Unit-6
(SQL) Structured Query Language

Structured Query Language (SQL) was the first commercial language for relational model of database. It is a database query language used for storing and managing data in Relational DBMS. SQL uses the terms **table, row** and **column** for the formal relational model terms relation, tuple and attribute respectively. The main SQL command for data definition is the **CREATE** statement, which can be used to create schemas, tables etc. Today almost all RDMS (MySQL, Oracle, Ms.Access etc.) use SQL as the standard database query language.

⊛ CREATE TABLE Command in SQL:

CREATE TABLE Command tells the database system to create a new table. This command is used to specify a new relation by giving it a name and specifying its attributes and initial constraints. The attributes are specified first and each attribute is given a name, a data type to specify it's domain of values and possibly attribute constraints, such as NOT NULL. The basic syntax of CREATE TABLE statement is as follows:-

```
CREATE TABLE table_name(
        column 1 datatype,
        column 2 datatype,
           ⋮
        column N datatype,
        PRIMARY KEY (one or more columns)
);
```

In brackets comes the list defining each column in the table and what sort of data type it is. The relations declared through CREATE TABLE command are called base tables (or base relations); this means that the table and its rows are actually created and stored as a file by DBMS.

Example: Creating table for Employee.

```
CREATE TABLE EMPLOYEE(
        Fname    VARCHAR(15)   NOT NULL,
        Lname    VARCHAR(15)   NOT NULL,
        Ssn      CHAR(9),
        Bdate    DATE,
        Sex      CHAR,
        Salery   DECIMAL(10,2),
        Dno      INT           NOT NULL,
        PRIMARY KEY (Ssn)
);
```

⊛. Attribute Data Types and Domains in SQL:-

The basic data types available for attributes include numeric, character string, bit string, Boolean, date, and time.

Numeric → Numeric data types include integer numbers of various sizes (INTEGER, INT, and SMALLINT) and floating point numbers of various precision (FLOAT and DOUBLE PRECISION). Formatted numbers can be declared by using DECIMAL(i,j) where i is the total no. of decimal digits and j is the no. of digits after decimal point.

Character-string → Character-string data types are either fixed length — CHAR(n) or CHARACTER(n), where n is the number of characters or varying length — VARCHAR(n) or CHAR VARYING(n) or CHARACTER VARYING(n), where n is the maximum number of characters.

Bit-string → Bit-string data types are either of fixed length n-BIT(n) or varying length — BIT VARYING(n), where n is the maximum number of bits. The default for n, the length of a character string or bit string is 1. Literal bit strings are placed between single quotes but preceded by a B to distinguish them from character strings; for example, B'10101'.

Boolean → A Boolean data type has the traditional values of TRUE or FALSE. In SQL, because of the presence of NULL values, a third possible value for a Boolean data type is UNKNOWN.

Date → The DATE data type has ten positions, and it's components are YEAR, MONTH and DAY in the form YYYY-MM-DD. The TIME data type has at least eight positions, with the components HOUR, MINUTE, and SECOND in the form of HH:MM:SS. Only valid dates and times should be allowed by the SQL implementation.

Domains in SQL → A domain can be declared and the domain name can be used with attribute specification. This makes it easier to change the data type for a domain that is used by numerous attributes in a schema, and improves schema readability. For example we can create a domain SSN_TYPE by the following statement;

```
CREATE DOMAIN SSN_TYPE AS CHAR(9);
```

We can use SSN_TYPE in place of CHAR(9). A domain can also have an optional default specification via a DEFAULT clause. Notice that domains may not be available in some implementations of SQL.

⊛. ALTER and DROP commands →

ALTER command → The alter command is used to modify an existing database, table, view or other database objects that might need to change during the life cycle of a database. Let's suppose that we have completed our database design and it has been implemented. Our database users are using it and then they realize some of the vital information was left out in the design phase. They don't want to lose the existing data but just want to incorporate the new information. The alter command comes in handy in such situations. We can use the alter command to change the data type of a field say string to numeric, change the field name to new name or even add a new column in a table.

