File Edit View History Bookmarks Tools Help

SQL Functions

www.pgrocer.net/Cis50/oracle3.html

NUMERIC FUNCTIONS:

Numeric Functions	Processing
ROUND	Rounds numeric data
TRUNC	Truncates numeric data
MOD	Returns the numeric remainder that results from a divide
CEIL	Finds nearest whole number greater then or equal to the data/field being tested CEIL(data/field)
FLOOR	Finds nearest whole number less than or equal to the data/field being tested FLOOR(data/field)
POWER	Exponentiation - where m is the number and n is the power Power(m,n)
SQRT	Square root of n SQRT(data/field)
SIGN	Returns 0 if data 0, 1 if data >0 and -1 if data <0 SIGN(data)

When working with numbers the ROUND function is used to round a number to a certain number of digits/decimals and the TRUNC function is used to truncate a number to a certain number of digits/decimals. The format is ROUND or TRUNC and then in parenthesis the number or field, the number of digits). In the first example, I rounded and truncated a number to two decimal places using dual. In the second example, I rounded and truncated the result of dividing yrgoal by 12 to three decimal places.

SQL CODE:

```
SQL> SELECT ROUND(123.456,2), TRUNC(123.456,2)
  2  FROM dual;

ROUND(123.456,2) TRUNC(123.456,2)
---------------- ----------------
          123.46           123.45

SQL> SELECT ROUND(yrgoal/12,3), TRUNC(yrgoal/12,3), yrgoal/12
  2  FROM donor;

ROUND(YRGOAL/12,3) TRUNC(YRGOAL/12,3) YRGOAL/12
------------------ ------------------ ---------
            41.667             41.666 41.666667
            33.333             33.333 33.333333

             8.333              8.333 8.3333333
             4.167              4.166 4.1666667
             4.167              4.166 4.1666667

6 rows selected.
```

MOD allows you to code to return the remainder that occurs after division. The format is MOD(the number or the column you are dividing into, the number or the column that you are dividing by). In the first example, I am dividing 17 by 5 and the remainder of 2 is shown. In the second example, I am dividing yrgoal by 12 and returning the remainder for each row.

Type here to search

9:40 AM
9/12/2019

```
---------------- ----------------
         123.46           123.45

SQL> SELECT ROUND(yrgoal/12,3), TRUNC(yrgoal/12,3), yrgoal/12
  2  FROM donor;

ROUND(YRGOAL/12,3) TRUNC(YRGOAL/12,3) YRGOAL/12
------------------ ------------------ ---------
            41.667             41.666 41.666667
            33.333             33.333 33.333333

             8.333              8.333 8.3333333
             4.167              4.166 4.1666667
             4.167              4.166 4.1666667

6 rows selected.
```

MOD allows you to code to return the remainder that occurs after division. The format is MOD(the number or the column you are dividing into, the number or the column that you are dividing by). In the first example, I am dividing 17 by 5 and the remainder of 2 is shown. In the second example, I am dividing yrgoal by 12 and returning the remainder for each row.

SQL CODE:

```
SQL> SELECT MOD(17,5)
  2  FROM dual;

MOD(17,5)
---------
        2

SQL> SELECT yrgoal, MOD(yrgoal, 12), yrgoal/12
  2  FROM donor;

   YRGOAL MOD(YRGOAL,12) YRGOAL/12
--------- -------------- ---------
      500              8 41.666667
      400              4 33.333333

      100              4 8.3333333
       50              2 4.1666667
       50              2 4.1666667

6 rows selected.
```

CEIL and FLOOR are used to find the nearest whole number either greater than (CEIL) or less than (FLOOR) the decimal number being examined. In the example below I am looking for the ceiling and floor for the number 123.456.

SQL CODE:

```
SQL> SELECT CEIL(123.456), FLOOR(123.456)
  2  FROM dual;

CEIL(123.456) FLOOR(123.456)
------------- --------------
          124            123
```

File Edit View History Bookmarks Tools Help

SQL Functions

www.pgrocer.net/Cis50/oracle3.html

```
05-JUN-99 03-JAN-98
05-JUN-99 04-MAR-92
05-JUN-99 04-MAR-92
05-JUN-99 04-APR-98

6 rows selected.
```

Date Functions	Processing
ROUND	Rounds date according to specifications
TRUNC	Truncates date according to specifications
MONTHS_BETWEEN	Returns the number of months between two dates MONTHS_BETWEEN(date1,date2)
ADD_MONTHS	Increases/decreases months -n is number of months to increase or decrease ADD_MONTH(date,n)
NEXT_DAY	Next day of the week from date specified - day is the day you of the week you are matching date to NEXT_DAY(date,day)
LAST_DAY	Returns last day of the month

The ROUND and TRUNC functions round or truncate the date. With no entry the default is day, optionally MONTH or YEAR can be used with ROUND and TRUNC. Note MONTH and YEAR should be enclosed in single quotes.

SQL CODE:

```
SQL> SELECT sysdate, ROUND(sysdate), TRUNC(sysdate)
  2  FROM dual;

SYSDATE   ROUND(SYS TRUNC(SYS
--------- --------- ---------
05-JUN-99 06-JUN-99 05-JUN-99

SQL> SELECT datefst, ROUND(datefst,'MONTH'), TRUNC(datefst,'MONTH')
  2  FROM donor;

DATEFST   ROUND(DAT TRUNC(DAT
--------- --------- ---------
03-JUL-98 01-JUL-98 01-JUL-98
24-MAY-97 01-JUN-97 01-MAY-97
03-JAN-98 01-JAN-98 01-JAN-98
04-MAR-92 01-MAR-92 01-MAR-92
04-MAR-92 01-MAR-92 01-MAR-92
04-APR-98 01-APR-98 01-APR-98

SQL> SELECT datefst, ROUND(datefst, 'YEAR'), TRUNC(datefst, 'YEAR')
  2  FROM donor;

DATEFST   ROUND(DAT TRUNC(DAT
--------- --------- ---------
03-JUL-98 01-JAN-99 01-JAN-98
24-MAY-97 01-JAN-97 01-JAN-97
03-JAN-98 01-JAN-98 01-JAN-98
04-MAR-92 01-JAN-92 01-JAN-92
04-MAR-92 01-JAN-92 01-JAN-92
04-APR-98 01-JAN-98 01-JAN-98
```

Type here to search

9:45 AM 9/12/2019

Title: Sep 12-9:45 AM (3 of 26)

File Edit View History Bookmarks Tools Help

SQL Functions

www.pgrocer.net/Cis50/oracle3.html

```
05-JUN-99 07-JUN-99 04-MAR-92 09-MAR-92
05-JUN-99 07-JUN-99 04-MAR-92 09-MAR-92
05-JUN-99 07-JUN-99 04-APR-98 06-APR-98

6 rows selected.
```

The LAST_DAY function returns the last day of the month.

