

Key Data Warehousing Features in Oracle10g: A Comparative Performance Analysis

An Oracle White Paper

April 2005

Key Data Warehousing Features in Oracle10g: A Comparative Performance Analysis

Executive Overview	3
Introduction	3
SYSTEM AND SCHEMA CONFIGURATION	4
Schema Properties	4
Data Properties	4
Hardware Configuration	5
Data Warehouse Workload	5
Initial Load	5
Incremental Loads	6
Adding New Data	7
Deleting Old Data	10
Results for Incremental Load	10
Query Performance	11
Star Query: Example #1	11
Star Query: Example #2	12
Results of Star Query Performance	13
Additional simple queries	13
Conclusion	15
APPENDIX A: TEST SCHEMA	16
APPENDIX B: EXECUTION PLANS FOR QUERIES	19
Star Query Example #1: Oracle10g (Star Transformation)	19
Star Query Example #1: Generic (Dynamic Bitmap Indexes)	19
Star Query Example #1: Generic (Hash Join)	19
Star Query Example #2: Oracle10g (Star Transformation)	20
Star Query Example #2: Generic (Dynamic Bitmap Indexes)	20
Star Query Example #2: Generic (Hash Join)	20
APPENDIX C: INIT.ORA PARAMETERS	21
APPENDIX D: TABLE SELECTIVITY IN STAR QUERIES	22

Key Data Warehousing Features in Oracle10g: A Comparative Performance Analysis

EXECUTIVE OVERVIEW

Oracle10g's data-warehousing features provide significant performance advantages over competing databases. This paper focuses on two key features of Oracle10g, range partitioning and bitmap indexes, and demonstrates how these two features alone provide order-of-magnitude performance benefits for typical load and query operations in a data warehouse.

INTRODUCTION

Comparing the database technologies of multiple vendors is a challenging task. Every database vendor can cite multiple features and capabilities in their own product, and every database vendor claims that these features make their product better than the next vendor's.

This white-paper takes a different approach; rather than describing Oracle10g's data-warehousing features, this paper measures the performance benefits of two of Oracle10g's key features (range partitioning and bitmap indexes). The paper compares the performance of basic data-warehouse load and query operations for two databases. One database leverages the range partitioning and bitmap index features of Oracle10g, while the other database uses more generic relational database features (hash partitioning and b-tree indexes). The two databases are identical in every other respect. The purpose of this paper is to quantify the performance benefits of Oracle's key data-warehousing features over competing technologies.

The workload consists of fundamental data-warehousing operations: creating the data warehouse, maintaining the data warehouse, and querying the data warehouse. In each of these steps, the benefits of bitmap indexes and range partitioning are apparent. While this paper's workload is simpler than complex real-world data warehouses, the core advantages of these features extend fully to more complex environments. The majority of Oracle's data warehouse customers today use both range partitioning and bitmap indexes in their real-world environments.

Prospective customers are encouraged to experiment with these key features to understand their true benefits. To support such experimentation, this paper documents the schema and all of the SQL operations used in this workload.

SYSTEM AND SCHEMA CONFIGURATION

This test was conducted using two databases. Except for differences in the partitioning and indexing strategies, the databases are otherwise identical. Specifically, both databases:

- use identical software (Oracle10g release 2)
- have identical tuning parameters (provided in Appendix C)
- have identical schema, except partitioning and indexing methods (Schema Properties described below),
- use identical hardware (Hardware Configuration described below)
- use identical storage¹
- use identical data (Data Properties described below).

The database which uses the exclusive Oracle10g features will, henceforth in this paper, be referred to as **Oracle10g**, and the other database will be referred to as **Generic**.

Schema Properties

The schema used in this performance experiment is a star schema containing sales data. This schema consists of one fact table, SALES, and five dimension tables: PRODUCTS, CUSTOMERS, CHANNELS, PROMOTIONS, and TIMES. The complete schema is provided in Appendix A.

The SALES table contains approximately 300 million rows, and contains sales spanning 3 years (2002-2004).

Each database has five indexes on the SALES table: one index for each foreign-key column.

SALES is partitioned and indexed as follows:

SALES table properties	Oracle10g	Generic
Partitioning method	Range on TIME_ID	Hash on CUST_ID
# of partitions	36 (one per month)	32
Index Type	Bitmap Local	B*Tree Local

Table 1: Comparison of partitioning and indexing strategies

Data Properties

A data generator was used to create data for the initial and incremental data loads. The generated data had the following characteristics, in order to make the data as realistic as possible:

¹ For these tests the internal disk was used. This is not an optimal solution for a data warehouse and certain operations would perform better on an adequately sized attached storage configuration.

- Complete referential integrity between the SALES table and its dimension tables.
- The volume of sales data grows by 20% each year
- Within each year, the data was skewed so that November, December, and January have above-average sales volumes, and April, June, and August have below-average sales volumes.
- The data is consistent with the business model described by the schema; for example, a given customer could not have more purchases in the SALES table than would be allowed by that customer's credit limit in the CUSTOMERS tables.

Hardware Configuration

The hardware used in these performance tests was an HP Proliant DL380 G3 with 6 GB RAM, 2 CPUs of 3 GHz, running Redhat Enterprise Linux 3.0.

DATA WAREHOUSE WORKLOAD

The workload chosen for this test consists of three phases. The first phase is the initial loading of three years' worth of data, which includes building indexes. The second phase is an incremental load of one new month's worth of data, including maintaining indexes, and purging the oldest month of data. The third and final phase of the test is executing typical queries.

Initial Load

Three years of data, from Jan 2002 to Nov 2004, were loaded into the data warehouse using Oracle10g's External Table feature. Since this is a simple loading of rows from files to the database, there is no significant difference in performance between the Oracle10g database and the Generic database. The most time-consuming portion is loading of the SALES table, which took approximately 27 minutes in both databases.

