Oracle 11g for Developers & DBAs: What You Need to Know

Presented to: Rocky Mountain Oracle Users Group
Training Days 2013

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
- 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)
- (more improvements)

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 Enhancements

- New auto-commit default on “EXIT”
  - Use SET EXITCOMMIT OFF to alter behavior
- Use SET ERRORLogging ON to enable logging of SQL and PL/SQL errors into log table
  - SPERRORLOG default table name
  - SET ERRORL ON TABLE myschema.mytable IDENTIFIER 'mylog’ (may also specify TRUNCATE)
- Use SET ESCCHAR to allow special characters in file names (SET ESCCHAR {@ | ? | % | $ | OFF} before issuing SPOOL, START, @, RUN, or EDIT)
- SQL*Plus now displays CLOB & BFILE data

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)));
```

Virtual Column AS Restrictions

- Output must be a Oracle scalar (single) datatype value or XMLType
- Output may not be LOB, LONG RAW, user-defined type, or Oracle-supplied type
- May not use a sequence
- May only reference columns in the same table
- May not refer to another virtual column
- May use deterministic user-defined functions; but then the virtual column cannot be used as partitioning key column
- Virtual Column AS definitions ARE NOT copied when a CREATE TABLE AS SELECT ... is executed; the current column value is copied to the new table

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)
  – Outputs of all grouping columns followed by new columns generated by PIVOT
  – Requires a subquery or CTE data source
- UNPIVOT performs similar 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
```

PIVOT Multiple Columns

```
select * from (select
    ename,hiredate,job,deptno,income from newemp) qry1
  pivot (avg(income)
  for deptno
  in ( 10 AS ACCOUNTING,
       20 AS RESEARCH,
       30 AS SALES))
  order by job

ENAME HIREDATE   JOB     ACCOUNTING  RESEARCH     SALES
FORD   03-DEC-01 ANALYST             30000
SCOTT  09-DEC-02 ANALYST             30000
SMITH  17-DEC-00 CLERK               8000
JAMES  03-DEC-01 CLERK                            9500
MILLER 23-JAN-02 CLERK     13000
```

PIVOT Multiple Aggregates

```
select * from
(select job,deptno,income from newemp) query1
   pivot (avg(income) avg ,max(income) max
   for deptno
   in ( 10 AS ACCTG,
        20 AS RSRCH))
   order by job

JOB          ACCTG_AVG ACCTG_MAX RSRCH_AVG RSRCH_MAX
ANALYST                          30000     30000
CLERK        13000     13000      9500     11000
MANAGER      24500     24500     29750     29750
PRESIDENT    50000     50000
SALESMAN
```

PIVOT XML

```
select * from
   (select job,deptno,income from newemp) query1
     pivot xml (avg(income) incomes
     for deptno
     in ( ANY))
     order by job
JOB       DEPTNO_XML
ANALYST   <PivotSet><item><column name = "DEPTNO">20</
column><column name =
"INCOMES">30000</column></item></PivotSet>
CLERK
<PivotSet><item><column name = "DEPTNO">10</column><column name =
"INCOMES">13000</column></item><item><column name = "DEPTNO">20</
column><column
name = "INCOMES">9500</column></item><item><column name =
"DEPTNO">30</column><column name = "INCOMES">9500</column></item></
PivotSet>
```

Simple Use of UNPIVOT

```
select empno, colname || '=' || colval value
  from emp
  unpivot include nulls
   (colval
    for (colname)
    in (ENAME AS 'ENAME', JOB AS 'JOB')
   )
  order by empno,colname

EMPNO VALUE
7369  ENAME=SMITH
7369  JOB=CLERK
** more **
```

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
  (when data changes, data automatically refreshes)
- 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
```

Results Cache Parameters

- result_cache_mode
  - Result cache may be enabled three ways: hint, alter session or alter system (def. manual)
- result_cache_max_size
  - Size of result cache allocated from shared pool (maint separately not flushed with shared pool)
- result_cache_max_result
  - Highest percentage of result cache that may be used by a single result set (default is 5%)
- result_cache_remote_expiration
  - Number of minutes result cache resultset based upon a remote object is considered valid

Result Cache Catalog Support

- V_$RESULT_CACHE_DEPENDENCY
- V_$RESULT_CACHE_MEMORY
- V_$RESULT_CACHE_OBJECTS
- V_$RESULT_CACHE_STATISTICS
- V$CLIENT_RESULT_CACHE_STATS

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
 1
 Blast Off!
```

<-Values “4” and “2” do not appear in the output

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 Common Table Expressions

- Oracle has allowed hierarchical queries using CONNECT BY syntax for many years
- SQL 2003 introduced an ISO syntax for recursive SQL allowing hierarchical queries and more
- Common Table Expressions (CTEs) were first added to Oracle in Oracle 9i via the WITH clause
- Recursive CTEs use base queries called "anchor members" and "recursive members"
- Anchor members generate rows of base data
- Recursive members process anchor member data which is then reprocessed recursively until no rows are produced

Simple Recursive Subquery

```
WITH CountDown(N) AS
( select 10 as N from dual
  UNION ALL
  select N - 1 AS N
   from CountDown
   where N > 0
)
SELECT * FROM CountDown;
```