SQL CODE:

```
SQL> SELECT sysdate, LAST_DAY(sysdate), datefst, LAST_DAY(datefst)
  2  FROM donor;

SYSDATE   LAST_DAY( DATEFST   LAST_DAY(
--------- --------- --------- ---------
05-JUN-99 30-JUN-99 03-JUL-98 31-JUL-98
05-JUN-99 30-JUN-99 24-MAY-97 31-MAY-97
05-JUN-99 30-JUN-99 03-JAN-98 31-JAN-98
05-JUN-99 30-JUN-99 04-MAR-92 31-MAR-92
05-JUN-99 30-JUN-99 04-MAR-92 31-MAR-92
05-JUN-99 30-JUN-99 04-APR-98 30-APR-98
```

Conversion functions:

SQL includes functions that convert a value from one datatype to a different datatype. These functions all include a format model which allows a variety of return formats. Oracle can automatically convert VARCHAR2 to either NUMBER or DATE and can convert CHAR to NUMBER if the character string is a valid number and CHAR to DATE if the character string is in the default format of DD-MON-YY. For other conversions the following functions can be used.

Conversion function	Processing
TO_CHAR	A number or date can be converted to VARCHAR2 using the format specified. When converting to date, there is a fm toggle element within the fmt format that can be used to remove leading 0s or padding spaces. TO_CHAR(data to convert [,'fmt']) TO_CHAR(number to convert [,'fmt']
TO_NUMBER	A character string containing digits is convert to number
TO_DATE	A character string containing a date is converted to a date according to the format specified - default is DD-MMM-YY

The following example automatically converts a VARCHAR2 field to a number which is used in a calculation.

SQL CODE:

```
SQL> SELECT idno, zip, idno + zip
  2  FROM donor;

IDNO  ZIP    IDNO+ZIP
----- ----- ---------
11111 02345     13456
12121 02045     14166
22222 02045     24267
23456 02720     26176
33333 02720     36053
34567 02720     37287
```

Type here to search

9:46 AM
9/12/2019

Title: Sep 12-9:46 AM (4 of 26)

Conversion function	Processing
TO_CHAR	A number or date can be converted to VARCHAR2 using the format specified. When converting to date, there is a fm toggle element within the fmt format that can be used to remove leading 0s or padding spaces. TO_CHAR(data to convert [,'fmt']) TO_CHAR(number to convert [,'fmt']
TO_NUMBER	A character string containing digits is convert to number
TO_DATE	A character string containing a date is converted to a date according to the format specified - default is DD-MMM-YY

The following example automatically converts a VARCHAR2 field to a number which is used in a calculation.

SQL CODE:

```
SQL> SELECT idno, zip, idno + zip
  2  FROM donor;

IDNO  ZIP    IDNO+ZIP
----- ----- ---------
11111 02345     13456
12121 02045     14166
22222 02045     24267
23456 02720     26176
33333 02720     36053
34567 02720     37287
```

When working with dates, there are a wide variety of formats that can be used. These formats also include time. The following list covers some of the formatting possibilities - check Oracle or a text for a full list.

Date Format Code	Processing
Y or YY or YYY	Last 1, 2 or 3 digits of the year
YYYY	Year
Q	Quarter (returns 1, 2, 3, 4 for the four quarters)
MM	Month (returns 1 - 12)
Month	Name of month
W	Week of the month
DDD	Day of the year
DD	Day of the month
D	Day of the week (1-7)
DY	Three letter abbreviation of day of week
DAY	Day of the week
HH or HH12	Hour using 12 hour clock
HH24	Hour using 24 hour clock
MI	Minutes
SS	Seconds

Examples in SQL CODE:

```
SQL> SELECT TO_CHAR(datefst,'YYY')
  2  FROM donor;
```

```
992
998

6 rows selected.

SQL> SELECT datefst, TO_CHAR(datefst,'Q')
  2  FROM donor;

DATEFST   TO_CHAR(DATEFST,'Q')
--------- -------------------------------------
03-JUL-98 3
24-MAY-97 2
03-JAN-98 1
04-MAR-92 1
04-MAR-92 1
04-APR-98 2

SQL> SELECT datefst, TO_CHAR(datefst,'DDD'), TO_CHAR(datefst,'DAY')
  2  FROM donor;

DATEFST   TO_CHAR(DATEFST,'DDD')
--------- ---------------------------------------------------------
TO_CHAR(DATEFST,'DAY')
--------------------------------------------------------------------
03-JUL-98 184
FRIDAY

24-MAY-97 144
SATURDAY

03-JAN-98 003
SATURDAY

04-MAR-92 064
WEDNESDAY

04-MAR-92 064
WEDNESDAY

04-APR-98 094
SATURDAY

SQL> SELECT TO_CHAR(datefst, 'fmDD Month YYYY'), datefst
  2  FROM donor;

TO_CHAR(DATEFST,'FMDDMONTHYYYY')                                            DATEFST
--------------------------------------------------------------------------- ---------
3 July 1998                                                                 03-JUL-98
24 May 1997                                                                 24-MAY-97
3 January 1998                                                              03-JAN-98
4 March 1992                                                                04-MAR-92
4 March 1992                                                                04-MAR-92
4 April 1998                                                                04-APR-98

6 rows selected.
```

There are a wide variety of numeric formatting possibilities that can be used to display numeric numbers. They are used with the TO-CHAR conversion to a VARCHAR2 field. These formatting options allow the insertion of editing characters such as commas, dollar signs etc. Some of the formats are shown below. For a complete reference see Oracle help or a text book. The format is TO_CHAR(number to be formatted, 'fmt').

File Edit View History Bookmarks Tools Help

SQL Functions

www.pgrocer.net/Cis50/oracle3.html

```
TO_CHAR(DATEFST,'FMDDMONTHYYYY')                                                DATEFST
------------------------------------------------------------------------------ ---------
3 July 1998                                                                     03-JUL-98
24 May 1997                                                                     24-MAY-97
3 January 1998                                                                  03-JAN-98
4 March 1992                                                                    04-MAR-92
4 March 1992                                                                    04-MAR-92
4 April 1998                                                                    04-APR-98

6 rows selected.
```

There are a wide variety of numeric formatting possibilities that can be used to display numeric numbers. They are used with the TO-CHAR conversion to a VARCHAR2 field. These formatting options allow the insertion of editing characters such as commas, dollar signs etc. Some of the formats are shown below. For a complete reference see Oracle help or a text book. The format is TO_CHAR(number to be formatted, 'fmt').

