



# ***Oracle 12c and Oracle 11gR2 New Features For Developers***



**Presented by: John Jay King**

**Download this paper from: <http://www.kingtraining.com>**



- Learn new Oracle 12c and Oracle 11gR2 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



- John King – Partner, King Training Resources
- Oracle Ace Director 
- Member Oak Table Network 
- Providing training to Oracle and IT community for over 25 years – <http://www.kingtraining.com>
- “Techie” who knows Oracle, ADF, 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
- Charter member of IOUG



- Environment changes
- New/improved SQL & PL/SQL statements
- SQL & PL/SQL Results Caches
- Java, JDBC, and SQLJ improvements
- New Analytic (and other) Functions
- Java and XML Enhancements
- Pro\*C/Pro\*COBOL & OCI Enhancements
- Edition-Based Redefinition (EBR)



- Oracle 11g R1 August 2007
- Oracle 11g R2 September 2009
- Oracle 12c R1 June 2013



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

- Multi-tenant Architecture:  
(first architecture change to Oracle since V6 in 1988!)
  - Container Database (CDB)
  - Pluggable Database(s) (PDB)
- Performance Improvements:
  - Improved optimization
  - Enhanced Statistics & New Histograms
  - Adaptive Execution Plans
- More cool stuff (watch OOW announcements...)

*DBA Stuff is not covered further in this presentation  
(lots of other presentations though)*



- SELECT improvements: Top-n & Pagination, pattern matching, outer join improvements
- Table definition improvements: expanded columns, identity columns, default improvements, invisible columns
- PL/SQL in WITH clause
- Temporal Validity
- Online DML operations
- Truncate CASCADE
- EBR improvements



- 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!



```
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



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

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



- 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



```
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	



```
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



- 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 index mytab_ix invisible;
```

```
alter index mytab_ix visible;
```



- 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



```
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



```
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 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



```
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;
```



```
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



- 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



- 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 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)



- 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



```
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;
/
```



- 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



- 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)



```
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**



- 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



```
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
```



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

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



- 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 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))



- 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)
```



- 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 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



- 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,lv1)
  as (select last_name, employee_id, manager_id, 1 lv12
      from hr.employees where manager_id is null
      union all
      select emp.last_name, emp.employee_id,
             emp.manager_id, ec.lv1+1
      from hr.employees emp, empConnect ec
      where emp.manager_id = ec.employee_id)
  SEARCH DEPTH FIRST BY last_name SET order_by
select lv1,lpad(' ',3*lv1, ' ')||last_name empname
from empConnect
order by order_by
```



- 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,...)



- 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



- 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



- 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



- 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.



