SQL (Structured **Q**uery **L**anguage)

Module 1 (Basics of SQL)

Contents

Chapter 1

Overview: Basics

When you begin doing SQL, It's a good idea to have a mental map of the terrain you'll be passing through and the equipment that you'll be using throughout your journey; always by your side and helping you reach your destination. You'll need to know the basics of what SQL is and how they work.

1.1 Structured Query Language

An Introduction, Structured Query Language or SQL, is a programming nomenclature used to do set operations to organize and retrieve information in relational databases, based on "set theory and relational algebra". Also it's "relational data language that provides a consistent, English keyword-oriented set of facilities for query, data definition, data manipulation, and data control."

SQL is the programming language for relational databases like MySQL, Oracle, Sybase, SQL Server, Postgre, etc. Other non-relational databases (also called **NoSQL**) databases like MongoDB, DynamoDB, etc do not use SQL. SQL is case **insensitive, b**ut it is a recommended practice to use keywords (like SELECT, UPDATE, CREATE, etc) in capital letters and use user defined things (liked table name, column name, etc) in small letters.

For short it's standard Database language which is used to **create**, **maintain** and **retrieve** the relational database. A **relational database** simply means the data is stored as well as retrieved in the form of relations (**tables)** **see Table 1:** shows the relational database with only one relation called STUDENT which stores ROLL_NO, NAME, ADDRESS, PHONE and AGE of students**.**

STUDENT

ROLL_NO	NAME	ADDRESS	PHONE	AGE
1	RAM	DELHI	9455123451	18
2	RAMESH	GURGAON	9652431543	18
3	SUJIT	ROHTAK	9156253131	20
4	SURESH	DELHI	9156768971	18

TABLE 1

Terminologies Relations

Attribute: the properties that define a relation. e.g.; ROLL_NO, NAME etc.
Tuple: Each row in the relation is known as tuple. The above relation contains 4 tuples, one of which is shown as:
Cardinality: The number of tuples in a relation is known as cardinality. The STUDENT relation defined above has cardinality 4.

1	RAM	DELHI	9455123451	18
(1)	(2)	(3)	(4)	(5)

Degree: The number of attributes in the relation is known as degree of the relation. The STUDENT relation defined above has degree 5.
Column: represents the set of values for a particular attribute. The column ROLL_NO is extracted from relation STUDENT.

Data Definition Language: It is used to define the structure of the database. e.g; CREATE TABLE, ADD COLUMN, DROP COLUMN and so on.
Data Query Language: It is used to extract the data from the relations. e.g.; SELECT.
So first we will consider the Data Query Language. A generic query to retrieve from a relational database is:

```
1. SELECT [DISTINCT] Attribute_List FROM R1,R2....RM
2. [WHERE condition]
3. [GROUP BY (Attributes) [HAVING condition]]
4. [ORDER BY(Attributes) [DESC]];
```

If you want to retrieve from a relational database, Part of the query represented by statement 1 is **required**; the statements written inside "[ ]" are optional.

Here are some cases that might be helpful,

Case 1: If we want to retrieve attributes ROLL_NO and NAME of all students, the query will be:

- **SELECT** ROLL_NO, NAME **FROM** STUDENT;
-

ROLL_NO	NAME
1	RAM
2	RAMESH
3	SUJIT
4	SURESH

Case 2: If we want to retrieve **ROLL_NO** and **NAME** of the students whose **ROLL_NO** is greater than 2, the query will be:

- **SELECT** ROLL_NO, NAME **FROM** STUDENT
  WHERE ROLL_NO>2;
-

ROLL_NO	NAME
3	SUJIT
4	SURESH

CASE 3: If we want to retrieve all attributes of students, we can write * in place of writing all attributes as:

- **SELECT * FROM** STUDENT
  WHERE ROLL_NO>2;
-

ROLL_NO	NAME	ADDRESS	PHONE	AGE
3	SUJIT	ROHTAK	9156253131	20
4	SURESH	DELHI	9156768971	18

CASE 4: If we want to represent the relation in ascending order by **AGE,** we can use ORDER BY clause as:

- **SELECT * FROM** STUDENT **ORDER BY** AGE;
  Can also be SELECT *** FROM** STUDENT **ORDER BY** AGE ASC/DESC; (Ascending/Descending)
-

ROLL_NO	NAME	ADDRESS	PHONE	AGE
1	RAM	DELHI	9455123451	18
2	RAMESH	GURGAON	9652431543	18
4	SURESH	DELHI	9156768971	18
3	SUJIT	ROHTAK	9156253131	20

CASE 5: If we want to retrieve distinct values of an attribute or group of attribute, **DISTINCT** is used as in:
Note: Without DISTINCT the Address "DELHI" will be repeated.