Numeric Format	Processing
9	Numeric postions (determines width of display)
0	Used to display leading zeros (example: 09999)
$	Floating dollar sign
.	Decimal point (insert in display where needed)
,	Comma (insert in display where needed)
MI	Minus sign for negative numbers (displays on right)
PR	Negative numbers in parenthesis
B	Display zeros as blank

Examples of numeric formatting in SQL CODE:

```
1  SELECT TO_CHAR(yrgoal,'$9,999.99'), TO_CHAR(yrgoal*-1,'9999PR')
2* FROM donor
SQL> /

TO_CHAR(YR TO_CHA
---------- ------
   $500.00  <500>
   $400.00  <400>

   $100.00  <100>
    $50.00   <50>
    $50.00   <50>

1  SELECT TO_CHAR(yrgoal*100,'$99,999.99'), TO_CHAR(yrgoal*-1,'9999MI')
2* FROM donor
SQL> /

TO_CHAR(YRG TO_CH
----------- -----
 $50,000.00  500-
 $40,000.00  400-

 $10,000.00  100-
  $5,000.00   50-
  $5,000.00   50-
```

Type here to search

9:50 AM
9/12/2019

```
1  SELECT TO_CHAR(yrgoal*100,'$99,999.99'), TO_CHAR(yrgoal*-1,'9999MI')
2* FROM donor
SQL> /

TO_CHAR(YRG TO_CH
----------- -----
 $50,000.00  500-
 $40,000.00  400-

 $10,000.00  100-
  $5,000.00   50-
  $5,000.00   50-

SQL> SELECT TO_CHAR(yrgoal,'0999'), TO_CHAR(yrgoal,'B999.99')
  2  FROM donor;

TO_CH TO_CHAR
----- --------
 0500  500.00
 0400  400.00

 0100  100.00
 0050   50.00
 0050   50.00

6 rows selected.
```

The NVL function converts null values to actual values. The format is:
NVL(the source that may contain null, the value to be used)

SQL CODE:

```
SQL> SELECT yrgoal, NVL(yrgoal,0)
  2  FROM donor;

   YRGOAL NVL(YRGOAL,0)
--------- -------------
      500           500
      400           400
                      0
      100           100
       50            50
50 50

SQL> SELECT yrgoal, NVL(yrgoal, 10)
  2  FROM donor;

   YRGOAL NVL(YRGOAL,10)
--------- --------------
      500            500
      400            400
                      10
      100            100
       50             50
       50             50
```

SQL use of functions

Functions are used in SQL as they are in many languages to manipulate, modify, format, convert and calculate with data. Officially, Oracle functions are accepting arguments and returning values. Functions can be the kind that operates on each row individually, effecting the display of data for that row or functions that operate on groups. Functions that work with characters, either convert or manipulating are called character functions. This handout will deal with some of the character, numeric, and date functions available in Oracle.

Character Functions:

Character Functions	Processing
INITCAP	Converts the first alphabetic character of each word to uppercase
UPPER	Converts alphabetic characters to upper case
LOWER	Converts alphabetic characters to lower case
LTRIM	Trims blank characters to the left
RTRIM	Trims blank characters to the right
LPAD	Pads the right with a string until n characters are reached LPAD(data/field, n, string)
RPAD	Pads the right with a string until n characters are reached RPAD(data/field, n, string)
CONCAT	Concatenates two fields - similiar to || CONCAT(data/field, data/field)
SUBSTR	Returns a substring of the original starting at character m, for n characters (if no n, all characters till the end are returned) SUBSTR(data/field, m [,n])
INSTR	Finds a particular characters, m and returns it's position INSTR(data/field, m)
LENGTH	Returns the number of characters in a the data or field
REPLACE	Replace all occurrences of str1 in the data/field with str2 REPLACE(data/field, str1, str2)
SOUNDEX	Used to look for a column that sounds like the data/field entered

start

Substr(flda, formula, formula
function function

instr

In the first examples I am looking at INITCAP, UPPER and LOWER. In these examples, the word dog is capitalized, the word dog is converted to upper case and the word DOG is converted to lower case. In the example involving the donor table, name is converted to upper case, stadr is converted to lower case, and state is converted to an initial capital. Notice the word in the FROM clause of these selects. DUAL is a system file that contains a 1 character dummy field. It can be used when you want to do a select and see the results but you do not want to do it on a specific table. To see DUAL, you can use the DESCRIBE or DESC command to show it. Note: Dual can also be refered to as SYS.DUAL.

SQL CODE:

```
SQL> desc dual;

 Name                            Null?    Type
 ------------------------------- -------- ----
 DUMMY                                    VARCHAR2(1)
```

Now back to the SQL code for INITCAP, UPPER and LOWER.

File Edit View History Bookmarks Tools Help

Functions cont.

www.pgrocer.net/Cis50/oracle3a.html

SQL Functions: CASE/IF...THEN...ELSE Structure and Nesting

DECODE function:

The DECODE function allows for the implementation of the IF...THEN...ELSE or the CASE structure within SQL. The format is:

```
DECODE(column or expression, search value, resulting action [, search2 value, resulting action2,...][,default]
```

The following code checks or DECODES the city. If the city is Seekonk, then 100 is added to the yrgoal. If the city is Providence then 125 is added to the yrgoal. If the city is Fall River then 150 is added to the yrgoal The default, if there is a different city, is not to add anything to yrgoal. The result of the calculation is shown in the column that is labeled NEW_GOAL.

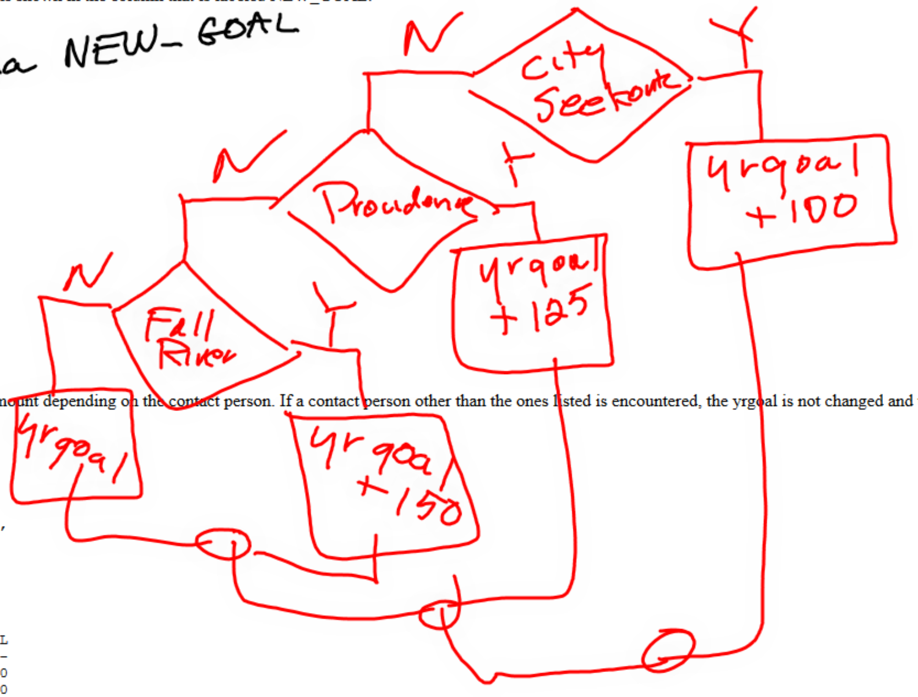
SQL CODE:

formula NEW_GOAL

```
SQL> SELECT name, city, yrgoal,
  2  DECODE(city, 'Seekonk', yrgoal+100,
  3               'Providence', yrgoal+125,
  4               'Fall River', yrgoal+150,
  5                             yrgoal)
  6          NEW_GOAL
  7  FROM donor;

NAME            CITY          YRGOAL  NEW_GOAL
--------------- ---------- --------- ---------
Stephen Daniels Seekonk          500       600
Jennifer Ames   Providence       400       525
Carl Hersey     Providence
Susan Ash       Fall River       100       250
Nancy Taylor    Fall River        50       200
Robert Brooks   Fall River        50       200

6 rows selected.
```



In the code below, the yrgoal is multiplied by a different amount depending on the contact person. If a contact person other than the ones listed is encountered, the yrgoal is not changed and the current year goal is the NEW_GOAL.