- 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



- 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



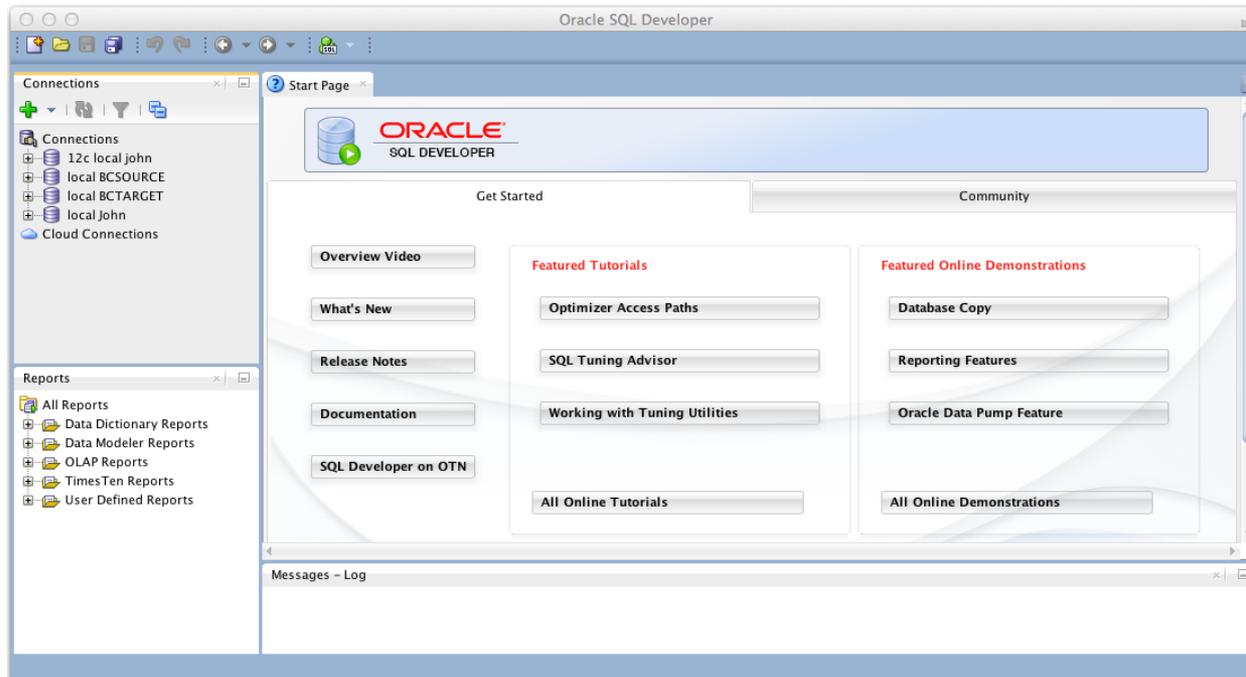
- Oracle 11gR2 Edition-Based Redefinition adds a new non-schema "edition" of an application including all of the original edition's PL/SQL, views, and synonyms; the new edition may be modified as desired then tested and deployed without impacting the users of the original edition
- Once the new edition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



- SELECT improvements: Top-n & Pagination, pattern matching, outer join improvements
- Table definition improvements: expanded columns, identity columns, default improvements, invisible columns
- PL/SQL in WITH clause
- Temporal Validity
- Online DML operations
- Truncate CASCADE
- EBR improvements



- Oracle SQL Developer 4.0 Early Adopter 2 is now available for download
- Many new features & supports Oracle 12c (still a couple of “wrinkles” in Early Adopter release ...)





- Oracle 12c adds “top-n” type queries and paginated queries
  - FETCH FIRST/LAST nn ROWS  
FIRST/LAST n PERCENT ROWS
  - OFFSET nn ROWS
- Optimizer uses analytics under the covers to make this work



- Original query uses “rownum” – note sequence of data

```
select ename,sal from emp
       where rownum < 5 order by sal desc;
```

--

<b>ENAME</b>	<b>SAL</b>
JONES	2975
ALLEN	1600
WARD	1250
SMITH	800



- Here the first five rows (by value) are selected; note no need for analytics

```
select ename,sal from emp
       order by sal desc
       fetch first 5 rows only;
```

<b>ENAME</b>	<b>SAL</b>
-----	-----
KING	5000
SCOTT	3000
FORD	3000
JONES	2975
BLAKE	2850



- The OFFSET clause may start processing at a given row; when (optionally) paired with FETCH allows pagination in query

```
select ename,sal from emp
order by sal desc
offset 2 rows
fetch first 5 rows only;
```

<b>ENAME</b>	<b>SAL</b>
-----	-----
<b>FORD</b>	<b>3000</b>
<b>JONES</b>	<b>2975</b>
<b>BLAKE</b>	<b>2850</b>
<b>CLARK</b>	<b>2450</b>
<b>ALLEN</b>	<b>1600</b>



- Top-N may use a percentage rather than a number of rows

```
select ename,sal from emp
order by sal desc
offset 2 rows
fetch first 5 percent rows only;
```

<b>ENAME</b>	<b>SAL</b>
-----	-----
SCOTT	3000



- Enhanced ability to use Regular Expressions enabled by Oracle 12c's MATCH\_RECOGNIZE
- Using syntax similar to the MODEL clause and Analytics rows may be compared to other rows using Regular Expressions (beyond capabilities of LAG/LEAD)



- MATCH\_RECOGNIZE includes:
  - PARTITION Segregate data
  - ORDER BY Order with partitions
  - MEASURES Define output columns
  - AFTER Return single/multiple rows
  - PATTERN Define regular expression
  - DEFINE Specify expression tags



