

Top Ten Things About Oracle Database 10g Release 2

*An Oracle White Paper
September 2005*

NOTE:

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Top Ten Things About Oracle Database 10g Release 2

INTRODUCTION

It is now a couple of months since the release of Oracle Database 10g Release 2. In this paper and accompanying presentation, we'll take a look at ten of the features/functions I've personally believe will provide great benefit – out of the box reasons for upgrading to Oracle Database 10g Release 2. This is not a comprehensive list of all of the new features/functions in Oracle Database 10g Release 2, that would require a book sized white paper and three or four days to present. Rather this is a short list of ten high impact, immediate payback features. These features were chosen as the result of answering questions on the site <http://asktom.oracle.com/> and customer engagements using the new database software.

Briefly, the features we will take a look at are:

- Conditional Compilation
- Full Database Transports
- XQuery
- SGA Direct Attach Query
- Asynchronous Commits
- Transporting AWR Data
- Transparent Data Encryption
- Online Transportable Tablespaces
- LOG ERRORS DML enhancement
- Audit in XML

So, in this paper we'll be taking a look at these features in particular and pointing you to where you can get more in depth information on each.

This white paper is meant to be accompanied by a presentation as well as a set of scripts that demonstrate many of the features in turn. These are available under the Oracle Open World San Francisco 2005 downloads on <http://otn.oracle.com>.

CONDITIONAL COMPILATION

PL/SQL has lots of new stuff. The first thing I noticed was conditional compilation, something I missed from my days as a C programmer. Conditional compilation is the ability to have code be effectively ignored by the compiler (or not) at will. Sounds “not too useful” at first, but it is. With conditional compilation:

- You can leave your really slow debug code in your application now – and turn it on and off at will.
- You can program assertions as you might in C. Each subprogram can test that the values of its inputs, for example, and verify that they meet some criteria. These tests can be active during the whole of the development cycle and inactive for production. However, they remain as formal documentation of the conditions upon which the unit depends, and can be simply re-activated for debugging a production-time bug.
- You can write version independent code – by compiling one set of code for version X and a different set of code for version Y – without having two sets of code. Check out the new DBMS_DB_VERSION supplied package.

You can support best practices during unit testing. For production, helper subprograms will be declared only in a package's body. But when you want to invoke them from a unit testing framework, they need to be declared in the specification. By conditionally compiling these declarations, you can have your cake and eat it, too: you honor your modular design by hiding what should be hidden in your shipped code; but you allow separate compilation units to see the hidden elements for testing purposes. Further, you can cause helper subprograms to produce expected (data-dependent) exceptions so that you can test the robustness of code that calls your code without the effort of contriving the exceptional data conditions.

The following shows a quick view of what conditional compilation implies and how it works.

```
SQL> create or replace procedure p
  2  as
  3  begin
  4      $IF $$debug_code $THEN
  5          dbms_output.put_line( 'Our debug code' );
  6          dbms_output.put_line( 'Would go here' );
  7      $END
  8      dbms_output.put_line( 'And our real code here' );
  9  end;
 10  /
Procedure created.
```

/* Notice how the 'debug' code is not printed,

```

* and the code compiles without the $$debug_code
* value being defined.*/

```

```

SQL> exec p
And our real code here
PL/SQL procedure successfully completed.

```

```

/* By simply enabling the variable debug_code, we can enable
that debug code. Note that 'debug_code' is my name; it is not a
special name; you can define your own variables at will */

```

```

SQL> alter procedure P compile
2  plsql_ccflags = 'debug_code:true' reuse settings;

```

Procedure altered.

```

SQL> exec p
Our debug code
Would go here
And our real code here
PL/SQL procedure successfully completed.

