Oracle 11g for Developers: What You Need to Know

Presented by: John Jay King
Download this paper from: http://www.kingtraining.com

Session Objectives

- Learn new Oracle 11g features that are geared to developers
- Know how existing database features have been improved in Oracle
- Become aware of some DBA-oriented features that impact developers

Who Am I?

- John King – Partner, King Training Resources
- Oracle Ace & member Oak Table Network OakTable.net
- Providing training to Oracle and IT community for over 20 years – http://www.kingtraining.com
- “Techie” who knows Oracle, SQL, Java, and PL/SQL pretty well (along with many other topics)
- Leader in Service Oriented Architecture (SOA)
- Member of ODTUG (Oracle Development Tools User Group) Board of Directors
- Member of IOUG
- Member of RMOUG (but I live in Arizona!)

Oracle 11g R1

- Environment changes
- XML enhancements
- New/improved SQL statements
- New features in PL/SQL
- SQL & PL/SQL Results Caches
- Java, JDBC, and SQLJ improvements
- Pro* and OCI enhancements

Oracle 11g R2

- Results Cache Improvements
- New Analytic Functions
- XML Enhancements
- Java Enhancements
- Pro*C/Pro*COBOL Enhancements
- Edition-Based Redefinition (EBR)

Oracle 11g Preview

- iSqlPlus and SQLPLUSW gone
  (SQL*Plus & SQL Developer still there)
- Virtual Columns
- XML DB Binary XMLTYPE
- SQL Pivot/Unpivot
- REGEXP_COUNT
- PL/SQL compiler enhancement
- Assign sequence numbers in PL/SQL
- PL/SQL CONTINUE
- Trigger improvements
- New JDBC driver support Java 5 (1.5) & 6

Goodbye iSQL*Plus & sqlplusw

- Oracle11g does not include iSQL*Plus
- Oracle 11g does not include the windows version of SQL*Plus (sqlplusw.exe)
- Oracle 11g still includes SQL*Plus (command line)
- Oracle 11g fully supports Oracle SQL Developer (introduced in Oracle 10g)
- Oracle SQL Developer is Oracle's suggested mechanism for SQL and PL/SQL development
- SQL*Plus has been enhanced to deal with BLOB, CLOB, and BFILE data more effectively

Introducing Virtual Columns

- Beginning with Oracle 11g tables may now include virtual columns (dynamic values; not stored)
- Virtual columns obtain their value by evaluating an expression that might use:
  - Columns from the same table
  - Constants
  - Function calls (user-defined functions or SQL functions)
- Virtual columns might be used to:
  - Eliminate some views
  - Control table partitioning (DBA stuff)
  - Manage the new "binary" XMLType data
- Virtual columns may be indexed!

Creating Virtual Column

```
CREATE TABLE NEWEMP
       (EMPNO NUMBER(4) NOT NULL,
        ENAME VARCHAR2(10),
        JOB VARCHAR2(9),
        MGR NUMBER(4),
        HIREDATE DATE,
        SAL NUMBER(7, 2),
        COMM NUMBER(7, 2),
        INCOME NUMBER(9,2)
           GENERATED ALWAYS
           AS (NVL("SAL",0)+NVL("COMM",0))
                        VIRTUAL,
        DEPTNO NUMBER(2));
```

- Datatype defaults if not specified (based upon expression)
- Expression result appears as data in table but is generated
- “generated always” and “virtual” not required, but add clarity

Adding Virtual Columns

- Oracle 11g also allows specification of Virtual Columns via ALTER TABLE

```
alter table myemp
   add (totpay as
        (nvl(sal,0)+nvl(comm,0)));
```

PIVOT/UNPIVOT

- Oracle joins other vendors by adding the PIVOT clause to the SELECT statement
- Adding a PIVOT clause to a SELECT allows rotation of rows into columns while performing aggregation to create cross-tabulation queries
- The PIVOT clause:
  - Computes aggregations (implicit GROUP BY of all columns not in PIVOT clause)
  - Output of all implicit grouping columns followed by new columns generated by PIVOT
- UNPIVOT performs the same activity but converts columns into ROWS (does not “undo” PIVOT)
- Clever developers have used PL/SQL and/or CASE to achieve PIVOT results before, but now it is part of Oracle's standard SQL