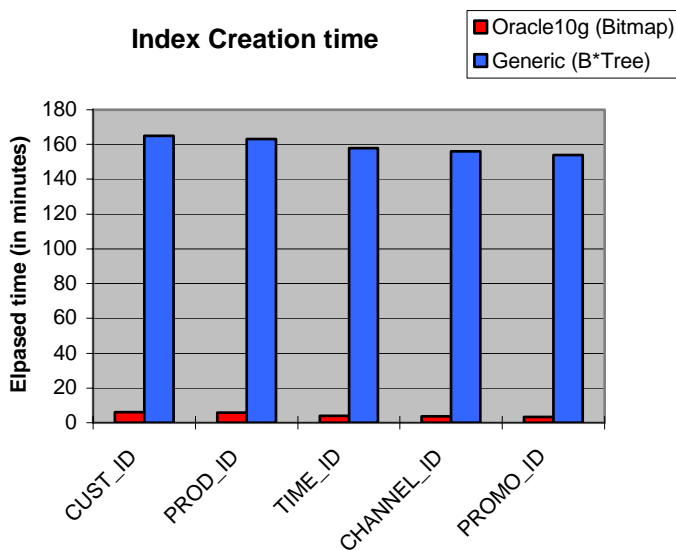
The next step in the initial load is to create all the necessary indexes. As is common in many star-schema data warehouses, all of the foreign key columns of the fact table are indexed. The Oracle10g database used bitmap indexes for this purpose, whereas the Generic database used b-tree indexes. The bitmap indexes provide significant savings in terms of both index-creation time and the space required. As shown in the following table, creating 5 b-tree indexes took much longer² and used approximately nine times more space as compared to bitmap indexes:

² Creating b-tree indexes is one of the operations that has definitely suffered from the less than optimal storage configuration. Note though that the difference is large enough to realize that with optimal storage the creation of b-tree indexes would still take more time.

Column Name	Elapsed Time (hh:mi:ss)		Space (MB)	
	Bitmap	B-Tree	Bitmap	B-Tree
CUST_ID	0:06:09	2:45:12	1,054	5,177
PROD_ID	0:05:43	2:43:37	1,202	4,796
TIME_ID	0:04:00	2:38:00	348	5,862
CHANNEL_ID	0:03:43	2:36:45	230	4,009
PROMO_ID	0:03:36	2:34:48	149	4,621
TOTAL	0:23:11	13:18:22	2,983	24,465

Table 2: Comparison of creation time and space consumption of bitmap and b-tree indexes

The following charts depict the comparison:



Indexed column of SALES table

Oracle10g's bitmap indexes are true bitmap indexes in every aspect. Other databases have 'dynamic bitmap index' capabilities, but in those databases the indexes are stored on disk as b-tree indexes. Databases with dynamic bitmap indexes do not receive any of the space savings or index-creation time savings obtained by Oracle10g's true bitmap indexes.

Incremental Loads

After the initial creation, the data warehouse is kept up to date by loading new data, and purging old data. In this scenario, a new month's worth of data is being loaded, and the oldest month's data is being purged. This is commonly called a 'rolling window' operation, because the data warehouse is being maintained so that it always has the most recent three years of data online, and during each monthly load cycle the three-year window 'rolls forward' by one month.

The key Oracle10g feature that enables efficient incremental loading is range partitioning. Since this data warehouse is adding new data on a monthly basis, the SALES table has been partitioned by month. Thus, adding a new month's data involves adding a new partition, and purging an old month's data involves dropping an existing partition. This partitioning strategy provides tremendous performance benefits, as the tests described below will illustrate.

Some data warehouses are loaded more frequently than once per month. However, while this paper's scenario illustrates a monthly data-load, the techniques outlined here are applicable to any time-based load scenario. Oracle customers use range partitioning to efficiently load their data warehouses monthly, weekly, daily, or even hourly or more frequently. By choosing an appropriate range partition granularity, range partitioning can vastly improve the performance of any load process.

Adding New Data

The data to be added to the SALES table of the data warehouse is for the month of December 2004. This new data is in a flat file and will be loaded using the External Tables feature of Oracle10g. The number of rows to be loaded for December 2004 is 12,558,000.

Oracle10g Database

Since the SALES table in Oracle10g database is range partitioned on TIME_ID by month, the new data will all go into one partition - namely the partition corresponding to December 2004. This partition does not exist yet, but can be easily added to the SALES table. Moreover, since the bitmap indexes on the SALES table are partitioned such that each partition of the index corresponds to only one partition of the table (this is known as a 'local' index), adding a new partition will not affect the indexes for any other partitions. Adding new data is done as follows:

1. Add an empty partition for December 2004 by running the following SQL

```
ALTER TABLE sales ADD PARTITION sales_dec_2004
VALUES LESS THAN (TO_DATE('01-jan-2005', 'dd-mon-yyyy'));
```

2. Create a non-partitioned table SALES_TEMP_DEC_2004

```
CREATE TABLE sales_temp_dec_2004 AS
SELECT * FROM sales WHERE ROWNUM < 1;
```

3. Load the data into the table SALES_TEMP_DEC_2004 by running the following command (where SALESXT is the external table that is defined on the flat files which contain data for December 2004):

```
INSERT INTO sales_temp_dec_2004
SELECT * FROM salesxt;
```

4. Create a bitmap index on each of the foreign keys of SALES_TEMP_DEC_2004

```
CREATE BITMAP INDEX sales_cust_id_bix_dec_2004
ON sales_temp_dec_2004 (cust_id)
NOLOGGING PARALLEL;
```

Similarly create the other four indexes as well.

5. Perform an exchange partition. The following command is a DDL statement and it will merge the bitmap indexes on the table SALES_TEMP_DEC_2004 with the corresponding local partitioned indexes on SALES table.

```
ALTER TABLE sales EXCHANGE PARTITION sales_dec_2004
WITH TABLE sales_temp_dec_2004
INCLUDING INDEXES WITHOUT VALIDATION;
```

6. Drop the table SALES_TEMP_DEC_2004.

Steps 1, 2, 5, and 6 are DDL statements, which execute extremely quickly. These steps executed in less than 1 second total.

Step 3 (loading into a non-partitioned table using external table) required 2 minutes and 6 seconds, while step 4 (creation of bitmap indexes on SALES_TEMP_DEC_2004 tables) required 29 seconds.

