Oracle 12c New Features For Developers & DBAs

Presented by: John Jay King

Download these slides from: http://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
- I help customers use technology through training and consulting in Oracle and other topics (http://www.kingtraining.com)
- "Techie" who knows Oracle, ADF, SQL, Java, and PL/SQL pretty well (along with many other topics)
- Member of AZORA, ODTUG, IOUG, and RMOUG
- One of those "dog-spoiling" people

King Training Resources

- Providing customized training solutions since 1988 in the US and internationally
- Oracle topics include: SQL, PL/SQL, Database, Cloud, APEX, ADF, MAF, Forms, Reports, Pro*C/Pro*COBOL
- Non-Oracle topics include: UX, Web Services, IoT, REST, Blockchain, Cloud Foundry, Java, JavaScript, HTML5, CSS, jQuery, COBOL, .NET, SQL Server, DB2, Business Analyst, more
- Visit us at www.kingtraining.com for more information and free downloads of presentations and code
- Contact Peggy at 1.303.798.5727 to schedule training today (email: peggy@kingtraining.com)

Arizona, USA

500+ Technical Experts Helping Peers Globally

ORACLE ACE PROGRAM

---

3 Membership Tiers bit.ly/OracleACEProgram

- Oracle ACE Director
- Oracle ACE
- Oracle ACE Associate

Connect:

oracle-ace_ww@oracle.com

Facebook.com/oracleaces

@oracleace

Nominate yourself or someone you know: **acenomination.oracle.com**

"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 12c R2	Fall 2016 for Cloud March 2017 for all
• Oracle 18.1	Coming soon…

Oracle 12c – Exciting DBA Stuff

- 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 (watch OOW announcements…)

What is Multi-Tenant?

- Multi-Tenant architecture is designed to achieve two specific goals:
  - Improved performance
  - Ease of management and consolidation
- Multi-tenant has two types of databases:
  - Container Database (CDB) - "Main" database contains up to 252 PDBs (Oracle EE) or exactly one PDB (Oracle SE)
  - Pluggable Database (PDB) - "Application" databases containing application/function-specific users and data

CDB & PDBs Share

- Single SGA
- Single set of database processes
- Single database to patch and/or upgrade (CDB)
- Single database to backup (CDB)
- Single configured container as standby database
- Single configuration for High-Availability, Data Guard, or RAC

What's the Big Deal?

- Less memory required
- Less space required
- For example:
  - Before Oracle 12c: 30 database instances might require approximately 20 background processes (each) to run; or, about 600 processes
  - 12c: 30 PDBs share 20 background processes (that's it)

Reasons to use Multi-Tenant

- Fast provisioning of new database or copy of existing database
- Fast redeployment to new platform
- Quickly patch and upgrade database version ONCE for all PDBs
- Patch/upgrade unplugging PDB from one CDB and plugging into CDB at later version
- Machine can run more databases as PDBs
- No changes required to user applications

Oracle In-Memory Database

- Oracle introduced new pay-for “In Memory Option” as part of Oracle Database 12.1.0.2 (so far only for Oracle EE)
- Oracle database normally stores data in tables; one row after another (on disk, pulled into memory for processing)
- In Memory Option ALSO stores table data in columnar format in memory
- Data in columnar format can speed some queries significantly

In-Memory “Secret Sauce”

- Both row and columnar formats are in memory at the same time; the optimizer decides which data store will work best
- SGA “In-Memory Area” (new pool) stores as much as will fit
- Tables are added to memory with ALTER TABLE xxx IN MEMORY - pivots data and adds to columnar store
- In-Memory is part of database; transparent to applications once tables added

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 & 12.2.0)

New SQL Developer

- Oracle SQL Developer 17.3 is now available for download (Sept 29!)
- Many new features & supports Oracle 12c (still one or two “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
```

Pagination

- 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
```

APPROX_COUNT_DISTINCT

- Oracle 12.1.0.2 documented aggregate first added internally to Oracle 11g
- Provides approximate value without actually processing all of the rows

```
select count(distinct cust_id) from sh.sales;
COUNT(DISTINCTCUST_ID)
----------------------
7059
      Elapsed: 00:00:00.614
select approx_count_distinct(cust_id) from sh.sales;
APPROX_COUNT_DISTINCT(CUST_ID)
------------------------------
7014
      Elapsed: 00:00:00.074
```

