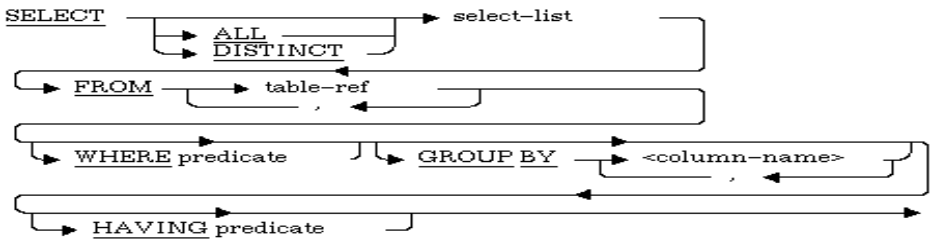
Chapter 5 (SELECT Statements)



"Few things are worse than that moment during an argument when you realize you're wrong."

5.1 What is a SELECT statement

Most of the SQL operations performed on a database in a typical organization are SELECT statements, which enable users to retrieve data from tables. The SELECT statement asks the database a question, which is why it's also known as a query.

```
SELECT  [DISTINCT | UNIQUE] (*, columnname [ AS alias], …)
        FROM      tablename
        [WHERE    condition]
        [GROUP BY group_by_expression]
        [HAVING   group_condition]
        [ORDER BY columnname];
```

The only clauses required for the SELECT statement are SELECT and FROM. Square brackets indicate optional portions of the statement. SQL statements can be entered over several lines or on one line. Most SQL statements are entered with each clause on a separate line to improve readability and make editing easier.

The examples in this section are based on the following data:

```
DROP TABLE patient;
CREATE  TABLE Patient
(
      Patient_id  NUMBER PRIMARY KEY,
      Fname       VARCHAR2(20),
      Lname       VARCHAR2(20),
      Gender            CHAR,
      DOB         DATE,
      salary            NUMBER ,
```

```
                city                          VARCHAR2(20),
                state                       VARCHAR2(20)
);

INSERT INTO patient values (111,'john','Doe','m','11-FEB-1978',25000,
'Davis','CA');
INSERT INTO patient values (112,'john','Smith','m','01-MAR-1981',40000,
'Davis','CA');
INSERT INTO patient values (113,'jill','Crane','m','12-APR-
1999',50000,'Reno','NV');
INSERT INTO patient values (114,'billy','Bob','f','05-MAY-1985',60000,'Las
Vegas','NV');
INSERT INTO patient values (115,'dove','Grime','f','04-JUN-
1960',20000,'Sacramento','CA');
```

5.2 What is a function

A function is a predefined block of code that accepts one or more arguments— values listed inside parentheses— and then returns a single value as output. The nature of an argument depends on the syntax of the function being executed. Single- row functions return one row of results for each record processed. By contrast, multiple- row functions return only one result per group or category of rows processed, such as counting the number of books published by each publisher.

Any single- row function can be nested inside other single- row functions. When nesting functions, you should remember the following important rules: ?All arguments required for each function must be provided. ?Every opening parenthesis must have a corresponding closing parenthesis. ?The nested, or inner, function is evaluated first. The inner function's result is then passed to the outer function, and the outer function is executed.

5.3 Simple Select clause

The asterisk (*) is a symbol that instructs Oracle to include all columns in the table. This symbol can be used only in the SELECT clause of a SELECT statement. If you need to view or display all columns in a table, typing an asterisk is much simpler than typing the name of each column. If the table contains a large number of fields, the results might look cluttered, or maybe, the table contains sensitive data you don't want other users to see. In these situations, you can instruct Oracle to return only specific columns in the results. Choosing specific columns in a SELECT statement is called projection. You can select one column— or as many as all columns— contained in the table.

```
SELECT * FROM patient;

--separate each of the columns with a comma.
```

```
SELECT fname, lname FROM patient;
```

PATIENT_ID	FNAME	LNAME	GENDER	DOB	SALARY	CITY	STATE
111	john	Doe	m	11-FEB-78	25000	Davis	CA
112	john	Smith	m	01-MAR-81	40000	Davis	CA
113	jill	Crane	m	12-APR-99	50000	Reno	NV
114	billy	Bob	f	05-MAY-85	60000	Las Vegas	NV
115	dove	Grime	f	04-JUN-60	20000	Sacramento	CA

FNAME	LNAME
john	Doe
john	Smith
jill	Crane
billy	Bob
dove	Grime

5.4 Alias

Sometimes a column name is a vague indicator of the data that's displayed. To better describe the data displayed in the output, you can substitute a column alias for the column name in query results. You need to keep some guidelines in mind when using a column alias. If the column alias contains spaces or special symbols, or if you don't want it to appear in all uppercase letters, you must enclose it in quotation marks (" "). By default, column headings shown in query results are capitalized. Using quotation marks overrides this default setting. If the column alias consists of only one word without special symbols, it doesn't need to be enclosed in quotation marks

```
--Can modify the headings with your choice of alias. This does not change the underlying
--columnname in the table. It only displays a different heading for this one SQL statement.
SELECT fname, lname AS lastname FROM patient;


--Can also come up with an alias without using the AS keyword.
SELECT fname, lname  lastname FROM patient;


--If an alias is comprised of multiple words then enclose it in double quotes.
SELECT fname, lname  "Last Name"  FROM patient;
```

```
SQL> SELECT fname, lname AS lastname FROM patient;

FNAME                LASTNAME
-------------------- --------------------
john                 Doe
john                 Smith
jill                 Crane
billy                Bob
dove                 Grime

SQL> SELECT fname, lname  lastname FROM patient;

FNAME                LASTNAME
-------------------- --------------------
john                 Doe
john                 Smith
jill                 Crane
billy                Bob
dove                 Grime

SQL> SELECT fname, lname  "Last Name"  FROM patient;

FNAME                Last Name
-------------------- --------------------
john                 Doe
john                 Smith
jill                 Crane
billy                Bob
dove                 Grime
```

5.5 Concatenation

Use the concatenation operator (| |) to concatenate, or combine, data from columns with string literals.

```
--Notice the comma is removed and everything is displayed as a single columns
SELECT fname || lname || gender FROM patient;
-- spaces do not matter
--SELECT fname||lname||gender FROM patient;
--can add spaces to make the info more readable. Also can add an alias
SELECT fname || '   '|| lname ||': ' ||  gender  AS Info FROM patient;
```

```
SQL> SELECT fname || lname || gender FROM patient;

FNAME||LNAME||GENDER
------------------------------------
johnDoem
johnSmithm
jillCranem
billyBobf
doveGrimef

SQL> -- spaces do not matter
SQL> --SELECT fname||lname||gender FROM patient;
SQL> SELECT fname || '   '|| lname ||': ' ||  gender  AS Info FROM patient;

INFO
-------------------------------------------
john   Doe: m
john   Smith: m
jill   Crane: m
billy   Bob: f
dove   Grime: f
```

The CONCAT function can also be used to concatenate data from two columns. The main difference between the concatenation operator and the CONCAT function is that you can combine a long list of columns and string literals with the concatenation operator. By contrast, you can combine only two items (columns or string literals) with the CONCAT function.