- 1st query is anchor; generates the value of 10
- 2nd query uses the CTE name (CountDown) to process output of the previous execution
  - Each 2nd query recursion processes previous output
  - Recursion stops when no rows satisfy the 2nd query

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
```

Recursive CTE vs CONNECT BY

- Are Recursive Common Table Expressions just a more-complex way to do CONNECT BY?
- Recursive CTE is more powerful
  - Comparisons may be more-complex than simple parent-child relationships
  - Looping may be controlled
  - Values may be calculated or expressions may be used to modify data
- Some CONNECT BY features are difficult to do (notably CONNECT_BY_ISLEAF)
- One Oracle ACE (Iggy Fernandez) has a script to solve Sudoku puzzles using Recursive CTEs

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,…)
- With Oracle 11gR2 the EXECUTE privilege may be granted for Directory objects

External Directory Syntax

```
CREATE TABLE …
...
ORGANIZATION EXTERNAL
( TYPE ORACLE_LOADER
  DEFAULT DIRECTORY MY_EXTERN_DIR
  ACCESS PARAMETERS
  ( RECORDS DELIMITED BY NEWLINE
    PREPROCESSOR MY_EXTERN_DIR2:'execute_unzip_exec.sh'
      ...
  )
  LOCATION (MY_EXTERN_DIR:'my_table_dat.gz')
)
…
```

Data Pump Improvements

- Oracle provides “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
- Data Pump support more compression options
  - METDATA_ONLY (same as 10gR2)
  - DATA_ONLY (data compressed, not metadata)
  - ALL (both data and metadata compressed)
  - NONE (default)
- Disable direct-path (if necessary)

```
impdp … data_options=disable_append_hint
```

11g Partitioning Improvements

- Partitioning is key functionality for managing large volumes of data
- Oracle 11g adds new capabilities including:
  - Interval Partitioning
  - Reference Partitioning
  - Virtual Column Partitioning
  - Extensions to Composite Partitioning

Interval Partitioning

- Oracle 11g Interval Partitioning adds the exciting capability of having partitions be defined automatically as they are needed
- A drawback to partitioning by something like dates is that additional management is required to create new partitions for each new period
- Beginning with Oracle 11g Interval Partitioning will create partitions as needed to hold inserted data (real-time; no manual intervention needed)

Interval Partitioning Syntax

```
CREATE TABLE service_orders (so_date DATE, ...)
 PARTITION BY RANGE (so_date)
 INTERVAL(NUMTOYMINTERVAL(1,'month')
 (PARTITION p_first VALUES LESS THAN ('01-JAN-2008');
```

- Data inserts that are outside the bounds of existing partitions will cause creation of the necessary new partition

```
INSERT INTO service_orders (so_date DATE, ...)
VALUES ('06-MAR-2008',...);
```

Reference Partitioning

- It is sometimes useful to use the same partitioning strategy for multiple tables
- Multiple tables partitioned in the same way cause overhead due to redundancy
- Oracle 11g REF partitioning (Reference Partitioning) allows a child table to inherit the partitioning strategy of the parent through the Primary Key - Foreign Key mechanism
- REF partitioning allows more-intuitive functionality that provides better performance and greater maintainability

Reference Partitioning Syntax

```
CREATE TABLE SALES_ORDERS
( SO_ID NUMBER NOT NULL,  ...
  CONSTRAINT PK_SO_ID PRIMARY KEY (SO_ID)
  ...
  PARTITION BY RANGE (so_id) ( ... )
  ...
  );
CREATE TABLE SALES_ORDER_LINES
(... SO_ID NUMBER, ...
 CONSTRAINT FK_SOL_SO_ID FOREIGN KEY (SO_ID)
 REFERENCES SALES_ORDERS (SO_ID)
 ...
 PARTITION BY REFERENCE (FK_SOL_SO_ID)
 ...
);
```

Virtual Column Partitioning

- First, Oracle 11g added Virtual Columns
- Then, Oracle 11g added the ability to use virtual columns as partitioning keys

```
CREATE TABLE dda_accounts
(
 dda_account_no number(12) not null,
 dda_account_name varchar2(40) not null,
 ...
 dda_account_branch number(4) generated always as
 (to_number(substr(to_char(dda_account_no),1,2)))
 partition by list (dda_account_branch)
 ...
)
```

Compost Partitioning Extensions

- 11g allows “Composite Partitioning” based upon two dimensions
- Partitioning is determined using a distinct value pair for the two dimensions
- Composite partitioning may be used to complement multi-column range partitioning
- Composite Partitioning may create pairs of:
  - List – Range
  - Range – Range
  - List – Hash
  - List - List

Misc. Tablespace Changes

- Command available to shrink temporary tablespace online

```
ALTER TABLESPACE TEMPSPACE SHRINK SPACE
```

- Specify tablespace when creating global temporary tables

