TUTORIAL ON **SQL**

Ritwik Raj
PhD Candidate
Industrial & Systems Engineering

University at Buffalo The State University of New York

Introduction

- Stands for Structured Query Language
- A standard language for storing, manipulating and retrieving data in databases
- Can be used in the following database systems: MySQL, SQL Server, MS Access, Oracle, Sybase, Informix, Postgres, etc.
- Installation:
  https://www.microsoft.com/en-us/sql-server/sql-server-downloads

- In this tutorial, the focus is on learning how to write SQL queries

Resources:

- https://www.w3schools.com/sql/default.asp
- https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_all
- https://en.wikipedia.org/wiki/SQL

What Can SQL do?

SQL can

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

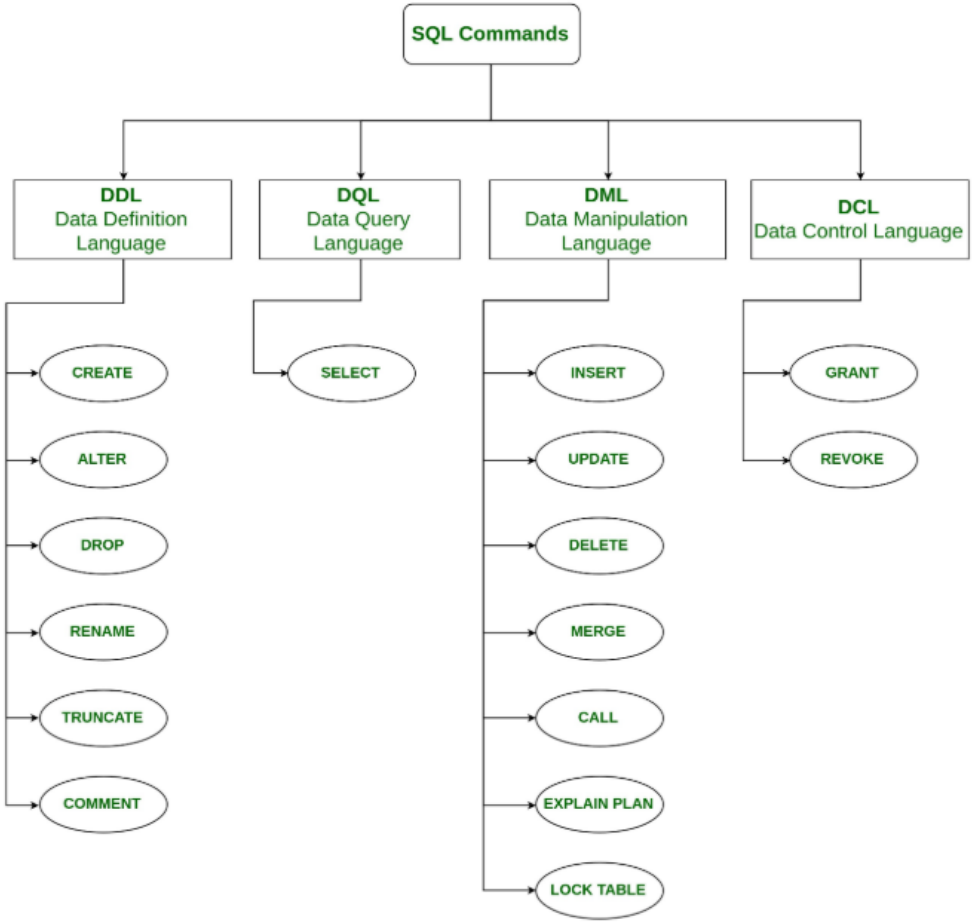
Database Tables

- A table is an object
- A database most often contains one or more tables
- Each table is identified by a unique name (e.g. "Customers" or "Orders")
- Columns have various attributes, such as column name and data type
- Rows contain records or data for the columns
- For this tutorial, the database available at w3schools.com has been used, and it has following tables:
    - Customers
    - Categories
    - Employees
    - OrderDetails
    - Orders
    - Products
    - Shippers
    - Suppliers

A Sample “Customers” Table:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

SQL functions fit into four broad categories:



A few things to note:

- SQL keywords are NOT case sensitive: select is the same as SELECT
- Some database systems require a semicolon at the end of each SQL statement
- Semicolon is the standard way to separate each SQL statement in database systems that allow more than one SQL statement to be executed in the same call to the server

