SQL Cheat Sheet Vol.2

Created by Techavilly

What is Database?

A database is an organized collection of data, stored and retrieved digitally from a remote or local computer system. Databases can be vast and complex, and such databases are developed using fixed design and modeling approaches.

What is DBMS?

DBMS stands for Database Management System. DBMS is a system software responsible for the creation, retrieval, updation, and management of the database. It ensures that our data is consistent, organized, and is easily accessible by serving as an interface between the database and its end-users or application software.

What is RDBMS? How is it different from DBMS?

RDBMS stands for Relational Database Management System. The key difference here, compared to DBMS, is that RDBMS stores data in the form of a collection of tables, and relations can be defined between the common fields of these tables. Most modern database management systems like MySQL, Microsoft SQL Server, Oracle, IBM DB2, and Amazon Redshift are based on RDBMS.

What is SQL?

SQL stands for Structured Query Language. It is the standard language for relational database management systems. It is especially useful in handling organized data comprised of entities (variables) and relations between different entities of the data.

What is the difference between SQL and MySQL?

SQL is a standard language for retrieving and manipulating structured databases. On the contrary, MySQL is a relational database management system, like SQL Server, Oracle or IBM DB2, that is used to manage SQL databases.

What are Tables and Fields?

A table is an organized collection of data stored in the form of rows and columns. Columns can be categorized as vertical and rows as horizontal. The columns in a table are called fields while the rows can be referred to as records.

7. What are Constraints in SQL?

Constraints are used to specify the rules concerning data in the table. It can be applied for single or multiple fields in an SQL table during the creation of the table or after creating using the ALTER TABLE command. The constraints are:

- **NOT NULL** - Restricts NULL value from being inserted into a column.
- **CHECK** - Verifies that all values in a field satisfy a condition.
- **DEFAULT** - Automatically assigns a default value if no value has been specified for the field.
- **UNIQUE** - Ensures unique values to be inserted into the field.
- **INDEX** - Indexes a field providing faster retrieval of records.
- **PRIMARY KEY** - Uniquely identifies each record in a table.
- **FOREIGN KEY** - Ensures referential integrity for a record in another table.

What is a Primary Key?

The PRIMARY KEY constraint uniquely identifies each row in a table. It must contain UNIQUE values and has an implicit NOT NULL constraint.
A table in SQL is strictly restricted to have one and only one primary key, which is comprised of single or multiple fields (columns).

What is a UNIQUE constraint?

A UNIQUE constraint ensures that all values in a column are different. This provides uniqueness for the column(s) and helps identify each row uniquely. Unlike primary key, there can be multiple unique constraints defined per table. The code syntax for UNIQUE is quite similar to that of PRIMARY KEY and can be used interchangeably.

What is a Foreign Key?

A FOREIGN KEY comprises of single or collection of fields in a table that essentially refers to the PRIMARY KEY in another table. Foreign key constraint ensures referential integrity in the relation between two tables.
The table with the foreign key constraint is labeled as the child table, and the table containing the candidate key is labeled as the referenced or parent table.

What is a Join? List its different types.

The SQL Join clause is used to combine records (rows) from two or more tables in a SQL database based on a related column between the two.

- **(INNER) JOIN:** Retrieves records that have matching values in both tables involved in the join. This is the widely used join for queries.

Query Sample

```
SELECT *
FROM Table_A
JOIN Table_B;
SELECT *
FROM Table_A
INNER JOIN Table_B;
```

- **LEFT (OUTER) JOIN:** Retrieves all the records/rows from the left and the matched records/rows from the right table.

Query Sample

```
SELECT *
FROM Table_A A
LEFT JOIN Table_B B
ON A.col = B.col;
```

- **RIGHT (OUTER) JOIN:** Retrieves all the records/rows from the right and the matched records/rows from the left table.

Query Sample

```
SELECT *
FROM Table_A A
RIGHT JOIN Table_B B
ON A.col = B.col;
```

- **FULL (OUTER) JOIN:** Retrieves all the records where there is a match in either the left or right table.