PIVOT Example

```
select * from
  (select job,deptno,income from newemp) query1
    pivot (avg(income)
    for deptno in (10 AS ACCOUNTING,
                   20 AS RESEARCH,
                   30 AS SALES))
    order by job;

Job        ACCOUNTING RESEARCH     SALES
ANALYST         30000
CLERK                13000     9500      9500
MANAGER         24500    29750     28500
PRESIDENT       50000
SALESMAN                 19500
```

UNPIVOT Example

```
select * from pivot_emp_table
  unpivot include nulls
    (avgpay for dept in (ACCOUNTING,RESEARCH,SALES))
  order by job;

JOB               DEPT              AVGPAY
ANALYST           ACCOUNTING
ANALYST           RESEARCH          30000
ANALYST           SALES
   /*** more rows ***/
SALESMAN          ACCOUNTING
SALESMAN          RESEARCH
SALESMAN          SALES             19500
```

New SQL Functions

- New functions have also been added to Oracle 11g including:

– CUBE_TABLE	Extracts two-dimensional table from a cube or dimension
– REGEXP_COUNT	Count occurrences of string
– XMLCAST	Cast XML data to SQL datatype
– XMLEXISTS	Determine if XQuery returns values
– XMLDIFF	Used to compare two XMLType documents
– XMLPATCH	Used to patch an XMLType document

Oracle 11g Read-Only Tables

- Beginning with Oracle 11g the database supports read-only table mode

```
alter table myTable read only;
```

```
alter table myTable read write;
```

  - When a table is in read only mode INSERT, UPDATE, DELETE, and MERGE fail
  - However, SELECT, CREATE INDEX, and other commands that do not alter data are allowed

Invisible Indexes

- Sometimes the optimizer selects the wrong index
  - Beginning with Oracle 11g it is possible to make an index “invisible” to the optimizer
  - Use ALTER TABLE to make it visible/invisible

```
create index mytab_ix on mytab(mykey) invisible
```

```
alter intex mytab_ix invisible;
```

```
alter index mytab_ix visible;
```

Results Caching

- Caching is nothing new to Oracle;
  Oracle has cached data for a long time now
- What' s new is the caching of results…
- This is similar to how a Materialized View works but is more-dynamic
- New "result_cache" hint asks Oracle to cache query results

Result Cache – Test Query

```
select cust_last_name || ', ' || cust_first_name cust_name
      ,cust_city
      ,prod_id
      ,count(*) nbr_sales
 from sh.customers cust
    join sh.sales sales
      on cust.cust_id = sales.cust_id
 where country_id = 52789
   and prod_id in (120,126)
 group by cust_last_name,cust_first_name,cust_city,prod_id
 having count(*) > 10
 order by cust_name,nbr_sales;
```

- This query was run three times in succession with timing turned on; resulting timings were
  - Elapsed: 00:00:00.67
  - Elapsed: 00:00:00.46
  - Elapsed: 00:00:00.37

Using Result Cache

```
select /*+ result_cache */ cust_last_name || ', ' || cust_first_name
   cust_name
       ,cust_city
       ,prod_id
       ,count(*) nbr_sales
 from sh.customers cust
    join sh.sales sales
      on cust.cust_id = sales.cust_id
 where country_id = 52789
   and prod_id in (120,126)
 group by cust_last_name,cust_first_name,cust_city,prod_id
 having count(*) > 10
 order by cust_name,nbr_sales;
```

- This query was run three times in succession with timing turned on; resulting timings were
  - Elapsed: 00:00:00.23
  - Elapsed: 00:00:00.01
  - Elapsed: 00:00:00.03

PL/SQL Result Cache

- PL/SQL allows specification of a result_cache for function/procedure calls
- Add the clause "result_cache" just before the "AS/IS" keyword in the Function and/or Procedure definition
  (Oracle 11g R1 also used now-obsolete "relies_on" clause)
- The results of a call to the Function or Procedure with a specific set of input parameters is stored for later re-use

PL/SQL Result Cache - Code

```
CREATE OR REPLACE FUNCTION RESULT_CACHE_ON
   (in_cust_id sh.customers.cust_id%type,  in_prod_id
   sh.sales.prod_id%type)
RETURN number
RESULT_CACHE -- RELIES_ON (SH.CUSTOMERS, SH.SALES)
authid definer
AS
 sales number(7,0);
BEGIN
select count(*) nbr_sales  into sales
 from sh.customers cust join sh.sales sales
      on cust.cust_id = sales.cust_id
 where cust.cust_id = in_cust_id
  and  prod_id = in_prod_id;
 return sales;
EXCEPTION
  when no_data_found then return 0;
END RESULT_CACHE_ON;
```

