# 4. Structural Query Language(SQL)

**SQL (Structured Query Language):**

Structured Query Language is a database computer language designed for managing data in relational database management systems(RDBMS), and originally based upon Relational Algebra.

Its scope includes data query and update, schema creation and modification, and data access control. SQL was one of the first languages for Edgar F. Codd's relational model in his influential 1970 paper, "A Relational Model of Data for Large Shared Data Banks"[3] and became the most widely used language for relational databases.

- IBM developed SQL in mid of 1970's.
- Oracle incorporated in the year 1979.
- SQL used by IBM/DB2 and DS Database Systems.
- SQL adopted as standard language for RDBS by ASNI in 1989.

**DATA TYPES:**

1. **CHAR (Size):** This data type is used to store character strings values of fixed length. The size in brackets determines the number of characters the cell can hold. The maximum number of character is 255 characters.

2. **VARCHAR (Size) / VERCHAR2 (Size)**: This data type is used to store variable length alphanumeric data. The maximum character can hold is 2000 character.

3. **NUMBER (P, S):** The NUMBER data type is used to store number (fixed or floating point). Number of virtually any magnitude may be stored up to 38 digits of precision. Number as large as $9.99 * 10^{124}$. The precision (p) determines the number of places to the right of the decimal. If scale is omitted then the default is zero. If precision is omitted, values are stored with their original precision up to the maximum of 38 digits.

4. **DATE:** This data type is used to represent date and time. The standard format is DD-MM-YY as in 17-SEP-2009. To enter dates other than the standard format, use the appropriate functions. Date time stores date in the 24-Hours format. By default the time in a date field is 12:00:00 am, if no time portion is specified. The default date for a date field is the first day the current month.

5. **LONG:** This data type is used to store variable length character strings containing up to 2GB. Long data can be used to store arrays of binary data in ASCII format. LONG values cannot be indexed, and the normal character functions such as SUBSTR cannot be applied.

6. **RAW:** The RAW data type is used to store binary data, such as digitized picture or image. Data loaded into columns of these data types are stored without any further conversion. RAW data type can have a maximum length of 255 bytes. LONG RAW data type can contain up to 2GB.

**Meanings:**

**DDL** is Data Definition Language statements. Some examples:

**CREATE** - to create objects in the database

**ALTER** - alters the structure of the database

**DROP** - delete objects from the database

**TRUNCATE** - remove all records from a table, including all spaces allocated for the records and removed

**COMMENT** - add comments to the data dictionary

**GRANT** - gives user's access privileges to database

**REVOKE** - withdraw access privileges given with the GRANT command

**DML** is Data Manipulation Language statements. Some examples:

**SELECT** - retrieve data from the a database

**INSERT** - insert data into a table

**UPDATE** - updates existing data within a table

**DELETE** - deletes all records from a table, the space for the records remain

**CALL** - call a PL/SQL or Java subprogram

**EXPLAIN PLAN** - explain access path to data

**LOCK TABLE** - control concurrency

**DCL** is Data Control Language statements. Some examples:

**COMMIT** - save work done

**SAVEPOINT** - identify a point in a transaction to which you can later roll back

**ROLLBACK** - restore database to original since the last COMMIT

**SET TRANSACTION** - Change transaction options like what rollback segment to use

There are five types of SQL statements. They are:

1. DATA DEFINITION LANGUAGE (DDL)

2. DATA MANIPULATION LANGUAGE (DML)

3. DATA RETRIEVAL LANGUAGE (DRL) OR DATA QUERY LANGUAGE (DQL)

4. TRANSATIONAL CONTROL LANGUAGE (TCL)

5. DATA CONTROL LANGUAGE (DCL)

**1. DATA DEFINITION LANGUAGE (DDL):** The Data Definition Language (DDL) is used to create and destroy databases and database objects. These commands will primarily be used by database administrators during the setup and removal phases of a database project. Let's take a look at the structure and usage of four basic DDL commands:

1. CREATE 2. ALTER 3. DROP 4. RENAME

**1. CREATE:**

**(a)CREATE TABLE:** This is used to create a new relation and the corresponding

***Syntax:*** CREATE TABLE relation_name

(field_1 data_type(Size),field_2 data_type(Size), .. . );

***Example:***

SQL>CREATE TABLE Student (sno NUMBER(3),sname CHAR(10),class CHAR(5));

**Example:**
SQL> CREATE TABLE Emp1 ( EmpNo Number(10)PRIMARY KEY,
EName VarChar2(15), Job Char(10),deptname varchar2(10),deptno number(9),Hiredate Date, salary number(8),exp number(5)));

**RESULT:** Table created.

**(b)CREATE TABLE..AS SELECT….:** This is used to create the structure of a new relation from the structure of an existing relation.

***Syntax:*** CREATE TABLE (relation_name_1, field_1,field_2,.....field_n) AS SELECT field_1,field_2,...........field_n FROM relation_name_2;

***Example:*** SQL>CREATE TABLE std(rno,sname) AS SELECT sno,sname FROM student;

**** DESC:** It is used to describe a schema as well as to retrieve rows from table in descending order.

**SYNTAX:** DESC

EX: SQL> DESC EMP1;

```
 NAME                          NULL?    TYPE
 ----------------------------------------- -------- --------------------------
 EMPNO                          NOT NULL   NUMBER(10)
 ENAME                                     VARCHAR2(15)
 JOB                                       CHAR(10)
 DEPTNAME                                  VARCHAR2(10)
 DEPTNO                                    NUMBER(9)
 HIREDATE                                  DATE
 SALARY                                    NUMBER(8)
 EXP                                       NUMBER(5)
```

**2. ALTER:**

**(a)ALTER TABLE …ADD…:** This is used to add some extra fields into existing relation.

***Syntax:*** ALTER TABLE relation_name ADD(new field_1 data_type(size), new field_2 data_type(size),..);

***Example :*** SQL>ALTER TABLE emp1 ADD(Address CHAR(10));

TABLE ALTERED.

