ORACLE
CloudWorld

Hints and Tips for Fast and Predictable Query Performance with Star Schemas

Session LRN3511

Juergen Mueller
Mihajlo Tekic

October 17–20, 2022
Caesars Forum and The Venetian
Las Vegas, NV

ORACLE
REAL-WORLD PERFORMANCE

Real-World Performance Sessions

Tuesday **11:00-11:45AM**	LRN3511 Hints and Tips for Fast and Predictable Query Performance with Star Schemas
Wednesday **02:30-03:15PM**	LRN1483 Solving Performance Problems Using Automatic Workload Repository
Online	oracle.com/goto/oll/rwp

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 one hundred years of experience combined

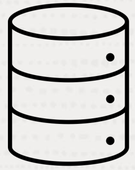
How We Work

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

What We Do

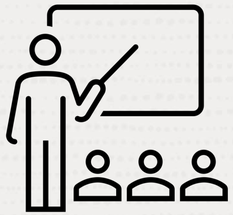
Customer Engagement

- Design Review
- Performance Project



Database Development

- Applications
- Tools



Customer Education

- Online
- Focused

ORACLE
CloudWorld

Hints and Tips for Fast and Predictable Query Performance with Star Schemas

Session LRN3511

Juergen Mueller
Mihajlo Tekic

October 17–20, 2022
Caesars Forum and The Venetian
Las Vegas, NV

ORACLE
REAL-WORLD PERFORMANCE

Agenda

Dimensional Model

Overview

Worked Example

From minutes to seconds

What you must do

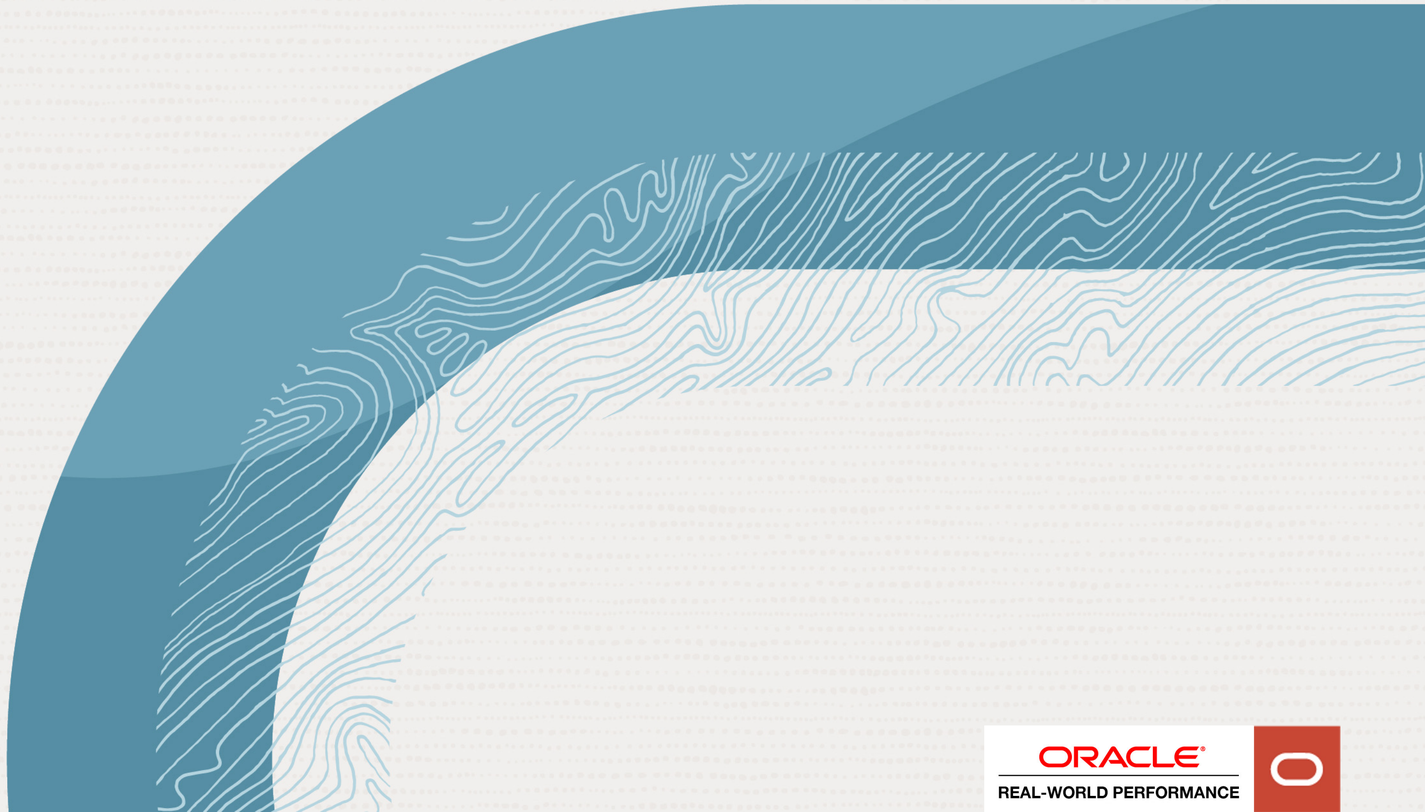
Star Query Execution on ADW

Analysis of our worked example on ADW

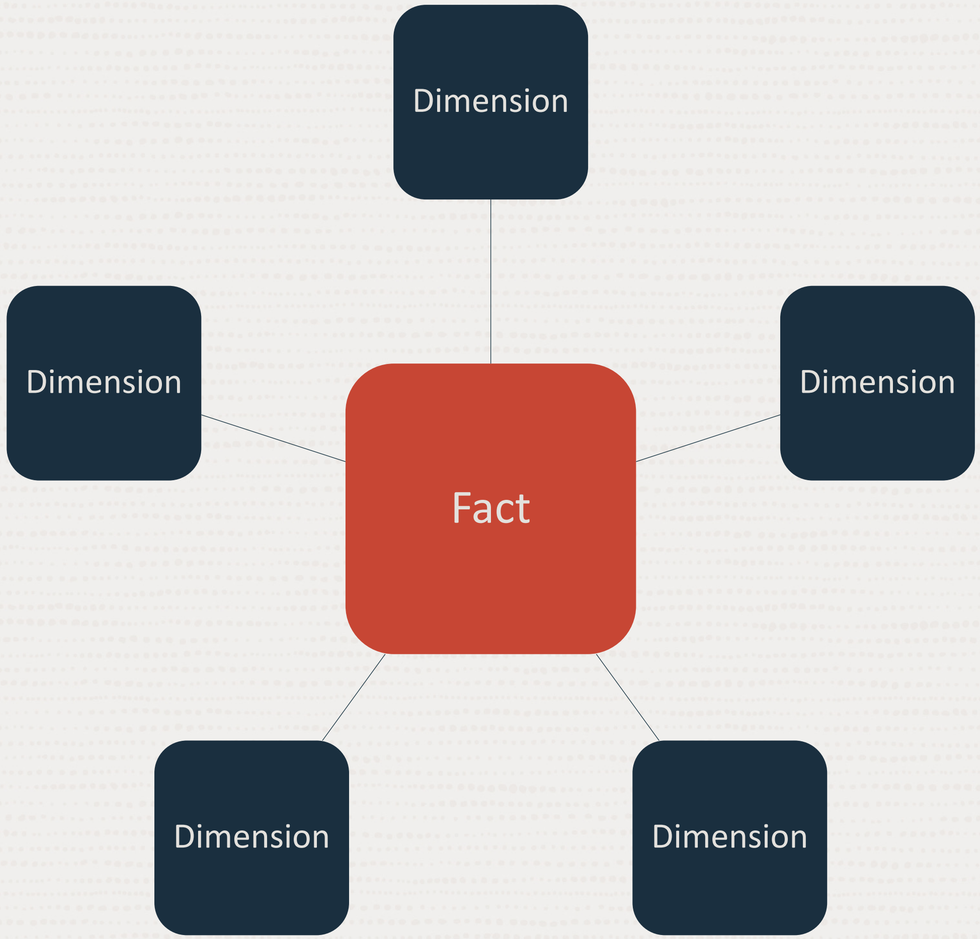
Summary

Dimensional Model

Overview



Why a Dimensional Model?



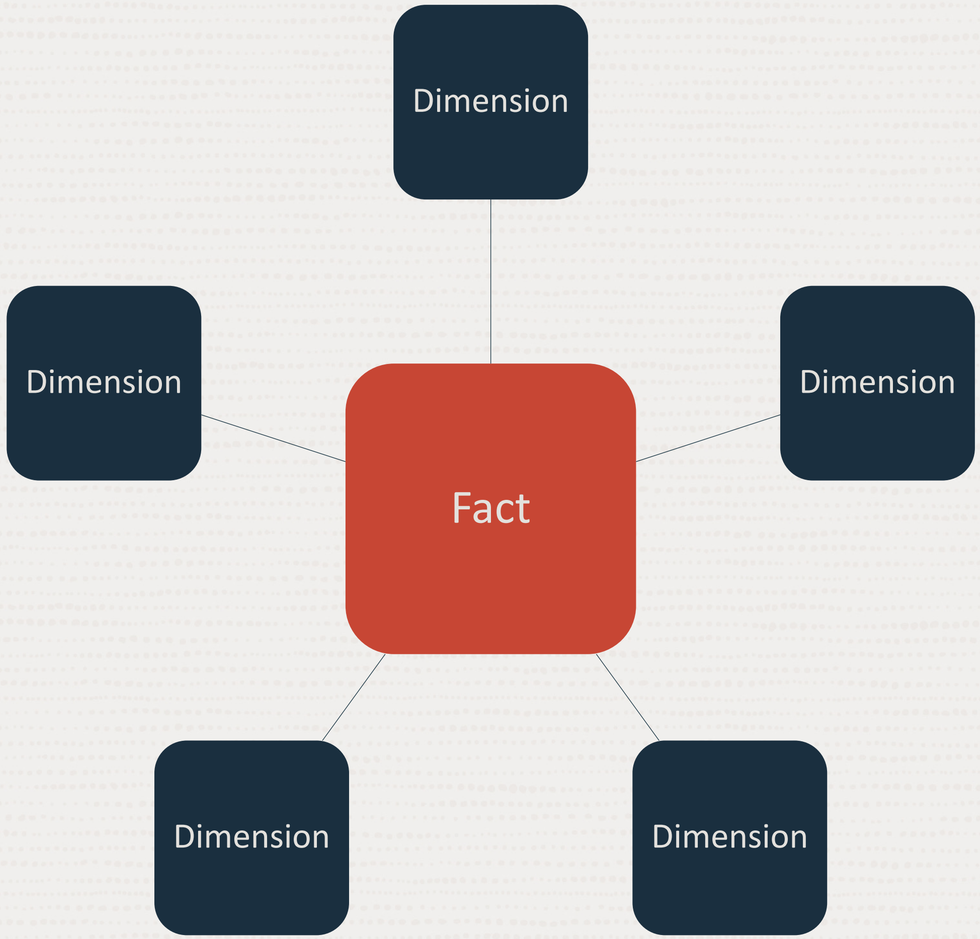
Predictable and standard format

Star join framework withstands unexpected changes in user behavior

Performance

- Well established performance algorithms for this model

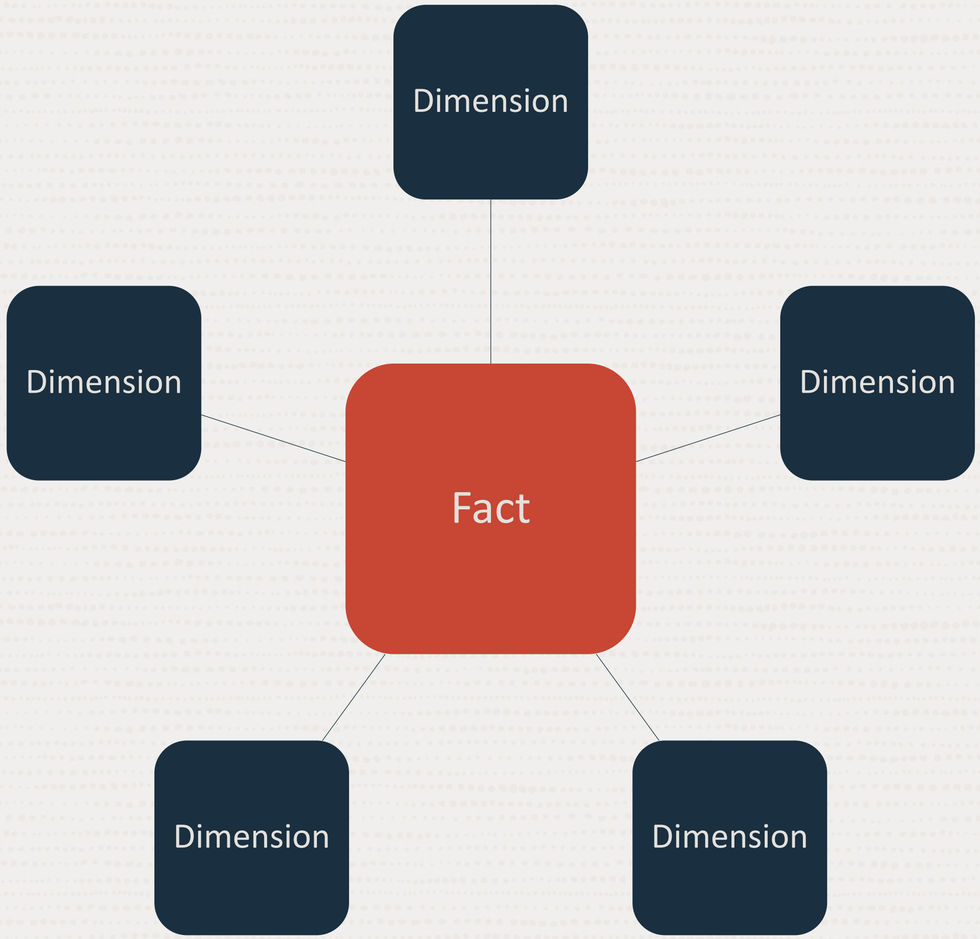
Star Schema Characteristics



Fact tables

- Contain business events called measures
  - Generally numerical values
    - Sales
    - Orders
    - Logins
- Large
- Append only
- Historical in nature

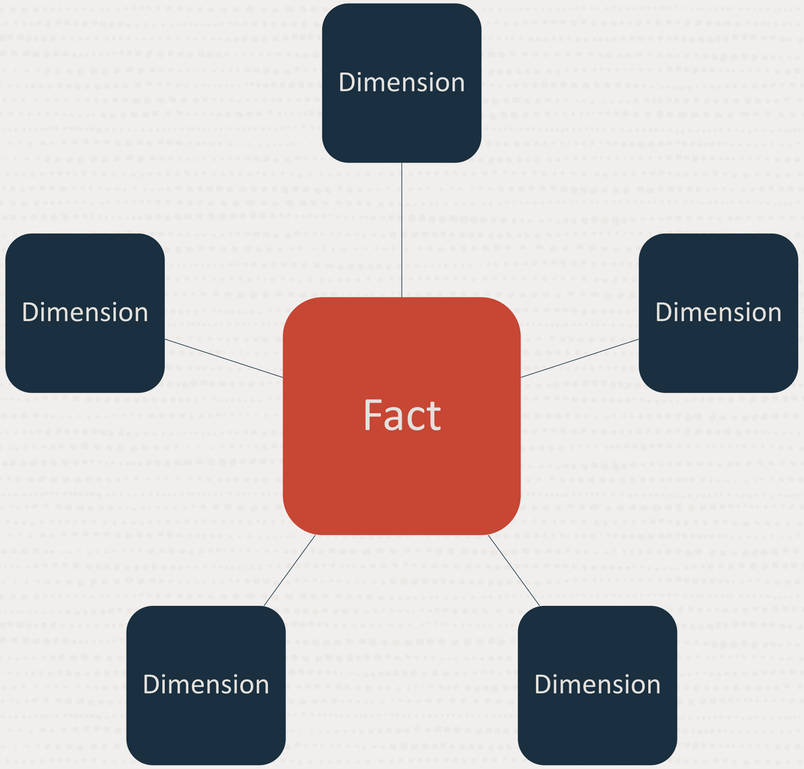
Star Schema Characteristics



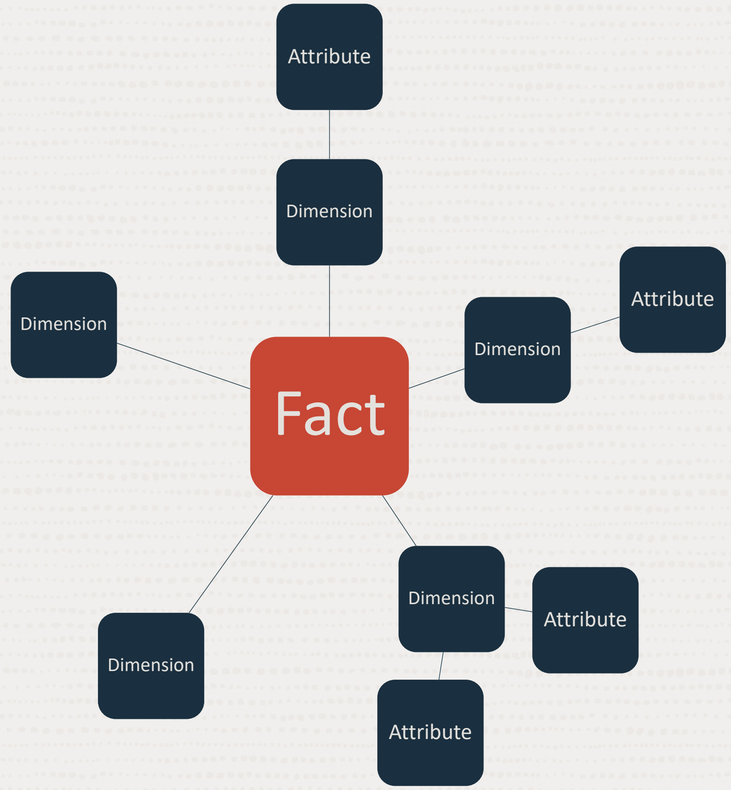
Dimension tables

- Contain reference information about facts called attributes
  - when
  - what
  - where
  - who
  - how
  - why
- Small compared to Fact
- Static or slowly changing

