**Syllabus: -**

**Introduction**

Structure Query Language (SQL) is a database query language used for storing and managing data in Relational DBMS. SQL was the first commercial language introduced for E.F Codd's Relational model of database. Today almost all RDBMS (MySQL, Oracle, Infomix, Sybase, MS Access) use SQL as the standard database query language. SQL is used to perform all types of data operations in RDBMS.

According to ANSI (American National Standards Institute), it is the standard language for relational database management systems. SQL statements are used to perform tasks such as update data on a database, or retrieve data from a database

Although most database systems use SQL, most of them also have their own additional proprietary extensions that are usually only used on their system. However, the standard SQL commands such as “Select”, “Insert”, “Update”, “Delete”, “Create”, and “Drop” can be used to accomplish almost everything that one needs to do with a database.

**SQL is used for the following:**

* Modifying database table and index structures;
* Adding, updating and deleting rows of data; and
* Retrieving subsets of information from within relational database management systems.

**Domain Type In SQL**

A domain determines the **type of data values** that are permitted for the attribute, and thus serves as an attribute constraint.

SQL provides a number of domain types to assign to attributes. The standard domain types include data values for characters, numerals, currency, dates, times, and Boolean entries (a logical value of either true or false). Most RDBMSs also accept the BLOB (binary large object) domain type, which stores binary objects such as graphics.

Choosing the correct domain type is critical to the accuracy of a database.  
It is important to choose the right domain type for an attribute. For example, zip codes appear as numbers to us, so there is the temptation to attach the numerical domain to them. However, zip codes in the north-eastern section of the United States begin with a zero. This would pose problems in a database since the numerical domain drops leading zeros.  
Do not think of time periods, such as “1st Quarter Profits,” “2nd Quarter Profits,” as distinct entities. Instead, create an entity called 'Profits' and use attributes with the date domain to distinguish time periods.

**SQL has the following domain types:**

* **char**(n) (or **character**(n)): fixed-length character string, with user-specified length.
* **varchar**(n) (or **character varying**): Variable length character strings, with user-specified maximum length n
* **int** or **integer**: an integer (length is machine-dependent).
* **Smallint** : a small integer (length is machine-dependent).
* **numeric(p,d)**: Fixed point number, with user-specified precision of p digits, with d digits to the right of decimal point. E.g., **numeric** (*3, 1*) allows 44.5 to be stored exactly but not 444.5.
* **real** or **double precision**: floating-point or double-precision floating-point numbers, with machine-dependent precision.
* **float**(n): floating-point, with user-specified precision of at least *n* digits.
* **date**: a calendar date, containing four-digit year, month, and day of the month.
* **time**: the time of the day in hours, minutes, and seconds.

**Schema Definition in SQL**

[SQL Schema](https://www.quest.com/community/blogs/b/database-management/posts/using-database-schemas-in-sql-server) as a logical collection of database objects such as tables, views, stored procedures, functions, indexes, triggers etc. A schema is associated with a username which is known as the schema owner, who is the owner of the logically related database objects. Schema always belong to a single database whereas a database can have single or multiple schemas.

Schemas are similar to separate namespaces or containers which stores database objects or used to handle database files. Schemas may be assigned security permissions, making them an effective method for distinguishing and defending database objects based on user access privileges. It increases the database's stability for security-related management.

The database object owner is a schema, and we define schema owners. We can have a single or multiple schema owner. It provides the following benefits:

* We can quickly transfer ownership of a SQL schema to another user
* We can share a schema among multiple users
* It allows you to move database objects among the schemas
* We get more control over database objects access and security

For example, a DB administrator provides your user’s name and password for a database. Your username is EMPLOYEE1. Let say you logged on to the DB and created a table named EMP\_TABLE.EMPLOYEE1 will be schema name for that table and it is the owner of the table as well.

Consider student is the sample database. if the database has two schema such as sales and modules. If want to access the object in the module schema you use format like Schema\_name.object\_name within the database like module.order .

**Built-in Schema**

SQL Server comes with a certain predefined schema that shares the same names as the built-in database functions and users. It persists primarily for backward compatibility. Here are a few examples of built-in schema:

* dbo
* guest
* sys
* INFORMATION\_SCHEMA

Objects in the schemas mentioned above cannot be dropped or removed. If you prefer to exclude schemas from the database, they will never appear on a new database.

**Creating a Schema**

**Syntax:**

CREATE SCHEMA schemaname [AUTHORIZATION ownername]

GO

**Example**

USE [db1]

GO

CREATE SCHEMA [schema\_one]

GO

CREATE TABLE schema\_one.student1 (name varchar(10), rollno int, age int, );

The above query will create a schema named as schema\_one and with user dbo as the owner of the schema.

Further, the CREATE command will create the table named student1 under the schema\_one schema.

**Data definition language**

**DDL commands:**

These commands are used to create, modify and remove database objects. They are auto com mitted i.e they do not need the COMMIT command to save the changes made to the database permanently. the change made are permanent after the successful execution of these statements.

DDL commands are as follows,  
1. CREATE   
2. DROP   
3. ALTER   
4. RENAME   
5. TRUNCATE

1. **CREATE COMMAND**

Create command is used for creating objects in the database.

**a). create database**

the sql create database command is used to create new sql database by allocating appropriate storage space for the new database in the physical database

syntax:

CREATE DATABASE <Database name>

Example :

Create database bank,

**b) create table**

The create table statement is used to create table object. In sql server fully qualified table name are in the format:[database].[schema].[table]

**Syntax**

CREATE TABLE <table\_name>  
(    column\_name1 datatype,  
     column\_name2 datatype,  
     .     .  
     .  
     column\_name\_n datatype  
);

Example:

Create table bank\_dept. branch( branched varchar(10),

location varchar(20),

branch\_manager varchar(20));

here bank\_dept. is the name of schema.

1. **DROP COMMAND**

Drop command is used to remove entire database objects from the database.It removes entire data structure from the database.

**Syntax:**  
DROP TABLE <table\_name>;  
OR  
DROP DATABASE <database\_name>;

Example

create table one

(

colnoe int,

col2 varchar(5));

drop table one

**3.ALTER COMMAND**

Alter command is used to alter or modify the structure of the database .It modifies an existing database object. Using this command, you can add additional column, drop existing column and even change the data type of columns.

1. **Adding the new column**

Syntax:

Alter table<tablename> add<column name> <data type(size)>;

create table student1

(

Roll\_no int,

name varchar(20),

grade varchar(20),

sub varchar(20),

mark varchar(10));

alter table student1

add address varchar(20);

1. **Removing column from table**

Syntax

Alter table<table name> drop column<column name>

alter table student1

drop column address;

**c)modify data type of column**

syntax

alter table <table name> alter column <column name data type (size)>;

alter table student1

alter column mark int;

1. **Remove or delete a table**

Syntax

Drop table <table name>

create table one

( colnoe int,

col2 varchar(5));

drop table one

with schema……..

create schema abc

create table abc.two

( colnoe int,

col2 varchar(5));

drop table abc.two

**4. RENAME COMMAND**

Rename command  is used to rename an object.

It renames a database table.

**Syntax:**

Exec sp\_rename ‘old\_table\_name’, ‘new\_table\_name’;  
  
**Example:**

create table three

( colnoe int,

col2 varchar(5));

exec sp\_rename 'three' , 'three\_new';

**5. TRUNCATE COMMAND**

Truncate command  is used to delete all the rows from the table permanently. It removes all the records from a table, including all spaces allocated for the records. This command is same as DELETE command, but TRUNCATE command does not generate any rollback data.

**Syntax:**

Truncate table<table name>  
  
**Example:**

create table three

( colnoe int,

col2 varchar(5));

insert into three values(5,'a');

select \* from three;

truncate table three;

select \* from three;

**Practical No: - 1**

1. **Create database bank**
2. **Create table customer with appropriate attributes custid, name, address,city,ph\_number and email and perform All the DDL operation given above**.

**Data manipulation language**

**DML commands:**

These commands are used to append, change or remove data from a table. COMMIT statement should be executed to make the changes made to the permanent to the database. DML describes the portion of SQL that allows you to manipulate or control your data. It includes the following…

1. SELECT   
2. INSERT  
3. DELETE  
4. UPDATE

1. INSERT (INSERT RECORD INTO THE TABLE)

The INSERT statement is used to insert a new record at a time into an existing table or relation.it has the general form

Syntax: INSERT INTO <table name> values(<value list>)

Example…….

