Module 2

CO2	Use Structured Query Language for building and manipulating databases.		
M2.01	Illustrate the basic features of SQL	5	Applying
M2.02	Illustrate the advanced features of SQL	6	Applying

Contents:

SQL: Data Definition and Data Types, Specifying Constraints, Basic Retrieval Queries, Insert, Delete and Update Statements.
Advanced Features of SQL: Complex SQL Retrieval Queries, Assertions, Triggers, Views, Schema change statements.

SQL is a database computer language designed for the retrieval and management of data in a relational database. **SQL** stands for **Structured Query Language**.

Features of SQL

- It is a nonprocedural language as it specifies what to be retrieved rather than how to retrieve it.
- It can be used for defining the structure of the data, modifying data in the database and specifying security constraints.
- It is an interactive query language that helps the users to execute complicate queries faster.
- The commands resemble English statements, making it easier to learn and understand.

Categories of SQL commands

1. DDL(Data Definition Language)
2. DML (Data Manipulation Language)
3. DQL(Data Query Language)
4. DCL (Data Control Language)
5. TCL (Transaction Control Language)

DDL (Data Definition Language)
DDL statements are used to alter/modify a database or table structure and schema.
CREATE – create a new Table, database, schema
ALTER – alter existing table, column description
DROP – delete existing objects from database

DML (Data Manipulation Language)
DML statements affect records in a table. These are basic operations we perform on data such as selecting a few records from a table, inserting new records, deleting unnecessary records, and updating/modifying existing records.
DML statements include the following:

SELECT – select records from a table
INSERT – insert new records
UPDATE – update/Modify existing records
DELETE – delete existing records

DATA QUERY Language (DQL)
In some books SELECT is under DQL.

DCL (Data Control Language)
DCL statements control the level of access that users have on database objects.
GRANT – allows users to read/write on certain database objects
REVOKE – keeps users from read/write permission on database objects

TCL (Transaction Control Language)
TCL statements allow you to control and manage transactions to maintain the integrity of data within SQL statements.
COMMIT Transaction – commits a transaction

ROLLBACK Transaction – ROLLBACK a transaction in case of any error

SAVEPOINT – Temporarily saves a transaction.

Data types in SQL

1. int or integer- represent numbers
2. smallint- range is less than int
3. char – fixed length character. For example if ABC is stored for an attribute of char(8), then the string is padded with 5 blanks to the right when saving in database.
4. varchar- variable length character. Here shorter string is not padded with blank characters.
5. **numeric(p,s) –** represent floating point numbers. p is the total number of digits and s is the number of digits after decimal point. For example numeric(3,2), then 333.23 and 567.34 are allowed. But 23.334 is not allowed.
6. **Boolean –** true ,false,NULL(to represent unknown)
7. Date and time- date is in form YYYY-MM-DD and time as HH:MM:SS
8. Timestamp- is used to represent date and time together.Format YYYY-MM-DD- HH:MM:SS

RDBMS

RDBMS stands for Relational Database Management System.

RDBMS is the basis for SQL, and for all modern database systems such as MS SQL Server, IBM DB2, Oracle, MySQL, and Microsoft Access.

The data in RDBMS is stored in database objects called tables. A table is a collection of related data entries and it consists of columns and rows.

The basic syntax of this CREATE DATABASE statement is as follows −

```
CREATE DATABASE DatabaseName;
```

Example

If you want to create a new database <testDB>, then the CREATE DATABASE statement would be as shown below −

```
SQL> CREATE DATABASE testDB;
```

Once a database is created, you can check it in the list of databases as follows

Show databases;

```
SQL> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| AMROOD             |
| TUTORIALSPOINT     |
| mysql              |
| orig               |
| test               |
| testDB             |
+--------------------+
7 rows in set (0.00 sec)
```

CREATE TABLE command with constraints NULL, DEFAULT,CHECK, PRIMARY KEY,UNIQUE, referential Integrity

```
CREATE TABLE <table_name> (<attribute1> <data_type1> [constraint1],
                           <attribute2> <data_type2> [constraint2],
                           :
                           <attributen> <data_typen> [constraintn],
                           [table_constraint1]
                           :
                           [table_constraintn]
                         );
```

where,

- `table_name` is the name of new relation
- $attribute_i$ is the attribute of relation
- $data\_type_i$ is the data type of values of the attribute
- $constraint_i$ is any of the column-level constraints defined on the corresponding attribute