/**TO CHECK WHETHER COLUMN ADD OR NOT CHECK USING DESC COMMAND**/

SQL> DESC EMP1;
```
 NAME                          NULL?    TYPE
```

```
----------------------------------------- -------- ---------------
EMPNO                          NOT NULL NUMBER (10)
ENAME                                  VARCHAR2 (15)
JOB                               CHAR (10)
DEPTNAME                                  VARCHAR2 (10)
DEPTNO                                 NUMBER (9)
HIREDATE                                 DATE
SALARY                                 NUMBER (8)
EXP                               NUMBER (5)
ADDRESS                                  CHAR (10)
```

**(b)ALTER TABLE…MODIFY…:** This is used to change the width as well as data type of fields of existing relations.

***Syntax:*** ALTER TABLE relation_name MODIFY (field_1 newdata_type(Size), field_2 newdata_type(Size),....field_newdata_type(Size));

***Example:***SQL>ALTER TABLE emp1 MODIFY(ename VARCHAR2(20),salary NUMBER(5));

**TABLE ALTERED.**

```
SQL> DESC EMP1;
 NAME                           NULL?   TYPE
 ----------------------------------------- -------- -----------------
 EMPNO                          NOT NULL NUMBER(10)
 ENAME                                  VARCHAR2(20)
 JOB                               CHAR(10)
 DEPTNAME                                  VARCHAR2(10)
 DEPTNO                                 NUMBER(9)
 HIREDATE                                 DATE
 SALARY                                 NUMBER(5)
 EXP                               NUMBER(5)
 ADDRESS                                  CHAR(10)
```

**3. DROP TABLE:** This is used to delete the structure of a relation. It permanently deletes the records in the table.

***Syntax:*** DROP TABLE relation_name;

***Example:*** SQL>DROP TABLE EMP1;

Table dropped;

```
SQL> SELECT * FROM TAB;

TNAME                    TABTYPE  CLUSTERID
------------------------------ ------- ----------
BONUS                    TABLE
DEPT                     TABLE
EMP                      TABLE
EMPLOYEE1                  TABLE
SALGRADE                   TABLE

5 ROWS SELECTED.
```

**4. RENAME:** It is used to modify the name of the existing database object.

***Syntax:*** RENAME TABLE old_relation_name TO new_relation_name;

***Example:*** SQL>RENAME TABLE EMP1 TO EMP2;

Table renamed.

**5. TRUNCATE:** This command will remove the data permanently. But structure will not be removed.

***Syntax:*** TRUNCATE TABLE <Table name>

***Example*** TRUNCATE TABLE EMP1;

**<u>Difference between Truncate & Delete:-</u>**

- ✓ By using truncate command data will be removed permanently & will not get back where as by using delete command data will be removed temporally & get back by using roll back command.
- ✓ By using delete command data will be removed based on the condition where as by using truncate command there is no condition.
- ✓ Truncate is a DDL command & delete is a DML command.

**<u>2. DATA MANIPULATION LANGUAGE (DML):</u>** The Data Manipulation Language (DML) is used to retrieve, insert and modify database information. These commands will be used by all database users during the routine operation of the database. Let's take a brief look at the basic DML commands:

**1. INSERT** **2. UPDATE** **3. DELETE**

**1. INSERT INTO:** This is used to add records into a relation. These are three type of INSERT INTO queries which are as

**a) Inserting a single record**

***Syntax:*** INSERT INTO relationname(field_1,field_2,.field_n)VALUES (data_1,data_2,........data_n);

*Example:* SQL>INSERT INTO student(sno,sname,class,address)VALUES (1,'Ravi','M.Tech','Palakol');

**b) Inserting all records from another relation**

*Syntax:* INSERT INTO relation_name_1 SELECT Field_1,field_2,field_n FROM relation_name_2 WHERE field_x=data;

*Example:* SQL>INSERT INTO EMP1 SELECT sno,sname FROM student WHERE name = 'Ramu';

**c) Inserting multiple records**

*Syntax:* INSERT INTO relation_name field_1,field_2,.....field_n) VALUES (&data_1,&data_2,........&data_n);

*Example:* SQL>INSERT INTO EMP1(empno,ename,job,deptname,deptno,hiredate,salary,exp,address) VALUES (&empno,&ename,&job,&deptname,&deptno,&hiredate,&salary,&exp,&address)

```
Enter value for empno: 101
Enter value for ename: Ramesh
Enter value for job: asst.prof
Enter value for deptname: it
Enter value for deptno:10
Enter value for hiredate:10-12-2011
Enter value for salay:20000
Enter value for exp:2
Enter value for address:wgl
```

**2. UPDATE-SET-WHERE:** This is used to update the content of a record in a relation.

*Syntax:* SQL>UPDATE relation name SET Field_name1=data,field_name2=data, WHERE field_name=data;

*Example:* SQL>UPDATE emp1 SET ename = 'kumar' WHERE empno=1;

**3. DELETE-FROM**: This is used to delete all the records of a relation but it will retain the structure of that relation.