```
CREATE GLOBAL TEMPORARY TABLE XXX
(
  . . .
)
  TABLESPACE mygtempspace;
```

11gR2 Java Enhancements

- Oracle’s Java Virtual Machine (JVM), Java debugger, and JDBC library have been enhanced to include IPv6 compatability
- Java API for Oracle Georaster datatype
- Note: The JDBC driver package is now oracle.jdbc; oracle.jdbc.driver package has been deprecated use **oracle.jdbc.OracleDriver** driver class

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?

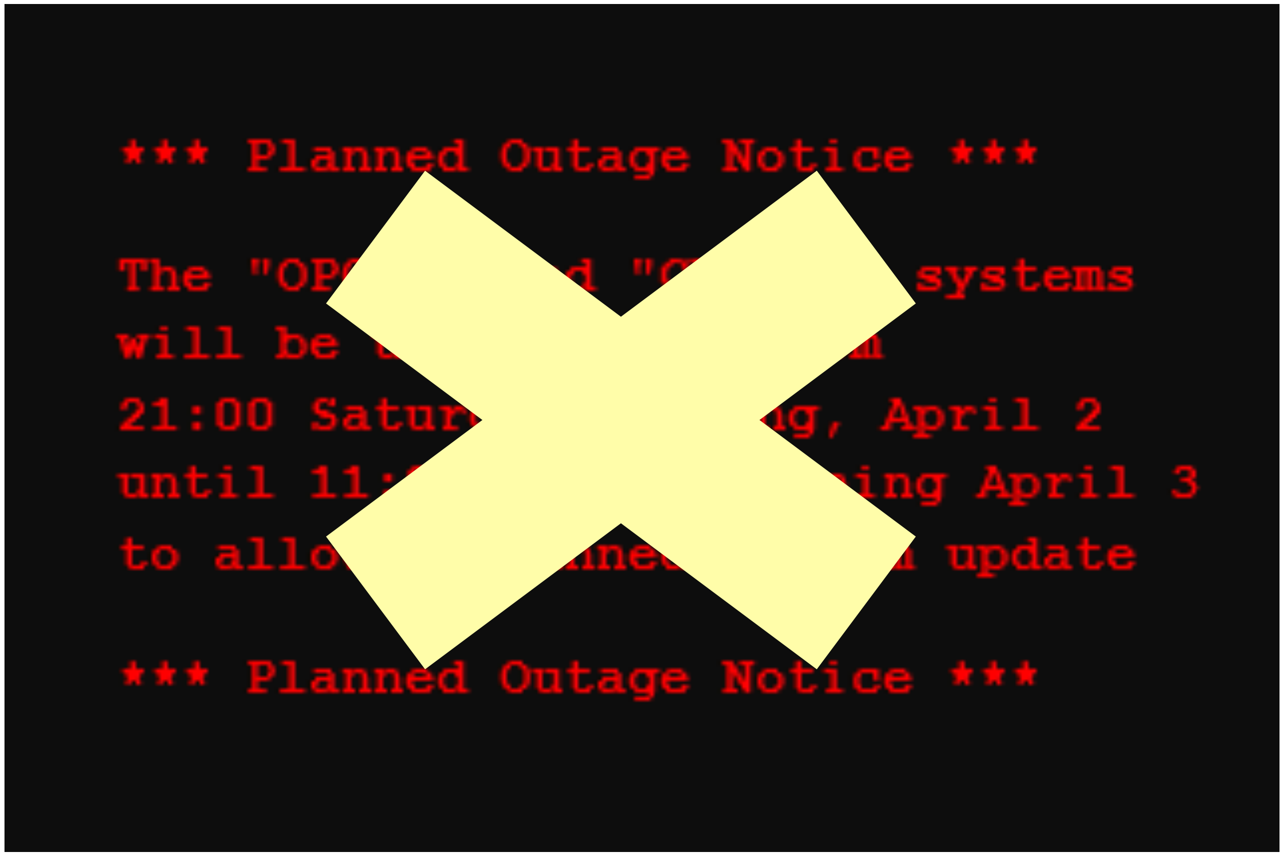
- Oracle 11gR2 Edition-Based Redefinition adds a new non-schema "edition" of an application including all of the original edition's PL/SQL, views, and synonyms; the new edition may be modified as desired then tested and deployed without impacting the users of the original edition
- Once the new edition 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

Planned Outages

"Can't live with them
- Can't live without them"

But…

- What if you could drastically reduce the downtime outages require?



*** Planned Outage Notice ***

The "OP d "C systems
will be m
21:00 Satur g, April 2
until 11: ing April 3
to allo ne update

*** Planned Outage Notice ***

Online Application Upgrade

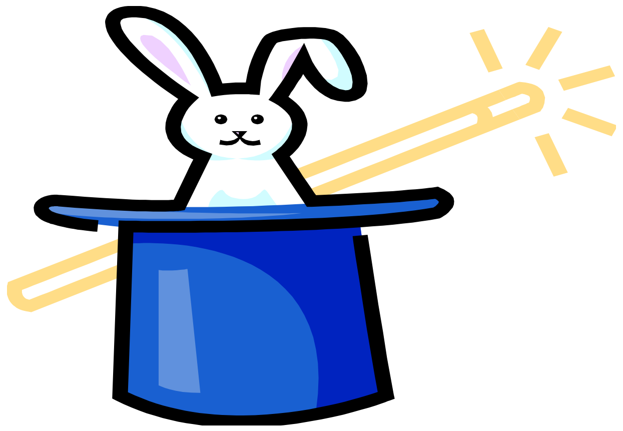
- The quest to eliminate downtime has led to a desire to provide "Online Application Upgrade"
  - An application need not be taken down when upgrades are applied
  - Users of the existing system continue uninterrupted
  - Users of the upgraded system use new code immediately

EBR to the Rescue!

- Edition-Based Redefinition (EBR) has been described as the “killer feature” of 11gR2
- EBR provides a revolutionary ability to manage change of stored PL/SQL
  - Applications need not be taken down when upgrades/changes are applied
  - Eliminating downtime required to upgrade

Is EBR Magic?

- It just seems like it!
- Oracle 11gR2's Edition-Based Redefinition uses a non-schema "edition" of an application; including PL/SQL, views, and synonyms
  - A new edition may be created without impacting users of the current edition
  - New editions may be modified as desired then tested and deployed; again without impacting users of the original edition
  - When a new edition is ready for complete rollout it may be released to all users

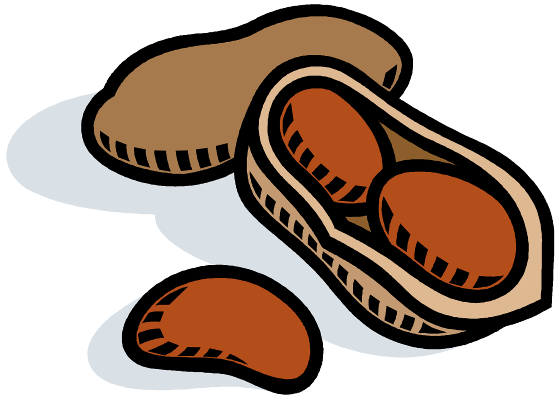


How?

- Administrators install, alter, compile, and verify Stored Procedures, Triggers, Grants, and Views (PL/SQL) using new (child) edition
- Existing systems continue uninterrupted with old (parent) edition
- Early adopters use new (child) code base others use existing (parent) code
- All users have access to upgraded system immediately after set by administrator

In a Nutshell

- EBR provides
  - Insulation mechanism where changes are made in privacy to the post-upgrade edition
  - Control mechanism allows users to actively use different editions
  - Support for “Hot Rollover” providing high availability



What is EBR?

- EBR provides a revolutionary ability to manage changes to “editionable objects”
- Editionable object types:
  – PL/SQL objects of all kinds
  – Synonyms
  – Views

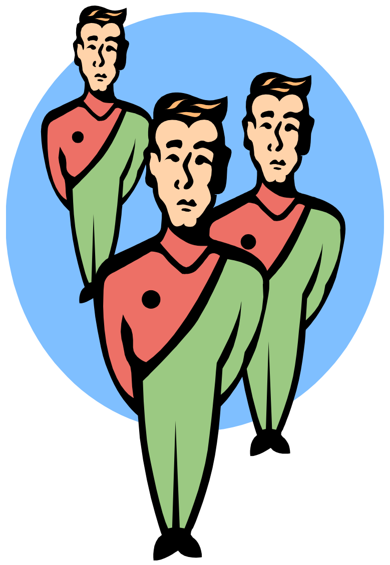
EBR Support

- EBR is safe, secure, and part of Oracle 11g (both EE and SE)
- EBR is built-in and is without additional cost
- EBR may require considerable design investment to work well

Oracle E-Business Suite 12.2 uses EBR to drastically reduce planned outages

How?

- Non-schema "edition" of a database's PL/SQL, views, and synonyms
  - New edition may be modified as desired then tested and deployed without impacting users of the original edition
  - Once the new edition is ready for complete rollout, it may be released



New 11g Objects

- Editions
  - All pre-upgrade editionable objects exist in a parent edition
  - New editions inherit from parent
  - Post-upgrade objects exist in the child edition
- Editioning Views
  - Different projection of table for each edition
  - Each edition sees only its own columns
  - Changes are made safely writing only to new columns or new tables not seen in old edition
- Cross-Edition Triggers
  - Keep "old" and "new" editions synchronized

EBR Edition

- Non-schema objects that have no owner (says SYS in dba_objects but not “owned”)
- Editions are part of 11gR2; used or not
- Name of default edition is ORA$BASE
- Database starts using single edition
- Each new edition must be the child of an existing edition

