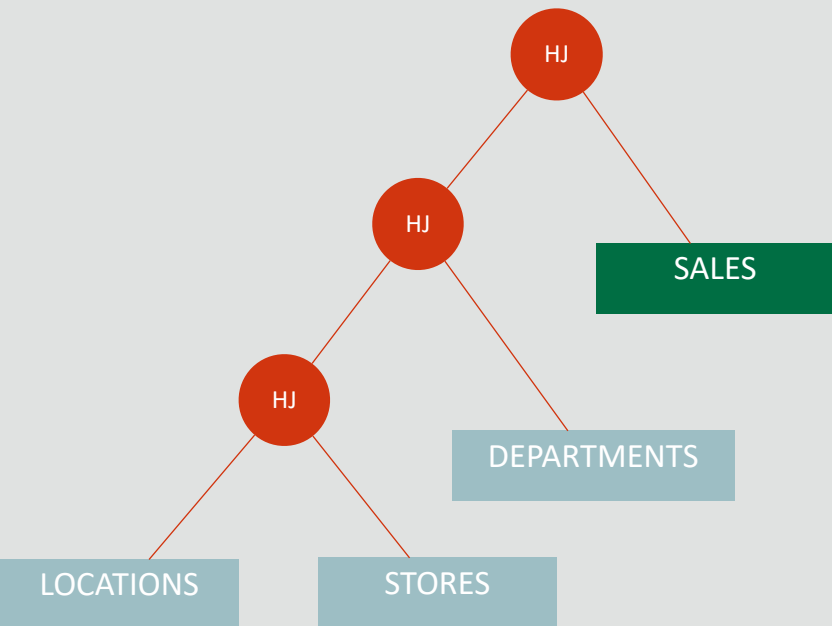
Baseline: Join order and join method



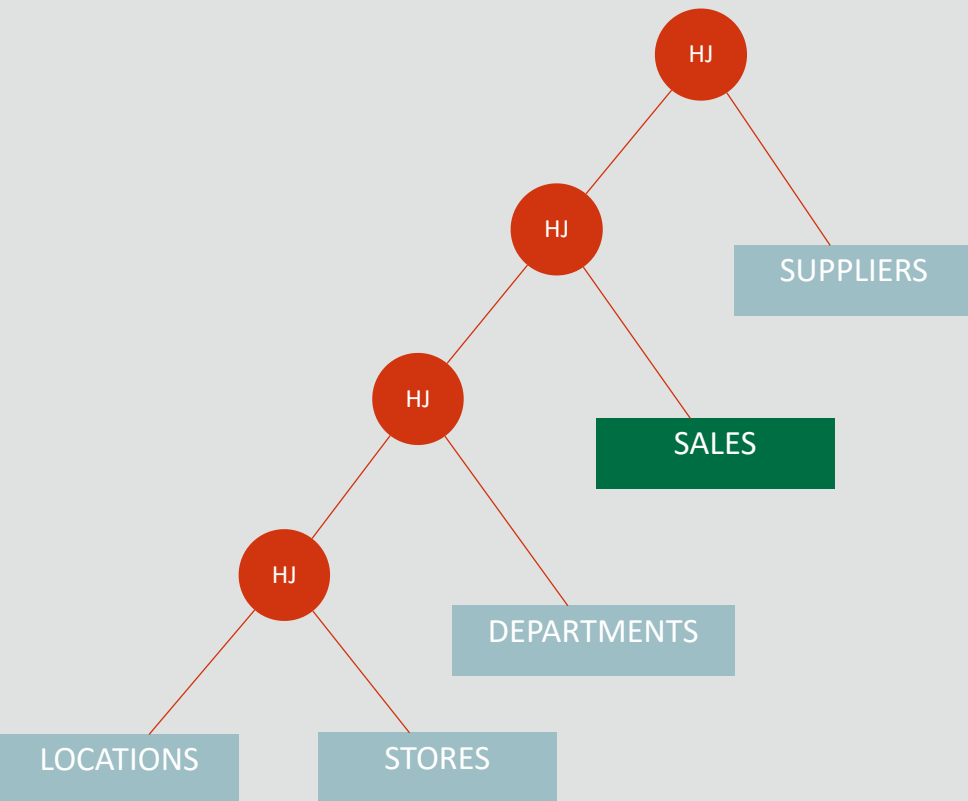
Baseline: Join order and join method



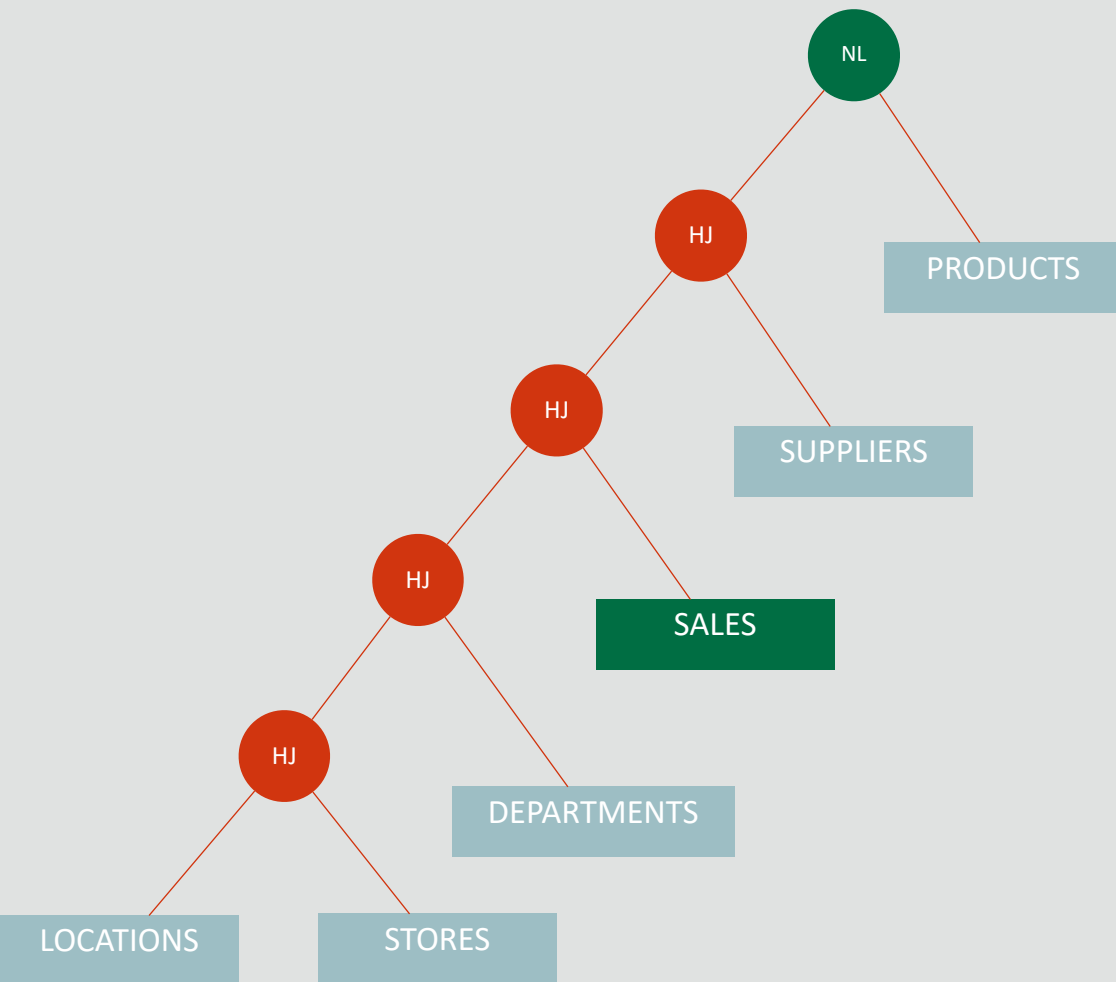
Baseline: Join order and join method



Baseline: Join order and join method

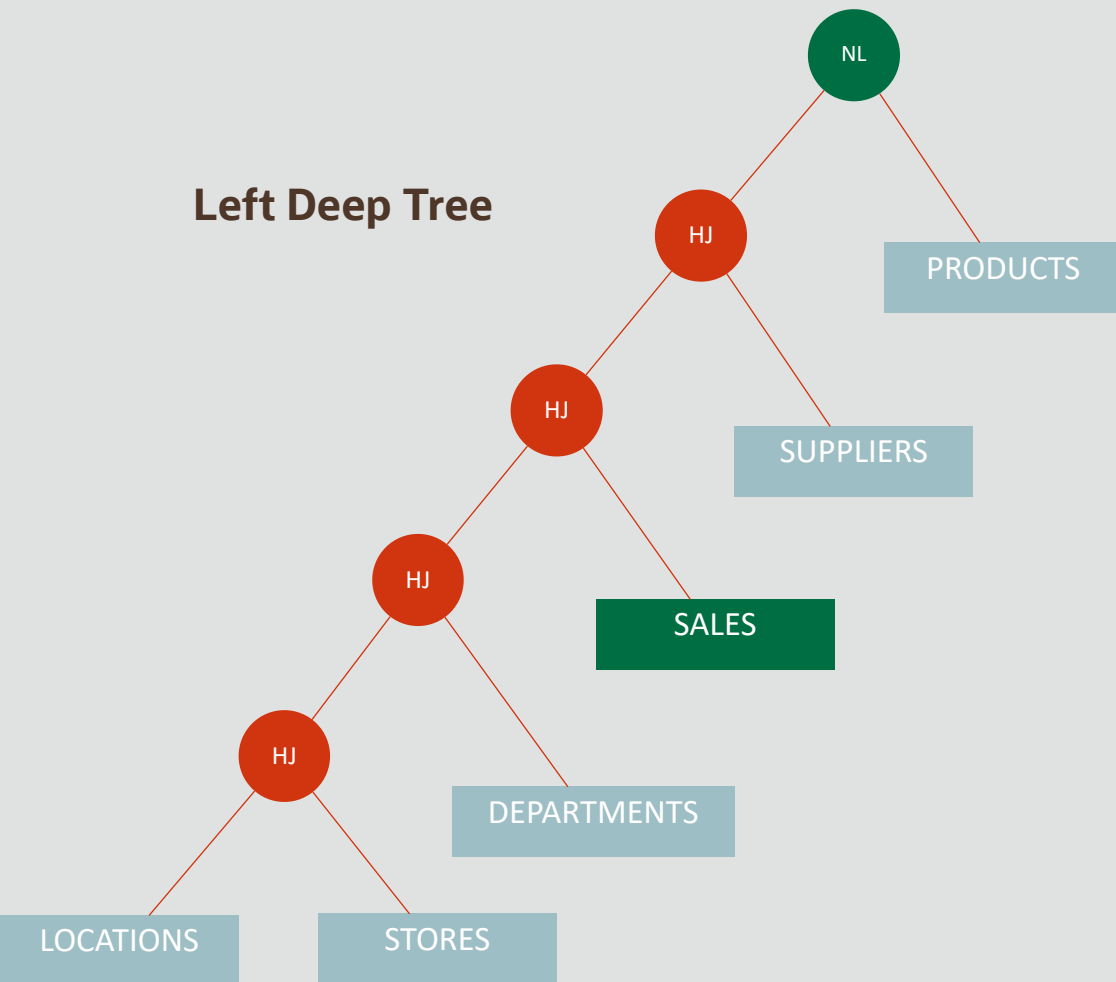


Baseline: Join order and join method



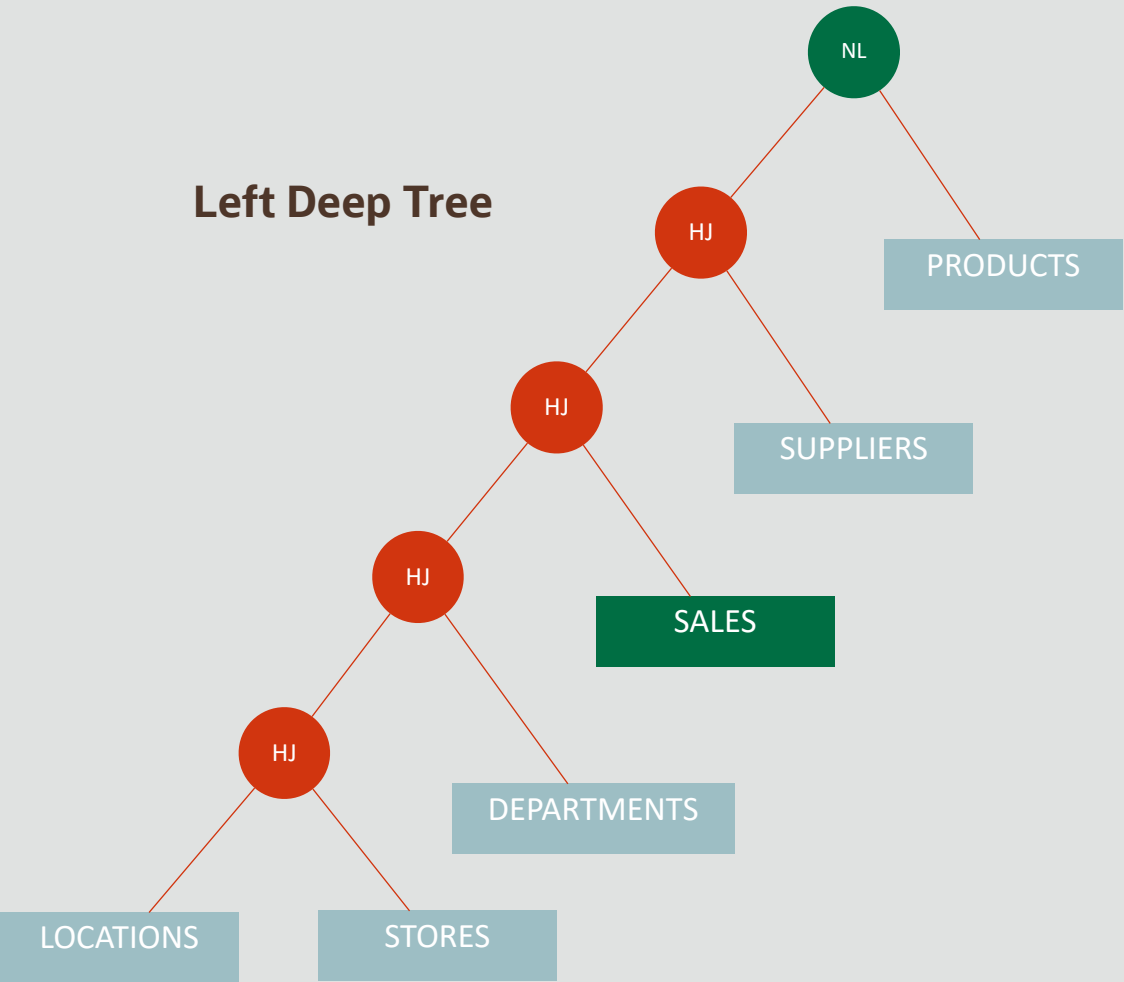
Baseline: Join order and join method

Left Deep Tree



Baseline: Join order and join method

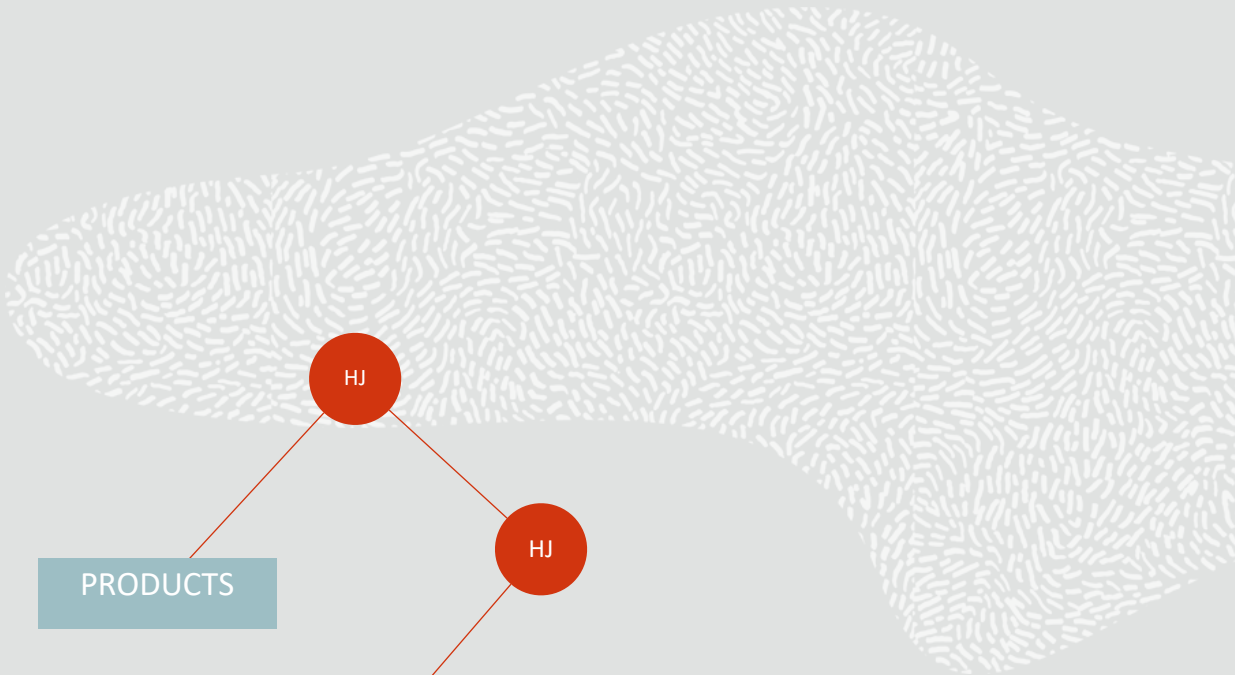
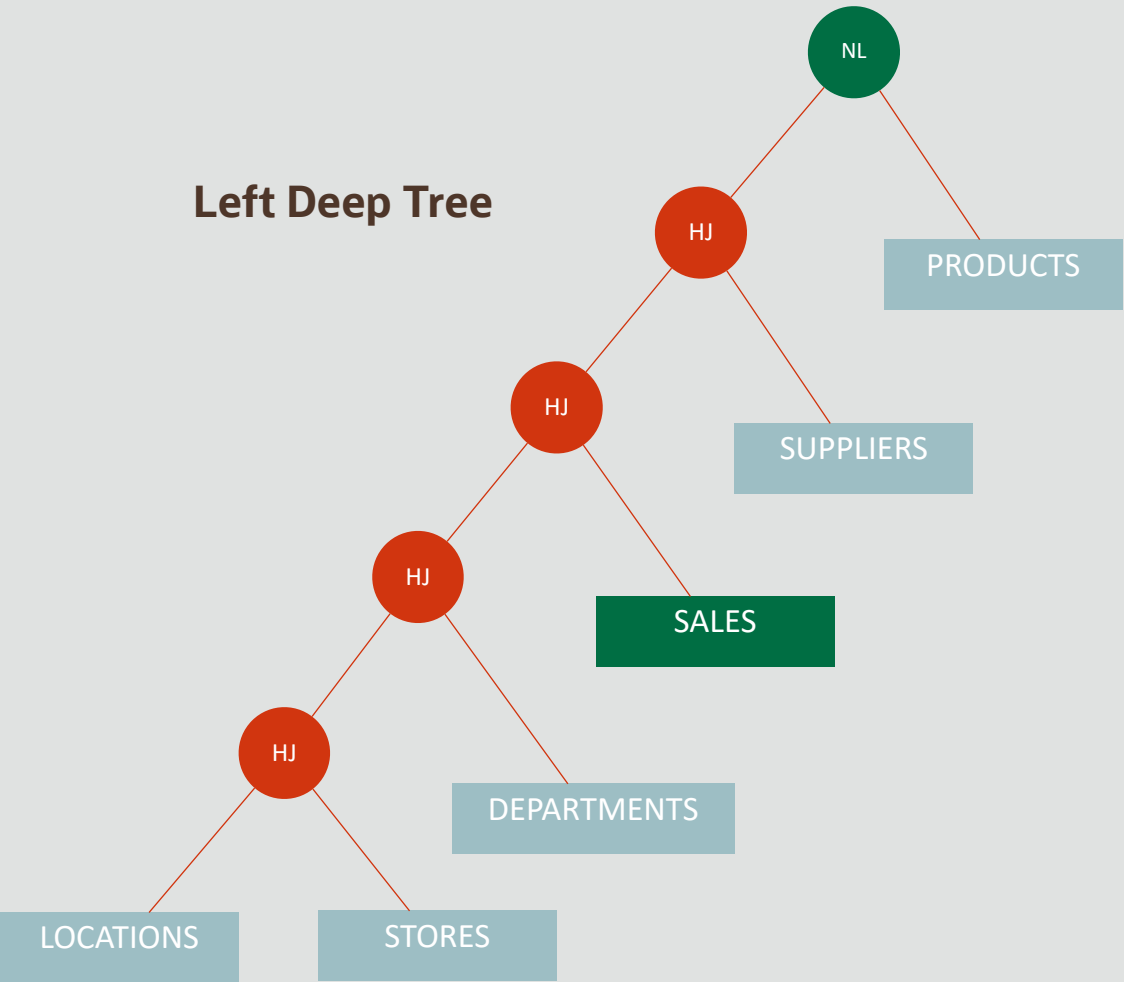
Left Deep Tree



