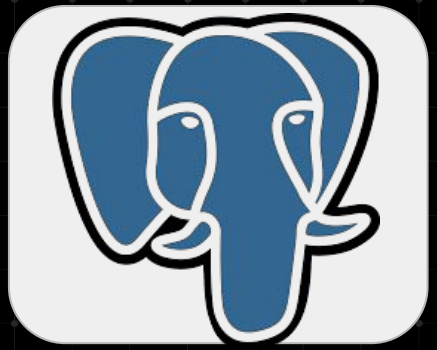


PostgreSQL Database Week

The Tech Stuff

Session Dates

01

Day 1
Intro to SQL

02

Day 2
Filtering Data

03

Day 3
Aggregating Data

04

Day 4
Joins &
Subqueries

05

Day 5
Data Manipulations

06

Day 6
Real-World
Scenarios

01

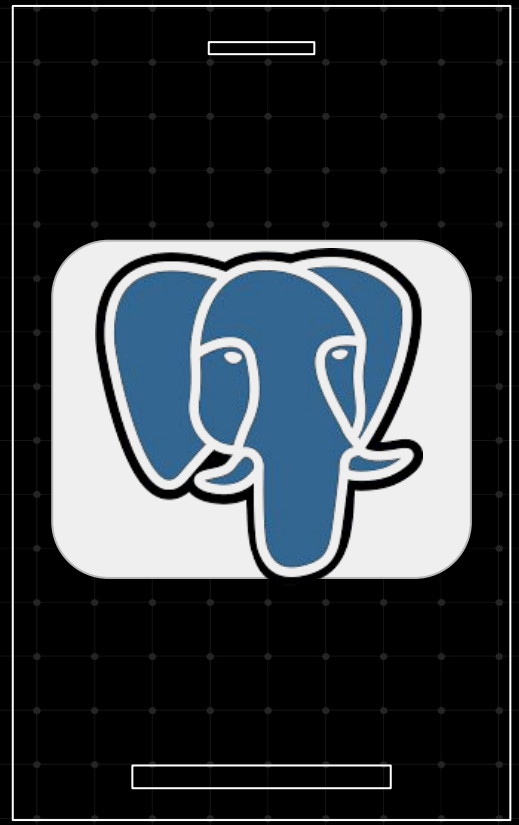
Introduction to PostgreSQL & SQL

Day 1

1. Overview of PostgreSQL and SQL.
2. Installation and setup of PostgreSQL.
3. Introduction to basic SQL commands: SELECT, INSERT, UPDATE, DELETE.
4. Running queries using PostgreSQL CLI (psql).

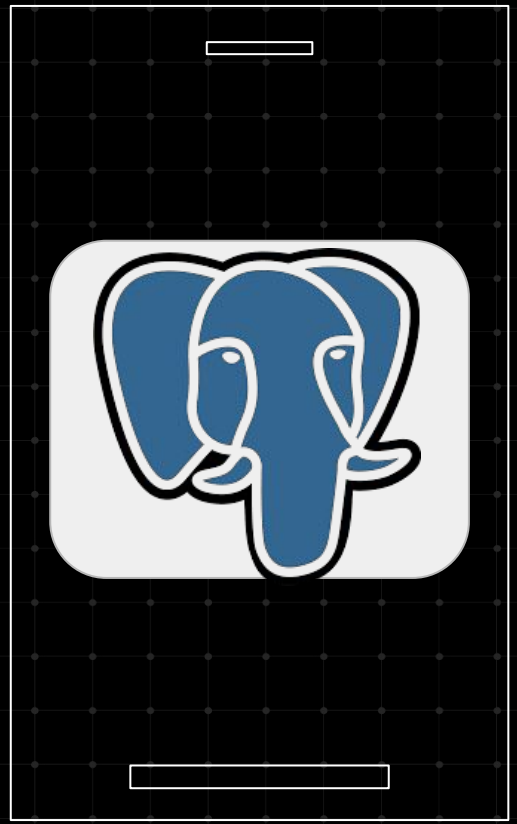
Introduction to SQL

The standard language used for managing and manipulating relational databases. It provides a way to communicate with and manage the data stored within a relational database system.