The concatenation operator is usually preferred because it's not limited to two items. If you need to combine more than two items with the CONCAT function, you must nest a CONCAT function inside another CONCAT function.

```
--Instead of the pipe symbols, the concat function can also be used. It only takes two arguments.
SELECT CONCAT (fname, lname) FROM patient;

SELECT CONCAT (fname, lname || gender)  FROM patient;



--Can embed one concat inside of another.
SELECT CONCAT (fname,CONCAT( lname, gender||lname))  as  heading FROM
patient;
```

```
SQL> --Instead of the pipe symbols, the concat function can also be used. I
y takes two arguments
SQL> SELECT CONCAT (fname, lname) FROM patient;

CONCAT(FNAME,LNAME)
---------------------------------------
johnDoe
johnSmith
jillCrane
billyBob
doveGrime

SQL> SELECT CONCAT (fname, lname || gender)  FROM patient;

CONCAT(FNAME,LNAME||GENDER)
---------------------------------------
johnDoem
johnSmithm
jillCranem
billyBobf
doveGrimef

SQL> --Can embed one concat inside of another
SQL> SELECT CONCAT (fname,CONCAT( lname, gender||lname))  as  heading FROM
nt;

HEADING
------------------------------------------------------------
johnDoemDoe
johnSmithmSmith
jillCranemCrane
billyBobfBob
doveGrimefGrime
```

5.6 LTRIM/RTRIM/TRIM

You can use the LTRIM function to remove a specific string of characters from the left side of data values. The syntax of the LTRIM function is LTRIM(c, s), where c represents the field to modify, and s represents the string to remove from the left side of data.

Oracle also supports the RTRIM function to remove specific characters from the right side of data values. The syntax of the RTRIM function is RTRIM(c, s). The c represents the field to modify, and s represents the string to remove from the right side of data. The TRIM function has several variations that can potentially replace both RTRIM and LTRIM.

```
--This record is being inserted so that we can remove the hyphens using the trim function.
INSERT INTO patient VALUES (777,'---john----', ' Doe','f','05-DEC-
1990',90000, 'Davis','CA');


--Removes the dashes from the left side and right side: Keep in mind this is for display purposes
only.
-- It doesn't change the actual data in the table.
SELECT LTRIM(fname,'-'), RTRIM(fname,'-') FROM patient;

--Can also use the trim function with LEADING or TRAILING options. Without those options it will
--look at both the left and the right sides.
SELECT TRIM(LEADING '-' FROM fname),TRIM (TRAILING'-' FROM fname),
TRIM('-'  FROM  fname) FROM patient;
```

```
e trim function
SQL> INSERT INTO patient VALUES (777,'---john----', ' Doe','f','05-DEC-1990
00, 'Davis','CA');

1 row created.

SQL> --removes the trims from the left side and right side: Keep in mind th
 for display. It doesn't actually change
SQL> --the date in the table
SQL> SELECT LTRIM(fname,'-'), RTRIM(fname,'-') FROM patient;

LTRIM(FNAME,'-')     RTRIM(FNAME,'-')
-------------------- --------------------
john                 john
john                 john
jill                 jill
billy                billy
dove                 dove
john----             ---john

6 rows selected.

SQL> SELECT TRIM(LEADING '-' FROM fname),TRIM (TRAILING'-' FROM fname), TRI
  FROM  fname) FROM patient;

TRIM(LEADING'-'FROMF TRIM(TRAILING'-'FROM TRIM('-'FROMFNAME)
-------------------- -------------------- --------------------
john                 john                 john
john                 john                 john
jill                 jill                 jill
billy                billy                billy
dove                 dove                 dove
john----             ---john              john

6 rows selected.
```

5.7 DISTINCT or UNIQUE

The DISTINCT/UNIQUE keyword eliminates duplicate values in the results.

```
SELECT city, state  FROM patient;

--Gets rid of the duplicate cities.
SELECT DISTINCT city FROM patient;

--Gets rid of the duplicate city, state combination like a composite key.
SELECT DISTINCT city, state FROM patient;

--INVALID. Must apply DISTINCT to the entire row.
SELECT city, DISTINCT state FROM patient;

--Can also use the UNIQUE keyword instead of distinct.
SELECT UNIQUE city, state FROM patient;
```

```
SQL> SELECT city, state  FROM patient;

CITY                 STATE
-------------------- --------------------
Davis                CA
Davis                CA
Reno                 NV
Las Vegas            NV
Sacramento           CA
Davis                CA

6 rows selected.

SQL> --Gets rid of the duplicate cities
SQL> SELECT DISTINCT city FROM patient;

CITY
--------------------
Las Vegas
Davis
Sacramento
Reno

SQL> --Gets rid of the duplicate city, state combination like a composite k
SQL> SELECT DISTINCT city, state FROM patient;

CITY                 STATE
-------------------- --------------------
Sacramento           CA
Las Vegas            NV
Reno                 NV
Davis                CA

SQL> --INVALID. Must apply DISTINCT to the entire row
SQL> SELECT city, DISTINCT state FROM patient;
SELECT city, DISTINCT state FROM patient
             *
ERROR at line 1:
ORA-00936: missing expression
```

```
SQL> --can also use the UNIQUE keyword instead of distinct
SQL> SELECT UNIQUE city, state FROM patient;

CITY                 STATE
-------------------- --------------------
Sacramento           CA
Las Vegas            NV
Reno                 NV
Davis                CA

SQL>
```

✓ *CHECK 5A*

1. Create the person table and insert some records before starting with the questions.
2. Display the contents of the Person table. Use the alias Last name.
3. Concatenate the firstname, lastname and the salary separated by a space, using the concat function.
4. Display all the unique lastnames from the Person table.