**a) DELETE-FROM**: This is used to delete all the records of relation.

*Syntax:* SQL>DELETE FROM relation_name;

*Example:* SQL>DELETE FROM EMP1;

**b) DELETE -FROM-WHERE:** This is used to delete a selected record from a relation.

*Syntax:* SQL>DELETE FROM relation_name WHERE condition;

*Example:* SQL>DELETE FROM emp1 WHERE empno = 2;

**3. DRL(DATA RETRIEVAL LANGUAGE):** Retrieves data from one or more tables.

**1. SELECT FROM:** To display all fields for all records.

*Syntax :* SELECT * FROM relation_name;

*Example :* SQL> select * from dept;

```
     DEPTNO     DNAME          LOC
-------- ----------- ----------
     10  ACCOUNTING          NEW YORK
     20  RESEARCH    DALLAS
```

Example: SQL> select * from emp1;

```
 EMPNO  ENAME  JOB DEPTNAME DEPTNO HIREDATE SALARY EXP
ADDRESS

 101  ramesh   asst.prof  it   10 10-DEC-11 20000  2  wgl

 102  ramu     asst.prof  it   10 05-JUL-12 10000  0 wgl

 103  rakesh   asst.prof  it   10 12-AUG-11  5000  0 hnk
```

**2. SELECT FROM:** To display a set of fields for all records of relation.

*Syntax:* SELECT a set of fields FROM relation_name;

*Example:* SQL> select empno, ename from emp1;

```
   EMPNO ENAME
---------- --------------------
     101 ramesh
     102 ramu
     103 rakesh
```

**3. SELECT - FROM -WHERE:** This query is used to display a selected set of fields for a selected set of records of a relation.

*Syntax:* SELECT a set of fields FROM relation_name WHERE condition;

*Example:* SQL> select * FROM emp1 WHERE deptno<=20;

SQL> select * from emp1 where deptno<20;

```
EMPNO ENAME  JOB DEPTNAME DEPTNO HIREDATE SALARY EXP
ADDRESS
--------------------------------------------------------------
101 ramesh  it   asst.prof  10 10-DEC-11  20000    2 wgl
102 ramu    it   asst.prof  10 05-JUL-12  10000    0 wgl
103 rakesh  it  asst.prof   10 12-AUG-11  5000     0 hnk
```

**4. SELECT - FROM -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 a set of fields FROM relation_name GROUP BY field_name;

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

```
                EMPNO          SUM (SALARY)
                ------    ----------
                 101      20000
                 102      10000
                 103      5000

          4 rows selected.
```

**5. SELECT - FROM -ORDER BY:** This query is used to display a selected set of fields from a relation in an ordered manner base on some field.

***Syntax:*** SELECT a set of fields FROM relation_name
ORDER BY field_name;

***Example:*** SQL> SELECT empno, ename, job FROM emp1 ORDER BY job;

```
                EMPNO          ENAME     JOB
                ------    ---------    --------
                 4             RAVI              MANAGER
                 2             aravind   Manager
                 1             sagar      clerk
                 3             Laki      clerk
          4rows selected.
```

**8. UNION:** This query is used to display the combined rows of two different queries, which are having the same structure, without duplicate rows.

***Syntax:*** SELECT field_1,field_2,....... FROM relation_1 WHERE (Condition) UNION SELECT field_1,field_2,....... FROM relation_2 WHERE (Condition);

***Example:***

```
        SQL> SELECT * FROM STUDENT;
                SNO            SNAME
                -----          -------
                1              kumar
                2              ravi
                3              ramu

        SQL> SELECT * FROM STD;
                SNO            SNAME
                -----          -------
                3              ramu
                5              lalitha
                9              devi
                1              kumar

SQL> SELECT * FROM student UNION SELECT * FROM std;
                SNO            SNAME
                ----           ------
                 1             kumar
                 2             ravi
                 3             ramu
                 5             lalitha
                 9             devi
```

**9. INTERSECT:** This query is used to display the common rows of two different queries, which are having the same structure, and to display a selected set of fields out of them.

***Syntax:*** SELECT field_1,field_2,.. FROM relation_1 WHERE
(Condition) INTERSECT SELECT field_1,field_2,.. FROM relation_2
WHERE(Condition);

***Example :*** SQL> SELECT * FROM student INTERSECT SELECT * FROM std;

```
        SNO            SNAME
        ----           -------
        1              Kumar
```

**10. MINUS:** This query is used to display all the rows in relation_1,which are not having in the relation_2.

***Syntax:*** SELECT field_1,field_2,......FROM relation_1
WHERE(Condition) MINUS SELECT field_1,field_2,.....
FROM relation_2 WHERE(Conditon);

**SQL>** SELECT * FROM student MINUS SELECT * FROM std;

```
SNO            SNAME
----           -------
2              RAVI
3              RAMU
```

**3. TRANSATIONAL CONTROL LANGUAGE (T.C.L):**

A transaction is a logical unit of work. All changes made to the database can be referred to as a transaction. Transaction changes can be mode permanent to the database only if they are committed a transaction begins with an executable SQL statement & ends explicitly with either role back or commit statement.

**1. COMMIT:** This command is used to end a transaction only with the help of the commit command transaction changes can be made permanent to the database.

***Syntax:*** SQL>COMMIT;

***Example:*** SQL>COMMIT;

**2. SAVE POINT**: Save points are like marks to divide a very lengthy transaction to smaller once. They are used to identify a point in a transaction to which we can latter role back. Thus, save point is used in conjunction with role back.

***Syntax:*** SQL>SAVE POINT ID;

***Example:*** SQL>SAVE POINT xyz;

**3. ROLE BACK:** A role back command is used to undo the current transactions. We can role back the entire transaction so that all changes made by SQL statements are undo (or) role back a transaction to a save point so that the SQL statements after the save point are role back.