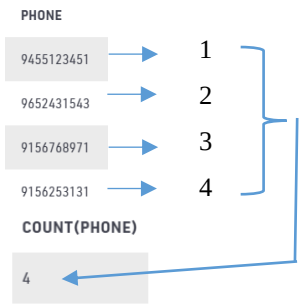
- **SELECT DISTINCT** ADDRESS **FROM** STUDENT;
-

ADDRESS
DELHI
GURGAON
ROHTAK

AGGRATION FUNCTIONS performs a calculation on a set of values, and returns a single value, it also ignore null values and often used with the GROUP BY clause of the SELECT statement. All aggregate functions are deterministic. In other words, aggregate functions return the same value each time that they are called, when called with a specific set of input values. Lastly it is used to perform mathematical operations on data values of a relation.

Here are some common aggregation functions used in SQL are:

- **COUNT:** is used to count the number of column in a relation
  - SELECT COUNT (PHONE) FROM STUDENT;
  -



- SUM: SUM function is used to add the values of an attribute in a relation. e.g;
  - **SELECT SUM** (AGE) **FROM** STUDENT;
  -

AGE
18
18
18
20

  -

SUM(AGE)
74

As we have seen above, all aggregation functions return only 1 column.

AVERAGE: It gives the average values of the tuples. It is also defined as sum divided by count values.
Syntax: AVG (attributename)
OR
Syntax: SUM(attributename)/COUNT(attributename)

The above mentioned syntax also retrieves the average value of tuples.
MAXIMUM: It extracts the maximum value among the set of tuples.
Syntax: MAX (attributename)
MINIMUM: It extracts the minimum value amongst the set of all the tuples.
Syntax: MIN (attributename)

GROUP BY: is used to group the tuples of a relation based on an attribute or group of attribute. It is always combined with aggregation function which is computed on group. e.g.;

- **SELECT** ADDRESS, **SUM**(AGE) **FROM** STUDENT
  GROUP BY (ADDRESS);
-

ROLL_NO	NAME	ADDRESS	PHONE	AGE
1	RAM	DELHI	9455123451	18
2	RAMESH	GURGAON	9652431543	18
4	SURESH	DELHI	9156768971	18
3	SUJIT	ROHTAK	9156253131	20

-

ADDRESS	SUM(AGE)
DELHI	36
GURGAON	18
ROHTAK	20

If we try to execute the query given below, it will result in error because although we have computed SUM(AGE) for each address, there are more than 1 ROLL_NO for each address we have grouped. So it can't be displayed in result set. We need to use aggregate functions on columns after SELECT statement to make sense of the resulting set whenever we are using GROUP BY.

- SELECT ROLL_NO, ADDRESS, SUM(AGE) FROM STUDENT
  GROUP BY (ADDRESS);

1.2 Data Types

Just like any other programming languages, SQL also has certain data types available, a particular kind of data item, as defined by the values it can take or the operations that can be performed on it. It just simply mean it is an attribute of data which tells the compiler or interpreter how the programmer intends to use the data.
Here are some of the data types,

Binary Datatypes is a data type consisting of categorical data that can take exactly two possible values, such as "A" and "B", or "heads" and "tails". It has four subtypes **Binary** that has Maximum length of 8000 bytes (Fixed-Length binary data), **Varbinary** that has Maximum length of 8000 bytes (Variable Length binary data), **Varbinary(max)** that has Maximum length of 231 bytes (Variable Length binary data), and lastly **Image** that has Maximum length of 2,147,483,647 bytes (Variable Length binary data).
Take note;
There are two ways to store data :

1. **Variable length**
2. **Fixed length**.

Variable length data format won't use max length every time while fixed one will use max provided fixed length every time.
Fixed-length binary data types (RAW, BINARY, and CHAR(x) FOR BIT DATA) are SQL data types with a driver-defined maximum length.
Variable-length binary data types (VARBINARY, LONG-VARBINARY, and LONG VARCHAR FOR BIT DATA) are SQL data types with a driver-defined maximum variable length.

Numeric Datatypes used for numerical data without a decimal point and is typically grouped in to two, which is **Exact** and **Appropriate**. **Exact** numeric types, values where the precision and scale need to be preserved.

Data Type	From	To
Bigint	-9,223,372,036,853,775,808	9,223,372,036,854,775,807
Int	-2,147,483,648	2,147,483,647
Smallint	-32,768	32,767
Tinyint	0	255
Bit	0	1
Decimal	$-10^{38} + 1$	$10^{38} - 1$
Numeric	$-10^{38} + 1$	$10^{38} - 1$
Money	-922,337,203,685,477.5808	922,337,203,685,477.5808
Smallmoney	-214,748.3648	214,748.3648

Approximate numeric types, values where the precision needs to be preserved and the scale can be floating.

