DSS Performance in Oracle Database 10*g* Release 2

An Oracle White Paper
July 2005

ORACLE

Executive Overview 3
Introduction 3
Test Environment 4
In Memory Sort 6
Hash-Based Aggregation 9
Partitioned Tables 10
Object Checkpointing 14
SQL Built-in Apply Functions for Data Mining 16
Enhancements for Support Vector Machines (SVM) Algorithm 20
Conclusion 23

EXECUTIVE OVERVIEW

Data warehouses today require high performance database management systems as backbones. Oracle Database 10*g* Release 2 delivers new features and enhancements that provide significant performance improvements to satisfy the growing data warehousing requirements. The performance gains take effiect automatically after upgrading to Oracle Database 10*g* Release 2 without any application changes or additional deployment costs. This technical white paper describes these new features and enhancements, and demonstrates the performance gains they deliver.

INTRODUCTION

Every day, organizations amass billions of bytes of data about their business, millions of individual facts about their customers, operations, products, and employees. Organizations poised to experience the greatest success will be those that can effectively leverage their data to improve day-to-day operational process, build strong relationships with stake holders, and most importantly, satisfy the demands of their customers. Data warehouses and business intelligence have become cornerstones for many successful enterprises.

As companies seek competitive advantage by striving to obtain more business intelligence, their data warehouses increase in size. The exploding data warehouse volumes and increased complexity is stretching the capabilities of database management systems. The pressure is on to provide faster, better information and to react quickly to changing business needs.

Oracle Database 10*g* Release 2 rises to the challenge delivering high performance database product to meet today's fast growing data warehousing requirement. New features and enhancements are designed to achieve transparent performance gains in the area of data warehouse and business intelligence. After upgrading to Oracle Database 10*g* Release 2, these new features and enhancements will integrate seamlessly with existing data warehousing applications without any new implementation or intervention from the data warehousing team.

This technical white paper describes these new features and enhancements. Performance gains are demonstrated via concrete comparisons of query performances in Oracle Database 10*g* Release 1 and Oracle Database 10*g* Release 2. In the subsequent sections of the paper, the test environment is described first, followed by descriptions of the tests characterizing the performance gains delivered

by the new features including the new in-memory sort algorithm, the hash aggregation feature, the object check-pointing feature, and enhancements for partitioned tables. Two data mining features will then be explained and the performance improvements will be demonstrated. Finally, we conclude the paper.

TEST ENVIRONMENT

Tests in this white paper were conducted on a system with 2x2.8 GHz Intel Xeon processors with Hyper-Threading technology. The databases were on an OCFS file system stripped over 10 disks in a single disk array. The database schema, portrayed by Figure 1, consists of the Sales History schema from Oracle Sample Schema and a few other independent tables that were added for data mining purposes.

The SALES table is populated with 7 years worth of sales information, from Jan 1995 to Dec 2001, totaling about 10GB of data. Both the SALES table and COSTS table are compressed and are range-partitioned on time_id, and the total number of partitions is 20. The table below lists the number-of-rows and average-row-length statistics for each table.

Table	Number of Rows	Average Row Length
SALES	300,000,000	31
COSTS	14,600,000	20
CUSTOMERS	50,000	138
PRODUCTS	10,000	241
TIMES	1461	177
PROMOTIONS	501	67
COUNTRIES	19	33
CHANNELS	5	19
PRODUCTS_AFFINITY_CARD_B	50,000	1,151
PRODUCTS_AFFINITY_CARD_A	2,000,000	1,151
SUPPLEMENTARY_DEMOGRAPHICS_B	32,561	107
SUPPLEMENTARY_DEMOGRAPHICS_T	16,281	107
SUPPLEMENTARY_DEMOGRAPHICS_A	97,684	107