PRODUCTS

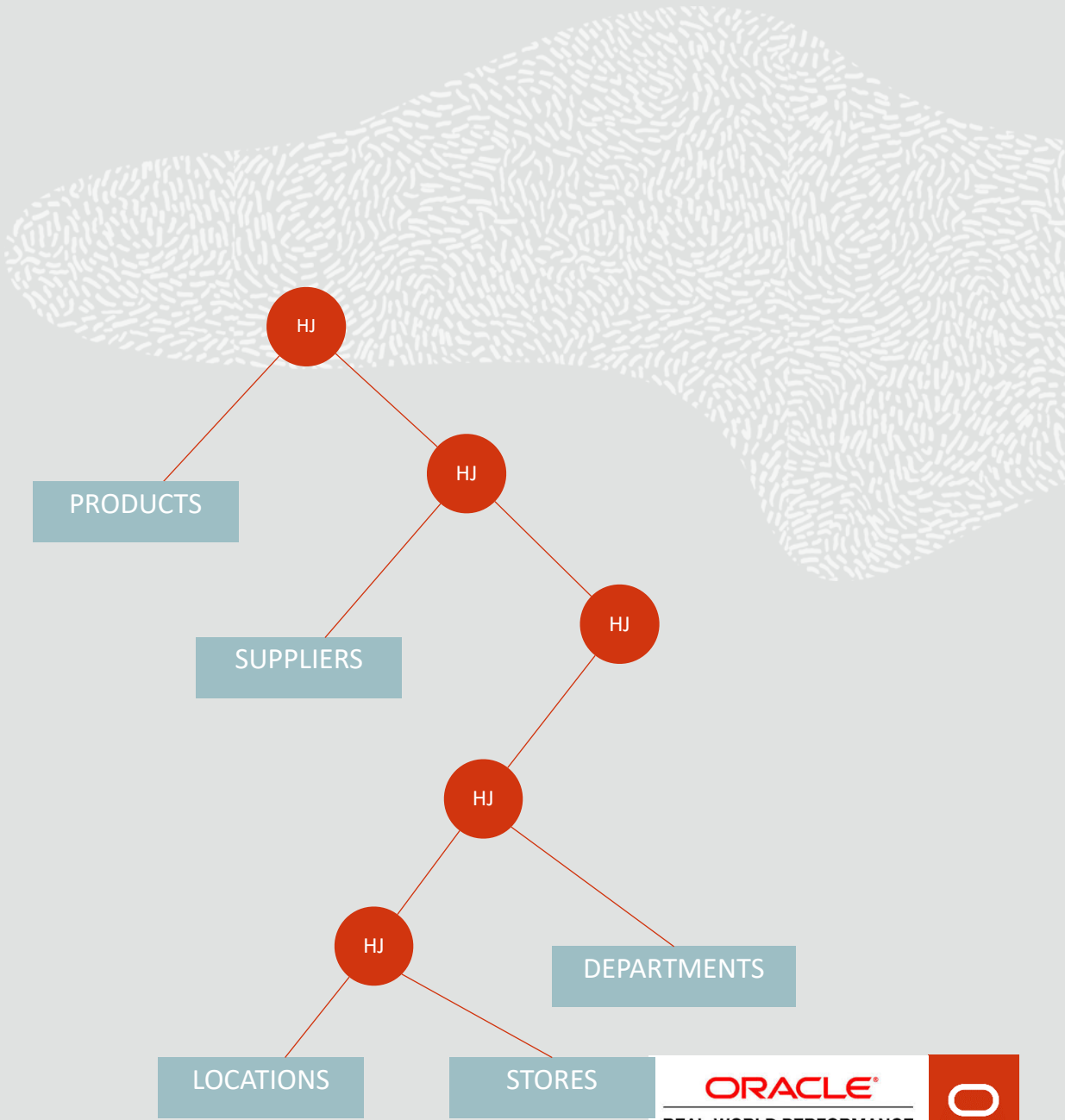
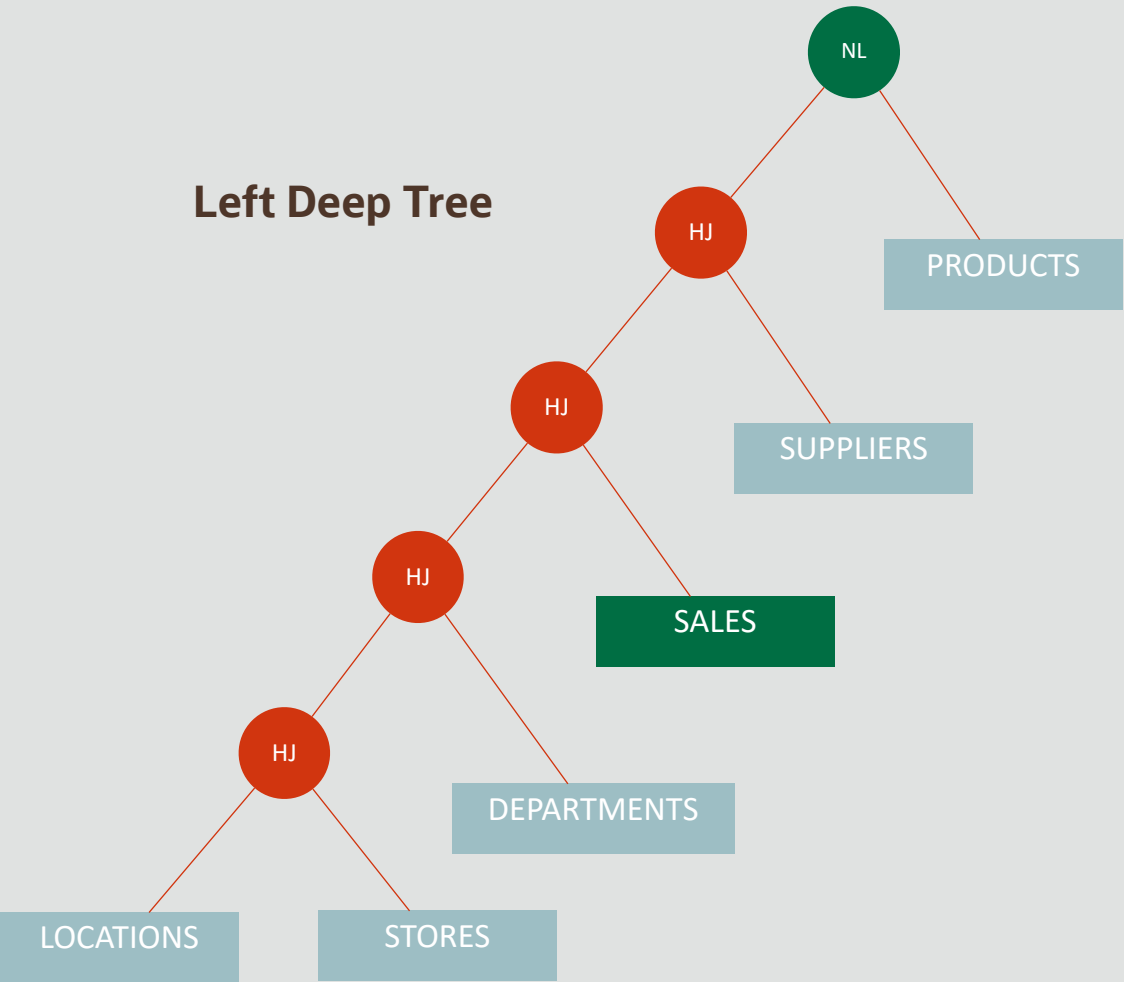
Baseline: Join order and join method

Left Deep Tree



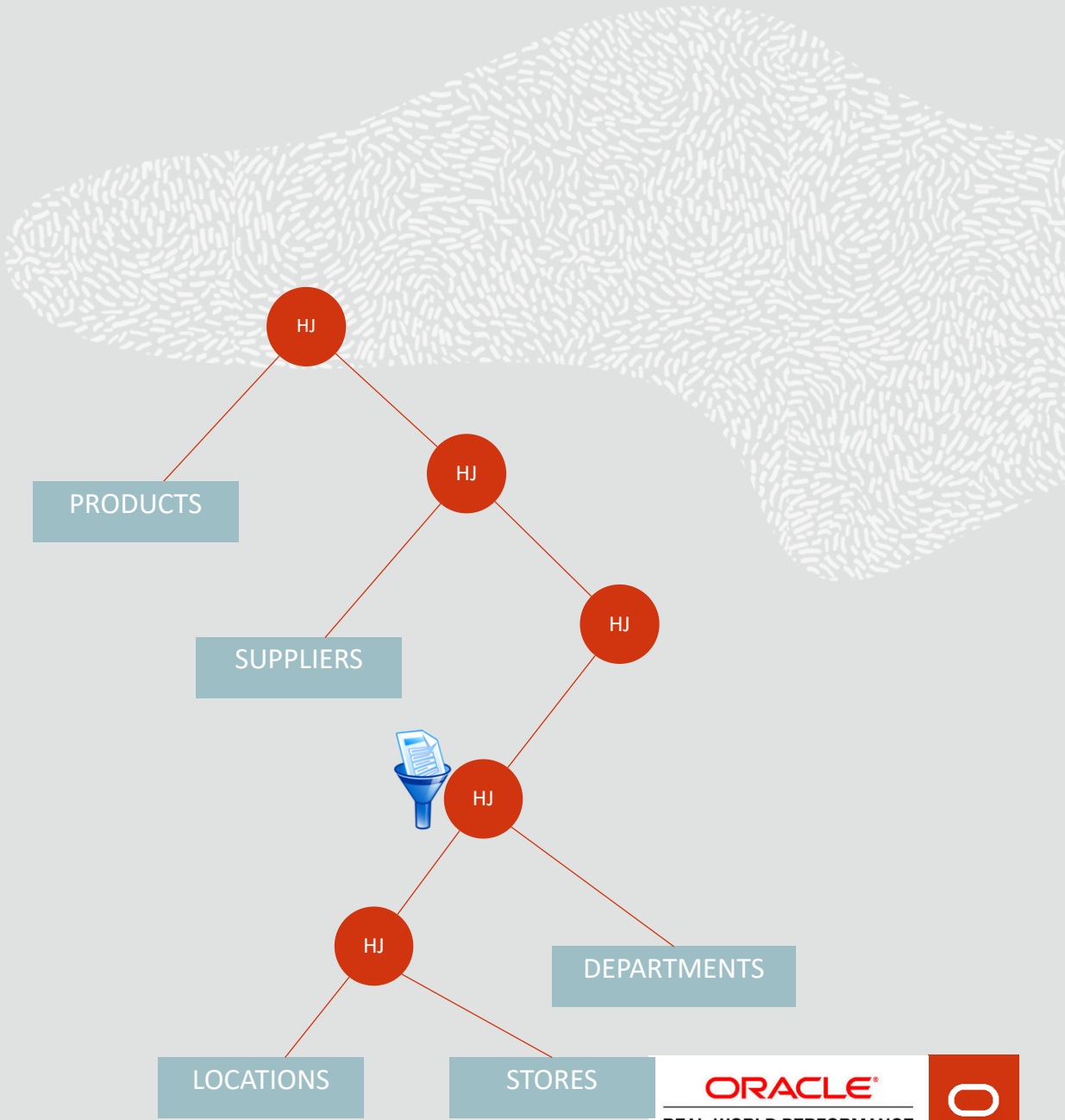
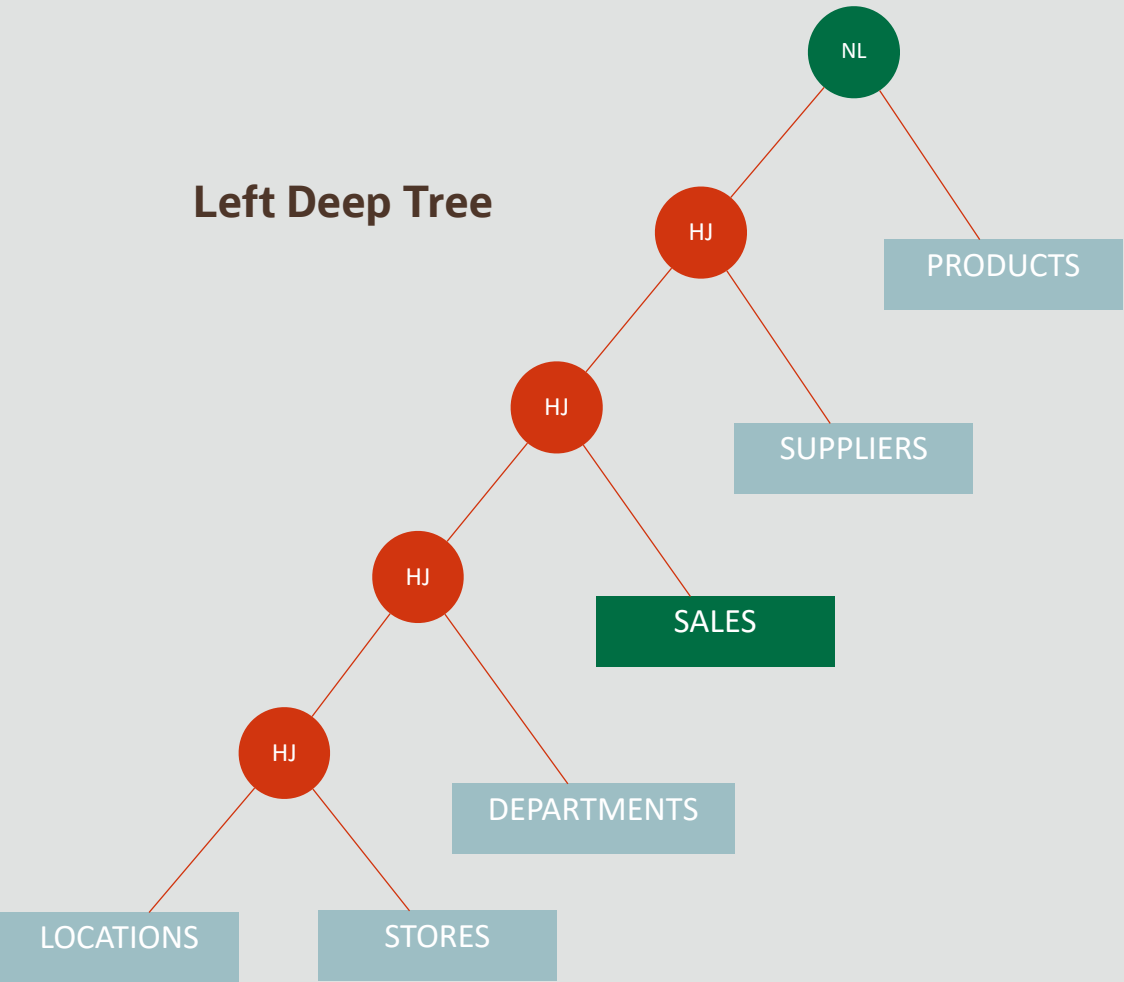
Baseline: Join order and join method

Left Deep Tree



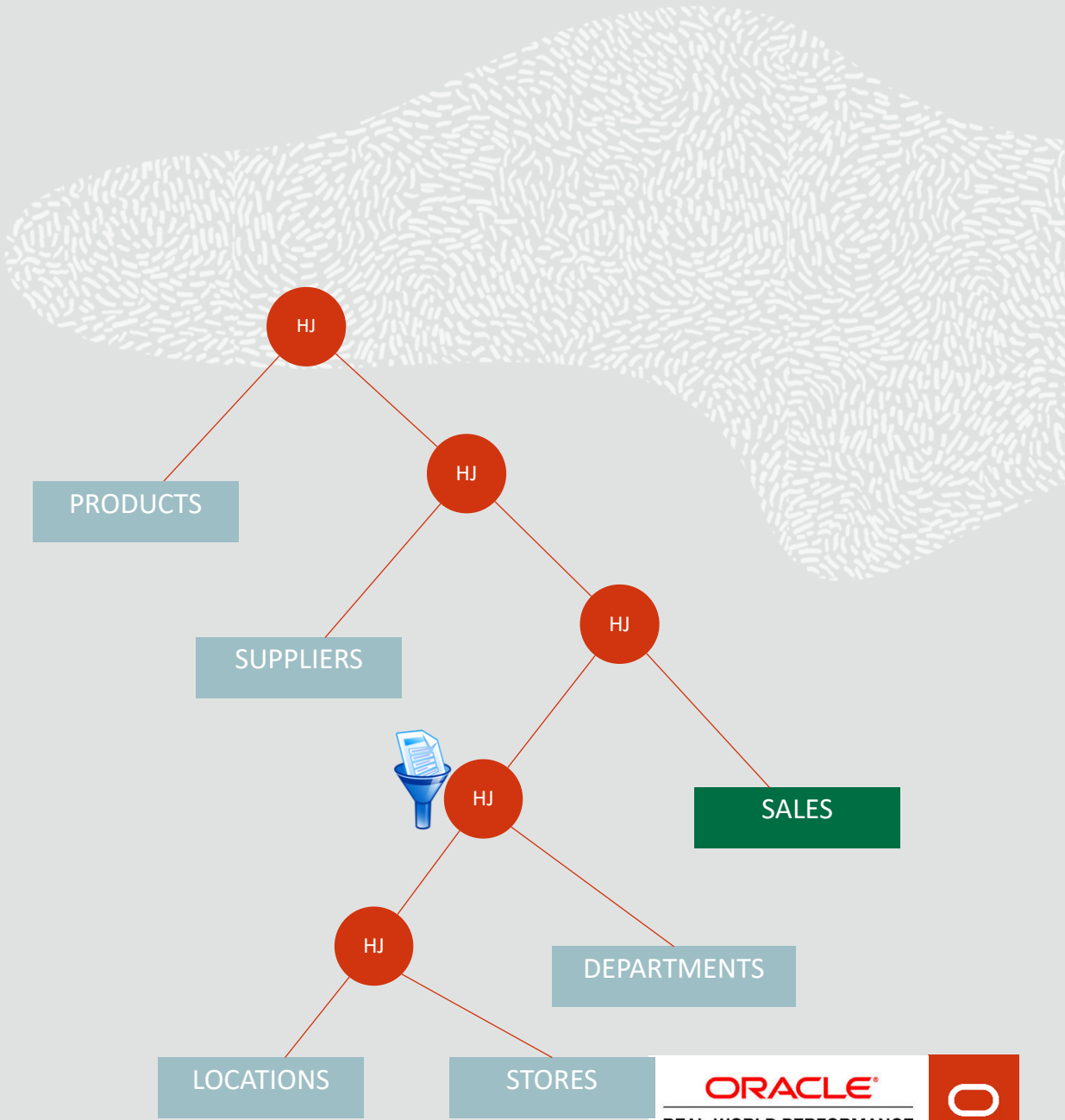
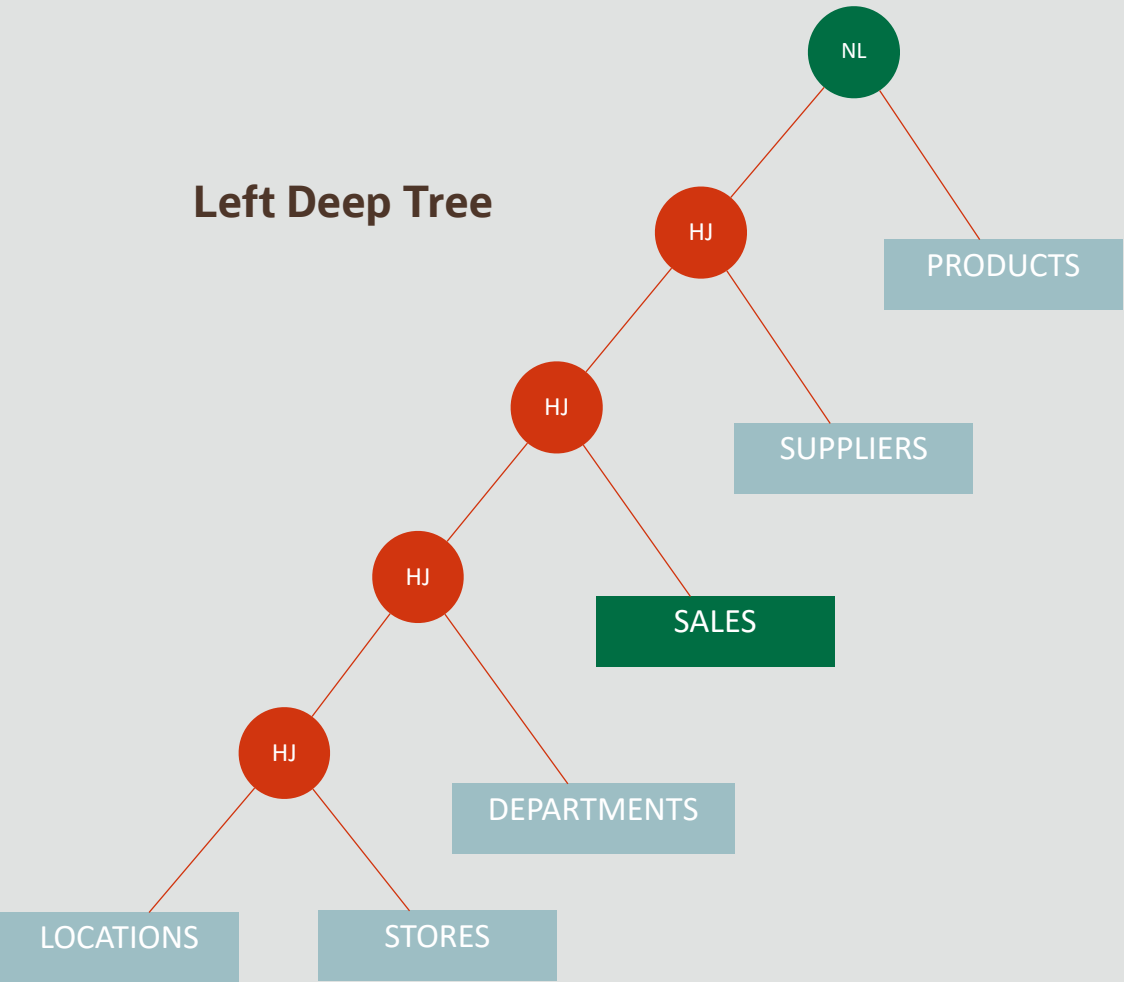
Baseline: Join order and join method