Data Type	**From**	**To**
Float	-1.79E + 308	1.79E + 308
Int	-3.40E + 38	3.40E + 38

Character String Datatype is a data type that accepts character strings of a fixed length.

Data Type	Description
Char	Maximum length of 8000 characters (Fixed-Length non-Unicode Characters)
Varchar	Maximum length of 8000 characters (Variable-Length non-Unicode Characters)
Varchar(max)	Maximum length of 231 characters (Variable-Length non-Unicode Characters)
Text	Maximum length of 2,147,483,647 characters (Variable-Length non-Unicode Characters)

Unicode Character String Datatype used to store Unicode data in the UTF-16 form. They behave similarly to char and varchar.

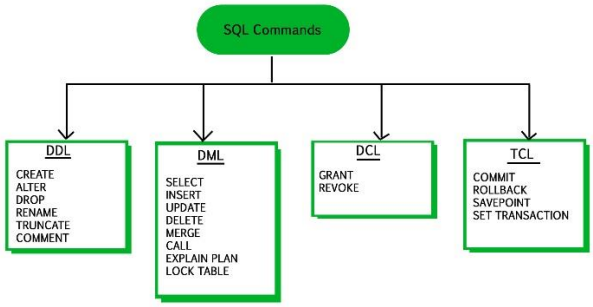
Data Type	Description
nchar	Maximum length of 4000 characters (Fixed-Length Unicode Characters)
nvarchar	Maximum length of 4000 characters (Variable-Length Unicode Characters)
nvarchar(max)	Maximum length of 231 characters (Variable-Length Unicode Characters)

Date and Time Datatype used for values that contain both date and time parts. MySQL retrieves and displays DATETIME values in ' YYYY-MM-DD hh:mm:ss ' format. The supported range is '1000-01-01 00:00:00' to '9999-12-31 23:59:59' .

Data Type	From	To
Datetime	Jan 1, 1753	Dec 31, 9999
Smalldatetime	Jan 1, 1900	June 6, 2079
Date	Stores a date like June 30, 1991	
Time	Stores a time of day like 12:30 P.M	

1.3 DDL, DML, TCL and DCL

In this article, we'll be discussing Data Definition Language, Data Manipulation Language, Transaction Control Language, and Data Control Language.



DDL (Data Definition Language) is used to define the database structure or schema. DDL commands can be used to add, remove, or modify tables within in a database. DDLs used in database applications are considered a subset of SQL. The database system implements integrity constraints that can be tested with minimal overhead.
Domain Constraints : A domain of possible values must be associated with every attribute (for example, integer types, character types, date/time types). Declaring an attribute to be of a particular domain acts as the constraints on the values that it can take.
Referential Integrity : There are cases where we wish to ensure that a value appears in one relation for a given set of attributes also appear in a certain set of attributes in another relation i.e. Referential Integrity. For example, the department listed for each course must be one that actually exists.
Assertions : An assertion is any condition that the database must always satisfy. Domain constraints and Integrity constraints are special form of assertions.
Authorization : We may want to differentiate among the users as far as the type of access they are permitted on various data values in database. These differentiation are expressed in terms of Authorization. The most common being
read authorization – which allows reading but not modification of data ;
insert authorization – which allow insertion of new data but not modification of existing data
update authorization – which allows modification, but not deletion.

Here are some commands (DDL):

- CREATE : to create objects in database
- ALTER : alters the structure of database
- DROP : delete objects from database
- RENAME : rename an objects

Note:
Integrity constraints are a set of rules. ... Integrity constraints ensure that the data insertion, updating, and other processes have to be performed in such a way that data integrity is not affected. Thus, integrity constraint is used to guard against accidental damage to the database.

DML (Data Manipulation Language) commands permitting users to manipulate data in a database. This manipulation involves inserting data into database tables, retrieving existing data, deleting data from existing tables and modifying existing data. DML is mostly incorporated in SQL databases.

DML are of two types –

- **Procedural DMLs** : require a user to specify what data are needed and how to get those data.
- **Declarative DMLs** (also referred as Non-procedural DMLs) : require a user to specify what data are needed without specifying how to get those data.
  Declarative DMLs are usually easier to learn and use than procedural DMLs. However, since a user does not have to specify how to get the data, the database system has to figure out an efficient means of accessing data.

