What is SQL:

Structured Query language (SQL) pronounced as **"S-Q-L" or sometimes as "See-Quel"** is actually the standard language for dealing with Relational Databases.

SQL programming can be effectively used to insert, search, update, delete database records.

That doesn't mean SQL cannot do things beyond that.

In fact it can do lot of things including, but not limited to, optimizing and maintenance of databases.

What is NoSQL:

NoSQL is an upcoming category of Database Management Systems. Its main characteristic is its non-adherence to Relational Database Concepts. NOSQL means "Not only SQL".

Concept of NoSQL databases grew with internet giants such as Google, Facebook, Amazon etc who deal with gigantic volumes of data.

When you use relational database for massive volumes of data , the system starts getting slow in terms of response time.

To overcome this , we could of course "scale up" our systems by upgrading our existing hardware.

The alternative to the above problem would be to distribute our database load on multiple hosts as the load increases.

This is known as "scaling out".

NOSQL database are non-relational databases that scale out better than relational databases and are designed with web applications in mind.

They do not use SQL to query the data and do not follow strict schemas like relational models.With NoSQL, ACID (Atomicity, Consistency, Isolation, Durability) features are not guaranteed always.

Relational databases have the following advantages over NOSQL databases:

- SQL(relational) databases have a mature data storage and management model . This is crucial for enterprise users.
- SQL databases support the notion of views which allow users to only see data that they are authorized to view. The data that they are not authorized to see is kept hidden from them.
- SQL databases support stored procedure sql which allow database developers to implement part of the business logic into the database.
- SQL databases have better security models compared to NoSQL databases.

SQL:

Schema: A database schema is a way to logically group objects such as tables, views, stored procedures etc. Think of a schema as a container of objects. You can assign a user login permissions to a single schema so that the user can only access the objects they are authorized to access.

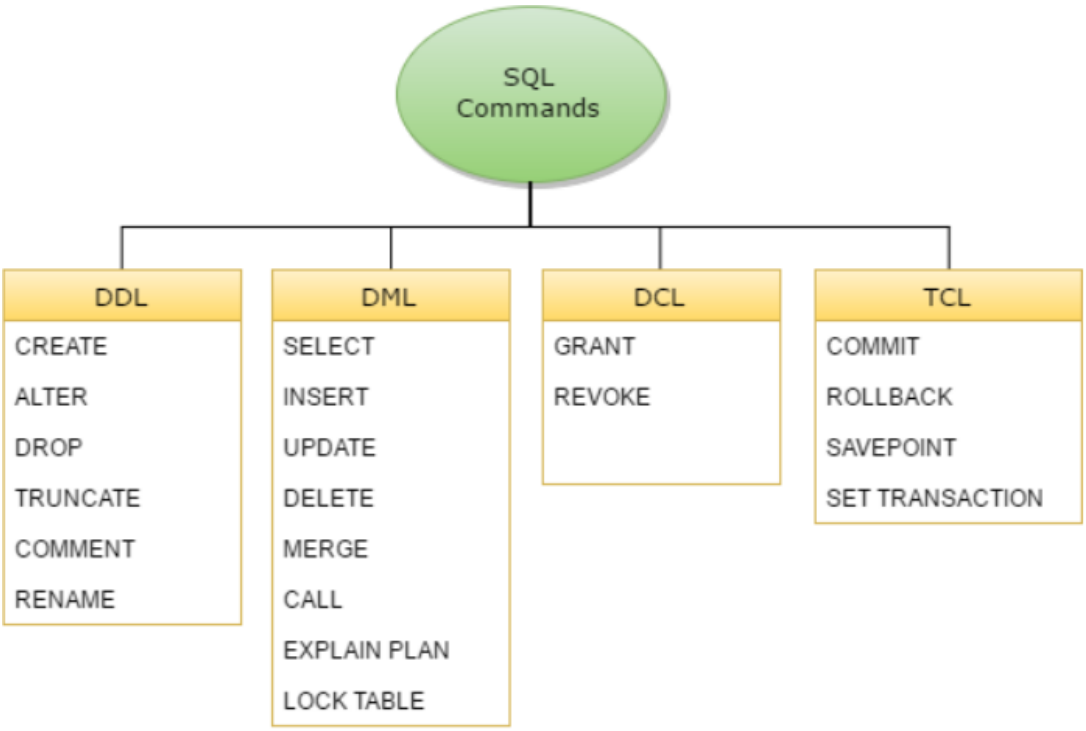
Schema refers to the organization of data as a blueprint of how the **database** is constructed.

