Experiment No: 3

Title: Implementation of different types of functions with suitable examples.

- Number Function
- Aggregate Function
- Character Function
- Conversion Function
- Date Function

Objective:
NUMBER FUNCTION:

Abs(n) :Select abs(-15) from dual;

Exp(n): Select exp(4) from dual;

Power(m,n): Select power(4,2) from dual;

Mod(m,n): Select mod(10,3) from dual;

Round(m,n): Select round(100.256,2) from dual;

Trunc(m,n): ;Select trunc(100.256,2) from dual;

Sqrt(m,n);Select sqrt(16) from dual;

Develop aggregate plan strategies to assist with summarization of several data entries.

Aggregative operators: In addition to simply retrieving data, we often want to perform some computation or summarization. SQL allows the use of arithmetic expressions. We now consider a powerful class of constructs for computing aggregate values such as MIN and SUM.

1. Count: COUNT following by a column name returns the count of tuple in that column. If DISTINCT keyword is used then it will return only the count of unique tuple in the column. Otherwise, it will return count of all the tuples (including duplicates) count (*) indicates all the tuples of the column.

Syntax: COUNT (Column name)

Example: SELECT COUNT (Sal) FROM emp;

2. SUM: SUM followed by a column name returns the sum of all the values in that column.

Syntax: SUM (Column name)

Example: SELECT SUM (Sal) From emp;

3. AVG: AVG followed by a column name returns the average value of that column values.

Syntax: AVG (n1, n2...)

Example: Select AVG (10, 15, 30) FROM DUAL;

4. MAX: MAX followed by a column name returns the maximum value of that column.

Syntax: MAX (Column name)

Example: SELECT MAX (Sal) FROM emp;

```
SQL> select deptno, max(sal) from emp group by deptno;

        DEPTNO      MAX (SAL)
        ------   --------
        10        5000
        20        3000
        30        2850

SQL> select deptno, max (sal) from emp group by deptno having max(sal)<3000;

    DEPTNO     MAX(SAL)
    ------     ---------
      30     2850
```

5. MIN: MIN followed by column name returns the minimum value of that column.

Syntax: MIN (Column name)

Example: SELECT MIN (Sal) FROM emp;

```
SQL>select deptno,min(sal) from emp group by deptno having min(sal)>1000;

   DEPTNO  MIN (SAL)
   ------    ---------
    10    1300
```

CHARACTER FUNCTION:

initcap(char) : select initcap(“hello”) from dual;

lower (char): select lower (‘HELLO’) from dual;

upper (char) :select upper (‘hello’) from dual;

ltrim (char,[set]): select ltrim (‘cseit’, ‘cse’) from dual;

rtrim (char,[set]): select rtrim (‘cseit’, ‘it’) from dual;

replace (char,search): select replace(‘jack and jue’,‘j’,‘bl’) from dual;

CONVERSION FUNCTIONS:

To_char: TO_CHAR (number) converts n to a value of VARCHAR2 data type, using the optional number format fmt. The value n can be of type NUMBER, BINARY_FLOAT, or BINARY_DOUBLE.

```
SQL>select to_char(65,'RN')from dual;

LXV
```

To_number : TO_NUMBER converts expr to a value of NUMBER data type.

```
SQL>Select to_number ('1234.64') from Dual;
1234.64
```

To_date:TO_DATE converts char of CHAR, VARCHAR2, NCHAR, or NVARCHAR2 data type to a value of DATE data type.

```
SQL>SELECT TO_DATE('January 15, 1989, 11:00 A.M.')FROM DUAL;

TO_DATE
-----------
15-JAN-89
```

STRING FUNCTIONS:

Concat: CONCAT returns char1 concatenated with char2. Both char1 and char2 can be any of the datatypes

```
SQL>SELECT CONCAT(‘ORACLE’,’CORPORATION’)FROM DUAL;
        ORACLECORPORATION
```

Lpad: LPAD returns expr1, left-padded to length n characters with the sequence of characters in expr2.

```
SQL>SELECT LPAD(‘ORACLE’,15,’*’)FROM DUAL;
        *********ORACLE
```

Rpad: RPAD returns expr1, right-padded to length n characters with expr2, replicated as many times as necessary.

```
SQL>SELECT RPAD (‘ORACLE’,15,’*’)FROM DUAL;
        ORACLE*********
```