SELECT

- The SELECT statement is used to select data from a database
- The data returned is stored in a result table, called the result-set
- Syntax Examples:
  - SELECT CustomerName, City FROM Customers;
  - SELECT * FROM Customers;

SELECT DISTINCT

- 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
- Syntax Examples:
  - SELECT DISTINCT Country FROM Customers;
  - SELECT COUNT(DISTINCT Country) FROM Customers;

WHERE

- The WHERE clause is used to filter records
- The WHERE clause is used to extract only those records that fulfill a specified condition
- Syntax Examples:
  - SELECT * FROM Customers WHERE Country='Mexico';
  - SELECT * FROM Customers WHERE CustomerID=1;

Operators in The WHERE Clause

Operator	Description
=	Equal
<>	Not equal. **Note:** In some versions of SQL this operator may be written as !=
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
BETWEEN	Between an inclusive range
LIKE	Search for a pattern
IN	To specify multiple possible values for a column

AND, OR and NOT

- The WHERE clause can be combined with AND, OR, and NOT operators
- The AND operator displays a record if all the conditions separated by AND is 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
- Syntax Examples:
  - SELECT * FROM Customers WHERE Country='Germany' AND City='Berlin';
  - SELECT * FROM Customers WHERE City='Berlin' OR City='München';
  - SELECT * FROM Customers WHERE NOT Country='Germany';
  - SELECT * FROM Customers
    WHERE Country='Germany' AND (City='Berlin' OR City='München');
  - SELECT * FROM Customers
    WHERE NOT Country='Germany' AND NOT Country='USA';

ORDER BY

- The ORDER BY keyword is used to sort the result-set in ascending or descending order
- The ORDER BY keyword sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword
- Syntax Examples:
  - SELECT * FROM Customers ORDER BY Country;
  - SELECT * FROM Customers ORDER BY Country DESC;
  - SELECT * FROM Customers ORDER BY Country, CustomerName;
  - SELECT * FROM Customers ORDER BY Country ASC, CustomerName DESC;

INSERT INTO

- The INSERT INTO statement is used to insert new records in a table
- 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
  - 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
- Syntax Examples:
  - INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode, Country) VALUES ('Cardinal', 'Tom B. Erichsen', 'Skagen 21', 'Stavanger', '4006', 'Norway');
  - INSERT INTO Customers (CustomerName, City, Country) VALUES ('Cardinal', 'Stavanger', 'Norway');
  - INSERT INTO Customers VALUES ('Cardinal', 'Tom B. Erichsen', 'Skagen 21', 'Stavanger', '4006', 'Norway');

NULL

- A field with a NULL value is a field with no value.
- If a field in a table is optional, it is possible to insert a new record or update a record without adding a value to this field. Then, the field will be saved with a NULL value.
- It is not possible to test for NULL values with comparison operators, such as =, <, or <>.
- We will have to use the IS NULL and IS NOT NULL operators instead.
- Syntax Examples:
  - SELECT * FROM Customers WHERE Address IS NULL;
  - SELECT * FROM Customers WHERE Address IS NOT NULL;

UPDATE

- The UPDATE statement is used to modify the existing records in a table
- Syntax Examples:
  - UPDATE Customers SET ContactName = 'Alfred Schmidt', City= ‘Frankfurt’ WHERE CustomerID = 1;
  - UPDATE Customers SET ContactName= ‘Juan’ WHERE Country = 'Mexico';
  - UPDATE Customers SET ContactName= 'Juan'; (CAREFUL! All records will be updated.)

DELETE

- The DELETE statement is used to delete existing records in a table
- Syntax Examples:
  - DELETE FROM Customers WHERE CustomerName='Alfreds Futterkiste';

LIMIT

- MySQL supports the LIMIT clause to select a limited number of records
- Syntax Examples:
  - SELECT * FROM Customers LIMIT 3;
  - SELECT * FROM Customers WHERE Country='Germany' LIMIT 3;

MIN() and MAX()

- The MIN() function returns the smallest value of the selected column
- The MAX() function returns the largest value of the selected column
- Syntax Examples:
  - SELECT MIN(Price) AS SmallestPrice FROM Products;
  - SELECT MAX(Price) AS LargestPrice FROM Products;

COUNT(), AVG() and SUM()

