



SESSION 440: PARLEZ-VOUS KLINGON RECURSION SQL FOR DATABASE MAGICIANS.DOC

04/15/2008 AT 4:45 PM - 5:45 PM

Author Name: Alexander Kopač, DBI

-- -- --

ABSTRACT:

-- -- --

Often when the words 'recursive query' are spoken, a shiver of fear runs down the spines of application developers and DBAs. Fear not!

In your near-future work life, you may also hear "Data Disguise Required" meaning that "use of data derived from relationships for purposes of testing, research, training, or publication will be confined to content that is disguised to ensure the anonymity of the individuals involved".

This session provides an easy, step by step, explanation of SQL "recursive queries" to spin thru tables and to quickly generate test data. Learn how recursive queries can be used to generate and refresh data without the need to write and compile C, COBOL, or Java programs.

We begin by explaining "recursive queries" in simple terms and give lots of examples to use in your office, and then the session progresses to showing data masking techniques and to help you become a "Master of Data Disguise".

-- -- --

Key Topics:

-- -- --

- * Explain "recursive queries" in RDBMS engines in simple terms
- * Explain recursion as used within Oracle in simple terms

Parlez-Vous Klingon? Recursion SQL for Database Magicians

- * Show some examples to search thru tables
- * Show some examples of some common syntax errors
- * RDBMS recursive SQL comparisons between Oracle & DB2
- * Show some techniques to generate/insert test data and/or masking data

-- -- --

Objectives:

-- -- --

- Objective 1: Learn how to create simple SQL recursive queries replacing hundreds of lines of SQL code
- Objective 2: Learn how DBA's & application programmers can use SQL recursive queries for overall performance
- Objective 3: Learn various data masking techniques & why data type selection in SQL is important for performance

-- -- --

Prerequisite:

-- -- --

Basic RDBMS understanding coupled with a detailed knowledge of SQL semantics helpful

-- -- --

Presentation Credits / Resources :

-- -- --

http://www.dba-oracle.com/t_with_clause.htm

Oracle WITH clause - Oracle Tips by Burleson Consulting

http://www.dba-Oracle.com/t_recursive_sql_statspack.htm

Oracle recursive SQL and STATSPACK - Oracle Tips by Burleson Consulting

http://www.remote-dba.net/oracle_10g_tuning/t_oracle_with_sql_clause.htm

Oracle WITH clause to simplify complex SQL - Oracle Tips by Burleson Consulting

<http://www.gennick.com/with.html>

Understanding the WITH Clause by Jonathan Gennick, 1-July-2003

Joe Celko's Trees and Hierarchies in SQL for Smarties - Joe Celko

Elsevier Ltd MAY-2004 ISBN-13: 978-1-55860-920-4 ISBN-10: 1-55860-920-2

[Birchall] Graeme Birchall, "DB2 9 Cookbook", at
http://mysite.verizon.net/Graeme_Birchall/id1.html

Parlez-Vous Klingon? Recursion SQL for Database Magicians

DB2 SQL Cookbook Downloads Page: Graeme Birchall

http://mysite.verizon.net/Graeme_Birchall/id1.html

http://mysite.verizon.net/Graeme_Birchall/cookbook/DB2V91CK.PDF

http://mysite.verizon.net/Graeme_Birchall/cookbook/HTM_SQL.html

http://mysite.verizon.net/Graeme_Birchall/cookbook/DB2V91CL.PDF

-- -- --

Learning Goals:

-- -- --

1. Learn how to create simple SQL recursive queries replacing hundreds of lines of SQL code
2. Learn how DBA's & application programmers can use SQL recursive queries for overall performance and database efficiency
3. Learn how to quickly create 'safe' test data in order to meet auditing and compliance requirements

-- -- --

INTRODUCTION:

-- -- --

"Any sufficiently advanced technology is indistinguishable from magic."

Sir Arthur C. Clarke, "Profiles of The Future", 1961 (Clarke's third law)
English physicist & science fiction author (1917 -)

From Wikipedia, the free encyclopedia:

Arthur C. Clarke formulated the following three "laws" of prediction:

1. When a distinguished but elderly scientist states that something is possible, he is almost certainly right. When he states that something is impossible, he is very probably wrong.
2. The only way of discovering the limits of the possible is to venture a little way past them into the impossible.
3. Any sufficiently advanced technology is indistinguishable from magic.

This presentation is a simple beginner view, a basic primer & starting point for your continued success.

Regrettably, the subject matter is vast and the material is therefore incomplete.

This session provides comparisons & examples for both Oracle & DB2 practitioners.

The author sincerely hopes you will find this paper & the examples useful; have fun!

Parlez-Vous Klingon? Recursion SQL for Database Magicians

-- -- --

Here's a Recursion Dictionary Definition:

-- -- --

"An algorithmic technique where a function, in order to accomplish a task, calls itself with some part of the task; self-referencing."

Paul E. Black and Patrick Rodgers, "recursion", in Dictionary of Algorithms and Data Structures [online], Paul E. Black, ed., U.S. National Institute of Standards and Technology. 21 August 2006. Available from:

<http://www.nist.gov/DADs/HTML/recursion.html>

-- -- --

Recursive Queries in most RDBMS engines routinely have two parts:

-- -- --

- 1) A Base Part(s) where query is a simple/single SQL stmt solved directly.
- 2) A Recursive Part (or parts) that typically have three parts or more components.
 - A. This part divides the query into one or more smaller query parts.
 - B. Next, the SQL query calls some process/function recursively on each smaller query part(s).
 - C. Finally, the query combine answer set of each part(s) into a final solution.

-- -- --

Here's the industry defined resources for SQL Standards & 'Recursion' Definitions:

-- -- --

SQL (1986), SQL2 (1992), SQL 3 (1999).

See: http://en.wikipedia.org/wiki/SQL_Standardization

SQL was adopted as a standard by ANSI (American National Standards Institute) in 1992 and ISO (International Organization for Standardization) 1987.

The SQL standard has gone through a number of revisions:

| Year | NAME | Alias | Comments |
|------|----------|--------|---|
| 1986 | SQL-86 | SQL-87 | First published by ANSI. Ratified by ISO in 1987. |
| 1989 | SQL-89 | | Minor revision. |
| 1992 | SQL-92 | SQL2 | Major revision (ISO 9075). |
| 1999 | SQL:1999 | SQL3 | Added regular expression matching, recursive queries, triggers, non-scalar types and some object-oriented features. (The last two are somewhat controversial and not yet widely supported.) |

Parlez-Vous Klingon? Recursion SQL for Database Magicians

2003 SQL:2003 Introduced XML-related features, window functions, standardized sequences and columns with auto-generated values (including identity-columns).

2006 SQL:2006 ISO/IEC 9075-14:2006 defines ways in which SQL can be used in conjunction with XML. It defines ways of importing and storing XML data in an SQL database, manipulating it within the database and publishing both XML and conventional SQL-data in XML form. In addition, it provides facilities that permit applications to integrate into their SQL CODE the use of XQuery; the XML Query Language published by the World Wide Web Consortium (W3C), to concurrently access ordinary SQL-data and XML documents.

The SQL standard is not freely available.

SQL:2003 and SQL:2006 may be purchased from ISO or ANSI.

A late draft of SQL:2003 is available as a ZIP archive from Whitemarsh Information Systems Corporation. The ZIP archive contains a number of PDF files that define the parts of the SQL:2003 specification.

