9. INTERACTIVE SQL PART - III

COMPUTATIONS DONE ON TABLE DATA

None of the techniques used till now allows display of data from a table after some arithmetic has been done with it.

Computations may include displaying an employee's name and the employee's salary from the Employee_Master table along with the **annual salary** of the employee (**i.e.** Salary*12). The arithmetic (Salary * 12) is an example of table data arithmetic.

Arithmetic and logical operators give a new dimension to SQL sentences.

Arithmetic Operators

Oracle allows arithmetic operators to be used while viewing records from a table or while performing Data Manipulation operations such as Insert, Update and Delete. These are:

+	Addition	*	Multiplication
-	Subtraction	**	Exponentiation
/	Division	( )	Enclosed operation

Example 1:
List the fixed deposits held by the customers and also show what will be the amount payable by the bank if the fixed deposits are cancelled by the **end of the day**.

Synopsis:

Tables:	FD_DTLS
Columns:	FD_NO, TYPE, PERIOD, OPNDT, DUEDT, AMT, INTRATE, DUEAMT
Technique:	**Functions:** ROUND(), **Operators:** *, -, /, **Clauses:** WHERE, **Others:** SYSDATE

Solution:

```
SELECT FD_NO, TYPE, PERIOD  OPNDT, DUEDT, AMT, INTRATE, DUEAMT,
       ROUND(AMT + (AMT * ROUND(SYSDATE – OPNDT)/365 * (INTRATE/100)), 2)
  FROM FD_DTLS WHERE DUEDT > SYSDATE;
```

Output:

```
FD_NO   TYPE    PERIOD OPNDT      DUEDT          AMT   INTRATE   DUEAMT
ROUND(AMT+(AMT*ROUND(SYSDATE-OPNDT)/365*(INTRATE/100)),2)
F6      S          732 19-JUL-03 20-JUL-05     5000         9  5902.47
                                                    5429.04
F7      S          366 27-JUL-03 27-JUL-04     5000         8   5401.1
                                                    5372.6
```

Explanation:
Here, **ROUND(AMT + (AMT * ROUND(SYSDATE – OPNDT)/365 * (INTRATE/100)),2)** is **not** a column in the table **FD_DTLS**. However, the arithmetic specified is done on the contents of the columns **AMT, OPNDT and INTRATE** of the table **FD_DTLS** and displayed in the output of the query.

By default, the Oracle engine will use the column names of the table **FD_DTLS** as column headers when displaying column output on the VDU screen.

9. INTERACTIVE SQL PART - III

COMPUTATIONS DONE ON TABLE DATA

None of the techniques used till now allows display of data from a table after some arithmetic has been done with it.

Computations may include displaying an employee's name and the employee's salary from the Employee_Master table along with the **annual salary** of the employee (i.e. Salary*12). The arithmetic (Salary * 12) is an example of table data arithmetic.

Arithmetic and logical operators give a new dimension to SQL sentences.

Arithmetic Operators

Oracle allows arithmetic operators to be used while viewing records from a table or while performing Data Manipulation operations such as Insert, Update and Delete. These are:

+	Addition	*	Multiplication
-	Subtraction	**	Exponentiation
/	Division	( )	Enclosed operation

Example 1:
List the fixed deposits held by the customers and also show what will be the amount payable by the bank if the fixed deposits are cancelled by the **end of the day**.

Synopsis:

Tables:	FD_DTLS
Columns:	FD_NO, TYPE, PERIOD, OPNDT, DUEDT, AMT, INTRATE, DUEAMT
Technique:	**Functions:** ROUND(), **Operators:** *, -, /, **Clauses:** WHERE, **Others:** SYSDATE

Solution:

```
SELECT FD_NO, TYPE, PERIOD  OPNDT, DUEDT, AMT, INTRATE, DUEAMT,
       ROUND(AMT + (AMT * ROUND(SYSDATE – OPNDT)/365 * (INTRATE/100)), 2)
  FROM FD_DTLS WHERE DUEDT > SYSDATE;
```

Output:

```
FD_NO   TYPE    PERIOD OPNDT      DUEDT          AMT   INTRATE   DUEAMT
ROUND(AMT+(AMT*ROUND(SYSDATE-OPNDT)/365*(INTRATE/100)),2)
F6      S          732 19-JUL-03  20-JUL-05     5000         9  5902.47
                                                    5429.04
F7      S          366 27-JUL-03  27-JUL-04     5000         8   5401.1
                                                    5372.6
```

Explanation:
Here, **ROUND(AMT + (AMT * ROUND(SYSDATE – OPNDT)/365 * (INTRATE/100)),2)** is **not** a column in the table **FD_DTLS**. However, the arithmetic specified is done on the contents of the columns **AMT, OPNDT and INTRATE** of the table **FD_DTLS** and displayed in the output of the query.

By default, the Oracle engine will use the column names of the table **FD_DTLS** as column headers when displaying column output on the VDU screen.

Since there are no columns with the arithmetic expression applied on the table **FD_DTLS**, the Oracle engine performs the required arithmetic and uses the **formula** as the **default** column header when displaying output as seen above.

Renaming Columns Used With Expression Lists

Rename the default output column names with an **alias**, when required.

Syntax:

```
SELECT <ColumnName> <AliasName>, <ColumnName> <AliasName>
    FROM <TableName>;
```

Example 2:
List the fixed deposits held by the customers and also show what will be the amount received if the fixed deposits are cancelled on the same day. Use Alias to rename the calculative column to **Pre-Maturity Amount**.

Synopsis:

Tables:	FD_DTLS
Columns:	FD_NO, TYPE, PERIOD, OPNDT, DUEDT, AMT, INTRATE, DUEAMT
Technique:	**Functions:** ROUND(), **Operators:** *, -, /, **Clauses:** WHERE, **Others:** ALIAS, SYSDATE

Solution:

```
SELECT FD_NO, TYPE, PERIOD, OPNDT, DUEDT, AMT, INTRATE, DUEAMT,
        ROUND(AMT + (AMT * ROUND(SYSDATE - OPNDT)/365 * (INTRATE/100)), 2)
            "Pre-Maturity Amount"
    FROM FD_DTLS WHERE DUEDT > SysDate;
```

Output:

```
FD_NO   TYPE    PERIOD OPNDT      DUEDT          AMT   INTRATE    DUEAMT
  Pre Maturity Amount
F6      S          732 19-JUL-03 20-JUL-05     5000         9   5902.47
                5429.04
F7      S          366 27-JUL-03 27-JUL-04     5000         8    5401.1
                 5372.6
```

Explanation:
Here, **ROUND(AMT + (AMT * ROUND(SYSDATE – OPNDT)/365 * (INTRATE/100)),2)** is renamed to alias "Pre-Maturity Amount".

Logical Operators

Logical operators that can be used in SQL sentences are:

The AND Operator:

The AND operator allows creating an SQL statement based on two or more conditions being met. It can be used in any valid SQL statement such as select, insert, update, or delete. The AND operator requires that each condition must be met for the record to be included in the result set.

The Oracle engine will process all rows in a table and display the result only when **all** of the conditions specified using the **AND** operator are satisfied.

Example 3:
Display all those transactions performed today for amount ranging between 500 and 5000 **both inclusive.**

Synopsis:

Tables:	TRANS_MSTR
Columns:	All Columns
Technique:	**Functions:** TO_CHAR(), **Operators:** AND, **Clauses:** WHERE, **Others:** SYSDATE

Solution:
```
SELECT * FROM TRANS_MSTR WHERE AMT >= 500 AND AMT <= 5000
    AND TO_CHAR(DT, 'DD/MM/YYYY') = TO_CHAR(SYSDATE, 'DD/MM/YYYY');
```

Output:
```
TRANS_NO ACCT_NO DT                  TYPE PARTICULAR       DR CR    AMT  BALANCE
-------- ------- ------------------- ---- ---------------- -- -- ------ --------
T7       CA7     14-MAR-2004 B       Initial Payment  D       2000     2000
```

Explanation:
Here, the AND operator is used to compare the value held in the amount field with a constant. Only those transactions carried out today, that satisfy this comparison, are shown. This is done by comparing transaction dates with the current date after converting them to characters.