Create table account

(

acno varchar(10),

Custid varchar(10),

Type varchar(10),

Branched varchar(10),

Balance numeric(10)

)

--------------------

Create table account

(acno varchar(10),

Custid varchar(10),

Type varchar(10),

Branched varchar(10),

Balance numeric(10))

* Insert values

Insert into account values('101',01,'saving','ktm',4000)

Insert into account values('102',02,'saving','Dhading',7000)

1. **SELECT (retrieving record from table)**

The select statement retrieves data from database and returns in the form of query rules.

Syntax: SELECT \* FROM <Table name>;

Example…..

select \* from account

1. **DELETE (Deleting records from table)**

Delete statement used to delete one or more rows from the table specified in the delete clause. the condition for deletion of rows has to be mentioned using where clause. You cannot delete a row which is related to another table using a foreign key constraint.in that case you have to delete the child rows first.

Syntax: delete from <table name> where <condition list>

Example…

delete from account where acno='101'

select \* from account

1. **UPDATE(Updating records into the Table)**

The update statement is used to modify one or more records of specific relation. The records to be modified are specified by a where clause and new value for the column is specified using set clause.

Syntax: update <table name> set <value\_list>where<condition>

Example….

select \* from account

update account set balance=balance+2000 where custid='2';

select \* from account

**Practical No: - 2**

1. **Create database bank**
2. **Create table transactions having attributes txnid, acno,type,**

**date-txn ,amount and perform All the DML operation given above.**

=======================================================

1. **The select clause**

The SELECT clause that begins each SELECT statement specifies the data items to be retrieved by the query. The items are usually specified by a select list, a list of select items separated by commas. Each select item in the list generates a single column of query results, in left-to-right order. A select item can be one of the following:

• A column name, identifying a column from the table(s) named in the FROM clause. When a column name appears as a select item, SQL simply takes the value of that column from each row of the database table and places it in the corresponding row of query results.

• A constant, specifying that the same constant value is to appear in every row of the query results.

• A SQL expression, indicating that SQL must calculate the value to be placed into the

query results, as specified by the expression.

Example:…..

Select \* from account;

SELECT acno,custid,type FROM account;

**2.Where clause**

The WHERE clause is used to specify the rows you want to retrieve. Here are some examples of simple queries that use the WHERE clause:

SELECT acno,custid,type,branched,balance from account

where balance > 6000;

SELECT acno,custid,type,branched,balance from account

where acno= '101';

**3.The From clause**

The FROM clause consists of the keyword FROM, followed by a list of table specifications separated by commas. Each table specification identifies a table or view containing data to be retrieved by the query. These tables are called the *source tables* of the query because they are the source of all of the data in the query results.

Example……

select \* from account

SELECT acno,custid,type,branched,balance,(balance-300) from account

SELECT acno,custid,type,branched,balance,(balance-300) as update\_balance from account

**4. THE RENAME OPERATION**

Rename command is used to rename an object. It changes the name of a user-created object in the current database. This object can be a table, index, column, alias data type etc.

**A. To renames a database table.**

**Syntax:**

Exec sp\_rename ‘old\_table\_name’, ‘new\_table\_name’;

**Example:**

create table three

( colnoe int,

col2 varchar(5));

exec sp\_rename 'three' , 'three\_new';

**B. Renaming a column**

The following example renames the grade column in the student table to class .

USE db1;

go

select \* from student

GO

EXEC sp\_rename 'student.grade', 'class', 'COLUMN';

GO

**C. Renaming an alias data type**

**=====================**

**Create unedified data type**

create type stud\_roll from int

create type stud\_name from varchar(10)

create type stud\_sub from varchar(10)

create type stud\_grade from int

create table student\_datatype

( rollno stud\_roll ,

name stud\_name,

subject stud\_sub,

grade stud\_grade);

select \* from student\_datatype

Now rename alias datatype

The following example renames the stud\_grade'alias data type to grade.

USE db1;

GO

EXEC sp\_rename N'stud\_grade', N'grade', N'USERDATATYPE';

GO

**D.To rename database**

1. System databases cannot be renamed.
2. The database name cannot be changed while other users are accessing the database.
3. In SQL Server, you can set a database in single user mode to close any open connections.

USE master;

GO

ALTER DATABASE db5new SET SINGLE\_USER WITH ROLLBACK IMMEDIATE

GO

ALTER DATABASE db5new MODIFY NAME = db5 ;

GO

ALTER DATABASE db5 SET MULTI\_USER

GO

**Practical No: - 3**

5. create databse dbone rename it

6. create table branch\_one

Having column branched, location, branch\_manager

And perform the rename operation on it(rename table,column,datatype)

**Tuple Variables**

Tuple variables Declares alternative names of the relation. In oracle it is describe as table aliases or correlation names*.*

create table employee

(

empid varchar(10),

nationalidno varchar(10),

contactid int,

loginid varchar(20),

managerid varchar(10),

title varchar(15)

)

insert into employee values('101','201',1,'emp@1','301','mkt asst');

insert into employee values('102','202',2,'emp@2',302,'tool mgr');

insert into employee values('103','203',3,'emp@3',303,'mkt mgr');

insert into employee values('104','204',4,'emp@4',304,'dgn eng');

insert into employee values('105','205',5,'emp@5',305,'prdn supvr');

insert into employee values('106','206',6,'emp@6',306,'prdn techn');

select e.empid , e.nationalidno , e.contactid

from employee e

where empid='105'

**String Operation**

SQL specifies strings by enclosing them in single quotes, for example, ’department’.

select 'Ishan Nepal '

**Pattern matching using the operator like**

When retrieving data, you can view selected rows that match a specific pattern. For example, you are asked to create a report that displays name of all student beginning with T. you can do this by using the LIKE keyword.

The LIKE keyword is used to search a string by using wildcards. Wildcards are special characters such as \*, %. These characters are used to match patterns.

**Following are the wild character used with LIKE keyword ….**

% Represents any string of zero or more characters

\_ Represents any single character

[] Represents any single character within specified range

[^] represents any single character not within the specified range

Patterns are case sensitive; that is, uppercase characters do not match lowercase

characters, or vice versa. To illustrate pattern matching, we consider the following

examples:

• ‘Hello%’ matches any string beginning with “Hello”.

• ‘%Comp%’ matches any string containing “Comp” as a substring, for

**Example**

1. **LIKE ‘ish%’ => all name that begin with “ish”**

select \* from employee

select firstname,lastname,salary

from employee

where firstname like 'ish%'

1. **LIKE ‘%i’ => all name that ends with “i”**

select \* from employee

select firstname,lastname,salary

from employee

where lastname like '%i'

1. **LIKE ‘%pa%’ => all name that have letter ‘pa’ in them**

select \* from employee

select firstname,lastname,salary

from employee

where lastname like '%ar%'

1. **LIKE ‘\_h%’ => all two letter names start with ‘h’**

select \* from employee

select firstname,lastname,salary

from employee

where lastname like '\_h%'

1. **LIKE ‘[td]%’ => all names that begins with “t”or “d”**

select \* from employee

select firstname,lastname,salary

from employee

where firstname like '[td]%'

1. **LIKE ‘[a-s]%’ => all names that begins with any letter from “a” through “s”**

select \* from employee

select firstname,lastname,salary

from employee

where firstname like '[a-s]%'

1. **LIKE ‘[a-s]%sh’ => all names that begins with any letter from “a” through “s” and end with “sh”**

select \* from employee

select firstname,lastname,salary

from employee

where firstname like '[a-s]%sh'

1. **LIKE ‘a[^c]%’ => all names that begins with “a” and not having “ c” as second letter .**

select \* from employee

select firstname,lastname,salary

from employee

where firstname like 'd[^c]%'

**SQL also permits a variety of functions on character strings, such as**

1. **Concatenating (using “+”)**

Concatenation is the operation where two strings are joined to make one string. Or example , the strings snow and balls can be concatenated to display the output snowball.

Select 'snow'+'ball'

Example :

create database db1

create table employee

(

empid varchar(10),

firstname varchar(20),

lastname varchar(20),

email varchar(30),

phone\_no numeric,

hiredate date,

salary numeric

)

Insert into employee values('101','Ram',Thapa,'tek@abc',9702,'2019',5000)

Insert into employee values('102','suraj','chaudhari','ss@abc',9802,'2015',6000)

Insert into employee values('103','manjil','maharjan','mm@abc',9666,'2014',7000)

Insert into employee values('104','dinesh','mallik','dn@abc',4567,'2013',8000)

Insert into employee values('105','amar','bohara','am@abc',9333,'2012',9000)

Select firstname + ' have last name = ' + lastname as 'lastname'

from employee

Select empid ,concat(firstname,lastname) as 'combine name'

from employee

1. **Extracting substrings(using SUBSTR function)**

It extracts a string of determined length , it has the general form as follows…..