SQL CODE:

```
SQL> SELECT name, yrgoal, contact,
  2  DECODE(contact, 'John Smith', yrgoal*1.1,
  3                  'Susan Jones', yrgoal*1.15,
  4                  'Amy Costa', yrgoal*1.2,
  5                  'John Adams', yrgoal*1.25,
  6                                yrgoal)
  7         NEW_GOAL
  8  FROM donor;

NAME               YRGOAL CONTACT       NEW_GOAL
--------------- --------- ------------ ---------
Stephen Daniels       500 John Smith         550
Jennifer Ames         400 Susan Jones        460
Carl Hersey               Susan Jones
Susan Ash             100 Amy Costa          120
Nancy Taylor           50 John Adams        62.5
Robert Brooks          50 Amy Costa           60
```

Type here to search

10:05 AM 9/12/2019

File Edit View History Bookmarks Tools Help

Functions cont.

www.pgrocer.net/Cis50/oracle3a.html

```
--------------- --------- ------------ ---------
Stephen Daniels       500 John Smith         550
Jennifer Ames         400 Susan Jones        460
Carl Hersey               Susan Jones
Susan Ash             100 Amy Costa          120
Nancy Taylor           50 John Adams        62.5
Robert Brooks          50 Amy Costa           60

6 rows selected.
```

Nesting FUNCTIONS:

Functions can be nested one within another. The two examples I have chosen accomplish nothing worth while but illustrate nested functions. With nested functions, the inner function is evaluated first and then evaluation moves out with the outer function being evaluated last.

For example, a variety of date functions are used in this SELECT. The first thing that is done is to add 6 months to datefst using the ADD_MONTHS. Then the MONTHS_BETWEEN evaluates the number between the new datefst which had the 6 months added and the datefst. Finally the answer is converted to character using TO_CHAR.

SQL CODE:

```
1  SELECT TO_CHAR(MONTHS_BETWEEN(ADD_MONTHS(datefst,6),datefst)), datefst
  2* FROM donor
SQL> /

TO_CHAR(MONTHS_BETWEEN(ADD_MONTHS(DATEFS DATEFST
---------------------------------------- ---------
6                                        03-JUL-98
6                                        24-MAY-97
6                                        03-JAN-98
6                                        04-MAR-92
6                                        04-MAR-92
6                                        04-APR-98

6 rows selected.
```

In the example below, character functions are used. The first thing is to determine the length of state, the inner function. Next that length is used to take the substring starting with the 2nd (the length of state) and taking 4 characters. Finally the 4 characters are concatenated with the literal CITY=.

SQL CODE:

```
1  SELECT city, state, CONCAT('CITY=',SUBSTR(city,LENGTH(state),4))
  2* FROM DONOR
SQL> /
```

3rd 2nd 1st

```
CITY       ST CONCAT('C
---------- -- ---------
Seekonk    MA CITY=eeko
Providence RI CITY=rovi
Providence RI CITY=rovi
Fall River MA CITY=all
Fall River MA CITY=all
Fall River MA CITY=all

6 rows selected.
```

Type here to search

10:11 AM
9/12/2019

SQL CODE:

```
SQL> SELECT SUBSTR(datefst,4,3), datefst
  2  FROM donor;

SUB DATEFST
--- ---------
JUL 03-JUL-98
MAY 24-MAY-97
JAN 03-JAN-98
MAR 04-MAR-92
MAR 04-MAR-92
APR 04-APR-98

6 rows selected.
```

Two types of functions: 1) acts on individual records 2) acts on groups of records

The INSTR function allows me to locate a particular character in a data string or data in a column. In the example shown, I want to find the lower case letter e in the city column. The format is INSTR(the data or column being used, the character you are looking for).

```
SQL> SELECT city, INSTR(city,'e')
  2  FROM donor;

CITY       INSTR(CITY,'E')
---------- ---------------
Seekonk                  2
Providence               7
Providence               7
Fall River               9
Fall River               9
Fall River               9

6 rows selected.
```

Using LENGTH, I can find the number of characters in a string or in a column. In this example, I am looking for the number of characters in a name. To do that simply put the data string or name of the column in parenthesis after the function LENGTH.

SQL CODE:

```
SQL> SELECT name, LENGTH(name)
  2  FROM donor;

NAME            LENGTH(NAME)
--------------- ------------
Stephen Daniels           15
Jennifer Ames             13
Carl Hersey               11
Susan Ash                  9
Nancy Taylor              12
Robert Brooks             13

6 rows selected.
```

The REPLACE allows the user to replace a particular string of data in a column with another string of data. For example, if you changed a code and you want to replace all AB embedded in the inventory code to XY. This could be done with

File Edit View History Bookmarks Tools Help

Group functions

www.pgrocer.net/Cis50/oracle3b.html

Group Functions using SQL

Group functions produce results based on a group or set of rows. For example, with group functions you can get counts, sums/totals, averages, maximums, minimums, etc. If you include the distinct clause then you will evaluate only non-duplicate values while if you include ALL (the default) you will evaluate all values. The data types may be VARCHAR2, CHAR, NUMBER or DATE for the functions COUNT, MAX and MIN. Null values will be ignored by all group functions except COUNT(*). The NVL function can be used to overcome this.

FUNCTION list:

- COUNT - counts the group
- SUM - totals the group
- AVG - averages the group
- MAX - finds maximum in the group
- MIN - find minimum in the group
- STDDEV - finds the standard deviation
- VARIANCE - finds the variance

The only group function that deals with null values is COUNT(*). Other functions can be forced to deal with null values by using the NVL function.

SQL CODE:

```
SQL> SELECT COUNT(*) FROM donor;

 COUNT(*)
---------
        6
SQL> SELECT COUNT(NVL(yrgoal,0)), AVG(NVL(yrgoal,0)), SUM(NVL(yrgoal,0))
  2  FROM donor;

COUNT(NVL(YRGOAL,0)) AVG(NVL(YRGOAL,0)) SUM(NVL(YRGOAL,0))
-------------------- ------------------ ------------------
                   6          183.33333               1100

SQL>
```

The rest of these examples will ignore null values if the column being tested contains a null value.

