C.P.PATEL & F.H.SHAH COMMERCE COLLEGE
(MANAGED BY SARDAR PATEL EDUCATION TRUST)
BCA, BBA (ITM) & PGDCA PROGRAMME:

BBA (ITM) Semester V - Paper Code: UM05CBBI09

UNIT 3- Data Constraints and Built-in Functions

Sr. No.	Topics
1	Data constraints – Introduction, Type of data constraints (Not Null, Unique, Primary Key, Foreign Key and Check);
2	ALTER TABLE to add/remove constraints;
3	Scalar Functions: Numeric (Abs, Floor, Mod, Power, Round, Sign, Sqrt, Trunc), Character (Chr, Ascii, Concat, Initcap, Lower, Substr, Trim, Upper),
4	Date (Add_Months, Last_Day, Next_Day, Months_Between),
5	Conversion (To_Number, To_Char And To_Date);
6	Aggregate Functions: (Avg, Count, Max, Min, Sum), Miscellaneous: (Nvl, Decode).

- **Constraints:-**

  Besides the cell name, cell length and cell data type, there are other parameters i.e. other data constraints that can be passed to the DBA at cell creation time.

  These data constraints will be connected to a cell by the DBA as flags. Whenever a user attempts to load a cell with data, the DBA will check the data being loaded into the cell against the data constraints defined at the time the cell was created. If data being loaded fails any of the data constraints checks by the DBA, the DBA will not load the data into the cell, rejects the entered record, and will flash an error message to the user.

  The constraints can be either placed at the column level or at the table level.

  COLUMN LEVEL CONSTRAINTS

  If the constraints are defined along with the column definition, it is called column level constraint. Column level constraint can be applied to any one of the column at a time i.e. they are local to a specific column. If the constraint spans across multiple columns, the user will have to use table level constraints.

  TABLE LEVEL CONSTRAINTS

  If the data constraint attached to a specific cell in a table references the contents of other cells in the table then the user will have to use table level constraints. Table level constraints are stored as a part of the table definition.

- **NULL VALUE CONCEPTS:**

  While creating tables, if a row lacks a data value for a particular column, that value is said to be *null*. Columns of any data types may contain null values unless the column was defined *not null* when table was created.

Principles of NULL values:

- Setting a null value is appropriate when the actual value is unknown, or when a value would not be meaningful.
- A null value is not equivalent to a value of zero.
- A null value will evaluate to null in any expression. E.g. null multiply by 10 is null.
- When a column name is defined as not null, then that column becomes a mandatory column. It implies that the user is forced to enter data into the column.

Example:

```
CREATE TABLE client_mst

(client_no VARCHAR2(6) NOT NULL,

name VARCHAR2(15) NOT NULL,

address1 VARCHAR2(20));
```

- **UNIQUE KEY CONCEPTS:**

A unique key is similar to a primary key, except that the purpose of unique key is to ensure that information in the column for each row is unique. A table may have many unique keys.

```
CREATE TABLE dept

(deptno  NUMBER(2) UNIQUE,

dname   VARCHAR2(9), loc VARCHAR2(10) );
```

OR

```
CREATE TABLE dept

(deptno  NUMBER(2) CONSTRAINT pk_dept UNIQUE,

dname   VARCHAR2(9), loc VARCHAR2(10) )
```

OR

```
CREATE TABLE dept

(deptno  NUMBER(2), dname   VARCHAR2(9),

loc VARCHAR2(10),

CONSTRAINT pk_dept UNIQUE(deptno) )
```

- **CHECK CONSTRAINTS:**

The CHECK constraint explicitly defines a condition. To satisfy the constraint, each row in the table must make the condition either TRUE or unknown (due to a null). The condition of a CHECK constraint can refer to any column in the table, but it cannot refer to columns of other tables.

Whenever Oracle evaluates a CHECK constraint condition for a particular row, any column names in the condition refer to the column values in that row.

