Upgrading from Oracle 9i to Oracle Database 11g:

A Real World Customer Experience

An Oracle White Paper
October 2008

Upgrading to Oracle Database 11g

EXECUTIVE OVERVIEW

An Oracle database upgrade is often considered as a risky task by database IT people. Upgrade tests have to be run, all applications have to be verified and validated in the new database environment and performance at the same level or better has to be ensured and achieved.

Upgrading directly from Oracle 9i to Oracle Database 11g will not only guarantee Premium Support until August 2012 – it will offer you many useful features to make this upgrade the most predictable and risk free database upgrade ever, ensuring excellent performance in the new Oracle Database 11g environment.

This technical Oracle White Paper will cover a whole Proof-of-Concept (PoC) done with one of the largest retail companies of the world. The goals of this "Move on to Oracle Database 11g" PoC were to test the easiest way to upgrade 400 databases directly from Oracle 9.2.0.8 to Oracle Database 11.1.0.6, and to evaluate Oracle Database 11g's performance features such as Oracle Tuning Pack, SQL Plan Management, Real Application Testing with Database Replay, SQL Performance Analyzer, and PL/SQL Native Compilation. Thereby it should be assured that nightly OLTP batch runs complete in the same amount of time, or faster, than in the current production environment.

The overall results were truly amazing. Besides achieving faster performance with Oracle Database 11g compared to Oracle 9i, there were no application changes necessary – and the whole upgrade project required far fewer working hours and resources compared to previous database upgrades. After evaluating all PoC results the customer started upgrading all 400 Oracle 9i databases directly to Oracle Database 11.1.0.7 and will complete the whole "Move on to 11g" project by spring 2009. This will also include the actual rollout and upgrade of more than 150 Oracle Application Server instances to Application Server 10.1.2.0.3. Oracle Grid Control's Provisioning Pack will be used for the rollout of the database and the application server software.

THE 3 PHASES OF PROJECT “MOVE ON TO 11G”

Project “Move on to Oracle Database 11g” was conducted from May to September 2008 at one of the world’s largest operating retail customers, running approximately 400 Oracle databases on Oracle 9.2.0.8. Project targets had been defined as:

1. Find the best solution for upgrading 400 Oracle 9i databases on 140 servers directly to Oracle Database 11g in a minimum amount of time and with very low manual intervention needed. Average database size is 120GB.
2. Ensure that highly critical nightly batch jobs finish in the same amount of time that they do in Oracle 9i. These nightly jobs are accumulated OLTP runs and process the complete merchandise planning for several thousand retail stores. The reference run completes in 1:45h. These jobs are extremely CPU intensive – they use a new 16 CPU server to full capacity during the whole run. In addition, approximately 25GB of redo log information is generated for each database every night.
3. Determine whether any application changes will be required for the move to Oracle Database 11g. In the past this has caused a lot of effort, such as during the migration from Oracle 8i to Oracle 9i Unicode.
4. Evaluate the new Oracle Database 11g performance features including SQL Tuning and Access Advisor, SQL Plan Management, SQL Performance Analyzer, Database Replay, and PL/SQL Native Compilation, and their ability both to improve performance and to reduce the manual effort required by the upgrade to Oracle Database 11g.

The project had been divided into 3 phases:

1. Setup of the reference system and creation of a repeatable production workload
2. Upgrade to Oracle Database 11g and comparison of performance figures without any optimizations
3. Performance optimization leveraging the Oracle Database 11g features
    a. Initialization parameter tuning
    b. Detection of changing execution plans with SQL Performance Analyzer
    c. Load simulation with Real Application Testing: capture a complete workload and replay it against an upgraded database
    d. Preserving Oracle 9i-like execution plans with SQL Plan Management

e. Improving overall PL/SQL performance with PL/SQL Native Compilation

f. SQL query and workload optimization with SQL Tuning and Access Advisors

g. Switch off any tuning enhancements in Oracle Database 11g and compare the results to Oracle 9i

Setup of the reference system

One IBM P670 system with 16 CPUs and 32 GB of RAM has been provided for test purposes. This system is identical to the production systems. The operating system is AIX 5.3 TL8, and the storage subsystem is an EMC DMX2000.

In production each server runs 3 databases in parallel, each of them hosting 10 retail stores. The current production database release is Oracle Database 9.2.0.8.

One ORACLE_HOME containing 9.2.0.8 and another home containing Oracle Database 11.1.0.6 were installed on the reference system. To leverage the new Real Application Testing pack according to Metalink-Note: 560977.1, the following patches were applied:

- Oracle Database 11.1.0.6:
  Patch 6865809: SQL Performance Analyzer – 11g can extract 9i trace information
- Oracle Database 9.2.0.8:
  Patch 6973309: Capture will be available in 9.2.0.8

Three different 9.2.0.8 production databases were copied and restored to the reference system using RMAN. A repeatable nightly batch run was also provided. This batch run is an accumulated OLTP run of several hours and is representative of a real world scenario.

Upgrading to Oracle Database 11g

Upgrading directly to Oracle Database 11g is supported for Oracle 9.2.0.4 and all later releases. It can be done either with the Database Upgrade Assistant (DBUA), a graphical user interface tool, or manually using the upgrade scripts. Upgrade with the DBUA is the Oracle-recommended method because it performs all necessary checks and changes.

In this case the customer had choosen the manual upgrade method instead of the DBUA. The DBUA would have been able to complete the whole upgrade process unattended in silent mode except for running operating system checks as well as controling the backup with RMAN and applying required database patches. Therefore a complex shell script from previous upgrade projects had been customized to accomodate the Oracle Database 11g requirements.

The manual upgrade on the command line with scripts can be divided into the following steps:

- Sanity maintenance of the Oracle 9i installation
    - Drop a possibly existing table SYS.PLAN_TABLE
    - Apply patch 6973309 to get Database Capture functionality in Oracle 9i. Please see Metalink-Note: 560977.1 for further details.
    - Drop all entries in OBJ$ of type=10 – these are leftovers from database links that have been created and dropped in the past:

      ```
      delete from obj$ where (name,owner#) in (
        select o.name,u.user#  from user$ u, obj$ o
          where u.user# (+)=o.owner# and o.type#=10
              and not exists
                (select p_obj#
                  from dependency$
                  where p_obj# = o.obj#)
        );
      ```

    - Prior to upgrading, apply the new time zone V4 files via a patch – otherwise an upgrade to Oracle Database 11g cannot be started. In 2007 the begin date and end date for the Daylight Savings Time (DST) has been changed for 7 time zones. These time zone definitions are important for the correct processing of the `TIMEZONE` datatype within the Oracle server. Metalink-Note: 413671.1 points to the correct patch number for each release and answers all questions concerning DST patches.
    - Run the utlu111i.sql script which is delivered within the Oracle Database 11g installation in the `$ORACLE_HOME/rdbms/admin` directory. This script must be run in the environment of the source database from SQL*Plus, while connected as user `SYS AS SYSDBA`. This script will examine the database to be upgraded, and

will display recommendations about parameters requiring change or removal.

- If an SPFILE is used than an editable init.ora should be created: `CREATE PFILE='...' FROM SPFILE='...';`
  All recommended changes should be applied to this init.ora file to meet the Oracle Database 11g requirements. Existing underscore parameters and database events should be removed from the initialization file as well.
  When upgrading an Oracle 9i database directly to Oracle Database 11g the initialization parameter `COMPATIBLE` has to be set to at least `10.0.0`. To enable all Oracle Database 11g new features and use the new optimizer `COMPATIBLE` should be set to `11.1.0`.

- For safety reasons it is strongly recommended to check `REGISTRY$DATABASE` for `TZ_VERSION`=4. If the time zone patches were not applied to the Oracle 9.2 database and the attempt was made to upgrade directly to Oracle Database 11g, then the database has to be opened in `STARTUP UPGRADE` mode in the new environment with an increased `COMPATIBLE` setting (`COMPATIBLE` has to be set to at least `10.0.0` or higher). At this point an upgrade will fail because of the missing time zone patches. But a downgrade is supported only to Oracle Database 10g – so it will not be possible to clean-up the Oracle 9.2 database and start the upgrade process again. In this case the whole database will have to be restored from a backup.

- Analyze the complete data dictionary and all default user schemas immediately prior to the upgrade. This will speed up the upgrade significantly, and should be done for users `SYS`, `SYSTEM`, `MDSYS`, `DBSNMP`, `OUTLN`, `SI_INFORMTN_SCHEMA`, `ORDPLUGINS`, `ORDSYS`, `LBACSYS`, `WKSYS`, `XDB`, `CTXSYS` and `WMSYS` as long as they exist in the database.:

  ```
  EXEC DBMS_STATS.GATHER_SCHEMA_STATS('SYS',
  options => 'GATHER',
  estimate_percent=>DBMS_STATS.AUTO_SAMPLE_SIZE,
  method_opt => 'FOR ALL COLUMNS SIZE AUTO',
  cascade => TRUE);
  ```

- To minimize the downtime during the upgrade process one can switch the database to `NOARCHIVELOG` mode. Once the upgrade has been completed successfully the `ARCHIVELOG` mode has to be switched on again.

- During the upgrade from Oracle 9i to Oracle Database 10g/11g all synonyms will be recompiled during the upgrade process. For each synonym this will take approximately 0.1 seconds. If a large number of synonyms have been created in the database then the time needed to recompile all synonyms must be added to the upgrade time.