PL/SQL Result Cache - Timings

```
 1* select result_cache_on(4977,120) from dual
RESULT_CACHE_ON(4977,120)
-------------------------
                       14
Elapsed: 00:00:00.40

  1* select result_cache_on(4977,120) from dual
RESULT_CACHE_ON(4977,120)
-------------------------
                       14
Elapsed: 00:00:00.00

  1* select result_cache_on(4977,120) from dual
RESULT_CACHE_ON(4977,120)
-------------------------
                       14
Elapsed: 00:00:00.01
```

PL/SQL Enhancements

- Oracle 11g's changes to PL/SQL are very interesting to the developer:
  - PL/SQL has been improved to include all of the XMLType, BLOB, Regular Expression, and other functionality added to SQL
  - Improvements have been made to the compiler
  - New PL/SQL data types
  - Sequence number use is easier
  - "continue" added for loop control
  - CALL syntax has improved

Compiler Enhancement

- In previous releases, the PL/SQL compiler required a standalone “C” compiler
- Oracle 11g now provides a native compiler for PL/SQL eliminating the need for a separate compiler

```
ALTER PROCEDURE my_proc COMPILE
  PLSQL_CODE_TYPE=NATIVE REUSE SETTINGS;
ALTER PROCEDURE my_proc COMPILE
  PLSQL_CODE_TYPE=INTERPRETED
         REUSE SETTINGS;
ALTER SESSION SET
  PLSQL_CODE_TYPE=NATIVE;
ALTER SESSION SET
  PLSQL_CODE_TYPE=INTERPRETED;
```

Compound Triggers

- Compound triggers allow the same code to be shared across timing points

  (previously accomplished using packages most of the time)

- Compound triggers have unique declaration and code sections for timing point

- All parts of a compound trigger share a common state that is initiated when the triggering statement starts and is destroyed when the triggering statement completes (even if an error occurs)

Compound Trigger Timing

- If multiple compound triggers exist for the same table; they fire together:
  - All before statement code fires first
  - All before row code fires next
  - All after row code fires next
  - All after statement code finishes
- The sequence of trigger execution can be controlled only using the FOLLOWS clause

Compound Trigger Syntax

```
CREATE TRIGGER compound_trigger
  FOR UPDATE OF sal ON emp
    COMPOUND TRIGGER
  -- Global Declaration Section
  BEFORE STATEMENT IS
  BEGIN …
  BEFORE EACH ROW IS
  BEGIN …
  AFTER EACH ROW IS
  BEGIN …
END compound_trigger;
/
```

TRIGGER … FOLLOWS

- Oracle 11g adds the “FOLLOWS” clause to trigger creation allowing control over the sequence of execution when multiple triggers share a timing point
- FOLLOWS indicates the including trigger should happen after the named trigger(s); the named trigger(s) must already exist
- If some triggers use “FOLLOWS” and others do not; only the triggers using “FOLLOWS” are guaranteed to execute in a particular sequence

How FOLLOWS Works

- FOLLOWs only distinguishes between triggers at the same timing point:
  - BEFORE statement
  - BEFORE row
  - AFTER row
  - AFTER statement
  - INSTEAD OF
- In the case of a compound trigger, FOLLOWS applies only to portions of triggers at the same timing point (e.g. if a BEFORE ROW simple trigger names a compound trigger with FOLLOWS the compound trigger must have a BEFORE ROW section and vice versa

FOLLOWS Syntax

```
CREATE OR REPLACE TRIGGER myTrigger
    BEFORE/AFTER/INSTEAD OF  someEvent
    FOR EACH ROW
    FOLLOWS someschema.otherTrigger
    WHEN (condition=true)
    /* trigger body */
```

- FOLLOWS may specify a list (and designate sequence)

```
FOLLOWS otherTrigger1, otherTrigger2, etc
```

New PL/SQL Datatypes

- Oracle 11g adds three new PL/SQL datatypes: Simple_integer, Simple_float, Simple_double
  - The three new datatypes take advantage of native compilation features providing faster arithmetic via direct hardware implementation
  - SIMPLE_INTEGER provides a binary integer that is neither checked for nulls nor overflows
  - SIMPLE_INTEGER values may range from -2147483648 to 2147483647 and is always NOT NULL
  - Likewise, SIMPLE_FLOAT and SIMPLE_DOUBLE provide floating point without null or overflow checks

