SUB QUERIES

- An SQL sub query is a query inside a query.
- Sub-queries are also known as nested queries or inner queries.
- They are used in SQL to retrieve data based on the results of another query.

Example:

- An **employees table** in an **employees_data database**.
  To get the data of those earning more than the average wage, you run the following query and subquery:

```
SELECT * FROM employees
WHERE wage > (SELECT AVG(wage) FROM employees)
```

- the main query selected everything from the employees table
- the sub-query (SELECT AVG(wage) FROM employees) got the average wage of the employees
- the WHERE clause (WHERE wage >) is responsible for getting every employee with a salary less than the average wage.
- Another example of sub-query use;

```
SELECT name, country, wage FROM employees
WHERE country IN (SELECT country
        FROM employees
        WHERE country = 'USA') ;
```

- sub-queries are not limited to the SELECT statement only.
- You can use sub-queries in all the CRUD operations of SQL – INSERT, SELECT, UPDATE, and DELETE.

SQL CONSTRAINTS

- Constraints are the rules enforced on the data columns of a table.
- These are used to limit the type of data that can go into a table.
- This ensures the accuracy and reliability of the data in the database.
- Constraints could be either on a **column level** or a table level. The **column level** constraints are **applied only to one column**, whereas the **table level** constraints are **applied to the whole table**.
- Constraints help in maintaining the accuracy, integrity, and reliability of a database.
- Constraints can be imposed at the time of the creation of the table or after its creation as well.

- The following constraints are commonly used in SQL:

1) **NOT NULL** - Ensures that a column cannot have a NULL value
2) **UNIQUE** - Ensures that all values in a column are different
3) **PRIMARY KEY** - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
4) **FOREIGN KEY** - Prevents actions that would destroy links between tables
5) **CHECK** - Ensures that the values in a column satisfies a specific condition
6) **DEFAULT** - Sets a default value for a column if no value is specified
7) **CREATE INDEX** - Used to create and retrieve data from the database very quickly

Syntax for creating a table with constraints

```
CREATE TABLE sample_table
(
column1 data_type(size) constraint_name,
column2 data_type(size) constraint_name,
column3 data_type(size) constraint_name,
....
);
```

sample_table: Name of the table to be created.

data_type: Type of data that can be stored in the field.

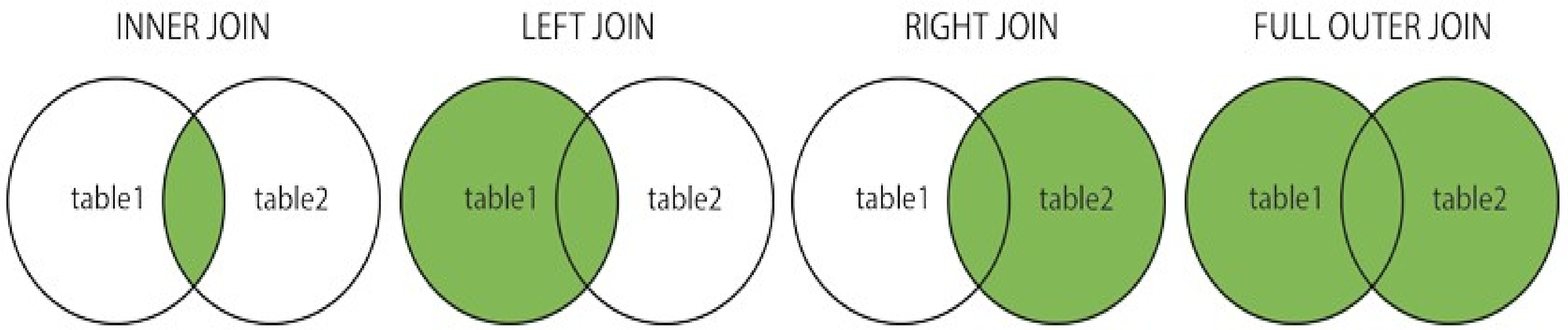
constraint_name: Name of the constraint. for example- NOT NULL, UNIQUE, PRIMARY KEY etc.

```
CREATE TABLE Student
(
ID int(6) NOT NULL,
NAME varchar(10) NOT NULL,
ADDRESS varchar(20)
);
```

- the above query creates a table Student with the fields ID and NAME as NOT NULL.
- This means, these two fields cannot be empty/null any time we insert a new row into the table.