Example

The following code block is an example, which creates a CUSTOMERS table with an ID as a primary key and NOT NULL are the constraints showing that these fields cannot be NULL while creating records in this table −

```
SQL> CREATE TABLE CUSTOMERS(
   ID   INT              NOT NULL,
   NAME VARCHAR (20)     NOT NULL,
   AGE  INT              NOT NULL,
   ADDRESS  CHAR (25) ,
   SALARY   DECIMAL (18, 2),
   PRIMARY KEY (ID)
);
```

Command to view the structure of relation

```
DESC table_name;
```

```
SQL> DESC CUSTOMERS;
+---------+---------------+------+-----+---------+-------+
| Field   | Type          | Null | Key | Default | Extra |
+---------+---------------+------+-----+---------+-------+
| ID      | int(11)       | NO   | PRI |         |       |
| NAME    | varchar(20)   | NO   |     |         |       |
| AGE     | int(11)       | NO   |     |         |       |
| ADDRESS | char(25)      | YES  |     | NULL    |       |
| SALARY  | decimal(18,2) | YES  |     | NULL    |       |
+---------+---------------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```

Specifying Constraints

1. Primary key Constraints
2. UNIQUE

3. CHECK
4. N**OT NULL**
5. Foreign key

SQL PRIMARY KEY Constraint

The `PRIMARY KEY` constraint uniquely identifies each record in a table.

Primary keys must contain UNIQUE values, and cannot contain NULL values.

A table can have only ONE primary key; and in the table, this primary key can consist of single or multiple columns (fields).

Syntax

```
<attribute> <data_type> PRIMARY KEY
```

OR

```
PRIMARY KEY (<attribute>)
```

If primary key consist of more than one attribute, then

```
PRIMARY KEY (<attribute1>, <attribute2>, ... , <attributen>)
```

2. UNIQUE

It ensures that the set of attributes have UNIQUE values.

```
<attribute> <data_type> UNIQUE
```

OR

```
UNIQUE (<attribute>)
```

If UNIQUE constraint is applied to more than one attribute, then

```
UNIQUE (<attribute1>, <attribute2>, ... , <attributen>)
```

3. CHECK

It ensures that value inserted in an attribute must satisfy a given expression.

```
<attribute> <data_type> CHECK(<expression>)
```

4. NOT NULL

This constraint is used to specify that an attribute will not accept null values.

Syntax:

```
<attribute> <data_type> NOT NULL
```

5. **FOREIGN KEY constraint**

It ensures that the foreign key value in referencing relation must exist in the primary key attribute of the referenced relation.

```
<attribute> <data_type> REFERENCES <referenced_relation>
(<key_attribute>)
```

Example

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(25) NOT NULL,
    FirstName varchar(25),
    Age int,
    PRIMARY KEY (ID)
);

CREATE TABLE Orders (
    OrderID int NOT NULL,
    OrderNumber int NOT NULL,
    PersonID int,
    PRIMARY KEY (OrderID),
    FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)
);
```

SQL ALTER TABLE Statement

The `ALTER TABLE` statement is used to add, delete, or modify columns in an existing table.

The `ALTER TABLE` statement is also used to add and drop various constraints on an existing table.

ALTER TABLE - ADD Column

To add a column in a table, use the following syntax:

```
ALTER TABLE table_name ADD column_name datatype;
```

Example: The following SQL adds an "Email" column to the "Customers" table:

```
ALTER TABLE Customers ADD Email varchar(25);
```

ALTER TABLE - DROP COLUMN

To delete a column in a table, use the following syntax (notice that some database systems don't allow deleting a column):

```
ALTER TABLE table_name DROP column_name;
```

Example: The following SQL deletes the "Email" column from the "Customers" table:

```
ALTER TABLE Customers DROP COLUMN Email;
```

ALTER TABLE - MODIFY COLUMN

To delete a column in a table, use the following syntax (notice that some database systems don't allow deleting a column):

```
ALTER TABLE table_name MODIFY column_name datatype;
```

Example: The following SQL modify the width of column "Email" in "Customers" table:

```
ALTER TABLE Customers MODIFY Email varchar(40);
```