- The COUNT() function returns the number of rows that matches a specified criteria
- The AVG() function returns the average value of a numeric column
- The SUM() function returns the total sum of a numeric column
- Syntax Examples:
  - SELECT COUNT(ProductID) FROM Products;
  - SELECT AVG(Price) FROM Products;
  - SELECT SUM(Quantity) FROM OrderDetails;

LIKE

- The LIKE operator is used in a WHERE clause to search for a specified pattern in a column
- There are two wildcards used in conjunction with the LIKE operator:
  - % - The percent sign represents zero, one, or multiple characters
  - _ - The underscore represents a single character
- Syntax Examples:
  - SELECT * FROM Customers WHERE CustomerName LIKE 'a%';
  - SELECT * FROM Customers WHERE CustomerName LIKE '%a';
  - SELECT * FROM Customers WHERE CustomerName LIKE '%or%';
  - SELECT * FROM Customers WHERE CustomerName LIKE '_r%';
  - SELECT * FROM Customers WHERE CustomerName LIKE 'a_%_%';
  - SELECT * FROM Customers WHERE ContactName LIKE 'a%o';
  - SELECT * FROM Customers WHERE CustomerName NOT LIKE 'a%';

IN

- The IN operator allows you to specify multiple values in a WHERE clause
- The IN operator is a shorthand for multiple OR conditions
- Syntax Examples:
  - SELECT * FROM Customers
    WHERE Country IN ('Germany', 'France', 'UK');
  - SELECT * FROM Customers
    WHERE Country NOT IN ('Germany', 'France', 'UK');
  - SELECT * FROM Customers
    WHERE Country IN (SELECT Country FROM Suppliers);

BETWEEN

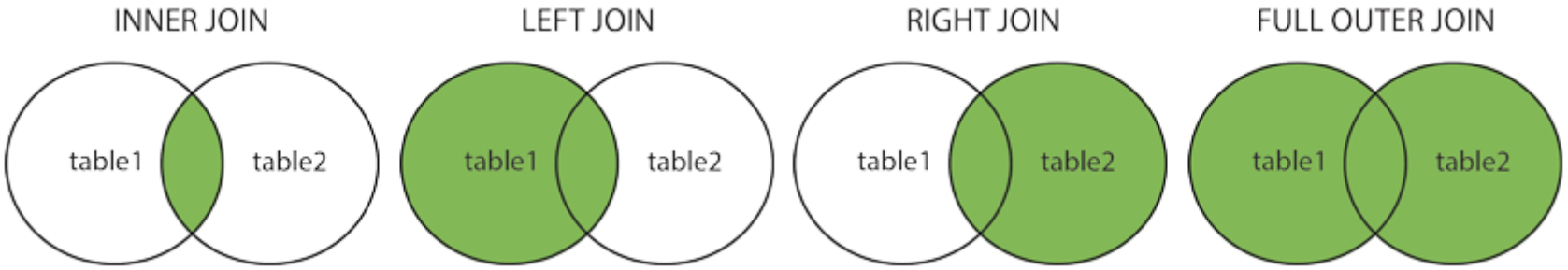
- The BETWEEN operator selects values within a given range. The values can be numbers, text, or dates.
- The BETWEEN operator is inclusive: begin and end values are included.
- Syntax Examples:
  - SELECT * FROM Products WHERE Price BETWEEN 10 AND 20;
  - SELECT * FROM Products WHERE Price NOT BETWEEN 10 AND 20;
  - SELECT * FROM Products WHERE (Price BETWEEN 10 AND 20) AND NOT CategoryID IN (1,2,3);
  - SELECT * FROM Products
    WHERE ProductName BETWEEN 'Carnarvon Tigers' AND 'Mozzarella di Giovanni‘ ORDER BY ProductName;
  - SELECT * FROM Orders WHERE OrderDate BETWEEN ‘1996-07-03’ AND ‘1996-07-04’;

Aliases

- SQL aliases are used to give a table, or a column in a table, a temporary name
- Aliases are often used to make column names more readable
- An alias only exists for the duration of the query
- Syntax Examples:
  - SELECT CustomerID as ID, CustomerName AS Customer FROM Customers;
  - SELECT CustomerName AS Customer, ContactName AS [Contact Person] FROM Customers;

JOIN

- A JOIN clause is used to combine rows from two or more tables, based on a related column between them
- Here are the different types of the JOINs in SQL:
  - (INNER) JOIN: Returns records that have matching values in both tables
  - LEFT (OUTER) JOIN: Return all records from the left table, and the matched records from the right table
  - RIGHT (OUTER) JOIN: Return all records from the right table, and the matched records from the left table
  - FULL (OUTER) JOIN: Return all records when there is a match in either left or right table



