

Disclaimer: Document is still under review.

Upgrade to 11g Performance Best Practices

Author: Uday Moogala
Last Updated: 10/9/2009 4:18:00 PM
Draft / Version: 1.0

Copyright © 2009 Oracle Corporation
All Rights Reserved

ORACLE®

1. Document Control

1.1 Change Record

Date	Email	Version	Change Reference
02-Oct-09	Uday.Moogala@oracle.com	1.0	First draft

1.2 Contributors

Email	Role	Content
Uday.Moogala@oracle.com	Author	
lester.gutierrez@oracle.com	Contributor	Known Issues
deepak.bhatnagar@oracle.com	Contributor	Known Issues
avasarala.murthy@oracle.com	Contributor	Known Issues

1.3 Reviewers

Name`	Position
Isam Alyousfi	Senior Director
Lester Gutierrez	Senior Architect
Jinsoo Eo	Senior Principle
Deepak Bhatnagar	Manager
Murthy Avasarala	Principle Engineer

Table of Contents

1.	Document Control.....	2
1.1	Change Record.....	2
1.2	Contributors	2
1.3	Reviewers.....	2
2.	Purpose.....	5
3.	Pre-Upgrade	6
3.1	Gather dictionary statistics:.....	6
3.2	Manage CBO statistics:.....	6
3.3	Gather performance Metrics	6
3.4	Save executions plans	6
3.5	Test.....	7
3.5.1	Capture and Replay Workload:.....	7
3.5.2	Data Masking	7
4.	Post Upgrade	8
4.1	Init.ora parameters:	8
4.2	Gather Statistics:	8
4.3	Disable Automatic Optimizer Statistics Collection	8
4.4	Plan Stability using SQL Plan Mangement (SPM).....	8
4.4.1	SQL Plan Baseline capture:	9
4.4.2	SQL Plan Baseline Selection:	10
4.4.3	SQL Plan Baseline Evolve:.....	10
4.5	Detect and Tune Performance Changes using SQL Performance Analyzer (SPA).....	10
4.5.1	SPA for Analyzing Performance.....	11
4.5.2	SPA for Upgrade Simulation	12
4.6	Real-Time SQL Monitoring.....	12
4.7	Incident Packaging Service.....	13
5.	11g New Performance Related Features.....	14
5.1	Optimizer Enhancements	14
5.1.1	Extended Statistics:	14

5.1.2	Pending statistics.....	15
5.1.3	Enhanced Statistics Collection for Partitioned Objects.....	16
5.1.4	Adaptive Cursor Sharing.....	16
5.1.5	Misc CBO Enhancements	16
5.2	Partitioning Enhancements	16
5.3	Advanced Compression:	17
6.	Fixes and Workarounds for Known Issues	18
6.1	Mandatory Patches.....	18
6.2	Optional Fixes and W/A	19
7.	Appendix: 11g Miscellaneous Enhancements	20
8.	References:.....	21

2. Purpose

This document intends to provide high level performance best practices to assist in upgrading Ebiz Suite database to Oracle Database 11g. This document will attempt to compile most major enhancements that are useful during upgrade to 11g, but will not cover every specific scenario.

Details of features that will help the upgrade go smoothly, new 11g features that you can uptake, and any Fixes and Workarounds for Known Issues that are not already covered in other Upgrade related document will be discussed in this document. This document is in addition to the Oracle 11gR1 Upgrade Companion (Note 601807.1). It is not a replacement of any Upgrade documents that are published by Oracle.

Following details are discussed in the document:

- Pre-Upgrade
 - Post Upgrade
 - 11g New Performance Related Features
 - Fixes and Workarounds for Known Issues
 - 11g Miscellaneous Enhancements
 - References
-

3. Pre-Upgrade

As a pre-upgrade step try to preserve as much information as possible about the current state of the system to act as baseline to allow comparisons of pre and post upgrade data. Following can be done (it is not a complete list) as pre-upgrade steps:

3.1 Gather dictionary statistics:

Gathering dictionary statistics will help improve upgrade performance.

```
SQL> EXEC DBMS_STATS.GATHER_DICTIONARY_STATS;
```

Gather fixed object statistics only if you see poor performance in the querying dynamic performance views, example, v\$ views. To gather the fixed objects stats:

```
SQL> EXEC DBMS_STATS.GATHER_FIXED_OBJECTS_STATS;
```

Further details on fixed objects statistics can be found in *Note 798257.1: Fixed Objects Statistics Considerations*

3.2 Manage CBO statistics:

Any new database release has significant improvements to CBO because of which majority of the queries perform well, but we do often see some regression issues. By taking backup of statistics, if need be, we can import into upgraded database as a temporary workaround while we resolve any sql regression. To make full backup of your CBO statistics:

- Run scripts `coe_create_user_coecestats.sql` and `coe_backup_cbo_stats.sql` connected as SYS.
- make an export of schema owner COECBOSTATS if pre-upgrade instance will be destroyed after the upgrade

For further details and scripts please see *Note 465787.1: Managing CBO Stats during an upgrade to 10g or 11g*

3.3 Gather performance Metrics

Gather Performance metrics to act as baseline of core business processes and some batch programs.