12.2: Six More Approximates

- APPROX_COUNT_DISTINCT_AGG
  Aggregations of approximate distinct counts
- APPROX_COUNT_DISTINCT_DETAIL
  Input values to APPROX_DISTINCT_AGG
- APPROX_MEDIAN
  Approximate Median
- APPROX_PERCENTILE
  Approximate Percentile
- APPROX_PERCENTILE_AGG
  Aggregations of approximate percentiles
- APPROX_PERCENTILE_DETAIL
  Input values to APPROX_PERCENT_AGG

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

- 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

Sample MATCH_RECOGNIZE

- The code on the following pages creates a report illustrating sales patterns (highs & lows) for a specific product over time
- Given five periods of data showing sales of:

2229 1191 1333 887 2148

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
```

Results

- 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

- 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

- 12c increases max size of VARCHAR2, NVARCHAR2, and RAW to 32,767
- Stored out of line as SECUREFILE CLOB when > 4k
- Now matches PL/SQL variables
- Not default required DBA action:
  - MAX_SQL_STRING_SIZE set to EXTENDED
  - COMPATIBLE must be 12.0.0.0.0+
  - Probably requires system restart to change
  - Once set cannot be undone

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
```

Enhanced Column DEFAULT

- 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)

Example Defaults

```
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
```

128-Byte Max Name Size

- Oracle 12.2 expands max size of many names from 30 to 128 bytes
- Requires COMPATIBLE init parameter of 12.2 or higher; otherwise max size is still 30
- Schema, table, and column names may all be up to 128 bytes long
- Some Oracle products might not support long names

Long Table & User Names

- Both User and Table names may be 128

```
SQL> create user ridiculously_long_user_name_now_works
  2  identified by whoops;

User RIDICULOUSLY_LONG_USER_NAME_NOW_WORKS created.
```

```
SQL> create table ridiculously_long_table_name_now_works
  2  (ridiculously_long_column_name1 number(4) generated as
identity,
  3   ridiculously_line_column_name2 varchar2(400)
  4   );

Table RIDICULOUSLY_LONG_TABLE_NAME_NOW_WORKS created.
```

Data Dictionary

```
SQL> desc ridiculously_long_table_name_now_works
Name                                      Null?    Type
----------------------------------------- -------- ------------
RIDICULOUSLY_LONG_COLUMN_NAME1 NOT NULL NUMBER(4)
RIDICULOUSLY_LINE_COLUMN_NAME2          VARCHAR2(400)
```

```
SQL> desc all_tab_cols
Name                    Null?    Type
----------------------- -------- -------------
OWNER                   NOT NULL VARCHAR2(128)
TABLE_NAME              NOT NULL VARCHAR2(128)
COLUMN_NAME             NOT NULL VARCHAR2(128)
DATA_TYPE                        VARCHAR2(128)
DATA_TYPE_MOD                    VARCHAR2(3)
DATA_TYPE_OWNER                  VARCHAR2(128)
```

Long Names in PL/SQL

- Here PL/SQL is used to load the column

```
declare
   plsql_also_supports_rediculously_long_names number := 0;
begin
   FOR j IN 1..5 LOOP
       plsql_also_supports_rediculously_long_names := j;
       insert into rediculously_long_table_name_now_works
       (rediculously_line_column_name2)
       values
       (plsql_also_supports_rediculously_long_names);
     END LOOP;
     dbms_output.put_line('nbr rows = ' ||
plsql_also_supports_rediculously_long_names);
END;
/
nbr rows = 5
```

Long Names in SQL

- Use the long names in SQL like any other

```
SQL> select * from rediculously_long_table_name_now_works;
REDICULOUSLY_LONG_COLUMN_NAME1 REDICULOUSLY_LINE_COLUMN_NAME2
------------------------------ ------------------------------
                             1 1
                             2 2
                             3 3
                             4 4
                             5 5
```

```
SQL> delete from rediculously_long_table_name_now_works
  2    where rediculously_long_column_name1 > 5;
5 rows deleted.
```

Column-Level Collation

- Oracle 12.2 allows specifications of collation comparisons at column level (data-bound collation)
  - Case insensitive BINARY_CI
  - Accent insensitive BINARY_AI