INNER JOIN

- The INNER JOIN keyword selects records that have matching values in both tables
- Syntax Examples:
  - SELECT Orders.OrderID, Customers.CustomerName FROM Orders INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID;
  - SELECT Orders.OrderID, Customers.CustomerName, Shippers.ShipperName FROM ((Orders INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID) INNER JOIN Shippers ON Orders.ShipperID = Shippers.ShipperID);

LEFT JOIN

- The LEFT JOIN keyword returns all records from the left table (table1), and the matched records from the right table (table2)
- The result is NULL from the right side, if there is no match
- Syntax Examples:
  - SELECT Customers.CustomerName, Orders.OrderID
    FROM Customers LEFT JOIN Orders
    ON Customers.CustomerID = Orders.CustomerID
    ORDER BY Customers.CustomerName;
- **Note:** The LEFT JOIN keyword returns all records from the left table (Customers), even if there are no matches in the right table (Orders).

RIGHT JOIN

- The RIGHT JOIN keyword returns all records from the right table (table2), and the matched records from the left table (table1)
- The result is NULL from the left side, when there is no match
- Syntax Examples:
  - SELECT Orders.OrderID, Employees.LastName, Employees.FirstName FROM Orders RIGHT JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID ORDER BY Orders.OrderID;
- **Note:** The RIGHT JOIN keyword returns all records from the right table (Employees), even if there are no matches in the left table (Orders).

FULL OUTER JOIN

- The FULL OUTER JOIN keyword return all records when there is a match in either left (table1) or right (table2) table records
- Syntax Examples:
  - SELECT Customers.CustomerName, Orders.OrderID
    FROM Customers
    FULL OUTER JOIN Orders ON Customers.CustomerID=Order
    .CustomerID ORDER BY Customers.CustomerName;
- **Note:** The FULL OUTER JOIN keyword returns all the rows from the left table (Customers), and all the rows from the right table (Orders). If there are rows in "Customers" that do not have matches in "Orders", or if there are rows in "Orders" that do not have matches in "Customers", those rows will be listed as well.

UNION

- The UNION operator is used to combine the result-set of two or more SELECT statements
- Each SELECT statement within UNION must have the same number of columns
- The columns must also have similar data types
- The columns in each SELECT statement must also be in the same order
- Syntax Examples:
  - SELECT City FROM Customers UNION
    SELECT City FROM Suppliers ORDER BY City;
  - SELECT City FROM Customers UNION ALL
    SELECT City FROM Suppliers ORDER BY City;
  - SELECT City, Country FROM Customers WHERE Country='Germany' UNION SELECT City, Country FROM Suppliers WHERE Country='Germany' ORDER BY City;

GROUP BY

- The GROUP BY statement is often used with aggregate functions (COUNT, MAX, MIN, SUM, AVG) to group the result-set by one or more columns
- Syntax Examples:
  - SELECT COUNT(CustomerID), Country FROM Customers GROUP BY Country;
  - SELECT COUNT(CustomerID), Country FROM Customers GROUP BY Country ORDER BY COUNT(CustomerID) DESC;
  - SELECT Shippers.ShipperName, COUNT(Orders.OrderID) AS NumberOfOrders FROM Orders
    LEFT JOIN Shippers ON Orders.ShipperID = Shippers.ShipperID GROUP BY ShipperName;

HAVING

- The HAVING clause was added to SQL because the WHERE keyword could not be used with aggregate functions
- Syntax Examples:
  - SELECT COUNT(CustomerID), Country FROM Customers GROUP BY Country HAVING COUNT(CustomerID) > 5;
  - SELECT COUNT(CustomerID), Country FROM Customers GROUP BY Country HAVING COUNT(CustomerID) > 5 ORDER BY COUNT(CustomerID) DESC;

INSERT INTO SELECT

- The INSERT INTO SELECT statement copies data from one table and inserts it into another table
- INSERT INTO SELECT requires that data types in source and target tables match
- The existing records in the target table are unaffected
- Syntax Examples:
  - INSERT INTO Customers (CustomerName, City, Country) SELECT SupplierName, City, Country FROM Suppliers;
  - INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode, Country) SELECT SupplierName, ContactName, Address, City, PostalCode, Country FROM Suppliers;