SUBSTRING(string, start\_position, length)

Select SUBSTRING('Ramthapamagar',1 ,6) as name

**Select empid,SUBSTRING(firstname,1 ,3) as fname\_substring from employee**

1. **Finding the length of strings**

The LEN() function returns the length of a string.it has general form as follows

Syntax: LEN(string)

Select len('Ramthapamagar') as length

**Select empid,firstname,lastname,len(firstname) as fname\_length from employee**

1. **Converting strings to uppercase and lowercase**

select lower(‘RAM’)

select lower('RAM')

select upper('Ram')

Select empid,firstname,lastname,upper(firstname) as fname\_uppercase

from employee

1. **Removing spaces using trim**
2. Ltrim

Select ltrim (' Ram') as space\_remove\_l

1. Rtrim

Select rtrim ('Ram ') as space\_remove\_r

1. **Ascii**

**Returns the ascii code of the leftmost letter**

Select ascii('ABC') as ascii\_column

select empno, empname,dept, empname+desig as combination from employee

select empno,empname, SUBSTRING(desig,2,4) from employee

select empno,empname,desig,len(empname) as emplength from employee

select empno,empname,desig,len(empname) as emplength,UPPER(empname) as upperempname, ascii(empno) as asciivalue from employee

1. **Char**

**Returns the character equivalent of the ascii code**

Select char(65) as ascii\_code

1. **Charindex**

**Returns the string position of the specified pattern in the expression**

Select charindex ('e','hello' ) as position

select empno,empname,desig,CHARINDEX('e',empname) as indexline from employee

**Practical No: - 6**

**Q) Create student table having appropriate attributes and insert 10 dummy data and perform the matching operation using like keywords and use all above given char function on it(concayanate,substring,length,upper,lower,trim,ascii,char,charindex etc….)**

**Ordering the display of tuples**

You can use ORDER BY clause of the SELECT statement to display the data in a specific order. Data can be display in ascending or descending order of values in a given column.

The order of rows returned by a select statement is, by default, undefined. You can use the ORDER BY clause to sort the rows. The default ordering is ascending.

The syntax for using ORDER BY clause in the select statement is:

SELECT select\_list

From Table\_name

ORDER BY order\_by\_expression [ASC/DESC]

EXAMPLE:

USE db1

CREATE TABLE PRODUCTS

(

MFR\_ID CHAR(3),

PRODUCT\_ID CHAR(5),

DESCRIPTION VARCHAR(30),

PRICE DECIMAL(9,2),

QTY\_ON\_HAND INTEGER

);

INSERT INTO PRODUCTS (MFR\_ID, PRODUCT\_ID, DESCRIPTION, PRICE, QTY\_ON\_HAND)

VALUES ('ACI', '41007', 'Size 7 Widget', 225.00, 250);

INSERT INTO PRODUCTS (MFR\_ID, PRODUCT\_ID, DESCRIPTION, PRICE, QTY\_ON\_HAND)

VALUES ('ACI', '41001', 'Size 8 Widget', 100.00, 200);

INSERT INTO PRODUCTS (MFR\_ID, PRODUCT\_ID, DESCRIPTION, PRICE, QTY\_ON\_HAND)

VALUES ('ACI', '41002', 'Size 6 Widget', 200.00, 100);

INSERT INTO PRODUCTS (MFR\_ID, PRODUCT\_ID, DESCRIPTION, PRICE, QTY\_ON\_HAND)

VALUES ('ACI', '41003', 'Size 5 Widget', 300.00, 100);

INSERT INTO PRODUCTS (MFR\_ID, PRODUCT\_ID, DESCRIPTION, PRICE, QTY\_ON\_HAND)

VALUES ('ACI', '41004', 'Size 4 Widget', 900.00, 200);

INSERT INTO PRODUCTS (MFR\_ID, PRODUCT\_ID, DESCRIPTION, PRICE, QTY\_ON\_HAND)

VALUES ('ACI', '41005', 'Size 4 Widget', 800.00, 700);

select \* from products

select MFR\_ID, PRODUCT\_ID, DESCRIPTION, PRICE, QTY\_ON\_HAND

from products

order by PRODUCT\_ID

select MFR\_ID, PRODUCT\_ID, DESCRIPTION, PRICE, QTY\_ON\_HAND

from products

order by PRODUCT\_ID asc

select MFR\_ID, PRODUCT\_ID, DESCRIPTION, PRICE, QTY\_ON\_HAND

from products

order by PRODUCT\_ID desc

select MFR\_ID, PRODUCT\_ID, DESCRIPTION, PRICE, QTY\_ON\_HAND

from products

order by PRODUCT\_ID,PRICE

select MFR\_ID,PRICE , DESCRIPTION, PRODUCT\_ID, QTY\_ON\_HAND

from products

order by PRICE,PRODUCT\_ID

**Duplicate Tuples**

If a query includes the primary key of a table in its select list, then every row of query results will be unique. If the primary key is not included in the query results, duplicate rows can occur. For example, suppose you made this request: List the PRODUCT\_ID of product table*.*

INSERT INTO PRODUCTS (MFR\_ID, PRODUCT\_ID, DESCRIPTION, PRICE, QTY\_ON\_HAND)

VALUES ('ACI', '41005', 'Size 10 Widget', 100.00, 700);

INSERT INTO PRODUCTS (MFR\_ID, PRODUCT\_ID, DESCRIPTION, PRICE, QTY\_ON\_HAND)

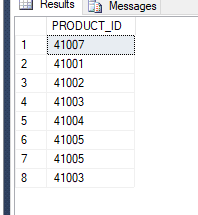
VALUES ('ACI', '41003', 'Size 4 Widget', 800.00, 700);

select \* from products

select \* from products

**select PRODUCT\_ID**

**from products**

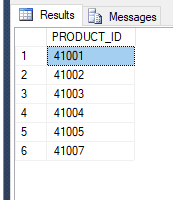


The query results have eight rows but two of them are exact duplicates of one another.

You can eliminate duplicate rows of query results by inserting the keyword DISTINCT in the SELECT statement just before the select list. Here is a version of the previous query that produces the results you want:

**select DISTINCT PRODUCT\_ID**

**from products**



**Practical No: - 7**

**Q) create table student with appropriate attributes**

**Order student data in ascending and descending order and eliminate the duplicate value in mark column by using DISTINCT keyword**

**Set Operation**

In an organization data related to different entities is stored in multiple tables. If you want to view data from the multiple tables together, you need to combine or compare the result sets. SQL server provides the various operations, such as UNION, EXCEPT AND INTERSECT to combine as well as compare the result set known as set opertion.

Set operator allows combining result from two or more select statements.it may look similar to sql join but there is big difference. SQL joins try to combine the table columns wise i.e horizontally or the resultant table contain more columns from the parent table on which join are made.

In case of set operation which are made on a particular set of column from both the tables, the resultant table happens to contain more row then the parent tables or set operation try to combine table vertically. As such they are known as vertical join .

You must follow some basic rules while using the UNION,EXCEPT, and INTERSECT operators to combine the result sets these rules are

1. The number and the sequence of the columns must be the same in all queries
2. The data types of the columns in all the queries must be compatable.

**Set operation UNION**

The UNION operator is used to combine data of the two or more queries into a single result set. The syntax of the UNION operator is

<query expression>

UNION [ALL]

<query expression>

UNION [ALL]

<query expression>

UNION [ALL]

[……n]

Example

create table employee\_setop

(

empid varchar(10),

custfname varchar(30),

custlname varchar(30),

email varchar(30),

phone\_no numeric,

hiredate date,

salary numeric

)

Insert into employee\_setop values('101','Ram', Thapa,'tek@abc',9702,'2019',5000)

Insert into employee\_setop values('102','suraj', 'cha','ss@abc',9802,'2015',6000)

Insert into employee\_setop values('103','manjil', 'mah','mm@abc',9666,'2014',7000)

Insert into employee\_setop values('104','dinesh', 'mal','dn@abc',4567,'2013',8000)

Insert into employee\_setop values('105','amar', 'boh','am@abc',9333,'2012',9000)

Insert into employee\_setop values('106','madan', 'gm','md@abc',9333,'2012',9000)

create table department

(

deptid varchar(10),

deptname varchar(20),

custfname varchar(30),

custlname varchar(30),

email varchar(30),

phone\_no numeric,

hiredate date,

)

Insert into department values('d101', 'markating','Ram', Thapa,'tek@abc',9702,'2019')

Insert into department values('d102', 'quality c','suraj', 'cha','ss@abc',9802,'2015')

Insert into department values('d103', 'HR','manjil', 'mah','mm@abc',9666,'2014')