-- -- --

SQL Recursion in Oracle:

-- -- --

Recursion SQL in Oracle now uses the "WITH" SQL clause; (the SQL-99 standard):

Added as valid syntax beginning with Oracle Version 9.2

Recursion SQL also known as the "Subquery Factoring Clause"

Inline views can also be expressed using "WITH" SQL clause

SQL2 did not support recursive queries

SQL3 supports recursive queries: "WITH" SQL clause

-- -- --

SQL Recursion in DB2

-- -- --

Recursion SQL in DB2 uses a "WITH" SQL clause aka Common Table Expression (CTE) in use since DB2 V5 for LUW & DB2 V8 for z/OS. CTE's are defined at beginning of a DB2 query using "WITH" SQL clause.

-- -- --

Some Recursion SQL Factoids

-- -- --

Recursion SQL is commonly used to solve the following problems:

-- -- --

Bill of Materials

Parlez-Vous Klingon? Recursion SQL for Database Magicians

Hierarchies

Network Planning

Organization Charts

Airline/Transportation Reservation Systems

Catalog Referential Integrity (RI) Searches - Tables & Views

-- -- --

More Recursion SQL Factoids

-- -- --

Recursion SQL is commonly used for solving 'Tree Structure' problems:

-- -- --

Acyclic Graph Example:

Master Health Insurance Policy with

Skydiving Rider

Un-Insured Motorist Rider

Accidental Death & Dismemberment Rider

Cyclic Graph Example:

CITY Travel Pairs - Travel Destinations

New York CITY-London-Rome-New York CITY

San Jose-San Francisco-Santa Clara-Los Angeles-San Jose

-- -- --

Note: Cyclic SQL Queries must take care to prevent 'looping' backwards !

-- -- --

From Wikipedia, the free encyclopedia:

http://en.wikipedia.org/wiki/Directed_acyclic_graph:

In computer science and mathematics, a directed acyclic graph, also called a dag or DAG, is a directed graph with no directed cycles; that is, for any vertex v , there is no nonempty directed path starting and ending on v . DAGs appear in models where it doesn't make sense for a vertex to have a path to itself; for example, if an edge $u \rightarrow v$ indicates that v is a part of u , such a path would indicate that u is a part of itself, which is impossible. Every directed acyclic graph corresponds to a partial order on its vertices, in which $u \leq v$ is in the partial order exactly when there exists a directed path from u to v in the graph. However, many different directed acyclic graphs may represent the same partial order in this way. Among these graphs, the one with the fewest edges is the transitive reduction and the one with the most edges is the transitive closure.

Parlez-Vous Klingon? Recursion SQL for Database Magicians

http://en.wikipedia.org/wiki/Cycle_graph:

In graph theory, a cycle graph, is a graph that consists of a single cycle, or in other words, some number of vertices connected in a closed chain. The cycle graph with n vertices is called Cn. The number of vertices in a Cn equals the number of edges, and every vertex has degree two, that is, every vertex has exactly two edges incident with it.

-- -- --

Even More Recursion SQL Factoids

-- -- --

Here are some common 'TREE STRUCTURE' types:

| DIVERGENT | CONVERGENT | BALANCED | UNBALANCED | RECURSIVE |
|-----------|------------|----------|------------|-----------|
| ===== | ===== | ===== | ===== | ===== |
| A1A | A1A | A1A | A1A | A1A<--+ |
| | | | | |
| +----+ | +----+ | +----+ | +----+ | +----+ |
| | | | | |
| B2B C3C | B2B C3C | B2B C3C | B2B C3C | B2B C3C>+ |
| | | | | |
| +----+ | +----++ | +----+ | +----+ | +----+ |
| | | | | |
| D4D E5E | D4D E5E | E5E F6F | D4D E5E | E5E D4D |

-- -- --

Here's a common Recursion SQL development processes: ;>)

-- -- --

1. ESTABLISH TEMPORARY TABLE (CREATOR/SCHEMA)
2. INITIALIZE QUERY (AT LEAST ONE ROW)
3. UNION ALL (ON SELF REFERENCING NAME ("CTE"))
4. START RECURSIVE QUERY (WITH ENDPOINT)
5. RETURN FINAL RESULTS (MAYBE AGGREGATE)
6. POST/BRAG SUCCESS MSG ON:
BOSS' DESK;
ON YOUR BLOG;
NEXT CUBICLE OVER;
GO TO CHURCH, ETC.

Parlez-Vous Klingon? Recursion SQL for Database Magicians

-- -- --

Recursion SQL in Oracle:

-- -- --

Recursion in Oracle has "Many Internal/External Uses"

-- -- --

SYS queries in Oracle are often known as recursive !

While inserting a row into an Oracle table that does not have enough space, the Oracle Server makes recursive calls in order to allocate space dynamically when dictionary managed tablespaces are used.

-- -- --

Oracle Metalink note 232443.1 also divides CPU time into 3 parts:

"parse time cpu"

time spent in parsing

"recursive cpu usage"

time spent in recursive calls such as PL/SQL calls from sql

"cpu other"

All the remaining cpu time

-- -- --

Recursion in Oracle has "Many Internal/External Uses"

-- -- --

Here's some examples:

-- -- --

```
SELECT
  STATISTIC#
  NAME,
  CLASS,
  VALUE,
  STAT_ID
from V$SYSSTAT
where upper(name)
  like '%CPU%';
```

-- -- --

Parlez-Vous Klingon? Recursion SQL for Database Magicians

STATISTIC# NAME

| CLASS | VALUE | STAT_ID |
|-------------------------------|-------|------------|
| 8 recursive cpu usage | | |
| 1 | 15883 | 4009879262 |
| 11 CPU used when call started | | |
| 128 | 13789 | 572264820 |
| 12 CPU used by this session | | |
| 1 | 18270 | 24469293 |

-- -- --

STATISTIC# NAME

| CLASS | VALUE | STAT_ID |
|--|-------|------------|
| 43 IPC CPU used by this session | | |
| 32 | 0 | 4247517299 |
| 46 global enqueue CPU used by this session | | |
| 32 | 0 | 3469911798 |
| 161 gc CPU used by this session | | |
| 40 | 0 | 4093034494 |

-- -- --

STATISTIC# NAME

| CLASS | VALUE | STAT_ID |
|-------|-------|---------|
|-------|-------|---------|

Parlez-Vous Klingon? Recursion SQL for Database Magicians

```
331 parse time cpu
```

```
64      770  206905303
```

```
7 rows selected.
```

```
-- -- --
```

Recursion in Oracle has "Many Internal/External Uses"

```
-- -- --
```

```
DESCRIBE v$sysstat ;
```

| Name | Null? | Type |
|------------|-------|--------------|
| STATISTIC# | | NUMBER |
| NAME | | VARCHAR2(64) |
| CLASS | | NUMBER |
| VALUE | | NUMBER |
| STAT_ID | | NUMBER |

```
-- -- --
```

```
SELECT 'v$sysstat',  
       COUNT(*)  
FROM v$sysstat ;
```

| 'V\$SYSSTA | COUNT(*) |
|------------|----------|
| v\$sysstat | 350 |

```
-- -- --
```

Here's a way to see where Oracle stores an internal measurement of time spent in recursive calls such as PL/SQL calls (SQL functions & stored procedures) using the VIEW V\$SYSSTAT :

```
-- -- --
```

```
select name,  
       value  
from v$sysstat  
where upper(name) like '%CPU%' ;
```

| NAME | VALUE |
|------------------------------|-------|
| recursive cpu usage | 15883 |
| CPU used when call started | 13789 |
| CPU used by this session | 18270 |
| IPC CPU used by this session | 0 |

Parlez-Vous Klingon? Recursion SQL for Database Magicians

```
global enqueue CPU used by this session 0
gc CPU used by this session             0
parse time cpu                          770
```

7 rows selected.