***Syntax:*** ROLE BACK( current transaction can be role back)

ROLE BACK to save point ID;

***Example:*** SQL>ROLE BACK;

SQL>ROLE BACK TO SAVE POINT xyz;

**4. DATA CONTROL LANGUAGE (D.C.L)**:

DCL provides uses with privilege commands the owner of database objects (tables), has the soul authority ollas them. The owner (data base administrators) can allow other data base uses to access the objects as per their requirement

**1. GRANT:** The GRANT command allows granting various privileges to other users and allowing them to perform operations with in their privileges
***For Example***, if a uses is granted as 'SELECT' privilege then he/she can only view data but cannot perform any other DML operations on the data base object GRANTED privileges can also be withdrawn by the DBA at any time

***Syntax:*** SQL>GRANT PRIVILEGES on object_name To user_name;

***Example***: SQL>GRANT SELECT, UPDATE on emp1 To hemanth;

**2. REVOKE:** To with draw the privileges that has been GRANTED to a uses, we use the REVOKE command

***Syntax:*** SQL>REVOKE PRIVILEGES ON object-name FROM user_name;

***Example:*** SQL>REVOKE SELECT, UPDATE ON emp FROM ravi;

**Defining integrity constraints in the alter table command:**

***Syntax:*** **ALTER TABLE** Table_Name **ADD PRIMARY KEY** (column_name);

***Example:*** **ALTER TABLE** student **ADD PRIMARY KEY** (sno);

(Or)

***Syntax:*** **ALTER TABLE** table_name **ADD CONSTRAINT** constraint_name **PRIMARY KEY**(colname)

***Example:*** **ALTER TABLE** student **ADD CONSTRAINT** SN **PRIMARY KEY(**SNO**)**

**<u>Dropping integrity constraints in the alter table command:</u>**

***Syntax:*** **ALTER TABLE** Table_Name **DROP** constraint_name;

***Example:*** **ALTER TABLE** student **DROP PRIMARY KEY**;

(or)

***Syntax:*** **ALTER TABLE** student **DROP CONSTRAINT** constraint_name**;**

***Example:*** **ALTER TABLE** student **DROP CONSTRAINT** SN**;**

## 2. Queries (along with sub Queries)

***Selecting data from sailors table***

```
SQL> select * from sailors;
     SID SNAME       AGE    RATING
--------- ---------- --------- ---------
      22 dustin       7       45
      29 brutus       1       33
      31 lubber       8       55
      32 andy         8      25.5
      58 rusty       10       35
      64 horatio      7       35
      71 zorba       10       40
      74 horatio      9       40
      85 art          3     25.5
      95 bob          3      63.5

10 rows selected.
```

***Selecting data from reserves table***

```
SQL> select * from reserves;

            SID      BID       DAY
          --------- --------- ------
      22      101       10-OCT-98
      22      102       10-OCT-98
      22      103       10-AUG-98
      22      104       10-JUL-98
      31      102       11-OCT-98
      31      103       11-JUN-98
      31      104       11-DEC-98
      64      101       09-MAY-98
      64      102       09-AUG-98
      74      104       09-AUG-98
10 rows selected.
```

***Selecting data from boat table***

```
SQL> select * from boats;
     BID BNAME              COLOR
--------- ------------------- ----------
     101 interlake          blue
     102 interlake          red
     103 clipper            green
     104 marine             red
```

***Q: find the names of sailors who have reserved boat 103.***

```
SQL> select s.sname from sailors s where s.sid in (select r.sid
                                                   from reserves r where r.bid=103);

          SNAME
          -------------
          dustin
          lubber

          2 rows selected.
```

***Q: find the names of sailors who have reserved a red boat.***

```
SQL> select s.sname from sailors s where s.sid in
          (select r.sid from reserves r where r.bid in (select b.bid
                                                  from boats b where b.color='red'));
          SNAME
          -------------------
          dustin
          lubber
          horatio
          horatio

          4 rows selected.
```

***Q: Find the name and age of the oldest sailors.***

```
SQL> select MAX(s.age)from sailors s;

          MAX(S.AGE)
          ----------
               10
```

***Q: Count the number of sailors.***

```
SQL> select COUNT(s.age)from sailors s

COUNT(S.AGE)
------------
        10
```

***Q: Count the number of different sailors names.***

```
SQL> select COUNT(distinct s.sname)* from sailors s

COUNT(DISTINCTS.SNAME)
----------------------
                 9
```

***Q: find the names of sailors who have not reserved a red boat.***

```
SQL> select s.sname
  2  from sailors s
  3  where s.sid not in (select r.sid
  4              from reserves r
  5              where r.bid in (select b.bid
  6                          from boats b
  7                          where b.color='red'))
             SNAME
             --------
             brutus
             andy
             rusty
             zorba
             art
             bob
6 rows selected.
```

***Q: find the names of sailors who have not reserved boat 103.***

```
SQL> select s.sname from sailors s where exists(select * from
                                              reserves r where r.bid=103 and r.sid=s.sid);

             SNAME
             -------
             dustin
             lubber

             2 rows selected.
```

***Q: find the names of sailors who have reserved at least one boat.***

```
SQL> select s.sname from sailors s, reserves r where
                                                                          s.sid=r.sid;

             SNAME
             ----------
             dustin
             dustin
             dustin
             dustin
             lubber
             lubber
             lubber
             horatio
             horatio
             horatio

             10 rows selected.
```

***Q: Compute increments for the ratings of persons who have sailed two different boats on the same day.***

```
SQL> select s.sname,s.rating+1 As rating from sailors s,
                            reserves r1, reserves r2 where s.sid=r1.sid AND s.sid=r2.sid
                                             AND r1.day=r2.day AND r1.bid<>r2.bid

         SNAME                  RATING
         ------------------- ---------
         dustin                 46
         dustin                 46
```

***Q: Find the names of sailors who have reserved a red or a green boat.***

```
SQL> select s.sname from sailors s, reserves r,boats b where s.sid=r.sid AND r.bid=b.bid
AND (b.color='red' OR
                                                                              b.color='green')

         SNAME
         -------------------
         dustin
         dustin
         dustin
         lubber
         lubber
         lubber
         horatio
         horatio

         8 rows selected.
```

***Q: find the all sids of sailors who have rating 10 or have reserved boat 104..***

```
SQL> select s.sid from sailors s where s.rating=10 union
   select r.sid from reserves r where r.bid=104;

    SID
         ------
    22
    31
    74
```

***Q: Find the number of reservations for each red boat.***

```
SQL> select b.bid,count(*)As sailorcount from boats b,
                                   reserves r where r.bid=b.bid AND b.color='red' group by b.bid;

             BID SAILORCOUNT
             --------- -----------
                  102           3
                  104           3
```

