King Training Resources

Oracle8/8i Differences for Developers: What you need to know...

John King
King Training Resources
6341 South Williams Street
Littleton, CO 80121-2627 USA
www.kingtraining.com
800.252.0652 or 303.798.5727

Objectives

Learn new Oracle8/8i features geared to developers

Be aware of new non-object features in Oracle8
and Oracle8i

Be aware of new object-oriented features in
Oracle8 and Oracle8i

Learn selected features of Oracle 8i Release 2 (8.1.6)

Non-Object Features

- Max. size of character columns increased
- New large-object datatype support
- SQL and PL/SQL for large objects and directories
- Deferred constraints
- Read-only views / INSTEAD OF triggers for views
- External Procedures and Advanced Queuing
- Bulk-Bind
- RETURNING clause on UPDATE and DELETE
- CASE statement for conditional SQL
- CUBE and ROLLUP extensions to GROUP BY, “Analytic functions”
- New index types and indexing using functions/expressions
- DDL and Database event triggers
- Materialized views
- PL/SQL invoker rights
- Autonomous transactions
- Temporary Tables
- DBA-oriented new features

Max. Size of Character Cols.

- CHAR columns may now be up to 2000 bytes long
  (old limit was 255 bytes)
- VARCHAR/VARCHAR2 columns may now be up to 4000 bytes long
  (old limit was 2000 bytes)
- LONG and LONG RAW remain the same
- Sizes still do not match PL/SQL, potential for truncation of long values still very real

National-Language Characters

- Oracle8 provides two character datatypes specifically designed for Unicode-standard multi-byte character data

– NCHAR	Fixed length multi-byte Max. size 2000 bytes (typically 1000 chars.)
– NVARCHAR2	Var. length multi-byte Max. size 4000 bytes (typically 2000 chars.)

- The “national” character set to be used is set by the DBA when the database is created

LOB Support

- Oracle8 provides four types of Large Objects (LOBs) in addition to LONG and LONG RAW, each allows storage of up to 4GB

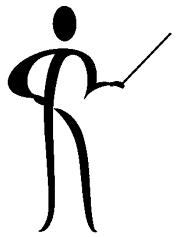
– BFILE	Reference host system file (BFILE is read-only)
– BLOB	Internal binary large object
– CLOB	Internal character large object
– NCLOB	Internal national character large object

- CREATE TABLE adds a LOB specification to help describe LOBs

LOB Rules

- May have multiple LOB’s in a row
- Oracle transaction backup/recovery covers BLOB, CLOB, and NCLOB (internal LOBs)
- Oracle transaction backup/recovery does not cover BFILE (external LOB)
- BLOB, CLOB, and NCLOB data may be stored together with table row data, or, a pointer to the data may be stored

SQL LOB-related Functions



- CLOB and BLOB data may be initialized by calling EMPTY_CLOB() and EMPTY_BLOB() respectively
- BFILEs may be named using BFILENAME(‘dirname’,’file.nam’)
- Generic directories (for portability) may be created using
  CREATE DIRECTORY

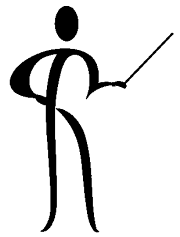
PL/SQL Support for LOBs

- PL/SQL provides a built-in package named DBMS_LOB for processing and manipulating LOBs allowing three basic types of processing:
  - Read/Examine LOB values (all LOBs)
  - Alter values in CLOB, BLOB, or NCLOB
  - Read values in a BFILE
- Several new PL/SQL exceptions have been created to handle problems associated with DBMS_LOB activity
- PL/SQL manipulation of LOBs is more powerful and straight-forward than direct SQL manipulation

Deferred Constraints

- Sometimes working around Referential Integrity constraints makes logic complex
- Oracle8 lets constraints be DEFERRABLE
- Deferred constraints allow DML to do things that might normally be disallowed
- Constraints are still enforced at COMMIT
- Constraints may be:
  - DEFERRABLE INITIALLY IMMEDIATE
  - DEFERRABLE INITIALLY DEFERRED
- ALTER SESSION may be used to defer constraints (only if marked DEFERRABLE)