-- -- --

Recursion in Oracle has "Many Internal/External Uses"

-- -- --

Oracle uses "Recursive SQL" that is normally "hidden"; these internal Oracle SQL functions are routinely used for items such as:

Parsing a new SQL statement for validating table/column names and security checking

Re-caching paged-out data from the Oracle Dictionary Cache

Referential Integrity (RI) checking

Space management functions needed for "Create Table" DDL & Insert DML

SQL within PL/SQL Stored procedures & SQL Functions

For more details see Don Burleson's work:

http://www.dba-Oracle.com/t_recursive_sql_statspack.htm

(You can often see this hidden stuff with vendor native database auditing tools)

-- -- --

DON'T

GET

SCARED...

-- -- --

HERE'S A WORKING WONDER WHOPPER & PRACTICAL RECURSION SQL . . .

- "TURNS AN ICON FROM RED TO GREEN..."

-- -- --

```
WITH SUMMARYD AS ( SELECT DATE(O.PURCHASE_DT) AS PURCHASE_DT, O.PURCHASE_NBR,
I.LAST_SHIP_TRK_NBR AS TRACKING_NBR, I.STATUS_MSG, I.QUANTITY, I.ORIENTATION,
CAST(REPLACE(CASE P.PURCHASE_TYPE_ID WHEN 1 THEN COALESCE(( SELECT UPPER(R.REFERENCE) FROM
O_PACKAGE_REFERENCE PR, O_REFERENCE R WHERE PR.PACKAGE_ID = P.PACKAGE_ID AND
R.REFERENCE_ID = PR.REFERENCE_ID AND R.REFERENCE_TYPE_ID = :1n), COALESCE(( SELECT
UPPER(R.REFERENCE) FROM O_PACKAGE_REFERENCE PR, O_REFERENCE R WHERE PR.PACKAGE_ID =
P.PACKAGE_ID AND R.REFERENCE_ID = PR.REFERENCE_ID AND R.REFERENCE_TYPE_ID = :1n),
'Unknown, Unknown')) WHEN 2 THEN COALESCE(( SELECT 'Stock PURCHASE (#' || R.REFERENCE ||
')' FROM O_PACKAGE_REFERENCE PR, O_REFERENCE R WHERE PR.PACKAGE_ID = P.PACKAGE_ID AND
R.REFERENCE_ID = PR.REFERENCE_ID AND R.REFERENCE_TYPE_ID = :1n), 'Stock PURCHASE') WHEN 3
```

Parlez-Vous Klingon? Recursion SQL for Database Magicians

```
THEN 'stock PURCHASE (by Sales Rep)' ELSE 'unknown' END, ' (' , '(') AS VARCHAR(1024)) AS
PURCHASE_FORM, O.PURCHASE_ID, O.SUPPLIER_ID, P.PACKAGE_ID, PI.SEQUENCE_NBR, I.MASTER_ID,
I.PRODUCT_ID, I.ITEM_ID, I.ITEM_STATUS_ID FROM A_USER U, O_PURCHASE O, O_PACKAGE P,
O_PACKAGE_ITEM PI, O_ITEM I , O_ITEM_STATUS_CD FRST WHERE U.USER_ID = ? AND O.ACCT_ID =
U.ACCT_ID AND P.PURCHASE_ID = O.PURCHASE_ID AND PI.PACKAGE_ID = P.PACKAGE_ID AND
P.PACKAGE_TYPE_ID NOT IN (18) AND I.ITEM_ID = PI.ITEM_ID AND FRST.ITEM_STATUS_ID =
I.ITEM_STATUS_ID AND I.ITEM_STATUS_ID != 9 AND O.PURCHASE_STATUS_ID != 1 AND
DATE(PURCHASE_DT) BETWEEN ? AND ? AND SUPPLIER_ID = ? AND ( I.ITEM_STATUS_ID = ? OR
I.ITEM_STATUS_ID = ? OR I.ITEM_STATUS_ID = ? OR I.ITEM_STATUS_ID = ? OR I.ITEM_STATUS_ID =
? OR I.ITEM_STATUS_ID = ? OR I.ITEM_STATUS_ID = ? ) AND
I.PRODUCT_ID NOT IN ( SELECT BI.PRODUCT_ID FROM HP.P_BUNDLEITEM BI WHERE NOT EXISTS (
SELECT 1 FROM HP.P_BUNDLE B WHERE B.BUNDLE_ID = BI.BUNDLE_ID AND B.MASTER_ID =
I.MASTER_ID) AND P.PACKAGE_TYPE_ID = :ln ) AND I.ITEM_STATUS_ID != :ln AND
O.PURCHASE_STATUS_ID != 1 ) SELECT D.PURCHASE_DT, S.LONG_DESC_TXT AS SUPPLIER,
D.PURCHASE_ID, D.PURCHASE_FORM, D.PURCHASE_NBR, D.TRACKING_NBR, D.STATUS_MSG, D.QUANTITY,
D.ORIENTATION, D.PURCHASE_ID, D.SUPPLIER_ID, D.PACKAGE_ID, D.SEQUENCE_NBR, D.MASTER_ID,
D.PRODUCT_ID, D.ITEM_ID, D.ITEM_STATUS_ID, FRST.SHORT_DESC_TXT AS ITEM_STATUS_CD,
FRST.LONG_DESC_TXT AS DETAIL_STATUS_CD, FRST.SEQUENCE_NBR AS STATUS_PRIORITY,
LIQUOR.THUMB_LIQUOR_URL AS STATUS_ICON, M.LONG_DESC_TXT AS PRODUCT, CASE M.MASTER_TYPE_ID
WHEN 1 THEN ( SELECT F.SHORT_DESC_TXT FROM F_FRAME F WHERE F.FRAME_ID = D.PRODUCT_ID) WHEN
2 THEN ( SELECT C.SHORT_DESC_TXT FROM F_CLIPON C WHERE C.CLIPONID = D.PRODUCT_ID) WHEN 3
THEN ( SELECT CASE D.ORIENTATION WHEN 'L' THEN U.LONG_DESC_TXT || ' (OS)' WHEN 'R' THEN
U.LONG_DESC_TXT || ' (OD)' ELSE U.LONG_DESC_TXT END FROM C_CONTACT_BOTTLE_UNIT U WHERE
U.CONTACT_BOTTLE_UNITID = D.PRODUCT_ID) WHEN 4 THEN 'Unknown' WHEN 5 THEN 'Unknown' WHEN 6
THEN ( SELECT L.SHORT_DESC_TXT || ' (' || T.SHORT_DESC_TXT || ')' FROM L_LABEL_SERVICE L,
L_LABEL_SERVICE_TYPE T WHERE L.LABEL_SERVICE_ID = D.PRODUCT_ID AND T.LABEL_SERVICE_TYPE_ID
= L.LABEL_SERVICE_TYPE_ID) WHEN 7 THEN ( SELECT L.LONG_DESC_TXT || ' (' ||
O.SHORT_DESC_TXT || ')' FROM O_FINAL_BOTTLE_PURCHASE LO, L_FINAL_BOTTLE L,
L_BOTTLE_ORIENTATION O WHERE LO.FINAL_BOTTLE_PURCHASE_ID = D.PRODUCT_ID AND
L.FINAL_BOTTLE_ID = LO.FINAL_BOTTLE_ID AND O.BOTTLE_ORIENTATION_ID = LO.ORIENTATION_ID)
WHEN 8 THEN ( SELECT P.SHORT_DESC_TXT FROM F_PART P WHERE P.PART_ID = D.PRODUCT_ID) WHEN 9
THEN ( SELECT A.SHORT_DESC_TXT FROM P_ASSORTED A WHERE A.ASSORTED_ID = D.PRODUCT_ID) ELSE
'Unknown' END AS DESCRIPTION, CASE M.MASTER_TYPE_ID WHEN 3 THEN ( SELECT CASE WHEN
D.ORIENTATION IN ('F', 'B') THEN COALESCE((SELECT DISTINCT '**NO COST**' FROM
O_PACKAGE_REFERENCE PR, O_REFERENCE R WHERE PR.PACKAGE_ID = D.PACKAGE_ID AND
R.REFERENCE_ID = PR.REFERENCE_ID AND R.REFERENCE_TYPE_ID = :ln), '') END FROM
C_CONTACT_BOTTLE_UNIT U WHERE U.CONTACT_BOTTLE_UNITID = D.PRODUCT_ID) ELSE '' END AS
FREE_INDICATOR FROM SUMMARYD D, O_ITEM_STATUS_CD FRST, P_LIQUOR IMG, P_SUPPLIER S,
P_MASTER M WHERE FRST.ITEM_STATUS_ID = D.ITEM_STATUS_ID AND LIQUOR.LIQUOR_ID =
FRST.LIQUOR_ID AND S.SUPPLIER_ID = D.SUPPLIER_ID AND M.MASTER_ID = D.MASTER_ID AND
UPPER(D.PURCHASE_FORM) LIKE ? PURCHASE BY DATE(D.PURCHASE_DT) DESC, SUPPLIER,
D.PURCHASE_ID, D.PURCHASE_NBR, D.PURCHASE_FORM, D.PACKAGE_ID, D.SEQUENCE_NBR ;
```