Left Deep Tree



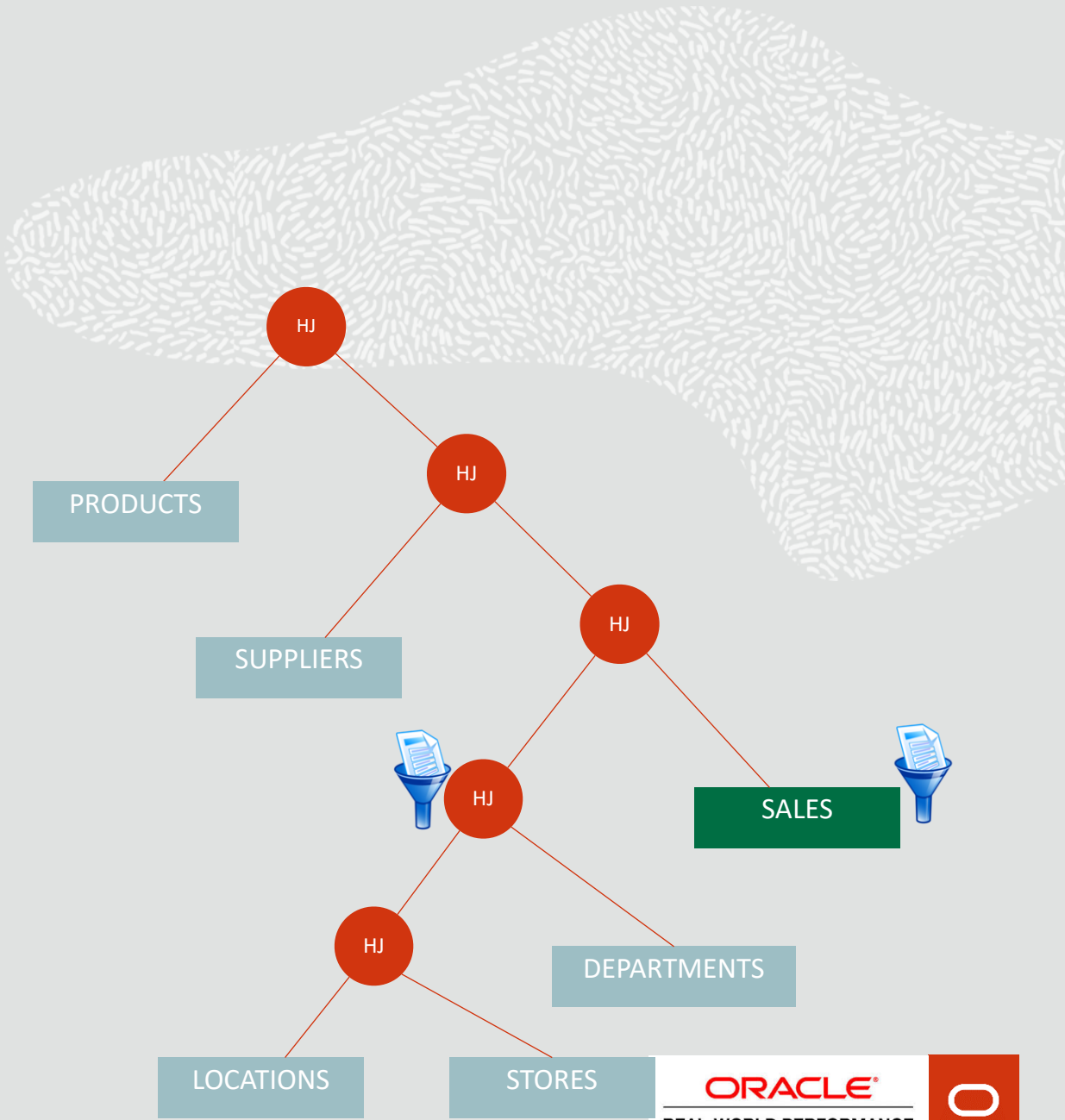
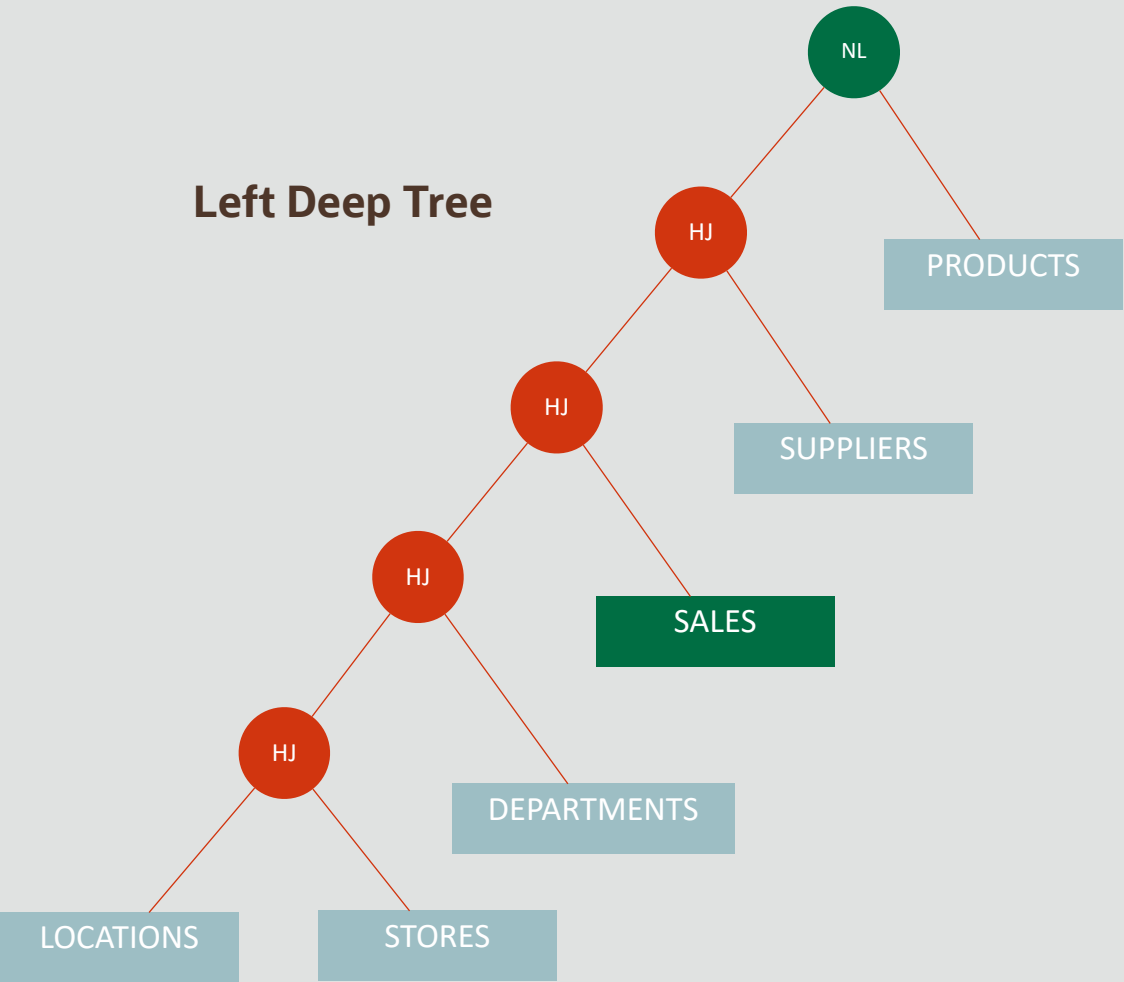
Baseline: Join order and join method

Left Deep Tree



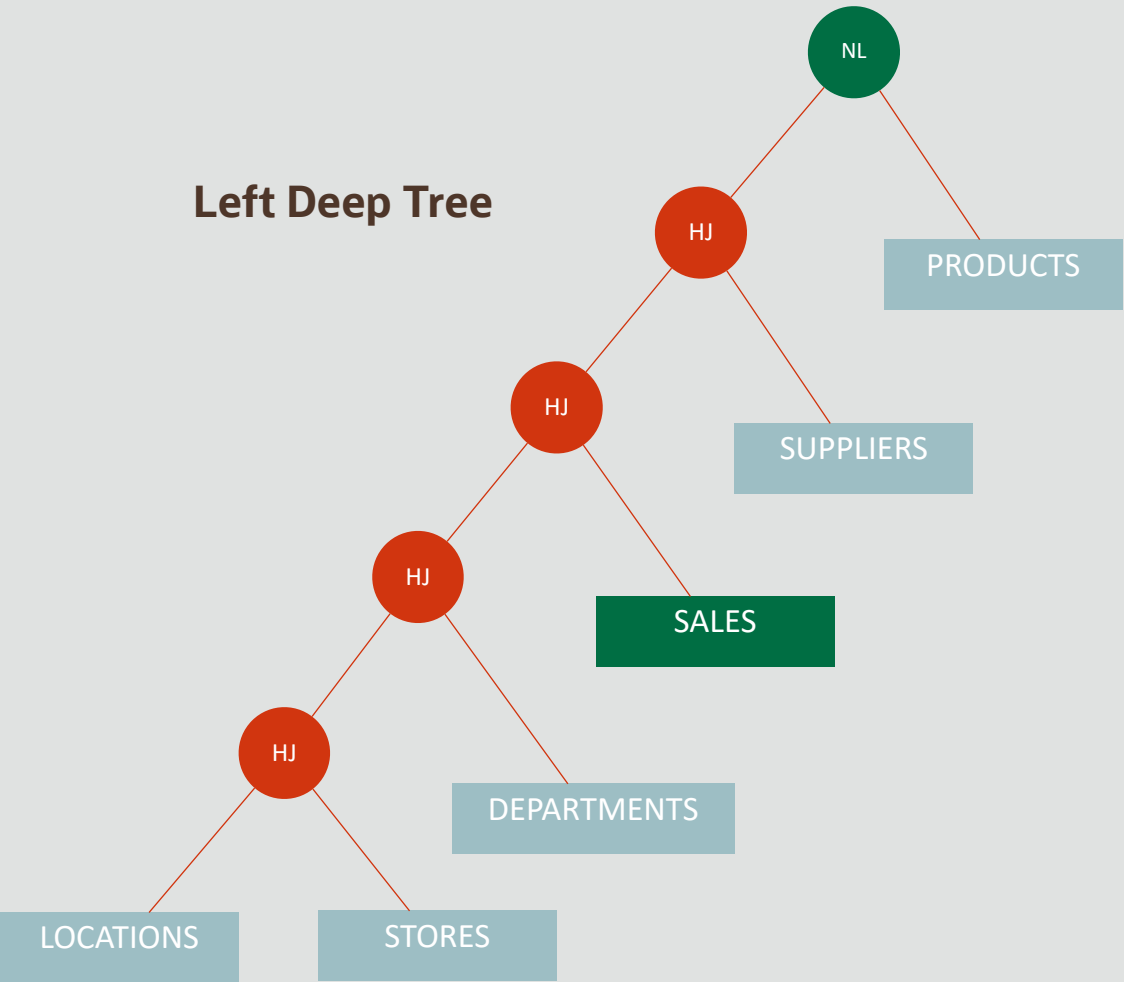
Baseline: Join order and join method

Left Deep Tree

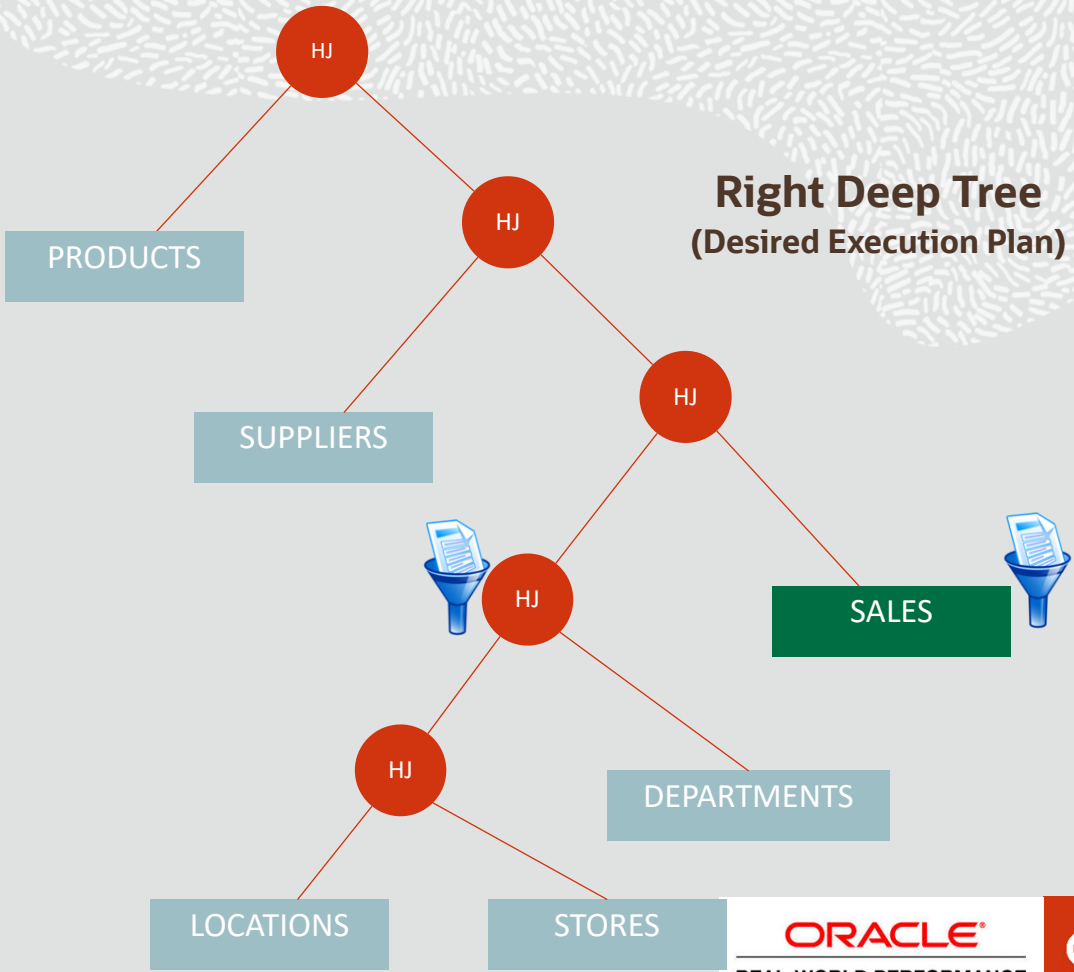


Baseline: Join order and join method

Left Deep Tree

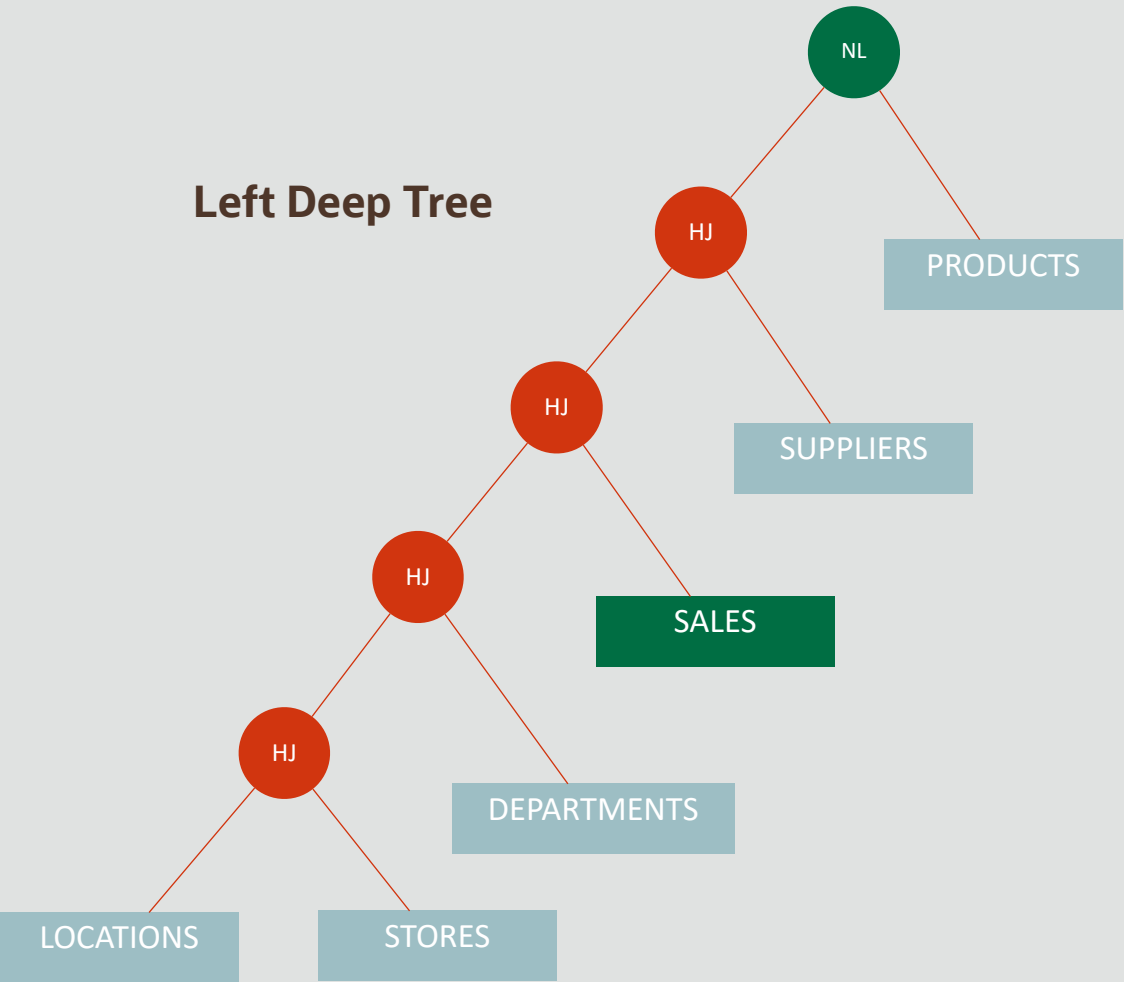


Right Deep Tree
(Desired Execution Plan)