- A complete online backup of the source database must be taken prior to starting the upgrade. Oracle Recovery Manager (RMAN) is the recommended tool to backup the database. Please make sure that your backup can be restored and recovered to the desired point in time before you proceed. Testing and verifying the restore and recovery operation as a fallback scenario is vital to the upgrade process.
- The Oracle Database 11g software should be installed while the above mentioned sanity operations are being executed. Metalink's Recommended Patches section should be queried and Metalink-Note: 738538.1 should be consolidated for possible alerts and important things to know. If there is a patch set available for the target (upgraded) release, then this should be applied prior to upgrading for two reasons: To decrease downtime by requiring only one recompilation run, and to preserve the possibility of downgrading to Oracle Database 10g.
- A new listener has to be created. This can be done with Oracle Network Configuration Assistant (NETCA).
- When switching to the new Oracle Database 11g environment, verify that LIBPATH and ORA_NLS10 have been set correctly to point to the new $ORACLE_HOME.
- The database will now be started in `STARTUP UPGRADE` mode. This will disable system triggers and replication dependency tracking, and will suppress unwanted error messages such as `ORA-904: table or view does not exist`. This will make it easier to screen the spool file for errors during the upgrade.

  ```
  SQL> spool /tmp/upgrade.log
  SQL> startup upgrade pfile='<new-init.ora>'
  ```
- Once an Oracle 9i database has opened in `STARTUP UPGRADE` mode the new SYSAUX tablespace has to be created. This is only necessary when upgrading from Oracle 9i; in Oracle Database 10g the SYSAUX tablespace already exists:

  ```
  CREATE TABLESPACE SYSAUX
   DATAFILE '/oradata/sysaux_01.dbf' size 2048M
   EXTENT MANAGEMENT LOCAL
   SEGMENT SPACE MANAGEMENT AUTO
   ONLINE;
  ```
- Run the upgrade script `catupgrd.sql`. This will upgrade all data dictionary objects and all components installed in the database.

  ```
  SQL> @?/rdbms/admin/catupgrd.sql
  ```

  Once it has completed, the upgrade script will shutdown the database so that a subsequent `STARTUP` will be necessary to proceed:

```
SQL> startup pfile=’<new-init.ora>’
```

- If an SPFILE has been used before then it should be created from the new init.ora:

  ```
  SQL> create spfile=’...’ from pfile=’<new-init.ora>’;
  ```

- When upgrading from Oracle Database 10g to Oracle Database 11g a script must be executed to upgrade the Automatic Workload Repository (AWR) to the new database release:

  ```
  SQL> @?/rdbms/admin/catuppst.sql
  ```

  This step is not necessary when upgrading from Oracle 9i because AWR was first introduced in Oracle Database 10g.

- Recompilation is the last step of the upgrade process. It will be done in parallel automatically, based on the number of CPUs available in the system:

  ```
  SQL> @?/rdbms/admin/utlrp.sql
  ```

  The default would be to recompile with one parallel thread fewer than the number of CPU cores installed in the system. If there are other databases and applications running on this server it may be useful to limit the number of parallel threads even more:

  ```
  SQL> @?/rdbms/admin/utlprp 7
  ```

  This will start the parallel recompilation with 7 parallel threads.

- Finally, a run of the post-upgrade status script `utlu111s.sql` is required to verify that all components have been upgraded successfully to Oracle Database 11g:

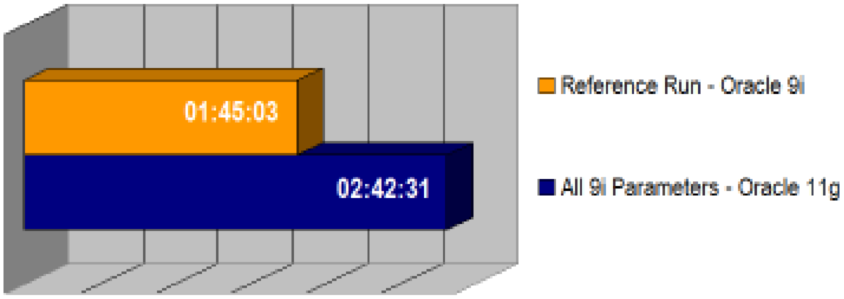
  ```
  SQL> @?/rdbms/admin/utlu111s.sql

  Oracle Database 11.1 Post-Upgrade Status Tool           08-13-2008 10:07:44

  Component                             Status         Version  HH:MM:SS
  Oracle Server                          VALID      11.1.0.6.0  03:06:48
  JServer JAVA Virtual Machine           VALID      11.1.0.6.0  00:15:43
  Oracle Workspace Manager               VALID      11.1.0.6.0  00:01:38
  Oracle XDK                             VALID      11.1.0.6.0  00:35:35
  Oracle XML Database                    VALID      11.1.0.6.0  00:04:28
  Oracle Database Java Packages          VALID      11.1.0.6.0  00:00:33
  Oracle Multimedia                      VALID      11.1.0.6.0  00:05:49
  Spatial                                VALID      11.1.0.6.0  00:08:53
  Gathering Statistics                                          00:08:18

  Total Upgrade Time: 04:27:49
  ```

The database has now been successfully upgraded to Oracle Database 11g!

Test run 1 – Workload on 11.1.0.6 - no parameter changes and no new features

For test run 1 the database has just been upgraded from Oracle 9i to Oracle Database 11g without any parameter changes except required init.ora parameter changes such as `diagnostic_dest`. Even special underscore parameters used in the Oracle 9i environment have been reused and not removed.

A reference run averaging 5 independent test runs on the same system against a 9.2.0.8 production database clone was used as the basis for comparison. Average run time in Oracle 9i was 1:45:03.



Without any adjustments, using all Oracle 9i related initialization parameters (including those that are obsolete or superseded), and without any new statistics or tuning features, the workload completed in 2:42:31 directly after the upgrade to Oracle Database 11g.

While the upgrade went very smoothly, some tuning is clearly needed to achieve the performance goals set forth for this project.

Performance Optimization

The performance optimization goal had been to achieve identical completion times for the nightly OLTP batch runs. Therefore three different areas of possible performance optimizations have been identified:

- Initialization parameter recommendations
- SQL Plan Management
- SQL Tuning Advisor and Automatic SQL Tuning

Initialization parameter recommendations

The following init.ora parameter recommendations were identified and implemented during the tests:

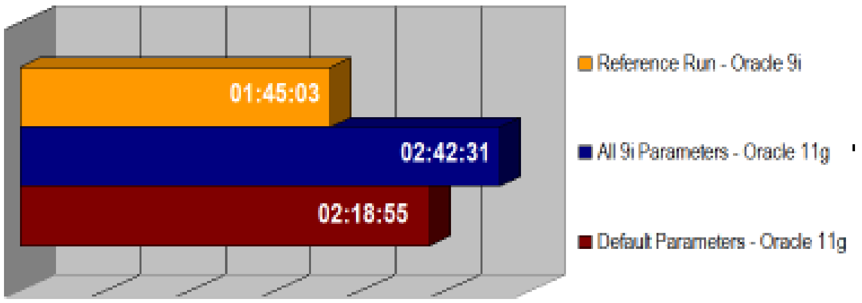
- Remove all Oracle 9i specific parameters from the initialization parameter file, such as:
  - `_ALWAYS_ANTI_JOIN='OFF'`
  - `_ALWAYS_SEMI_JOIN ='OFF'`
  - `_SHARED_POOL_RESERVED_MIN_ALLOC=4000`
  - `_UNNEST_SUBQUERY=FALSE`
  - `DISK_ASYNCH_IO=FALSE`
  - `SORT_AREA_SIZE=4194304`

- Change a few parameters to values required by Oracle Database 11g:
  - `LOG_ARCHIVE_FORMAT='_%r_%t_%s.arc'`
    The '%r' option is required since Oracle Database 10g specifies the resetlogs-ID within the naming format for the archive logs.
  - `TIMED_STATISTICS=TRUE`
    Always set this parameter to `true` to get useful and correct performance information.

- Set new parameters:
  - `CURSOR_SPACE_FOR_TIME=TRUE`
  - `DIAGNOSTIC_DEST='/m161/oracle/admin'`
    Specifies the new Automatic Diagnostic Repository location for all traces and dumps. This replaces parameters such as background_dump_dest and user_dump_dest.
  - `OPTIMITER_INDEX_COST_ADJUST=75`
    This parameter was modified to favor full table scans over index scans – the workload in this case is more CPU bound – IO is not an issue.
  - `OPTIMIZER_CAPTURE_SQL_PLAN_BASELINES=FALSE`
  - `OPTIMIZER_USE_SQL_PLAN_BASELINES=FALSE`
  - `PGA_AGGREGATE_TARGET=1000M`
  - `RECYCLEBIN=OFF`
    The recycle bin has been switched off because it won't get used in the current production environment.
  - `SESSION_CACHED_CURSORS=500`
  - `SHARED_POOL_SIZE=1250M`
  - `_DISABLE_FLASHBACK_ARCHIVER=1`
    The background process FBDA has been switched off because Total Recall is a feature which is not used in the production environment.

Test run 2 – Oracle Database 11g default parameters in the init.ora

For the second test run all unnecessary initialization parameters were removed from the init.ora file, and other parameters were set to the defaults recommended for Oracle Database 11g.

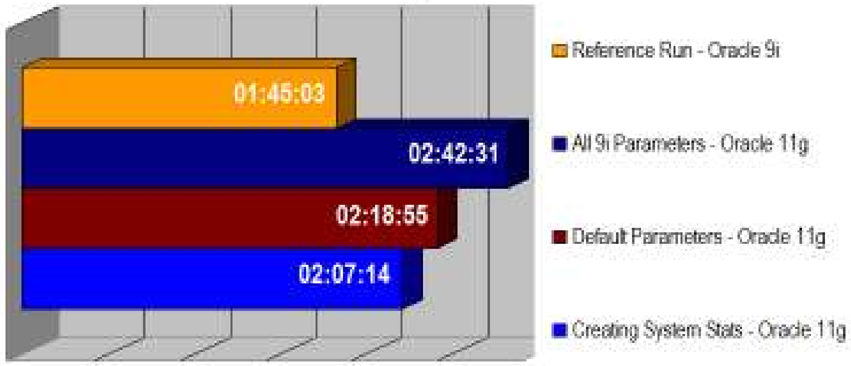


The performance improvement from taking these basic steps was substantial, demonstrating why it is always recommended to remove previous release specific database parameters after upgrading. However, more work was needed to achieve the performance goals for the project.