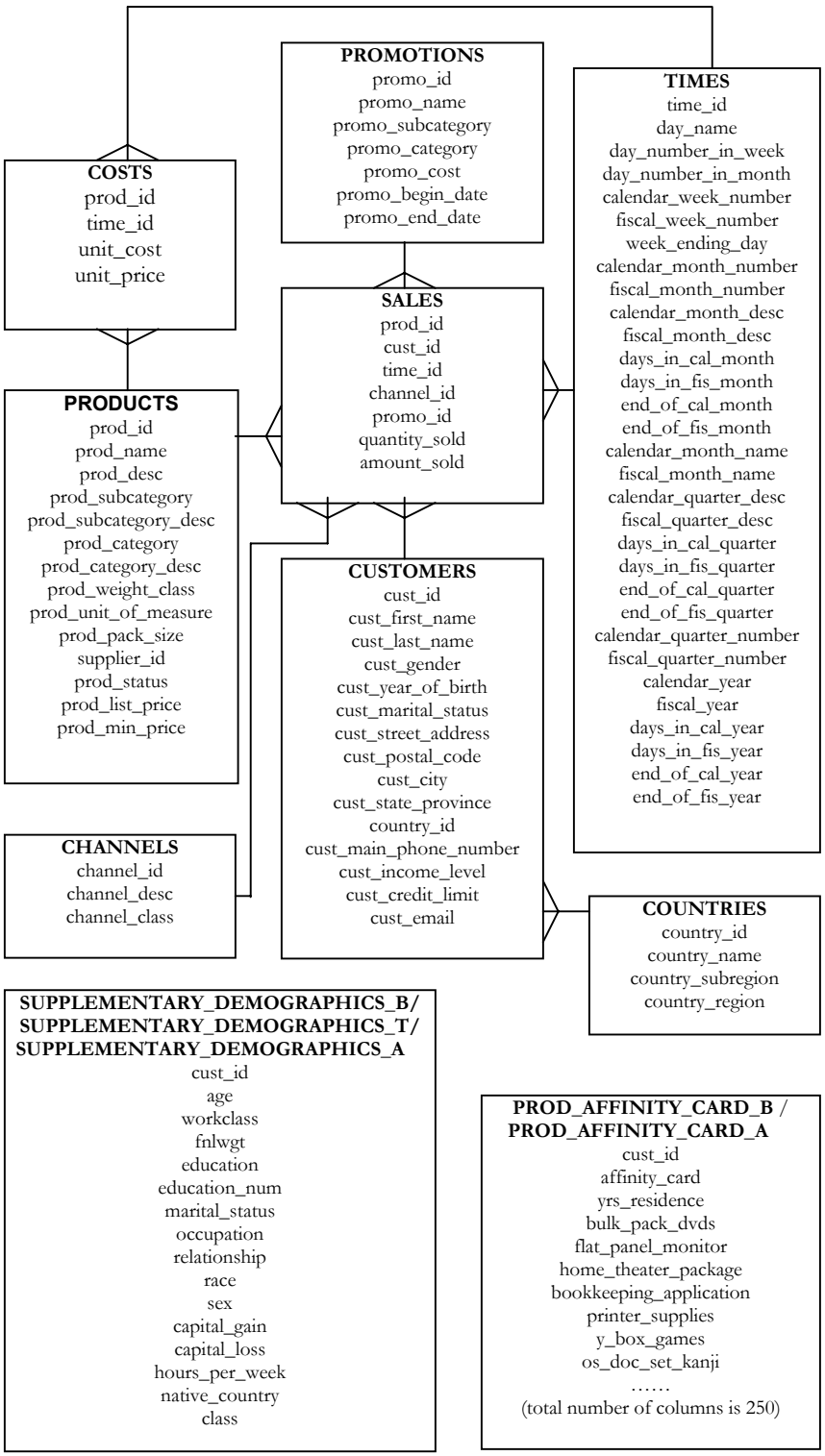

Figure 1 Database Schema

Each of the following sections introduces a new feature in Oracle Database 10*g* Release 2 with test cases built upon the above schema.

IN MEMORY SORT

Oracle Database 10*g* Release 2 introduces a new in-memory sort algorithm which sorts faster and has better memory system locality than the existing sort algorithm in Oracle Database 10*g* Release 1.

The new in-memory sort algorithm improves performances for a variety of queries. It is used for in the following cases:

- queries with an "order by" clause
- queries using "sort merge join"
- queries using "partition outer join"
- create index

The new in-memory sort algorithm does not support columns that have garbage collection, segmented streams, or larger than 30,000 bytes, even if these are the non-key columns. For key columns, the new in-memory sort does not support columns that require semantic comparison, such as logical rowid and time with time zone.

Under some circumstances, the performance of the current sort algorithm degrades as the available memory increases and reaches a certain threshold. The new in-memory sort algorithm addresses this problem because of its good memory system locality.

The first group of tests shows performance improvement of a couple of typical queries that benefit from the new in-memory sort. These tests were run with workarea_size_policy set to AUTO. The second group of tests demonstrates the memory system locality of the new in-memory sort algorithm, by setting workarea_size_policy to MANUAL running with varying sort_area_size's and measuring the elapsed time for one query.

The following queries were used for the performance evaluation.

```
create index sales_cust_id on sales (cust_id) parallel
compute statistics nologging;
```

We denote this query as "**INDEX**" query.

For select statement, the following query was considered:

```
select prod_id, time_id, unit_cost from costs order by
unit_cost;
```

To avoid outputting 14,600,000 rows on screen, the following query was actually ran:

```
select count(*) from (select /*+ no_merge */ prod_id,
time_id, unit_cost from costs order by unit_cost);
```

The NO_MERGE hint prevents Oracle from merging an inline view into a potentially non-collocated SQL statement so that "order by unit_cost" is actually executed.

We denote this query as **"SELECT"** query.

Each of the above queries was run in both Oracle Database 10*g* Release 1 and Oracle Database 10*g* Release 2 and their execution statistics are compared. The execution plan in Oracle Database 10*g* Release 2 is the same as the execution plan in Oracle Database 10*g* Release 1, although the underlying implementation for in-memory sort is different.