The OR Operator:

The OR condition allows creating an SQL statement where records are returned when any one of the conditions are met. It can be used in any valid SQL statement such as select, insert, update, or delete. The OR condition requires that any of the conditions must be met for the record to be included in the result set.

The Oracle engine will process all rows in a table and display the result only when **any of** the conditions specified using the **OR** operator is satisfied.

Example 4:
Display the customers whose belong to Information Technology or are self-employed.

Synopsis:

Tables:	CUST_MSTR, ADDR_DTLS
Columns:	CUST_NO, FNAME, MNAME, LNAME
Technique:	**Operators:** LIKE, AND, OR, **Clauses:** WHERE, **Others:** CONCAT

Solution:
```
SELECT CUST_NO, FNAME || ' ' || MNAME || ' ' || LNAME "Customers"
    FROM CUST_MSTR, ADDR_DTLS
        WHERE CUST_MSTR.CUST_NO = ADDR_DTLS.CODE_NO
            AND (OCCUP = 'Information Technology' OR OCCUP = 'Self Employed')
            AND CUST_NO LIKE 'C%';
```

Output:
```
CUST_NO  Customers
-------- ------------------------
C1       Ivan Nelson Bayross
C10      Namita S. Kanade
C7       Anil Arun Dhone
```

Explanation:
Here, the **OR** operator is used to compare the value held in the **OCCUP** field. If the comparison condition is satisfied then only those customers who belong to Information Technology or are self-employed are shown. The **LIKE** operator is used to avoid display of those rows held in the **CUST_MSTR** table, which identify corporates.

Combining the AND and OR Operator:

The **AND** and **OR** conditions can be combined in a single SQL statement. It can be used in any valid SQL statement such as select, insert, update, or delete.

When combining these conditions, it is important to use brackets so that the database knows what order to evaluate each condition.

The Oracle engine will process all rows in a table and display the result only when **all** of the conditions specified using the **AND** operator are satisfied and when **any of** the conditions specified using the **OR** operator are satisfied.

Example 5:
Display all the customers whose **last name** is **Bayross** and are **less than 25** yrs old **or** all those customers who are **more than 25** but **less than 50** yrs old.

Synopsis:

Tables:	CUST_MSTR, ADDR_DTLS
Columns:	CUST_NO, FNAME, MNAME, LNAME
Technique:	**Operators:** LIKE, AND, OR, **Clauses:** WHERE, **Others:** CONCATENATE

Solution:
```
SELECT CUST_NO, FNAME || ' ' || MNAME || ' ' || LNAME "Customers",
       ROUND((SYSDATE - DOB_INC)/365) "Age" FROM CUST_MSTR
   WHERE (ROUND((SYSDATE - DOB_INC)/365) < 25 AND LNAME='Bayross')
      OR (ROUND((SYSDATE - DOB_INC)/365) > 25
          AND ROUND((SYSDATE - DOB_INC)/365) < 50) AND CUST_NO LIKE 'C%';
```

Output:
```
CUST_NO  Customers                Age
-------------------------------------
C2       Chriselle Ivan Bayross   22
C3       Mamta Arvind Muzumdar    29
C4       Chhaya Sudhakar Bankar   28
C5       Ashwini Dilip Joshi      26
C8       Alex Austin Fernandes    42
C10      Namita S. Kanade         26
6 rows selected.
```

Explanation:
This would return all the records where the value calculated by the arithmetic expression **i.e.** age is less than 25 and the value held in the field **LNAME** is **Bayross**. This will also return those records where the value calculated by the arithmetic expression **i.e.** age is more than 25 but less than 50. The brackets determine what order the **AND / OR** conditions are evaluated in.

The NOT Operator:

The Oracle engine will process all rows in a table and display only those records that **do not** satisfy the condition specified.

Example 6:
List the accounts details of those accounts which are **neither** Singly and **nor** Joint Accounts.

Synopsis:

Tables:	ACCT_MSTR
Columns:	ACCT_NO, TYPE, OPR_MODE, OPNDT, CURBAL, STATUS
Technique:	**Operators:** NOT, OR, **Clauses:** WHERE

Solution:
```
SELECT ACCT_NO, TYPE, OPR_MODE, OPNDT, CURBAL, STATUS
    FROM ACCT_MSTR WHERE NOT (OPR_MODE = 'SI' OR OPR_MODE = 'JO');
```

Output:
```
ACCT_NO  TYPE  OPR_MODE  OPNDT       CURBAL  STATUS
-------  ----  --------  ----------  ------  ------
CA4      CA    AS        05-FEB-03   2000    A
SB6      SB    ES        27-FEB-03   500     A
CA7      CA    AS        14-MAR-03   2000    A
CA10     CA    AS        19-APR-03   2000    A
```

Explanation:
The Oracle engine **will not** display rows from the ACCT_MSTR table where the value of the field **OPR_MODE** is either **SI** (Single) or **JO** (Joint). This means that all those records, which satisfy the condition specified using the NOT operator, will not be shown.

Range Searching

In order to select data that is within a range of values, the **BETWEEN** operator is used. The **BETWEEN** operator allows the selection of rows that contain values within a specified lower and upper limit. The range coded after the word **BETWEEN** is **inclusive**.

The lower value must be coded first. The two values in between the range must be linked with the keyword **AND**. The **BETWEEN** operator can be used with both character and numeric data types. However, the data types cannot be mixed **i.e.** the lower value of a range of values from a character column and the other from a numeric column.

Example 7:
List the transactions performed in months of January to March.

Synopsis:

Tables:	TRANS_MSTR
Columns:	All Columns
Technique:	**Functions:** TO_CHAR(),**Operators:** BETWEEN, **Clauses:** WHERE

Solution:
```
SELECT * FROM TRANS_MSTR WHERE TO_CHAR(DT, 'MM') BETWEEN 01 AND 03;
```

Equivalent to:
```
SELECT * FROM TRANS_MSTR
    WHERE TO_CHAR(DT, 'MM') >= 01 AND TO_CHAR(DT, 'MM') <= 03;
```

Output:

TRANS_NO	ACCT_NO	DT	TYPE	PARTICULAR	DR_CR	AMT	BALANCE
T1	SB1	05-JAN-03	C	Initial Payment	D	500	500
T2	CA2	10-JAN-03	C	Initial Payment	D	2000	2000
T3	SB3	22-JAN-03	C	Initial Payment	D	500	500
T4	CA4	05-FEB-03	B	Initial Payment	D	2000	2000
T5	SB5	15-FEB-03	B	Initial Payment	D	500	500
T6	SB6	27-FEB-03	C	Initial Payment	D	500	500
T7	CA7	14-MAR-03	B	Initial Payment	D	2000	2000
T8	SB8	29-MAR-03	C	Initial Payment	D	500	500

8 rows selected.

Explanation:
The above select will retrieve all those records from the ACCT_MSTR table where the value held in the DT field is between 01 and 03 (both values inclusive). This is done using TO_CHAR() function which extracts the month value from the DT field. This is then compared using the AND operator.

Example 8:
List all the accounts, which have not been accessed in the fourth quarter of the financial year.

Synopsis:

Tables:	TRANS_MSTR
Columns:	ACCT_NO
Technique:	**Functions:** TO_CHAR(),**Operators:** NOT, BETWEEN, **Clauses:** WHERE

Solution:

```
SELECT DISTINCT
FROM TRANS_MSTR
    WHERE TO_CHAR(DT, 'MM') NOT BETWEEN 01 AND 04;
```

Output:

```
ACCT_NO
SB9
```

Explanation:
The above select will retrieve all those records from the ACCT_MSTR table where the value held in the DT field is not between 01 and 04 (both values inclusive). This is done using TO_CHAR() function which extracts the month value from the DT field and then compares them using the not and the between operator.