Example SIMPLE_INTEGER

```
declare
--  mytestvar pls_integer := 2147483645;
  mytestvar simple_integer := 2147483645;
begin
  loop
     mytestvar := mytestvar + 1;
     dbms_output.put_line('Value of mytestvar is now '
                              || mytestvar);
     exit when mytestvar < 10;
  end loop;
end;
Results in:
Value of mytestvar is now 2147483646
Value of mytestvar is now 2147483647
Value of mytestvar is now -2147483648
```

Without SIMPLE_INTEGER

- If the “mytestvar” variable is switched to PLS_INTEGER, an ORA-1426 NUMERIC OVERFLOW exception occurs

```
Error report:
   ORA-01426: numeric overflow
   ORA-06512: at line 7
   01426. 00000 -  "numeric overflow"
   *Cause:    Evaluation of an value expression causes
   an overflow/underflow.
   *Action:   Reduce the operands.
   Value of mytestvar is now 2147483646
   Value of mytestvar is now 2147483647
```

Sequences in PL/SQL

- Sequence values NEXTVAL and CURRVAL may be use in PL/SQL assignment statement

```
myvar := myseq.nextval;
```

CONTINUE

- CONTINUE “iterates” a loop; branching over the rest of the code in the loop and returning to the loop control statement

```
begin
    dbms_output.put_line('Counting down to blastoff!');
    for loopctr in reverse 1 .. ctr loop
      if loopctr in (4,2) then
          continue;
      end if;
      dbms_output.put_line(to_char(loopctr));
    end loop;
    dbms_output.put_line('Blast Off!');
end;
Counting down to blastoff!
6
5
3        <-Values “4” and “2” do not appear in the output
1
Blast Off!
```

REGEXP_COUNT

- REGEXP_COUNT counts the number of times a pattern occurs in a source string

```
select ename,regexp_count(ename,'l',1,'i') from emp;
SMITH    0
ALLEN    2
WARD     0
JONES    0
MARTIN   0
BLAKE    1
/** more rows ***/
MILLER   2
```

  - string expression and/or column to match pattern
  - Regular Expression pattern
  - Beginning position in the source string (default=1)
  - Match parameters (i = case insensitive, c = case sensitive, m = multiple line source delimited by ‘^’ or ‘$’, n = matches ‘.’ newline characters (default no), and x = ignore whitespace characters (default is to match)

CALL with Mixed Parameters

- PL/SQL allows function and procedure parameters to be specified in two ways; by position and by name
- With Oracle 11g SQL, parameter types may now be mixed
- Given the following function:

```
CREATE OR REPLACE
FUNCTION TEST_CALL (inval1 IN NUMBER, inval2 IN
  NUMBER,
     inval3 IN NUMBER) RETURN NUMBER AS
BEGIN
  RETURN inval1 + inval2 + inval3;
END TEST_CALL;
```

- The following calls all now work:

```
test_call(vara,varb,varc)
test_call(inval3=>varc,inval1=>vara,inval2=>varb)
test_call(vara,inval3=>varc,inval2=>varb)
```

Non-PL/SQL Development

- Pro*C++ and Pro*COBOL improvements include:
  - Supports DB2-style array INSERT and SELECT syntax
  - Client-Side Query Cache & Oracle Outlines work
  - Oracle 11g Java Enhancements include:
    - Java SE 5 (JDK 1.5) is new base level
    - JIT enabled by default; automatic native compile
    - JDBC 4.0 supported
  - Microsoft .NET and Visual Studio .NET 2005
  - PL/SQL Debugging in Visual Studio .NET 2005
  - Designer and integration using Data Windows via Visual Studio .NET 2005 DDEX
  - Oracle Data Provider for .NET (ODP.NET)
- PHP Enhancements
  - Zend Technologies collaboration; Zend Core for Oracle may be downloaded from OTN

New Analytics (11gR2)

- Oracle 11gR2 has improved upon the already-impressive analytic functions first introduced in Oracle 8i adding:
  - LISTAGG
  - NTH_VALUE

LISTAGG (11gR2)