Restrictions on CHECK Constraints:

- The condition must be a BOOLEAN expression that can be evaluated using the values in the row being inserted or updated.
- The condition cannot contain sub queries or sequences.
- The condition cannot include the SYSDATE, UID, USER or USERENV SQL functions.

Example:

CREATE TABLE dept
(deptno NUMBER(6) CONSTRAINT check_deptno CHECK (deptno BETWEEN 10 AND 99), dname VARCHAR2(9) CONSTRAINT check_dname CHECK (dname = UPPER(dname)),
loc VARCHAR2(10) CONSTRAINT check_loc CHECK (upper(loc) IN ('DALLAS','BOSTON','NEW YORK','CHICAGO'))

CHECK_DEPTNO	Ensures that no department numbers are less than 10 or greater than 99.
CHECK_DNAME	Ensures that all department names are in uppercase.
CHECK_LOC	Restricts department locations to Dallas, Boston, New York, or Chicago in any case.

CREATE TABLE client_mst

(client_no varchar2(6) CONSTRAINTS ck_clientno CHECK (client_no LIKE 'C%'), client_name varchar2(12))

CK_CLIENTNO	Ensures that client numbers must have first character as character 'C' in it.

- **PRIMARY KEY CONCEPTS:**

A primary key is one or more column(s) in a table that is used to identify each row in the table.

A multiple column primary key is called a *composite primary key.*

- No primary key value can appear in more than one row in the table.
- No column that is part of the primary key can contain a null.

CREATE TABLE dept

(deptno NUMBER(2) CONSTRAINTS pk_dept PRIMARY KEY,

dname VARCHAR2(9), loc VARCHAR2(10))

OR

CREATE TABLE dept

(deptno NUMBER(2) PRIMARY KEY,

dname VARCHAR2(9), loc VARCHAR2(10))

OR

CREATE TABLE dept

(deptno NUMBER(2), dname VARCHAR2(9),

loc VARCHAR2(10) CONSTRAINTS pk_dept PRIMARY KEY (deptno))

Defining composite primary key:

CREATE TABLE stud

(sno NUMBER(2), sname VARCHAR2(9), class VARCHAR2(10) CONSTRAINTS pk_dept PRIMARY KEY (sno, sname))

OR

CREATE TABLE stud

(sno NUMBER(2), sname VARCHAR2(9), class VARCHAR2(10) PRIMARY KEY (sno,sname))

- **FOREIGN KEY CONCEPTS:**

  Foreign key represents relationship between tables. A foreign key is a column (or a group of columns) whose values are derived from the primary key of the same or some other table.

  The existence of a foreign key implies that the table with the foreign key is related to the primary key table from which the foreign key is derived. A foreign key must have a corresponding primary key value in the primary key table.

The foreign key/references constraints:

- Rejects and INSERT or UPDATE of a value, if a corresponding value does not exist in the primary key table;
- Rejects a DELETE, if it would invalidate a REFERENCES constraints
- Must reference a PRIMARY KEY or UNIQUE column(s) in primary key table;
- Must reference the PRIMARY KEY of the primary key table if no column or group of columns is specified in the constraint;
- Must reference a table and not a view or cluster
- Requires that you own the primary key table, have the column-level REFERENCE privilege on the referenced column in the primary key table
- Doesn't restrict how other constraints ma reference the same table
- Requires that the FOREIGN KEY column(s) and the CONSTRAINT column(s) have matching data types
- May references the same table named in the CREATE TABLE statement.
- Must not reference the same column more than once

  Example:

Consider the *stud* as a primary key table:

```
CREATE TABLE stud
(s_no NUMBER(3) PRIMARY KEY,
s_name   VARCHAR2(9), address VARCHAR2(15))
```