Parent-Child Editions

- Parent edition

  Represents objects prior to changes (pre-upgrade)

- Child edition

  Represents objects after changes (post-upgrade)

Object Identification

- Oracle 11g R2 (and beyond) database objects are identified as:

  edition_name.schema_name.object_name

- Before Oracle 11g R2 database objects were identified as:

  schema_name.object_name

Need for Editioning Views

- If only PL/SQL is changing from edition to edition; Editioning Views are not required
- If Tables might have column definitions added, removed, or altered between editions; then, Editioning Views are necessary to make sure each Edition's users see only relevant data
- If Editioning Views will be created and it is desirable to execute multiple editions in production; Cross-Edition Triggers may needed to keep data synchronized

EBR Editioning Views

- Represent projected data from an underlying table (unaltered and unfiltered)
- May not use: Joins, Functions, Operators, Group By, Order By, or Distinct (or anything causing view to misrepresent the data)
- Act like tables and may have triggers and other table-like features
- Are referenced by ALL application code rather than the base tables; applications “think” the Editioning View is the base table

Editioning View Performance

- Since Editioning Views merely PROJECT column data; the optimizer converts all activity to use the underlying table
  - SQL referencing Editioning Views will get EXACTLY the same execution plan as SQL using the base table
  - There is no additional performance cost (other than statement parsing) involved with Editioning Views
  - Forward and reverse Cross-Edition triggers will have some performance impact

EBR Cross-Edition Trigger

- Used to propagate changes between editions
  - Changes in Parent propagated to Child (Forward)
  - Changes in Child propagated to Parent (Reverse)
- Cross-Edition Triggers allow shops to make changes like adding columns or changing table definitions without causing an outage
- Cross-Edition Triggers are temporary

Oracle Catalog Support

- v$session (session_edition_id column)
- dba_editions
- dba_edition_comments
- dba_editioning_views
- dba_editioning_views_ae
- dba_editioning_view_cols
- dba_editioning_view_cols_ae
- dba_triggers
- dba_objects (edition_name column)
- more...

Putting EBR Together

- Cross-Edition Triggers allow shops to make changes like adding columns or changing table definitions without causing an outage
- In the past, some changes (e.g. alter column datatype) would cascade through an application and bringing the change to production would require locking table data and shutting down applications to propagate the change

Note: Oracle re-engineered Alter Table DDL to make most column additions non-blocking and improved dependency-tracking to allow “fine-grain dependency” in support of EBR

EBR in Use

- Current edition's (version1/release1) views represent table data
- A new edition (version2/release2) is created building new Editioning Views in the schema; Editioning View references the new/changed columns using the old column names
- Cross-edition triggers fire in one edition (or the other, but only one edition); typically changes to the current edition's data will be copied/forwarded to the new version

EBR and Triggers

- Rows/changed added in the current version will forward new data into the new version; the **dbms_parallel_execute** package (11g R2) is used to make changes in smaller chunks limiting locking issues
- Updates to the current version/release fire Cross-Edition triggers forwarding all changes to the new version/release
- Both current version/release and the new version/ release are running simultaneously

Managing Transition

- Changes may be installed, verified, compiled, and validated without impacting the current system
- When ready; administrator may "cut over" to the new version/release by simply setting the edition for users

EBR and Table Modifications

- Post-Upgrade edition processing must not interfere with any Pre-Upgrade edition processing
- Column datatype change requires:
  - Creating a new column in the base table so that both sets of code are still valid
  - Creating Cross-Edition triggers to keep values “in-sync” if pre-upgrade and post-upgrade editions will be in use at the same time
  - Populating new column values

EBR Access

- Access to editions is user based
- To allow a user access to enable editions