- LISTAGG provides lists of lower-level columns after aggregation

```
select department_id,
       listagg(last_name, ', ')
       within group
       (order by last_name) dept_employees
       from hr.employees
       where department_id in (20,30)
       group by department_id
       order by department_id;

   DEPARTMENT_ID  DEPT_EMPLOYEES
   -------------  -----------------------------------------


              20  Fay, Hartstein

              30  Baida, Colmenares, Himuro, Khoo,
                  Raphaely, Tobias
```

NTH_VALUE (11gR2)

- NTH_VALUE simplifies the process of retrieving the “n-th” value

```
select distinct department_id
    ,first_value(salary)  ignore nulls
       over (partition by department_id order by salary desc
       rows between unbounded preceding and unbounded following)
      "1st"
      ,nth_value(salary,2) ignore nulls
       over (partition by department_id  order by salary desc
       rows between unbounded preceding and unbounded following)
      "2nd"
      ,nth_value(salary,3) ignore nulls
       over (partition by department_id  order by salary desc
       rows between unbounded preceding and unbounded following)
      "3rd"
      from hr.employees
      where department_id = 80
      order by department_id, "1st", "2nd", "3rd";

DEPARTMENT_ID        1st        2nd        3rd
------------- ---------- ---------- ----------
           80      14000      13500      12000
```

Recursive Subquery

- Oracle’s CONNECT BY has allowed definition of a hierarchical relationship for years; now an ISO-standard option is available:

```
with empConnect(last_name,employee_id,manager_id,lvl)
     as (select last_name, employee_id, manager_id, 1 lvl2
             from hr.employees where manager_id is null
             union all
         select emp.last_name, emp.employee_id,
             emp.manager_id, ec.lvl+1
             from hr.employees emp, empConnect ec
             where emp.manager_id = ec.employee_id)
     SEARCH DEPTH FIRST BY last_name SET order_by
select lvl,lpad(' ' ,3*lvl, ' ')||last_name empname
    from empConnect
    order by order_by
```

External Directory Features

- With Oracle 11gR2 the EXECUTE privilege may be granted for Directory objects; allowing execution of code stored in host operating system files
- Pre-processing programs may be specified for External files used via Oracle Loader
(perhaps to unzip, decrypt, translate,…)

Data Pump “Legacy Mode”

- Oracle 11gR2 has provided “legacy mode” for Oracle Data Pump
- Allows execution of existing Import/Export scripts
- When Data Pump recognizes Import/Export parameters it automatically switches to “legacy mode” and executes as desired

11gR2 XML Enhancements

- Binary XML has been enhanced with significant performance improvements
- Default XMLType storage is now Binary using SecureFile (used to be Unstructured)
- Unstructured XMLType is “deprecated”
- XMLIndex improved allowing indexing for all XMLTypes and for fragments via XPath and partitioning
- Partitioning now allowed for XMLType data

Binary XML

- Oracle continues its XML leadership in Oracle 11g
- Biggest change is the addition of a new “binary” XMLType
  - “binary xml” is a third method for storing XML data in the database
  - “structured” and “unstructured” XMLType still supported
  - Oracle 11g’s XML processors includes a binary XML encoder, decoder, and token manager
  - XML 1.0 text may be parsed via SAX events with or without a corresponding schema into “binary” XML form
  - “binary” XMLType allows optimization of some XML applications by reducing memory and CPU expense

Next-Gen. LOB: Securefile

- Oracle 11g provides a new, more-secure, faster mechanism for storing Large Objects (e.g. XMLType data)
- LOB column specifications in CREATE TABLE or ALTER TABLE include STORE AS SECUREFILE
- SECUREFILE provides compression and encryption for Large OBjects (LOBs)
  - Oracle 11g will detect duplicate LOB data and conserve space by only storing one copy ("de-duplication" if SECUREFILE is specified).
  - PL/SQL packages and OCI functions have been added to take advantage of SECUREFILE LOBs
  - SECUREFILE lobs provide higher performance through reduced size and resource use.