The total time required to add a new month's worth of data to the Oracle10g database was 2 minutes and 36 seconds.

After the exchange partition operation has executed successfully, the new sizes of the bitmap indexes are given below:

Index Name	New Size of the Bitmap Index in MB
SALES_CUST_ID_BIX	1,061
SALES_PROD_ID_BIX	1,216
SALES_TIME_ID_BIX	352
SALES_CHANNEL_ID_BIX	232
SALES_PROMO_ID_BIX	149
TOTAL	3,010

The incremental difference in space used by indexes in the Oracle10g database was 3,010 MB – 2,983 MB = 27 MB.

Generic Database

Since the SALES table in Generic database is hash partitioned on CUST_ID, the incoming rows from December 2004 will be distributed among all 32 partitions of the SALES table. Inserting the rows for December 2004 from the flat files into SALES table can be done in two ways:

1. The indexes on the SALES table could be maintained during the load process. The index maintenance is done in parallel, but this approach can be costly because all of the indexes are maintained at the same time which requires large amounts of temporary space and additionally slows down the load times by a considerable margin.
2. All the indexes on the SALES table can be dropped prior to loading the data. Once the new data has been loaded, the indexes are recreated one-by-one. Since the indexes are built one at a time, the total amount of temporary space is minimized. This approach is chosen for this workload because this approach takes less time.

Therefore, the steps required for "add new data" operation on the Generic database are:

1. Drop the indexes on SALES table. This operation takes a few seconds to finish.
2. Load December 2004 data into the SALES table in parallel, using external tables. The operation requires approximately 6 minutes.
3. Recreate the five b-tree indexes on the SALES table. The time required to create these indexes, and their result sizes are summarized in the table below³:

Index name	Elapsed Time (hh:mi:ss)	Space Used (MB)
SALES_CUST_ID_IX	2:45:35	5,408
SALES_PROD_ID_IX	2:40:17	5,010
SALES_TIME_ID_IX	2:37:28	6,124
SALES_CHANNEL_ID_IX	2:33:56	4,189
SALES_PROMO_ID_IX	2:37:36	4,828
TOTAL	13:14:52	25,559

1. Enable the constraints (NOVALIDATE) on SALES table (instantaneous).

Hence, Total time for the "Add new Data" operation on the Generic database is 800 minutes. The incremental difference in space used by indexes on the Generic database = 25,559 – 24,465 = 1,094 MB.

Results for "Adding New Data"

The following table shows the performance benefits of the Oracle10g database over Generic database percentage wise:

³ Note again that the less than optimal storage configuration has resulted in a slower than necessary index creation.

% Savings of Oracle10g over Generic	
Incremental SPACE required for Indexes	TIME
97.53%	99.75%

Deleting Old Data

The data to be deleted from the SALES table of the data warehouse is typically the oldest data, which in this case is the January 2002 data. The total number of rows for January 2002 is 7,475,000.

Oracle10g Database

Since the Oracle10g database is range partitioned on month, all we need to do is drop the partition corresponding to January 2002 in the SALES table. The indexes are local so none of the other index partitions are affected. We can do this simply by issuing the following SQL:

```
ALTER TABLE SALES DROP PARTITION SALES_JAN_2002;
```

The time taken for "Deleting Old Data" on the Oracle10g database is 1 second.

Generic Database

The Generic database is hash partitioned on CUST_ID, so the data corresponding to January 2002 can be spread over more than one partitions (possibly all). Hence, we can only issue a delete command. Since DELETE is a DML statement, it will use the rollback segment. In our case the amount of space used for the rollback segment = 4,334 MB.

The following statement can be executed in parallel:

```
DELETE FROM SALES WHERE TIME_ID < TO_DATE('01-FEB-2002','DD-MON-YYYY');
```

The time taken for "Deleting Old Data" in the Generic database is 7 hours, 51 minutes 31 seconds, or 28,291 seconds.

Results for "Deleting Old Data"

The following table summarizes the performance benefits of using Oracle10g versus the Generic database:

% Savings of Oracle10g over Generic	
SPACE required for Rollback segment	TIME
100%	99.99%

Results for Incremental Load

The total amount of time for an incremental load is simply the sum of the time for adding new data plus the time for dropping the old data. The Oracle10g database

completed the operation in a matter of minutes with minimal additional space requirements. The generic database required several hours with significant additional space requirements for rollback and additional index space.

Query Performance

The key requirement for almost any data warehouse is query performance. Both bitmap indexes and range partitioning accelerate the performance of typical data warehouse queries.

In a star-schema data warehouse, most queries will be star queries, in which the fact table is joined with two or more dimension tables and the results of the join is subsequently aggregated.

Star Query: Example #1

A typical star query is given below:

```
SELECT p.prod_name, SUM(s.amount_sold)
FROM sales s, products p, channels ch, promotions pm
WHERE s.prod_id = p.prod_id
AND s.channel_id = ch.channel_id
AND s.promo_id = pm.promo_id
AND ch.channel_desc = 'Catalog'
AND pm.promo_category = 'flyer'
AND p.prod_subcategory = 'Shorts - Men'
GROUP BY p.prod_name;
```

The cardinality and the selectivity (using the predicates in the above query) are given in Appendix D. This query is executed in parallel in both databases.

Oracle10g Database (Star Transformation with bitmap indexes)

Oracle10g has a highly optimized algorithm for executing star queries called the 'star transformation'. This algorithm leverages Oracle's bitmap indexes to efficiently join the dimension tables to the fact table.

The execution plan for this query (and all subsequent queries) is given in Appendix B.

The time taken to execute the first example query on the Oracle10g database with Star Transformation (using bitmap indexes) is 39 seconds.

Generic Database (Star Transformation with B-tree indexes)

Some generic database systems also provide a star-transformation algorithm, but rely on b-tree indexes instead of bitmap indexes. This use of b-tree indexes is often referred to as 'dynamic bitmap indexes', since the b-tree indexes are dynamically converted into bitmap representations during query execution.