```
alter user someschema enable editions
```

(dba_users.editions_enabled=Y or N)

- To create a new edition and make it available to a user

```
create edition yyy as child of xxx
-- Requires CREATE ANY EDITION privilege
grant use on edition yyy to someuser
```

EBR Planning and Readying

- EBR use should be planned carefully
- When Editioning Views will be used a “readying” process is followed:
  - Tables renamed and then replaced using Editioning Views with the original table name
  - Drop triggers from base tables
  - Create triggers on editioning views
  - Recompile PL/SQL and Views using tables
  - then test, test, and test some more

Editioning Rules

- A schema object of an editionable type is editioned if its owner is editions-enabled; otherwise, it is potentially editioned
- A schema object of a noneditionable type is always noneditioned, even if its owner is editions-enabled
- Non-editioned objects may not depend upon editioned objects
- Editioning Views must have same owner as base table

“NE on E” Prohibition

- A simple concept keeps data making sense

Non-Editioned objects may not depend upon Editioned objects

Non-Editioned Item Issues, 1

- Public Synonyms cannot refer to editioned objects
- Function-based Indexes cannot depend upon editioned functions
- Materialized Views cannot refer to editioned objects

Non-Editioned Item Issues, 2

- Tables cannot have columns based upon user-defined types (collection/ADT) whose owner is editions-enabled
  (Editioned ADTs may not be evolved)
- Non-editioned subprograms cannot reference an editioned subprogram
- Editioning Views may not be part of Foreign Key definitions

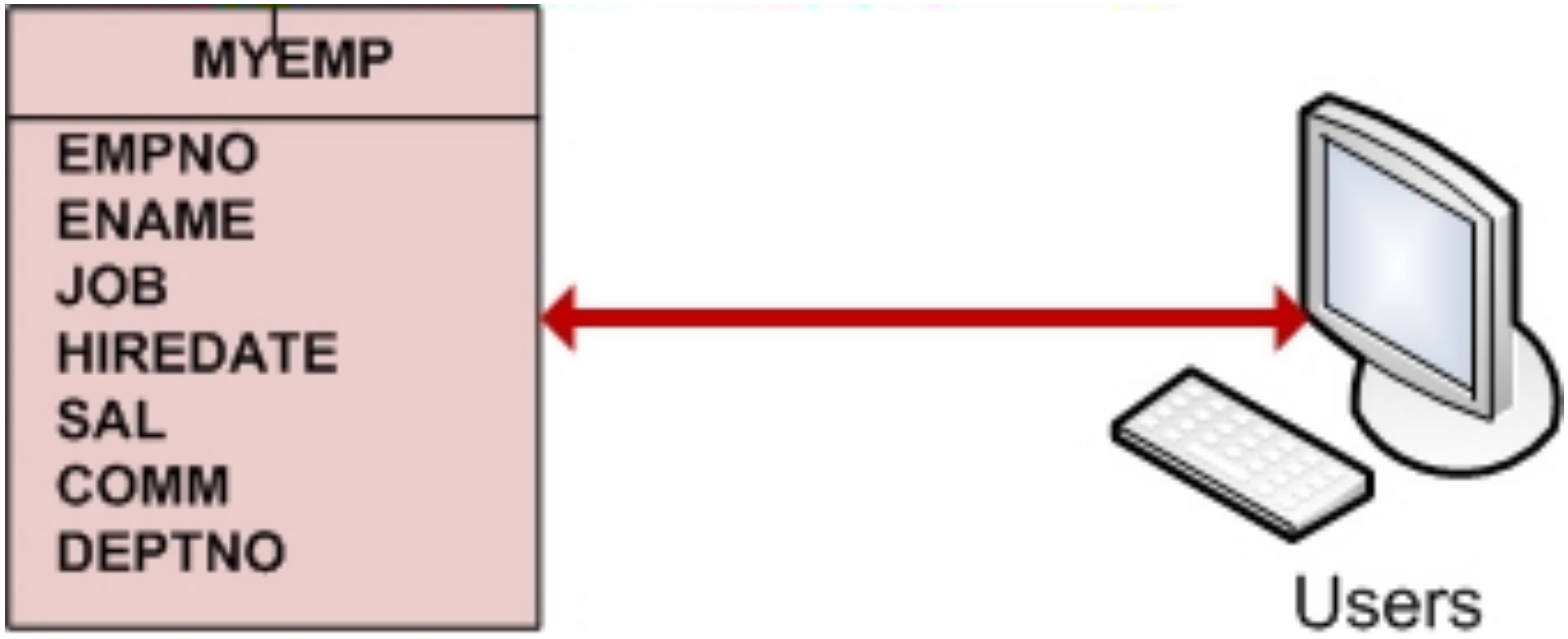
User Grants

