Data Warehouse Performance Enhancements with Oracle9*i*

An Oracle White Paper
April 2001

ORACLE

Data Warehouse Performance Enhancements with Oracle9*i*

INTRODUCTION

Oracle9*i* provides significant performance improvements for basic data warehouse (DWH) functionality. Since no schema or application code has to be changed to leverage most of these improvements, nearly all customers will benefit from them simply by upgrading their Oracle installation. This paper illustrates the Oracle9*i* enhancements by performing typical data warehousing tasks on a test data warehouse and comparing Oracle8*i* and Oracle9*i* results. All the tasks were performed on a single carefully controlled system, ensuring a fair comparison between Oracle8*i* and Oracle9*i*. From loading data, changing it through a transformation process, creating materialized views, refreshing them and finally accessing the data, we find that Oracle9*i* significantly improves performance and system utilization.

RESOURCE REQUIREMENTS AND DEPENDENCIES

The tests comparing Oracle8*i* to Oracle9*i* were done on the same hardware system, using the same amount of memory, I/O subsystem and number of CPUs. Each test was monitored to prove that the results are correct and were not skewed by other workloads running at the same time. All timing information is presented in the format mi:ss.

Hardware

System Configuration: Sun Microsystems sun4u 8-slot Sun Enterprise 4000/5000

- 12 x 400 MHZ CPUs
- 10 GB Memory
- 128 x Disks, Raw Devices, 64KB Stripe Unit Size with a maximum of 90 MB/sec. throughput

Database

Two 30 GB databases based on a TPC-H schema were set up with equivalent initialization parameters (init.ora), except those parameters that were necessary to enable the new features.

- Oracle9*i* (9iopt), Total System Global Area 662,995,316 Bytes
- Oracle8*i* (8iopt), Total System Global Area 616,460,448 Bytes

BASIC PERFORMANCE ENHANCEMENTS

This paper shows that performance improvements with Oracle9*i* are not exclusively related to new features that need to be explicitly activated. Many of the improvements will be experienced without any change to application code or database schema. For that reason many statements running in existing Oracle8*i* DWH environments will automatically benefit after the migration to Oracle9*i*. We will go through a data warehouse process, starting with the loading, transformation and building phase, then aggregate data in Materialized Views, refresh these views and access data with typical query statements. At the end we will examine how long the complete process takes in Oracle8*i* and Oracle9*i*, presenting the performance improvement for each of the enhancements.

Merge / Upsert

Merge / Upsert Enhancements

In a data warehouse environment, tables (typically fact tables) need to be refreshed periodically with new data arriving from on-line systems. This new data may contain changes to existing rows in tables of the warehouse and/or new rows that need to be inserted. If a row in the new data corresponds to an item that already exists in the table, an UPDATE is performed; if the row's primary key does not exist in the table, an INSERT is performed. Prior to Oracle9*i* these operations were expressed either as a sequence of DMLs (INSERT/UPDATE) or as PL/SQL loops deciding, for each row, whether to insert or update the data. But both approaches often suffer from deficiencies in performance and usability. By extending SQL with a new syntax - the MERGE-statement - which combines the sequence of a conditional INSERT and UPDATE in one single atomic statement, Oracle9*i* overcomes these deficiencies and makes the implementation of warehousing applications more simple and intuitive.

The 'IF ROW EXISTS -THEN-UPDATE ELSE-INSERT' - conditional sequence is also referred to as UPSERT.

Merge / Upsert Results

The performance improvements which can be achieved by using the new atomic MERGE-statement for the upsert process in Oracle9*i*, instead of using a sequence of conditional INSERT and UPDATE-statements with the same semantics, depend mainly on the size of the source data containing the new rows to be merged, compared to the size of the destination table in which the new arrivals will be merged. Changing many rows of the destination table will benefit more from the upsert than changing just a few rows. If the upsert is performed in one MERGE-statement, the source table has only to be scanned once. Whereas using a sequence of INSERT/UPDATE the source table has to be scanned twice.

If further constraints like additional procedural checks or transformations, which can not be expressed in standard SQL, are necessary, performing a merge in PL/SQL is from a performance perspective by far the least attractive choice. The PL/SQL-method is significantly slower than MERGE or UPSERT/INSERT which both use set-oriented processing logic.

If the ETL (Extraction/Transformation/Loading) process requires a procedural transformation before the upsert, using the new parallelizable PL/SQL table functions in stream mode and pipelining the results to a subsequent Merge operation may result in significant performance gains.

In our tests we compared an upsert of 3 million new customer records into a customer table containing 4.5 million rows. The new MERGE-statement running in Oracle9*i* was compared to a sequence of conditional UPDATE and INSERT-statements running in Oracle8*i*. In both cases 1.5 million records were inserted and 1.5 million records were updated. In both scenarios Oracle's parallel DML capability was used to execute the statement with a parallel degree of 16. Apart from the performance gain due to minimized scanning of the source table, internally optimized index maintenance operations for rows inserted with parallel DML also contribute to the performance improvement. The same internal optimization also leads to performance improvements with 'normal' parallelized INSERTs in Oracle9*i* compared to the same INSERT-statement in Oracle8*i*.

Test	Oracle8i		Oracle9i		Performance	
					Diff.	%Gain
Upsert	update/insert	03:28	merge	02:24	01:04	30.7%

Figure 1: Upsert test results

External Tables

External Tables Enhancement

In Oracle9*i* the new External Table feature enables the use of external data as a 'virtual table'. With this interface, external data can be queried in parallel and joined directly to internal, regular tables without requiring the external data to be loaded into the database beforehand.

Using External Tables enables pipelining of ETL-processes, so that the transformation phase will not be blocked by the loading phase. The transformation process can be merged with the loading process without any interruption of the data streaming. It is no longer necessary to store the data in a staging area for comparison or transformation purposes.

The main difference between External Tables and regular tables is, that externally organized tables are read-only. No DML operations are possible and no indexes can be created on them. In Oracle9*i* External Tables are a complement to the existing SQL*Loader functionality. They are especially useful for environments where the complete external source has to be joined with existing database objects and transformed in a complex manner or where the external data volume is large and used only once.

SQL*Loader, on the other hand, might still be the better choice for loading data where additional indexing of the staging table is necessary. This is true for operations where the data is used in independent complex transformations or the data is only partially used in further processing.

External Tables Results

The test done for External Tables was complementary to the upsert test.
In Oracle8*i* 3 steps need to be performed to upload new records to a warehouse schema:

1. Load the data from external files into a database staging table (CUSTOMER_MERGE in our case)
2. Create a unique index on the staging table (required for update)
3. Merge the data to the destination table (CUSTOMER) using UPDATE/INSERT

In Oracle9*i* the whole process can be reduced to 2 steps:

1. Create External Table metadata in the database (CREATE TABLE … ORGANIZATION external)
2. Use the External Table (CUSTOMER_MERGE) directly as source for the Oracle9*i* MERGE-statement.