- Gather System level performance stats using tools like OS Watcher, and AWR. Please see *Note 301137.1: OS Watcher User Guide*.

- You can also export AWR data to do historical analysis. One option is to export AWR specific tables, and import to another database. Another option is to use script supplied by Oracle to export and import AWR data (supplied since 10.2.0.4). These scripts are `awrextr.sql` and `awrload.sql` located in `$ORACLE_HOME/rdbms/admin` directory.

3.4 Save executions plans

Oracle SQL Plan Management (SPM) features allows capturing execution plans from cursor cache of any SQL statement (into SQL Tuning Set) along with bind variables and compilation environment. Execution plans can also be captured from Trace files and AWR data using SPM.

Example of creating and capturing SQL Tuning Set:

```
SQL> DBMS_SQLTUNE.CREATE_SQLSET(sqlset_name => 'APPS_STS');
```

```
SQL> DBMS_SQLTUNE.CAPTURE_CURSOR_CACHE_SQLSET(  
    sqlset_name => 'APPS_STS',
```

```
time_limit => 600,  
repeat_interval => 5);
```

These saved execution plans can be used

- When you encounter performance issue in upgraded database. While support and development working on the issue, the saved execution plan can be used.
- As a baseline in upgraded database using SPM to ensure plan stability and later can automatically or manually evolve best execution plan. For more details please see SPM section 4.5.
- For Upgrade Simulation using Oracle SQL Performance Analyzer (SPA). Please see SPA section 4.6

3.5 Test

Make sure to do the test runs of upgrade and back out procedures to figure out any issues with the upgrade process before actual upgrade.

During the test run on pre-upgraded database, you can capture the production workload and recreate this workload in test environments using **Database Replay** feature available as part of Real Application Testing (RAT) option. You can also use **Data Masking** feature to mask the productions' sensitive data (such as credit card, salary, SSN etc) allowing you to safely and securely use production data in testing environment.

3.5.1 Capture and Replay Workload:

Database Replay allows users to perform comprehensive testing of database and infrastructure changes using real application workloads. It allows capture of production workload including concurrency, think time, and transactions dependencies, and allows users to replay the workload on a test system with the exact same production characteristics so that all problems can be identified and remediate in test before deploying the change to production. Database Replay consists of following four main steps:

1. Workload Capture
2. Workload Processing
3. Replay the captured Workload
4. Perform Regression Analysis and Report

For detailed explanation of above steps please refer to *Metalink Note 748895.1: How to Use Database Replay Feature to Help With the Upgrade*. Also see *Metalink Note 560977.1: Real Application Testing Now Available for Earlier Releases*

3.5.2 Data Masking

With Oracle Data Masking, sensitive information such as credit card or social security numbers can be replaced with realistic values, allowing production data to be safely used for development, testing, or sharing with out-source or off-shore partners for other non-production purposes. Oracle Data Masking uses a library of templates and format rules, consistently transforming data in order to maintain referential integrity for applications.

Please refer to Oracle documentation for further details.

4. Post Upgrade

Once the upgrade is complete, fresh backup of the database should be taken and identify new features to uptake in new upgraded database environment.

4.1 Init.ora parameters:

Following metalink notes should be followed while setting init.ora parameter.

Metalink note for Apps Release 12 (#396009.1)

Metalink note for Apps Release 11i (#216205.1)

4.2 Gather Statistics:

Once the upgrade is done gather following statistics:

- Gather fixed object statistics

```
SQL> EXEC DBMS_STATS.GATHER_FIXED_OBJECTS_STATS;
```

- Gather stats for Ebiz schema. Follow stats gathering strategy (using FND_STATS) you normally follow for ebiz schema.

Note:

- In 11g AUTO_SAMPLE_SIZE is very fast compared to earlier versions and gives accuracy of close to 100 % sample size. AUTO_SAMPLE_SIZE uses a new Hash-based Sampling for Column Statistics. But, at this we do not have any recommendation as we did not do thorough testing of Apps.
- We do not recommend you to gather system statistics for Apps.

4.3 Disable Automatic Optimizer Statistics Collection

In Oracle Database 11g Automatic optimizer statistics collection runs as part of the automated maintenance tasks infrastructure (AutoTask) and is enabled by default to run in all predefined maintenance windows. The AutoTask statistics collection replaces the Oracle Database 10g GATHER_STATS_JOB. Automatic optimizer statistics collection is enabled by default.

Ebiz Customer should disable this task:

```
BEGIN
  DBMS_AUTO_TASK_ADMIN.DISABLE(
    client_name => 'auto optimizer stats collection',
    operation => NULL,
    window_name => NULL);
END;
```

Along with above task Oracle 11g includes two more automated database maintenance tasks:

- Auto Space Advisor
- SQL Tuning Advisor

You can optionally disable above advisors once the upgraded database is stable. Please refer to *Metalink Note 756734.1: 11g: Scheduler Maintenance Tasks or Autotasks*.

