



CSE 311L(Database Management System)

LAB-Week 01

Instructor – Asif Ahmed Neloy

Objective:

After completing this lesson, you should be able to do the following:

- ▶ Create database tables
- ▶ Describe the data types that can be used when specifying column definition
- ▶ Table naming rules & Fields Datatypes

Query Language is a technique of accessing data from the database. It's mainly of two types :

- A. **Procedural Query Language** : This is a formal way of accessing the database. The information regarding what has to be accessed and how it has to be accessed are provided along with the queries, so that the data can be accessed from the database.

Relational Algebra is a type of Procedural Query Language. It consists of a set of operators which take one or two relations as input and a new relation is provided as the output. RA is the component of relational query engine. RA also provides a framework for query optimization. SQL queries are internally translated into RA operations first.

Relational Algebra eliminates duplicate tuples from the result by default.

- B. **Non-Procedural Query Language** : This is an informal way of accessing the data from the database. In the case of Non-Procedural Query Language, information is required only about what data has to be accessed from the database. **SQL** and **Tuple Relational Calculus** are types of Non-Procedural Query Language.

SQL is a standard language for accessing and manipulating databases.

What is SQL?

- SQL stands for Structured Query Language
- SQL lets you access and manipulate databases
- SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987

What Can SQL do?

- SQL can execute queries against a database
- SQL can retrieve data from a database
- SQL can insert records in a database
- SQL can update records in a database
- SQL can delete records from a database
- SQL can create new databases
- SQL can create new tables in a database
- SQL can create stored procedures in a database
- SQL can create views in a database
- SQL can set permissions on tables, procedures, and views

SQL is a Standard - BUT....

Although SQL is an ANSI/ISO standard, there are different versions of the SQL language. However, to be compliant with the ANSI standard, they all support at least the major commands (such as SELECT, UPDATE, DELETE, INSERT, WHERE) in a similar manner.

SQL Process

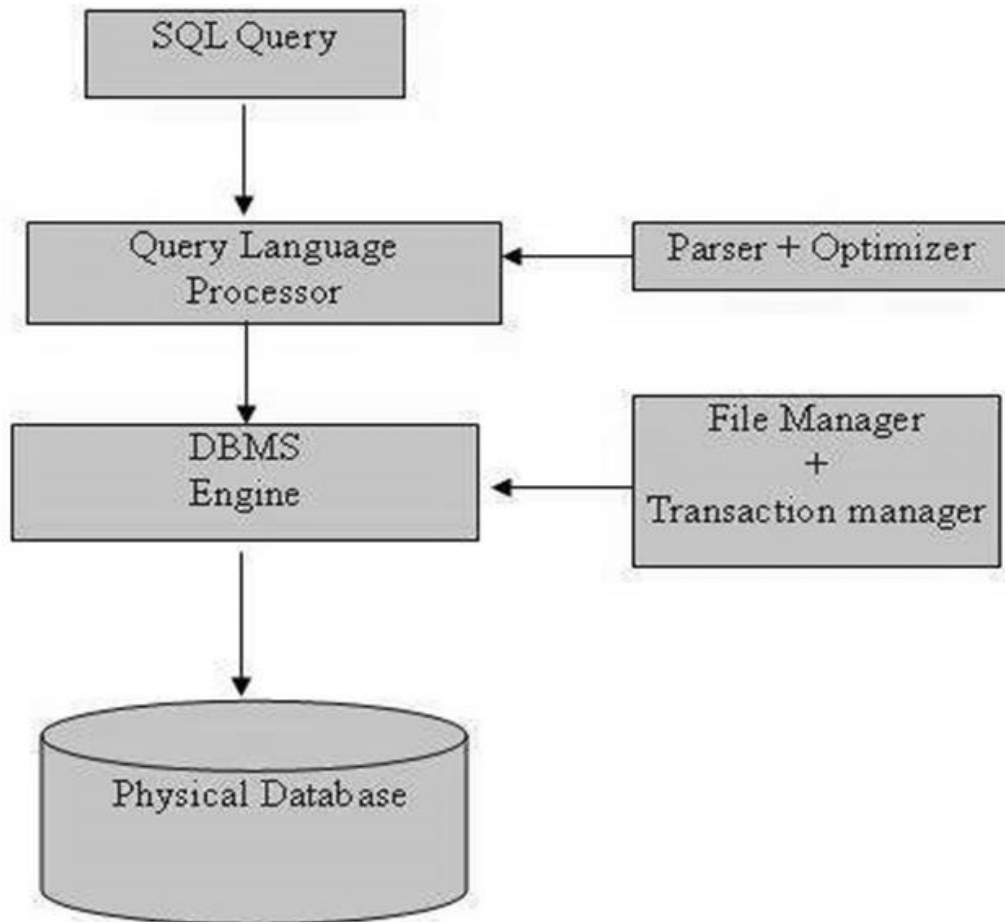
When you are executing an SQL command for any RDBMS, the system determines the best way to carry out your request and SQL engine figures out how to interpret the task. There are various components included in this process.

