

Oracle 12c

New Features For Developers



COLLABORATE15

TECHNOLOGY AND APPLICATIONS FC
FOR THE ORACLE COMMUNITY



APRIL 12-16, 2015
MANDALAY BAY
RESORT & CASINO

#C15LV

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





Session Objectives

- Learn new Oracle 12c 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 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 other topics)
- Member of AZORA, ODTUG, IOUG, and RMOUG

“Recent” Releases

- Oracle 11g R1 August 2007
- Oracle 11g R2 September 2009
- Oracle 12c R1 June 2013
- Oracle 12c R1.0.2 June 2014



- Oracle In-Memory Database
- 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
 - “Heat” maps
 - Adaptive Execution Plans
- More cool stuff (review OOW announcements...)

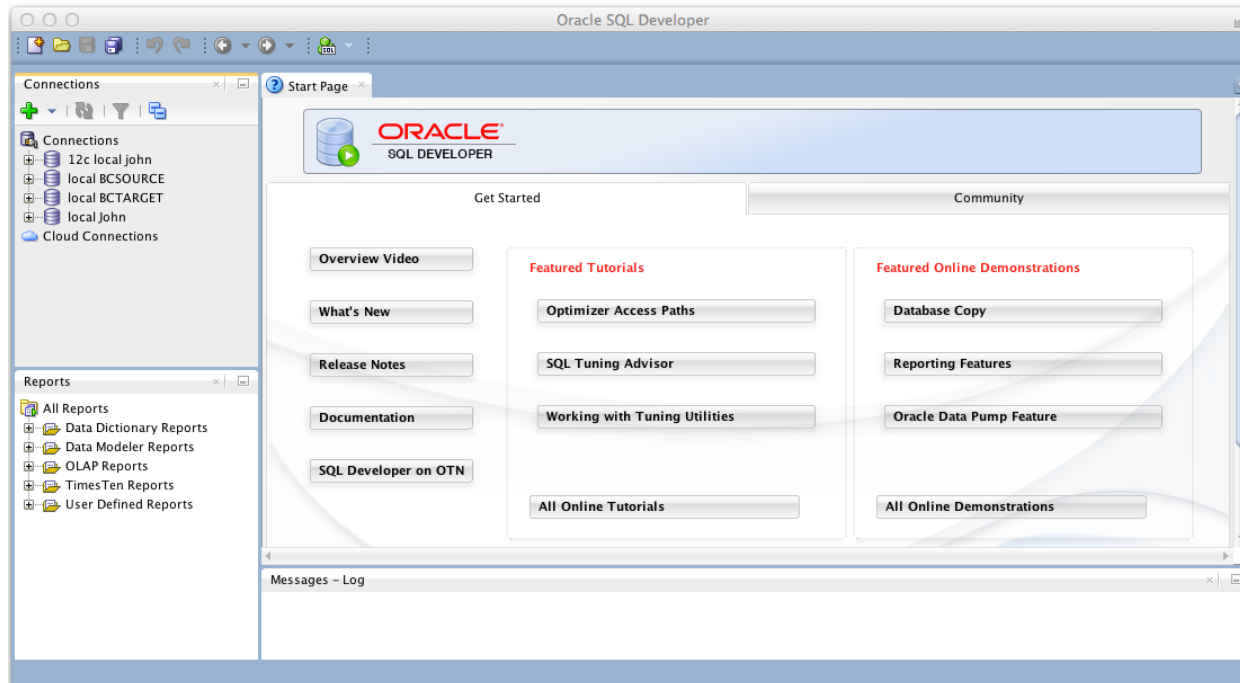
Oracle 12c New Features

- 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
- JSON in the database (12.1.0.2)



New SQL Developer

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



Top-N & Pagination

- 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

Top-N: Base Query

- Original query; note row sequence

```
select ename,sal from emp order by sal desc;
```

ENAME	SAL
-----	-----
KING	5000
FORD	3000
SCOTT	3000
JONES	2975
BLAKE	2850
CLARK	2450
ALLEN	1600
TURNER	1500
MILLER	1300
WARD	1250
*** more ***	
SMITH	800

Top-N: Using Rownum

- Original query uses “rownum” – note sequence of data (oops, wrong rows...)