The Oracle9*i* Step 1 above is easy, since SQL*Loader generates the complete CREATE TABLE-syntax on demand based on an existing Oracle8*i* loader control file. Using the Oracle9*i* approach, two maintenance tasks that were necessary in Oracle8*i* can be omitted: data load and index creation. Unlike the UPDATE-statement, the MERGE-statement requires no unique index on the source table.

In our scenario the time for Oracle8*i* loading and indexing steps was:

1. Loading (3 million rows, 16-fold direct parallel) 19 seconds
2. Indexing stage table (16-fold parallel) 14 seconds

Thus 33 seconds are saved by using Oracle9*i* External Tables. The test also proved that using the External Table as a direct source instead of an internal table for the MERGE resulted only in a very modest performance degradation: the MERGE finished in 2:27 compared to 2:24 using the internal table.

Test	Oracle8*i* Style		Oracle9*i* Style		Performance	
					Diff.	%Gain
External	load+index+ upsert (*)	02:57	upsert external	02:27	00:30	16.9 %

Figure 2: External Tables test results (*): results taken from Oracle9i-MERGE for fair comparison

Group By

Variable Length Aggregates

GROUP BY is a very important basic operation for DWH applications. In Oracle9*i* the GROUP BY has been enhanced to use Variable Length Aggregates.

In Oracle8*i*, the execution engine uses Oracle numbers as aggregation accumulators during GROUP BY, Cube and Rollup processing. The sort engine is populated with records constructed by concatenation of GROUP BY keys and a list of aggregate work areas. The aggregation happens as the sort operation consumes rows with the same GROUP BY keys. Oracle8*i* aggregates use 23 bytes for SUM, 31 bytes for AVG and 54 bytes for VARIANCE to accumulate the values being aggregated. The sort records can grow in size, which, in turn leads to a quick exhaustion of the in-memory sort area.

For the majority of applications the use of full length Oracle numbers for aggregations is clearly a waste of space. If we know that a specific Oracle number may only require 7 bytes instead of 22 bytes and reduce the aggregate work area(s), a 3-fold space gain can be achieved. This may directly translate to performance gains. By utilizing space so much more efficiently, the sort engine may never spill to disk, but even if it does, it would read and write 3 times less data in this example.

Aggregation Test Results

As the number of groups processed in a GROUP BY query increases, so does the advantage of Oracle9*i* over Oracle8*i*. For a very small number of groups in a query, the difference between the two releases is not noteworthy. We chose a query with 8 aggregate operators (GroupBy.sql) for our test and fixed the amount of memory to be used by setting SORT_AREA_SIZE to 20MB and 30MB. With 20MB, both Oracle8*i* and Oracle9*i* spilled to disk. In this case, Oracle9*i* used almost half the disk space that Oracle8*i* did, and was 10% faster. With 30MB, Oracle9*i* fit in memory, whereas Oracle8*i* still spilled to disk, and in this case, Oracle9*i* was 20% faster. The degree of parallelism we used in both cases was 24, which means that each of the 24 processes used 20MB of memory in the first case, and 30MB of memory in the second scenario.

Test			Oracle8*i*		Oracle9*i*		Performance	
	DOP	Sort	Temp	Time	Temp	Time	Diff.	%Gain
1	24	20M	1.35G	02:46	0.75G	02:29	00:17	10.2%
2	24	30M	1.12G	02:29	-	01:56	00:30	20.1%

Figure 3: GROUP BY test results. DOP = Degree of Parallelism,
Sort = SORT_AREA_SIZE

Dynamic Memory Management

PGA Memory for dedicated servers

Prior to Oracle9*i*, memory work areas like SORT_AREA_SIZE, HASH_AREA_SIZE, BITMAP_MERGE_AREA_SIZE and CREATE_BITMAP_AREA_SIZE had to be set manually. Depending on the kind of application running on the system and the workload, a fixed setting for these parameters could have caused problems. When the setting was too low, Oracle could not perform time consuming operations in memory and had to write to disk. When the setting was too high and too many processes consumed too much memory, the operation system could potentially end up in swapping memory to disk. In both cases the user would have to face a performance degradation. Since applications on a database can have very different profiles in terms of memory consumption, the task of setting these parameters correctly turned out to be quite difficult.

Oracle9*i* introduces Dynamic Memory Management. To enable this mode, the DBA simply needs to specify the total size of PGA memory for the Oracle instance (Dedicated Server only). This is done by setting the new initialization parameter PGA_AGGREGATE_TARGET. The specified number (e.g. 2G) is a global target for the Oracle instance. Oracle controls memory consumption across all database server processes and guarantees that the target will not be exceeded. Within this target, when the workload is low, the work processes can get the optimal amount of PGA memory.

Dynamic Memory Management Results

We created different statements based on joins (HASH_AREA_SIZE) and GROUP BY aggregations (SORT_AREA_SIZE). The first statement Dyn_01.sql is very similar to the test cases done for GROUP BY (see GroupBy.sql) except that it reads more partitions from Lineitem. The other statements are based on Single or Joined Aggregated Materialized Views (Mav10.sql, Mav12.sql, Mav30.sql). During the tests, we observed that the new feature Variable Length Aggregates cross benefits from Dynamic Memory Management by processing many sort operations completely in memory. With both features enabled we could see many more cases where Oracle8*i* had to write sort information to disk and Oracle9*i* did not. Mav10.sql is an Aggregated Materialized Join View, joining Lineitem and Parts (hash join) and having a lot of AVG, SUM and VARIANCE aggregations with predicates on Parts and Lineitem, resulting in 2.4M rows. Mav12.sql is also an Aggregated Materialized Join View on Lineitem and Parts with no predicates, resulting in 2.5M rows. Mav30.sql is a Single Table Aggregated Materialized View on Orders containing 3M rows.