These components are –

- Query Dispatcher
- Optimization Engines
- Classic Query Engine
- SQL Query Engine, etc.

A classic query engine handles all the non-SQL queries, but a SQL query engine won't handle logical files.

Following is a simple diagram showing the SQL Architecture:



SQL Commands

The standard SQL commands to interact with relational databases are CREATE, SELECT, INSERT, UPDATE, DELETE and DROP. These commands can be classified into the following groups based on their nature:

DDL - Data Definition Language

Command	Description
CREATE	Creates a new table, a view of a table, or other object in the database.
ALTER	Modifies an existing database object, such as a table.
DROP	Deletes an entire table, a view of a table or other objects in the database.

DML - Data Manipulation Language

Command	Description
SELECT	Retrieves certain records from one or more tables.
INSERT	Creates a record.
UPDATE	Modifies records.
DELETE	Deletes records.

DCL - Data Control Language

Command	Description
GRANT	Gives a privilege to user.
REVOKE	Takes back privileges granted from user.

What is RDBMS?

RDBMS stands for **R**elational **D**atabase **M**anagement **S**ystem. RDBMS is the basis for SQL, and for all modern database systems like MS SQL Server, IBM DB2, Oracle, MySQL, and Microsoft Access.

A Relational database management system (RDBMS) is a database management system (DBMS) that is based on the relational model as introduced by E. F. Codd.

What is a table?

The data in an RDBMS is stored in database objects which are called as **tables**. This table is basically a collection of related data entries and it consists of numerous columns and rows.

Remember, a table is the most common and simplest form of data storage in a relational database. The following program is an example of a CUSTOMERS table:

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

What is a field?

Every table is broken up into smaller entities called fields. The fields in the CUSTOMERS table consist of ID, NAME, AGE, ADDRESS and SALARY.

A field is a column in a table that is designed to maintain specific information about every record in the table.

What is a Record or a Row?

A record is also called as a row of data is each individual entry that exists in a table. For example, there are 7 records in the above CUSTOMERS table. Following is a single row of data or record in the CUSTOMERS table:

```

+-----+-----+-----+-----+
|  1  | Ramesh   |  32 | Ahmedabad | 2000.00 |
+-----+-----+-----+-----+

```

A record is a horizontal entity in a table.

What is a column?

A column is a vertical entity in a table that contains all information associated with a specific field in a table.

For example, a column in the CUSTOMERS table is ADDRESS, which represents location description and would be as shown below:

```

+-----+
| ADDRESS |
+-----+
| Ahmedabad |
| Delhi     |
| Kota      |
| Mumbai    |
| Bhopal    |
| MP        |
| Indore    |
+-----+

```

What is a NULL value?

A NULL value in a table is a value in a field that appears to be blank, which means a field with a NULL value is a field with no value.

It is very important to understand that a NULL value is different than a zero value or a field that contains spaces. A field with a NULL value is the one that has been left blank during a record creation.

SQL Constraints

Constraints are the rules enforced on data columns on a table. These are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the database.

Constraints can either be column level or table level. Column level constraints are applied only to one column whereas, table level constraints are applied to the entire table.