Introduction to SQL

SQL allows users to perform various operations such as querying data, updating records, inserting new data, and deleting records. It is essential for interacting with databases and is widely used in various applications, from small-scale projects to large enterprise systems.



SQL Keys

Primary

A unique identifier for a record in a database table. It ensures that each row in the table is uniquely identifiable, preventing duplicate records.

Foreign

A field (or a group of fields) in a database table that creates a link between two tables. It establishes a relationship between the records in the two tables, enforcing referential integrity.

Primary Key Rules

Uniqueness

The values in the primary key column(s) must be unique across all rows in the table. This ensures that no two rows share the same primary key value.

Not Null

The primary key column(s) cannot contain NULL values. Every record in the table must have a valid and unique primary key value.

Immutable

The value of a primary key should not change frequently. It should remain stable to maintain the integrity of the relationships between tables.

PostgreSQL

PostgreSQL is a powerful, open-source relational database management system (RDBMS) known for its advanced features and extensibility. It is designed to handle a wide range of workloads, from single-machine applications to large-scale web applications with many concurrent users. PostgreSQL is renowned for its robustness, reliability, and support for complex queries and data types.

Data Types in SQL

1) INT , INTEGER

2) VARCHAR

3) BOOLEAN

4) DATE

5) NUMERIC

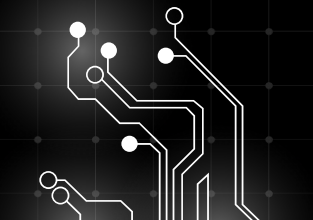
6) TEXT

Creating Tables

In PostgreSQL, creating a table involves defining the structure of the table, including its columns, data types, and constraints. A table is a fundamental structure in a relational database, where data is stored in rows and columns.

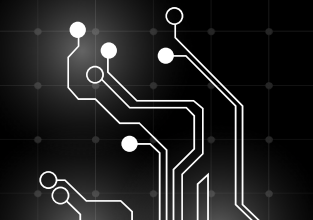
Table Structure

```
CREATE TABLE table_name (
    column1 data_type CONSTRAINTS,
    column2 data_type CONSTRAINTS,
    ...
);
```



Example

```
CREATE TABLE Employees (
    EmployeeID SERIAL PRIMARY KEY,
    FirstName VARCHAR(50) NOT NULL,
    LastName VARCHAR(50) NOT NULL,
    DepartmentID INT,
    HireDate DATE,
    Salary NUMERIC(10, 2) CHECK (Salary > 0)
);
```

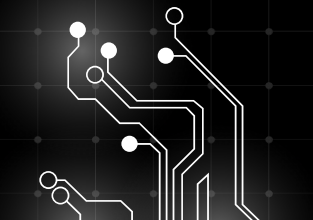


Dropping Tables

Dropping a table is like removing the shelf completely. This deletes the table and all the data stored in it.

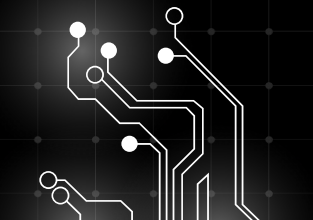
Table Structure

DROP TABLE table_name;