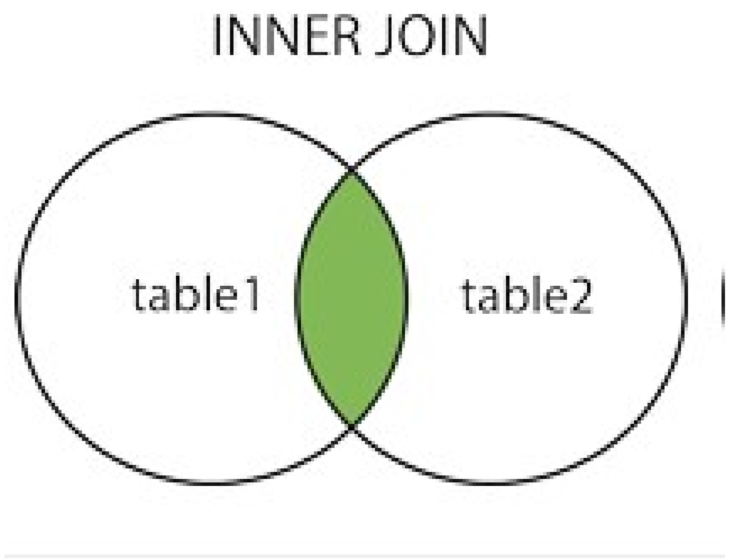
JOINS

- SQL Join statement is used **to combine data or rows from two or more tables** based on a **common column** between them.
- The different types of the JOINs in SQL:
- **(INNER) JOIN**: Returns records that have matching values in both tables
- **LEFT (OUTER) JOIN**: Returns all records from the left table, and the matched records from the right table
- **RIGHT (OUTER) JOIN**: Returns all records from the right table, and the matched records from the left table
- **FULL (OUTER) JOIN**: Returns all records when there is a match in either left or right table


INNER JOIN
table1
table2
LEFT JOIN
table1
table2
RIGHT JOIN
table1
table2
FULL OUTER JOIN
table1
table2

INNER JOIN

- An INNER JOIN returns **only the rows that have matching values in both tables**.
  It excludes rows from either table that do not have corresponding matches in the other table.



- Syntax:

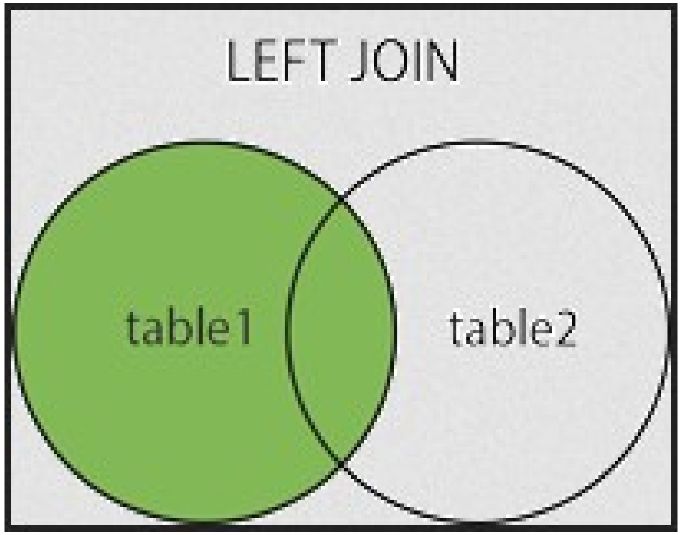
```
SELECT table1.column_name ,  table2.column_name
FROM table1
INNER JOIN table2 ON table1.column_name = table2.column_name;
```

- Example:
  Let's consider two tables: "customers" and "orders." We want to retrieve all orders with their corresponding customer information.