```

For more information on this feature, see http://download-east.oracle.com/docs/cd/B19306_01/appdev.102/b14261/whatsnew.htm#sthref27

FULL DATABASE TRANSPORTS

Oracle Database 8i introduced transportable tablespaces. Oracle 9i permitted us to use multiple blocksize tablespaces with this feature. Oracle Database 10g Release 1 introduced the cross platform transportable tablespace, but Oracle Database 10g Release 2 takes it to the next level. Now you can transport an entire database across platforms that share the same endianness (byte ordering). That means you can move an entire database (not just transport individual tablespaces) from HP-UX to Apple Macintosh, or from Open VMS to Solaris x86. No more dump and reload.

```

SQL> select endian_format, platform_name
2      from v$transportable_platform
3      order by endian_format
4      /

```

ENDIAN_FORMAT	PLATFORM_NAME
---------------	---------------

Big	HP-UX IA (64-bit)
	Solaris[tm] OE (32-bit)
	Apple Mac OS
	HP-UX (64-bit)
	IBM zSeries Based Linux

	AIX-Based Systems (64-bit)
	Solaris[tm] OE (64-bit)
	IBM Power Based Linux
Little	Solaris Operating System (x86)
	Microsoft Windows 64-bit for AMD
	Linux 64-bit for AMD
	Microsoft Windows IA (64-bit)
	HP Open VMS
	HP Tru64 UNIX
	Linux IA (32-bit)
	Microsoft Windows IA (32-bit)
	Linux IA (64-bit)

For more information on this feature, please see http://download-east.oracle.com/docs/cd/B19306_01/server.102/b14214/chapter1.htm#sthref254 and http://download-east.oracle.com/docs/cd/B19306_01/backup.102/b14191/dbxpitrn.htm#BRADV05432

XQUERY

In addition Oracle 10g Release 2 adds a native implementation of XQuery, W3C's emerging standard for querying XML. This new feature introduces two new functions, XMLQUERY and XMLTABLE. XMLQUERY gives you the ability to query XMLDB repository documents, or XML views over relational data, using the XQuery language, from an SQL client. XMLTABLE maps the result of an XQuery to relational rows and columns, so that result can be queried using SQL.

For more information on this feature, it is suggested you see:

- http://www.oracle.com/technology/pub/articles/10gdba/nanda_10gr2dba_part1.html#xquery, a white paper by Arup Nanda with many good examples
- <http://www.oracle.com/technology/oramag/oracle/05-jan/o15industry.html>, an interview with Jim Melton and Steve Buxton of Oracle on this technology.
- <http://www.w3.org/TR/xquery/>, the specification for XQuery
- http://download-east.oracle.com/docs/cd/B19306_01/server.102/b14214/chapter1.htm#sthref174, in the new features guide

SGA DIRECT ATTACH

This is a new feature that allows Oracle Enterprise Manager to attach directly to the SGA and query relevant performance information – even when you cannot yourself log into the database. Prior releases of Oracle could do this via arcane commands and “oradebug”, but it was typically done at the request of support to provide them with diagnostic information. Now you can use the same technique via Enterprise manager to give you yourself the ability to diagnose performance or “hung database” issues.

This does not replace the normal use of V\$ views in performance tools but rather gives us a way to diagnose a database that cannot be logged into – or upon getting logged in is so slow to respond as to make diagnoses of an issue impossible.

For further reading, I suggest

http://www.oracle.com/technology/pub/articles/10gdba/nanda_10gr2dba_part2.html#sga

ASYNCHRONOUS COMMITS

Normally, when a Java, C, or Visual Basic application issues a COMMIT, that causes a wait – specifically, a log file sync wait. This wait is due to the client waiting for the log writer process to flush the redo to disk – to make the transaction *permanent*. Normally this is exactly what you want to have happen. When you commit, it should be permanent.

However, there are exceptions to all rules. What about a system that is processing incoming records as fast as possible, perhaps from a sensor or a network feed? This program’s goal in life is to batch up a couple records, INSERT them, COMMIT them (to make them visible), and continue on. It doesn’t really care to wait for the COMMIT to complete, in fact, by waiting this program is not running as fast as it can.