For the INDEX query, the execution plan is:

```
 Id   Operation                           Name    Pstart   Pstop
-------------------------------------------------------------------
   0  CREATE INDEX STATEMENT
   1   PX COORDINATOR
   2    PX SEND QC (ORDER)                :Q1001
   3     INDEX BUILD NON UNIQUE
   4      SORT CREATE INDEX
   5       PX RECEIVE
   6        PX SEND RANGE                 :Q1000
   7         PX BLOCK ITERATOR                      1       20
   8          TABLE ACCESS FULL           SALES      1       20
```

The following table compares the performance of the INDEX query:

	Elapsed Time (s)	User CPU(%)
Oracle Database 10*g* Release 1	1432	88.54
Oracle Database 10*g* Release 2	706	55.98

Because of the new sort algorithm, the INDEX query in Oracle Database 10*g* Release 2 is 50% faster than Oracle Database 10*g* Release 1. Furthermore, the percentage of CPU that is busy running the INDEX query in Oracle Database 10*g* Release 2 (55.98%) is much less than the percentage of CPU that is busy running the INDEX query in Oracle Database 10*g* Release 1 (88.54%), allowing more concurrent queries.

For the SELECT query, the execution plan is:

```
 Id   Operation                  Name            Pstart   Pstop
-------------------------------------------------------------------
   0  SELECT STATEMENT
   1   SORT AGGREGATE
   2    PX COORDINATOR
   3     PX SEND QC (ORDER)      :Q1001
   4      SORT AGGREGATE
   5       VIEW
   6        SORT ORDER BY
   7         PX RECEIVE
   8          PX SEND RANGE     :Q1000
   9           PX BLOCK ITERATOR                1        20
  10            TABLE ACCESS FULL   COSTS       1        20
```

The following table compares the performance of the SELECT query.

	Elapsed Time (s)	User CPU(%)
Oracle Database 10*g* Release 1	19	83.21
Oracle Database 10*g* Release 2	10	73.33

The SELECT with order-by query is 47% faster and consumes less CPU time than the equivalent query in Oracle Database 10*g* Release 1.

Unlike the existing sort algorithm in Oracle Database 10*g* Release 1, the new sort algorithm in Oracle Database 10*g* Release 2 has good memory system locality. It generates less cache and TLB misses than the existing sort algorithm in Oracle Database 10*g* Release 1. As the memory available to the sort is increased, the performance of queries using the new sort algorithm improves.

Using the INDEX query as an example, we vary the sort_area_size and compare the query performance for Oracle Database 10*g* Release 1 and Oracle Database 10*g* Release 2. The following table captures the elapsed time in seconds for the INDEX query using different sort_area_size.

SORT_AREA_SIZE	1 MB	10 MB	100 MB
Oracle Database 10*g* Release 1	1318	1389	1653
Oracle Database 10*g* Release 2	911	804	715

In none of the above cases, the index creation is completely in memory. However, because of the good memory system locality of the new in-memory sort algorithm, the improvement of the INDEX query in Oracle Database 10*g* Release 2 over Oracle Database 10*g* Release 1 increases from 31% to 57% as we increase the sort_area_size from 1MB to 100MB.

In conclusion, the new in-memory sort algorithm introduced in Oracle Database 10*g* Release 2 significantly improves query performance in terms of elapsed time while consuming less CPU and allowing more concurrent queries. Because of the better memory system locality afforded by the new in-memory sort algorithm, the performances of queries using the new in-memory sort algorithm improve as the available memory increases.

HASH-BASED AGGREGATION

Data aggregation is a frequent operation in data warehousing and OLAP environments. Data is either aggregated on the fly or pre-computed and stored as materialized views. In either case, data aggregation brings together records with the same values of the group-by columns and aggregates them.

Data aggregation in Oracle Database 10*g* Release 1 uses a sort-based approach. In Oracle Database 10*g* Release 2, a hash-based scheme has been introduced. It works by building a hash table on the data based on values of the group-by columns and finding the records with same values of the group-by columns by probing the hash table. The hash-based scheme significantly improves the performance of data aggregation.

The performance improvement of the hash-based approach is illustrated by the following query that computes the revenue contribution by individual customers and lists the most significant revenue contributor first.

```
select c.cust_last_name last_name,
       s.revenue        revenue
from customers c,
        (select cust_id,
                sum(amount_sold)     revenue
        from sales
        group by cust_id ) s
where s.cust_id = c.cust_id
order by  s.revenue desc;
```

The query is CPU intensive. To ensure fair comparison, the new in-memory sort feature in Oracle Database 10*g* Release 2 was turned off. The following table compares query performance in Oracle Database 10*g* Release 1 and Oracle Database 10*g* Release 2.

	Elapsed Time (s)
Oracle Database 10*g* Release 1	395
Oracle Database 10*g* Release 2	171

With the hash-based scheme for data aggregation, running the query in Oracle Database 10*g* Release 2 is 57% faster than running the query in Oracle Database 10*g* Release 1.

The significant performance improvement brought by the hash-based aggregation feature in Oracle Database 10*g* Release 2 has large impact in the DSS area as a result of the frequent usage of the data-aggregation operation.

PARTITIONED TABLES

There are two major changes to partitioning in Oracle Database 10*g* Release 2. To increase scalability and flexibility, the number of partitions per table has been increased from 64K to 1 million. The second change is the implementation of more effective pruning techniques to ensure that we access only the partitions that satisfy any query predicates. We illustrate the pruning techniques in Oracle Database 10*g* Release 2 from two different aspects -- partition pruning for disjunct OR predicates and partition pruning using index access path.