This sample monster recursive query is used in daily production by the world's largest provider for private practice eyecare professionals; (courtesy the folks at www.eyefinity.com).

-- -- --

RECURSION SQL:

-- -- --

NEATNESS IS REQUIRED !

ALIGN FOR SELF-DOCUMENTATION !

-- -- --

Review of Oracle "WITH" query_name Clause:

-- -- --

Parlez-Vous Klingon? Recursion SQL for Database Magicians

Oracle's WITH query_name clause allows you assign a NAME to a sub-query block.

-- -- --

Here's some EXAMPLES WITH SYS.DUAL Usage:

(Single Alias) query_name :

-- -- --

```
WITH o1 AS (SELECT * FROM sys.dual)      SELECT * FROM o1 ;
```

```
WITH x AS (SELECT * FROM sys.dual)      SELECT * FROM x ;
```

```
WITH x AS (SELECT * FROM sys.dual)      SELECT * FROM x ;
```

```
with x as (select 99 from sys.dual)      select * from x ;
```

-- -- --

Here's the corresponding SYS.DUAL Query Results:

-- -- --

Connected to:

Oracle Database 10g Enterprise Edition Release 10.2.0.2.0 - Production

With the Partitioning, OLAP and Data Mining options

```
D
-
X
```

```
D
-
X
```

```
D
-
X
```

```
          99
-----
          99
```

-- -- --

Here's a script that uses the Oracle "WITH" Clause:

Parlez-Vous Klingon? Recursion SQL for Database Magicians

Here the Recursive query is used to load a table :

-- -- --

-- =====

```
-- SQLPLUS alex/alex @i00.SQL > i00.TXT
-- NOTEPAD i00.TXT
```

```
SELECT '00', SYSTIMESTAMP FROM SYS.DUAL ;
```

```
SELECT USER FROM SYS.DUAL ;
SELECT CAST ( '22-DEC-2008' AS TIMESTAMP WITH LOCAL TIME ZONE ) FROM SYS.DUAL ;
```

```
SELECT '01', SYSTIMESTAMP FROM SYS.DUAL ;
```

```
WITH RECURSIVE_TEMP_TBL_A AS ( SELECT 99999 AS USER_NBR , 'John' AS FIRST_NAME ,
'Adams' AS LAST_NAME , TO_DATE ( '28.02.1996' , 'DD.MM.YYYY' ) AS DATE_OF_HIRE FROM
SYS.DUAL UNION SELECT 99999 USER_NBR , 'Benjamin' FIRST_NAME , 'Franklin' LAST_NAME ,
TO_DATE ( '28.02.1976' , 'DD.MM.YYYY' ) DATE_OF_HIRE FROM SYS.DUAL UNION SELECT 99999
USER_NBR , 'Theodore' FIRST_NAME , 'Roosevelt' LAST_NAME , TO_DATE ( '28.02.1905'
, 'DD.MM.YYYY' ) DATE_OF_HIRE FROM SYS.DUAL UNION SELECT 99999 USER_NBR , 'Patrick'
FIRST_NAME , 'Henry' LAST_NAME , TO_DATE ( '28.02.1970' , 'DD.MM.YYYY' ) DATE_OF_HIRE FROM
SYS.DUAL ) SELECT * FROM RECURSIVE_TEMP_TBL_A ;
```

```
SELECT '02', SYSTIMESTAMP FROM SYS.DUAL ;
```

```
EXIT ;
```

-- =====

-- -- --

Here are the Recursive query load table RESULTS:

-- -- --

-- =====

SQL*Plus: Release 10.2.0.2.0 - Production on Sun Feb 24 21:32:58 2008

Copyright (c) 1982, 2005, Oracle. All Rights Reserved.

Connected to:

Oracle Database 10g Enterprise Edition Release 10.2.0.2.0 - Production
with the Partitioning, OLAP and Data Mining options

```
'0 SYSTIMESTAMP
```

```
-----
00 24-FEB-08 09.32.58.625000 PM -06:00
```

```
USER
```

```
-----
ALEX
```

```
CAST('22-DEC-2008' AS TIMESTAMP WITH LOCAL TIME ZONE)
```

```
-----
22-DEC-20 08.00.00.000000 AM
```

Parlez-Vous Klingon? Recursion SQL for Database Magicians

```
'0 SYSTIMESTAMP
-----
01 24-FEB-08 09.32.58.640000 PM -06:00

  USER_NBR FIRST_NA LAST_NAME DATE_OF_H
-----
    99999 Benjamin Franklin 28-FEB-76
    99999 John      Adams    28-FEB-96
    99999 Patrick  Henry   28-FEB-70
    99999 Theodore Roosevelt 28-FEB-05
```

```
'0 SYSTIMESTAMP
-----
02 24-FEB-08 09.32.58.640000 PM -06:00
```

Disconnected from Oracle Database 10g Enterprise Edition Release 10.2.0.2.0 - Production
With the Partitioning, OLAP and Data Mining options

```
-- =====
```

```
-- -- --
```