"Great minds discuss ideas, Average minds discuss events, Small minds discuss people"

5.8 DUAL table

Any of the single- row functions covered in this chapter can be used with the DUAL table

Example 5.8a (Dual table)

```
--The dual table contains only one column called dummy.
DESC DUAL;

--There is only a single row of information in this table which is (X).
SELECT * FROM DUAL;
```

```
DESC DUAL
Name  Null Type
----- ---- -----------
DUMMY      VARCHAR2(1)

DUMMY
-----
X
```

Example 5.8b (Using the Dual table)

```
--The reason why we want to use the dual table is because there is only one record. In this
--instance, notice it displays the literal text (hello) fror every record
SELECT 'hello', fname FROM patient;
SELECT 'hello' FROM patient;

--Displays hello only one time
SELECT 'hello' FROM dual;
```

```
SQL> --The reason why we want to use the dual table is because there is onl
 record. In this
SQL> --instance, notice it displays the literal text (hello) fror every rec
SQL> SELECT 'hello', fname FROM patient;

'HELL FNAME
----- --------------------
hello john
hello john
hello jill
hello billy
hello dove
hello ---john----

6 rows selected.

SQL> SELECT 'hello' FROM patient;

'HELL
-----
hello
hello
hello
hello
hello
hello

6 rows selected.

SQL> --Displays hello only one time
SQL> SELECT 'hello' FROM dual;

'HELL
-----
hello
```

```
--When displaying the square root, we only want to see the result one time. We don't want it
--to be repeated for each row of a table. It is redundant which is why we would use the dual table.
SELECT SQRT(4), fname FROM patient;

SELECT SQRT(4) FROM patient;

SELECT SQRT(4) FROM dual;
```

```
SQL> --When displaying the square root, we only want to see the result one
 We don't want it
SQL> --to be repeated for each row of a table. It is redundant
SQL> SELECT SQRT(4), fname FROM patient;

   SQRT(4) FNAME
---------- --------------------
         2 john
         2 john
         2 jill
         2 billy
         2 dove
         2 ---john----

6 rows selected.

SQL> SELECT SQRT(4) FROM patient;

   SQRT(4)
----------
         2
         2
         2
         2
         2
         2

6 rows selected.

SQL> SELECT SQRT(4) FROM dual;

   SQRT(4)
----------
         2
```

5.9 INITCAP

The initcap function sets the first character in each word to uppercase and the rest to lowercase.
The syntax for the initcap function is: initcap(string1) where string1 is the string argument whose first character in each word will be converted to uppercase and all remaining characters converted to lowercase.

```
--Does not change, just displays. First letter is upper cased.
SELECT INITCAP('tech on the net')  FROM dual;

--First letter is uppercased.
SELECT INITCAP('GEORGE SOROS' ) FROM dual;
SELECT INITCAP(fname) FROM patient;
```

```
SQL> --Does not change, just displays. First letter is upper cased
SQL> SELECT INITCAP('tech on the net')  FROM dual;

INITCAP('TECHON
---------------
Tech On The Net

SQL> --First letter is uppercased
SQL> SELECT INITCAP('GEORGE SOROS' ) FROM dual;

INITCAP('GEO
------------
George Soros

SQL> SELECT INITCAP(fname) FROM patient;

INITCAP(FNAME)
--------------------
John
John
Jill
Billy
Dove
---John----

6 rows selected.
```

5.10 SUBSTR, INSTR and REPLACE

The INSTR (instring) function searches a string for a specified set of characters or a sub-string, and then returns a numeric value representing the first character position in which the substring is found. If the substring doesn't exist in the string value, a 0 (zero) is returned. Two arguments must be provided to the INSTR function: the string value to search and the characters or substring (enclosed in single quotes) to locate. Two optional arguments are also available: start position, indicating on which character of the string value the search should begin, and occurrence, which is the instance of the search value to locate (that is, first occurrence, second occurrence, and so on). By default, the search begins at the beginning of the string value, and the position of the first occurrence is located.

Example 5.10a (Instr)

```
--Return first occurrence of 'e' .
SELECT INSTR ('Tech on the net', 'e') FROM DUAL;

--The first occurrence of 'e'.
SELECT INSTR ('Tech on the net', 'e', 1, 1) FROM DUAL;
```

```
SQL> --Return first occurrence of "e"
SQL> SELECT INSTR ('Tech on the net', 'e') FROM DUAL;

INSTR('TECHONTHENET','E')
-------------------------
                        2

SQL>
SQL> --The first occurrence of 'e'
SQL> SELECT INSTR ('Tech on the net', 'e', 1, 1) FROM DUAL;

INSTR('TECHONTHENET','E',1,1)
-----------------------------
                            2
```

```
--The second occurrence of 'e'.
SELECT INSTR ('Tech on the net', 'e', 1, 2) FROM DUAL;

--The third occurrence of 'e'.
SELECT INSTR ('Tech on the net', 'e', 1, 3) FROM DUAL;

--Looks for the first occurrence of the letter (l).
SELECT  INSTR(fname,'l') , fname FROM PATIENT;
```

```
SQL> --The second occurrence of 'e'
SQL> SELECT INSTR ('Tech on the net', 'e', 1, 2) FROM DUAL;

INSTR('TECHONTHENET','E',1,2)
-----------------------------
                           11

SQL>
SQL> --The third occurrence of 'e'
SQL> SELECT INSTR ('Tech on the net', 'e', 1, 3) FROM DUAL;

INSTR('TECHONTHENET','E',1,3)
-----------------------------
                           14

SQL>
SQL> SELECT  INSTR(fname,'l') , fname FROM PATIENT;

INSTR(FNAME,'L') FNAME
---------------- --------------------
               0 john
               0 john
               3 jill
               3 billy
               0 dove
               0 ---john----

6 rows selected.
```

Example 5.10b (Substr)