INSTEAD OF View Triggers



- To facilitate UPDATE and INSERT logic against views, Oracle8 provides the INSTEAD OF trigger

```
CREATE OR REPLACE TRIGGER xxx
                   INSTEAD OF INSERT ON myview
                 or INSTEAD OF UPDATE
                 or INSTEAD OF DELETE
        DECLARE
                         …
        BEGIN
                         …
        /* Code to manipulate necessary tables using :NEW values */
        …
        END;
```

External Procedures

- PL/SQL Version 8 allows external procedures not written in PL/SQL
  - Program is written/purchased and installed as an executable in the host environment
  - Program is a “Dynamic Link” program similar to those supported by Windows and Solaris
  - Net8 (formerly SQL*Net) listener is modified to watch for the external process
  - A LIBRARY is created in Oracle8 (use CREATE LIBRARY) identifying the path
  - A PL/SQL procedure (known as a Wrapper Procedure) is created to act as an interface
  - PL/SQL can execute as a procedure

Advanced Queuing

- Oracle8 Advanced Queuing is more flexible than DBMS_PIPE and DBMS_ALERT for communicating between sessions
- Each session ENQUEUEs data into a table using a new set PL/SQL built-ins

– DBMS_AQ	Used to create queue tables and control who uses them
– DBMS_AQADM	DBMS_AQ.ENQUEUE & DBMS_AQ.DEQUEUE to add/remove entries

RETURNING Clause

- The RETURNING clause may be added to INSERT, UPDATE and DELETE allowing access to values AFTER the DML statement has completed (and after triggers may have modified data) saving a network round-trip

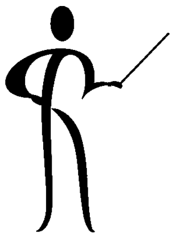
```
UPDATE emp
SET SAL = SAL * 1.1
WHERE JOB = ‘CLERK’
RETURNING SAL INTO :NEWSAL;
```

Subqueries Anywhere!

- Oracle8i allows the use of subqueries just about anywhere in the SQL statement
- Here are four oddball statements that would not be possible in earlier versions

```
select ename,job,sal,(select avg(sal) from emp where job = main.job) jobavgsal
                        from emp main;
select ename,sal from emp
  where sal between (select avg(sal) from emp where job = 'SALESMAN')
                      and (select avg(sal) from emp where job = 'ANALYST');
select deptno from dept
  where (select avg(sal) from emp) > (select avg(sal) from emp
                                                      where emp.deptno = dept.deptno);
select ename,sal from emp
  order by (select dname from dept where dept.deptno = emp.deptno),ename;
```

CASE Expression (8.1.6)



- Oracle has added the CASE expression to allow more complex processing than DECODE (ANSI/ISO standard)
- CASE allows IF…THEN…ELSE logic to be placed anywhere in SQL that a column or literal can go
- CASE syntax is as follows:

```
CASE WHEN condition1   THEN expression1
     WHEN condition2   THEN expresssion2
      …
     WHEN conditionn   THEN expressionn
     ELSE expression
END
```

- One WHEN THEN pair is required, ELSE is optional (default is NULL), END is required

CASE Example (8.1.6)

```
select ename,sal,case when job = 'CLERK' then 'GLUE'
                      when job = 'MANAGER' then 'SUPER'
                      else job
                     end job_x
      from emp
      where case when sal < 1000 then sal + 2000
                 when sal < 2000 then sal + 1000
                 else sal
            end    > 2900
      order by case when sal < 1000 then sal + 9000
                    when sal < 2000 then sal + 7000
                    else sal
               end
    ENAME           SAL        JOB_X
    ------------    ----------  -----------------
    JONES           2975       SUPER
    FORD            3000       ANALYST
    SCOTT           3000       ANALYST
    KING            5000       PRESIDENT
    JAMES            950       GLUE
```