Ltrim: Returns a character expression after removing leading blanks.

```
SQL>SELECT LTRIM(‘SSMITHSS’,’S’)FROM DUAL;
        MITHSS
```

Rtrim: Returns a character string after truncating all trailing blanks

```
SQL>SELECT RTRIM(‘SSMITHSS’,’S’)FROM DUAL;
        SSMITH
```

Lower: Returns a character expression after converting uppercase character data to lowercase.

```
SQL>SELECT LOWER(‘DBMS’)FROM DUAL;
        dbms
```

Upper: Returns a character expression with lowercase character data converted to uppercase

```
SQL>SELECT UPPER(‘dbms’)FROM DUAL;
        DBMS
```

Length: Returns the number of characters, rather than the number of bytes, of the given string expression, excluding trailing blanks.

```
SQL>SELECT LENGTH(‘DATABASE’)FROM DUAL;
        8
```

Substr: Returns part of a character, binary, text, or image expression.

```
SQL>SELECT SUBSTR(‘ABCDEFGHIJ’3,4)FROM DUAL;
        CDEF
```

Instr: The INSTR functions search string for substring. The function returns an integer indicating the position of the character in string that is the first character of this occurrence.

```
SQL>SELECT INSTR('CORPORATE FLOOR','OR',3,2)FROM DUAL;
        14
```

DATE FUNCTIONS:

Sysdate:

```
SQL>SELECT SYSDATE FROM DUAL;
        29-DEC-08
```

next_day:

```
SQL>SELECT NEXT_DAY(SYSDATE,'WED')FROM DUAL;
        05-JAN-09
```

add_months:

```
SQL>SELECT ADD_MONTHS(SYSDATE,2)FROM DUAL;
        28-FEB-09
```

last_day:

```
SQL>SELECT LAST_DAY(SYSDATE)FROM DUAL;
        31-DEC-08
```

months_between:

```
SQL>SELECT MONTHS_BETWEEN(SYSDATE,HIREDATE)FROM EMP;
 4
```

Least:

```
SQL>SELECT LEAST('10-JAN-07','12-OCT-07')FROM DUAL;
        10-JAN-07
```

Greatest:

```
SQL>SELECT GREATEST('10-JAN-07','12-OCT-07')FROM DUAL;
        10-JAN-07
```

Trunc:

```
SQL>SELECT TRUNC(SYSDATE,'DAY')FROM DUAL;
        28-DEC-08
```

Round:

```
SQL>SELECT ROUND(SYSDATE,'DAY')FROM DUAL;
        28-DEC-08
```

to_char:

```
SQL> select to_char(sysdate, "dd\mm\yy") from dual;
        24-mar-05.
```

to_date:

```
SQL> select to date (sysdate, "dd\mm\yy") from dual;
        24-mar-o5.
```

LAB PRACTICE ASSIGNMENT:

Create a table EMPLOYEE with following schema:

(Emp_no, E_name, E_address, E_ph_no, Dept_no, Dept_name,Job_id, Designation , Salary)

Write SQL statements for the following query.

1. List the E_no, E_name, Salary of all employees working for MANAGER.
2. Display all the details of the employee whose salary is more than the Sal of any IT PROFF..
3. List the employees in the ascending order of Designations of those joined after 1981.
4. List the employees along with their Experience and Daily Salary.
5. List the employees who are either ‘CLERK’ or ‘ANALYST’ .
6. List the employees who joined on 1-MAY-81, 3-DEC-81, 17-DEC-81,19-JAN-80 .
7. List the employees who are working for the Deptno 10 or20.
8. List the Enames those are starting with ‘S’ .
9. Dislay the name as well as the first five characters of name(s) starting with ‘H’
10. List all the emps except ‘PRESIDENT’ & ‘MGR” in asc order of Salaries.

Experiment No: 4

Title : Implementation of different types of operators in SQL.

- Arithmetic Operator
- Logical Operator
- Comparision Operator
- Special Operator
- Set Operator

Objective:

To learn different types of operator.

Theory:

ARIHMETIC OPERATORS:

(+) : Addition - Adds values on either side of the operator .