You can use the SUBSTR function to return a substring (a portion of a string). The syntax of this function is SUBSTR(c, p, l), where c represents the character string, p represents the beginning character position for the extraction, and l represents the length of the string to return in the query results.

```
--Extracts two characters, starting from the 6th position.
SELECT  SUBSTR ('This is a test', 6, 2) FROM DUAL;

-- If the second parameter is omitted, substr will return the entire string.
SELECT  SUBSTR ('This is a test', 6) FROM DUAL;
```

```
SQL> --extracts two characters starting from the 6th position
SQL> SELECT  SUBSTR ('This is a test', 6, 2) FROM DUAL;

SU
--
is

SQL> -- If the second parameter is omitted, substr will return

SQL> SELECT  SUBSTR ('This is a test', 6) FROM DUAL;

SUBSTR('T
---------
is a test
```

```
--Extracts four characters, starting from the 1st position.
SELECT  SUBSTR ('TechOnTheNet', 1, 4) FROM DUAL;

--If start_position is a negative number, then it starts from  the end of the string and counts backwards.
SELECT  SUBSTR ('TechOnTheNet', -3, 3) FROM DUAL;
SELECT  SUBSTR ('TechOnTheNet', -6, 3) FROM DUAL;
SELECT  SUBSTR ('TechOnTheNet', -8, 2) FROM DUAL;
```

```
SQL> --extracts four characters starting from the 1st position
SQL> SELECT  SUBSTR ('TechOnTheNet', 1, 4) FROM DUAL;

SUBS
----
Tech

SQL>
SQL> --If start_position is a negative number, then substr sta
SQL> -- the end of the string and counts backwards.
SQL> SELECT  SUBSTR ('TechOnTheNet', -3, 3) FROM DUAL;

SUB
---
Net

SQL> SELECT  SUBSTR ('TechOnTheNet', -6, 3) FROM DUAL;

SUB
---
The

SQL> SELECT  SUBSTR ('TechOnTheNet', -8, 2) FROM DUAL;

SU
--
On
```

```
SELECT  SUBSTR (fname,2) , fname FROM patient;
```

```
SQL> SELECT  SUBSTR (fname,2) , fname FROM patient;

SUBSTR(FNAME,2)      FNAME
-------------------- --------------------
ohn                  john
ohn                  john
ill                  jill
illy                 billy
ove                  dove
--john----           ---john----

6 rows selected.
```

Example 5.10c (Replace)

The REPLACE function is similar to the "search and replace" function used in many programs. It looks for the occurrence of a specified string of characters and, if found, substitutes it with another set of characters. The syntax of the REPLACE function is REPLACE(c, s, r), where c represents the field to search, s represents the string of characters to find, and r represents the string of characters to substitute for s.

```
-- Gets rid of everything in the second argument. It is looking for the exact pattern.
SELECT REPLACE ('123123tech', '123') FROM DUAL;

-- Doesn't matter where the 123 is.
SELECT  REPLACE('123tech123', '123') FROM DUAL;

--Replaces all every 2 with a 3.
SELECT REPLACE('222tech', '2', '3') FROM DUAL;

-- Gets rid of all the zeros.
SELECT REPLACE('0000123', '0') FROM DUAL;

-- Replaces the zeros with blank spaces.
SELECT REPLACE('0000123', '0', '  ') FROM DUAL;
```

```
SQL> -- Gets rid of everything in the second argument. It is l
ct pattern. For example
SQL> --if the second argument is 122, it does not get rid of a
ch a a pattern does not --- exist
SQL> SELECT  REPLACE ('123123tech', '123') FROM DUAL;

REPL
----
tech

SQL> -- Doesn't matter where the 123 is
SQL> SELECT  REPLACE('123tech123', '123') FROM DUAL;

REPL
----
tech

SQL> --replaces all every 2 with a 3
SQL> SELECT  REPLACE('222tech', '2', '3') FROM DUAL;

REPLACE
-------
333tech

SQL> -- gets rid of all the zeros
SQL> SELECT  REPLACE('0000123', '0') FROM DUAL;

REP
---
123

SQL> -- Replaces the zeros with blank spaces
SQL> SELECT  REPLACE('0000123', '0', ' ') FROM DUAL;

REPLACE('00
-----------
        123
```

SELECT REPLACE(fname,'bill',’kill’) changed, fname FROM patient;

```
SQL> SELECT  REPLACE(fname,'bill','kill') changed, fname  FRO

CHANGED    FNAME
---------- --------------------
john       john
john       john
jill       jill
killy      billy
dove       dove
---john--- ---john----
-


6 rows selected.
```

Example 5.10d (combination of functions)

```
DELETE FROM patient;
INSERT INTO patient VALUES (999,'Bond, James', '','f','05-DEC-
1990',90000, 'Davis','CA');

SELECT INSTR(fname,',')-1, INSTR(fname,',')+ 1 FROM patient;
--Column contains lastname and first name separated by a comma. Want only the last name (Bond).
SELECT SUBSTR(fname,1, INSTR(fname,',')-1) FROM patient;

-- Want to extract the first name. This includes leading spaces.
SELECT SUBSTR(fname,INSTR(fname,',')+1) FROM patient;

--Get rid of the spaces.
SELECT LTRIM(SUBSTR(fname,INSTR(fname,',')+1)) FROM patient;
```

```
SQL> DELETE FROM patient;

6 rows deleted.

SQL> INSERT INTO patient VALUES (999,'Bond, James', '','f','05-DEC-1990',90000,
'Davis','CA');

1 row created.

SQL>
SQL> SELECT INSTR(fname,',')-1, INSTR(fname,',')+ 1 FROM patient;

INSTR(FNAME,',')-1 INSTR(FNAME,',')+1
------------------ ------------------
                 4                  6

SQL> --column contains lasname and first name separated by a comma
SQL> -- Want only the last name (Bond)
SQL> SELECT   SUBSTR(fname,1, INSTR(fname,',')-1) FROM patient;

SUBSTR(FNAME,1,INSTR
--------------------
Bond

SQL>
SQL> -- Want to extract the first name. This includes leading spaces
SQL> SELECT   SUBSTR(fname,INSTR(fname,',')+1) FROM patient;

SUBSTR(FNAME,INSTR(F
--------------------
 James

SQL>
SQL> --Get rid of the spaces
SQL> SELECT   LTRIM(SUBSTR(fname,INSTR(fname,',')+1)) FROM patient;

LTRIM(SUBSTR(FNAME,I
--------------------
James
```

✓ *CHECK 5B*

1. Display the first three letters of the last name. Make sure the first letter is in uppercase.
2. Identify the location of the letter 'e' in all the firstnames.