Syntax conventions:
Case Insensitive
Write keywords uppercase letter for better practice
White space insensitive
Use statement terminator(;)

Comments:

/* this is an inline or multiline comment */

SQL Commands:



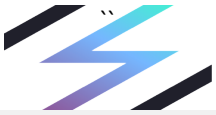
Data Type: The data type is a label and a guideline for SQL to understand what type of data is expected inside of each column, and it also identifies how SQL will interact with the stored data.

MySQL Data Types

In MySQL there are three main data types: text, number, and date.

Text data types:

Data type	**Description**



CHAR(size)	Holds a fixed length string (can contain letters, numbers, and special characters). The fixed size is specified in parenthesis. Can store up to 255 characters
VARCHAR(size)	Holds a variable length string (can contain letters, numbers, and special characters). The maximum size is specified in parenthesis. Can store up to 255 characters. **Note:** If you put a greater value than 255 it will be converted to a TEXT type
TEXT	Holds a string with a maximum length of 65,535 bytes

Number data types:

Data type	**Description**
INT(size)	-2147483648 to 2147483647 normal. 0 to 4294967295 UNSIGNED*. The maximum number of digits may be specified in parenthesis
FLOAT(size,d)	A small number with a floating decimal point. The maximum number of digits may be specified in the size parameter. The maximum number of digits to the right of the decimal point is specified in the d parameter
DOUBLE(size,d)	A large number with a floating decimal point. The maximum number of digits may be specified in the size parameter. The maximum number of digits to the right of the decimal point is specified in the d parameter
DECIMAL(size,d)	A DOUBLE stored as a string , allowing for a fixed decimal point. The maximum number of digits may be specified in the size parameter. The maximum number of digits to the right of the decimal point is specified in the d parameter

*The integer types have an extra option called UNSIGNED. Normally, the integer goes from an negative to positive value. Adding the UNSIGNED attribute will move that range up so it starts at zero instead of a negative number.

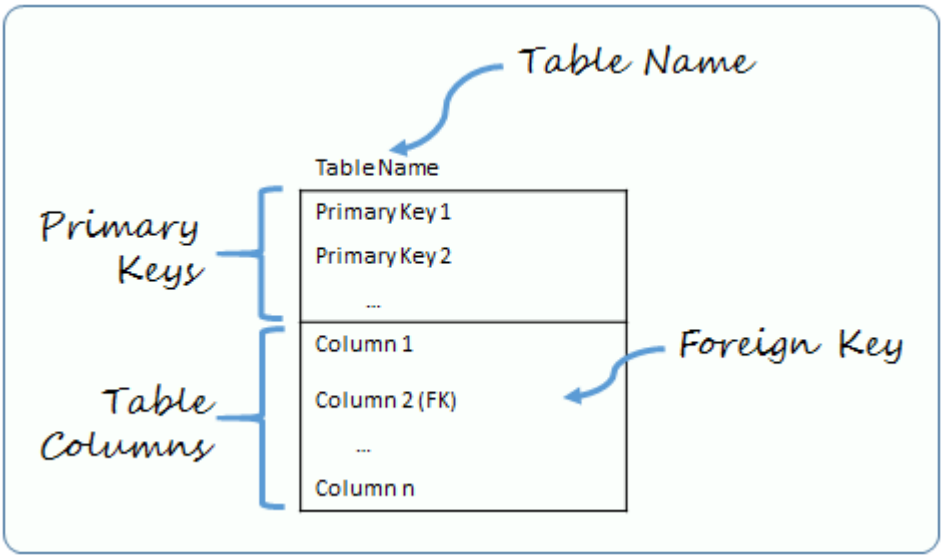
Date data types:

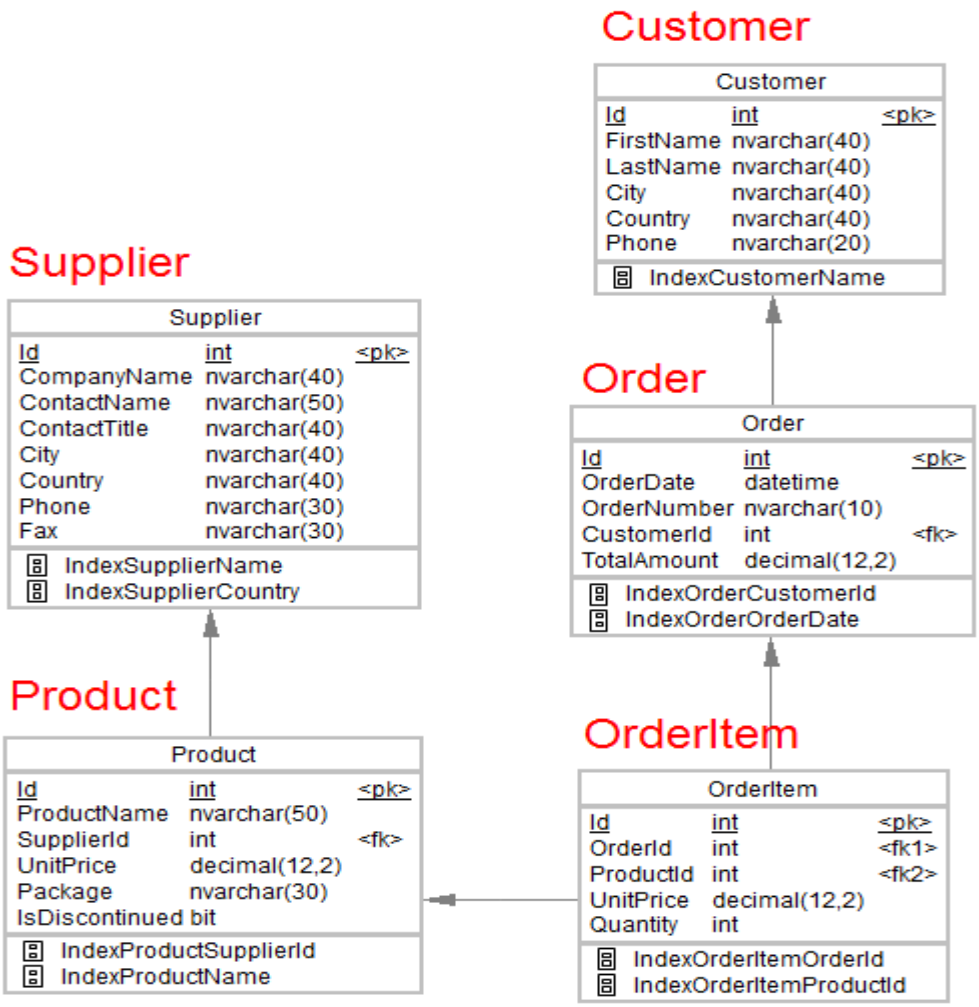
Data type	**Description**
DATE()	A date. Format: YYYY-MM-DD **Note:** The supported range is from '1000-01-01' to '9999-12-31'
DATETIME()	*A date and time combination. Format: YYYY-MM-DD HH:MI:SS **Note:** The supported range is from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'

TIMESTAMP()	*A timestamp. TIMESTAMP values are stored as the number of seconds since the Unix epoch ('1970-01-01 00:00:00' UTC). Format: YYYY-MM-DD HH:MI:SS **Note:** The supported range is from '1970-01-01 00:00:01' UTC to '2038-01-09 03:14:07' UTC
TIME()	A time. Format: HH:MI:SS **Note:** The supported range is from '-838:59:59' to '838:59:59'
YEAR()	A year in two-digit or four-digit format. **Note:** Values allowed in four-digit format: 1901 to 2155. Values allowed in two-digit format: 70 to 69, representing years from 1970 to 2069

*Even if DATETIME and TIMESTAMP return the same format, they work very differently. In an INSERT or UPDATE query, the TIMESTAMP automatically set itself to the current date and time. TIMESTAMP also accepts various formats, like YYYYMMDDHHMISS, YYMMDDHHMISS, YYYYMMDD, or YYMMDD.