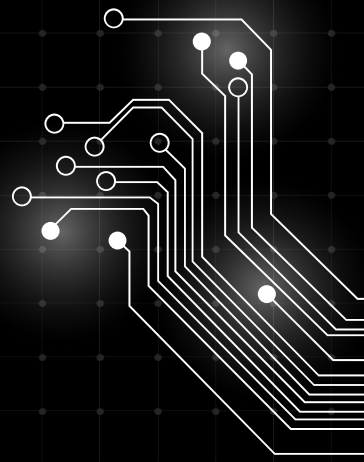


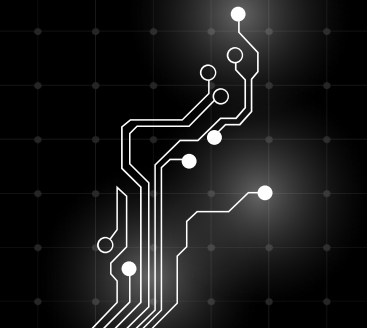
Example

```
DROP TABLE Books;
```



SELECT





SELECT Command

The SELECT command is used to retrieve data from one or more tables. You can specify which columns to retrieve, filter rows using conditions, and even aggregate data.

Structure

Syntax

SELECT column1, column2
FROM table_name WHERE
condition;

Example

SELECT FirstName, LastName
FROM Employees WHERE
DepartmentID = 1;

INSERT

INSERT Command

The INSERT command is used to add new records to a table.

Structure

Syntax

```
INSERT INTO table_name
(column1, column2) VALUES
(value1, value2);
```

Example

```
INSERT INTO Employees
(FirstName, LastName,
DepartmentID, HireDate, Salary)
VALUES ('John', 'Doe', 1,
'2024-01-01', 60000);
```

UPDATE

UPDATE Command

The UPDATE command is used to modify existing records in a table.

Structure

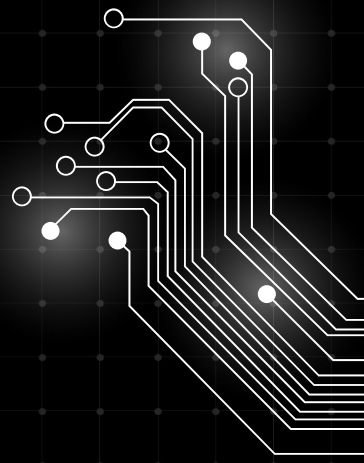
Syntax

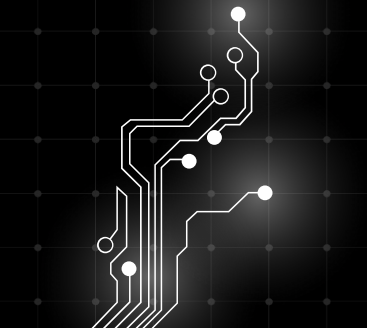
UPDATE table_name SET column1 = value1, column2 = value2 WHERE condition;

Example

UPDATE Employees SET Salary = 65000 WHERE EmployeeID = 1;

DELETE





DELETE Command

The DELETE command is used to remove records from a table.

Structure

Syntax

```
DELETE FROM table_name
WHERE condition;
```

Example

```
DELETE FROM Employees
WHERE EmployeeID = 1;
```

ALIAS

ALIASES

Aliases in SQL are temporary names given to columns or tables in a query.

They make the output of a query easier to read, especially when dealing with complex expressions or when joining tables with similar column names.

ALIASES

Aliases are often used to rename columns in the result set or to make SQL queries more concise and understandable.

Alias Types

Column Alias

Column aliases are used to rename the columns in the result set of a query. The new name is assigned using the AS keyword, though the AS keyword is optional in many SQL databases.

Table Alias

Table aliases are used to assign a temporary name to a table, which is particularly useful when you have long table names or when performing self-joins (joining a table with itself).

Structure - Column

Syntax

```
SELECT column_name AS
alias_name
FROM table_name;
```

Example

```
SELECT FirstName AS Name,
Salary AS MonthlySalary
FROM Employees;
```

Structure - Table

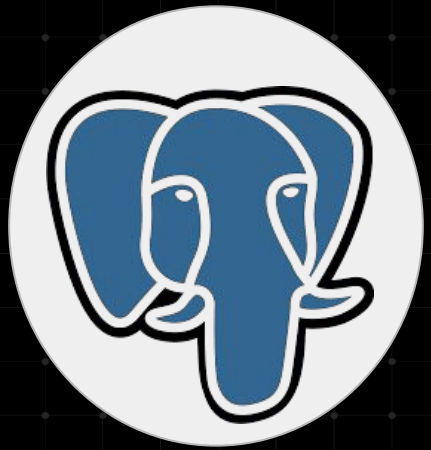
Syntax

```
SELECT column_name
FROM table_name AS
alias_name;
```

Example

```
SELECT e.FirstName,
d.DepartmentName
FROM Employees AS e
JOIN Departments AS d ON
e.DepartmentID =
d.DepartmentID;
```

END OF DAY 1!



