ORACLE

A Day of Real-World Performance

Andrew Holdsworth, Tom Kyte, Graham Wood

ORACLE

ORACLE
REAL-WORLD PERFORMANCE

Why is My SQL Slow ?

ORACLE
ORACLE REAL-WORLD PERFORMANCE

Data Warehouse Death Spiral

- HW CPU Sizing 10X
  - Sized like an OLTP System
- I/O Sizing 10X
  - Sized by Space requirements
  - Cannot use Parallel Query
- Using the the incorrect Query Optimization Techniques 10X
  - Over Indexed Database
  - Data Loads and ETL running to Slow
- System Over loaded to Make the CPU look Busy
  - 100s of Concurrent Queries taking Hours to Execute

ORACLE

ORACLE REAL-WORLD PERFORMANCE

Extreme Data Warehouse Workloads

Defined by:

- Analytics / BI queries
- Process large numbers of rows
- Append-only
- Resource intensive
  - Parallel Processing Required
  - Recruit all Available HW for a single task

ORACLE

ORACLE REAL-WORLD PERFORMANCE

Data Loading
Anatomy of an External Table

```
create table FAST_LOAD
(
column definition list ...
)
organization external
(type oracle_loader
default directory SPEEDY_FILESYSTEM
preprocessor exec_file_dir:'zcat.sh'
characterset 'ZHS16GBK'
badfile ERROR_DUMP:'FAST_LOAD.bad'
logfile ERROR_DUMP:'FAST_LOAD.log'
(
file column mapping list ...
)
location
(file_1.gz, file_2.gz, file_3.gz, file_4.gz )
reject limit 1000
parallel 4
/
```

External Table Definition

Reference the Mount Point

Uncompress the data using a secure wrapper

The Characterset must match the Characterset of the Files

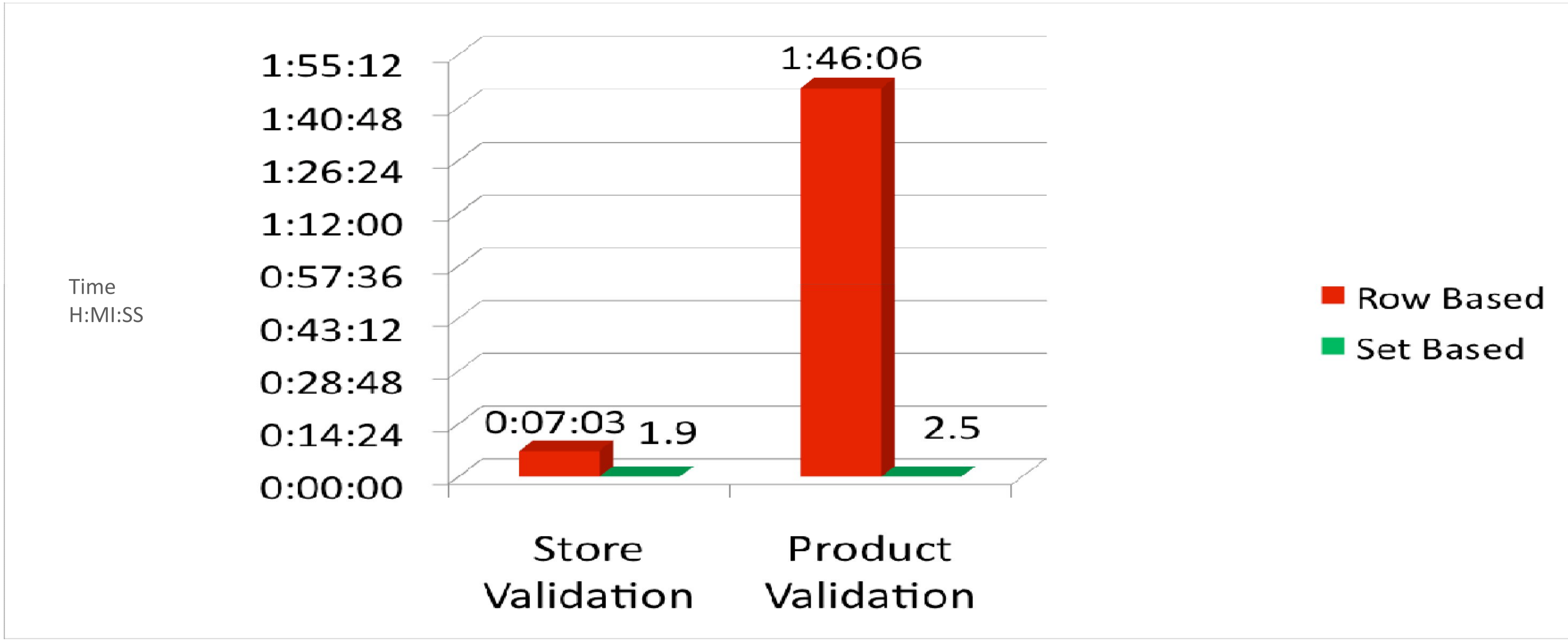
Note Compressed Files

Parallel should match or be less than the number of Files

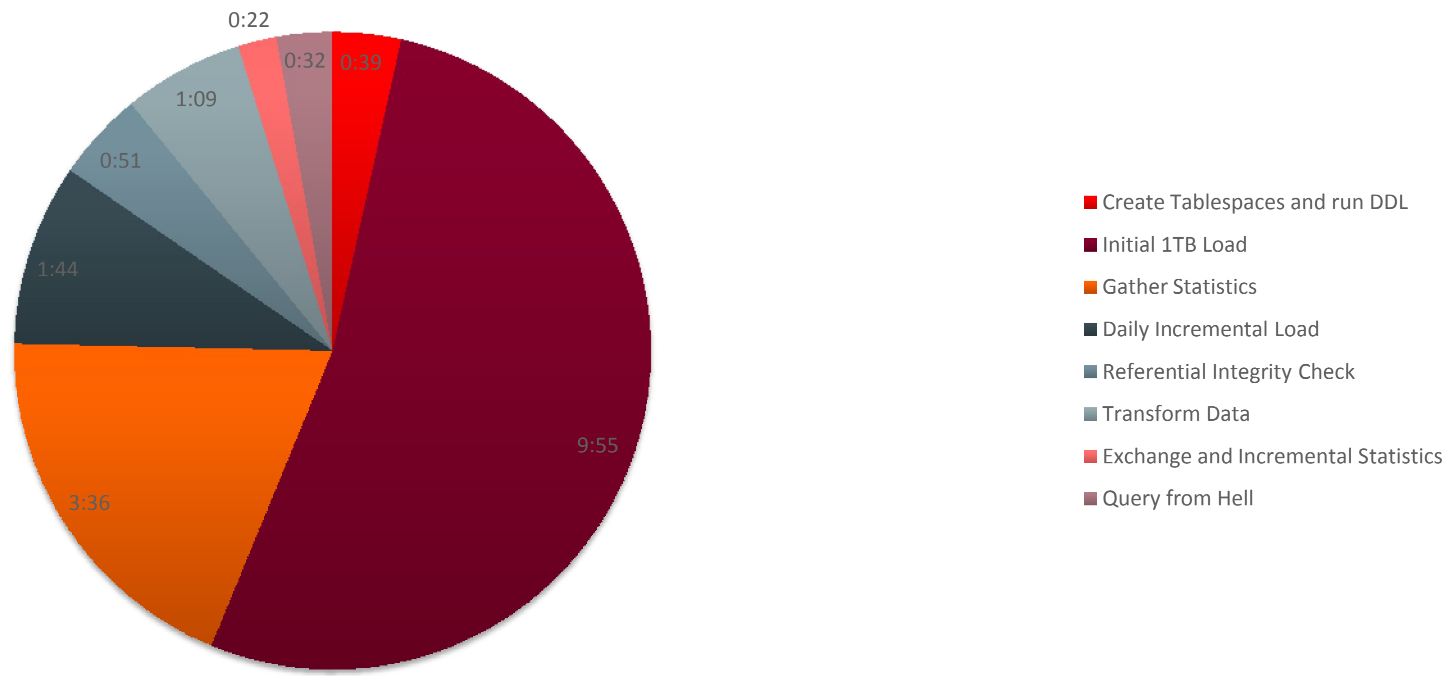
ORACLE

ORACLE REAL-WORLD PERFORMANCE

Validation Example
Set based processing vs. row by row



1 Terabyte Loaded and Ready To Go In 20 Minutes



Requirements for Interactive Performance for DW Query

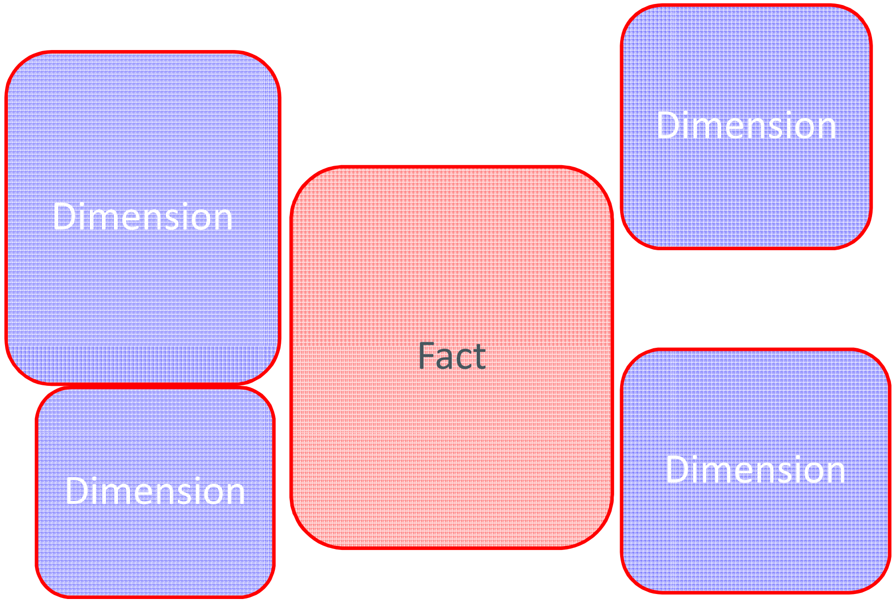
Business Goals

- Analytics at the Speed of Thought
- Predictable Response Times
- No runaway queries
- Most frequent implementation is Star/Snowflake or Dimensional Schema

ORACLE
ORACLE REAL-WORLD PERFORMANCE

Why a Dimensional Schema?

- Dimensional schemas are schemas in which data is organized into *facts, dimensions*
- "Facts" represent events, such as sales, logins, orders, etc.
- Dimensions contain reference information about facts
- Fact tables are denormalized tables that store data for multiple dimensions
- Provides ability to retrieve all "interesting" detailed information from a single table with only joins to smaller dimension tables

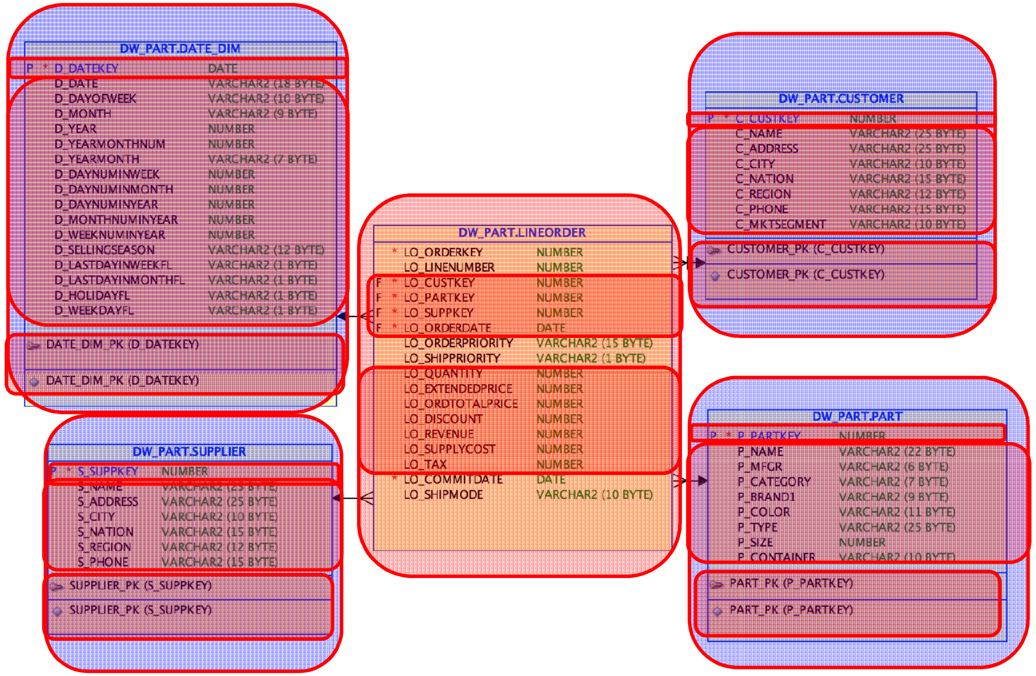


ORACLE

ORACLE REAL-WORLD PERFORMANCE

What is a Dimensional Schema?