Here's same query with COMMON SYNTAX error - "explicitly specifying logon/schema name":

```
-- -- --
```

```
SELECT USER FROM SYS.DUAL ;
```

```
WITH ALEX.RECURSIVE_TEMP_TBL_A AS ( SELECT 99999 AS USER_NBR , 'John' AS FIRST_NAME ,
'Adams' AS LAST_NAME , TO_DATE ( '28.02.1996' , 'DD.MM.YYYY' ) AS DATE_OF_HIRE FROM
SYS.DUAL UNION SELECT 99999 USER_NBR , 'Benjamin' FIRST_NAME , 'Franklin' LAST_NAME ,
TO_DATE ( '28.02.1976' , 'DD.MM.YYYY' ) DATE_OF_HIRE FROM SYS.DUAL UNION SELECT 99999
USER_NBR , 'Theodore' FIRST_NAME , 'Roosevelt' LAST_NAME , TO_DATE ( '28.02.1905'
, 'DD.MM.YYYY' ) DATE_OF_HIRE FROM SYS.DUAL UNION SELECT 99999 USER_NBR , 'Patrick'
FIRST_NAME , 'Henry' LAST_NAME , TO_DATE ( '28.02.1970' , 'DD.MM.YYYY' ) DATE_OF_HIRE FROM
SYS.DUAL ) SELECT * FROM RECURSIVE_TEMP_TBL_A ;
```

```
USER
```

```
-----
ALEX
```

```
WITH ALEX.RECURSIVE_TEMP_TBL_A AS ( SELECT 99999 AS USER_NBR , 'John' AS FIRST_NAME ,
'Adams' AS LAST_NAME , TO_DATE ( '28.02.1996' , 'DD.MM.YYYY' ) AS DATE_OF_HIRE FROM
SYS.DUAL UNION SELECT 99999 USER_NBR , 'Benjamin' FIRST_NAME , 'Franklin' LAST_NAME ,
TO_DATE ( '28.02.1976' , 'DD.MM.YYYY' ) DATE_OF_HIRE FROM SYS.DUAL UNION SELECT
99999 USER_NBR , 'Theodore' FIRST_NAME , 'Roosevelt' LAST_NAME , TO_DATE (
'28.02.1905' , 'DD.MM.YYYY' ) DATE_OF_HIRE FROM SYS.DUAL UNION SELECT 99999 USER_NBR ,
'Patrick' FIRST_NAME , 'Henry' LAST_NAME , TO_DATE ( '28.02.1970' , 'DD.MM.YYYY' )
DATE_OF_HIRE FROM SYS.DUAL ) SELECT * FROM RECURSIVE_TEMP_TBL_A
*
```

```
ERROR at line 1:
ORA-00905: missing keyword
```

It just doesn't like ALEX !

```
-- -- --
```

Here's a review of DB2 "WITH" Clause

```
-- -- --
```

Parlez-Vous Klingon? Recursion SQL for Database Magicians

The WITH clause in DB2 defines a Common Table Expression (CTE) that is defined at the beginning of a recursion query (available in DB2 LUW since version 5 ; in DB2 for z/OS since V8). Once defined, CTE's can be referenced by NAME within the rest of the SQL statement. (Do not confuse "Isolation Level" specification " WITH CS or WITH UR").

-- -- --

Here's a quick Review of Oracle "Connect by Prior" for performing recursion:

-- -- --

Here's the "Connect by Prior" setup Oracle SQL:

-- -- --

```
CREATE TABLE ALEX.DEPARTMENTS2 ( DEPT_ID INTEGER, DEPT_NAME VARCHAR2(30), PART_COUNT
INTEGER, SUPER_DEPT INT ) ;
INSERT INTO ALEX.DEPARTMENTS2 VALUES (1, 'Production Dayshift', 10, 05 ) ;
INSERT INTO ALEX.DEPARTMENTS2 VALUES (2, 'Production Dayshift', 20, 05 ) ;
INSERT INTO ALEX.DEPARTMENTS2 VALUES (3, 'Production Dayshift', 30, 05 ) ;
INSERT INTO ALEX.DEPARTMENTS2 ( DEPT_ID , DEPT_NAME , PART_COUNT , SUPER_DEPT ) VALUES (1,
'Production Nightshift', 40, 05 ) ;
INSERT INTO ALEX.DEPARTMENTS2 ( DEPT_ID , DEPT_NAME , PART_COUNT , SUPER_DEPT ) VALUES (2,
'Production Nightshift', 50, 05 ) ;
INSERT INTO ALEX.DEPARTMENTS2 ( DEPT_ID , DEPT_NAME , PART_COUNT , SUPER_DEPT ) VALUES (3,
'Production Nightshift', 60, 05 ) ;
SELECT * FROM ALEX.DEPARTMENTS2 ;
```

-- -- --

Here's the " CONNECT BY PRIOR " Recursion SQL in Oracle Example:

-- -- --

```
SELECT SUM(PART_COUNT) FROM ALEX.DEPARTMENTS2 CONNECT BY PRIOR SUPER_DEPT = DEPT_ID START
WITH DEPT_NAME = 'Production Dayshift';
EXIT ;
```

-- -- --

Here's the " CONNECT BY PRIOR " Recursion SQL in Oracle Results :

-- -- --

| DEPT_ID | DEPT_NAME | PART_COUNT | SUPER_DEPT |
|---------|-----------------------|------------|------------|
| 1 | Production Dayshift | 10 | 5 |
| 2 | Production Dayshift | 20 | 5 |
| 3 | Production Dayshift | 30 | 5 |
| 1 | Production Nightshift | 40 | 5 |
| 2 | Production Nightshift | 50 | 5 |
| 3 | Production Nightshift | 60 | 5 |

Parlez-Vous Klingon? Recursion SQL for Database Magicians

6 rows selected.

```
SUM(PART_COUNT)
-----
              60
```

-- -- --

Here's an equivalent in DB2 "Recursion using CTE" SQL

-- -- --

Here's the setup DB2 SQL:

-- -- ----- -- --

```
connect to samp195 ;
```

```
DROP TABLE ALEX.DEPARTMENTS ;
```

```
DROP TABLE ALEX.DEPARTMENTS2 ;
```

```
CREATE TABLE ALEX.DEPARTMENTS ( DEPT_ID INTEGER, DEPT_NAME VARCHAR(30), PART_COUNT
INTEGER, SUPER_DEPT INT ) ;
```

```
INSERT INTO ALEX.DEPARTMENTS VALUES (1, 'Production Dayshift', 10, 05 ) ;
```

```
INSERT INTO ALEX.DEPARTMENTS VALUES (2, 'Production Dayshift', 20, 05 ) ;
```

```
INSERT INTO ALEX.DEPARTMENTS VALUES (3, 'Production Dayshift', 30, 05 ) ;
```

```
INSERT INTO ALEX.DEPARTMENTS ( DEPT_ID , DEPT_NAME , PART_COUNT , SUPER_DEPT ) VALUES (1,
'Production Nightshift', 40, 05 ) ;
```

```
INSERT INTO ALEX.DEPARTMENTS ( DEPT_ID , DEPT_NAME , PART_COUNT , SUPER_DEPT ) VALUES (2,
'Production Nightshift', 50, 05 ) ;
```

```
INSERT INTO ALEX.DEPARTMENTS ( DEPT_ID , DEPT_NAME , PART_COUNT , SUPER_DEPT ) VALUES (3,
'Production Nightshift', 60, 05 ) ;
```

```
SELECT * FROM ALEX.DEPARTMENTS ;
```

-- -- --

Here's the equivalent "CTE" DB2 SQL:

-- -- --

```
WITH ALEX.TEMPTAB ( DEPT_ID, PART_COUNT, SUPER_DEPT) AS ( SELECT TOP.DEPT_ID,
TOP.PART_COUNT, TOP.SUPER_DEPT FROM ALEX.DEPARTMENTS TOP WHERE DEPT_NAME='Production
Dayshift' UNION ALL SELECT BOTTOM.DEPT_ID, BOTTOM.PART_COUNT, BOTTOM.SUPER_DEPT FROM
ALEX.DEPARTMENTS BOTTOM, ALEX.TEMPTAB SUPER WHERE BOTTOM.SUPER_DEPT = SUPER.DEPT_ID )
SELECT SUM(PART_COUNT) FROM ALEX.TEMPTAB ;
```

```
terminate ;
```

-- -- --

Here's the equivalent DB2 "Recursion using CTE" SQL Results:

Parlez-Vous Klingon? Recursion SQL for Database Magicians

--- --

connect to samp195

| DEPT_ID | DEPT_NAME | PART_COUNT | SUPER_DEPT |
|---------|-----------------------|------------|------------|
| 1 | Production Dayshift | 10 | 5 |
| 2 | Production Dayshift | 20 | 5 |
| 3 | Production Dayshift | 30 | 5 |
| 1 | Production Nightshift | 40 | 5 |
| 2 | Production Nightshift | 50 | 5 |
| 3 | Production Nightshift | 60 | 5 |

6 record(s) selected.

```
WITH ALEX.TEMP TAB ( DEPT_ID, PART_COUNT, SUPER_DEPT) AS ( SELECT TOP.DEPT_ID,
TOP.PART_COUNT, TOP.SUPER_DEPT FROM ALEX.DEPARTMENTS TOP WHERE DEPT_NAME='Production
Dayshift' UNION ALL SELECT BOTTOM.DEPT_ID, BOTTOM.PART_COUNT, BOTTOM.SUPER_DEPT FROM
ALEX.DEPARTMENTS BOTTOM, ALEX.TEMP TAB SUPER WHERE BOTTOM.SUPER_DEPT = SUPER.DEPT_ID )
SELECT SUM(PART_COUNT) FROM ALEX.TEMP TAB
```

1

SQL0347W The recursive common table expression "ALEX.TEMP TAB" may contain an infinite loop. SQLSTATE=01605

60

1 record(s) selected with 1 warning messages printed.

--- --

Here's the " Order By OVER " Recursion SQL in Oracle Example:

--- --

```
-- SQLPLUS alex/alex @AA9.SQL > AA9.TXT
```

```
DROP TABLE ALEX.TBL1 ;
COMMIT ;
```

```
CREATE TABLE ALEX.TBL1 ( GIMME_A_# INTEGER ) ;
COMMIT ;
```

```
DESCRIBE ALEX.TBL1 ;
```

```
INSERT into ALEX.TBL1 values (1) ;
```

```
INSERT into ALEX.TBL1 values (2) ;
```

```
INSERT into ALEX.TBL1 values (4) ;
```

```
INSERT into ALEX.TBL1 values (5) ;
```

```
INSERT into ALEX.TBL1 values (3) ;
```

```
COMMIT ;
```

```
select MIN ( GIMME_A_#)
```

```
over (order by GIMME_A_# desc) from ALEX.TBL1 ;
```

Parlez-Vous Klingon? Recursion SQL for Database Magicians

```
select max ( GIMME_A_#)
       over (order by  GIMME_A_#)      from ALEX.TBL1 ;
```

```
select
  max ( GIMME_A_#)
  over (order by  GIMME_A_#),
  min ( GIMME_A_#)
  over (order by  GIMME_A_# desc)
from ALEX.TBL1 ;
```

-- -- --

Here's the " Order By OVER " Recursion SQL in Oracle Results :

-- -- --

| Name | Null? | Type |
|-----------|-------|------------|
| GIMME_A_# | | NUMBER(38) |

MIN(GIMME_A_#)OVER(ORDERBYGIMME_A_#DESC)

| |
|---|
| 5 |
| 4 |
| 3 |
| 2 |
| 1 |

MAX(GIMME_A_#)OVER(ORDERBYGIMME_A_#)

| |
|---|
| 1 |
| 2 |
| 3 |
| 4 |
| 5 |

MAX(GIMME_A_#)OVER(ORDERBYGIMME_A_#)

Parlez-Vous Klingon? Recursion SQL for Database Magicians

```
-----  
                    5  
                    4  
                    3  
                    2  
                    1  
MIN(GIMME_A_#)OVER(ORDERBYGIMME_A_#DESC)  
-----  
                    5  
                    4  
                    3  
                    2  
                    1
```

-- -- --

Here's a review of Oracle SYS.DUAL Table:

SYS.DUAL USAGE IS COMMON IN LEARNING & USING Oracle RECURSION STMTS.

-- -- --

SYS.DUAL table is a special in-memory table that can be used to show values in the Oracle registers.

-- -- --

SYS.DUAL Examples:

-- -- --

```
SQL> DESCRIBE TABLE SYS.DUAL
```

| NAME | Null? | Type |
|-------|-------|-------------|
| DUMMY | | VARCHAR2(1) |

-- -- --

```
SQL> SELECT USER FROM dual ;
```

-- -- --

```
USER
```

```
-----  
ALEX
```

Parlez-Vous Klingon? Recursion SQL for Database Magicians

-- -- --

Here's more notes on Oracle's SYS.DUAL :

-- -- --

Table SYS.DUAL gets created during Oracle database creation.

SYS.DUAL has a schema name user SYS, but is accessible by the name DUAL to all users.

SYS.DUAL contains only ONE column, DUMMY (defined as VARCHAR2(1)) and ONE row with a value 'X'. ***

Sometimes, SYS.DUAL can have more than one row.

The kernel knows that SYS.DUAL is a single row, single column table, but PL/SQL engine doesn't know.

Reportedly, some Oracle Applications patches can cause multiple rows to be placed into the SYS.DUAL table.

Note: Do not INSERT/DELETE/UPDATE SYS.DUAL !

The ADADMIN utility gives you options to check the SYS.DUAL table for multiple rows that can cause issues.

-- -- --

Here's the equivalent in DB2 - SYSIBM.SYSDUMMY1 View/Table

SYSIBM.SYSDUMMY1 USAGE IS COMMON IN LEARNING & USING DB2 RECURSION STMTS

-- -- --

SYSIBM.SYSDUMMY1 view/table is a special in-memory table that can be used to show values in the DB2 registers.

-- -- --

```
C:\Program Files\IBM\SQLLIB\BIN> DB2 DESCRIBE TABLE SYSIBM.SYSDUMMY1
```

```
Column NAME  schema      Data type NAME  Length  Scale  Nulls
```

```
-----  
IBMREQD      SYSIBM      CHARACTER          1       0      No
```

```
1 record(s) selected.
```

-- -- --

```
C:\Program Files\IBM\SQLLIB\BIN> db2 select user from sysibm.sysdummy1
```

```
C:\Program Files\IBM\SQLLIB\BIN> db2 select current sqlid from sysibm.sysdummy1
```

```
1
```

Parlez-Vous Klingon? Recursion SQL for Database Magicians

ALEXANDER KOPAC

-- -- --

-- -- --

Here's a review of Oracle "WITH" query_name Clause

-- -- --

Oracle's WITH query_name clause allows you assign a NAME to a sub-query block

SYS.DUAL Usage:

(Single Alias) query_name :

WITH o1 AS (SELECT * FROM dual) SELECT * FROM o1 ;

WITH x AS (SELECT * FROM dual) SELECT * FROM x ;

WITH x AS (SELECT * FROM sys.dual) SELECT * FROM x ;

with x as (select 99 from dual) select * from x ;

SQL> WITH o1 AS (SELECT SYSDATE FROM SYS.DUAL) SELECT * FROM o1 ;

-- -- --

More Review of Oracle "CONNECT BY" & SYS.DUAL USAGE

-- -- --

SQL> select * from

 (select level as lvl from sys.dual connect by level <= 7) ;

 LVL

 1

 2

 3

 4

 5

 6

 7

7 rows selected.

-- -- --

Parlez-Vous Klingon? Recursion SQL for Database Magicians

Review of Oracle "CONNECT BY" & SYS.DUAL USAGE

-- -- --

```
SQL> select * from
      ( select level as lvl from sys.dual connect by level <= 7 ) ;
```

LVL

1
2
3
4
5
6
7

7 rows selected.

-- -- --

```
SQL> select * from
      ( select level as lvl from sys.dual connect by level <= 7 ) ,
      ( select level as lvl from sys.dual connect by level <= 7 ) ;
```

-- -- --

Note: 49 rows selected.