Query Sample

- ## What is a Self-Join?

  A **self JOIN** is a case of regular join where a table is joined to itself based on some relation between its own column(s). Self-join uses the INNER JOIN or LEFT JOIN clause and a table alias is used to assign different names to the table within the query.

Query Sample

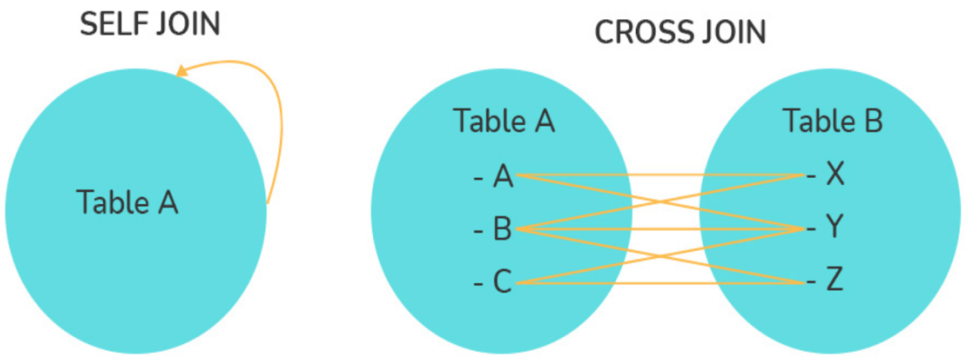
```
SELECT A.emp_id AS "Emp_ID",A.emp_name AS "Employee",
B.emp_id AS "Sup_ID",B.emp_name AS "Supervisor"
FROM employee A, employee B
WHERE A.emp_sup = B.emp_id;
```

- ## What is a Cross-Join?

Cross join can be defined as a cartesian product of the two tables included in the join. The table after join contains the same number of rows as in the cross-product of the number of rows in the two tables. If a WHERE clause is used in cross join then the query will work like an INNER JOIN.

Query Sample

```
SELECT stu.name, sub.subject
FROM students AS stu
CROSS JOIN subjects AS sub;
```



Write a SQL statement to perform SELF JOIN for 'Table_X' with alias 'Table_1' and 'Table_2', on columns 'Col_1' and 'Col_2' respectively.

SELECT * FROM Table_X AS Table_1, Table_X AS Table_2 WHERE Table_1.Col_1 = Table_2.Col_2;

- ## What is Data Integrity?

Data Integrity is the assurance of accuracy and consistency of data over its entire life-cycle and is a critical aspect of the design, implementation, and usage of any system which stores, processes, or retrieves data. It also defines integrity constraints to

enforce business rules on the data when it is entered into an application or a database.

- ## What is a Query?

A query is a request for data or information from a database table or combination of tables. A database query can be either a select query or an action query.

Query Sample

```
SELECT fname, lname
FROM myDb.students
WHERE student_id = 1;
```

- ## What is a Subquery? What are its types?

A subquery is a query within another query, also known as a **nested query** or **inner query**. It is used to restrict or enhance the data to be queried by the main query, thus restricting or enhancing the output of the main query respectively. For example, here we fetch the contact information for students who have enrolled for the maths subject:

Query Sample

```
SELECT name, email, mob, address
FROM myDb.contacts
WHERE roll_no IN (
SELECT roll_no
 FROM myDb.students
 WHERE subject = 'Maths');
```

There are two types of subquery- **Correlated** and **Non-Correlated**.

- A **correlated** subquery cannot be considered as an independent query, but it can refer to the column in a table listed in the FROM of the main query.
- A **non-correlated** subquery can be considered as an independent query and the output of the subquery is substituted in the main query.

Write a SQL query to update the field "status" in table "applications" from 0 to 1.

UPDATE application SET status = 1 WHERE status = 0;

Write a SQL query to select the field "app_id" in table "applications" where "app_id" less than 1000.

SELECT app_id FROM applications WHERE app_id < 1000

- ## What are some common clauses used with SELECT query in SQL?