ALTER TABLE – ADD CONSTRAINT

```
ALTER TABLE <table_name> ADD [CONSTRAINT <constraint_name>]
<Cons>;
```

where,

CONSTRAINT is a keyword
constraint_name is a name given to constraint
Cons can be any of the constraints discussed earlier

Example: Check whether book title is NULL

```
ALTER TABLE BOOK ADD CHECK(Book_title <> '');
```

```
ALTER TABLE BOOK ADD CONSTRAINT Cons_1 UNIQUE(ISBN);
```

ALTER TABLE – DROP CONSTRAINT

Any of the constraint already defined can be dropped.

```
ALTER TABLE BOOK DROP CONSTRAINT Cons_1;
```

Data Manipulation Language (DML)

Data Manipulation Language (DML) commands are used for retrieving and manipulating data stored in the database.

- Retrieval of data stored in the database
- Insertion of data into the database.
- Modification of data stored in the database.
- Deletion of data stored in the database.

Basic retrieval and manipulation commands are SELECT, INSERT,UPDATE and DELETE

SELECT statement

The basic syntax of the SELECT statement is as follows −

SELECT column1, column2, …. , columnN FROM table_name;

Here, column1, column2... are the fields of a table whose values you want to fetch. If you want to fetch all the fields available in the field, then you can use the following syntax.

```
SELECT * FROM table_name;
```

Example

Consider the CUSTOMERS table having the following records −

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

The following code is an example, which would fetch the ID, Name and Salary fields of the customers available in CUSTOMERS table.

```
SQL> SELECT ID, NAME, SALARY FROM CUSTOMERS;
```

This would produce the following result −

```
+----+----------+----------+
| ID | NAME     | SALARY   |
+----+----------+----------+
|  1 | Ramesh   |  2000.00 |
|  2 | Khilan   |  1500.00 |
|  3 | kaushik  |  2000.00 |
|  4 | Chaitali |  6500.00 |
|  5 | Hardik   |  8500.00 |
|  6 | Komal    |  4500.00 |
|  7 | Muffy    | 10000.00 |
+----+----------+----------+
```

If you want to fetch all the fields of the CUSTOMERS table, then you should use the following query.

```
SQL> SELECT * FROM CUSTOMERS;
```

This would produce the result as shown below.

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

The SQL **DISTINCT** keyword is used in conjunction with the SELECT statement to eliminate all the duplicate records and fetching only unique records.

Command used to display only unique values for the attribute category.

```
SQL> SELECT DISTINCT SALARY FROM CUSTOMERS;
```

Instead of DISTINCT, ALL keyword is specified, the result retains all duplicate values.

```
SQL> SELECT ALL SALARY FROM CUSTOMERS;
```

The SQL **WHERE** clause is used to specify a condition while fetching the data from a single table or by joining with multiple tables. If the given condition is satisfied, then only it returns a specific value from the table.

The WHERE clause is not only used in the SELECT statement, but it is also used in the UPDATE, DELETE statement, etc.

The basic syntax of the SELECT statement with the WHERE clause is as shown below.

Instead of DISTINCT, ALL keyword is specified, the result retains all duplicate values.

```
SQL> SELECT column1, column2, columnN
FROM table_name
WHERE [condition];
```

The following code is an example which would fetch the ID, Name and Salary fields from the CUSTOMERS table, where the salary is greater than 2000 −

```
SQL> SELECT ID, NAME, SALARY
FROM CUSTOMERS
WHERE SALARY > 2000;
```

The relational operators (=, <>, <, <=, >, >=) can be used to specify the conditions .

We can also use AND, OR , NOT , BETWEEN, NOT BETWEEN, IN operators.

```
SQL> SELECT ID, NAME, SALARY
FROM CUSTOMERS
WHERE SALARY BETWEEN  2000 AND 6500;
```

IN operator is used to select a set of values. The IN operator selects the values that match an value in a given list of values.

```
SQL> SELECT ID, NAME, SALARY
FROM CUSTOMERS
WHERE SALARY IN ( 2000, 6500);
```

Aliasing and tuple variable

The basic syntax of a **table** alias is as follows.

```
SELECT column1, column2....
FROM table_name AS alias_name
WHERE [condition];
```

The basic syntax of a **column** alias is as follows.

```
SELECT column_name AS alias_name
FROM table_name
WHERE [condition];
```

Consider the following two tables.