SQL CODE:

```
SQL> SELECT COUNT(yrgoal), AVG(yrgoal), SUM(yrgoal)
  2  FROM donor;

COUNT(YRGOAL) AVG(YRGOAL) SUM(YRGOAL)
------------- ----------- -----------
            5         220        1100
SQL> SELECT MIN(yrgoal), MAX(yrgoal)
  2  FROM donor;

MIN(YRGOAL) MAX(YRGOAL)
----------- -----------
         50         500

SQL> SELECT COUNT(datefst), MAX(datefst), MIN(datefst)
```

Type here to search

10:16 AM
9/12/2019

```
SQL> SELECT MIN(yrgoal), MAX(yrgoal)
  2  FROM donor;

MIN(YRGOAL) MAX(YRGOAL)
----------- -----------
         50         500

SQL> SELECT COUNT(datefst), MAX(datefst), MIN(datefst)
  2  FROM donor;

COUNT(DATEFST) MAX(DATEF MIN(DATEF
-------------- --------- ---------
             6 03-JUL-98 04-MAR-92

SQL> SELECT COUNT(DISTINCT(yrgoal)) FROM donor;

COUNT(DISTINCT(YRGOAL))
-----------------------
                      4

SQL> SELECT SUM(DISTINCT(yrgoal))
  2  FROM donor;

SUM(DISTINCT(YRGOAL))
---------------------
                 1050

SQL> SELECT STDDEV(yrgoal), VARIANCE(yrgoal)
  2  FROM donor;

STDDEV(YRGOAL) VARIANCE(YRGOAL)
-------------- ----------------
      213.8925            45750
```

Instead of treating the whole table as the group, the GROUP BY clause can be used to make groups based on particular criteria. In the first example below, I am printing the state and a count of the number of cities in each state. The select line has state and count(city) and the GROUP BY clause has state (note that state did not have to be included in the select by, only in the group by). SQL will produce one display line for each state or group. Including the GROUP BY function means that you will only get grouped results, you will not see individual groups. A WHERE clause can be used to exclude things from the GROUPS as shown in the second example below. One other important thing to note, you must use the column not the column alias in the group by clause. One other note, the group by clause causes sorting to be in ascending order by the column in the group by clause, to override this, you can use the order by clause. This is shown in the third example which is ordered by state desc.

SQL CODE:

```
SQL> SELECT state, COUNT(city)
  2  FROM donor
  3  GROUP BY state;

ST COUNT(CITY)
-- -----------
MA           4
RI           2

SQL>

SQL> SELECT state, COUNT(city)
  2  FROM donor
  3  WHERE city !='Seekonk'
  4  GROUP BY state;

ST COUNT(CITY)
-- -----------
```

File Edit View History Bookmarks Tools Help

Group functions

www.pgrocer.net/Cis50/oracle3b.html

```
STDDEV(YRGOAL) VARIANCE(YRGOAL)
-------------- ----------------
      213.8925            45750
```

Instead of treating the whole table as the group, the GROUP BY clause can be used to make groups based on particular criteria. In the first example below, I am printing the state and a count of the number of cities in each state. The select line has state and count(city) and the GROUP BY clause has state (note that state did not have to be included in the select by, only in the group by). SQL will produce one display line for each state or group. Including the GROUP BY function means that you will only get grouped results, you will not see individual groups. A WHERE clause can be used to exclude things from the GROUPS as shown in the second example below. One other important thing to note, you must use the column not the column alias in the group by clause. One other note, the group by clause causes sorting to be in ascending order by the column in the group by clause, to override this, you can use the order by clause. This is shown in the third example which is ordered by state desc.

SQL CODE:

```
SQL> SELECT state, COUNT(city)
  2  FROM donor
  3  GROUP BY state;

ST COUNT(CITY)
-- -----------
MA           4
RI           2

SQL>

SQL> SELECT state, COUNT(city)
  2  FROM donor
  3  WHERE city !='Seekonk'
  4  GROUP BY state;

ST COUNT(CITY)
-- -----------
MA           3
RI           2

SQL> SELECT state, COUNT(city)
  2  FROM donor
  3  GROUP BY state
  4  ORDER BY state desc;

ST COUNT(CITY)
-- -----------
RI           2
MA           4

SQL>
```

You can also use a function in the order by, let's say I wanted to order by the count of the cities in descending order as opposed to the state name.

SQL CODE:

```
SQL> SELECT state, COUNT(city)
  2  FROM donor
  3  GROUP by state
  4  ORDER BY COUNT(city) desc;

ST COUNT(CITY)
-- -----------
MA           4
```

Type here to search

10:19 AM
9/12/2019

```
STDDEV(YRGOAL)  VARIANCE(YRGOAL)
--------------  ----------------
      213.8925             45750
```

Instead of treating the whole table as the group, the GROUP BY clause can be used to make groups based on particular criteria. In the first example below, I am printing the state and a count of the number of cities in each state. The select line has state and count(city) and the GROUP BY clause has state (note that state did not have to be included in the select by, only in the group by). SQL will produce one display line for each state or group. Including the GROUP BY function means that you will only get grouped results, you will not see individual groups. A WHERE clause can be used to exclude things from the GROUPS as shown in the second example below. One other important thing to note, you must use the column not the column alias in the group by clause. One other note, the group by clause causes sorting to be in ascending order by the column in the group by clause, to override this, you can use the order by clause. This is shown in the third example which is ordered by state desc.

SQL CODE:

```
SQL> SELECT state, COUNT(city)
  2  FROM donor
  3  GROUP BY state;

ST COUNT(CITY)
-- -----------
MA           4
RI           2

SQL>

SQL> SELECT state, COUNT(city)
  2  FROM donor
  3  WHERE city !='Seekonk'
  4  GROUP BY state;

ST COUNT(CITY)
-- -----------
MA           3
RI           2

SQL> SELECT state, COUNT(city)
  2  FROM donor
  3  GROUP BY state
  4  ORDER BY state desc;

ST COUNT(CITY)
-- -----------
RI           2
MA           4

SQL>
```

As we are forming groups eliminate Seekonk

You can also use a function in the order by, let's say I wanted to order by the count of the cities in descending order as opposed to the state name.

SQL CODE:

```
SQL> SELECT state, COUNT(city)
  2  FROM donor
  3  GROUP by state
  4  ORDER BY COUNT(city) desc;

ST COUNT(CITY)
-- -----------
MA           4
```

Title: Sep 12-10:20 AM (16 of 26)

```
ST COUNT(CITY)
-- -----------
MA           4
RI           2

SQL>

SQL> SELECT state, COUNT(city)
  2  FROM donor
  3  WHERE city !='Seekonk'
  4  GROUP BY state;

ST COUNT(CITY)
-- -----------
MA           3
RI           2

SQL> SELECT state, COUNT(city)
  2  FROM donor
  3  GROUP BY state
  4  ORDER BY state desc;

ST COUNT(CITY)
-- -----------
RI           2
MA           4

SQL>
```

You can also use a function in the order by, let's say I wanted to order by the count of the cities in descending order as opposed to the state name.

