CHAPTER

1

SQL Overview

SQL tutorial gives unique learning on **Structured Query Language** and it helps to make practice on SQL commands which provides immediate results. SQL is a language of database, it includes database creation, deletion, fetching rows and modifying rows etc.

SQL is an ANSI (American National Standards Institute) standard, but there are many different versions of the SQL language.

What is SQL?

SQL is Structured Query Language, which is a computer language for storing, manipulating and retrieving data stored in relational database.

SQL is the standard language for Relation Database System. All relational database management systems like MySQL, MS Access, Oracle, Sybase, Informix, postgres and SQL Server use SQL as standard database language.

Also, they are using different dialects, such as:

- MS SQL Server using T-SQL,
- Oracle using PL/SQL,
- MS Access version of SQL is called JET SQL (native format) etc.

Why SQL?

- Allows users to access data in relational database management systems.
- Allows users to describe the data.
- Allows users to define the data in database and manipulate that data.
- Allows to embed within other languages using SQL modules, libraries & pre-compilers.
- Allows users to create and drop databases and tables.

- Allows users to create view, stored procedure, functions in a database.

- Allows users to set permissions on tables, procedures and views

History:

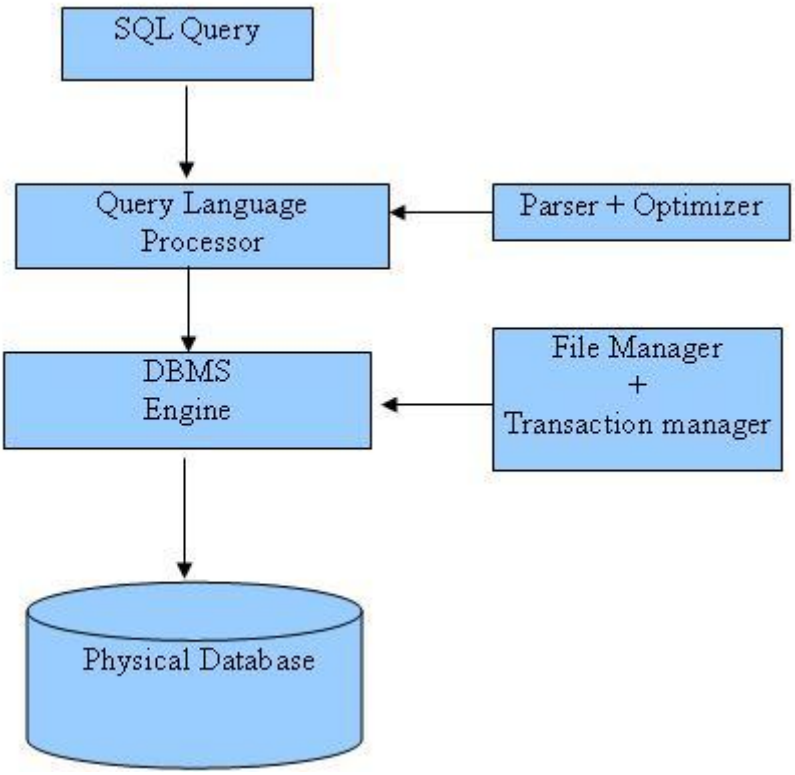
- **1970 --** Dr. E. F. "Ted" of IBM is known as the father of relational databases. He described a relational model for databases.
- **1974 --** Structured Query Language appeared.
- **1978 --** IBM worked to develop Codd's ideas and released a product named System/R.
- **1986 --** IBM developed the first prototype of relational database and standardized by ANSI. The first relational database was released by Relational Software and its later becoming Oracle.

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 the process. These components are Query Dispatcher, Optimization Engines, Classic Query Engine and SQL Query Engine, etc. Classic query engine handles all non-SQL queries, but SQL query engine won't handle logical files.

Following is a simple diagram showing 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 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 database
ALTER	Modifies an existing database object, such as a table.
DROP	Deletes an entire table, a view of a table or other object in the database.

DML - Data Manipulation Language:

Command	Description
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

DQL - Data Query Language:

Command	Description
SELECT	Retrieves certain records from one or more tables

CHAPTER

2

SQL RDBMS Concepts

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 table?

The data0 in RDBMS is stored in database objects called **tables**. The table is a collection of related data entries and it consists of columns and rows.

Remember, a table is the most common and simplest form of data storage in a relational database. Following is the 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 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 record or row?

A record, also called 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 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 consist of the following:

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

What is 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 one that has been left blank during record creation.

SQL Constraints:

Constraints are the rules enforced on data columns on 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 could be column level or table level. Column level constraints are applied only to one column, whereas table level constraints are applied to the whole table.

Following are commonly used constraints available in SQL:

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

NOT NULL Constraint:

By default, a column can hold NULL values. If you do not want a column to have a NULL value, then you need to define such constraint on this column specifying that NULL is now not allowed for that column.

A NULL is not the same as no data, rather, it represents unknown data.