COLLATE BINARY_CI

- Use COLLATE BINARY_CI to make columns case-insensitive

```
CREATE TABLE EMP_CASE_INSENSITIVE
       (EMPNO NUMBER(4) NOT NULL,
        ENAME VARCHAR2(10) COLLATE BINARY_CI,
        JOB VARCHAR2(9),
        MGR NUMBER(4),
        HIREDATE DATE,
        SAL NUMBER(7, 2),
        COMM NUMBER(7, 2),
        DEPTNO NUMBER(2));

Table EMP_CASE_INSENSITIVE created.
```

Test Data Inserts

```
INSERT INTO EMP_CASE_INSENSITIVE VALUES
 (8301,'SMITH’,'DEVOPS’,7902,
   TO_DATE('17-DEC-2016’,'DD-MON-YYYY'),900,NULL, 20);
1 row inserted.
INSERT INTO EMP_CASE_INSENSITIVE VALUES
  (8302, 'smith’, 'DEVOPS’, 7902,
    TO_DATE('17-DEC-2016’,'DD-MON-YYYY'),1000, NULL, 20);
1 row inserted.
INSERT INTO EMP_CASE_INSENSITIVE VALUES
  (8303, 'Smith’,'DEVOPS’, 7902,
    TO_DATE('17-DEC-2016’,'DD-MON-YYYY'),1100,NULL, 20);
1 row inserted.
INSERT INTO EMP_CASE_INSENSITIVE VALUES
  (8304, 'sMiTh’, 'DEVOPS’, 7902,
    TO_DATE('17-DEC-2016’,'DD-MON-YYYY'),1200,NULL, 20);
1 row inserted.
```

Example Case-Insensitivity

```
select empno,ename from EMP_CASE_INSENSITIVE
  where ename = 'SMITH';

     EMPNO ENAME
---------- ----------
      8301 SMITH
      8302 smith
      8303 Smith
      8304 sMiTh
```

Accent-Insensitivity

- To make column comparisons accent insensitive use COLLATE BINARY_AI
- COLLATE BINARY_AI is the same as BINARY_CI except it ignores accent marks

Default Collation

- A table's default collation (for new columns) may be set via CREATE/ALTER TABLE
- Requires INIT PARAMETER MAX_STRING_SIZE = EXTENDED

```
CREATE TABLE sometable
...
DEFAULT COLLATION BINARY_CI
...
```

```
ALTER TABLE sometable
...
DEFAULT COLLATION BINARY_CI
...
```

Session-Specific Sequence

- 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;
```

  - 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

Invisible Columns

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

COL$ View

- 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,'XXXXXXXXXXX') 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)
```

SQL*Plus – CSV Output

- SQL*Plus can now generate CSV output

```
set markup csv on
select employee_id, last_name || ', '
       || first_name  empname, phone_number
   from hr.employees  order by empname;
set markup csv off
```

```
"EMPLOYEE_ID","EMPNAME","PHONE_NUMBER"
174,"Abel, Ellen","011.44.1644.429267"
166,"Ande, Sundar","011.44.1346.629268"
/* more rows here */
200,"Whalen, Jennifer","515.123.4444"
149,"Zlotkey, Eleni","011.44.1344.429018"

107 rows selected.
```

SQL*Plus History

- SQL*Plus now has a HISTORY command like the ones in Linux and SQL Developer

```
SQL> set history on
SQL> select * from emp;
SQL> select * from dept;
SQL> select * from hr.employees where rownum < 1;
```

```
SQL> history
  1  select * from emp;
  2  select * from dept;
  3  select * from hr.employees where rownum < 1;
```

SQL*Plus Script Performance

- SQL*Plus now allows setting of performance related paramenters when running scripts from the command line

```
ARRAYSIZE = 100

LOBPREFETCH = 16384

PAGESIZE = 50000

ROWPREFETCH = 2

STATEMENTCACHE = 20
```

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
/
```

PL/SQL UDF

- 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;
/
```

How Did They Rate?