"The trouble with our times is that the future is not what it used to be. "

5.11 LPAD, RPAD

The LPAD function can be used to pad, or fill in, the area to the left of a character string with a specific character— or even a blank space. The syntax of the LPAD function is LPAD(c, l, s), where **c** represents the character string to pad, l represents the length of the character string after padding, and s represents the symbol or character (enclosed in single quotes) to use as padding, RPAD is similar except that it pads from the right hand side.

```
--Pad the left hand side with dots.
SELECT LPAD(fname, 20, '.')  FROM patient;

--Pad the right hand side with dots.
SELECT RPAD(fname,20,'.') FROM patient;
```

```
SQL> --pad the left hand side with dots
SQL> SELECT LPAD(fname, 20, '.')  FROM patient;

LPAD(FNAME,20,'.')
--------------------
.........Bond, James

SQL> --pad the right hand side with dots
SQL> SELECT RPAD(fname,20,'.') FROM patient;

RPAD(FNAME,20,'.')
--------------------
Bond, James.........
```

TRUNC, ROUND, FLOOR, CEIL

The syntax of the ROUND function is ROUND (d, u), where d represents the date data, or field, to round, and u represents the unit to use for rounding. A date can be rounded by the unit of month or year.

Example 5.12a (Round)

```
SELECT ROUND(12.5) FROM DUAL;
SELECT ROUND(12.1) FROM DUAL;
SELECT ROUND(12.56,1) FROM DUAL;
SELECT ROUND(12.54,1) FROM DUAL;
```

```
SQL> SELECT ROUND(12.5) FROM DUAL;
ROUND(12.5)
-----------
         13
SQL> SELECT ROUND(12.1) FROM DUAL;
ROUND(12.1)
-----------
         12
SQL> SELECT ROUND(12.56,1) FROM DUAL;
ROUND(12.56,1)
--------------
          12.6
SQL> SELECT ROUND(12.54,1) FROM DUAL;
ROUND(12.54,1)
--------------
          12.5
```

Example 5.12b (Trunc)

At times you need to truncate, rather than round, numeric data. You can use the TRUNC (truncate) function to truncate a numeric value to a specific position. Any numbers after that position are simply removed (truncated). The syntax of the TRUNC function is TRUNC(n, p), where n represents the numeric data or field to truncate, and p represents the position of the digit where data should be truncated.

```
SELECT TRUNC(12.549,2) FROM DUAL;
SELECT TRUNC(12.5) FROM DUAL;
SELECT TRUNC(12.1) FROM DUAL;
SELECT TRUNC(12.56,1) FROM DUAL;
SELECT TRUNC(12.54,1) FROM DUAL;
SELECT TRUNC(12.549,2) FROM DUAL;
```

```
SQL> SELECT TRUNC(12.549,2) FROM DUAL;
TRUNC(12.549,2)
---------------
          12.54
SQL> SELECT TRUNC(12.5) FROM DUAL;
TRUNC(12.5)
-----------
         12
SQL> SELECT TRUNC(12.1) FROM DUAL;
TRUNC(12.1)
-----------
         12
SQL> SELECT TRUNC(12.56,1) FROM DUAL;
TRUNC(12.56,1)
--------------
          12.5
SQL> SELECT TRUNC(12.54,1) FROM DUAL;
TRUNC(12.54,1)
--------------
          12.5
SQL> SELECT TRUNC(12.549,2) FROM DUAL;
TRUNC(12.549,2)
---------------
          12.54
```

Example 5.12c (Ceil)

Returns the ceiling value (next highest integer above a number).
The syntax for the ceil function is: ceil(number)

```
SELECT CEIL(12.5) FROM DUAL;

SELECT CEIL(12.1) FROM DUAL;

SELECT CEIL(13.9) FROM DUAL;
```

```
SQL> SELECT CEIL(12.5) FROM DUAL;
CEIL(12.5)
----------
        13
SQL> SELECT CEIL(12.1) FROM DUAL;
CEIL(12.1)
----------
        13
SQL> SELECT CEIL(13.9) FROM DUAL;
CEIL(13.9)
----------
        14
```

Example 5.12d (Floor)

The FLOOR function rounds the specified number down.
The syntax for the floor function is: floor(number)

```
SELECT FLOOR(12.5) FROM DUAL;

SELECT FLOOR(12.1) FROM DUAL;

SELECT FLOOR(13.9) FROM DUAL;
```

```
SQL> SELECT FLOOR(12.5) FROM DUAL;
FLOOR(12.5)
-----------
         12
SQL> SELECT FLOOR(12.1) FROM DUAL;
FLOOR(12.1)
-----------
         12
SQL> SELECT FLOOR(13.9) FROM DUAL;
FLOOR(13.9)
-----------
         13
```

5.12 Arithmetic

Simple arithmetic operations, such as multiplication (*), division (/), addition (þ), and subtraction (-), can be used in the SELECT clause of a query. Keep in mind that Oracle adheres to the standard order of operations: 1. Moving from left to right in the arithmetic equation, any required multiplication and

division operations are solved first. 2. Addition and subtraction operations are solved after multiplication and division, again moving from left to right in the equation. To override this order of operations, you can use parentheses to enclose the portion of the equation that should be calculated first.

Example 5.13a (Basic math)

```
--Does not change the underlying table.
SELECT salary, salary +200, salary/2, salary * 2, salary-200 FROM
patient;
```

```
SQL> SELECT salary, salary +200, salary/2, salary * 2, salary-200 FROM patient;

    SALARY SALARY+200   SALARY/2   SALARY*2 SALARY-200
---------- ---------- ---------- ---------- ----------
     90000      90200      45000     180000      89800

SQL>
```

Example 5.13b (Mod)

The MOD (modulus) function returns only the remainder of a division operation.

```
SELECT salary, MOD(salary, 2) FROM patient;
```

```
SQL> SELECT salary, MOD(salary, 2) FROM patient;

    SALARY MOD(SALARY,2)
---------- -------------
     90000             0
```

Example 5.13c (ABS)

The ABS (absolute) function returns the absolute, or positive, value of the numeric values supplied as the argument.

```
SELECT ABS(0) FROM DUAL;
SELECT ABS(10) FROM DUAL;
SELECT ABS(-10) FROM DUAL;
```

```
SQL> SELECT ABS(0) FROM DUAL;

    ABS(0)
----------
         0

SQL> SELECT ABS(10) FROM DUAL;

   ABS(10)
----------
        10

SQL> SELECT ABS(-10) FROM DUAL;

  ABS(-10)
----------
        10
```