The lack of bitmap indexes causes this query execution strategy to be less efficient. Using the execution plan in Appendix B, the time taken to execute the first example query on the Generic database is 94 seconds.

Generic Database (Hash Join)

Other database systems may lack the ability to do ‘dynamic bitmap indexing’, so the next best alternative for executing a star query is to do a hash join. The time taken to execute the first example query on the Generic database using hash joins is 262 seconds.

Star Query: Example #2

This query is a slight modification of Example #1. There is now an additional predicate on the TIME dimension table. This query is given below, with the extra predicate in bold:

```
select p.prod_name, sum(s.amount_sold)
from sales s, products p, channels ch, promotions pm, times t
where s.prod_id = p.prod_id
and s.channel_id = ch.channel_id
and s.promo_id = pm.promo_id
and s.time_id = t.time_id
and ch.channel_desc = 'Catalog'
and pm.promo_category = 'flyer'
and t.calendar_quarter_desc = '2000-Q2'
and p.prod_subcategory = 'Shorts - Men'
group by p.prod_name;
```

Oracle10g Database (Star Transformation)

As with the first example star query, Oracle10g uses the star transformation to execute this second star query. However, since the SALES table is range partitioned on TIME_ID and there is a predicate on TIME dimension, Oracle additionally uses ‘partition pruning’ to improve the performance of this query. When accessing the SALES table, Oracle only needs to access three partitions of the SALES table (corresponding to the three months in 2000-Q2). Since less data from the SALES table needs to be processed, this query is more efficient than the first star query.

The time taken to execute the second example query in the Oracle10g database with Star Transformation (using bitmap indexes) is 5 seconds.

Generic Database (Dynamic Bitmap Indexes)

In a database which lacks range partitioning, this new predicate on the TIME dimension table has relatively little impact on the overall performance. Unlike range partitioning, hash partitioning does not allow typical queries to take advantage of partition pruning.

Time taken to execute the second example query on the Generic database is 62 seconds.

Generic Database (Hash Join)

As earlier in the case of Basic Query, the execution time of this star query is based upon an execution plan which hash-joins all of the tables. This query is also not significantly impacted by the additional predicate on the TIME table.

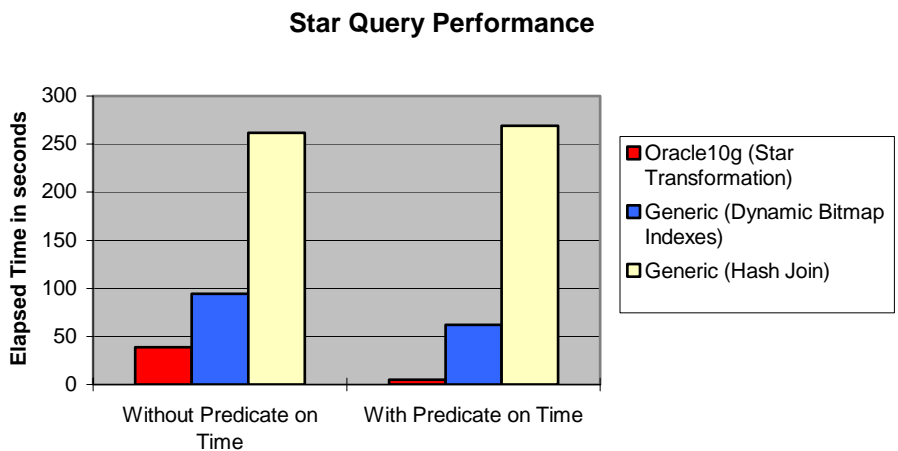
Time taken to execute the second example query on the Generic database (using hash join) is 269 seconds.

Results of Star Query Performance

Following table shows the elapsed time in seconds for Star Query Performance

STAR QUERY	Oracle10g	Generic (Dynamic bitmap indexes)	Generic (Hash Join)
Example #1	39	94	262
Example #2	5	62	269

The following chart summarizes the above results for star queries.



One key aspect to note about these query results is that dynamic bitmap indexes do not provide the same query performance Oracle10g’s real bitmap indexes. While dynamic bitmap indexes can be used in “star transformation” strategies for executing star queries, dynamic bitmap indexes are still based upon b-tree indexes and considerable IO costs are associated with accessing the much-larger b-tree indexes.

Additional simple queries

The following single-table queries on the SALES table illustrate the additional advantages of bitmap indexes over b-tree indexes. Three queries were tested:

1. `select count(*) from sales;`
2. `select count(*)
from sales
and promo_id = 714
and channel_id = 'S';`
3. `select count(*)
from sales
and promo_id = 714
and time_id = to_date('20-MAY-2004', 'DD-MON-YYYY')
and channel_id = 'S';`

Oracle10g Database

1. The first query uses the bitmap fast full index scan on the smallest bitmap index (SALES_PROMO_ID_BIX) and executes in 0.62 seconds. This query is very efficient because this query can be evaluated simply by reading a small portion of a bitmap index.
2. The second query uses bitmap indexes on the promo_id and the channel_id columns (SALES_PROMO_ID_BIX, SALES_CHANNEL_ID_BIX). Bitmap indexes are very efficient for AND'ing multiple predicates together. This query executes in 0.13 seconds.
3. The third query uses a similar strategy as the second query, except that this query also benefits from Oracle10g's partition-pruning. This query takes only 0.04 second to finish.

Generic Database

1. The first query uses a index fast full scan on the smallest index (SALES_CHANNEL_ID_IX) to get the result in 82 seconds. This query is less efficient because the b-tree index is so much larger than the bitmap index⁴.
2. The second query used an index range scan on the b-tree index SALES_PROMO_ID_IX and an access by ROWID on SALES_CHANNEL_ID_IX. The second query finishes in 77 seconds. As in the first query, the Generic Database is less efficient than the Oracle10g database in this query because of the overhead in processing the large b-tree indexes.
3. The third query uses index range scan, and accesses rows from the SALES table. This query finishes in 64 seconds.

⁴ Note again that the less than optimal storage configuration has resulted in a slower query execution for this scenario. However, even with an optimal storage configuration the query time would still not match the Oracle10g scenario.