Dimensional Schemas



Star Schema



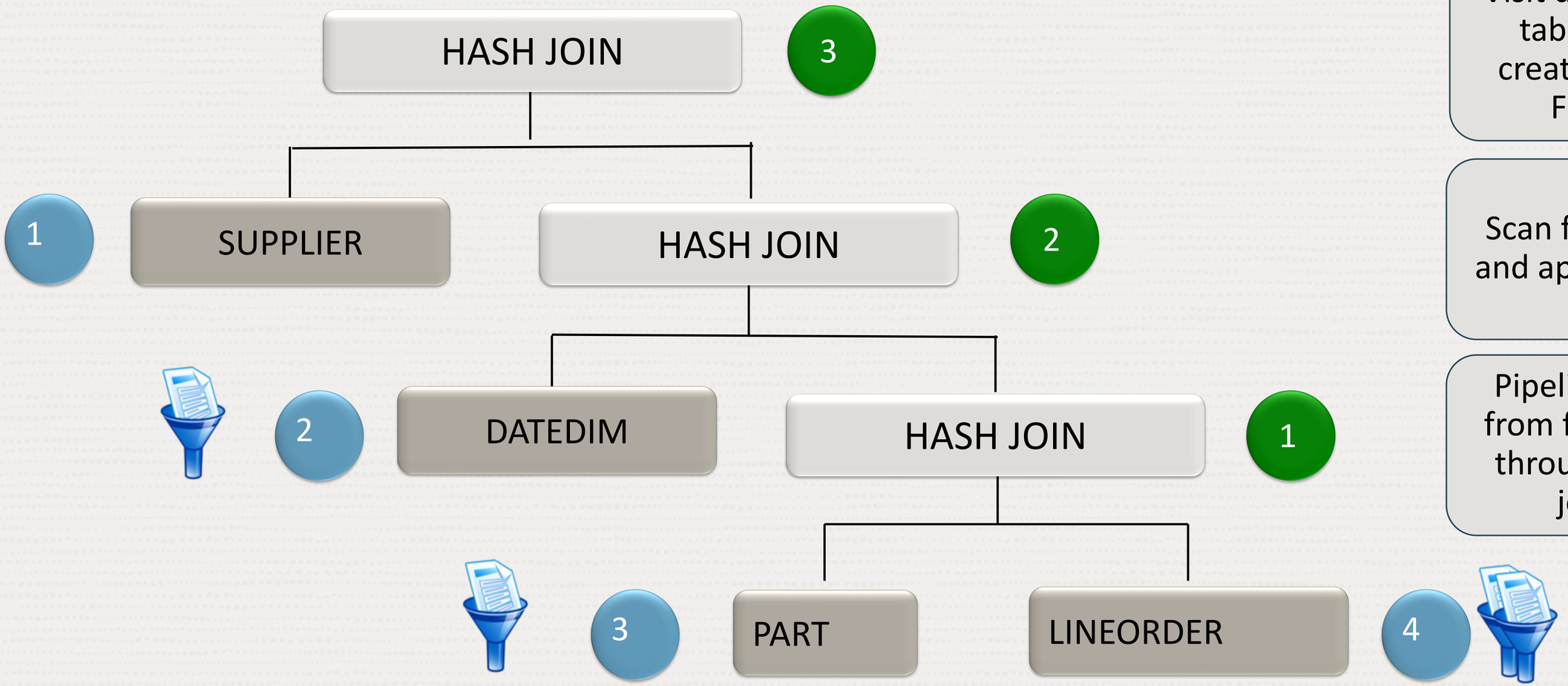
Star Schema derivative

Shape and Structure of a Typical Dimensional Query

```
SELECT d_sellingseason, p_category, s_region,
  SUM(lo_extendedprice)
    FROM    lineorder
                JOIN    customer        ON lo_custkey = c_custkey
                JOIN    date_dim        ON lo_orderdate = d_datekey
                JOIN    part            ON lo_partkey = p_partkey
                JOIN    supplier        ON lo_suppkey = s_suppkey
        WHERE  d_year IN (1993, 1994, 1995)
        AND    p_container in ('JUMBO PACK')
GROUP BY    d_sellingseason, p_category, s_region
ORDER BY    d_sellingseason, p_category, s_region
```

- Choose your **fact** table
- Complete the star by defining relationships with **joins** to dimension tables
- Choose **filter** criteria based upon dimension attributes
- Choose **measures** for aggregation
- Choose **segmentation/roll up** columns
- Choose **grouping** requirements
- Choose **ordering** requirements

Shape of the Execution Plan – Right Deep Tree



Visit dimension tables and create Bloom Filters

Scan fact table and apply filters

Pipeline rows from fact table through hash joins

Shape of the Execution Plan – Right Deep Tree

Operation	Name	Lin...
SELECT STATEMENT		0
SORT GROUP BY		1
HASH JOIN		2
TABLE ACCESS STORAGE FULL	SUPPLIER	3
HASH JOIN		4
JOIN FILTER CREATE	:BF0001	5
PART JOIN FILTER CREATE	:BF0000	6
TABLE ACCESS STORAGE FULL	DATE_DIM	7
HASH JOIN		8
JOIN FILTER CREATE	:BF0002	9
TABLE ACCESS STORAGE FULL	PART	10
JOIN FILTER USE	:BF0001	11
JOIN FILTER USE	:BF0002	12
PARTITION RANGE JOIN-FILTER		13
TABLE ACCESS STORAGE FULL	LINEORDER	14

Visit dimension tables and create Bloom Filters

Scan fact table and apply filters

Pipeline rows from fact table through hash joins

Things You Must Do to Ensure Optimal Execution Plans

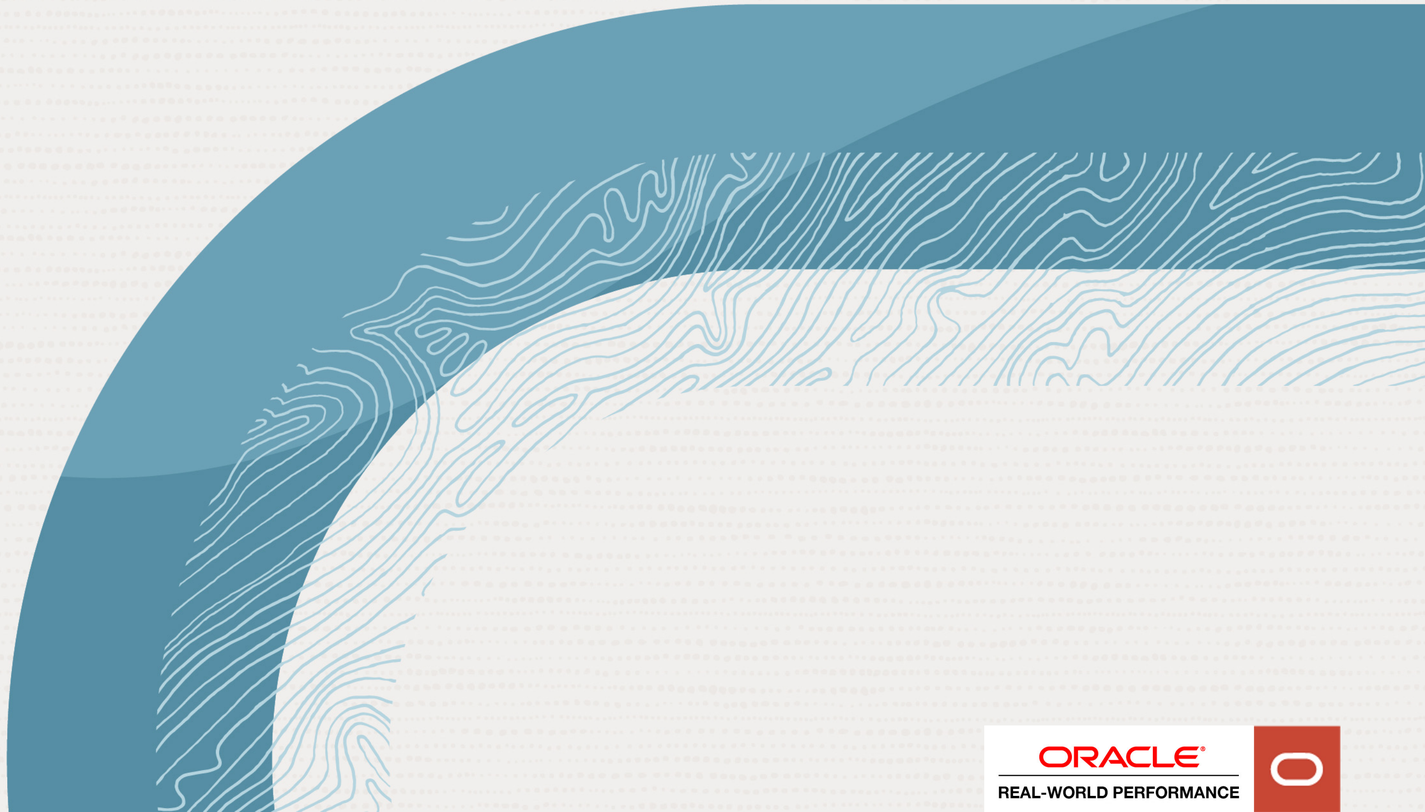
- Statistics
- Constraints
- Partitioning
- Data Types

… all of them are equally important

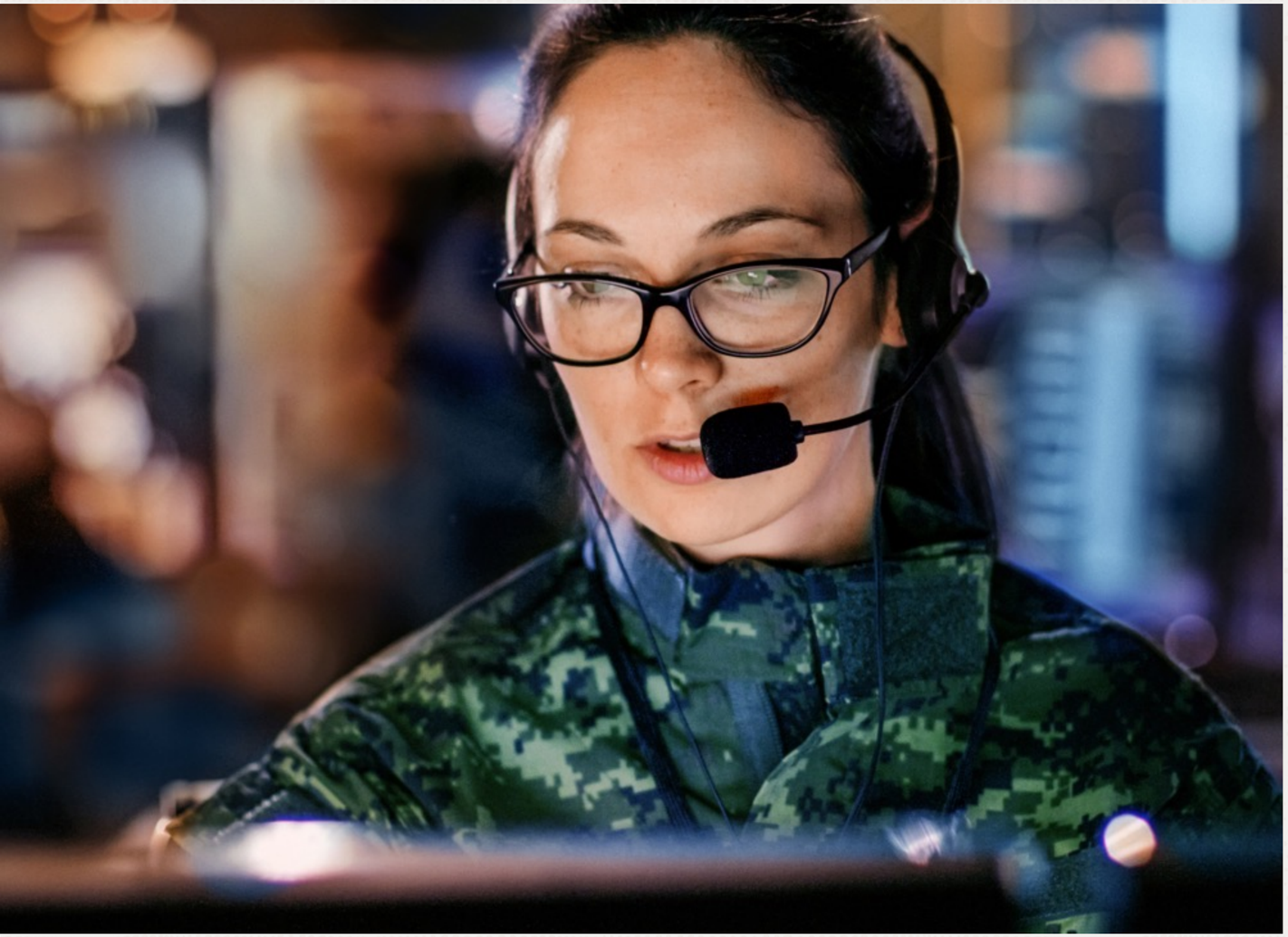
Star Query Execution

Worked example: From minutes to seconds

What you must do?



Scenario



A star query completes in about 5 minutes.

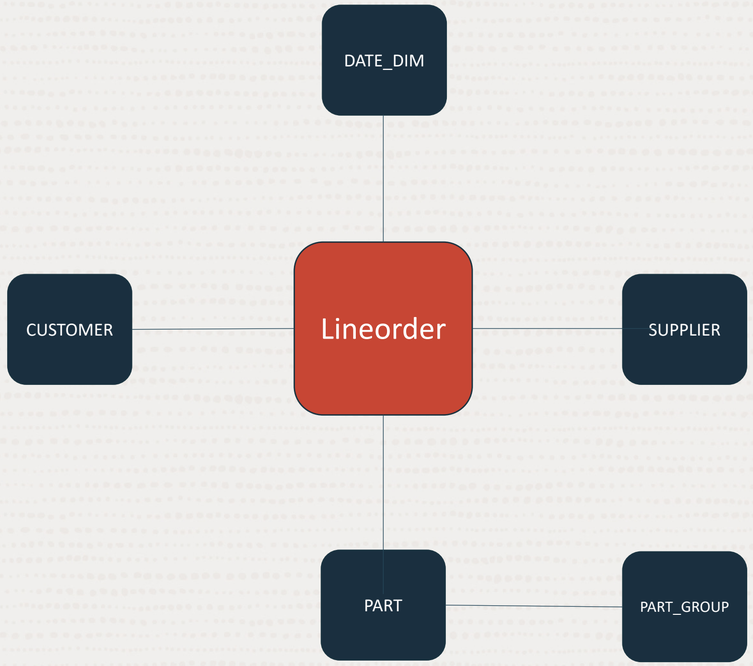
Goal: It needs to complete in 1s or better

Our task:

Optimize the query to complete in 1s or better.
Query modifications are not allowed.

Our Query

```
SELECT  d_month, d_year, s_nation,
        SUM(lo_quantity) lo_quantity,
        SUM(lo_revenue) lo_revenue,
        SUM(lo_supplycost) lo_supplycost
        FROM lineorder
                JOIN customer      ON lo_custkey = c_custkey
                JOIN date_dim      ON lo_orderdate = d_datekey
                JOIN part          ON lo_partkey = p_partkey
                JOIN part_group    ON p_mfgr=pg_mfgr
                                  AND p_category=pg_category
                                  AND p_brand1=pg_brand1
                JOIN supplier      ON lo_suppkey = s_suppkey
        WHERE   d_year = 2007
        AND     s_region = 'ASIA'
        AND     s_nation in ('CHINA','INDIA','JAPAN','VIETNAM')
        AND     pg_type in (1,2,3)
        AND     p_size<12
GROUP BY        d_month, d_year, s_nation
ORDER BY        d_month, d_year, s_nation;
```



Initial performance

Initial running time: 256 seconds

General

Status	Completed
SQL Text	SELECT /*+ monitor */ d_month, d_year, ...
Execution Started	Oct 5, 2022 4:53:25 PM GMT-07:00
Last Refresh Time	Oct 5, 2022 4:57:41 PM GMT-07:00
Execution ID	16777217
User Name	DW01@OCWDW
Fetch Calls	5

Time & Wait

Duration	4.27 m
Database Time	5.07 m
PL/SQL & Java	0 s
Activity	100%

I/O

Buffer Gets	35 M
I/O Requests	4,913 K
I/O Bytes	37.6 GB

Plan Statistics | SQL Text | Activity | Metrics | Optimizer Environment | Outline

Plan Hash Value 1890571844 Plan Size 23 Navigate to

View Option: Tabular Execution Plan | Plan Notes | More Columns

Operation	Object	Informatic	Line ID	Timeline	Executions	Est. Rows	Rows	Activity
SELECT STATEMENT			0		1		48	
SORT GROUP BY (SEL$703A7920)			1		1	13	48	
NESTED LOOPS			2		1	13	173 K	
NESTED LOOPS			3		1	13	173 K	
HASH JOIN			4		1	13	1,085 K	1.60%
JOIN FILTER CREATE	:BF0000		5		1	90	7,157 K	
NESTED LOOPS			6		1	90	7,157 K	0.80%
NESTED LOOPS			7		1	90	7,157 K	
VIEW (SEL$7AE8FF08)	VW_GBF_40		8		1	1	106 K	
HASH GROUP BY (SEL$7AE8FF08)			9		1	1	106 K	
HASH JOIN			10		1	1	106 K	
JOIN FILTER CREATE	:BF0001		11		1	1	211 K	
TABLE ACCESS STORAGE FULL (SEL$7AE8FF	PART		12		1	1	211 K	0.40%
JOIN FILTER USE	:BF0001		13		1	600	600	
TABLE ACCESS STORAGE FULL (SEL$7AE8FF	PART_GROUP		14		1	600	600	
PARTITION RANGE ALL			15		106 K	90	7,157 K	0.80%
INDEX RANGE SCAN (SEL$703A7920)	LINEORDER1_PART_N		16		8,692 K	90	7,157 K	48.40%
TABLE ACCESS BY LOCAL INDEX ROWID (SEL$703A	LINEORDER		17		12 M	90	7,157 K	47.20%
JOIN FILTER USE	:BF0000		18		1	365	365	
TABLE ACCESS STORAGE FULL (SEL$703A7920)	DATE_DIM		19		1	365	365	
TABLE ACCESS BY INDEX ROWID BATCHED (SEL$703A7	SUPPLIER		20		1,085 K	1	173 K	
INDEX RANGE SCAN (SEL$703A7920)	SUPPLIER_PK		21		1,085 K	1	1,085 K	0.80%
INDEX RANGE SCAN (SEL$703A7920)	CUSTOMER_PK		22		174 K	1	173 K	

ORACLE REAL-WORLD PERFORMANCE

Worked example: Progress

	Fix Category	Running Time
Initial Performance		256 seconds

Initial performance

General

Status	Completed
SQL Text	SELECT /*+ monitor */ d_month, d_year, ...
Execution Started	Oct 5, 2022 4:53:25 PM GMT-07:00
Last Refresh Time	Oct 5, 2022 4:57:41 PM GMT-07:00
Execution ID	16777217
User Name	DW01@OCWDW
Fetch Calls	5

Time & Wait

Duration	4.27 m
Database Time	5.07 m
PL/SQL & Java	0 s
Activity	100%

I/O

Buffer Gets	35 M
I/O Requests	4,913 K
I/O Bytes	37.6 GB

Most of the database time spent on accessing (through an index) data in LINEORDER table.