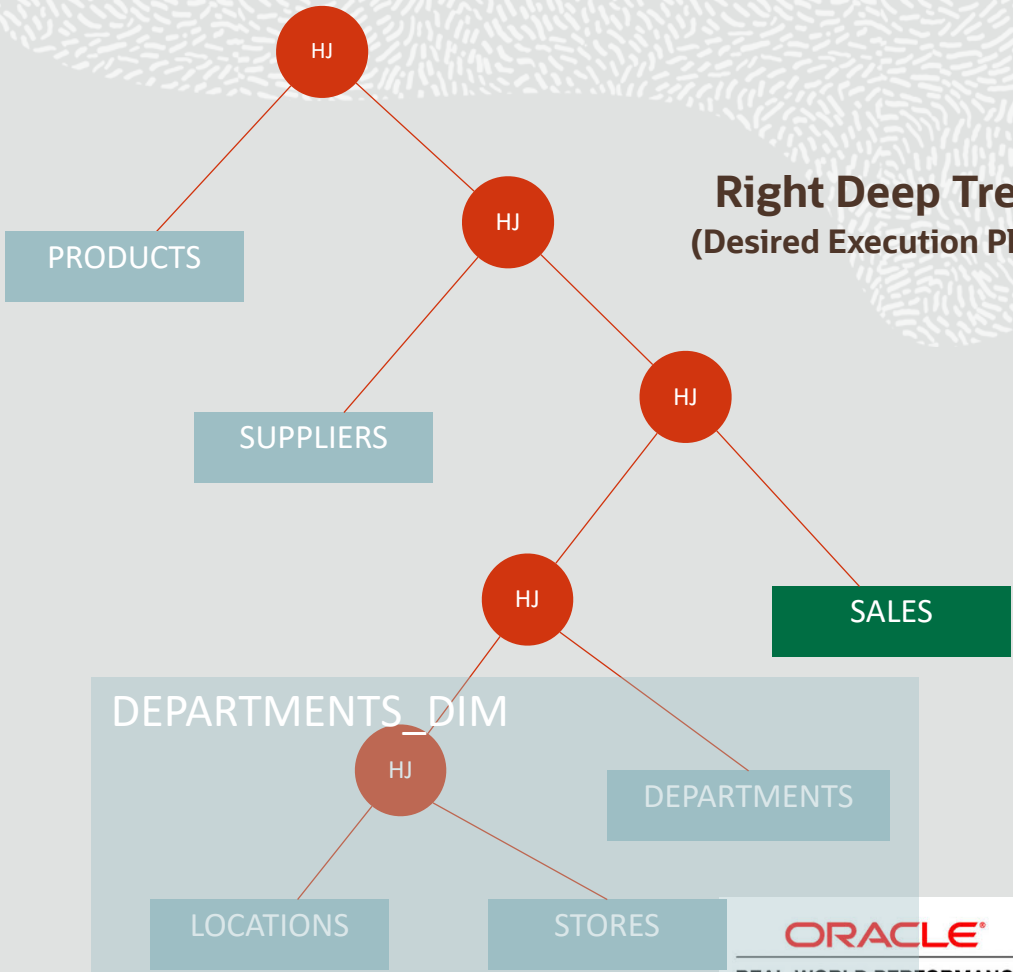


Baseline: Join order and join method

Left Deep Tree

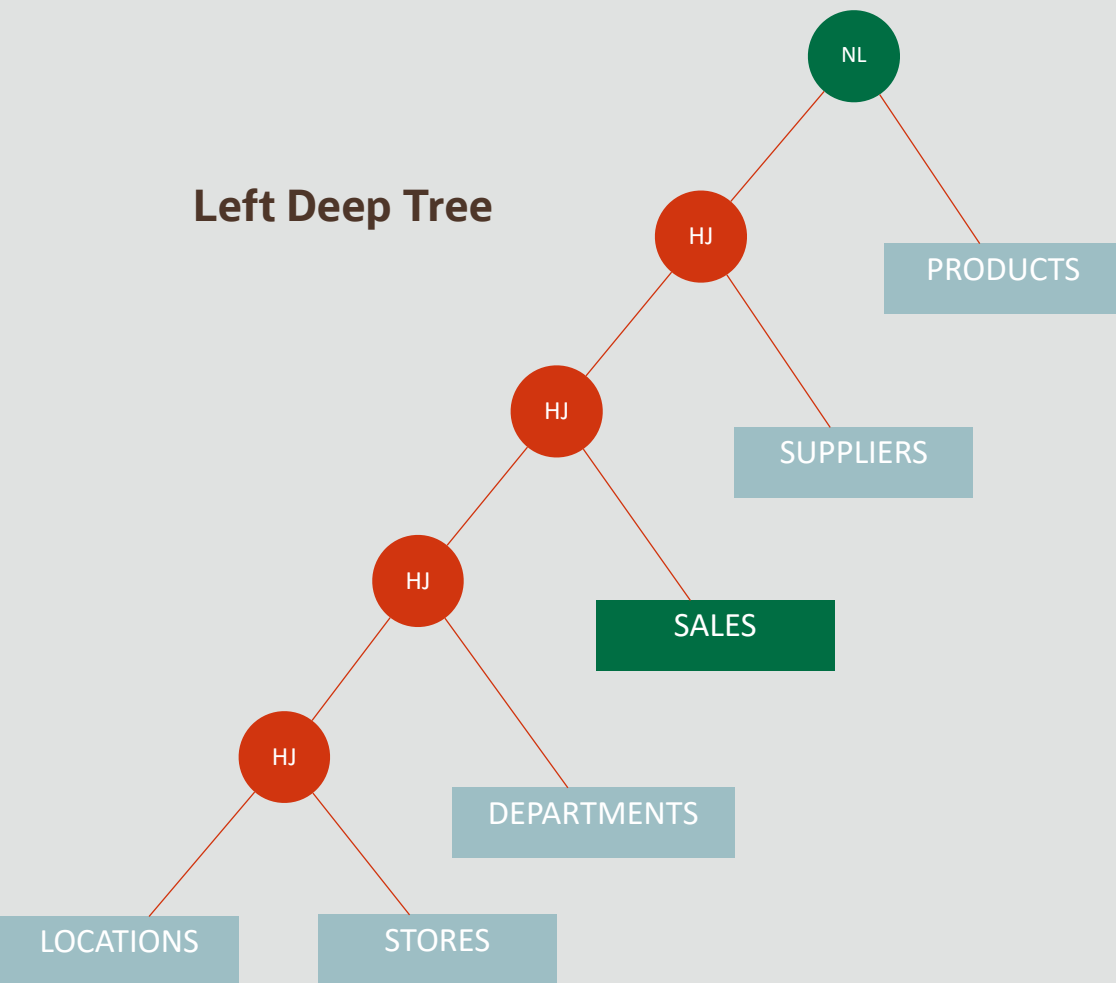


Right Deep Tree
(Desired Execution Plan)

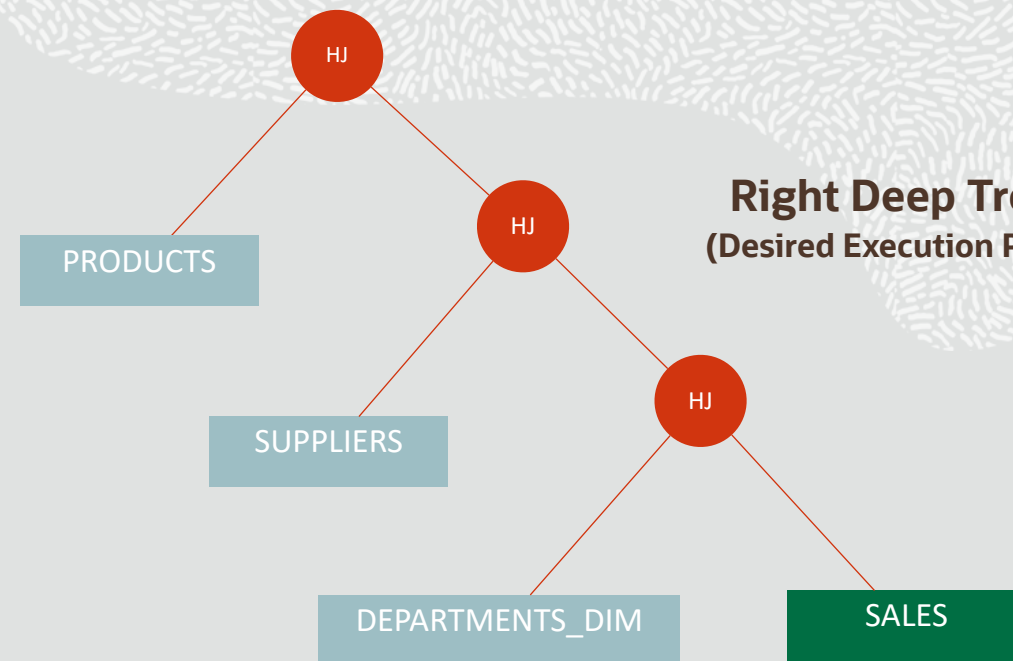


Baseline: Join order and join method

Left Deep Tree

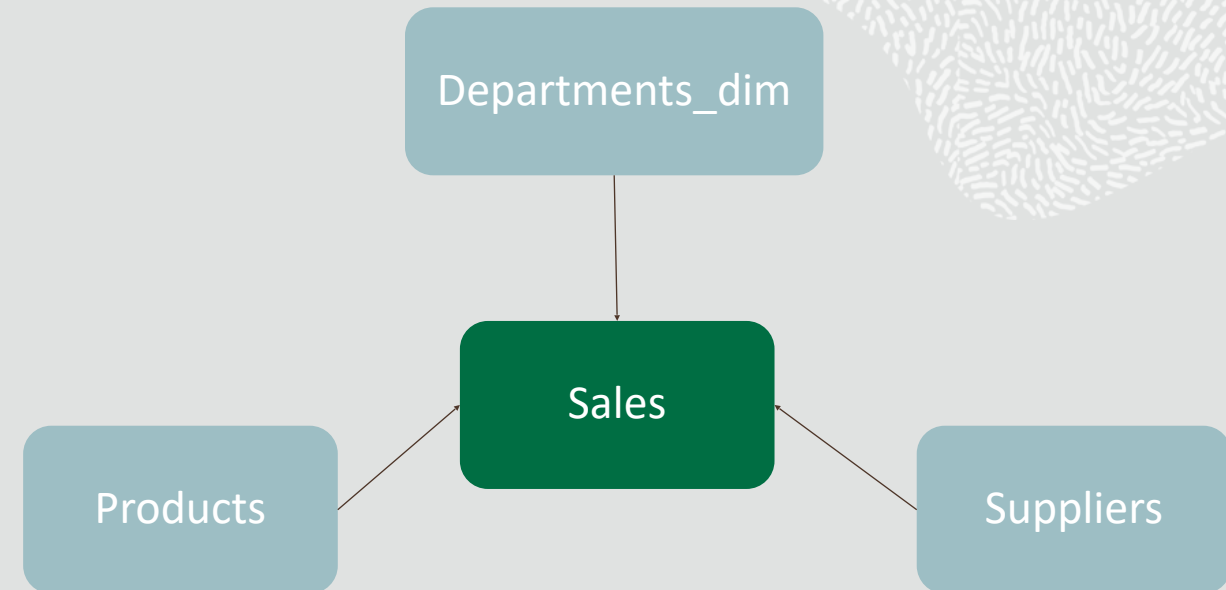


Right Deep Tree
(Desired Execution Plan)



Schema and SQL Statement

```
SELECT
  CATEGORY_ID
, COUNTRY
, SUM(CTRL) as CTRL
, SUM(QUANTITY) AS S_Q
FROM
  (SELECT
    P.CATEGORY_ID
  , SP.COUNTRY
  , CASE
      WHEN SP.CATEGORY='CAT_' || P.CATEGORY_ID
      THEN 1
      ELSE 0
    END as CTRL
  , QUANTITY
  FROM
    SALES SL
  INNER JOIN DEPARTMENTS_DIM D ON SL.DEPARTMENT_ID=D.DEPARTMENT_ID
  INNER JOIN PRODUCTS P ON SL.PRODUCT_ID=P.PRODUCT_ID
  INNER JOIN SUPPLIERS SP ON SL.SUPPLIER_ID=SP.SUPPLIER_ID
  WHERE (D.CITY) IN ( 'CITY_1', 'CITY_2', 'CITY_3' )
  AND D.STORE_TYPE=0
  )
GROUP BY CATEGORY_ID, COUNTRY
```



Demo

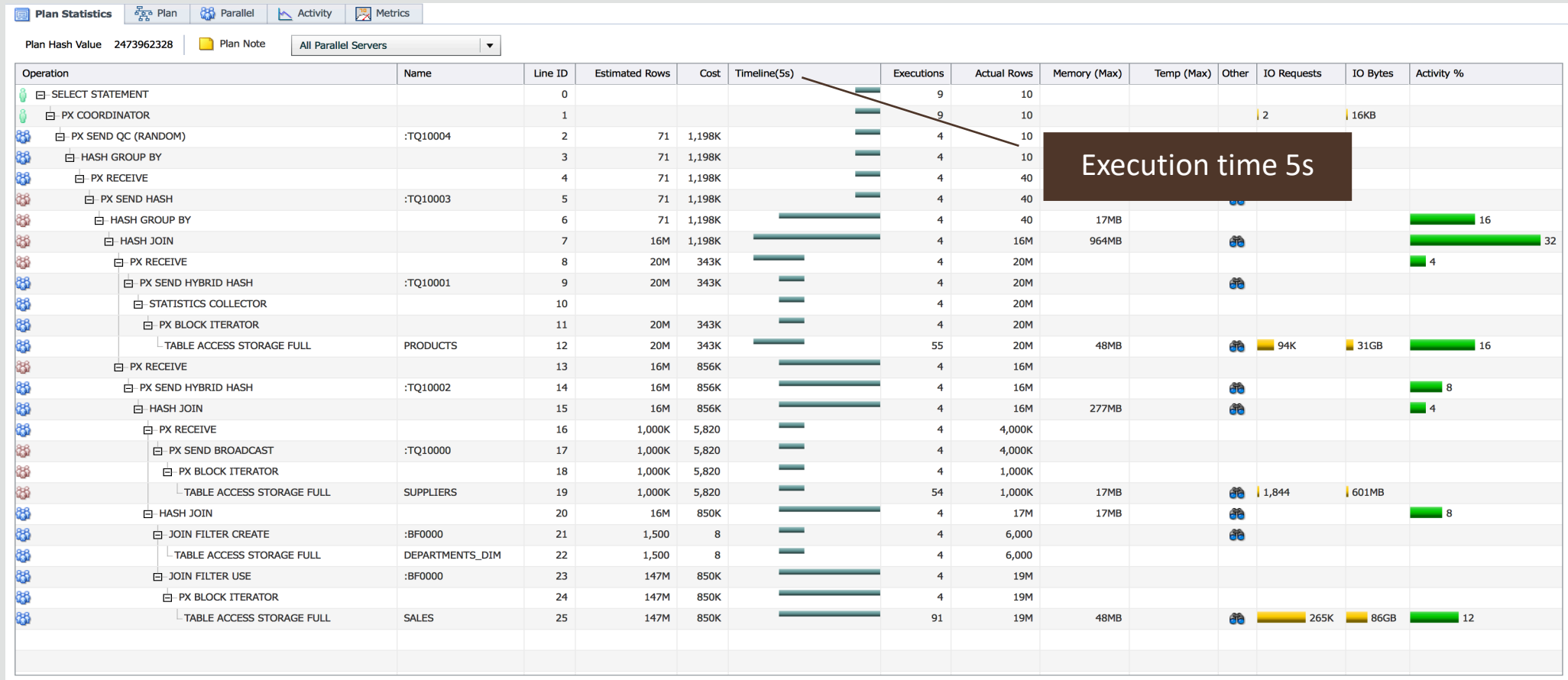
Desired Run

Denormalize the join between LOCATIONS, STORES and DEPARTMENTS

Plan Statistics													
Plan Hash Value 2473962328 Plan Note All Parallel Servers													
Operation	Name	Line ID	Estimated Rows	Cost	Timeline(5s)	Executions	Actual Rows	Memory (Max)	Temp (Max)	Other	IO Requests	IO Bytes	Activity %
SELECT STATEMENT		0				9	10						
PX COORDINATOR		1				9	10				2	16KB	
PX SEND QC (RANDOM)	:TQ10004	2	71	1,198K		4	10						
HASH GROUP BY		3	71	1,198K		4	10	6MB					
PX RECEIVE		4	71	1,198K		4	40						
PX SEND HASH	:TQ10003	5	71	1,198K		4	40						
HASH GROUP BY		6	71	1,198K		4	40	17MB					16
HASH JOIN		7	16M	1,198K		4	16M	964MB					32
PX RECEIVE		8	20M	343K		4	20M						4
PX SEND HYBRID HASH	:TQ10001	9	20M	343K		4	20M						
STATISTICS COLLECTOR		10				4	20M						
PX BLOCK ITERATOR		11	20M	343K		4	20M						
TABLE ACCESS STORAGE FULL	PRODUCTS	12	20M	343K		55	20M	48MB			94K	31GB	16
PX RECEIVE		13	16M	856K		4	16M						
PX SEND HYBRID HASH	:TQ10002	14	16M	856K		4	16M						8
HASH JOIN		15	16M	856K		4	16M	277MB					4
PX RECEIVE		16	1,000K	5,820		4	4,000K						
PX SEND BROADCAST	:TQ10000	17	1,000K	5,820		4	4,000K						
PX BLOCK ITERATOR		18	1,000K	5,820		4	1,000K						
TABLE ACCESS STORAGE FULL	SUPPLIERS	19	1,000K	5,820		54	1,000K	17MB			1,844	601MB	
HASH JOIN		20	16M	850K		4	17M	17MB					8
JOIN FILTER CREATE	:BF0000	21	1,500	8		4	6,000						
TABLE ACCESS STORAGE FULL	DEPARTMENTS_DIM	22	1,500	8		4	6,000						
JOIN FILTER USE	:BF0000	23	147M	850K		4	19M						
PX BLOCK ITERATOR		24	147M	850K		4	19M						
TABLE ACCESS STORAGE FULL	SALES	25	147M	850K		91	19M	48MB			265K	86GB	12