***Q: Find the minimum age of the sailor.***

```
SQL> select min(s.age) from sailors s;
             MIN(S.AGE)
             ----------
                    1
```

***Q: Find the sum of the rating of sailors.***

```
SQL> select sum(s.rating)from sailors s;
             SUM(S.RATING)
             -------------
                   397.5
```

***Q: find the id and names of sailors who have reserved id=22 or age<25.***

```
SQL> select sid,sname from sailors where sid=22 or age<25

              SID SNAME
      -- --------
       22 dustin
```

**3) Queries using Aggregate functions (COUNT, SUM, AVG, MAX and MIN), GROUP BY, HAVING and Creation and dropping of Views.**

**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 emp1;

**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 emp1;

**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
```

**VIEW:** In SQL, a view is a virtual table based on the result-set of an SQL statement.

A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

You can add SQL functions, WHERE, and JOIN statements to a view and present the data as if the data were coming from one single table.

A view is a virtual table, which consists of a set of columns from one or more tables. It is similar to a table but it doest not store in the database. View is a query stored as an object.

***Syntax:*** CREATE VIEW view_name AS SELECT set of fields FROM relation_name WHERE (Condition)

***1. Example:***

```
SQL>CREATE VIEW employee AS SELECT empno,ename,job FROM EMP
    WHERE job = 'clerk';
    View created.
SQL> SELECT * FROM EMPLOYEE;
    EMPNO       ENAME       JOB
    ----        ------      -------
    7369        SMITH       CLERK
    7876        ADAMS       CLERK
    7900        JAMES       CLERK
    7934        MILLER      CLERK
```

***2.Example:***

```
CREATE VIEW [Current Product List] AS
SELECT ProductID,ProductName
FROM Products
WHERE Discontinued=No
```

**DROP VIEW**: This query is used to delete a view , which has been already created.

***Syntax:*** DROP VIEW View_name;

***Example :*** SQL> DROP VIEW EMPLOYEE;

View dropped

**4. Queries using Conversion functions (to_char, to_number and to_date), string functions (Concatenation, lpad, rpad, ltrim, rtrim, lower, upper, initcap, length, substr and instr), date functions (Sysdate, next_day, add_months, last_day, months_between, least, greatest, trunc, round, to_char, to_date)**

**1. 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
```

**2. 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
```

**<u>3. Date functions:</u>**

**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.
```

**Consider the following relations with underlined primary keys.**
**Product(<u>P_code,</u> Description, Stocking_date, QtyOnHand, MinQty, Price, Discount,V_code)**
**Vendor(<u>V_code</u>, Name, Address, Phone)**
**Here a vendor can supply more than one product but a product is supplied by only one vendor. Write SQL queries for the following:**

- **i. List the names of all the vendors who supply more than one product.**
- **ii. List the details of the products whose prices exceed the average product price.**

**iii. List the Name, Address and Phone of the vendors who are currently not supplying any product.**

**Ans:**

i. Select Name from Vendor Where V_code in (Select V_code from Product group by V_code having count (V_code) > 1)

ii. Select * from Product Where Price > (Select avg (Price)from Product)

iii. Select Name, Address, Phone From Vendor Where V_code not in (Select V_code from Product)

**Consider the following relations**
**Physician (rgno, phyname, addr, phno)**
**Patient (ptname, ptaddr)**
**Visits(rgno, ptname, dateofvisit, fees-charged)**
**Answer the following in SQL :**

i. **Define the tables. Identify the keys and foreign keys.**

ii. **Create an assertion that the total fees charged for a patient can not be more than Rs.1000/- assuming that patients can visit the same doctor more than once.**

iii. **Create a view Patient_visits(name, times) where name is the name of the patient and times is the number of visits of a patient.**

iv. **Display the ptname, ptaddr of the patient(s) who have visited more than one physician in the month of May 2000 in ascending order of ptname.**

**Ans:**

i.
```
CREATE TABLE PHYSICIAN ( RGNO VARCHAR2(5) PRIMARY
KEY, PHYNAME VARCHAR2(15),
ADDR VARCHAR2(25),
PHNO VARCHAR2(10));
CREATE TABLE PATIENT
( PTNAME VARCHAR2(15) PRIMARY KEY,
PTADDR VARCHAR2(25));
CREATE TABLE VISITS
( RGNO VARCHAR2(5) REFERENCES PHYSICIAN(RGNO),
PTNAME VARCHAR2(15) REFERENCES, PATIENT(PTNAME),
DATEOFVISIT DATE,
FEE-CHARGED NUMBER(8,2),
CONSTRAINT VISITS_PK PRIMARY KEY(RGNO, PTNAME));
```

ii.
```
CREATE ASSERTION CHECK ((SELECT
SUM(FEES_CHARGED) FROM VISITS WHERE PTNAME
=NEW.PTNAME AND RGNO = NEW.RGNO) <= 1000)
```

iii.
```
CREATE VIEW PATIENT_VISITS (NAME, TIMES) ASSELECT
PTNAME, COUNT(PTNAME) FROM VISITS GROUP BY
PTNAME;
```

iv.
```
(SELECT * FROM PATIENT WHERE PTNAME IN (SELECT
PTNAME FROM VISITS A, VISITS B WHERE A.PTNAME =
B.PTNAME AND A.RGNO <> B.RGNO AND A.DATEOFVISIT
BETWEEN TO_DATE('01-MAY-2000') AND TO_DATE('31-MAY-
2000')) ORDER BY PTNAME
```

