SQL Terms

1.**Query**
A query is a request to retrieve or manipulate data in a database. Think of it as asking the database a question, like "Show me all the players_name in Spain Football League.

2.**Database**
A database is a structured collection of data. It's like a digital filing cabinet where all your data is stored and organized for easy access.

3.**Table**
A table is a set of rows and columns in a database, similar to an Excel spreadsheet. Each row is a record, and each column is a field of data.

4. **Primary Key**
A primary key is a unique identifier for each record in a table. It ensures that every row can be uniquely identified, like a Social Security Number for your data.

5. **Foreign Key**
A foreign key is a field in a table that links to the primary key in another table. It's how different tables are connected, enabling you to relate data across tables.

6. **Schema**
The structure of your database, including how tables, fields, and relationships are organized. It's the blueprint that defines how data is stored.

7. **Index**
This is a database structure that improves the speed of data retrieval operations. Think of it like an index in a book that helps you find information quickly.

8. **SQL Injection**
SQL Injection is a type of attack where malicious SQL code is used to access or manipulate a database. Understanding this term is crucial for database security.

9. **Join**
This is a command used to combine data from two or more tables based on a related column between them. Joins are essential for querying across multiple tables.

10. **Aggregate Function**
A function that performs a calculation on a set of values and returns a single value, like SUM, COUNT, AVG, MAX, or MIN. It's useful for summarizing data.

SQL Joins

1.**Inner Joins**
When to use: When you need to retrieve records that have matching values in both tables.
Example: Finding customers who have placed orders.

```
SELECT
c.customer_id,
c.customer_name,
o.order_id,
o.order_date
FROM
customers c
INNER JOIN
orders o ON c.customer_id = o.customer_id;
```

2.**Left Join**
When to use: When you need all records from the left table and the matched records from the right table. Unmatched records from the right table will be NULL.
Example: Find all customers, including those who haven't placed any orders.

```
SELECT
c.customer_id,
c.customer_name,
o.order_id,
o. order_date
FROM
customers c
LEFT JOIN
orders o ON c.customer_id = o.customer_id;
```

3.**Right Join**
When to use: When you need all records from the right table and the matched records from the left table. Unmatched records from the left table will be NULL.
Example: Listing all orders and the customers who placed them, if any.

```
SELECT
o.order_id,
o.order_date,
c.customer_id,
c.customer_name
FROM
customers c
RIGHT JOIN
orders o ON c.customer_id = o.customer_id;
```

4.**FULL Join**
When to use: When you need all the records when there is a match on either the left or right table. Records with no match will have NULLs
Example: Finding all customers and all orders, including those without matches.

```
SELECT
c.customer_id,
c.customer_name,
o.order_id,
o. order_date
FROM
customers c
FULL OUTER JOIN
orders o ON c.customer_id = o.customer_id;
```

5.**Cross Join**
When to use: When you need to return the Cartesian product of the two tables. Be cautious as this can result in a large number of rows.
Example: Pairing each customer with every product.

```
SELECT
c.customer_id,
c.customer_name,
o.order_id,
o.order_date
FROM
customers c
CROSS JOIN
orders o;
```

6.**Self Join**
When to use: When you need to join a table with itself to compare rows within the same table.
Example: Finding pairs of employees who have the same manager.

```
SELECT
e1.employee_name AS employee_1,
e2.employee_name AS employee_2,
e1.manager_id
FROM
employees e1
JOIN
employees e2 ON e1.manager_id = e2.manager_id
WHERE
e1.employee_id <> e2.employee_id;
```