Example 5.13d (Power)

The POWER function raises the number in first argument to the power indicated as the second argument. The syntax of the POWER function is POWER(x, y), where x represents the number you're raising, and y represents the power to which you're raising it.

```
SELECT POWER(5,3) FROM DUAL;

SELECT POWER(5,0) FROM DUAL;
```

```
SQL> SELECT POWER(5,3) FROM DUAL;

POWER(5,3)
----------
       125

SQL> SELECT POWER(5,0) FROM DUAL;

POWER(5,0)
----------
         1
```

Example 5.13e (SQRT)

The SQRT function returns the square root of a number

```
SELECT SQRT(10) FROM DUAL;
```

```
SQL> SELECT SQRT(10) FROM DUAL;

  SQRT(10)
----------
3.16227766
```

5.13 GREATEST, LEAST

The greatest function returns the greatest value in a list of expressions. The syntax for the greatest function is: greatest(expr1, expr2, ... expr_n) where expr1, expr2, .expr_n are expressions that are

evaluated by the greatest function. If the datatypes of the expressions are different, all expressions will be converted to whatever datatype expr1 is.

The least function returns the smallest value in a list of expressions.The syntax for the least function is: least(expr1, expr2, ... expr_n) where expr1, expr2, .expr_n are expressions that are evaluated by the least function. If the datatypes of the expressions are different, all expressions will be converted to whatever datatype expr1 is.

```
SELECT GREATEST(10,60,90,3) FROM DUAL;
SELECT LEAST(10,60,90,3) FROM DUAL;
```

```
SQL> SELECT GREATEST(10,60,90,3) FROM DUAL;

GREATEST(10,60,90,3)
--------------------
                  90

SQL> SELECT LEAST(10,60,90,3) FROM DUAL;

LEAST(10,60,90,3)
-----------------
                3
```

5.14 Date functions

Oracle has a number of functions that apply to a date

Sysdate	Returns the current date/time
ADD_MONTHS	Function to add a number of months to a date. For example: add_months(SYSDATE,3) returns 3 months after sysdate. This could be rounded to below is the resulting month has fewer days than the month this function is applied to.
+,- (plus/minus)	In Oracle you can add or substract a number of days from a date. Example: sysdate+5 means systemdate/time plus 5 days
GREATEST	With the greatest function you can select the date/time that is the highest in a range of date/times. Example: greatest (sysdate+4,sysdate,sysdate-5) = sysdate+4.
LEAST	With the least function you can select the earliest date/time in a range of date/times. Example: least (sysdate+4,sysdate,sysdate-5) = sysdate-5.
LAST_DAY	Returns the last_day of a month based on the month the passed date is in. Example: last_day(sysdate) returns the last day of this month.
MONTHS_BETWEEN	Returns the number of months between two dates. The number is not rounded. Example: months_between(sysdate, to_date('01-01-2007','dd-mm-yyyy')) returns the number of months since jan 1, 2007.
NEXT_DAY	Date of next specified date following a date NEXT_DAY(,) Options are SUN, MON, TUE, WED, THU, FRI, and SAT SELECT NEXT_DAY(SYSDATE, 'FRI') FROM dual;
ROUND	Returns date rounded to the unit specified by the format model. If you omit the format, the date is rounded to the nearest day ROUND(,) SELECT ROUND(TO_DATE('27-OCT-00'),'YEAR') NEW_YEAR FROM dual;
TRUNC	Convert a date to the date without time (0:00h) Example: TRUNC(sysdate) returns today without time.
TO_CHAR(date,format_mask)	Converts a date to a string using a format mask.

Example 5.15a (Months_between)

The syntax is MONTHS_ BETWEEN(d1, d2), where d1 and d2 are the two dates in question, and d2 is subtracted from d1.To eliminate the decimal portion of the output and return only the number of whole months between the two dates, you can nest the MONTHS_ BETWEEN function inside the TRUNC function,

```
SELECT DOB, MONTHS_BETWEEN(SYSDATE, DOB) FROM patient;

SELECT DOB, TRUNC(MONTHS_BETWEEN(SYSDATE, DOB)) FROM patient;

--Can also find out the years by dividing the results of months_between into 12.
SELECT DOB, TRUNC(MONTHS_BETWEEN(SYSDATE, DOB)/12) FROM patient;

SELECT DOB, ROUND(MONTHS_BETWEEN(SYSDATE, DOB)/12) FROM patient;
```

```
SQL> SELECT dob, MONTHS_BETWEEN(SYSDATE, dob) FROM patient;

DOB       MONTHS_BETWEEN(SYSDATE,DOB)
--------- ---------------------------
05-DEC-90                   252.24253

SQL> SELECT dob, TRUNC(MONTHS_BETWEEN(SYSDATE, dob)) FROM patient;

DOB       TRUNC(MONTHS_BETWEEN(SYSDATE,DOB))
--------- ----------------------------------
05-DEC-90                                252

SQL> --can also find out the years by dividing the results of months_between i
o 12
SQL> SELECT dob, TRUNC(MONTHS_BETWEEN(SYSDATE, dob)/12) FROM patient;

DOB       TRUNC(MONTHS_BETWEEN(SYSDATE,DOB)/12)
--------- -------------------------------------
05-DEC-90                                    21

SQL> SELECT dob, ROUND(MONTHS_BETWEEN(SYSDATE, dob)/12) FROM patient;

DOB       ROUND(MONTHS_BETWEEN(SYSDATE,DOB)/12)
--------- -------------------------------------
05-DEC-90                                    21
```

Example 5.15b (Add_months)

The syntax of the ADD_ MONTHS function is ADD_ MONTHS(d, m), where d rep-resents the beginning *date* for the calculation, and m represents the number of months to add to the date.

```
SELECT DOB, ADD_MONTHS(DOB,4) FROM patient;
SELECT DOB, ADD_MONTHS(DOB,-14) FROM patient;
```

```
SQL> SELECT dob, ADD_MONTHS(dob,4) FROM patient;

DOB       ADD_MONTH
--------- ---------
05-DEC-90 05-APR-91

SQL> SELECT dob, ADD_MONTHS(dob,-14) FROM patient;

DOB       ADD_MONTH
--------- ---------
05-DEC-90 05-OCT-89
```

