Skills for Hire
Data Analyst Program
Week 2

<u>MySQL fundamentals</u>

SQL is a database computer language designed for the retrieval and management of data in relational database. SQL stands for Structured Query Language.

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

DDL Data Definition Language:

- 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:

- INSERT Creates a record
- UPDATE Modifies records
- DELETE Deletes records

DCL Data Control Language:

- GRANT Gives a privilege to user
- REVOKE Takes back privileges granted from user

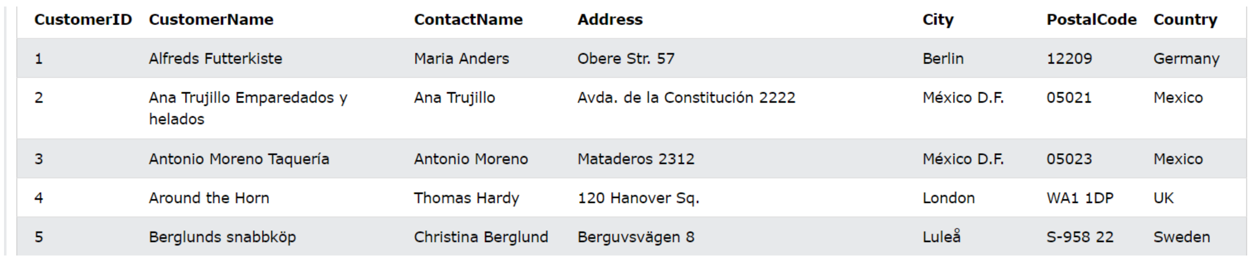
DQL Data Query Language:

- SELECT Retrieves certain records from one or more tables

Relational Database Management System.

RDBMS is the basis for SQL and for all modern database systems like MS SQL Server, IBM DB2, Oracle, MySQL, and Microsoft Access.

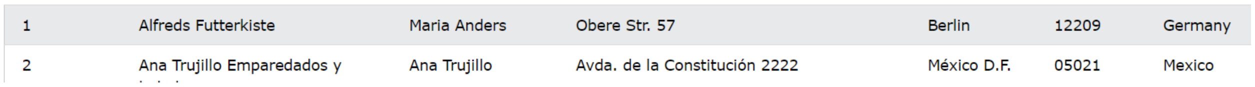
- The data in RDBMS is stored in database objects called **tables.** The table is a collection of related data entries and it consists of columns and rows. Example of a table named **CUSTOMER**



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

- Every table is broken up into smaller entities called **fields.** The fields in the CUSTOMERS table consist of CustomerID, CustomerName, ContactName, Address, City, PostalCode and Country.

- 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	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico

- 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.

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.

- 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.

SQL Syntax:

Run the following queries on either of the following sites:

- https://www.w3schools.com/sql/trysql.asp?filename=trysql_asc
- https://www.programiz.com/sql/online-compiler/

- **(You may choose any other editor or software of your choice)**

Data Types in SQL

- String – CHAR, VARCHAR
- Integer – INT
- Float – FLOAT
- Date - DATE

Basic SQL Operations

Operations	Syntax	Example (You can copy these queries written in this column on sql complier and practice)

Create a table	CREATE TABLE *table_name* **(**
	    column1 datatype NOT NULL,
	    column2 datatype,
	    column3 datatype,
	PRIMARY KEY (*column1*)
	);
		CREATE TABLE Persons (
		    PersonID int NOT NULL,
		    LastName varchar(255),
		    FirstName varchar(255),
		    Address varchar(255),
		    City varchar(255),
		   PRIMARY KEY (PersonID)
		);

Insert	INSERT INTO *table_name* (*column1*, *column2*, *column3*, ...)
	VALUES (*value1*, *value2*, *value3*, ...);
		INSERT INTO Persons (PersonID, LastName, FirstName,Address, City)
		VALUES (1,'Santiago', 'Amy', 'John Street', 'Brooklyn'),
		(2,'Cooper', 'Sheldon', 'Downing Street', 'Medford');

Alter a table	ALTER TABLE *table_name* ADD *column_name datatype*;	Alter table Persons ADD Salary int;
Drop a column	ALTER TABLE *table_name* DROP COLUMN *column_name*;	Alter table Persons DROP Column Salary;

Select	SELECT *column1*, *column2, ...*
	FROM *table_name*;
	WHERE *condition*
	ORDER BY *col list*
	LIMIT *row limit*
	
	SELECT * FROM *table_name*;
		Select LastName,FirstName from Persons where PersonID = 1;
		