- Here's how the three options stacked up:

	1st Run	2nd Run	3rd Run
Function in WITH	0.854	0.825	0.929
Compiled Function in database	2.018	1.945	1.928
Compiled UDF Function in database	0.667	0.602	0.664

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');
```

Online DDL

- 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
  - ALTER DATABASE MOVE DATAFILE (….) TO (….)

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

UTL_CALL_STACK Functions

- 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();
```

UTL_CALL_STACK Results

```
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 File

```
<?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>
```

JSON File

```
{"myBooks":
  [ {"book":
     "name":"Learning XML",
     "author":"Eric T. Ray",
     "publisher":"O'Reilly"
     },
     {"book":
     "name":"XML Bible",
     "author":"Elliotte Rusty Harold",
     "publisher":"IDG Books"
     },
     {"book":
     "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 SQL

- 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)
);
```

JSON and DML

```
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 ****
```

Generating JSON

- Oracle 12.2 provides functions for generating JSON from SQL
  - JSON_OBJECT
  - JSON_ARRAY
  - JSON_OBJECTAGG
  - JSON_ARRAYAGG

JSON_OBJECT

- JSON_OBJECT builds JSON objects using name-value pairs: name (literal or expr.) and a value (usually expr.)

```
select json_object('empId’  value employee_id
  , 'empName'     value last_name || ', ' || first_name
  , 'phoneNumber' value phone_number
  , 'deptId'      value department_id
  , 'pay’ value json_object('salary' value salary
                           ,'commPct' value commission_pct)
     FORMAT JSON) emp_rows
from hr.employees
where department_id < 30;
```

JSON_OBJECT Output

```
EMP_ROWS
--------------------------------------------------
{"empId":200,"empName":"Whalen,
Jennifer","phoneNumber":"515.123.4444","deptId":10,"pay":{"s
alary":4400,"commPct":null}}
{"empId":201,"empName":"Hartstein,
Michael","phoneNumber":"515.123.5555","deptId":20,"pay":{"sa
lary":13000,"commPct":null}}
{"empId":202,"empName":"Fay,
Pat","phoneNumber":"603.123.6666","deptId":20,"pay":{"salary
":6000,"commPct":null}}
```

JSON_OBJECT and NULLs, 1

- JSON_OBJECT allows NULL elements to be omitted using the ABSENT ON NULL

```
select json_object('empId' value employee_id
    , 'empName'     value last_name || ', ' || first_name
    , 'phoneNumber' value phone_number
    , 'deptId'      value department_id
    , 'pay'  value json_object('salary' value salary,
          'commPct' value commission_pct absent on null)
    FORMAT JSON) emp_rows
from hr.employees  where department_id < 30;
```

JSON_OBJECT and NULLs, 2

- JSON_OBJECT ABSENT ON NULL output

```
EMP_ROWS
--------------------------------------------------
{"empId":200,"empName":"Whalen,Jennifer”,
   "phoneNumber":"515.123.4444","deptId":10,
   "pay":{"salary":4400}}
{"empId":201,"empName":"Hartstein, Michael",
   "phoneNumber":"515.123.5555","deptId":20,
   "pay":{"salary":13000}}
{"empId":202,"empName":"Fay,    Pat",
   "phoneNumber":"603.123.6666","deptId":20,
   "pay":{"salary":6000}}
```

JSON_ARRAY

- JSON_ARRAY builds a JSON array using provided values and expressions

```
select json_object('employeeIdId' value employee_id
   , 'empName'     value last_name || ', ' || first_name
   , 'phoneNumber' value phone_number
   , 'deptId'  value department_id
   ,'pay' value json_array(salary,commission_pct)) json_out
  from hr.employees
  where commission_pct is not null and salary < 6500;
JSON_OUT
{"employeeIdId":166,"empName":"Ande, Sundar",
 "phoneNumber":"011.44.1346.629268","deptId":80,
 "pay":[6400,0.1]}
{"employeeIdId":167,"empName":"Banda, Amit",
 "phoneNumber":"011.44.1346.729268","deptId":80,
 "pay":[6200,0.1]}  … more …
```

JSON_OBJECTAGG

- JSON_OBJECTAGG aggregates the JSON resulting from multiple rows

```
select json_objectagg( last_name || ', ' || first_name value
salary) json_output
  from hr.employees
  where department_id < 50;

JSON_OUTPUT
--------------------------------------------------
{"Whalen, Jennifer":4400,"Hartstein, Michael":13000,
 "Fay, Pat":6000,"Raphaely, Den":11000,"Khoo, Alexander":
 3100,"Baida, Shelli":2900,"Tobias, Sigal":2800,"Himuro,
 Guy":2600,"Colmenares, Karen":2500,"Mavris, Susan":6500}
```

JSON_ARRAYAGG