The foreign key table:

```
CREATE TABLE mark
(s_no NUMBER(3) REFERENCES stud(s_no),
mark1 NUMBER(3), mark2 NUMBER(3))
OR
CREATE TABLE mark
(s_no NUMBER(3),
mark1 NUMBER(3), mark2 NUMBER(3),
FOREIGN KEY (s_no) REFERENCES stud(s_no))
OR
CREATE TABLE mark
(s_no NUMBER(3),
mark1 NUMBER(3), mark2 NUMBER(3))
```

After creating table add a foreign key constraints

```
ALTER TABLE mark ADD CONSTRAINTS fk FOREIGN KEY (s_no)
REFERENCES stud(s_no)
```

- **DEFAULT VALUE CONCEPTS:**

At the time of cell creation a 'default value' can be assigned to it. When the user is loading a 'record' with values and leaves this cell empty, the DBA will automatically load this cell with the default value specified. The data type of default value should be match with the type of the column.

Example:

```
CREATE TABLE stud
(sno number(3), name varchar2(12), gender varchar2(1) DEFAULT 'M')
```

Whenever user will not enter the data in the field *gender* the DBA will automatically takes the value '*M'* for that field.

➢ **ALTER TABLE TO ADD/REMOVE CONSTRAINTS:**

❖ *To add a primary key constraint after creation of a table:*
ALTER TABLE <tablename> ADD PRIMARY KEY (column(s))

It will add a primary key constraint to a specified column(s) specified after PRIMARY KEY. If there are more then one column to specify then all the columns are separated by commas and will create composite primary key.

❖ *To remove primary key constraint:*
ALTER TABLE <tablename> DROP PRIMARY KEY;

Will remove primary key constraint from a table.

❖ *To add a constraint after creation of a table:*
ALTER TABLE <tablename> ADD CONSTRAINTS <cons_name> CHECK <condition>

Will add constraint to a table column level or table level depends on the condition user specifies after CHECK. If check condition with one column will create a column level constraint and for more than one column will create table level constraint

❖ *To remove user defined constraint:*
ALTER TABLE <tablename> DROP CONSTRAINTS <const_name>

Will remove the constraint, which was defined on the table.

⇒ **GROUP FUNCTIONS:**

▪ **AVG**
Syntax: AVG ([DISTINCT|ALL] n)
Purpose: Returns average value of *n*.
Example
SELECT AVG (sal) "Average" FROM emp
Average

2077.21429

▪ **MIN**
Syntax: MIN([DISTINCT|ALL] expr)
Purpose: Returns minimum value of *expr*.
Example
SELECT MIN(hiredate) "Minimum Date" FROM emp
MinimumDate

17-DEC-80

▪ **MAX**
Syntax: MAX([DISTINCT|ALL] expr)
Purpose: Returns maximum value of *expr*.
Example
SELECT MAX(sal) "Maximum Salary" FROM emp
MaximumSalary

5000

- **COUNT**

  Syntax: COUNT ({* | [DISTINCT|ALL] expr})

  Purpose: Returns the number of rows in the query.

  If you specify the *expr* this function return rows where *expr* is NOT NULL. You can count either all rows or distinct value of *expr*.

  If you specify the asterisk (*), this function returns all rows, including duplicates and nulls.

Examples	Output
SELECT COUNT(*) "Total" FROM emp	18
SELECT COUNT(job) "Count" FROM emp	14
SELECT COUNT(DISTINCT job) "Jobs" FROM emp	5

- **SUM**

  Syntax: SUM([DISTINCT|ALL] n)

  Purpose: Returns sum of values of *n*.

  Example: SELECT SUM(sal) "Total" FROM emp

```
Total
----------
29081
```

⇒ **NUMERIC FUNCTIONS:**

- **ABS**

  Syntax: ABS(n)

  Purpose: Returns the absolute value of *n*

  Example:

```
SELECT ABS(-15) "Absolute" FROM DUAL
Absolute
----------
      15
```

- **POWER**

  Syntax: POWER(m, n)

  Purpose: Returns *m* raised to the *n*th power. The base *m* and the exponent *n* can be any numbers, but if *m* is negative, *n* must be an integer.