Bulk Bind

- For years, PL/SQL developers have chafed under the restriction that SELECT statements returning more than one row must use a cursor
- Pro*C, Pro*COBOL, and Pro*Fortran have allowed SELECT … INTO arrays for years
- Selecting directly into an array avoids the overhead and network traffic associated with cursors
- Bulk bind now allows the processing of arrays via:
  - BULK COLLECT added to SELECT, FETCH, and RETURNING clauses
  - New for FORALL process for collections (PL/SQL tables)

BULK COLLECT

- Fetches all result rows directly into the PL/SQL tables one operation, removing cursor processing (works in SELECT, FETCH, and RETURNING)
- Careful! Enough memory must be available

```
declare
  type insal_type is table of many_emps.sal%type
     index by binary_integer;
  type inename_type is table of many_emps.ename%type
     index by binary_integer;
  insal insal_type;
  inename inename_type;
begin
     select ename,sal
        bulk collect into inename,insal
        from many_emps
        where deptno = 10;
```

FORALL

- FORALL is a new automatically incrementing process for executing one DML using PL/SQL table values

```
declare
  type insal_type is table of many_emps.sal%type
     index by binary_integer;
  type inename_type is table of many_emps.ename%type
     index by binary_integer;
  insal insal_type;
  inename inename_type;
begin
     …
   FORALL idx IN inename.FIRST .. inename.LAST
         insert into bonus (ename,sal)
           values (inename(idx),insal(idx));
end;
/
```

Partition Support in DML

- If an installation is using partitions, DML statements may refer to a desired partition thus reducing the search time required to identify rows
- Most partitioned tablespaces use indexes to separate the partitions by key value
- When an index is being used by Oracle, it will automatically search only the appropriate partitions
- Careful! Hard-coded partition numbers may create a maintenance issue...

```
DELETE FROM xxx PARTITION (yyy) WHERE … ;
```

Oracle 8.1.6 Aggregates

- AVG
- CORR
- COUNT
- COVAR_POP
- COVAR_SAMP
- GROUPING
- MAX
- MIN
- REGR_AVGX
- REGR_AVGY
- REGR_COUNT
- REGR_INTERCEPT
- REGR_R2
- REGR_SLOPE
- REGR_SXX
- REGR_SYY
- REGR_SXY
- STDDEV
- STDDEV_POP
- STDDEV_SAMP
- SUM
- VAR_POP
- VAR_SAMP
- VARIANCE

Oracle8i Version 2 (8.1.6) Analytic Functions

- Oracle 8.1.6 includes a new set of functions designed to provide expanded support for data mining operations -
  (this topic is too rich to fully cover in the context of this paper)
- The analytic functions are divided into four "families"
- Lag/Lead Compares values of rows to other rows in same table: LAG, LEAD
- Ranking Supports "top n" queries: CUME_DIST, DENSE_RANK, NTILE, PERCENT_RANK, RANK, ROW_NUMBER
- Reporting Aggregate Compares aggregates to non-aggregates (pct of total):
  RATIO_TO_REPORT
- Window Aggregate Moving average type queries: FIRST_VALUE, LAST_VALUE
- The analytic functions allow users to divide query result sets into ordered groups of rows called partitions
  (not the same as database partitions)

Oracle8i Version 2 (8.1.6) Analytic Function Clauses

- Along with the new functions came new clauses (again, too rich to cover completely here):

analytic_function ( ) OVER (analytic clause)

  – Analytic clause
    Query_partition_clause-Order_by clause-Windowing clause
  – Query partition clause
    PARTITION BY list,of,cols
  – Windowing clause
    RANGE … or **ROWS ...**
  – Order by clause
    ORDER BY col,list

CUBE and ROLLUP

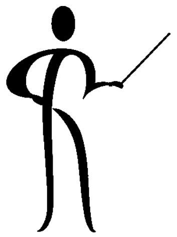
- CUBE and ROLLUP extend GROUP BY
- ROLLUP builds subtotal aggregates at any level, including grand total
- CUBE extends ROLLUP to calculate all possible combinations of subtotals for a GROUP BY
- Cross-tabulation reports are easy with CUBE
- Oracle8i Release 2 (Oracle version 8.1.6) began release in February 2000, it’s new “Analytic” functions include: ranking, moving aggregates, period comparisons, ratio of total, and cumulative aggregates