```
JSON_OUTPUT
--------------------------------------------------------------------------------
{"departmentName":"Administration","deptId":10,"numberEmps":
1,"deptEmps":["Whalen, Jennifer"]}
{"departmentName":"IT","deptId":60,"numberEmps":4,"deptEmps"
:["Austin, David","Ernst, Bruce","Lorentz,
Diana","Pataballa, Valli"]}
{"departmentName":"Marketing","deptId":20,"numberEmps":1,"de
ptEmps":["Fay, Pat"]}
{"departmentName":"Purchasing","deptId":30,"numberEmps":5,"d
eptEmps":["Baida, Shelli","Colmenares, Karen","Himuro,
Guy","Khoo, Alexander","Tobias, Sigal"]}
{"departmentName":"Sales","deptId":80,"numberEmps":4,"deptEm
ps":["Ande, Sundar","Banda, Amit","Johnson, Charles","Kumar,
Sundita"]}
{"departmentName":"Shipping","deptId":50,"numberEmps":41,"de
ptEmps":["Atkinson, Mozhe","Bell, Sarah","Bissot,
Laura","Bull, Alexis","Cabrio, Anthony","Chung,
*** more ***
```

JSON_ARRAYAGG Output

- JSON_ARRAYAGG builds a JSON array using GROUPed data from an aggregate

```
select json_object('departmentName' value department_name
    , 'deptId'  value depts.department_id
    , 'numberEmps' value count(emps.employee_id)
    , 'deptEmps' value
       json_arrayagg( last_name || ', ' || first_name
                      order by last_name,first_name))
     json_output
  from hr.employees emps join hr.departments depts
     on depts.department_id = emps.department_id
  where salary < 6500
  group by depts.department_name,depts.department_id
  order by depts.department_name;
```

System READ Privilege

- Oracle 12.1.0.2 added two new privileges READ and READ ANY TABLE
- SELECT privilege (been there forever) also allows locking to occur
  - LOCK TABLE ...
  - SELECT ... FOR UPDATE
- READ privilege allows SELECT statements but does not allow statements to lock rows

```
GRANT READ ON xxx.yyy TO user,role;
GRAND READ ANY TABLE  TO user,role;
```

READ Privilege at Work

```
SQL> grant read on dept to t1;
```

```
SQL> select * from john.dept;
    DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        40 OPERATIONS     BOSTON
```

```
SQL> select * from john.dept for update;
select * from john.dept for update
                    *
ERROR at line 1:
ORA-01031: insufficient privileges
```

Merge Partitions

- Partitions may be merged easily

```
alter table ORDERS
  merge partitions P2014Q1,P2014Q2,P2014Q3,P2014Q4
      into P2014;
```

```
alter table ORDERS
   merge partitions P2014Q1 to P2014Q4 into P2014;
```

Partition Split

- Partitions may be split

```
alter table ORDERS split partition P2016 into
   (partition P2016Q1 values
      less than to_date('01.04.2016','DD.MM.YYYY')),
   (partion P2016Q2 values
      less than to_date('01.07.2016','DD.MM.YYYY')),
   (partition P2016Q3 values
      less than to_date('01.10.2016','DD.MM.YYYY')),
   (partition P2016Q4 values
      less than to_date('01.01.2017','DD.MM.YYYY');
```

Partition Add/Drop/Truncate

```
alter table ORDERS add
 partition P2017Q1 value
     less than to_date('01.04.2017','DD.MM.YYYY')),
 partition P2017Q1 value
     less than to_date('01.07.2017','DD.MM.YYYY')),
 partition P2017Q1 value
     less than to_date('01.10.2017','DD.MM.YYYY')),
 partition P2017Q1 value
     less than to_date('01.01.2018','DD.MM.YYYY')),
```