  Example:

```
SELECT POWER(3,2) "Raised" FROM DUAL
 Raised
----------
      9
```

- **ROUND**

  Syntax: ROUND(n[,m])

  Purpose: Returns n rounded to m places right of the decimal point; if m is omitted, to 0 places. m can be negative to round off digits left of the decimal point. m must be an integer.

  Example: SELECT ROUND(15.193,1) "Round" FROM DUAL

```
Round
----------
 15.2
```

  SELECT ROUND(15.193,-1) "Round" FROM DUAL

```
 Round
----------
   20
```

- **TRUNC**

  Syntax: TRUNC(n[,m])

  Purpose: Returns n truncated to m decimal places; if m is omitted, to 0 places. m can be negative to truncate (make zero) m digits left of the decimal point.

  Example:

```
SELECT TRUNC(15.79,1) "Truncate" FROM DUAL

  Truncate
 ----------
   15.7
SELECT TRUNC(15.79,-1) "Truncate" FROM DUAL
  Truncate
 ----------
    10
```

- **SQRT**

  Syntax: SQRT(n)

  Purpose: Returns square root of n. The value n cannot be negative. SQRT returns a "real" result.

  Example:

```
SELECT SQRT(26) "Square root" FROM DUAL
Square root
-----------
5.09901951
```

- **CEIL**

  Syntax: CEIL(n)

  Purpose: Returns smallest integer greater than or equal to n.

  Example:

```
SELECT CEIL(15.1) "Ceiling" FROM DUAL
    Ceiling
 ----------
      16
SELECT CEIL(-15.1) "Ceiling" FROM DUAL
    Ceiling
 ----------
     -15
```

- **FLOOR**

  Syntax: FLOOR(n)

  Purpose: Returns largest integer equal to or less than n.

  Example:

```
SELECT FLOOR(15.7) "Floor" FROM DUAL
```

Floor

15

- **EXP**

Syntax: EXP(n)
Purpose: Returns e raised to the nth power; e = 2.71828183 ...
Example:
SELECT EXP(4) "e to the 4th power" FROM DUAL
e to the 4th power

54.59815

- **LN**

Syntax: LN(n)
Purpose: Returns the natural logarithm of n, where n is greater than 0.
Example:
SELECT LN(95) "Natural log of 95" FROM DUAL
Natural log of 95

4.55387689

- **LOG**

Syntax: LOG(m,n)

Purpose: Returns the logarithm, base m, of n. The base m can be any positive number other than 0 or 1 and n can be any positive number.

Example:

SELECT LOG(10,100) "Log base 10 of 100" FROM DUAL

Log base 10 of 100

2

- **MOD**

Syntax: MOD(m,n)
Purpose: Returns remainder of m divided by n. Returns m if n is 0.
Example:
SELECT MOD(11,4) "Modulus" FROM DUAL
Modulus

3

- **GREATEST**

Syntax: GREATEST(expr [,expr] ...)

Purpose: Returns the greatest of the list of *expr*s. All *expr*s after the first are implicitly converted to the datatype of the first *exprs* before the comparison. Oracle compares the *expr*s using non-padded comparison semantics. Character comparison is based on the value of the character in the database character set.

One character is greater than another if it has a higher value. If the value returned by this function is character data, its datatype is always VARCHAR2.

Example:

SELECT GREATEST('HARRY','HARRIOT','HAROLD') "GREATEST" FROM DUAL;

GREATEST

HARRY

- **LEAST**

Syntax: LEAST(expr [,expr] ...)

Purpose: Returns the least of the list of *expr*s. All *expr*s after the first are implicitly converted to the datatype of the first *expr* before the comparison. Oracle compares the *expr*s using non-padded comparison semantics. If the value returned by this function is character data, its datatype is always VARCHAR2.