Enter the asynchronous commit in Oracle Database 10g Release 2. We can now say “commit, but don’t wait” or “commit, and please do wait”. If you use the commit but don’t wait mode, however, you must be prepared at some point in the future to have data that was “committed” be lost. If you commit but don’t wait and the system fails, that data may not have been committed. Whether this is acceptable to you, only you can decide. But there are many classes of applications—including high speed data ingest programs whose only goal is to get the stream of data into the database—where commit but don’t wait is not only acceptable but very desirable performance-wise.

As a small example, I wrote the Java routine:

```
import java.sql.*;
public class instest
{
    static public void main(String args[]) throws Exception
```

```

{
    DriverManager.registerDriver(new
oracle.jdbc.driver.OracleDriver());
    Connection
        conn = DriverManager.getConnection
            ("jdbc:oracle:thin:@desktop:1521:oral0gr2",
            "scott","tiger");
    conn.setAutoCommit( false );

    Statement sql_trace =
        conn.createStatement();
    sql_trace.execute
    ( "truncate table t" );
    System.out.println( "Table truncated" );

    sql_trace.execute
    ( "begin " +
      "  dbms_monitor.session_trace_enable( WAITS=>TRUE );" +
      "end;" );

    // create table t(x char(2000))
    // create index t_idx on t(x)
    PreparedStatement pstmt =
        conn.prepareStatement
        ("insert into t(x) values(?)" );
    pstmt.setString( 1, "x" );

    PreparedStatement commit =
        conn.prepareStatement
        ("commit work write batch nowait");
        // ("commit work write immediate wait");

    for( int i = 0; i < 1000; i++ )
    {
        pstmt.executeUpdate();
        commit.executeUpdate();
    }
    conn.close();
}
}

```

Note how the commit statement I'm using is either COMMIT WORK WRITE BATCH NOWAIT or COMMIT WORK WRITE IMMEDIATE WAIT. The former is the new feature – it shows how to commit without waiting for the commit to be finalized. The latter is the way it has historically happened. When I ran the test program (the CREATE TABLE is a comment before the INSERT statement for reference), I observed a lot of time spent waiting for “log file sync” waits when using the WAIT option, and virtually no time spent waiting with the NOWAIT option. The results are shown here.

Event waited on	Times Waited	Max. Wait	Total Waited
-----	-----	-----	-----
log file sync (nowait)	19	0.02	0.04
log file sync (wait)	1017	0.04	1.43

This will make a measurable difference in high-speed data load programs that need to commit periodically during the load.

TRANSPORT AWR DATA

This new feature provides the oft requested ability to move and consolidate AWR (Automatic Workload Repository) data across databases. Rather than impact the production system by analyzing the performance data on it, you may not “export” the AWR data for any period of time and import it into another database for analysis.

This is accomplished via three new routines:

- `DBMS_SWRF_INTERNAL.AWR_EXTRACT`
- `DBMS_SWRF_INTERNAL.AWR_LOAD`
- `DBMS_SWRF_INTERNAL.MOVE_TO_AWR`

The extract routine allows you to specify a begin and end snapshot period to unload and creates a file in the file system. This file can then be loaded into the target database via the load routine and then moved into the actual AWR tables using the `MOVE_TO_AWR` routine.

For further information, I suggest:

- http://www.oracle.com/technology/pub/articles/10gdba/nanda_10gr2dba_part3.html#awr, a white paper by Arup Nanda on this feature
- http://download-east.oracle.com/docs/cd/B19306_01/server.102/b14214/chapter1.htm#sthref749, the new features guide.

TRANSPARENT DATA ENCRYPTION

Now for the revolutionary feature in Oracle Database 10g Release 2 – transparent data encryption. This is the ability to easily and – as the name implies – transparently have information stored encrypted. Authorized users need not deal with encryption keys; the data will be decrypted (and encrypted) for them quite transparently. The data is stored encrypted on disk, so that even if someone steals your database, the information is protected. My description and account of my experience with transparent data encryption are not intended to be a thorough “here is everything you can do” example for this new feature, but rather it shows how it is set up and how you would use it.

The first thing I had to do was to set up a wallet. This is where the encryption keys will be stored – and this file is password protected.