Here are some commands (DML) :

- SELECT: retrieve data from the database
- INSERT: insert data into a table
- UPDATE: update existing data within a table
- DELETE: deletes all records from a table, space for the records remain

TCL (Transaction Control Language): used to manage transactions in the database. These are used to manage the changes made by DML-statements. It also allows statements to be grouped together into logical transactions. Simply means this command is used to manage the changes made by DML statements.

Here are some commands(TCL):

- COMMIT: Commit command is used to permanently save any transaction into the database.
- ROLLBACK: This command restores the database to last committed state. It is also used with savepoint command to jump to a savepoint in a transaction.
- SAVEPOINT: Savepoint command is used to temporarily save a transaction so that you can rollback to that point whenever necessary.

DCL (Data Control Language) : is a syntax similar to a computer programming language used to control access to data stored in a database (Authorization). In particular, it is a component of Structured Query Language (SQL). DCL is used to control user access in a database, thus related to security issues; using this it allows or restricts the user from accessing data in database schema.

Here are some commands(DCL):

- GRANT: allow specified users to perform specified tasks.
- REVOKE: cancel previously granted or denied permissions.

1.4 Transactions

Is a sequence of operations performed (using one or more SQL statements) on a database as a single logical unit of work. The effects of all the SQL statements in a transaction can be either all committed (applied to the database) or all rolled back (undone from the database). It group a set of tasks into a single execution unit. Each transaction begins with a specific task and ends when all the tasks in the group successfully complete. If any of the tasks fail, the transaction fails. Therefore, a transaction has only two results: *success* or *failure*. A **database transaction**, by definition, must be atomic, consistent, isolated and durable.

To understand the concept of a transaction, consider a banking database. Suppose a bank customer transfers money from his savings account (SB a/c) to his overdraft account (OD a/c), the statement will be divided into four blocks:

- Debit SB a/c.
- Credit OD a/c.
- Record in Transaction Journal
- End Transaction

The SQL statement to debit SB a/c is as follows :

```
UPDATE sb_accounts
SET balance = balance - 1000
WHERE account_no = 932656 ;
```

The SQL statement to credit OD a/c is as follows :

```
UPDATE od_accounts
SET balance = balance + 1000
WHERE account_no = 933456 ;
```

The SQL statement for record in transaction journal is as follows :

```
INSERT INTO journal VALUES
(100896, 'Tansaction on Benjamin Hampshair a/c', '26-AUG-08' 932656, 933456, 1000);
```

The SQL statement for End Transaction is as follows :

```
COMMIT WORK;
```

A database transaction must be atomic, consistent, isolated and durable. Bellow we have discussed these four points.

Atomic : A transaction is a logical unit of work which must be either completed with all of its data modifications, or none of them is performed.

Consistent : At the end of the transaction, all data must be left in a consistent state.

Isolated : Modifications of data performed by a transaction must be independent of another transaction. Unless this happens, the outcome of a transaction may be erroneous.

Durable : When the transaction is completed, effects of the modifications performed by the transaction must be permanent in the system.

Often these four properties of a transaction are acronymed as **ACID**.

We have explained the above four properties of a translation with the following example :

Example of Fund Transfer

- Transaction to transfer $1000 from account X to account Y:
    1. read(X)
    2. X = X –1000
    3. write(X)
    4. read(Y)
    5. Y = Y + 1000
    6. write(Y)
- Atomicity requirement — if the transaction fails after step 3 and before step 6, the system should ensure that its updates are not reflected in the database, else an inconsistency will result.
- Consistency requirement – the sum of X and Y is unchanged by the execution of the transaction.
- If between steps 3 and 6, another transaction is allowed to access the partially updated database, it will see an inconsistent database (the sum X + Y will be less than it should be).
    1. Isolation can be ensured trivially by running transactions serially, that is one after the other.
    2. However, executing multiple transactions concurrently has significant benefits, as we will see later.

- Durability requirement — once the user has been notified that the transaction has completed (i.e., the transfer of the $1000 has taken place), the updates to the database by the transaction must persist despite failures.

Beginning a Transaction

A transaction is beginning to initiate the execution of multiple SQL statements. Beginning of a transaction guarantees the atomicity of a transaction. After beginning, either it can be committed to making the modifications permanent or rolled back to undo the changes to leave the database unaltered.

Committing a Transaction

By committing a transaction, it is closed explicitly and modifications performed by the transaction is made permanent.

Rolling Back a Transaction

By rolling back a transaction, a transaction is explicitly closed and any modifications made by the transaction is discarded.