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

LEFT JOIN(LEFT OUTER JOIN)

- A LEFT JOIN returns all the **rows from the left table** and the **matched rows from the right table**.
- If there is no match in the right table, NULL values are returned for the right table's columns.



- Syntax:

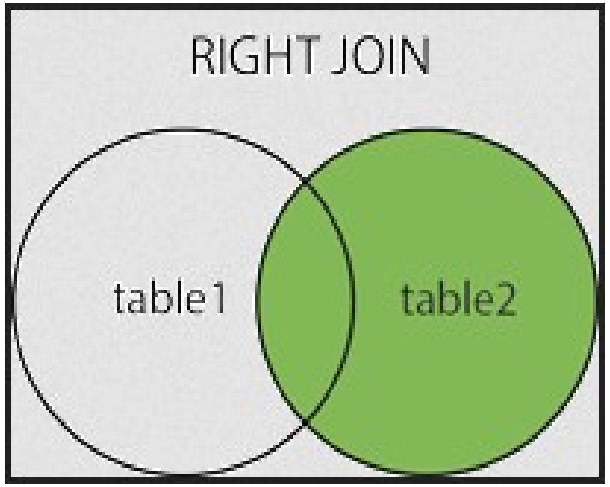
```
SELECT table1.column_name, table2.column_name
FROM table1
LEFT JOIN table2
ON table1.column_name = table2.column_name;
```

- Example:
  Let's consider two tables: "departments" and "employees." We want to retrieve all departments and the employees assigned to each department.

```
SELECT departments.department_name, employees.employee_name
FROM departments
LEFT JOIN employees
ON departments.department_id = employees.department_id;
```

RIGHT JOIN (RIGHT OUTER JOIN)

- A RIGHT JOIN is similar to a LEFT JOIN but **returns all the rows from the right table and the matched rows from the left table.**
  If there is no match in the left table, NULL values are returned for the left table's columns.



- Syntax:

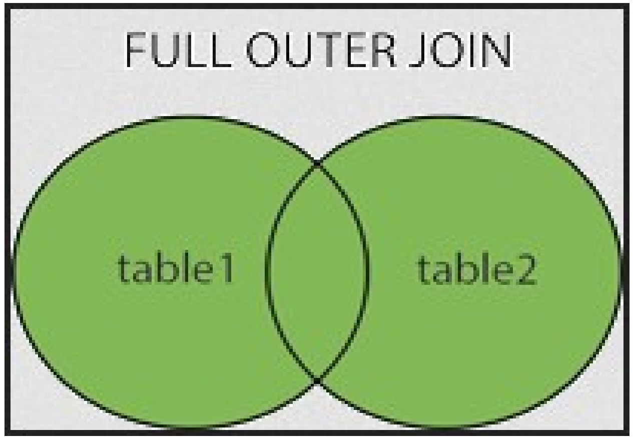
```
SELECT table1.column_name, table2.column_name
FROM table1
RIGHT JOIN table2
ON table1.column_name = table2.column_name;
```

- Example:
  Let's consider two tables: "orders" and "order_items." We want to retrieve all order items and their corresponding order information.

```
SELECT order_items.item_id, orders.order_date
FROM order_items
RIGHT JOIN orders
ON order_items.order_id = orders.order_id;
```

FULL JOIN (FULL OUTER JOIN):

- A FULL JOIN **returns all rows when there is a match in either the left or right table**.
  If there is no match, NULL values are returned for the columns from the table that doesn't have a match.