-- -- --

Review of Oracle DBMS_RANDOM.* Package

-- -- --

DBMS_RANDOM package available Oracle version 8+ is faster than generators written in PL/SQL since DBMS_RANDOM calls Oracle's internal random number generator.

-- -- --

Review of Oracle "DBMS_RANDOM.STRING" & SYS.DUAL USAGE

```
dbms_random.string(opt, len) ;
```

-- -- --

DBMS_RANDOM.STRING function generates random character strings.

First parameter is OPT that allows modest control over the type of characters being generated.

Parlez-Vous Klingon? Recursion SQL for Database Magicians

The following are the allowable values to be passed as the OPT parameter.

```
'a','A' : Alpha characters only - Mixed case
'u','U' : Upper case alpha characters only
'l','L' : Lower case alpha characters only
'p','P' : Any printable characters
'x','X' : Any alpha-numeric characters (upper)
```

Second parameter is the number of characters to be generated.

The maximum value is 4000.

-- -- --

This generates 45 random Printable characters:

-- -- --

```
SELECT DBMS_RANDOM.STRING('P',45) VALUE FROM SYS.DUAL ;
```

This generates 45 random Uppercase characters:

```
SELECT DBMS_RANDOM.STRING('U',45) VALUE FROM SYS.DUAL ;
```

-- -- --

Review of Oracle DBMS_RANDOM.STRING & SYS.DUAL USAGE

```
dbms_random.STRING ( opt, len );
```

-- -- --

```
SQL> SELECT DBMS_RANDOM.STRING('U',45) VALUE FROM SYS.DUAL ;
```

VALUE

CPUNVQMSHIZTSWIQUIUWINNBOACQLPBUCALBUWGGGJWLT

```
SQL> SELECT DBMS_RANDOM.STRING('P',45) VALUE FROM SYS.DUAL ;
```

VALUE

qmEQn.\SFr_Zr.:4RPEAFkOeYs ^^;'A?I(q79}~wkbtV

-- -- --

Review of Oracle DBMS_RANDOM.STRING & SYS.DUAL USAGE

```
dbms_random.STRING ( opt, len );
```

-- -- --

Parlez-Vous Klingon? Recursion SQL for Database Magicians

```
SQL> select dbms_random.string('U', 20) as xxx from dual ;
```

```
XXX
```

```
-----  
LNZBRIUVBTQDPLEPFOZC
```

```
SQL> select dbms_random.string('U', 20) yyy from dual ;
```

```
YYY
```

```
-----  
KFJGYRHSSIQXUPOWUQOW
```

```
SQL> WITH o1 AS (select dbms_random.string('U', 20) as xxx from sys.dual ) SELECT * FROM  
o1 ;
```

```
XXX
```

```
-----  
PXXSIFQMDPVBEHEJFGTK
```

```
SQL> WITH o1 AS (select dbms_random.string('P', 20) YYY from sys.dual) SELECT * FROM o1 ;
```

```
YYY
```

```
-----  
]Gep@2}C'bKnMsmG#e%
```

```
-- -- --
```

Review of Oracle DBMS_RANDOM.STRING & SYS.DUAL USAGE

```
dbms_random.STRING ( opt, len );
```

```
-- -- --
```

```
DROP TABLE POSTAL_CODE ;
```

```
CREATE TABLE POSTAL_CODE (
```

```
ZIP_CODE          VARCHAR2(5)  NOT NULL,
```

```
STATE_ABBREV      VARCHAR2(2)  NOT NULL,
```

```
STATE_NAME        VARCHAR2(30) NOT NULL,
```

```
CITY_NAME         VARCHAR2(30)  ) ;
```

```
INSERT INTO POSTAL_CODE VALUES ('12345', 'NV', 'Nevada', 'Las Vegas') ;
```

```
INSERT INTO POSTAL_CODE VALUES ('12346', 'NV', 'Nevada', 'Las Cruces') ;
```

```
INSERT INTO POSTAL_CODE VALUES ('12346', 'NV', 'Nevada', (select dbms_random.string('P', 30)  
from sys.dual)) ;
```

```
INSERT INTO POSTAL_CODE VALUES ('12347', 'NV', 'Nevada', (select dbms_random.string('P', 30)  
from sys.dual)) ;
```

Parlez-Vous Klingon? Recursion SQL for Database Magicians

```
INSERT INTO POSTAL_CODE VALUES (  
  (MOD(ABS(DBMS_RANDOM.RANDOM),100000) + 10000 ),  
  (SELECT DBMS_RANDOM.STRING('U',2) VALUE FROM SYS.DUAL),  
  (SELECT DBMS_RANDOM.STRING('U',30) VALUE FROM SYS.DUAL),  
  (SELECT DBMS_RANDOM.STRING('P',30) VALUE FROM SYS.DUAL) ) ;  
COMMIT ;  
SELECT * FROM POSTAL_CODE ;
```

```
=====
```

| ZIP_C | ST | STATE_NAME | CITY_NAME |
|-------|----|--------------------------------|----------------------------------|
| 34961 | CU | GQGMUCRFVOLGLYATPILTZBARFMLEAO | 2? \>u7wpxhhd=D,A.FjJ)LJ,` `n |

Review of Oracle DBMS_RANDOM.STRING & SYS.DUAL USAGE

```
  dbms_random.STRING ( opt, len );  
-- -- --  
DROP TABLE POSTAL_CODE ;  
DROP TABLE ZIPCODE_NEW ;  
CREATE TABLE POSTAL_CODE (  
  ZIP_CODE          VARCHAR2(5) NOT NULL,  
  STATE_ABBREV      VARCHAR2(2) NOT NULL,  
  STATE_NAME        VARCHAR2(30) NOT NULL,  
  CITY_NAME         VARCHAR2(30) ) ;  
CREATE TABLE ZIPCODE_NEW (  
  ZIP_CODE          VARCHAR2(5) NOT NULL,  
  STATE_ABBREV      VARCHAR2(2) NOT NULL,  
  CITY_NAME         VARCHAR2(30) ) ;  
INSERT INTO POSTAL_CODE VALUES ('12345', 'NV', 'Nevada', 'Las Vegas') ;  
INSERT INTO POSTAL_CODE VALUES ('12346', 'NV', 'Nevada', 'Las Cruces') ;  
INSERT INTO POSTAL_CODE VALUES ('12346', 'NV', 'Nevada', (select dbms_random.string('P',  
30) from dual)) ;  
INSERT INTO POSTAL_CODE VALUES ('12347', 'NV', 'Nevada', (select dbms_random.string('P',  
30) from dual)) ;
```

Parlez-Vous Klingon? Recursion SQL for Database Magicians

```
INSERT INTO ZIPCODE_NEW      SELECT ZIP_CODE, STATE_ABBREV, CITY_NAME  FROM POSTAL_CODE ;
SELECT * FROM ZIPCODE_NEW ;
SELECT * FROM POSTAL_CODE ;
exit ;
```

```
=====
ZIP_C ST CITY_NAME
-----
```

```
12345 NV Las Vegas
```

```
12346 NV Las Cruces
```

```
12346 NV w(p[ ;Y]#DwB*`we~fLQ1^Tt^&6b"!h
```

```
12347 NV ka/Cs*1?t$St0<auWa()jcuw"~?T^<
```

```
ZIP_C ST STATE_NAME          CITY_NAME
-----
```

```
12345 NV Nevada
```

```
Las Vegas
```

```
12346 NV Nevada
```

```
Las Cruces
```

```
12346 NV Nevada
```

```
w(p[ ;Y]#DwB*`we~fLQ1^Tt^&6b"!h
```

```
12347 NV Nevada
```

```
ka/Cs*1?t$St0<auWa()jcuw"~?T^<
```

```
-- -- --
```

Review of Oracle DBMS_RANDOM.STRING & SYS.DUAL USAGE

```
dbms_random.STRING ( opt, len );
```

```
-- -- --
```

You can use DBMS_RANDOM.STRING to generate random strings.