The next step was to create and use updated system statistics.

Test run 3 – Influence of creating system statistics

Creating system statistics within a workload period gives the database knowledge about the IO performance of the system. These system statistics can also be exported, preserved, and imported into a test system.



System statistics should be created based upon a workload period for at least a few hours.

```
        exec DBMS_STATS.GATHER_SYSTEM_STATS('start');
```

and later:

```
        exec DBMS_STATS.GATHER_SYSTEM_STATS('stop');
```

The results can be monitored by querying `SYS.AUX_STATS$`.

SQL Plan Management

SQL Plan Management (SPM) is a new feature in Oracle Database 11g. It introduces the infrastructure and services needed to support plan maintenance, and to performance verification of new SQL execution plans. To accomplish this, the optimizer maintains a history of plans for individual SQL statements that are executed more than once. Only well-known and proven SQL execution plans will be kept in the SQL Plan Baseline. Manual feeding of plans for a set of SQL statements is also supported.

- `DBMS_SPM.LOAD_PLANS_FROM_SQLSET('mySQLTuningSet1')`
- `DBMS_SPM.LOAD_PLANS_FROM_CURSOR_CACHE(<sql-id>)`

The SQL Plan Baseline and the SQL Plan History are part of the SQL Management Base residing in the SYSAUX tablespace. Its size and its historic extension can be controlled:

- `DBMS_SPM.CONFIGURE('SPACE_BUDGET_PERCENT',20);`
- `DBMS_SPM.CONFIGURE('PLAN_RETENTION_WEEKS',15);`

Two initialization parameters control the SQL Plan Baseline:

- `optimizer_use_sql_plan_baselines = true | false`
- `optimizer_capture_sql_plan_baselines = true | false`

As long as `optimizer_use_sql_plan_baselines` is set to its default value of `true`, only verified or manually loaded plans stored in the SQL Plan Baseline will be used for recurring statements.

Transporting known execution plans

The ability to load SQL execution plans directly into the SQL Plan Baseline is ideally suited to the database upgrade scenario. This ensures that all well proven execution plans can be transported to an upgraded database, thereby reducing the risk of plan regressions to nearly zero.

Upgrading from Oracle 9i to Oracle Database 11g

In Oracle 9i, all statements including their execution plans can be collected and stored into a SQL Tuning Set (STS). However, the SQL Tuning Set cannot be loaded directly into the SQL Plan Baseline in the upgraded database. Therefore another strategy is needed to help reduce the risk of plan regressions. It requires that the upgraded database run for a certain amount of time with a re-parameterized optimizer setting and with the ability to accept execution plans of recurring SQL statements into the SQL Plan Baseline:

- `optimizer_capture_sql_plan_baselines = true`
- `optimzer_features_enable='9.2.0'`

Now the application should be run against the database. It is not important to produce a high load but to ensure that all important statements will be issued more than once.

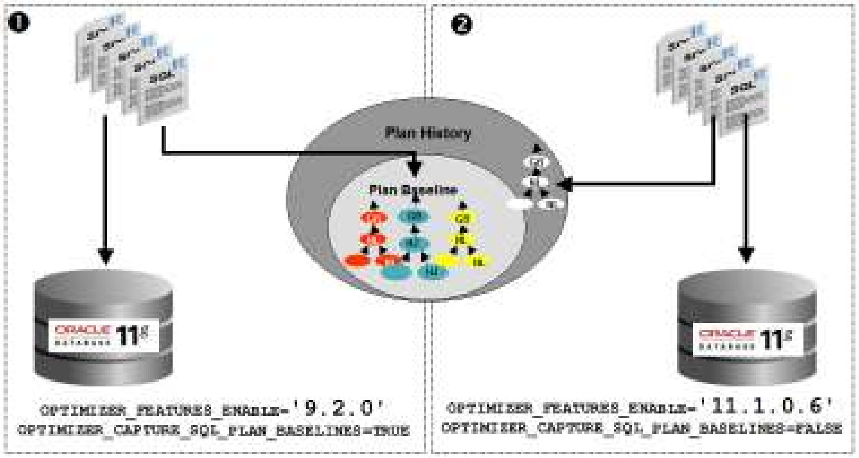

❶
❷
Plan History
Plan Baseline
ORACLE DATABASE 11g
ORACLE DATABASE 11g
OPTIMIZER_FEATURES_ENABLE='9.2.0'
OPTIMIZER_CAPTURE_SQL_PLAN_BASELINES=TRUE
OPTIMIZER_FEATURES_ENABLE='11.1.0.6'
OPTIMIZER_CAPTURE_SQL_PLAN_BASELINES=FALSE

Figure 1: Using SQL Plan Management for a Oracle 9i upgrade

During the second phase the optimizer will now be parameterized to its release-matching level but `optimizer_capture_sql_plan_baselines` will be switched off. New execution plans now considered as equal or better than the stored plans from the SQL Plan Baseline will be recorded to the Plan History – but the optimizer will use the verified plans from the SQL Plan Baseline. At this point the DBA will verify the recorded plans from the Plan History and decide whether to move them into the SQL Plan Baseline or let them purge automatically.

- `optimizer_capture_sql_plan_baselines = false`
- `optimzer_features_enable=’11.1.0.6’`

Upgrading from Oracle Database 10g to Oracle Database 11g

Moving from Oracle Database 10g to Oracle Database 11g is even less risky because all execution plans can be transported within a SQL Tuning Set and loaded directly into the SQL Plan Baseline on the upgraded database. Please note that the use of SQL Tuning Sets requires either a license for the Tuning Pack or the Real Application Testing Pack.

All plans have to be captured on the source database into a STS. This can be done by capturing the cursor cache or selecting statements from the AWR.

```
BEGIN
   DBMS_SQLTUNE.CREATE_SQLSET('STS102WKLD');
   DBMS_SQLTUNE.CAPTURE_CURSOR_CACHE_SQLSET(
       sqlset_name => 'STS102WKLD',
       time_limit =>  120,
       repeat_interval =>  5);
END;
/
```

In the above example the cursor cache will be polled 5 times within a time period of 120 seconds. Subsequently the SQL Tuning Set `STS102WKLD` containing all SQL statements including their execution plans has to be packaged into a staging table:

```
BEGIN
   DBMS_SQLTUNE.CREATE_STGTAB_SQLSET(
         table_name => 'STGTAB102');
   DBMS_SQLTUNE.PACK_STGTAB_SQLSET(
         sqlset_name => 'STS102WKLD',
         staging_table_name => 'STGTAB102');
END;
/
```

and moved via Data Pump export/import or a database link to the target Oracle Database 11g system.

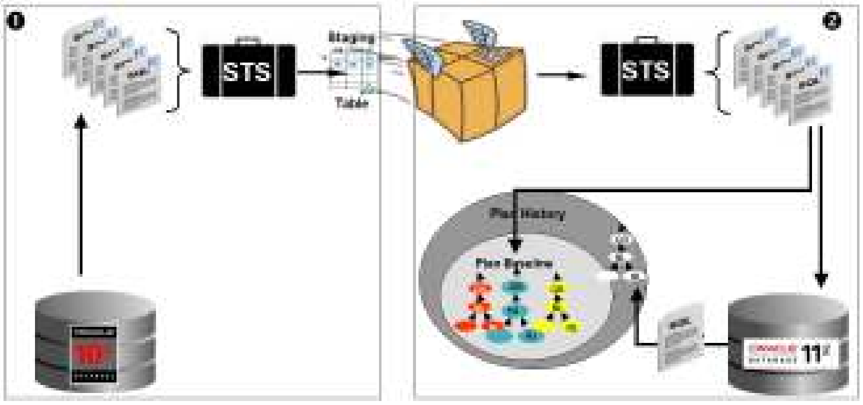


Figure 2: Transporting execution plans from 10g to 11g into the SQL Plan Baseline

There it will be unpacked and the containing execution plans will be written into the SQL Plan Baseline.

```
DECLARE
  my_plans pls_integer;
BEGIN
   DBMS_SQLTUNE.UNPACK_STGTAB_SQLSET(
         sqlset_name => 'STS102WKLD',
         sqlset_owner => '%',
         replace => TRUE,
         staging_table_name  => 'STGTAB102');
  my_plans := DBMS_SPM.LOAD_PLANS_FROM_SQLSET(
         sqlset_name => 'STS102WKLD',
         sqlset_owner => 'SYS',
         basic_filter => 'sql_text like ''%''',
         fixed => 'YES',
         enabled => 'YES',
         commit_rows => 1);
END;
/
```

Preserving a SQL Plan Management

It can be useful to preserve SQL Plan Baselines, to load them into a test system or into a restored database for another Database Replay run, to verify performance in Oracle Database 11g with Oracle 9i-like execution plans. Therefore all plans from the baselines will be loaded into a staging table and this table will be exported:

```
DECLARE
  spm number;
BEGIN
  DBMS_SPM.CREATE_STGTAB_BASELINE(
     table_name=>'SPM_SAVED',
     table_owner=>'SYSTEM');
  spm := DBMS_SPM.PACK_STGTAB_BASELINE(
     table_name=>'SPM_SAVED',
     table_owner=>'SYSTEM');
END;
/

$ exp system/manager file=bls.dmp tables=SPM_SAVED
```

Once the system has been reset the stored baselines will be imported and unpacked again using the `DBMS_SPM.UNPACK_STGTAB_BASELINE` procedure. Now a captured workload can be replayed with the optimizer set to 11.10.6:

```
$ imp system/manager file=bls.dmp full=y

DECLARE
  spm number;
BEGIN
  spm := DBMS_SPM.UNPACK_STGTAB_BASELINE(
     table_name=>'SPM_SAVED',
     table_owner=>'SYSTEM');
END;
/
```

Conclusion

SQL Plan Management is a great new feature to ensure plan stability upon a database upgrade. Especially when moving from Oracle Database 10g to Oracle Database 11g, it will reduce the risk of plan regressions to zero.