02

Filtering & Sorting Data

Day 2

1. Using WHERE to filter data.
2. Sorting results with ORDER BY.
3. Working with text, numbers, and date functions.

WHERE CLAUSE

The WHERE clause in SQL is used to filter records in a query based on specific conditions. It allows you to retrieve only the rows that meet the criteria defined in the WHERE clause, making it a powerful tool for narrowing down your query results.

Basic Syntax

```
SELECT column1, column2
FROM table_name
WHERE condition;
```

Example

```
SELECT FirstName, LastName
FROM Employees
WHERE DepartmentID = 2;
```

Common Operators

=	!= , <>	> , <
Equal to	Not equal to	Greater and less than
<= , >=	LIKE	BETWEEN
Greater or Equal, less or equal	Pattern matching with wildcards	Between two values (inclusive)

Example

```
SELECT FirstName, LastName
FROM Employees
WHERE DepartmentID = 2 AND Salary >
50000;
```

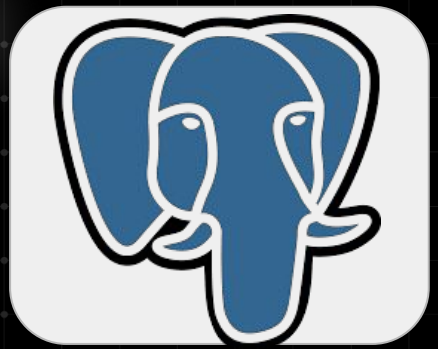
LIKE Operator

%

Represents zero, one, or multiple characters.

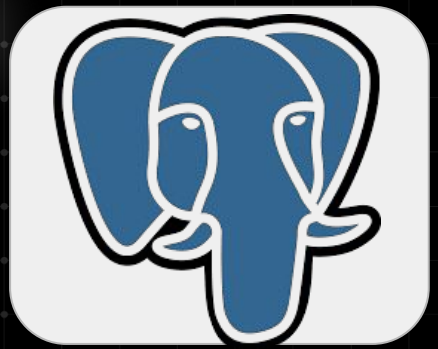
_

Represents a single character.



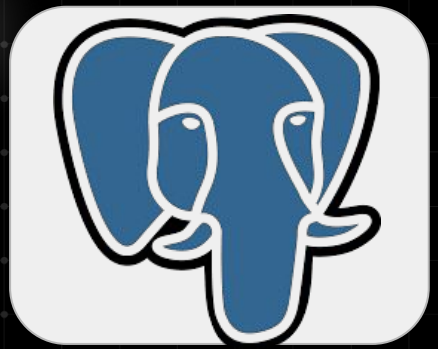
Example 01

```
SELECT FirstName, LastName
FROM Employees
WHERE FirstName LIKE 'Jo%';
```



Example 02

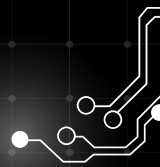
```
SELECT FirstName, LastName
FROM Employees
WHERE FirstName LIKE '___n';
```



Example 03

```
SELECT FirstName, LastName
FROM Employees
WHERE LastName LIKE '_ar%';
```

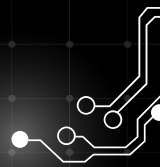
Sorting Results with ORDER BY



Description:

The ORDER BY clause is used to sort the result set of a query by one or more columns. You can sort the results in ascending (ASC) or descending (DESC) order.