		Select * from Persons;

Update	UPDATE *table_name*
	SET *column1* = *value1*, *column2* = *value2*, ...
	WHERE *condition*;
		UPDATE Persons
		SET LastName = 'Peralta', FirstName= 'Jake'
		WHERE PersonID = 1;

Delete Drop	DELETE FROM *table_name* WHERE *condition*; DROP TABLE *table_name*;	DELETE FROM Persons WHERE LastName='Peralta';

Auto Increment	CREATE TABLE *table_name* (
	    column1 int NOT NULL AUTO_INCREMENT,
	   column2 varchar(255) NOT NULL,
	   column3 varchar(255),
	    PRIMARY KEY (*column1*)
	);	

Alias	SELECT *column_name* AS *alias_name*
	FROM *table_name;*
		SELECT PersonID AS ID, LastName AS LName
		FROM Persons;

SQL Integer Operations

1. **Min/Max**
   ```
   SELECT MIN(Price) AS SmallestPrice
   FROM Products;
   SELECT MAX(Price) AS BiggestPrice
   FROM Products;
   ```

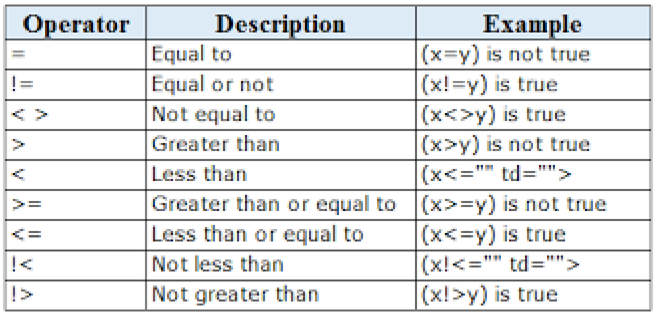
2. **ROUND() function rounds the number up or down depends upon the second argument D and number itself(digit after D decimal places >=5 or not).**
   FLOOR() function rounds the number, towards zero, always down.
   CEILING() function rounds the number, away from zero, always up.

```
Select ROUND(1.415,2),FLOOR(1.415),CEILING(1.415);

+----------------+--------------+----------------+

| ROUND(1.415,2) | FLOOR(1.415) | CEILING(1.415) |

+----------------+--------------+----------------+

|           1.42 |            1 |              2 |
```

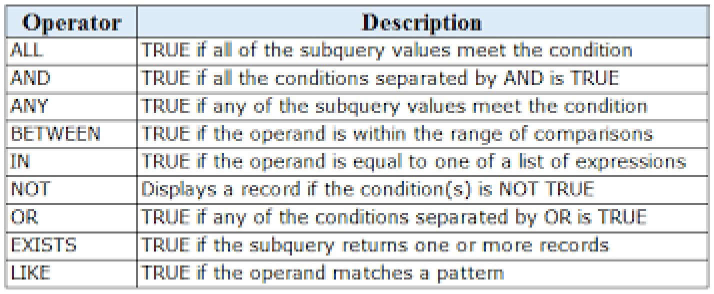
String Operations

1. Syntax: SELECT 'Geeks' || ' ' || 'forGeeks' FROM dual;
   Output: ‘GeeksforGeeks’
2. Syntax: SELECT char_length('Hello!');
   Output: 6
3. Syntax: SELECT CONCAT_WS('_', 'geeks', 'for', 'geeks');
   Output: geeks_for_geeks
4. Syntax: LENGTH('GeeksForGeeks');
   Output: 13
5. SELECT REVERSE('geeksforgeeks.org');
   Output: ‘gro.skeegrofskeeg’

SQL Where clause with comparison operators



Operator	Description	Example
=	Equal to	(x=y) is not true
!=	Equal or not	(x!=y) is true
< >	Not equal to	(x<>y) is true
>	Greater than	(x>y) is not true
<	Less than	(x<="" td="">
>=	Greater than or equal to	(x>=y) is not true
<=	Less than or equal to	(x<=y) is true
!<	Not less than	(x!<="" td="">
!>	Not greater than	(x!>y) is true



Operator	Description
ALL	TRUE if all of the subquery values meet the condition
AND	TRUE if all the conditions separated by AND is TRUE
ANY	TRUE if any of the subquery values meet the condition
BETWEEN	TRUE if the operand is within the range of comparisons
IN	TRUE if the operand is equal to one of a list of expressions
NOT	Displays a record if the condition(s) is NOT TRUE
OR	TRUE if any of the conditions separated by OR is TRUE
EXISTS	TRUE if the subquery returns one or more records
LIKE	TRUE if the operand matches a pattern