Pattern Matching

The use of the LIKE predicate

The comparison operators discussed so far have compared one value, exactly to one other value. Such precision may not always be desired or necessary. For this purpose Oracle provides the **LIKE** predicate.

The **LIKE** predicate allows comparison of one string value with another string value, which is not identical. This is achieved by using wildcard characters. Two wildcard characters that are available are:

For character data types:

- % allows to match any string of any length (including zero length)
- _ allows to match on a single character

Example 9:
List the customers whose names begin with the letters 'Ch'.

Synopsis:

Tables:	CUST_MSTR
Columns:	FNAME, LNAME, DOB_INC
Technique:	**Operators:** LIKE, **Clauses:** WHERE, **Others:** ALIAS

Solution:
```
SELECT FNAME, LNAME, DOB_INC "BIRTHDATE", OCCUP FROM CUST_MSTR
    WHERE FNAME LIKE 'Ch%';
```

Output:
```
FNAME      LNAME      Birthday   OCCUP
---------- ---------- ---------- ----------
Chriselle  Bayross    29-OCT-82  Service
Chhaya     Bankar     06-OCT-76  Service
```

Explanation:
In the above example, all those records where the value held in the field FNAME begins with **Ch** are displayed. The **%** indicates that any number of characters can follow the letters **Ch**.

Example 10:
List the customers whose names have the second character as **a** or **s**.

Synopsis:

Tables:	CUST_MSTR
Columns:	FNAME, LNAME, DOB_INC
Technique:	**Operators:** LIKE, **Clauses:** WHERE, **Others:** ALIAS

Solution:
```
SELECT FNAME, LNAME, DOB_INC "Birthday", OCCUP FROM CUST_MSTR
    WHERE FNAME LIKE '_a%' OR FNAME LIKE '_s%';
```

Output:
```
FNAME      LNAME      Birthday   OCCUP
---------- ---------- ---------- ----------
Mamta      Muzumdar   28-AUG-75  Service
Ashwini    Joshi      20-NOV-78  Business
Hansel     Colaco     01-JAN-82  Service
Ashwini    Apte       19-APR-79  Service
Namita     Kanade     10-JUN-78  Self Employed
```

Explanation:
In the above example, all those records where the value held in the field FNAME contains the second character as **a** or **s** are displayed. The **_a** and **_s** indicates that only one character can precede the character a or s. The **%** indicates that any number of characters can follow the letters **Ch**.

Example 11:
List the customers whose names begin with the letters **Iv** and it is a four letter word.

Synopsis:

Tables:	CUST_MSTR
Columns:	FNAME, LNAME, DOB_INC
Technique:	**Operators:** LIKE, **Clauses:** WHERE, **Others:** ALIAS

Solution:
SELECT FNAME, LNAME, DOB_INC "Birthday", OCCUP **FROM** CUST_MSTR
WHERE FNAME **LIKE** 'Iv__'; **(i.e. two underscore characters)**

Output:

```
FNAME  LNAME     Birthday  OCCUP
------ --------- --------- -------------
Ivan   Bayross   25-JUN-52 Self Employed
```

Explanation:
In the above example, all those records where the value held in the field FNAME begins with **Iv** are displayed. The **__ (i.e. two underscore characters)** indicates that only two characters can follow the letters **Iv**. This means the whole word will only be four characters.

The IN and NOT IN predicates:

The arithmetic operator (=) compares a single value to another single value. In case a value needs to be compared to a list of values then the **IN** predicate is used. The IN predicate helps reduce the need to use multiple OR conditions

Example 12:
List the customer details of the customers named Hansel, Mamta, Namita and Aruna.

Synopsis:

Tables:	CUST_MSTR
Columns:	FNAME, LNAME, DOB_INC
Technique:	**Operators:** IN, **Clauses:** WHERE, **Others:** ALIAS

Solution:
SELECT FNAME, LNAME, DOB_INC "birthday", OCCUP **FROM** CUST_MSTR
WHERE FNAME **IN**('Hansel', 'Mamta', 'Namita', 'Aruna');

Output:

```
FNAME  LNAME    Birthday  OCCUP
------ -------- --------- -------------
Mamta  Muzumdar 28-AUG-75 Service
Hansel Colaco   01-JAN-82 Service
Namita Kanade   10-JUN-78 Self Employed
```

Explanation:
The above example, displays all those records where the FNAME field holds any one of the four specified values.

The **NOT IN** predicate is the opposite of the IN predicate. This will select all the rows where values **do not** match the values in the list.

Example 13:
List the customer details of the customers other then Hansel, Mamta, Namita and Aruna.

Synopsis:

Tables:	CUST_MSTR
Columns:	FNAME, LNAME, DOB_INC
Technique:	**Operators:** NOT, IN, **Clauses:** WHERE, **Others:** ALIAS

Solution:
SELECT FNAME, LNAME, DOB_INC "Birthday", OCCUP **FROM** CUST_MSTR
WHERE FNAME **NOT IN**('Hansel', 'Mamta', 'Namita', 'Aruna');

Output:

FNAME	LNAME	Birthday	OCCUP
Ivan	Bayross	25-JUN-52	Self Employed
Chriselle	Bayross	29-OCT-82	Service
Chhaya	Bankar	06-OCT-76	Service
Ashwini	Joshi	20-NOV-78	Business
Anil	Dhone	12-OCT-83	Self Employed
Alex	Fernandes	30-SEP-62	Executive
Ashwini	Apte	19-APR-79	Service

7 rows selected.

Explanation:
In the above example by just changing the predicate to **NOT IN** the Select statement will now retrieve all the rows where the FNAME is **not in** the values specified. In other words, information about customers whose names are **not Hansel, Mamta, Namita, Aruna** will be displayed.

The Oracle Table - DUAL

DUAL is a table owned by SYS. SYS owns the data dictionary, and DUAL is part of the data dictionary. Dual is a small Oracle worktable, which consists of only one row and one column, and contains the value x in that column. Besides arithmetic calculations, it also supports **date** retrieval and it's formatting.

Often a simple calculation needs to be done, for example, 2*2. The only SQL verb to cause an output to be written to a VDU screen is **SELECT**. However, a SELECT must have a table name in its FROM clause, otherwise the SELECT **fails**.

When an arithmetic exercise is to be performed such as 2*2 or 4/2 and so on, there is no table being referenced, only **numeric literals** are being used.

To facilitate such calculations via a SELECT, Oracle provides a **dummy** table called **DUAL**, against which SELECT statements that are required to manipulate numeric literals can be fired, and appropriate output obtained.

The structure of the dual table if viewed is as follows:
DESC DUAL;

Output:

NAME	Null?	TYPE
Dummy		VARCHAR2(1)

If the dual table is queried for records the output is as follows:
SELECT * FROM DUAL;

Output:

D
X

Example 14:
SELECT 2*2 FROM DUAL;

Output:

```
      2*2
---------
        4
```

SYSDATE

SYSDATE is a pseudo column that contains the current date and time. It requires no arguments when selected from the table DUAL and returns the current date.

Example 15:
SELECT SYSDATE FROM DUAL;

Output:

```
SYSDATE
---------
01-JUL-04
```

ORACLE FUNCTIONS

Oracle Functions serve the purpose of manipulating data items and returning a result. Functions are also capable of accepting user-supplied variables or constants and operating on them. Such variables or constants are called **arguments**. *Any number of arguments (***or** no arguments at all) can be passed to a function in the following format:

Function_Name(argument1, argument2,..)