To demonstrate the partition pruning improvement for disjunct OR predicates, the following query is used.

```
select  p.promo_name                  promo_name,
        (s.profit - p.promo_cost)     profit
from  promotions p,
        ( select promo_id,
sum(sales.QUANTITY_SOLD * (costs.UNIT_PRICE -
costs.UNIT_COST)) profit
        from sales, costs
          where
            (
(sales.time_id BETWEEN TO_DATE('01-JAN-1998','DD-MON-YYYY',
'NLS_DATE_LANGUAGE = American') and TO_DATE('01-JAN-
1999','DD-MON-YYYY', 'NLS_DATE_LANGUAGE = American'))
      OR
(sales.time_id BETWEEN TO_DATE('01-JAN-2001','DD-MON-YYYY',
'NLS_DATE_LANGUAGE = American') and TO_DATE('01-JAN-
2002','DD-MON-YYYY', 'NLS_DATE_LANGUAGE = American'))
)
      AND sales.time_id = costs.time_id
```

```
        AND sales.prod_id = costs.prod_id
    group by promo_id
)  s
where
    s.promo_id = p.promo_id
order by  profit desc;
```

The query joins the SALES and COSTS tables. The SALES table is partitioned by range on the column TIME_ID. One of the conditions in the query are two predicates on TIME_ID which are OR-ed together.

In Oracle Database 10*g* Release 1, the query is transformed using OR-expansion into two sub-queries followed by a concatenation. Each sub-query joins the SALES and COSTS tables using one branch of the OR predicate. As a result, the COSTS table is accessed twice, in addition to the concatenation operation. This is shown in the following query execution plan. The parts of the plan that deal with the disjunct OR predicates are highlighted.

```
Id Operation                                Name Pstart Pstop
-----------------------------------------------------------------
 0 SELECT STATEMENT
 1  PX COORDINATOR
 2   PX SEND QC (ORDER)                   :Q1006
 3    SORT ORDER BY
 4     PX RECEIVE
 5      PX SEND RANGE                     :Q1005
 6       HASH JOIN
 7        PX RECEIVE
 8         PX SEND HASH                   :Q1003
 9          VIEW
10           SORT GROUP BY
11            PX RECEIVE
12             PX SEND HASH               :Q1002
13              SORT GROUP BY
14               CONCATENATION
15                HASH JOIN
16                 PX BLOCK ITERATOR                1   20
17                  TABLE ACCESS FULL       COSTS     1   20
18                 PX RECEIVE
19                  PX SEND BROADCAST LOCAL :Q1000
20                   BUFFER SORT
21                    PX BLOCK ITERATOR              17  20
22                     TABLE ACCESS FULL    SALES     17  20
```

```
23                HASH JOIN
24                 PX BLOCK ITERATOR                 1   20
25                  TABLE ACCESS FULL       COSTS     1   20
26                 PX RECEIVE
27                  PX SEND BROADCAST LOCAL :Q1001
28                   BUFFER SORT
29                    PX BLOCK ITERATOR               5    9
30                     TABLE ACCESS FULL   SALES      5    9
31         PX RECEIVE
32          PX SEND HASH                    :Q1004
33           PX BLOCK ITERATOR
34            TABLE ACCESS FULL            PROMOTIONS
```

In Oracle Database 10*g* Release 2, the OR predicate is used directly to prune the partitions in SALES table and a single join between the SALES and COSTS table is performed, as shown in the following query execution plan. The hash aggregation feature in Oracle Database 10*g* Release 2 was turned off to ensure fair comparison. The parts of the plan that deal with the disjunct OR predicates are again highlighted.

```
 Id  Operation                             Name        Pstart Pstop
-------------------------------------------------------------------
  0 SELECT STATEMENT
  1  PX COORDINATOR
  2   PX SEND QC (ORDER)          :TQ10005
  3    SORT ORDER BY
  4     PX RECEIVE
  5      PX SEND RANGE                   :TQ10004
  6       HASH JOIN
  7        PX RECEIVE
  8           PX SEND HASH               :TQ10002
  9            VIEW
 10             SORT GROUP BY
 11              PX RECEIVE
 12               PX SEND HASH           :TQ10001
 13                SORT GROUP BY
 14                 HASH JOIN
 15                  PX BLOCK ITERATOR                 1   20
 16                     TABLE ACCESS FULL     COSTS     1   20
 17                  PX RECEIVE
 18                     PX SEND BROADCAST LOCAL :TQ10000
 19                      PX BLOCK ITERATOR              KEY  KEY
 20                       TABLE ACCESS FULL   SALES     KEY  KEY
 21         PX RECEIVE
```

```
22          PX SEND HASH               :TQ10003
23           PX BLOCK ITERATOR
24            TABLE ACCESS FULL        PROMOTIONS
```

In Oracle Database 10*g* Release 1, more data was accessed because of the less effective pruning. In addition, the hash joins spill to disk, further increasing the number of the physical read requests and adding large number of the physical write requests. In Oracle Database 10*g* Release 2, however, as a result of the better pruning and decreased number of physical IO requests, the performance of the query is improved significantly. Specifically, the elapsed time of the query is improved about 30%; the user CPU time of the query, the amount of time that CPUs are busy running the query, totaled over all processors, is improved 23.4%.