Test			Oracle8*i*		Oracle9*i*		Performance	
	DOP	Sort	Temp	Time	Temp	Time	Diff.	%Gain
Dyn_01	12	20M	1.74G	03:26	-	02:47	00:39	18.9%
Mav10	12	20M	1.71G	02:13	-	01:38	00:35	26.3%
Mav12	12	20M	12.41G	35:02	5.01G	23:04	10:58	31.2%
Mav30	12	10M	7.21G	05:53	-	03:50	02:03	34.8%

Figure 4: Dynamic Memory Management test results. DOP = Degree of Parallelism, Sort = SORT_AREA_SIZE (only for Oracle8i)

Materialized Views, Fast Refresh

Optimized Execution Plan for Fast Refresh

Fast Refresh for all different types of Materialized Views has been improved in Oracle9*i*. Materialized Join Views and Aggregated Materialized Join Views benefit most from this improvement. An optimal execution plan has a big impact on the execution time for a Fast Refresh. Prior to Oracle9*i* the refresh routines contained hard coded hints. Depending on the amount of data that had to be refreshed, the execution plan may not be optimal in some cases. The problem is that statistics on most of the objects accessed by a Fast Refresh, like all_sumdelta, Materialized Views or snapshots, are missing. Before executing the Fast Refresh, cardinality statistics are computed on the fly and used by the optimizer, so that Oracle9*i* can intelligently optimize the operation for best performance.

Fast Refresh Results

Many optimizations and improvements have been implemented in Oracle9*i*. It is now possible to perform a Fast Refresh for all types of operations on all types of Materialized Views. In Oracle8*i* some restrictions exist, e.g. only 'insert append' or 'direct loads' will work for Fast Refresh on Aggregated Materialized Join Views. For fair comparison, MAV12 is used only for a small insert, the second result (insert of 179,000 rows) is taken from a refresh of Aggregated Single Table Materialized View (MAV40).

Test	Oracle8*i*		Oracle9*i*		Performance 9*i*	
		Time		Time	Diff.	%Gain
3Rows(Mav12)		01:20		00:20	00:60	75.0%
179,000 Rows (Mav40)		00:57		00:22	00:35	61.4%

Figure 5: Results of Fast Refresh

PL/SQL

PL/SQL Enhancements

Oracle9i has important PL/SQL performance enhancements. In addition to PL/SQL performance enhancements which require changes to existing application code - like table functions, extended bulk operations, etc. - there are also some significant performance improvements which are transparent to existing applications. The most important are:

- optimized execution of user-defined functions in SQL-statements
- native compilation of PL/SQL

Optimized execution of user-defined functions in SQL-statements:

When a user-defined PL/SQL-function is embedded in a SQL-statement, a context switch between the SQL-statement executor and the PL/SQL-engine happens for each matching row of the query. If there are many matching rows, the invocation time for the function becomes a significant portion of the overall execution time. In Oracle9*i* this invocation mechanism has been thoroughly optimized resulting in a significant reduced number of instructions.

Native compilation of PL/SQL:

In Oracle8*i*, PL/SQL program units are compiled to byte code at compile time. The byte code has to be interpreted by the PL/SQL-engine during execution. This mechanism is still available in Oracle9*i*. But now PL/SQL program units can also be compiled natively. Native compilation was first introduced to stored Java program units in Oracle8*i* and has been applied to PL/SQL in Oracle9*i*. After a PL/SQL library unit has been natively compiled, it is stored as a shared library in the file system. At execution time, the shared library is dynamically linked to the executing Oracle process. Native compilation will in particular benefit PL/SQL programs that handle complex programming logic (complex loops, complex computations, etc.). In this situation, a performance improvement of factor 2 - 5 is realistic.

PL/SQL Results

Our case study shows the Oracle9*i* performance improvement for PL/SQL functions embedded into SQL. A simple function (function.sql) which computes the discounted order price given the extended price and discount rate stored in the retrieved rows, was applied in the SELECT-list of the statement for 200,000 and 400,000 records.

The invocation time for the PL/SQL-function made up a significant part of the overall execution time, because the function was simple, consisting of only one instruction. The test showed 41% overall performance improvement for this statement. The default byte code compilation was used. Native compilation would have improved the overall performance only slightly in this test case due to the simplicity of the function.

Test	Oracle8*i*		Oracle9*i*		Performance 9*i*	
		Time		Time	Diff.	%Gain
200,000 Rows		00:40		00:24	00:16	40.0%
400,000 Rows		01:21		00:48	00:33	40.7%

Figure 6: PL/SQL test results

Index Skip Scan

Index Skip Scan Enhancements

In releases prior to Oracle9*i*, a composite index would only be used if either the leading index prefix column was included in the predicate of the statement or an index full scan was executed. With

Oracle9*i*, the optimizer can use a composite index even if the prefix column value is not being used in the query. The optimizer uses an algorithm called 'Index Skip Scan' to retrieve ROWIDs for values that do not use the prefix column. Skip scans reduce the need to add an index to support occasional queries which do not reference the prefix column of an existing index. This is useful when a lot of change activity takes place since the existence of too many indexes degrade the performance of DML-operations. The algorithm is also advantageous in cases where no clear strategy can be found for selecting the columns to use as the prefix columns in a composite index.

The prefix column should be the most discriminating, but also the most frequently referenced in queries. Sometimes, these two requirements are met by two different columns in a composite index, forcing a compromise or the use of multiple indexes.

During a skip scan, the B*-tree is being probed for each distinct value in the prefix column. For each prefix column value, the normal search algorithm is performed. The result is a series of searches through subsets of the index, each of which appear to result from a query using a specific value of the prefix column.

Index Skip Scan Results

A concatenated index, I_ORDERS_SKIP on ORDERS table, consisting of columns O_ORDERPRIORITY, O_ORDERSTATUS, O_TOTALPRICE was created (index_skip.sql). The cardinality of the table ORDERS is 45,000,000. The number of distinct values of each indexed column is:

O_ORDERPRIORITY	:	5
O_ORDERSTATUS	:	3
O_ORDERTOTAL_PRICE	:	31078592

The following query was tested against the table:

```
SELECT      o_orderpriority,
            o_orderstatus,
            SUM(o_totalprice)
FROM        orders
WHERE       o_orderstatus = 'F' AND
            o_totalprice BETWEEN 0 AND 2000
GROUP BY    o_orderpriority, o_orderstatus
```

The query filtered 64,771 rows (0.14%). The results below show the immense value of the new access path.

Test	Oracle8*i*		Oracle9*i*		Performance 9*i*	
					diff.	%Gain
64,000 R.	serial index full scan	04:22	skip scan	00:02	04:20	99.2%
64,000 R.	parallel index full scan	00:23	skip scan	00:02	00:21	91.3%
64,000 R.	parallel full table scan	01:04	skip scan	00:02	01:02	96.8%

Figure 7: Index Skip Scan test results

Bitmap Join Index

Bitmap Join Index Enhancements

Oracle has supported persistent bitmap indexes as well as B*-tree indexes since Oracle 7.3. In all releases since then, bitmap indexes have been enhanced significantly. The major enhancement for Oracle9*i* is the ability to build a bitmap index on a table based on columns of another table. This index type is called a Bitmap Join Index. A Bitmap Join Index can be a single- or multi-column index and can combine columns of different tables. Bitmap Join Indexes materialize precomputed join results in a very efficient way. Typical usage in a data warehouse would be to create Bitmap Join Indexes on a fact table in a star or snow flake schema on one or more columns of one or more dimension tables. This could improve star query processing times dramatically, especially when a star query has filter predicates on low cardinality attributes of different dimension tables and the combination of these attributes is highly selective on the fact table.