Optimization and Load Simulation with Real Application Testing

Oracle Real Application Testing (RAT) pack consists of two components:

- SQL Performance Analyzer (SPA)
  SPA can be used to predict SQL performance deviations before end-users can be impacted. It executes each SQL statement stored in a SQL Tunings Set using the production context, and compares 'before' and 'after' execution plans and run-time statistics. SPA's goal is to identify the set of SQL statements with improved and/or regressed performance.
- Database Replay
  With Database Replay a real workload can be captured on a production system and replayed on a test system with production characteristics including concurrency, synchronization and dependencies. It helps to assess the impact of change on workload throughput. The goal of Database Replay is comprehensive testing of all sub-systems of the database server using a real production workload.

Licensing of Real Application Testing includes SQL Tuning Sets (STS) as well.

The next two sections will show the usage and benefit of SQL Performance Analyzer and Database Replay for a database upgrade project.

SQL Performance Analyzer – Easy detection of changed execution plans

One of the biggest challenges in upgrade projects is to find out how known SQL statements will perform after a release change. SPA is the right tool to find out which SQL statements will improve or regress and which execution plans will change in the new release - before the upgrade. Therefore all SQL statements have to be captured in the current database system.

Test goal

The initial goal when using the SQL Performance Analyzer is to find out which execution plans will change in the new release and establish an easy performance comparison mechanism.

Test setup

A complete nightly batch run will be the basis for the test evaluation. This run defines the test baseline when run against a clone of the Oracle 9.2.0.8 production database. All necessary information will be gathered through SQL trace event 10046. The trace information will be transported and preprocessed according to an object mapping table. Once preprocessed, this information can be reused as often as required. Setup and parameters can be changed, SQL Profiles can be created, the SQL Plan Management can be used – and all changes can be verified immediately.

Capturing SQL statements

First, it is important to switch on statistics timing either by setting it in the initialization parameter file or by issuing:

```
ALTER SYSTEM SET TIMED_STATISTICS=TRUE;
```

Capturing SQL in Oracle Database 9.2.0.8 will be done with SQL tracing – either by DBMS_SUPPORT (see Metalink Note:62294.1) or by using event 10046 to also gather all bind variables:

```
ALTER SYSTEM SET EVENTS ‘10046 TRACE NAME CONTEXT
FOREVER, LEVEL 4’;
```

Now all SQL for all newly connected sessions will be captured in trace files. Tracing can later be switched off with:

```
ALTER SYSTEM SET EVENTS ‘10046 TRACE NAME CONTEXT OFF’;
```

NOTE: It is strongly recommented that you empty the directory specified by `USER_DUMP_DEST` before starting SQL tracing. It is likely that a huge amount of trace information will be gathered as part of this process. In this case approximately 60-80 GB of trace files were written to disk. Results will vary depending on the database and workload.

In addition to the SQL traces, a mapping table must be created and transported via export/import to the evaluation database:

```
  CREATE TABLE MAPPING_TABLE AS
         SELECT OBJECT_ID ID, OWNER,
                SUBSTR(OBJECT_NAME, 1, 30) NAME
           FROM DBA_OBJECTS
          WHERE OBJECT_TYPE NOT IN
         ('CONSUMER GROUP', 'EVALUATION
           CONTEXT', 'FUNCTION', 'INDEXTYPE',
          'JAVA CLASS', 'JAVA DATA', 'JAVA
           RESOURCE', 'LIBRARY', 'LOB',
          'OPERATOR', 'PACKAGE', 'PACKAGE
           BODY', 'PROCEDURE', 'QUEUE',
          'RESOURCE PLAN', 'SYNONYM',
          'TRIGGER', 'TYPE', 'TYPE BODY')
UNION ALL
  SELECT USER_ID ID, USERNAME OWNER, NULL NAME
    FROM DBA_USERS;
```

Loading statements into a SQL Tuning Set

Once imported into the 11g database, the mapping table is used to make the 9i trace information fit into an 11g STS.

```
DECLARE
   syscur DBMS_SQLTUNE.SQLSET_CURSOR;
BEGIN
   DBMS_SQLTUNE.CREATE_SQLSET('SPA_STS');
   OPEN syscur FOR
     SELECT VALUE(p) FROM
     TABLE(DBMS_SQLTUNE.SELECT_SQL_TRACE(
      directory => 'SPA_DIR',
      file_name => '%ora%',
      mapping_table_name => 'MAPPING_TABLE',
      select_mode=>dbms_sqltune.single_execution)) p;
      DBMS_SQLTUNE.LOAD_SQLSET(
         sqlset_name => 'SPA_STS',
         populate_cursor => syscur,
         commit_rows => 5);
   CLOSE syscur;
END;
/
```

If many trace files were generated then it will make sense to split them up equally into separate sub-directories and load them in parallel. As a general rule, an ideal number of directories across which to distribute the traces would be one fewer than CPUs in the server. In this case, parallel loading reduced the time to load all 2760 individual statements into the STS from 9.5 hours down to 1.5 hours. If this strategy has been used to populate STSs in parallel then a merge operation is required at the end of the load:

```
DECLARE
   cur DBMS_SQLTUNE.SQLSET_CURSOR;
BEGIN
   DBMS_SQLTUNE.CREATE_SQLSET('SPA_MAIN_STS');
   OPEN cur FOR
     SELECT VALUE(p)
      FROM TABLE(
        DBMS_SQLTUNE.SELECT_SQLSET('SQLSET1')) p;
   DBMS_SQLTUNE.LOAD_SQLSET(
      sqlset_name => 'SPA_MAIN_STS',
      populate_cursor => cur,
      load_option => 'MERGE',
      update_option => 'ACCUMULATE');
END;
/
```

This action has to be done for `SQLSET1` … `SQLSETn`.

The following query can be used to monitor how many unique SQL statements have already been loaded into the STS:

```
SELECT STATAMENT_COUNT
   FROM DBA_SQLSET
   WHERE NAME=’SPA_STS’;
```

In addition, the view `DBA_SQLSET_STATEMENTS` can be queried to monitor details on specific SQLs loaded into an STS.

Preservation of the SQL Tuning Set

Populating the SQL Tuning Set has to be just done once. Hence this STS should be saved in an exportable staging table. If the test system can be setup to a certain point in time then this STS can be reused as often as required.

```
-- Create the staging table
-- Please note that the name of the mapping table
-- currently cannot exceed 19 characters

BEGIN
  DBMS_SQLTUNE.CREATE_STGTAB_SQLSET(
     table_name => 'SPA_STS_STGTAB' );
END;
/

-- Load the SQL Set into the mapping table

BEGIN
  DBMS_SQLTUNE.PACK_STGTAB_SQLSET(
     sqlset_name => 'SPA_STS',
     staging_table_name => 'SPA_STS_STGTAB');
END;
/
```

The staging table `SPA_STS_STGTAB` can now be exported, stored in a save location and re-imported at any time to the test system. Once it has been imported it can be unpacked:

```
BEGIN
  DBMS_SQLTUNE.UNPACK_STGTAB_SQLSET(
     sqlset_name => '%',
     replace => TRUE,
     staging_table_name => 'SPA_STS_STGTAB');
END;
/
```

Creating and parameterizing an SPA analysis task

To analyze the SQL statements stored in the STS with SPA, an analysis task has to be created first:

```
DECLARE
  tname VARCHAR2(100);
BEGIN
  tname := DBMS_SQLPA.CREATE_ANALYSIS_TASK(
             sqlset_name=>'SPA_STS',
             task_name=>'SPA_TASK_9i_11g',
             description=>'Move on from 9i to 11g');
END;
/
```

The analysis task should be parameterized to allow for control over the analysis:

```
BEGIN
   DBMS_SQLPA.SET_ANALYSIS_TASK_PARAMETER(
     task_name => 'SPA_TASK_9i_11g',
     parameter => 'workload_impact_threshold',
     value     => 0);
   DBMS_SQLPA.SET_ANALYSIS_TASK_PARAMETER(
     task_name => 'SPA_TASK_9i_11g',
     parameter => 'sql_impact_threshold',
     value     => 10);
end;
/
```

`WORKLOAD_IMPACT_THRESHOLD` defines the minimum threshold, as a percentage, for a SQL to impact the workload before SPA will label it improved or regressed. This parameter can be set to zero, since the 9i STS we are using for SPA does not contain execution frequency information.

`SQL_IMPACT_THRESHOLD` sets the minimum threshold as a percentage for a SQL's per-execution performance to be impacted before SPA will label it improved or regressed (in addition to the workload impact threshold). The default value for this parameter is 1. In this case it has been set to 10 to avoid flooding the report with tiny, irrelevant regressions. Setting these two values will cause SPA to justify improvement or regression findings for any SQL whose individual performance increases or decreases by at least ten percent, ignoring the rest of the workload.