```
GRANT USE ON EDITION xxx
     TO USER
```

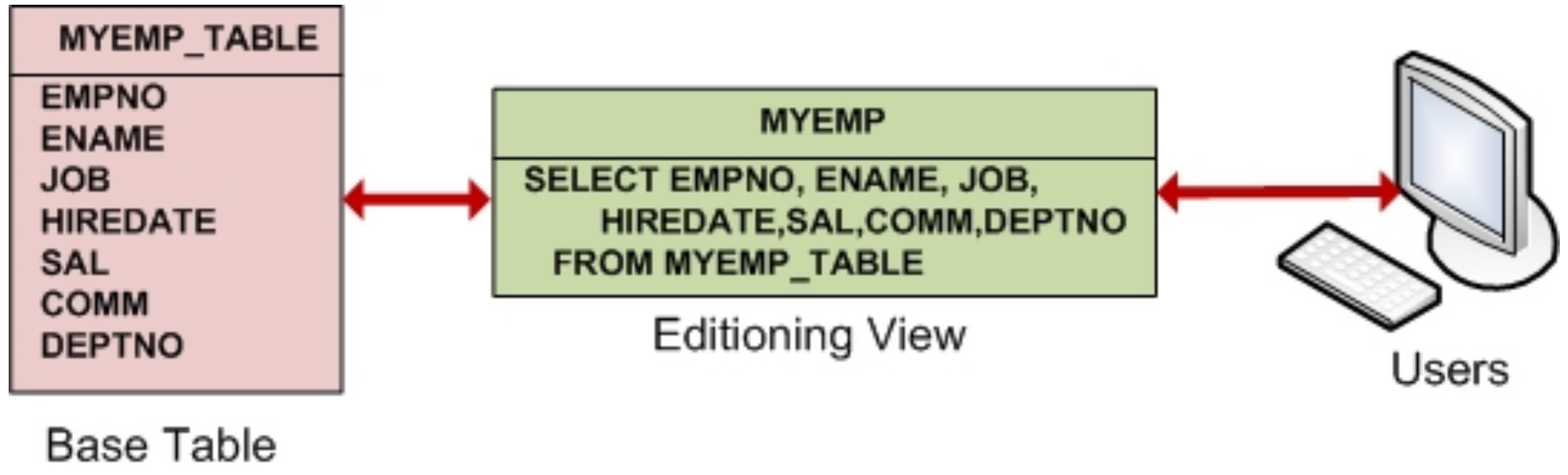
```
GRANT USE ON EDITION xxx
     TO PUBLIC
```

– (automatically performed by ALTER