SQL CODE:

```
SQL> SELECT state, COUNT(city)
  2  FROM donor
  3  GROUP by state
  4  ORDER BY COUNT(city) desc;

ST COUNT(CITY)
-- -----------
MA           4
RI           2
```

You can also group within a group. In this example, I grouped by state and then by contact within state. This means that first the rows will be grouped by state and then on contact within state. The SUM of yrgoals is for each contact within each state. Note again that with the GROUP BY I can only display grouped information so the only things in the select can be the columns I group by and group functions. Again, I could have eliminated a particular contact, as shown in the second example below, by the use of the WHERE clause. In the example, I eliminated contact John Adams from the results.

SQL CODE:

```
SQL> SELECT state, contact, SUM(yrgoal)
  2  FROM donor
  3  GROUP BY state, contact;

ST CONTACT      SUM(YRGOAL)
-- ------------ -----------
MA Amy Costa            150
```

Title: Sep 12-10:24 AM (17 of 26)

```
ST COUNT(CITY)
-- -----------
RI           2
MA           4

SQL>
```

You can also use a function in the order by, let's say I wanted to order by the count of the cities in descending order as opposed to the state name.

SQL CODE:

```
SQL> SELECT state, COUNT(city)
  2  FROM donor
  3  GROUP by state
  4  ORDER BY COUNT(city) desc;

ST COUNT(CITY)
-- -----------
MA           4
RI           2
```

You can also group within a group. In this example, I grouped by state and then by contact within state. This means that first the rows will be grouped by state and then on contact within state. The SUM of yrgoals is for each contact within each state. Note again that with the GROUP BY I can only display grouped information so the only things in the select can be the columns I group by and group functions. Again, I could have eliminated a particular contact, as shown in the second example below, by the use of the WHERE clause. In the example, I eliminated contact John Adams from the results.

SQL CODE:

```
SQL> SELECT state, contact, SUM(yrgoal)
  2  FROM donor
  3  GROUP BY state, contact;

ST CONTACT      SUM(YRGOAL)
-- ------------ -----------
MA Amy Costa            150
MA John Adams            50
MA John Smith           500
RI Susan Jones          400

SQL> SELECT state, contact, SUM(yrgoal)
  2  FROM donor
  3  WHERE contact  !='John Adams'
  4  GROUP BY state, contact;

ST CONTACT      SUM(YRGOAL)
-- ------------ -----------
MA Amy Costa            150
MA John Smith           500
RI Susan Jones          400
```

A tricky part of grouping is when you can use the WHERE clause and when you must use the HAVING clause. My understanding is that if you restrict prior to forming the groups the WHERE clause can be used. In the example above I eliminated contact John Adams. However if you restrict the actual groups such as in the example below where I was testing to see if SUM(yrgoal) was > 100 then you need to use the HAVING clause. The third example below, brings the WHERE and the HAVING together with the WHERE eliminating JOHN ADAMS and the HAVING eliminating groups that do not have a SUM(yrgoal) > 100.

SQL CODE:

File Edit View History Bookmarks Tools Help

Group functions

www.pgrocer.net/Cis50/oracle3b.html

```
ST COUNT(CITY)
-- -----------
MA           4
RI           2
```

You can also group within a group. In this example, I grouped by state and then by contact within state. This means that first the rows will be grouped by state and then on contact within state. The SUM of yrgoals is for each contact within each state. Note again that with the GROUP BY I can only display grouped information so the only things in the select can be the columns I group by and group functions. Again, I could have eliminated a particular contact, as shown in the second example below, by the use of the WHERE clause. In the example, I eliminated contact John Adams from the results.

SQL CODE:

```
SQL> SELECT state, contact, SUM(yrgoal)
  2  FROM donor
  3  GROUP BY state, contact;

ST CONTACT      SUM(YRGOAL)
-- ------------ -----------
MA Amy Costa            150
MA John Adams            50
MA John Smith           500
RI Susan Jones          400

SQL> SELECT state, contact, SUM(yrgoal)
  2  FROM donor
  3  WHERE contact  !='John Adams'
  4  GROUP BY state, contact;

ST CONTACT      SUM(YRGOAL)
-- ------------ -----------
MA Amy Costa            150
MA John Smith           500
RI Susan Jones          400
```

A tricky part of grouping is when you can use the WHERE clause and when you must use the HAVING clause. My understanding is that if you restrict prior to forming the groups the WHERE clause can be used. In the example above I eliminated contact John Adams. However if you restrict the actual groups such as in the example below where I was testing to see if SUM(yrgoal) was > 100 then you need to use the HAVING clause. The third example below, brings the WHERE and the HAVING together with the WHERE eliminating JOHN ADAMS and the HAVING eliminating groups that do not have a SUM(yrgoal) > 100.

SQL CODE:

```
SQL> SELECT state, contact, SUM(yrgoal)
  2  FROM donor
  3  WHERE SUM(yrgoal) > 100
  4  GROUP by state, contact;
WHERE SUM(yrgoal) > 100
      *
ERROR at line 3:
ORA-00934: group function is not allowed here

SQL> SELECT state, contact, SUM(yrgoal)
  2  FROM donor
  3  GROUP BY state, contact
  4  HAVING SUM(yrgoal) > 100;

ST CONTACT      SUM(YRGOAL)
-- ------------ -----------
MA Amy Costa            150
```

Type here to search

10:33 AM 9/12/2019

File Edit View History Bookmarks Tools Help

Group functions

www.pgrocer.net/Cis50/oracle3b.html

```
  4  GROUP BY state, contact;

ST CONTACT       SUM(YRGOAL)
-- ------------ -----------
MA Amy Costa             150
MA John Smith            500
RI Susan Jones           400
```

A tricky part of grouping is when you can use the WHERE clause and when you must use the HAVING clause. My understanding is that if you restrict prior to forming the groups the WHERE clause can be used. In the example above I eliminated contact John Adams. However if you restrict the actual groups such as in the example below where I was testing to see if SUM(yrgoal) was > 100 then you need to use the HAVING clause. The third example below, brings the WHERE and the HAVING together with the WHERE eliminating JOHN ADAMS and the HAVING eliminating groups that do not have a SUM(yrgoal) > 100.

SQL CODE:

```
SQL> SELECT state, contact, SUM(yrgoal)
  2  FROM donor
  3  WHERE SUM(yrgoal) > 100
  4  GROUP by state, contact;
WHERE SUM(yrgoal) > 100
      *
ERROR at line 3:
ORA-00934: group function is not allowed here

SQL> SELECT state, contact, SUM(yrgoal)
  2  FROM donor
  3  GROUP BY state, contact
  4  HAVING SUM(yrgoal) > 100;

ST CONTACT       SUM(YRGOAL)
-- ------------ -----------
MA Amy Costa             150
MA John Smith            500
RI Susan Jones           400

SQL> SELECT state, contact, SUM(yrgoal)
  2  FROM donor
  3  WHERE contact != 'John Adams'
  4  GROUP BY state, contact
  5  HAVING SUM(yrgoal) > 100
  6  ORDER by SUM(yrgoal);

ST CONTACT       SUM(YRGOAL)
-- ------------ -----------
MA Amy Costa             150
RI Susan Jones           400
MA John Smith            500

SQL>
```

The two examples below show the minimum or the average yrgoals by state (example 1) and by contact within state (example 2).