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

--

ENAME	SAL
JONES	2975
ALLEN	1600
WARD	1250
SMITH	800

- Note: use of rownum, RANK, or DENSE_RANK in dynamic view (select from (subquery)) may be used to get correct rows

Top-N: First nn ROWS

- 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;
```

ENAME	SAL
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;
```

ENAME	SAL
-----	-----
FORD	3000
JONES	2975
BLAKE	2850
CLARK	2450
ALLEN	1600

Top-N: Percentage

- 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;
```

ENAME	SAL
-----	-----
SCOTT	3000

Matching Patterns

- 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

Sample Code 1

- 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))
```

Sample Code 2

```

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

Outer Join Improvements

- 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



Outer Join Example

```
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
```

CROSS & OUTER APPLY

- Oracle 12c adds the ability to JOIN values in a generated table collection to regular tables using correlated column values:
 - 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

Example APPLY - Setup

```
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;
/
```



Example APPLY

```
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;
```


- Lateral inline views introduce a new keyword allowing correlated references to other tables in a join
 - Correlated tables appear to the left of the inline view in the query's FROM list
 - Correlation names may be used anywhere within the inline view a correlation name usually occurs
(e.g. SELECT, FROM, WHERE, ...)

Example Lateral Inline View

- Here is an example using a lateral inline view; this syntax would fail without the “LATERAL” keyword

```
select last_name,first_name,department_name
       from hr.employees e, LATERAL(select *
                                   from hr.departments d
                                   where e.department_id
                                       = d.department_id) ;
```

New Column Sizes

- Oracle 12c increases the maximum size of three data types to 32,767 (4,000 before)
 - VARCHAR2
 - NVARCHAR2
 - RAW
- Not default: DBA sets max_sql_string_size EXTENDED (COMPATIBLE must be 12.0.0.0.0+)
- Stored out of line as SECUREFILE CLOB when > 4k
- Now matches PL/SQL variables

Identity Columns

- 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
 - IDENTITY column resets if table is dropped and recreated

Identity Example 1

```
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
```

Identity Example 2


```
create table id_test1
(id number generated as identity (
    start with 10 increment by 11),
    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
-----
   10 A
   21 B
   32 C
```



Session-Specific Sequence

- CREATE SEQUENCE now offers a SESSION parameter causing a sequence to be reset in each session where it is used

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



- Rows in Global Temporary Tables exist either for the life of the session or transaction
- While particularly useful for GTTs; session-specific sequences are NOT limited to GTTs

- 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

Invisible Columns

- 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 (why?? See next page)

- What happens when a column is marked invisible?
- The database marks column number to 0

```
SELECT c.name,c.type#,c.col#,c.intcol#,c.segcol#,
       TO_CHAR (c.property,'XXXXXXXXXXXX') AS property
FROM sys.col$ c, sys.obj$ o, sys.user$ u
WHERE c.obj# = o.obj#
AND o.owner# = u.user#
AND u.name = 'MYUSER'
AND o.name = 'MYTABLE' ;
```

- Col# is set to 0
- Property is set to x'40000020'

Invisible Column Example 1

```
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)

Invisible Column Example 2

```
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)

PL/SQL in WITH

- Oracle 12c allows definition of PL/SQL Functions and Procedures using SQL's Common Table Expression (WITH)
 - Defining PL/SQL locally reduces SQL-PL/SQL context-switching 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
 - PL/SQL Result Cache no use in Local PL/SQL

Example PL/SQL in WITH

```
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 allows functions to be defined using “PRAGMA UDF” to specify that a function will be used in SELECTS (behaving similar to function in WITH)
- This optimizes code for use within a SELECT or other SQL

Probably not a good option for functions also used from PL/SQL !

Example PL/SQL UDF

```
create or replace function times_42(inval number)
  return number
as
  pragma udf;
begin
  return inval * 42;
end;
/
```

Temporal Validity

- 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

Temporal Validity Example

```
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

TRUNCATE ... CASCADE

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

```
TRUNCATE TABLE ID_TEST1 CASCADE;
```

- Child table referential security must specify "ON DELETE CASCADE" or statement will fail

UTL_CALL_STACK

- Oracle has provided PL/SQL debug aids for a long time; perhaps your shop uses one: `dbms_utility.format_call_stack`, `dbms_utility.format_error_backtrace`, or `dbms_utility.format_error_stack`
- Oracle 12c adds `UTL_CALL_STACK` providing greater insight into the stack

- See documentation for a complete list of subprograms – here are a few:
 - CONCATENATE_SUBPROGRAM
Concatenated unit name
 - DYNAMIC_DEPTH
Number of subprograms on call stack
 - LEXICAL_DEPTH
Lexical nesting level of subprogram
 - UNIT_LINE
Line number in backtrace unit

Using UTL_CALL_STACK

