What is SQL

- Structured Query Language
- Method of accessing and retrieving data in relational databases
- MySQL is Oracle's Open-Source database management software (RDBMS)

Terms

- Querying - retrieves data from a table
- Output - data returned froma SQL query
- Table - this is what stores the data. It is made of columns and rows
- Database - this stores multiple tables as well as views, stored procedures, and functions
- ETL (Extract, transform, load)

SELECT

- * = everything
- Syntax:
    - *SELECT column1, column2, ...*
    - *FROM table_name;*
- ; = end of query

SELECT DISTINCT

- The SELECT DISTINCT statement is used to return only distinct (different) values.
- Syntax:
    - *SELECT DISTINCT column1, column2, ...*
    - *FROM table_name;*

WHERE Clause

- The WHERE clause is used to filter records.
- It is used to extract only those records that fulfill a specified condition.
- Syntax:
    - SELECT column1, column2, ...
    - FROM table_name

    - WHERE condition;
- Works with =, <, >, !, etc
- Works with logical operators: AND, OR, etc
- Still uses PEMDAS
- % anything before or after
    - ‘a%’ = starts with a
    - ‘%a’ = ends with a
    - ‘%a%’ = has a anywhere
- _ number of char before or after
    - ‘_a’ = has one character before a
    - ‘a__’ = has two chars after a

Group By

- The GROUP BY statement groups rows that have the same values into summary rows, like "find the number of customers in each country".
- 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:
    - SELECT COUNT(CustomerID), Country
    - FROM Customers
    - GROUP BY Country;

Order By

- The ORDER BY keyword is used to sort the result-set in ascending or descending order.
- Syntax:
    - SELECT column1, column2, ...
    - FROM table_name
    - ORDER BY column1, column2, ... ASC|DESC;

Having vs Where

- The HAVING clause was added to SQL because the WHERE keyword cannot be used with aggregate functions such as AVG, MAX, etc
- HAVING is placed **after** GROUP BY
- Syntax:
    - SELECT column_name(s)

    - FROM table_name
    - WHERE condition
    - GROUP BY column_name(s)
    - HAVING condition
    - ORDER BY column_name(s);

Limit

- The LIMIT clause is used to specify the number of records to return.
- The LIMIT clause is useful on large tables with thousands of records. Returning a large number of records can impact performance.
- Syntax:
    - SELECT column_name(s)
    - FROM table_name
    - WHERE condition
    - LIMIT number;

Aliasing

- 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 that query.
- An alias is created with the AS keyword.
- Syntax when alias used on **column**:
    - SELECT column_name AS alias_name
    - FROM table_name;
- Syntax when alias used on **table**:
    - SELECT column_name(s)
    - FROM table_name AS alias_name;

Inner Join

- The INNER JOIN keyword selects records that have matching values in both tables.
- Syntax:
    - SELECT column_name(s)
    - FROM table1
    - INNER JOIN table2
    - ON table1.column_name = table2.column_name;

Left Join

- The LEFT JOIN keyword returns all records from the left table (table1), and the matching records (if any) from the right table (table2).
- Syntax:
    - SELECT column_name(s)
    - FROM table1
    - LEFT JOIN table2
    - ON table1.column_name = table2.column_name;

Right Join

- The RIGHT JOIN keyword returns all records from the right table (table2), and the matching records (if any) from the left table (table1).
- Syntax:
    - SELECT column_name(s)
    - FROM table1
    - RIGHT JOIN table2
    - ON table1.column_name = table2.column_name;

Self Join

- A self join is a regular join, but the table is joined with itself.
- Syntax:
    - SELECT column_name(s)
    - FROM table1 T1, table1 T2
    - WHERE condition;

Union

- The UNION operator is used to combine the result-set of two or more SELECT statements.
- Every SELECT statement within UNION must have the same number of columns
- The columns must also have similar data types
- The columns in every SELECT statement must also be in the same order
- Syntax:

    - SELECT column_name(s) FROM table1
    - UNION
    - SELECT column_name(s) FROM table2;
- **UNION ALL** allows duplicate values

Length

- The LENGTH() function returns the length of a string (in bytes).
- Syntax: LENGTH(string)

Upper/Lower

- The LOWER() function converts a string to lower-case.
- The UPPER() function converts a string to upper-case.
- Syntax: LOWER(text)/UPPER(text)

Trim/Left Trim/Right Trim

- The TRIM() function removes leading and trailing spaces from a string.
- The LTRIM() function removes leading spaces from a string.
- The RTRIM() function removes trailing spaces from a string.
- Syntax: L/R/TRIM(string)

Left/Right

- The LEFT() function extracts a number of characters from a string (starting from left).
- The RIGHT() function extracts a number of characters from a string (starting from right).
- Syntax: LEFT(string, number_of_chars), RIGHT(string, number_of_chars)

Substring

- The SUBSTRING() function extracts a substring from a string (starting at any position).
- Syntax: SUBSTRING(string, start, length) or SUBSTRING(string FROM start FOR length)

Locate

- The LOCATE() function returns the position of the first occurrence of a substring in a string.
- If the substring is not found within the original string, this function returns 0.
- This function performs a case-insensitive search.
- Syntax: LOCATE(substring, string, start)

Concat

- The CONCAT() function adds two or more expressions together.
- Syntax: CONCAT(expression1, expression2, expression3,...)