Example:

For example, the following SQL creates a new table called CUSTOMERS and adds five columns, three of which, ID and NAME and AGE, specify not to accept NULLs:

```
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)
);
```

If CUSTOMERS table has already been created, then to add a NOT NULL constraint to SALARY column in Oracle and MySQL, you would write a statement similar to the following:

```
ALTER TABLE CUSTOMERS
   MODIFY SALARY  DECIMAL (18, 2) NOT NULL;
```

DEFAULT Constraint:

The DEFAULT constraint provides a default value to a column when the INSERT INTO statement does not provide a specific value.

Example:

For example, the following SQL creates a new table called CUSTOMERS and adds five columns. Here, SALARY column is set to 5000.00 by default, so in case INSERT INTO statement does not provide a value for this column. then by default this column would be set to 5000.00.

```
CREATE TABLE CUSTOMERS(
       ID   INT              NOT NULL,
       NAME VARCHAR (20)     NOT NULL,
       AGE  INT              NOT NULL,
       ADDRESS  CHAR (25) ,
       SALARY   DECIMAL (18, 2) DEFAULT 5000.00,
       PRIMARY KEY (ID)
);
```

If CUSTOMERS table has already been created, then to add a DFAULT constraint to SALARY column, you would write a statement similar to the following:

```
ALTER TABLE CUSTOMERS
   MODIFY SALARY  DECIMAL (18, 2) DEFAULT 5000.00;
```

Drop Default Constraint:

To drop a DEFAULT constraint, use the following SQL:

```
ALTER TABLE CUSTOMERS
   ALTER COLUMN SALARY DROP DEFAULT;
```

UNIQUE Constraint:

The UNIQUE Constraint prevents two records from having identical values in a particular column. In the CUSTOMERS table, for example, you might want to prevent two or more people from having identical age.

Example:

For example, the following SQL creates a new table called CUSTOMERS and adds five columns. Here, AGE column is set to UNIQUE, so that you can not have two records with same age:

```
CREATE TABLE CUSTOMERS(
       ID   INT              NOT NULL,
```

```
        NAME VARCHAR (20)     NOT NULL,
        AGE  INT              NOT NULL UNIQUE,
        ADDRESS  CHAR (25) ,
        SALARY   DECIMAL (18, 2),
        PRIMARY KEY (ID)
);
```

If CUSTOMERS table has already been created, then to add a UNIQUE constraint to AGE column, you would write a statement similar to the following:

```
ALTER TABLE CUSTOMERS
   MODIFY AGE INT NOT NULL UNIQUE;
```

You can also use following syntax, which supports naming the constraint in multiple columns as well:

```
ALTER TABLE CUSTOMERS
   ADD CONSTRAINT myUniqueConstraint UNIQUE(AGE, SALARY);
```

DROP a UNIQUE Constraint:

To drop a UNIQUE constraint, use the following SQL:

```
ALTER TABLE CUSTOMERS
   DROP CONSTRAINT myUniqueConstraint;
```

If you are using MySQL, then you can use the following syntax:

```
ALTER TABLE CUSTOMERS
   DROP INDEX myUniqueConstraint;
```

PRIMARY Key:

A primary key is a field in a table which uniquely identifies each row/record in a database table. Primary keys must contain unique values. A primary key column cannot have NULL values.

A table can have only one primary key, which may consist of single or multiple fields. When multiple fields are used as a primary key, they are called a **composite key**.

If a table has a primary key defined on any field(s), then you can not have two records having the same value of that field(s).

Note: You would use these concepts while creating database tables.

Create Primary Key:

Here is the syntax to define ID attribute as a primary key in a CUSTOMERS table.

```
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)
);
```

To create a PRIMARY KEY constraint on the "ID" column when CUSTOMERS table already exists, use the following SQL syntax:

```
ALTER TABLE CUSTOMER ADD PRIMARY KEY (ID);
```

NOTE: If you use the ALTER TABLE statement to add a primary key, the primary key column(s) must already have been declared to not contain NULL values (when the table was first created).

For defining a PRIMARY KEY constraint on multiple columns, use the following SQL syntax:

```
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, NAME)
);
```

To create a PRIMARY KEY constraint on the "ID" and "NAMES" columns when CUSTOMERS table already exists, use the following SQL syntax:

```
ALTER TABLE CUSTOMERS
   ADD CONSTRAINT PK_CUSTID PRIMARY KEY (ID, NAME);
```

Delete Primary Key:

You can clear the primary key constraints from the table, Use Syntax:

```
ALTER TABLE CUSTOMERS DROP PRIMARY KEY ;
```

FOREIGN Key:

A foreign key is a key used to link two tables together. This is sometimes called a referencing key.

Foreign Key is a column or a combination of columns whose values match a Primary Key in a different table.

The relationship between 2 tables matches the Primary Key in one of the tables with a Foreign Key in the second table.

