SQL FOR DATA SCIENCE

SQL: Structured Query Language is a standard computer language for relational database management and data manipulation. SQL is a non procedural language it is a declarative language.

Relational database management systems: Microsoft SQL server, MySQL, PostgreSQL, SQLite, … (All of them has some differences in their sintaxis)

Data Model Building Blocks.
Entity: Person, things or events, unique and distinct.
Attribute: A characteristic of an entity**.**
Relationship: Describe association between entities:

- One to many
- Many to many
- One to One

SELECT

We need to specify what you want and where you want to select ir from.

```
SELECT pro_name FROM products;
```

```
SELECT pro_name
FROM products
LIMIT 100;
```

Oracle

```
SELECT pro_name
FROM products
WHERE ROWNUM 100;
```

Creating tables

```
CREATE TABLE student (
  ID char(8) PRIMARY KEY
  name varchar(15) NOT NULL
  );
```

Insert vales

```
INSERT INTO student (ID, name) VALUES
           (123, ‘Esteban’),
           (456, ‘Heber’;
```

Showing databases

```
SHOW DATABASES;
```

Creating databases

```
CREATE DATABASE mydb;
```

Deleting databases

```
DROP DATABASES mydb;
```

Using databases

```
USE mydb;
```

Creating temporary table

Temporary tables can be used to store intermediate results of complex queries, which can improve query performance by reducing the amount of data that needs to be processed.

```
CREATE TEMPORARY TABLE students_temp AS (
  SELECT * FROM students
  WHERE ID = 123
);
```

Adding comments.

single line: - - hello
section: /* hello */

WHERE

WHERE clause is used to filter the results of a SELECT, UPDATE, or DELETE statement. It specifies a condition that must be met for a row to be included in the result set or affected by the statement.

```
SELECT column_name, column_name
FROM table_name
WHERE column_name operator value;
```

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
IS NULL	Is a null value

Example

```
- - creating table
CREATE TABLE student(
    ID char(10) PRIMARY KEY,
    name varchar(10),
    age INTEGER NOT NULL
  );
```

```
- - inserting some records
INSERT INTO student (ID, name, age) VALUES
    ('S001', 'John', 20),
    ('S002', 'Jane', 22),
    ('S003', 'Mike', 25),
    ('S004', 'Mary', 21),
    ('S005', 'David', 24)
```

```
- - showing everything
SELECT * FROM student;
```

ID	name	age
S001	John	20
S002	Jane	22
S003	Mike	25
S004	Mary	21
S005	David	24

```
- - using WHERE clause
SELECT name
FROM student
WHERE ID = 'S001';
```

name
John

```
- - another example
SELECT name, age
FROM student
WHERE age BETWEEN 20 AND 22;
```

name	age
John	20
Jane	22
Mary	21

IN Operator

```
SELECT name, age
FROM student
WHERE age IN (20, 22, 25);
```

name	age
John	20
Jane	22
Mike	25

NOT operator

```
SELECT *
FROM employess
WHERE NOT city = ‘London’ AND
NOT city = ‘Seattle’ :
```

WildCards in SQL

Wildcard	Action
‘%Pizza’	Grabs anything ending with the word pizza
‘Pizza%’	Grabs anything after the word pizza
‘%Pizza%’	Grabs anything before and after the word pizza

The underscore (_) wildcard in SQL is used as a single-character wildcard. It can be used in a LIKE statement to match any single character in a string. For example, the following SQL statement would return all rows where the 'Name' column starts with the letter 'S':

```
SELECT * FROM table_name\
WHERE Name LIKE 'S_%';
```

This query will match the names starting with 'S' and have any single character after that

ORDER BY

The ORDER BY clause in SQL is used to sort the results of a query in ascending or descending order based on one or more columns. The basic syntax for the ORDER BY clause is

```
SELECT * FROM Employees
ORDER BY LastName DESC, FirstName ASC;
```

Math Operator.

SQL supports a variety of mathematical operators that can be used in queries to perform mathematical calculations on columns or values.

```
SELECT EmployeeID, (Salary - OldSalary) / OldSalary * 100 AS SalaryIncrease
FROM Employees;
```

Aggregate Functions

Aggregate functions in SQL are used to perform calculations on multiple rows and return a single value.