Example:

SELECT LEAST('HARRY','HARRIOT','HAROLD') "LEAST" FROM DUAL;

LEAST

HAROLD

- **SIGN**

Syntax: SIGN(m)
Purpose: If *n*<0, the function returns -1; if *n*=0, the function returns 0; if *n*>0, the function returns 1.

```
SELECT SIGN(-15) "Sign" FROM DUAL
    Sign
--------
      -1
SELECT SIGN(15) "Sign" FROM DUAL
    Sign
--------
      -1
SELECT SIGN(0) "Sign" FROM DUAL
    Sign
--------
       0
```

⇒ **CHARACTER FUNCTIONS:**

- **UPPER**

Syntax: UPPER(char)

Purpose: Returns *char*, with all letters uppercase. The return value has the same datatype as the argument *char*.

Example:
SELECT UPPER('Large') "Uppercase" FROM DUAL
Uppercase

LARGE

- **LOWER**

Syntax: LOWER(char)
Purpose: Returns *char*, with all letters lowercase. The return value has the same datatype as the argument *char* (CHAR or VARCHAR2).
Example:
SELECT LOWER ('SCOTT') "Lowercase" FROM DUAL
Lowercase
---
scott

- **INITCAP**

Syntax: INITCAP(char)
Purpose: Returns *char*, with the first letter of each word in uppercase, all other letters in lowercase. Words are delimited by white space.
Example:
SELECT INITCAP ('the soap') "Capitals" FROM DUAL;
Capitals

The Soap

- **LENGTH**

Syntax: LENGTH(char)
Purpose: Returns the length of *char* in characters. If *char* has datatype CHAR, the length includes all trailing blanks. If *char* is null, this function returns null.
Example
SELECT LENGTH('SEMCOM') "Length in characters" FROM DUAL
Length in characters

6

- **SUBSTR**

Syntax: SUBSTR(char, m [,n])

Purpose: Returns a portion of *char*, beginning at character *m*, *n* characters long. If *m* is 0, it is treated as 1. If *m* is positive, Oracle counts from the beginning of *char* to find the first character. If *m* is negative, Oracle counts backwards from the end of *char*. If *n* is omitted, Oracle returns all characters to the end of *char*. If n is less than 1, a null is returned.

Floating point numbers passed as arguments to substr are automatically converted to integers.

SELECT SUBSTR ('ABCDEFG',3.1,4) "Subs" FROM DUAL
Subs

CDEF

SELECT SUBSTR ('ABCDEFG',-5,4) "Subs" FROM DUAL

Subs

CDEF

- **LPAD**

Syntax: LPAD (char1,n [,char2])

Purpose: Returns *char1*, left-padded to length *n* with the sequence of characters in *char2*; *char2* defaults to a single blank space. If length of *char1* is longer than *n*, this function returns the portion of *char1* that fits in *n*.

The argument *n* is the total length of the return value as it is displayed on your terminal screen. In most character sets, this is also the number of characters in the return value. However, in some multi-byte character sets, the display length of a character string can differ from the number of characters in the string.

Example:

SELECT LPAD ('Page 1',15,'*.') "LPAD example" FROM DUAL

LPAD example

..*.*.*Page 1

- **RPAD**

Syntax: RPAD(char1, n [,char2])

Purpose: Returns *char1*, right-padded to length *n* with *char2*, replicated as many times as necessary; *char2* defaults to a single blank space. If length of *char1* is longer than *n*, this function returns the portion of *char1* that fits in *n*.

The argument *n* is the total length of the return value as it is displayed on your terminal screen. In most character sets, this is also the number of characters in the return value. However, in some multi-byte character sets, the display length of a character string can differ from the number of characters in the string.