4.4 Plan Stability using SQL Plan Mangement (SPM)

Typically every upgrade brings some plan instability because of Optimizer related enhancements. Standard practices for plan stability typically have been to use hints, Stored Outlines, SQL Profiles, stats import, setting OFE, and set optimizer parameter to influence plans.

But, Oracle 11g ensures plan stability by using SQL Plan Management. With SPM, SQL execution plans will be recorded, SQL Baseline will be created, and then *only verified and accepted plans will be used*. All this can be achieved using DBMS_SPM built-in package.

Ebiz customers with big installations are encouraged to use SPM to make upgrade go smoother. SPM allows controlled plan evolution by only using a new plan after it has been verified to perform better than the current plan. Some small amount of preparation is needed to ensure that execution plans are preserved for use after the Upgrade, but for best results, use SQL Plan Management as part of your upgrade strategy.

Using SPM, capture SQL statements into a SQL Tuning Set (STS) before performing the upgrade. STS captures SQL statements text, execution plan, execution statistics, and bind variables. Once the upgrade is completed, transform the STS into SQL Plan Baselines in the upgraded database. On the initial execution of SQL statement in the upgraded database, CBO identifies that a SQL Plan Baseline exists and decides to use it if its execution plan is more efficient. So, essentially, avoiding the execution plan regressions.

SPM can be used to:

- Save execution plans and selectively use these saved plans in *upgraded* databases while development is working on a fix for (a particular SQL's) performance issue. This way all the SQLs can take advantage new 11g features, and only regressed SQL can use execution plans from baseline while issue is being worked on.
- Save execution plans of critical processes as a pre-upgrade step and start using them as baseline in upgraded database. Later, plans can be evolved in a controlled fashion eliminating regression issues for critical processes.

The SQL Plan Management has 3 phases: Capture, Selection, and Evolve.

4.4.1 SQL Plan Baseline capture:

Baseline is stored in SQL Management Base in tablespace SYSAUX. Plan Baseline can be captures in two ways:

- Automatic capture of execution plans by setting `OPTIMIZER_CAPTURE_SQL_PLAN_BASELINES` to `TRUE`. With automatic plan capture enabled, the SPM repository will be automatically populated for any repeatable SQL statement
- Manual Plan Loading or Bulk load of execution plans using
 - `DBMS_SPM.LOAD_PLANS_FROM_SQLSET`
 - `DBMS_SPM.LOAD_PLANS_FROM_CURSOR_CACHE`

4.4.1.1 Bulk Load from a SQL Tuning Set:

Bulk loading execution plans from a STS is an excellent way to guarantee no plan changes as part of a database upgrade. The following four steps is all it takes to upgrade from 10g:

Step 1: In an Oracle Database 10g Release 2 create an STS that includes the execution plan for each of the SQL statements.

```
SQL> DBMS_SQLTUNE.CREATE_SQLSET(sqlset_name => 'APPS_STS');

SQL> DBMS_SQLTUNE.CAPTURE_CURSOR_CACHE_SQLSET(
      sqlset_name => 'APPS_STS',
      time_limit   => 600,
      repeat_interval => 5);
```

Step 2: Load the STS into a staging table and export the staging table into a flat file.

```
BEGIN
  DBMS_SQLTUNE.CREATE_STGTAB_SQLSET(
    table_name => 'APPS_STSSTG_TBL'
    ,schema_name => 'APPS'
    ,tablespace_name => 'APPS'
  );

  DBMS_SQLTUNE.PACK_STGTAB_SQLSET(
    sqlset_name => 'APPS_STS'
    ,sqlset_owner => 'SYS'
    ,staging_table_name => 'APPS_STSSTG_TBL'
    ,staging_schema_owner => 'APPS'
  );
END;
```

Now export the staging tables using Oracle Datadump into a flat file.

Step 3: Import the staging table from a flat file into an Oracle Database 11g and unload the STS.

Now Import the staging table using Oracle Datadump into a *upgraded* database.

```

BEGIN
  DBMS_SQLTUNE.UNPACK_STGTAB_SQLSET(
    sqlset_name => 'APPS_STS'
    ,sqlset_owner => 'SYS'
    ,replace => TRUE
    ,staging_table_name => 'APPS_STSSTG_TBL'
    ,staging_schema_owner => 'APPS'
  );
END;
/

```

Step 4: Use EM or DBMS_SPM.LOAD_PLANS_FROM_SQLSET to load the execution plans into the SQL Management Base.

```

Declare
  Plan_cnt NUMBER;
BEGIN
  plan_cnt :=
    DBMS_SPM.LOAD_PLANS_FROM_SQLSET(
      sqlset_name => 'APPS_STS'
      ,sqlset_owner => 'SYS'
    );
END;
/