Oracle Functions can be clubbed together depending upon whether they operate on a single row or a group of rows retrieved from a table. Accordingly, functions can be classified as follows:

Group Functions (Aggregate Functions)

Functions that act on a **set of values** *are called* **Group Functions**. *For example,* **SUM**, *is a function, which calculates the total set of numbers. A group function returns a single result row for a group of queried rows.*

Scalar Functions (Single Row Functions)

Functions that act on **only one value** *at a time are called* **Scalar Functions**. *For example,* **LENGTH**, *is a function, which calculates the length of one particular string value. A single row function returns one result for every row of a queried table or view.*

Single row *functions can be further grouped together by the data type of their arguments and return values. For example,* **LENGTH** *relates to the* **String** *Data type. Functions can be classified corresponding to different data types as:*

String Functions **: For String** Data type
Numeric Functions: For **Number** Data type
Conversion Functions: For **Conversion** of one Data type to another.
Date Functions: For **Date** Data type

Aggregate Functions

AVG: *Returns an average value of* **'n'**, **ignoring** null values in a column.

Syntax:

```
AVG ([<DISTINCT>|<ALL>] <n>)
```

Example:
SELECT AVG(CURBAL) "Average Balance" **FROM** ACCT_MSTR;

Output:
```
Average Balance
---------------
           1100
```

Note

In the above SELECT statement, **AVG** function is used to calculate the average balance of all accounts branch wise. The selected column is renamed as **Average Balance** in the output.

MIN: Returns a minimum value of **expr**.

Syntax:
```
MIN([<DISTINCT>|<ALL>] <expr>)
```

Example:
SELECT MIN(CURBAL) "Minimum Balance" **FROM** ACCT_MSTR;

Output:
```
Minimum Balance
---------------
            500
```

COUNT(expr): Returns the number of rows where **expr** is not null.

Syntax:
```
COUNT([<DISTINCT>|<ALL>] <expr>)
```

Example:
SELECT COUNT(ACCT_NO) "No. Of Accounts" **FROM** ACCT_MSTR;

Output:
```
No. Of Accounts
---------------
             10
```

COUNT(*): Returns the number of rows in the table, including duplicates and those with nulls.

Syntax:
```
COUNT(*)
```

Example:
SELECT COUNT(*) "No. Of Records" **FROM** ACCT_MSTR;

Output:
```
No. of Records
--------------
            10
```

MAX: Returns the maximum value of **expr**.

Syntax:
```
MAX([<DISTINCT>|<ALL>] <expr>)
```

Example:
SELECT MAX(CURBAL) "Maximum Balance" **FROM** ACCT_MSTR;

Output:

```
Maximum Balance
---------------
           2000
```

SUM: Returns the sum of the values of **'n'**.

Syntax:

```
SUM([<DISTINCT>|<ALL>] <n>)
```

Example:

```
SELECT SUM(CURBAL) "Total Balance" FROM ACCT_MSTR;
```

Output:

```
Total Balance
-------------
        11000
```

Numeric Functions

ABS: Returns the absolute value of **'n'**.

Syntax:

```
ABS(n)
```

Example:

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

Output:

```
Absolute
--------
      15
```

POWER: Returns **m** raised to the **n**th power. **n** must be an integer, else an error is returned.

Syntax:

```
POWER(m,n)
```

Example:

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

Output:

```
Raised
------
     9
```

ROUND: Returns **n**, rounded to **m** places to the right of a decimal point. If **m** is omitted, **n** is rounded to 0 places. **m** can be negative to round off digits to the left of the decimal point. **m** must be an integer.

Syntax:

```
ROUND(n[,m])
```

Example:

```
SELECT ROUND(15.19,1) "Round" FROM DUAL;
```

Output:

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

SQRT: Returns square root of **n**. If **n<0**, NULL. SQRT returns a **real** result.

Syntax:

```
SQRT(n)
```

Example:

```
SELECT SQRT(25) "Square Root" FROM DUAL;
```

Output:

```
Square Root
-----------
          5
```

EXP: Returns **e** raised to the **nth** power, where **e = 2.71828183**.

Syntax:

```
EXP(n)
```

Example:

```
SELECT EXP(5) "Exponent" FROM DUAL;
```

Output:

```
  Exponent
-----------
148.413159
```

EXTRACT: Returns a value extracted from a date or an interval value. A DATE can be used only to extract YEAR, MONTH, and DAY, while a timestamp with a time zone datatype can be used only to extract TIMEZONE_HOUR and TIMEZONE_MINUTE.

Syntax:

```
EXTRACT( {year | month | day | hour | minute | second | timezone_hour |
          timezone_minute | timezone_region | timezone_abbr}
    FROM { date_value | interval_value } )
```

Example:

```
SELECT EXTRACT(YEAR FROM DATE '2004-07-02') "Year",
   EXTRACT(MONTH FROM SYSDATE) "Month" FROM DUAL;
```

Output:

```
 Year  Month
-----------
 2004      7
```

GREATEST: Returns the greatest value in a list of expressions.

Syntax:

```
GREATEST(expr1, expr2, ... expr_n)
```

where, **expr1**, **expr2**, ... **expr_n** are expressions that are evaluated by the greatest function.

Example:

```
SELECT GREATEST(4, 5, 17) "Num", GREATEST('4', '5', '17') "Text" FROM DUAL;
```

Output:

```
 Num  Text
-----------
  17     5
```

LEAST: Returns the least value in a list of expressions.

Syntax:

```
LEAST(expr1, expr2, ... expr_n)
```

where, **expr1**, **expr2**, ... **expr_n** are expressions that are evaluated by the least function.

Example:

```
SELECT LEAST(4, 5, 17) "Num", LEAST('4', '5', '17') "Text" FROM DUAL;
```

Output:

```
 Num  Text
---------
   4    17
```

Note

In the **GREATEST()** and **LEAST()** function if the datatypes of the expressions are different, all expressions will be converted to whatever is datatype of the first expression in the list. If the comparison is based on a character comparison, one character is considered greater than another if it has a higher character set value.

MOD: Returns the remainder of a first number divided by second number passed a parameter. If the second number is zero, the result is the same as the first number.

Syntax:

```
MOD(m, n)
```

Example:

```
SELECT MOD(15, 7) "Mod1", MOD(15.7, 7) "Mod2" FROM DUAL;
```

Output:

```
 Mod1  Mod2
-----------
    1   1.7
```

TRUNC: Returns a number truncated to a certain number of decimal places. The decimal place value must be an integer. If this parameter is omitted, the TRUNC function will truncate the number to 0 decimal places.

Syntax:

```
TRUNC(number, [decimal_places])
```

Example:

```
SELECT TRUNC(125.815, 1) "Trunc1", TRUNC(125.815, -2) "Trunc2" FROM DUAL;
```

Output:

```
 Trunc1  Trunc2
---------------
  125.8     100
```

FLOOR: Returns the largest integer value that is equal to or less than a number.

Syntax:

```
FLOOR(n)
```

Example:

```
SELECT FLOOR(24.8) "Flr1", FLOOR(13.15) "Flr2" FROM DUAL;
```

Output:

```
  Flr1   Flr2
------ ------
    24     13
```

CEIL: Returns the smallest integer value that is greater than or equal to a number.

Syntax:

```
CEIL(n)
```

Example:

```
SELECT CEIL(24.8) "Ceil1",CEIL(13.15) "Ceil2" FROM DUAL;
```

Output:

```
 Ceil1  Ceil2
------ ------
    25     14
```

Note

Several other Numeric functions are available in Oracle. These include the following:

- ACOS(), ASIN(), ATAN(), ATAN2(),
- COS(), COSH(), SIN(), SINH(), TAN(), TANH(),
- COVAR_POP(), COVAR_SAMP(), VAR_POP(), VAR_SAMP(),
- CORR(), SIGN()

String Functions

LOWER: Returns char, with all letters in lowercase.

Syntax:

```
LOWER(char)
```

Example:

```
SELECT LOWER('IVAN BAYROSS') "Lower" FROM DUAL;
```

Output:

```
Lower
---------------
ivan bayross
```

INITCAP: Returns a string with the first letter of each word in **upper case.**

Syntax:

```
INITCAP(char)
```

Example:

```
SELECT INITCAP('IVAN BAYROSS') "Title Case" FROM DUAL;
```

Output:

```
Title Case
------------------
Ivan Bayross
```

UPPER: Returns char, with all letters forced to uppercase.

