

SINGLE ROW FUNCTIONS -- CHAPTER 3 --

- There are two main types of functions, single-row and multi-row functions.
- MULTIPLE ROW functions manipulate groups of rows to give one result per group of rows.
- SINGLE ROW functions operate on single rows only and return one result per row. There are different types of these functions and they include those that process Characters, Number, Data, and Conversion.
 - Can manipulate data
 - Accept arguments and return one value
 - Act on each row returned
 - Return one result per row
 - Can Modify the data-type (type casting).
 - Can be nested.
- The syntax for single-row functions is as follows:

```
Function_name (column | expression, [arg1, arg2,...])
```

- Where column is any database named column, expression is any character string or calculated expression and arg1, arg2, etc is any argument to be used by the function.

- **Character Functions:**

- (1) Case conversion functions and
 - LOWER – Converts alpha characters to lower case.
 - LOWER(column | expression)
 - UPPER – Converts alpha characters to upper case.
 - UPPER(column | expression)
 - INITCAP – Capitalizes the first alpha character of a single word.
 - INITCAP(column | expression)

```
SQL> run
1 select empno, initcap(ename), lower(ename), upper(ename)
2* from emp
```

EMPNO	INITCAP (EN	LOWER (ENAM	UPPER (ENAM
7369	Smith	smith	SMITH
7499	Allen	allen	ALLEN
7521	Ward	ward	WARD
7566	Jones	jones	JONES
321	Bobby	bobby	BOBBY

- (2) Character manipulation functions.
 - CONCAT – Concatenates the first character value to the second character value. Note that this is equivalent to using the concatenation operator (||).
 - CONCAT(column1 | expression1, column2 | expression2,...)

```
SQL> r
1 select empno, concat('Hello ', concat('Employee: ', ename))
2* from emp
```

```

EMPNO CONCAT('HELLO',CONCAT('EMP
-----
7369 Hello Employee: SMITH
7499 Hello Employee: ALLEN
7521 Hello Employee: WARD
7566 Hello Employee: JONES
```

- **SUBSTR** – Returns specified characters from character value starting at character position m, n characters long (if m is negative, the count starts from the end of the character value. If n is omitted, all characters to the end of the of the string are returned.

- SUBSTR(column | expression, m[n])
-

```
SQL> r
1 select ename, substr(ename, 0,2), substr(ename, 3)
2* from emp
```

```

ENAME      SU SUBSTR(E
-----
SMITH      SM ITH
ALLEN      AL LEN
WARD       WA RD
JONES      JO NES
```

- **LENGTH** – Returns the number of characters in value

- LENGTH(column | expression)

```
SQL> run
1 select ename, length(ename)
2* from emp
```

```

ENAME      LENGTH(ENAME)
-----
SMITH      5
ALLEN      5
WARD       4
```

- **INSTR** – Returns the numeric position of a named character

- INSTR(column | expression, m)

```

ENAME      INSTR(ENAME, 'M')
-----
SMITH      2
ALLEN      0
ADAMS      4
```

- Note that when the character does not exist, this function returns a zero. Note also that in the event that the character exists more than once only the position of the first one (left to right) is returned.
- LPAD – Pads the character value right-justified to a total width of n character positions. Note that there IS also an **RPAD** function that works similarly but pads from the right...
 - PAD(column | expression n, 'string')

```
SQL> run
1  select ename, lpad(ename, 20, '# ')
2* from emp
```

ENAME	LPAD (ENAME, 20, '# ')
SMITH	# # # # # # # # #SMITH
ALLEN	# # # # # # # # #ALLEN
WARD	# # # # # # # # # WARD
JONES	# # # # # # # # #JONES
BOBBY	# # # # # # # # #BOBBY

- TRIM – Enables to trim heading or trailing characters (or both) from a character string. If trim_character or trim_source is a character literal, you must enclose it in single quotes. This is a feature available from Oracle8i onward.
 - TRIM(leading | trailing | both, trim_character FROM trim_source)

```
SQL> r
1  select trim('R' from 'RRRKR MITH')
2  from emp
3* where empno > 9000
```

```
TRIM('
-----
KRMITH
KRMITH
KRMITH
```

- Note that this function is acting on a string literal in this case (but it could be a column containing a string) and removes all occurrences of 'R' that are contiguous. Note that the right-most 'R' in the string is protected by the 'K' that precedes it.