Following are some of the most commonly used constraints available in SQL:

- [NOT NULL Constraint](#): Ensures that a column cannot have a NULL value.
- [DEFAULT Constraint](#): Provides a default value for a column when none is specified.
- [UNIQUE Constraint](#): Ensures that all the values in a column are different.
- [PRIMARY Key](#): Uniquely identifies each row/record in a database table.
- [FOREIGN Key](#): Uniquely identifies a row/record in any another database table.
- [CHECK Constraint](#): The CHECK constraint ensures that all values in a column satisfy certain conditions.
- [INDEX](#): Used to create and retrieve data from the database very quickly.

Data Integrity

The following categories of data integrity exist with each RDBMS:

- **Entity Integrity**: There are no duplicate rows in a table.
- **Domain Integrity**: Enforces valid entries for a given column by restricting the type, the format, or the range of values.
- **Referential integrity**: Rows cannot be deleted, which are used by other records.
- **User-Defined Integrity**: Enforces some specific business rules that do not fall into entity, domain or referential integrity.

Database Normalization

Database normalization is the process of efficiently organizing data in a database. There are two reasons of this normalization process:

- Eliminating redundant data. For example, storing the same data in more than one table.
- Ensuring data dependencies make sense.

Both these reasons are worthy goals as they reduce the amount of space a database consumes and ensures that data is logically stored. Normalization consists of a series of guidelines that help guide you in creating a good database structure.

Normalization guidelines are divided into normal forms; think of a form as the format or the way a database structure is laid out. The aim of normal forms is to organize the database structure, so that it complies with the rules of first normal form, then second normal form and finally the third normal form.

It is your choice to take it further and go to the fourth normal form, fifth normal form and so on, but in general, the third normal form is more than enough.

- [First Normal Form \(1NF\)](#)

- [Second Normal Form \(2NF\)](#)
- [Third Normal Form \(3NF\)](#)

Database – First Normal Form (1NF)

The First normal form (1NF) sets basic rules for an organized database:

- Define the data items required, because they become the columns in a table.
- Place the related data items in a table.
- Ensure that there are no repeating groups of data.
- Ensure that there is a primary key.

First Rule of 1NF

You must define the data items. This means looking at the data to be stored, organizing the data into columns, defining what type of data each column contains and then finally putting the related columns into their own table.

For example, you put all the columns relating to locations of meetings in the Location table, those relating to members in the MemberDetails table and so on.

Second Rule of 1NF

The next step is ensuring that there are no repeating groups of data. Consider we have the following table:

```
CREATE TABLE CUSTOMERS(
    ID    INT                NOT NULL,
    NAME  VARCHAR (20)       NOT NULL,
    AGE   INT                NOT NULL,
    ADDRESS CHAR (25),
    ORDERS  VARCHAR(155)
);
```

So, if we populate this table for a single customer having multiple orders, then it would be something as shown below:

ID	NAME	AGE	ADDRESS	ORDERS
100	Sachin	36	Lower West Side	Cannon XL-200
100	Sachin	36	Lower West Side	Battery XL-200

100	Sachin	36	Lower West Side	Tripod Large
-----	--------	----	-----------------	--------------

But as per the 1NF, we need to ensure that there are no repeating groups of data. So, let us break the above table into two parts and then join them using a key as shown in the following program:

CUSTOMERS Table

```
CREATE TABLE CUSTOMERS(
    ID    INT                NOT NULL,
    NAME  VARCHAR (20)       NOT NULL,
    AGE   INT                NOT NULL,
    ADDRESS CHAR (25),
    PRIMARY KEY (ID)
);
```

This table would have the following record:

ID	NAME	AGE	ADDRESS
100	Sachin	36	Lower West Side

ORDERS Table

```
CREATE TABLE ORDERS(
    ID    INT                NOT NULL,
    CUSTOMER_ID INT          NOT NULL,
    ORDERS VARCHAR(155),
    PRIMARY KEY (ID)
);
```

This table would have the following records:

ID	CUSTOMER_ID	ORDERS
10	100	Cannon XL-200
11	100	Battery XL-200

12	100	Tripod Large
----	-----	--------------

Third Rule of 1NF

The final rule of the first normal form, create a primary key for each table which we have already created.