Table 1 − CUSTOMERS Table is as follows.

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
```

```
| 1 | Ramesh   | 32 | Ahmedabad |  2000.00 |
| 2 | Khilan   | 25 | Delhi     |  1500.00 |
| 3 | kaushik  | 23 | Kota      |  2000.00 |
| 4 | Chaitali |  25 | Mumbai    |  6500.00 |
| 5 | Hardik   |  27 | Bhopal    |  8500.00 |
| 6 | Komal    |  22 | MP        |  4500.00 |
| 7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

Table 2 − ORDERS Table is as follows.

```
+-----+---------------------+-------------+--------+
|OID  | DATE                | CUSTOMER_ID | AMOUNT |
+-----+---------------------+-------------+--------+
| 102 | 2009-10-08 00:00:00 |           3 |   3000 |
| 100 | 2009-10-08 00:00:00 |           3 |   1500 |
| 101 | 2009-11-20 00:00:00 |           2 |   1560 |
| 103 | 2008-05-20 00:00:00 |           4 |   2060 |
+-----+---------------------+-------------+--------+
```

Now, the following code block shows the usage of a **table alias**.

```
SQL> SELECT C.ID, C.NAME, C.AGE, O.AMOUNT
   FROM CUSTOMERS AS C, ORDERS AS O
   WHERE  C.ID = O.CUSTOMER_ID;
```

This would produce the following result.

```
+----+----------+-----+--------+
| ID | NAME     | AGE | AMOUNT |
+----+----------+-----+--------+
|  3 | kaushik  |  23 |   3000 |
|  3 | kaushik  |  23 |   1500 |
|  2 | Khilan   |  25 |   1560 |
|  4 | Chaitali |  25 |   2060 |
+----+----------+-----+--------+
```

Following is the usage of a **column alias**.

```
SQL> SELECT  ID AS CUSTOMER_ID, NAME AS CUSTOMER_NAME
   FROM CUSTOMERS
WHERE SALARY IS NOT NULL;
```

This would produce the following result.

```
+-------------+---------------+
| CUSTOMER_ID | CUSTOMER_NAME |
+-------------+---------------+
|           1 | Ramesh        |
|           2 | Khilan        |
|           3 | kaushik       |
|           4 | Chaitali      |
|           5 | Hardik        |
|           6 | Komal         |
|           7 | Muffy         |
+-------------+---------------+
```

String Operations

The SQL **LIKE** clause is used to compare a value to similar values using wildcard operators. There are two wildcards used in conjunction with the LIKE operator.

- The percent sign (%)
- The underscore (_)

The percent sign represents zero, one or multiple characters. The underscore represents a single number or character.

⇒ 'A%' matches any string beginning with character *A*
⇒ '%A' matches any string ending with character *A*
⇒ 'A%A' matches any string beginning and ending with character *A*
⇒ '%AO%' matches any string with character *AO* appearing anywhere in a string as substring
⇒ 'A_ _' matches any string beginning with character *A* and followed by exactly two characters

⇒ '_ _A' matches any string ending with character *A* and preceded by exactly two characters
⇒ 'A_ _ _D' matches any string beginning with character *A*, ending with the character *D* and with exactly three characters in between

```
SQL> SELECT ID, NAME
 FROM CUSTOMERS
Where NAME LIKE '%K;
```

This command is used to retrieve customer name and ID whose name ends with K.

The below command is used to retrieve customer name and ID whose name starts with K.

```
SQL> SELECT ID, NAME
 FROM CUSTOMERS
Where NAME LIKE 'K%;
```

The below command is used to retrieve customer name and ID whose name starts with K followed by 5 characters.

```
SQL> SELECT ID, NAME
 FROM CUSTOMERS
Where NAME LIKE 'K_ _ _ _ _;
```

Similarly NOT LIKE operator can also use.