Desired Run

Denormalize the join between LOCATIONS, STORES and DEPARTMENTS



Desired Run

Denormalize the join between LOCATIONS, STORES and DEPARTMENTS

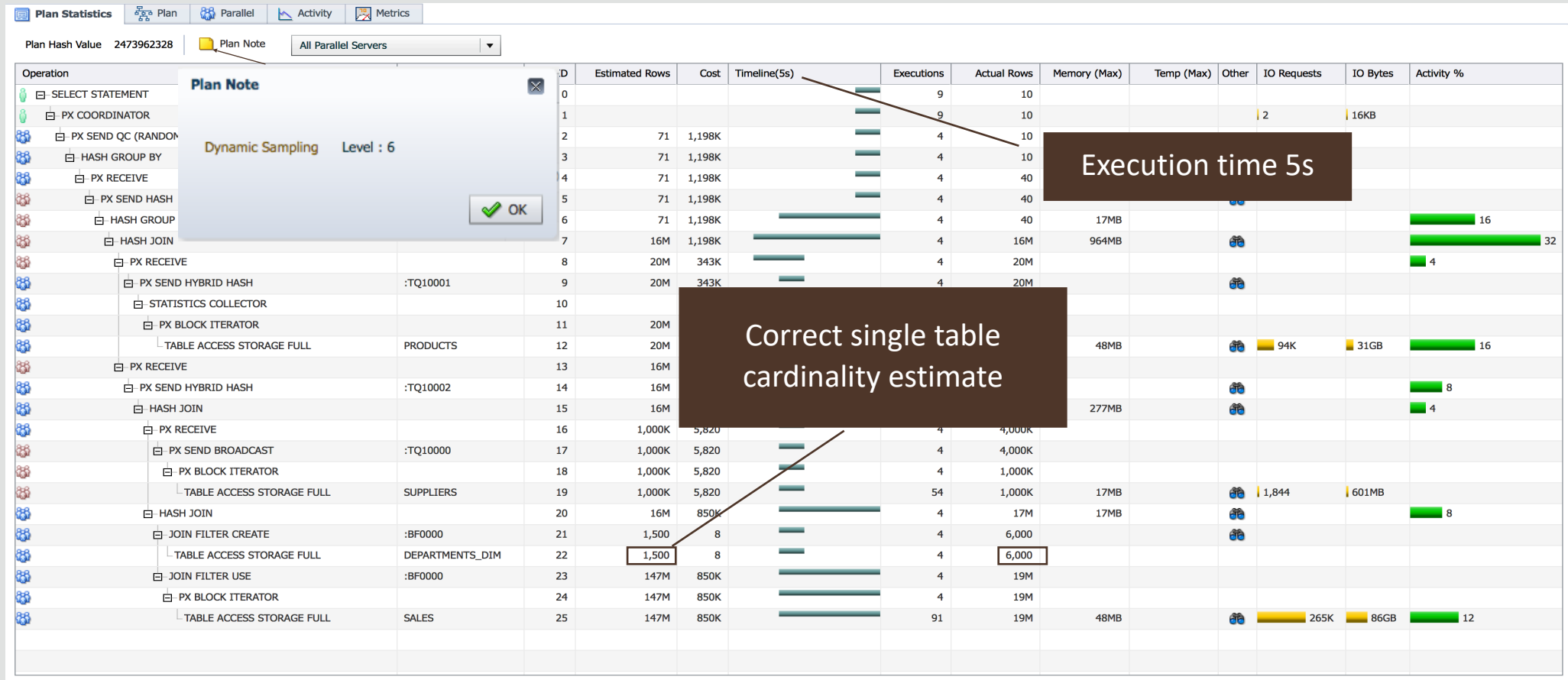
Plan Statistics													
Plan Hash Value 2473962328 Plan Note All Parallel Servers													
Operation	Name	Line ID	Estimated Rows	Cost	Timeline(5s)	Executions	Actual Rows	Memory (Max)	Temp (Max)	Other	IO Requests	IO Bytes	Activity %
SELECT STATEMENT		0				9	10						
PX COORDINATOR		1				9	10				2	16KB	
PX SEND QC (RANDOM)	:TQ10004	2	71	1,198K		4	10						
HASH GROUP BY		3	71	1,198K		4	10						
PX RECEIVE		4	71	1,198K		4	40						
PX SEND HASH	:TQ10003	5	71	1,198K		4	40						
HASH GROUP BY		6	71	1,198K		4	40	17MB					16
HASH JOIN		7	16M	1,198K		4	16M	964MB					32
PX RECEIVE		8	20M	343K		4	20M						4
PX SEND HYBRID HASH	:TQ10001	9	20M	343K		4	20M						
STATISTICS COLLECTOR		10											
PX BLOCK ITERATOR		11	20M										
TABLE ACCESS STORAGE FULL	PRODUCTS	12	20M					48MB			94K	31GB	16
PX RECEIVE		13	16M										
PX SEND HYBRID HASH	:TQ10002	14	16M										8
HASH JOIN		15	16M					277MB					4
PX RECEIVE		16	1,000K	5,820		4	4,000K						
PX SEND BROADCAST	:TQ10000	17	1,000K	5,820		4	4,000K						
PX BLOCK ITERATOR		18	1,000K	5,820		4	1,000K						
TABLE ACCESS STORAGE FULL	SUPPLIERS	19	1,000K	5,820		54	1,000K	17MB			1,844	601MB	8
HASH JOIN		20	16M	850K		4	17M	17MB					
JOIN FILTER CREATE	:BF0000	21	1,500	8		4	6,000						
TABLE ACCESS STORAGE FULL	DEPARTMENTS_DIM	22	1,500	8		4	6,000						
JOIN FILTER USE	:BF0000	23	147M	850K		4	19M						
PX BLOCK ITERATOR		24	147M	850K		4	19M						
TABLE ACCESS STORAGE FULL	SALES	25	147M	850K		91	19M	48MB			265K	86GB	12

Execution time 5s

Correct single table
cardinality estimate

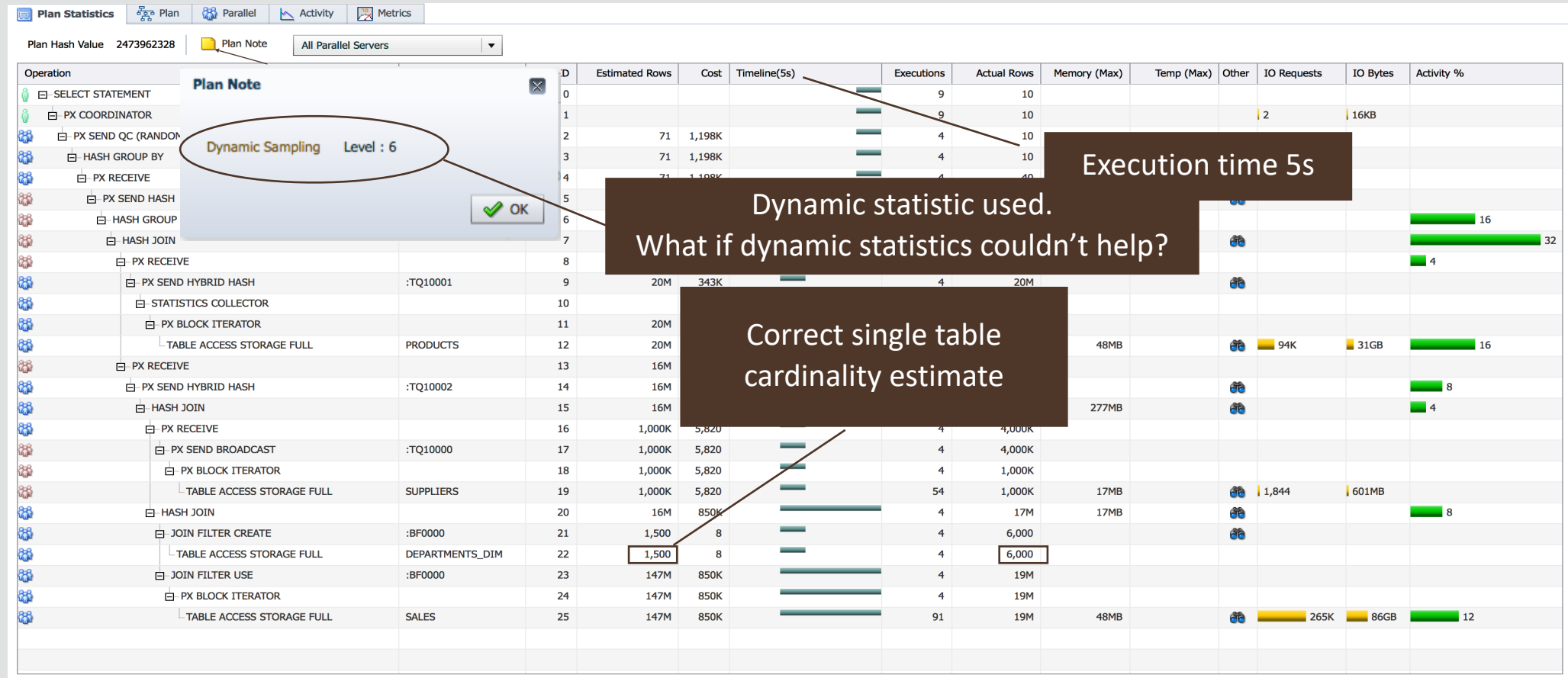
Desired Run

Denormalize the join between LOCATIONS, STORES and DEPARTMENTS



Desired Run

Denormalize the join between LOCATIONS, STORES and DEPARTMENTS



Desired Run

Denormalize the join between LOCATIONS, STORES and DEPARTMENTS

Plan Statistics													
Plan Hash Value 383059184 Plan Note													
Operation	Name	Line ID	Estimated Rows	Cost	Timeline(81s)	Executions	Actual Rows	Memory (Max)	Temp (Max)	Other	IO Requests	IO Bytes	Activity %
SELECT STATEMENT		0				9	10						
PX COORDINATOR		1				9	10						
PX SEND QC (RANDOM)	:TQ10002	2	71	861K		4	10						
HASH GROUP BY		3	71	861K		4	10	6MB					
PX RECEIVE		4	71	861K		4	40						
PX SEND HASH	:TQ10001	5	71	861K		4	40						
HASH GROUP BY		6	71	861K		4	40	17MB					3.14
NESTED LOOPS		7	9,091	861K		4	16M						.31
NESTED LOOPS		8	9,091	861K		4	16M						
HASH JOIN		9	9,091	856K		4	16M	4GB	788MB		3,152	2GB	4.72
PX RECEIVE		10	9,337	850K		4	66M						.63
PX SEND BROADCAST	:TQ10000	11	9,337	850K		4	66M						.63
HASH JOIN		12	9,337	850K		4	17M	17MB					
JOIN FILTER CREATE	:BF0000	13	1	8		4	6,000						
TABLE ACCESS STORAGE FULL	DEPARTMENTS_DIM	14	1	8		4	6,000						
JOIN FILTER USE	:BF0000	15	147M	850K		4	19M						
PX BLOCK ITERATOR		16	147M	850K		4	19M						
TABLE ACCESS STORAGE FULL	SALES	17	147M	850K		91	19M	48MB			265K	86GB	1.89
PX BLOCK ITERATOR		18	1,000K	5,820		4	1,000K						
TABLE ACCESS STORAGE FULL	SUPPLIERS	19	1,000K	5,820		54	1,000K	19MB			1,844	601MB	.63
INDEX UNIQUE SCAN	PRODUCTS_PK	20	1	1		19M	16M				4,849K	37GB	46
TABLE ACCESS BY INDEX ROWID	PRODUCTS	21	1	2		24M	16M				921K	7GB	42

Desired Run

Denormalize the join between LOCATIONS, STORES and DEPARTMENTS

Plan Statistics													
Plan Hash Value 383059184 Plan Note													
Operation	Name	Line ID	Estimated Rows	Cost	Timeline(81s)	Executions	Actual Rows	Memory (Max)	Temp (Max)	Other	IO Requests	IO Bytes	Activity %
SELECT STATEMENT		0				9	10						
PX COORDINATOR		1				9	10						
PX SEND QC (RANDOM)	:TQ10002	2	71	861K		4	10						
HASH GROUP BY		3	71	861K		4	10						
PX RECEIVE		4	71	861K		4	40						
PX SEND HASH	:TQ10001	5	71	861K		4	40						
HASH GROUP BY		6	71	861K		4	40	17MB					3.14
NESTED LOOPS		7	9,091	861K		4	16M						.31
NESTED LOOPS		8	9,091	861K		4	16M						
HASH JOIN		9	9,091	856K		4	16M	4GB	788MB		3,152	2GB	4.72
PX RECEIVE		10	9,337	850K		4	66M						.63
PX SEND BROADCAST	:TQ10000	11	9,337	850K		4	66M						.63
HASH JOIN		12	9,337	850K		4	17M	17MB					
JOIN FILTER CREATE	:BF0000	13	1	8		4	6,000						
TABLE ACCESS STORAGE FULL	DEPARTMENTS_DIM	14	1	8		4	6,000						
JOIN FILTER USE	:BF0000	15	147M	850K		4	19M						
PX BLOCK ITERATOR		16	147M	850K		4	19M						
TABLE ACCESS STORAGE FULL	SALES	17	147M	850K		91	19M	48MB			265K	86GB	1.89
PX BLOCK ITERATOR		18	1,000K	5,820		4	1,000K						
TABLE ACCESS STORAGE FULL	SUPPLIERS	19	1,000K	5,820		54	1,000K	19MB			1,844	601MB	.63
INDEX UNIQUE SCAN	PRODUCTS_PK	20	1	1		19M	16M				4,849K	37GB	46
TABLE ACCESS BY INDEX ROWID	PRODUCTS	21	1	2		24M	16M				921K	7GB	42

Execution time 81s

Desired Run

Denormalize the join between LOCATIONS, STORES and DEPARTMENTS

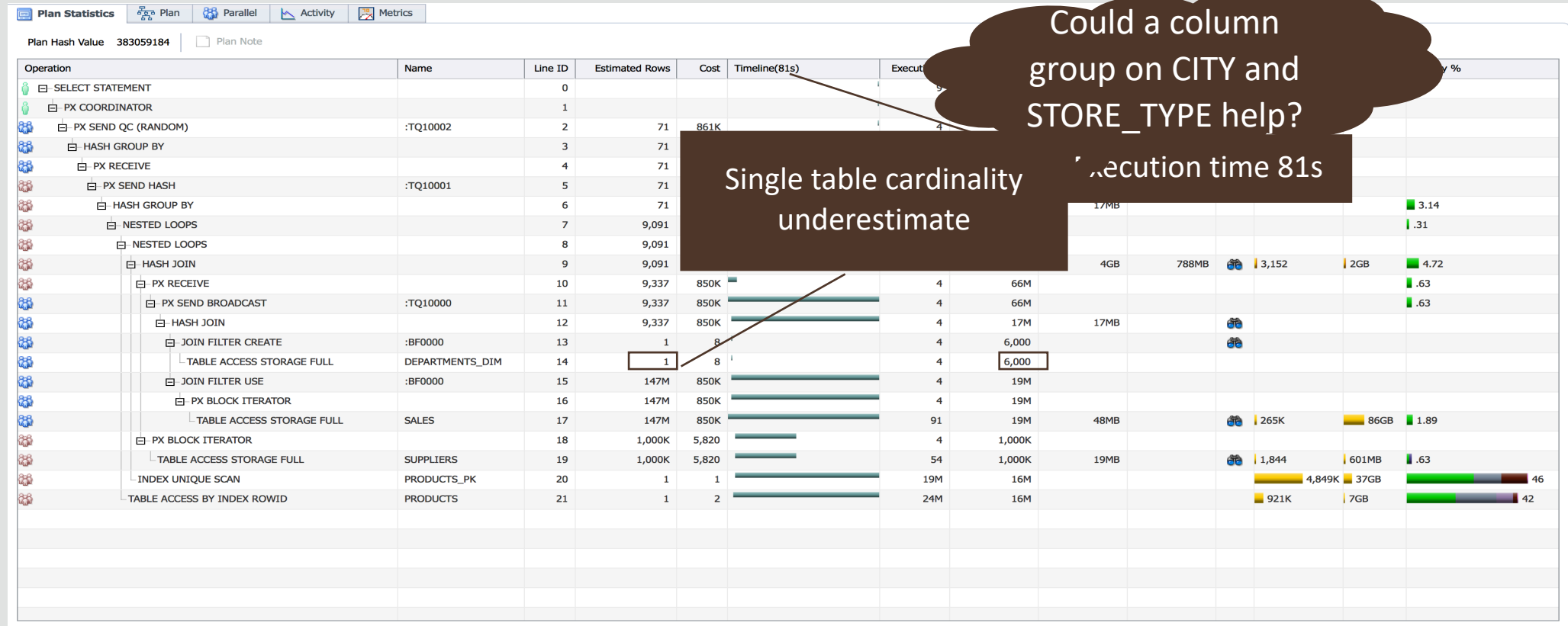
Plan Statistics													
Plan Hash Value 383059184 Plan Note													
Operation	Name	Line ID	Estimated Rows	Cost	Timeline(81s)	Executions	Actual Rows	Memory (Max)	Temp (Max)	Other	IO Requests	IO Bytes	Activity %
SELECT STATEMENT		0				9	10						
PX COORDINATOR		1				9	10						
PX SEND QC (RANDOM)	:TQ10002	2	71	861K		4	10						
HASH GROUP BY		3	71										
PX RECEIVE		4	71										
PX SEND HASH	:TQ10001	5	71										
HASH GROUP BY		6	71					17MB					3.14
NESTED LOOPS		7	9,091										.31
NESTED LOOPS		8	9,091										
HASH JOIN		9	9,091					4GB	788MB		3,152	2GB	4.72
PX RECEIVE		10	9,337	850K		4	66M						.63
PX SEND BROADCAST	:TQ10000	11	9,337	850K		4	66M						.63
HASH JOIN		12	9,337	850K		4	17M	17MB					
JOIN FILTER CREATE	:BF0000	13	1	8		4	6,000						
TABLE ACCESS STORAGE FULL	DEPARTMENTS_DIM	14	1	8		4	6,000						
JOIN FILTER USE	:BF0000	15	147M	850K		4	19M						
PX BLOCK ITERATOR		16	147M	850K		4	19M						
TABLE ACCESS STORAGE FULL	SALES	17	147M	850K		91	19M	48MB			265K	86GB	1.89
PX BLOCK ITERATOR		18	1,000K	5,820		4	1,000K						
TABLE ACCESS STORAGE FULL	SUPPLIERS	19	1,000K	5,820		54	1,000K	19MB			1,844	601MB	.63
INDEX UNIQUE SCAN	PRODUCTS_PK	20	1	1		19M	16M				4,849K	37GB	46
TABLE ACCESS BY INDEX ROWID	PRODUCTS	21	1	2		24M	16M				921K	7GB	42

Single table cardinality underestimate

Execution time 81s

Desired Run

Denormalize the join between LOCATIONS, STORES and DEPARTMENTS



Analysis: Column group on DEPARTMENTS_DIM(CITY, STORE_TYPE)

```
SELECT dbms_stats.create_extended_stats(USER, 'DEPARTMENTS_DIM', '(CITY,STORE_TYPE)') from dual;

EXEC dbms_stats.gather_table_stats(user, 'DEPARTMENTS_DIM', METHOD_OPT=>'FOR COLUMNS (CITY,STORE_TYPE) SIZE 2048 FOR ALL COLUMNS SIZE AUTO');

PL/SQL procedure successfully completed.
```

```
SELECT column_name, num_distinct, num_nulls, histogram, num_buckets
FROM user_tab_col_statistics
WHERE table_name = 'DEPARTMENTS_DIM';
```

COLUMN_NAME	NUM_DISTINCT	NUM_NULLS	HISTOGRAM	NUM_BUCKETS
DEPARTMENT_ID	50000	0	NONE	1
DEPARTMENT_NAME	50000	0	NONE	1
STORE_ID	5000	0	NONE	1

CITY	2080	0	HYBRID	254
STORE_TYPE	5	0	FREQUENCY	5
SYS_STUMZODT3BT6MDTDUS_1OYH3M#	2080	0	HYBRID	1878

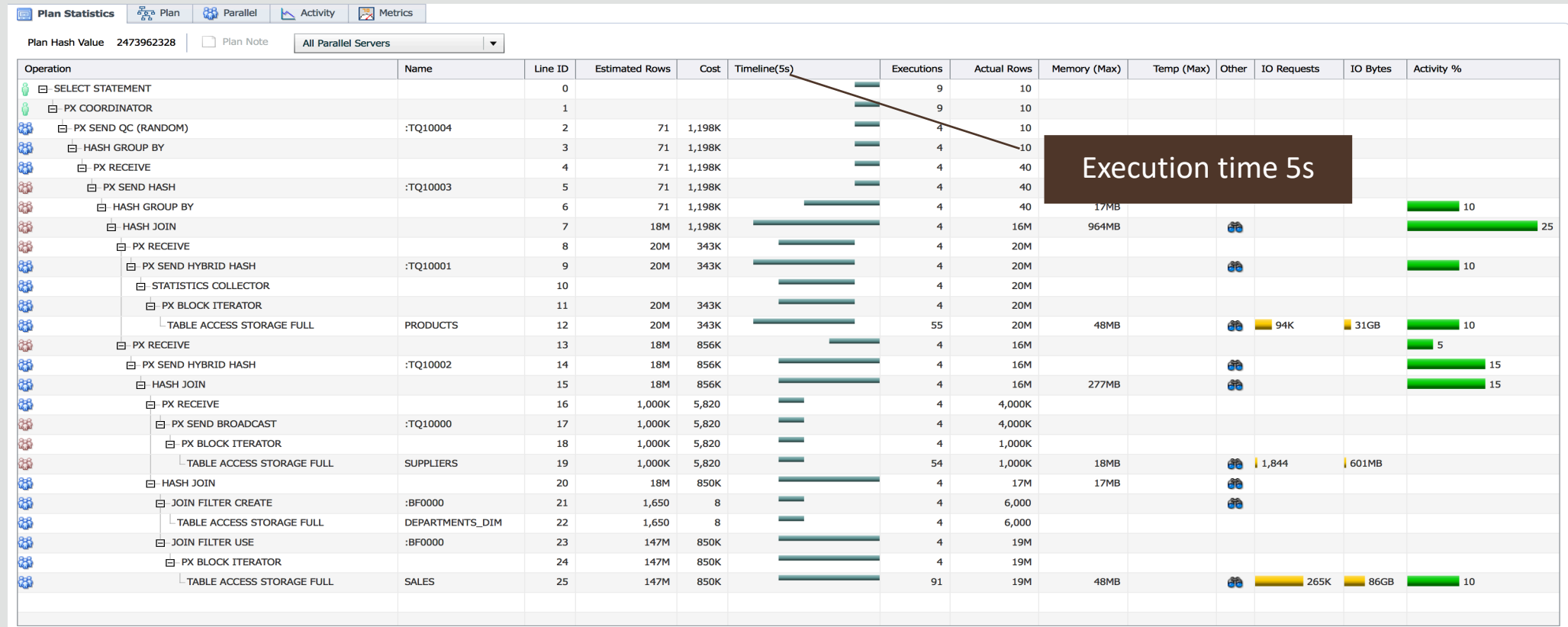
Desired Run

Denormalize the join between LOCATIONS, STORES and DEPARTMENTS

Plan Statistics													
Plan Hash Value 2473962328													
All Parallel Servers													
Operation	Name	Line ID	Estimated Rows	Cost	Timeline(5s)	Executions	Actual Rows	Memory (Max)	Temp (Max)	Other	IO Requests	IO Bytes	Activity %
SELECT STATEMENT		0				9	10						
PX COORDINATOR		1				9	10						
PX SEND QC (RANDOM)	:TQ10004	2	71	1,198K		4	10						
HASH GROUP BY		3	71	1,198K		4	10	6MB					
PX RECEIVE		4	71	1,198K		4	40						
PX SEND HASH	:TQ10003	5	71	1,198K		4	40						
HASH GROUP BY		6	71	1,198K		4	40	17MB					10
HASH JOIN		7	18M	1,198K		4	16M	964MB					25
PX RECEIVE		8	20M	343K		4	20M						
PX SEND HYBRID HASH	:TQ10001	9	20M	343K		4	20M						10
STATISTICS COLLECTOR		10				4	20M						
PX BLOCK ITERATOR		11	20M	343K		4	20M						
TABLE ACCESS STORAGE FULL	PRODUCTS	12	20M	343K		55	20M	48MB			94K	31GB	10
PX RECEIVE		13	18M	856K		4	16M						5
PX SEND HYBRID HASH	:TQ10002	14	18M	856K		4	16M						15
HASH JOIN		15	18M	856K		4	16M	277MB					15
PX RECEIVE		16	1,000K	5,820		4	4,000K						
PX SEND BROADCAST	:TQ10000	17	1,000K	5,820		4	4,000K						
PX BLOCK ITERATOR		18	1,000K	5,820		4	1,000K						
TABLE ACCESS STORAGE FULL	SUPPLIERS	19	1,000K	5,820		54	1,000K	18MB			1,844	601MB	
HASH JOIN		20	18M	850K		4	17M	17MB					
JOIN FILTER CREATE	:BF0000	21	1,650	8		4	6,000						
TABLE ACCESS STORAGE FULL	DEPARTMENTS_DIM	22	1,650	8		4	6,000						
JOIN FILTER USE	:BF0000	23	147M	850K		4	19M						
PX BLOCK ITERATOR		24	147M	850K		4	19M						
TABLE ACCESS STORAGE FULL	SALES	25	147M	850K		91	19M	48MB			265K	86GB	10