ROLLUP Example

```
select nvl(to_char(deptno),'Grand') deptid,
         nvl(job,'Total') job,
         sum(sal) as sal
       from emp
       group by rollup (deptno,job);
```

```
DEPTID JOB                SAL
------ --------- ----------
10     CLERK             1300
. . .
10     Total             8750
20     ANALYST           6000
. . .
20     Total            10875
30     CLERK              950
. . .
30     Total             9400
Grand  Total            29025
```

Using GROUPING



```
select decode(grouping(deptno),1,'All Departments',deptno) deptno
      ,decode(grouping(job),1,'All Jobs',job) job
      ,sum(sal) as sal
    from emp
    group by rollup (deptno,job)
```

```
      DEPTNO                    JOB              SAL
---------------------------- --------- ----------
10                          CLERK             1300
. . .
10                          All Jobs          8750
20                          ANALYST           6000
. . .
20                          All Jobs         10875
30                          CLERK              950
. . .
30                          All Jobs          9400
All Departments             All Jobs         29025
```

CUBE Example

```
select decode(grouping(deptno),1,'All Departments',deptno) deptno
      ,decode(grouping(job),1,'All Jobs',job) job
      ,sum(sal) as sal
    from emp
    group by cube (deptno,job)
```

```
DEPTNO                        JOB              SAL
----------------------------- --------- ----------
10                            CLERK           1300
. . .
10                            All Jobs        8750
. . .
All Departments               ANALYST         6000
All Departments               CLERK           4150
All Departments               MANAGER         8275
All Departments               PRESIDENT       5000
All Departments               SALESMAN        5600
All Departments               All Jobs       29025
```

Function/Expression-Based Index

- Indexes may be defined for column values after execution of a function or expression
- This provides the ability to use an index in common situations
- Using function/expression indexes requires that the transaction have QUERY_REWRITE_ENABLED =TRUE and for transactions enabling user-defined functions QUERY_REWRITE_INTEGRITY = TRUSTED

```
CREATE INDEX … ON EMP (UPPER(ENAME)) …
CREATE INDEX … ON NEMP (NVL(SAL,0)+NVL(COMM,0)) …
```

```
SELECT … WHERE UPPER(ENAME) = UPPER(:hostvar) …
SELECT … WHERE NVL(SAL,0)+NVL(COMM,0) > 1000 …
```

DDL and Database Triggers

- DDL triggers fire due to CREATE, ALTER, or DROP statements:
  - BEFORE CREATE or AFTER CREATE
  - BEFORE ALTER or AFTER ALTER
  - BEFORE DROP or AFTER DROP
- Database event triggers fire when system-level events occur:
  - LOGON
  - LOGOFF
  - SERVERERROR
  - STARTUP
  - SHUTDOWN

Materialized Views

- Allow a view's results to be stored as materialized in the database for use by subsequent SQL statements
- View materialization is refreshed periodically or upon demand
- Oracle8i Release 2 (8.1.6) allows an ORDER BY clause

```
create materialized view dept_summary
         refresh start with sysdate next sysdate + 1
       as
         select dname,count(*), nbr_emps,
                sum(nvl(sal,0)) tot_sal
               from emp,dept
               where emp.deptno(+) = dept.deptno
               group by dname
```

Invoker Rights

- By default, stored PL/SQL is executed under the security domain of the userid used to compile the stored PL/SQL
- Occasionally, it might be useful to require the user executing stored PL/SQL to have the security authorization to perform all actions contained within the code
- Oracle8i provides a new clause on the CREATE statements allowing control over the security domain used at execution

```
create procedure xxx (parameter list)
          AUTHID DEFINER              -- default, works like existing PL/SQL
          as … pl/sql block …

create procedure yyy (parameter list)
          AUTHID CURRENT_USER         -- new with Oracle8i
          as … pl/sql block …
```