Case

- The CASE expression goes through conditions and returns a value when the first condition is met (like an if-then-else statement). So, once a condition is true, it will stop reading and return the result. If no conditions are true, it returns the value in the ELSE clause.
- If there is no ELSE part and no conditions are true, it returns NULL.
- Syntax:
    - CASE
    - WHEN condition1 THEN result1
    - WHEN condition2 THEN result2
    - WHEN conditionN THEN resultN
    - ELSE result
    - END;

Subqueries

- a Subquery can be simply defined as a query within another query. In other words we can say that a Subquery is a query that is embedded in WHERE clause of another SQL query.
- You can place the Subquery in a number of SQL clauses: WHERE clause, HAVING clause, FROM clause. Subqueries can be used with SELECT, UPDATE, INSERT, DELETE statements along with expression operator. It could be equality operator or comparison operator such as =, >, =, <= and Like operator.
- A subquery is a query within another query. The outer query is called as main query and inner query is called as subquery.

- The subquery generally executes first when the subquery doesn't have any co-relation with the main query, when there is a co-relation the parser takes the decision on the fly on which query to execute on precedence and uses the output of the subquery accordingly.
- Subquery must be enclosed in parentheses.
- Subqueries are on the right side of the comparison operator.
- Use single-row operators with single row Subqueries. Use multiple-row operators with multiple-row Subqueries.
- Syntax:
    - SELECT column_name [, column_name]
    - FROM table1 [, table2]
    - WHERE column_name
    - OPERATOR (SELECT column_name [,column_name] FROM table1 [, table2] [WHERE]);

Window Functions (over and partition by)

- Window functions apply to aggregate and ranking functions over a particular window (set of rows). OVER clause is used with window functions to define that window. OVER clause does two things:
    - Partitions rows to form a set of rows. (PARTITION BY clause is used)
    - Orders rows within those partitions into a particular order. (ORDER BY clause is used)
- Syntax:
    - SELECT column_name1,
    - window_function(column_name2)
    - OVER([PARTITION BY column_name1] [ORDER BY column_name3]) AS new_column
    - FROM table_name;
    -
    -
    - window_function= any aggregate or ranking function
    - column_name1= column to be selected
    - column_name2= column on which window function is to be applied
    - column_name3= column on whose basis partition of rows is to be done
    - new_column= Name of new column
    - table_name= Name of table

Ranking Window Functions

- RANK() – As the name suggests, the rank function assigns rank to all the rows within every partition. Rank is assigned such that rank 1 given to the first row and rows having same value are assigned same rank. ***For the next rank after two same rank values, one rank value will be skipped.*** For instance, if two rows share rank 1, the next row gets rank 3, not 2.
- DENSE_RANK() – It assigns rank to each row within partition. Just like rank function first row is assigned rank 1 and rows having same value have same rank. The difference between RANK() and DENSE_RANK() is that in DENSE_RANK(), for ***the next rank after two same rank, consecutive integer is used, no rank is skipped***.
- ROW_NUMBER() – gives each row a unique number. It numbers rows from one to the total rows. The rows are put into groups based on their values. Each group is called a partition. In each partition, rows get numbers one after another. No two rows have the same number in a partition. This makes ROW_NUMBER() different from RANK() and DENSE_RANK(). ROW_NUMBER() ***uniquely identifies every row with a sequential integer number***. This helps with different kinds of data analysis.

CTE (common table expression)

- A Common Table Expression (CTE) in SQL is a temporary result set that can be referenced within a SELECT, INSERT, UPDATE, or DELETE statement. CTEs are defined using the WITH keyword and allow you to create a named, reusable subquery within your SQL statement. They provide a way to simplify complex queries and make them more readable.
- You would use a CTE when you want to:
    - Simplify complex queries: Break down a complex SQL statement into smaller, more manageable parts to improve readability and maintainability.
    - Avoid duplicating subqueries: Reuse a result set across multiple parts of a query without rewriting the same subquery multiple times.
    - Create recursive queries: When you need to perform recursive operations, such as traversing hierarchical data structures.
    - Improve query organization: Organize your SQL statements by separating logical sections, making it easier to understand and debug.
- Syntax:
    - WITH cte_name (column1, column2, ...) AS (
    - SELECT ...
    - FROM ...
    - WHERE ...
    -)