Insert into department values('d104', 'Operation','dinesh', 'mal','dn@abc',4567,'2013')

Insert into department values('d105', 'Operation','amar', 'boh','am@abc',9333,'2012')

Insert into department values('d106', 'markating','vijay', 'shah','tek@abc',9702,'2019')

Insert into department values('d107', 'markating','manish', 'roka','tek@abc',9702,'2019')

Insert into department values('d108', 'markating','manish', 'magar','tek@abc',9702,'2019')

Select custfname from employee\_setop

UNION

Select custfname from department

Select custfname from employee\_setop

UNION ALL

Select custfname from department

**Set operation INTERSECT**

The INTERSECT operator returns the common rows after comparing two result sets. If the common rows are not found , then NULL value is returned. The syntax for using INTERSECT operator is

<query expression>

INTERSECT

<query expression>

Example

Select custfname from employee\_setop

INTERSECT

Select custfname from department

**Set operation EXCEPT**

The except operator compares two result sets and returns the data from first result set that is not found in the second result set. The syntax for using EXCEPT operator is

<query expression>

EXCEPT

<query expression>

Select custfname from employee\_setop

EXCEPT

Select custfname from department

**Practical No: - 6**

**Q) create table teacher and student with appropriate attributes**

**And perform the set operation like union, union all, intersect and except**

**Aggregate Functions**

The Aggregate functions, on execution , summirze the values of column or a group of columns, and produces a single value.

**Syntax:**

SELECT aggregate\_function ([ALL|DISTINCT] expression )

From table\_name

SQL offers following built-in aggregate functions:

• SUM() computes the total of a column.

• AVG() computes the average value in a column.

• MIN() finds the smallest value in a column.

• MAX() finds the largest value in a column.

• COUNT() counts the number of values in a column. (NULL values are not counted.)

create table employee\_agg

(

empid varchar(10),

custfname varchar(30),

custlname varchar(30),

email varchar(30),

phone\_no numeric,

hiredate date,

salary numeric

)

Insert into employee\_agg values('101','Ram', Thapa,'tek@abc',9702,'2019',5000)

Insert into employee\_agg values('102','suraj', 'cha','ss@abc',9802,'2015',6000)

Insert into employee\_agg values('103','manjil', 'mah','mm@abc',9666,'2014',7000)

Insert into employee\_agg values('104','dinesh', 'mal','dn@abc',4567,'2013',8000)

Insert into employee\_agg values('105','amar', 'boh','am@abc',9333,'2012',9000)

Insert into employee\_agg values('106','madan', 'gm','md@abc',9333,'2012',9000)

**SUM () :- Computing a Column Total**

The SUM() column function/aggregate function computes the sum of a column of data values. The data in the column must have a numeric type (such as integer, decimal, floating point, or money). The result of the SUM() function has the same basic data type as the data in the column, but the result may have a higher precision.

Example

SELECT SUM(salary) as total

FROM employee\_agg;

**AVG ():- Computing a Column Average**

The AVG() column function/aggregate function computes the average of a column of data values. As with the SUM() function, the data in the column must have a numeric type. Because the AVG() function adds the values in the column and then divides by the number of values, its result may have a different data type than that of the values in the column.

For example, if you apply the AVG() function to a column of integers, the result will be either a decimal or a floating point number, depending on the brand of DBMS you are using.

Here are some examples of the AVG() column function:

SELECT AVG(salary) as agerage\_salary

FROM employee\_agg;

**MIN() and MAX():- Finding Extreme Values (MIN and MAX)**

The MIN() and MAX() column functions find the smallest and largest values in a column, respectively. The data in the column can contain numeric, string, or date/time information. The result of the MIN() or MAX() function has exactly the same data type as the data in the column. Here are some examples that show the use of these column functions:

create table department2

(

deptid varchar(10),

deptname varchar(20),

custfname varchar(30),

custlname varchar(30),

email varchar(30),

phone\_no numeric,

hiredate date,

salary numeric

)

Insert into department2 values('d101', 'markating','Ram', moktan,'ram@abc',9702,'2019',2000)

Insert into department2 values('d102', 'quality c','suraj', 'cha','ss@abc',9802,'2015',2222)

Insert into department2 values('d103', 'HR','manjil', 'mah','mm@abc',9666,'2014',3000)

Insert into department2 values('d104', 'Operation','dinesh', 'mal','dn@abc',4567,'2013',8956)

Insert into department2 values('d105', 'Operation','amar', 'boh','am@abc',9333,'2012',7000)

Insert into department2 values('d106', 'markating','vijay', 'shah','tek@abc',9702,'2019',5623)

Insert into department2 values('d107', 'markating','manish', 'roka','tek@abc',9702,'2019',4444)

SELECT MIN(deptid), MAX(salary)

FROM department2 ;

SELECT MIN(deptid), MAX(100 \* salary)

FROM department2 ;

**COUNT ( ):- Counting Data Values**

The COUNT() column function counts the number of data values in a column. The data in the column can be of any type. The COUNT() function always returns an integer, regardless of the data type of the column. Here are some examples of queries that use the COUNT() column function:

SELECT COUNT(deptid)

FROM department2;

SELECT COUNT(deptid)

FROM department2

WHERE salary > 5000;

**Null Values**

A null value is an *indicator* that tells SQL (and the user) that the data is missing or not applicable. As a convenience, a missing piece of data is often said to have the value NULL. But the NULL value is not a real data value like 0, 473.83, or “Sam Clark.” Instead, it’s a signal, or a reminder, that the data value is missing or unknown.

If a field in a table is optional, it is possible to insert a new record or update a record without adding a value to this field. Then, the field will be saved with a NULL value.

A NULL value is different from a zero value or a field that contains spaces. A field with a NULL value is one that has been left blank during record creation!

To retrieve the rows with NULL values in the column, Transact-SQL includes the operator feature IS NULL.

To retrieve only the rows with no NULL values in the column Transact-SQL includes the operator feature IS NOT NULL.

Create table student

(

Name varchar(20),

Rollno int,

Subject varchar(30),

Address varchar(40),

Mobile\_no numeric null

)

Insert into student(name,rollno,subject,address)

values('Ram',10,'english','Dhading')

Insert into student(name,rollno,subject,address)

values('Amar',11,'math','mahendranagar')

Insert into student(name,rollno,subject,mobile\_no)

values('Amar',11,'math',9702101432)

select \* from student

select \* from student

WHERE Mobile\_no IS NULL;

Create table student1

(

Name varchar(20),

Rollno int,

Subject varchar(30),

Address varchar(40),

Mobile\_no numeric

)

Insert into student1(name,rollno,subject,address)

values('Ram',10,'english','Dhading')

Insert into student1(name,rollno,subject,address)

values('Amar',11,'math','mahendranagar')

Insert into student1(name,rollno,subject,mobile\_no)

values('Amar',11,'math',9702101432)

select \* from student1

select \* from student1

WHERE Mobile\_no IS NULL;

SELECT name,rollno,subject,address,Mobile\_no

FROM student1

WHERE Mobile\_no IS NOT NULL

Create table student2

(

Name varchar(20),

Rollno int,

Subject varchar(30),

Address varchar(40),

Mobile\_no numeric not null

)

Insert into student2(name,rollno,subject,address)

values('Ram',10,'english','Dhading')

Insert into student2(name,rollno,subject,address,Mobile\_no)

values('Ram',10,'english','Dhading',123)

select \* from student2

WHERE Mobile\_no IS NULL;

**Practical no:7**

**Q:- create a table employee\_agg having attributes empid ,custfname ,custlname ,email,**

**phone\_no,hiredate , and salary And perform the operation of aggregate function like SUM() , AVG(), MIN(), MAX() and COUNT() on salary attributes.**

**Nested Subqueries**

**Subqueries:**

A subquery is an SQL statement that is used within another SQL statement or query within a query. Subqueries are nested inside the WHERE or HAVING clause of the SELECT, INSERT, UPDATE and DELETE statements.

The query that represents the parent query is called outer query, and the query that represent the subquery is called inner query. The database engine executes the inner query first and returns the result to the outer query to calculate the result set.