Compared to Materialized Join Views in Oracle8*i*, Bitmap Join Indexes are much more space efficient because they store the ROWIDs in a compressed form. Another possible advantage is the fact that Bitmap Join Indexes can be combined with other bitmap indexes on the same table to process a complex query and they are maintained automatically like any other index (but unlike Materialized Views, which require a refresh). Therefore, in many cases, the use of Bitmap Join Indexes might be the better choice in Oracle9 *i*.

Bitmap Join Index Results

In our test case, we created a combined Bitmap Join Index on table PARTSUPP based on the dimension table columns PARTS (P_TYPE) and SUPPLIER(S_NATIONKEY). The table cardinalities were :

SUPPLIER	:	6,000,000 rows
PARTS	:	300,000 rows
PARTSUPP	:	24,000,000 rows

The column cardinalities were:

PARTS(P_TYPE)	:	150 distinct values
SUPPLIER(S_NATIONKEY)	:	25 distinct values

Query issued:

```
SELECT  COUNT (DISTINCT ps_suppkey),
                AVG (ps_supplycost),
                MAX (ps_supplycost),
                MIN (ps_supplycost)
FROM     partsupp ps, supplier s, parts p
WHERE   ps.ps_suppkey = s.s_suppkey     AND
                ps.ps_partkey=p.p_partkey  AND
                s.s_nationkey   = 24   AND
                p.p_type='medium burnished steel';
```

The query returned 6475 rows on PARTSUPP. It was run with a parallel degree of 16. Oracle9*i* used a Bitmap Join Index access path. The access path giving the best results in Oracle8*i* was a hash join of all three tables, chosen by the optimizer by default. Note that the 00:07 seconds achieved in Oracle9*i* was the result when reading all blocks from disk; with all blocks cached, the query finished in 00:02 seconds.

Test	Oracle8*i*		Oracle9*i*		Performance	
					Diff.	%Gain
Bitmap Join Index		01:02		00:07	00:55	88.7%

Figure 8: Bitmap Join Index test results

CONCLUSION

The results of the Oracle9*i* versus Oracle8*i* comparison presented in this paper show that Oracle9*i* improves the performance for all test cases. The tasks tested are very common and are normally executed when loading, transforming, aggregating, refreshing and selecting data in a data warehouse. A customer will benefit from basic performance enhancements like Variable Length Aggregates, Dynamic Memory Management, Fast Refresh and PL/SQL performance enhancements without changing the schema or the application code. Applying features like Index Skip Scan, External Tables, Merge and Bitmap Join Indexes will improve the performance even more.

Using Oracle8*i* with our test data set and hardware, a data loading, aggregating and refreshing cycle would require 55 minutes and 14 seconds. Here are the activities performed:

- External Table/direct Upsert	04:01
- Variable Length Aggregates	05:15 (sum of times for 2 test cases)
- Create Materialized View (mav12)	35:02
- Create Materialized View (mav30)	05:53
- Fast Refresh	02:17 (sum of times for 2 test cases)
- PL/SQL	01:21 (second test case)
- Index Skip Scan	00:23 (parallel index full scan)
- Join (without bitmap join index)	01:02

In Oracle9*i*, the same tasks would require 35 minutes and 18 seconds:

- External Table/direct Upsert	02:27
- Variable Length Aggregates	04:25
- Create Materialized View (mav12)	23:04
- Create Materialized View (mav30)	03:50
- Fast Refresh	00:42 (sum of times for 2 test cases)
- PL/SQL	00:48 (second test case)
- Index Skip Scan	00:02
- Join (with bitmap join index)	00:07

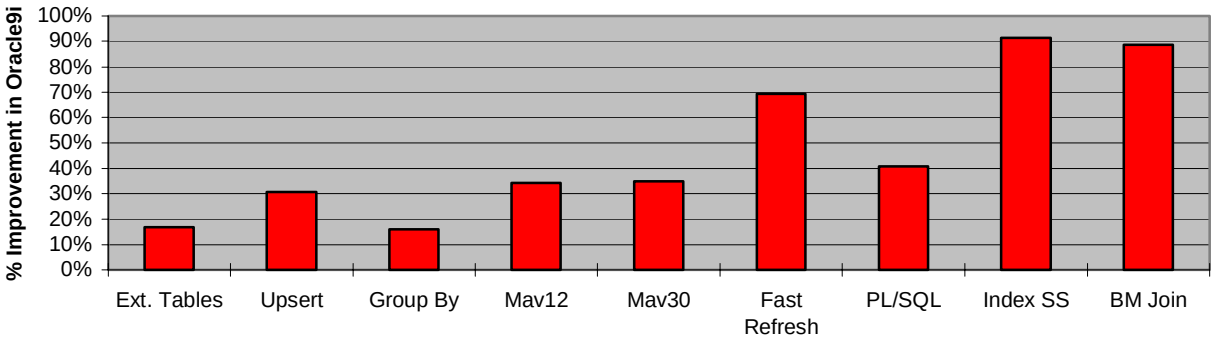

Figure 9: Performance enhancement of Oracle9*i* for all test results

We are confident that the performance benefits of Oracle9 *i* over Oracle8*i* will be similar or even better for larger scale data warehouses than with the 30GB database we used in this study. For long-running operations such as materialized view creation, the performance enhancements translate to very significant time savings for very large data warehouses - likely hours of time saved for every cycle of a large data warehouse. With the major time savings on long-running operations, windows for refresh processing can be reduced, enhancing data availability for users. In addition, data can be refreshed more frequently, enhancing data currency for better business decision making. Very large data warehouses will also benefit from the disk savings enabled by bitmap join indexes and index skip scan.

Oracle has long been the leading database for data warehouses. The dozens of data warehousing enhancements added to Oracle starting with Oracle 7.3 have created a rich and robust feature set for decision support tasks. The important performance enhancements of Oracle9 *i* extend Oracle's leadership, making Oracle an even stronger platform for data warehousing.