- Dimensional schemas are either ***star schemas*** or ***snowflake schemas***
- Schemas consist of fact tables and dimension tables
- The Fact table stores *measures*; i.e., order quantity, net price, etc.
- Dimension tables store *attributes* to describe facts; i.e., month, customer name, etc.
- Tables are joined using *keys*
- Dimensional queries are designed to run on ***dimensional schemas***



ORACLE | ORACLE REAL-WORLD PERFORMANCE

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

ORACLE

ORACLE REAL-WORLD PERFORMANCE

Star Query Race Demo

The Goal

- Access the fact table once
- Filter out all the rows you're **NOT** interested in as early as possible
- i.e. maximize row rejection

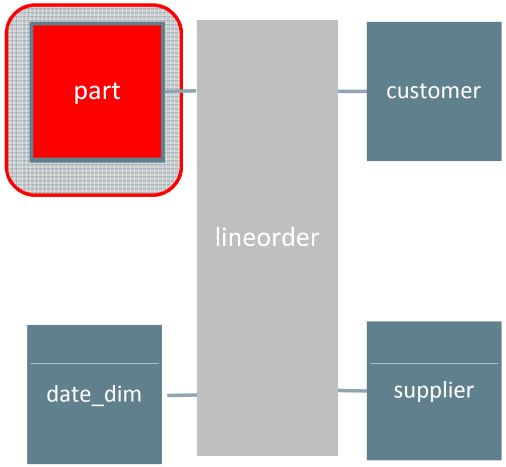
ORACLE

ORACLE REAL-WORLD PERFORMANCE

Star Query Execution Plans

Nested Loops with B*Tree Indexes

1 Build Filters



Operation	Object Name	Predicate information
SELECT STATEMENT		
SORT GROUP BY		
NESTED LOOPS		
NESTED LOOPS		
NESTED LOOPS		
NESTED LOOPS		
TABLE ACCCESS BY LOCAL INDEX ROWID	PART	P_CONTAINER = 'JUMBO PACK'
INDEX RANGE_SCAN	PART_CONTAINER_N	
PARTITION RANGE ALL		LO_ORDERDATE = D_DATEKEY
TABLE ACCESS BY LOCAL INDEX ROWID	LINEORDER	
INDEX RANGE_SCAN	LO_PART_N	
TABLE ACCESS BY INDEX ROWID	DATE_DIM	D_YEAR IN (1993, 1994, 1995)
INDEX UNIQUE SCAN	DATE_DIM_PK	LO_ORDERDATE = D_DATEKEY
TABLE ACCESS BY INDEX ROWID	SUPPLIER	
INDEX UNIQUE SCAN	SUPPLIER_PK	LO_SUPPKEY = S_SUPPKEY

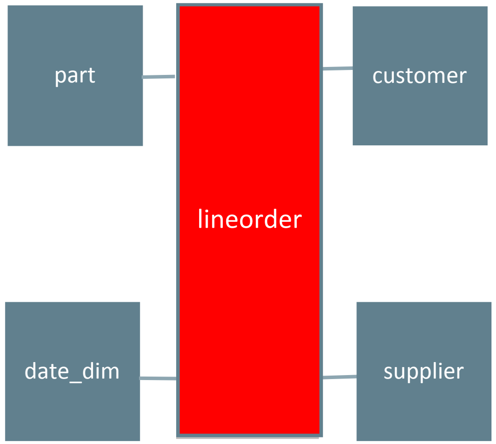
```
SELECT          d_sellingseason, p_category, s_region,
                sum(lo_extendedprice)
                FROM    lineorder
                    JOIN    customer        ON lo_custkey = c_custkey
                    JOIN    date_dim        ON lo_orderdate = d_datekey
                    JOIN    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
```

ORACLE

ORACLE
REAL-WORLD PERFORMANCE

Nested Loops with B*Tree Indexes

2. Extract Rows from the Fact table



Operation	Object Name	Predicate information
SELECT STATEMENT		
SORT GROUP BY		
NESTED LOOPS		
NESTED LOOPS		
NESTED LOOPS		
NESTED LOOPS		
TABLE ACCCESS BY LOCAL INDEX ROWID	PART	P_CONTAINER = 'JUMBO PACK'
INDEX RANGE_SCAN	PART_CONTAINER_N	
PARTITION RANGE ALL		LO_ORDERDATE = D_DATEKEY
TABLE ACCESS BY LOCAL INDEX ROWID	LINEORDER	
INDEX RANGE_SCAN	LO_PART_N	
TABLE ACCESS BY INDEX ROWID	DATE_DIM	D_YEAR IN (1993, 1994, 1995)
INDEX UNIQUE SCAN	DATE_DIM_PK	LO_ORDERDATE = D_DATEKEY
TABLE ACCESS BY INDEX ROWID	SUPPLIER	
INDEX UNIQUE SCAN	SUPPLIER_PK	LO_SUPPKEY = S_SUPPKEY

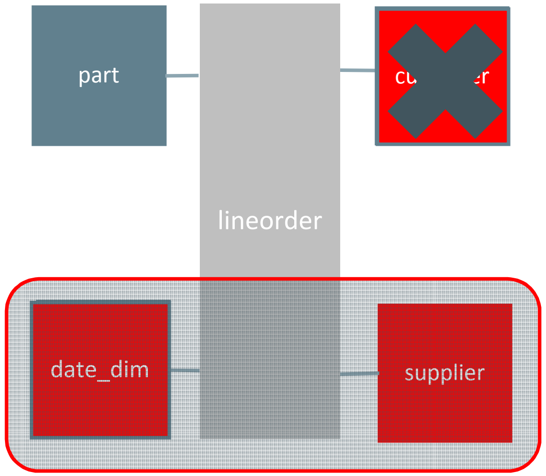
```
SELECT        d_sellingseason, p_category, s_region,
              sum(lo_extendedprice)
              FROM    lineorder
                JOIN    customer         ON lo_custkey = c_custkey
                JOIN    date_dim         ON lo_orderdate = d_datekey
                JOIN    part             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
```

ORACLE

ORACLE REAL-WORLD PERFORMANCE

Nested Loops with B*Tree Indexes

3. Join to Dimensions to Project Additional Columns



Operation	Object Name	Predicate information
SELECT STATEMENT		
SORT GROUP BY		
NESTED LOOPS		
NESTED LOOPS		
NESTED LOOPS		
NESTED LOOPS		
TABLE ACCCESS BY LOCAL INDEX ROWID	PART	P_CONTAINER = 'JUMBO PACK'
INDEX RANGE_SCAN	PART_CONTAINER_N	
PARTITION RANGE ALL		LO_ORDERDATE = D_DATEKEY
TABLE ACCESS BY LOCAL INDEX ROWID	LINEORDER	
INDEX RANGE_SCAN	LO_PART_N	
TABLE ACCESS BY INDEX ROWID	DATE_DIM	D_YEAR IN (1993, 1994, 1995)
INDEX UNIQUE SCAN	DATE_DIM_PK	LO_ORDERDATE = D_DATEKEY
TABLE ACCESS BY INDEX ROWID	SUPPLIER	
INDEX UNIQUE SCAN	SUPPLIER_PK	LO_SUPPKEY = S_SUPPKEY

```
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
```

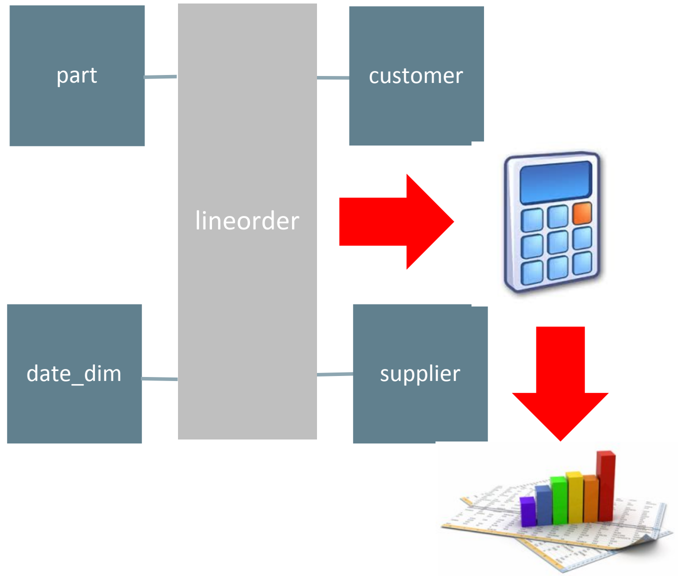
ORACLE

ORACLE REAL-WORLD PERFORMANCE

Nested Loops with B*Tree Indexes

4. Aggregate/Sort Row and Return Results

Operation	Object Name	Predicate information
SELECT STATEMENT		
SORT GROUP BY		
NESTED LOOPS		
NESTED LOOPS		
NESTED LOOPS		
NESTED LOOPS		
TABLE ACCCESS BY LOCAL INDEX ROWID	PART	P_CONTAINER = 'JUMBO PACK'
INDEX RANGE_SCAN	PART_CONTAINER_N	
PARTITION RANGE ALL		LO_ORDERDATE = D_DATEKEY
TABLE ACCESS BY LOCAL INDEX ROWID	LINEORDER	
INDEX RANGE_SCAN	LO_PART_N	
TABLE ACCESS BY INDEX ROWID	DATE_DIM	D_YEAR IN (1993, 1994, 1995)
INDEX UNIQUE SCAN	DATE_DIM_PK	LO_ORDERDATE = D_DATEKEY
TABLE ACCESS BY INDEX ROWID	SUPPLIER	
INDEX UNIQUE SCAN	SUPPLIER_PK	LO_SUPPKEY = S_SUPPKEY

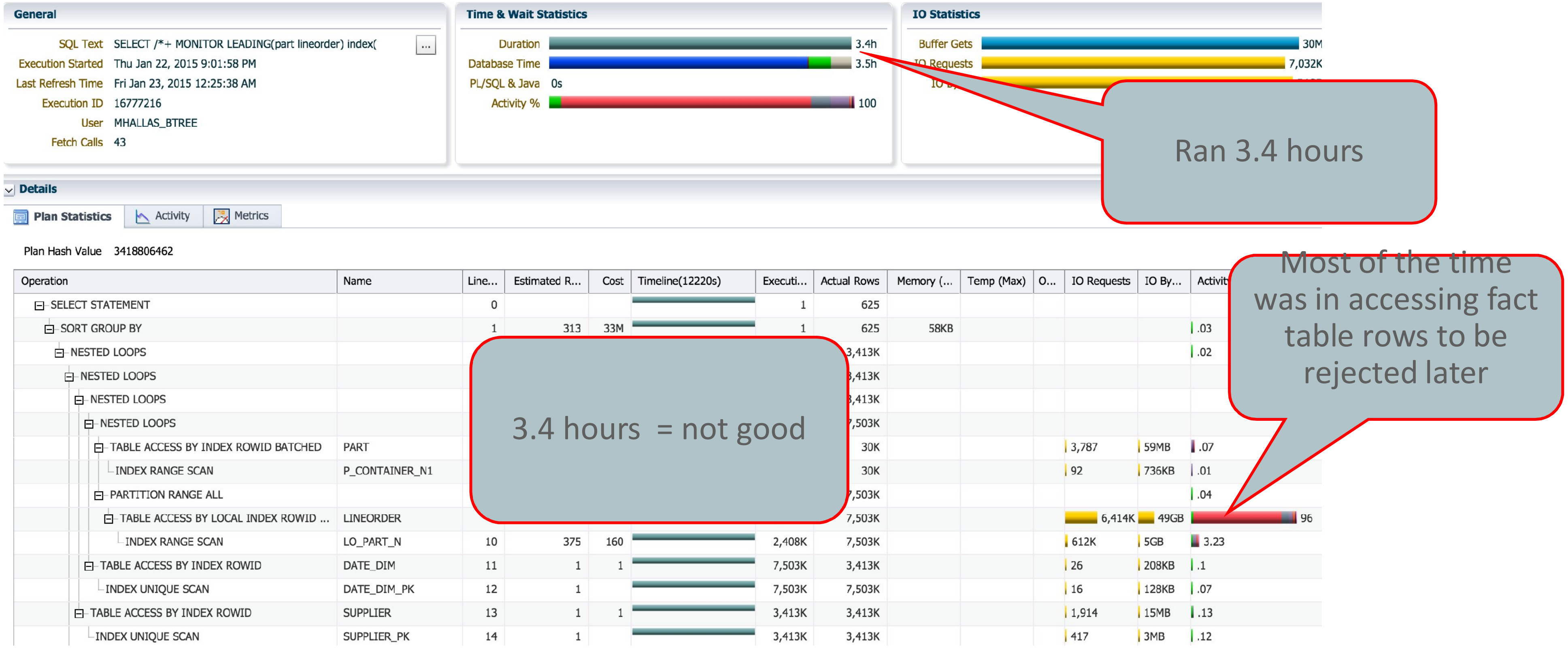



```
SELECT      d_sellingseason, p_category, s_region,
            sum(lo_extendedprice)
            FROM    lineorder
              JOIN    customer       ON lo_custkey = c_custkey
              JOIN    date_dim       ON lo_orderdate = d_datekey
              JOIN    part           ON lo_partkey = p_partkey
              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
```

REAL-WORLD PERFORMANCE

Nested Loops with B*Tree Indexes

Things to Think About



ORACLE

ORACLE
REAL-WORLD PERFORMANCE

B*Tree Index with Nested Loops Joins Summary

Technique	Primary Fact Table Access Method	Requirements	Pros	Cons
B*Tree Indexes with NL Joins	• B*Tree index access • Nested Loops joins	• Indexes on fact table	Decent performance if number of rows is very small and all data accessed is satisfied from memory	Algorithmically weak; can’t get fact table rows fast enough

ORACLE

ORACLE
REAL-WORLD PERFORMANCE

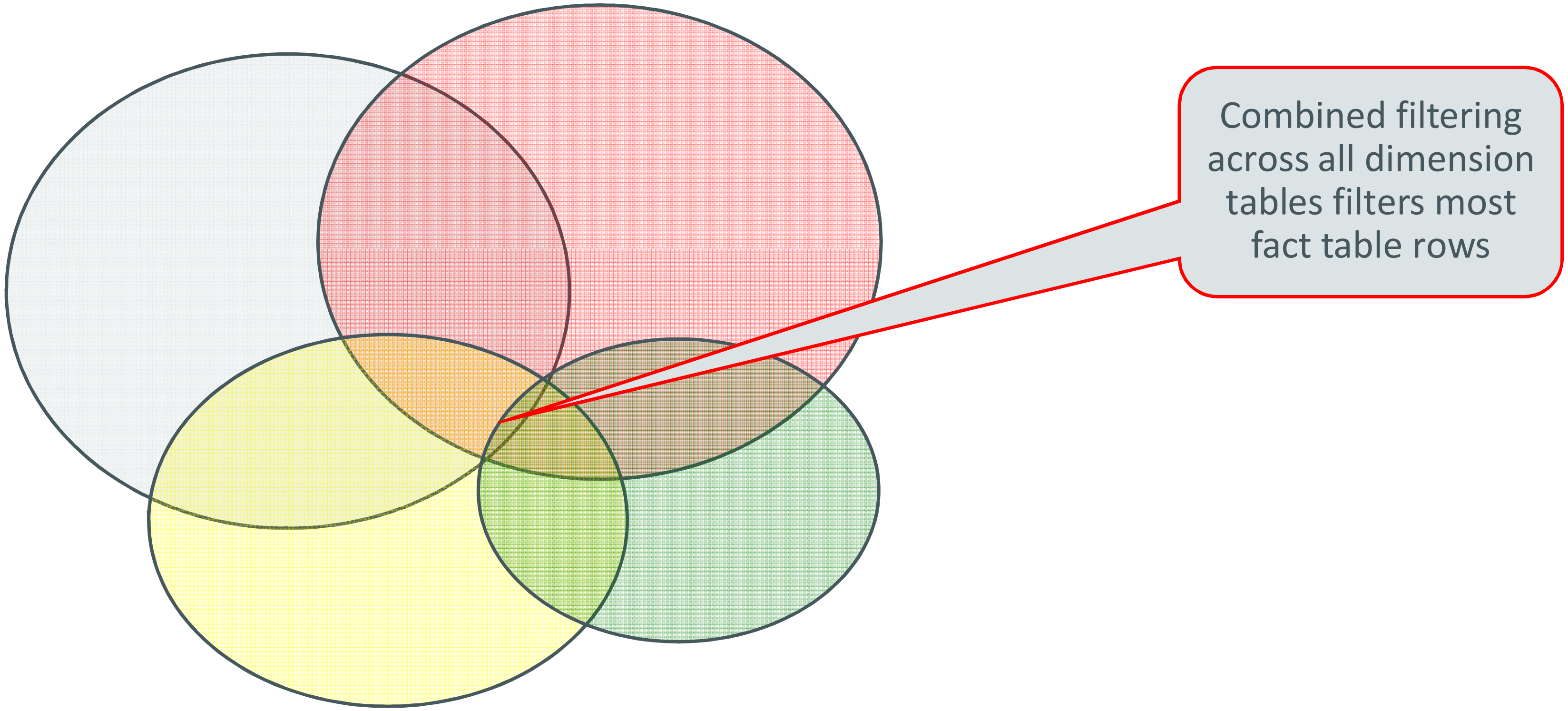
Star Transformation with Bit Mapped Indexes

```
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
```

```
SELECT lo_orderdate, lo_partkey, lo_suppkey,
       lo_extendedprice
       FROM lineorder
       WHERE lo_orderdate IN
            (SELECT d_datekey
                    FROM  date_dim
                    WHERE d_year     IN ( 1993,1994,1995 ))
       AND   lo_partkey IN
            (SELECT p_partkey
                    FROM part
                    WHERE p_container IN ('JUMBO PACK' ))
```