- The code on the following pages creates a report illustrating sales patterns for a specific product over time



- SELECT uses query in from clause to aggregate SH.SALES data by prod\_id and day (truncated time\_id)

```
select * from
(select prod_id, trunc(time_id) time_id,
sum(amount_sold) amount_sold from sh.sales
where prod_id = 148
and extract(year from time_id) in (2000,2001)
group by prod_id, trunc(time_id))
```



```
match_recognize (  
  partition by prod_id  
  order by time_id  
  measures to_char(strt.time_id,'yyyy-mm-dd') as  
start_date,  
  to_char(last(down.time_id),'yyyy-mm-dd') as bottom_date,  
  to_char(last(up.time_id) ,'yyyy-mm-dd') as end_date,  
  last(round(down.amount_sold)) as bottom_amt,  
  last(round(up.amount_sold)) as end_amt  
  --one row per match  
  after match skip to last up  
  pattern (strt down+ up+)  
  define  
    down as down.amount_sold < prev(down.amount_sold) ,  
    up as up.amount_sold > prev(up.amount_sold)  
  ) matcher  
order by matcher.prod_id, matcher.start_date
```



- Here are the results and a sample of the data to see what happened
- Two result rows:

148	2000-01-18	2000-01-23	2000-01-27	1191	1333
148	2000-01-27	2000-02-02	2000-02-14	887	2148

- Matching base data rows:

148	18-JAN-00	2229
148	23-JAN-00	1191
148	27-JAN-00	1333
148	02-FEB-00	887
148	14-FEB-00	2148



- Oracle 12c expands the use of the “traditional” Oracle Outer Join syntax (+) to make it more useful
- The (+) notation to create null rows may now be used for multiple tables & columns



```
select region_name, country_name, department_name, city,
count(employee_id) nbr_emps
  from hr.regions r, hr.countries c, hr.locations l,
       hr.departments d, hr.employees e
 where r.region_id = c.region_id(+)
       and c.country_id = l.country_id(+)
       and l.location_id = d.location_id(+)
       and d.department_id = e.department_id(+)
 group by region_name, country_name, department_name, city
 order by region_name, country_name, department_name, city
```



- Oracle 12c adds the ability to JOIN values in a generated table collection to regular tables using:
  - CROSS APPLY                      Join table to generated collection when values match
  - OUTER APPLY                      Join table to generated collection when values match and create matches for non-match rows too



```
create or replace type name_table_type
    as table of varchar2(100);
create or replace function department_employees
    (in_department_id varchar2)
    return name_table_type
is
    mynames name_table_type;
begin
    select cast(collect(last_name || ', ' || first_name)
        as name_table_type)
        into mynames
    from hr.employees
    where department_id = in_department_id;
    return mynames;
end;
/
```



```
select *  
  from hr.departments d  
      cross apply  
      department_employees(d.department_id) dept_emps;
```

```
select *  
  from hr.departments d  
      outer apply  
      department_employees(d.department_id) dept_emps;
```

```
select department_name  
      ,department_employees(department_id) deptemps  
  from hr.departments;
```



- Oracle 12c increases the maximum size of three datatypes to 32,767 (4,000 before)
  - VARCHAR2
  - NVARCHAR2
  - RAW
- Not default: DBA must set init.ora  
max\_sql\_string\_size EXTENDED
- Stored out of line as CLOB when > 4k
- Now matches PL/SQL variables



- Oracle has had SEQUENCES for years; the IDENTITY column allows use of a SEQUENCE as part of a column definition (much like some competitor databases)
  - Use “GENERATED AS IDENTITY” clause
  - Default starts with 1 increments by 1
  - May set values using START WITH and INCREMENT BY



```
create table id_test1
(id number generated as identity,
 col1 varchar2(10));
--
insert into id_test1 (col1) values ('A');
insert into id_test1 (col1) values ('B');
insert into id_test1 (col1) values ('C');
--
select * from id_test1;
  ID COL1
-----
   1 A
   2 B
   3 C
```



```
create table id_test1
(id number generated as identity (
  start with 10 increment by 11),
 coll varchar2(10));
--
insert into id_test1 (coll) values ('A');
insert into id_test1 (coll) values ('B');
insert into id_test1 (coll) values ('C');
--
select * from id_test1;
  ID COL1
-----
    10 A
    21 B
    32 C
```