For example,

create table employee\_detail

(

empid varchar(10),

empname varchar(30),

designation varchar(30),

salary numeric,

deptno varchar(20)

)

Insert into employee\_detail values('101','Ram','executive',5000,'D101')

Insert into employee\_detail values('102','suraj','manager',6000,'D102')

Insert into employee\_detail values('103','manjil','clerk',7000,'D103')

Insert into employee\_detail values('104','dinesh','clerk ',8000,'D104')

Insert into employee\_detail values('105','amar','executive', 9000,'D105')

Insert into employee\_detail values('106','madan','assistent',3000,'D106')

Insert into employee\_detail values('10','aman','clerk',8000,'D107')

Insert into employee\_detail values('109','shiv','assistent',5000,'D104')

Insert into employee\_detail values('107','raj','clerk',8000,'D101')

Insert into employee\_detail values('108','Aryan','dgm',8000,'D103')

**Q) find all the employees who have the same designation as manjil.**

select \* from employee\_detail

Select designation from employee\_detail

Where empname='manjil'

Select \* from employee\_detail

Where designation='clerk'

Select \* from employee\_detail

Where designation=(Select designation from employee\_detail

Where empname='manjil')

**IN /NOT IN operator**

Sometimes, you want to retrieve data after specifying a set of values to check whether the specified value matches any data of the table. This type of operation is performed by using IN and NOT IN keywords. The IN operator allows you to specify multiple values in a WHERE clause. It is a shorthand for multiple OR conditions.

**Syntax:**

SELECT column\_list

From table\_name

Where expression list\_operator(‘value list’)

Example:

Select empname,designation,salary

from employee\_detail

Where salary in (4000,5000,6000,8000)

Select empname,designation,salary

from employee\_detail

Where salary not in (4000,5000,6000,8000)

**Set Membership**

SQL allows testing tuples for membership in a relation. The **in** connective tests for set membership, where the set is a collection of values produced by a **select** clause. The **not in** connective tests for the absence of set membership.

If subquery returns more than one value, you might need to match a column value with any of the values in the list return by the inner query.to perform this task, you need to use IN keyword.

**Example:**

**Q) Write a query to display the employee’s name, designation and salary for all employees from a table employee\_detail with any employee whose name start from T.**

Select empname,designation,salary

from employee\_detail

Where empname in ( SELECT empname

FROM employee\_detail

WHERE empname LIKE 't%' )

=========================================================

**Q) Write a query to display the employee’s name from a table employee\_detail** **with any employee who have minim salary and should be grouped by department number**.

SELECT empname

from employee\_detail

WHERE salary IN

(

SELECT MIN(salary)

from employee\_detail

GROUP BY deptno

)

======================================================== **Q) Write a query to display the employee’s detail from a table employee\_detail** **with any employee who have designation clerk and dgm**.

SELECT \* from employee\_detail

WHERE designation IN

(

SELECT designation

from employee\_detail

WHERE designation ='clerk' and designation ='dgm'

)

=========================================================

**Q) Write a query to display the employee’s name from a table employee\_detail** **with any employee who have designation clerk and salary having greater then 5000**.

SELECT \* from employee\_detail

WHERE designation IN ( SELECT designation

from employee\_detail

WHERE designation='clerk'

AND salary>5000 );

**Q2) Write a query to display the name of student who live in Butwal**

create table student4

(

rollno int,

fname varchar(30),

lname varchar(30),

school varchar(30),

addr varchar(30),

phno numeric

)

Insert into student4 values(1,'Ram','gm', 'new horizen','butwal',9702101432)

Insert into student4 values(2,'sam','sh', 'appize','butwal',9802101432)

Insert into student4 values(3,'raj','rk','sunrise','ktm',9802101432)

Insert into student4 values(4,'amar','bh', 'new horizen','mahendranagar',9802501432)

select \* from student4

SELECT fname ,lname

from student4 WHERE addr IN ( SELECT addr

from student4

where addr='butwal')

**Set Comparison**

The nested subquery have ability to compare sets by using the comparison operator like = (equal to), > (greater then), >= (greater then or equal to), < (less then), <= (less than or equal to), <> (not equal to).It can compare a value to a set of values.

create table department3

(

depno varchar(10),

deptname varchar(20),

custfname varchar(30),

custlname varchar(30),

email varchar(30),

phone\_no numeric,

hiredate date,

salary numeric

)

Insert into department2 values('d101', 'markating','Ram', Thapa,'tek@abc',9702,'2019',2000)

Insert into department2 values('d102', 'quality c','suraj', 'cha','ss@abc',9802,'2015',2222)

Insert into department2 values('d103', 'HR','manjil', 'mah','mm@abc',9666,'2014',3000)

Insert into department2 values('d104', 'Operation','dinesh', 'mal','dn@abc',4567,'2013',8956)

Insert into department2 values('d105', 'Operation','amar', 'boh','am@abc',9333,'2012',7000)

Insert into department2 values('d106', 'markating','vijay', 'shah','tek@abc',9702,'2019',5623)

Insert into department2 values('d107', 'markating','manish', 'roka','tek@abc',9702,'2019',4444)

Insert into department2 values('d107', 'markating','jhon', 'viju','jh@abc',9802,'2019',5000)

**Q) display the employees name and salary whose deptno is equal to department id in department3 table whose deptid is d103**

SELECT empname,salary

from employee\_detail

WHERE deptn= ( SELECT deptid

from department3

WHERE deptid ='d103'

)

**Q) display empname,salary, deptno from employee\_detail whose salary is equal to salary in department2 table whose deptid is d105**

SELECT empname,salary,deptno

from employee\_detail

WHERE salary= (

SELECT salary

from department2

WHERE deptid ='d105'

)

**Q) display employee name and salary from employee\_detail table whose salary is greater than that of the employee in department3 table whose name is jhon**

SELECT empname,salary

from employee\_detail

WHERE salary > ( SELECT salary

from department3

WHERE custfname ='jhon' )

**Q) display employee name and salary from employee\_detail table whose salary is not greater than that of the employee in department3 table whose name is jhon**

SELECT \*

from employee\_detail

WHERE salary <> ( SELECT salary

from department2

WHERE custfname ='jhon' )

While using subqueries you can use <,>,<=,>=,<> comparison operator to create a condition that checks the value returned by the sub query. When subquery returns more than one value, you might need to apply operators to all the values returned by the subquery.

To perform this task, you can modify the comparison operator in the subquery. SQL server provides the ALL and ANY keywords that can be used to modify the existing comparison operator.

The following list shows the operators that can be used with the all and any keywords.

1. >ALL => Means greater then the maximum value in the list.

Eg. >ALL(10,20,30) means greater then 30

1. >ANY => Means greater then the minimum value in the list.

Eg. >ANY(10,20,30) means greater then 10

1. =ALL => Eg. =ALL(10,20,30) means equal to either 10 or 20 or

30

1. <>ANY => Means not equal to any value in the list.

Eg. <>ANY(10,20,30) means not equal to 10, or 20, or 30

1. <>ALL => Means not equal to all the value in the list.

Eg. <>ALL(10,20,30) means not equal to 10, and , and 20,

And 30

**Test for empty relation**

You can use a subquery to check if a set of records exists. For this, you need to use the EXISTS clause with a subquery. The EXISTS keyword always returns a TRUE or False value. The **exists** construct returns the value **true** if the argument subquery is nonempty.

The EXISTS clause checks for the existence of rows according to the condition specified in the inner query and passes the existence status to the outer query.

**The syntax for using the EXIT keyword in the SELECT statement is:**

**SELECT column-list**

**FROM table\_name**

**WHERE EXISTS (**

**SELECT column FROM table\_name**

**WHERE [conditional expression]**

**)**

==============================================================

**Create table office**

**(**

**City varchar(30),**

**Region varchar(30),**

**Target float,**

**Sales float,**

**Office\_no int,**

**)**

INSERT INTO OFFICE (CITY, REGION, TARGET, SALES, Office\_no )

VALUES ('butwal', 'Western', 275000, 0.00, 01);

INSERT INTO OFFICE (CITY, REGION, TARGET, SALES, Office\_no )

VALUES ('ktm', 'estern', 200000, 0.00, 2);

INSERT INTO OFFICE (CITY, REGION, TARGET, SALES, Office\_no )

VALUES ('dang', 'mid western', 300000, 0.00, 3);

INSERT INTO OFFICE (CITY, REGION, TARGET, SALES, Office\_no )

VALUES ('mahendranagar', ' far Western', 100000.00, 0.00, 4);

INSERT INTO OFFICE (CITY, REGION, TARGET, SALES, Office\_no )

VALUES ('ktm', ' estern', 100000, 12, 5);

INSERT INTO OFFICE (CITY, REGION, TARGET, SALES, Office\_no )

VALUES ('ktm', ' estern', 100000, 19, 6);

select \* from office

SELECT \* FROM office

WHERE EXISTS ( SELECT CITY

FROM office

WHERE city = 'ktm'

)

SELECT \* FROM office

WHERE NOT EXISTS ( SELECT CITY

FROM office

WHERE city = 'ktm'

)