DATABASE SET DEFAULT EDITION)

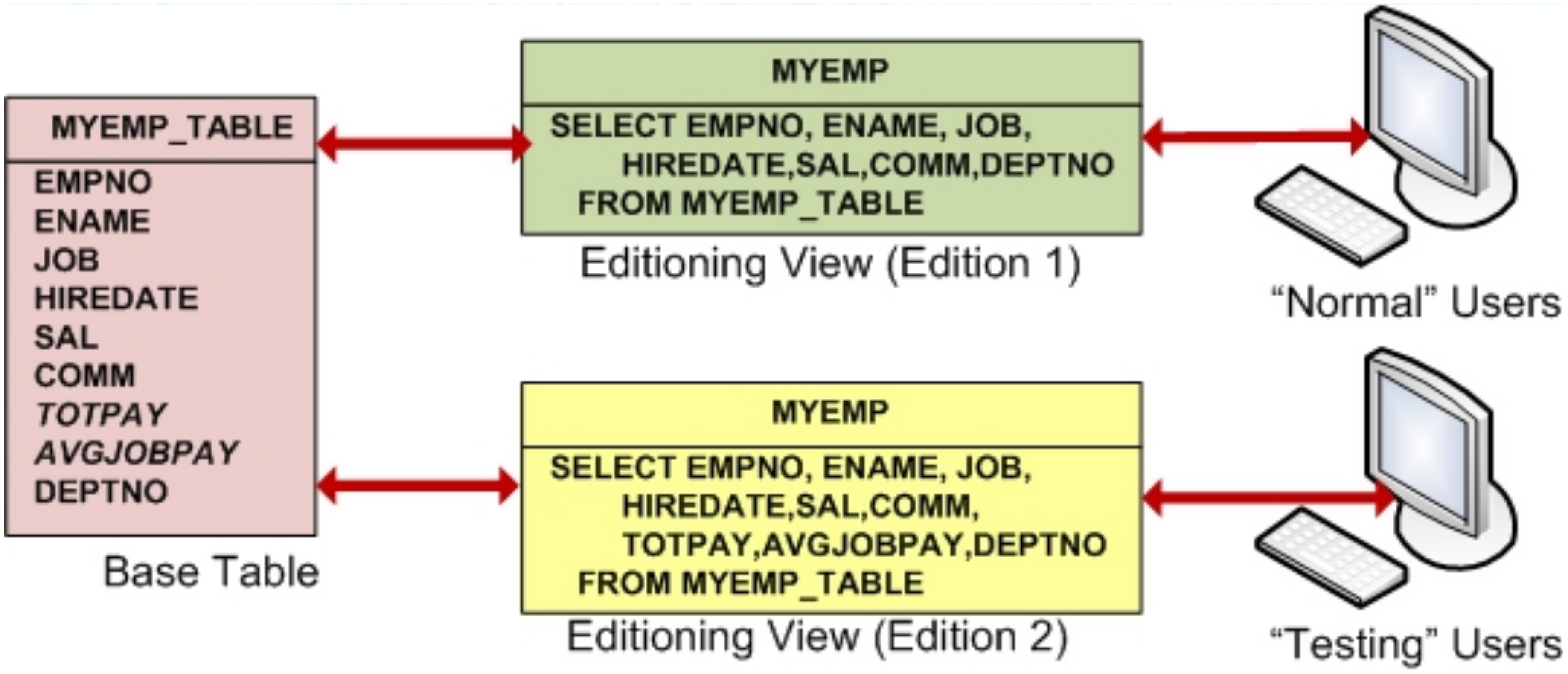
EBR – Not Used Yet



EBR – Edition 1



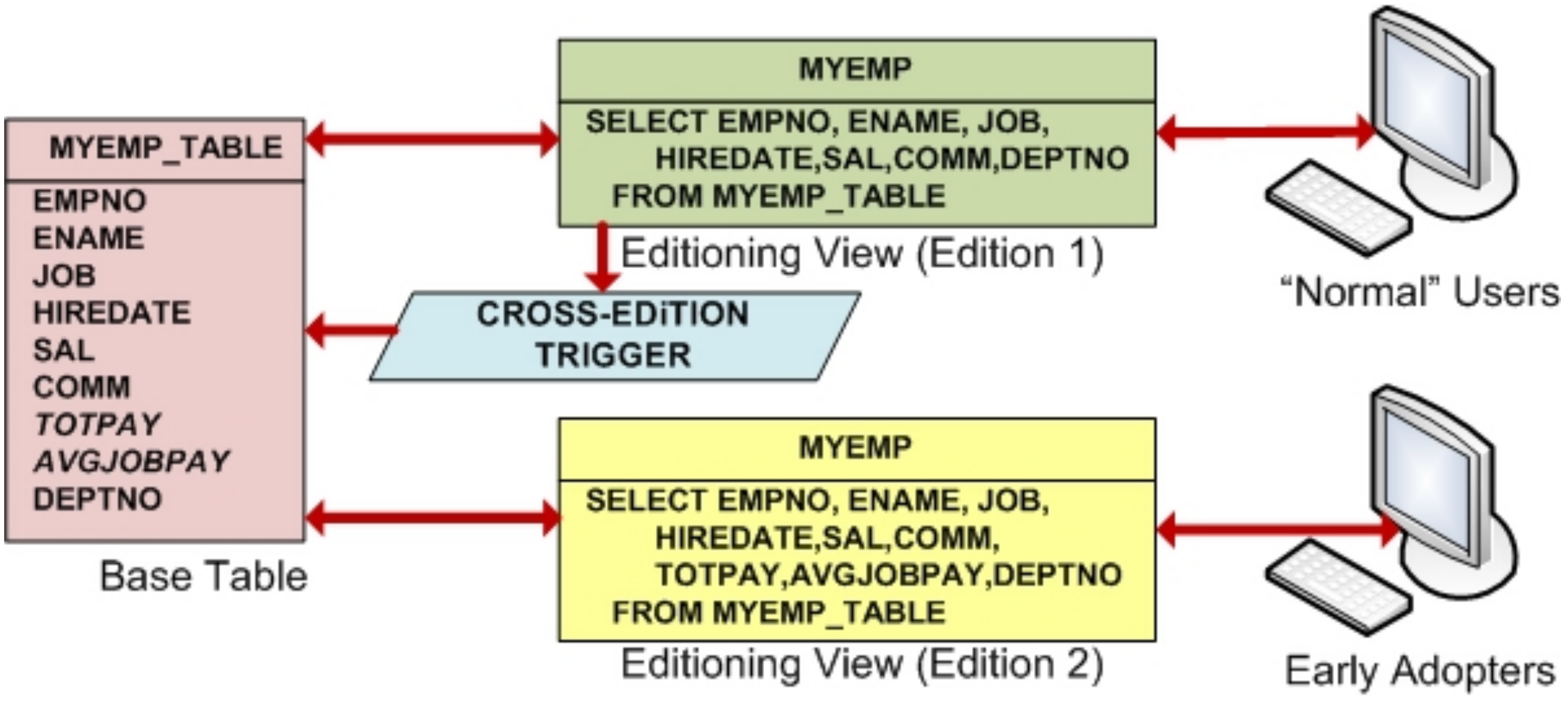
EBR – Edition 2