Tables: It is a set of rows and columns. Table structure need to create before using it.





SQL CREATE TABLE Statement

The CREATE TABLE statement is used to create a new table in a database.

```
CREATE TABLE table_name (
    column1 datatype,
    column2 datatype,
    column3 datatype,
   ....
);
```

the column parameters specify the names of the columns of the table.

The datatype parameter specifies the type of data the column can hold (e.g. varchar, integer, date, etc.).

Let's create a table persons:

```
CREATE TABLE Persons (
    PersonID int NOT NULL,
    LastName varchar(255) NOT NULL,
```

```
    FirstName varchar(255),
    Age int
);
```

SQL INSERT INTO Statement

The INSERT INTO statement is used to insert new records in a table.

INSERT INTO Syntax

It is possible to write the INSERT INTO statement in two ways.

The first way specifies both the column names and the values to be inserted:

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

If you are adding values for all the columns of the table, you do not need to specify the column names in the SQL query. However, make sure the order of the values is in the same order as the columns in the table. The INSERT INTO syntax would be as follows:

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

Example: Let's insert data into the table persons:

```
insert into persons values (1,'Hansen','Ola',30)

insert into persons values (2,'Svendson','Svendson',23)

insert into persons values (3,'Pettersen','Kari',20)
```

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, which may consist of single or multiple fields.

SQL PRIMARY KEY on CREATE TABLE

The following SQL creates a PRIMARY KEY on the "PersonID" column when the "Persons" table is created:

```
CREATE TABLE Persons (
    PersonID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
```

```
    PRIMARY KEY (PersonID)
);
```

To allow naming of a PRIMARY KEY constraint, and for defining a PRIMARY KEY constraint on multiple columns, use the following SQL syntax:

```
CREATE TABLE Persons (
    PersonID    int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    CONSTRAINT PK_Person PRIMARY KEY (PersonID,LastName)
);
```

SQL PRIMARY KEY on ALTER TABLE

To create a PRIMARY KEY constraint on the "ID" column when the table is already created, use the following SQL:

```
ALTER TABLE tablename
ADD PRIMARY KEY (columnname);
```

Example: As we already have table persons, we alter table to add primary key on column personID

```
alter table persons

add primary key (PersonID)
```

DROP a PRIMARY KEY Constraint

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

```
ALTER TABLE tableName
DROP PRIMARY KEY;
```

Example: If want to drop the primary key on table persons

```
alter table persons

Drop primary key;
```

SQL FOREIGN KEY Constraint

A FOREIGN KEY is a key used to link two tables together.

A FOREIGN KEY is a field (or collection of fields) in one table that refers to the PRIMARY KEY in another table.

The table containing the foreign key is called the child table, and the table containing the candidate key is called the referenced or parent table.

Look at the following two tables:

"Persons" table:

PersonID	LastName	FirstName	Age
1	Hansen	Ola	30
2	Svendson	Tove	23
3	Pettersen	Kari	20

"Orders" table:

OrderID	OrderNumber	PersonID
1	77895	3
2	44678	3
3	22456	2
4	24562	1

Notice that the "PersonID" column in the "Orders" table points to the "PersonID" column in the "Persons" table.

The "PersonID" column in the "Persons" table is the PRIMARY KEY in the "Persons" table.

The "PersonID" column in the "Orders" table is a FOREIGN KEY in the "Orders" table.

The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.

The FOREIGN KEY constraint also prevents invalid data from being inserted into the foreign key column, because it has to be one of the values contained in the table it points to.