- **Number Functions:**

- ROUND – Rounds a value to a specified decimal
 - ROUND(45.926, 2) → 45.93
- TRUNC – Truncates a value to specified decimal
 - TRUNC(45.926, 2) → 45.92
- MOD – Returns remainder of division

- MOD(1600, 300) → 100

```
SQL> r
 1 select trunc(45.845, 1), round(45.845), round(45.845,2),
    mod(45, 6)
 2 from emp
 3* where empno > 9000
```

TRUNC(45.845,1)	ROUND(45.845)	ROUND(45.845,2)	MOD(45.5,2.3)
-----	-----	-----	-----
45.8	46	45.85	1.8

- Note that the MOD function is not restricted to integer values. Any floating point values are allowed as arguments.
- **Working With DATES:**
- Oracle stores date in an internal numeric format: century, year, month, day, hours, minutes, seconds. The default date format is DD-MON-YY (example: 22-02-75 for February 22, 1975?)
- The function **SYSDATE** returns the current time/date.
- **DUAL** is a dummy table used to view SYSDATE. (this will come in handy a lot!). It contains only one column and one row and is useful for returning single values of functions.

```
SQL> select SYSDATE
 2 from DUAL;
```

```
SYSDATE
-----
29-AUG-02
```

- Arithmetic with DATES:
 - OPERATION RESULT DESCRIPTION
 - Date + number Date Add a number of days to a date
 - Date - number Date Subtracts a number of days to a date
 - Date - Date # of Days Subtracts one date form another
 - Date + number/24 Date Adds a number of hours to a date

```
SQL> r
 1 select ename, round((SYSDATE - hiredate)/7) WEEKS
 2 from emp
 3* where deptno = 10
```

ENAME	WEEKS
-----	-----
CLARK	1107
KING	1084
BLACK	46

- Date Definitions: Date functions operate on Oracle date. All date functions return a value of DATE type with the exception of MONTHS_BETWEEN which returns a numeric value.
 - MONTHS_BETWEEN(date1, date2) : Finds the number of months between date1 and date2. Note that the result can be either negative or positive depending on the argument's values.
 - ADD_MONTHS(date, n) : Adds a number of calendar months to date. The number of n must be an integer value and as such, may be either negative or positive.
 - NEXT_DAY(date, 'char') : Finds the date of the next specified day of the week ('char') following date. The number of char may be a number representing a day or character string.
 - LAST_DAY(date): Finds the date of the last day of the month that contains date.

```
SQL> r
1  select months_between('22-feb-1975','28-aug-2002') *-1
2      "NumberOfMontsIHaveLivedSoFar",
3      round(months_between('22-feb-1975','28-aug-
4      2002'))/12*-1
5      "YearsIHaveLivedSoFar",
6      next_day(SYSDATE, 'FRIDAY')
7*   "Tomorrow:"
7* from dual
```

```
NumberOfMontsIHaveLivedSoFar  YearsIHaveLivedSoFar  Tomorrow:
-----
330.19355                      27.5  30-AUG-02
```

- Note in the example above, that the order in which the dates appear is important...in this case, we had to multiply the result by -1 in order to get positive results.
- ROUND(date [, 'fmt']) : Returns date rounded to the unit specified by the format model fmt. If the format model fmt is omitted, date is rounded to the nearest day.
- TRUNC(date [, 'fmt']) : Returns date with the time portion of the day truncated to the unit specified by the format model fmt. If the format model fmt is omitted, date is truncated to the nearest day.

```
SQL> r
1  SELECT empno, hiredate,
2  ROUND(hiredate, 'MONTH'), TRUNC(hiredate, 'MONTH')
3  FROM emp
4* WHERE hiredate LIKE '%81%'
```

EMPNO	HIREDATE	ROUND(HIR	TRUNC(HIR
7499	20-FEB-81	01-MAR-81	01-FEB-81
7521	22-FEB-81	01-MAR-81	01-FEB-81
7566	02-APR-81	01-APR-81	01-APR-81
7698	01-MAY-81	01-MAY-81	01-MAY-81
7782	09-JUN-81	01-JUN-81	01-JUN-81
7839	17-NOV-81	01-DEC-81	01-NOV-81