  Some common SQL clauses used in conjuction with a SELECT query are as follows:

  - **WHERE** clause in SQL is used to filter records that are necessary, based on specific conditions.
  - **ORDER BY** clause in SQL is used to sort the records based on some field(s) in ascending (**ASC**) or descending order (**DESC)**.

    Query Sample

    ```
    SELECT *
    FROM myDB.students
    WHERE graduation_year = 2019
    ORDER BY studentID DESC;
    ```

- **GROUP BY** clause in SQL is used to group records with identical data and can be used in conjunction with some aggregation functions to produce summarized results from the database.
- **HAVING** clause in SQL is used to filter records in combination with the GROUP BY clause. It is different from WHERE, since the WHERE clause cannot filter aggregated records.

  Query Sample

  ```
  SELECT COUNT(studentId), country
  FROM myDB.students
  WHERE country != "INDIA"
  GROUP BY country
  HAVING COUNT(studentID) > 5;
  ```

- ## What are UNION, MINUS and INTERSECT commands?

  The **UNION** operator combines and returns the result-set retrieved by two or more SELECT statements.
  The **MINUS** operator in SQL is used to remove duplicates from the result-set

obtained by the second SELECT query from the result-set obtained by the first SELECT query and then return the filtered results from the first.
The **INTERSECT** clause in SQL combines the result-set fetched by the two SELECT statements where records from one match the other and then returns this intersection of result-sets.

Certain conditions need to be met before executing either of the above statements in SQL -

- Each SELECT statement within the clause must have the same number of columns
- The columns must also have similar data types
- The columns in each SELECT statement should necessarily have the same order

Query Sample

```
SELECT name FROM Students
UNION
SELECT name FROM Contacts;
SELECT name FROM Students
UNION ALL
SELECT name FROM Contacts;
```

Query Sample

```
SELECT name FROM Students
MINUS
SELECT name FROM Contacts;
```

Query Sample

```
SELECT name FROM Students
INTERSECT
SELECT name FROM Contacts;
```

Write a SQL query to fetch "names" that are present in either table "accounts" or in table "registry".

Answer: SELECT names FROM accounts UNION SELECT names from registry;

Write a SQL query to fetch "names" that are present in "accounts" but not in table "registry".

Answer: SELECT names FROM accounts MINUS SELECT names FROM registry;

What is Cursor? How to use a Cursor?

A database cursor is a control structure that allows for the traversal of records in a database. Cursors, in addition, facilitates processing after traversal, such as retrieval, addition, and deletion of database records. They can be viewed as a pointer to one row in a set of rows.

Working with SQL Cursor:

1. **DECLARE** a cursor after any variable declaration. The cursor declaration must always be associated with a SELECT Statement.
2. Open cursor to initialize the result set. The **OPEN** statement must be called before fetching rows from the result set.
3. **FETCH** statement to retrieve and move to the next row in the result set.
4. Call the **CLOSE** statement to deactivate the cursor.
5. Finally use the **DEALLOCATE** statement to delete the cursor definition and release the associated resources.