Example:

SELECT LPAD ('Page 1',15,'*.') "LPAD example" FROM DUAL

LPAD example

Page 1*.*.*.*.*

- **LTRIM**

Syntax: LTRIM(char1 [,set])

Purpose: Removes characters from the left of *char*, with all the leftmost characters that appear in *set* removed; *set* defaults to a single blank space. Oracle begins scanning *char* from its first character and removes all characters that appear in set until reaching a character not in set and then returns the result string.

Example:

SELECT LTRIM('xyxXxyLAST WORD','xy') "LTRIM example" FROM DUAL

LTRIM example

Xxy LAST WORD

- **RTRIM**
  Syntax: RTRIM(char [,set])
  Purpose: Returns *char*, with all the rightmost characters that appear in *set* removed; *set* defaults to a single blank space. RTRIM works similarly to LTRIM.
  Example:
  SELECT RTRIM('BROWNINGyxXxy','xy') "RTRIM e.g." FROM DUAL
  RTRIM e.g

  BROWNINGyxX
- **CHR**
  Syntax: CHR(n [USING NCHAR_CS])
  Purpose: Returns the character having the binary equivalent to *n* in either the database character set or the national character set.
  Example:
  SELECT CHR(67)||CHR(65)||CHR(84) "Char" FROM DUAL;
  Cha

  CAT
- **CONCAT**
  Syntax: CONCAT (char1, char2)
  Purpose: Returns *char1* concatenated with *char2*. This function is equivalent to the concatenation operator (||).
  Example:
  SELECT CONCAT('CPPATEL','FHSHAH') "CONCAT" FROM DUAL
  CONCAT

  CPPATELFHSHAH
- **REPLACE**
  Syntax: REPLACE(char,search_string[,replacement_string])
  Purpose: Returns *char* with every occurrence of *search_string* replaced with *replacement_string*. If *replacement_string* is omitted or null, all occurrences of *search_string* are removed. If *search_string* is null, char is returned.
  Example:
  SELECT REPLACE('JACK and JUE','J','BL') "Changes" FROM DUAL Changes
  ---
  BLACK and BLUE

- **ASCII**
  Syntax: ASCII(char)
  Purpose: Returns the decimal representation in the database character set of the first byte of *char*.

Example:

```
SELECT ASCII('Q') FROM DUAL
ASCII('Q')
----------
       81
```

- **INSTR**

  Syntax: INSTR (char1,char2 [,n[,m]])

  Purpose: Searches *char1* beginning with its *n*th character for the *m*th occurrence of *char2* and returns the position of the character in *char1* that is the first character of this occurrence. If *n* is negative, Oracle counts and searches backward from the end of *char1*. The value of *m* must be positive. The default values of both *n* and *m* are 1, meaning Oracle begins searching at the first character of *char1* for the first occurrence of *char2*. The return value is relative to the beginning of *char1*, regardless of the value of *n*, and is expressed in characters. If the search is unsuccessful (if *char2* does not appear *m* times after the *n*th character of *char1*) the return value is 0.

  Examples

```
SELECT INSTR('CORPORATE FLOOR','OR', 3, 2) "Instr" FROM DUAL
Instr
----------
        14
```

- **SOUNDEX**

  Syntax: SOUNDEX(char)

  Purpose: Returns a character string containing the phonetic representation of *char*. This function allows you to compare words that are spelled differently, but sound alike in English.

  Example:

```
SELECT ename FROM emp WHERE SOUNDEX(ename) = SOUNDEX('SMYTHE')
  ENAME
----------
SMITH
```

⇒ **DATE FUNCTIONS:**

- **ADD_MONTHS**

  Syntax: ADD_MONTHS(d,n)

  Purpose: Returns the date *d* plus *n* months. The argument *n* can be any integer. If *d* is the last day of the month or if the resulting month has fewer days than the day component of *d*, then the result is the last day of the resulting month. Otherwise, the result has the same day component as *d.*