SQL FOREIGN KEY on CREATE TABLE

The following SQL creates a FOREIGN KEY on the "PersonID" column when the "Orders" table is created:

Example: As we do not have orders table, let's create using below syntax.

```
CREATE TABLE Orders(

    OrderID int NOT NULL,
```

```
    OrderNumber int NOT NULL,

    PersonID int,

    PRIMARY KEY (OrderID),

    FOREIGN KEY (PersonID) REFERENCES persons(PersonID)

   );
```

To allow naming of a FOREIGN KEY constraint, and for defining a FOREIGN KEY constraint on multiple columns, use the following SQL syntax:

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

SQL FOREIGN KEY on ALTER TABLE

To create a FOREIGN KEY constraint on the "PersonID" column when the "Orders" table is already created, use the following SQL:

```
ALTER TABLE Orders
ADD FOREIGN KEY (PersonID) REFERENCES Persons(PersonID);
```

DROP a FOREIGN KEY Constraint

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

```
ALTER TABLE Orders
DROP FOREIGN KEY foreign_key_name;
```

*Expand table structure and see the foreign_key_name on database.

SQL DROP TABLE Example

The following SQL statement drops the existing table "persons":

```
DROP TABLE tableName;
```

Example: DROP TABLE persons;

*If you have a foreign key associated with table, first we need to delete the corresponding table first or remove foreign key association first and remove the original table.

Example: Persons table is associated with Orders table with foreign key, if now we drop persons table we will get error. In this case we can do below two ways

1. First remove foreign key connection with orders table
2. Or drop Orders table (Not suggested in many cases)

SQL TRUNCATE TABLE

The TRUNCATE TABLE statement is used to delete the data inside a table, but not the table itself.

TRUNCATE TABLE *table_name*;

Example: truncate table persons

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*;

Now let's add data of birth column to our persons table

ALTER TABLE Persons

ADD DateOfBirth date;

Now look into the table structure for new column.

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 *column_name*;

Let's drop the data of birth column of persons table

ALTER TABLE Persons

DROP Column DateOfBirth;

ALTER TABLE - ALTER/MODIFY COLUMN

To change the data type of a column in a table, use the following syntax:

ALTER TABLE *table_name*
MODIFY COLUMN *column_name datatype*;

We added date of birth to persons table with datatype as date, so let's change type to year

-- As we dropped the column , re adding the column

ALTER TABLE Persons

ADD DateOfBirth date;

-- Modifying the column datatype to year

ALTER TABLE Persons

MODIFY COLUMN DateOfBirth year;

SQL SELECT Statement

The SELECT statement is used to select data from a database table.

To select all columns from a table:

SELECT * FROM *table_name*;

Example: select * from film

To select particular columns from a table

SELECT *column1*, *column2, ...*
FROM *table_name*;

Example: select title, release_year from film

SQL SELECT DISTINCT Statement

The SELECT DISTINCT statement is used to return only distinct (different) values.

Inside a table, a column often contains many duplicate values; and sometimes you only want to list the different (distinct) values.

SELECT DISTINCT *column1*, *column2, ...*
FROM *table_name*;

Example: select distinct special_features from film

SQL WHERE Clause

The WHERE clause is used to filter records.

The WHERE clause is used to extract only those records that fulfill a specified condition.

SELECT *column1*, *column2, ...*
FROM *table_name*
WHERE *condition*;

Example: select * from film where rental_duration = 3

select title,description from film where rental_duration = 3

SQL AND, OR and NOT Operators

The WHERE clause can be combined with AND, OR, and NOT operators.

The AND and OR operators are used to filter records based on more than one condition:

- The AND operator displays a record if all the conditions separated by AND are TRUE.
- The OR operator displays a record if any of the conditions separated by OR is TRUE.

The NOT operator displays a record if the condition(s) is NOT TRUE.

AND Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 AND condition2 AND condition3 ...;
```

Example: select * from film where rental_duration = 3 and rating ='G'

OR Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 OR condition2 OR condition3 ...;
```

Example: select * from film where rental_duration = 3 or rating ='G'

NOT Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE NOT condition;
```

Example:select * from film where NOT replacement_cost = 22.99

SQL SELECT TOP Clause

The SELECT TOP clause is used to specify the number of records to return.

The SELECT TOP clause is useful on large tables with thousands of records. Returning a large number of records can impact on performance.