```
alter table ORDERS drop partitions
  P2010Q1, P2010Q2, P2010Q3, P2010Q4;
```

```
alter table ORDERS truncate partitions
  P2010Q1, P2010Q2, P2010Q3, P2010Q4;
```

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 and Types may be used with editioning
  - Virtual Columns may be used with EBR

EBR & Materialized Views

- CREATE/ALTER MATERIALIZED VIEW now add the ability to specify use with editioning by specifying the Edition(s) to be used:
  - UNUSABLE BEFORE
    - CURRENT EDITION
    - EDITION XXX
  - UNUSABLE BEGINNING
    - CURRENT EDITION
    - EDITION XXX
    - NULL EDITION

EBR & Types

- CREATE/ALTER TYPE now add the ability to specify use with editioning by specifying the Edition(s) to be used:
  - UNUSABLE BEFORE
    - CURRENT EDITION
    - EDITION XXX
  - UNUSABLE BEGINNING
    - CURRENT EDITION
    - EDITION XXX
    - NULL EDITION

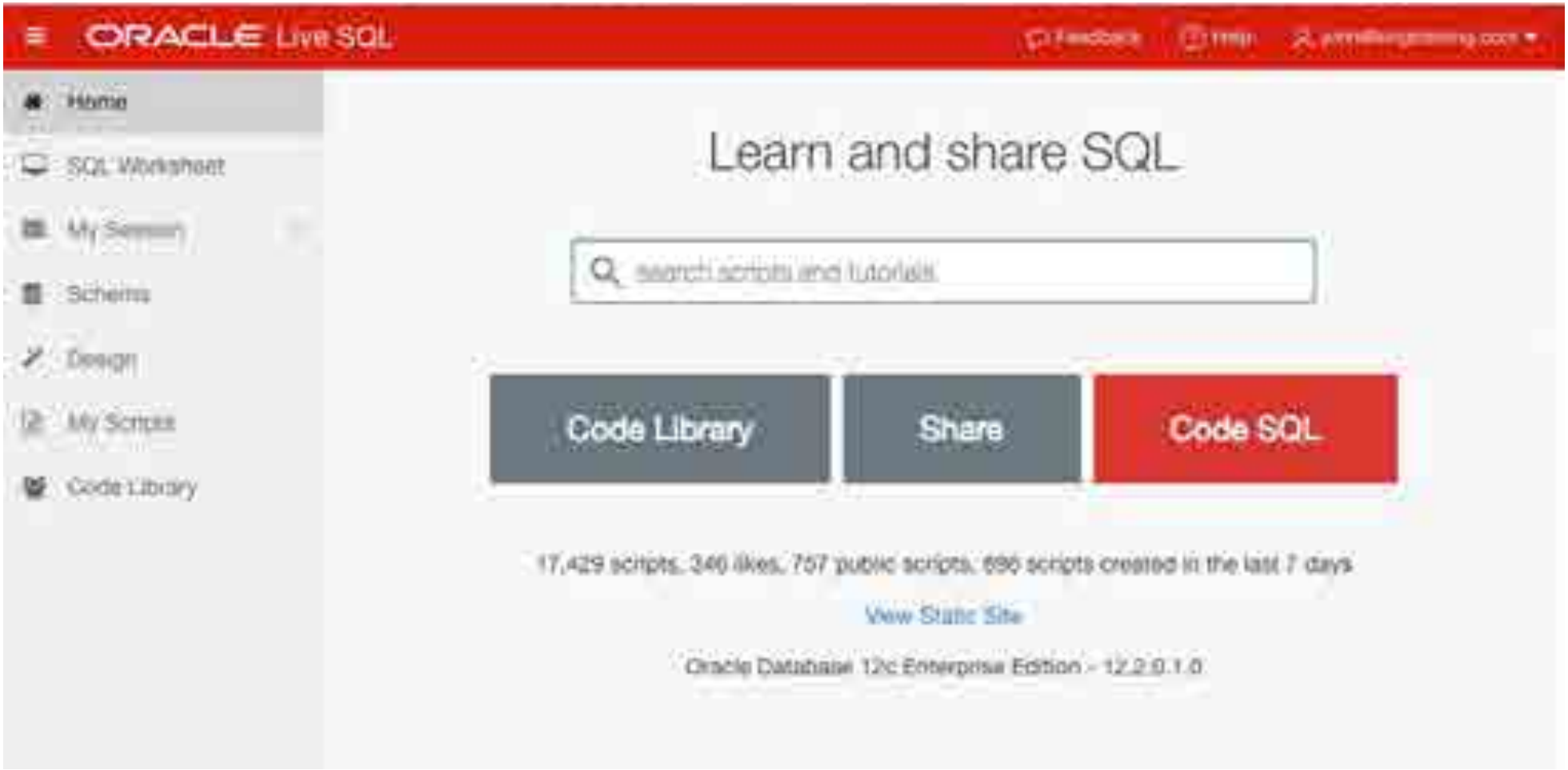
EBR & Virtual Columns

- 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

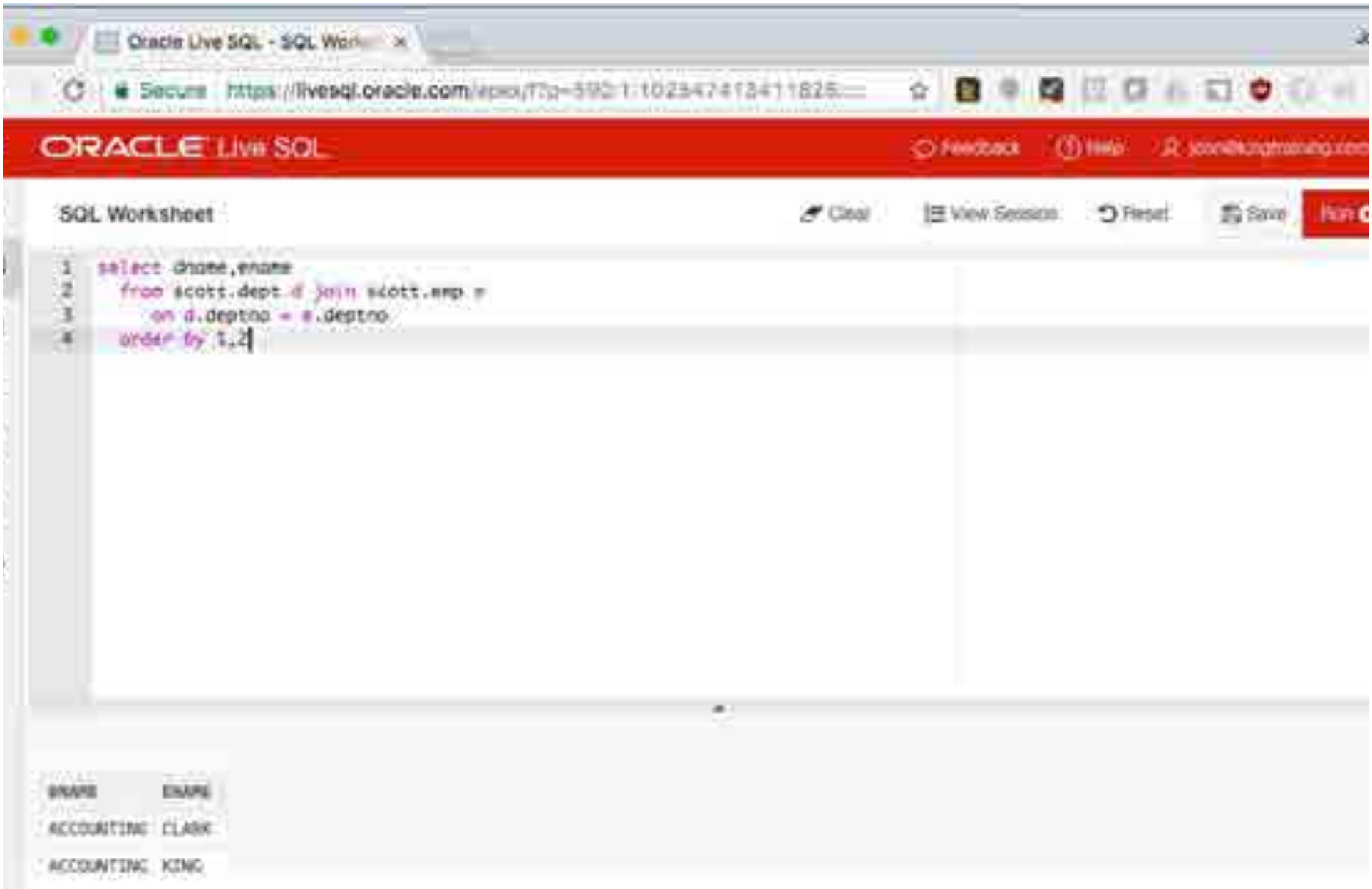
livesql.oracle.com

- Oracle provides free "live" 12.2 SQL tool
  - Includes available code library (cut & paste capab
  - Ability to save scripts and share
  - Online database design