Customers

customer_id	first_name	last_name	Age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

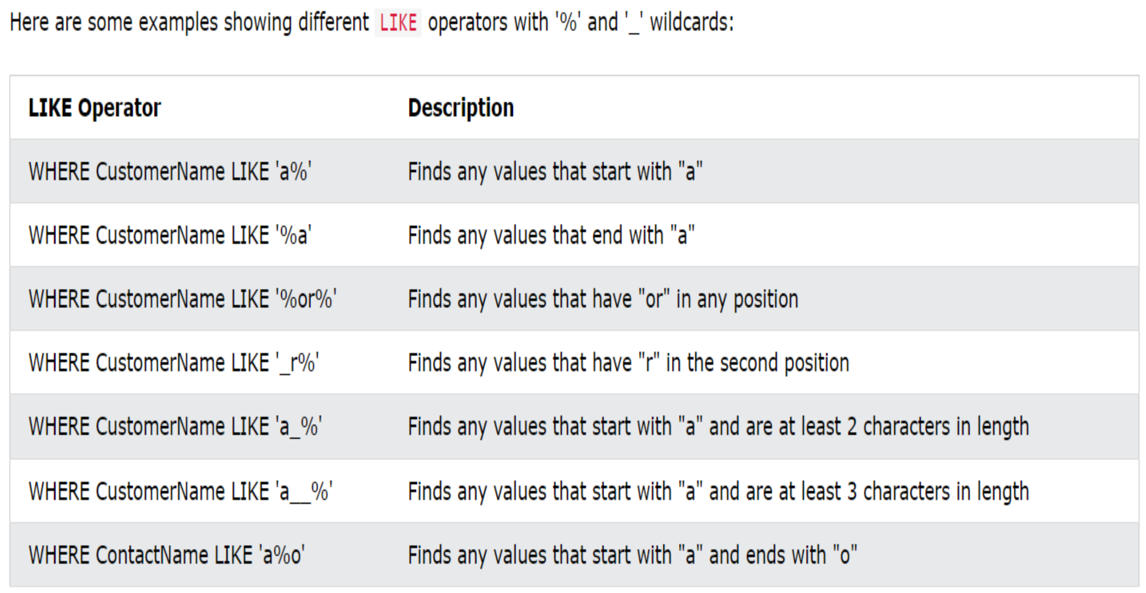
- **(Comparison Operator)** Select the ids of customers whose age is 25 and less.
  - Select customer_id from Customers where age<=25;
- **AND**
  - SELECT * FROM Customers WHERE Age=22 AND Country='UK';
- **NOT**
  - SELECT * FROM Customers WHERE NOT Country='UAE';
- **OR**
  - SELECT * FROM Customers WHERE Age=22 OR Age='31';

- **LIKE & WILD CARDS**

The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

There are two wildcards often used in conjunction with the LIKE operator:

- The percent sign (%) represents zero, one, or multiple characters
- The underscore sign (_) represents one, single character



Here are some examples showing different LIKE operators with '%' and '_' wildcards:

LIKE Operator	Description
WHERE CustomerName LIKE 'a%'	Finds any values that start with "a"
WHERE CustomerName LIKE '%a'	Finds any values that end with "a"
WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in any position
WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE CustomerName LIKE 'a_%'	Finds any values that start with "a" and are at least 2 characters in length
WHERE CustomerName LIKE 'a__%'	Finds any values that start with "a" and are at least 3 characters in length
WHERE ContactName LIKE 'a%o'	Finds any values that start with "a" and ends with "o"

Example (Output shown below)

- **Customer with first name starting from j**

  SELECT * FROM Customers WHERE first_name LIKE 'j%';

customer_id	first_name	last_name	Age	country
1	John	Doe	31	USA
4	John	Reinhardt	25	UK

- **Customers with a Last Name that starts with “R” and ends with “N”:**

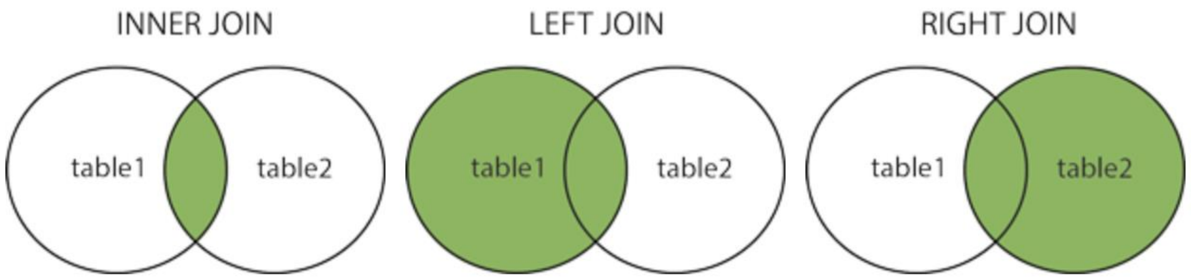
  SELECT * FROM Customers WHERE last_name LIKE 'r%n';