```

4.4.2 SQL Plan Baseline Selection:

Each time a SQL statement is compiled, the optimizer first uses the traditional cost-based search method to build a best-cost plan. If the initialization parameter OPTIMIZER_USE_PLAN_BASELINES is set to TRUE (default value) then before the cost based plan is executed the optimizer will try to find a matching plan in the SQL statement's SQL plan baseline; this is done as in-memory operation, thus introducing no measurable overhead to any application. If a match is found then it proceeds with this plan. Otherwise, if no match is found, the newly generated plan will be added to the plan history; it will have to be verified before it can be accepted as a SQL plan baseline. Instead of executing the newly generated plan the optimizer will cost each of the accepted plans for the SQL statement and pick the one with the lowest cost (note that a SQL plan baseline can have more than one verified/accepted plan for a given statement). However, if a change in the system (such as a dropped index) causes all of the accepted plans to become non reproducible, the optimizer will use the newly generated cost-based plan.

4.4.3 SQL Plan Baseline Evolve:

When the optimizer finds a new plan for a SQL statement, the plan is added to the plan history as a non-accepted plan that needs to be verified before it can become an accepted plan. It is possible to evolve a SQL statement's execution plan using Oracle Enterprise Manager or by running the command-line function DBMS_SPM.EVOLVE_SQL_PLAN_BASELINE.

For further details on SPM:

[SQL Plan Management in Oracle Database 11g](#)

Metalink Note.456518.1: SQL Plan Mangement
Database 11g Performance Tuning Guide

Note.789888.1: HOW TO LOAD SQL PLANS INTO SPM FROM AWR.

Note 801033.1: HOW TO MOVE 10gR2 EXECUTION PLANS AND LOAD INTO 11g SPM.

Note.790039.1: HOW TO DROP PLANS FROM SPM REPOSITRY

4.5 Detect and Tune Performance Changes using SQL Performance Analyzer (SPA)

For testing SQL execution plans use SQL Performance Analyzer (SPA) on the upgraded databases, however, please note that this is not a substitute for full functional and stress Testing. In addition, SQL Tuning Advisor can be used to tune the regressed SQL statements.

Starting with Oracle9i, an extended SQL trace can be generated to capture SQL execution performance for key workloads for input to SPA. For Oracle Database 10g Release 2, the more efficient incremental cursor cache capture should be used to capture SQL into a SQL Tuning Set for input to SPA. This may require licensing the Real Application Testing option and Oracle Database Diagnostic and Tuning Management Packs, but makes the task of testing upgrades significantly easier and hence is the recommended best practice.

For patch numbers please see Metalink Note:560977.1

SPA is a great tool to study the impact of any parameter changes or data structure changes on SQLs (captured using SQL Tuning Set). DBAs can analyze the performance of SQL, fix any SQL regressed (can use SQL Tuning Advisor), and then promote to production, drastically reducing the risk.

4.5.1 SPA for Analyzing Performance

- Allows one to capture a representative set of SQL workload from a pre-change environment (via SQL Tuning Set – STS). Following shows populating SQL Tuning Set from cursor cache, and creating and executing SPA task to get before image.

```
BEGIN
  DBMS_SQLTUNE.CREATE_SQLSET(
    sqlset_name => 'APPS_SPA');

  DBMS_SQLTUNE.CAPTURE_CURSOR_CACHE_SQLSET(
    sqlset_name => 'APPS_SPA_STS'
    ,time_limit  => 600
    ,repeat_interval => 5
  );
END;
/

-- Create a SPA Task
DECLARE
  Spa_name  VARCHAR2(100);
BEGIN
  spa_name :=
    DBMS_SQLPA.CREATE_ANALYSIS_TASK(
      sqlset_name => 'APPS_SPA_STS'
      ,basic_filter => NULL
      ,order_by => NULL
      ,top_sql => NULL
      ,description => 'Apps SPA Task Creation'
      ,sqlset_owner => 'SYS'
    );
END;
/

-- Execute the SPA task
BEGIN
  DBMS_SQLPA.EXECUTE_ANALYSIS_TASK(
    task_name => 'SPATASK_01'
    ,execution_type => 'Apps Test'
    ,execution_name => 'APPS_SPA_BEFORE'
    ,execution_desc => 'Apps SPA Before'
    ,execution_params => DBMS_ADVISOR.ARGLIST('TEST_EXECUTE', 'FULL')
  );
END;
/
```

- Execute and capture the same workload after the environment or parameter change.

```
BEGIN
  DBMS_SQLPA.EXECUTE_ANALYSIS_TASK(
    task_name => 'TASK_01'
    ,execution_type => 'Apps Test'
    ,execution_name => 'APPS_SPA_AFTER'
    ,execution_desc => 'Apps SPA After'
    ,execution_params => DBMS_ADVISOR.ARGLIST('TEST_EXECUTE', 'FULL')
  );
END;
/
```