Execution of the SPA trail with the Oracle 9i performance data

Processing the SQL information in the STS from the 9i database means just copying statistics from the STS into the SPA task. It does not execute any SQL statements and therefore it is a lightweight operation completing very quickly.

```
BEGIN
   DBMS_SQLPA.EXECUTE_ANALYSIS_TASK(
     task_name => 'SPA_TASK_9i_11g',
     execution_name => 'EXEC_SPA_TASK_9i',
     execution_type => 'CONVERT SQLSET',
     execution_desc => 'Convert 9i Workload');
END;
/
```

Execution of the SPA trail with the Oracle Database 11g performance data

Creating a SPA trail with the Oracle Database 11g performance data will now execute each statement in the freshly prepared test database.

```
BEGIN
   DBMS_SQLPA.EXECUTE_ANALYSIS_TASK(
     task_name => 'SPA_TASK_9i_11g',
     execution_name => 'EXEC_SPA_TASK_11g',
     execution_type => 'TEST EXECUTE',
     execution_desc => 'Test 9i Workload in 11g');
END;
/
```

This step will be much more resource intensive because it actually executes every statement. It can be monitored by running the following query:

```
select sofar, totalwork from v$advisor_progress
where task_id = <tid>;
```

Reporting / Comparison

Once the tasks have been executed the results can be compared on several metrics: `PARSE_TIME, ELAPSED_TIME, CPU_TIME, USER_IO_TIME, BUFFER_GETS, DISK_READS, DIRECT_WRITES` and `OPTIMIZER_COST`.

```
BEGIN
   DBMS_SQLPA.EXECUTE_ANALYSIS_TASK(
     task_name => 'SPA_TASK_9i_11g',
     execution_name => 'Compare 9i 11g CPU_TIME',
     execution_type => 'COMPARE PERFORMANCE',
     execution_params =>
       DBMS_ADVISOR.ARGLIST(
               'comparison_metric',
               'cpu_time',
               'execution_name1','EXEC_SPA_TASK',
               'execution_name2','TEST 9i 11g CPU'),
     execution_desc => 'Compare 9i to 11g CPU_TIME');
END;
/
```

Display a report on this comparison:

```
SELECT
  xmltype(DBMS_SQLPA.REPORT_ANALYSIS_TASK
 ('SPA_TASK_9i_11g', 'html', 'typical', 'summary',
   null, 350, 'Compare 9i to 11g CPU_TIME'))
 .getclobval(2,2)
FROM dual;
```

This can now be done for different metrics. The `CPU_TIME` and `BUFFER_GETS` metrics are the most meaningful when testing with Oracle 9i data.

Results of the SQL Performance Analyzer

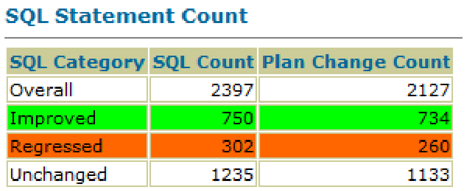
The SQL Performance Analyzer was used throughout the whole performance evaluation phase of this project. Once the statements were loaded from the source database they could be reiterated as often and as many times as required. This made SPA an extremely helpful tool when tuning a database because it took just a re-execute of the SPA task to verify the results.

An example shows the optimization of optimizer parameters with SPA. The best settings had been found by just re-executing the SPA task again and again with different initialization parameters set. Please be aware that this does require knowledge about the application and its SQL statements.

In this case especially, the high-load regressed execution plans were checked in detail in the SPA reports. The evaluation SQL Tuning Set consisted of 2397 unique SQL statements. The optimization was targeted for `CPU_TIME`, because `BUFFER_GETS` are relatively cheap for this specific workload.

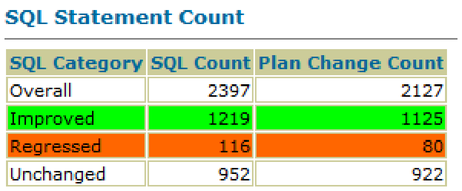
The operation of recommended 10gR2 parameters such as `_optimizer_cost_based_transformation=off` and `_new_initial_join_order=false` were verified as well. But, the results showed clearly that those are not necessary in Oracle Database 11g, and that furthermore the use of these parameters will lead to worse execution results.

Reports on `CPU_TIME`	Report on `BUFFER_GETS`
`optimizer_features_enable=9.2.0`	



SQL Statement Count

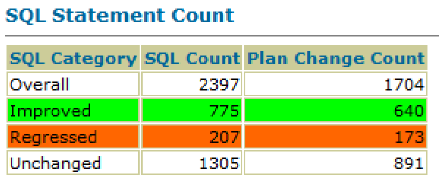
SQL Category	SQL Count	Plan Change Count
Overall	2397	2127
Improved	750	734
Regressed	302	260
Unchanged	1235	1133



SQL Statement Count

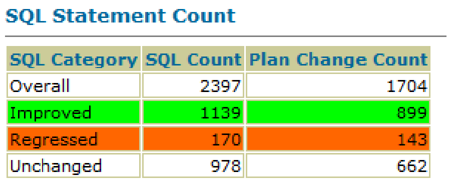
SQL Category	SQL Count	Plan Change Count
Overall	2397	2127
Improved	1219	1125
Regressed	116	80
Unchanged	952	922

`optimizer_features_enable=11.1.0.6`



SQL Statement Count

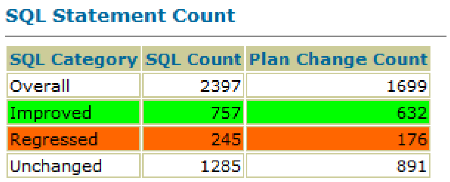
SQL Category	SQL Count	Plan Change Count
Overall	2397	1704
Improved	775	640
Regressed	207	173
Unchanged	1305	891



SQL Statement Count

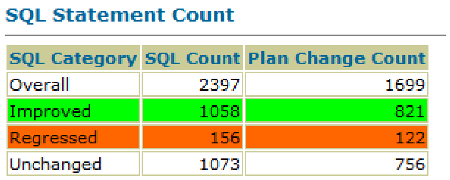
SQL Category	SQL Count	Plan Change Count
Overall	2397	1704
Improved	1139	899
Regressed	170	143
Unchanged	978	662

`_optimizer_cost_based_transformation=off`



SQL Statement Count

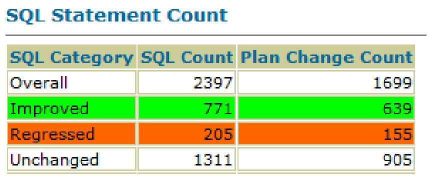
SQL Category	SQL Count	Plan Change Count
Overall	2397	1699
Improved	757	632
Regressed	245	176
Unchanged	1285	891



SQL Statement Count

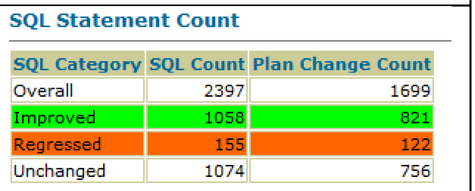
SQL Category	SQL Count	Plan Change Count
Overall	2397	1699
Improved	1058	821
Regressed	156	122
Unchanged	1073	756

`_new_initial_join_order=false`



SQL Statement Count

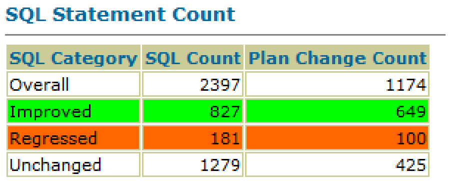
SQL Category	SQL Count	Plan Change Count
Overall	2397	1699
Improved	771	639
Regressed	205	155
Unchanged	1311	905



SQL Statement Count

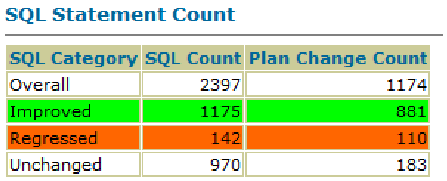
SQL Category	SQL Count	Plan Change Count
Overall	2397	1699
Improved	1058	821
Regressed	155	122
Unchanged	1074	756

`optimizer_index_cost_adj=75`



SQL Statement Count

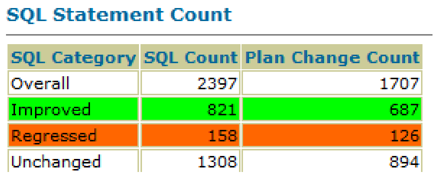
SQL Category	SQL Count	Plan Change Count
Overall	2397	1174
Improved	827	649
Regressed	181	100
Unchanged	1279	425



SQL Statement Count

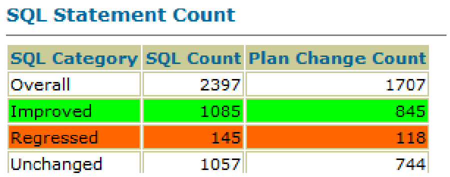
SQL Category	SQL Count	Plan Change Count
Overall	2397	1174
Improved	1175	881
Regressed	142	110
Unchanged	970	183

`optimizer_mode=first_rows_10`



SQL Statement Count

SQL Category	SQL Count	Plan Change Count
Overall	2397	1707
Improved	821	687
Regressed	158	126
Unchanged	1308	894

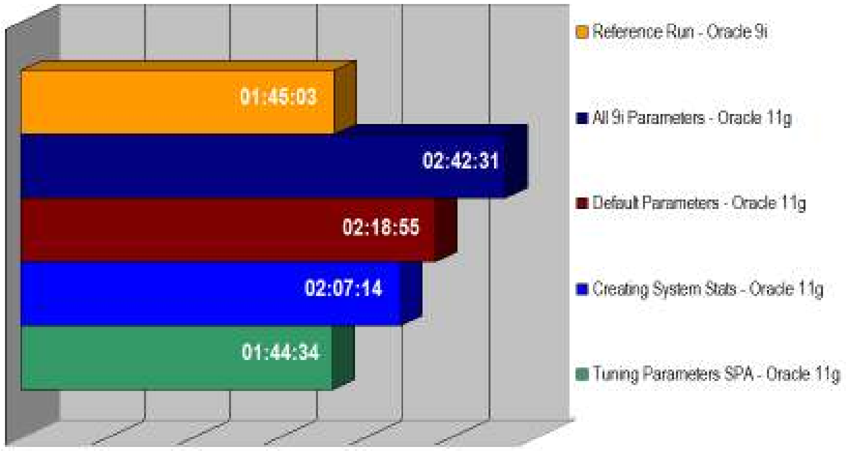


SQL Statement Count

SQL Category	SQL Count	Plan Change Count
Overall	2397	1707
Improved	1085	845
Regressed	145	118
Unchanged	1057	744

Test run 4 – Optimization of the init.ora with SPA

SPA is the ideal tool to optimize the init.ora. The load has just to be done once and the SPA task can be reiterated again and again.