NL Join between the results from the join between PART and PART_GROUP and LINEORDER

Plan Statistics | SQL Text | Activity | Metrics | Optimizer Environment | Outline

Plan Hash Value 1890571844 Plan Size 23 Navigate to View Option Tabular Execution Plan Plan Notes

Operation	Object		Executions	Est. Rows	Rows	Activity
SELECT STATEMENT		1	1		48	
SORT GROUP BY (SEL$703A7920)			1	13	48	
NESTED LOOPS		2	1	13	173 K	
NESTED LOOPS		3	1	13	173 K	
HASH JOIN		4	1	13	1,085 K	1.60%
JOIN FILTER CREATE	:BF0000	5	1	90	7,157 K	
NESTED LOOPS		6	1	90	7,157 K	0.80%
NESTED LOOPS		7	1	90	7,157 K	
VIEW (SEL$7AE8FF08)	VW_GBF_40	8	1	1	106 K	
HASH GROUP BY (SEL$7AE8FF08)		9	1	1	106 K	
HASH JOIN		10	1	1	106 K	
JOIN FILTER CREATE	:BF0001	11	1	1	211 K	
TABLE ACCESS STORAGE FULL (SEL$7AE8FF	PART	12	1	1	211 K	0.40%
JOIN FILTER USE	:BF0001	13	1	600	600	
TABLE ACCESS STORAGE FULL (SEL$7AE8FF	PART_GROUP	14	1	600	600	
PARTITION RANGE ALL		15	106 K	90	7,157 K	0.80%
INDEX RANGE SCAN (SEL$703A7920)	LINEORDER1_PART_N	16	8,692 K	90	7,157 K	48.40%
TABLE ACCESS BY LOCAL INDEX ROWID (SEL$703A	LINEORDER	17	12 M	90	7,157 K	47.20%
JOIN FILTER USE	:BF0000	18	1	365	365	
TABLE ACCESS STORAGE FULL (SEL$703A7920)	DATE_DIM	19	1	365	365	
TABLE ACCESS BY INDEX ROWID BATCHED (SEL$703A7	SUPPLIER	20	1,085 K	1	173 K	
INDEX RANGE SCAN (SEL$703A7920)	SUPPLIER_PK	21	1,085 K	1	1,085 K	0.80%
INDEX RANGE SCAN (SEL$703A7920)	CUSTOMER_PK	22	174 K	1	173 K	

Initial performance

General

Status	Completed
SQL Text	SELECT /*+ monitor */ d_month, d_year, ...
Execution Started	Oct 5, 2022 4:53:25 PM GMT-07:00
Last Refresh Time	Oct 5, 2022 4:57:41 PM GMT-07:00
Execution ID	16777217
User Name	DW01@OCWDW
Fetch Calls	5

Time & Wait

Duration	4.27 m
Database Time	5.07 m
PL/SQL & Java	0 s
Activity	100%

I/O

Buffer Gets	35 M
I/O Requests	4,913 K
I/O Bytes	37.6 GB

Most of the database time spent on accessing (through an index) data in LINEORDER table.

Plan Statistics | SQL Text | Activity | Metrics | Optimizer Environment | Outline

Plan Hash Value 1890571844 Plan Size 23 Navigate to View Option Tabular Execution Plan Plan Notes

Operation	Object						Activity
SELECT STATEMENT						48	
SORT GROUP BY (SEL$703A7920)						48	
NESTED LOOPS			2			173 K	
NESTED LOOPS			3			173 K	
HASH JOIN			4			1,085 K	1.60%
JOIN FILTER CREATE	:BF0000				1	7,157 K	
NESTED LOOPS					1	7,157 K	0.80%
NESTED LOOPS					1	7,157 K	
VIEW (SEL$7AE8FF08)	VW_GBF_40				1	106 K	
HASH GROUP BY (SEL$7AE8FF08)					1	106 K	
HASH JOIN					1	106 K	
JOIN FILTER CREATE	:BF0001				1	211 K	
TABLE ACCESS STORAGE FULL (SEL$7AE8FF	PART				1	211 K	0.40%
JOIN FILTER USE	:BF0001				1	600	600
TABLE ACCESS STORAGE FULL (SEL$7AE8FF	PART_GROUP				1	600	600
PARTITION RANGE ALL				106 K	90	7,157 K	0.80%
INDEX RANGE SCAN (SEL$703A7920)	LINEORDER1_PART_N			692 K	90	7,157 K	48.40%
TABLE ACCESS BY LOCAL INDEX ROWID (SEL$703A	LINEORDER			12 M	90	7,157 K	47.20%
JOIN FILTER USE	:BF0000				1	365	365
TABLE ACCESS STORAGE FULL (SEL$703A7920)	DATE_DIM				1	365	365
TABLE ACCESS BY INDEX ROWID BATCHED (SEL$703A7	SUPPLIER			085 K	1	173 K	
INDEX RANGE SCAN (SEL$703A7920)	SUPPLIER_PK			085 K	1	1,085 K	0.80%
INDEX RANGE SCAN (SEL$703A7920)	CUSTOMER_PK			174 K	1	173 K	

NL Join between the results from the join between PART and PART_GROUP and LINEORDER

The optimizer estimated 1 row to be produced after filtering the data from PART table. The actual number of rows is 211K

Because the optimizer estimated relatively small number of rows will be produced by row source 8, it chose NL join as a join method to join LINEORDER table.

The driving side produces 106K rows (estimated 1) hence row source 15 is executed 106K times.

The first big cardinality underestimate is the estimate for table PART

If the optimizer estimates correct cardinality for PART, will the join method be more efficient?

LET'S SEE …

What You Must Do: Statistics

Think about skew

Think about correlation

Do not rely on Dynamic Statistics alone

Think about how and when to gather statistics

Statistics must be representative

Step 1: Fix the cardinality estimate for PART

Fix the cardinality estimate for table PART (estimate 1, actual 211K)

- When addressing a problem with single table cardinality estimates, the first place to look is the state of the table statistics. Are they representative?

```
SELECT table_name, num_rows FROM user_tables WHERE table_name='PART';

TABLE_NAME           NUM_ROWS
-------------------- ----------
PART                          1
```

Statistics for table PART indicate there is 1 row in the table

```
SELECT COUNT(*) FROM part;

  COUNT(*)
----------
    959616
```

The actual number of rows in table PART is 959,616

```
EXEC dbms_stats.gather_table_stats(ownname => user,tabname => 'PART');
```

Gather statistics for PART table

Step 1: Statistics on PART

Step1: running time: 211 seconds

General

Status	Completed
SQL Text	SELECT /*+ monitor */ d_month, d_year, ...
Execution Started	Oct 5, 2022 4:59:22 PM GMT-07:00
Last Refresh Time	Oct 5, 2022 5:02:53 PM GMT-07:00
Execution ID	16777216
User Name	DW01@OCWDW
Fetch Calls	5

Time & Wait

Duration	3.52 m
Database Time	4.56 m
PL/SQL & Java	0 s
Activity	100%

I/O

Buffer Gets	9,463 K
I/O Requests	4,415 K
I/O Bytes	33.7 GB

Plan Statistics | SQL Text | Activity | Metrics | Optimizer Environment | Outline

Plan Hash Value 667865441 Plan Size 19 Navigate to View Option Tabular Execution Plan Plan Notes More Columns

Operation	Object	Informatic	Line ID	Timeline	Executions	Est. Rows	Rows	Activity
SELECT STATEMENT			0		1		48	
SORT GROUP BY (SEL$703A7920)			1		1		48	
NESTED LOOPS			2		1		173 K	
HASH JOIN			3		1	264	173 K	
TABLE ACCESS STORAGE FULL (SEL$703A7920)	DATE_DIM		4		1	365	365	
HASH JOIN			5		1	1,849	1,136 K	0.97%
VIEW (SEL$7AE8FF08)	VW_GBF_40		6		1	644	106 K	
HASH GROUP BY (SEL$7AE8FF08)			7		1	644	106 K	
HASH JOIN			8		1	644	106 K	
JOIN FILTER CREATE	:BF0000		9		1	600	600	
TABLE ACCESS STORAGE FULL (SEL$7AE8FF08)	PART_GROUP		10		1	600	600	
JOIN FILTER USE	:BF0000		11		1	211 K	107 K	
TABLE ACCESS STORAGE FULL (SEL$7AE8FF08)	PART		12		1	211 K	107 K	
NESTED LOOPS			13		1	1,725 K	8,581 K	
TABLE ACCESS STORAGE FULL (SEL$703A7920)	SUPPLIER		14		1	575	2,862	
PARTITION RANGE ALL			15		2,862	2,999	8,581 K	0.49%
TABLE ACCESS BY LOCAL INDEX ROWID BATCHED (	LINEORDER		16		235 K	2,999	8,581 K	74.27%
INDEX RANGE SCAN (SEL$703A7920)	LINEORDER1_SUPP_FBI_N		17		235 K	32	8,581 K	24.27%
INDEX RANGE SCAN (SEL$703A7920)	CUSTOMER_PK		18		173 K	1	173 K	

The estimated cardinality for PART is correct

Worked example: Progress

	Fix Category	Running Time
Initial Performance		256 seconds
Step 1: Statistics on PART	Statistics	211 seconds

Step 1: Statistics on PART

General

Status	Completed
SQL Text	SELECT /*+ monitor */ d_month, d_year, ...
Execution Started	Oct 5, 2022 4:59:22 PM GMT-07:00
Last Refresh Time	Oct 5, 2022 5:02:53 PM GMT-07:00
Execution ID	16777216
User Name	DW01@OCWDW
Fetch Calls	5

Time & Wait

Duration	3.52 m
Database Time	4.56 m
PL/SQL & Java	0 s
Activity	100%

I/O

Buffer Gets	9,463 K
I/O Requests	4,415 K
I/O Bytes	33.7

Most of the database time spent on accessing (through LINEORDER1_SUPP_FBI_N index) data in LINEORDER table.

Plan Statistics | SQL Text | Activity | Metrics | Optimizer Environment | Outline

Plan Hash Value 667865441 Plan Size 19 Navigate to View Option Tabular Execution Plan Plan Notes

Operation	Object	Informatic	Line ID	Timeline	Executions	Est. Rows	Rows	Activity
SELECT STATEMENT			0		1		48	
SORT GROUP BY (SEL$703A7920)			1		1	34	48	
NESTED LOOPS			2		1	264	173 K	
HASH JOIN			3		1	264	173 K	
TABLE ACCESS STORAGE FULL (SEL$703A7920)	DATE_DIM		4		1	365	365	
HASH JOIN			5		1	1,849	1,136 K	0.97%
VIEW (SEL$7AE8FF08)	VW_GBF_40		6		1	644	106 K	
HASH GROUP BY (SEL$7AE8FF08)			7		1	644	106 K	
HASH JOIN			8		1	644	106 K	
JOIN FILTER CREATE	:BF0000		9		1	600	600	
TABLE ACCESS STORAGE FULL (SEL$7AE8FF08)	PART_GROUP		10		1	600	600	
JOIN FILTER USE	:BF0000		11		1	211 K	107 K	
TABLE ACCESS STORAGE FULL (SEL$7AE8FF08)	PART		12		1	211 K	107 K	
NESTED LOOPS			13		1	1,725 K	8,581 K	
TABLE ACCESS STORAGE FULL (SEL$703A7920)	SUPPLIER		14		1	575	2,862	
PARTITION RANGE ALL			15		2,862	2,999	8,581 K	0.49%
TABLE ACCESS BY LOCAL INDEX ROWID BATCHED (	LINEORDER		16		235 K	2,999	8,581 K	74.27%
INDEX RANGE SCAN (SEL$703A7920)	LINEORDER1_SUPP_FBI_N		17		235 K	32	8,581 K	24.27%
INDEX RANGE SCAN (SEL$703A7920)	CUSTOMER_PK		18		173 K	1	173 K	

Step 1: Statistics on PART

General

Status	Completed
SQL Text	SELECT /*+ monitor */ d_month, d_year, ...
Execution Started	Oct 5, 2022 4:59:22 PM GMT-07:00
Last Refresh Time	Oct 5, 2022 5:02:53 PM GMT-07:00
Execution ID	16777216
User Name	DW01@OCWDW
Fetch Calls	5

Time & Wait

Duration	3.52 m
Database Time	4.56 m
PL/SQL & Java	0 s
Activity	100%

I/O

Buffer Gets	9,463 K
I/O Requests	4,415 K
I/O Bytes	33.7

Most of the database time spent on accessing (through LINEORDER1_SUPP_FBI_N index) data in LINEORDER table.

Plan Statistics | SQL Text | Activity | Metrics | Optimizer Environment

NL Join between SUPPLIER and LINEORDER

Plan Hash Value 667865441 Plan Size 19 Navigate to

View Option: Tabular Execution Plan Plan Notes

Operation	Object		Executions	Est. Rows	Rows	Activity
SELECT STATEMENT		1	1		48	
SORT GROUP BY (SEL$703A7920)		1	1	34	48	
NESTED LOOPS		2	1	264	173 K	
HASH JOIN		3	1	264	173 K	
TABLE ACCESS STORAGE FULL (SEL$703A7920)	DATE_DIM	4	1	365	365	
HASH JOIN		5	1	1,849	1,136 K	0.97%
VIEW (SEL$7AE8FF08)	VW_GBF_40	6	1	644	106 K	
HASH GROUP BY (SEL$7AE8FF08)		7	1	644	106 K	
HASH JOIN		8	1	644	106 K	
JOIN FILTER CREATE	:BF0000	9	1	600	600	
TABLE ACCESS STORAGE FULL (SEL$7AE8FF08)	PART_GROUP	10	1	600	600	
JOIN FILTER USE	:BF0000	11	1	211 K	107 K	
TABLE ACCESS STORAGE FULL (SEL$7AE8FF08)	PART	12	1	211 K	107 K	
NESTED LOOPS		13	1	1,725 K	8,581 K	
TABLE ACCESS STORAGE FULL (SEL$703A7920)	SUPPLIER	14	1	575	2,862	
PARTITION RANGE ALL		15	2,862	2,999	8,581 K	0.49%
TABLE ACCESS BY LOCAL INDEX ROWID BATCHED (	LINEORDER	16	235 K	2,999	8,581 K	74.27%
INDEX RANGE SCAN (SEL$703A7920)	LINEORDER1_SUPP_FBI_N	17	235 K	32	8,581 K	24.27%
INDEX RANGE SCAN (SEL$703A7920)	CUSTOMER_PK	18	173 K	1	173 K	

Step 1: Statistics on PART

General

Status	Completed
SQL Text	SELECT /*+ monitor */ d_month, d_year, ...
Execution Started	Oct 5, 2022 4:59:22 PM GMT-07:00
Last Refresh Time	Oct 5, 2022 5:02:53 PM GMT-07:00
Execution ID	16777216
User Name	DW01@OCWDW
Fetch Calls	5

Time & Wait

Duration	3.52 m
Database Time	4.56 m
PL/SQL & Java	0 s
Activity	100%

I/O

Buffer Gets	9,463 K
I/O Requests	4,415 K
I/O Bytes	33.7

Most of the database time spent on accessing (through LINEORDER1_SUPP_FBI_N index) data in LINEORDER table.

NL Join between SUPPLIER and LINEORDER

The estimated cardinality for SUPPLIER is 575 which is low enough for the optimizer to choose NL join. The actual is 2,862.