**Test for the absence of duplicate tuples**

SQL includes a Boolean function for testing whether a subquery has duplicate tuples in its result. The **unique** construct returns the value **true** if the argument subquery contains no duplicate tuples. UNIQUE predicate evaluates to True only if two rows returned by the subquery are identical. In other words, the UNIQUE predicate evaluates to True only if all the rows that its subquery returns are unique.

create table student5

(

rollno int,

teacherid varchar(20),

fname varchar(30),

lname varchar(30),

school varchar(30),

addr varchar(30),

phno numeric

)

Insert into student5 values(1, 't101','Ram','gm', 'new horizen','butwal',9702101432)

Insert into student5 values(2, 't101','sam','sh', 'appize','butwal',9802101432)

Insert into student5 values(3, 't102','raj','rk','sunrise','ktm',9802101432)

Insert into student5 values(4, 't102','amar','bh', 'new horizen','mahendranagar',9802501432)

Insert into student5(rollno,teacherid,fname,lname,school,addr)

values(5, 't103','rajesh','bh', 'new horizen','mahendranagar')

create table teacher

(

teacherid varchar(20),

teacher\_name varchar(30),

addr varchar(30),

subject varchar(30),

phno numeric

)

Insert into teacher values( 't101','vijay', 'mumbai', 'english',9702101432)

Insert into teacher values('t101','vipul', 'delhi', 'comp',9802101432)

Insert into teacher values( 't102','manoj', 'amd', 'math',9802101432)

Insert into teacher values( 't102','anthony','raj','science',802501432)

select T.phno

from teacher as T

where 1<=(select R.phno

from student5 as R

where R.fname='Ram'

)

select teacher\_name,addr

from teacher

where 1<=( select phno

from student5

where fname='amar'

)

select teacher\_name,addr,phno

from teacher

where 1<=(

select rollno

from student5,teacher

where student5.phno=teacher.phno

)

Note that if a phno is not available of student having name Ram, the subquery would return an empty result, and the **unique** predicate would evaluate to true on the empty set.

select T.phno

from teacher as T

where 1<=(select R.phno

from student5 as R

where R.fname='rajesh'

)

**Views:**

A view is a **virtual table** in the database whose contents are defined by a query. To the database user, the view appears just like a real table, with a set of named columns and rows of data. But unlike a real table, a view does not contain any data, but it derives its data from the underlying tables. SQL creates the illusion of the view by giving the view a name like a table name and storing the definition of the view in the database.

**Creating a View**

The CREATE VIEW statement, is used to create a view. This statement assigns a name to the view and specifies the query that defines the view. To create the view successfully, you must have permission to access all of the tables referenced in the query. In some DBMSs you must also have permission to create views.

The CREATE VIEW statement can optionally assign a name to each column in the newly created view. If a list of column names is specified, it must have the same number of items as the number of columns produced by the query. Note that only the column names are specified; the data type, length, and other characteristics of each column are derived from the definition of the columns in the source tables.

If the list of column names is omitted from the CREATE VIEW statement, each column in the view takes the name of the corresponding column in the query. The list of column names must be specified if the query produces two columns with identical names and in some DBMS products, if the query

includes calculated columns.

**Syntax**

**CREATE VIEW view\_name AS  
 SELECT column1, column2, ...  
 FROM table\_name  
 WHERE condition;**

**Example**

create table sales\_detail

(

empid varchar(10),

empname varchar(30),

age date,

quota numeric,

sales numeric,

Office\_no int,

)

Insert into sales\_detail values('101','Ram','2018',500000,20000,1)

Insert into sales\_detail values('102','suraj','2016',440000,20000,2)

Insert into sales\_detail values('103','manjil','2014',700000,5000,3)

Insert into sales\_detail values('104','dinesh','2015',80000,70000,4)

Insert into sales\_detail values('105','amar','2016',900000,10000,5)

Insert into sales\_detail values('106','madan','2014',300000,20000,6)

Insert into sales\_detail values('110','aman','2017',800000,6000,7)

Insert into sales\_detail values('109','shiv','2018',500000,8800,8)

======================================================

Create table office1

(

City varchar(30),

Region varchar(30),

Target float,

Sales float,

Office\_no int

)

INSERT INTO OFFICE1 (CITY, REGION, TARGET, SALES, Office\_no )

VALUES ('butwal', 'Western', 275000, 0.00, 01);

INSERT INTO OFFICE1 (CITY, REGION, TARGET, SALES, Office\_no )

VALUES ('ktm', 'estern', 200000, 0.00, 2);

INSERT INTO OFFICE (CITY, REGION, TARGET, SALES, Office\_no )

VALUES ('dharan', 'estern', 200000, 0.00, 2);

INSERT INTO OFFICE1 (CITY, REGION, TARGET, SALES, Office\_no )

VALUES ('dang', 'mid western', 300000, 0.00, 3);

INSERT INTO OFFICE1 (CITY, REGION, TARGET, SALES, Office\_no )

VALUES ('mahendranagar', ' far Western', 100000.00, 0.00, 4);

INSERT INTO OFFICE1 (CITY, REGION, TARGET, SALES, Office\_no )

VALUES ('ktm', ' estern', 100000, 12, 5);

INSERT INTO OFFICE1 (CITY, REGION, TARGET, SALES, Office\_no )

VALUES ('ktm', ' estern', 100000, 19, 6);

========================================================================

CREATE VIEW vw\_sales\_detail

AS

SELECT empid,empname,age,quota,sales,office\_no

From sales\_detail

Select \* from vw\_sales\_detail

==================================================================

CREATE VIEW vw\_office

AS

SELECT city,region,target,sales,office\_no

From office1

Select \* from vw\_office

Select city,region from vwoffice

=================================================================

CREATE VIEW vw\_sales\_detail2

AS

SELECT \* FROM sales\_detail

WHERE Office\_no IN (1, 2, 3)

Select \* from vw\_sales\_detail2

==================================================================

CREATE VIEW vwEASTOFFICES

AS

SELECT \* FROM OFFICE1

WHERE REGION = ' estern ';

Select \* from vwEASTOFFICES

==================================================================

CREATE VIEW vwsl

As

SELECT empid, empname, sales

from sales\_detail

WHERE sales > ( SELECT sales

from sales\_detail

WHERE empname ='Ram' )

Select \* from vwsl

create table student4

(

rollno int,

fname varchar(30),

lname varchar(30),

school varchar(30),

sub varchar(20),

mark numeric,

addr varchar(30),

phno numeric

)

Insert into student4 values(1,'Ram','gm', 'new horizen', 'eng',55,'butwal',9702101432)

Insert into student4 values(2,'sam','sh', 'appize', 'math',70,'butwal',9802101432)

Insert into student4 values(3,'raj','rk','sunrise', 'science',88,'ktm',9802101432)

Insert into student4 values(4,'amar','bh', 'new horizen', 'social',75,'mahendranagar',9802501432)

select \* from student4

CREATE VIEW vw\_student4

AS

SELECT fname ,lname

from student4

WHERE addr IN ( SELECT addr

from student4

where addr='butwal')

**Update of a View: -**

View do not maintain a separate copy of the data, but only display data present in the base table. therefore, you cannot modify the base table by modifying the data in the view. Following restriction exist while inserting, updating or deleting data through view

1. You cannot modify data in the view if the modification affects more than one underlaying table. However, you can modify data in a view if the modification affects one table at a time.

Create table office

(

City varchar(30),

Region varchar(30),

Target float,

Sales float,

Office\_no int,

)

INSERT INTO OFFICE (CITY, REGION, TARGET, SALES, Office\_no )

VALUES ('butwal', 'Western', 275000, 0.00, 01);

INSERT INTO OFFICE (CITY, REGION, TARGET, SALES, Office\_no )

VALUES ('ktm', 'estern', 200000, 0.00, 2);

INSERT INTO OFFICE (CITY, REGION, TARGET, SALES, Office\_no )

VALUES ('dang', 'mid western', 300000, 0.00, 3);

INSERT INTO OFFICE (CITY, REGION, TARGET, SALES, Office\_no )

VALUES ('mahendranagar', ' far Western', 100000.00, 0.00, 4);

INSERT INTO OFFICE (CITY, REGION, TARGET, SALES, Office\_no )

VALUES ('ktm', ' estern', 100000, 12, 5);

INSERT INTO OFFICE (CITY, REGION, TARGET, SALES, Office\_no )

VALUES ('ktm', ' estern', 100000, 19, 6);

CREATE VIEW vwSales\_office

AS

SELECT City,Region,Target,Sales,Office\_no

From OFFICE

select \* from vwSales\_office

select \* from office1

select \* from sales\_detail

update vwSales\_office

set city='dharan'

where Office\_no=1

select \* from vwSales\_office

update vwSales\_office

set Region='estern'

where Office\_no=1;

select \* from vwSales\_office

update vwSales\_office

set city='kk'

where Office\_no=2;

select \* from vwSales\_office

2) you cannot change a column that is the result of a calculation, such as computed column or an aggregate function

create view sss

AS

SELECT City,Region,Target,Sales,Office\_no, Sales+1000 as updated\_sale

From office

select \* from sss

drop view sss

update sss

set updated\_sale=5000

where Office\_no=1;

**Altering View:-**

create view vwoff

AS

SELECT City,Region,Target,Sales,Office\_no, Sales+1000 as updated\_sale

From office

alter view vwoff

AS

SELECT City,Region,Target,Sales,Office\_no

From office

**Renaming View:-**

sp\_rename vwSales\_office,vwSales\_office\_new

select \* from vwSales\_office\_new

**dropping View:-**

drop view sss

**Joined Relations: Join types and Conditions**

**Joins:**

When an SQL query executes, it returns a result set. A result set is a set of rows retrieved from a table. As a database developer, you may need to retrieve data from more than one table and display it in a single result set. In such a case, different columns in the result set can obtain data from different tables.