Results of Simple Queries

The following table summarizes the results for the simple queries:

Simple Query	Elapsed Time (in seconds)	
	Oracle10g (A)	Generic (B)
Query 1	0.62	82
Query 2	0.13	77
Query 3	0.04	64

CONCLUSION

Using two key features, range partitioning and bitmap indexes, this paper demonstrates how Oracle10g can provide tremendous performance benefits over competing technological approaches. Not only do these performance benefits translate directly into higher end-user satisfaction (when the end-users queries complete more quickly), but these features require less disk space, cause less system utilization, and reduce maintenance processing which allows for data warehouses to be built and managed at lower costs. Oracle10g is the market-leading database for data warehousing, and this paper illustrates why more customers choose Oracle10g for their warehouse than any other database.

APPENDIX A: TEST SCHEMA

SQL> describe sales;

Column Name	Null?	Type
PROD_ID	NOT NULL	NUMBER(6)
CUST_ID	NOT NULL	NUMBER
TIME_ID	NOT NULL	DATE
CHANNEL_ID	NOT NULL	CHAR(1)
PROMO_ID	NOT NULL	NUMBER(6)
QUANTITY_SOLD	NOT NULL	NUMBER(3)
AMOUNT_SOLD	NOT NULL	NUMBER(10,2)

SQL> SELECT count(*) FROM sales;

```
      COUNT(*)
-----
292,282,479
```

SQL> describe products;

Column Name	Null?	Type
PROD_ID	NOT NULL	NUMBER(6)
PROD_NAME	NOT NULL	VARCHAR2(50)
PROD_DESC	NOT NULL	VARCHAR2(4000)
PROD_SUBCATEGORY	NOT NULL	VARCHAR2(50)
PROD_SUBCAT_DESC	NOT NULL	VARCHAR2(2000)
PROD_CATEGORY	NOT NULL	VARCHAR2(50)
PROD_CAT_DESC	NOT NULL	VARCHAR2(2000)
PROD_WEIGHT_CLASS		NUMBER(2)
PROD_UNIT_OF_MEASURE		VARCHAR2(20)
PROD_PACK_SIZE		VARCHAR2(30)
SUPPLIER_ID		NUMBER(6)
PROD_STATUS	NOT NULL	VARCHAR2(20)
PROD_LIST_PRICE	NOT NULL	NUMBER(8,2)
PROD_MIN_PRICE	NOT NULL	NUMBER(8,2)

SQL> SELECT count(*) FROM products;

```
      COUNT(*)
-----
10,000
```

SQL> describe promotions;

Column Name	Null?	Type
PROMO_ID	NOT NULL	NUMBER(6)
PROMO_NAME	NOT NULL	VARCHAR2(20)
PROMO_SUBCATEGORY	NOT NULL	VARCHAR2(30)
PROMO_CATEGORY	NOT NULL	VARCHAR2(30)
PROMO_COST	NOT NULL	NUMBER(10,2)
PROMO_BEGIN_DATE	NOT NULL	DATE
PROMO_END_DATE	NOT NULL	DATE

SQL> SELECT count(*) FROM promotions;

```
      COUNT(*)
-----
1,001
```

SQL> describe customers;

Column Name	Null?	Type
CUST_ID	NOT NULL	NUMBER
CUST_FIRST_NAME	NOT NULL	VARCHAR2(20)
CUST_LAST_NAME	NOT NULL	VARCHAR2(40)
CUST_GENDER		CHAR(1)
CUST_YEAR_OF_BIRTH		NUMBER(4)
CUST_MARITAL_STATUS		VARCHAR2(20)
CUST_STREET_ADDRESS	NOT NULL	VARCHAR2(40)
CUST_POSTAL_CODE	NOT NULL	VARCHAR2(10)
CUST_CITY	NOT NULL	VARCHAR2(30)

```
CUST_STATE_PROVINCE          VARCHAR2(40)
COUNTRY_ID                   NOT NULL CHAR(2)
CUST_MAIN_PHONE_NUMBER       VARCHAR2(25)
CUST_INCOME_LEVEL            VARCHAR2(30)
CUST_CREDIT_LIMIT            NUMBER
CUST_EMAIL                   VARCHAR2(30)
```

```
SQL> SELECT count(*) FROM customers;
```

```
      COUNT(*)
-----
      1,000,000
```

```

SQL> describe times;
Column Name                               Null?    Type
-----
TIME_ID                                   NOT NULL DATE
DAY_NAME                                  NOT NULL VARCHAR2(9)
DAY_NUMBER_IN_WEEK                        NOT NULL NUMBER(1)
DAY_NUMBER_IN_MONTH                       NOT NULL NUMBER(2)
CALENDAR_WEEK_NUMBER                     NOT NULL NUMBER(2)
FISCAL_WEEK_NUMBER                       NOT NULL NUMBER(2)
WEEK_ENDING_DAY                           NOT NULL DATE
CALENDAR_MONTH_NUMBER                    NOT NULL NUMBER(2)
FISCAL_MONTH_NUMBER                      NOT NULL NUMBER(2)
CALENDAR_MONTH_DESC                       NOT NULL VARCHAR2(8)
FISCAL_MONTH_DESC                        NOT NULL VARCHAR2(8)
DAYS_IN_CAL_MONTH                        NOT NULL NUMBER
DAYS_IN_FIS_MONTH                        NOT NULL NUMBER
END_OF_CAL_MONTH                          NOT NULL DATE
END_OF_FIS_MONTH                          NOT NULL DATE
CALENDAR_MONTH_NAME                       NOT NULL VARCHAR2(9)
FISCAL_MONTH_NAME                         NOT NULL VARCHAR2(9)
CALENDAR_QUARTER_DESC                    NOT NULL CHAR(7)
FISCAL_QUARTER_DESC                      NOT NULL CHAR(7)
DAYS_IN_CAL_QUARTER                      NOT NULL NUMBER
DAYS_IN_FIS_QUARTER                      NOT NULL NUMBER
END_OF_CAL_QUARTER                       NOT NULL DATE
END_OF_FIS_QUARTER                       NOT NULL DATE
CALENDAR_QUARTER_NUMBER                  NOT NULL NUMBER(1)
FISCAL_QUARTER_NUMBER                    NOT NULL NUMBER(1)
CALENDAR_YEAR                             NOT NULL NUMBER(4)
FISCAL_YEAR                               NOT NULL NUMBER(4)
DAYS_IN_CAL_YEAR                         NOT NULL NUMBER
DAYS_IN_FIS_YEAR                         NOT NULL NUMBER
END_OF_CAL_YEAR                           NOT NULL DATE
END_OF_FIS_YEAR                           NOT NULL DATE