Plan Statistics | SQL Text | Activity | Metrics | Optimizer Environment

Plan Hash Value 667865441 Plan Size 19 Navigate to

View Option: Tabular Execution Plan Plan Notes

Operation	Object							Activity
SELECT STATEMENT							48	
SORT GROUP BY (SEL$703A7920)			1				48	
NESTED LOOPS			2				173 K	
HASH JOIN			3			264	173 K	
TABLE ACCESS STORAGE FULL (SEL$703A7920)	DATE_DIM		4			365	365	
HASH JOIN			5			1,849	1,136 K	0.97%
VIEW (SEL$7AE8FF08)	VW_GBF_40		6			644	106 K	
HASH GROUP BY (SEL$7AE8FF08)			7			644	106 K	
HASH JOIN			8			644	106 K	
JOIN FILTER CREATE	:BF0000		9			600	600	
TABLE ACCESS STORAGE FULL (SEL$7AE8FF08)	PART_GROUP		10			600	600	
JOIN FILTER USE	:BF0000		11		1	211 K	107 K	
TABLE ACCESS STORAGE FULL (SEL$7AE8FF08)	PART		12		1	211 K	107 K	
NESTED LOOPS			13		1	1,725 K	8,581 K	
TABLE ACCESS STORAGE FULL (SEL$703A7920)	SUPPLIER		14		1	575	2,862	
PARTITION RANGE ALL			15		2,862	2,999	8,581 K	0.49%
TABLE ACCESS BY LOCAL INDEX ROWID BATCHED (	LINEORDER		16		235 K	2,999	8,581 K	74.27%
INDEX RANGE SCAN (SEL$703A7920)	LINEORDER1_SUPP_FBI_N		17		235 K	32	8,581 K	24.27%
INDEX RANGE SCAN (SEL$703A7920)	CUSTOMER_PK		18		173 K	1	173 K	

Step 1: Statistics on PART

General

Status	Completed
SQL Text	SELECT /*+ monitor */ d_month, d_year, ...
Execution Started	Oct 5, 2022 4:59:22 PM GMT-07:00
Last Refresh Time	Oct 5, 2022 5:02:53 PM GMT-07:00
Execution ID	16777216
User Name	DW01@OCWDW
Fetch Calls	5

Time & Wait

Duration	3.52 m
Database Time	4.56 m
PL/SQL & Java	0 s
Activity	100%

I/O

Buffer Gets	9,463 K
I/O Requests	4,415 K
I/O Bytes	33.7

Most of the database time spent on accessing (through LINEORDER1_SUPP_FBI_N index) data in LINEORDER table.

Plan Statistics | SQL Text | Activity | Metrics | Optimizer Environment

Plan Hash Value 667865441 Plan Size 19 Navigate to

View Option: Tabular Execution Plan Plan Notes

NL Join between SUPPLIER and LINEORDER

The estimated cardinality for SUPPLIER is 575 which is low enough for the optimizer to choose NL join. The actual is 2,862.

Operation	Object							Activity
SELECT STATEMENT							48	
SORT GROUP BY (SEL$703A7920)			1				48	
NESTED LOOPS			2				173 K	
HASH JOIN			3			264	173 K	
TABLE ACCESS STORAGE FULL (SEL$703A7920)	DATE_DIM		4			365	365	
HASH JOIN			5			1,849	1,136 K	0.97%
VIEW (SEL$7AE8FF08)	VW_GBF_40		6			644	106 K	
HASH GROUP BY (SEL$7AE8FF08)			7			644	106 K	
HASH JOIN			8			644	106 K	
JOIN FILTER CREATE	:BF0000		9			600	600	
TABLE ACCESS STORAGE FULL (SEL$7AE8FF08)	PART_GROUP		10			600	600	
JOIN FILTER USE	:BF0000		11		1	211 K	107 K	
TABLE ACCESS STORAGE FULL (SEL$7AE8FF08)	PART		12		1	211 K	107 K	
NESTED LOOPS			13		1	1,725 K	8,581 K	
TABLE ACCESS STORAGE FULL (SEL$703A7920)	SUPPLIER		14		1	575	2,862	
PARTITION RANGE ALL			15		2,862	2,999	8,581 K	0.49%
TABLE ACCESS BY LOCAL INDEX ROWID BATCHED (	LINEORDER		16		235 K	2,999	8,581 K	74.27%
INDEX RANGE SCAN (SEL$703A7920)	LINEORDER1_SUPP_FBI_N		17		235 K	32	8,581 K	24.27%
INDEX RANGE SCAN (SEL$703A7920)	CUSTOMER_PK		18		173 K	1	173 K	

Step 1: Statistics on PART

General

Status	Completed
SQL Text	SELECT /*+ monitor */ d_month, d_year, ...
Execution Started	Oct 5, 2022 4:59:22 PM GMT-07:00
Last Refresh Time	Oct 5, 2022 5:02:53 PM GMT-07:00
Execution ID	16777216
User Name	DW01@OCWDW
Fetch Calls	5

Time & Wait

Duration	3.52 m
Database Time	4.56 m
PL/SQL & Java	0 s
Activity	100%

I/O

Buffer Gets	9,463 K
I/O Requests	4,415 K
	33.7 GB

Plan Statistics | SQL Text | Activity | Metrics | Optimizer Environment | Outline

Plan Hash Value 667865441 Plan Size 19 Navigate to

Operation	Object	Informatic	Line ID	Timeline	Executions			
SELECT STATEMENT			0					
SORT GROUP BY (SEL$703A7920)			1					
NESTED LOOPS			2		1			
HASH JOIN			3		1	264	173 K	
TABLE ACCESS STORAGE FULL (SEL$703A7920)	DATE_DIM		4		1	365	365	
HASH JOIN			5		1	1,849	1,136 K	0.97%
VIEW (SEL$7AE8FF08)	VW_GBF_40		6		1	644	106 K	
HASH GROUP BY (SEL$7AE8FF08)			7		1	644	106 K	
HASH JOIN			8		1	644	106 K	
JOIN FILTER CREATE	:BF0000		9		1	600	600	
TABLE ACCESS STORAGE FULL (SEL$7AE8FF08)	PART_GROUP		10		1	600	600	
JOIN FILTER USE	:BF0000		11		1	211 K	107 K	
TABLE ACCESS STORAGE FULL (SEL$7AE8FF08)	PART		12		1	211 K	107 K	
NESTED LOOPS			13		1	1,725 K	8,581 K	
TABLE ACCESS STORAGE FULL (SEL$703A7920)	SUPPLIER		14		1	575	2,862	
PARTITION RANGE ALL			15		2,862	2,999	8,581 K	0.49%
TABLE ACCESS BY LOCAL INDEX ROWID BATCHED (	LINEORDER		16		235 K	2,999	8,581 K	74.27%
INDEX RANGE SCAN (SEL$703A7920)	LINEORDER1_SUPP_FBI_N		17		235 K	32	8,581 K	24.27%
INDEX RANGE SCAN (SEL$703A7920)	CUSTOMER_PK		18		173 K	1	173 K	

Filter Predicates

("SUPPLIER"."S_REGION"='ASIA' AND INTERNAL_FUNCTION("SUPPLIER"."S_NATION"))

Close

The estimated cardinality represents the number of rows that optimizer expects to be returned from SUPPLIER after the filter predicates are applied:

```
    s_region = 'ASIA'
AND s_nation in ('CHINA','INDIA','JAPAN','VIETNAM')
```

Step 2: Fix the cardinality estimate for SUPPLIER

Fix the cardinality estimate for table SUPPLIER (estimated 575, actual 2,862)

- When addressing a problem with single table cardinality estimates, the first place to look is the state of the table statistics. Are they representative?

```
SELECT table_name, num_rows FROM user_tables WHERE table_name='SUPPLIER';

TABLE_NAME           NUM_ROWS
-------------------- ----------
SUPPLIER                  18000
```

The statistics for SUPPLIER indicate there 18,000 rows in the table

```
SELECT COUNT(*) FROM supplier;

  COUNT(*)
----------
     18000
```

The actual number is also 18,000.

The table level statistics for SUPPLIER seem to be up to date.

But, are they representative ?

Step 2: Fix the cardinality estimate for SUPPLIER

Fix the cardinality estimate for table SUPPLIER (estimated 575, actual 2,862)

- When addressing a problem with single table cardinality estimates, the first place to look is the state of the table statistics. Are they representative?
- A **selectivity** of a filter predicate indicates the estimated proportion of rows in the table that will be returned after the filter predicate is applied.
- The **estimated cardinality**, or the number of rows expected to be produced from the table after a filter predicate is applied, is a product of the number of rows in the table and the selectivity of the filter predicate.
- When filters on multiple columns are used, by default, the final selectivity is calculated by multiplying the individual selectivities of all filter predicates.

Step 2: Fix the cardinality estimate for SUPPLIER

Fix the cardinality estimate for table SUPPLIER (estimated 575, actual 2,862)

```
SELECT column_name, num_distinct
FROM user_tab_col_statistics
WHERE table_name='SUPPLIER'
  AND column_name IN ('S_REGION','S_NATION');

COLUMN_NAME          NUM_DISTINCT
-------------------- ------------
S_NATION                       25
S_REGION                        5
```

Assuming no histograms are present.

Selectivity for `s_nation in ('CHINA')` is 1/NUM_DISTINCT, that is 1/25
Selectivity for `s_nation in ('CHINA','INDIA','JAPAN','VIETNAM')` is 4/25

Assuming no histograms are present.

Selectivity for `s_region in ('ASIA')` is 1/NUM_DISTINCT, that is 1/5

Selectivity when both filter predicates are evaluated is 4/25 x 1/5 = 0.032

The estimated cardinality when both filter predicates are evaluated is 18,000 (rows) x 0.032 = 576 (approx)

With histograms in place the calculations are different

```
EXPLAIN PLAN FOR
SELECT * FROM supplier
WHERE s_region = 'ASIA'
  AND s_nation in ('CHINA','INDIA','JAPAN','VIETNAM')

-----------------------------------------------------
| Id  | Operation                  | Name     | Rows  |
-----------------------------------------------------
|   0 | SELECT STATEMENT           |          |   575 |
|   1 |  TABLE ACCESS STORAGE FULL| SUPPLIER |   575 |
-----------------------------------------------------
```

Step 2: Fix the cardinality estimate for SUPPLIER

Fix the cardinality estimate for table SUPPLIER (estimated 575, actual 2,862)

Think about correlation

- What is the possibility that a row in SUPPLIER will have s_region='ASIA' and s_nation in ('CHINA','INDIA','JAPAN','VIETNAM')?
- Is there a country named Japan outside of Asia?

```
SELECT COUNT(DISTINCT s_region) as distinct_region
     , COUNT(DISTINCT s_nation) as distinct_nation
     , COUNT(DISTINCT s_region||' '||s_nation) as distinct_pairs
FROM supplier;
```

```
DISTINCT_REGION DISTINCT_NATION DISTINCT_PAIRS
--------------- --------------- --------------
              5              25             25
```

If the number of distinct pairs of values between s_region and s_nation is close to the number of distinct values of either s_region or s_nation, we can say that the values in these two columns are correlated.

s_region and s_nation are strongly correlated.

Create a correlated column group on s_region and s_nation!
Link: Managing Extended Statitics

Step 2: Fix the cardinality estimate for SUPPLIER

Fix the cardinality estimate for table SUPPLIER (estimated 575, actual 2,862)

```
SELECT DBMS_STATS.CREATE_EXTENDED_STATS(user,'SUPPLIER','(S_REGION,S_NATION)') FROM DUAL;

DBMS_STATS.CREATE_EXTENDED_STATS(USER,'SUPPLIER','(S_REGION,S_NATION)')
-----------------------------------------------------------------------
SYS_STU8TQAAMA8C62YIA0QMZ_A26F

EXEC DBMS_STATS.GATHER_TABLE_STATS(user,'SUPPLIER');

SELECT column_name, histogram FROM user_tab_col_statistics WHERE table_name='SUPPLIER' AND column_name IN
('S_REGION','S_NATION','SYS_STU8TQAAMA8C62YIA0QMZ_A26F');

COLUMN_NAME                              HISTOGRAM
---------------------------------------- ---------------
SYS_STU8TQAAMA8C62YIA0QMZ_A26F           NONE
S_NATION                                 FREQUENCY
S_REGION                                 FREQUENCY

EXPLAIN PLAN FOR
SELECT * FROM supplier
WHERE s_region = 'ASIA'
  AND s_nation in ('CHINA','INDIA','JAPAN','VIETNAM');

EXEC DBMS_STATS.GATHER_TABLE_STATS(user,'SUPPLIER');

SELECT column_name, histogram FROM user_tab_col_statistics WHERE table_name='SUPPLIER' AND column_name IN
('S_REGION','S_NATION','SYS_STU8TQAAMA8C62YIA0QMZ_A26F');

COLUMN_NAME                              HISTOGRAM
---------------------------------------- ---------------
SYS_STU8TQAAMA8C62YIA0QMZ_A26F           FREQUENCY
S_NATION                                 FREQUENCY
S_REGION                                 FREQUENCY
```

Extended statistic (column group) created, but no stats have been gathered.

Gather statistics for SUPPLIER table.

No histogram for the column group

If any of the columns that are used in the column group has histograms, in order for the column group to be considered by the optimizer, it needs to have histogram too.

Run EXPLAIN PLAN of a query that uses filter predicates on the columns used in the column group will populate SYS.COL_USAGE$

When stats are gathered Oracle uses SYS.COL_USAGE$ to find columns that will benefit from histograms.
In this case, it will find the column group that we created and will create a histogram on it

The histogram on the column group has been created

Step 2: Column group on SUPPLIER(S_REGION, S_NATION)

Step2: running time: 185 seconds

General

Status	Completed
SQL Text	SELECT /*+ monitor */ d_month, d_year, ...
Execution Started	Oct 5, 2022 5:04:42 PM GMT-07:00
Last Refresh Time	Oct 5, 2022 5:07:45 PM GMT-07:00
Execution ID	16777216
User Name	DW01@OCWDW
Fetch Calls	5

Time & Wait

Duration	3.05 m
Database Time	3.62 m
PL/SQL & Java	0 s
Activity	100%

I/O

Buffer Gets	34 M
I/O Requests	3,801 K
I/O Bytes	29.1 GB

The estimated cardinality for SUPPLIER is correct

Plan Statistics | SQL Text | Activity | Metrics | Optimizer Environment | Outline

Plan Hash Value 4039374634 Plan Size 18 Navigate to View Option Tabula Plan Notes More Columns

Operation	Object	Informatic	Line ID	Timeline	Executions	Est. Rows	Rows	Activity
SELECT STATEMENT			0		1		48	
SORT GROUP BY (SEL$CCD74004)			1		1	34	48	0.56%
HASH JOIN			2		1	1,315	173 K	
TABLE ACCESS STORAGE FULL (SEL$CCD74004)	DATE_DIM		3		1	365	365	
NESTED LOOPS			4		1	9,201	1,136 K	
HASH JOIN			5		1	9,201	1,136 K	2.23%
TABLE ACCESS STORAGE FULL (SEL$CCD74004)	SUPPLIER		6		1	2,862	2,862	
NESTED LOOPS			7		1	58 K	7,157 K	
NESTED LOOPS			8		1	58 K	7,157 K	
HASH JOIN			9		1	644	106 K	
JOIN FILTER CREATE	:BF0000		10		1	600	600	
TABLE ACCESS STORAGE FULL (SEL$CCD7400	PART_GROUP		11		1	600	600	
JOIN FILTER USE	:BF0000		12		1	211 K	109 K	
TABLE ACCESS STORAGE FULL (SEL$CCD7400	PART		13		1	211 K	109 K	0.56%
PARTITION RANGE ALL			14		106 K	90	7,157 K	1.68%
INDEX RANGE SCAN (SEL$CCD74004)	LINEORDER1_PART_N		15		8,692 K	90	7,157 K	41.90%
TABLE ACCESS BY LOCAL INDEX ROWID (SEL$CCD	LINEORDER		16		11 M	90	7,157 K	53.07%
INDEX RANGE SCAN (SEL$CCD74004)	CUSTOMER_PK		17		1,137 K	1	1,136 K	

Worked example: Progress

	Fix Category	Running Time
Initial Performance		256 seconds
Step 1: Statistics on PART	Statistics	211 seconds
Step 2: Column group on SUPPLIER	Statistics	185 seconds

Step 2: Column group on SUPPLIER(S_REGION,S_NATION)

General

Status	Completed
SQL Text	SELECT /*+ monitor */ d_month, d_year, ...
Execution Started	Oct 5, 2022 5:04:42 PM GMT-07:00
Last Refresh Time	Oct 5, 2022 5:07:45 PM GMT-07:00
Execution ID	16777216
User Name	DW01@OCWDW
Fetch Calls	5

Time & Wait

Duration	3.05 m
Database Time	3.62 m
PL/SQL & Java	0 s
Activity	100%

I/O

Buffer Gets	34 M
I/O Requests	3,801 K
I/O Bytes	29.1 GB

Plan Statistics | SQL Text | Activity | Metrics | Optimizer Environment | Outline

Plan Hash Value 4039374634 Plan Size 18 Navigate to View Option Tabular Execution Plan Plan Notes More Columns

Operation	Object	Informatic	Line ID	Timeline	Executions	Est. Rows	Rows	Activity
SELECT STATEMENT			0		1		48	
SORT GROUP BY (SEL$CCD74004)			1		1	34	48	
HASH JOIN			2		1	1,315	173 K	
TABLE ACCESS STORAGE FULL (SEL$CCD74004)	DATE_DIM		3		1	365	365	
NESTED LOOPS			4		1	9,201	1,136 K	
HASH JOIN			5		1	9,201	1,136 K	
TABLE ACCESS STORAGE FULL (SEL$CCD74004)	SUPPLIER		6		1	2,862	2,862	
NESTED LOOPS			7		1	58 K	7,157 K	
NESTED LOOPS			8		1	58 K	7,157 K	
HASH JOIN			9		1	644	106 K	
JOIN FILTER CREATE	:BF0000		10		1	600	600	
TABLE ACCESS STORAGE FULL (SEL$CCD7400	PART_GROUP		11		1	600	600	
JOIN FILTER USE	:BF0000		12		1	211 K	109 K	
TABLE ACCESS STORAGE FULL (SEL$CCD7400	PART		13		1	211 K	109 K	0.56%
PARTITION RANGE ALL			14		106 K	90	7,157 K	1.68%
INDEX RANGE SCAN (SEL$CCD74004)	LINEORDER1_PART_N		15		8,692 K	90	7,157 K	41.90%
TABLE ACCESS BY LOCAL INDEX ROWID (SEL$CCD	LINEORDER		16		11 M	90	7,157 K	53.07%
INDEX RANGE SCAN (SEL$CCD74004)	CUSTOMER_PK		17		1,137 K	1	1,136 K	

Most of the database time spent on accessing (through LINEORDER1_PART_N index) data in LINEORDER table.