With SPA the following parameters had been found to optimize the workload processing:

```
       optimizer_index_cost_adj=75
       optimizer_mode=first_rows_10
```

This lead to excellent execution timings, beating the Oracle 9i workload completion for the first time during this proof of concept.

Conclusion SQL Performance Analyzer

The SQL Performance Analyzer is a great tool to detect plan regressions before upgrading and evaluating different remedies. It does not require application of a patch to the source systems, which means that no additional downtime on the production system is needed to use SPA, and it does not change actual data. Once all statements have been collected into a SQL Tuning Set and transported to the target system it can be evaluated again and again - effortless.

A minor disadvantage of SPA currently does not take into account how often a statement was executed originally. Therefore it does not allow benchmarking of a single statement in comparison to the overall workload.

Database Replay

Database Capture and Replay was introduced with Oracle Database 11g. This option allows capture of a full workload running against a database and – after preprocessing it – replay of that workload against an Oracle Database 11g database on any platform. The capture can be run in an Oracle 9.2.0.8, Oracle Database 10.2.0.2, 10.2.0.3, 10.2.0.4 or Oracle Database 11.1 environment, but pre-10.2.0.4 databases require a patch to enable the capture functionality. Metalink-Note: 560977.1 points directly to the required patch numbers. In an Oracle Database 10.2.0.4 the script

```
@$ORACLE_HOME/rdbms/admin/wrrenbl.sql
```

has to be run to enable workload capture capabilities. This will automatically set the `PRE_11G_ENABLE_CAPTURE` initialization parameter to `TRUE`.
Capture functionality can always been disabled by changing the initialization parameter `PRE_11G_ENABLE_CAPTURE` to `FALSE`.

Test goal

The initial goal when using Database Replay is to find out how the new system will perform with a complete nightly batch load. Therefore the whole workload will be captured when run against the Oracle 9.2.0.8 database. Once the captured workload has been preprocessed it can be used for evaluation in the Oracle Database 11g environment as often as required. Due to the fact that the captured workload will actually change data, it is important to restore the upgraded database to a status that will allow it to process the complete workload again and again.

Test setup

A complete nightly batch run will be the basis for the test evaluation. This run defines the test baseline when run against a clone of the Oracle 9.2.0.8 production database. All necessary information will be captured in binary capture files. These files have to be preprocessed once. They can be reused and they are platform independent – so the workload information is useful not only for a database evaluation, but also to evaluate OS features, a storage subsystem, or other changes to the operating environment.

To replay the captured files, workload replay clients have to be started on the OS level. The database has to be reset to a specific point in time for a successful processing and completion of the recorded workload. Several options can be used to achieve this:

- Backup the upgrade database with RMAN and restore it for each replay.
    - Advantage:
      Is well tested and does not add any overhead to the test.
    - Disadvantage:
      Takes longer to complete (approximately 1 hour).

- Enable Flashback Database – this requires the Archivelog mode to be switched on – and create a guaranteed restore point. Then the workload can be replayed and afterwards the database can be flashed back to the guaranteed restore point.
  - Advantage:
    Easy to setup and extremely fast.
  - Disadvantage:
    Adds additional overhead to the test load because of the creation of the flashback logs. Also requires additional disk space (estimate: as much as archivelogs will get created).
- Snapshot Standby – a physical standby database will be converted temporarily into a Snapshot Standby database. After the workload has been replayed it will be converted back into a physical standby. Data Guard will then synchronize all changes from the production database to the standby.
  - Advantage:
    Easy to setup and extremely fast.
  - Disadvantage:
    Requires a physical standby database usable for testing purposes. For this project there was no physical standby database available and the workload will only be completely replayable if the database will be rolled back to the point in time of the capture run.

Workload Capture

For a successful workload capture an empty capture directory must be defined and the database should be started in RESTRICTED mode before the capture begins. This will enable the capture process to catch all user calls. As soon as the capture starts, the database will be switched automatically to UNRESTRICTED mode allowing users to connect.

```
CREATE OR REPLACE DIRECTORY RAT_DIR as '/oracle/rat';

STARTUP RESTRICTED;
```

Filters can be added to the capture for either exclusion or inclusion of specific users:

```
BEGIN
   DBMS_WORKLOAD_CAPTURE.ADD_FILTER(
      fname => 'FILTER_FOR_USER_SYS',
      fattribute => 'USER',
      fvalue => 'SYS');
END;
/
```

The workload capture can be started and monitored:

```
BEGIN
   DBMS_WORKLOAD_CAPTURE.START_CAPTURE(
      name => 'RAT_9208',
      dir => 'RAT_DIR');
END;
/
SELECT * FROM DBA_WORKLOAD_CAPTURES;
```

If it has not been limited to run for a specific duration then it can be stopped with:

```
EXECUTE DBMS_WORKLOAD_CAPTURE.FINISH_CAPTURE;
```

If the capture has done on an Oracle Database 10g or Oracle Database 11g release database afterwards the AWR should be exported for later comparisons:

```
BEGIN
  DBMS_WORKLOAD_CAPTURE.EXPORT_AWR(
     capture_id => 5);
END;
/
```

Preprocessing

The captured workload then has to be preprocessed before it can be transported and replayed on any operation system:

```
EXECUTE DBMS_WORKLOAD_REPLAY.PROCESS_CAPTURE(
           capture_dir => 'RAT_DIR');
```

Workload Replay

The workload replay can only be done in an Oracle Database 11g system. To start a successful workload replay, the target database must be reset to the same point in time when the capture was started. Otherwise it is possible that not all calls will succeed. The exported AWR snapshot should be imported for later comparison:

```
DECLARE db_id number;
BEGIN
 db_id := DBMS_WORKLOAD_CAPTURE.IMPORT_AWR(
            capture_id => 5,
            staging_schema => 'TEST');
END;
/
```

The next step should be creating valid and current statistics and an AWR snapshot:

```
exec dbms_stats.gather_dictionary_stats;
exec dbms_stats.gather_fixed_objects_stats;
exec dbms_stats.gather_system_stats('start');
exec dbms_stats.gather_system_stats('stop');
```

New object statistics should be created:

```
exec dbms_auto_task_immediate.gather_optimizer_stats;
```

The job can be monitored with:

```
SELECT job_name,state
FROM dba_scheduler_jobs
WHERE program_name='GATHER_STATS_PROG';
```

Once current statistics have been created the replay can be initialized:

```
BEGIN
   DBMS_WORKLOAD_REPLAY.INITIALIZE_REPLAY(
      replay_name => 'TEST_REPLAY1',
      replay_dir => 'RAT_DIR');
END;
/
```

The Workload Replay Clients (WRC) have to be calibrated at the operating system level. This will estimate the number of WRCs to start:

```
$ wrc mode=calibrate replaydir=/oracle/rat

Recommendation:
Consider using at least 1 clients divided among 1 CPU(s).

Workload Characteristics:
- max concurrency: 23 sessions
- total number of sessions: 343

Assumptions:
- 1 client process per 50 concurrent sessions
- 4 client process per CPU
- think time scale = 100
- connect time scale = 100
- synchronization = TRUE
```

The replay can be parameterized in several modes:

- SYNCHRONIZATION = TRUE | FALSE
  The COMMIT order in the captured workload will be honored, meaning that actions will be executed only after dependent COMMITs have completed.

- CONNECT_TIME_SCALE = 0 [%] - 100 [%]
  Optional parameter for specifying the elapsed time from when the workload capture has been started to when the session connects. This is interpreted as a percentage value and used for increasing or decreasing the number of users.

- THINK_TIME_SCALE = 0 [%] - 100 [%]
  Optional parameter for regulating the elapsed time between two user calls. Setting it to 0 would lead to the fastest possible replay.