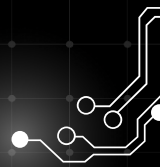
Sorting Results with ORDER BY



Basic Syntax:

```
SELECT column1, column2
FROM table_name
ORDER BY column1 [ASC|DESC], column2 [ASC|DESC];
```

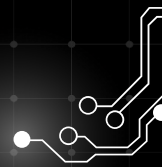
Sorting Results with ORDER BY



Example:

```
SELECT FirstName, LastName, Salary
FROM Employees
ORDER BY Salary DESC;
```

Sorting With Multiple Columns



Example:

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

Working with Text, Numbers, and Date Functions

Text Functions

1.1 Length

LENGTH

Returns the length of a string

Example:

```
SELECT LENGTH('Hello') AS LengthOfString;
```

Upper & Lower

1.2

UPPER & LOWER

Converts a string to upper or lower case.

Example:

```
SELECT UPPER('hello') AS UpperCase, LOWER('WORLD') AS LowerCase;
```

CONCAT

1.3

CONCAT

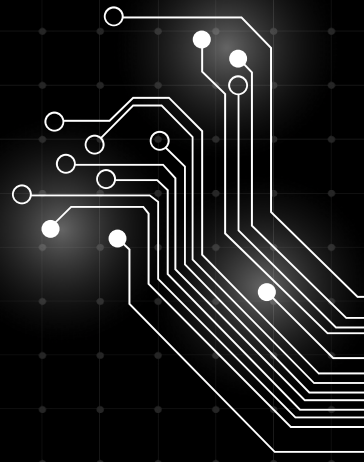
Combines two or more strings.

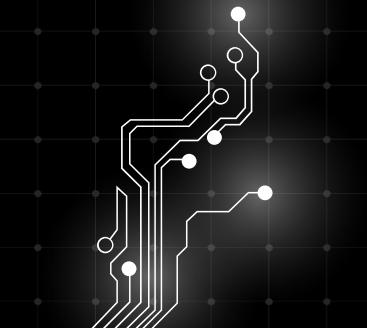
Example:

```
SELECT SUBSTRING('PostgreSQL', 1, 4) AS SubString;
```

SUBSTRING

1.4





SUBSTRING

Extracts a substring from a string.

Example:

```
SELECT CONCAT(FirstName, ' ', LastName) AS FullName
FROM Employees;
```

Number Functions 2.1 Round

ROUND

Rounds a number to a specified number of decimal places.

Example:

```
SELECT ROUND(123.4567, 2) AS RoundedNumber;
```

Ceil & Floor

2.2

CEIL & FLOOR

CEIL rounds a number up to the nearest integer, and FLOOR rounds a number down.

Example:

```
SELECT CEIL(123.4) AS Ceiling, FLOOR(123.4) AS Floor;
```

ABS 2.3

ABS

Returns the absolute value of a number.

Example:

```
SELECT ABS(-15) AS AbsoluteValue;
```

MOD 2.4

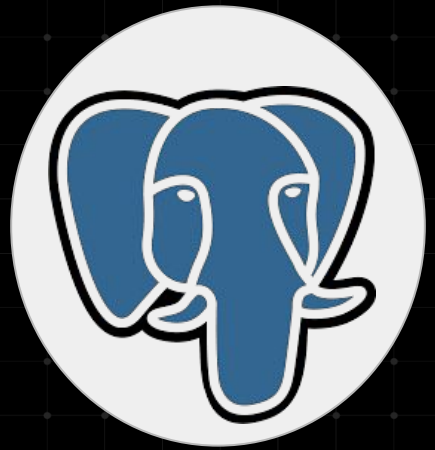
MOD

Returns the remainder of a division operation.

Example:

```
SELECT MOD(10, 3) AS Remainder;
```

END OF DAY 2!



03

Aggregating Data

Day 3

1. Introduction to aggregate functions: COUNT, SUM, AVG, MIN, MAX.
2. Using GROUP BY to aggregate data.
3. Filtering grouped data with HAVING.
4. Practical examples and exercises.