```

```

SQL> SELECT count(*) FROM times;

```

```

          COUNT(*)
-----
          2,557

```

APPENDIX B: EXECUTION PLANS FOR QUERIES

Star Query Example #1: Oracle10g (Star Transformation)

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	Pstart	Pstop
0	SELECT STATEMENT		760	35720	11955 (3)	00:02:24		
1	TEMP TABLE TRANSFORMATION							
2	LOAD AS SELECT	SYS_TEMP_0FD9D6604_5CC9C4						
3	TABLE ACCESS FULL	PRODUCTS	270	13500	69 (2)	00:00:01		
4	PX COORDINATOR							
5	PX SEND QC (RANDOM)	:TQ10002	760	35720	11886 (3)	00:02:23		
6	HASH GROUP BY		760	35720	11886 (3)	00:02:23		
7	PX RECEIVE		6743	309K	11885 (3)	00:02:23		
8	PX SEND HASH	:TQ10001	6743	309K	11885 (3)	00:02:23		
9	HASH JOIN		6743	309K	11885 (3)	00:02:23		
10	BUFFER SORT							
11	PX RECEIVE		270	8640	2 (0)	00:00:01		
12	PX SEND BROADCAST	:TQ10000	270	8640	2 (0)	00:00:01		
13	TABLE ACCESS FULL	SYS_TEMP_0FD9D6604_5CC9C4	270	8640	2 (0)	00:00:01		
14	PX PARTITION RANGE ALL		188K	2761K	11882 (3)	00:02:23	1	36
15	TABLE ACCESS BY LOCAL INDEX ROWID	SALES	188K	2761K	11882 (3)	00:02:23	1	36
16	BITMAP CONVERSION TO ROWIDS							
17	BITMAP AND							
18	BITMAP MERGE							
19	BITMAP KEY ITERATION							
20	BUFFER SORT							
21	TABLE ACCESS FULL	CHANNELS	1	12	3 (0)	00:00:01		
22	BITMAP INDEX RANGE SCAN	CHANNEL_ID_INDX					1	36
23	BITMAP MERGE							
24	BITMAP KEY ITERATION							
25	BUFFER SORT							
26	TABLE ACCESS FULL	PROMOTIONS	125	1375	4 (0)	00:00:01		
27	BITMAP INDEX RANGE SCAN	PROMO_ID_INDX					1	36
28	BITMAP MERGE							
29	BITMAP KEY ITERATION							
30	BUFFER SORT							
31	TABLE ACCESS FULL	SYS_TEMP_0FD9D6604_5CC9C4	1	13	2 (0)	00:00:01		
32	BITMAP INDEX RANGE SCAN	PROD_ID_INDX					1	36

Star Query Example #1: Generic (Dynamic Bitmap Indexes)

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	Pstart	Pstop
0	SELECT STATEMENT		760	65360	1235G(100)	999:59:59		
1	TEMP TABLE TRANSFORMATION							
2	LOAD AS SELECT	SYS_TEMP_0FD9D6606_67EA71						
3	TABLE ACCESS FULL	PRODUCTS	270	13500	69 (2)	00:00:01		
4	PX COORDINATOR							
5	PX SEND QC (RANDOM)	:TQ10003	760	65360	1235G(100)	999:59:59		
6	HASH GROUP BY		760	65360	1235G(100)	999:59:59		
7	PX RECEIVE		760	65360	1235G(100)	999:59:59		
8	PX SEND HASH	:TQ10002	760	65360	1235G(100)	999:59:59		
9	HASH GROUP BY		107K	9010K	1235G(100)	999:59:59		
10	HASH JOIN							
11	BUFFER SORT							
12	PX RECEIVE		1	12	3 (0)	00:00:01		
13	PX SEND BROADCAST	:TQ10000	1	12	3 (0)	00:00:01		
14	TABLE ACCESS FULL	CHANNELS	1	12	3 (0)	00:00:01		
15	HASH JOIN		536K	37M	1235G(100)	999:59:59		
16	BUFFER SORT							
17	PX RECEIVE		270	8640	2 (0)	00:00:01		
18	PX SEND BROADCAST	:TQ10001	270	8640	2 (0)	00:00:01		
19	TABLE ACCESS FULL	SYS_TEMP_0FD9D6606_67EA71	270	8640	2 (0)	00:00:01		
20	PX PARTITION HASH ALL		12M	512M	1235G(100)	999:59:59	1	32
21	TABLE ACCESS BY LOCAL INDEX ROWID	SALES	12M	512M	1235G(100)	999:59:59	1	32
22	BITMAP CONVERSION TO ROWIDS							
23	BITMAP AND							
24	BITMAP MERGE							
25	BITMAP KEY ITERATION							
26	BUFFER SORT							
27	TABLE ACCESS FULL	SYS_TEMP_0FD9D6606_67EA71	1	13	2 (0)	00:00:01		
28	BITMAP CONVERSION FROM ROWIDS							
29	INDEX RANGE SCAN	PROD_ID_INDX			166 (2)	00:00:02	1	32
30	BITMAP MERGE							
31	BITMAP KEY ITERATION							
32	BUFFER SORT							
33	TABLE ACCESS FULL	PROMOTIONS	125	1375	4 (0)	00:00:01		
34	BITMAP CONVERSION FROM ROWIDS							
35	INDEX RANGE SCAN	PROMO_ID_INDX			135 (33)	00:00:02	1	32