- THINK_TIME_AUTO_CORRECT = TRUE | FALSE
  Optional parameter correcting THINK_TIME_SCALE between calls when user calls take longer to complete during replay than during capture.

```
BEGIN
   DBMS_WORKLOAD_REPLAY.PREPARE_REPLAY(
      SYNCHRONIZATION => TRUE);
```

```
       END;
       /
```

The database is now waiting for the workload replay clients to connect against it. Workload Replay clients have to be started at the command line prompt – the previous calibrate will suggest a number of wrc-clients to start:

```
       $ wrc userid=system password=mypwd
             replaydir=/oracle/rat
```

Then the actual workload replay should be started:

```
       BEGIN
          DBMS_WORKLOAD_REPLAY.START_REPLAY;
       END;
       /
```

It can be monitored either in Enterprise Manager Database Control or with the view DBA_WORKLOAD_REPLAYS:

```
       SELECT id, name, status,
              start_time, end_time, num_clients
         FROM dba_workload_replays;
```

A workload replay can be canceled as follows:

```
       BEGIN
         DBMS_WORKLOAD_REPLAY.CANCEL_REPLAY ();
       END;
       /
```

Reporting

A script will display the replay overall statistics:

```
set long 10000000
set lines 128
column output format a120
set trimspool on
spool report_replay_text.txt
set heading off
set pages 0
select dbms_workload_replay.report(&&1,'TEXT') output from
dual;
spool off

set heading off
set pages 0
set long 1000000000
spool /tmp/replay_&&1.html
select dbms_workload_replay.report(&&1,'HTML') from dual;
spool off
```

Conclusion

Database Replay is very helpful in verifying the application-level functional completeness for an upgrade of either a patch or full release. It gives many useful hints about how the target system will perform by re-executing the captured workload. And it can be used to evaluate not only the database, but also the system or storage performance compared to the current setup.

In this case, Database Replay confirmed that no application changes were needed as a result of the upgrade to Oracle Database 11g. Application changes had caused a lot of effort in the past, such as during the migration from Oracle 8i to Oracle 9i.

The Database Replay results gave the customer great confidence in moving forward to the new release.

PL/SQL Native Compilation

Oracle PL/SQL is normally an interpreted language, but Native Compilation has been available since Oracle 9i. Starting with Oracle Database 11g no external compiler is necessary, which makes Native Compilation more efficient and easier to use. Simply set the initialization parameters:

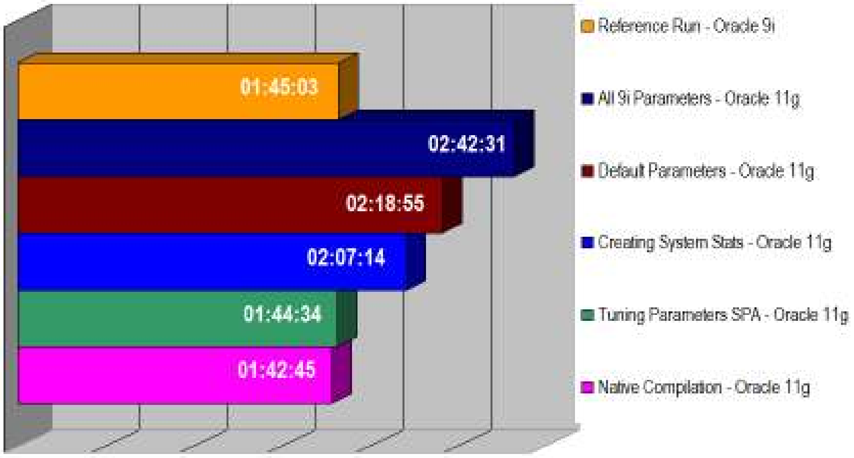
```
plsql_code_type=native
plsql_optimization_level=3
```

to the appropriate values and PL/SQL code can then be recompiled schema by schema with the following procedure:

```
exec DBMS_UTILITY.COMPILE_SCHEMA('<username>');
```

Test run 5 – PL/SQL Native Compilation

All schemas in the database containing PL/SQL procedures and packages were recompiled. After doing this the workload completed nearly 2 minutes faster than without Native Compilation.



Conclusion

Even though most PL/SQL code in this specific customer scenario just issues a lot of SQL statements, the Native Compilation of PL/SQL code lead to a slightly better overall completion timing of the workload. The improvement can be even more dramatic if procedures, functions, and packages perform computations and more intensive PL/SQL processing.

SQL Tuning Advisor and Automatic SQL Tuning

Available starting with Oracle Database 10g, the database package Tuning Pack interacts closely with the Diagnostic Pack and is capable of creating SQL Profiles to make SQL statements perform faster without changing either the SQL or the application code. Each unique SQL statement gets a hash value assigned to it upon normalization, and a SQL Profile can be bound to the statement on this `sql_id`. To create a SQL Profile the optimizer must be set into Comprehensive Mode. This will allow it to calculate all possibilities in a specific amount of time to execute this particular statement faster during regular database operation. Such SQL Profiles consist of additional information and notes for the optimizer – they are not static like Stored Outlines – and they are persistent. As of Oracle Database 10gR2 those SQL Profiles can be transported from one database to another within a Staging Table.

Beginning with Oracle Database 11g, the process of choosing high-load SQL statements manually and scheduling the SQL Tuning Advisor to analyze these statements can be automated by Automatic SQL Tuning. Every time (by default every 60 minutes) that an AWR snapshot is gathered and stored in the Automatic Workload Repository, the Automatic SQL Tuning process will start calculating SQL Profiles for the high-load SQL statements marked in this AWR snapshot. It can implement all SQL Profiles automatically so that they will apply immediately if one of the profiled statements will be executed again.

Automatic SQL Tuning can be enabled and disabled with:

```
BEGIN
  DBMS_AUTO_TASK_ADMIN.ENABLE(
    client_name => 'sql tuning advisor',
    operation => NULL,
    window_name => NULL);
END;
/

BEGIN
  DBMS_AUTO_TASK_ADMIN.DISABLE(
    client_name => 'sql tuning advisor',
    operation => NULL,
    window_name => NULL);
END;
/
```

Automatic SQL Tuning can be parameterized to accept SQL Profiles automatically without any manual intervention necessary:

```
BEGIN
  DBMS_SQLTUNE.SET_TUNING_TASK_PARAMETER(
    task_name => 'SYS_AUTO_SQL_TUNING_TASK',
    parameter => 'ACCEPT_SQL_PROFILES',
    value => 'TRUE');
END;
/
```

The tuning results will be shown using the following report:

```
variable my_rept CLOB;

BEGIN
  :my_rept :=DBMS_SQLTUNE.REPORT_AUTO_TUNING_TASK(
    begin_exec => NULL,
    end_exec => NULL,
    type => 'TEXT',
    level => 'TYPICAL',
    section => 'ALL',
    object_id => NULL,
    result_limit => NULL);
END;
/

print :my_rept
```

The report would look like this, in which the Automatic SQL Tuning Advisor has found seven tuning candidates across two AWR snapshots:

```
-------------------------------------------------------------------
GENERAL INFORMATION SECTION
-------------------------------------------------------------------
Tuning Task Name                        : SYS_AUTO_SQL_TUNING_TASK
Tuning Task Owner                       : SYS
Workload Type                           : Automatic High-Load SQL W
Execution Count                         : 2
Current Execution                       : EXEC_20
Execution Type                          : TUNE SQL
Scope                                   : COMPREHENSIVE
Global Time Limit(seconds)              : 3600
Per-SQL Time Limit(seconds)             : 1200

MY_REPT
-------------------------------------------------------------------
Completion Status                       : COMPLETED
Started at                              : 08/27/2008 22:00:02
Completed at                            : 08/27/2008 22:05:19
Number of Candidate SQLs                : 7
Cumulative Elapsed Time of SQL (s)      : 1052
-------------------------------------------------------------------
```

The report contains four findings: one SQL Profile with a benefit of 90%, one index with a benefit of 98% and restructuring hints for two statements.

```
-----------------------------------------------------------------
              Global SQL Tuning Result Statistics
-----------------------------------------------------------------
MY_REPT
-----------------------------------------------------------------
Number of SQLs Analyzed                      : 7
Number of SQLs in the Report                 : 4
Number of SQLs with Findings                 : 4
Number of SQLs with SQL profiles recommended : 1
Number of SQLs with Index Findings           : 1
Number of SQLs with SQL Restructure Findings : 2
---------------------------------------------------------------------------
SQLs with Findings Ordered by Maximum (Profile/Index) Benefit, Object ID
---------------------------------------------------------------------------
objID  SQL ID        statistics profile(benefit) index(benefit) restructure
------ ------------- ---------- ---------------- -------------- -----------
    13 bgjxn875surdu                                     98.10%
    12 6p8uzacmr06kr                      90.37%
     9 a9cq9s90q2k5w                                                      1
    14 9b87y60nh1jcg                                                      3

---------------------------------------------------------------------------
Tables with New Potential Indices (ordered by schema, number of times, tab)
---------------------------------------------------------------------------
    Schema Name         Table Name                  Index Name     Nb Time
----------------------- --------------------------- -------------- --------
              USER_ABC M123_AUSZEICH_DAT           IDX$$_00010006        1
```

Following these global tuning results the report will reveal all tuning recommendations in detail.

For later reuse, and to have the ability to re-import SQL Profiles upon a restore operation of the database during the test cycles, the SQL Profiles will be written into a staging table and exported. They will be imported and unpacked again after the test system restore has finished. This procedure can also be used to transport SQL Profiles from a test system to a production database.