Star Transformation with Bitmap Indexes

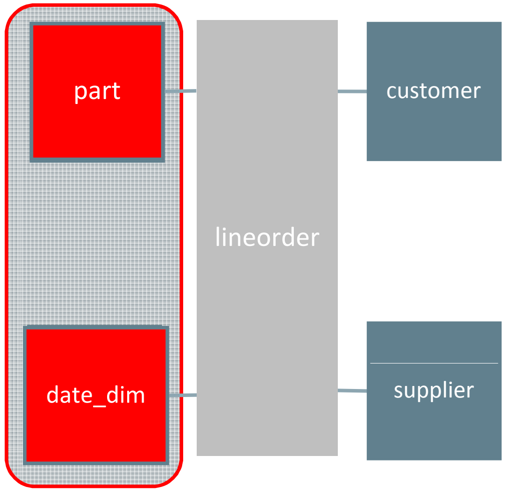
When Bitmap Indexes are Effective



Execution Method for Star Transformation

1. Build Filters

Operation	Object Name	Predicate information
SELECT STATEMENT		
TEMP TABLE TRANSFORMATION		
LOAD AS SELECT	SYS_TEMP_0FD9FCA09_7D1FC714	
TABLE ACCESS FULL	DATE_DIM	D_YEAR IN (1993, 1994, 1995)
LOAD AS SELECT	SYS_TEMP_0FD9FCA0A_7D1FC714	
TABLE ACCESS FULL	PART	P_CONTAINER = 'JUMBO PACK'
SORT GROUP BY		
HASH JOIN		LO_PARTKEY = P_PARTKEY
TABLE ACCESS FULL	SYS_TEMP_0FD9FCA0A_7D1FC714	
HASH JOIN		LO_ORDERDATE = D_DATEKEY
TABLE ACCESS FULL	SYS_TEMP_0FD9FCA09_7D1FC714	
HASH JOIN		LO_SUPPKEY = S_SUPPKEY
TABLE ACCESS FULL	SUPPLIER	
VIEW	VW_ST_F981A0CC	
NESTED LOOPS		
PARTITION RANGE SUBQUERY		
BITMAP CONVERSION TO ROWIDS		
BITMAP AND		
BITMAP MERGE		
BITMAP KEY ITERATION		
BUFFER SORT		
TABLE ACCESS FULL	SYS_TEMP_0FD9FCA09_7D1FC714	
BITMAP INDEX RANGE SCAN	LO_DATE_B	LO_ORDERDATE = D_DATEKEY
BITMAP MERGE		
BITMAP KEY ITERATION		
BUFFER SORT		
TABLE ACCESS FULL	SYS_TEMP_0FD9FCA0A_7D1FC714	
BITMAP INDEX RANGE SCAN	LO_PART_B	LO_PARTKEY = P_PARTKEY
TABLE ACCESS BY USER ROWID	LINEORDER	




```
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
```

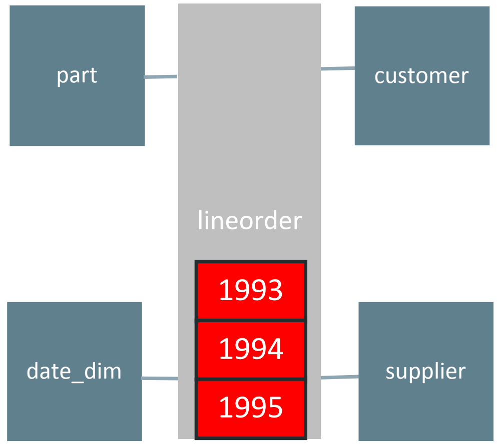
ORACLE

ORACLE REAL-WORLD PERFORMANCE

Execution Method for Star Transformation

2. Extract Rows from the Fact table

Operation	Object Name	Predicate information
SELECT STATEMENT		
TEMP TABLE TRANSFORMATION		
LOAD AS SELECT	SYS_TEMP_0FD9FCA09_7D1FC714	
TABLE ACCESS FULL	DATE_DIM	D_YEAR IN (1993, 1994, 1995)
LOAD AS SELECT	SYS_TEMP_0FD9FCA0A_7D1FC714	
TABLE ACCESS FULL	PART	P_CONTAINER = 'JUMBO PACK'
SORT GROUP BY		
HASH JOIN		LO_PARTKEY = P_PARTKEY
TABLE ACCESS FULL	SYS_TEMP_0FD9FCA0A_7D1FC714	
HASH JOIN		LO_ORDERDATE = D_DATEKEY
TABLE ACCESS FULL	SYS_TEMP_0FD9FCA09_7D1FC714	
HASH JOIN		LO_SUPPKEY = S_SUPPKEY
TABLE ACCESS FULL	SUPPLIER	
VIEW	VW_ST_F981A0CC	
NESTED LOOPS		
PARTITION RANGE SUBQUERY		
BITMAP CONVERSION TO ROWIDS		
BITMAP AND		
BITMAP MERGE		
BITMAP KEY ITERATION		
BUFFER SORT		
TABLE ACCESS FULL	SYS_TEMP_0FD9FCA09_7D1FC714	
BITMAP INDEX RANGE SCAN	LO_DATE_B	LO_ORDERDATE = D_DATEKEY
BITMAP MERGE		
BITMAP KEY ITERATION		
BUFFER SORT		
TABLE ACCESS FULL	SYS_TEMP_0FD9FCA0A_7D1FC714	
BITMAP INDEX RANGE SCAN	LO_PART_B	LO_PARTKEY = P_PARTKEY
TABLE ACCESS BY USER ROWID	LINEORDER	




```
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
```

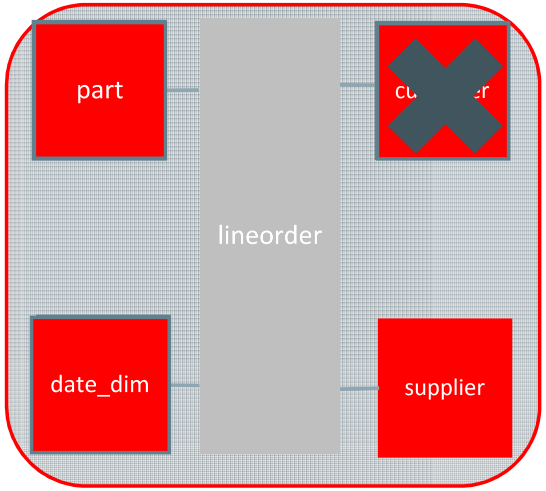
ORACLE

ORACLE
REAL-WORLD PERFORMANCE

Execution Method for Star Transformation

3. Join Back to Dimensions to Project Additional Columns

Operation	Object Name	Predicate information
SELECT STATEMENT		
TEMP TABLE TRANSFORMATION		
LOAD AS SELECT	SYS_TEMP_0FD9FCA09_7D1FC714	
TABLE ACCESS FULL	DATE_DIM	D_YEAR IN (1993, 1994, 1995)
LOAD AS SELECT	SYS_TEMP_0FD9FCA0A_7D1FC714	
TABLE ACCESS FULL	PART	P_CONTAINER = 'JUMBO PACK'
SORT GROUP BY		
HASH JOIN		LO_PARTKEY = P_PARTKEY
TABLE ACCESS FULL	SYS_TEMP_0FD9FCA0A_7D1FC714	
HASH JOIN		LO_ORDERDATE = D_DATEKEY
TABLE ACCESS FULL	SYS_TEMP_0FD9FCA09_7D1FC714	
HASH JOIN		LO_SUPPKEY = S_SUPPKEY
TABLE ACCESS FULL	SUPPLIER	
VIEW	VW_ST_F981A0CC	
NESTED LOOPS		
PARTITION RANGE SUBQUERY		
BITMAP CONVERSION TO ROWIDS		
BITMAP AND		
BITMAP MERGE		
BITMAP KEY ITERATION		
BUFFER SORT		
TABLE ACCESS FULL	SYS_TEMP_0FD9FCA09_7D1FC714	
BITMAP INDEX RANGE SCAN	LO_DATE_B	LO_ORDERDATE = D_DATEKEY
BITMAP MERGE		
BITMAP KEY ITERATION		
BUFFER SORT		
TABLE ACCESS FULL	SYS_TEMP_0FD9FCA0A_7D1FC714	
BITMAP INDEX RANGE SCAN	LO_PART_B	LO_PARTKEY = P_PARTKEY
TABLE ACCESS BY USER ROWID	LINEORDER	



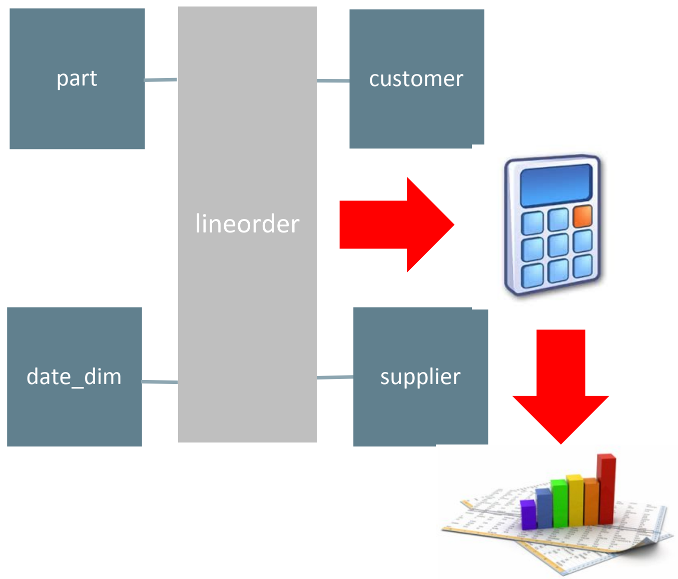

```
SELECT      d_sellingseason, p_category, s_region,
            sum(lo_extendedprice)
            FROM   lineorder
              JOIN   customer         ON lo_custkey = c_custkey
              JOIN   date_dim         ON lo_orderdate = d_datekey
              JOIN   part             ON lo_partkey = 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
```

ORACLE
ORACLE REAL-WORLD PERFORMANCE

Execution Method for Star Transformation

4. Aggregate/Sort Rows and Return Results

Operation	Object Name	Predicate information
SELECT STATEMENT		
TEMP TABLE TRANSFORMATION		
LOAD AS SELECT	SYS_TEMP_0FD9FCA09_7D1FC714	
TABLE ACCESS FULL	DATE_DIM	D_YEAR IN (1993, 1994, 1995)
LOAD AS SELECT	SYS_TEMP_0FD9FCA0A_7D1FC714	
TABLE ACCESS FULL	PART	P_CONTAINER = 'JUMBO PACK'
SORT GROUP BY		
HASH JOIN		LO_PARTKEY = P_PARTKEY
TABLE ACCESS FULL	SYS_TEMP_0FD9FCA0A_7D1FC714	
HASH JOIN		LO_ORDERDATE = D_DATEKEY
TABLE ACCESS FULL	SYS_TEMP_0FD9FCA09_7D1FC714	
HASH JOIN		LO_SUPPKEY = S_SUPPKEY
TABLE ACCESS FULL	SUPPLIER	
VIEW	VW_ST_F981A0CC	
NESTED LOOPS		
PARTITION RANGE SUBQUERY		
BITMAP CONVERSION TO ROWIDS		
BITMAP AND		
BITMAP MERGE		
BITMAP KEY ITERATION		
BUFFER SORT		
TABLE ACCESS FULL	SYS_TEMP_0FD9FCA09_7D1FC714	
BITMAP INDEX RANGE SCAN	LO_DATE_B	LO_ORDERDATE = D_DATEKEY
BITMAP MERGE		
BITMAP KEY ITERATION		
BUFFER SORT		
TABLE ACCESS FULL	SYS_TEMP_0FD9FCA0A_7D1FC714	
BITMAP INDEX RANGE SCAN	LO_PART_B	LO_PARTKEY = P_PARTKEY
TABLE ACCESS BY USER ROWID	LINEORDER	




```
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
```

ORACLE

ORACLE REAL-WORLD PERFORMANCE

Star Transformation
Things to Think About

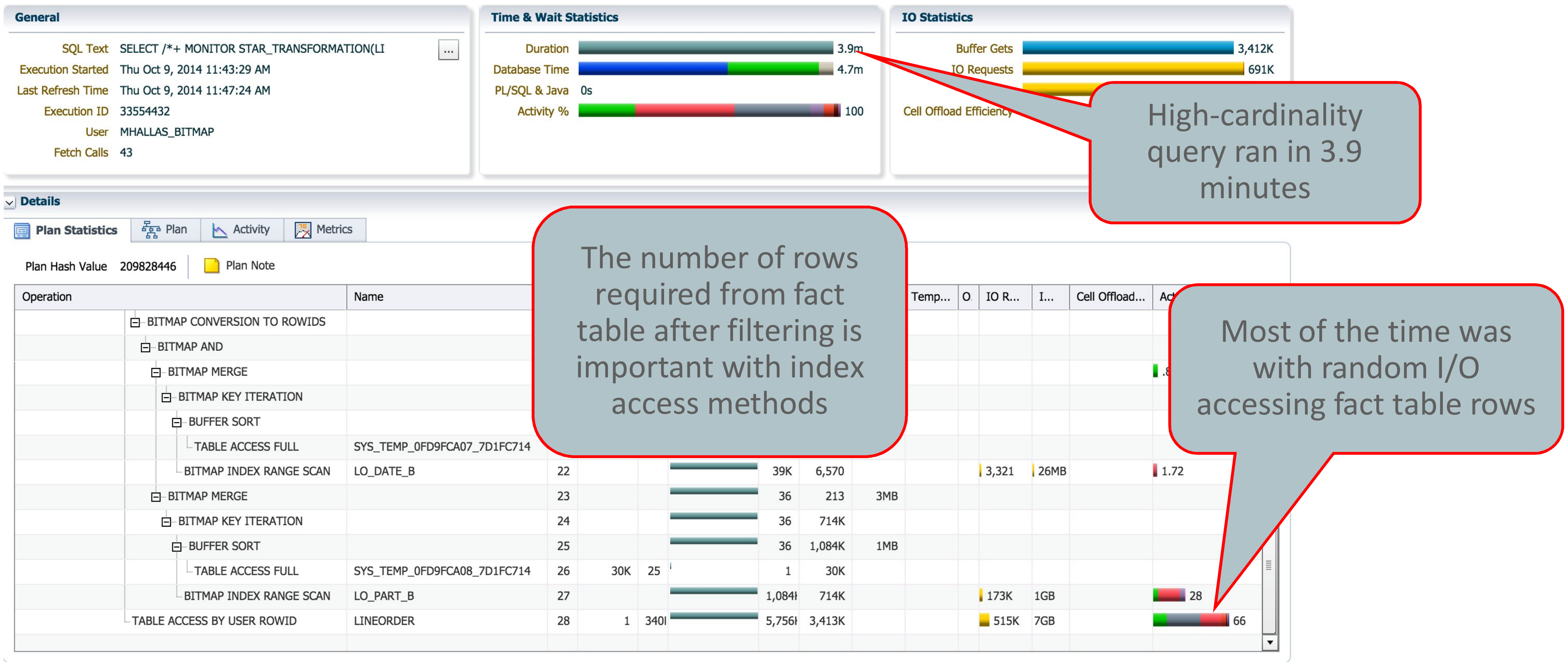
Operation	Object Name	Predicate information
SELECT STATEMENT		
TEMP TABLE TRANSFORMATION		
LOAD AS SELECT	SYS_TEMP_0FD9FCA09_7D1FC714	
TABLE ACCESS FULL	DATE_DIM	D_YEAR IN (1993, 1994, 1995)
LOAD AS SELECT	SYS_TEMP_0FD9FCA0A_7D1FC714	
TABLE ACCESS FULL	PART	P_CONTAINER = 'JUMBO PACK'
SORT GROUP BY		
HASH JOIN		LO_PARTKEY = P_PARTKEY
TABLE ACCESS FULL	SYS_TEMP_0FD9FCA0A_7D1FC714	
HASH JOIN		LO_ORDERDATE = D_DATEKEY
TABLE ACCESS FULL	SYS_TEMP_0FD9FCA09_7D1FC714	
HASH JOIN		LO_SUPPKEY = S_SUPPKEY
TABLE ACCESS FULL	SUPPLIER	
VIEW	VW_ST_F981A0CC	
NESTED LOOPS		
PARTITION RANGE SUBQUERY		
BITMAP CONVERSION TO ROWIDS		
BITMAP AND		
BITMAP MERGE		
BITMAP KEY ITERATION		
BUFFER SORT		
TABLE ACCESS FULL	SYS_TEMP_0FD9FCA09_7D1FC714	
BITMAP INDEX RANGE SCAN	LO_DATE_B	LO_ORDERDATE = D_DATEKEY
BITMAP MERGE		
BITMAP KEY ITERATION		
BUFFER SORT		
TABLE ACCESS FULL	SYS_TEMP_0FD9FCA0A_7D1FC714	
BITMAP INDEX RANGE SCAN	LO_PART_B	LO_PARTKEY = P_PARTKEY
TABLE ACCESS BY USER ROWID	LINEORDER	