Autonomous Transactions

- Autonomous transactions allow a COMMIT/ROLLBACK transaction sequence within a code block that is not connected to the COMMIT/ROLLBACK in the outer transaction
- Place the following line in **the declarative section** of any anonymous PL/SQL block, Procedure, or Function

```
pragma autonomous_transaction;
```

Temporary Tables

- Temporary Tables provide a table that is visible to a single transaction or session
- All DML and TRUNCATE TABLE may be used
- Indexes and synonyms may be created for them too
- Temporary Table definitions may be shared by many transactions (ON COMMIT DELETE ROWS) or sessions (ON COMMIT PRESERVE ROWS), but, each transaction or session gets its own copy of the data
- Data is deleted when the transaction or session ends
- Transactions generate UNDO information for Temporary Tables, but, not REDO information

Java

- This topic is covered by many other papers, here is a synopsis
- Oracle8i includes a Java Virtual Machine specifically engineered by Oracle to provide multi-threaded support of Java applications instead of having separate JVMs for each bit of Java
- Oracle also supports the creation of stored procedures using Java
- Java support for programming includes: Java stored procedures, Enterprise Java Bean 1.0 support, and support for CORBA 2.0
- Java support for SQL includes: JDBC and SQLJ. SQLJ statements are translated by an SQLJ Preprocessor before Java code is submitted to JDBC

  Direct JDBC support is more complex, but, yields more control.

Interesting DBA-Oriented Stuff

- Partitioning: Spreads large tables across multiple files/devices predictably good for very large tables
- Reverse-key indexes:Reverses value of keys -- good for keys with tightly-clustered values
- Index-organized tables: Table data stored, in key sequence (in the key) -- makes speedy “lookup”
- ROWID format change: Should not impact most applications
- Direct-path loading from OCI
- ALTER TABLE DROP COLUMN/SET UNUSED
- Oracle8i Release 2 (8.1.6) adds significant new security features

Object Features

- User-defined datatypes
- Encapsulated attributes and methods
- Nested Tables
- Varrays
- Object tables
- Object views
- REF and VALUE functions

User-Defined Datatypes

- Oracle8's Object option provides the capability to begin realizing the promise of object-orientation by making Oracle8 an Object-Relational Database (ORDBMS)
- User-defined datatypes may be as simple as a group data item, or as complex as class-type object

User-Defined Object (Create)

```
create or replace type address as object
( house_number          number(6),
  street1               varchar2(30),
  street2               varchar2(30),
  city                  varchar2(20),
  state                 varchar2(20),
  mailcode              varchar2(15),
  country               varchar2(20)
);
```

User-Defined Object (Use)

```
create table purchase_order
( po_number             number(6) not null primary key,
  customer_name         varchar2(30) not null,
  shipping_address      address,
  billing_address       address
);
```

Encapsulated Attributes & Methods, 1

```
create or replace type cust_order_type as object
( po_number            number(6),
  customer             varchar2(30),
  billing_address      address,
  shipping_address     address,
  order_date           date,
  member function days_old return number,
  pragma restrict_references (days_old,wnds,wnps)
);
```

Encapsulated Attributes & Methods, 2

```
create or replace type body cust_order_type
 as member function days_old
  return number
is
begin
  return sysdate - order_date;
end days_old;
end;
```

Nested Tables (Create Type)

- Associated data to be stored as a unit

```
create or replace type deptemp as object
(EMPNO              NUMBER(4),
 ENAME              VARCHAR2(10),
 JOB                VARCHAR2(9),
 MGR                NUMBER(4),
 HIREDATE           DATE,
 SAL                NUMBER(7,2),
 COMM               NUMBER(7,2)
);
/
create or replace type deptemps as table of deptemp;
/
```

Nested Tables (Create Table)

```
create table department
(deptno        number(2) not null,
 dname         varchar2(15),
 loc           varchar2(20),
 employees     deptemps)
 nested table employees store as emps;
```