  - Available sample schemas or build your own

livesql.oracle.com - Options



livesql.oracle.com - SQL



Oracle 12.2

- Oracle 12.2 was released for Oracle Exadata Express Cloud Service users in October 2016 (OOW)
- Oracle DBaaS uses got access to 12.2 in November 2016
- On-premise versions of 12.2 available for download as of March 2017

Overview of 12.2 Features

- JSON generating functions
- Analytic Views
- Max number of PDBs from 252 to 4096
- PDB memory and resource management
- Local UNDO for PDBs and "hot clone"
- SQL*Plus history and csv output
- Partition tables online
- READ-only partitions/subpartitions
- Oracle sharding
- Partitioned External Tables

New Oracle Versioning

- Oracle is changing the way they number products!
  - Products now released more-frequently
  - Each year will start with a release using the last two digits of the year; updates will be “dot” releases
  - Oracle 12.2.0.1.0 will be followed by
  - Oracle 18.1 (coming soon!)

Wrapping it all Up

- Oracle 12c has added significant new functionality to the already robust Oracle database environment; release 12.1.0.2 and 12.2.0 add even more
- Oracle 12c represents the first major architectural change to Oracle since Version 6
- With the release of Oracle 12c R2 it's probably time for your shop to finally move off 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 2018
February 20-22, 2018 (Tuesday-Thursday)
Westin Hotel
10600 Westminster Blvd
Westminster, CO 80020
Tracks
• Application Development
• Business Intelligence
• Database Administration
• DBA Deep Dive
• Database Tools of the Trade
• Hyperion
• Middleware
• Professional Empowerment
RMOUG
www.rmoug.org

COLLABORATE 18

TECHNOLOGY AND APPLICATIONS FORUM FOR THE ORACLE COMMUNITY

Save the Date

COLLABORATE 18 registration will open on Wednesday, November 8.

Call for Speakers

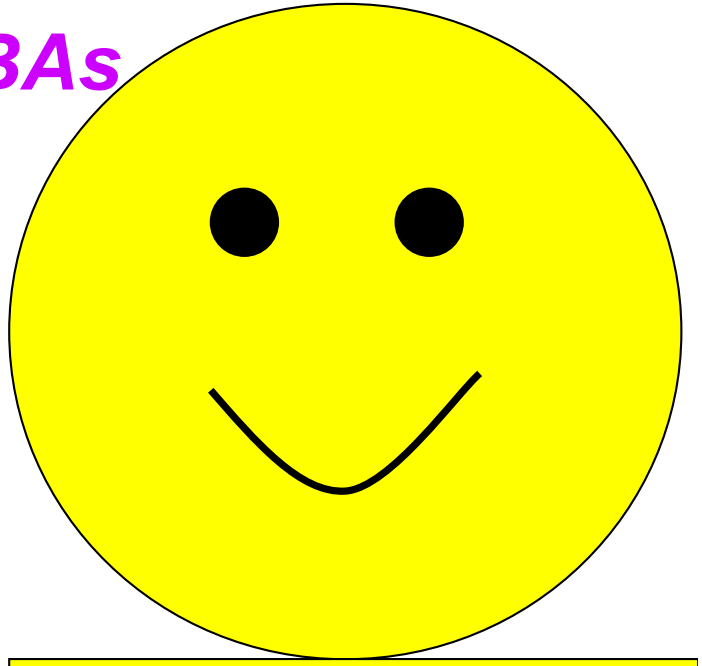
Submit your session presentation! The Call for Speakers is open until Friday, October 20

collaborate.ioug.org


ODTUG
Kscope18
ORLANDO, FLORIDA • JUNE 10-14
Register Now
www.kscope18.odtug.com
ORLANDO

Please Complete Session Evaluations

Oracle 12c
New Features For Developers & DBAs



Thanks for your attention!

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

Twitter: @royaltwit

Linked In: https://www.linkedin.com/in/john-king-4175603

Today's slides and examples are on the web:

http://www.kingtraining.com

- End