Syntax:

```
ALTER TABLE `table_name` ADD COLUMN `column_name`
                                    `data_type`;
```

DROP command → The DROP command is used to delete a database from server or to delete an object (like Table, Column) from a database.

Syntax:

To drop table: DROP TABLE `table_name`;

#Specifying Constraints:

Constraints: Constraints are the rules that we can apply on the type of data in a table. The available constraints in SQL are NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY, CHECK, DEFAULT. We can specify constraints at the time of creating the table as below syntax.

```
CREATE TABLE table_name(
        column1 data_type(size) constraint_name,
        column2 data_type(size) constraint_name,
        :
);
```

Specifying Attribute Constraints and Attribute Defaults:

Because SQL allows NULLs as attribute values, a constraint NOT NULL may be specified if NULL is not permitted for a particular attribute. This is always implicitly specified for any other attributes whose values are required not to be NULL.

It is also possible to define a default value for an attribute by appending the clause DEFAULT <value> to an attribute definition. The default value is included in new tuple if an explicit value is not provided for that attribute. If no default clause is specified, the default value NULL for attributes that do not have the NOT NULL constraint.

Another type of constraint can restrict attribute or domain values using the CHECK clause. For example, suppose that department numbers are restricted to integer numbers between 1 and 20; then we can change the attribute declaration of Dnumber in the DEPARTMENT table to the following:

```
Dnumber INT NOT NULL CHECK (Dnumber > 0 AND Dnumber < 21);
```

⊛. Specifying Key and Referential Integrity Constraints:-

Because keys and referential integrity constraints are very important, there are some special clauses within the CREATE TABLE statement to specify them. The **PRIMARY KEY** clause specifies one or more attributes that makes up the primary key of a relation. For example, the primary key of DEPARTMENT can be specified as;

```
Dnumber INT PRIMARY KEY
```

The **UNIQUE** clause specifies alternate (unique) keys, also known as candidate keys. The **UNIQUE** clause can also be specified directly for a unique key if it is single attribute as;

```
Dname VARCHAR(15) UNIQUE
```

Referential integrity is specified via the **FOREIGN KEY** clause. A referential integrity constraint can be violated when tuples are inserted or deleted, or when a foreign key or primary key attribute value is updated.

#Basic Retrival Queries:

⊛. SELECT-FROM-WHERE Structure:-

The basic form of the select-from-where block is formed of the three clauses SELECT, FROM and WHERE and has the following form:

```
SELECT  <attribute list>
FROM    <table list>
WHERE   <condition> ;
```

where,

<attribute list> is a list of attribute names whose values are to be retrived by the query.

<table list> is a list of the relation names required to process the query.

<condition> is a conditional (Boolean) expression that identifies the tuples to be retrived by the query.

Example1: Retrive the birth date and address of the employee whose name is 'John B. Smith'.

Solution:

```
SELECT  Bdate, Address
FROM    EMPLOYEE
WHERE   Fname='John' AND Minit='B' AND Lname='Smith';
```

⇒ This query in the example1 involves only the EMPLOYEE relation listed in the FROM clause. The query selects individual EMPLOYEE tuples that satisfy the condition of WHERE clause, then projects the result on the Bdate and Address attributes listed in the SELECT clause.

Example 2: Retrive the name and address of all employees who work for the 'Research' department.

Solution:

```
SELECT  Fname, Lname, Address
FROM    EMPLOYEE, DEPARTMENT
WHERE   Dname='Research' AND Dnumber=Dno;
```

⊛. Ambiguous Attribute Names, Aliasing, Renaming, and Tuple Variables

Ambiguous Attribute Names: In SQL, the same name can be used for two (or more) attributes as long as the attributes are in different tables. If this is the case, and a multitable query refers to two or more attributes with the same name then we must **qualify** the attribute name with the relation name to prevent ambiguity. This is done by prefixing the relation name to the attribute name and separating the two by a period.

Example :-

```
SELECT   EMPLOYEE.Fname, Address
FROM     EMPLOYEE, DEPARTMENT
WHERE    Dname = 'Research' AND Dnumber = Dno;
```

Here, writing EMPLOYEE.Fname we are qualifying attribute name Fname with relation name EMPLOYEE (i.e, we are prefixing relation name EMPLOYEE to attribute name Fname) to specify which one we are referring to either EMPLOYEE or DEPARTMENT. EMPLOYEE.Fname shows we are refering to EMPLOYEE. This helps to prevent from ambiguity.