- Syntax:

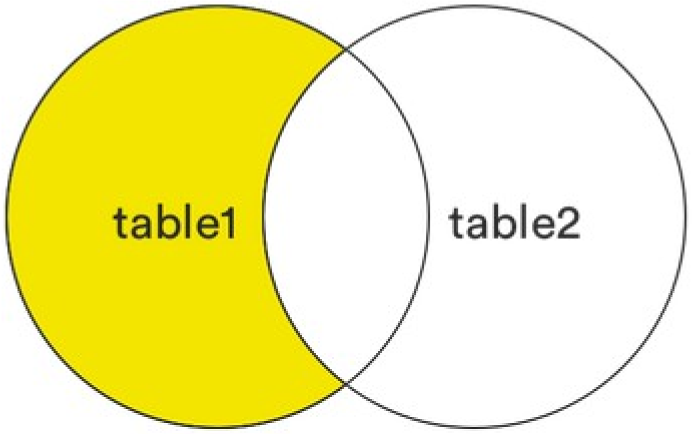
```
SELECT table1.column_name, table2.column_name
FROM table1
FULL JOIN table2
ON table1.column_name = table2.column_name;
```

- Example:
  Let's consider two tables: "products" and "sales." We want to retrieve all product sales data, including products with no sales and sales with no associated product.

```
SELECT products.product_name, sales.sale_amount
FROM products
FULL JOIN sales
ON products.product_id = sales.product_id;
```

LEFT [OUTER] JOIN without the intersection

- This clause returns all records from the left table (table1) that matched the records on the right table(table2), but exclude those records exist in both tables.



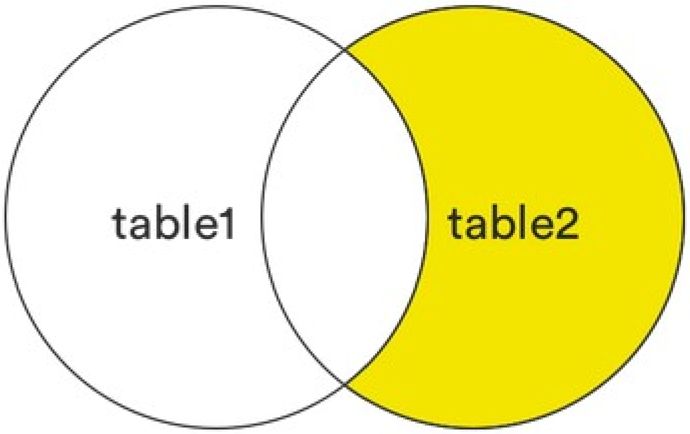
LEFT JOIN with no intersection

SQL syntax:

```
SELECT *
FROM table1
LEFT JOIN table2
ON table1.col1 = table2.col2
WHERE table2.col2 IS NULL;
```

RIGHT [OUTER] JOIN without the intersection

- This clause returns all records from the right table (table2) that matched those in the left table(table 1), but exclude those records that exist in both tables.



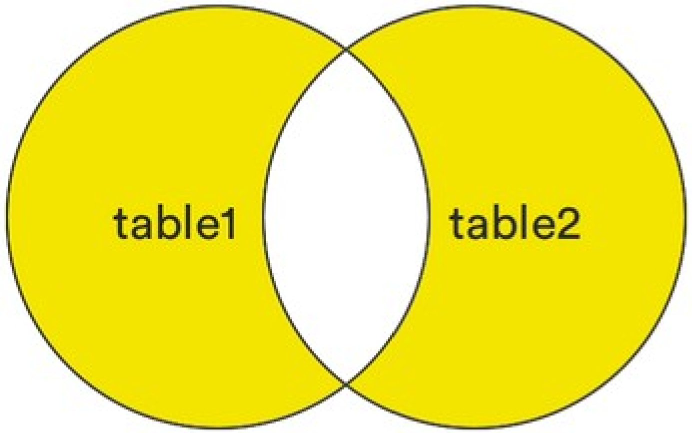
RIGHT JOIN with no intersection

SQL syntax:

```
SELECT *
FROM table1
RIGHT JOIN table2
ON table1.col1 = table2.col2
WHERE table1.col1 IS NULL;
```

FULL [OUTER] JOIN without the intersection.

- This clause **returns all rows when there is a match in either the left or right table but excluding those are in common between two tables, or those records exist in both tables.**



OUTER JOIN with no intersection

SQL syntax:

```
SELECT *
FROM table1
FULL JOIN table2
ON table1.col1 = table2.col2
WHERE table1.col1 IS NULL
OR table2.col2 IS NULL;
```