```
SELECT EMPNO, ENAME, JOB,
       HIREDATE,SAL,COMM,DEPTNO
  FROM MYEMP_TABLE
```

```
SELECT EMPNO, ENAME, JOB,
       HIREDATE,SAL,COMM,
       TOTPAY,AVGJOBPAY,DEPTNO
  FROM MYEMP_TABLE
```

EBR – Edition 2 + Trigger



Planning for EBR Adoption

- Complexity of implementation matters!

PL/SQL Only	Pretty easy, not much planning required
Editioning Views representing table data but only one version in use at a time	More complex; requires planning of table-view creation, miscellaneous changes
Editioning Views represent tables with simultaneous changes allowed; requiring use of Editioning Triggers	Much more complex; testing must be planned thoroughly; probably requires schedule for retirement of Editioning Triggers

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**

Here Comes Oracle 12c !

- Sometime in the near future Oracle will release the Oracle 12c database
  - I will discuss some of Oracle 12c's new features but will not be able to demo them
  - Any information I have has been shared by Oracle at Oracle Open World 2012 and UKOUG 2012; all of it is subject to change and may be inaccurate
- CONSULT THE RELEASE DOCUMENTATION WHEN IT BECOMES AVAILABLE !!!

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

RMOUG

ROCKY MOUNTAIN ORACLE USERS GROUP

Training Days 2014

2 Days of inexpensive Oracle-related training in Denver !!

February 11-12

February 10: University day: More low-cost training!
Check the website for details

www.rmoug.org

See you in Denver Colorado!

COLLABORATE 13

TECHNOLOGY AND APPLICATIONS FORUM
FOR THE ORACLE COMMUNITY

April 2013 – Get Ready to Go!

ODTUG Kscope13

NEW ORLEANS, LA ⚜ JUNE 23-27

Topics

- Application Express
- ADF and Fusion Development
- Developer's Toolkit
- The Database
- Building Better Software
- Business Intelligence
- Essbase
- Planning
- Financial Close
- EPM Reporting
- EPM Foundations & Data Management
- EPM Business Content

ODTUG - Leading Developers into the Future with Oracle Tools

www.kscope13.com

King Training Resources

Oracle 11g for Developers & DBAs: What You Need to Know

To contact the author:

John King

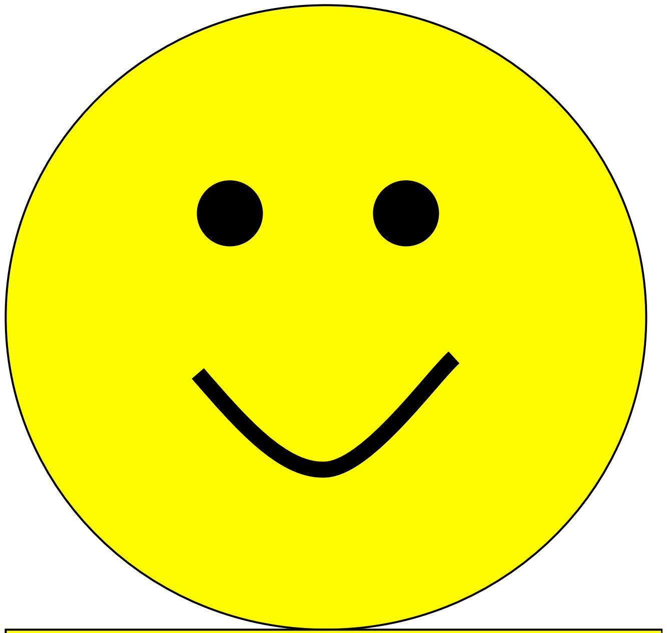
King Training Resources

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