- Many systems take advantage of Oracle Global Temporary Tables
- Rows in Global Temporary Tables either for the life of the session or transaction
- CREATE SEQUENCE now offers a SESSION parameter allowing a sequence to be reset each time the Global Temporary Table is reinitialized (default is GLOBAL)

```
create sequence session_sample_seq  
start with 1 increment by 1  
session;
```



- Oracle 12c enhances the capabilities of column default settings
  - Columns may be set to a default when NULL values are INSERTed
  - Column default values may be based upon a SEQUENCE (.nextval or .currval)



```
drop sequence default_test_seq;
drop table default_test;
create sequence default_test_seq start with 1 increment by 1;
create table default_test
(id number default default_test_seq.nextval not null,
 col1 varchar2(10) ,
 col2 varchar2(10)default on null 'N/A' not null);
insert into default_test (col1,col2) values ('A',null);
insert into default_test (col1) values ('B');
insert into default_test (col1,col2) values ('C','test');
select * from default_test;
```

ID	COL1	COL2
1	A	N/A
2	B	N/A
3	C	test



- Columns may be marked “INVISIBLE” in CREATE/ALTER table
- Invisible columns do not appear in SQL\*Plus DESCRIBE or SQL Developer column display (does show in SQL Developer table column list)
- Invisible columns may be inserted into or omitted from INSERT statements
- When made visible columns appear at end of table



```
drop table invisible_test;  
create table invisible_test (  
  id number,  
  col1 varchar2(10),  
  col2 varchar2(10) invisible,  
  col3 varchar2(10));
```

```
desc invisible_test;
```

```
Name Null Type
```

```
-----  
ID          NUMBER  
COL1        VARCHAR2 (10)  
COL3        VARCHAR2 (10)
```



```
insert into invisible_test
(col1,col2,col3) values (1,'a','a');
insert into invisible_test
(col1,col3) values (2,'b');
insert into invisible_test values (3,'c');
select * from invisible_test;
alter table invisible_test modify col2 visible;
desc invisible_test;
```

Name	Null	Type
----	----	-----
ID		NUMBER
COL1		VARCHAR2 (10)
COL3		VARCHAR2 (10)
COL2		VARCHAR2 (10)



- Oracle 12c allows definition of PL/SQL Functions and Procedures using SQL's Common Table Expression (WITH)
  - Defining PL/SQL locally reduces context-switch costs
  - Local PL/SQL overrides stored PL/SQL with the same name
  - Local PL/SQL is not stored in the database
  - Local PL/SQL is part of the same source code as the SQL that uses it



```
with function times_42(inval number)
  return number
as
begin
  return inval * 42;
end;
select channel_id,count(*) nbr_rows,
       sum(quantity_sold) qtysold,
       sum(times_42(cust_id)) cust42
  from sh.sales
 group by channel_id
 order by channel_id
/
```



- Oracle 12c adds options to CREATE TABLE, ALTER TABLE, and SELECT allowing use of time dimensions in conjunction with FLASHBACK QUERY
  - Periods are defined using TIMESTAMP columns
  - CREATE/ALTER TABLE's PERIOD clause specifies period starting and ending times
  - SELECT statements AS OF PERIOD FOR clause allows selection of rows falling within periods



```
CREATE TABLE temporal_emp_test(  
    employee_id NUMBER,  
    last_name    VARCHAR2(50),  
    start_time   TIMESTAMP,  
    end_time     TIMESTAMP,  
    PERIOD FOR my_time_period (start_time, end_time));  
INSERT INTO temporal_emp_test  
    VALUES (1000, 'King', '01-Jan-10', '30-Jun-11');  
INSERT INTO temporal_emp_test  
    VALUES (1001, 'Manzo', '01-Jan-11', '30-Jun-11');  
INSERT INTO temporal_emp_test  
    VALUES (1002, 'Li', '01-Jan-12', null);  
SELECT * from temporal_emp_test AS OF PERIOD  
    FOR my_time_period TO_TIMESTAMP('01-Jun-10');  
SELECT * from temporal_emp_test VERSIONS PERIOD FOR  
    my_time_period BETWEEN TO_TIMESTAMP('01-Jun-10')  
        AND TO_TIMESTAMP('02-Jun-10');
```