Star Query Example #1: Generic (Hash Join)

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	Pstart	Pstop
0	SELECT STATEMENT		230	26450	117K (8)	00:23:34		
1	PX COORDINATOR							
2	PX SEND QC (RANDOM)	:TQ10003	230	26450	117K (8)	00:23:34		
3	HASH GROUP BY		230	26450	117K (8)	00:23:34		
4	PX RECEIVE		230	26450	117K (8)	00:23:34		
5	PX SEND HASH	:TQ10002	230	26450	117K (8)	00:23:34		
6	HASH GROUP BY		230	26450	117K (8)	00:23:34		
7	HASH JOIN		497K	54M	117K (8)	00:23:33		
8	BUFFER SORT							
9	PX RECEIVE		125	1375	4 (0)	00:00:01		
10	PX SEND BROADCAST	:TQ10000	125	1375	4 (0)	00:00:01		
11	TABLE ACCESS FULL	PROMOTIONS	125	1375	4 (0)	00:00:01		
12	HASH JOIN		3977K	394M	117K (8)	00:23:33		
13	BUFFER SORT							
14	PX RECEIVE		270	16740	72 (2)	00:00:01		
15	PX SEND BROADCAST	:TQ10001	270	16740	72 (2)	00:00:01		
16	MERGE JOIN CARTESIAN		270	16740	72 (2)	00:00:01		
17	TABLE ACCESS FULL	CHANNELS	1	12	3 (0)	00:00:01		
18	BUFFER SORT		270	13500	69 (2)	00:00:01		
19	TABLE ACCESS FULL	PRODUCTS	270	13500	69 (2)	00:00:01		
20	PX BLOCK ITERATOR	SALES	473M	18G	115K (6)	00:23:06	1	32
21	TABLE ACCESS FULL		473M	18G	115K (6)	00:23:06	1	32

Star Query Example #2: Oracle10g (Star Transformation)

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	Pstart	Pstop
0	SELECT STATEMENT		560	30800	2653 (19)	00:00:32		
1	TEMP TABLE TRANSFORMATION							
2	LOAD AS SELECT	SYS_TEMP_0FD9D6605_5CC9C4						
3	TABLE ACCESS FULL	PRODUCTS	270	13500	69 (2)	00:00:01		
4	FX COORDINATOR							
5	FX SEND QC (RANDOM)	:TQ10002	560	30800	2584 (19)	00:00:32		
6	HASH GROUP BY		560	30800	2584 (19)	00:00:32		
7	FX RECEIVE		560	30800	2583 (19)	00:00:32		
8	FX SEND HASH	:TQ10001	560	30800	2583 (19)	00:00:32		
9	HASH JOIN		560	30800	2583 (19)	00:00:32		
10	BUFFER SORT							
11	FX RECEIVE		270	8640	2 (0)	00:00:01		
12	FX SEND BROADCAST	:TQ10000	270	8640	2 (0)	00:00:01		
13	TABLE ACCESS FULL	SYS_TEMP_0FD9D6605_5CC9C4	270	8640	2 (0)	00:00:01		
14	FX PARTITION RANGE SUBQUERY		15654	351K	2581 (19)	00:00:31	KEY	KEY
15	TABLE ACCESS BY LOCAL INDEX ROWID	SALES	15654	351K	2581 (19)	00:00:31	KEY	KEY
16	BITMAP CONVERSION TO ROWIDS							
17	BITMAP AND							
18	BITMAP MERGE							
19	BITMAP KEY ITERATION							
20	BUFFER SORT							
21	TABLE ACCESS FULL	CHANNELS	1	12	3 (0)	00:00:01		
22	BITMAP INDEX RANGE SCAN	CHANNEL_ID_INDX					KEY	KEY
23	BITMAP MERGE							
24	BITMAP KEY ITERATION							
25	BUFFER SORT							
26	TABLE ACCESS FULL	TIMES	91	1456	7 (0)	00:00:01		
27	BITMAP INDEX RANGE SCAN	TIME_ID_INDX					KEY	KEY
28	BITMAP MERGE							
29	BITMAP KEY ITERATION							
30	BUFFER SORT							
31	TABLE ACCESS FULL	PROMOTIONS	125	1375	4 (0)	00:00:01		
32	BITMAP INDEX RANGE SCAN	PROMO_ID_INDX					KEY	KEY
33	BITMAP MERGE							
34	BITMAP KEY ITERATION							
35	BUFFER SORT							
36	TABLE ACCESS FULL	SYS_TEMP_0FD9D6605_5CC9C4	1	13	2 (0)	00:00:01		
37	BITMAP INDEX RANGE SCAN	PROD_ID_INDX					KEY	KEY

Star Query Example #2: Generic (Dynamic Bitmap Indexes)