	Elapsed Time (s)	User CPU Time (s)	Physical Read IO Requests	Physical Write IO Requests
Oracle Database 10*g* Release 1	417	1334	87668	20772
Oracle Database 10*g* Release 2	294	1021	17402	72

Oracle Database 10*g* Release 2 makes more efficient use of query predicates that specify non-contiguous partition ranges to prune unneeded partitions. We illustrate this by considering the following query that computes the total AMOUNT_SOLD for a list of the customers during two non-contiguous time periods. For the purpose of illustration, we've created a global index SALES_GIDX on the column CUST_ID of the partitioned table SALES. The query predicate specifies two disjoint partition ranges using the OR predicate. In Oracle Database 10*g* Release 2, Oracle will access only the rows belonging to the partitions specified by the OR predicate. Oracle achieves this by maintaining a list of valid partitions specified by the predicates. In Oracle Database 10*g* Release 1, a single range is maintained for the potentially valid partitions instead, resulting in less effective pruning.

```
select cust_id, sum(amount_sold)
from sales
where
(
(time_id BETWEEN TO_DATE('01-JAN-1997','DD-MON-YYYY',
'NLS_DATE_LANGUAGE = American') and TO_DATE('01-JAN-
1998','DD-MON-YYYY', 'NLS_DATE_LANGUAGE = American'))
```

```
OR
(time_id BETWEEN TO_DATE('01-JAN-2001','DD-MON-YYYY',
'NLS_DATE_LANGUAGE = American') and TO_DATE('01-JAN-
2002','DD-MON-YYYY', 'NLS_DATE_LANGUAGE = American'))
)
AND
cust_id in (50, 60, 70, 80, 1000, 1100, 1590, 4500, 80000,
250000, 350000, 400100, 430000)
group by cust_id;
```

The following table summarizes the performance results for the query in both Oracle Database 10*g* Release 1 and Oracle Database 10*g* Release 2.

	Elapsed Time (s)	Physical Read IO Requests
Oracle Database 10*g* Release 1	31	75968
Oracle Database 10*g* Release 2	10	25436

Because of the more effective pruning, the number of physical read requests for the query significantly decreases in Oracle Database 10*g* Release 2, resulting in 67% elapsed time improvement.

The implementation of more effective pruning techniques in Oracle Database 10*g* Release 2, as illustrated by the two examples in this section, greatly improves performance of queries involving partitioned tables. These partition-pruning enhancements, along with the increase of the number-of-partitions-per-table limit, further improves scalability and flexibility, allowing large datasets with large number of partitions to be efficiently handled.

OBJECT CHECKPOINTING

To make optimal use of hardware resources Oracle Parallel Query can use direct path reads. Direct path reads allow the query slaves to bypass the data cache for read requests and to issue read requests up to the size allowed by the Operating System. However, before an object can be accessed through direct path reads, dirty buffers of the object must be written back to disk via an object-checkpoint request. Oracle Database 10*g* Release 1 handles the object-checkpoint request by issuing a tablespace checkpoint for the tablespace the object belongs to, writing out all dirty buffers of the entire tablespace. Since a large number of objects may reside in the same tablespace, this implementation may cause large number of unnecessary disk writes. A checkpoint request for a target object may cause the dirty buffers of other

objects residing in the same tablespace to be written back to disk, in addition to the dirty buffers of the target object.

To eliminate this inefficiency in Oracle Database 10*g* Release 2 a checkpoint request for a target object will only write out the dirty buffers of that object, without incurring any additional writes for other objects' dirty buffers.

To illustrate the performance benefit of the new Oracle Database 10*g* Release 2 implementation, we assume that the following update statement needs to be issued to for the PRODUCTS_AFFINITY_CARD_A table:

```
update products_affinity_card_a set home_theater_package=0
where yrs_residence <2;
```

The update statement modifies 808,760 rows. Immediately after executing the update statement, the buffer cache contains around 84,000 dirty pages from the table PRODUCTS_AFFINITY_CARD_A.

Right after the update occurs, the following parallel select statement is run to calculate the maximum average cost of products.

```
select max(avg_costs)
from (select avg(unit_cost) avg_costs
      from costs
      group by prod_id);
```

Oracle Database 10*g* Release 1 needs to checkpoint the dirty buffers of table PRODUCTS_AFFINITY_CARD_A before executing the select statement, while Oracle Database 10*g* Release 2 is able to execute the select statement directly without writing dirty pages to disk.