Step 2: Column group on SUPPLIER(S_REGION,S_NATION)

General

Status	Completed
SQL Text	SELECT /*+ monitor */ d_month, d_year, ...
Execution Started	Oct 5, 2022 5:04:42 PM GMT-07:00
Last Refresh Time	Oct 5, 2022 5:07:45 PM GMT-07:00
Execution ID	16777216
User Name	DW01@OCWDW
Fetch Calls	5

Time & Wait

Duration	3.05 m
Database Time	3.62 m
PL/SQL & Java	0 s
Activity	100%

I/O

Buffer Gets	34 M
I/O Requests	3,801 K
I/O Bytes	29.1 GB

Plan Statistics | SQL Text | Activity | Metrics | Optimizer Environment | Outline

Plan Hash Value 4039374634 Plan Size 18 Navigate to View Option Tabular Execution Plan Plan Notes More Columns

Operation	Object		Executions	Est. Rows	Rows	Activity
SELECT STATEMENT		1	1		48	
SORT GROUP BY (SEL$CCD74004)		2	1	34	48	
HASH JOIN		3	1	1,315	173 K	
TABLE ACCESS STORAGE FULL (SEL$CCD74004)	DATE_DIM	4	1	365	365	
NESTED LOOPS		5	1	9,201	1,136 K	
HASH JOIN		6	1	9,201	1,136 K	
TABLE ACCESS STORAGE FULL (SEL$CCD74004)	SUPPLIER	7	1	2,862	2,862	
NESTED LOOPS		8	1	58 K	7,157 K	
NESTED LOOPS		9	1	58 K	7,157 K	
HASH JOIN		10	1	644	106 K	
JOIN FILTER CREATE	:BF0000	11	1	600	600	
TABLE ACCESS STORAGE FULL (SEL$CCD7400	PART_GROUP	12	1	600	600	
JOIN FILTER USE	:BF0000	13	1	211 K	109 K	
TABLE ACCESS STORAGE FULL (SEL$CCD7400	PART	14	1	211 K	109 K	0.56%
PARTITION RANGE ALL		15	106 K	90	7,157 K	1.68%
INDEX RANGE SCAN (SEL$CCD74004)	LINEORDER1_PART_N	16	8,692 K	90	7,157 K	41.90%
TABLE ACCESS BY LOCAL INDEX ROWID (SEL$CCD	LINEORDER	17	11 M	90	7,157 K	53.07%
INDEX RANGE SCAN (SEL$CCD74004)	CUSTOMER_PK	18	1,137 K	1	1,136 K	

NL Join between the results from the join between PART and PART_GROUP and LINEORDER

Most of the database time spent on accessing (through LINEORDER1_PART_N index) data in LINEORDER table.

Step 2: Column group on SUPPLIER(S_REGION,S_NATION)

General

Status	Completed
SQL Text	SELECT /*+ monitor */ d_month, d_year, ...
Execution Started	Oct 5, 2022 5:04:42 PM GMT-07:00
Last Refresh Time	Oct 5, 2022 5:07:45 PM GMT-07:00
Execution ID	16777216
User Name	DW01@OCWDW
Fetch Calls	5

Time & Wait

Duration	3.05 m
Database Time	3.62 m
PL/SQL & Java	0 s
Activity	100%

I/O

Buffer Gets	34 M
I/O Requests	3,801 K
I/O Bytes	29.1 GB

Plan Statistics | SQL Text | Activity | Metrics | Optimizer Environment | Outline

Plan Hash Value 4039374634 Plan Size 18 Navigate to View Option Tabular Execution Plan Plan Notes More Columns

Operation	Object		Executions	Est. Rows	Rows	Activity
SELECT STATEMENT		1	1		48	
SORT GROUP BY (SEL$CCD74004)		2			48	0.56%
HASH JOIN		3			173 K	
TABLE ACCESS STORAGE FULL (SEL$CCD74004)	DATE_DIM				365	
NESTED LOOPS		4			1,136 K	
HASH JOIN		5			1,136 K	
TABLE ACCESS STORAGE FULL (SEL$CCD74004)	SUPPLIER	6			2,862	
NESTED LOOPS		7	1		7,157 K	
NESTED LOOPS		8	1	58 K	7,157 K	
HASH JOIN		9	1	644	106 K	
JOIN FILTER CREATE	:BF0000	10	1	600	600	
TABLE ACCESS STORAGE FULL (SEL$CCD7400	PART_GROUP	11	1	600	600	
JOIN FILTER USE	:BF0000	12	1	211 K	109 K	
TABLE ACCESS STORAGE FULL (SEL$CCD7400	PART	13	1	211 K	109 K	0.56%
PARTITION RANGE ALL		14	106 K	90	7,157 K	1.68%
INDEX RANGE SCAN (SEL$CCD74004)	LINEORDER1_PART_N	15	8,692 K	90	7,157 K	41.90%
TABLE ACCESS BY LOCAL INDEX ROWID (SEL$CCD	LINEORDER	16	11 M	90	7,157 K	53.07%
INDEX RANGE SCAN (SEL$CCD74004)	CUSTOMER_PK	17	1,137 K	1	1,136 K	

NL Join between the results from the join between PART and PART_GROUP and LINEORDER

Join cardinality underestimate for the join between PART and PART_GROUP (estimated 644, actual 106K)

Single table cardinality estimates are spot on

Step 2: Column group on SUPPLIER(S_REGION,S_NATION)

General

Status	Completed
SQL Text	SELECT /*+ monitor */ d_month, d_year, ...
Execution Started	Oct 5, 2022 5:04:42 PM GMT-07:00
Last Refresh Time	Oct 5, 2022 5:07:45 PM GMT-07:00
Execution ID	16777216
User Name	DW01@OCWDW
Fetch Calls	5

Time & Wait

Duration	3.05 m
Database Time	3.62 m
PL/SQL & Java	0 s
Activity	100%

I/O

Buffer Gets	34 M
I/O Requests	3,801 K
I/O Bytes	29.1 GB

Plan Statistics | SQL Text | Activity | Metrics | Optimizer Environment | Outline

Plan Hash Value 4039374634 Plan Size 18 Navigate to

Plan Notes More Columns

Multi-column join

Operation	Object	Informatic	Line ID	Timeline	Executions				
SELECT STATEMENT			0		1			48	
SORT GROUP BY (SEL$CCD74004)			1		1		34	48	0.56%
HASH JOIN			2				1,315	173 K	
TABLE ACCESS STORAGE FULL (SEL$CCD74004)	DATE_DIM		3				365	365	
NESTED LOOPS			4				9,201	1,136 K	
HASH JOIN			5				9,201	1,136 K	2.23%
TABLE ACCESS STORAGE FULL (SEL$CCD74004)	SUPPLIER		6				2,862	2,862	
NESTED LOOPS			7				58 K	7,157 K	
NESTED LOOPS			8				58 K	7,157 K	
HASH JOIN			9				644	106 K	
JOIN FILTER CREATE	:BF0000		10				600	600	
TABLE ACCESS STORAGE FULL (SEL$CCD7400	PART_GROUP		11				600	600	
JOIN FILTER USE	:BF0000		12				211 K	109 K	
TABLE ACCESS STORAGE FULL (SEL$CCD7400	PART		13				211 K	109 K	0.56%
PARTITION RANGE ALL			14		106 K		90	7,157 K	1.68%
INDEX RANGE SCAN (SEL$CCD74004)	LINEORDER1_PART_N		15		8,692 K		90	7,157 K	41.90%
TABLE ACCESS BY LOCAL INDEX ROWID (SEL$CCD	LINEORDER		16		11 M		90	7,157 K	53.07%
INDEX RANGE SCAN (SEL$CCD74004)	CUSTOMER_PK		17		1,137 K		1	1,136 K	

Access Predicates

"PART"."P_MFGR"="PG_MFGR" AND "PART"."P_CATEGORY"="PG_CATEGORY" AND "PART"."P_BRAND1"="PG_BRAND1"

Close

ORACLE REAL-WORLD PERFORMANCE

What You Must Do: Constraints

- NOT NULL Constraints on Join Keys
- Primary Key Constraints on Dimension Join Keys
- Foreign Key Constraints on Fact Join Keys

What You Must Do: Primary Key and Foreign Key Constraints

```
alter table customer
add constraint customer_pk
  primary key (c_custkey)
  RELY;

alter table lineorder
add constraint lo_customer_pk
  foreign key (lo_custkey)
  references
  customer (c_custkey)
  RELY
  DISABLE NOVALIDATE;

alter system
 set query_rewrite_integrity=TRUSTED;
```

- There must be a primary key on the dimension table
- There must be a foreign key on the fact table
- The state of the constraint depends on trust in the ETL process and volume of data
- Constraints must be in RELY state
- It is *not* necessary to enforce constraints on the fact table
- You need to tell the optimizer you can trust constraints in the RELY state

What You Must Do: Validating ETL/ELT

```
SELECT *
FROM lineorder
LEFT OUTER JOIN customer
  ON lo_custkey = c_custkey
WHERE c_custkey IS NULL;
```

- How do we validate our data when our constraints are not enforced?
- In other words, when constraints are in RELY mode, how to we ensure we can rely on the quality of data being inserted into our fact table?
- This SQL checks for rows in lineorder for values of lo_custkey which do not exist in the customer dimension table

What You Must Do: Validating ETL/ELT

```
SELECT *
FROM lineorder
LEFT OUTER JOIN customer
  ON lo_custkey = c_custkey
LEFT OUTER JOIN date_dim
  ON lo_orderdate = d_datekey
LEFT OUTER JOIN part
  ON lo_partkey = p_partkey
LEFT OUTER JOIN supplier
  ON lo_suppkey = s_suppkey
WHERE c_custkey   IS NULL
  OR d_datekey    IS NULL
  OR p_partkey    IS NULL
  OR s_suppkey    IS NULL;
```

- We can also validate the rows in lineorder against multiple dimensions
- Check the lineorder table for rows which contain keys that do not exist in the dimension tables

What You Must Do: Constraints

How the constraints help the optimizer?

- Improve cardinality estimates
- Opens possibility for additional transformation such as : join elimination and other query transformation techniques

Step 3: Fix the join cardinality estimate

Fix the cardinality estimate for the join between PART and PART_GROUP

- Create a **primary key** on PART_GROUP
- Create a **foreign key** on PART **referencing** PART_GROUP
- Create a **primary key** on PART
- Create a **foreign key** on LINEORDER **referencing** PART

You'll need to tell the optimizer it can trust constraints in RELY state by setting :

```
query_rewrite_integrity=TRUSTED;
```

```
ALTER TABLE part_group ADD CONSTRAINT part_group_pk PRIMARY KEY (pg_mfgr, pg_category, pg_brand1) RELY;

ALTER TABLE part ADD CONSTRAINT part_group_fk FOREIGN KEY(p_mfgr, p_category, p_brand1) REFERENCING part_group RELY;

ALTER TABLE part ADD CONSTRAINT part_pk PRIMARY KEY(p_partkey) RELY;

ALTER TABLE lineorder ADD CONSTRAINT lineorder_part_fk FOREIGN KEY (lo_partkey) REFERENCING part RELY DISABLE NOVALIDATE;
```

Step 3: Constraints on PART, PART_GROUP and LINE ORDER

Step3: running time: 4 seconds

General

Status	Completed
SQL Text	SELECT /*+ monitor */ d_month, d_year, ...
Execution Started	Oct 5, 2022 5:09:41 PM GMT-07:00
Last Refresh Time	Oct 5, 2022 5:09:45 PM GMT-07:00
Execution ID	16777217
User Name	DW01@OCWDW
Fetch Calls	5

Time & Wait

Duration	4 s
Database Time	3.85 s
PL/SQL & Java	0 s
Activity	100%

I/O

Buffer Gets	881 K
I/O Requests	6,231
I/O Bytes	5.3 GB
Cell Offload Efficiency	93%

Plan Statistics | SQL Text | Activity | Metrics | Optimizer Environment | Outline

Plan Hash Value 3197623652 Plan Size 16 Navigate to

View Option: Tabular Execution Plan | Plan Notes | More Columns

Operation	Object	Informatic	Line ID	Timeline	Executions	Est. Rows	Rows	Activity
SELECT STATEMENT			0		1		48	
SORT GROUP BY (SEL$CCD74004)			1		1	34	48	
NESTED LOOPS			2		1	185 K	173 K	
HASH JOIN			3		1	185 K	173 K	
TABLE ACCESS STORAGE FULL (SEL$CCD74004)	PART_GROUP		4		1	600	600	
HASH JOIN			5		1	432 K	287 K	
JOIN FILTER CREATE	:BF0000		6		1	211 K	211 K	
TABLE ACCESS STORAGE FULL (SEL$CCD74004)	PART		7		1	211 K	211 K	
HASH JOIN			8		1	1,227 K	337 K	
TABLE ACCESS STORAGE FULL (SEL$CCD74004)	SUPPLIER		9		1	2,862	2,862	
HASH JOIN			10		1	7,715 K	2,120 K	
TABLE ACCESS STORAGE FULL (SEL$CCD74004)	DATE_DIM		11		1	365	365	
JOIN FILTER USE	:BF0000		12		1	54 M	14 M	
PARTITION RANGE ALL			13		1	54 M	14 M	
TABLE ACCESS STORAGE FULL (SEL$CCD7400	LINEORDER		14		82	54 M	14 M	75.00%
INDEX RANGE SCAN (SEL$CCD74004)	CUSTOMER_PK		15		174 K	1	173 K	25.00%

Worked example: Progress

	Fix Category	Running Time
Initial Performance		256 seconds
Step 1: Statistics on PART	Statistics	211 seconds
Step 2: Column group on SUPPLIER	Statistics	185 seconds
Step 3: Constraints on PART, PART_GROUP and LINEORDER	Constraints	4 seconds

Step 3: Constraints on PART, PART_GROUP and LINE ORDER

General

Status	Completed
SQL Text	SELECT /*+ monitor */ d_month, d_year, ...
Execution Started	Oct 5, 2022 5:09:41 PM GMT-07:00
Last Refresh Time	Oct 5, 2022 5:09:45 PM GMT-07:00
Execution ID	16777217
User Name	DW01@OCWDW
Fetch Calls	5

Time & Wait

Duration	4 s
Database Time	3.85 s
PL/SQL & Java	0 s
Activity	100%

I/O

Buffer Gets	881 K
I/O Requests	6,231
I/O Bytes	5.3 GB
Cell Offload Efficiency	93%

Most of the DB time spent on scanning data in LINEORDER table

25% of DB Time spent on accessing data in CUSTOMER table (index access path)

Plan Statistics | SQL Text | Activity | Metrics | Optimizer Environment | Outline

Plan Hash Value 3197623652 Plan Size 16 Navigate to

Tabular Execution Plan

Operation	Object	Informatic	Line ID	Timeline	Executions	...ws	Rows	Activity
SELECT STATEMENT			0				48	
SORT GROUP BY (SEL$CCD74004)			1			34	48	
NESTED LOOPS			2		1	185 K	173 K	
HASH JOIN			3		1	5 K	173 K	
TABLE ACCESS STORAGE FULL (SEL$CCD74004)	PART_GROUP		4		1	0	600	
HASH JOIN			5		1		287 K	
JOIN FILTER CREATE	:BF0000		6		1		211 K	
TABLE ACCESS STORAGE FULL (SEL$CCD74004)	PART		7		1		211 K	
HASH JOIN			8		1	1,227	337 K	
TABLE ACCESS STORAGE FULL (SEL$CCD74004)	SUPPLIER		9		1	2,862	2,862	
HASH JOIN			10		1	7,715 K	2,120 K	
TABLE ACCESS STORAGE FULL (SEL$CCD74004)	DATE_DIM		11		1	365	365	
JOIN FILTER USE	:BF0000		12		1	54 M	14 M	
PARTITION RANGE ALL			13		1	54 M	14 M	
TABLE ACCESS STORAGE FULL (SEL$CCD7400	LINEORDER		14		82	54 M	M	75.00%
INDEX RANGE SCAN (SEL$CCD74004)	CUSTOMER_PK		15		174 K	1	173 K	25.00%

Step 3: Constraints on PART, PART_GROUP and LINE ORDER

General

Status	Completed
SQL Text	SELECT /*+ monitor */ d_month, d_year, ...
Execution Started	Oct 5, 2022 5:09:41 PM GMT-07:00
Last Refresh Time	Oct 5, 2022 5:09:45 PM GMT-07:00
Execution ID	16777217
User Name	DW01@OCWDW
Fetch Calls	5

Time & Wait

Duration	4 s
Database Time	3.85 s
PL/SQL & Java	0 s
Activity	100%

I/O

Buffer Gets	881 K
I/O Requests	6,231
I/O Bytes	5.3 GB
Cell Offload Efficiency	93%

Plan Statistics | SQL Text | Activity | Metrics | Optimizer Environment | Outline

Save to Script

```
SELECT
    /*+ monitor */
        d_month, d_year, s_nation,
        SUM(lo_quantity) lo_quantity,
        SUM(lo_revenue) lo_revenue,
        SUM(lo_supplycost) lo_supplycost
        FROM lineorder l
                JOIN customer      ON lo_custkey = c_custkey
                JOIN date_dim      ON lo_orderdate = d_datekey
                JOIN part          ON lo_partkey = p_partkey
                JOIN part_group pg ON p_mfgr=pg_mfgr and p_category=pg_category and p_brand1=pg_brand1
                JOIN supplier      ON lo_suppkey = s_suppkey
        WHERE   d_year = 2007
        AND     s_region = 'ASIA'
    AND      s_nation in ('CHINA','INDIA','JAPAN','VIETNAM')
        AND     pg_type in (1,2,3)
        AND     p_size<12
GROUP BY        d_month, d_year, s_nation
ORDER BY        d_month, d_year, s_nation
```

CUSTOMER table joined to LINEORDER. But, there are no other columns from CUSTOMER table used in the query.