APPENDIX

Initialization Files

init8iopt.ora

```
*********************************************************************************************
db_file_multiblock_read_count    = 64
audit_trail                      = FALSE
compatible                       = 8.1.7
control_files = ("/private5/mbender/dbs/control8iopt1.ora",
                 "/private5/mbender/dbs/control8iopt2.ora")
db_block_buffers                 = 20000
db_block_size                    = 8192
db_files                         = 1023
db_file_multiblock_read_count    = 64
db_name                          = 8iopt
distributed_transactions         = 20
dml_locks                        = 100000
enqueue_resources                = 50000
hash_area_size                   = 10000000
job_queue_processes              = 2
large_pool_size                  = 200000000
log_buffer                       = 8388608
max_dump_file_size               = 500000
max_rollback_segments            = 650
nls_date_format                  = YYYY-MM-DD
open_cursors                     = 1024
optimizer_index_cost_adj         = 20
parallel_broadcast_enabled       = true
parallel_execution_message_size  = 8192
parallel_max_servers             = 200
partition_view_enabled           = true
processes                        = 400
query_rewrite_enabled            = true
query_rewrite_integrity          = trusted
sessions                         = 400
shared_pool_size                 = 200000000
sort_area_size                   = 10000000
transactions                     = 512
transactions_per_rollback_segment= 20
user_dump_dest                   = "/private5/mbender/udump8i"

*********************************************************************************************
```

init9iopt.ora

```
*********************************************************************************************
audit_trail                     = FALSE
compatible                      = 8.2.0
control_files = ("/private5/mbender/dbs/control9iopt1.ora",
                 "/private5/mbender/dbs/control9iopt2.ora")
db_block_buffers                = 20000
db_block_size                   = 8192
db_files                        = 1023
_disable_multiple_block_sizes=true
db_file_multiblock_read_count   = 64
db_name                         = 9iopt
distributed_transactions        = 20
dml_locks                       = 100000
enqueue_resources               = 50000
hash_area_size                  = 10000000
  # hash_area_size : needed only for comparison to 8i
  # (Group By tests), with Oracle9i Dynamic Memory
  # Management, the setting is not necessary
large_pool_size                 = 200000000
log_buffer                      = 8388608
max_dump_file_size              = 500000
max_rollback_segments           = 650
nls_date_format                 = YYYY-MM-DD
open_cursors                    = 1024
optimizer_index_cost_adj        = 20
parallel_broadcast_enabled      = true
parallel_execution_message_size = 8192
parallel_max_servers            = 200
partition_view_enabled          =true
pga_aggregate_target            = 4g
processes                       = 400
query_rewrite_enabled           = true
query_rewrite_integrity         = trusted
sessions                        = 400
shared_pool_size                = 200000000
sort_area_size                  = 10000000
  # sort_area_size : needed only for comparison to 8i
  # (Group By tests), with Oracle9i Dynamic Memory
  # Management, the setting is not necessary
transactions                    = 512
transactions_per_rollback_segment= 20
user_dump_dest                  = "/private5/mbender/udump9i"

*********************************************************************************************
```

Creation Statements

merge / upsert

```
**************************************************************************************************
Rem upsert9i.sql
Rem
Rem    NAME
Rem    upsert9i.sql
Rem
Rem    DESCRIPTION
Rem      Modifies the table customer using the new 9i MERGE-statement
Rem      Destination table: CUSTOMER                    : 4.5 M. Rows
Rem      Source table: CUSTOMER_MERGE                   : 3    M. Rows
Rem      1.5  M. New Entries (Insert) 1.5 / 1.5 M. existing (Update)
Rem      Parallel Degree: 16
Rem
Rem    MODIFIED   (MM/DD/YY)
Rem    batzenbe     03/24/01 - Created
Rem

alter session enable parallel dml;

MERGE /*+ PARALLEL(O,16) */ INTO CUSTOMER O
USING CUSTOMER_MERGE N
ON (O.C_CUSTKEY=N.C_CUSTKEY)
WHEN MATCHED THEN
UPDATE SET
 O.C_COMMENT = N.C_COMMENT
,O.C_NAME    = N.C_NAME
,O.C_ADDRESS = N.C_ADDRESS
,O.C_PHONE   = N.C_PHONE
WHEN NOT MATCHED THEN
INSERT
( C_CUSTKEY
,C_MKTSEGMENT
,C_NATIONKEY
,C_NAME
,C_ADDRESS
,C_PHONE
,C_ACCTBAL
,C_COMMENT )
VALUES
( N.C_CUSTKEY
,N.C_MKTSEGMENT
,N.C_NATIONKEY
,N.C_NAME
,N.C_ADDRESS
,N.C_PHONE
,N.C_ACCTBAL
,N.C_COMMENT );

commit;

**************************************************************************************************
```

```
**********************************************************************************************
Rem upsert8i.sql
Rem
Rem    NAME
Rem    upsert8i.sql
Rem
Rem    DESCRIPTION
Rem     Modifies the table customer using 8i UPDATE/INSERT-statements
Rem     Destination table: CUSTOMER              : 4.5 M. Rows
Rem     Source table: CUSTOMER_MERGE       : 3    M. Rows
Rem     1.5  M. New Entries (Insert) 1.5 / 1.5 M. existing (Update)
Rem     Parallel Degree: 16
Rem
Rem    MODIFIED   (MM/DD/YY)
Rem    batzenbe     03/24/01 - Created
Rem

alter session enable parallel dml;

UPDATE /*+ PARALLEL(N,16) */
(
SELECT /*+ PARALLEL(N,16) PARALLEL(O,16)  */
   O.C_COMMENT AS OLD_C_COMMENT
  ,N.C_COMMENT AS NEW_C_COMMENT
  ,O.C_NAME    AS OLD_C_NAME
  ,N.C_NAME    AS NEW_C_NAME
  ,O.C_ADDRESS AS OLD_C_ADDRESS
  ,N.C_ADDRESS AS NEW_C_ADDRESS
  ,O.C_PHONE   AS OLD_C_PHONE
  ,N.C_PHONE   AS NEW_C_PHONE
FROM CUSTOMER_MERGE N
    ,CUSTOMER O
WHERE O.C_CUSTKEY=N.C_CUSTKEY
)
SET
  OLD_C_COMMENT = NEW_C_COMMENT
 ,OLD_C_NAME    = NEW_C_NAME
 ,OLD_C_ADDRESS = NEW_C_ADDRESS
 ,OLD_C_PHONE   = NEW_C_PHONE;

INSERT /*+ PARALLEL(CUSTOMER,16) */  INTO CUSTOMER
SELECT /*+ PARALLEL(N,16) */ * FROM CUSTOMER_MERGE N
WHERE  NOT EXISTS
(SELECT '' FROM CUSTOMER O
WHERE N.C_CUSTKEY=O.C_CUSTKEY);

commit;

**********************************************************************************************
```