To retrieve data from multiple tables, SQL server allows you to apply joins. Joins allow you to view data from related tables in a single result set. You can join more then one table based on a common attribute.

**Types of Joins:**

1. **Inner Joins**

An inner join retrieves records from multiple tables after comparing values present in a common column.

When inner join is applied, only the rows with values satisfying the join condition in the common column are displayed. The rows in the both the tables that do not satisfy the join condition are not displayed.

**Example:**

create table employee\_master53

(

empno varchar(5),

fname varchar(8),

lname varchar(8),

dept varchar(8),

desig varchar(9),

branchno varchar(5)

);

insert into employee\_master53 values('e001','Ram',Thapa,'oper','asistance','b001')

insert into employee\_master53 values('e002','sadik','patankar','develp','manager','b002')

insert into employee\_master53 values('e003','gorakh','talekar','qualityc','analyst','b003');

insert into employee\_master53 values('e004','vipul','navadkar','oper','analyst','b004');

insert into employee\_master53 values('e006','ishani','agrawal','oper','analyst','b006');

select \* from employee\_master53

create table branch\_master53

(

name varchar(10),

location varchar(30),

branchno varchar(5)

);

insert into branch\_master53 values('Ram','millinium buss park','b001');

insert into branch\_master53 values('sadik', 'mumbai forth','b002');

insert into branch\_master53 values('gorakh', 'BKC','b003');

insert into branch\_master53 values('tejus', 'lower parel','b005');

insert into branch\_master53 values('piyush', 'masjid van','b007');

select \* from employee\_master53

select \* from branch\_master53

select e.empno,e.fname,e.dept,e.desig,d.location

from employee\_master53 as e JOIN branch\_master53 as d

ON e.branchno=d.branchno

1. **Outer Joins**

In comparison to an inner join, an outer join displays the result set containing all the rows from one table and matching rows from the other table.

For example, if you create an outer join on a table A and table B , it will show you all the records of table A and only those records from table B for which the condition on the common column holds true .

**An outer join is an following types…….**

* **Left outer join**
* **Right outer join**
* **Full outer join**

**Left outer join:**

A left outer join returns all rows from the table specified on the left side of the LEFT OUTER JOIN and matching rows from the table specified on the right side.

It displays NULL for the columns of the table specified on right side where it does not find any matching records.

select \* from employee\_master53

select \* from branch\_master53

select e.fname,e.dept, d.location

from employee\_master53 as e LEFT OUTER JOIN branch\_master53 as d

ON e.branchno=d.branchno

**Right outer join:**

A right outer join returns all rows from the table specified on the right side of the RIGHT OUTER JOIN and matching rows from the table specified on the left side.

It displays NULL for the columns of the table specified on left side where it does not find any matching records.

select \* from employee\_master53

select \* from branch\_master53

select e.fname,e.dept, d.location

from employee\_master53 as e RIGHT OUTER JOIN branch\_master53 as d

ON e.branchno=d.branchno

**Full outer join:**

A full outer join is a combination of right outer and left outer join. This join returns all the matching and non-matching rows from both the tables. However, the matching records are displayed only once. In case of non-matching rows, a NULL value is displayed for the columns for which data is not available.

It displays NULL for the columns of the table specified on left side where it does not find any matching records.

select \* from employee\_master53

select \* from branch\_master53

select e.fname,e.dept, d.location

from employee\_master53 as e FULL OUTER JOIN branch\_master53 as d

ON e.branchno=d.branchno

1. **Cross join:**

Cross join also known as the cartesian product. It joins each row of one table with each row of the other table. The number of rows in the result set is equal to the number of rows in first table multiplied by the number of rows in the second table.

**Example:**

select \* from employee\_master53

select \* from branch\_master53

select e.fname,e.dept, d.location,d. branchno

from employee\_master53 as e CROSS JOIN branch\_master53 as d

**4) Equi join:**

Equi join is type of INNER JOIN where the comparison operator is equality. However, in equi join, only the equality operator is used to specify the join condition, whereas you can use conditional operator to specify the join in inner join.

select \* from employee\_master53

select \* from branch\_master53

select \* from employee\_master53 as e JOIN branch\_master53 as d

ON e.branchno=d.branchno

**// below is inner join**

select \* from employee\_master53

select \* from branch\_master53

select e.empno,e.fname,e.dept, d.location,d.branchno

from employee\_master53 as e JOIN branch\_master53 as d

ON e.branchno>d.branchno

**5) Self join:**

In self join, a table is joined with itself. As a result, one row in a table corelates with another row in a same table. In a self-join, a table name is used twice in the query. Therefore, to differentiate between two instances of a single table, the table is given two alias names.

create table employee\_self

(

empid int,

title varchar(40),

managerID numeric

);

insert into employee\_self values(1,'production technician-wc60',16)

insert into employee\_self values(2,'marketing assistent',6)

insert into employee\_self values(3,'Engineering manager',12)

insert into employee\_self values(4,'senior tool manager',3)

insert into employee\_self values(5,'tool designer',263)

insert into employee\_self values(6,'markating manager',109)

insert into employee\_self values(7,' production supervisor -wc60',21)

insert into employee\_self values(8,'production technician-wc10',185)

insert into employee\_self values(9,'Design engineer',3)

insert into employee\_self values(10,'production technician-wc10',185)

select \* from employee\_self

select emp.empid,emp.title as emp\_designation,emp.managerid,mgr.title as manager\_designation

from employee\_self as emp, employee\_self as mgr

where emp.managerid=mgr.empid

create table employee\_self

(

empid int,

title varchar(40),

managerID numeric

);

**Embedded SQL**

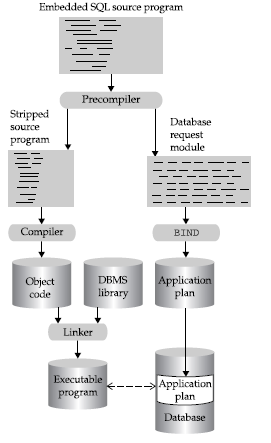
SQL is a language and can be used programmatically, but it would be incorrect to call SQL a programming language. SQL lacks even the most primitive features of real programming languages. It has no provision for declaring variables, no GOTO statement, no IF statement for testing conditions, no FOR, DO, or WHILE statements to construct loops, no block structure, and so on. SQL is a database *sublanguage* that handles special-purpose database management tasks.

So , to write a program that accesses a database, you must start with a conventional programming language such as COBOL, PL/I, FORTRAN, Pascal, C, C++, or Java, or a scripting language such as Perl, PHP, or Ruby, and then add SQL to the program.

In embedded SQL, SQL statements are embedded directly into the

program’s source code, intermixed with the other programming language statements. Special embedded SQL statements are used to retrieve data into the program. A special SQL precompiler accepts the combined source code and, along with other programming tools, converts it into an executable program.

An embedded SQL program contains a mix of SQL and programming language statements, so it can’t be submitted directly to a compiler for the programming language. Instead, it moves through a multistep development process, shown in Figure below.



**FIGURE: -The embedded SQL development process**

1. The embedded SQL source program is submitted to the SQL precompiler, a programming tool. The precompiler scans the program, finds the embedded SQL statements, and processes them. A different precompiler is required for each programming language supported by the DBMS. Commercial SQL products typically offer precompilers for one or more languages, including C, Pascal, COBOL, FORTRAN, Ada, PL/I, RPG, and various assembly languages.