Syntax:

```
UPPER (char)
```

Example:
SELECT UPPER('Ms. Carol') "Capitalised" FROM DUAL;

Output:
```
Capitalised
-----------
MS. CAROL
```

SUBSTR: Returns a portion of characters, beginning at character **m**, and going upto character **n**. If **n** is omitted, the result returned is upto the last character in the string. The first position of char is 1.

Syntax:
SUBSTR(<string>, <start_position>, [<length>])

where, **string** is the source string.
start_position is the position for extraction. The first position in the string is always 1.
length is the number of characters to extract.

Example:
SELECT SUBSTR('SECURE',3,4) "Substring" FROM DUAL;

Output:
```
Substring
---------
CURE
```

ASCII: Returns the NUMBER code that represents the specified character. If more than one character is entered, the function will return the value for the first character and ignore all of the characters after the first.

Syntax:
ASCII(<single_character>)

where, **single_character** is the specified character to retrieve the NUMBER code for.

Example:
SELECT ASCII('a') "ASCII1", ASCII('A') "ASCII2" FROM DUAL;

Output:
```
ASCII1  ASCII2
------  ------
    97      65
```

COMPOSE: Returns a Unicode string. It can be a **char, varchar2, nchar, nvarchar2, clob**, or **nclob**.

Syntax:
COMPOSE(<single>)

Below is a listing of **unistring** values that can be combined with other characters in the compose function.

Unistring Value	Resulting character
UNISTR('\0300')	grave accent (`)
UNISTR('\0301')	acute accent (')
UNISTR('\0302')	circumflex (^)
UNISTR('\0303')	tilde (~)
UNISTR('\0308')	umlaut (¨)

Example:
SELECT COMPOSE('a' || UNISTR('\0301')) "Composed" FROM DUAL;

Output:

```
Composed
--------
á
```

DECOMPOSE: Accepts a string and returns a Unicode string.

Syntax:

```
DECOMPOSE(<single>)
```

Example:

```
SELECT DECOMPOSE(COMPOSE('a' || UNISTR('\0301'))) "Decomposed" FROM DUAL;
```

Output:

```
Decomposed
----------
á
```

INSTR: Returns the location of a substring in a string.

Syntax:

```
INSTR(<string1>, <string2>, [<start_position>], [<nth_appearance>])
```

where, **string1** is the string to search.
string2 is the substring to search for in string1.
start_position is the position in **string1** where the search will start. If omitted, it defaults to 1. The first position in the string is 1. If the **start_position** is negative, the function counts back **start_position** number of characters from the end of **string1** and then searches towards the beginning of **string1**.
nth_appearance is the **nth** appearance of **string2**. If omiited, it defaults to 1.

Example:

```
SELECT INSTR('SCT on the net', 't') "Instr1", INSTR('SCT on the net', 't', 1, 2) "Instr2"
  FROM DUAL;
```

Output:

```
 Instr1  Instr2
-------- -------
      8      14
```

TRANSLATE: Replaces a sequence of characters in a string with another set of characters. However, it replaces a single character at a time. For example, it will replace the 1st character in the string_to_replace with the 1st character in the replacement_string. Then it will replace the 2nd character in the string_to_replace with the 2nd character in the replacement_string, and so on.

Syntax:

```
TRANSLATE(<string1>, <string_to_replace>, <replacement_string>)
```

where, **string1** is the string to replace a sequence of characters with another set of characters.
string_to_replace is the string that will be searched for in string1.
All characters in the **string_to_replace** will be replaced with the corresponding character in the **replacement_string**.

Example:

```
SELECT TRANSLATE('1sct523', '123', '7a9') "Change" FROM DUAL;
```

Output:

```
Change
-------
7sct5a9
```

LENGTH: Returns the length of a word.

Syntax:

```
LENGTH(word)
```

Example:
SELECT LENGTH('SHARANAM') "Length" FROM DUAL;

Output:

```
Length
----------
         8
```

LTRIM: Removes characters from the left of char with initial characters removed upto the first character not in set.

Syntax:

```
LTRIM(char[,set])
```

Example:
SELECT LTRIM('NISHA','N') "LTRIM" FROM DUAL;

Output:

```
LTRIM
-----
ISHA
```

RTRIM: Returns char, with final characters removed after the last character not in the set. 'set' is optional, it defaults to spaces.

Syntax:

```
RTRIM (char,[set])
```

Example:
SELECT RTRIM('SUNILA','A') "RTRIM" FROM DUAL;

Output:

```
RTRIM
-----
SUNIL
```

TRIM: Removes all specified characters either from the beginning or the ending of a string.

Syntax:

```
TRIM( [leading | trailing | both [<trim_character> FROM ] ]  <string1> )
```

where, **leading** - remove **trim_string** from the front of **string1**.
trailing - remove **trim_string** from the end of **string1**.
both - remove **trim_string** from the front and end of **string1**.
If none of the above option is chosen, the **TRIM** function will remove **trim_string** from both the front and end of **string1**.
trim_character is the character that will be removed from string1. If this parameter is omitted, the trim function will remove all leading and trailing spaces from string1.
string1 is the string to trim.

Example 1:
SELECT TRIM(' Hansel ') "Trim both sides" FROM DUAL;

Output:

```
Trim both sides
--------------
Hansel
```

Example 2:

SELECT TRIM(LEADING 'x' **FROM** 'xxxHanselxxx') "Remove prefixes" **FROM DUAL;**

Output:

```
Remove prefixes
---------------
Hanselxxx
```

Example 3:

SELECT TRIM(BOTH 'x' **FROM** 'xxxHanselxxx') "Remove prefixes N suffixes" **FROM DUAL;**

Output:

```
Remove prefixes N suffixes
--------------------------
Hansel
```

Example 4:

SELECT TRIM(BOTH '1' **FROM** '123Hansel12111') "Remove string" **FROM DUAL;**

Output:

```
Remove string
-------------
23Hansel112
```

LPAD: Returns **char1**, left-padded to length **n** with the sequence of characters specified in **char2**. If **char2** is not specified Oracle uses blanks by default.

Syntax:

```
LPAD(char1,n [,char2])
```

Example:

SELECT LPAD('Page 1',10,'*') "LPAD" **FROM DUAL;**

Output:

```
LPAD
----------
****Page1
```

RPAD: Returns **char1**, right-padded to length **n** with the characters specified in **char2**. If **char2** is not specified, Oracle uses blanks by default.

Syntax:

```
RPAD(char1,n[,char2])
```

Example:

SELECT RPAD(FNAME,10,'x') "RPAD Example" **FROM** CUST_MSTR
WHERE FNAME = 'Ivan';

Output:

```
RPAD Example
------------
Ivanxxxxxx
```

VSIZE: Returns the number of bytes in the internal representation of an expression.

Syntax:

```
VSIZE(<expression>)
```

Example:
SELECT VSIZE('SCT on the net') "Size" **FROM DUAL;**

Output:
```
Size
----------
   14
```

Conversion Functions

TO_NUMBER: Converts **char**, a **CHARACTER** value expressing a number, to a NUMBER datatype.

Syntax:

```
TO_NUMBER(char)
```

Example:
UPDATE ACCT_MSTR **SET** Curbal = Curbal + **TO_NUMBER(SUBSTR**('$100',2,3));

Output:
```
10 rows updated.
```

Note

Here, the value 100 will be added to every accounts current balance in the Acct_Mstr table.

TO_CHAR (number conversion): Converts a value of a **NUMBER** datatype to a **character** datatype using the optional format string. TO_CHAR() accepts a number (**n**) and a numeric format (**fmt**) in which the number has to appear. If **fmt** is omitted, **n** is converted to a char value exactly long enough to hold all significant digits.