external tables

```
**********************************************************************************************
Rem upsert_external.sql
Rem
Rem    NAME
Rem    upsert_external.sql
Rem
Rem    DESCRIPTION
Rem      Modifies the table customer using the new 9i MERGE- and External Table feature
Rem      Destination table: CUSTOMER                    : 4.5 M. Rows
Rem      Source table: CUSTOMER_EXTERNAL: 3    M. Rows (External Table)
Rem      1.5  M. New Entries (Insert) 1.5 / 1.5 M. existing (Update)
Rem      Parallel Degree: 16
Rem
Rem    Consists of two steps:
Rem    1.) Creating External Table
Rem    2.) Merge using External Table directly
Rem    MODIFIED   (MM/DD/YY)
Rem    batzenbe     03/27/01 - Created
Rem

DROP DIRECTORY CUSTOMER_DIR ;
CREATE DIRECTORY CUSTOMER_DIR AS '/private/test/ext_load_dat' ;

DROP TABLE CUSTOMER_EXTERNAL;

CREATE TABLE CUSTOMER_EXTERNAL
(
  C_CUSTKEY NUMBER,
  C_MKTSEGMENT CHAR(10),
  C_NATIONKEY NUMBER,
  C_NAME VARCHAR2(25),
  C_ADDRESS VARCHAR2(40),
  C_PHONE CHAR(15),
  C_ACCTBAL NUMBER,
  C_COMMENT VARCHAR2(117)
)
ORGANIZATION external
(
  TYPE oracle_loader
  DEFAULT DIRECTORY CUSTOMER_DIR
  ACCESS PARAMETERS
  (
    RECORDS DELIMITED BY NEWLINE CHARACTERSET US7ASCII
    BADFILE 'CUSTOMER_DIR':'cust_external.bad'
    LOGFILE 'CUSTOMER_DIR':'cust_external.log'
    FIELDS LDRTRIM
            (C_CUSTKEY      POSITION (1:10)                CHAR (10)
            ,C_MKTSEGMENT   POSITION (11:20)       CHAR (10)
            ,C_NATIONKEY    POSITION (21:23)       CHAR (3)
            ,C_NAME         POSITION (24:41)       CHAR (18)
            ,C_ADDRESS      POSITION (42:81)       CHAR (40)
            ,C_PHONE        POSITION (82:96)       CHAR (15)
            ,C_ACCTBAL      POSITION (97:104)      CHAR (8)
            ,C_COMMENT      POSITION (105:134)     CHAR(30))
  )
  location
```

```
  (
'TPCD.CUST_MERGE_VAR_PAR_SYS_P8653.DAT'
,'TPCD.CUST_MERGE_VAR_PAR_SYS_P8654.DAT'
,'TPCD.CUST_MERGE_VAR_PAR_SYS_P8655.DAT'
,'TPCD.CUST_MERGE_VAR_PAR_SYS_P8656.DAT'
,'TPCD.CUST_MERGE_VAR_PAR_SYS_P8657.DAT'
,'TPCD.CUST_MERGE_VAR_PAR_SYS_P8658.DAT'
,'TPCD.CUST_MERGE_VAR_PAR_SYS_P8659.DAT'
,'TPCD.CUST_MERGE_VAR_PAR_SYS_P8660.DAT'
,'TPCD.CUST_MERGE_VAR_PAR_SYS_P8661.DAT'
,'TPCD.CUST_MERGE_VAR_PAR_SYS_P8662.DAT'
,'TPCD.CUST_MERGE_VAR_PAR_SYS_P8663.DAT'
,'TPCD.CUST_MERGE_VAR_PAR_SYS_P8664.DAT'
,'TPCD.CUST_MERGE_VAR_PAR_SYS_P8665.DAT'
,'TPCD.CUST_MERGE_VAR_PAR_SYS_P8666.DAT'
,'TPCD.CUST_MERGE_VAR_PAR_SYS_P8667.DAT'
,'TPCD.CUST_MERGE_VAR_PAR_SYS_P8668.DAT'
  )
)REJECT LIMIT UNLIMITED;

alter session enable parallel dml;

MERGE /*+ PARALLEL(O,16) */ INTO CUSTOMER O
USING CUSTOMER_EXTERNAL N
ON (O.C_CUSTKEY=N.C_CUSTKEY)
WHEN MATCHED THEN
UPDATE SET
 O.C_COMMENT = N.C_COMMENT
,O.C_NAME    = N.C_NAME
,O.C_ADDRESS = N.C_ADDRESS
,O.C_PHONE   = N.C_PHONE
WHEN NOT MATCHED THEN
INSERT
( C_CUSTKEY
,C_MKTSEGMENT
,C_NATIONKEY
,C_NAME
,C_ADDRESS
,C_PHONE
,C_ACCTBAL
,C_COMMENT )
VALUES
( N.C_CUSTKEY
,N.C_MKTSEGMENT
,N.C_NATIONKEY
,N.C_NAME
,N.C_ADDRESS
,N.C_PHONE
,N.C_ACCTBAL
,N.C_COMMENT );

commit;

********************************************************************************************
```

group by

```
**************************************************************************************************
Rem GroupBy.sql
Rem
Rem    NAME
Rem      GroupBy.sql
Rem
Rem    DESCRIPTION
Rem      This query joins lineitem and parts with a large number of
Rem      GROUP BY records as output.
Rem
Rem    MODIFIED   (MM/DD/YY)
Rem    mbender     03/07/01 - Created
Rem

alter session set sort_area_size = 20000000;
alter table lineitem parallel 24;
alter table parts parallel 24;

select count(*) from
(
select
       p_brand
      ,p_container
      ,p_size
      ,l_shipmode
      ,l_shipinstruct
      ,p_name
      ,sum(l_quantity) as sum_qty
      ,sum(l_extendedprice) as sum_base_price
      ,variance(l_extendedprice) as var_base_price
      ,sum(l_extendedprice * (1 - l_discount)) as sum_disc_price
      ,variance(l_extendedprice * (1 - l_discount)) as var_disc_price
      ,sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as sum_charge
      ,variance(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as var_charge
      ,avg(l_quantity) as avg_qty
      ,avg(l_extendedprice) as avg_price
      ,avg(l_discount) as avg_disc
      ,avg(l_extendedprice * (1 - l_discount)) as avg_disc_price
      ,avg(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as avg_charge
      ,count(*) as count#
from
       lineitem,
       parts
where
       p_partkey = l_partkey
       and p_container in ('SM CASE', 'SM BOX', 'SM PACK', 'SM PKG')
       and l_shipdate between to_date( '1995-01-01', 'YYYY-MM-DD')
       and to_date( '1997-06-28', 'YYYY-MM-DD')
group by p_name
       ,p_brand
       ,p_container
       ,p_size
       ,l_shipmode
       ,l_shipinstruct);
```