Is the join to CUSTOMER table needed in the query?

Can the optimizer eliminate the join automatically?

Yes if there is FK from LINEORDER to CUSTOMER.
- Join elimination transformation

Step 4: Eliminate the join to CUSTOMER

Help the optimizer safely eliminate the join to CUSTOMER

- Create a **primary key** on CUSTOMER
- Create a **foreign key** on LINEORDER **referencing** CUSTOMER

You'll need to tell the optimizer it can trust constraints in RELY state by setting :

```
query_rewrite_integrity=TRUSTED;
```

```
ALTER TABLE customer ADD CONSTRAINT customer_pk PRIMARY KEY (c_custkey) RELY;

ALTER TABLE lineorder ADD CONSTRAINT lineorder_cust_fk FOREIGN KEY (lo_custkey) REFERENCING customer RELY DISABLE NOVALIDATE;
```

Step 4: PK on CUSTOMER – FK on LINEORDER

Step4: running time: 4 seconds

General

Status	Completed
SQL Text	SELECT /*+ monitor */ d_month, d_year, ...
Execution Started	Oct 5, 2022 5:10:22 PM GMT-07:00
Last Refresh Time	Oct 5, 2022 5:10:26 PM GMT-07:00
Execution ID	16777218
User Name	DW01@OCWDW
Fetch Calls	5

Time & Wait

Duration	4 s
Database Time	3.49 s
PL/SQL & Java	0 s
Activity	100%

I/O

Buffer Gets	708 K
I/O Requests	5,758
I/O Bytes	5.4 GB
Cell Offload Efficiency	94%

Plan Statistics | SQL Text | Activity | Metrics | Optimizer Environment | Outline

Plan Hash Value 2536171388 Plan Size 14 Navigate to View Option Tabular Execution Plan Plan Notes More Columns

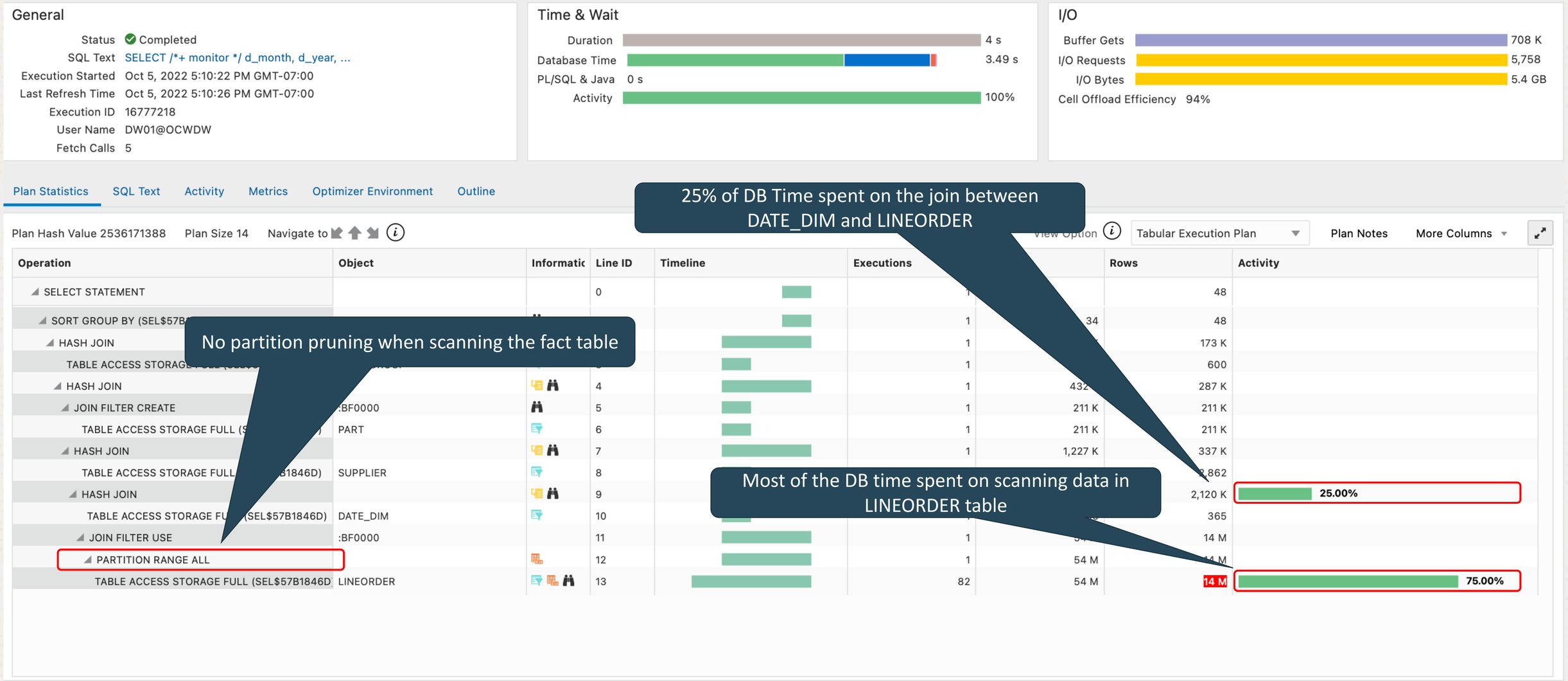
Operation	Object	Informatic	Line ID	Timeline	Executions	Est. Rows	Rows	Activity
SELECT STATEMENT			0		1		48	
SORT GROUP BY (SEL$57B1846D)			1		1	34	48	
HASH JOIN			2		1	185 K	173 K	
TABLE ACCESS STORAGE FULL (SEL$57B1846D)	PART_GROUP		3		1	600	600	
HASH JOIN			4		1	432 K	287 K	
JOIN FILTER CREATE	:BF0000		5		1	211 K	211 K	
TABLE ACCESS STORAGE FULL (SEL$57B1846D)	PART		6		1	211 K	211 K	
HASH JOIN			7		1	1,227 K	337 K	
TABLE ACCESS STORAGE FULL (SEL$57B1846D)	SUPPLIER		8		1	2,862	2,862	
HASH JOIN			9		1	7,715 K	2,120 K	25.00%
TABLE ACCESS STORAGE FULL (SEL$57B1846D)	DATE_DIM		10		1	365	365	
JOIN FILTER USE	:BF0000		11		1	54 M	14 M	
PARTITION RANGE ALL			12		1	54 M	14 M	
TABLE ACCESS STORAGE FULL (SEL$57B1846D	LINEORDER		13		82	54 M	14 M	75.00%

The join to CUSTOMER table is eliminated

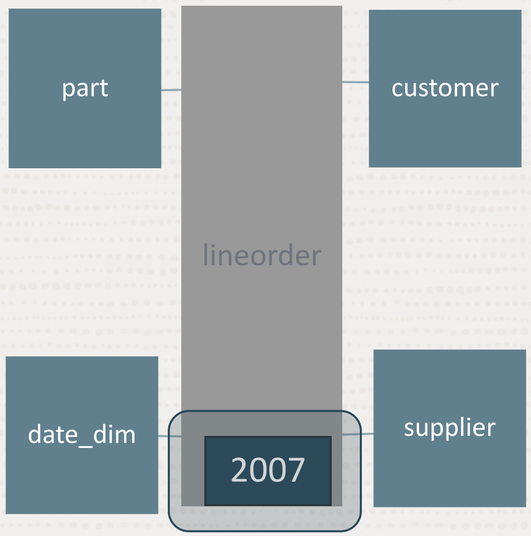
Worked example: Progress

	Fix Category	Running Time
Initial Performance		256 seconds
Step 1: Statistics on PART	Statistics	211 seconds
Step 2: Column group on SUPPLIER	Statistics	185 seconds
Step 3: Constraints on PART, PART_GROUP and LINEORDER	Constraints	4 seconds
Step 4: PK on CUSTOMER and FK on LINEORDER referencing CUSTOMER	Constraints	4 seconds

Step 4: PK on CUSTOMER – FK on LINEORDER



What You Must Do: Partitioning



- Typically RANGE or INTERVAL
  - Typically fact tables in star schemas are partitioned on a column of DATE data type
- Reduces the number of rows extracted from the fact table (i.e., early filtering)
- Improves manageability

Step 5: Correct the partitioning strategy

The query is interested only in data from 2007 (d_year=2007)

The partition pruning reduces the amount of data being scanned from the fact table

- LINEORDER is partitioned on LO_COMMITDATE

```
SELECT * FROM USER_PART_KEY_COLUMNS WHERE name='LINEORDER';

NAME                 OBJEC COLUMN_NAME          COLUMN_POSITION COLLATED_COLUMN_ID
-------------------- ----- -------------------- --------------- ------------------
LINEORDER            TABLE LO_COMMITDATE                      1
```

Partition pruning doesn't take place because LINEORDER is not partitioned on LO_ORDERDATE

- LINEORDER is joined to DATE_DIM on LO_ORDERDATE

```
SELECT * FROM USER_PART_KEY_COLUMNS WHERE name='LINEORDER';

NAME                 OBJEC COLUMN_NAME          COLUMN_POSITION COLLATED_COLUMN_ID
-------------------- ----- -------------------- --------------- ------------------
LINEORDER            TABLE LO_COMMITDATE                      1
```

Step 5: LINEORDER partitioned on LO_ORDERD

Step5: running time: 4 seconds

General

Status	Completed
SQL Text	SELECT /*+ monitor */ d_month, d_year, ...
Execution Started	Oct 5, 2022 5:11:04 PM GMT-07:00
Last Refresh Time	Oct 5, 2022 5:11:08 PM GMT-07:00
Execution ID	16777219
User Name	DW01@OCWDW
Fetch Calls	5

Time & Wait

Duration	4 s
Database Time	3.47 s
PL/SQL & Java	0 s
Activity	100%

I/O

Buffer Gets	708 K
I/O Requests	5,741
I/O Bytes	5.4 GB
Cell Offload Efficiency	94%

Plan Statistics | SQL Text | Activity | Metrics | Optimizer Environment | Outline

Plan Hash Value 2536171388 Plan Size 14 Navigate to View Option Tabular Execution Plan Plan Notes More Columns

Operation	Object	Informatic	Line ID	Timeline	Executions	Est. Rows	Rows	Activity
SELECT STATEMENT			0		1		48	
SORT GROUP BY (SEL$57B1846D)			1		1	34	48	
HASH JOIN			2		1	185 K	173 K	
TABLE ACCESS STORAGE FULL (SEL$57B1846D)	PART_GROUP		3		1	600	600	
HASH JOIN			4		1	433 K	287 K	
JOIN FILTER CREATE	:BF0000		5		1	211 K	211 K	
TABLE ACCESS STORAGE FULL (SEL$57B1846D)	PART		6		1	211 K	211 K	25.00%
HASH JOIN			7		1	1,216 K	337 K	
TABLE ACCESS STORAGE FULL (SEL$57B1846D)	SUPPLIER		8		1	2,862	2,862	
HASH JOIN			9		1	7,715 K	2,120 K	25.00%
TABLE ACCESS STORAGE FULL (SEL$57B1846D)	DATE_DIM		10		1	365	365	
JOIN FILTER USE	:BF0000		11		1	54 M	14 M	
PARTITION RANGE ALL			12		1	54 M	14 M	
TABLE ACCESS STORAGE FULL (SEL$57B1846D)	LINEORDER		13		80	54 M	14 M	50.00%

Still no partition pruning. But why?

Worked example: Progress

	Fix Category	Running Time
Initial Performance		256 seconds
Step 1: Statistics on PART	Statistics	211 seconds
Step 2: Column group on SUPPLIER	Statistics	185 seconds
Step 3: Constraints on PART, PART_GROUP and LINEORDER	Constraints	4 seconds
Step 4: PK on CUSTOMER and FK on LINEORDER referencing CUSTOMER	Constraints	4 seconds
Step 5: LINEORDER partitioned on LO_ORDERDATE	Partitioning	4 seconds

Step 5: LINEORDER partitioned on LO_ORDERDATE

General

Status	Completed
SQL Text	SELECT /*+ monitor */ d_month, d_year, ...
Execution Started	Oct 5, 2022 5:11:04 PM GMT-07:00
Last Refresh Time	Oct 5, 2022 5:11:08 PM GMT-07:00
Execution ID	16777219
User Name	DW01@OCWDW
Fetch Calls	5

Time & Wait

Duration	4 s
Database Time	3.47 s
PL/SQL & Java	0 s
Activity	

I/O

Buffer Gets	708 K
I/O Requests	5,741
I/O Bytes	5.4 GB

INTERNAL_FUNCTION indicates D_DATEKEY and LO_ORDERDATE are not of same data type

Plan Statistics | SQL Text | Activity | Metrics | Optimizer Environment | Outline

Plan Hash Value 2536171388 Plan Size 14 Navigate to View Option Tabular Execution Plan Plan Notes More Columns

Operation	Object	Informatic	Lin			Est. Rows	Rows	Activity
SELECT STATEMENT			0		1		48	
SORT GROUP BY (SEL$57B1846D)			1		1	34	48	
HASH JOIN			2		1	185 K	173 K	
TABLE ACCESS STORAGE FULL (SEL$57B1846D)	PART_GROUP		3		1	600	600	
HASH JOIN			4		1	433 K	287 K	
JOIN FILTER CREATE	:BF0000		5		1	211 K	211 K	
TABLE ACCESS STORAGE FULL (SEL$57B1846D)	PART		6		1	211 K	211 K	25.00%
HASH JOIN			7		1	1,216 K	337 K	
TABLE ACCESS STORAGE FULL (SEL$57B1846D)	SUPPLIER		8		1	2,862	2,862	
HASH JOIN			9		1	7,715 K	2,120 K	25.00%
TABLE ACCESS STORAGE FULL (SEL$57B1846D)	DATE_DIM		10		1	365	365	
JOIN FILTER USE	:BF0000		11		1	54 M	14 M	
PARTITION RANGE ALL					1	54 M	14 M	
TABLE ACCESS STORAGE FULL (SEL$57B1846D)	LINEORDER				80	54 M	14 M	50.00%

Access Predicates

"D_DATEKEY"=INTERNAL_FUNCTION("L"."LO_ORDERDATE")

Close

Still no partition pruning

What You Must Do: Data Types

- Data types need to be the same on Primary Key and Foreign Key columns
- Data type precision needs to be the same on Primary Key and Foreign Key columns
- Avoid runtime data type conversion

```
SELECT column_name, data_type FROM USER_TAB_COLUMNS WHERE table_name='LINEORDER' AND column_name='LO_ORDERDATE’;

COLUMN_NAME          DATA_TYPE
-------------------- --------------------
LO_ORDERDATE         DATE

SELECT column_name, data_type FROM USER_TAB_COLUMNS WHERE table_name='DATE_DIM' AND column_name='D_DATEKEY';

COLUMN_NAME          DATA_TYPE
-------------------- --------------------
D_DATEKEY            TIMESTAMP(6)
```

Data type mismatch on the join columns between LINEORDER and DATE_DIM

Step 6: Correct data type on DATE.D_DATEKEY

Step6: running time: 1 second

General

Status	Completed
SQL Text	SELECT /*+ monitor */ d_month, d_year, ...
Execution Started	Oct 5, 2022 5:12:32 PM GMT-07:00
Last Refresh Time	Oct 5, 2022 5:12:33 PM GMT-07:00
Execution ID	16777221
User Name	DW01@OCWDW
Fetch Calls	5

Time & Wait

Duration	1 s
Database Time	1.34 s
PL/SQL & Java	0 s
Activity	100%

I/O

Buffer Gets	117 K
I/O Requests	972
I/O Bytes	910.9 MB
Cell Offload Efficiency	80%

Plan Statistics | SQL Text | Activity | Optimizer Environment | Outline

Plan Hash Value 601704106 Plan Size 17 Navigate to View Option Tabular Execution Plan Plan Notes More Columns

Operation	Object	Informatic	Line ID	Timeline	Executions	Est. Rows	Rows	Activity
SELECT STATEMENT			0		1		48	
SORT GROUP BY (SEL$57B			1			34	48	
HASH JOIN			2			K	173 K	
TABLE ACCESS STORAGE FULL (SEL$5	GROUP		3			00	600	
HASH JOIN			4			K	287 K	
JOIN FILTER CREATE	:BF0000		5			K	211 K	
TABLE ACCESS STORAGE FULL (	PART		6			K	211 K	
HASH JOIN			7			K	337 K	
TABLE ACCESS STORAGE FULL	846D)	SUPPLIER	8			862	2,862	
HASH JOIN			9		1	8,197 K	2,120 K	100.00%
JOIN FILTER CREATE	:BF0002		10		1	365	365	
PART JOIN FILTER CREA	:BF0001		11		1	365	365	
TABLE ACCESS STORAG ULL (SEL$57B1846D	DATE_DIM		12		1	365	365	
JOIN FILTER USE	:BF0000		13		1	54 M	2,120 K	
JOIN FILTER USE	:BF0002		14		1	54 M	8,190 K	
PARTITION RANGE JOIN-FILTER			15		1	54 M	8,190 K	
TABLE ACCESS STORAGE FULL (SEL$57B1846	LINEORDER		16		12	54 M	8,190 K	

Partition pruning when scanning the fact table

Now we have a right deep tree plan. Partition pruning kicks in.