```
create or replace procedure Print_Call_Stack
```

```
As
```

```
    DEPTH pls_integer := UTL_CALL_STACK.dynamic_depth();
```

```
    procedure printheaders is
```

```
        /* more code */
```

```
    procedure print is
```

```
        begin
```

```
            printheaders;
```

```
            for stunit in reverse 1..DEPTH loop
```

```
                dbms_output.put_line(
```

```
                    rpad( UTL_CALL_STACK.lexical_depth(stunit), 10 )
```

```
                    || rpad( stunit, 7)
```

```
                    || rpad(to_char(UTL_CALL_STACK.unit_line(stunit),
                                '99'), 9 )
```

```
                    || UTL_CALL_STACK.concatenate_subprogram
```

```
            end loop;
```

```
        /* more code */
```


Anatomy of Test Package

- The example package illustrates code nested within code:

```
package body TestPkg is
  procedure proc_a is
    procedure proc_b is
      procedure proc_c is
        procedure proc_d is
          Print_Call_Stack();
```

```
begin TestPkg.proc_a; end;
```

Error report -

```
ORA-06501: PL/SQL: program error
```

```
ORA-06512: at "JOHN.TESTPKG", line 11
```

```
ORA-06512: at "JOHN.TESTPKG", line 14
```

```
ORA-06512: at "JOHN.TESTPKG", line 17
```

```
ORA-06512: at "JOHN.TESTPKG", line 20
```

```
ORA-06512: at line 1
```

```
06501. 00000 - "PL/SQL: program error"
```

*Cause: This is an internal error message. An error has been detected in a PL/SQL program.

*Action: Contact Oracle Support Services

```
TESTPKG.PROC_A
```

```
TESTPKG.PROC_A.PROC_B
```

```
TESTPKG.PROC_A.PROC_B.PROC_C
```

```
TESTPKG.PROC_A.PROC_B.PROC_C.PROC_D
```

```
PRINT_CALL_STACK
```

```
PRINT_CALL_STACK.PRINT
```

12c (12.1.0.2) and JSON

- 12c patch-set 2 (12.1.0.2) adds JSON data
- JSON documents are stored as VARCHAR2, CLOB, or BLOB data type
- JSON data works with all existing Oracle features including SQL and Analytics
- 12c supports path-based queries of JSON data stored in the database, JSON Path Language, and JSON Path Expressions
- JSON is used in SQL via SQL/JSON views
- JSON documents may be indexed

JSON-XML Similarities

- JSON is text only, just like XML and thus is an excellent vehicle for data interchange—JSON and XML are both plain text
- JSON and XML are “human readable” and “self-describing”
- JSON and XML are hierarchical (data sets nested within data sets)
- JSON and XML offer validation capability; XML’s is more mature and capable today

JSON-XML Dissimilarities

- XML is verbose, JSON is shorter
- JSON does not end tags, required in XML
- JSON is quicker to read and write
- Reading XML documents requires “walking the DOM” – JSON does not
- JSON works more easily and is faster than XML when working with AJAX
- XML documents must be tested for “well-formed”-ness before processing

```
<?xml version="1.0"?>
<myBooks>
  <book>
    <name>Learning XML</name>
    <author>Eric T. Ray</author>
    <publisher>O'Reilly</publisher>
  </book>
  <book>
    <name>XML Bible</name>
    <author>Elliotte Rusty Harold</author>
    <publisher>IDG Books</publisher>
  </book>
  <book>
    <name>XML by Example</name>
    <author>Sean McGrath</author>
  </book>
</myBooks>
```