- A comparison report is then produced that identifies SQL regressions

```
BEGIN
  DBMS_SQLPA.EXECUTE_ANALYSIS_TASK(
    task_name => 'TASK_01'
    ,execution_type => 'compare pre/post changes'
    ,execution_name => 'APPS_SPA_COMPARE'
  );
END;
```

```

        ,execution_desc => 'Apps SPA 01 Compare'
        ,execution_params =>
            DBMS_ADVISOR.ARGLIST(
                'EXECUTION_NAME1', 'APPS_SPA_BEFORE'
                , 'EXECUTION_NAME2', 'APPS_SPA_AFTER'
                , 'COMPARISON_METRIC', 'ELAPSED_TIME'
            )
    );
END;
/

SET LINESIZE 2000
SET SERVEROUTPUT ON SIZE UNLIMITED
DECLARE
    sparep CLOB;
BEGIN
    SELECT DBMS_SQLPA.REPORT_ANALYSIS_TASK('TASK_01')
        INTO sparep
        FROM DUAL;
    DBMS_OUTPUT.PUT_LINE(sparep);
END;
/

```

4.5.2 SPA for Upgrade Simulation

SPA can also be used to see the impact of optimizer version change, because of upgrade, on the SQL workload via *'Upgrade Simulation Panel'*. To do that:

- Capture workload into STS
 - Load into staging table and export to a flat file
 - Transport and import STS into Database 11g
- Above three steps are described with example in SQL Plan Management section.
- In 11g's EM use SPA panel and select 'Upgrade Simulation Panel'
 - Create SPA Task and select appropriate Optimizer versions.
 - Use SPA panel to view the comparison report.

Note:

1. SPA does not track the source of the SQL (module, action, etc.). Therefore, it may be challenging to associate a poor performing SQL statement with a business activity.
2. SPA results on SQL statements making use of global temporary tables will be less reliable since Optimizer assumes zero rows in global temp table during SPA task execution. Also, a SPA session would not have client info and Org setting which will influence the performance of a SQL statement.

4.6 Real-Time SQL Monitoring

The real-time SQL monitoring feature of Oracle Database enables you to monitor the performance of SQL statements while they are executing. By default, SQL monitoring is automatically started when a SQL statement runs parallel, or when it has consumed at least 5 seconds of CPU or I/O time in a single execution. Once monitoring is initiated, an entry is added to the dynamic performance view V\$SQL_MONITOR. This entry tracks key performance metrics collected for the execution, including the elapsed time, CPU time, number of reads and writes, I/O wait time and various other wait times. These statistics are refreshed in near real-time as the statement executes, generally once every second.

Ebiz customer facing performance issues can use this feature to monitor the sql performance. This help customers and support to quickly identify the bottleneck in poorly performing SQL statements.

You can capture Real-Time SQL Monitoring report in html format using DBMS_SQLTUNE.REPORT_SQL_MONITOR package.

If you want to get a SQL Monitor report for a statement you just ran in your session (similar to dbms_xplan.display_cursor) then use this command:

```

select
    DBMS_SQLTUNE.REPORT_SQL_MONITOR(
        session_id=>sys_context('userenv','sid'),
        report_level=>'ALL') as report
from dual;

```

4.7 Incident Packaging Service

A DBA can automatically and easily gather all diagnostic data (traces, health check reports, SQL test cases, and more) pertaining to a critical error and package the data into a zip file suitable for transmission to Oracle Support. Because all diagnostic data relating to a critical error are tagged with that error's incident number, the DBA does not have to search through trace files and other files to determine the files that are required for analysis; the incident packaging service identifies all required files automatically and adds them to the package.

Please see Oracle Database Administrator's Guide for more information about these components. Also see:
Metalink Note.443529.1: 11g Quick Steps to Package and Send Critical Error Diagnostic Information to Support
Metalink Note.738732.1: ADR How to Package Diagnostic Information in 11g

5. 11g New Performance Related Features

5.1 Optimizer Enhancements

5.1.1 Extended Statistics:

Oracle database 11g now provides mechanism to gather *statistics on a group of columns or expression*. In the real world examples we often have correlation between two or more columns in table and this correlation can affect the selectivity of the column group.

Oracle uses workload statistics to determine the column groups, but you can also create manually using `dbms_stats.CREATE_EXTENDED_STATS` procedure or using `method_ops` in `dbms_stats.gather%` procedures.

Ebiz suites `FND_STATS` package is being enhanced to uptake this feature.

For example, in ebiz apps, consider `FND_LOOKUP_VALUES` table in which column `lookup_type` and `view_application_id` are correlated. The extended statistics are used by Oracle to estimate the combined selectivity of the predicates. The new statistics will provide additional information that the optimizer needs to choose better execution plans using more accurate counts of rows.

To create a column group based on `FND_LOOKUP_VALUES` (`lookup_type`, `view_application_id`) you can use `DBMS_STATS.create_extended_stats` procedure.