To ensure fair comparison, the in-memory sort and hash aggregation features in Oracle Database 10*g* Release 2 were disabled, causing both Oracle Database 10*g* Release 1 and Oracle Database 10*g* Release 2 to execute the select query with the following execution plan:

```
 Id   Operation                          Name          Pstart     Pstop
-------------------------------------------------------------------------
   0  SELECT STATEMENT
   1   SORT AGGREGATE
   2    PX COORDINATOR
   3     PX SEND QC (RANDOM)             :TQ10001
   4      SORT AGGREGATE
   5       VIEW
   6        SORT GROUP BY
   7         PX RECEIVE
```

```
 8          PX SEND HASH                :TQ10000
 9           SORT GROUP BY
10            PX BLOCK ITERATOR                          1         20
11             TABLE ACCESS FULL         COSTS           1         20
```

With Oracle Database 10*g* Release 1, executing the parallel select query right after the update takes 27 seconds, while for Oracle Database 10*g* Release 2 it takes only 13 seconds. This is because the update statement updated a table existing in the same tablespace as the object accessed by the select query. In case of Oracle Database 10*g* Release 1, additional time is spend on writing out dirty buffers from PRODUCTS_AFFINITY_CARD_A table before executing the select query.

The new implementation in Oracle Database 10*g* Release 2 significantly improves parallel-query performance because a direct-path access to an object won't be slowed down by check pointing dirty buffers of other objects in the same tablespace.

SQL BUILT-IN APPLY FUNCTIONS FOR DATA MINING

The data mining process involves *building* a model using a training dataset as the first step. This model is then *applied* to new datasets to generate predictions. Since an existing model can be applied to many different new datasets, the APPLY operation is frequently invoked by customers to mine their databases. As a result, the performance and usability of the APPLY operation is very important.

In data mining terminology, APPLY is also called SCORING, hence the input dataset (table) for the APPLY operation can be called SCORING data (table). In Oracle Database 10*g* Release 1, APPLY using PL/SQL API is a "batch" scoring procedure that accepts a table as input, and provides a table as output. APPLY using JAVA API offers a record apply operation. Some of the drawbacks of the implementation in Oracle Database 10*g* Release 1 include:

- The performance is limited by the complex join(s) between the model tables and the scoring tables. Multiple materializations of intermediate results inhibit seamless SQL query processing. Many of the mappings and transformations can be so expensive as to dominate the time to perform an APPLY.
- Since the model is not shared across multiple invocations of the same apply operation, it needs to be loaded into memory every time APPLY is invoked.
- Furthermore, a novice user needs to be familiar with the JAVA and PL/SQL API before using it.

Oracle Database 10*g* Release 2 standardizes the APPLY functionality by offering SQL built-in model scoring functions. Some of the benefits of the Oracle Database 10*g* Release 2 implementation include:

- APPLY performance is improved because of the change in implementation afforded by the SQL built-in functions. In some cases, the improvement is dramatic.
- Existing query execution functionality, such as shared cursors to cache the model metadata, allows model sharing across different APPLY invocations.
- Providing a SQL built-in function for APPLY greatly improves usability. A novice user is able to quickly learn how to apply an existing model to a new dataset. Deployment of models within the context of existing applications becomes straightforward since existing SQL statements can easily be enhanced with these new APPLY functions. A SQL built-in also enables pipelining of results involving data mining predictions, providing the ability to return a few results quickly to the end user.

While the SQL built-in scoring functions in Oracle Database 10*g* Release 2 can apply models built from different data mining algorithms, the Adaptive Bayes Network (ABN) algorithm is used to build the model for this paper. ABN is a classification algorithm. The classification model built through the ABN algorithm from a training dataset can be used to classify each record in an apply dataset by indicating which class the record belongs to. For example, we may want to predict whether a customer has an affinity card based on customers' purchasing history, such as how many bulk pack DVDs the customer has bought, how many flat panel monitors the customer owns, and how many bookkeeping applications the customer has bought, etc. In such a case, each customer belongs to one of the following two classes – group of customers who have affinity cards, and group of customers who don't have affinity cards. Affinity card is called the target, and customers' purchasing history is called the predictors. Such kind of classification task begins with a training dataset for which we already know whether this customer has an affinity card. The relationship between the predictors and the target in the training data is stored in a model that can then be applied to new datasets to predict whether these new customers would have an affinity card.

For both Oracle Database 10*g* Release 2 and Oracle Database 10*g* Release 1, we use same training dataset to build an ABN model that was then applied to same scoring dataset. We measure and compare the APPLY performance for Oracle Database 10*g* Release 2 and Oracle Database 10*g* Release 1.

Our training dataset PRODUCTS_AFFINITY_CARD_B has 250 columns and 50,000 rows. The table contains an ID column (CUST_ID), a target (AFFINITY_CARD), and 248 predictors such as YRS_RESIDENCE, BULK_PACK_DVDS, and FLAT_PANEL_MONITOR, etc. Building the ABN

model 'ABN_Clas' in both Oracle Database 10*g* Release 1 and Oracle Database 10*g* Release 2 takes about 40 minutes.