```
{ "myBooks" :
  [ {
    "name": "Learning XML",
    "author": "Eric T. Ray",
    "publisher": "O'Reilly"
  },
  {
    "name": "XML Bible",
    "author": "Elliotte Rusty Harold",
    "publisher": "IDG Books"
  },
  {
    "name": "XML by Example",
    "author": "Sean McGrath",
    "publisher": "Prentice-Hall"
  }
  ]
}
```

Oracle as JSON Data Store

- JSON documents are stored in the database using existing data types
 - VARCHAR2, CLOB and BLOB for character mode JSON
 - External JSON data sources accessible through external tables
 - JSON in file system (also HDFS) can be accessed via external tables

- JSON content is accessible from SQL via new operators
 - JSON_VALUE Used to query a scalar value from a JSON document
 - JSON_TABLE Used to query JSON document and create relational-style columns
 - JSON_EXISTS Used in query to see if JSON path exists in document IS JSON Used to validate JSON, usually in CHECK constraint
- JSON operators use JSON Path language to navigate JSON objects

JSON Check Constraint

```
create table deptj
(id raw(16) not null,
 dept_info clob constraint deptjson
                check (dept_info is json)
) ;
```

```
insert into deptj values
(sys_guid(),
'{"departments":{
  "DEPTNO": 10, "DNAME": "ACCOUNTING", "LOC": "NEW YORK",
  "deptemps": [
    { "EMPNO": 7782,
      "ENAME": "CLARK",
      "JOB": "MANAGER",
      "MGR": 7839,
      "HIREDATE": "09-JUN-81",
      "pay": {
        "SAL": 2450,
        "COMM": null},
      "DEPTNO": "10"
    },
    /* more */
```

Simple JSON Query

```
select dept_info
from deptj;
```

DEPT_INFO

```
-----
{"departments": {
  "DEPTNO": 10,
  "DNAME": "ACCOUNTING",
  "LOC": "NEW YORK",
  "deptemps": [
    {
      "EMPNO": 7782,
      "ENAME": "CLARK",
      **** more ****
```

Query with JSON_VALUE

```
select json_value(dept_info, '$.departments.DNAME')  
from deptj;
```

DNAME

ACCOUNTING

RESEARCH

SALES

OPERATIONS



Query with JSON_TABLE

```
select dname,ename,job,sal
from deptj, json_table(dept_info,'$.departments'
    columns (dname varchar2(15) path '$.DNAME'
    ,nested path '$.deptemps[*]'
    columns (ename varchar2(20) path '$.ENAME'
    ,job varchar2(20) path '$.JOB'
    ,nested path '$.pay'
    columns (sal number path '$.SAL')
    )
) );
```

DNAME	ENAME	JOB	SAL
-----	-----	-----	-----
ACCOUNTING	CLARK	MANAGER	2450
ACCOUNTING	KING	PRESIDENT	5000
**** more ****			

EBR Improvements

- 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 11gR2 implementation of EBR:
 - Public Synonyms may point to editioned objects
 - Materialized Views, Types, and Virtual Columns may reference editioned objects

- CREATE/ALTER MATERIALIZED VIEW
now add the ability to specify use with
editioning:
 - UNUSABLE BEFORE
 - CURRENT EDITION
 - EDITION XXX
 - UNUSABLE BEGINNING
 - CURRENT EDITION
 - EDITION XXX
 - NULL EDITION

- CREATE/ALTER TYPE now add the ability to specify use with editioning:
 - UNUSABLE BEFORE
 - CURRENT EDITION
 - EDITION XXX
 - UNUSABLE BEGINNING
 - CURRENT EDITION
 - EDITION XXX
 - NULL EDITION

- Non-editioned Virtual Columns may depend upon editioned objects
 - May specify expression is to be resolved by searching the specified edition:
 - CURRENT EDITION
 - EDITION XXX
 - NULL EDITION
 - May use UNUSABLE EDITION or UNUSABLE BEGINNING clause (see previous page) to limit Virtual Columns “visibility” into editions

Wrapping it all Up

- Oracle 12c has added significant new functionality to the already robust Oracle database environment; release 12.1.0.2 adds even more
- 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
- I am still actively testing the new features presented here (and some others); your mileage may vary; watch for future editions of this talk or blog posts for more

RMOUG Training Days 2016

February 9-11, 2016

(Tuesday-Thursday)

Denver Convention Center



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



COLLABORATE 16 – IOUG Forum

April 10 – 14, 2016

**Mandalay Bay
Las Vegas, NV**





ODTUG
Kscope15

HOLLYWOOD, FLORIDA • JUNE 21-25



Paper 785: Oracle 12c New Features For Developers

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>

- End