**Consider the following relations with key underlined**
**Customer (<u>C#</u>, Cname, Address)**
**Item (<u>I#</u>, Iname, Price, Weight)**
**Order (<u>O#</u>, C#, I#, Quantity)**
**Write SQL queries for the following:**

a. **List the names of customers who have ordered items weighing more than 1000 and only those.**

b. **List the names of customers who have ordered atleast one item priced over Rs.500.**

c. **Create a view called "orders" that has the total cost of every order.**

Ans:

a.
```
SELECT CNAME FROM CUSTOMER WHERE C# IN
(SELECT C# FROM ORDER
MINUS
SELECT C# FROM ORDER, ITEM
WHERE ORDER.I# = ITEM.I# AND WEIGHT <= 1000)
```

b.
```
SELECT CNAME FROM CUSTOMER WHERE C# IN
(SELECT C# FROM ORDER WHERE I# IN
(SELECT I# FROM ITEM WHERE PRICE > 500))
```

**OR**

```
SELECT DISTINCT CNAME FROM CUSTOMER, ORDER, ITEM
WHERE CUSTOMER.C# = ORDER.C# AND ORDER.I# =
ITEM.I# AND PRICE > 500
```

c.
```
CREATE VIEW ORDERS (O#, TOTALCOST) AS
SELECT O#, SUM(QUANTITY * PRICE)
FROM ORDER, ITEM WHERE ORDER.I# = ITEM.I#
GROUP BY O#
```

**Consider the following relations with key underlined**
**lives (person_name, street, city)**
**works (person_name, company_name, salary)**
**located (company_name, city)**
**manages (person_name, manager_name)**
**Answer the following using SQL:**

i. **Find the names and city of persons who work for manager John.**

ii. **Find the names of persons who live in the same city as the company they work for.**

iii. **John's manager has changed. The new manager is Anna.**

iv. **Susan doesn't work anymore.**

v. **Create a view BangWork (person_name, company_name, manager_name) of all people who work in Bangalore in ascending order of person name**

Ans:

i. Select person_name, city from lives, manages where manager_name = 'John' and lives.person_name = manages.person_name;

ii. Select person_name from lives works, located where lives.person_name = works.person_name and works.company_name = located.company_name and lives.city = located. city;

iii. Update manages set manager_name = 'Anna' where manager_name = 'John';

iv. delete from works where person_name = 'Susan';

v. create view BangWork as select person_name, company_name, manager_name from work where city = 'Bangalore' order by person_name;

**Consider the following relations with underlined primary keys**
**Product(<u>P_code</u>, Description, Stocking_date, QtyOnHand, MinQty, Price, Discount,**
**<u>V_code</u>)**
**Vendor(<u>V_code</u>, Name, Address, Phone)**
**Here a vendor can supply more than one product but a product is supplied by only one vendor. Write SQL queries for the following :**

i. **List the names of all the vendors who supply more than one product.**

ii. **List the details of the products whose prices exceed the average product price.**

iii. **List the Name, Address and Phone of the vendors who are currently not supplying any product. (3 x 3)**

**Ans:**

i. Select name from Vendor where V_code in (Select V_code from Product group by V_code having count (V_code) >1)

ii. Select * from Product where price > (select avg(Price) from product)

iii. Select Name, Address, Phone from Vendor v where not exists ( select * from Product p where p.V_code = v.V_code)

**Consider the following tables.**
**EMP (emp_no, name, salary, supervisor_no, sex_code, dept_code)**
**DEPT (dept_cd, dept_name)**
**Write down queries in SQL for getting following information:**
**(i) Employees getting more salary than their supervisor.**
**(ii) Department name and total number of employees in each department who earn more than average salary for their department.**
**(iii) Department(s) having maximum employees earning more than 25000.**
**(iv) Name of employee(s) who earn maximum salary in their organization.**
**Ans:**

**(i)** Employees getting more salary than their supervisor.
Again it will be solved by using self join concept.
SELECT E1.name, E2.name Supervisor F ROM EMPAS E1, EMP AS E2
WHERE E1.supervisor_no = E2.emp_no AND E1.salary>E2.salary;

**(ii)** Department name and total number of employees in each department who earn more than average salary for their department. This will use the correlated query concept.
SELECT dept_name, COUNT(*) FROM EMP E, DEPT
WHERE E.dept_code=DEPT.dept_cd AND salary>(SELECT AVG(salary)FROM EMP WHERE EMP.dept-code=E.dept_code) GROUP BY dept_name;

**(iii)** Departments having maximum employees earning more than 25000.
SELECT dept_name, COUNT(*) FROM EMP E,DEPT WHERE E.dept_code=DEPT.dept_cd AND salary>25000 GROUP BY dept_name *HAVING COUNT(*)>=ALL(SELECT COUNT(*)* FROM EMP GROUP BY dept_code);

**(iv)** Name of employee(s) who earn maximum salary in the organization.
SELECT name FROM EMP WHERE salary=(SELECT MAX (salary) FROM EMP);

Consider the insurance database, where the primary keys are underlined.

person (<u>ss#</u>, name, address)
car (<u>license</u>, year, model)
accident (<u>date</u>, <u>driver</u>, damage-amount)
owns (<u>ss#</u>, <u>license</u>)
log (<u>license</u>, <u>date</u>, <u>driver</u>)

Construct the following SQL queries for this relational database.

**(i)** Find the total number of people whose cars were involved in accidents in 1989.
**(ii)** Find the number of accidents in which the cars belonging to "John Smith" were involved.
**(iii)** Add a new customer to the database.
**(iv)** Add a new accident record for the Toyota belonging to "Jones" **(8)**

**Ans:(i)** Select count(driver) from accident where date is between '01-Jan-89'and '31-DEC-89';
**(ii)** Select count(*) from accident,person,owns,log where name='John Smith' and person.ss# = owns.ss# and owns.License= log.License and log.driver = accident.driver
**(iii)** insert into person values(1,'Raj', 'A10 RajNagar Gzb');
**(iv)** insert into car values('L20', 1982, 'Toyata');
insert into person values('S1', 'Jones', 'A2 Kavi Nagar Gzb');
insert into owns values ('S1', 'L20');
insert into accident values (Sysdate,"Jones",10000)

## Views

A view is object that gives the user a logical view of data from an underlying table or tables. You can restrict what users can view by allowing them to see only a few attributes/columns from a table.