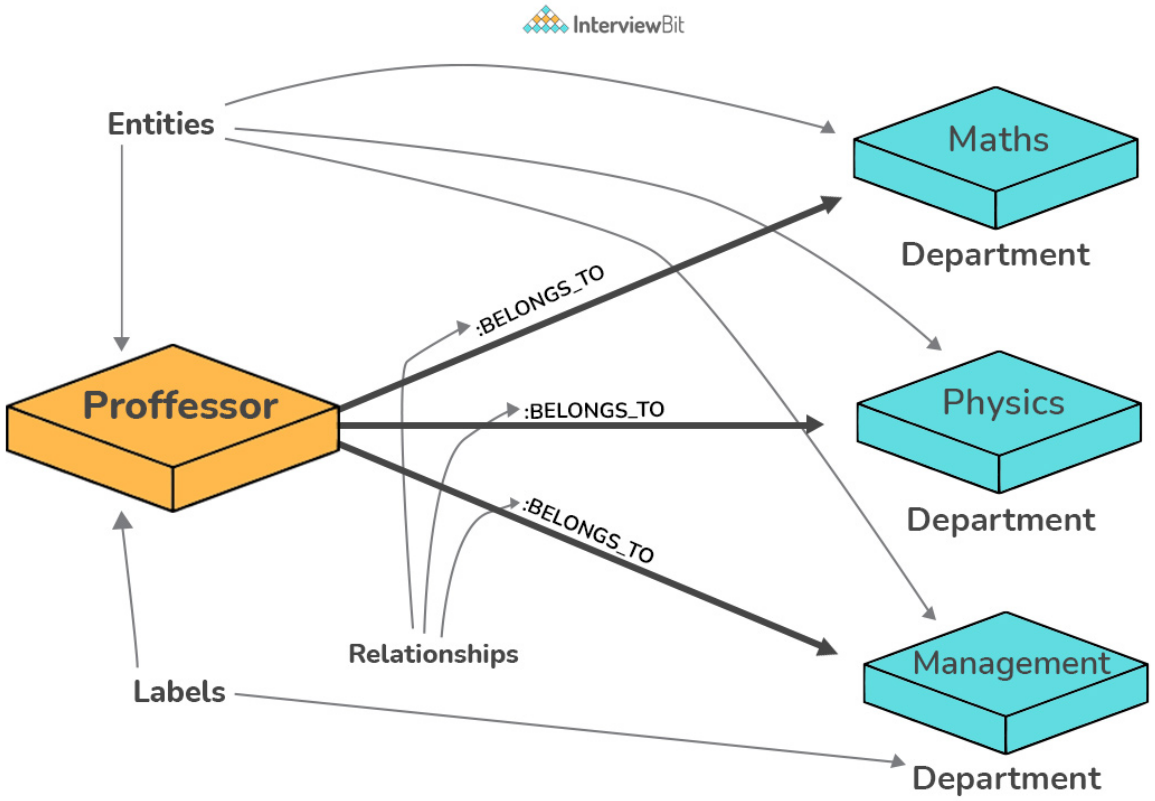
Query Sample

```
DECLARE @name VARCHAR(50)
DECLARE db_cursor CURSOR FOR
SELECT name
FROM myDB.students
WHERE parent_name IN ('Sara', 'Ansh')
OPEN db_cursor
FETCH next
FROM db_cursor
INTO @name
CLOSE db_cursor
DEALLOCATE db_cursor
```

What are Entities and Relationships?

Entity: An entity can be a real-world object, either tangible or intangible, that can be easily identifiable. For example, in a college database, students, professors, workers, departments, and projects can be referred to as entities. Each entity has some associated properties that provide it an identity.

Relationships: Relations or links between entities that have something to do with each other. For example - The employee's table in a company's database can be associated with the salary table in the same database.



List the different types of relationships in SQL.

- **One-to-One** - This can be defined as the relationship between two tables where each record in one table is associated with the maximum of one record in the other table.
- **One-to-Many & Many-to-One** - This is the most commonly used relationship where a record in a table is associated with multiple records in the other table.
- **Many-to-Many** - This is used in cases when multiple instances on both sides are needed for defining a relationship.
- **Self-Referencing Relationships** - This is used when a table needs to define a relationship with itself.

What is an Alias in SQL?

An alias is a feature of SQL that is supported by most, if not all, RDBMSs. It is a temporary name assigned to the table or table column for the purpose of a particular SQL query. In addition, aliasing can be employed as an obfuscation technique to secure the real names of database fields. A table alias is also called a correlation name.

An alias is represented explicitly by the AS keyword but in some cases, the same can be performed without it as well. Nevertheless, using the AS keyword is always a good practice.