- Assume it takes 5ms to do a random IO
- If we need 5 rows from the fact table and they're not in the buffer cache, how long would it take to extract the rows we want?
- What if we need to extract 1,000,000 rows?

ORACLE
ORACLE REAL-WORLD PERFORMANCE

Star Transformation with Bitmap Indexes

Things to Think About



ORACLE

ORACLE REAL-WORLD PERFORMANCE

Star Transformation Summary

Technique	Primary Fact Table Access Method	Requirements	Pros	Cons
B*Tree Indexes with NL Joins	• B*Tree index access • Nested Loops joins	• Indexes on fact table	Decent performance if number of rows is very small and all data accessed is satisfied from memory	Algorithmically weak; can't get fact table rows fast enough
Star transformation	• Rowid from bitmap index • Bitmap merge • Star transformation	• star_transformation_enabled • query_rewrite_integrity • PK/FK constraints • NOT NULL constraints • Bitmap indexes on fact table	Excellent performance if number of rows is small and all data accessed is satisfied from memory	Poor performance if number of rows from fact table is high and requires random I/O

Bloom Filters – before we get into the next part…

- Efficient way to filter data
- Bloom Filters created from dimension tables and applied to fact table during scan
- Utilizes swap join optimization and yields right-deep plans
- Filtered data is pipelined to hash joins