CREATE TABLE

- The CREATE TABLE statement is used to create a new table in a database
- 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.)
- Syntax Examples:
  - CREATE TABLE Persons (
    PersonID int,
    LastName varchar(255),
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255)
    );

CREATE TABLE Using Another Table

- A copy of an existing table can be created using a combination of the CREATE TABLE statement and the SELECT statement.
- The new table gets the same column definitions. All columns or specific columns can be selected.
- If you create a new table using an existing table, the new table will be filled with the existing values from the old table.
- Syntax Examples:
  - CREATE TABLE Persons AS
    SELECT LastName, FirstName, BirthDate FROM Employees
    WHERE EmployeeID < 7;

ALTER TABLE

- 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
- Syntax Examples:
  - ALTER TABLE Persons
    ADD DateOfBirth date;
  - ALTER TABLE Persons
    ALTER COLUMN DateOfBirth year;
  - ALTER TABLE Persons
    DROP COLUMN DateOfBirth;

Constraints

- SQL constraints are used to specify rules for data in a table
- Constraints can be specified when the table is created with the CREATE TABLE statement, or after the table is created with the ALTER TABLE statement
- The following constraints are commonly used in SQL:
  - NOT NULL - Ensures that a column cannot have a NULL value
  - UNIQUE - Ensures that all values in a column are different
  - PRIMARY KEY - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
  - FOREIGN KEY - Uniquely identifies a row/record in another table
  - CHECK - Ensures that all values in a column satisfies a specific condition
  - DEFAULT - Sets a default value for a column when no value is specified
  - INDEX - Used to create and retrieve data from the database very quickly

NOT NULL

- The NOT NULL constraint enforces a column to NOT accept NULL values
- This enforces a field to always contain a value, which means that you cannot insert a new record, or update a record without adding a value to this field
- Syntax Examples:
  - CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255) NOT NULL,
    Age int
    );

UNIQUE

- The UNIQUE constraint ensures that all values in a column are different
- Syntax Examples:
  - CREATE TABLE Persons (
    ID int NOT NULL UNIQUE,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int
    );
  - ALTER TABLE Persons
    ADD UNIQUE (ID);

PRIMARY KEY

- The PRIMARY KEY constraint uniquely identifies each record in a database 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
- Syntax Examples:
  - ```
    CREATE TABLE Persons (
        ID int NOT NULL PRIMARY KEY,
        LastName varchar(255) NOT NULL,
        FirstName varchar(255),
        Age int
    );
    ```
  - ```
    ALTER TABLE Persons
    ADD PRIMARY KEY (ID);
    ```

FOREIGN KEY

- 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
- Syntax Examples:
  - CREATE TABLE Orders (
    OrderID int NOT NULL PRIMARY KEY,
    OrderNumber int NOT NULL,
    PersonID int FOREIGN KEY REFERENCES Persons(PersonID)
    );
  - ALTER TABLE Orders
    ADD FOREIGN KEY (PersonID) REFERENCES Persons(PersonID);

CHECK

- The CHECK constraint is used to limit the value range that can be placed in a column
- Syntax Examples:
  - CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int CHECK (Age>=18)
    );
  - ALTER TABLE Persons
    ADD CHECK (Age>=18);

DEFAULT

- The DEFAULT constraint is used to provide a default value for a column
- The default value will be added to all new records IF no other value is specified
- Syntax Examples:
  - CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    City varchar(255) DEFAULT 'Sandnes'
    );
  - ALTER TABLE Persons
    ALTER COLUMN City SET DEFAULT 'Sandnes';

CREATE INDEX

- The CREATE INDEX statement is used to create indexes in tables.
- Indexes are used to retrieve data from the database very fast. The users cannot see the indexes, they are just used to speed up searches/queries.
- Syntax Examples:
  - CREATE INDEX idx_lastname
    ON Persons (LastName);
  - CREATE INDEX idx_pname
    ON Persons (LastName, FirstName);

AUTO INCREMENT

- Auto-increment allows a unique number to be generated automatically when a new record is inserted into a table
- Often this is the primary key field that we would like to be created automatically every time a new record is inserted
- Syntax Examples:
  - CREATE TABLE Persons (
    ID int NOT NULL AUTO_INCREMENT,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    PRIMARY KEY (ID)
    );

Thank you

Questions?