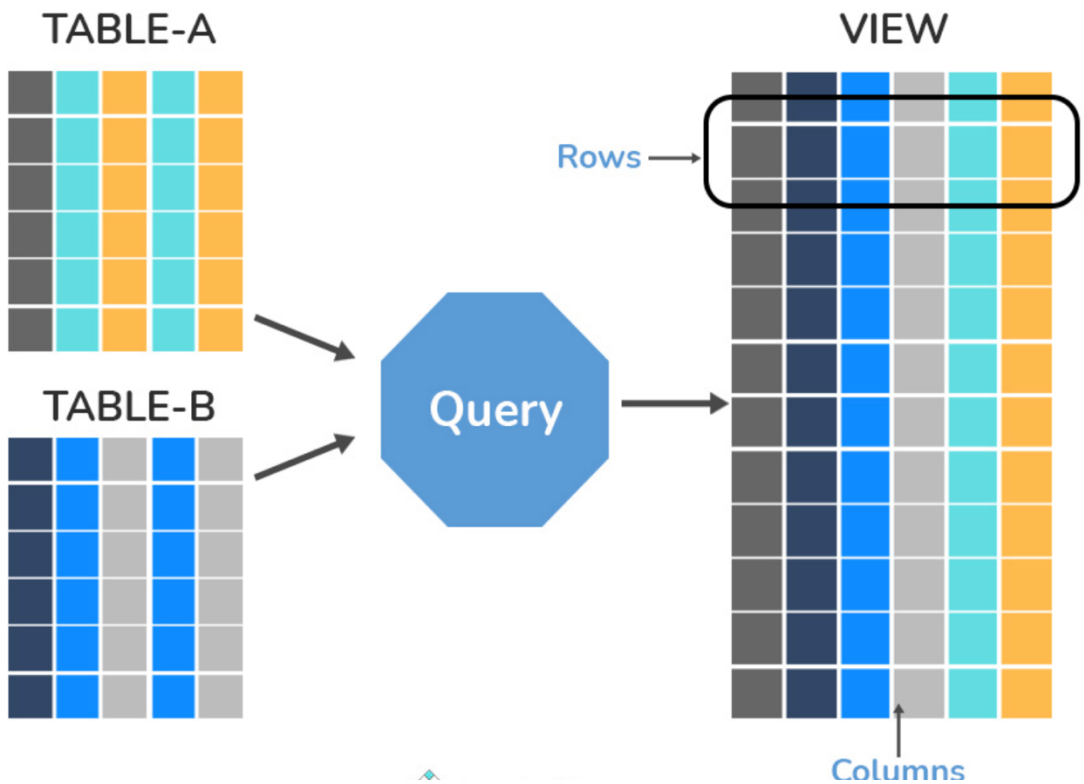
Write an SQL statement to select all from table "Limited" with alias "Ltd".

Answer: SELECT * FROM Limited AS Ltd;

```
SELECT A.emp_name AS "Employee"
B.emp_name AS "Supervisor"
FROM employee A, employee B
WHERE A.emp_sup = B.emp_id;
```

What is a View?

A view in SQL is a virtual table based on the result-set of an SQL statement. A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.



What is Denormalization?

Denormalization is the inverse process of normalization, where the normalized schema is converted into a schema that has redundant information. The performance is improved by using redundancy and keeping the redundant data consistent. The reason for performing denormalization is the overheads produced in the query processor by an over-normalized structure.

29. What are the various forms of Normalization?

Normal Forms are used to eliminate or reduce redundancy in database tables. The different forms are as follows:

- **First Normal Form:**
  A relation is in first normal form if every attribute in that relation is a **single-valued attribute**. If a relation contains a composite or multi-valued attribute, it violates the first normal form. Let's consider the following **students** table. Each student in the table, has a name, his/her address, and the books they issued from the public library -

Students Table

Student	Address	Books Issued	Salutation
Sara	Amanora Park Town 94	Until the Day I Die (Emily Carpenter), Inception (Christopher Nolan)	Ms.
Ansh	62nd Sector A-10	The Alchemist (Paulo Coelho), Inferno (Dan Brown)	Mr.
Sara	24th Street Park Avenue	Beautiful Bad (Annie Ward), Woman 99 (Greer Macallister)	Mrs.
Ansh	Windsor Street 777	Dracula (Bram Stoker)	Mr.