Following is the small demonstration of multi-column stats impact on `FND_LOOKUP_VALUES` query.

```
SQL> select VIEW_APPLICATION_ID,lookup_type, count(*) from FND_LOOKUP_VALUES group by
VIEW_APPLICATION_ID,lookup_type order by 3 desc;
```

```
VIEW_APPLICATION_ID LOOKUP_TYPE                COUNT(*)
-----
3 NO_POSTAL_CODE                13698
...
3 AE_AREA_CODES                 1350
```

```
--
-- Compute statistics on FND_LOOKUP_VALUES
--
```

```
SQL> EXEC apps.FND_STATS.GATHER_TABLE_STATS('APPLSYS', 'FND_LOOKUP_VALUES', percent=>100);
```

```
--
-- Now run query against the table
-- Note the number of rows expected are 12
--
```

```
SQL> explain plan for select * from FND_LOOKUP_VALUES where VIEW_APPLICATION_ID=3 and
lookup_type='NO_POSTAL_CODE';
```

Explained.

```
SQL> select plan_table_output from table(dbms_xplan.display('plan_table',null,'serial'));
```

```
PLAN_TABLE_OUTPUT
```

```
-----
Plan hash value: 3344024189
```

```
-----
| Id | Operation                | Name                | Rows | Bytes | Cost (%CPU)| Time     |
-----
|  0 | SELECT STATEMENT         |                     |  12  | 1680 |  13  (0)| 00:00:01 |
|  1 | TABLE ACCESS BY INDEX ROWID | FND_LOOKUP_VALUES  |  12  | 1680 |  13  (0)| 00:00:01 |
|*  2 | INDEX RANGE SCAN        | FND_LOOKUP_VALUES_U2 |  12  |      |    3  (0)| 00:00:01 |
-----
```

```
Predicate Information (identified by operation id):
-----
```

```
2 - access("LOOKUP_TYPE"='NO_POSTAL_CODE' AND "VIEW_APPLICATION_ID"=3)
```

```
--
-- Now create extended statistics on the column_group view_application_id, lookup_type
--
```

```
DECLARE
```

```

l_cg_name VARCHAR2(30);
BEGIN
  l_cg_name := DBMS_STATS.create_extended_stats(ownname => 'APPLSYS',
                                               tabname => 'FND_LOOKUP_VALUES',
                                               extension => '(VIEW_APPLICATION_ID, LOOKUP_TYPE)');

  DBMS_OUTPUT.PUT_LINE(l_cg_name);
END;
/

```

```

--
-- Now gather statistics again, but this time with column group
--

```

```

SQL> exec dbms_stats.gather_table_stats
      ('APPLSYS','FND_LOOKUP_VALUES',
       method_opt => 'FOR ALL COLUMNS SIZE 1
                    FOR COLUMNS LANGUAGE size skewonly
                    FOR COLUMNS VIEW_APPLICATION_ID size skewonly
                    FOR columns (VIEW_APPLICATION_ID, LOOKUP_TYPE) size skewonly',
       estimate_percent => 100);

```

```

--
-- Now run the sql again
-- Not that now the estimated rows are 12580 which is very close to actual
--

```

```

SQL> explain plan for select * from FND_LOOKUP_VALUES where VIEW_APPLICATION_ID=3 and
lookup_type='NO_POSTAL_CODE'

```

Explained.

```

SQL> select plan_table_output from table(dbms_xplan.display('plan_table',null,'serial'));

```

PLAN_TABLE_OUTPUT

Plan hash value: 691804408

```

-----
| Id | Operation          | Name                | Rows  | Bytes | Cost (%CPU)| Time     |
-----
|  0 | SELECT STATEMENT   |                     | 12580 | 1867K | 2729  (2)| 00:00:33 |
|*  1 | TABLE ACCESS FULL| FND_LOOKUP_VALUES  | 12580 | 1867K | 2729  (2)| 00:00:33 |
-----

```

Predicate Information (identified by operation id):

```

-----
1 - filter("LOOKUP_TYPE"='NO_POSTAL_CODE' AND "VIEW_APPLICATION_ID"=3)

```

Many SQLs would get benefit out of this feature because optimizer will come up with better plan as it has statistics about multi-column relationships. The multi-column statistics are stored as virtual columns.

Note that we have seen high parse time issues when virtual columns are involved, but not for all cases. On a case by case basis, we are using workaround to currently set `_replace_virtual_columns` to false. Please see known issues section for issues.

5.1.2 Pending statistics

Allows you to test the impact of statistics changes before publishing for use by CBO at runtime. `OPTIMIZER_PENDING_STATISTICS` initialization parameter should be set to true for the CBO to use pending statistics.

This feature can be used in scenarios where a particular SQL statements performance has degraded and you wanted to gather statistics. You can gather statistics on tables (can use new features like extended stats) and study the impact of it on SQL statement. If SQL statement performance is improved with new statistics, then you can publish the statistics for production use.

5.1.3 Enhanced Statistics Collection for Partitioned Objects

Oracle 11g includes improvements to statistics collection for partitioned objects so untouched partitions are not rescanned. This significantly increases the speed of statistics collection on large tables where some of the partitions contain static data. This enhancement is transparent to the user. Ebiz customer need not do anything special to take advantage of this feature.