SET operations

UNION, INTERSECT, DIFFERENCE

The UNION operation is used to retrieve tuples from more than one relation and it also eliminates duplicate records. For example, the command to find the union of all the tuples with Price less than *40* and all the tuples with Price greater than *30* from the BOOK relation can be specified as

```
(SELECT *
FROM BOOK
WHERE Price<40)
UNION
(SELECT *
FROM BOOK
WHERE Price>30);
```

The INTERSECT operation is used to retrieve common tuples from more than one relation. For example, the command to find the intersection of all the tuples with Price less than *40* and all the tuples with Price greater than *30* from the BOOK relation can be specified as

```
(SELECT *
FROM BOOK
WHERE Price<40)
INTERSECT
(SELECT *
FROM BOOK
WHERE Price>30);
```

The MINUS operation is used to retrieve those tuples present in one relation, which are not present in other relation. For example, the command to find the difference (using MINUS operation) of all the tuples with Price less than *40* and all the tuples with Price greater than *30* from the BOOK relation can be specified as

```
(SELECT *
FROM BOOK
WHERE Price<40)
MINUS
(SELECT *
FROM BOOK
WHERE Price>30);
```

The UNION, INTERSECT, and MINUS operations automatically eliminate the duplicate tuples from the result. However, if all the duplicate tuples are to be retained, the ALL keyword can be used with these operations. For example, the command to retain duplicate tuples during UNION operation can be specified as

```
(SELECT *
FROM BOOK
WHERE Price<40)
UNION ALL
(SELECT *
FROM BOOK
WHERE Price>30);
```


ORDER BY clause

Syntax

SELECT column-list
FROM table_name
[WHERE condition]
[ORDER BY column1, column2, .. columnN] [ASC | DESC];

Below command display the details of customers in the order of ascending order of name. default ordering is ascending order.

```
SQL> SELECT * FROM CUSTOMERS
  ORDER BY NAME;
```

Below command display the details of customers in the order of descending order of name.

```
SQL> SELECT * FROM CUSTOMERS
  ORDER BY NAME DESC;
```

Below command display the details of customers in the order of descending order of name and ascending order of salary.

```
SQL> SELECT * FROM CUSTOMERS
  ORDER BY NAME DESC, SALARY ASC;
```

INSERT INTO statement

The SQL **INSERT INTO** Statement is used to add new rows of data to a table in the database.

Syntax

There are two basic syntaxes of the INSERT INTO statement which are shown below.

INSERT INTO TABLE_NAME (column1, column2, column3,...columnN) VALUES (value1, value2, value3,...valueN);

Here, column1, column2, column3,...columnN are the names of the columns in the table into which you want to insert the data.

You may not need to specify the column(s) name in the SQL query if you are adding values for all the columns of the table. But make sure the order of the values is in the same order as the columns in the table.

The **SQL INSERT INTO** syntax will be as follows −

```
INSERT INTO TABLE_NAME VALUES (value1,value2,value3,...valueN);
```

Example

```
INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY) VALUES (1, 'Ramesh', 32, 'Ahmedabad', 2000.00 );

INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY) VALUES (2, 'Khilan', 25, 'Delhi', 1500.00 );
```

```
INSERT INTO CUSTOMERS  VALUES (7, 'Muffy', 24, 'Indore', 10000.00 );
```

To insert multiple records

```
insert into customers values (1, 'Ramesh', 32, 'Ahmedabad', 2000.00 ),( 2, 'Khilan', 25, 'Delhi', 1500.00);
```

The SQL UPDATE Statement

Syntax

The basic syntax of the UPDATE query with a WHERE clause is as follows −

```
UPDATE table_name
SET column1 = value1, column2 = value2...., columnN = valueN
WHERE [condition];
```

You can combine N number of conditions using the AND or the OR operators.

Example

Consider the CUSTOMERS table having the following records −

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
+----+----------+-----+-----------+----------+
```

The following query will update the ADDRESS for a customer whose ID number is 6 in the table.

```
SQL> UPDATE CUSTOMERS
SET ADDRESS = 'Pune'
WHERE ID = 3;
```

Now, the CUSTOMERS table would have the following records −

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Pune      |  2000.00 |
+----+----------+-----+-----------+----------+
```

Delete command

The DELETE statement is used to delete existing records in a table.

Syntax

```
DELETE FROM table_name WHERE condition;
```

Example

Consider the CUSTOMERS table having the following records −

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
+----+----------+-----+-----------+----------+
```

The following code has a query, which will DELETE a customer, whose ID is 6.

```
SQL> DELETE FROM CUSTOMERS
WHERE ID = 2;
```

Now, the CUSTOMERS table would have the following records −

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
+----+----------+-----+-----------+----------+
```