As we can observe, the Books Issued field has more than one value per record, and to convert it into 1NF, this has to be resolved into separate individual records for each book issued. Check the following table in 1NF form -

Students Table (1st Normal Form)

Student	Address	Books Issued	Salutation
Sara	Amanora Park Town 94	Until the Day I Die (Emily Carpenter)	Ms.

Student	Address	Books Issued	Salutation
Sara	Amanora Park Town 94	Inception (Christopher Nolan)	Ms.
Ansh	62nd Sector A-10	The Alchemist (Paulo Coelho)	Mr.
Ansh	62nd Sector A-10	Inferno (Dan Brown)	Mr.
Sara	24th Street Park Avenue	Beautiful Bad (Annie Ward)	Mrs.
Sara	24th Street Park Avenue	Woman 99 (Greer Macallister)	Mrs.
Ansh	Windsor Street 777	Dracula (Bram Stoker)	Mr.

- **Second Normal Form:**

A relation is in second normal form if it satisfies the conditions for the first normal form and does not contain any partial dependency. A relation in 2NF has **no partial dependency**, i.e., it has no non-prime attribute that depends on any proper subset of any candidate key of the table. Often, specifying a single column Primary Key is the solution to the problem. Examples -

Example 1 - Consider the above example. As we can observe, the Students Table in the 1NF form has a candidate key in the form of [Student, Address] that can uniquely identify all records in the table. The field Books Issued (non-prime attribute) depends partially on the Student field. Hence, the table is not in 2NF. To convert it into the 2nd Normal Form, we will partition the tables into two while specifying a new ***Primary Key*** attribute to identify the individual records in the Students table. The ***Foreign Key*** constraint will be set on the other table to ensure referential integrity.

Students Table (2nd Normal Form)

Student_ID	Student	Address	Salutation
1	Sara	Amanora Park Town 94	Ms.
2	Ansh	62nd Sector A-10	Mr.
3	Sara	24th Street Park Avenue	Mrs.
4	Ansh	Windsor Street 777	Mr.

Books Table (2nd Normal Form)

Student_ID	Book Issued
1	Until the Day I Die (Emily Carpenter)
1	Inception (Christopher Nolan)
2	The Alchemist (Paulo Coelho)
2	Inferno (Dan Brown)
3	Beautiful Bad (Annie Ward)

Student_ID	Book Issued
3	Woman 99 (Greer Macallister)
4	Dracula (Bram Stoker)

Example 2 - Consider the following dependencies in relation to R(W,X,Y,Z)

```
WX -> Y     [W and X together determine Y]
 XY -> Z     [X and Y together determine Z]
```

Here, WX is the only candidate key and there is no partial dependency, i.e., any proper subset of WX doesn't determine any non-prime attribute in the relation.

- **Third Normal Form**

A relation is said to be in the third normal form, if it satisfies the conditions for the second normal form and there is **no transitive dependency** between the non-prime attributes, i.e., all non-prime attributes are determined only by the candidate keys of the relation and not by any other non-prime attribute.

Example 1 - Consider the Students Table in the above example. As we can observe, the Students Table in the 2NF form has a single candidate key Student_ID (primary key) that can uniquely identify all records in the table. The field Salutation (non-prime attribute), however, depends on the Student Field rather than the candidate key. Hence, the table is not in 3NF. To convert it into the 3rd Normal Form, we will once again partition the tables into two while specifying a new ***Foreign Key*** constraint to identify the salutations for individual records in the Students table. The ***Primary Key*** constraint for the same will be set on the Salutations table to identify each record uniquely.