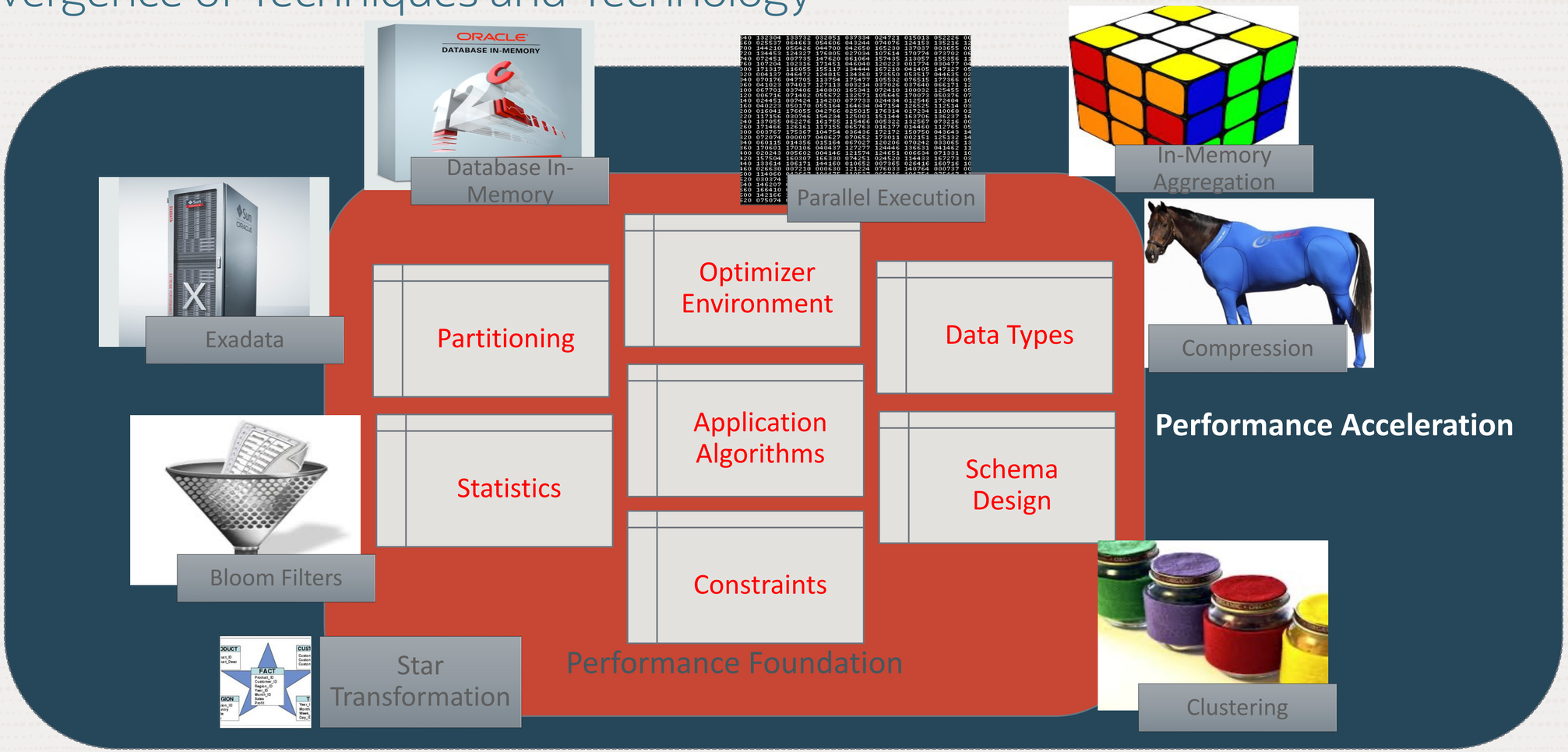
Bloom filters are used to pre-filter data from the fact table while the data is scanned

Worked example: Progress

	Fix Category	Running Time
Initial Performance		256 seconds
Step 1: Statistics on PART	Statistics	211 seconds
Step 2: Column group on SUPPLIER	Statistics	185 seconds
Step 3: Constraints on PART, PART_GROUP and LINEORDER	Constraints	4 seconds
Step 4: PK on CUSTOMER and FK on LINEORDER referencing CUSTOMER	Constraints	4 seconds
Step 5: LINEORDER partitioned on LO_ORDERDATE	Partitioning	4 seconds
Step 6: Correct data types	Data types	1 second

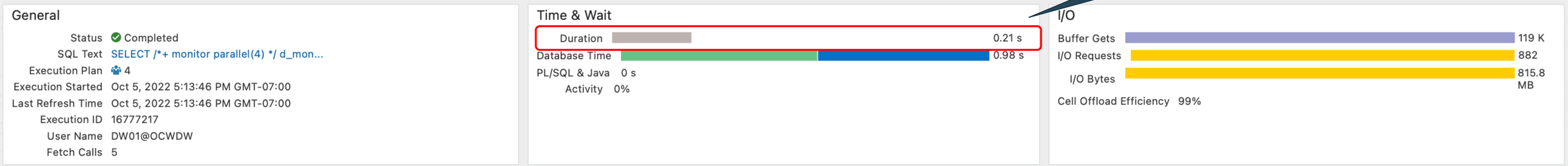
The Dimensional Model

Convergence of Techniques and Technology

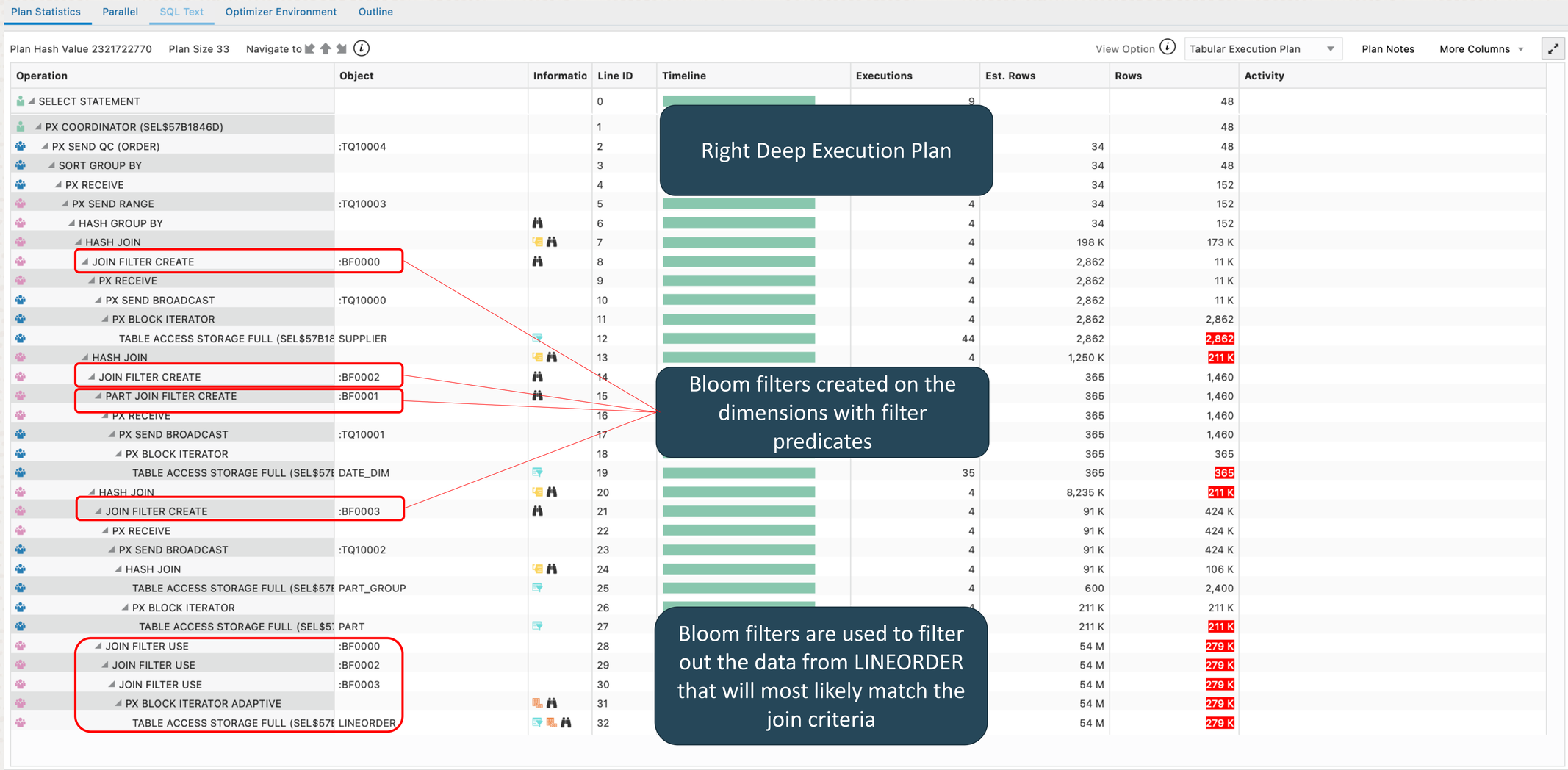


Step 7: Parallel execution

Step7: running time: 0.21 seconds



Step 7: Parallel execution



Worked example: Progress

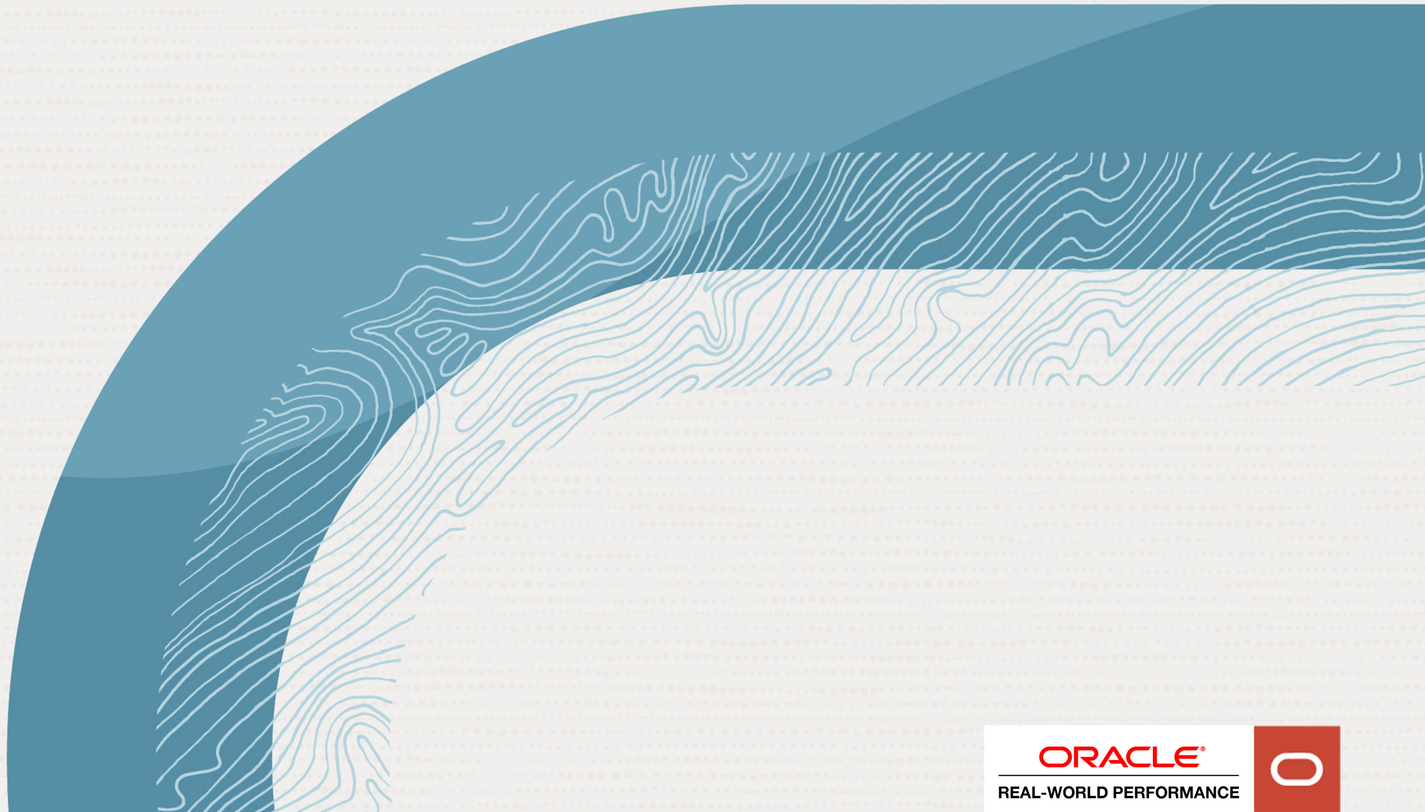
	Fix Category	Running Time
Initial Performance		256 seconds
Step 1: Statistics on PART	Statistics	211 seconds
Step 2: Column group on SUPPLIER	Statistics	185 seconds
Step 3: Constraints on PART, PART_GROUP and LINEORDER	Constraints	4 seconds
Step 4: PK on CUSTOMER and FK on LINEORDER referencing CUSTOMER	Constraints	4 seconds
Step 5: LINEORDER partitioned on LO_ORDERDATE	Partitioning	4 seconds
Step 6: Correct data types	Data types	1 second
Step 7: Parallel Execution	Other technologies	0.21s

What Do You Gain by Following the Prescription?

- Better cardinality estimates
- Better execution plans
- More access paths available
- Ability for the optimizer to perform many transformations and optimizations (join elimination, materialized view rewrites, In-Memory Aggregation transformation, and many more)
- Partition pruning
- Exploit other technologies for optimal performance
  - Parallel Execution
  - Materialized Views
  - Compression
  - Database In-Memory

Star Query Execution on Autonomous Data Warehouse

Analysis of the default performance of our worked example



ORACLE
REAL-WORLD PERFORMANCE

Star Query on Autonomous Data Warehouse

Analyze out of the box performance of the worked example when running on Autonomous Data Warehouse (ADW)

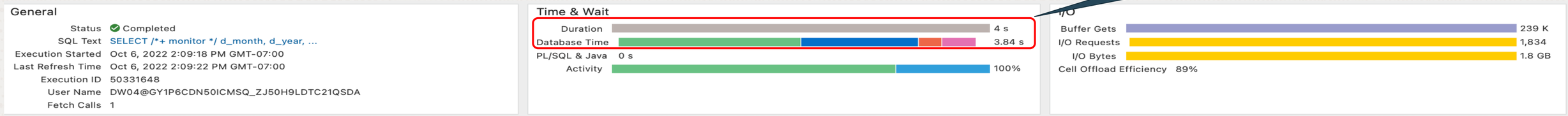
The session that executes the query will use two different services

- Low Service (serial execution)
- Medium Service (parallel execution with DoP 4)

ADW: Out of the Box Performance (Serial)

- The query is executed by a session connected to the Low service
  - SQL statements executed using the Low service run serially

Initial running time: 4 seconds



ADW: Out of the Box Performance (Serial

Cardinality estimate for PART is spot on:
- High frequency stats reduce the chance for stale stats

The execution plan is different and uses In-Memory Aggregation (Vector Transformation)
In ADW IMA is enabled and can take advantage of the data stored in columnar store that resides on flash disks

Plan Statistics | SQL Text | Activity | Metrics | Optimizer Environment | Outline

Plan Hash Value 802987796 Plan Size 44 Navigate to

Tabular Execution Plan | Plan Notes | More Columns

Operation	Object	Informatio	Line			t. Rows		Activity
KEY VECTOR CREATE BUFFERED	:KV0002		12				1	
VIEW (SEL$C9F9BE95)	VW_VTN_63105048		13			1	106 K	
HASH JOIN (SEL$C9F9BE95)			14			641	106 K	
JOIN FILTER CREATE	:BF0000		15			600	600	
TABLE ACCESS STORAGE FULL (SEL$C9F9BE95)	PART_GROUP		16			600	600	
JOIN FILTER USE	:BF0000		17			212 K	107 K	
TABLE ACCESS STORAGE FULL (SEL$C9F9BE95)	PART		18		1	212 K	107 K	
LOAD AS SELECT (CURSOR DURATION MEMORY) (SEL$1D	SYS_TEMP_0FD9EFFBD_2B216ACB		19		1		1	
HASH GROUP BY			20		1	25	4	
KEY VECTOR CREATE BUFFERED	:KV0003		21		1	25	4	
TABLE ACCESS STORAGE FULL (SEL$1D319852)	SUPPLIER		22		1	567	2,862	
SORT GROUP BY			23		1	213	48	
HASH JOIN			24		1	275	48	
MERGE JOIN CARTESIAN			25		1	300	48	
MERGE JOIN CARTESIAN			26		1	12	12	
MERGE JOIN CARTESIAN			27		1	1	1	
TABLE ACCESS STORAGE FULL (SEL$4578C471)	SYS_TEMP_0FD9EFFBA_2B216ACB		28		1	1	1	
BUFFER SORT			29		1	1	1	
TABLE ACCESS STORAGE FULL (SEL$4578C471)	SYS_TEMP_0FD9EFFBC_2B216ACB		30		1	1	1	
BUFFER SORT			31		1	12	12	
TABLE ACCESS STORAGE FULL (SEL$4578C471)	SYS_TEMP_0FD9EFFBB_2B216ACB		32		1	12	12	
BUFFER SORT			33		12	25	48	
TABLE ACCESS STORAGE FULL (SEL$4578C471)	SYS_TEMP_0FD9EFFBD_2B216ACB		34		1	25	4	
VIEW (SEL$ECC0E6B9)	VW_VT_F074A866		35		1	275	48	
VECTOR GROUP BY (SEL$ECC0E6B9)			36		1	275	48	
HASH GROUP BY			37		1	275		
KEY VECTOR USE	:KV0000		38		1	275	172 K	25.00%
KEY VECTOR USE	:KV0001		39		1	275	7,151 K	
KEY VECTOR USE	:KV0003		40		1	1,876	7,151 K	
KEY VECTOR USE	:KV0002		41		1	58 K	7,151 K	
PARTITION RANGE ALL			42		1	54 M	7,151 K	
TABLE ACCESS STORAGE FULL (SEL$ECC0E	LINEORDER		43		82	54 M	7,151 K	75.00%