Example 5.15c (Next_day)

The syntax of the NEXT_ DAY function is NEXT_ DAY(d, DAY), where d represents the starting date, and DAY represents the day of the week to identify.The LAST_ DAY function is similar to the NEXT_ DAY function, except it always determines the last day of the month for a given date.

```
--The second argument  represents the day as follows: --Sunday=1, Monday=2, Tuesday=3, …
SELECT DOB,NEXT_DAY(DOB,4) , TO_CHAR(DOB,'DY') , TO_CHAR(NEXT_DAY(DOB,4)
,'DY')FROM patient;
SELECT DOB, LAST_DAY(DOB) , DOB FROM patient;
```

```
SQL> SELECT dob, NEXT_DAY(dob,4) , TO_CHAR(dob,'DY') , TO_CHAR(NEXT_DAY(dob,4) ,
'DY')FROM patient;

DOB       NEXT_DAY( TO_ TO_
--------- --------- --- ---
05-DEC-90 12-DEC-90 WED WED

SQL> SELECT dob, LAST_DAY(dob) , dob FROM patient;

DOB       LAST_DAY( DOB
--------- --------- ---------
05-DEC-90 31-DEC-90 05-DEC-90
```

Example 5.15d (Round)

```
SELECT sysdate FROM dual;
SELECT ROUND(sysdate,'YEAR') FROM dual;
SELECT ROUND(sysdate,'MONTH') FROM dual;
SELECT ROUND(TO_DATE('27-oct-2011'),'YEAR') FROM dual;
SELECT ROUND(TO_DATE('27-JAN-2011'),'YEAR') FROM dual;
SELECT ROUND(TO_DATE('27-OCT-2011'),'MONTH') FROM dual;
SELECT ROUND(TO_DATE('01-OCT-2011'),'MONTH') FROM dual;
```

```
SQL> SELECT sysdate FROM dual;
SYSDATE
---------
12-DEC-11
SQL> SELECT ROUND(sysdate,'YEAR') FROM dual;
ROUND(SYS
---------
01-JAN-12
SQL> SELECT ROUND(sysdate,'MONTH') FROM dual;
ROUND(SYS
---------
01-DEC-11
SQL> SELECT ROUND(TO_DATE('27-oct-2011'),'YEAR') FROM dual;
ROUND(TO_
---------
01-JAN-12
SQL> SELECT ROUND(TO_DATE('27-JAN-2011'),'YEAR') FROM dual;
ROUND(TO_
---------
01-JAN-11
SQL> SELECT ROUND(TO_DATE('27-OCT-2011'),'MONTH') FROM dual;
ROUND(TO_
---------
01-NOV-11
SQL> SELECT ROUND(TO_DATE('01-OCT-2011'),'MONTH') FROM dual;
ROUND(TO_
---------
01-OCT-11
```

Example 5.15e (Greatest, Least)

```
SELECT sysdate, sysdate+5, sysdate-4 FROM dual;
SELECT GREATEST(sysdate, sysdate+5, sysdate-4) FROM dual;
SELECT LEAST(sysdate, sysdate+5, sysdate-4) FROM dual;
```

```
SQL> SELECT sysdate, sysdate+5, sysdate-4 FROM dual;
SYSDATE   SYSDATE+5 SYSDATE-4
--------- --------- ---------
12-DEC-11 17-DEC-11 08-DEC-11
SQL> SELECT GREATEST(sysdate, sysdate+5, sysdate-4) FROM dual;
GREATEST(
---------
17-DEC-11
SQL> SELECT LEAST(sysdate, sysdate+5, sysdate-4) FROM dual;
LEAST(SYS
---------
08-DEC-11
```

✓ *CHECK 5C*

1. Using a select statement, display the age (make everyone six months older) in years.

"It is not enough to succeed. Others must fail. "

5.15 NVL and NVL2 Functions

You can use the NVL function to address problems caused when performing arithmetic operations with fields that might contain NULL values. When a NULL value is used in a calculation, the result is always a NULL value. The NVL function is used to substitute a value for the existing NULL so that the calculation can be completed. The syntax of the NVL function is NVL(x, y), where y represents the value to substitute if x is NULL. In many cases, the substitute for a NULL value in a calculation is zero (0).

Example 5.16a (NVL)

```
DELETE FROM patient;
INSERT INTO patient values (211,'john','Doe','m','11-FEB-1978',NULL,
'Davis','CA');
INSERT INTO patient values (219,'Billy','smith','f','19-FEB-
1998',2000, 'Sacramento','CA');

SELECT fname, salary FROM patient;

--Every salary that is NULL is replaced with zero.
SELECT fname, NVL (salary,0)  FROM patient;

--Every salary that is NULL is replaced by the text (poor). Notice that to_char is used
--because both arguments have to be the same type (text).
SELECT fname, NVL(TO_CHAR(salary),'poor') "SALary"  FROM patient;
```

```
SQL> DELETE FROM patient;

1 row deleted.

SQL> INSERT INTO patient values (211,'john','Doe','m','11-FEB-1978',NULL, 'Davis
','CA');

1 row created.

SQL> INSERT INTO patient values (219,'Billy','smith','f','19-FEB-1998',2000, 'Sa
cramento','CA');

1 row created.

SQL> SELECT fname, salary FROM patient;

FNAME                    SALARY
-------------------- ----------
john
Billy                      2000

SQL> --Every salary that is null is replaced with zero
SQL> SELECT fname, NVL (salary,0)  FROM patient;

FNAME                NVL(SALARY,0)
-------------------- -------------
john                             0
Billy                         2000

SQL> --Every salary that is null is replaced by the text (poor). Notice that to_
char is used
SQL> --because both arguments have to be the same type (text)
SQL> SELECT fname, NVL(TO_CHAR(salary),'poor') "SALary"  FROM patient;

FNAME                SALary
-------------------- ----------------------------------------
john                 poor
Billy                2000
```

Example 5.16a (NVL2)