1. **BEGIN TRANSACTION**: It indicates the start point of an explicit or local transaction.
Syntax:

- BEGIN TRANSACTION transaction_name ;

2. **SET TRANSACTION:** Places a name on a transaction.
Syntax:

- SET TRANSACTION [READ WRITE | READ ONLY];

3. **COMMIT**: If everything is in order with all statements within a single transaction, all changes are recorded together in the database is called committed. The COMMIT command saves all the transactions to the database since the last COMMIT or ROLLBACK command.
Syntax:

- COMMIT;

4. ROLLBACK: If any error occurs with any of the SQL grouped statements, all changes need to be aborted. The process of reversing changes is called rollback. This command can only be used to undo transactions since the last COMMIT or ROLLBACK command was issued.

Syntax:

- ROLLBACK;

5. SAVEPOINT: creates points within the groups of transactions in which to ROLLBACK.
A SAVEPOINT is a point in a transaction in which you can roll the transaction back to a certain point without rolling back the entire transaction.
Syntax for Savepoint command:

- SAVEPOINT SAVEPOINT_NAME;

6. RELEASE SAVEPOINT:- This command is used to remove a SAVEPOINT that you have created.
Syntax:

- RELEASE SAVEPOINT SAVEPOINT_NAME

1.5 Views is nothing more than a SQL statement that is stored in the database with an associated name. A view is actually a composition of a table in the form of a predefined SQL query. It can contain all rows of a table or select rows from a table. A view can be created from one or many tables which depends on the written SQL query to create a view. It allow users to do the following

1. Structure data in a way that users or classes of users find natural or intuitive.
2. Restrict access to the data in such a way that a user can see and (sometimes) modify exactly what they need and no more.
3. Summarize data from various tables which can be used to generate reports.

Creating Views are created using the **CREATE VIEW** statement. Views can be created from a single table, multiple tables or another view.

Syntax:

```
CREATE VIEW view_name AS
SELECT column1, column2.....
FROM table_name
WHERE condition;
```

view_name: Name for the View
table_name: Name of the table
condition: Condition to select rows

To Show :
SELECT * FROM StudentNames;

Creating Views are created using the **CREATE VIEW** statement. Views can be created from a multiple tables, multiple tables or another view.

Syntax:

```
CREATE VIEW MarksView AS
SELECT StudentDetails.NAME,
StudentDetails.ADDRESS,
StudentMarks.MARKS
FROM StudentDetails, StudentMarks
WHERE StudentDetails.NAME =
StudentMarks.NAME;
To Show :
SELECT * FROM MarksView;
```

UPDATING VIEWS

There are certain conditions needed to be satisfied to update a view. If any one of these conditions is not met, then we will not be allowed to update the view.

1. The SELECT statement which is used to create the view should not include GROUP BY clause or ORDER BY clause.
2. The SELECT statement should not have the DISTINCT keyword.
3. The View should have all NOT NULL values.
4. The view should not be created using nested queries or complex queries.
5. The view should be created from a single table. If the view is created using multiple tables then we will not be allowed to update the view.
6. The SELECT clause may not contain summary functions.
7. The SELECT clause may not contain set functions.
8. The SELECT clause may not contain set operators.
9. The SELECT clause may not contain an ORDER BY clause.
10. The FROM clause may not contain multiple tables.
11. The WHERE clause may not contain subqueries.
12. The query may not contain HAVING.
13. Calculated columns may not be updated.
14. All NOT NULL columns from the base table must be included in the view in order for the INSERT query to function.

We can use the **CREATE OR REPLACE VIEW** statement to add or remove fields from a view.

Syntax:

```
CREATE OR REPLACE VIEW view_name AS
SELECT column1,coulmn2,..
FROM table_name
WHERE condition;
```

Inserting a row in a view

```
INSERT INTO view_name(column1, column2 , column3,..)
VALUES(value1, value2, value3..);
```

view_name: Name of the View

Deleting a row from a View

```
DELETE FROM view_name
WHERE condition;
```

view_name:Name of view from where we want to delete rows
condition: Condition to select rows

NOTE: The default value of NAME column is *null*.

Uses of a View :

A good database should contain views due to the given reasons:

1. **Restricting data access –**
   Views provide an additional level of table security by restricting access to a predetermined set of rows and columns of a table.
2. **Hiding data complexity –**
   A view can hide the complexity that exists in a multiple table join.
3. **Simplify commands for the user –**
   Views allows the user to select information from multiple tables without requiring the users to actually know how to perform a join.
4. **Store complex queries –**
   Views can be used to store complex queries.
5. **Rename Columns –**
   Views can also be used to rename the columns without affecting the base tables provided the number of columns in view must match the number of columns specified in select statement. Thus, renaming helps to to hide the names of the columns of the base tables.
6. **Multiple view facility –**
   Different views can be created on the same table for different users.