Desired Run

Denormalize the join between LOCATIONS, STORES and DEPARTMENTS



Desired Run

Denormalize the join between LOCATIONS, STORES and DEPARTMENTS

Plan Statistics													
Plan Hash Value 2473962328													
All Parallel Servers													
Operation	Name	Line ID	Estimated Rows	Cost	Timeline(5s)	Executions	Actual Rows	Memory (Max)	Temp (Max)	Other	IO Requests	IO Bytes	Activity %
SELECT STATEMENT		0				9	10						
PX COORDINATOR		1				9	10						
PX SEND QC (RANDOM)	:TQ10004	2	71	1,198K		4	10						
HASH GROUP BY		3	71	1,198K		4	10						
PX RECEIVE		4	71	1,198K		4	40						
PX SEND HASH	:TQ10003	5	71	1,198K		4	40						
HASH GROUP BY		6	71	1,198K		4	40	17MB					10
HASH JOIN		7	18M	1,198K		4	16M	964MB					25
PX RECEIVE		8	20M	343K		4	20M						
PX SEND HYBRID HASH	:TQ10001	9	20M	343K		4	20M						10
STATISTICS COLLECTOR		10				4	20M						
PX BLOCK ITERATOR		11	20M										
TABLE ACCESS STORAGE FULL	PRODUCTS	12	20M					48MB			94K	31GB	10
PX RECEIVE		13	18M										5
PX SEND HYBRID HASH	:TQ10002	14	18M										15
HASH JOIN		15	18M					277MB					15
PX RECEIVE		16	1,000K										
PX SEND BROADCAST	:TQ10000	17	1,000K										
PX BLOCK ITERATOR		18	1,000K	5,820		4	1,000K						
TABLE ACCESS STORAGE FULL	SUPPLIERS	19	1,000K	5,820		54	1,000K	18MB			1,844	601MB	
HASH JOIN		20	18M	850K		4	17M	17MB					
JOIN FILTER CREATE	:BF0000	21	1,650	8		4	6,000						
TABLE ACCESS STORAGE FULL	DEPARTMENTS_DIM	22	1,650	8		4	6,000						
JOIN FILTER USE	:BF0000	23	147M	850K		4	19M						
PX BLOCK ITERATOR		24	147M	850K		4	19M						
TABLE ACCESS STORAGE FULL	SALES	25	147M	850K		91	19M	48MB			265K	86GB	10

Execution time 5s

Very good single table
cardinality estimate

The art of finding excuses

The art of finding excuses

- “This approach requires schema changes...”

The art of finding excuses

- “This approach requires schema changes...”
- “This approach requires code changes (ETL, queries, etc) ...”

The art of finding excuses

- “This approach requires schema changes...”
- “This approach requires code changes (ETL, queries, etc) ...”
- “If we changed the code, we would have to test it ...”

The art of finding excuses

- “This approach requires schema changes...”
- “This approach requires code changes (ETL, queries, etc) ...”
- “If we changed the code, we would have to test it ...”
- “The person who developed this code is no longer with the company hence we cannot implement such a solution ...”

The art of finding excuses

- “This approach requires schema changes...”
- “This approach requires code changes (ETL, queries, etc) ...”
- “If we changed the code, we would have to test it ...”
- “The person who developed this code is no longer with the company hence we cannot implement such a solution ...”
- “It’s a 3rd party application ...”

The art of finding excuses

- “This approach requires schema changes...”
- “This approach requires code changes (ETL, queries, etc) ...”
- “If we changed the code, we would have to test it ...”
- “The person who developed this code is no longer with the company hence we cannot implement such a solution ...”
- “It’s a 3rd party application ...”
- “After much consideration the management decided to stick with the current solution ...”

Enter the Materialized View

- Traditionally the Materialized View (MV) is to

- Materialize expensive joins

- Summarize or rollup frequently aggregated data

- Some Key Properties

- A query can automatically be re-written by the optimizer to use an MV

- Transparent to user queries

- No data model changes required

- MV's can automatically be kept up to date by the database

Create an MV for the join between LOCATIONS, STORES and DEPARTMENTS

```
CREATE MATERIALIZED VIEW DEP_STORE_LOC_MV
PARALLEL 4
ENABLE QUERY REWRITE
AS
SELECT D.*
       , L.CITY
       , S.STORE_TYPE
from DEPARTMENTS D
     , STORES S
     , LOCATIONS L
WHERE D.STORE_ID=S.STORE_ID
AND S.LOCATION_ID=L.LOCATION_ID;
```

Benefit of Materialized View

- In this example

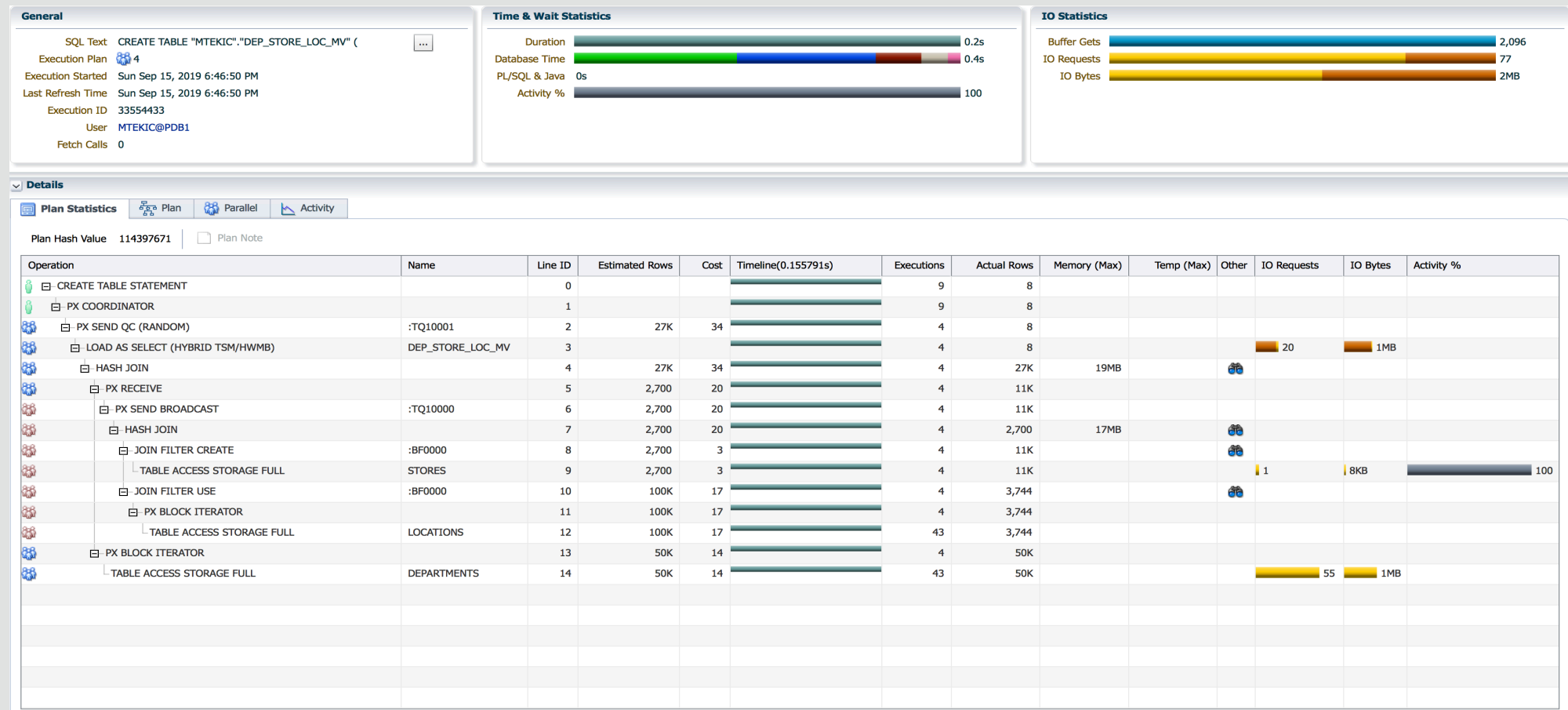
- The MV is not used to rollup aggregates

- The Joins in this case are not expensive

- The MV is used to improve cardinality estimates

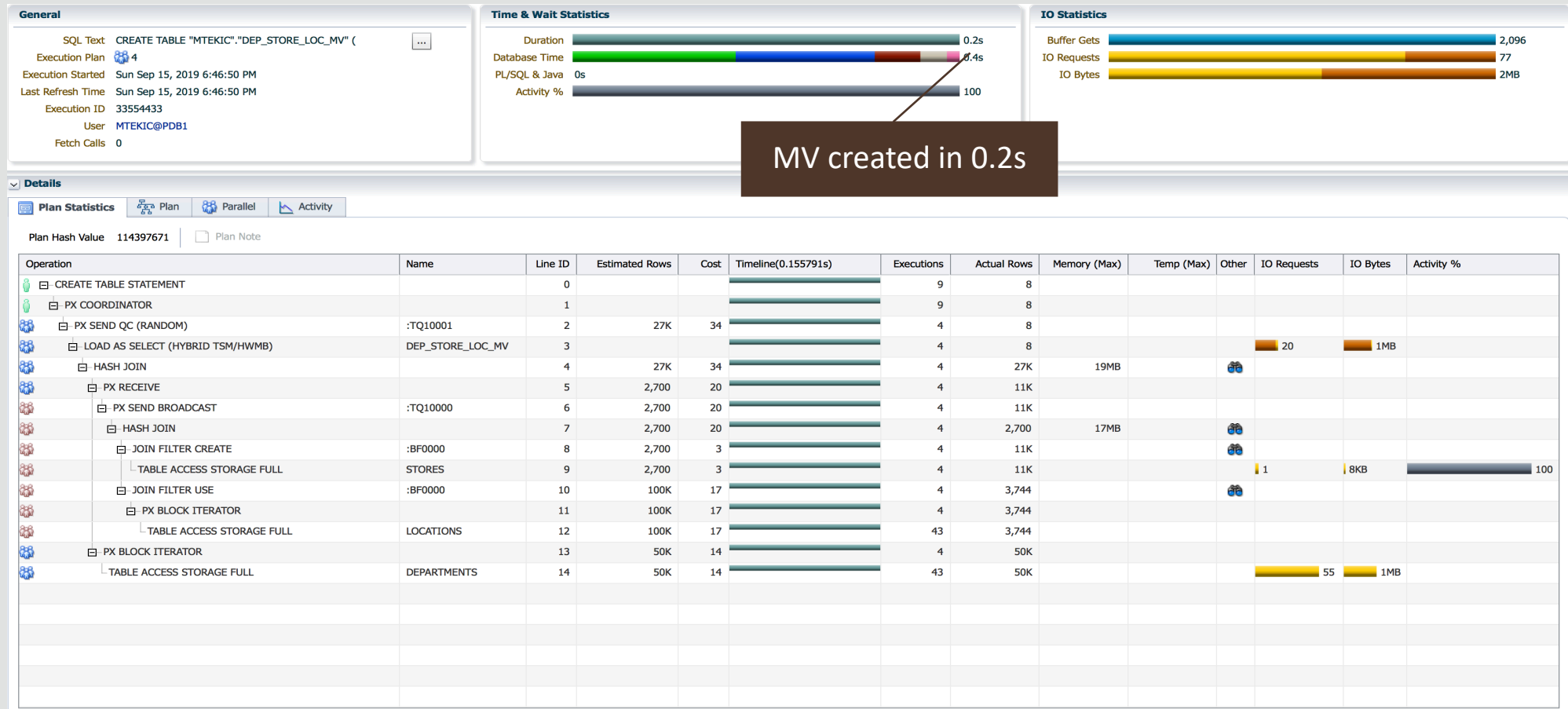
Query Rewrite for Materialized View

MV creation is fast



Query Rewrite for Materialized View

MV creation is fast



Materialized View

Cardinality underestimate still a problem

Monitored SQL Execution Details: azmv8ma1147ff

Overview

Details

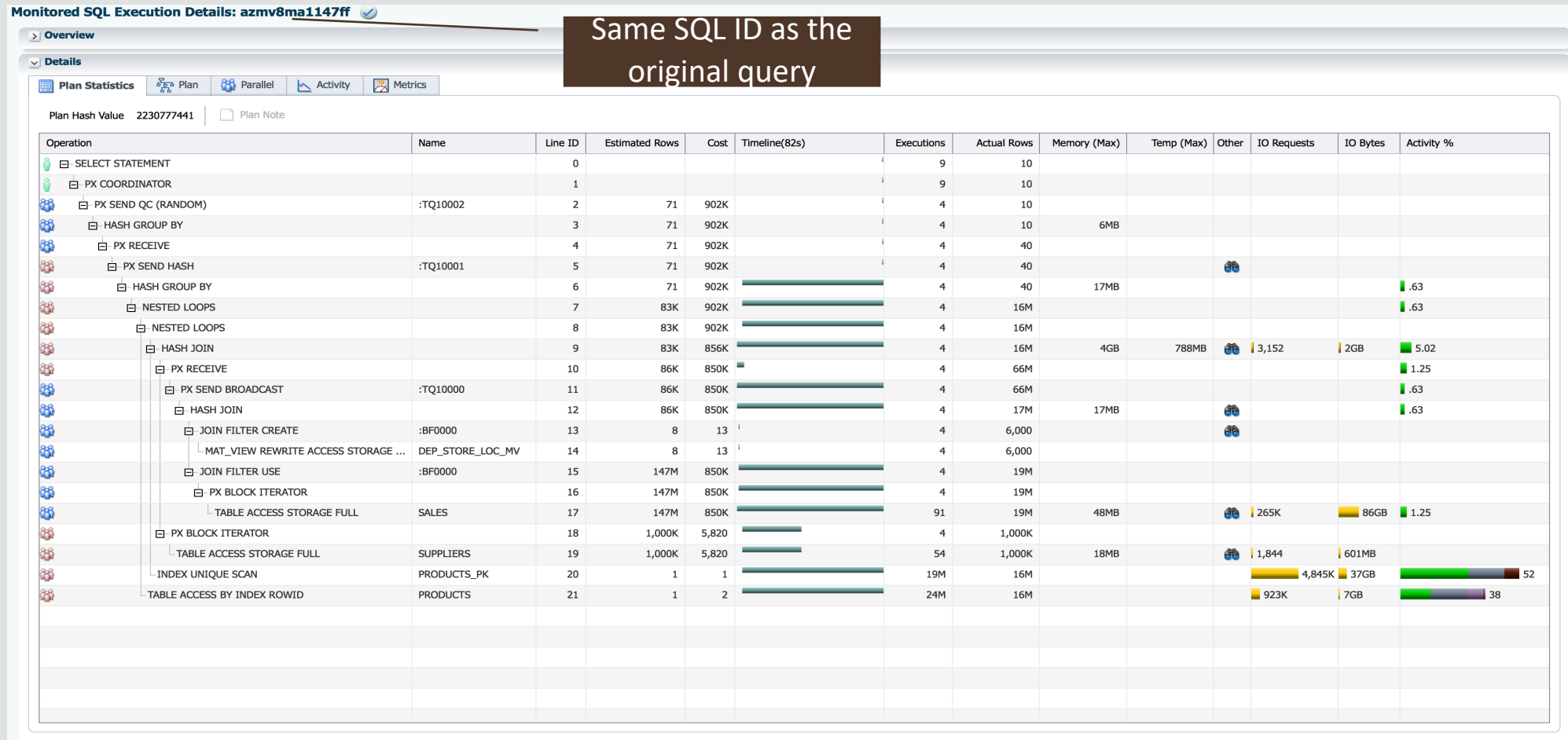
Plan Statistics Plan Parallel Activity Metrics

Plan Hash Value 2230777441 Plan Note

Operation	Name	Line ID	Estimated Rows	Cost	Timeline(82s)	Executions	Actual Rows	Memory (Max)	Temp (Max)	Other	IO Requests	IO Bytes	Activity %
SELECT STATEMENT		0				9	10						
PX COORDINATOR		1				9	10						
PX SEND QC (RANDOM)	:TQ10002	2	71	902K		4	10						
HASH GROUP BY		3	71	902K		4	10	6MB					
PX RECEIVE		4	71	902K		4	40						
PX SEND HASH	:TQ10001	5	71	902K		4	40						
HASH GROUP BY		6	71	902K		4	40	17MB					.63
NESTED LOOPS		7	83K	902K		4	16M						.63
NESTED LOOPS		8	83K	902K		4	16M						
HASH JOIN		9	83K	856K		4	16M	4GB	788MB		3,152	2GB	5.02
PX RECEIVE		10	86K	850K		4	66M						1.25
PX SEND BROADCAST	:TQ10000	11	86K	850K		4	66M						.63
HASH JOIN		12	86K	850K		4	17M	17MB					.63
JOIN FILTER CREATE	:BF0000	13	8	13		4	6,000						
MAT_VIEW REWRITE ACCESS STORAGE FULL	DEP_STORE_LOC_MV	14	8	13		4	6,000						
JOIN FILTER USE	:BF0000	15	147M	850K		4	19M						
PX BLOCK ITERATOR		16	147M	850K		4	19M						
TABLE ACCESS STORAGE FULL	SALES	17	147M	850K		91	19M	48MB			265K	86GB	1.25
PX BLOCK ITERATOR		18	1,000K	5,820		4	1,000K						
TABLE ACCESS STORAGE FULL	SUPPLIERS	19	1,000K	5,820		54	1,000K	18MB			1,844	601MB	
INDEX UNIQUE SCAN	PRODUCTS_PK	20	1	1		19M	16M				4,845K	37GB	52
TABLE ACCESS BY INDEX ROWID	PRODUCTS	21	1	2		24M	16M				923K	7GB	38