Function	Description
AVG ()	Averages a column of values
COUNT ()	Counts the number of values
MIN ()	Finds the minimum value
MAX ()	Finds the maximum value
SUM ()	Sums the column values

AVG() : returns the average of all the values in a specified column, NULL values are ingonored.

```
SELECT AVG(Salary) AS AverageSalary
FROM Employees;
```

COUNT(*) : counts all the rows in a table containig values or NULL values
COUNT(colum_name) : counts all the rows in a specific column ignoring NULL values.

```
-- example 1
SELECT COUNT(*) AS total
FROM Customers;

-- example 2
SELECT COUNT(CustomerID) AS total
FROM Cutomers;
```

SUM(): returns the sum of all the values in a specified column

```
SELECT SUM(Salary) AS TotalSalary FROM Employees;
```

MAX() : returns the maximum value in a specified column

```
SELECT Name, MAX(Price) as Price FROM Products;
```

COUNT (DISTINCT): he DISTINCT keyword can be added to the function to only count unique values.

```
SELECT COUNT(DISTINCT name) FROM customers;
```

Grouping Data

GROUP BY clause is used to group rows in a table based on one or more columns. The aggregate functions, such as SUM(), AVG(), COUNT() etc., can then be used to perform calculations on the grouped data.

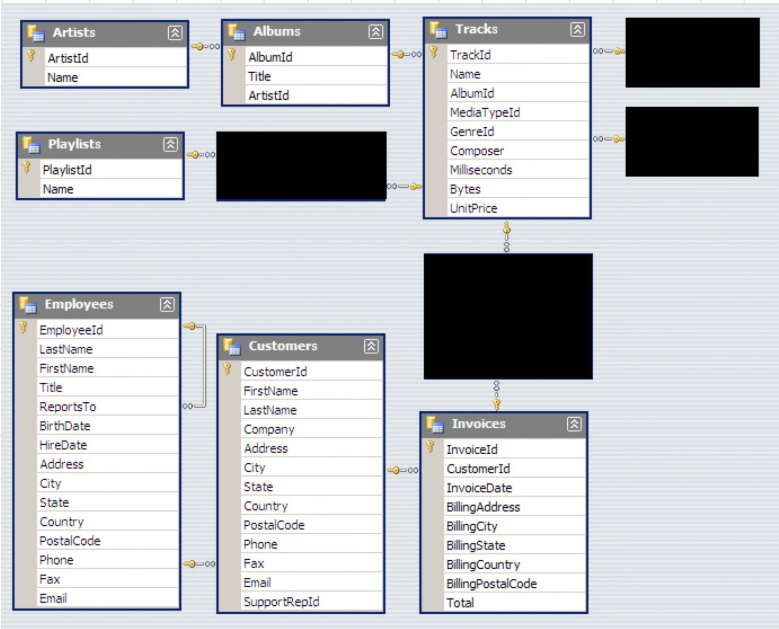
```
SELECT category, SUM(sales)
FROM sales
GROUP BY category;
```

HAVING

```
SELECT category, SUM(sales)
FROM sales
GROUP BY category
HAVING SUM(sales) > 1000;
```

Key SQL Clauses

Clause	Description	Required
SELECT	Columns or expressions to be returned	Yes
FROM	Table from which to retrieve data	Only if selecting data from a table
WHERE	Row-level filtering	No
GROUP BY	Group specification	Only if calculating aggregates by group
HAVING	Group-level filter	No
ORDER BY	Output sort order	No



Run Query: Find all the tracks that have a length of 5,000,000 milliseconds or more.

```
SELECT *
FROM Tracks
WHERE Millisecons >= 5000000;
```

Run Query: Find all the invoices whose total is between $5 and $15 dollars.

```
SELECT *
FROM Invoices
WHERE Total BETWEEN 5 AND 15;
```

Run Query: Find all the customers from the following States: RJ, DF, AB, BC, CA, WA, NY.

```
SELECT *
FROM Customers
WHERE State in ('RJ', 'DF', 'AB', 'BC', 'CA', 'WA', 'NY')
```

Run Query: Find all the invoices for customer 56 and 58 where the total was between $1.00 and $5.00.

```
SELECT *
FROM Invoices
WHERE (Total BETWEEN 1 AND 5) AND (CustomerId = 58 OR CustomerId = 56)
```

Run Query: Find all the tracks whose name starts with 'All'.

```
SELECT *
FROM Tracks
WHERE Name like 'All%'
```

Run Query: Find the albums with 12 or more tracks.

```
SELECT *, COUNT(AlbumId)
FROM Tracks
GROUP BY AlbumId
HAVING COUNT (*) >= 12
```