  Example:

```
SELECT ADD_MONTHS('24-JUN-04',2) "ADD MONTHS" FROM DUAL;
ADD MONTHS
----------
24-AUG-04
```

- **LAST_DAY**
  Syntax: LAST_DAY(d)
  Purpose: Returns the date of the last day of the month that contains *d*. You might use this function to determine how many days are left in the current month.
  Example:
  SELECT LAST_DAY('10-APR-95) "Last Date" FROM DUAL
  Last Date

  30-APR-95
- **MONTHS_BETWEEN**
  Syntax: MONTHS_BETWEEN(d1, d2)

  Purpose: Returns number of months between dates *d1* and *d2*. If *d1* is later than *d2*, result is positive; if earlier, negative. If *d1* and *d2* are either the same days of the month or both last days of months, the result is always an integer; otherwise Oracle calculates the fractional portion of the result based on a 31-day month and considers the difference in time components of *d1* and *d2*.

- **NEXT_DAY**
  Syntax: NEXT_DAY(d, char)

  Purpose: Returns the date of the first weekday named by *char* that is later than the date *d*. The argument *char* must be a day of the week in your session's date language. The return value has the same hours, minutes, and seconds component as the argument *d*.

  Example: This example returns the date of the next Tuesday after March 15, 1992.

  SELECT NEXT_DAY ('15-MAR-92','TUE') "NEXT DAY" FROM DUAL
  NEXT DAY

  17-MAR-92
  SELECT NEXT_DAY ('15-MAR-92',3) "NEXT DAY" FROM DUAL
  NEXT DAY

  17-MAR-92
  SELECT NEXT_DAY ('15-MAR-92','Tuesday') "NEXT DAY" FROM DUAL
  NEXT DAY

  17-MAR-92

- **Addition and subtraction of dates:**
  SELECT SYSDATE + 1 FROM DUAL;

  Will gives the next date from the current system date. It will add number of days to a system date.

  SELECT SYSDATE - 1 FROM DUAL;

  Will gives the previous date from the current system date. It will subtract number of days to a system date.

⇒ **CONVERSION FUNCTIONS:**

- **TO_NUMBER**

  Syntax: TO_NUMBER(char [,fmt])

  Purpose: Converts *char*, a value of CHAR or VARCHAR2 datatype containing a number in the format specified by the optional format model *fmt*, to a value of NUMBER datatype.

- **TO_CHAR**

  TO_CHAR (<expr>)

  Will convert the expression into its character expression.

  TO_CHAR, *date conversion*

  Syntax: TO_CHAR (d [, fmt])

  Purpose: Converts *d* of DATE datatype to a value of VARCHAR2 datatype in the format specified by the date format *fmt*. If you omit *fmt*, *d* is converted to a VARCHAR2 value in the default date format.

- **TO_DATE**

  Syntax: TO_DATE (char [,fmt])

  Purpose: Converts *char* of CHAR or VARCHAR2 datatype to a value of DATE datatype. The *fmt* is a date format specifying the format of *char*.

⇒ **MISCELLANEOUS FUNCTIONS:**

- **NVL**

  The NVL function can be used to return a value when a null occurs.

  For example, the expression NVL (COMM, 0) returns 0 if COMM is null or the value of COMM if it is not null. Here COMM is the field from emp table and type of that is NUMBER so will be replaced by ZERO.

  If user wants to convert the null value with any user define string value then first the field must been converted into the character type.

  For example, NVL (TO_CHAR (COMM),'Null Value') returns a string 'Null Value' if the COMM is null or the value of COMM if it is not null

- **DECODE**

SELECT ename, empno, DECODE (deptno, 10,'ACCT', 20,'SALES','PURCHASE') FROM EMP;

In above query the employee name, number and department number fields will be displayed but with deptno the DECODE function is used.

In that the deptno is the field name and of the record is having the value 10 then 'ACCT' will be displayed, if 20 then 'SALES' will be displayed and if not 10 or 20 then 'PURCHASE' will be displayed.