Students Table (3rd Normal Form)

Student_ID	Student	Address	Salutation_ID
1	Sara	Amanora Park Town 94	1
2	Ansh	62nd Sector A-10	2
3	Sara	24th Street Park Avenue	3
4	Ansh	Windsor Street 777	1

Books Table (3rd Normal Form)

Student_ID	Book Issued
1	Until the Day I Die (Emily Carpenter)
1	Inception (Christopher Nolan)
2	The Alchemist (Paulo Coelho)
2	Inferno (Dan Brown)
3	Beautiful Bad (Annie Ward)
3	Woman 99 (Greer Macallister)
4	Dracula (Bram Stoker)

Salutations Table (3rd Normal Form)

Salutation_ID	Salutation
1	Ms.
2	Mr.
3	Mrs.

What are the TRUNCATE, DELETE and DROP statements?

DELETE statement is used to delete rows from a table.

```
DELETE FROM Candidates
WHERE CandidateId > 1000;
```

TRUNCATE command is used to delete all the rows from the table and free the space containing the table.

```
TRUNCATE TABLE Candidates;
```

DROP command is used to remove an object from the database. If you drop a table, all the rows in the table are deleted and the table structure is removed from the database.

```
DROP TABLE Candidates;
```

What is the difference between DROP and TRUNCATE statements?

If a table is dropped, all things associated with the tables are dropped as well. This includes - the relationships defined on the table with other tables, the integrity checks and constraints, access privileges and other grants that the table has. To create and use the table again in its original form, all these relations, checks, constraints, privileges and relationships need to be redefined. However, if a table is truncated, none of the above problems exist and the table retains its original structure.

What is the difference between DELETE and TRUNCATE statements?

The **TRUNCATE** command is used to delete all the rows from the table and free the space containing the table.
The **DELETE** command deletes only the rows from the table based on the condition given in the where clause or deletes all the rows from the table if no condition is specified. But it does not free the space containing the table.

What are Aggregate and Scalar functions?

An aggregate function performs operations on a collection of values to return a single scalar value. Aggregate functions are often used with the GROUP BY and HAVING clauses of the SELECT statement. Following are the widely used SQL aggregate functions:

- **AVG ()** - Calculates the mean of a collection of values.
- **COUNT ()** - Counts the total number of records in a specific table or view.
- **MIN ()** - Calculates the minimum of a collection of values.
- **MAX ()** - Calculates the maximum of a collection of values.
- **SUM ()** - Calculates the sum of a collection of values.
- **FIRST ()** - Fetches the first element in a collection of values.
- **LAST ()** - Fetches the last element in a collection of values.

Note: All aggregate functions described above ignore NULL values except for the COUNT function.

A scalar function returns a single value based on the input value. Following are the widely used SQL scalar functions:

- **LEN ()** - Calculates the total length of the given field (column).
- **UCASE ()** - Converts a collection of string values to uppercase characters.
- **LCASE ()** - Converts a collection of string values to lowercase characters.

- **MID ()** - Extracts substrings from a collection of string values in a table.
- **CONCAT ()** - Concatenates two or more strings.
- **RAND ()** - Generates a random collection of numbers of a given length.
- **ROUND ()** - Calculates the round-off integer value for a numeric field (or decimal point values).
- **NOW ()** - Returns the current date & time.
- **FORMAT ()** - Sets the format to display a collection of values.

What is Collation? What are the different types of Collation Sensitivity?

Collation refers to a set of rules that determine how data is sorted and compared. Rules defining the correct character sequence are used to sort the character data. It incorporates options for specifying case sensitivity, accent marks, kana character types, and character width. Below are the different types of collation sensitivity:

- **Case sensitivity: A** and **a** are treated differently.
- **Accent sensitivity: a** and **á** are treated differently.
- **Kana sensitivity:** Japanese kana characters Hiragana and Katakana are treated differently.
- **Width sensitivity:** Same character represented in single-byte (half-width) and double-byte (full-width) are treated differently.

What is Pattern Matching in SQL?

SQL pattern matching provides for pattern search in data if you have no clue as to what that word should be. This kind of SQL query uses wildcards to match a string pattern, rather than writing the exact word. The LIKE operator is used in conjunction with **SQL Wildcards** to fetch the required information.