For my quick-and-dirty demonstration, I simply created a directory named “wallet” off of my Oracle home and put the following into my `sqlnet.ora` file:

```
WALLET_LOCATION=
(SOURCE=(METHOD=FILE)
(METHOD_DATA=
```

```

(DIRECTORY=/home/ora10gr2/wallet)
)
)

```

Once that was done, all I needed to do was to open the wallet:

```

SQL> alter system
      2  set encryption wallet open
      3  identified by foobar;
System altered.

```

This is something you must do at database startup once, when you start using this feature (or else the encryption keys won't be available to the system and the encrypted data will not be available).

Now, since I had not set up any encryption keys, I needed to do that:

```

SQL> alter system
      2  set encryption key
      3  identified by foobar;
System altered.

```

In this case I let the system generate the key, but I could easily have specified a key of my own choice – something I might do if I wanted to move this data around from system to system easily.

And that was it for setup. We were ready to store encrypted data:

```

SQL> create table t
      2  ( c1 varchar2(80),
      3    c2 varchar2(80) encrypt )
      4  tablespace test;
Table created.

```

```

SQL> insert into t values
      2  ( 'this_is_unencrypted',
      3    'this_is_encrypted' );
1 row created.

```

Now, this data was transparently encrypted on disk and decrypted when we queried it:

```

SQL> select *
      2  from t;

```

C1	C2
-----	-----
this_is_unencrypted	this_is_encrypted

Now, how can we confirm that this data was, in fact, encrypted? We used strings and grep (a Unix utility to search files) on the Oracle datafile to see if we could see the string. First, we made sure the block containing the encrypted data was written to disk:

```
SQL> alter system checkpoint;  
System altered.
```

And then we looked for the strings. We looked for any string in the encrypt.dbf datafile that had “this_is” in it:

```
$ strings -a encrypt.dbf|grep this_is  
this_is_unencrypted
```

As you can see, only the “this_is_unencrypted” string appeared. The other string was not visible because it was, in fact, scrambled before being stored.

This quick example exposes the tip of the iceberg with this new encryption capability. Transparent Data Encryption works with external table unloads, data pump, and so on, and GUI tools, including Oracle Wallet Manager, let you manage your wallet and passwords. There are also commands to “re-key” data in the event you feel the keys have been compromised. Transparent Data Encryption is, I believe, one of the more revolutionary features in Oracle Database 10g Release 2.

ONLINE TRANSPORT OF TABLESPACES

In the past, in order to transport a tablespace from database A to database B – you were required to make that tablespace read only for a period of time in database A. There were two main issues with that:

- Making a tablespace read only must wait for active transactions to complete – that could take a long time and on a busy system, might be something hard to accomplish
- Making a tablespace read only obviously prevents modifications to the data contained in it. That could well be unacceptable on your system.

Enter the new ability to transport tablespaces using your backups. You can now in effect restore a tablespace from database A to database B using RMAN. This gives you greater flexibility as to when the data is “as of”, since the restored file can be recovered to a specific point in time (using an SCN or a timestamp). That is, you can restore to database B a tablespace from database A “as of noon on Tuesday” – something not possible before using conventional transportable tablespaces.

For more information, please see http://download-east.oracle.com/docs/cd/B19306_01/server.102/b14214/chapter1.htm#sthref521 and http://download-east.oracle.com/docs/cd/B19306_01/backup.102/b14191/ontbltrn.htm#BRADV05141

LOG ERRORS

LOG ERRORS is a new clause on DELETE, INSERT, MERGE and UPDATE statements. LOG ERRORS allows for a bulk statement that affects many rows to record rows that failed processing – rather than just failing the entire statement! You can issue a statement such as “INSERT INTO T SELECT A,B,C FROM T2 LOG ERRORS INTO ERRLOG(‘BAD_ROWS_FOR_T’) REJECT LIMIT UNLIMITED”. What that does is log into the table BAD_ROWS_FOR_T any row that violates a constraint, for example it will log errors caused by column values that are too large, constraint violations (NOT NULL, unique, referential, and check constraints), errors raised during trigger execution, errors resulting from type conversion between a column in a subquery and the corresponding column of the table, partition mapping errors, and certain MERGE operation errors (ORA-30926: Unable to get a stable set of rows for MERGE operation.). When an error occurs, the row causing the failure is logged into a bad table along with the Oracle error number, the text of the error message, the type of operation (INSERT, UPDATE, or DELETE) as well as the ROWID of the failed row for UPDATES and DELETES. This new feature is destined to be my favorite Oracle 10g Release 2 new feature for sure! We all know that performing large bulk operations rather than row by row processing is superior for speed and resource usage (let alone ease of coding) but error logging of failed rows has always been an issue in the past. No more with this new feature in place.