- Some DDL statements may be performed ONLINE in Oracle 12c, eliminating the DML lock from earlier releases
  - DROP INDEX ... ONLINE
  - ALTER INDEX ... UNUSABLE ONLINE
  - ALTER TABLE ... SET UNUSED ... ONLINE ...
  - ALTER TABLE ... DROP ... ONLINE
  - ALTER TABLE ...  
MOVE PARTITION ... ONLINE
  - ALTER TABLE ...  
MOVE SUBPARTITION .... ONLINE



- Oracle 12c's TRUNCATE statement allows the use of CASCADE to eliminate values in tables that are referentially connected

```
TRUNCATE TABLE ID_TEST1 CASCADE;
```



- Time does not permit detailed EBR coverage
- Edition-Based Redefinition made its debut in Oracle 11g and provides an ability to significantly reduce downtime due to changes in PL/SQL and/or SQL
- Oracle 12c removes some limitations present in 11g'2 implementation of EBR:
  - Materialized Views and Types may be editioned
  - Virtual Columns may be used with EBR



- Oracle 11g and Oracle 12c have both added significant new functionality to the already robust Oracle database environment
- Oracle 12c represents the first major architectural change to Oracle since Version 6!
- With the release of Oracle 12c it's probably time for your shop to finally move to 11g R2
- 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 TRAINING DAYS 2014

February 5-7, 2014 - Denver Convention Center - Denver, Colorado



## Tracks

- Application Development
- Business Intelligence
- Database Administration
- DBA Deep Dive
- Database Tools of the Trade
- Hyperion
- Middleware
- Professional Empowerment

PHOTO CREDIT: Mike Landrum, SQL Developer and the "Data Tsunami" from i-Behavior

[www.rmoug.org](http://www.rmoug.org)



# COLLABORATE 14 – IOUG Forum

April 6 – 10, 2014

The  
Venetian  
Las Vegas,  
NV



# ODTUG Kscope14



SEATTLE, WASHINGTON • JUNE 22-26

**REGISTER TODAY!**

## TOPICS:

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



[www.kscope14.com](http://www.kscope14.com)



## **Oracle 12c and Oracle 11gR2 New Features For Developers - Session: UGF9765**

To contact the author:

**John King**

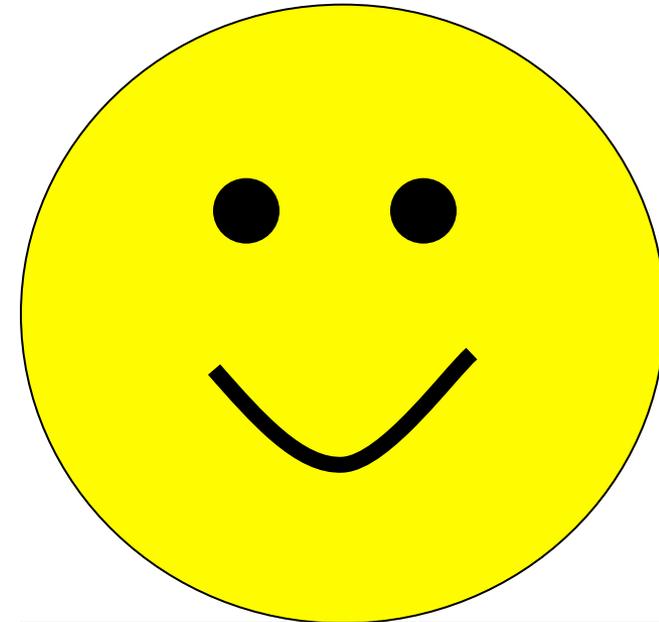
**King Training Resources**

P. O. Box 1780

Scottsdale, AZ 85252 USA

1.800.252.0652 - 1.303.798.5727

Email: [john@kingtraining.com](mailto:john@kingtraining.com)



**Thanks for your attention!**

Today's slides and examples are on the web:  
<http://www.kingtraining.com>



- End