ORACLE REAL-WORLD PERFORMANCE

ADW: Out of the Box Performance (Serial)

But, the cardinality estimate for SUPPLIER is still wrong due to the lack of extended statistics (correlated column group)

Plan Statistics | SQL Text | Activity | Metrics | Optimizer Environment | Outline

Plan Hash Value 802987796 Plan Size 44 Navigate to

Operation	Object	Informatio	Line ID	Timeline	Executions	Est. Rows		Activity
KEY VECTOR CREATE BUFFERED	:KV0002		12		1		1	
VIEW (SEL$C9F9BE95)	VW_VTN_63105048		13		1		106 K	
HASH JOIN (SEL$C9F9BE95)			14		1		106 K	
JOIN FILTER CREATE	:BF0000		15		1		600	
TABLE ACCESS STORAGE FULL (SEL$C9F9BE95)	PART_GROUP		16		1		600	
JOIN FILTER USE	:BF0000		17		1	K	107 K	
TABLE ACCESS STORAGE FULL (SEL$C9F9BE95)	PART		18		1	212 K	107 K	
LOAD AS SELECT (CURSOR DURATION MEMORY) (SEL$1D	SYS_TEMP_0FD9EFFBD_2B216ACB		19		1		1	
HASH GROUP BY			20		1	25	4	
KEY VECTOR CREATE BUFFERED	:KV0003		21		1	25	4	
TABLE ACCESS STORAGE FULL (SEL$1D319852)	SUPPLIER		22		1	567	2,862	
SORT GROUP BY			23		1	213	48	
HASH JOIN			24		1	275	48	
MERGE JOIN CARTESIAN			25		1	300	48	
MERGE JOIN CARTESIAN			26		1	12	12	
MERGE JOIN CARTESIAN			27		1	1	1	
TABLE ACCESS STORAGE FULL (SEL$4578C471)	SYS_TEMP_0FD9EFFBA_2B216ACB		28		1	1	1	
BUFFER SORT			29		1	1	1	
TABLE ACCESS STORAGE FULL (SEL$4578C471)	SYS_TEMP_0FD9EFFBC_2B216ACB		30		1	1	1	
BUFFER SORT			31		1	12	12	
TABLE ACCESS STORAGE FULL (SEL$4578C471)	SYS_TEMP_0FD9EFFBB_2B216ACB		32		1	12	12	
BUFFER SORT			33		12	25	48	
TABLE ACCESS STORAGE FULL (SEL$4578C471)	SYS_TEMP_0FD9EFFBD_2B216ACB		34		1	25	4	
VIEW (SEL$ECC0E6B9)	VW_VT_F074A866		35		1	275	48	
VECTOR GROUP BY (SEL$ECC0E6B9)			36		1	275	48	
HASH GROUP BY			37		1	275		
KEY VECTOR USE	:KV0000		38		1	275	172 K	25.00%
KEY VECTOR USE	:KV0001		39		1	275	7,151 K	
KEY VECTOR USE	:KV0003		40		1	1,876	7,151 K	
KEY VECTOR USE	:KV0002		41		1	58 K	7,151 K	
PARTITION RANGE ALL			42		1	54 M	7,151 K	
TABLE ACCESS STORAGE FULL (SEL$ECC0E	LINEORDER		43		82	54 M	7,151 K	75.00%

ORACLE REAL-WORLD PERFORMANCE

ADW: Out of the Box Performance (Serial)

Plan Statistics | SQL Text | Activity | Metrics | Optimizer Environment | Outline

Plan Hash Value 802987796 Plan Size 44 Navigate to View Option Tabular Execution Plan Plan Notes More Columns

Operation	Object	Informatio	Line ID	Timeline	Executions	Est. Rows	Rows	Activity
KEY VECTOR CREATE BUFFERED	:KV0002		12		1	1	1	
VIEW (SEL$C9F9BE95)	VW_VTN_63105048		13		1	641	106 K	
HASH JOIN (SEL$C9F9BE95)			14		1	641	106 K	
JOIN FILTER CREATE	:BF0000		15		1	600	600	
TABLE ACCESS STORAGE FULL (SEL$C9F9BE95)	PART_GROUP		16		1	600	600	
JOIN FILTER USE	:BF0000		17		1		107 K	
TABLE ACCESS STORAGE FULL (SEL$C9F9BE95)	PART		18		1	21	107 K	
LOAD AS SELECT (CURSOR DURATION MEMORY) (SEL$1D	SYS_TEMP_0FD9EFFBD_2B216ACB		19		1		1	
HASH GROUP BY			20		1			
KEY VECTOR CREATE BUFFERED	:KV0003		21		1			
TABLE ACCESS STORAGE FULL (SEL$1D319852)	SUPPLIER		22		1			
SORT GROUP BY			23		1			
HASH JOIN			24		1			
MERGE JOIN CARTESIAN			25		1			
MERGE JOIN CARTESIAN			26		1			
MERGE JOIN CARTESIAN			27		1	1	1	
TABLE ACCESS STORAGE FULL (SEL$4578C471)	SYS_TEMP_0FD9EFFBA_2B216ACB		28		1	1	1	
BUFFER SORT			29		1	1	1	
TABLE ACCESS STORAGE FULL (SEL$4578C471)	SYS_TEMP_0FD9EFFBC_2B216ACB		30		1	1	1	
BUFFER SORT			31		1	12	12	
TABLE ACCESS STORAGE FULL (SEL$4578C471)	SYS_TEMP_0FD9EFFBB_2B216ACB		32		1	12	12	
BUFFER SORT			33		12	25	48	
TABLE ACCESS STORAGE FULL (SEL$4578C471)	SYS_TEMP_0FD9EFFBD_2B216ACB		34		1	25	4	
VIEW (SEL$ECC0E6B9)	VW_VT_F074A866		35		1	275	48	
VECTOR GROUP BY (SEL$ECC0E6B9)			36		1	275	48	
HASH GROUP BY			37		1	275		
KEY VECTOR USE	:KV0000		38		1	275	172 K	25.00%
KEY VECTOR USE	:KV0001		39		1	275	7,151 K	
KEY VECTOR USE	:KV0003		40		1	1,876	7,151 K	
KEY VECTOR USE	:KV0002		41		1	58 K	7,151 K	
PARTITION RANGE ALL			42		1	54 M	7,151 K	
TABLE ACCESS STORAGE FULL (SEL$ECC0E6	LINEORDER		43		82	54 M	7,151 K	75.00%

Join cardinality between PART and PART_GROUP still bad due to lack of constraints

ORACLE REAL-WORLD PERFORMANCE

ADW: Out of the Box Performance (Serial)

Plan Statistics | SQL Text | Activity | Metrics | Optimizer Environment | Outline

Plan Hash Value 802987796 Plan Size 44 Navigate to

View Option: Tabular Execution Plan | Plan Notes | More Columns

Operation	Object	Informatio	Line ID	Timeline	Executions	Est. Rows	Rows	Activity
KEY VECTOR CREATE BUFFERED	:KV0002		12		1	1	1	
VIEW (SEL$C9F9BE95)	VW_VTN_63105048		13		1	641	106 K	
HASH JOIN (SEL$C9F9BE95)			14		1	641	106 K	
JOIN FILTER CREATE	:BF0000		15		1	600	600	
TABLE ACCESS STORAGE FULL (SEL$C9F9BE95)	PART_GROUP		16		1	600	600	
JOIN FILTER USE	:BF0000		17		1	212 K	107 K	
TABLE ACCESS STORAGE FULL (SEL$C9F9BE95)	PART		18		1	212 K	107 K	
LOAD AS SELECT (CURSOR DURATION MEMORY) (SEL$1D3	SYS_TEMP_0FD9EFFBD_2B216ACB		19		1		1	
HASH GROUP BY			20		1	25	4	
KEY VECTOR CREATE BUFFERED	:KV0003		21		1	25	4	
TABLE ACCESS STORAGE FULL (SEL$1D319852)	SUPPLIER		22		1	567	2,862	
SORT GROUP BY			23		1	213	48	
HASH JOIN			24		1	275	48	
MERGE JOIN CARTESIAN			25		1	300	48	
MERGE JOIN CARTESIAN			26		1	12	12	
MERGE JOIN CARTESIAN			27		1	1	1	
TABLE ACCESS STORAGE FULL (SEL$4578C471)	SYS_TEMP_0FD9EFFBA_2B216ACB		28		1	1	1	
BUFFER SORT			29		1	1	1	
TABLE ACCESS STORAGE FULL (SEL$4578C471)	SYS_TEMP_0FD9EFFBC_2B216ACB		30		1	1	1	
BUFFER SORT			31		1	12	12	
TABLE ACCESS STORAGE FULL (SEL$4578C471)	SYS_TEMP_0FD9EFFBB_2B216ACB				1	12	12	
BUFFER SORT					12	25	48	
TABLE ACCESS STORAGE FULL (SEL$4578C471)	SYS_TEMP_0FD9EFFBD_2B216A				1	25	4	
VIEW (SEL$ECC0E6B9)	VW_VT_F074A866				1	275	48	
VECTOR GROUP BY (SEL$ECC0E6B9)					1	275	48	
HASH GROUP BY					1	275		
KEY VECTOR USE	:KV0000				1	275	172 K	25.00%
KEY VECTOR USE	:KV0001				1	275	7,151 K	
KEY VECTOR USE	:KV0003		40		1	1,876	7,151 K	
KEY VECTOR USE	:KV0002		41		1	58 K	7,151 K	
PARTITION RANGE ALL			42		1	54 M	7,151 K	
TABLE ACCESS STORAGE FULL (SEL$ECC0E6	LINEORDER		43		82	54 M	7,151 K	75.00%

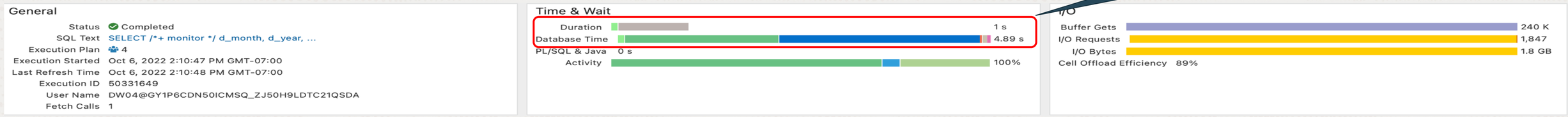
No partition pruning due to bad partitioning key

ORACLE REAL-WORLD PERFORMANCE

ADW: Out of the Box Performance (Parallel)

- The query is executed by a session connected to the Medium service
  - SQL statements executed using the Medium service run with DoP 4

PX running time: 1 seconds (out-of-the-box)



ADW: Out of the Box Performance (Parallel)

Plan Statistics | Parallel | SQL Text | Activity | Optimizer Environment | Outline

Plan Hash Value 523409758 Plan Size 75 Navigate to

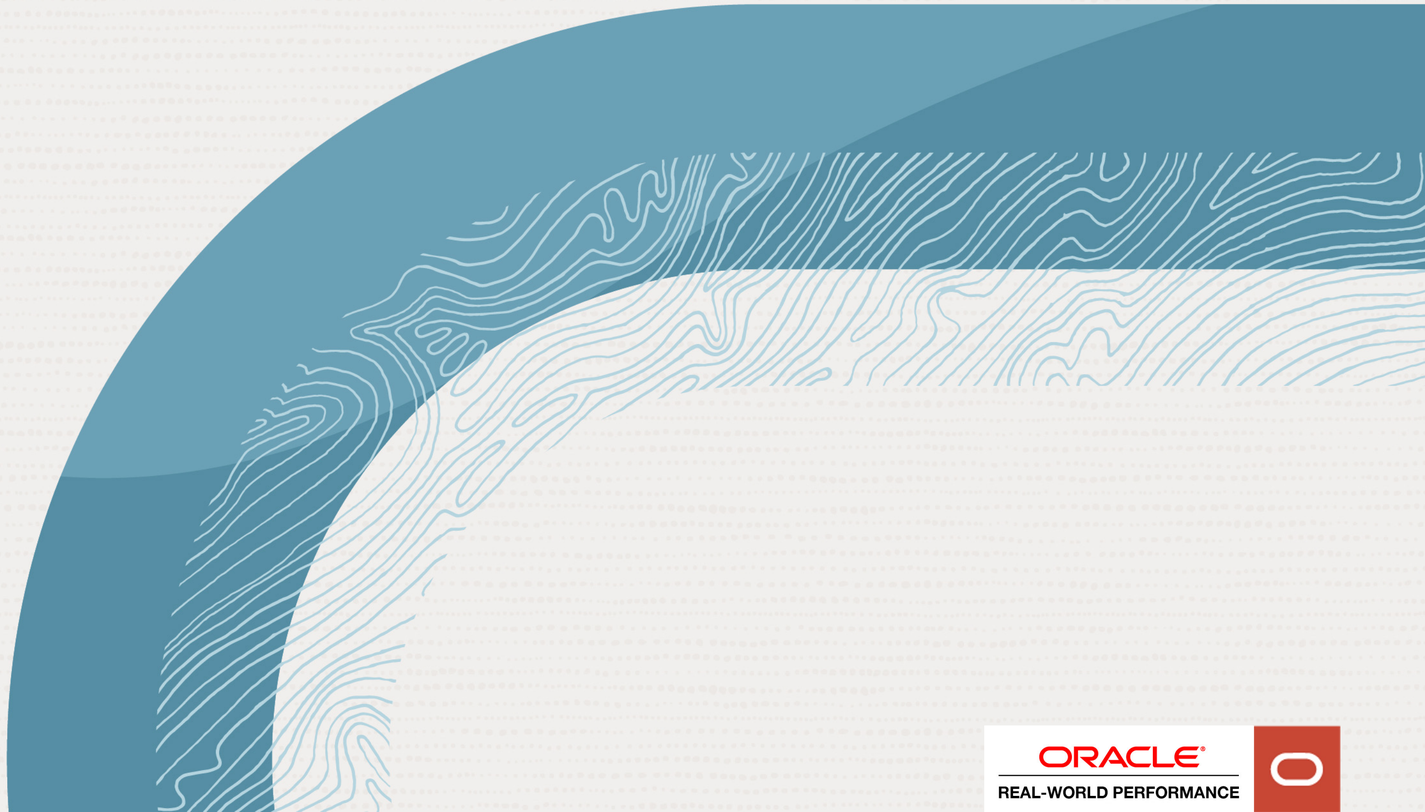
Operation	Object	Informatic	Lin			st. Ro	
PX COORDINATOR			22				
PX SEND QC (RANDOM)	:TQ30001		23				
HASH GROUP BY			24				
PX RECEIVE			25				
PX SEND HASH	:TQ30000		26				
KEY VECTOR CREATE BUFFERED	:KV0002		27				
VIEW (SEL$C9F9BE95)	VW_VTN_63105048		28				
HASH JOIN (SEL$C9F9BE95)			29		4	6	106 K
JOIN FILTER CREATE	:BF0000		30		4		2,400
TABLE ACCESS STORAGE FULL (SEL$C9F	PART_GROUP		31		4		2,400
JOIN FILTER USE	:BF0000		32		4		177 K
PX BLOCK ITERATOR			33		4		177 K
TABLE ACCESS STORAGE FULL (SEL$C9	PART		34		17		177 K
LOAD AS SELECT (SEL$1D319852)	SYS_TEMP_0FD9EFFCB_2B216ACB		35		1		2
PX COORDINATOR			36		9		4
PX SEND QC (RANDOM)	:TQ40001		37		4	25	4
HASH GROUP BY			38		4	25	4
PX RECEIVE			39		4	25	8
PX SEND HASH	:TQ40000		40		4	25	8
KEY VECTOR CREATE BUFFERED	:KV0003		41		4	25	8
PX BLOCK ITERATOR			42		4	2,862	2,862
TABLE ACCESS STORAGE FULL (SEL$1D3198	SUPPLIER		43		2	2,862	2,862
PX COORDINATOR			44		9		48
PX SEND QC (ORDER)	:TQ50004		45		4	213	48
SORT ORDER BY			46		4	213	48
PX RECEIVE			47		4	275	48
PX SEND RANGE	:TQ50003		48		4	275	48
HASH JOIN BUFFERED			49		4	275	48
PX RECEIVE			50		4	1	4
PX SEND BROADCAST	:TQ50000		51		4	1	4
PX BLOCK ITERATOR			52		4	1	1
TABLE ACCESS STORAGE FULL (SEL$4578	SYS_TEMP_0FD9EFFC8_2B216ACB		53		1	1	1

The execution plan is different and uses In-Memory Aggregation (Vector Transformation)
In ADW IMA is enabled and can take advantage of the data stored in columnar store that resides on flash disks

Cardinality estimate for SUPPLIER is now better because dynamic statistics kicked in automatically for every parallel query.

… and that's pretty much it

Summary



Summary

Get the fundamentals right and everything else falls into place

Autonomous Database can help address some problems:

- Augments bad/stale stats
- Dynamic statistics may correct some cardinality estimates (do not solely rely on dynamic statistics)
- Stats quality (every column has a histogram)
- Use of features like IMA, Columnar Cache in Flash is available

These features are enabled by default and kick-in automatically.

Where Autonomous Database cannot help:

- Suboptimal schema design
  - Wrong data types on join keys
  - Lack of constraints
  - Sub-optimal partitioning strategy