Syntax:

```
TO_CHAR (n[,fmt])
```

Example:
SELECT TO_CHAR(17145, '$099,999') "Char" **FROM DUAL;**

Output:
```
Char
----------
$017,145
```

TO_CHAR (date conversion): Converts a value of a **DATE** datatype to **CHAR** value. TO_CHAR() accepts a date, as well as the format (fmt) in which the date has to appear. **fmt** must be a date format. If **fmt** is omitted, the **date** is converted to a character value using the default date format, i.e. "DD-MON-YY".

Syntax:

```
TO_CHAR(date[,fmt])
```

Example:
SELECT TO_CHAR(DT, 'Month DD, YYYY') "New Date Format" **FROM** Trans_Mstr
WHERE Trans_No = 'T1';

Output:
```
New Date Format
---------------------
January   05, 2003
```

MBER datatype.

);

Mstr table.

aracter datatype
at (fmt) in which
nough to hold all

. TO_CHAR()
date format. If
e. "DD-MON-

DATE CONVERSION FUNCTIONS

The DATE data type is used to store date and time information. The DATE data type has special properties associated with it. It stores information about century, year, month, day, hour, minute and second for each date value.

The value in the column of a DATE data type, **is always** stored in a specific **default** format. This default format is 'DD-MON-YY HH:MI:SS'. Hence, when a date has to be inserted in a date field, its value has to be specified in the same format. Additionally, values of DATE columns are always displayed in the **default** format when **retrieved** from the table.

If data from a date column has to be viewed in any other format other than the default format, Oracle provides the **TO_DATE** function that can be used to specify the required format.

The same function can also be used for storing a date into a DATE field in a particular format (other than default). This can be done by specifying the date value, along with the format in which it is to be inserted. The TO_DATE() function also allows part insertion of a DATE value into a column, for example, only the day and month portion of the date value.

To enter the time portion of a date, the TO_DATE function must be used with a **format mask** indicating the time portion.

TO_DATE: Converts a character field to a date field.

Syntax:

```
TO_DATE(char [, fmt])
```

Example:

```
INSERT INTO CUST_MSTR(CUST_NO, FNAME, MNAME, LNAME, DOB_INC)
    VALUES('C1', 'Ivan', 'Nelson', 'Bayross',
        TO_DATE('25-JUN-1952 10:55 A.M.', 'DD-MON-YY HH:MI A.M.'));
```

Output:

```
1 rows created.
```

DATE FUNCTIONS

To manipulate and extract values from the date column of a table Oracle provides some date functions. These are discussed below:

ADD_MONTHS: Returns date after adding the number of months specified in the function.

Syntax:

```
ADD_MONTHS(d,n)
```

Example:

```
SELECT ADD_MONTHS(SYSDATE, 4) "Add Months" FROM DUAL;
```

Output:

```
Add Months
---------
01-NOV-04
```

LAST_DAY: Returns the last date of the month specified with the function.

Syntax:

```
LAST_DAY(d)
```

Example:

SELECT SYSDATE, LAST_DAY(SYSDATE) "LastDay" FROM DUAL;

Output:

```
SYSDATE   LastDay
--------- ---------
01-JUL-04 31-JUL-04
```

MONTHS_BETWEEN: Returns number of months between **d1** and **d2**.

Syntax:

```
MONTHS_BETWEEN(d1, d2)
```

Example:

SELECT MONTHS_BETWEEN('02-FEB-92', '02-JAN-92') "Months" **FROM DUAL;**

Output:

```
Months
----------
         1
```

NEXT_DAY: Returns the date of the first weekday named by **char** that is after the date named by date. **char** must be a day of the week.

Syntax:

```
NEXT_DAY(date, char)
```

Example:

SELECT NEXT_DAY('06-JULY-02', 'Saturday') "NEXT DAY" **FROM DUAL;**

Output:

```
NEXT DAY
---------
13-July-02
```

ROUND: Returns a date rounded to a specific unit of measure. If the second parameter is omitted, the **ROUND** function will round the date to the nearest day.

Syntax:

```
ROUND(date, [format])
```

Below are the valid format parameters:

Unit	Format parameters	Rounding Rule
Year	SYYYY, YYYY, YEAR, SYEAR, YYY, YY, Y	Rounds up on July 1st
ISO Year	IYYY, IY, I	
Quarter	Q	Rounds up on the 16th day of the second month of the quarter
Month	MONTH, MON, MM, RM	Rounds up on the 16th day of the month
Week	WW	Same day of the week as the first day of the year
IW	IW	Same day of the week as the first day of the ISO year
W	W	Same day of the week as the first day of the month
Day	DDD, DD, J	
Hour	HH, HH12, HH24	

Unit	Format parameters	Rounding Rule
Start day of the week	DAY, DY, D	
Minute	MI	

Example:

```
SELECT ROUND(TO_DATE('01-JUL-04'), 'YYYY') "Year" FROM DUAL;
```

Output:

```
Year
---------
01-JAN-05
```

NEW_TIME: Returns the date after converting it from **time zone1** to a date in **time zone2**.

Syntax:

```
NEW_TIME(date, zone1, zone2)
```

Value	Description	Value	Description
AST	Atlantic Standard Time	ADT	Atlantic Daylight Time
BST	Bering Standard Time	BDT	Bering Daylight Time
CST	Central Standard Time	CDT	Central Daylight Time
EST	Eastern Standard Time	EDT	Eastern Daylight Time
GMT	Greenwich Mean Time	HST	Alaska-Hawaii Standard Time
HDT	Alaska-Hawaii Daylight Time	MST	Mountain Standard Time
MDT	Mountain Daylight Time	NST	Newfoundland Standard Time
PST	Pacific Standard Time	PDT	Pacific Daylight Time
YST	Yukon Standard Time	YDT	Yukon Daylight Time

Example:

The following example converts an Atlantic Standard Time into a Mountain Standard Time:

```
SELECT NEW_TIME(TO_DATE('2004/07/01 01:45', 'yyyy/mm/dd HH24:MI'), 'AST', 'MST') "MST"
    FROM DUAL;
```

Output:

```
MST
---------
30-JUN-04
```

Note

Several other Date function are available in Oracle. These include the following:

- **DbTimeZone(), SessionTimeZone(), SysTimestamp(), Tz_Offset()**

The above Oracle date functions are **just a few** selected from the many date functions that are built into Oracle. These Oracle functions are commonly used in commercial application development.

MANIPULATING DATES IN SQL USING THE DATE()

A column of data type **Date** is always displayed in a default format, which is **'DD-MON-YY'**. If this default format is not used when entering data into a column of the **date** data type, Oracle **rejects the data** and returns an error message.

If a **date** has to be retrieved or inserted into a table in a format **other than** the default one, Oracle provides the **TO_CHAR** and **TO_DATE** functions to do this.

TO_CHAR

The TO_CHAR function facilitates the retrieval of data in a format different from the default format. It can also extract a part of the date, i.e. the date, month, or the year from the date value and use it for sorting or grouping of data according to the date, month, or year.

Syntax:

```
TO_CHAR(<date value> [,<fmt>])
```

where **date value** stands for the date and **fmt** is the specified format in which date is to be displayed.

Example 1:

```
SELECT TO_CHAR(SYSDATE, 'DD-MM-YY') FROM DUAL;
```

Output:

```
TO_CHAR(
--------
01-07-04
```

TO_DATE

TO_DATE converts a **char** value into a **date** value. It allows a user to insert date into a date column in any required format, by specifying the **character** value of the date to be inserted and its format.

Syntax:

```
TO_DATE(<char value>[,<fmt>])
```

where **char value** stands for the value to be inserted in the date column, and **fmt** is a date format in which the 'char value' is specified.

Example 2:

```
SELECT TO_DATE ('06/07/02', 'DD/MM/YY') FROM DUAL;
```

Output:

```
TO_DATE('
---------
06-JUL-02
```

Example 3:

List the transaction details in order of the months for account no. **SB9**. The **Transaction Date** should be displayed in **'DD/MM/YY'** format.