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	Pstart	Pstop
0	SELECT STATEMENT		760	80560	184G(100)	999:59:59		
1	TEMP TABLE TRANSFORMATION							
2	LOAD AS SELECT	SYS_TEMP_0FD9D6608_67EA71						
3	TABLE ACCESS FULL	PRODUCTS	270	13500	69 (2)	00:00:01		
4	FX COORDINATOR							
5	FX SEND QC (RANDOM)	:TQ10004	760	80560	184G(100)	999:59:59		
6	HASH GROUP BY		760	80560	184G(100)	999:59:59		
7	FX SEND HASH	:TQ10003	760	80560	184G(100)	999:59:59		
8	HASH JOIN		760	80560	184G(100)	999:59:59		
9	HASH GROUP BY		760	80560	184G(100)	999:59:59		
10	HASH JOIN		13411	1388K	184G(100)	999:59:59		
11	BUFFER SORT							
12	FX RECEIVE		1	12	3 (0)	00:00:01		
13	FX SEND BROADCAST	:TQ10000	1	12	3 (0)	00:00:01		
14	TABLE ACCESS FULL	CHANNELS	1	12	3 (0)	00:00:01		
15	HASH JOIN		67053	6155K	184G(100)	999:59:59		
16	BUFFER SORT							
17	FX RECEIVE		125	1375	4 (0)	00:00:01		
18	FX SEND BROADCAST	:TQ10001	125	1375	4 (0)	00:00:01		
19	TABLE ACCESS FULL	PROMOTIONS	125	1375	4 (0)	00:00:01		
20	HASH JOIN		536K	42M	184G(100)	999:59:59		
21	BUFFER SORT							
22	FX RECEIVE		270	8640	2 (0)	00:00:01		
23	FX SEND BROADCAST	:TQ10002	270	8640	2 (0)	00:00:01		
24	TABLE ACCESS FULL	SYS_TEMP_0FD9D6608_67EA71	270	8640	2 (0)	00:00:01		
25	FX PARTITION HASH ALL		12M	621M	184G(100)	999:59:59	1	32
26	TABLE ACCESS BY LOCAL INDEX ROWID	SALES	12M	621M	184G(100)	999:59:59	1	32
27	BITMAP CONVERSION TO ROWIDS							
28	BITMAP AND							
29	BITMAP MERGE							
30	BITMAP KEY ITERATION							
31	BUFFER SORT							
32	TABLE ACCESS FULL	SYS_TEMP_0FD9D6608_67EA71	1	13	2 (0)	00:00:01		
33	BITMAP CONVERSION FROM ROWIDS							
34	INDEX RANGE SCAN	PROD_ID_INDX			166 (2)	00:00:02	1	32
35	BITMAP MERGE							
36	BITMAP KEY ITERATION							
37	BUFFER SORT							
38	TABLE ACCESS FULL	TIMES	91	1456	7 (0)	00:00:01		
39	BITMAP CONVERSION FROM ROWIDS							
40	INDEX RANGE SCAN	TIME_ID_INDX			129 (35)	00:00:02	1	32

Star Query Example #2: Generic (Hash Join)

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	Pstart	Pstop
0	SELECT STATEMENT		230	32200	117K (8)	00:23:34		
1	FX COORDINATOR							
2	FX SEND QC (RANDOM)	:TQ10004	230	32200	117K (8)	00:23:34		
3	HASH GROUP BY		230	32200	117K (8)	00:23:34		
4	FX RECEIVE		230	32200	117K (8)	00:23:34		
5	FX SEND HASH	:TQ10003	230	32200	117K (8)	00:23:34		
6	HASH GROUP BY		230	32200	117K (8)	00:23:34		
7	HASH JOIN		41281	5643K	117K (8)	00:23:34		
8	BUFFER SORT							
9	FX RECEIVE		125	1375	4 (0)	00:00:01		
10	FX SEND BROADCAST	:TQ10000	125	1375	4 (0)	00:00:01		
11	TABLE ACCESS FULL	PROMOTIONS	125	1375	4 (0)	00:00:01		
12	HASH JOIN		330K	40M	117K (8)	00:23:34		
13	BUFFER SORT							
14	FX RECEIVE		91	1456	7 (0)	00:00:01		
15	FX SEND BROADCAST	:TQ10001	91	1456	7 (0)	00:00:01		
16	TABLE ACCESS FULL	TIMES	91	1456	7 (0)	00:00:01		
17	HASH JOIN		3977K	428M	117K (8)	00:23:33		
18	BUFFER SORT							
19	FX RECEIVE		270	16740	72 (2)	00:00:01		
20	FX SEND BROADCAST	:TQ10002	270	16740	72 (2)	00:00:01		
21	MERGE JOIN CARTESIAN		270	16740	72 (2)	00:00:01		
22	TABLE ACCESS FULL	CHANNELS	1	12	3 (0)	00:00:01		
23	BUFFER SORT		270	13500	69 (2)	00:00:01		
24	TABLE ACCESS FULL	PRODUCTS	270	13500	69 (2)	00:00:01		
25	FX BLOCK ITERATOR		473M	22G	115K (6)	00:23:06	1	32
26	TABLE ACCESS FULL	SALES	473M	22G	115K (6)	00:23:06	1	32

APPENDIX C: INIT.ORA PARAMETERS

The relevant init.ora parameters that are common to both database are listed below:

```
compatible=10.2.0.0.0
db_block_size=8192
db_file_multiblock_read_count=64
sga_target=512M
pga_aggregate_target=1G
open_cursors=100
processes=1024
sessions=1024
transactions_per_rollback_segment=1
open_links_per_instance=1
parallel_min_servers=4
parallel_max_servers=8
star_transformation_enabled=TRUE
undo_management='AUTO'
undo_tablespace='UNDOTBS1'
```

APPENDIX D: TABLE SELECTIVITY IN STAR QUERIES

Following table lists the selectivity of the tables based on the predicates of the star queries listed in the Query Performance section.

Total rows in SALES fact table = 292,282,479

Dimension Table	Predicate	Selectivity (in Dimension table)	Selectivity (in SALES fact table)	Cumulative Selectivity (in SALES table)
CHANNELS	channel_desc = 'Catalog'	1/5 (20%)	35,073,898 (12%)	35,073,898 (12%)
PRODUCTS	prod_subcategory = 'Shorts - Men'	143/10000 (1.43%)	4,793,316 (1.64%)	532,586 (0.18%)
PROMOTIONS	promo_category = 'flyer'	80/1001 (8%)	3,985,168 (1.36%)	8,032 (0.0027%)
TIMES	calendar_quarter_desc = '2000-Q2'	91/2557 (3.55%)	24,667,000 (8.44%)	1,470 (0.0005%)



Key Data Warehousing Features in Oracle10f

April 2005

Author: Neil Thombre, Cetin Ozbutun

Contributing Authors: Linan Jiang, Mark Van de Wiel

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
oracle.com

Copyright © 2005, Oracle. All rights reserved.

This document is provided for information purposes only and the contents hereof are subject to change without notice.

This document is not warranted to be error-free, nor subject to any other warranties or conditions, whether expressed orally or implied in law, including implied warranties and conditions of merchantability or fitness for a particular purpose. We specifically disclaim any liability with respect to this document and no contractual obligations are formed either directly or indirectly by this document. This document may not be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without our prior written permission. Oracle, JD Edwards, and PeopleSoft are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.