5.1.4 Adaptive Cursor Sharing

This is a new feature introduced in 11g, which is on by default and is transparent to the user, to alleviate issues with bind peeking feature introduced in 9i. Adaptive Cursor Sharing will improve plans for the queries containing bind variables. As a result of this features we will see more cursors for a SQL statement containing bind variables.

With bind peeking, Optimizer will peek at bind values during the initial execution of a SQL statement and generate an optimized execution plan for the peeked bind values. It is as if literals have been used in the SQL statement. For all subsequent execution of the same SQL statement, even with different bind variables, this generated execution plan will be used. However, if there is a data skew in the columns with binds, then the plan generated may not be optimal for all possible values of bind variables.

With Adaptive Cursor Sharing, multiple execution plans for a SQL statement with bind values will be stored and used depending on the bind values.

Please refer to *'Metalink Note 836256.1: Adaptive Cursor Sharing in 11G'* for detailed explanation of this feature.

5.1.5 Misc CBO Enhancements

Additional CBO enhancements which are out of scope of this document, but worth mentioning are:

- **Join Predicate Pushdown**
Join predicate pushdown is now available for queries with group-by, distinct, semi/anti-joined view.
- **Enhanced Partition Pruning**
Partition Pruning now uses bloom filtering instead of subquery pruning. While subquery was activated on cost based decision and consumed internal (recursive) resources, pruning based on bloom filtering is activated all the time without consuming additional resources. Bloom filtering gives better performance with large sets.
- **Distinct Elimination**
CBO removes the distinct if the query block is guaranteed to return distinct rows without it.
- **Native Implementation of Full Outer Joins**
Before Oracle Database 11g, ANSI full outer joins were converted into a UNION ALL query with two branches, one branch consist of LEFT OUTER JOIN AND other branch consist of NOT EXIST subquery. Oracle Database 11g introduced a native support for hash full outer joins. This improves the performance of a full outer join.
- **Stored Outlines**
Stored Outlines are deprecated in Oracle Database 11g. Oracle highly recommends the use of SQL plan baselines instead of the stored outlines.
- **New Density Calculation**
There is a new algorithm to calculate density in Oracle Database 11g when histograms are present on the columns. This shows up in the 10053 traces as NewDensity. This will affect cardinality ESTIMATE calculations. The NewDensity is not stored in the data dictionary. The old density values are stored in data dictionary. NewDensity is calculated at run time.

Please refer to [New Density calculation in 11g](#) white paper by A Dell'Era

5.2 Partitioning Enhancements

Oracle Database 11g has introduced several new partitioning methods:

Interval Partitioning
New Composite Partitionings
Reference Partitioning
Virtual Column-Based Partitioning
System Partitioning

Interval partitioning is essentially an extension of range partitioning which instructs the database to automatically create partitions of a specified interval when data inserted into the table exceeds all of the range partitions. You must specify at least one range partition. The range partitioning key value determines the high value of the range partitions, which is called the transition point, and the database creates interval partitions for data beyond that transition point.

Following new composite partitioning types are introduced in 11g:

Range-Range: Composite range-range partitioning enables logical range partitioning along two dimensions and typically are use full to store time-dependent data.

List-Range: Composite list-range partitioning enables logical range subpartitioning within a given list partitioning strategy.

List-Hash: Composite list-hash partitioning enables hash sub-partitioning of a list-partitioned object and is useful for large tables

List-List: Composite list-list partitioning enables logical list partitioning along two dimensions and again useful for large tables that are often access on difference dimensions.

With the Interval partitioning, the following composite partitioning methods are also supported: **Interval-Range, Interval-List and Interval-Hash.**

Provide some Ebiz specific examples

Along with above partitioning types, 11g provides Reference Partitioning, System Partitioning, and Virtual Column Based Partitioning. For details on these features please refer to '*Metalink Note 452447.1: 11g Partitioning Enhancements*'.

5.3 Advanced Compression:

The Oracle Database 11g Advanced Compression Option introduces a comprehensive set of compression capabilities to help customers maximize resource utilization and reduce costs.

Oracle 11g introduces supports regular structured data (numbers, characters), unstructured data (documents, spreadsheets, XML and other files), and other backup data.

The Oracle Advanced Compression option supports OLTP table compression among other thing i.e., it allows data to be compressed during all types of data manipulation operations, including conventional DML such as INSERT and UPDATE.

We do not have any recommendation for Ebiz customers regarding Advanced Compression feature as we did not do any performance benchmarking.