```
SELECT column_name(s)
FROM table_name
WHERE condition
LIMIT number;
```

Example:select title

from film

where rental_duration = 3

limit 5

SQL UPDATE Statement

The UPDATE statement is used to modify the existing records in a table.

UPDATE *table_name*
SET *column1* = *value1*, *column2* = *value2*, ...
WHERE *condition*;

Example: select * from film where replacement_cost = 26.99

update film set replacement_cost = 24.99 where replacement_cost = 26.99

Aggregations:

Famous aggregate functions available are MIN, MAX,COUNT, AVG,SUM.

MIN() function returns the smallest value of the selected column.

Syntax

SELECT MIN(*column_name*)
FROM *table_name*
WHERE *condition*;

Example: select distinct rental_rate from film where rental_duration = 6

select min(rental_rate) from film where rental_duration = 6

MAX() function returns the largest value of the selected column.

Syntax

SELECT MAX(*column_name*)
FROM *table_name*
WHERE *condition*;

Example: select Max(rental_rate) from film where rental_duration = 6

COUNT() function returns the number of rows that matches a specified criteria.

Syntax

SELECT COUNT(*column_name*)
FROM *table_name*
WHERE *condition*;

Example: select count(*) from film where rental_duration = 6

select count(distinct rental_rate) as count from film where rental_duration = 6

AVG() function returns the average value of a numeric column.

Syntax

SELECT AVG(*column_name*)
FROM *table_name*
WHERE *condition*;

Example: select avg(rental_rate) as average from film where rental_duration = 6

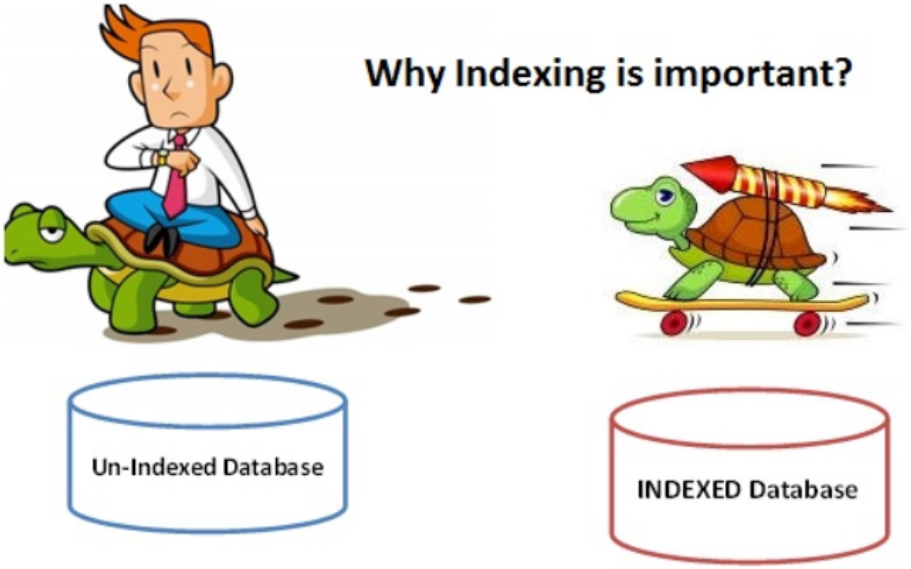
SUM() function returns the total sum of a numeric column.

Syntax

SELECT SUM(*column_name*)
FROM *table_name*
WHERE *condition*;

Example: select sum(rental_rate) as Total from film where rental_duration = 6

Index: It increase the query performance of database



- The slowness in the response time is usually due to the records being stored randomly in database tables.
- Search queries have to loop through the entire randomly stored records one after the other to locate the desired data.
- This results in poor performance databases when it comes to retrieving data from large tables
- Indexes come in handy in such situations. Indexes sort data in an organized sequential way.Think of an index as an alphabetically sorted list. It is easier to lookup names that have been sorted in alphabetical order than ones that are not sorted.
- INDEX's are created on the column(s) that will be used to filter the data.

- Using indexes on tables that are frequently updated can result in poor performance. This is because SQL creates a new index block every time that data is added or updated in the table. Generally, indexes should be used on tables whose data does not change frequently but is used a lot in select search queries.

Refer this link for more available functions in SQL:
https://www.w3schools.com/sql/default.asp