- **Conversion Functions:**

- In addition to Oracle datatypes, columns of tables in an Oracle8 database can be defined using ANSI dB2, and SQL/DS datatypes. However, the Oracle Server internally converts such datatypes to Oracle8 datatypes.
- In some cases, Oracle Server allows data of one datatype where it expects data of a different datatype. This is allowed when Oracle Server can automatically convert the data to the expected datatype. This datatype conversion can be done implicitly by Oracle Server or explicitly by the user.
- **Implicit** datatype conversions work according to the rules explained in the following sections.
- **Explicit** datatype conversions are done by using the conversion functions. The first datatype is the input datatype; the last datatype is the output datatype. It is recommended to always explicitly convert datatypes for clarity.
 - TO_CHAR(number | Date, [fmt], [nlsparams]): Converts a number or data value to a VARCHAR2 character string with format model fmt. For number conversions, the nlsparams parameter specifies the following characters, which are returned by number format elements:
 - Decimal Character
 - Group Separator
 - Local Currency Symbol
 - International Currency Symbol
 - TO_NUMBER(char, [fmt], [nlsparams]): Converts a character string containing digits to a number in the format specified by the optional format model 'fmt'.
 - TO_DATE(char, [fmt], [nlsparams]): Converts a character string representing a date to a date value according to the fmt specified. If fmt is omitted, the default format is used.
- NOTE: There is a [list of time/date formats in 3-30 and 3-31](#) of Study Guide.

```
SQL> r
  1  select ename, TO_CHAR(hiredate, 'fmDD Month YYYY') "Hire
Date"
  2* from emp
```

ENAME	Hire Date
SMITH	17 December 1980
ALLEN	20 February 1981

Another example:

```
SQL> r
  1  select ename,
  2  TO_CHAR(hiredate, 'fmDdspth "of" MONTH YYYY fmHH:MI:SS AM')
  3* from emp
```

ENAME	TO_CHAR(HIREDATE, 'FMDDSPTH"OF"MONTHYYYYFMHH:
SMITH	Seventeenth of DECEMBER 1980 12:00:00 AM
ALLEN	Twentieth of FEBRUARY 1981 12:00:00 AM

- Note that the 'fm' that precedes Ddspth is used to remove padded blanks or suppress leading zeroes. Also note that 'Dd' in Ddspth Cause the Day to be displayed with the first character capitalized. Further, the 'sp' means 'spelled out' and 'th' means 'display ordinal'.
- Using TO_CHAR() with numbers...(see 3-33 of Study Guide for formatting syntax).

```
SQL> r
  1  select TO_CHAR(sal, '$99,999')
  2  from emp
  3* where sal between 1000 and 3000
```

```
TO_CHAR(
-----
$1,500
$1,000
$1,000
$2,000
```

- Note that Oracle Server will display (#'s) if the value in a field is greater than that allowed by the formatting model.
- NVL Function converts null to an actual value. Datatypes that can be used as arguments are Date, Character, and Number. The syntax is: NVL(expr1, expr2) where expr1 is the source value or expression that may contain a null value, and expr2 is the target value for converting null into.

```
SQL> r
  1  select empno, ename, job, NVL(job, 'No Job Yet')
  2* from emp
```

EMPNO	ENAME	JOB	NVL(JOB, 'No Job Yet')
7566	JONES	MANAGER	MANAGER
9600	BUSH	ANALYST	
3434	SI		No Job Yet
3535	VIC_SANC		No Job Yet

```
SQL> r
  1  select empno, sal, sal*100, NVL(sal, 2), NVL(sal, 2)*100
  2  from emp
  3* where empno = 3434 or empno = 3535
```

EMPNO	SAL	SAL*100	NVL(SAL, 2)	NVL(SAL, 2)*100
3535			2	200
3434			2	200

- Note here that by using NVL, we transform NULL into a number 1, which is then multiplied by 100 resulting in a result of 200.

- DECODE Function: Facilitates conditional inquiries by doing the work of a CASE or IF-THEN-ELSE statement...here's the syntax:

```
DECODE (col/expression, search1, result1,
        [, search2, result2,...,], [, default])
```

- This function decodes *expression* after comparing it to each search value. If the *expression* is the same as *search*, result is returned.

```
SQL> SELECT job, sal,
2  DECODE(job, 'ANALYST', SAL*2.0,
3          'SALESMAN', SAL*3.0,
4          'MANAGER', SAL*5.0,
5          SAL)
6  REVISED_SALARY
7* FROM emp
```

JOB	SAL	REVISED_SALARY
ANALYST	1500	3000
SALESMAN	1000	3000
MANAGER	2000	10000

- Notice how REVISED_SALARY is incremented differently according to title as prescribed by the DECODE function. In this function, the 'job' inside the DECODE parentheses is the argument whose values you are to decode (i.e., 'ANALYST', 'SALESMAN', etc...), then the value that SAL*2.0 evaluates to is used as the resulting value placed under column 'REVISED_SALARY' which is itself an alias. The last 'SAL' inside the DECODE parentheses is the default value in the event that a value of 'job' is not matched...for example, if there is a title not included in the list of values to match.
- Functions can be nested and I know how to do that.