2. The precompiler produces two files as its output. The first file is the source program, stripped of its embedded SQL statements. In their place, the precompiler substitutes calls to the private DBMS routines that provide the runtime link between the program and the DBMS. Typically, the names and calling sequences of these routines are known only to the precompiler and the DBMS; they are not a public interface to the DBMS. The second file is a copy of all the embedded SQL statements used in the program. This file is sometimes called a *database request module* (DBRM).

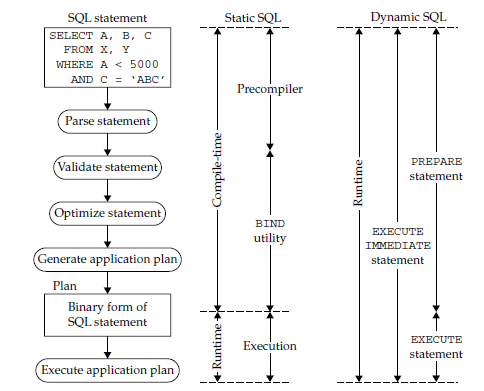
3. The source file output from the precompiler is submitted to the standard compiler for the host programming language (such as a C or COBOL compiler). The compiler processes the source code and produces object code as its output. Note that this step has nothing in particular to do with the DBMS or with SQL.

4. The *linker* accepts the object modules generated by the compiler, links them with various library routines, and produces an executable program. The library routines linked into the executable program include the private DBMS routines described in Step 2.

5. The database request module generated by the precompiler is submitted to a special BIND program. This program examines the SQL statements; parses, validates, and optimizes them; and produces an application plan for each statement. The result is a combined application plan for the entire program, representing a DBMS-executable version of its embedded SQL statements. The BIND program stores the plan in the database, usually assigning it the name of the application program that created it.

**Dynamic SQL**

Program containing embedded dynamic SQL statement must be precompiled like those containing static SQL, but unlike static SQL , dynamic SQL statements are constructed and prepared at run time .the SQL statement text is prepared and executed using either the PREPARE and EXECUTE statements, or the EXECUTE IMMIDDIATE statement. The statement can also be execute with cursor operations if it is select statement. Figure below shows the processing of SQL statement as example of dynamic SQL statement.



**FIGURE: - How the DBMS processes a SQL statement**

**1.** In static SQL statement goes through the first four steps of the process at compile-time. The BIND utility (or the equivalent part of the DBMS runtime system) analyzes the SQL statement, determines the best way to carry it out, and stores the application plan for the statement in the database as part of the program development process. When the static SQL statement is executed at runtime, the DBMS simply executes the stored application plan.

**2.** In dynamic SQL, the SQL statement to be executed isn’t known until runtime, so the DBMS cannot prepare for the statement in advance. When the program is actually executed, the DBMS receives the text of the statement to be dynamically executed (called the *statement string*) and goes through all five of the steps shown in Figure above at runtime.

**Transaction Control Language(Commit, Rollback)**

Transaction Control (TCL) statements are used to manage the changes made by DML, statements. It allows statements to be grouped together into logical transactions. It plays an important role in SQL. Transaction control language is used for managing the changes made by DML statements. It allows statements to be grouped together to form a logical transaction. There are lots of TCL commands which are used in SQL in which some are defined as follows:

**COMMIT:** Commit command is used for saving work done in database. It is the responsibility of the programmer to execute commits statement only after ensuring the correctness and integrity of data. You cannot cancel or undo the changes performed by a transaction after a commit transaction statement is issued because the database modifications are made perm anent after that.

A SQL statement that is executed successfully without any errors does not confirms that the Transaction is committed. Executing successfully means that the specific statement was:

1.parse

2. Found to be syntactically correct and a valid SQL statement

3. Does not throw any errors or exceptions.

However, until the transaction involving the statement is committed, the transaction can be rolled back, and all of the changes of the statement can be reversed.

A statement is said to be committed when the user gives the COMMIT statement explicitly. An implicit request occurs in either of the following conditions:

(1) After normal termination of an application using END statement

(2) Successful execution of a data definition language (DDL) operation.

The changes made by the SQL statement(s) of a transaction become permanent and visible to other users only after that transaction commits.

Example:

create table employee\_detail

(

empid varchar(10),

empname varchar(30),

designation varchar(30),

salary numeric,

deptno varchar(20)

)

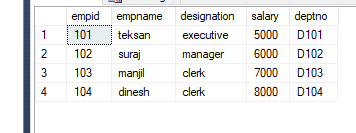
Insert into employee\_detail values('101','Ram','executive',5000,'D101')

Insert into employee\_detail values('102','suraj','manager',6000,'D102')

Insert into employee\_detail values('103','manjil','clerk',7000,'D103')

Insert into employee\_detail values('104','dinesh','clerk ',8000,'D104')

select \* from employee\_detail



Consider the scenario where the salary of employee named Ram is going to change to 3000 and salary of an employee named suraj is going to change 40000 .To perform these transaction , you need to execute following statement

Update employee\_detail

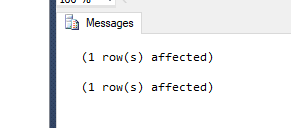
Set salary = 3000

where empname ='Ram'

Update employee\_detail

Set salary = 4000

where empname ='suraj'



Update employee\_detail

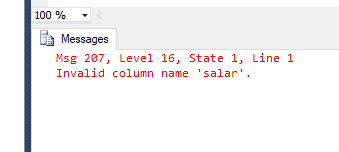
Set salar = 3000

where empname ='Ram'

Update employee\_detail

Set salary = 4000

where empname ='suraj'



when the preceding statement are executed ,either both should be executed successfully or none of them should be executed .if any of the statement fails to execute, the entire transmission should be Rolled Backed. Therefore ,you need to define the beginning and end of a transaction , as shown in following statement

begin transaction mytran

Update employee\_detail

Set salary= 3000

where empname ='Ram'

Update employee\_detail

Set salary = 4000

where empname ='suraj'

commit transaction mytran

**ROLLBACK**

**ROLLBACK:** The rollback commands cancels the modifications done by the recent DML statements and restore database to original state since the last COMMIT. The Roll back of the transactions can occur under the following circumstances

(1) Transaction can be rolled back due to a statement execution error.

(2) The Transaction can Rollback to a Savepoint if explicitly specified.

(3) Rollback of a transaction due to user request (ROLLBACK statement)

(4) Rollback of a transaction due to abnormal application termination like PC

shut down or accidentally closure of the SQL command line window.

(5) Rollback of incomplete transactions during database recovery

NOTE:- Save point command in SQL Server is used  to rollback the transaction into a certain point.

The syntax for SAVEPOINT command is as follows

**save transaction  save point\_name;**

The syntax for rolling back to a SAVEPOINT is as follows

**rollback transaction savepoint\_name;**

Example

create table emp

(empid int constraint PRIMARYKEY primary key,

empName varchar(15)

)

drop table emp

insert into emp values(1,'Ram')

insert into emp values(2,'aman')

insert into emp values(3,'vijay')

select \* from emp

begin transaction tr1

update emp set empName ='jay' where empid=1

commit tran tr1

begin transaction tr2

declare @id int;

set @id=5;

insert into emp values(@id,'d')

if(@id>10)

begin

print'An id greater than 10 is not valid; query is rolled back';

rollback tran tr2;

end

else

begin

print 'data is inserted'

end

select \* from emp

CREATE TABLE TestTable

(

ID INT NOT NULL,

name varchar(20),

mark numeric

)

TRUNCATE TABLE TestTable

BEGIN TRANSACTION

INSERT INTO TestTable VALUES ( 1,'Ram',50)

SAVE TRANSACTION first\_svpt -- this will create a savepoint-1

INSERT INTO TestTable VALUES ( 1,'sajan',75)

SAVE TRANSACTION second\_svpt -- this will create a savepoint-2

INSERT INTO TestTable VALUES ( 1,'amar',50)

ROLLBACK TRANSACTION first\_svpt -- this will rollback to the first savepoint named first\_svpt

COMMIT

SELECT \* FROM TestTable

=======================end of unit -3==============================

|  |
| --- |
| **Topics cover** |

Background, Data Definition Language: Domain Types in SQL, Schema Definition in SQL. Data Manipulation Language: The select Clause, The where Clause, The from Clause, The Rename Operation, Tuple Variables, String Operations, Ordering the Display of Tuples, Duplicate Tuples. Set Operations. Aggregate Functions. Null Values. Nested Subqueries: Set Membership, Set Comparison, Test for Empty Relations, Test for the Absence of Duplicate Tuples. Derived Relations: Views. Modification of the Database: Deletion, Insertion, Updates, Updates, Update of a view. Joined Relations: Join types and Conditions. Embedded SQL. Dynamic SQL. Transaction Control Language (Commit, Rollback).