With DML Error Logging, instead of the 100,000 row update/insert/whatever failing because a single row doesn’t quite work out, we can have the 99,999 successful rows go through and have the one bad row logged to a table! And – the best part – it is really quite easy and intuitive. Here is a quick demonstration. We’ll just take the SCOTT.EMP table:

```
ops$tkyte-ORA10GR2> create table emp
2  as
3  select * from scott.emp where 1=0;
```

Table created.

and using a new supplied package – we’ll generate the error log for it. You could create this by hand, you could specify the name via this API, you could do lots of stuff, this is the shortest way. It’ll default everything and just name the error log table ERR\$_<tablename> which seems sensible:

```
ops$tkyte-ORA10GR2> exec dbms_errlog.create_error_log(
'EMP' );
```

PL/SQL procedure successfully completed.

```
ops$tkyte-ORA10GR2> desc err$_emp
```

Name	Null?	Type
-----	-----	-----
ORA_ERR_NUMBER\$		NUMBER
ORA_ERR_MESG\$		VARCHAR2(2000)
ORA_ERR_ROWID\$		ROWID
ORA_ERR_OPTYP\$		VARCHAR2(2)
ORA_ERR_TAG\$		VARCHAR2(2000)
EMPNO		VARCHAR2(4000)
ENAME		VARCHAR2(4000)
JOB		VARCHAR2(4000)
MGR		VARCHAR2(4000)
HIREDATE		VARCHAR2(4000)
SAL		VARCHAR2(4000)
COMM		VARCHAR2(4000)
DEPTNO		VARCHAR2(4000)

There is the error logging table, they picked datatypes that would be “safe” for each column — if you inserted a JOB that was too long, no worries, it has to fit into a VARCHAR2(4000). EMPNO isn’t really a number in your input? No problem, it’ll fit into a VARCHAR2 as well. The other columns you see in here are useful too — the ORA-XXXX error number, the text of the error, the ROWID of the offending row(s) from an update or delete, the operating type (I/U/D) and optionally a ‘tag’ you provide as part of your SQL operation. That tag will be useful to determine what step of your ETL (extract/transform/load) process failed.

So, let’s try this out, we’ll add some constraints:

```
ops$tkyte-ORA10GR2> alter table emp add constraint
emp_check_sal check(sal > 900);
```

Table altered.

```
ops$tkyte-ORA10GR2> create trigger emp_trigger
  2  after insert on emp for each row
  3  begin
  4      if ( :new.ename = 'ALLEN' )
  5      then
  6          raise_application_error
  7              (-20024, 'Failed Validation' );
  8      end if;
  9  end;
10  /
```

Trigger created.

```
ops$tkyte-ORA10GR2> alter table emp add
  2  constraint emp_pk primary key(empno);
```

Table altered.

and using just the old fashioned INSERT, we can see what normally would happen:

```
ops$tkyte-ORA10GR2> insert /*+ APPEND */ into emp
  2  select * from scott.emp;
insert /*+ APPEND */ into emp
      select * from scott.emp
*
```

ERROR at line 1:

```
ORA-02290: check constraint
              (OPS$TKYTE.EMP_CHECK_SAL) violated
```

```
ops$tkyte-ORA10GR2> select * from emp;
no rows selected
```