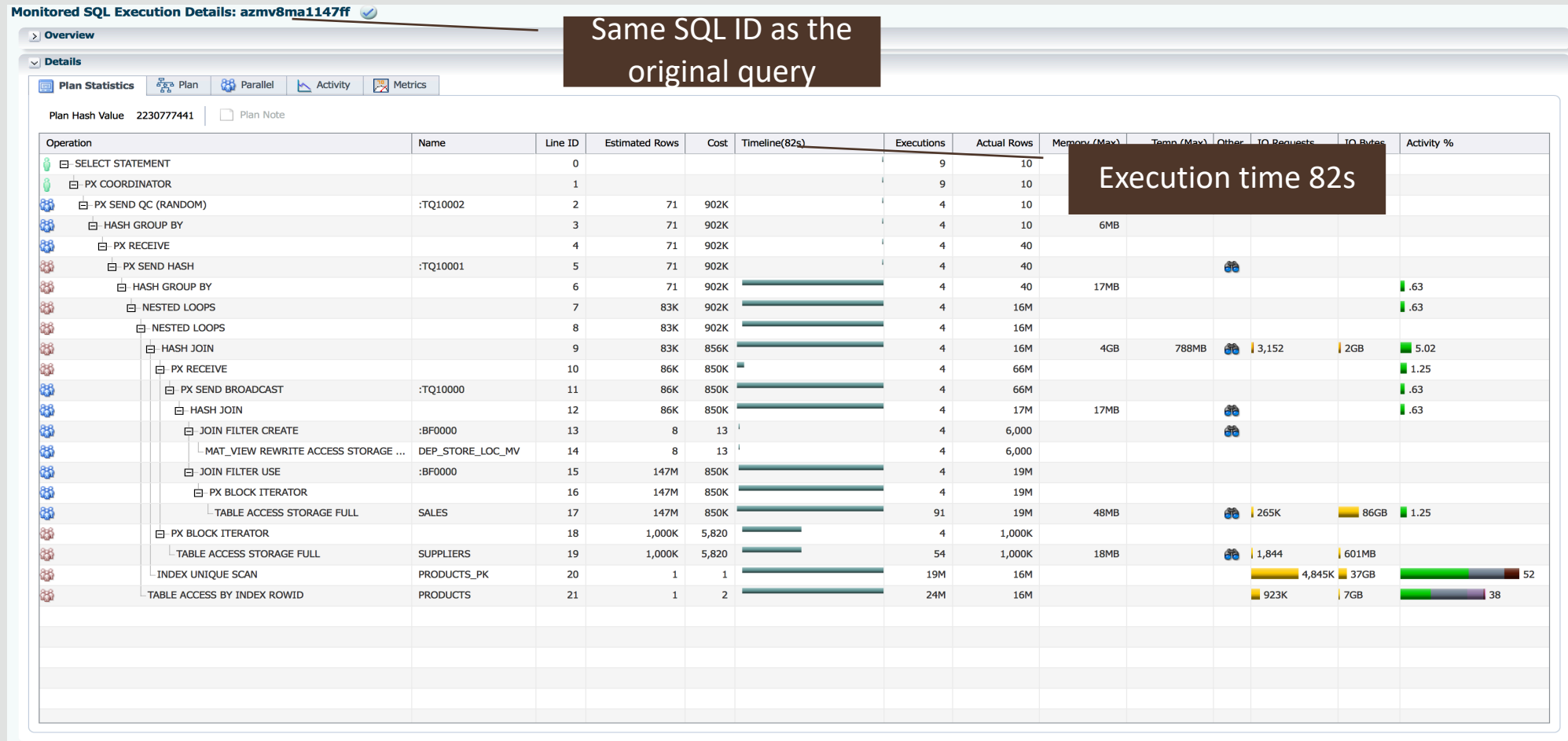
Materialized View

Cardinality underestimate still a problem



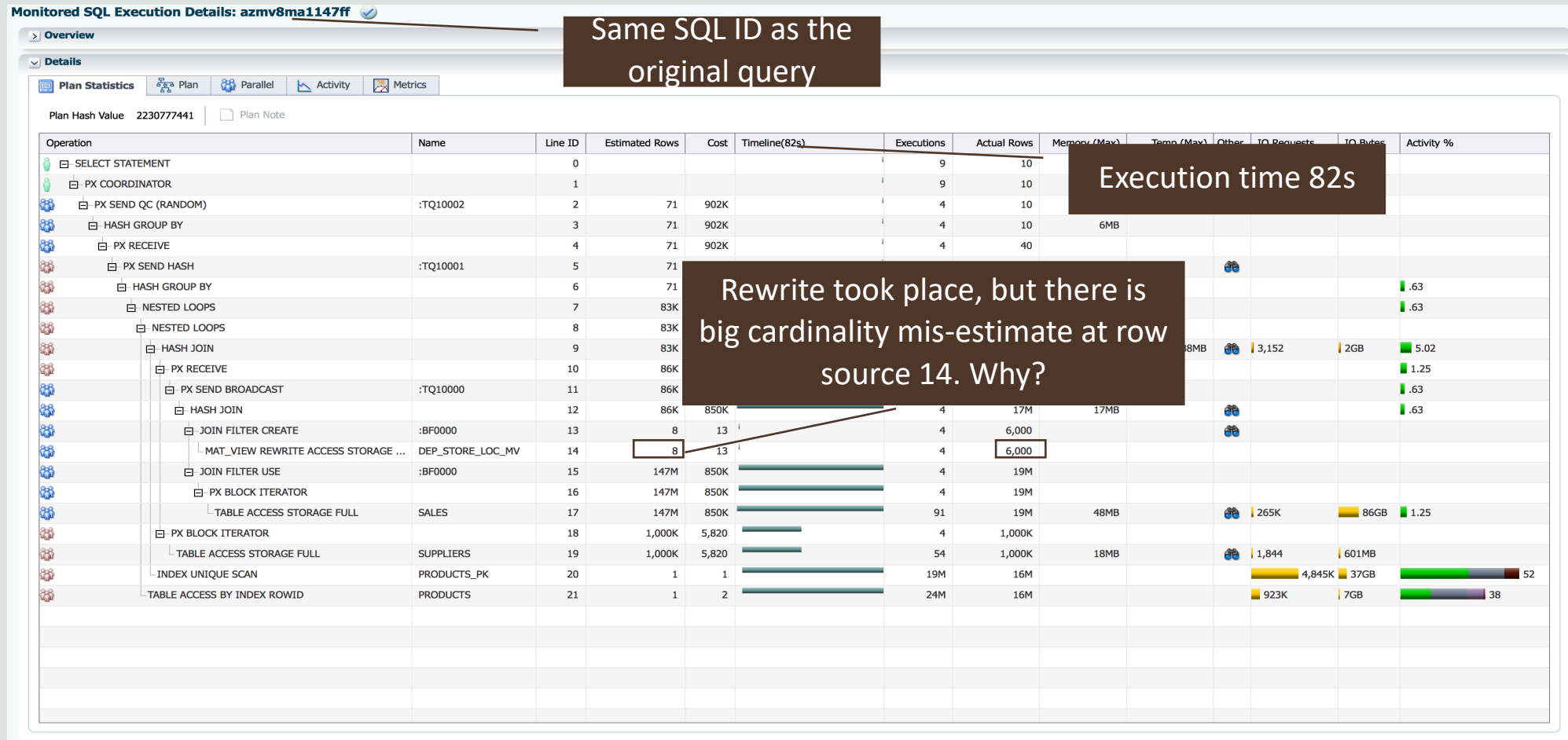
Materialized View

Cardinality underestimate still a problem



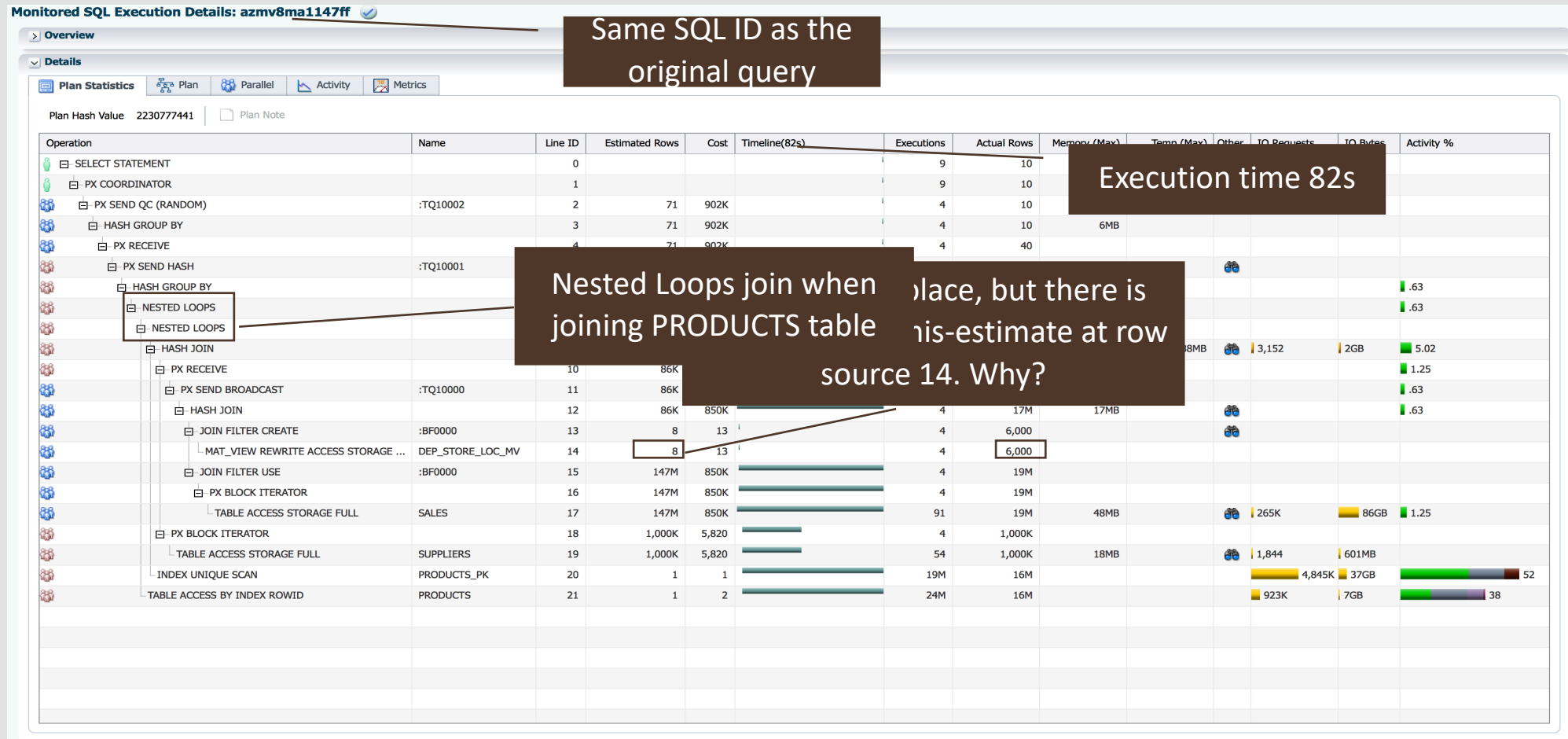
Materialized View

Cardinality underestimate still a problem



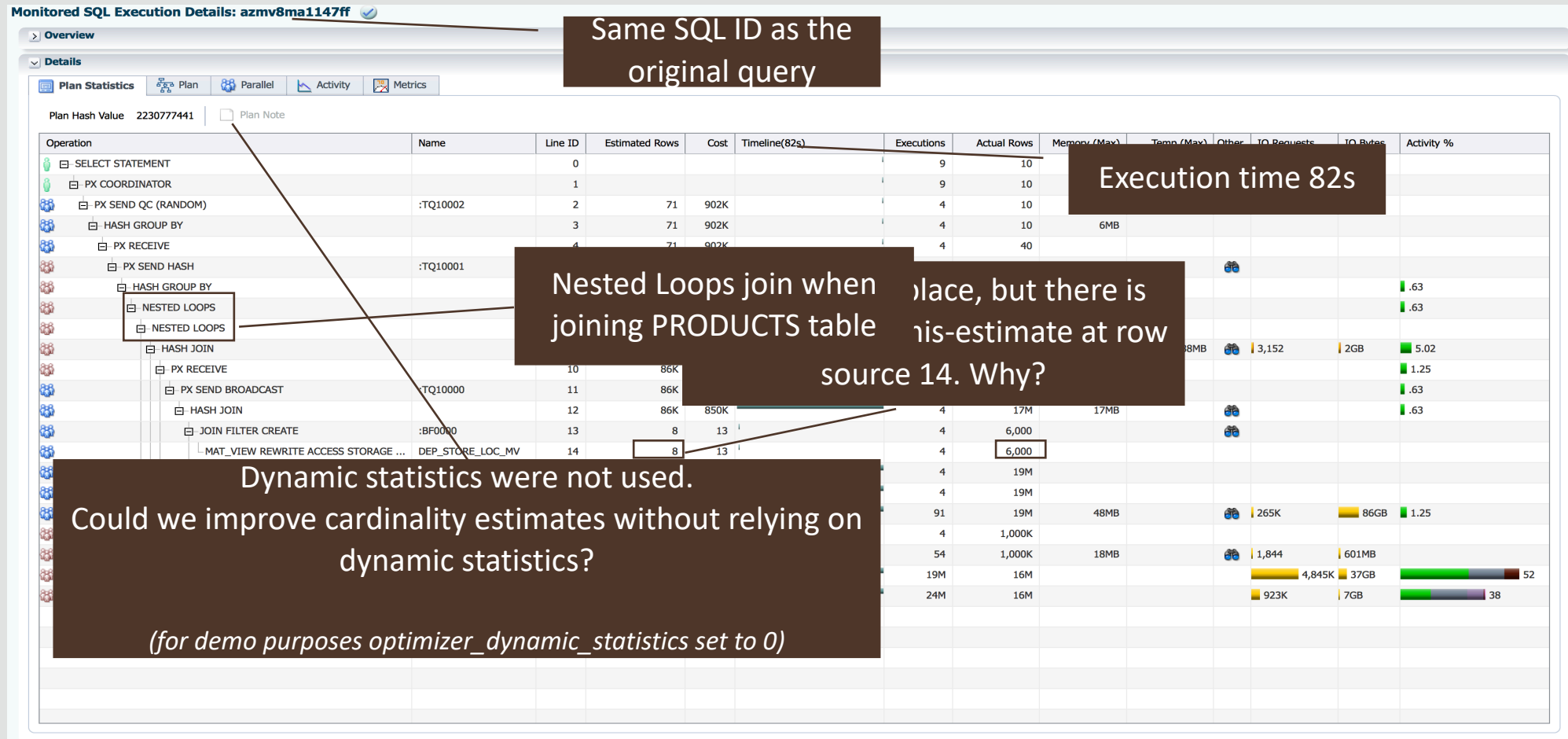
Materialized View

Cardinality underestimate still a problem



Materialized View

Cardinality underestimate still a problem



Analysis: Column group on DEP_STORE_LOC_MV(CITY, STORE_TYPE)



Analysis: Column group on DEP_STORE_LOC_MV(CITY, STORE_TYPE)

```
SELECT num_rows
FROM user_tab_statistics
WHERE table_name = 'DEP_STORE_LOC_MV';
```

NUM_ROWS
27000

```
SELECT column_name, num_distinct, num_nulls, num_buckets
FROM user_tab_col_statistics
WHERE table_name = 'DEP_STORE_LOC_MV'
AND column_name IN ('CITY','STORE_TYPE');
```

COLUMN_NAME	NUM_DISTINCT	NUM_NULLS	NUM_BUCKETS
CITY	2080	0	254
STORE_TYPE	5	0	5

```
SELECT count(DISTINCT CITY||' '||store_type) c_check
FROM DEP_STORE_LOC_MV;
```

C_CHECK
2080

Analysis: Column group on DEP_STORE_LOC_MV(CITY, STORE_TYPE)

```
SELECT num_rows
FROM user_tab_statistics
WHERE table_name = 'DEP_STORE_LOC_MV';
```

NUM_ROWS
27000

```
SELECT column_name, num_distinct, num_nulls, num_buckets
FROM user_tab_col_statistics
WHERE table_name = 'DEP_STORE_LOC_MV'
AND column_name IN ('CITY','STORE_TYPE');
```

COLUMN_NAME	NUM_DISTINCT	NUM_NULLS	NUM_BUCKETS
CITY	2080	0	254
STORE_TYPE	5	0	5

```
SELECT count(DISTINCT CITY||' '||store_type) c_check
FROM DEP_STORE_LOC_MV;
```

C_CHECK
2080

Indicates strong correlation
between CITY and STORE_TYPE

Analysis: Column group on DEP_STORE_LOC_MV(CITY, STORE_TYPE)

```
SELECT num_rows
FROM user_tab_statistics
WHERE table_name = 'DEP_STORE_LOC_MV';
```

NUM_ROWS
27000

```
SELECT column_name, num_distinct, num_nulls, num_buckets
FROM user_tab_col_statistics
WHERE table_name = 'DEP_STORE_LOC_MV'
AND column_name IN ('CITY','STORE_TYPE');
```

COLUMN_NAME	NUM_DISTINCT	NUM_NULLS	NUM_BUCKETS
CITY	2080	0	254
STORE_TYPE	5	0	5

```
SELECT count(DISTINCT CITY||' '||store_type) c_check
FROM DEP_STORE_LOC_MV;
```

C_CHECK
2080

Indicates strong correlation between CITY and STORE_TYPE

```
SELECT
dbms_stats.create_extended_stats(USER,'DEP_STORE_LOC_MV','(CITY,STORE_TYPE)')
from dual;
```

```
EXEC dbms_stats.gather_table_stats(user, 'DEP_STORE_LOC_MV', METHOD_OPT=>'FOR
COLUMNS (CITY,STORE_TYPE) SIZE 2048 FOR ALL COLUMNS SIZE AUTO');
```

PL/SQL procedure successfully completed.

```
SELECT column_name, num_distinct, num_nulls, histogram, num_buckets
FROM user_tab_col_statistics
WHERE table_name = 'DEP_STORE_LOC_MV';
```

COLUMN_NAME	NUM_DISTINCT	NUM_NULLS	HISTOGRAM	NUM_BUCKETS
DEPARTMENT_ID	27000	0	NONE	1
DEPARTMENT_NAME	27000	0	NONE	1
STORE_ID	2700	0	NONE	1
CITY	2080	0	HYBRID	254
STORE_TYPE	5	0	FREQUENCY	5
SYS_STUMZODT3BT6MDTDUS_10YH3M#	2080	0	HYBRID	1878