Database – Second Normal Form (2NF)

The Second Normal Form states that it should meet all the rules for 1NF and there must be no partial dependencies of any of the columns on the primary key:

Consider a customer-order relation and you want to store customer ID, customer name, order ID and order detail and the date of purchase:

```
CREATE TABLE CUSTOMERS(
    CUST_ID    INT                NOT NULL,
    CUST_NAME  VARCHAR (20)       NOT NULL,
    ORDER_ID   INT                NOT NULL,
    ORDER_DETAIL VARCHAR (20)     NOT NULL,
    SALE_DATE  DATETIME,
    PRIMARY KEY (CUST_ID, ORDER_ID)
);
```

This table is in the first normal form; in that it obeys all the rules of the first normal form. In this table, the primary key consists of the CUST_ID and the ORDER_ID. Combined, they are unique assuming the same customer would hardly order the same thing.

However, the table is not in the second normal form because there are partial dependencies of primary keys and columns. CUST_NAME is dependent on CUST_ID and there's no real link between a customer's name and what he purchased. The order detail and purchase date are also dependent on the ORDER_ID, but they are not dependent on the CUST_ID, because there is no link between a CUST_ID and an ORDER_DETAIL or their SALE_DATE.

To make this table comply with the second normal form, you need to separate the columns into three tables.

First, create a table to store the customer details as shown in the code block below:

```
CREATE TABLE CUSTOMERS(
    CUST_ID    INT                NOT NULL,
    CUST_NAME  VARCHAR (20)       NOT NULL,
    PRIMARY KEY (CUST_ID)
```

```
);
```

The next step is to create a table to store the details of each order:

```
CREATE TABLE ORDERS(  
    ORDER_ID    INT                NOT NULL,  
    ORDER_DETAIL VARCHAR (20)    NOT NULL,  
    PRIMARY KEY (ORDER_ID)  
);
```

Finally, create a third table storing just the CUST_ID and the ORDER_ID to keep a track of all the orders for a customer:

```
CREATE TABLE CUSTMERORDERS(  
    CUST_ID    INT                NOT NULL,  
    ORDER_ID    INT                NOT NULL,  
    SALE_DATE  DATETIME,  
    PRIMARY KEY (CUST_ID, ORDER_ID)  
);
```

Database – Third Normal Form (3NF)

A table is in a third normal form when the following conditions are met:

- It is in the second normal form.
- All non-primary fields are dependent on the primary key.

The dependency of these non-primary fields is between the data. For example, in the following table – the street name, city and the state are unbreakably bound to their zip code.

```
CREATE TABLE CUSTOMERS(  
    CUST_ID    INT                NOT NULL,  
    CUST_NAME  VARCHAR (20)    NOT NULL,  
    DOB        DATE,  
    STREET     VARCHAR(200),  
    CITY       VARCHAR(100),  
    STATE      VARCHAR(100),  
    ZIP        VARCHAR(12),  
    EMAIL_ID   VARCHAR(256),  
    PRIMARY KEY (CUST_ID)
```

```
);
```

The dependency between the zip code and the address is called as a transitive dependency. To comply with the third normal form, all you need to do is to move the Street, City and the State fields into their own table, which you can call as the Zip Code table.

```
CREATE TABLE ADDRESS(  
    ZIP          VARCHAR(12),  
    STREET       VARCHAR(200),  
    CITY         VARCHAR(100),  
    STATE        VARCHAR(100),  
    PRIMARY KEY (ZIP)  
);
```

The next step is to alter the CUSTOMERS table as shown below.

```
CREATE TABLE CUSTOMERS(  
    CUST_ID      INT          NOT NULL,  
    CUST_NAME    VARCHAR (20) NOT NULL,  
    DOB          DATE,  
    ZIP          VARCHAR(12),  
    EMAIL_ID     VARCHAR(256),  
    PRIMARY KEY (CUST_ID)  
);
```

The advantages of removing transitive dependencies are mainly two-fold. First, the amount of data duplication is reduced and therefore your database becomes smaller.