****

materialized views

****

```
Rem mav10.sql
Rem
Rem    NAME
Rem      mav10.sql
Rem
Rem    DESCRIPTION
Rem      This create statement creates an aggregated Materialized View
Rem       an index on the GROUP BY keys is not created automatically
Rem
Rem    MODIFIED   (MM/DD/YY)
Rem    mbender     03/07/01 - Created
Rem

create materialized view mav10
pctfree 1
pctused 99
initrans 10
tablespace ts_ind
storage (initial 1m next 1m freelists 12 freelist groups 2 maxextents unlimited pctincrease 0)
parallel (degree 12 )
nologging
using no index
enable query rewrite
as

-- statement continues
```

```
select
       p_brand
      ,p_size
      ,p_container
      ,l_shipmode
      ,l_shipinstruct
      ,l_quantity
      ,l_shipdate
      ,count(l_quantity) as countqty
      ,count(l_discount) as countdisc
      ,count(l_extendedprice) as count_price
      ,count(l_extendedprice * (1 - l_discount)) as cou_disc_price
      ,count(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as cou_charge
      ,sum(l_quantity) as sumqty
      ,sum(l_discount) as sumdisc
      ,sum(l_extendedprice) as sum_base_price
      ,sum(l_extendedprice * (1 - l_discount)) as revenue
      ,sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as sum_charge
      ,variance(l_extendedprice) as var_base_price
      ,variance(l_extendedprice * (1 - l_discount)) as var_disc_price
      ,variance(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as var_charge
      ,avg(l_quantity) as avg_qty
      ,avg(l_discount) as avg_disc
      ,avg(l_extendedprice) as avg_price
      ,avg(l_extendedprice * (1 - l_discount)) as avg_disc_price
      ,avg(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as avg_charge
from
       lineitem,
       parts
where
       p_partkey = l_partkey
       and p_container in ('SM CASE', 'SM BOX', 'SM PACK', 'SM PKG')
       and l_shipdate between to_date( '1995-01-01', 'YYYY-MM-DD')
       and to_date( '1995-12-26', 'YYYY-MM-DD')
group by
       p_brand
      ,p_size
      ,p_container
      ,l_shipmode
      ,l_shipinstruct
      ,l_quantity
      ,l_shipdate;

*********************************************************************************************
```

```
**************************************************************************************************
Rem mav12.sql
Rem
Rem    NAME
Rem      mav12.sql
Rem
Rem    DESCRIPTION
Rem      This statement creates an Aggregated Materialized View,
Rem       an index on the GROUP BY keys is not created automatically
Rem       since no predicates on lineitem or parts are provided. A lot of
Rem       hash and sort information has to be written down to temp
Rem
Rem    MODIFIED   (MM/DD/YY)
Rem    mbender     03/07/01 - Created
Rem

create materialized view mav12
pctfree 1
pctused 99
initrans 10
tablespace ts_ind
storage (initial 1m next 1m freelists 12 freelist groups 2 maxextents unlimited pctincrease 0)
parallel (degree 12 )
nologging
using no index
enable query rewrite
as
select
       p_brand
      ,p_container
      ,l_shipdate
      ,count(*) cnt
      ,count(l_quantity) as countqty
      ,count(l_discount) as countdisc
      ,count(l_extendedprice) as count_price
      ,count(l_extendedprice * (1 - l_discount)) as cou_disc_price
      ,count(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as cou_charge
      ,sum(l_quantity) as sumqty
      ,sum(l_discount) as sumdisc
      ,sum(l_extendedprice) as sum_base_price
      ,sum(l_extendedprice * (1 - l_discount)) as revenue
      ,sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as sum_charge
      ,avg(l_extendedprice * (1 - l_discount)) as avg_disc_price
from
       lineitem,
       parts
where
       p_partkey = l_partkey
group by
       p_brand
      ,p_container
      ,l_shipdate;

**************************************************************************************************
```

```
**************************************************************************************************
Rem mav30.sql
Rem
Rem    NAME
Rem      mav30.sql
Rem
Rem    DESCRIPTION
Rem      This is an aggregated materialized view on the table orders
Rem
Rem    MODIFIED   (MM/DD/YY)
Rem    batzenbe     03/22/01 - Created
Rem

create materialized view mav30
pctfree 1
initrans 10
tablespace ts_ind
storage (initial 1m next 1m freelists 12 freelist groups 2 maxextents unlimited pctincrease 0)
parallel (degree 12 )
nologging
enable query rewrite
as
select
  o_custkey
  ,count(*)         as count_ord_rev_cust
  ,sum(o_totalprice) as sum_ord_rev_cust
  ,max(o_totalprice) as max_ord_rev_cust
  ,min(o_totalprice) as min_ord_rev_cust
  ,avg(o_totalprice) as avg_ord_rev_cust
  ,variance(o_totalprice) as var_ord_rev_cust
  ,max(o_orderdate)  as max_order_date
  ,min(o_orderdate)  as min_order_date
from
       orders
group by
       o_custkey;

**************************************************************************************************
```

```
**************************************************************************************
Rem mav40.sql
Rem
Rem    NAME
Rem      mav30.sql
Rem
Rem    DESCRIPTION
Rem      This is an aggregated materialized view on the table lineitem
Rem
Rem    MODIFIED   (MM/DD/YY)
Rem    mbender     03/22/01 - Created
Rem


create materialized view ma_lineitem
pctfree 2
tablespace ts_default
parallel
nologging
REFRESH fast
ON demand
enable query rewrite
as select
count(*) as count_p_group,
              l_shipdate,
              l_returnflag,
              l_linestatus,
              sum(l_quantity) as sum_qty,
              sum(l_extendedprice) as sum_base_price,
              sum(l_extendedprice * (1 - l_discount)) as sum_disc_price,
              count(l_extendedprice * (1 - l_discount)) as count_disc_price,
              sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as sum_charge,
              count(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as count_charge,
              count(l_quantity) as count_qty,
              count(l_extendedprice) as count_price,
              sum(l_discount) as sum_disc,
              count(l_discount) as count_disc,
              count(*) as count_order
FROM
   lineitem
GROUP BY
  l_returnflag ,
  l_linestatus ,
  l_shipdate ;

**************************************************************************************
```