With the 'ABN_Clas' model built, it can be used to make predictions from an apply dataset. The apply table PRODUCTS_AFFINITY_CARD_A contains 2 million rows. The following PL/SQL program can be used in both Oracle Database 10*g* Release 1 and Oracle Database 10*g* Release 2 to apply the model 'ABN_Clas' to the apply table and store results in a result table APPLY_RESULT.

```
-- The APPLY table needs to be binned in the same way as
the training table.
-- abn_num is the bin table that we generated when binning
the training table.
BEGIN
  DBMS_DATA_MINING_TRANSFORM.XFORM_BIN_NUM (
    bin_table_name  => 'abn_num',
    data_table_name => 'PRODUCTS_AFFINITY_CARD_A',
    xform_view_name => 'abn_apply_prepared');
END;
/

-- materialize the prepared apply table for performance.
create table abn_apply_prepared_table as select * from
abn_apply_prepared;
alter table abn_apply_prepared_table parallel;

-- APPLY THE MODEL
BEGIN
  DBMS_DATA_MINING.APPLY(
    model_name          => 'ABN_Clas',
    data_table_name     => 'abn_apply_prepared_table',
    case_id_column_name => 'CUST_ID',
    result_table_name   => 'APPLY_RESULT');
END;
/
```

Although the above code can be used in both Oracle Database 10*g* Release 1 and Oracle Database 10*g* Release 2, the underlying implementation for DBMS_DATA_MINING.APPLY is different. In Oracle Database 10*g* Release 2, the SQL built-in function PREDICTION_SET is invoked as we invoke DBMS_DATA_MINING.APPLY. In addition, these SQL built-in functions

provided by Oracle Database 10*g* Release 2 can be invoked in a SQL statement, making the APPLY functionality very easy to use. For example, the following SQL statement

```
select cust_id, prediction(ABN_Clas using *) as my_pred
from abn_apply_prepared_table;
```

predicts whether a customer would have an affinity card, for each customer stored in abn_apply_prepared_table, using the pre-built classification model ABN_Clas. The SQL built-in function PREDICTION is simpler than PREDICTION_SET. It returns the best prediction for a classification model given a set of predictors, while the PREDICTION_SET function returns a varray of objects that contains all classes and the probability for each class. With 2-million customers stored in abn_apply_prepared_table, the above SQL query returns 2-million rows. To avoid measuring the elapsed time of outputting 2-million rows, the above query was modified in the following way to facilitate performance measuring:

```
select count(distinct prediction(ABN_Clas using *)) from
abn_apply_prepared_table;
```

When applying a model, Oracle Database 10*g* Release 2 customers could select from one of the three methods. The first method is to use a PL/SQL program, as shown in the above example. This method is denoted as Oracle Database 10*g* Release 2 (PL/SQL). The second method is to use a SQL query, like the one shown above. This method is denoted as Oracle Database 10*g* Release 2 (SQL). Finally, one could also use the JAVA API that is not benchmarked in this paper. Like the PL/SQL API, the JAVA API in Oracle Database 10*g* Release 2 layers on top of the SQL built-in apply functions.

The following table summarizes the performance results for APPLY in Oracle Database 10*g* Release 1 and APPLY in Oracle Database 10*g* Release 2, using the PL/SQL method and the SQL method described.

	Elapsed Time	Temp Space
Oracle Database 10*g* Release 1	1H59M36.56S	10G
Oracle Database 10*g* Release 2 (PL/SQL)	59.6S	0
Oracle Database 10*g* Release 2 (SQL)	10.38S	0

The SQL built-in scoring functions dramatically improve the performance of APPLY, in terms of both elapsed time and temporary space usage. The improvement on temporary space usage indicates dramatic savings on memory requirements. Using the SQL built-in scoring functions directly in a SQL statement

results an even faster APPLY operation and also a user interface that is much easier for novice users to grasp. The improved performance and enhanced usability for APPLY in Oracle Database 10*g* Release 2 allows large dataset to be easily and efficiently handled.

ENHANCEMENTS FOR SUPPORT VECTOR MACHINES (SVM) ALGORITHM

Introduced in Oracle Database 10*g* Release 1, Support Vector Machines (SVM) is a state-of-the-art data-mining algorithm that supports classification and regression tasks. Classification tasks predict a target class based on a set of predictors. For example, we may want to predict whether a person has high level of income or not based on predictors such as a customer's age, occupation, etc. In this case, a person belongs to one of the two target classes - high-income group or low-income group. Instead of predicting target class (discrete or categorical value), regression tasks predict numerical/continuous targets. For example, we may want to predict the actual income amount based on education, occupation, etc.

In SVM terminology, logical rows associated with analysis entities, such as customers, are viewed as patterns. Each pattern is a vector of predictor values and a target value. SVM looks for predictive patterns (support vectors). For example, to predict high or low income, the SVM classifier separates these two target classes via a decision boundary. This decision boundary represents a hyperplane in a high-dimensional space. In the case of low-dimensional data, non-linear kernel functions (e.g., Gaussian kernels) can be used to transform the input space into a high-dimensional feature space. The high-dimensional feature space then gives SVM models the flexibility to fit arbitrarily complex non-linear decision surfaces accurately. For high-dimensional data, a non-linear kernel transformation to a higher dimensional feature space is usually not necessary and linear kernels can be used.

As with any other classification or regression algorithm, a data mining process using SVM involves *building* a model to discover the relationship between the predictors and the target. After a model is built, model *testing* evaluates the accuracy. During testing, the model is applied to an independent dataset where target values are available. A model with satisfactory accuracy can be used to make predictions for a dataset where target values are unknown. This is called *APPLY*.