The second advantage is data integrity. When duplicated data changes, there is a big risk of updating only some of the data, especially if it is spread out in many different places in the database.

For example, if the address and the zip code data were stored in three or four different tables, then any changes in the zip codes would need to ripple out to every record in those three or four tables.

SQL Syntax

SQL is followed by a unique set of rules and guidelines called Sytax. This tutorial gives you a quick start with SQL by listing all the basic SQL Syntax.

All the SQL statements start with any of the keywords like SELECT, INSERT, UPDATE, DELETE, ALTER, DROP, CREATE, USE, SHOW and all the statements end with a semicolon (;).

The most important point to be noted here is that SQL is **case insensitive**, which means SELECT and select have same meaning in SQL statements. Whereas, MySQL makes difference in table names. So, if you are working with MySQL, then you need to give table names as they exist in the database.

Various Syntax in SQL

All the examples given in this tutorial have been tested with a MySQL server.

SQL SELECT Statement

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

SQL DISTINCT Clause

```
SELECT DISTINCT column1, column2....columnN
FROM   table_name;
```

SQL WHERE Clause

```
SELECT column1, column2....columnN
FROM   table_name
WHERE  CONDITION;
```

SQL AND/OR Clause

```
SELECT column1, column2....columnN
FROM   table_name
WHERE  CONDITION-1 {AND|OR} CONDITION-2;
```

SQL IN Clause

```
SELECT column1, column2....columnN
FROM   table_name
WHERE  column_name IN (val-1, val-2,...val-N);
```

SQL BETWEEN Clause

```
SELECT column1, column2....columnN
FROM   table_name
WHERE  column_name BETWEEN val-1 AND val-2;
```

SQL LIKE Clause

```
SELECT column1, column2....columnN
FROM   table_name
WHERE  column_name LIKE { PATTERN };
```

SQL ORDER BY Clause

```
SELECT column1, column2....columnN
FROM   table_name
WHERE  CONDITION
ORDER BY column_name {ASC|DESC};
```

SQL GROUP BY Clause

```
SELECT SUM(column_name)
FROM   table_name
WHERE  CONDITION
GROUP BY column_name;
```

SQL COUNT Clause

```
SELECT COUNT(column_name)
FROM   table_name
WHERE  CONDITION;
```

SQL HAVING Clause

```
SELECT SUM(column_name)
FROM   table_name
WHERE  CONDITION
GROUP BY column_name
HAVING (arithmetic function condition);
```

SQL CREATE TABLE Statement

```
CREATE TABLE table_name(
column1 datatype,
column2 datatype,
column3 datatype,
.....
columnN datatype,
PRIMARY KEY( one or more columns )
);
```

SQL DROP TABLE Statement

```
DROP TABLE table_name;
```

SQL CREATE INDEX Statement

```
CREATE UNIQUE INDEX index_name
ON table_name ( column1, column2,...columnN);
```

SQL DROP INDEX Statement

```
ALTER TABLE table_name
DROP INDEX index_name;
```

SQL DESC Statement

```
DESC table_name;
```

SQL TRUNCATE TABLE Statement

```
TRUNCATE TABLE table_name;
```

SQL ALTER TABLE Statement

```
ALTER TABLE table_name {ADD|DROP|MODIFY} column_name {data_type};
```

SQL ALTER TABLE Statement (Rename)

```
ALTER TABLE table_name RENAME TO new_table_name;
```

SQL INSERT INTO Statement

```
INSERT INTO table_name( column1, column2....columnN)  
VALUES ( value1, value2....valueN);
```

SQL UPDATE Statement

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

SQL DELETE Statement

```
DELETE FROM table_name  
WHERE {CONDITION};
```

SQL CREATE DATABASE Statement

```
CREATE DATABASE database_name;
```

SQL DROP DATABASE Statement

```
DROP DATABASE database_name;
```

SQL USE Statement

```
USE database_name;
```

SQL COMMIT Statement

```
COMMIT;
```