customer_id	first_name	last_name	Age	country
3	David	Robinson	22	UK

JOINS

Used to combine two or more tables based on a related column between them.

SELECT * FROM table1 INNER JOIN table2 ON condition;



- **INNER JOIN** Returns records that have matching values in both tables.

  ```
  SELECT column_name(s)
  FROM table1
  INNER JOIN table2
  ON table1.column_name = table2.column_name;
  ```

- **LEFT JOIN** Returns all records from the left table (table 1) and matching records from table 2. If table 1 has a record that is not in table 2, that record will still be returned.

  ```
  SELECT column_name(s)
  FROM table1
  LEFT JOIN table2
  ON table1.column_name = table2.column_name;
  ```

- **RIGHT JOIN** Returns all records from right table (table 2) and matching records from table 1. If table 2 has a record that is not in table 1, that record will still be returned. Typically, we prefer using LEFT JOIN because we write from left to right; it's just easier to think/ visualize this way because we are used to it.

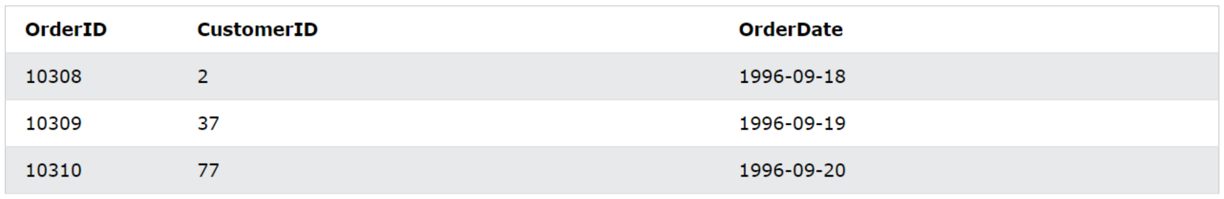
  ```
  SELECT column_name(s)
  FROM table1
  RIGHT JOIN table2
  ON table1.column_name = table2.column_name;
  ```

- **FULL (OUTER) JOIN** Returns all records when there is a match in left or right table records. It is not supported everywhere, but can be found using a combination of LEFT and RIGHT join with the UNION operator.

  ```
  ELECT column_name(s)
  FROM table1
  FULL OUTER JOIN table2
  ```

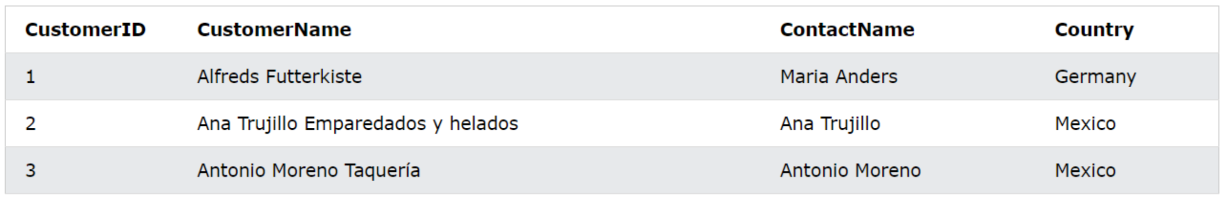
```
    ON table1.column_name = table2.column_name
    WHERE condition;
```

Examples (Live Demo):



OrderID	CustomerID	OrderDate
10308	2	1996-09-18
10309	37	1996-09-19
10310	77	1996-09-20

Then, look at a selection from the "Customers" table:



CustomerID	CustomerName	ContactName	Country
1	Alfreds Futterkiste	Maria Anders	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mexico

- ### Inner Join

```
SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate
FROM Orders
INNER JOIN Customers ON Orders.CustomerID=Customers.CustomerID;
```

- ### Left Join

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID
ORDER BY Customers.CustomerName;
```

- ### Right Join

```
SELECT Orders.OrderID, Employees.LastName, Employees.FirstName
FROM Orders
RIGHT JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID
ORDER BY Orders.OrderID;
```