6. Fixes and Workarounds for Known Issues

6.1 Mandatory Patches

Issue	Description	Fix
High Parse Time	Bug 8508056. High Parse time when virtual columns or function based index is present	Patch: 8508056
Cardinality issue	Bug 5483301. Low cardinality computed when searching for a missing value in a freq histogram with stale statistics	Patch 5483301
	Bug 8602840. Self-join with inequality join predicate can cause incorrect cardinality estimate	Patch: 8602840
Subquery unnesting	Bug 8580883. sub-optimal plan with subquery unnesting in DML statement though the cost of subquery unnesting with interleaved CVM is cheaper	Patch: 8580883
High 'latch: row cache objects' contention	Bug 7291739. High 'latch: row cache objects' contention on dc_rollback_segments together with high 'enq: US - contention'	Patch: 7291739
Wrong Results	Bug 8426380. Wrong results with join elimination (with OUTER joins and tables with Check Constraints)	Patch: 8447623
Compression		Please refer to Metalink Note 244040.1
Plan regression	Bug 6438892. Queries which include a ROWNUM predicate may get a sub-optimal plan, typically including an expensive NESTED LOOPS join.	Patch: 6438892
ORA-00700: KESQSMakesQL	Bug 8987298: APPS ST GSI : ORA-00700: KESQSMakesQL-INVSTAT :ELPSTIME & :CPUTIME	Patch 7974905
CATUPPST.SQL performance issue	Bug 8738008: CATUPPST.SQL performance issue	Patch 8738008
EBusiness Suite R12 Certification Patch Bundle #1	For 11.1.0.7	Patch 7684818

EBusiness Suite R12 Certification Patch Bundle #1 (Patch 7684818):

- Bug:4247037 RFID-EPC generation
- Bug:6263237 Package compilation can dump with fine grain dependencies
- Bug:6530141 False ORA-979 can occur on an UPDATE DML
- Bug:6598432 ORA-1466 from read-only transaction when client / server in different timezones
- Bug:6815733 OERI [qctctel] from cost based transformation with subquery in ORDER BY clause
- Bug:6870937 Small memory leak / OERI[729] using SHARED_CONTEXT_SENSITIVE RLS policy
- Bug:6972189 Invalid package bodies when using _load_without_compile option
- Bug:6991626 Datapump export fails with ORA-39126 / ORA-22813
- Bug:7000281 Difference in FORALL statement behaviour in 11g
- Bug:7111245 Object statistics can have excessive overhead
- Bug:7243270 utlrp.sql errors with ORA-2243 during upgrade
- Bug:7253531 A dump [ttci2u] passing LOB data over heterogeneous connection with multibyte
- Bug:7277741 Incorrectly escaped characters from XMLTransform
- Bug:7295298 Poor Subquery filter order / Queries against ALL_OBJECTS can be slow
- Bug:7319922 Inconsistent results with XML Transform for &
- Bug:7327166 Dump (under evacssr) from CASE expression over UNION ALL view
- Bug:7556778 ORA-25137 when trying to CAST() on bind variables
- Bug:7708340 PLS-908 / ORA-4052 from PLSQL RPC from 11g to 10.2 or earlier

11.1.0.7 Patch Set - Availability and Known Issues Note 738538.1

11.1.0.6 Base Release - Availability and Known Issues Note 454506.1

6.2 Optional Fixes and W/A

Issue	Description	Fix/Workaround
High Parse Time	High Parse time is seen in 'execute' phase of the query	W/A: "_fix_control"='4887636:off'
High Parse Time	Bug 8792821. Parse time issue when moving from 10.2.0.4 to 11.1.0.7	W/A: Pin the problematic cursor Bug is still open
row cache lock	Bug 8874285: SMON and the foreground process received deadlock between ROW CACHE LOCK AND TX enqueue	set event 10052 to disable clean-up of non-existent objects from 'obj\$' *and* run a script that would delete the non-existing objects from obj\$ at periodic interval(maybe every 5 mins). Also see WebIV note 21160.1.

7. Appendix: 11g Miscellaneous Enhancements

Feature	Brief Description
Invisible Indexes	<p>Allows an index to exist, but unused. Useful to testing the decommissioning of an index, or private testing.</p> <p>An invisible index is an alternative to making an index unusable or even to drop it. An invisible index is maintained for any DML operation but is not used by the optimizer unless you explicitly specify the index with a hint.</p> <p>Applications often have to be modified without being able to bring the complete application offline. Create invisible indexes temporarily for specialized non-standard operations, such as online application upgrades, without affecting the behavior of any existing application. Furthermore, invisible indexes can be used to test the removal of an index without dropping it right away, thus enabling a grace period for testing in production environments.</p>
Flush a single sql statement	<p>Useful to force a hard parse on a single statement.</p> <p><code>dbms_shared_pool.purge()</code></p> <p>Reference: http://kerryosborne.oracle-guy.com/2008/09/flush-a-single-sql-statement/</p>

8. References:

Oracle Database Upgrade Guide
Oracle Database Performance Tuning Guide

[Change - why upgrasde now presentation](#) -- Tom Kyte
[Upgrading to 11g – Best Practices](#) by Ashish Agrawal

Other Important Notes:

Note: 601807.1 Upgrade Companion 11g
Note: 429825.1 Complete Checklist for Manual Upgrades to 11g
Note: 454506.1 Known Issues and Alerts 11.1.0.6

Performance Testing

Note: 560977.1 Real Application Testing available for earlier releases
Note: 562899.1 Using SQL Performance Analyzer for upgd. 9.2 to 10.2

Note 554539.1: Using Database Partitioning with Oracle E-Business Suite