Views may be created for the following reasons :

- simplifies queries
- can be queried as a base table
- provides data security

## Creation of Views

**Syntax :**

```
CREATE VIEW viewname as
SELECT columnname, cloumnname
FROM tablename
WHERE columnname = expression list ;
```

**Example :**

*Create view on Book table which contains two fields title, and author name.*

```
SQL> create view V_Book as
     select title,author_name
     from book;
View created.
SQL> select * from V_Book;
```

**Output :**

| Title | Author_name |
|---|---|
| Oracle | Arora |
| DBMS | Basu |
| DOS | Sinha |
| ADBMS | Basu |
| Unix | Kapoor |

## Selecting Data from a View

**Example :** *Display all the titles of books written by author 'Basu'.*

```
SQL> select title from V_Book
         where author_name = 'Basu';
```

**Output :**

| Title |
|---|
| DBMS |
| ADBMS |

## Updatable Views

Views can also be used for data manipulation i.e. the user can perform Insert, Update, and the Delete operations on the view. The views on which data manipulation can be done are called *Updateable Views*, views that do not allow data manipulation are called *Readonly Views*. When you give a view name in the Update, Insert, or Delete statement, the modifications to the data will be passed to the underlying table.

For the view to be updatable, it should meet following criteria :

- The view must be created on a single table.
- The primary key column of the table should be included in the view.
- Aggregate functions cannot be used in the select statement.
- The select statement used for creating a view should not include *Distinct*, *Group by*, or *Having* clause.
- The select statement used for creating a view should not include subqueries.
- It must not use constant, strings or value expressions like total/5.

## Destroying a View

A view can be dropped by using the DROP VIEW command.

**Syntax :**

```
DROP VIEW viewname ;
```

**Example :**

```
DROP VIEW V_Book ;
```

## Triggers

A trigger is a statement that the system executes automatically as a side effect of a modification to the database. To design a trigger mechanism, we must meet two requirements :

1. Specify when a trigger is to be executed. This is broken up into an *event* that causes the trigger to be checked and as *condition* that must be satisfied for trigger execution to proceed.
2. Specify the *actions* to be taken when the trigger executes.

The above model of triggers is referred to as the **event-condition-action** model for trigger.

The database stores triggers just as if they were regular data, so that they are persistent and are accessible to all database operations. Once we enter a trigger into the database, the database system takes on the responsibility of executing it whenever the specified event occurs and the corresponding condition is satisfied.

## Need for Triggers

Triggers are useful mechanism for alterting humans for starting certain tasks automatically when certain conditions are met. For example, suppose that, instead of allowing negative account balances, the bank deals with overdrafts by setting the account balance to zero, and creating a loan in the amount of the overdraft. The bank gives this loan a loan number identical to the account number of the overdrawn account. For this example, the condition for executing the trigger is an update to the account relation that results in a negative balance value. Suppose that Jones withdrawal of some money from an account made the account balance negative. Let t denote the account tuple with a negative balance value. The actions to be taken are :

- Insert a new tuple *s* in the loan relation with
  s [loan_no] = t [account_no]
  s [branch_name] = t [branch_name]
  s [amount] = – t [balance]
- Insert a new tuple u in the borrower relation with
  u [customer_name] = "Jones"
  u [loan_no] = t [account_no]
- Set t [balance] to 0.

As another example of the use of triggers, suppose a warehouse wishes to maintain a minimum inventory of each item ; when the inventory level of an item falls below the minimum level, an order should be placed automatically. This is how the business rule can be implemented by triggers : on an update of the inventory level of an item, the trigger should compare the level with the minimum inventory level for the item, and if the level is at or below the minimum, a new order is added to an orders relation.

## Triggers in SQL

SQL based database systems use triggers widely. Example of SQL trigger is given below :

```
Create trigger overdraft_trigger after update on account new row as nrow
for each row
when nrow.balance < 0
begin atomic
    insert into borrower
     (select customer_name, account_no
     from depositor
     where nrow.account_no = depositor. account_no) ;
    insert into loan values
     (nrow.account_no, nrow.branch_name, -nrow.balance) ;
    update account set balance = 0
    where account.account_no = nrow.account_no ;
end
```

This trigger definition specifies that the trigger is initiated after any update of the relation account is executed. The referencing new row as clause creates a variable *nrow*, which stores the value of an updated row after the update. Then, for each row, when statement checks the value of balance whether it is less than zero or not. If it is, then *customer_name* and *account_no* of *depositor* for given *account_no* inserted into borrower relation. A new tuple with values *nrow.account_no*, *nrow.branch_name* and – *nrow.balance* is inserted into loan relation, and finally balance of that *account_no* is set to zero in *account* relation.

## When Not to Use Triggers

Triggers should be written with great care, since a trigger error detected at run time causes the failure of the insert/delete/update statement that set off the trigger. The action of one trigger can set off another trigger. In worst case, this could lead to an infinite chain of triggering. For example, suppose an insert trigger on a relation has an action that causes another (new) insert on the same relation. The insert action then triggers yet another insert action, and so on. Database system typically limit the length of such chains of triggers, and consider longer chains of triggering an error.

## 2.14 Join

Join is a query in which data is retrieved from two or more table. A join matches data from two or more tables, based on the values of one or more columns in each table.

### ➢ Need for joins

In a database where the tables are normalized, one table may not give you all the information about a particular entity. For example, the Employee table gives only the department ID, so if you want to know the department name and the manager name for each employee, then you will have to get the information from the Employee and Department table. In other words, you will have to join two tables. So for comprehensive data analysis, you must assemble data from several tables. The relational model having made you to partition your data and put them in different tables for reducing data redundancy and improving data independence - relies on the join operation to enable you to perform ad hoc queries that will combine the related data which resides in more than one table.