Nested Tables (Use)

```
insert into department
    (deptno,dname,loc,employees)
     select deptno,dname,loc,
          cast(multiset(select empno,ename,job,mgr,hiredate,sal,comm
                                        from emp where deptno = dept.deptno
                              )
                       as deptemps
                 )
      from dept;


select empno,ename
    from the (select employees from department where deptno = 20);
```

Varrays (Create Type)

- Varrays might be useful when data occurs a known number of times

```
create or replace type deptemp
as object
(EMPNO          NUMBER(4),
 ENAME          VARCHAR2(10),
 JOB            VARCHAR2(9),
 MGR            NUMBER(4),
 HIREDATE       DATE,
 SAL            NUMBER(7,2),
 COMM           NUMBER(7,2)
);
```

Varrays (Use)

```
create or replace type deptemps as varying array (10) of deptemp;
```

```
create table department
(deptno            number(2) not null,
 dname             varchar2(15),
 loc               varchar2(20),
 employees         deptemps
);
```

Object Tables (Create Type)

- Object tables may be created representing an object type

```
create or replace type emp_type
as object
(EMPNO          NUMBER(4),
 ENAME          VARCHAR2(10),
 JOB            VARCHAR2(9),
 MGR            NUMBER(4),
 HIREDATE       DATE,
 SAL            NUMBER(7,2),
 COMM           NUMBER(7,2)
);
```

Object Tables (Create/Insert)

```
create table my_emps of emp_type
(empno primary key not null, hiredate not null);
```

```
insert into my_emps
  values (emp_type(1234,'WU','NETHERO'
                              ,NULL,sysdate,60000,20000)
              );
```

Object Views

- Views may be based upon an object
- Object Views allow the use of object technology with existing relation tables
- Five steps to creating an Object View using EMP data as a nested table:
  - Define object identically to relational table
  - Define object view using relational table
  - Define new object type and object table
  - Define view using nested table syntax

Object Views:

Define Object

```
create or replace type jempobj as object
  (EMPNO        NUMBER(4),
   ENAME        VARCHAR2(10),
   JOB          VARCHAR2(9),
   MGR          NUMBER(4),
   HIREDATE     DATE,
   SAL          NUMBER(7,2),
   COMM         NUMBER(7,2),
   DEPTNO       NUMBER(2) )
/
```

Object Views:
Define Object View and Object Table

```
create or replace view jemp_obj_view of jempobj
  with object oid (empno) as
  select  empno,
          ename,
          job,
          mgr,
          hiredate,
          sal,
          comm,
          deptno
    from emp
/
```

```
create type jemp_n_table as table of jempobj;
/
```

Object Views:
Define View using Nested Table Syntax

```
create or replace view jemp_o_view (deptno,dname,emptab)
as
 select dept.deptno,dept.dname
        ,cast(multiset
           (select     emp.empno
                      ,emp.ename
                      ,emp.job
                      ,emp.mgr
                      ,emp.hiredate
                      ,emp.sal
                      ,emp.comm
                      ,emp.deptno
              from emp
              where emp.deptno = dept.deptno)
         as jemp_n_table)
   from dept
/
```

Object Views:
Use Relational Table via Object View

```
select ename from
   the  (select emptab from jemp_o_view where deptno = 20)
/
```

- This example shows how Object Views can be created using existing Relational Table data without changing the existing Relational Table in any way
- Your application can use object technology without converting your RDBMS!

REF and VALUE

- REF() get the address of an object in the database
- VALUE() returns an entire object rather than the attributes that make up the object

Conclusion

- Oracle8/8i has many new features of interest to the application developer
- Many important and useful features are available besides the object-oriented and Java features
- Developers can improve applications greatly by incorporating the new features, at the very least, we need to know what is possible so we recognize the new features when they show up on the job
- Oracle8 and Oracle8i are significant improvements to the database, very promising!

King Training Resources

To contact the author:

John King
King Training Resources
6341 South Williams Street
Littleton, CO 80121-2627 USA
1.800.252.0652 - 1.303.798.5727
Email: john@kingtraining.com