(-):Subtraction - Subtracts right hand operand from left hand operand .

(*):Multiplication - Multiplies values on either side of the operator .

(/):Division - Divides left hand operand by right hand operand .

(^):Power- raise to power of .

(%):Modulus - Divides left hand operand by right hand operand and returns remainder.

LOGICAL OPERATORS:

AND : The AND operator allows the existence of multiple conditions in an SQL statement's WHERE clause.

OR: The OR operator is used to combine multiple conditions in an SQL statement's WHERE clause.

NOT: The NOT operator reverses the meaning of the logical operator with which it is used. Eg: NOT EXISTS, NOT BETWEEN, NOT IN, etc. **This is a negate operator.**

COMPARISION OPERATORS:

(=): Checks if the values of two operands are equal or not, if yes then condition becomes true.

(!=): Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.

(< >): Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.

(>): Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true

(<): Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.

(>=): Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.

(<=): Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.

SPECIAL OPERATOR:

<u>BETWEEN:</u> The BETWEEN operator is used to search for values that are within a set of values, given the minimum value and the maximum value.

<u>IS NULL:</u> The NULL operator is used to compare a value with a NULL attribute value.

<u>ALL:</u> The ALL operator is used to compare a value to all values in another value set

<u>ANY:</u> The ANY operator is used to compare a value to any applicable value in the list according to the condition.

LIKE: The LIKE operator is used to compare a value to similar values using wildcard operators.It allows to use percent sign(%) and underscore (_) to match a given string pattern.

IN: The IN operator is used to compare a value to a list of literal values that have been specified.

EXIST: The EXISTS operator is used to search for the presence of a row in a specified table that meets certain criteria.

SET OPERATORS:

The Set operator combines the result of 2 queries into a single result. The following are the operators:

- Union
- Union all
- Intersect
- Minus

Union: Returns all distinct rows selected by both the queries

Union all: Returns all rows selected by either query including the duplicates.

Intersect: Returns rows selected that are common to both queries.

Minus: Returns all distinct rows selected by the first query and are not by the second

LAB PRACTICE ASSIGNMENT:

1. Display all the dept numbers available with the dept and emp tables avoiding duplicates.
2. Display all the dept numbers available with the dept and emp tables.
3. Display all the dept numbers available in emp and not in dept tables and vice versa.

Experiment No: 5

Title : Implementation of different types of Joins

- Inner Join
- Outer Join
- Natural Join..etc

Objective :

To implement different types of joins

Theory :

The SQL **Joins** clause is used to combine records from two or more tables in a database. A JOIN is a means for combining fields from two tables by using values common to each.The join is actually performed by the 'where' clause which combines specified rows of tables.

Syntax:

```
SELECT column 1, column 2, column 3...
FROM  table_name1, table_name2
WHERE  table_name1.column name  =  table_name2.columnname;
```

Types of Joins :

1. Simple Join
2. Self Join
3. Outer Join

Simple Join:

It is the most common type of join. It retrieves the rows from 2 tables having a common column and is further classified into

Equi-join :

A join, which is based on equalities, is called equi-join.

Example:

```
Select * from item, cust where item.id=cust.id;
```

In the above statement, item-id = cust-id performs the join statement. It retrieves rows from both the tables provided they both have the same id as specified by the where clause. Since the where clause uses the comparison operator (=) to perform a join, it is said to be equijoin. It combines the matched rows of tables. It can be used as follows:

- ✓ To insert records in the target table.
- ✓ To create tables and insert records in this table.
- ✓ To update records in the target table.
- ✓ To create views.

Non Equi-join:

It specifies the relationship between columns belonging to different tables by making use of relational operators other than'='.

Example:

Select * from item, cust where item.id<cust.id;

Table Aliases
Table aliases are used to make multiple table queries shorted and more readable. We give an alias name to the table in the 'from' clause and use it instead of the name throughout the query.

Self join:

Joining of a table to itself is known as self-join. It joins one row in a table to another. It can compare each row of the table to itself and also with other rows of the same table.

Example:

select * from emp x ,emp y where x.salary >= (select avg(salary) from x.emp
where x. deptno =y.deptno);

Outer Join:

It extends the result of a simple join. An outer join returns all the rows returned by simple join as well as those rows from one table that do not match any row from the table. The symbol(+) represents outer join.