Different types of Joins are :

- Inner Join
- Outer Join
- Natural Join

### 2.14.1 Inner Join

Inner Join returns the matching rows from the tables that are being joined.

Consider following two relations :

i) Employee (Emp_name, City)

ii) Employee_Salary (Emp_name, Department, Salary)

These two relations are shown in Fig 2.39 and Fig. 2.40.

| Employee | |
|---|---|
| **Emp_Name** | **City** |
| Hari | Pune |
| OM | Mumbai |
| Smith | Nashik |
| Jay | Solapur |

**Fig. 2.39 The Employee relation**

| Employee_Salary | | |
|---|---|---|
| **Emp_Name** | **Department** | **Salary** |
| Hari | Computer | 10000 |
| Om | IT | 7000 |
| Bill | Computer | 8000 |
| Jay | IT | 5000 |

**Fig. 2.40 The Employee_salary relation**

**Example 1 :**

select Employee.Emp_name, Employee_salary.Salary
from Employee **inner join** Employee_salary **on**
Employee.Emp_Name = Employee_salary.Emp_Name ;

Fig. 2.41 shows the result of above query

| Emp_Name | Salary |
|---|---|
| Hari | 10000 |
| Om | 7000 |
| Jay | 5000 |

**Fig. 2.41 The result of Employee inner join emploee_salary operation with selected fields from employee and employee_salary relation**

**Example 2 :**

select *
from Employee **inner join** Employee_Salary **on**
Employee.Emp_Name = Employee_Salary.Emp_Name ;

The result of above query is shown in Fig. 2.42.

| Emp_Name | City | Emp_Name | Department | Salary |
|---|---|---|---|---|
| Hari | Pune | Hari | Computer | 10000 |
| Om | Mumbai | Om | IT | 7000 |
| Jay | Solapur | Jay | IT | 5000 |

**Fig. 2.42 The result of Employee inner join Employee_salary operation with all fields from Employee and Employee relations**

As shown in Fig. 2.42 the result consists of the attributes of the left-hand-side relation followed by the attributes of the right-hand-side relation. Thus the Emp_name attribute appears twice in result first is from Employee relation and second is from Employee_salary relation.

### 2.14.2 Outer Join **[April-04, 4 Marks]**

When tables are joined using inner join, rows which contain matching values in the join predicate are returned. Sometimes, you may want both matching and non-matching rows returned for the tables that are being joined. This kind of an operation is known as an **outer join.**

An outer join is an extended form of the inner join. In this, the rows in one table having no matching rows in the other table will also appear in the result table with nulls.

### ➢ Types of outer join

The outer join can be any one of the following :

- Left outer
- Right outer
- Full outer

➢ **Join types and conditions**

Join operations take two relations and return another relation as the result. Each of the variants of the join operation in SQL consists of a *join type* and a *join condition*.

- ***Join type :*** It defines how tuples in each relation that do not match with any tuple in the other relation, are treated. Following Fig. 2.43 shows various allowed join types.

| Join Types |
|---|
| inner join |
| left outer join |
| right outer join |
| full outer join |

**Fig. 2.43 Join types**

- ***Join conditions :*** The join condition defines which tuples in the two relations match and what attributes are present in the result of the join.

Following Fig. 2.44 shows allowed join conditions.

| Join conditions |
|---|
| natural |
| on <predicate> |
| using (A1, A2, ...., An) |

**Fig. 2.44 Join conditions**

The use of join condition is mandatery for outer joins, but is optional for inner join (if it is omitted, a cartesian product results). Syntactically, the keyword **natural** appears before the join type, whereas the **on** and **using** conditions appear at the end of the join expression.

The join condition **using** (A1, A2, ......, An) is similar to the natural join condition, except that the join attributes are the attributes A1, A2, ....., An, rather than all attributes that are common to both relations. The attributes A1, A2, ...., An must consist of only attributes that are common to both relations, and they appear only once in the result of the join.

**Example :** Employee **full outer join** Emp_salary using (Emp_name)

#### 2.14.2.1 Left Outer Join

The left outer join returns matching rows from the tables being joined, and also non-matching rows from the left table in the result and places null values in the attributes that come from the right table.

**Example :**

```
select Employee.Emp_name, Salary
from Employee left outer join Employee_salary
on Employee.Emp_Name = Employee_salary.Emp_name ;
```

The result of above query is shown in Fig. 2.45.

| Emp_Name | Salary |
|---|---|
| Hari | 10000 |
| Om | 7000 |
| Jay | 5000 |
| Smith | null |

**Fig. 2.45 The result of Employee left outer join Employee_salary with selected fields from Employee and Employee_salary relations.**

➢ **Left outer join operation is computed as follows :**

First, compute the result of inner join as before. Then, for every tuple *t* in the left hand side relation Employee that does not match any tuple in the right-hand-side relation Employee_salary in the inner join, add a tuple *r* to the result of the join : The attributes of tuple *r* that are derived from the left-hand-side relation are filled with the from tuple *t*, and remaining attributes of *r* are filled with null values as shown in Fig. 2.45.

#### 2.14.2.2 Right Outer Join

The right outer join operation returns matching rows from the tables being joined, and also non-matching rows from the right table in the result and places null values in the attributes that comes from the left table.

**Example :**

```
Select Employee.Emp_Name, City, Salary from Employee right outer join
Employee_salary on Employee.Emp_name = Employee_salary.Emp_name ;
```

The result of preceding query is shown in Fig. 2.46.

| Emp_name | City | Salary |
|---|---|---|
| Hari | Pune | 10000 |
| Om | Mumbai | 7000 |
| Jay | Solapur | 5000 |
| Bill | null | 8000 |

**Fig. 2.46 The result of outer join operation with selected fields from Employee and Employee_salary relations**