If a table has a primary key defined on any field(s), then you can not have two records having the same value of that field(s).

Example:

Consider the structure of the two tables as follows:

CUSTOMERS table:

```
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)
);
```

ORDERS table:

```
CREATE TABLE ORDERS (
       ID          INT        NOT NULL,
       DATE        DATETIME,
       CUSTOMER_ID INT references CUSTOMERS(ID),
       AMOUNT     double,
       PRIMARY KEY (ID)
);
```

If ORDERS table has already been created, and the foreign key has not yet been set, use the syntax for specifying a foreign key by altering a table.

```
ALTER TABLE ORDERS
   ADD FOREIGN KEY (Customer_ID) REFERENCES CUSTOMERS (ID);
```

DROP a FOREIGN KEY Constraint:

To drop a FOREIGN KEY constraint, use the following SQL:

```
ALTER TABLE ORDERS

   DROP FOREIGN KEY;
```

CHECK Constraint:

The CHECK Constraint enables a condition to check the value being entered into a record. If the condition evaluates to false, the record violates the constraint and isn't entered into the table.

Example:

For example, the following SQL creates a new table called CUSTOMERS and adds five columns. Here, we add a CHECK with AGE column, so that you can not have any CUSTOMER below 18 years:

```
CREATE TABLE CUSTOMERS(

       ID   INT              NOT NULL,

       NAME VARCHAR (20)     NOT NULL,

       AGE  INT              NOT NULL CHECK (AGE >= 18),

       ADDRESS  CHAR (25) ,

       SALARY   DECIMAL (18, 2),

       PRIMARY KEY (ID)

);
```

If CUSTOMERS table has already been created, then to add a CHECK constraint to AGE column, you would write a statement similar to the following:

```
ALTER TABLE CUSTOMERS

   MODIFY AGE INT NOT NULL CHECK (AGE >= 18 );
```

You can also use following syntax, which supports naming the constraint in multiple columns as well:

```
ALTER TABLE CUSTOMERS

   ADD CONSTRAINT myCheckConstraint CHECK(AGE >= 18);
```

DROP a CHECK Constraint:

To drop a CHECK constraint, use the following SQL. This syntax does not work with MySQL:

```
ALTER TABLE CUSTOMERS

   DROP CONSTRAINT myCheckConstraint;
```

INDEX:

The INDEX is used to create and retrieve data from the database very quickly. Index can be created by using single or group of columns in a table. When index is created, it is assigned a ROWID for each row before it sorts out the data.

Proper indexes are good for performance in large databases, but you need to be careful while creating index. Selection of fields depends on what you are using in your SQL queries.

Example:

For example, the following SQL creates a new table called CUSTOMERS and adds five columns:

```
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)
);
```

Now, you can create index on single or multiple columns using the following syntax:

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

To create an INDEX on AGE column, to optimize the search on customers for a particular age, following is the SQL syntax:

```
CREATE INDEX idx_age
    ON CUSTOMERS ( AGE );
```

DROP an INDEX Constraint:

To drop an INDEX constraint, use the following SQL:

```
ALTER TABLE CUSTOMERS
   DROP INDEX idx_age;
```

Data Integrity:

The following categories of the 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 the normalization process:

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

Both of these are worthy goals as they reduce the amount of space a database consumes and ensure 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 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 third normal form.

It's your choice to take it further and go to fourth normal form, fifth normal form, and so on, but generally speaking, third normal form is enough.

- First Normal Form (1NF)
- Second Normal Form (2NF)
- Third Normal Form (3NF)

First Normal Form

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

- Define the data items required, because they become the columns in a table. Place 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 finally putting 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 follows:

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 1NF, we need to ensure that there are no repeating groups of data. So let us break above table into two parts and join them using a key as follows:

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.

Second Normal Form

Second normal form states that it should meet all the rules for 1NF and there must be no partial dependences 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 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 first normal form, in that it obeys all the rules of first normal form. In this table, the primary key consists of CUST_ID and ORDER_ID. Combined, they are unique assuming same customer would hardly order same thing.

However, the table is not in 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. Order detail and purchase date are also dependent on ORDER_ID, but they are not dependent on CUST_ID, because there's no link between a CUST_ID and an ORDER_DETAIL or their SALE_DATE.

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

First, create a table to store the customer details as follows:

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

Next, create a table to store 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 CUST_ID and ORDER_ID to keep 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)
);
```

Third Normal Form

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

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

The dependency of nonprimary fields is between the data. For example, in the below table, street name, city, and state are unbreakably bound to the 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 zip code and address is called a transitive dependency. To comply with third normal form, all you need to do is move the Street, City, and State fields into their own table, which you can call the Zip Code table:

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

Next, alter the CUSTOMERS table as follows:

```
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 twofold. 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's a big risk of updating only some of the data, especially if it's spread out in a number of different places in the database. For example, if address and zip code data were stored in three or four different tables, then any changes in zip codes would need to ripple out to every record in those three or four tables.