The SVM implementation in Oracle Database 10*g* Release 1 is comparable with other publicly available SVM tools (e.g., SVMLight, LIBSVM, SVMTorch, and SVMFu) in terms of speed and accuracy. But there are certain limitations inherent in the standard SVM approach. The training time for SVM models scales quadratically or even cubically with the number of data records. Additionally, models with non-linear kernels, and in some cases with linear

kernels, can grow very large in size, resulting in slow model build and apply performance.

Oracle Database 10*g* Release 2 combines active learning and stratified sampling to address the inherent scalability problems of the standard SVM approach. The method effectively selects the most informative examples to reduce the number of support vectors. As a result, the time to build a model and to apply an existing model significantly decreases. The active learning method offers approximate solutions, but for the majority of cases, the accuracy of these approximate solutions is comparable to (not significantly worse than) those of standard SVM models.

To illustrate the improvement made by the new method, we first build an SVM model using a training dataset in both Oracle Database 10*g* Release 1 and Oracle Database 10*g* Release 2 and compare the building time, temporary space usage, model size, as well as the number of support vectors. We then test both models to compare the model accuracy using a test dataset. Finally, we apply both models to a apply dataset and compare the elapsed time and temporary space usage.

Our classification task is to decide whether a person has a high income or not based on the following predictors: AGE, WORKCLASS, FNLWGT, EDUCATION, EDUCATION_NUM, MARITAL_STATUS, OCCUPATION, RELATIONSHIP, RACE, SEX, CAPITAL_GAIN, CAPITAL_LOSS, HOURS_PER_WEEK and NATIVE_COUNTRY.

The training dataset SUPPLEMENTARY_DEMOGRAPHICS_B has 32,561 rows. We build an SVM model with Gaussian kernels in both Oracle Database 10*g* Release 1 and Oracle Database 10*g* Release 2. The following table captures the performances of model build in both versions.

	Elapsed Time	Temp space (MB)	Model Size (MB)	#of Support vectors
Oracle Database 10*g* Release 1	10M47.40S	3	6	11,756
Oracle Database 10*g* Release 2	9.82S	1	.5625	300

Building an SVM model in Oracle Database 10*g* Release 2 uses much less time than Oracle Database 10*g* Release 1. In addition, Oracle Database 10*g* Release 2 requires much less temporary tablespace, indicating the new method reduces the memory requirement. The improvement in elapsed time and temporary-space usage in Oracle Database 10*g* Release 2 is a result of reduced model size.

To justify the reduced model size, we test both models using 16281-row test data SUPPLEMENTARY_DEMOGRAPHICS_T to compare model accuracy.

	Accuracy Computed From Confusion Matrix	Area Under ROC
Oracle Database 10*g* Release 1	.8469	.8935
Oracle Database 10*g* Release 2	.8227	.8673

There are different measures of model accuracy. The accuracy computed from confusion matrix for Oracle Database 10*g* Release 1 is 0.8469. This means that the number of times that the SVM model built in Oracle Database 10*g* Release 1 correctly predicts the target class is 0.8469 * 16281 where 16281 is the total number of rows in the testing dataset. The number of times the target attribute's class is incorrectly predicted is (1-0.8469) * 16281.

Another useful metric for evaluating classification models is the Receiver Operating Characteristic (ROC). The area under the ROC curve (AUC) measures the discriminating ability of a binary classification model. The larger the AUC is, the higher the likelihood that an actual positive case (high-income) will be assigned a higher probability of having high income than an actual negative case (low-income). The AUC measure is especially useful for datasets with unbalanced target distribution (that is, on target class dominates the other).

As we can see from the above table, the approximate model results in only a small loss of model accuracy.

Finally, to compare the APPLY performance, we use dataset SUPPLEMENTARY_DEMOGRAPHICS_A with 97,684 records. The following table summarizes the performance of APPLY in Oracle Database 10*g* Release 1 and Oracle Database 10*g* Release 2.

	Elapsed Time	Temp Space (MB)
Oracle Database 10*g* Release 1	6M 6.29S	7
Oracle Database 10 Release 2	36.51 S	0

As a result of reduced model size, the APPLY performance is significantly improved in Oracle Database 10*g* Release 2. It not only runs faster, but also requires less memory, as indicated by the temporary space usage. The newly introduced SQL built-in functions for APPLY in Oracle Database 10*g* Release 2,

explained in the previous section of this paper, also contribute to this performance improvement.

The SVM methodology in Oracle Database 10*g* Release 2 significantly improves the performance of the SVM algorithm for both build and apply, with a small loss of model accuracy. It also allows using SVM models on large size datasets that could not be handled in Oracle Database 10*g* Release 1.

CONCLUSION

The results speak for themselves. With enhancements and new features designed to meet and exceed the requirements of data warehousing applications, Oracle Database 10*g* Release 2 delivers significantly faster and higher performance. The transparent performance gains are achieved with no extra deployment costs by simply upgrading to Oracle Database 10*g* Release 2, allowing data warehousing teams efficiently and easily leveraging their ever increasing volumes of data to obtain more business intelligence.

DSS Performance in Oracle Database 10*g* Release 2
July 2005
Author: Linan Jiang
Contributing Authors: Xumin Nie

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