ORACLE

ORACLE REAL-WORLD PERFORMANCE

Bloom Filter

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

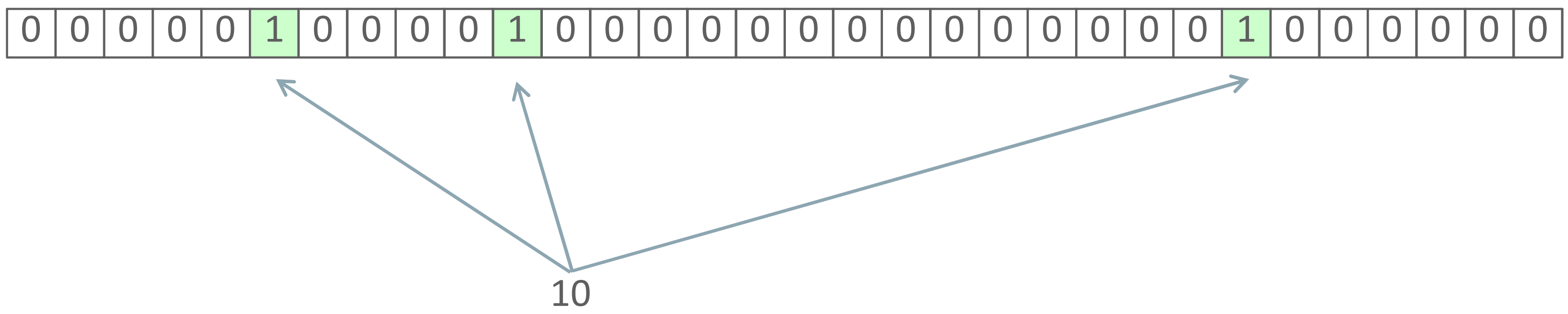
ORACLE

ORACLE
REAL-WORLD PERFORMANCE

Bloom Filter

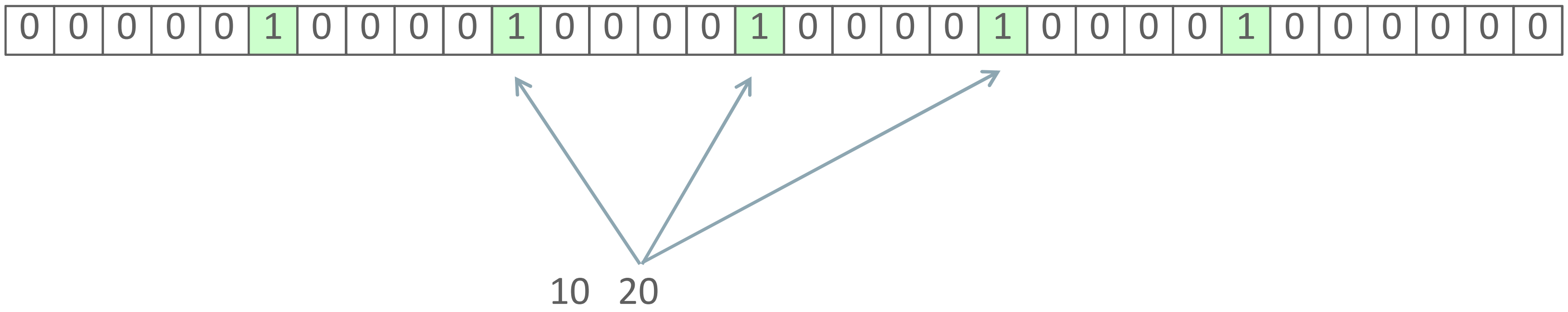
Build

This example uses 3 hash functions



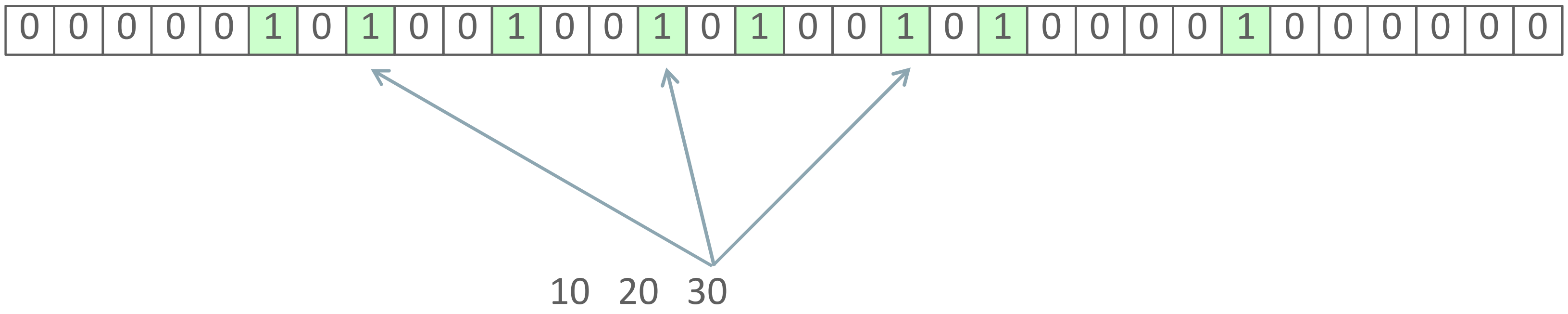
Bloom Filter

Build



Bloom Filter

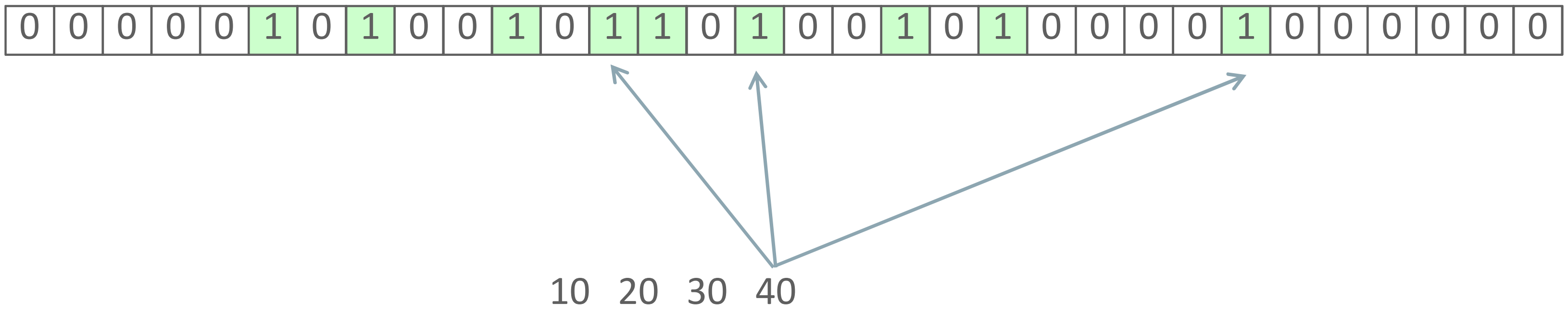
Build



ORACLE REAL-WORLD PERFORMANCE

Bloom Filter

Build



ORACLE

ORACLE REAL-WORLD PERFORMANCE

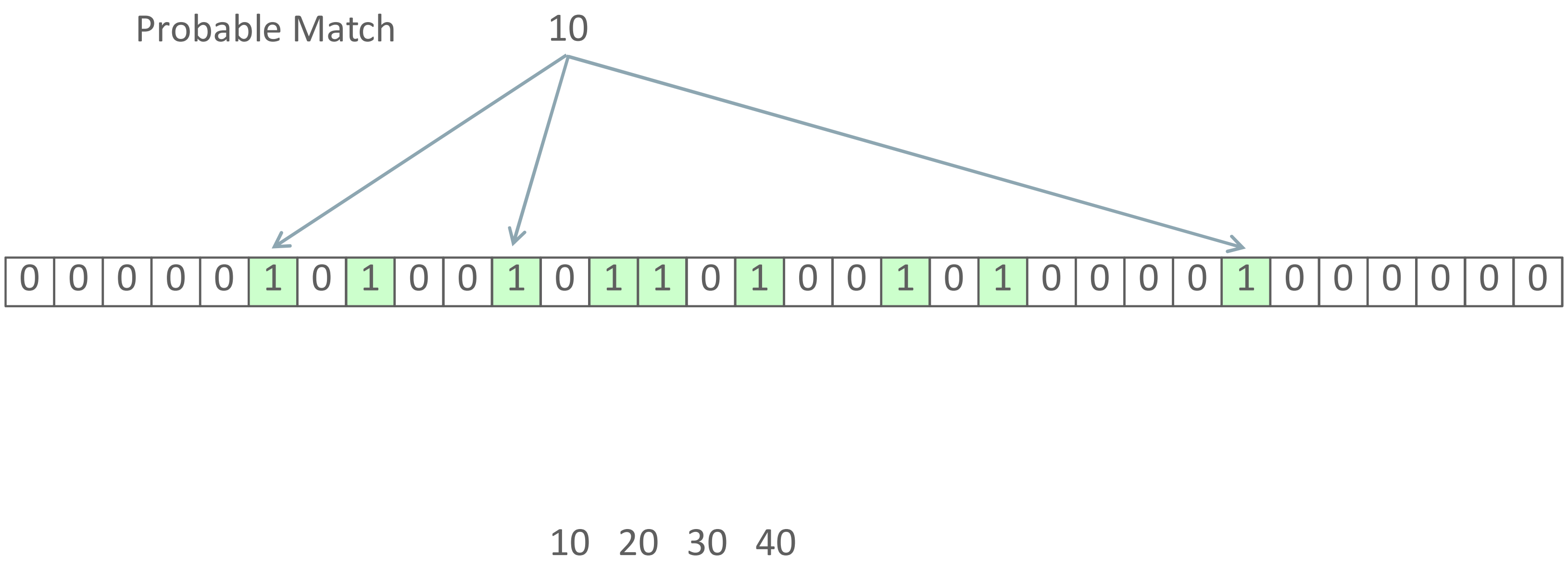
Bloom Filter

Bloom Filter passed Down

0	0	0	0	0	1	0	1	0	0	1	0	1	1	0	1	0	0	1	0	1	0	0	0	0	1	0	0	0	0	0	0

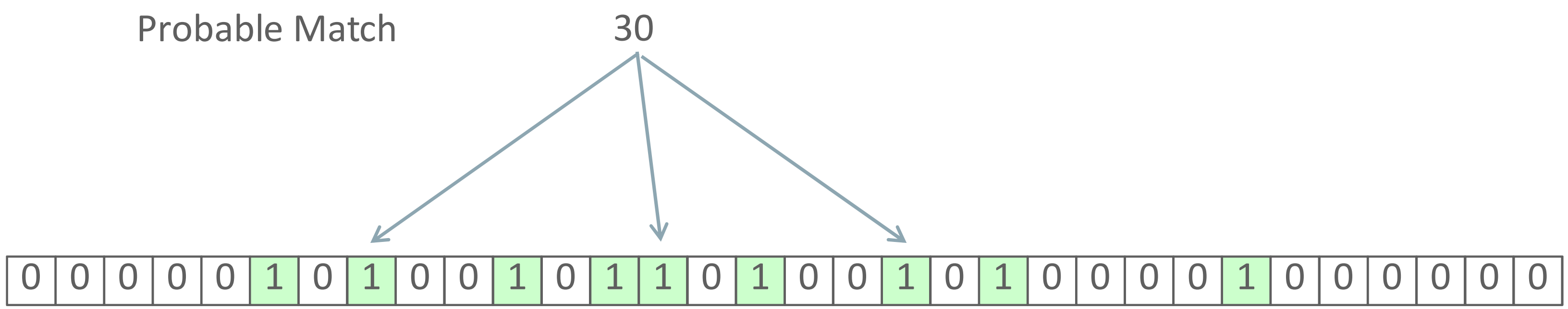
Bloom Filter

Test



Bloom Filter

Test



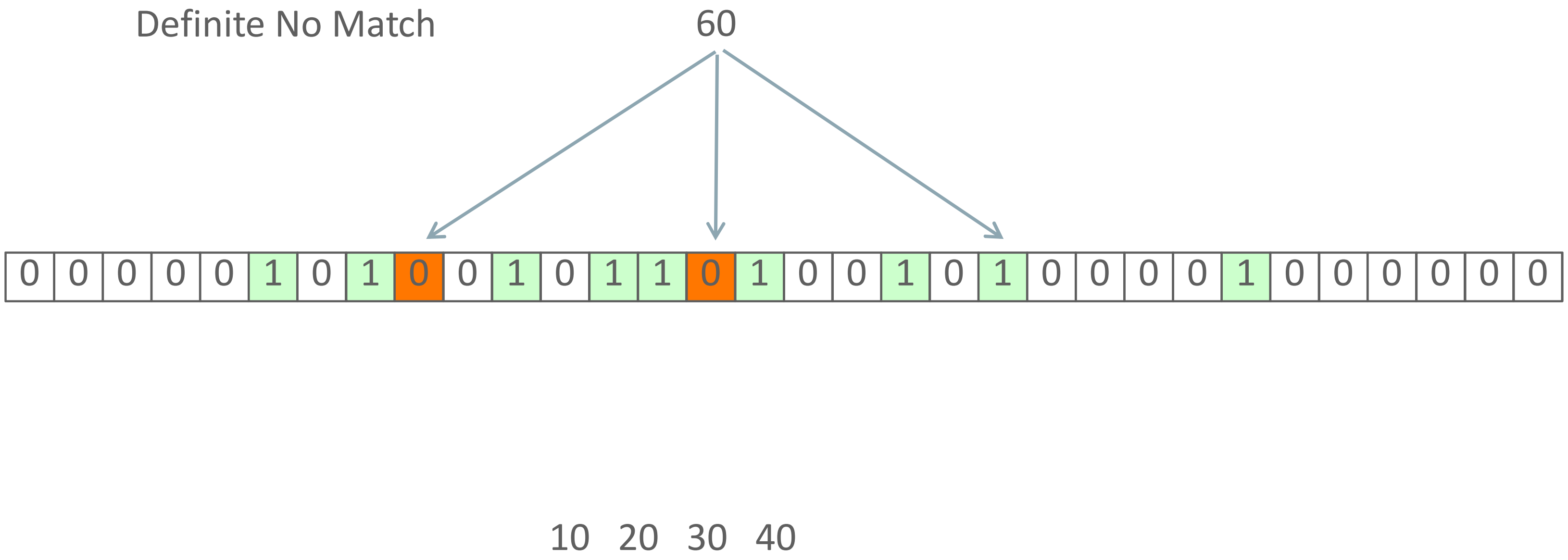
10 20 30 40

ORACLE

ORACLE REAL-WORLD PERFORMANCE

Bloom Filter

Test



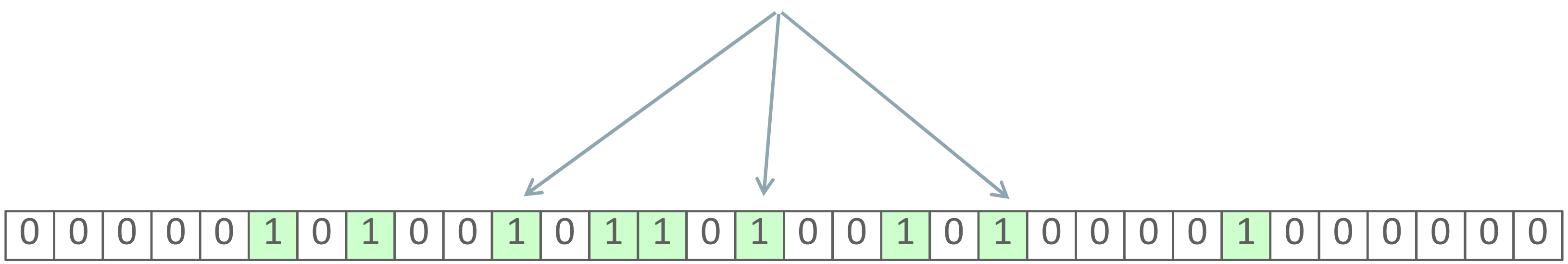
ORACLE

ORACLE
REAL-WORLD PERFORMANCE

Bloom Filter

Test

Probable Match



10 20 30 40

In this case, the match is in fact a false positive

Bloom Filter

Identifying in Plans

Bloom Filter create

Bloom Filter use

```
-------------------------------------------------------------------------------
| Id  | Operation                               | Name      | Rows  | Bytes |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                        |           |    63 |  7371 |
|   1 |  PX COORDINATOR                         |           |       |       |
|   2 |   PX SEND QC (ORDER)                    | :TQ10003  |    63 |  7371 |
|   3 |    SORT GROUP BY                        |           |    63 |  7371 |
|   4 |     PX RECEIVE                          |           |    63 |  7371 |
|   5 |      PX SEND RANGE                      | :TQ10002  |    63 |  7371 |
|   6 |       HASH GROUP BY                     |           |    63 |  7371 |
|*  7 |        HASH JOIN                        |           |   430 | 50310 |
|   8 |         JOIN FILTER CREATE              | :BF0000   |  4013 |  133K|
|   9 |          PX RECEIVE                     |           |  4013 |  133K|
|  10 |           PX SEND BROADCAST             | :TQ10000  |  4013 |  133K|
|  11 |            PX BLOCK ITERATOR            |           |  4013 |  133K|
|* 12 |             TABLE ACCESS INMEMORY FULL  | SUPPLIER  |  4013 |  133K|
|* 13 |         HASH JOIN                       |           | 10589 |  858K|
|  14 |          JOIN FILTER CREATE             | :BF0001   |    31 |  1023 |
|* 15 |           TABLE ACCESS INMEMORY FULL    | DATE_DIM  |    31 |  1023 |
|* 16 |          HASH JOIN                      |           |  845K|   40M|
|  17 |           JOIN FILTER CREATE            | :BF0002   |  2332 | 65296 |
|  18 |            PX RECEIVE                   |           |  2332 | 65296 |
|  19 |             PX SEND BROADCAST           | :TQ10001  |  2332 | 65296 |
|  20 |              PX BLOCK ITERATOR          |           |  2332 | 65296 |
|* 21 |               TABLE ACCESS INMEMORY FULL| PART      |  2332 | 65296 |
|  22 |           JOIN FILTER USE               | :BF0000   |  300M|  6294M|
|  23 |            JOIN FILTER USE              | :BF0001   |  300M|  6294M|
|  24 |             JOIN FILTER USE             | :BF0002   |  300M|  6294M|
|  25 |              PX BLOCK ITERATOR          |           |  300M|  6294M|
|* 26 |               TABLE ACCESS INMEMORY FULL| LINEORDER |  300M|  6294M|
-------------------------------------------------------------------------------
```

ORACLE

ORACLE REAL-WORLD PERFORMANCE

Choose Your Execution Method

Table Scans with Intelligent Filtering

- Queries extract many rows from Fact table
- Database size large

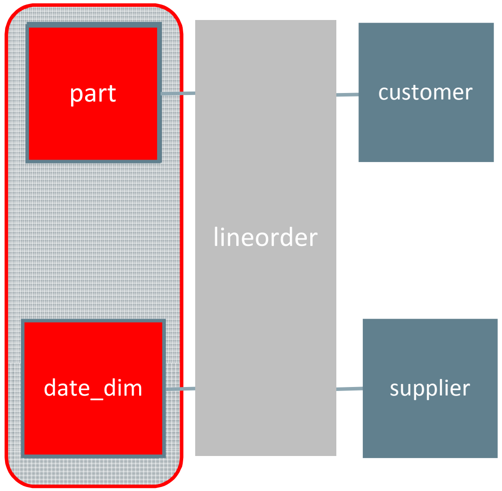
Exadata or Oracle Database In-Memory

ORACLE
ORACLE
REAL-WORLD PERFORMANCE

Intelligent Full Scans

1. Build Bloom Filters and Hash Tables from Dimensions

Operation	Object Name	Predicate information
SELECT STATEMENT		
SORT GROUP BY		
HASH JOIN		LO_SUPPKEY = S_SUPPKEY
TABLE ACCESS STORAGE FULL	SUPPLIER	
HASH JOIN		
JOIN FILTER CREATE	:BF0001	
PART JOIN FILTER CREATE	:BF0000	LO_ORDERDATE = D_DATEKEY
TABLE ACCESS STORAGE FULL	DATE_DIM	D_YEAR IN (1993, 1994, 1995)
HASH JOIN		
JOIN FILTER CREATE	:BF0002	LO_PARTKEY = P_PARTKEY
TABLE ACCESS STORAGE FULL	PART	P_CONTAINER = 'JUMBO PACK'
JOIN FILTER USE	:BF0001	
JOIN FILTER USE	:BF0002	
PARTITION RANGE JOIN-FILTER		
TABLE ACCESS STORAGE FULL	LINEORDER	:BF0000




```
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
```

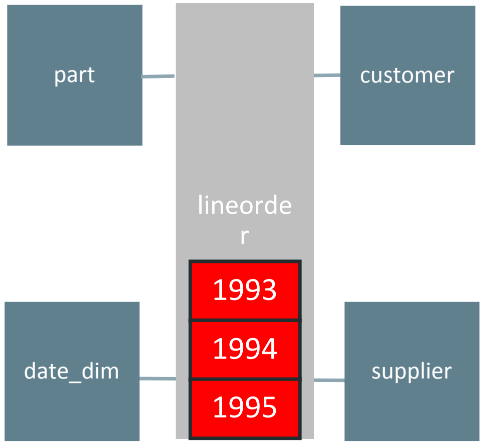
ORACLE

ORACLE REAL-WORLD PERFORMANCE

Intelligent Full Scans

2. Extract Rows from the Fact table

Operation	Object Name	Predicate information
SELECT STATEMENT		
SORT GROUP BY		
HASH JOIN		LO_SUPPKEY = S_SUPPKEY
TABLE ACCESS STORAGE FULL	SUPPLIER	
HASH JOIN		
JOIN FILTER CREATE	:BF0001	
PART JOIN FILTER CREATE	:BF0000	LO_ORDERDATE = D_DATEKEY
TABLE ACCESS STORAGE FULL	DATE_DIM	D_YEAR IN (1993, 1994, 1995)
HASH JOIN		
JOIN FILTER CREATE	:BF0002	LO_PARTKEY = P_PARTKEY
TABLE ACCESS STORAGE FULL	PART	P_CONTAINER = 'JUMBO PACK'
JOIN FILTER USE	:BF0001	
JOIN FILTER USE	:BF0002	
PARTITION RANGE JOIN-FILTER		
TABLE ACCESS STORAGE FULL	LINEORDER	:BF0000




```
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
```

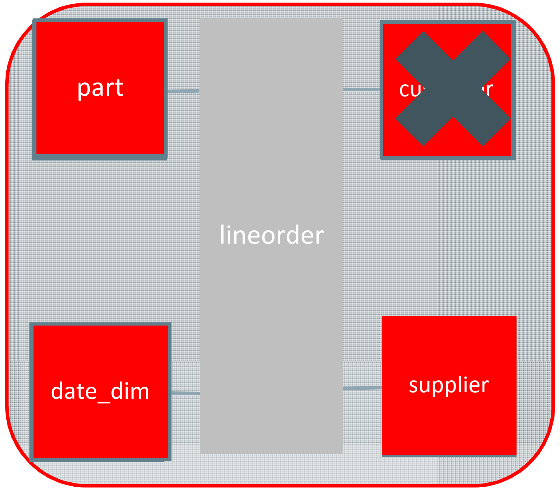
ORACLE

ORACLE
REAL-WORLD PERFORMANCE

Intelligent Full Scans

3. Join to Dimensions to Project additional columns

Operation	Object Name	Predicate information
SELECT STATEMENT		
SORT GROUP BY		
HASH JOIN		LO_SUPPKEY = S_SUPPKEY
TABLE ACCESS STORAGE FULL	SUPPLIER	
HASH JOIN		
JOIN FILTER CREATE	:BF0001	
PART JOIN FILTER CREATE	:BF0000	LO_ORDERDATE = D_DATEKEY
TABLE ACCESS STORAGE FULL	DATE_DIM	D_YEAR IN (1993, 1994, 1995)
HASH JOIN		
JOIN FILTER CREATE	:BF0002	LO_PARTKEY = P_PARTKEY
TABLE ACCESS STORAGE FULL	PART	P_CONTAINER = 'JUMBO PACK'
JOIN FILTER USE	:BF0001	
JOIN FILTER USE	:BF0002	
PARTITION RANGE JOIN-FILTER		
TABLE ACCESS STORAGE FULL	LINEORDER	:BF0000



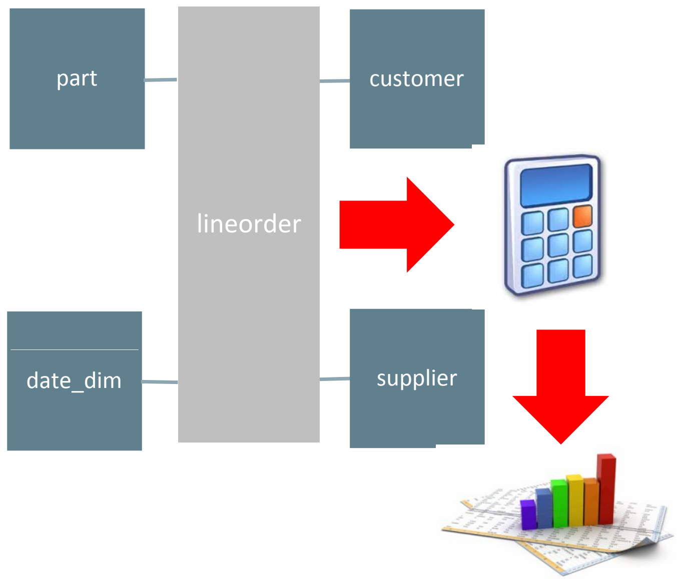

```
SELECT          d_sellingseason, p_category, s_region,
                sum(lo_extendedprice)
                FROM     lineorder
                  JOIN       customer             ON lo_custkey = s_custkey
                  JOIN       date_dim             ON lo_orderdate = d_datekey
                  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
```

ORACLE | ORACLE REAL-WORLD PERFORMANCE

Intelligent Full Scans

4. Aggregate/Sort Rows and Return Results

Operation	Object Name	Predicate information
SELECT STATEMENT		
SORT GROUP BY		
HASH JOIN		LO_SUPPKEY = S_SUPPKEY
TABLE ACCESS STORAGE FULL	SUPPLIER	
HASH JOIN		
JOIN FILTER CREATE	:BF0001	
PART JOIN FILTER CREATE	:BF0000	LO_ORDERDATE = D_DATEKEY
TABLE ACCESS STORAGE FULL	DATE_DIM	D_YEAR IN (1993, 1994, 1995)
HASH JOIN		
JOIN FILTER CREATE	:BF0002	LO_PARTKEY = P_PARTKEY
TABLE ACCESS STORAGE FULL	PART	P_CONTAINER = 'JUMBO PACK'
JOIN FILTER USE	:BF0001	
JOIN FILTER USE	:BF0002	
PARTITION RANGE JOIN-FILTER		
TABLE ACCESS STORAGE FULL	LINEORDER	:BF0000