Synopsis:

Tables:	TRANS_MSTR
Columns:	TRANS_NO, ACCT_NO, DT, PARTICULAR, DR_CR, AMT, BALANCE
Technique:	**Functions:** TO_CHAR(), **Clauses:** WHERE, ORDER BY

Solution:

```
SELECT TRANS_NO, ACCT_NO, TO_CHAR(DT, 'DD/MM/YY') "Transaction Date",
       PARTICULAR, DR_CR, AMT, BALANCE
  FROM TRANS_MSTR WHERE ACCT_NO = 'SB9' ORDER BY TO_CHAR(DT, 'MM');
```

Output:

TRANS_NO	ACCT_NO	Transaction Date	PARTICULAR	DR_CR	AMT	BALANCE
T9	SB9	05/04/03	Initial Payment	D	500	500
T10	SB9	15/04/03	CLR-204907	D	3000	3500
T11	SB9	17/04/03	Self	W	2500	1000
T13	SB9	05/06/03	CLR-204908	D	3000	4000

Output: (Continued)

```
TRANS_NO   ACCT_NO   Transaction Date   PARTICULAR      DR_CR  AMT    BALANCE
T14        SB9       27/06/03          Self            W      2500   1500
```

Explanation:
Here the value held in the DT field is formatted using the TO_CHAR() function to display the date in the **DD/MM/YY** format. The ordering of the output data set is based on the "**MONTH**" segment of the data in the column DT. This is done using the **TO_CHAR()** function, in the order by clause, extracting only the "**MONTH**" segment of the DT to sort on.

Example 4:
Insert the following data in the table CUST_MSTR, wherein the time component has to be stored along with the date in the column DOB_INC.

Cust_No	Fname	Lname	Dob_Inc
C100	Sharanam	Shah	03/Jan/1981 12:23:00

INSERT INTO CUST_MSTR (CUST_NO, FNAME, LNAME, DOB_INC)
VALUES('C100', 'Sharanam', 'Shah', **TO_DATE**('03/Jan/1981 12:23:00', 'DD/MON/YY hh:mi:ss'));

Output:

```
1 row created.
```

Special Date Formats Using TO_CHAR function

Sometimes, the date value is required to be displayed in special formats, for example, instead of 03-JAN-81, displays the date as 03rd of January, 1981. For this, Oracle provides **special attributes**, which can be used in the format specified with the **TO_CHAR** and **TO_DATE** functions. The significance and use of these characters are explained in the examples below.

All three examples below are based on the CUST_MSTR table

The query is as follows:

SELECT CUST_NO, FNAME, LNAME, DOB_INC
FROM CUST_MSTR **WHERE** CUST_NO **LIKE** 'C_';

Output:

```
CUST_NO  FNAME      LNAME      DOB_INC
C1       Ivan       Bayross    25-JUN-52
C2       Chriselle  Bayross    29-OCT-82
C3       Mamta      Muzumdar   28-AUG-75
C4       Chhaya     Bankar     06-OCT-76
C5       Ashwini    Joshi      20-NOV-78
C6       Hansel     Colaco     01-JAN-82
C7       Anil       Dhone      12-OCT-83
C8       Alex       Fernandes  30-SEP-62
C9       Ashwini    Apte       19-APR-79
9 rows selected.
```

Variations in this output can be achieved as follows:

Use of TH in the TO_CHAR() function:

DDTH places TH, RD, ND for the date (DD), for example, 2ND, 3RD, 08TH etc

SELECT CUST_NO, FNAME, LNAME, **TO_CHAR**(DOB_INC, 'DDTH-MON-YY') "DOB_DDTH"
FROM CUST_MSTR **WHERE** CUST_NO **LIKE** 'C_';

Output:

CUST_NO	FNAME	LNAME	DOB_DDTH
C1	Ivan	Bayross	25TH-JUN-52
C2	Chriselle	Bayross	29TH-OCT-82
C3	Mamta	Muzumdar	28TH-AUG-75
C4	Chhaya	Bankar	06TH-OCT-76
C5	Ashwini	Joshi	20TH-NOV-78
C6	Hansel	Colaco	01ST-JAN-82
C7	Anil	Dhone	12TH-OCT-83
C8	Alex	Fernandes	30TH-SEP-62
C9	Ashwini	Apte	19TH-APR-79

9 rows selected.

Use of SP in the TO_CHAR() function

DDSP indicates that the date (DD) must be displayed by spelling the date such as ONE, TWELVE etc.

```
SELECT CUST_NO, FNAME, LNAME, TO_CHAR(DOB_INC, 'DDSP') "DOB_DDSP"
    FROM CUST_MSTR WHERE CUST_NO LIKE 'C_';
```

Output:

CUST_NO	FNAME	LNAME	DOB_DDSP
C1	Ivan	Bayross	TWENTY-FIVE
C2	Chriselle	Bayross	TWENTY-NINE
C3	Mamta	Muzumdar	TWENTY-EIGHT
C4	Chhaya	Bankar	SIX
C5	Ashwini	Joshi	TWENTY
C6	Hansel	Colaco	ONE
C7	Anil	Dhone	TWELVE
C8	Alex	Fernandes	THIRTY
C9	Ashwini	Apte	NINETEEN

9 rows selected.

Use of 'SPTH' in the to_char function

SPTH displays the date (DD) with **th** added to the spelling fourteen**th**, twel**fth**.

```
SELECT CUST_NO, FNAME, LNAME, TO_CHAR(DOB_INC, 'DDSPTH') "DOB_DDSPTH"
    FROM CUST_MSTR WHERE CUST_NO LIKE 'C_';
```

Output:

CUST_NO	FNAME	LNAME	DOB_DDSPTH
C1	Ivan	Bayross	TWENTY-FIFTH
C2	Chriselle	Bayross	TWENTY-NINTH
C3	Mamta	Muzumdar	TWENTY-EIGHTH
C4	Chhaya	Bankar	SIXTH
C5	Ashwini	Joshi	TWENTIETH
C6	Hansel	Colaco	FIRST
C7	Anil	Dhone	TWELFTH
C8	Alex	Fernandes	THIRTIETH
C9	Ashwini	Apte	NINETEENTH

9 rows selected.

MISCELLANEOUS FUNCTIONS

UID: This function returns an integer value corresponding to the UserID of the user currently logged in.

Syntax:

```
UID [INTO <variable>]
```

where, **variable** will now contain the id number for the user's session.

Example:

```
SELECT UID FROM DUAL;
```

Output:

```
   UID
------
    61
```

USER: This function returns the **user name** of the user who has logged in. The value returned is in varchar2 data type.

Syntax:

```
USER
```

Example:

```
SELECT USER FROM DUAL;
```

Output:

```
USER
------------------------------
DBA_BANKSYS
```

SYS_CONTEXT: Can be used to retrieve information about Oracle's environment.

Syntax:

```
SYS_CONTEXT (<namespace>, <parameter>, [<length>])
```

where, **namespace** is an Oracle namespace that has already been created. If the **namespace** of **USERENV** is used, attributes describing the current Oracle session can be returned.
parameter is a valid attribute that has been set using the DBMS_SESSION.set_context procedure.
length is the length of the return value in bytes. If this parameter is omitted or if an invalid entry is provided, the **SYS_CONTEXT** function will default to **256 bytes**.