At first the staging table will be created. Note that it cannot be located in the SYS user schema:

```
begin
 DBMS_SQLTUNE.CREATE_STGTAB_SQLPROF (
  table_name=>'STAB_PROFILES',
  schema_name=>'SYSTEM');
end;
/
```

Then all SQL Profiles get written into the staging table:

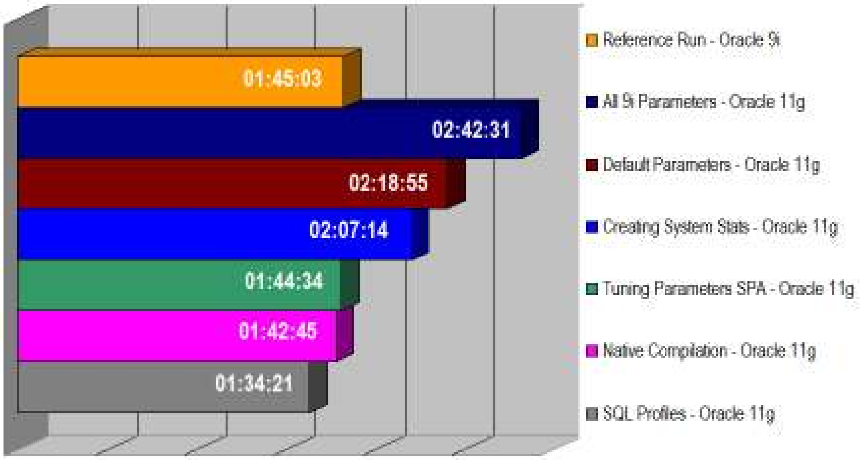
```
BEGIN
 DBMS_SQLTUNE.PACK_STGTAB_SQLPROF (
   profile_name => '%',
   profile_category => 'DEFAULT',
   staging_table_name => 'STAB_PROFILES',
   staging_schema_owner => 'SYSTEM');
END;
/
```

Now the Staging Table containing all the SQL Profiles can be exported with Data Pump Export and imported later when the test system has been restored again. It can also be moved via an existing database link to another database. Once unpack the SQL Profiles will applied to the target system:

```
BEGIN
 DBMS_SQLTUNE.UNPACK_STGTAB_SQLPROF (
   profile_name => '%',
   profile_category => 'DEFAULT',
   staging_table_name => 'STAB_PROFILES',
   staging_schema_owner => 'SYSTEM',
   replace => TRUE);
end;
/
```

Test run 6 – SQL Tuning Advisor and Automatic SQL Tuning

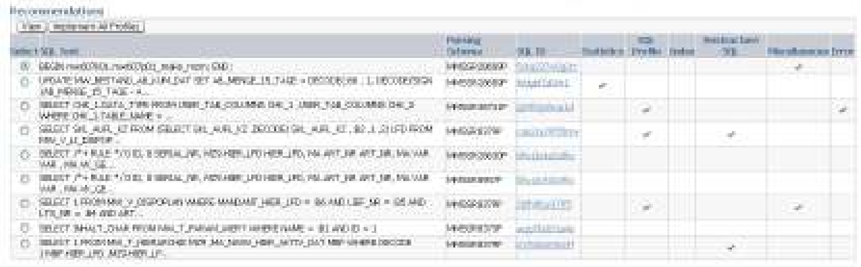
For test run 6 all five SQL Profiles gathered during past workloads runs were loaded into the database.



The above graph shows clearly the SQL Profiles improved performance of the application – *without the need to rewrite any SQL statement or change application code.* The Oracle 9i reference workload timings were outperformed by more than 10 minutes, or nearly 10%.

Conclusion

Statement tuning with the SQL Tuning Advisor is extremely efficient and very easy to use. During workload processing the database was monitored in Enterprise Manager Database Control, and high-load statements were selected manually and delivered to the SQL Tuning Advisor for optimization.



Select	SQL Text	Parsing Schema	SQL ID	Statistics	SQL Profile	Index	Restructure SQL	Miscellaneous	Error
●	BEGIN [illegible]; END;	[illegible]	[illegible]					✓	
○	UPDATE MW_BESTAND_AB_KUM_DAT SET AB_MENGE_15_TAGE = DECODE(:B8 , 1, DECODE(SIGN (AB_MENGE_15_TAGE - :4...	[illegible]	[illegible]	✓					
○	SELECT CHK_1.DATA_TYPE FROM USER_TAB_COLUMNS CHK_1 , USER_TAB_COLUMNS CHK_2 WHERE CHK_1.TABLE_NAME = ...	[illegible]	[illegible]		✓				✓
○	SELECT SHL_AUR_KZ FROM (SELECT SHL_AUR_KZ, DECODE(SHL_AUR_KZ , :B2 ,1 ,2) LFD FROM MW_V_LI_[illegible] ...	[illegible]	[illegible]		✓		✓		
○	SELECT /*+ RULE */ [illegible] ...	[illegible]	[illegible]						
○	SELECT /*+ RULE */ [illegible] ...	[illegible]	[illegible]						
○	SELECT 1 FROM [illegible] WHERE MANDANT_[illegible] AND [illegible] ...	[illegible]	[illegible]		✓			✓	
○	SELECT [illegible] FROM [illegible] WHERE NAME = :B1 AND ID = 1	[illegible]	[illegible]						
○	SELECT 1 FROM [illegible] WHERE DECODE [illegible] ...	[illegible]	[illegible]				✓		

Once verified, the recommended SQL Profiles were preserved and reloaded for further test runs. SQL Profiles do not require the application code to be changed, so they are the ideal method of tuning suboptimal SQL statements without changing anything. The SQL Profiles created by the SQL Tuning Advisor led to an average 80% better SQL performance for those statements identified as needing to be tuned.

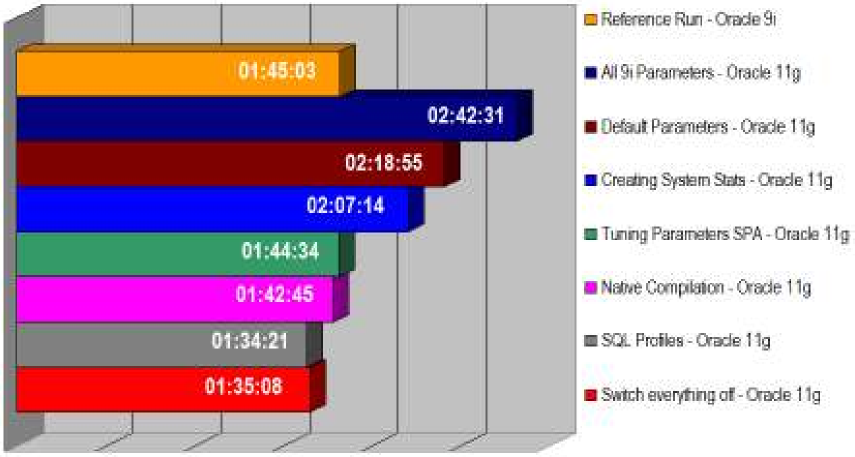
Final Test run – Switch off all automation and statistics features

Although the results of the Oracle 9i workload completion timings had already been improved in Oracle Database 11g, one final test run had to be done to have a reliable basis for comparison: the Oracle 9i databases needed to be run without timed_statistics or STATSPACK enabled. This is because with Oracle Database 11g the database has a lot of tuning mechanisms like AWR snapshots, advisor jobs and Automatic SQL Tuning in Oracle Database 11g switched on by default. For the final test run all those features had been disabled.

```
statistics_level=basic
_ash_enable=false
timed_statistics=false
```

SQL Profiles were deleted as well.

The results should not be misinterpreted. This test has been done only to compare the initial results with the Oracle Database 11g timing on an identical basis. For production purposes the full diagnostic and tuning functionality should and will be switched on.



It is remarkable that enabling all diagnostic and tuning features and using SQL Profiles had a much better effect on the overall performance, than running the workload with all tuning mechanisms switched off. The workload timings from the Final Test Run were not as good as the results from Test Run 6. This proves the claim wrong that disabling all Oracle performance features will speed up the database performance - and it impressively demonstrates the benefit a database application can gain from those features.

FINAL CONCLUSION

When this Proof-of-Concept was started, the main targets were to find the best way to upgrade 400 Oracle databases from Oracle 9.2.0.8 to Oracle 11.1.0.7 and to ensure that the original Oracle 9i completion timings could be achieved without requiring any application changes. Even a 10% decrease in overall performance would have been acceptable, but the PoC far exceeded that level of acceptability by delivering a 10% performance increase.

The most remarkable fact for the customer was that no application changes at all were necessary during this upgrade.

The upgrade itself was very easy and smooth – much smoother than was expected by the customer. The automation features like Automatic Statistics Gathering and Automatic SQL Tuning eased maintenance quite a bit during the regular operation of the database. And, the tuning recommendations were detected by the database – no manual intervention or additional tuning time was required.

Overall the results have been rated “excellent”. The actual move of the first 30 of 400 database will be done in January 2009 – all databases should have gone live by end of May 2009.

REFERENCES

1. Why Upgrade to Oracle Database 11g?
   http://www.oracle.com/technology/

2. Oracle Database 11g Upgrade Guide
   http://www.oracle.com/pls/db111/to_pdf?pathname=server.111/b28300.pdf

3. Oracle Database 11g Upgrade Companion
   https://metalink.oracle.com/metalink/plsql/ml2_documents.showDocument?p_database_id=NOT&p_id=601807.1

4. Complete checklist for manual upgrade to Oracle Database 11g:
   https://metalink.oracle.com/metalink/plsql/ml2_documents.showDocument?p_database_id=NOT&p_id=429825.1

5. Testing the SQL Performance Impact of an Oracle 9i/10g Release 1 to Oracle Database 10g Release 2 upgrade with SQL Performance Analyzer
   http://www.oracle.com/technology/products/manageability/database/pdf/owp_spa_9i_10g.pdf

6. Database Upgrade webpage on OTN:
   http://www.oracle.com/technology/products/database/oracle11g/upgrade/index.html

7. Upgrade Discussion Forum on OTN:
   http://forums.oracle.com/forums/forum.jspa?forumID=583

8. Upgrading from Oracle Database 9i to 10g – What to Expect from the Optimizer?
   http://www.oracle.com/technology/products/bi/db/10g/pdf/twp_bidw_optimizer_10gr2_0208.pdf

Upgrading from Oracle 9i to Oracle Database 11g:
A Real World Customer Experience
October 2008 – V1.1
Author: Mike Dietrich
Conributing Authors: Thomas Kempkens, Roy Swonger, Carol Tagliaferri, Carol Palmer

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