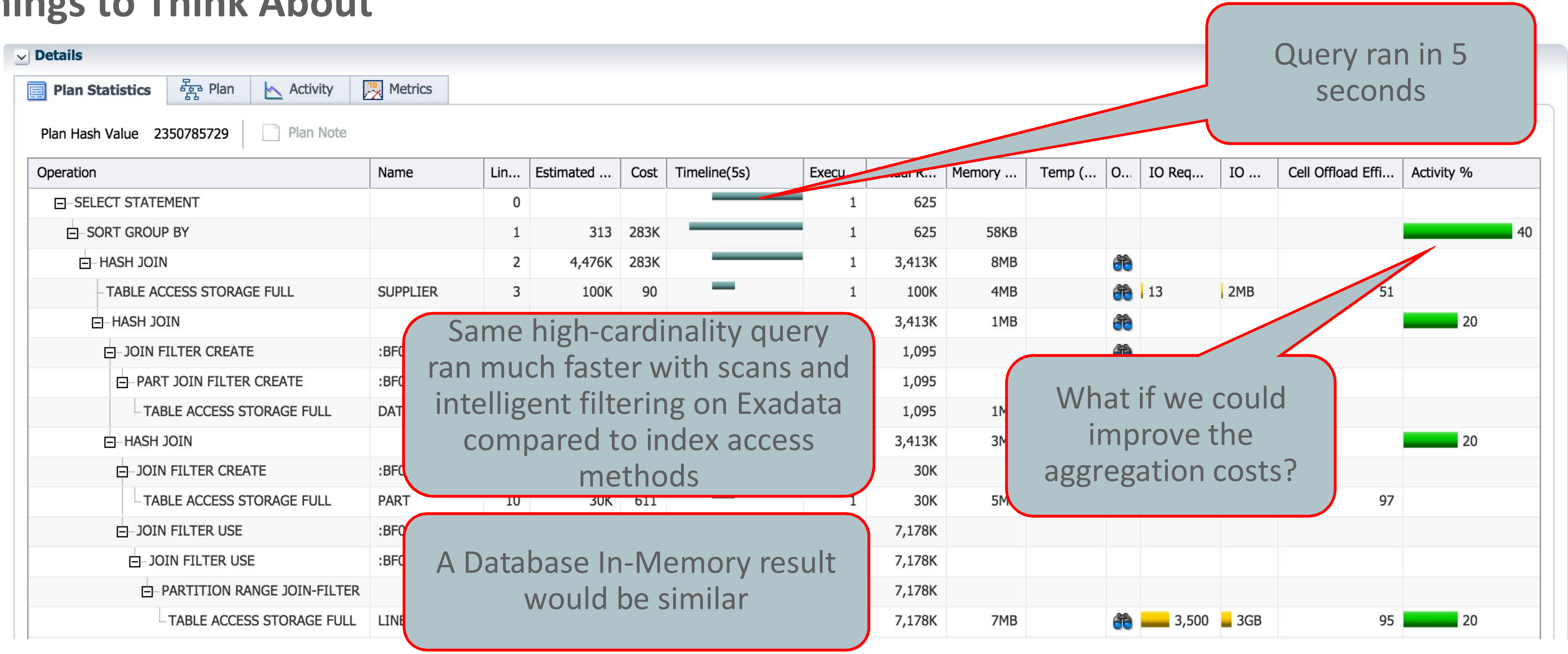



```
SELECT          d_sellingseason, p_category, s_region,
                sum(lo_extendedprice)
                FROM    lineorder
                   JOIN     customer        ON lo_custkey = c_custkey
                   JOIN     date_dim        ON lo_orderdate = d_datekey
                   JOIN     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
```

REAL-WORLD PERFORMANCE

Intelligent Filtering

Things to Think About



ORACLE

ORACLE REAL-WORLD PERFORMANCE

Full Scans with Intelligent Filtering Summary

Technique	Primary Fact Table Access Method	Requirements	Pros	Cons
B*Tree Indexes with NL Joins	• B*Tree index access • Nested Loops joins	• Indexes on fact table	Decent performance if number of rows is very small and all data accessed is satisfied from memory	Algorithmically weak; can't get fact table rows fast enough
Star transformation	• Rowid from bitmap index • Bitmap merge • Star transformation	• star_transformation_enabled • query_rewrite_integrity • PK/FK constraints • NOT NULL constraints • Bitmap indexes on fact table	Excellent performance if number of rows is small and all data accessed is satisfied from memory	Poor performance if number of rows from fact table is high and requires random I/O
Full Scans with Intelligent Filtering	• Full scans • Swap join optimization & right-deep tree • Bloom Filters • Pipelined hash joins	• Exadata or DBIM • cell_offload_processing • PK/FK constraints • NOT NULL constraints	Can handle high and low cardinality queries to achieve consistent response times	Infrastructure cost, scalability as concurrency increases

ORACLE

ORACLE REAL-WORLD PERFORMANCE

Table Scans with Intelligent Filtering

Things We Do for Performance

- Exploit Latest HW and SW technologies Exadata and Database In-Memory
  - Hundreds of GB/second
  - Millions->Billions of Rows/second
- Specialist Execution plans and algorithms, Swap join optimization and right-deep trees

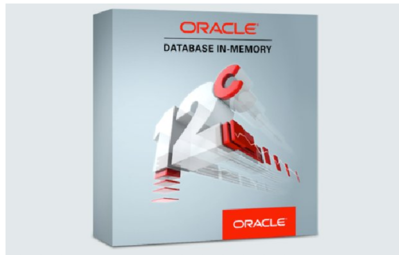
Exadata or Oracle Database In-Memory

Table Scans with Intelligent Filtering

How We Do It

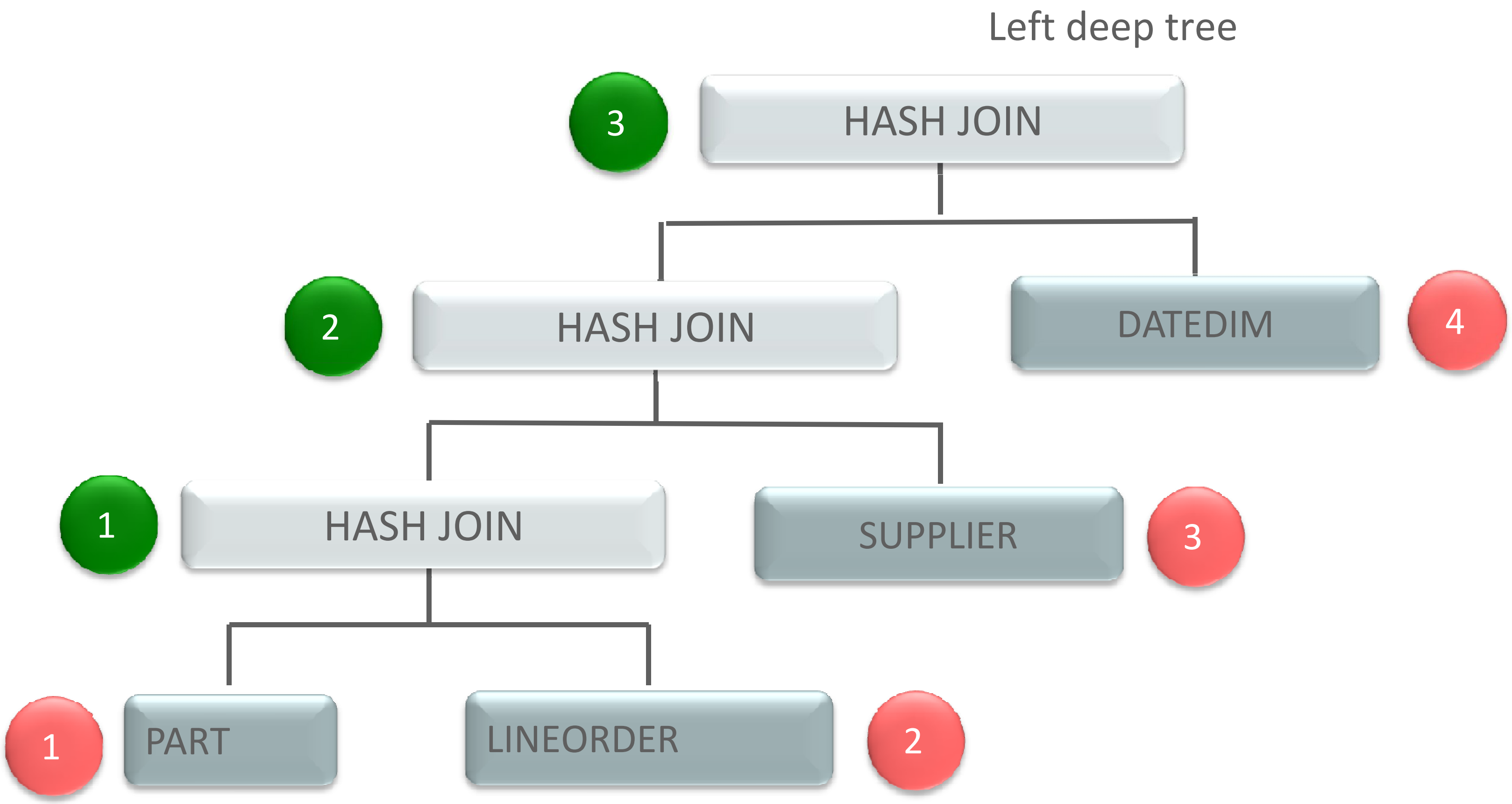
	Exadata	Database In Memory
Scans and Access	**Hardware**: CPUs, disks, flash, InfiniBand **Software**: Smart Scan, HCC, Storage Indexes	In-Memory columnar layout SIMD vector processing
Filtering & Evaluation	Bloom Filters pushed down to storage	Bloom Filters pushed down to column store

Exadata



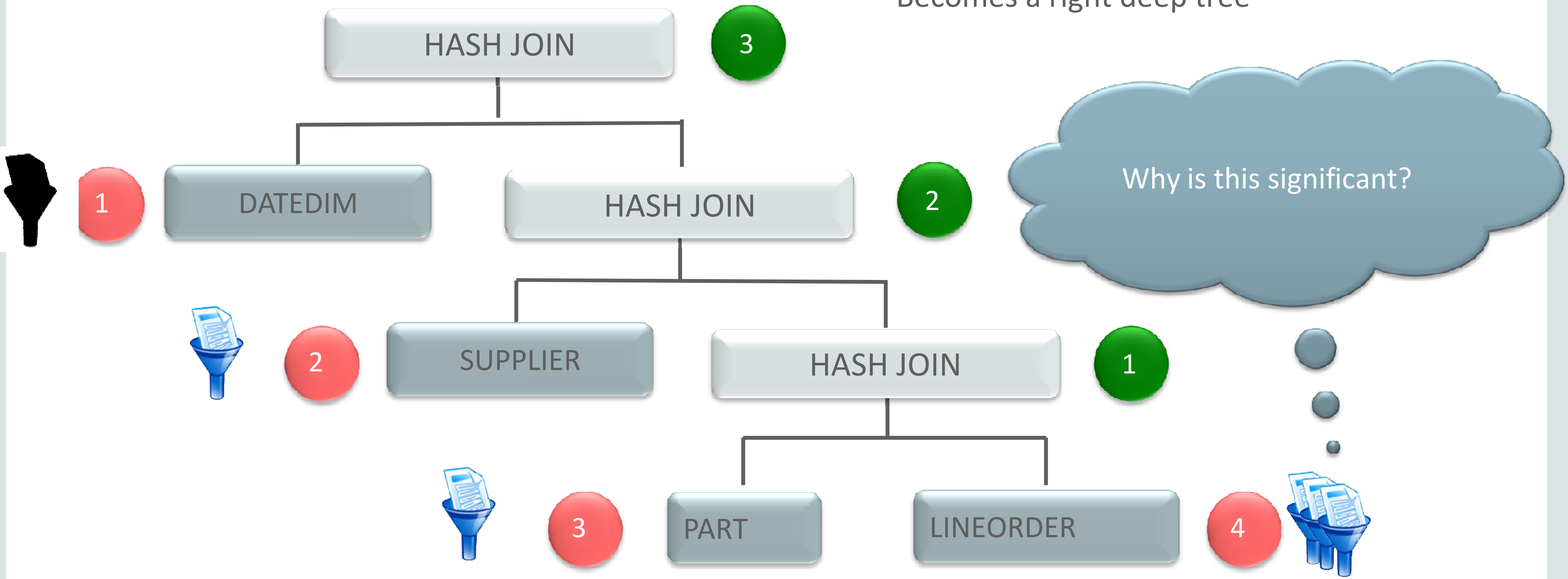
Database In Memory

Swap Join Input Optimization



Swap Join Input Optimization

Becomes a right deep tree

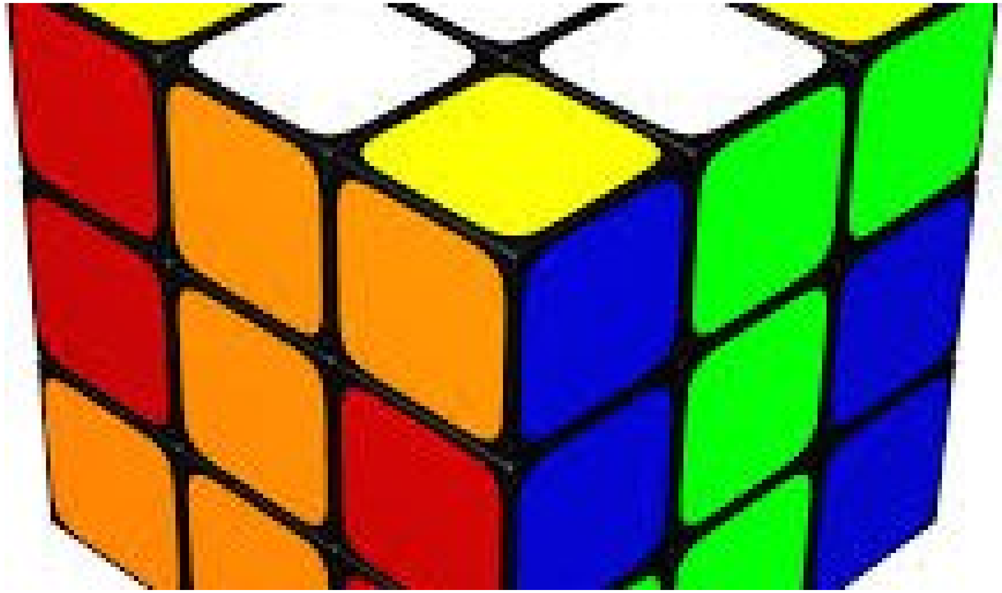


ORACLE
ORACLE REAL-WORLD PERFORMANCE

Optimizations after the Joins

Vector Transformation (In-Memory Aggregation)

- Queries extract many rows from Fact table
- Database size large
- Aggregation on low cardinality dimensions
- Optimizer costs the transformation



Oracle Database In-Memory

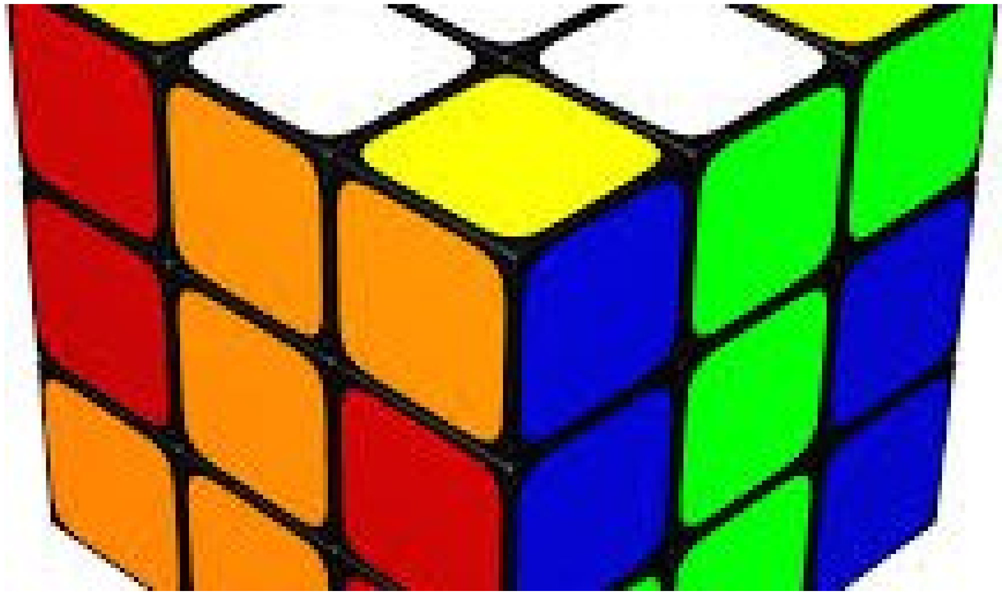
ORACLE

ORACLE REAL-WORLD PERFORMANCE

In-Memory Aggregation

Why Do It?