The NVL2 function is a variation of the NVL function with different options based on whether a NULL value exists. The syntax of the NVL2 function is NVL2(x, y, z), where **y** represents what should be substituted if x isn't NULL, and **z** represents what should be substituted if x is NULL. This variation gives you a little more flexibility when working with NULL values.

```
--NVL2 provides a value if it is not NULL and also provides a value if it is NULL. In this case,
--poor  if there is data and rich if it is NULL.
SELECT fname, salary, NVL2(TO_CHAR(salary),'poor','rich')  FROM
patient;
```

```
SQL> SELECT fname, salary, NVL2(TO_CHAR(salary),'poor','rich')  FROM patient;

FNAME                SALARY NVL2
-------------------- ---------- ----
john                        rich
Billy                     2000 poor
```

5.16 DECODE

The DECODE function takes a specified value and compares it to values in a list. If a match is found, the specified result is returned. If no match is found, a default result is returned. If no default result is defined, a NULL is returned. The syntax of the DECODE function is DECODE(V, L1, R1, L2, R2, ..., D), where V is the value you're searching for, L1 represents the first value in the list, R1 represents theresult to return if L1 and V are equivalent, and so on, and D is the default result to return if no match is found.

```
INSERT INTO patient values (311,'Jakey','Doey','U','11-FEB-1978',NULL,
'Davis','CA');
--Notice there are an odd number of arguments.
--If gender='m' then
--      display Male
--else if gender='f' then
--      display Female
SELECT gender, DECODE (gender,'m','MALE','f','FEMALE')  FROM patient;

--Notice there are an even number of arguments. The last argument pertains to the else condition.
--If gender='m' then
--      display Male
--else if gender='f' then
--      display Female
-- else
```

```
--      display unknown
SELECT gender, DECODE (gender,'m','MALE','f','FEMALE','UNKNOWN')  FROM
patient;
```

```
SQL> INSERT INTO patient values (311,'Jakey','Doey','U','11-FEB-1978',NULL, 'D
is','CA');

1 row created.

SQL> --Notice there are an odd number of arguments
SQL> --If gender='m' then
SQL> --        display Male
SQL> --else if gender='f' then
SQL> --        display Female
SQL> SELECT gender, DECODE (gender,'m','MALE','f','FEMALE')  FROM patient;

G DECODE
- ------
m MALE
f FEMALE
U

SQL>
SQL> --Notice there are an even number of arguments. The last argument pertain
to the else condition
SQL> --If gender='m' then
SQL> --        display Male
SQL> --else if gender='f' then
SQL> --        display Female
SQL> -- else
SQL> --        display unknown
SQL> SELECT gender, DECODE (gender,'m','MALE','f','FEMALE','UNKNOWN')  FROM pa
ent;

G DECODE(
- -------
m MALE
f FEMALE
U UNKNOWN
```

5.17 SIGN

The syntax for the sign function is: sign (number)

number is the number to test for its sign.

If number < 0, then sign returns -1.
If number = 0, then sign returns 0.
If number > 0, then sign returns 1.

```
DELETE FROM patient;
INSERT INTO patient values (111,'john','Doe','m','11-FEB-1978',25000,
'Davis','CA');
INSERT INTO patient values (112,'john','Smith','m','01-MAR-
1981',40000, 'Davis','CA');
INSERT INTO patient values (113,'jill','Crane','m','12-APR-
1999',500000,'Reno','NV');

SELECT salary, DECODE(SIGN(salary-40000),0,'Good money',-1,'Need
more','Donate') FROM patient;
```

```
SELECT salary, DECODE(SIGN(salary-40000),0,'Good money',-1,'Need
more',1,'Donate') FROM patient;
```

```
SQL> DELETE FROM patient;

3 rows deleted.

SQL> INSERT INTO patient values (111,'john','Doe','m','11-FEB-1978',25000, 'Da
s','CA');

1 row created.

SQL> INSERT INTO patient values (112,'john','Smith','m','01-MAR-1981',40000, '
vis','CA');

1 row created.

SQL> INSERT INTO patient values (113,'jill','Crane','m','12-APR-1999',500000,'
no','NV');

1 row created.

SQL> SELECT salary, DECODE(SIGN(salary-40000),0,'Good money',-1,'Need more','D
ate') FROM patient;

    SALARY DECODE(SIG
---------- ----------
     25000 Need more
     40000 Good money
    500000 Donate

SQL> SELECT salary, DECODE(SIGN(salary-40000),0,'Good money',-1,'Need more',1,
onate') FROM patient;

    SALARY DECODE(SIG
---------- ----------
     25000 Need more
     40000 Good money
    500000 Donate
```

5.18 CASE

CASE expression syntax is similar to an IF-THEN-ELSE statement. Oracle checks each condition starting from the first condition (left to right). When a particular condition is satisfied (WHEN part) the expression returns the tagged value (THEN part). If none of the conditions are matched, the value mentioned in the ELSE part is returned. The ELSE part of the expression is not mandatory. CASE expression will return NULL if nothing is satisfied.

case when <condition> then <value>
    when<condition> then <value>
    ...
    else<value>
end

```
SELECT  fname, salary,
        CASE  WHEN salary<40000 THEN 'Need More'
```

```
               WHEN salary=40000 THEN 'Okay'
               ELSE  'Donate'
         END
FROM patient;
```

```
SQL> SELECT  fname, salary, CASE WHEN salary<40000 THEN 'Need More'
  2                                                 WHEN salary=40000 THEN '
Okay'
  3                                                 ELSE  'Donate'
  4                                           END
  5  FROM patient;

FNAME                SALARY CASEWHENS
-------------------- ---------- ---------
john                  25000 Need More
john                  40000 Okay
jill                 500000 Donate
....
```

5.19 TO_NUMBER

The TO_ NUMBER function converts a value to a numeric data type, if possible. For example, the string value 2009 stored in a date or character string could be converted to a numeric data type to use in calculations. If the string being converted contains non-numeric characters, the function returns an error. It just so happens that in the following cases, it does not matter if a conversion is made because ORACLE is smart enough to make the conversion by itself.

```
SELECT '2009' * 2 FROM DUAL;
SELECT TO_CHAR(2009) * 2 FROM DUAL;
SELECT TO_NUMBER('2009') * 2 FROM DUAL;
```

```
SQL> SELECT '2009' * 2 FROM DUAL;

  '2009'*2
----------
      4018

SQL> SELECT TO_CHAR(2009) * 2 FROM DUAL;

TO_CHAR(2009)*2
---------------
           4018

SQL> SELECT TO_NUMBER('2009') * 2 FROM DUAL;

TO_NUMBER('2009')*2
-------------------
               4018
```