- Example:
    - WITH CTE_Example AS
    - (
    - SELECT gender, AVG(salary) avg_sal, MAX(salary) max_sal, MIN(salary) min_sal, COUNT(salary) count_sal
    - FROM employee_demographics dem
    - JOIN employee_salary sal
    - ON dem.employee_id = sal.employee_id
    - GROUP BY gender
    -)
    - SELECT AVG(avg_sal) # Average salary for male and female
    - FROM CTE_Example
    - ;

Temp Table

- Temporary Tables are most likely as Permanent Tables. Temporary Tables are Created in TempDB and are automatically deleted as soon as the last connection is terminated. Temporary Tables helps us to store and process intermediate results. Temporary tables are very useful when we need to store temporary data.
- Syntax:
    - CREATE TABLE #EmpDetails (id INT, name VARCHAR(25))
- Example:
    - CREATE TABLE temp_table
    - (first_name varchar(50),
    - last_name varchar(50),
    - favorite_movie varchar(100)
    -);
    -
    - INSERT INTO temp_table #Insert into temp table
    - VALUES('Alex', 'Freberg', 'Lord of the Rings')
    - ;

Stored Procedures

- A stored procedure is a prepared SQL code that you can save, so the code can be reused over and over again.
- So if you have an SQL query that you write over and over again, save it as a stored procedure, and then just call it to execute it.

- You can also pass parameters to a stored procedure, so that the stored procedure can act based on the parameter value(s) that is passed.
- Syntax:
    - CREATE PROCEDURE procedure_name
    - AS
    - sql_statement
    - GO;
    - EXEC procedure_name;
- Stored Procedure With One Parameter - The following SQL statement creates a stored procedure that selects Customers from a particular City from the "Customers" table:
    - CREATE PROCEDURE SelectAllCustomers @City nvarchar(30)
    - AS
    - SELECT * FROM Customers WHERE City = @City
    - GO;
    - EXEC SelectAllCustomers @City = 'London';

Triggers

- A trigger is a stored procedure in a database that automatically invokes whenever a special event in the database occurs. For example, a trigger can be invoked when a row is inserted into a specified table or when specific table columns are updated. In simple words, a trigger is a collection of SQL statements with particular names that are stored in system memory. It belongs to a specific class of stored procedures that are automatically invoked in response to database server events. Every trigger has a table attached to it.
- The following are the key differences between triggers and stored procedures:
    - Triggers cannot be manually invoked or executed.
    - There is no chance that triggers will receive parameters.
    - A transaction cannot be committed or rolled back inside a trigger.
- Syntax:
    - create trigger [trigger_name]
    - [before | after]
    - {insert | update | delete}
    - on [table_name]
    - [for each row]
    - [trigger_body]
- Example:
    - DELIMITER $$
    - CREATE TRIGGER employee_ insert
    - AFTER INSERT ON employee_salary
    - FOR EACH ROW

- BEGIN
- INSERT INTO employee_demographics (employee_id, first_name, last_name)
- VALUES (NEW.employee_id, NEW.first _name, NEW. last _name);
- END $$
- DELIMITER ;
-
- INSERT INTO employee_salary (employee_id, first_name, last. _name, occupation, salary, dept_id)
- VALUES(13, 'Jean-Ralphio', 'Saperstein', 'Exntertainment 720 CEO',1000000, NULL);

Events

- Events in MySQL are tasks that run according to a defined Schedule. events are executed based on schedule so it is also called a Scheduled event.
- Events can be created for one-time execution or for certain intervals. It consists of some MySQL statements which get executed on the happening of the event.
- In order to execute any event firstly we need to ensure that the event scheduler is set to ON. To check the status of the event scheduler we can execute the following Command,
  - SHOW VARIABLES
  - WHERE VARIABLE_NAME =
  - 'event_scheduler';
- Application of Event:
  - The event can be used for various optimization over collected data on a frequent basis like weekly, or monthly.
  - It can be used to track various details on available data.
  - This is used to maintain a large eCommerce-based database where there is a need of monitoring the stock units of available products.
  - It can be used to clean up log records of the visiting users on-site weekly or after a certain time,
- Syntax:
  - CREATE EVENT event_name
  - ON SCHEDULE schedule
  - DO
  - Event_body
- Example
  - DELIMITER $$
  - CREATE EVENT delete_retirees
  - ON SCHEDULE EVERY 30 SECOND

```
DO
BEGIN
        DELETE
        FROM employee_demographics
        WHERE age >= 60;
END $$
DELIMITER ;
```