- **Using the % wildcard to perform a simple search**

The % wildcard matches zero or more characters of any type and can be used to define wildcards both before and after the pattern. Search a student in your database with first name beginning with the letter K:

```
SELECT *
FROM students
WHERE first_name LIKE 'K%'
```

Omitting the patterns using the NOT keyword

Use the NOT keyword to select records that don't match the pattern. This query returns all students whose first name does not begin with K.

```
SELECT *
FROM students
WHERE first_name NOT LIKE 'K%'
```

Using the _ wildcard to match pattern at a specific position

The _ wildcard matches exactly one character of any type. It can be used in conjunction with % wildcard. This query fetches all students with letter K at the third position in their first name.

```
SELECT *
FROM students
WHERE first_name LIKE '__K
```

Matching patterns for a specific length

The _ wildcard plays an important role as a limitation when it matches exactly one character. It limits the length and position of the matched results. For example -

```
SELECT *
FROM students
WHERE first_name LIKE '___%'
```

```
SELECT *
FROM students
WHERE first_name LIKE '____'
```

What is PostgreSQL?

PostgreSQL was first called Postgres and was developed by a team led by Computer Science Professor Michael Stonebraker in 1986. It was developed to help developers build enterprise-level applications by upholding data integrity by making systems fault-tolerant. PostgreSQL is therefore an enterprise-level, flexible, robust, open-source, and object-relational DBMS that supports flexible workloads along with handling concurrent users. It has been consistently supported by the global developer community. Due to its fault-tolerant nature, PostgreSQL has gained widespread popularity among developers.

How do you define Indexes in PostgreSQL?

Indexes are the inbuilt functions in PostgreSQL which are used by the queries to perform search more efficiently on a table in the database. Consider that you have a table with thousands of records and you have the below query that only a few records can satisfy the condition, then it will take a lot of time to search and return those rows that abide by this condition as the engine has to perform the search operation on every single to check this condition. This is undoubtedly inefficient for a system dealing with huge data. Now if this system had an index on the column where we are applying search, it can use an efficient method for identifying matching rows by walking through only a few levels. This is called indexing.

How will you change the datatype of a column?

This can be done by using the ALTER TABLE statement as shown below:

Syntax:

```
ALTER TABLE tname
ALTER COLUMN col_name [SET DATA] TYPE new_data_type;
```

1. From which station names has bike_id 1710 started?

SELECT

DISTINCT start_station_name

FROM

`bigquery-public-data.london_bicycles.cycle_hire`

WHERE

bike_id = 1710;

2. How many bike_ids have ended at "Moor Street, Soho"?

SELECT

```
COUNT (DISTINCT bike_id) AS num_of_bikes
FROM
`bigquery-public-data.london_bicycles.cycle_hire`
WHERE
end_station_name = 'Moor Street, Soho';
```

3. What is the station_id for "Canton Street, Poplar"?

```
SELECT
DISTINCT start_station_id --can also use end_station_id
FROM
`bigquery-public-data.london_bicycles.cycle_hire`
WHERE
start_station_name = 'Canton Street, Poplar' --can also use end_station_name
```

4. What is the name of the station whose ID is 111?

```
SELECT
DISTINCT start_station_name --can also use end_station_name

FROM
`bigquery-public-data.london_bicycles.cycle_hire`
WHERE
start_station_id = 111 --can also use end_station_id
```

5. How many distinct bike_ids had trip durations greater than 2400 seconds (or 40 minutes)?

```
SELECT
COUNT(DISTINCT bike_id) AS num_of_bike_trips
FROM
`bigquery-public-data.london_bicycles.cycle_hire`
WHERE
duration > 2400 --schema indicates that trip duration is recorded in seconds


SELECT
    CASE
        WHEN genre = 'horror' THEN 'Will not watch'
        WHEN genre = 'documentary' THEN 'Will watch'
        ELSE 'watch with others'
        END AS watch_category, COUNT(movie_title) AS number of movies
FROM
    Movie data
GROUP BY
    1
```