A big “nothing” is what happens, the SQL either entirely happens or doesn’t happen at all. But, add “LOG ERRORS” and check it out:

```
ops$tkyte-ORA10GR2> insert /*+ APPEND */ into emp
      2  select * from scott.emp
      3  LOG ERRORS REJECT LIMIT UNLIMITED;

12 rows created.
```

Now, since anyone that can spell Oracle knows there are 14 rows in emp – something happened here. Something good in a way. We had errors (bad) that were not ignored (good) but logged off to the side (better) and we still took advantage of BULK OPERATIONS (best of all). We can review the bad records:

```
ops$tkyte-ORA10GR2> select ORA_ERR_NUMBER$,
      2      ORA_ERR_OPTYP$,
      3      EMPNO,
      4      ORA_ERR_MESG$
      5  from err$_emp;
```

```
ORA_ERR_NUMBER$ OR EMPNO ORA_ERR_MESG$
-----
      2290 I   7369  ORA-02290: chec
                        k constraint (O
                        PS$TKYTE.EMP_CH
                        ECK_SAL) violat
                        ed

      20024 I   7499  ORA-20024: Fail
                        ed Validation
                        ORA-06512: at "
                        OPS$TKYTE.EMP_T
                        RIGGER", line 4

                        ORA-04088: erro
                        r during execut
```

```

ion of trigger
'OPS$TKYTE.EMP_
TRIGGER'

```

It even works during UPDATES:

```

ops$tkyte-ORA10GR2> update emp
2      set sal = sal - 2000
3      where sal between 2000 and 3000
4      LOG ERRORS ('My Update')
5      REJECT LIMIT UNLIMITED;

3 rows updated.

```

But really, we didn't update three rows. Note this time I passed in a nice little name, which we can now use against this cumulative error log table (the failed inserts are still in there as well)

```

ops$tkyte-ORA10GR2> select ORA_ERR_NUMBER$,
2      ORA_ERR_OPTYP$,
3      ORA_ERR_TAG$,
4      ORA_ERR_ROWID$,
5      EMPNO,
6      ORA_ERR_MESG$
7      from err$_emp
8      where ora_err_tag$ is not null;

```

```

ORA OR ORA_ERR_T ORA_ERR_R EMPNO ORA_ERR_MESG$
----- --
2290 U  My Update AAAM04AAE 7698 ORA-02290: chec
      AAAAGdAAD      k constraint (O
                        PS$TKYTE.EMP_CH
                        ECK_SAL) violat
                        ed

2290 U  My Update AAAM04AAE 7782 ORA-02290: chec

```



```

AAAAGdAAE      k constraint (O
                PS$TKYTE.EMP_CH
                ECK_SAL) violat
                ed

```

We have the row data (EMPNO is filled in) as well as the rowid's of the bad rows. We can see what caused the error — the check constraint violation, and see how many rows violated the check constraint — two in this case. Our update would have modified 3 rows had all of the rows been “OK”, but it really only did 1.

This new feature changes everything. No more excuses in the future, BULK operations are in style, they are efficient and you'll be able to erase a ton of code (which believe it or not is a cool thing, erasing code).

AUDIT IN XML

This new feature provides the ability to have audit trail records written in the industry standard XML format instead of as “text report” as they were in the past when using the file system as your audit trail destination. So, now you may use any XML tool to inspect and query these audit trail entries – including the Oracle database suite of XML functionality. For example, there is a builtin V\$XML_AUDIT_TRAIL view that permits you to query the XML as if it were a relational table. Or you can use XQuery directly on the XML itself.

IN CONCLUSION

This is by no means a comprehensive, inclusive list of new Oracle Database 10g Release 2 features and functions, not close. Rather, it is a short list of compelling new features that I forecast will be used with great success. Many are transparent – like data encryption. Others make life easier – like the full database transport. Others still might change our entire approach to certain problems – like the LOG ERRORS clause. All in all – they are 10 reasons to consider getting on Oracle Database 10g Release 2.



White Paper Title
September 2005
Author: Thomas Kyte
Contributing Authors: Cary Millsap (<http://www.hotsos.com/>)

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

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

Copyright © 2005, Oracle. All rights reserved.

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

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