- Goal is to do extra work up-front while processing dimension tables to save time downstream
- Scans and filtering takes place in the DBIM column store
- Aggregation is performed as part of the fact table access
- Build a cube as we scan the fact table to avoid potentially costly aggregation



Oracle Database In-Memory

Choose Your Execution Method

Vector Transformation

```
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
```

```
SELECT key_vector(lo_orderdate), key_vector(lo_partkey), key_vector(lo_suppkey),
       sum(lo_extendedprice)
       FROM lineorder
       WHERE key_vector(lo_orderdate) IN
            (SELECT key_vector(d_datekey)
                    FROM  date_dim
                    WHERE d_year      IN ( 1993,1994,1995 ))
       AND    key_vector(lo_partkey) IN
            (SELECT key_vector(p_partkey)
                    FROM part
                    WHERE p_container IN ( 'JUMBO PACK' ))
       GROUP BY key_vector(lo_orderdate), key_vector(lo_partkey), key_vector(lo_suppkeu)
```

ORACLE

ORACLE REAL-WORLD PERFORMANCE

In-Memory Aggregation Summary

Technique	Primary Fact Table Access Method	Requirements	Pros	Cons
B*Tree Indexes with NL Joins	• B*Tree index access • Nested Loops joins	• Indexes on fact table	Decent performance if number of rows is very small and all data accessed is satisfied from memory	Algorithmically weak; can't get fact table rows fast enough
Star transformation	• Rowid from bitmap index • Bitmap merge • Star transformation	• star_transformation_enabled • query_rewrite_integrity • PK/FK constraints • NOT NULL constraints • Bitmap indexes on fact table	Excellent performance if number of rows is small and all data accessed is satisfied from memory	Poor performance if number of rows from fact table is high and requires random I/O
Full Scans with Intelligent Filtering	• Full scans • Swap join optimization & right-deep tree • Bloom Filters and pipelined hash joins	• Exadata or DBIM • cell_offload_processing • PK/FK constraints • NOT NULL constraints	Can handle high and low cardinality queries to achieve consistent response times	Infrastructure cost, scalability as concurrency increases
In-Memory Aggregation	• Full scans • Vector Transformation	• DBIM • PK/FK constraints • NOT NULL constraints	Excellent performance for both scan, filter, and aggregation	

ORACLE

ORACLE REAL-WORLD PERFORMANCE

Star Query Multi-User Demo

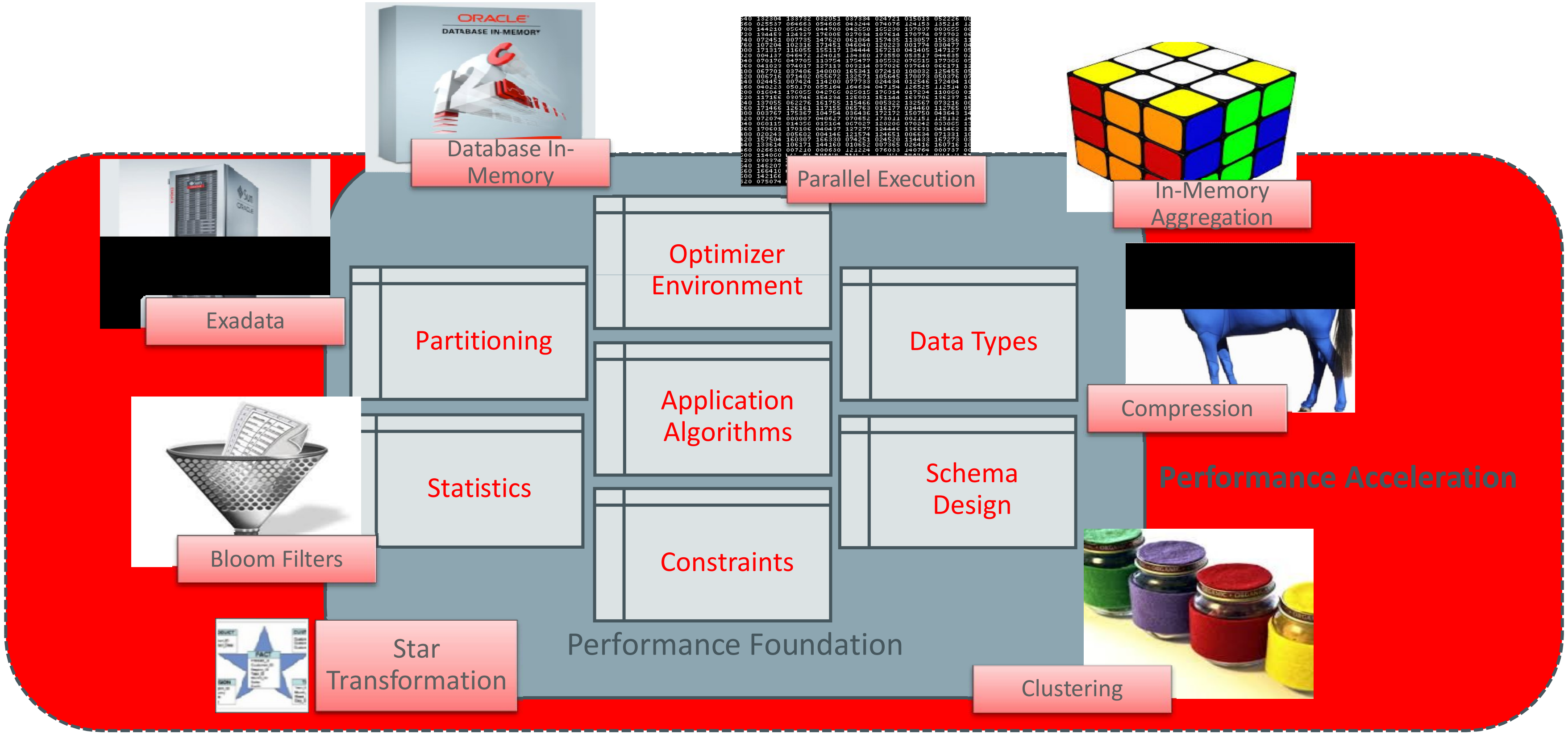
Part I

ORACLE

ORACLE REAL-WORLD PERFORMANCE

Prescription

Convergence of Techniques and Technology



ORACLE REAL-WORLD PERFORMANCE

The Prescription

Things You Must Do to Ensure Optimal Execution Plans

ORACLE

ORACLE
REAL-WORLD PERFORMANCE

The Prescription

What you must do

- Constraints
- Data Types
- Statistics
- Partitioning

ORACLE
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

ORACLE

ORACLE
REAL-WORLD PERFORMANCE

What You Must Do

NOT NULL Constraints

- For each row in lineorder, how many rows are returned from customer?
- ***Without*** constraints, what if lo_custkey is NULLable?
- Even if lo_custkey is NOT NULL, how many rows will join with customer? 0? 1? More than 1?
- NOT NULL constraints are essentially free, no reason not to implement
- Several optimizations depend on this information!

```
FROM     lineorder
JOIN     customer ON
            lo_custkey = c_custkey
```

```
SQL> desc lineorder
 Name            Null?          Type
 -----------     -------        --------
...
LO_CUSTKEY       NOT NULL       NUMBER
...

SQL> desc customer
 Name            Null?          Type
 ---------       ---------      --------
C_CUSTKEY        NOT NULL       NUMBER
....
```

ORACLE ORACLE REAL-WORLD PERFORMANCE

What You Must Do

Primary Key and Foreign Key Constraints

- 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

```
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;
```

With PK/FK constraints, exactly 1 row is returned from dimension table for a fact row

ORACLE

ORACLE REAL-WORLD PERFORMANCE

What You Must Do

Validating ETL/ELT

- 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

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

ORACLE
ORACLE REAL-WORLD PERFORMANCE

What You Must Do

Validating ETL/ELT

- 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

```
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;
```

ORACLE | ORACLE REAL-WORLD PERFORMANCE

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

```
from        T_TAB_WKO_04   a11
where ( a11.DIMENSION02, a11.DIMENSION08 ) in (
  select a12.BRAND_CODE, a12.STYLE_COLOR_CODE
                from       T_GASDM_LU_STYLE_COLOR a12)
and  ((a11.MONTH = 12 and a11.YEAR = 2013)
and to_char(to_number(a11.dimension01)) in ('24'))
```

```
FROM     lineorder
JOIN     customer ON
             lo_custkey = c_custkey
```

```
SQL> desc lineorder
 Name           Null?        Type
 -----------    -------      ---------
...
LO_CUSTKEY     NOT NULL      NUMBER
...

SQL> desc customer
 Name           Null?        Type
 ---------      ---------    ---------
C_CUSTKEY      NOT NULL      NUMBER(11)
....
```

Needs to be NUMBER

What You Must Do

Ensure Optimizer Statistics are Accurate and Representative

- Think about skew
- Think about correlation
- Do not rely on Dynamic Statistics alone
- Think about how and when to gather statistics

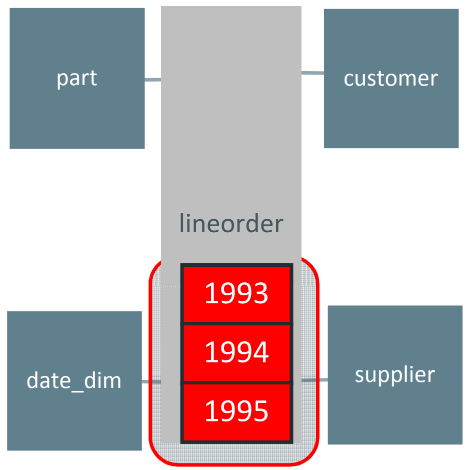
ORACLE

ORACLE REAL-WORLD PERFORMANCE

What You Must Do

Partition the Fact Table on the Time Dimension

- Typically RANGE or INTERVAL
- Reduces the number of rows extracted from the fact table (i.e., early filtering)
- Improves manageability



Applicable regardless of execution method

What You Must Do

Partition the Fact Table on the Time Dimension

Example: Interval partitioning

```
CREATE TABLE
 LINEORDER
(
 "LO_ORDERKEY" NUMBER NOT NULL ENABLE
,"LO_LINENUMBER" NUMBER
... other columns
)
partition by range
(
 LO_ORDERDATE
)
interval (numtoyminterval(1, 'MONTH'))
(
 partition R199201 values less than
   (to_date('19920201', 'YYYYMMDD'))
)
;
```