Aliasing and Tuple Variables:

We can rename the table names to shorter names by creating an alias for each table name to avoid repeated typing of long table names, which is called aliasing. An alias follow the keyword AS, as shown below in the example.

Example:- For each employee, retrive the employee's first and last name and the first and last name of his or her intermediate supervisor.

Solution:

```
SELECT   E.Fname, E.Lname, S.Fname, S.Lname
FROM     EMPLOYEE AS E, EMPLOYEE AS S
WHERE    E.Super_ssn = S.Ssn;
```

⇒ In this case we declare alternative relation names E and S, called aliases or tuple variables, for the EMPLOYEE relation.

Renaming: It is also possible to rename the relation attributes within the query in SQL by giving them aliases. For example, if we write

```
EMPLOYEE AS E(Fn, Mi, Ln, Ssn, Bd, Addr, Sex, Dno)
```

in the form of clause, then Fn becomes alias for Fname, Mi for Minit, Ln for Lname and so on. We can use this alias-naming or renamming mechanism in any SQL query to specify tuple variables for every table in the WHERE clause. This practice is recommended since it results in queries that are easier to comprehened.

⊛. Unspecified WHERE Clause and Use of the Asterisk:

Unspecified or a missing WHERE clause indicates no condition on tuple selection. Hence all tuples of the relation specified in the FROM clause qualify and are selected for the query result. If more than one relation is specified in the FROM clause and there is no WHERE clause, then the CROSS PRODUCT – all possible tuple combinations – of these relations is selected. If any condition is incorrect or unspecified then very large relations may result.

Example:- Select all ~~EMPLOYEE Ssns and all~~ combinations of EMPLOYEE Ssn and DEPARTMENT Dname in the database.

Solution:

```
SELECT  Ssn, Dname
FROM    EMPLOYEE, DEPARTMENT;
```

Use of Asterisk: To retrive all the attribute values of the selected tuples, we do not have to list the attribute names explicitly in SQL; we just specify an asterisk (*), which stands for all the attributes. This * can also be prefixed by the relation name or alias. For Example EMPLOYEE* refers to all attributes of the EMPLOYEE table.

⊛ Pattern matching and arithmetic operators:

Substring Pattern Matching:- This feature allows comparision conditions on only parts of a character string, using the LIKE comparision operator. This can be used for string pattern matching. Partial strings are specified using two reserved characters: % replaces an arbitrary number of zero or more characters, and the underscore (_) replaces a single character.

Patterns are case sensitive. Special characters (percent, underscore) can be included in patterns using an escape character '\' (backslash).

Examples:

i) 'Raj%' matches any string starting with "Raj".
ii) '%Raj' matches any string ending with "Raj".
iii) '__91' matches with any string ending with "91", with any two characters before that.
iv) '____' matches any string with exactly four characters.

Using 'LIKE' operator example: (also part of pattern matching topic)

Obtain roll numbers and names of all students whose names end with 'Mohan'.

```
SELECT  rollNo, name
FROM    STUDENT
WHERE   name LIKE '%Mohan';
```

Note: % & (_) are called wildcard operators

Arithmetic Operators:-

Operator	Operation	Description
+	Addition	Add values on either side of operator.
−	Subtraction	Used to subtract the right hand side value from left hand side value.
*	Multiplication	Multiplies the values present on each side of the operator.
/	Division	Divides left hand side value from the right hand side value.
%	Modulus	Returns the remainder.

(also called WHERE clause operators)

Comparision Operators:- =, >, <, <=, >=, <> or != , !>, !< etc. are comparision operators used in SQL. (not greater than)

Logical Operators:- AND, OR, NOT and following are used in SQL;
BETWEEN → Searches the values within the range mentioned.
EXISTS → Used to search for the row's presence in table.
LIKE → Compares a pattern using wildcard operators.
ALL → Used to compare specific value to all other values in set.
ANY → Compares a specific value to any of the values in set.