fast refresh

```
*********************************************************************************************
Rem  Fast Refresh (mav12.sql)
Rem
Rem    DESCRIPTION
Rem      Fast Refresh test
Rem      1. create the snapshot logs
Rem      2. create the materialized view (mav12)
Rem
Rem    MODIFIED   (MM/DD/YY)
Rem    mbender     03/22/01 - Created
Rem

create materialized view log on lineitem
tablespace ts_ind
parallel
with rowid
(
    l_shipdate
   ,l_quantity
   ,l_discount
   ,l_extendedprice
   ,l_tax
   ,l_partkey
)
including new values;

create materialized view log on parts
tablespace ts_ind
parallel
with rowid
(   p_partkey
   ,p_brand
   ,p_container
)
including new values;

Rem
Rem create the materialized view mav12 (see mav12.sql)
Rem snapshot logs must be there before creating the mav
Rem

Rem
Rem in Oracle8i only insert append is supported
Rem
alter session enable parallel dml;
insert into orders   select * from temp_o /* 1 Row */;
insert into lineitem select * from temp_l /* 2 Rows */;
commit;

Rem for Oracle8i
execute dbms_snapshot.refresh('MAV12','F',atomic_refresh=>false);

Rem for Oracle9i
execute dbms_snapshot.refresh('MAV12','F');
```

****

```
**************************************************************************************
Rem  Fast Refresh (mav40.sql)
Rem
Rem    DESCRIPTION
Rem      Fast Refresh test
Rem      1. create the snapshot logs
Rem      2. create the materialized view (mav40)
Rem
Rem    MODIFIED   (MM/DD/YY)
Rem    mbender     03/22/01 - Created
Rem

create materialized view log on lineitem
tablespace ts_ind
parallel
partition by range (l_shipdate)
(
,partition item1 values less than (to_date('1992-01-01','YYYY-MM-DD'))
,partition item2 values less than (to_date('1992-02-01','YYYY-MM-DD'))
,partition item3 values less than (to_date('1992-03-01','YYYY-MM-DD'))
…
,partition item82 values less than (to_date('1998-10-01','YYYY-MM-DD'))
,partition item83 values less than (to_date('1998-11-01','YYYY-MM-DD'))
,partition item84 values less than (MAXVALUE)
)
with rowid
(
   l_shipdate          ,
   l_returnflag        ,
   l_linestatus        ,
   l_quantity          ,
   l_extendedprice     ,
   l_discount          ,
   l_tax               ,
   l_suppkey           ,
   l_partkey           ,
   l_orderkey
)
including new values
;

Rem
Rem create the materialized view mav40 (see mav40.sql)
Rem snapshot logs must be there before creating the mav
Rem

alter session enable parallel dml;
insert into lineitem select * from temp_item_3 /* 179,000 Rows */;
commit;

execute dbms_snapshot.refresh('MA40');

**************************************************************************************
```

pl/sql

```
*********************************************************************************************
Rem function.sql
Rem
Rem    NAME
Rem    function.sql
Rem
Rem    DESCRIPTION
Rem     This script consists of two steps:
Rem     1.) Creating the function total_price
Rem     2.) Test SQL applying total_price to table orders
Rem
Rem
Rem    Prerequisite: Index on column l_orderkey
Rem    Subselect determines how many rows are retrieved
Rem    In this version: 400,000
Rem
Rem    MODIFIED   (MM/DD/YY)
Rem    batzenbe     03/22/01 - Created
Rem

CREATE OR REPLACE  FUNCTION
            total_price
           (
             ex_price in number
            ,discount in number
           )
            RETURN NUMBER DETERMINISTIC PARALLEL_ENABLE
           IS discount_price NUMBER(30,2);
            BEGIN
            discount_price:=ex_price*(1-discount);
            RETURN (discount_price);
            END;


select
        round(max(total_price(l_extendedprice,l_discount)),2) as max_total_price
       ,round(min(total_price(l_extendedprice,l_discount)),2) as min_total_price
       ,round(avg(total_price(l_extendedprice,l_discount)),2) as avg_total_price
from  lineitem l
where l_orderkey  > (
                              select max(l_orderkey)-400000
                              from lineitem
                            );

*********************************************************************************************
```

index skip scan

```
********************************************************************************************
Rem index_skip.sql
Rem
Rem    NAME
Rem    index_skip.sql
Rem
Rem    DESCRIPTION
Rem     This script consists of two steps:
Rem       1.) Create index I_ORDERS_SKIP
Rem       2.) Querying the table
Rem
Rem    Query filters 0,.14% of rows
Rem    By default the optimizer chooses Bitmap Join Index in 9i and index fast full scan in 8i
Rem    MODIFIED   (MM/DD/YY)
Rem    batzenbe     03/23/01 - Created
Rem

create index I_orders_skip
on orders
(
      o_orderpriority
    , o_orderstatus
    , o_totalprice
  )
nologging
parallel( degree 24)
storage (initial 10M next 10M pctincrease 0)
compute statistics
tablespace ts_ps;

select
       o_orderpriority
      ,o_orderstatus
      ,sum(o_totalprice)
from
       orders
where
       o_orderstatus='F'
and
       o_totalprice between 0 and 2000
group by
       o_orderpriority
      ,o_orderstatus;

********************************************************************************************
```

bitmap join index

```
********************************************************************************************
Rem bitmap_join.sql
Rem
Rem    NAME
Rem    bitmap_join.sql
Rem
Rem    DESCRIPTION
Rem     This script consists of two steps:
Rem       1.) Create index I_PARTSUP_BMJ (only Oracle9i )
Rem       2.) Querying the table
Rem
Rem
Rem    By default optimizer chooses Bitmap Join Index access in 9i and hash join in 8i
Rem    MODIFIED   (MM/DD/YY)
Rem    batzenbe     03/29/01 - Created
Rem


create bitmap index i_partsup_bmj
on partsupp(p.p_type,s.s_nationkey)
from partsupp ps,parts p,supplier s
where ps.ps_partkey=p.p_partkey
and ps.ps_suppkey=s.s_suppkey
local
nologging
parallel (degree 16)
storage (initial 1m next 1m pctincrease 0)
compute statistics;

select
count(distinct ps_suppkey)
,avg(ps_supplycost)
,max(ps_supplycost)
,min(ps_supplycost)
from partsupp ps,supplier s,parts p
where ps.ps_suppkey=s.s_suppkey
and   ps.ps_partkey=p.p_partkey
and s.s_nationkey=24
and p.p_type='medium burnished steel';

********************************************************************************************
```

DWH Performance Enhancements with Oracle9*i*
April 2001
Author: Bernhard Atzenberger, Marcus Bender
Contributing Authors: Cetin Ozbutun, Hakan Jakobsson, Benoit Dageville, Alexandra Karasik, Angela Amor, Herrmann Baer, Neil Thombre

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
www.oracle.com

Oracle Corporation provides the software
that powers the internet.