ORACLE

ORACLE REAL-WORLD PERFORMANCE

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

ORACLE
ORACLE REAL-WORLD PERFORMANCE

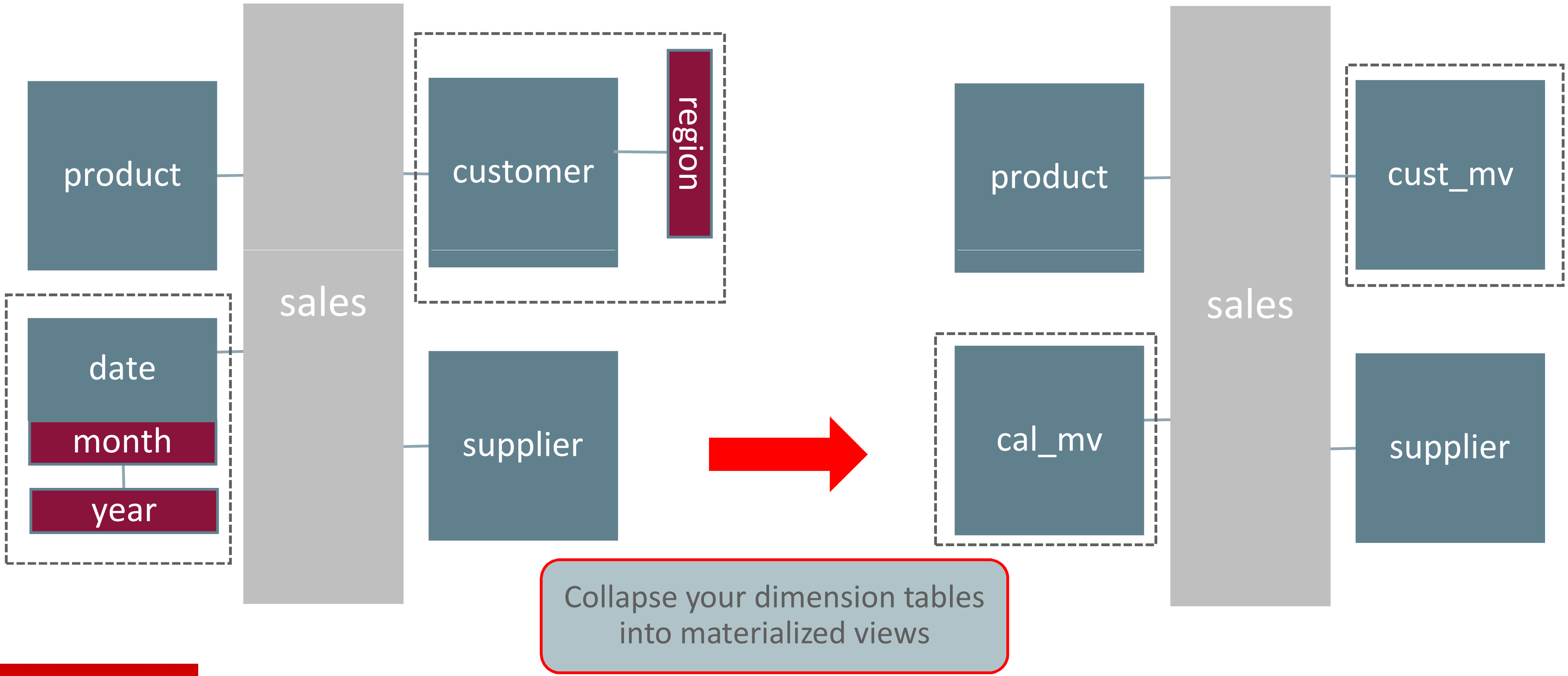
Star Query Fundamentals

ORACLE

ORACLE REAL-WORLD PERFORMANCE

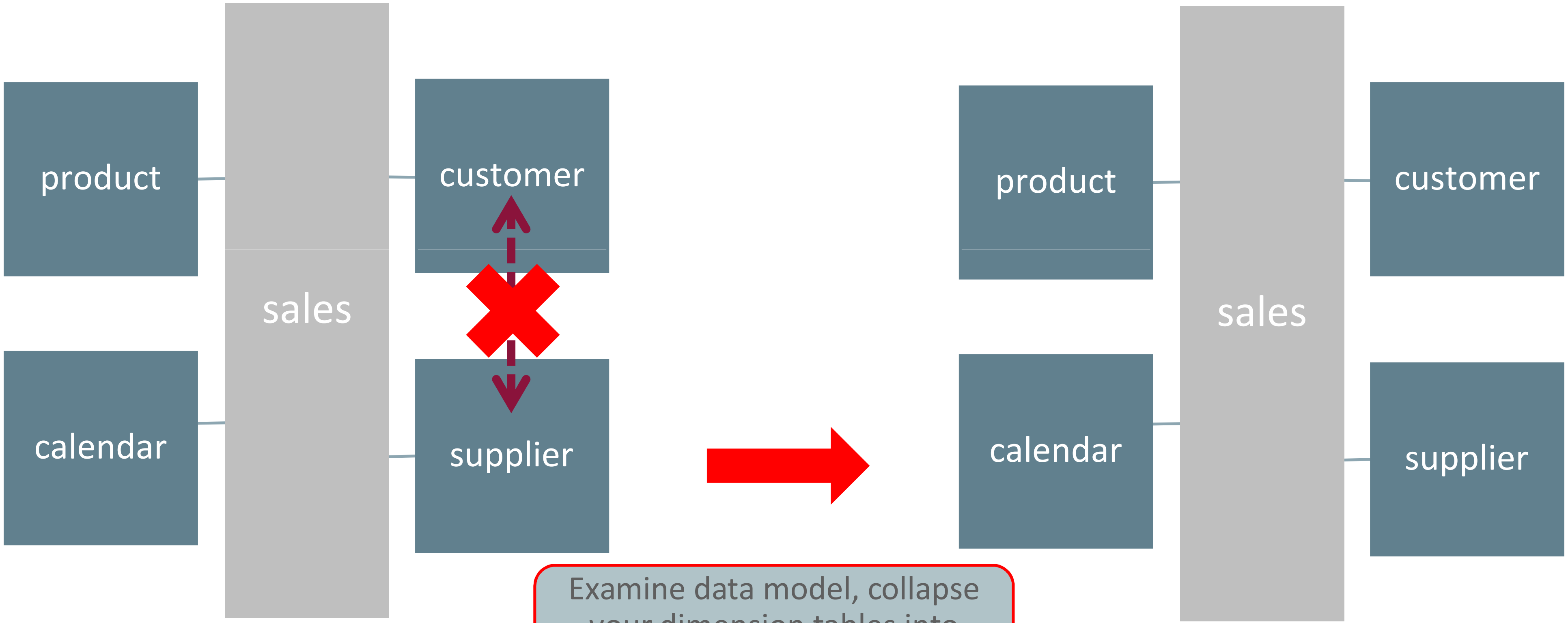
Edge Conditions

Snowflake Schema



Edge Conditions

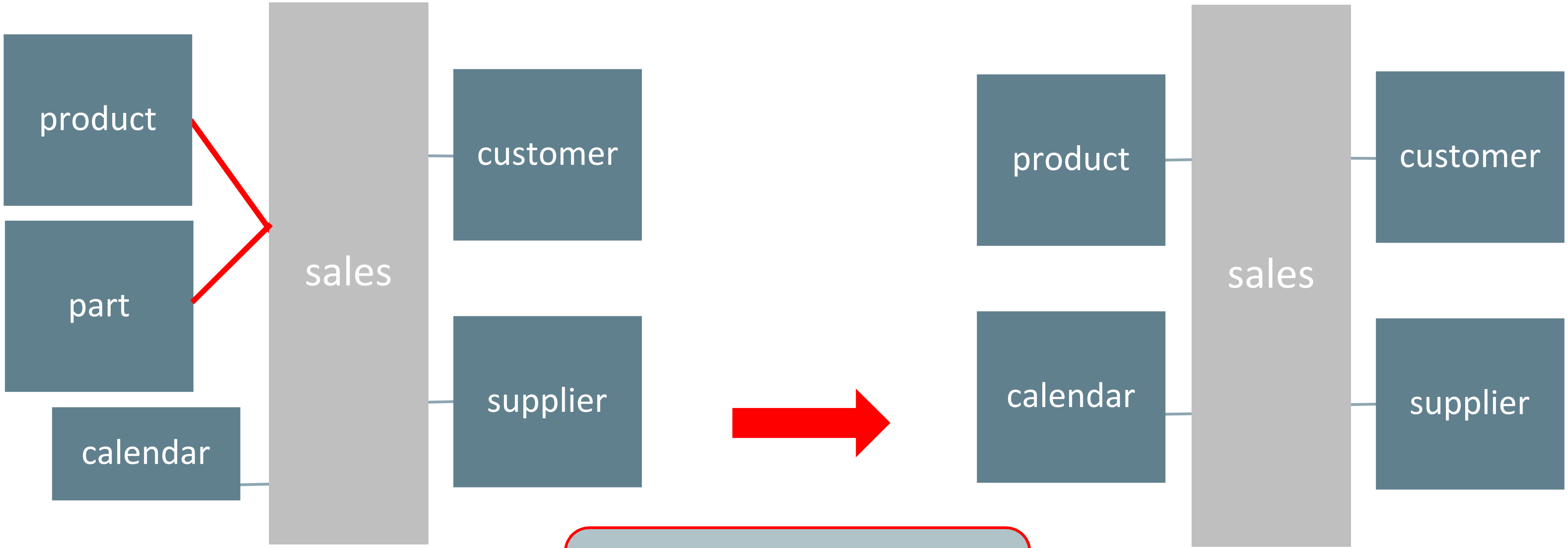
Relationships between Dimensions



Examine data model, collapse your dimension tables into materialized views

Edge Conditions

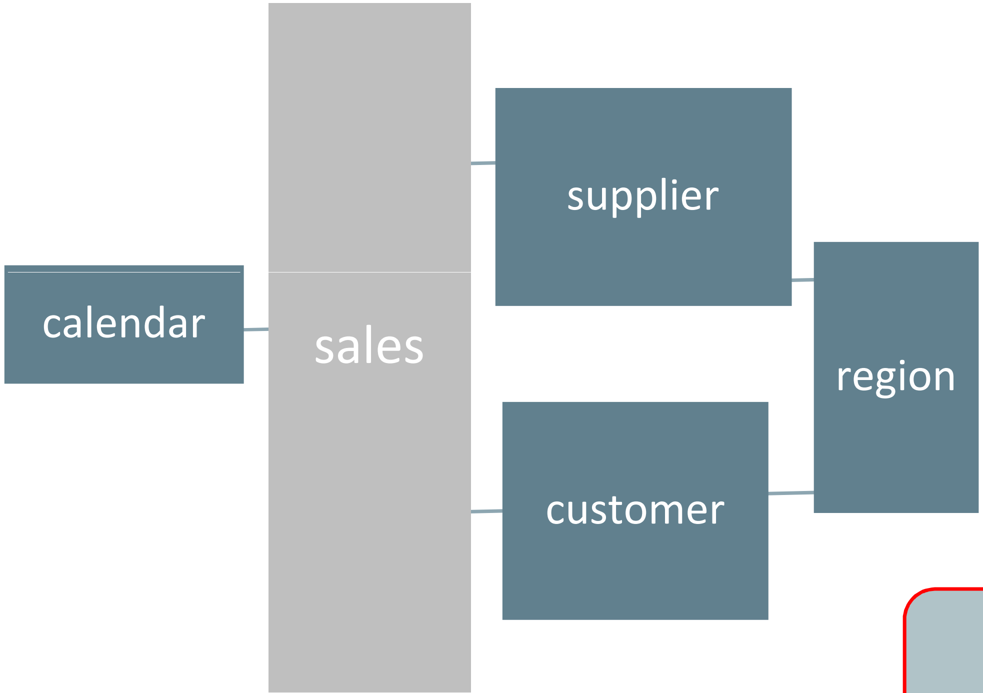
Common Join Columns



Examine data model, look to see if there are missing columns

Edge Conditions

Not “Completing” Joins



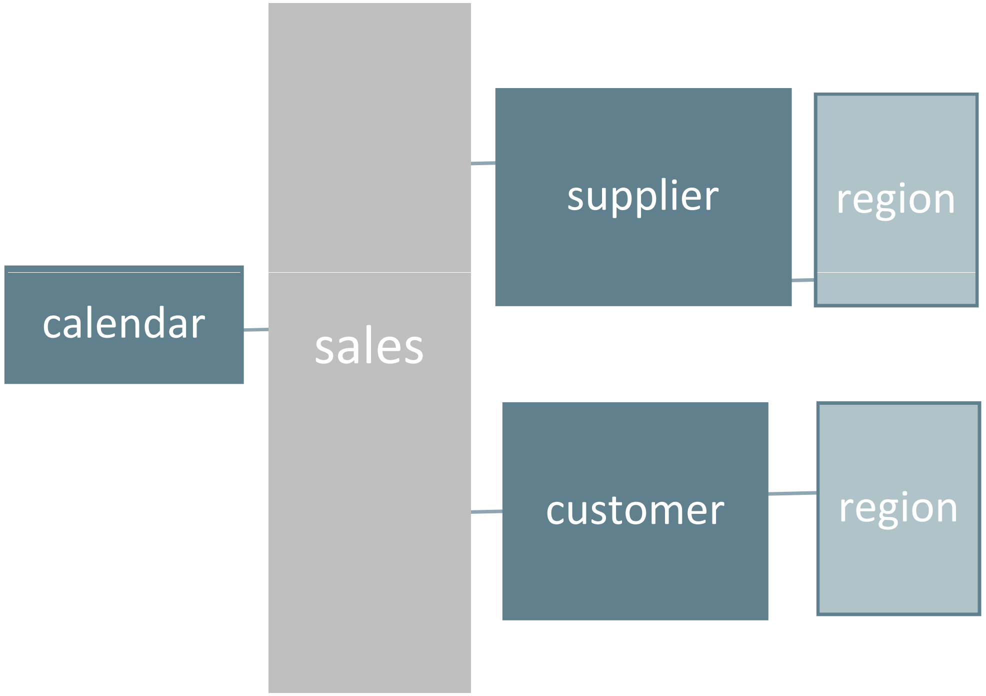

```
SELECT        d.year, s.nation, c.nation,
              SUM(l.extendedprice)
              FROM    lineorder l
                           JOIN          calendar d ON l.lo_orderdate = d_datekey
                           JOIN          supplier s ON l.lo_suppkey = s.suppkey
                           JOIN          customer c ON l.lo_suppkey = c.custkey
                           JOIN          region   r ON s.region_id  = r.region_id
              WHERE        d.year IN (1993, 1994, 1995)
              AND          r.region_code = 'ASIA'
              AND          c.region_id   = s.region_id
GROUP BY      d.year, s.nation, c.nation,
ORDER BY      d.year, s.nation, c.nation;
```

Snowflake schema with queries providing filter predicates once for both dimension joins and not completing joins

ORACLE REAL-WORLD PERFORMANCE

Edge Conditions

“Completing” Joins



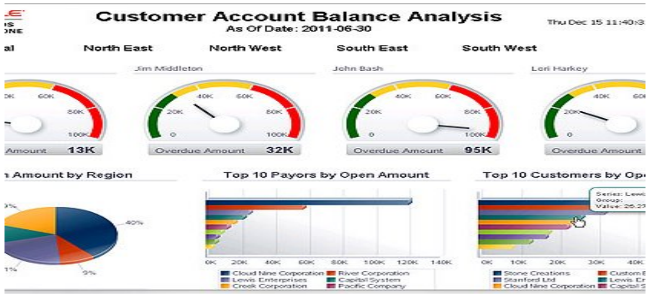

```
SELECT       d.year, s.nation, c.nation,
             SUM(l.extendedprice)
             FROM    lineorder l
                           JOIN          calendar d ON l.lo_orderdate = d_datekey
                           JOIN          supplier s ON l.lo_suppkey = s.suppkey
                           JOIN          customer c ON l.lo_suppkey = c.custkey
                           JOIN          region   r1 ON s.region_id  = r1.region_id
                           JOIN          region   r2 ON c.region_id  = r2.region_id
             WHERE         d.year IN (1993, 1994, 1995)
             AND           r1.region_code = 'ASIA'
             AND           r2.region_code = 'ASIA'
GROUP BY     d.year, s.nation, c.nation,
ORDER BY     d.year, s.nation, c.nation;
```

Join each dimension to outer table in snowflake schema

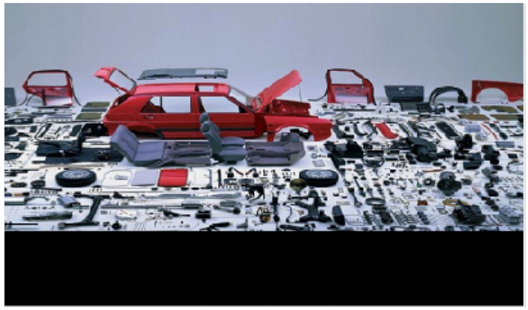
ORACLE

ORACLE REAL-WORLD PERFORMANCE

Recent Results 1000X Project



Baseline:	4.3 Hours
Code Changes:	4.3 Hours
Correct Usage:	29 Secs
Bug Fixes:	12 Secs
Final:	12 Secs
Speed up:	**1355.57**



Baseline:	2.4 Days
Code Changes:	27 Mins
Correct Usage:	7.5 Mins
Bug Fixes:	4.5 Mins
Final:	4.5 Mins
Speed up:	**768**

Baseline:	2.5 Hours
Code Changes:	2.5 Hours
Correct Usage:	4 Secs
Bug Fixes:	0.90 Secs
Final:	0.90 Secs
Speed up:	**9000**

ORACLE

ORACLE REAL-WORLD PERFORMANCE