Example: Retrive all employees in department 5 whose salery is between $30,000 and $40,000.

```
SELECT *          (specifies all)
FROM   EMPLOYEE
WHERE  (Salary BETWEEN 30000 AND 40000) AND Dno=5;
```

⊛ Comparisons involving NULL:

SQL represents "this has no value" by the special non-value NULL. NULL isn't a value it's just a representation that there is no any value. The result of comparing anything to NULL, even itself, is always NULL. A comparision to NULL is never true or false. Since NULL can never be equal to any value, it can never be unequal, either.

The correct way to write the queries is instead of using boolean comparision operators such as less-than and greater-than, equal-to and not-equal-to, these queries must be written with the special comparision operator **IS NULL**. The IS NULL operator tests whether a value is null or not null, and returns a boolean.

Example:- select * from table where column is not null;

(we can write in single line as this or multiple lines as we used before.)

⊛ Nested Queries:-

If a query is written inside a query, then it is called nested query. In nested queries the result of inner query is used in the execution of outer query. It is embedded within the WHERE clause, enclosed within parenthesis. There are mainly two types of nested queries:

- Independent nested queries → The execution of inner query is independent of outer query, but the result of inner query is used in execution of outer query. Various operators like IN, NOT IN, ANY, ALL etc. are used in writing independent nested queries.

ii) Co-related nested queries → The execution of inner query is dependent of outer query and the result of inner query is used in the execution of outer query. Various operators like AND are used in co-related nested queries.

For Example:

```
SELECT *
FROM CUSTOMERS
WHERE ID IN(
        SELECT ID
        FROM CUSTOMERS
        WHERE Salary > 4500);
```

⊛ Concept and example of INSERT, DELETE and UPDATE commands:

Insert Command → INSERT is used to add a single tuple (row) to a relation (table). We must specify the relation name and a list of values for the tuple.

Example:

```
INSERT INTO EMPLOYEE
VALUES ('Roshan', 'Bist', '2000-7-5', '50000', 4);
```

A second form of the INSERT statement allows the user to specify explicit attribute names that correspond to the values provided in the INSERT command. This is useful if a relation has many attributes but only a few of those attributes are assigned values in the new tuple.

Example:

```
INSERT INTO EMPLOYEE (Fname, Lname, Dno, Ssn)
VALUES ('Roshan', 'Bist', 4, '65329865');
```

DELETE Command → The DELETE command removes tuples from a relation. It includes a WHERE clause to select the tuples to be deleted. Tuples are explicitly deleted from only one table at a time. Depending on the number of tuples selected by the condition in the WHERE clause, zero, one, or several tuples can be deleted by a single DELETE command. A missing WHERE clause specifies that all tuples in the relation are to be deleted; however, the table remains in the database

as an empty table. We must use the DROP TABLE command to remove the table definition.

Example:

```
DELETE FROM EMPLOYEE
WHERE Lname='Bist';
```

UPDATE Command → The UPDATE command is used to modify attribute values of one or more selected tuples. As in the DELETE command, a WHERE clause in the UPDATE command selects the tuples to be modified from a single relation. An additional SET clause in the UPDATE command specifies the attributes to be modified and their new values.

For Example:- To change the location and controlling department number of project number 10 to 'Butwal' and 5, respectively we write the query as following;

```
UPDATE PROJECT
SET    Plocation='Butwal', Dnum=5
WHERE  Pnumber=10;
```

Several tuples can be modified with a single UPDATE command. Example: Employees in the 'Research' department a 10% raise in salary, the query is as follows:

```
UPDATE EMPLOYEE
SET    Salary=Salary*1.1
WHERE  Dno=5;
```

It is possible to specify NULL or DEFAULT as the new attribute value.

⊛. Concept of views:

A view in SQL terminology is a single table that is derived from other tables. These other tables can be base tables or previously defined views. A view does not necessarily exist in physical form; it is considered to be a virtual table, in contrast to base tables, whose tuples are always physically stored in the database. This limits the possible update operations that can be applied to views, but it does not provide any limitations on ~~quering~~ querying a view.