- Left outer join
- Right outer join
- Full outer join

LAB PRACTICE ASSIGNMENT:

Consider the following schema:

Sailors (sid, sname, rating, age)

Boats (bid, bname, color)

Reserves (sid, bid, day(date))

1. Find all information of sailors who have reserved boat number 101.
2. Find the name of boat reserved by Bob.
3. Find the names of sailors who have reserved a red boat, and list in the order of age.
4. Find the names of sailors who have reserved at least one boat.
5. Find the ids and names of sailors who have reserved two different boats on the same day.
6. Find the ids of sailors who have reserved a red boat or a green boat.
7. Find the name and the age of the youngest sailor.
8. Count the number of different sailor names.
9. Find the average age of sailors for each rating level.
10. Find the average age of sailors for each rating level that has at least two sailors.

Experiment No: 6

Title : Study & Implementation of

- Group by & Having Clause
- Order by Clause
- Indexing

***Objective*:**

To learn the concept of group functions

***Theory*:**

- **GROUP BY:** This query is used to group to all the records in a relation together for each and every value of a specific key(s) and then display them for a selected set of fields the relation.

Syntax: SELECT <set of fields> FROM <relation_name>
GROUP BY <field_name>;

Example: SQL> SELECT EMPNO, SUM (SALARY) FROM EMP GROUP BY
EMPNO;

GROUP BY-HAVING : The HAVING clause was added to SQL because the WHERE keyword could not be used with aggregate functions. The HAVING clause must follow the GROUP BY clause in a query and must also precede the ORDER BY clause if used.

Syntax:
```
SELECT column_name, aggregate_function(column_name) FROM table_name
WHERE column_name operator value
GROUP BY column_name
HAVING aggregate_function(column_name) operator value;
```

Example **:** SELECT Employees.LastName, COUNT(Orders.OrderID) AS NumberOfOrders
FROM (Orders
INNER JOIN Employees
ON Orders.EmployeeID=Employees.EmployeeID) GROUP BY LastName
HAVING COUNT (Orders.OrderID) > 10;

JOIN using GROUP BY: This query is used to display a set of fields from two relations by matching a common field in them and also group the corresponding records for each and every value of a specified key(s) while displaying.

Syntax: SELECT <set of fields (from both relations)> FROM relation_1,relation_2
WHERE relation_1.field_x=relation_2.field_y GROUP BY field_z;

Example:

SQL> SELECT empno,SUM(SALARY) FROM emp,dept
WHERE emp.deptno =20 GROUP BY empno;

- <u>**ORDER BY:**</u> This query is used to display a selected set of fields from a relation in an ordered manner base on some field.

Syntax: SELECT <set of fields> FROM <relation_name>
ORDER BY <field_name>;

Example: SQL> SELECT empno, ename, job FROM emp ORDER BY job;

JOIN using ORDER BY: This query is used to display a set of fields from two relations by matching a common field in them in an ordered manner based on some fields.

Syntax: SELECT <set of fields (from both relations)> FROM relation_1, relation_2
WHERE relation_1.field_x = relation_2.field_y ORDER BY field_z;

Example: SQL> SELECT empno,ename,job,dname FROM emp,dept
WHERE emp.deptno = 20 ORDER BY job;

- **INDEXING**: An *index* is an ordered set of pointers to the data in a table. It is based on the data values in one or more columns of the table. SQL Base stores indexes separately from tables.

  An index provides two benefits:

  - It improves performance because it makes data access faster.
  - It ensures uniqueness. A table with a unique index cannot have two rows with the same values in the column or columns that form the index key.

Syntax:

```
CREATE INDEX <index_name> on <table_name> (attrib1,attrib 2….attrib n);
```

Example:

```
CREATE INDEX id1 on emp(empno,dept_no);
```

<u>LAB PRACTICE ASSIGNMENT:</u>

Create a relation and implement the following queries.

1. Display total salary spent for each job category.
2. Display lowest paid employee details under each manager.
3. Display number of employees working in each department and their department name.
4. Display the details of employees sorting the salary in increasing order.
5. Show the record of employee earning salary greater than 16000 in each department.
6. Write queries to implement and practice the above clause.