Materialized View

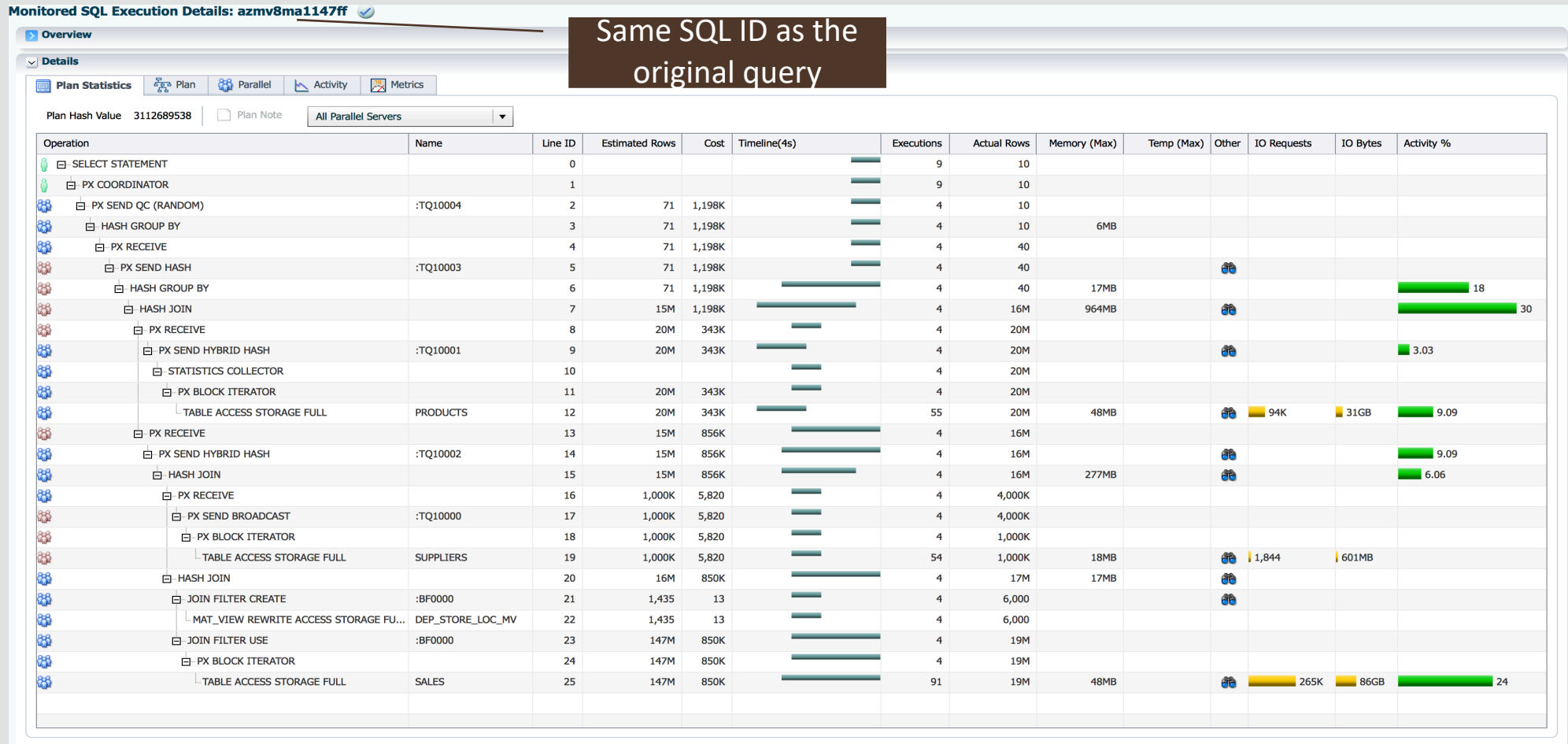
With extended statistics

Monitored SQL Execution Details: azmv8ma1147ff

Overview													
Details													
Plan Statistics													
Plan													
Parallel													
Activity													
Metrics													
Plan Hash Value 3112689538													
Plan Note													
All Parallel Servers													
Operation	Name	Line ID	Estimated Rows	Cost	Timeline(4s)	Executions	Actual Rows	Memory (Max)	Temp (Max)	Other	IO Requests	IO Bytes	Activity %
SELECT STATEMENT		0				9	10						
PX COORDINATOR		1				9	10						
PX SEND QC (RANDOM)	:TQ10004	2	71	1,198K		4	10						
HASH GROUP BY		3	71	1,198K		4	10	6MB					
PX RECEIVE		4	71	1,198K		4	40						
PX SEND HASH	:TQ10003	5	71	1,198K		4	40						
HASH GROUP BY		6	71	1,198K		4	40	17MB					18
HASH JOIN		7	15M	1,198K		4	16M	964MB					30
PX RECEIVE		8	20M	343K		4	20M						
PX SEND HYBRID HASH	:TQ10001	9	20M	343K		4	20M						3.03
STATISTICS COLLECTOR		10				4	20M						
PX BLOCK ITERATOR		11	20M	343K		4	20M						
TABLE ACCESS STORAGE FULL	PRODUCTS	12	20M	343K		55	20M	48MB			94K	31GB	9.09
PX RECEIVE		13	15M	856K		4	16M						
PX SEND HYBRID HASH	:TQ10002	14	15M	856K		4	16M						9.09
HASH JOIN		15	15M	856K		4	16M	277MB					6.06
PX RECEIVE		16	1,000K	5,820		4	4,000K						
PX SEND BROADCAST	:TQ10000	17	1,000K	5,820		4	4,000K						
PX BLOCK ITERATOR		18	1,000K	5,820		4	1,000K						
TABLE ACCESS STORAGE FULL	SUPPLIERS	19	1,000K	5,820		54	1,000K	18MB			1,844	601MB	
HASH JOIN		20	16M	850K		4	17M	17MB					
JOIN FILTER CREATE	:BF0000	21	1,435	13		4	6,000						
MAT_VIEW REWRITE ACCESS STORAGE FU...	DEP_STORE_LOC_MV	22	1,435	13		4	6,000						
JOIN FILTER USE	:BF0000	23	147M	850K		4	19M						
PX BLOCK ITERATOR		24	147M	850K		4	19M						
TABLE ACCESS STORAGE FULL	SALES	25	147M	850K		91	19M	48MB			265K	86GB	24

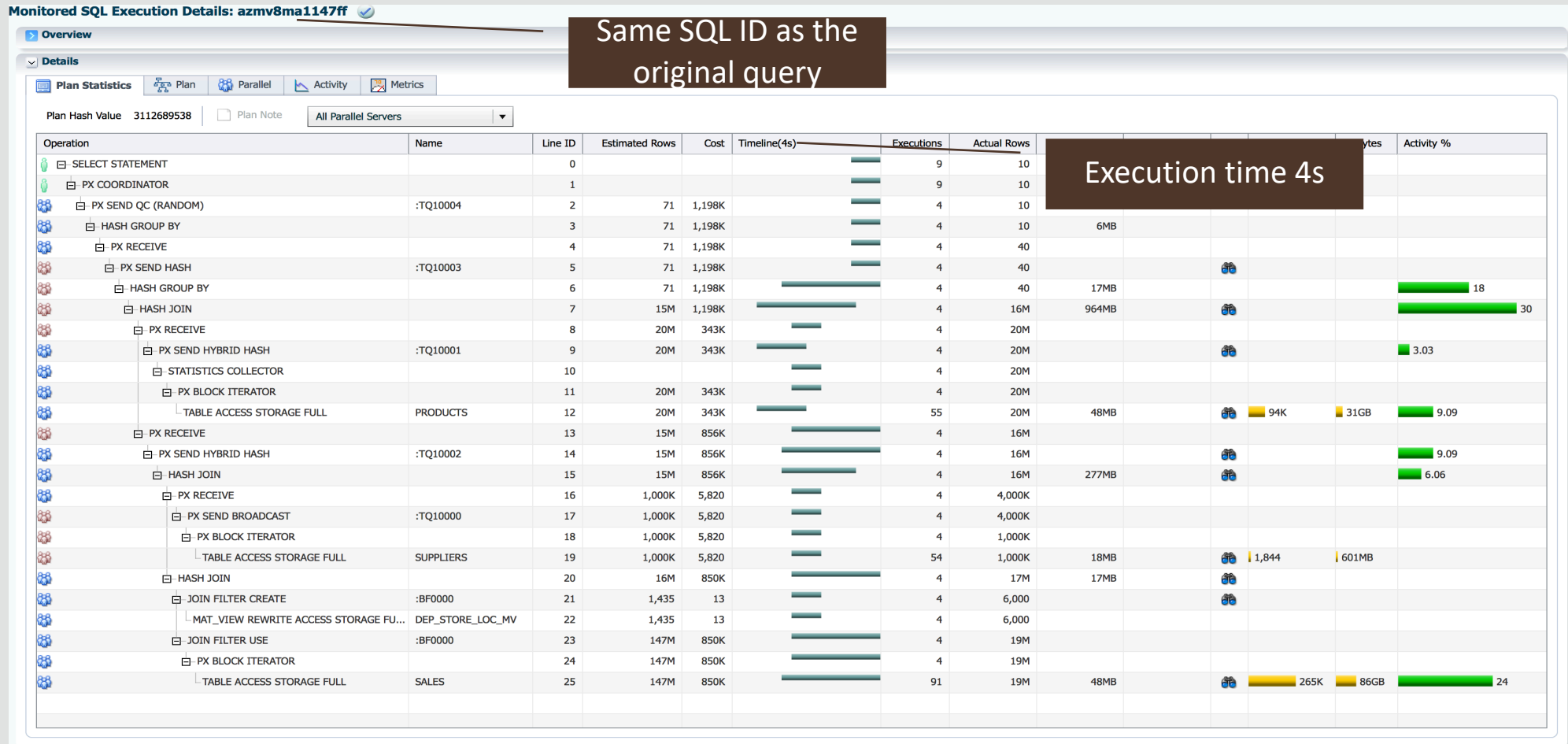
Materialized View

With extended statistics



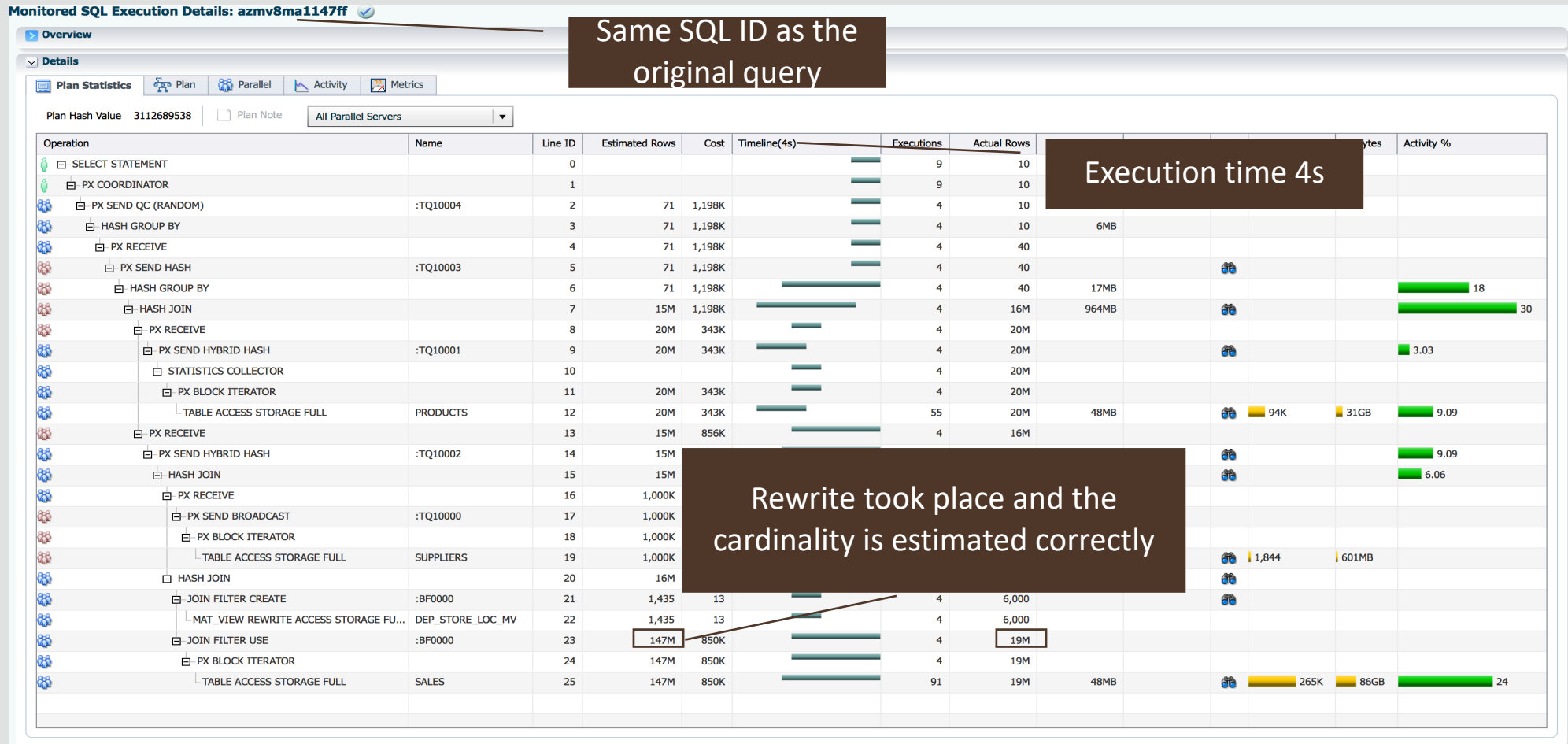
Materialized View

With extended statistics



Materialized View

With extended statistics



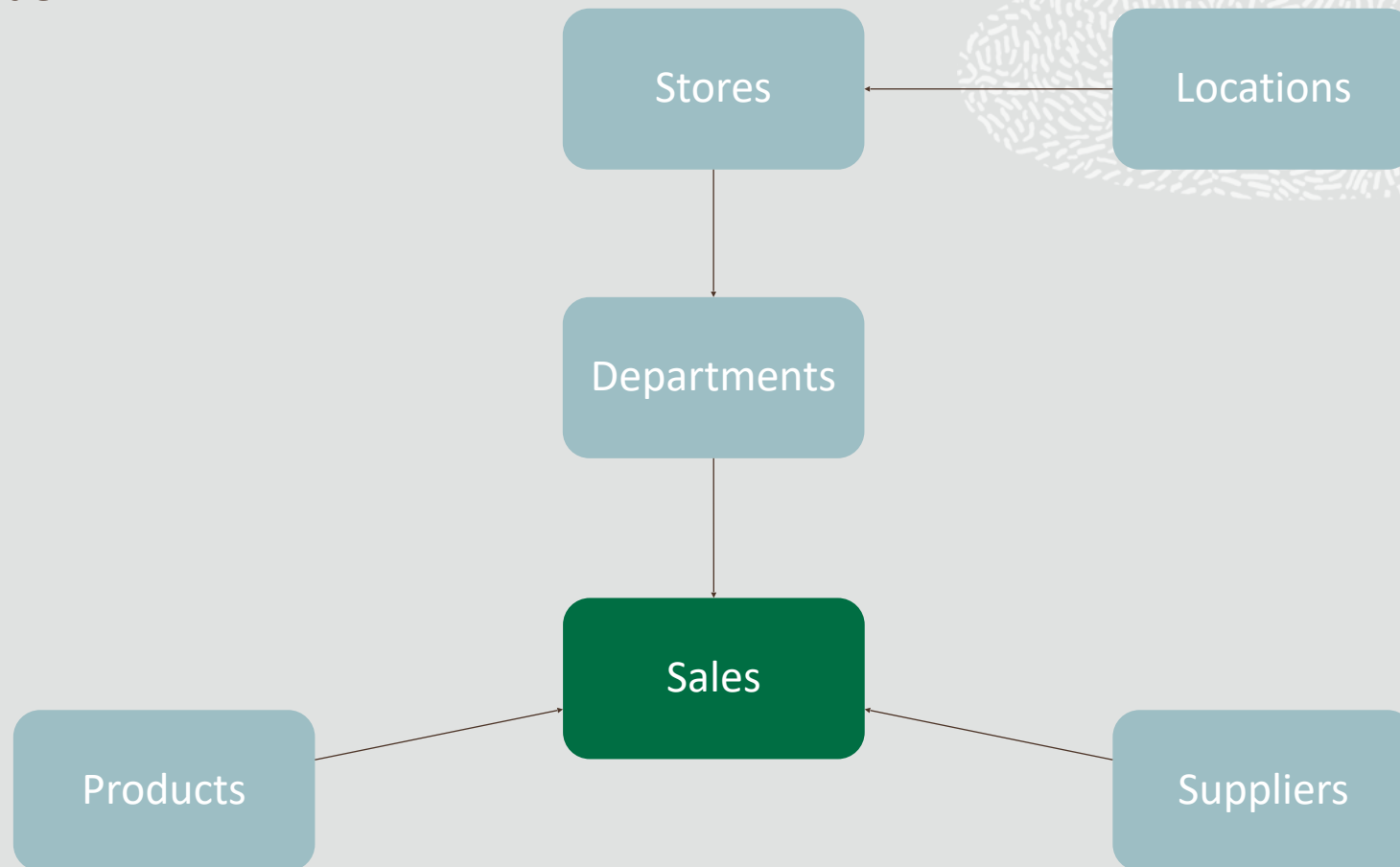
End Demo2



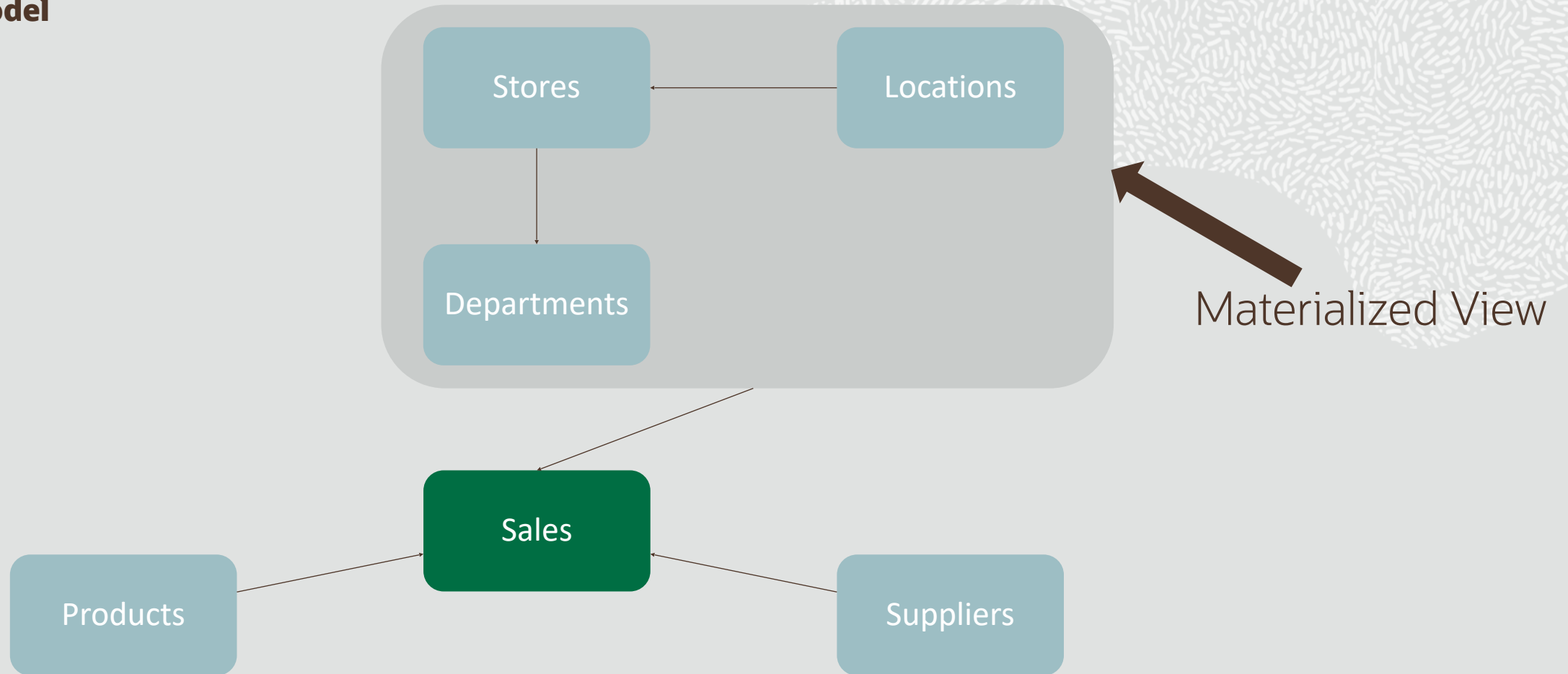
Data Model



Data Model



Data Model



One more thing...

Materialized View; Additional Consideration

- Since the original join cardinality was wrong, its cost is also wrong
- The cost could well be lower than that of the MV-based plan
- May well have to set the following parameter

`QUERY_REWRITE_ENABLED=FORCE`

To Summarize...

Summary: Avoid Guesswork

- When things go wrong, work to find the root cause.
- When you have root cause, you then have a platform to recommend or deliver a robust solution

Summary: Data Model

- Data model is important

The benefits of a good data model cannot be overstated

- You might not always have a perfect dimensional model

Not designed that way

Coercing some other system/model for data warehousing

- A poor data model can lead to sub-optimal execution plans

Wrong distribution methods

Large amounts of TEMP used

etc

Often these plans cannot take advantage of the performance features

Summary: Data Model

- In certain cases, data characteristics mean that statistics (including advanced statistic types) are not sufficient to estimate good join cardinality

Dynamic statistics may or may not help.

- A materialized view provides additional information to the optimizer, resulting in a good execution plan

The performance benefit in this example comes from the better execution plan, not the materializing of the joins

Summary: Benefits of the Materialized View

- No change to the underlying data model
- No change to the code
- The MV's that address this type of problem are typically built on DIMENSIONS

Generally small compared to the FACT table

Slowly changing

- Opens up the possibility of additional optimizations

Eg. Column Groups

- Leverage

Can benefit many different queries

Summary: Materialized View

- Another technique in your Performance Engineering arsenal
- Learn when/where/how to use it

Thank You

Robert Carlin
Mihajlo Tekic

Real-World Performance Team
Oracle Database Development

Safe Harbor

The preceding is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, timing, and pricing of any features or functionality described for Oracle's products may change and remains at the sole discretion of Oracle Corporation.

Statements in this presentation relating to Oracle's future plans, expectations, beliefs, intentions and prospects are "forward-looking statements" and are subject to material risks and uncertainties. A detailed discussion of these factors and other risks that affect our business is contained in Oracle's Securities and Exchange Commission (SEC) filings, including our most recent reports on Form 10-K and Form 10-Q under the heading "Risk Factors." These filings are available on the SEC's website or on Oracle's website at <http://www.oracle.com/investor>. All information in this presentation is current as of September 2019 and Oracle undertakes no duty to update any statement in light of new information or future events.