SQL CODE:

```
SQL> SELECT MIN(AVG(yrgoal))
  2  FROM donor
  3  GROUP BY state;
```

Type here to search

10:33 AM 9/12/2019

Title: Sep 12-10:33 AM (20 of 26)

```
SQL> SELECT state, contact, SUM(yrgoal)
  2  FROM donor
  3  WHERE SUM(yrgoal) > 100
  4  GROUP by state, contact;
WHERE SUM(yrgoal) > 100
      *
ERROR at line 3:
ORA-00934: group function is not allowed here

SQL> SELECT state, contact, SUM(yrgoal)
  2  FROM donor
  3  GROUP BY state, contact
  4  HAVING SUM(yrgoal) > 100;

ST CONTACT      SUM(YRGOAL)
-- ------------ -----------
MA Amy Costa            150
MA John Smith           500
RI Susan Jones          400

SQL> SELECT state, contact, SUM(yrgoal)
  2  FROM donor
  3  WHERE contact != 'John Adams'
  4  GROUP BY state, contact
  5  HAVING SUM(yrgoal) > 100
  6  ORDER by SUM(yrgoal);

ST CONTACT      SUM(YRGOAL)
-- ------------ -----------
MA Amy Costa            150
RI Susan Jones          400
MA John Smith           500

SQL>
```

The two examples below show the minimum or the average yrgoals by state (example 1) and by contact within state (example 2).

SQL CODE:

```
SQL> SELECT MIN(AVG(yrgoal))
  2  FROM donor
  3  GROUP BY state;

MIN(AVG(YRGOAL))
----------------
             175

SQL> SELECT MIN(AVG(yrgoal))
  2  FROM donor
  3  GROUP BY state, contact;

MIN(AVG(YRGOAL))
----------------
              50

SQL>
```

File Edit View History Bookmarks Tools Help

CIS150/50 Course Page

www.pgrocer.net/PFGCIS150.html

Oracle and SQL
CIS150/50

About This Course:

Need help or support?
Introduction to CIS150
Syllabus
Withdrawal Policy as of Fall 2017

Site Resources:

Notes and Handouts
Presentations
SmartBoard Presentations
Audio

Weekly Schedule:

Weekly schedule - CIS150/50

Using Oracle:

Installation and login

Links:

Links & Tutorials

	output that is produced unless you are told otherwise. Sometimes this involves showing data that was written to a file/table. Please keep copies of all work you submit until you receive your final grade at the end of the semester. Comment on how I want the assignment: You should take the commands you do in SQL and the results and paste them into a word document. It should be similiar to the way I prepared the notes. You then send me the word document. I do not want you to show the screen itself - they are really hard to read!
HELP schedule	We are starting the semester with this help schedule. We may make changes based on student need, so check back! These are the times that Joe Fletcher, my SI, is available. Monday 2-7 in K101 Tuesday 2-5 in K104 Wednesday 2-6 in K101 Thursday 2-5 in K101 Jayson Rios from the tutoring center is also available on Monday 11-3 and Wednesday 11-1 in New Bedford.
Week #2 Week of September 9th	**Please do not change any of the passwords we have given you. I need to be able to get to your site as does the IT staff if there is a problem.** See the help times above. These are the notes that I will attempt to get through this week. **More on maintaining an SQL table** **SQL functions** **More SQL functions** **Group SQL functions** I may go further, so do check back. You should also look at the accompanying PowerPoint presentations: **More on maintaining an SQL table** **Separate speaker notes for More on maintaining an SQL table** **SQL functions - part 1 character** **Separate speaker notes for SQL functions - part 1 characters** **SQL functions - part 2 numeric & date** **Separate speaker notes for SQL functions - part 2 numeric & date** **More SQL functions** (includes DECODE, ways to edit SQL code and nested functions) **Separate speaker notes for more SQL functions** **Group functions** **Separate speaker notes for group functions** Always check back and see how we are doing! ***Assignments:*** Here is a script to create the emp table: **Script to create emp table (createemp.sql)** You can enter edit createemp.sql or another name at the SQL prompt. It will go into the editor and ask if you want to create a new script. Enter yes and paste the script in and save it. Return to the prompt and enter @ createemp.sql and it should run and create the table for you. If you are using putty, remember to have the script text in notepad and highlight it. When you go into the editor the right click will paste it or you can uses shift + ins to paste. You can then write and exit. Back at the SQL prompt is where you run it with the @. **Inclass if statements (must be passed in by everyone)** I want you to play computer and show me the output you think would be generated as opposed to running the query and letting Oracle give you the answer. **Second SQL assignment** **Quiz 1** I am hoping your have already created a table that you can use for this quiz. If not you need to create one. I would suggest you create one that you can use with assignments from week 1 or week 2. Note that you need to show me the SQL and the results on all of the quiz questions. Check back!
	I recommend one of the text books in the syllabus, but a textbook is not required. There is enough material on the web and in my notes that I decided to make the book a recommendation. Please note that many students have found it useful to have a copy of the text. I do recommend that you get one of these books or another book dealing with Oracle SQL (we are using version 11g or 12c this semester) as a reference. We will not do chapter by chapter coverage, the text gives you additional information on topics and a reference if you have questions. ional database information