1ST parameter is "opt" which has some control over the type of characters being generated. It returns a string whose length is determined by the "len" argument ; ("opt" must be character). "len" max is 4000.

The following five options are available:

'a' or 'A': Upper and lower case alpha
characters

'l' or 'L': Lower case alpha characters

'p' or 'P': Any printable character

'u' or 'U': Upper case alpha characters

'x' or 'X': Upper alpha and numeric characters

Parlez-Vous Klingon? Recursion SQL for Database Magicians

-- -- --

Review of Oracle DBMS_RANDOM.STRING & SYS.DUAL USAGE

```
dbms_random.STRING ( opt, len );
```

-- -- --

You can use DBMS_RANDOM.STRING to generate random strings.

1ST parameter is "opt" which has some control over the type of characters being generated. It returns a string whose length is determined by the "len" argument ; "opt" must be character. "len" max is 4000

-- -- --

Review of Oracle DBMS_RANDOM.VALUE & SYS.DUAL USAGE

```
dbms_random.VALUE ( low#, high# );
```

-- -- --

```
-- DBMS_RANDOM.VALUE
```

```
col a format 9999
```

```
col l format 9999
```

```
col p format 9999
```

```
col u format 9999
```

```
col x format 9999
```

```
SELECT
```

```
DBMS_RANDOM.VALUE (          ) a,
```

```
DBMS_RANDOM.VALUE ( 0, 10 ) l,
```

```
DBMS_RANDOM.VALUE ( 99, 1000 ) p,
```

```
DBMS_RANDOM.VALUE ( 900, 9000 ) u,
```

```
DBMS_RANDOM.VALUE ( 9, 9999 ) x
```

```
FROM SYS.DUAL ;
```

```
SELECT
```

```
DBMS_RANDOM.VALUE (          ) ,
```

```
DBMS_RANDOM.VALUE ( 0, 10 ) ,
```

```
DBMS_RANDOM.VALUE ( 99, 1000 ) ,
```

```
DBMS_RANDOM.VALUE ( 900, 9000 ) ,
```

```
DBMS_RANDOM.VALUE ( 9, 9999 )
```

```
FROM SYS.DUAL ;
```

Parlez-Vous Klingon? Recursion SQL for Database Magicians

-- -- --

Review of Oracle DBMS_RANDOM.VALUE & SYS.DUAL USAGE

```
dbms_random.VALUE ( opt, len );
```

-- -- --

```
DBMS_RANDOM.VALUE
```

```
DBMS_RANDOM.VALUE ()
```

```
.32872613
```

```
DBMS_RANDOM.VALUE (0,10)
```

```
2.59607408
```

```
DBMS_RANDOM.VALUE (99,1000)
```

```
936.08742
```

-- -- --

Review of Oracle DBMS_RANDOM.VALUE & SYS.DUAL USAGE

```
dbms_random.VALUE ( opt, len );
```

-- -- --

-- Generate a random number between 1 & 100 :

```
SELECT DBMS_RANDOM.VALUE ( 1, 100) VALUE FROM SYS.DUAL ;
```

-- You can use TRUNC OR ROUND SQL functions to remove DECIMALS :

```
SELECT TRUNC (DBMS_RANDOM.VALUE  
  ( 1, 100)) VALUE FROM SYS.DUAL ;
```

```
SELECT ROUND (DBMS_RANDOM.VALUE  
  ( 1, 100)) VALUE FROM SYS.DUAL ;
```

-- Generate a random number between

1000 & 9999 :

```
SELECT ROUND (DBMS_RANDOM.VALUE  
  ( 1000, 9999)) VALUE FROM SYS.DUAL ;
```

Parlez-Vous Klingon? Recursion SQL for Database Magicians

-- -- --

Review of Oracle DBMS_RANDOM.RANDOM & SYS.DUAL USAGE

```
dbms_random.RANDOM ( opt, len );
```

-- -- --

```
MOD(ABS(DBMS_RANDOM.RANDOM),10)
```

9

```
MOD(ABS(DBMS_RANDOM.RANDOM),100)
```

62

```
MOD(ABS(DBMS_RANDOM.RANDOM),1000)
```

376

```
MOD(ABS(DBMS_RANDOM.RANDOM),10000)
```

3114

```
MOD(ABS(DBMS_RANDOM.RANDOM),100000)
```

29351

-- -- --

Review of Oracle VALUES Function

-- -- --

```
DROP TABLE ALEX.TABLE1 ;
```

```
CREATE TABLE ALEX.TABLE1 (
```

```
pid          NUMBER (5),
```

```
first_name  VARCHAR2 (20),
```

```
last_name   VARCHAR2 (25) ) ;
```

```
INSERT ALL INTO ALEX.TABLE1
```

```
( pid, first_name, last_name )
```

```
VALUES (1, 'Annie', 'Johnson')
```

```
INTO ALEX.TABLE1
```

Parlez-Vous Klingon? Recursion SQL for Database Magicians

```
( pid, first_name, last_name)
VALUES (2, 'George', 'Thomas')
INTO ALEX.TABLE1
```

```
( pid, first_name, last_name)
VALUES (3, 'Gina', 'Bushnell')
```

```
SELECT * FROM SYS.DUAL ;
```

```
SELECT * FROM ALEX.TABLE1 ;
```

```
EXIT;
```

```
-- -- --
```

FINAL THOUGHTS & RECOMMENDATIONS

```
-- -- --
```

Be Careful Out There... Test for Loops !

Create Suitable Indexes !

Check for Suitable SQL Access Paths !

Test Each Generated Column Separately !

Establish ENDPOINTS - ENSURE NO LOOPS !

Practice Neatness !

Be Careful Out There... Test, Test, Test !

Parlez-Vous Klingon? Recursion SQL for Database Magicians

-- -- --

Future Questions:

-- -- --

Parlez-Vous Klingon? Recursion SQL for Database Magicians

-- -- --

```

      |
      \|/
     \|/|/
    \|/|/|/
   \|/|/|/|/
  \|/~ ~//
  ( @ @ )

```

```

+------( )-----+
| Alex.Kopac@DB-Performance.com
| Alexander.Kopac@DB-Performance.com
| ALEXANDER_KOPAC@HOTMAIL.COM
|
|           512-249-2324 X113 OFFICE
| http://www.DB-Performance.com/
|           ooooo
+------( )-----0oooo-----+
      \| ( (
       \_) ) (
           (_/

```

Every day above ground IS a good one !

-- -- --

Biography:

-- -- --

Alexander Kopac is a Database Magician and Customer Support Manager for DBI in Austin, Texas and an IBM Certified Solutions Expert. He has over 15 yrs of RDBMS experience. Previously, he was a consultant for more than 10 years working with automotive manufacturing, airline fares marketing, finance & insurance company marketing systems. Alexander has delivered many presentations for International DB2 User Group (IDUG) Regional User Groups, many IDUG Europe 2001-6 & IDUG North America conferences 2002-7, CA-World, IBM's DB2 Data Management Technical Conference 2002, INFOTEC 2002-5, Oracle & ISACA regional User Groups, BMC Userworld 2007 and many others. Lately, he's had a lot of fun helping customers with Oracle and DB2 LUW auditing, compliance and database performance issues. He's a board member of the Austin Oracle Users Group (AOUG) and an IDUG North America Conference Planning Committee (NACPC) member 2004-7.