XML Indexes

- Replaces CTXSYS.CTXXPATH indexes
- XML-specific index type, indexes document XML structure
- Designed to improve indexing unstructured and hybrid XML
- Determines XPath expressions for a document's XML tags
- Indexes singleton (scalar) nodes and items that occur multiple times
- XMLIndex record document child, descendant, and attribute axes (hierarchy) information
- XMLIndex is be design general (like CTXXPATH) rather than specific like B-tree indexes
- XMLIndex applies to all possible XPath document targets
- XMLIndex may be used for XMLQuery, XMLTable, XMLExists, XMLCast, extract, extractValue, and existsNode
- XMLIndex helps anywhere in the query, not just in the WHERE clause

Creating XMLIndex

- The syntax to create an XMLIndex looks a little different from non-XML indexes; it is made up of three parts:
  - Path index   Indexes XML tags and identifies document fragments
  - Order index   Indexes the hierarchy of nodes
  - Value index   Values to match WHERE clauses (may be exact match or range)
- XMLIndex uses a “Path Table” to store the various node paths in an XML document; if not specified in the CREATE INDEX statement Oracle will generate a name for you

```
CREATE INDEX po_xmlindex_ix
   ON po_clob (OBJECT_VALUE)
   INDEXTYPE IS XDB.XMLIndex
   PARAMETERS ('PATH TABLE my_path_table');
```

Edition-Based Redefinition (EBR)

- The quest to eliminate downtime has led to a desire to provide "Online Application Upgrade" where an application need not be taken down when upgrades are applied
  - Users of the existing system continue uninterrupted
  - Users of the upgraded system use new code immediately

How?

- Oracle 11gR2 Edition-Based Redefinition adds a new non-schema "edition" of an application including all of the original edition's PL/SQL, views, and synonyms; the new edition may be modified as desired then tested and deployed without impacting the users of the original edition
- Once the new edition is ready for complete rollout it may be released
- This is accomplished by a combination of:
  - Editioning Views
    Showing the data "as of" a specific edition
  - Cross-Edition Triggers
    Triggers keeping "old" and "new" editions synchronized

More Information on EBR

- Edition-Based Redefinition is one of the most-exciting aspects of Oracle 11g R2 to get more information on this amazing new feature see:
  - White Paper on OTN: **http://www.oracle.com/technology/deploy /availability/pdf/edition_based_redefinition.pdf**
  - Tutorial on OTN: **http://www.oracle.com/technology/obe /11gr2_db_prod/appdev/ebr/ebr_otn.htm**
  - Bryn Llewellyn interview on Oracle Development Tools User Group (ODTUG) website **http://www.odtug.com**
  - My paper on **http://www.kingtraining.com**

Wrapping it all Up

- Oracle 11g adds significant new functionality to the already robust database environment
- With the production release of Oracle 11g R2 it's probably time for organizations to really get serious about moving off of earlier releases
- While an emphasis is sometimes placed on the features of Oracle that support the Data Base Administrator, this paper shows many Developer-oriented features of great usefulness

See you in Denver Colorado!

COLLABORATE13

TECHNOLOGY AND APPLICATIONS FORUM
FOR THE ORACLE COMMUNITY

April 2013 – Get Ready to Go!

Please Fill Out Session Evaluations

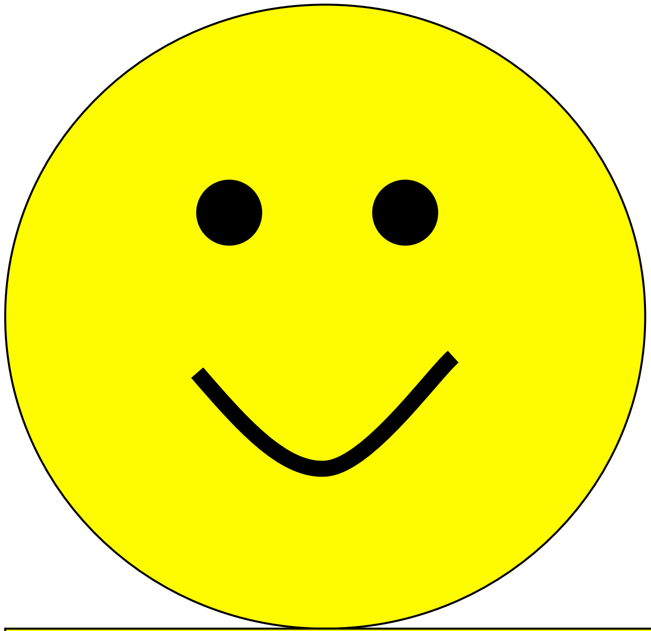
Oracle 11g for Developers: What You Need to Know

To contact the author:

John King

King Training Resources

P. O. Box 1780
Scottsdale, AZ 85252 USA
1.800.252.0652 - 1.303.798.5727
Email: john@kingtraining.com



Thanks for your attention!

Today’s slides and examples are on the web:

http://www.kingtraining.com