Type here to search

10:42 AM
9/12/2019

File Edit View History Bookmarks Tools Help

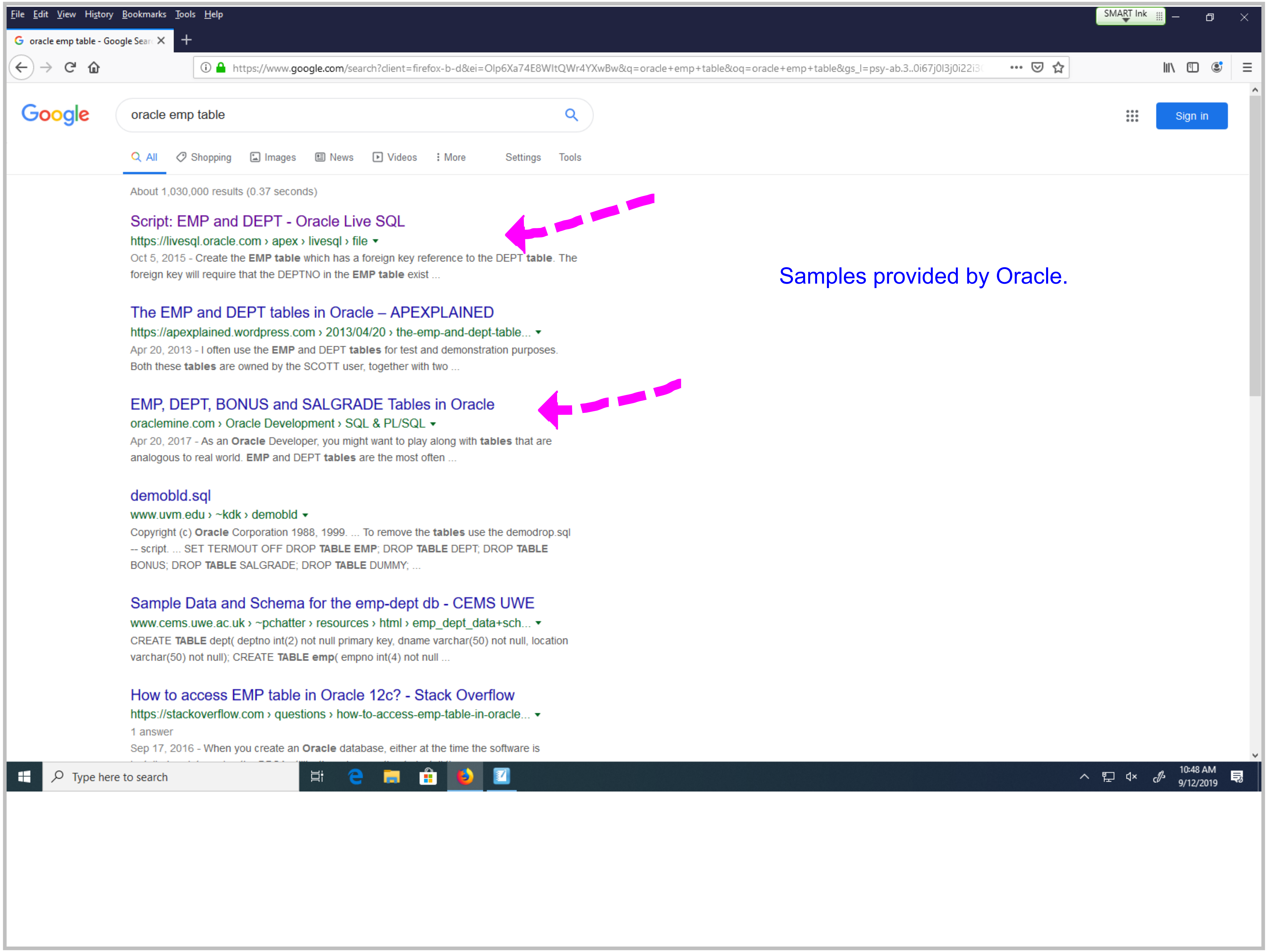
pgrocer.net/Cis50/createemp.sql

www.pgrocer.net/Cis50/createemp.sql

```
create table empcopy
(empno number(4), ename varchar2(10), job varchar2(9),
mgr number(4), hiredate date, sal number(7,2),
comm number(7,2), deptno number(2));
insert into empcopy values (7369, 'SMITH', 'CLERK', 7902, '17-DEC-80',800,NULL,20);
insert into empcopy values (7499, 'ALLEN', 'SALESMAN', 7698, '20-FEB-81',1600,300,30);
insert into empcopy values (7521, 'WARD', 'SALESMAN', 7698, '22-FEB-81', 1250, 500,30);
insert into empcopy values (7566, 'JONES', 'MANAGER', 7839, '02-APR-81', 2975,NULL,20);
insert into empcopy values (7654, 'MARTIN','SALESMAN', 7698,'28-SEP-81', 1250, 1400, 30);
insert into empcopy values (7698, 'BLAKE', 'MANAGER', 7839, '01-MAY-81', 2850,NULL, 30);
insert into empcopy values (7782, 'CLARK', 'MANAGER', 7839, '09-JUN-81', 2450, NULL, 10);
insert into empcopy values (7788, 'SCOTT', 'ANALYST', 7566, '19-APR-87', 3000, NULL, 20);
insert into empcopy values (7839, 'KING', 'PRESIDENT', NULL, '17-NOV-81', 5000, NULL, 10);
insert into empcopy values (7844,'TURNER','SALESMAN',7698, '08-SEP-81', 1500,0,30);
insert into empcopy values (7876, 'ADAMS','CLERK', 7788, '23-MAY-87',1100, NULL, 20);
insert into empcopy values (7900, 'JAMES', 'CLERK', 7698, '03-DEC-81', 950, NULL, 30);
insert into empcopy values (7902, 'FORD', 'ANALYST', 7566, '03-DEC-81', 3000,NULL,20);
insert into empcopy values (7934, 'MILLER', 'CLERK', 7782, '23-JAN-82', 1300, NULL, 10);
select * from empcopy;
```

Type here to search

10:42 AM
9/12/2019



Title: Sep 12-10:48 AM (24 of 26)

File Edit View History Bookmarks Tools Help

EMP, DEPT, BONUS and SALGR

oraclemine.com/emp-and-dept-tables-in-oracle/

```
INSERT INTO dept VALUES(10, 'ACCOUNTING', 'NEW YORK');
INSERT INTO dept VALUES(20, 'RESEARCH', 'DALLAS');
INSERT INTO dept VALUES(30, 'SALES', 'CHICAGO');
INSERT INTO dept VALUES(40, 'OPERATIONS', 'BOSTON');
```

EMP table values

```
INSERT INTO emp VALUES(
 7839, 'KING', 'PRESIDENT', null,
 to_date('17-11-1981','dd-mm-yyyy'),
 5000, null, 10 );

INSERT INTO emp VALUES(
 7698, 'BLAKE', 'MANAGER', 7839,
 to_date('1-5-1981','dd-mm-yyyy'),
 2850, null, 30);

INSERT INTO emp VALUES(
 7782, 'CLARK', 'MANAGER', 7839,
 to_date('9-6-1981','dd-mm-yyyy'),
 2450, null, 10);

INSERT INTO emp VALUES(
 7566, 'JONES', 'MANAGER', 7839,
 to_date('2-4-1981','dd-mm-yyyy'),
 2975, null, 20);

INSERT INTO emp VALUES(
 7788, 'SCOTT', 'ANALYST', 7566,
 to_date('13-JUL-87','dd-mm-rr') - 85,
 3000, null, 20);

INSERT INTO emp VALUES(
 7902, 'FORD', 'ANALYST', 7566,
 to_date('3-12-1981','dd-mm-yyyy'),
 3000, null, 20 );
```

Type here to search

10:58 AM
9/12/2019

OracleMine

Knowledge mine of Oracle Technologies

EMP and DEPT tables in Oracle

As an Oracle Developer, you might want to play along with tables that are analogous to real world. EMP and DEPT tables are the most often worked upon tables in Oracle. Mostly these tables are owned by user SCOTT. Along with EMP and DEPT, the other two commonly used tables are BONUS and SALGRADE.

Here I have given the table structures (DDL) and values (DML) of EMP, DEPT, BONUS and SALGRADE table.

Happy Programming!

Table Structures