The valid parameters for the namespace called **USERENV** are as follows:

Parameter	Explanation	Return Length
AUDITED_CURSORID	Returns the cursor ID of the SQL that triggered the audit	N/A
AUTHENTICATION_DATA	Authentication data	256
AUTHENTICATION_TYPE	Describes how the user was authenticated. Can be one of the following values: Database, OS, Network, or Proxy	30
BG_JOB_ID	If the session was established by an Oracle background process, this parameter will return the Job ID. Otherwise, it will return NULL.	30
CLIENT_IDENTIFIER	Returns the client identifier (global context)	64
CLIENT_INFO	User session information	64
CURRENT_SCHEMA	Returns the default schema used in the current schema	30
CURRENT_SQL	Returns the SQL that triggered the audit event	64
CURRENT_USER	Name of the current user	30
CURRENT_USERID	Userid of the current user	30
DB_NAME	Name of the database from the DB_NAME initialization parameter	30
ENTRYID	Available auditing entry identifier	30
EXTERNAL_NAME	External of the database user	256
HOST	Name of the host machine from which the client has connected	54

Parameter	Explanation	Return Length
CURRENT_SCHEMAID	Returns the identifier of the default schema used in the current schema	30
DB_DOMAIN	Domain of the database from the DB_DOMAIN initialization parameter	256
FG_JOB_ID	If the session was established by a client foreground process, this parameter will return the Job ID. Otherwise, it will return NULL.	30
GLOBAL_CONTEXT_MEMORY	The number used in the System Global Area by the globally accessed context	N/A
INSTANCE	The identifier number of the current instance	30
IP_ADDRESS	IP address of the machine from which the client has connected	30
ISDBA	Returns TRUE if the user has DBA privileges. Otherwise, it will return FALSE.	30
LANG	The ISO abbreviate for the language	62
LANGUAGE	The language, territory, and character of the session. In the following format: language_territory.characterset	52
NETWORK_PROTOCOL	Network protocol used	256
NLS_CALENDAR	The calendar of the current session	62
NLS_CURRENCY	The currency of the current session	62
NLS_DATE_FORMAT	The date format for the current session	62
NLS_DATE_LANGUAGE	The language used for dates	62
NLS_SORT	BINARY or the linguistic sort basis	62
NLS_TERRITORY	The territory of the current session	62
OS_USER	The OS username for the user logged in	30
PROXY_USER	The name of the user who opened the current session on behalf of SESSION_USER	30
PROXY_USERID	The identifier of the user who opened the current session on behalf of SESSION_USER	30
SESSION_USER	The database user name of the user logged in	30
SESSION_USERID	The database identifier of the user logged in	30
SESSIONID	The identifier of the auditing session	30
TERMINAL	The OS identifier of the current session	10

Example:

```
SELECT SYS_CONTEXT('USERENV', 'NLS_DATE_FORMAT') "SysContext" FROM DUAL;
```

Output:

```
SysContext
----------
DD-MON-RR
```

USERENV: Can be used to retrieve information about the current Oracle session. Although this function still exists in Oracle for backwards compatibility, it is recommended that the **SYS_CONTEXT** function is used instead.

Syntax:

```
USERENV(<parameter>)
```

where, **parameter** is the value to return from the current Oracle session.

The possible values are:

Parameter	Explanation
CLIENT_INFO	Returns user session information stored using the DBMS_APPLICATION_INFO package
ENTRYID	Available auditing entry identifier

INSTANCE	The identifier number of the current instance
ISDBA	Returns TRUE if the user has DBA privileges. Otherwise, it will return FALSE.
LANG	The ISO abbreviate for the language
LANGUAGE	The language, territory, and character of the session. In the following format: language_territory.characterset
SESSIONID	The identifier of the auditing session
TERMINAL	The OS identifier of the current session

Example:

```
SELECT USERENV('LANGUAGE') FROM DUAL;
```

Output:

```
USERENV('LANGUAGE')
----------------------------------------
AMERICAN_AMERICA.WE8MSWIN1252
```

COALESCE: Returns the first non-null expression in the list. If all expressions evaluate to null, then the *coalesce* function will return null.

Syntax:

```
COALESCE(<expr1>, <expr2>, ... <expr_n>)
```

Example:

```
SELECT COALESCE(FNAME, CUST_NO) Customers FROM CUST_MSTR;
```

The above coalesce statement is equivalent to the following IF-THEN-ELSE statement:

```
IF FNAME IS NOT NULL THEN
   Customers := FNAME;
ELSIF CUST_NO IS NOT NULL THEN
   Customers := CUST_NO;
ELSE
   Customers := NULL;
END IF;
```

Output:

```
CUSTOMERS
----------------------------------------
Ivan
Chriselle
Mamta
Chhaya
Ashwini
Hansel
Anil
Alex
Ashwini
Namita
011
012
013
014
```

Explanation:

In the above example, Oracle will display the first name i.e. the value held in the field FNAME if first name field holds a value. If does not hold a value, then Oracle will move on to the next column in the **COALESCE** function and display the value held in the next column i.e. CUST_NO if it hold a value.

In case the second column also does not hold a value, then Oracle will display null as an output.

SELF REVIEW QUESTIONS

FILL IN THE BLANKS

1. The Oracle engine will process all rows in a table and display the result only when any of the conditions specified using the _____ operator are satisfied.
2. The _____ predicate allows for a comparison of one string value with another string value which is not identical.
3. For character datatypes the _____ sign matches any string.
4. _____ is a small Oracle worktable, which consists of only one row and one column, and contains the value x in that column.
5. Functions that act on a set of values are called as _____ _____.
6. Variables or constants accepting by functions are called _____.
7. The _____ function returns a string with the first letter of each word in upper case.
8. The _____ function removes characters from the left of char with initial characters removed up to the first character not in set.
9. _____ returns the string passed as a parameter after right padding it to a specified length.
10. The _____ function converts char, a CHARACTER value expressing a number, to a NUMBER datatype.
11. The _____ function converts a value of a DATE datatype to CHAR value.
12. The _____ function returns number of months between two dates.
13. The _____ function returns an integer value corresponding to the UserID of the user currently logged in.

TRUE OR FALSE

14. The Oracle engine will process all rows in a table and display the result only when none of the conditions specified using the NOT operator are satisfied.
15. In order to select data that is within a range of values, the IN BETWEEN operator is used.
16. For character datatypes the percent sign matches any single character.
17. COUNT(expr) function returns the number of rows where expr is not null.
18. ROOT function returns square root of a numeric value.
19. The second parameter in the ROUND function specifies the number of digits after the decimal point.
20. The LOWER function returns char, with all letters in lowercase.
21. The UPPER function returns a string with the first letter of each word in upper case.

22. The LENGTH function returns the length of a word.
23. The LTRIM returns char, with final characters removed after the last character not in the set. 'set' is optional, it defaults to spaces.
24. LPAD returns the string passed as a parameter after left padding it to a specified length.
25. The TO_CHAR (date conversion) converts a value of a NUMBER datatype to a character datatype, using the optional format string.
26. The DATE data type is used to store date and time information.
27. The TO_DATE() function also disallows part insertion of a DATE value into a column.
28. The ADD_MONTHS function returns date after adding the number of months specified in the function.
29. The TO-DATE function allows a user to insert date into a date column in any required format, by specifying the character value of the date to be inserted and its format.

HANDS ON EXERCISES

Using the tables created previously generate the SQL statements for the operations mentioned below. The tables in user are as follows:

a. Client_Master
b. Product_Master
c. Salesman_Master
d. Sales_Order
e. Sales_Order_Details

1. **Perform the following computations on table data:**

a. List the names of all clients having 'a' as the second letter in their names.
b. List the clients who stay in a city whose First letter is 'M'.
c. List all clients who stay in 'Bangalore' or 'Mangalore'
d. List all clients whose BalDue is greater than value 10000.
e. List all information from the Sales_Order table for orders placed in the month of June.
f. List the order information for ClientNo 'C00001' and 'C00002'.
g. List products whose selling price is greater than 500 and less than or equal to 750.
h. List products whose selling price is more than 500. Calculate a new selling price as, original selling price * .15. Rename the new column in the output of the above query as new_price.
i. List the names, city and state of clients who are not in the state of 'Maharashtra'.
j. Count the total number of orders.
k. Calculate the average price of all the products.
l. Determine the maximum and minimum product prices. Rename the output as max_price and min_price respectively.
m. Count the number of products having price less than or equal to 500.
n. List all the products whose QtyOnHand is less than reorder level.

2. **Exercise on Date Manipulation:**

a. List the order number and day on which clients placed their order.
b. List the month (in alphabets) and date when the orders must be delivered.
c. List the OrderDate in the format 'DD-Month-YY'. e.g. 12-February-02.
d. List the date, 15 days after today's date.