Aggregate Functions

Aggregate functions in SQL are used to perform calculations on a set of values and return a single value. They are commonly used in conjunction with the GROUP BY clause to group rows that have the same values in specified columns into summary rows.

Common Functions

COUNT

Returns the number of rows that match a specified condition.

SUM

Returns the total sum of a numeric column.

AVG

Returns the average value of a numeric column.

MIN / MAX

Returns the smallest value in a column.
Returns the largest value in a column.

COUNT Example

```
SELECT COUNT(*) AS NumberOfEmployees
FROM Employees;
```

SUM Example

```
SELECT SUM(Salary) AS TotalSalaries

FROM Employees;
```

AVG Example

```
SELECT AVG(Salary) AS AverageSalary

FROM Employees;
```

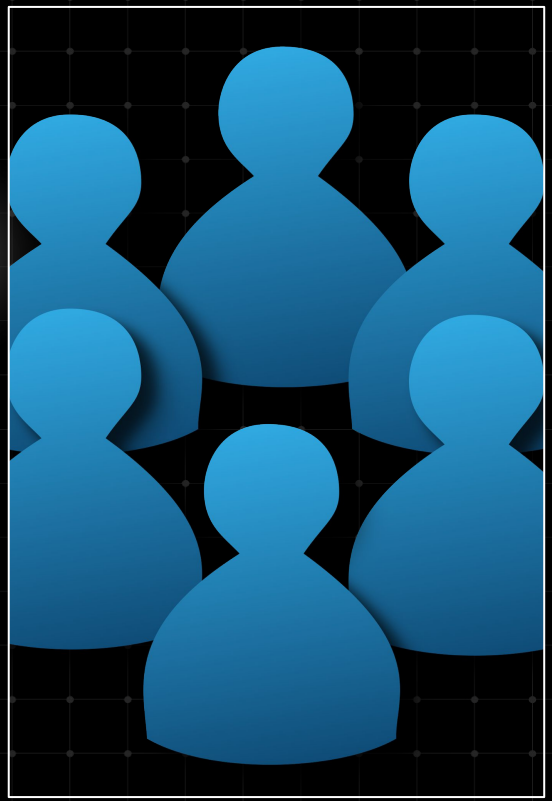
MIN Example

```
SELECT MIN(Salary) AS LowestSalary

FROM Employees;
```

MAX Example

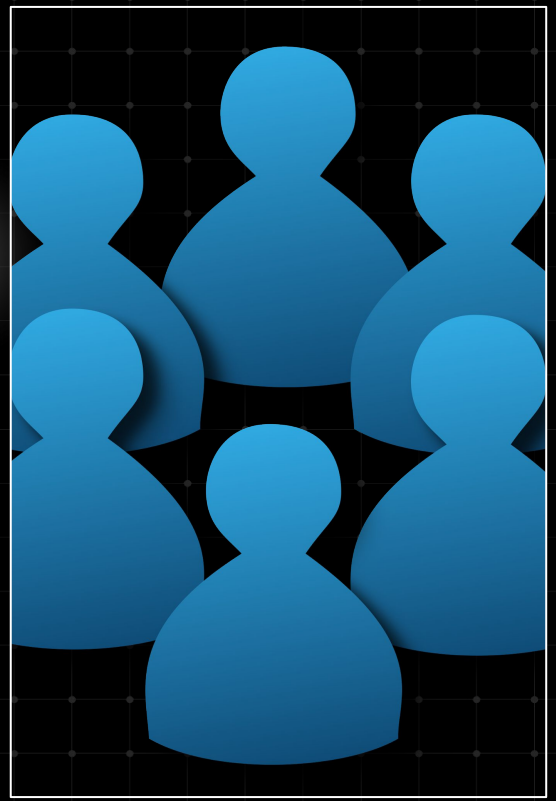
```
SELECT MAX(Salary) AS HighestSalary

FROM Employees;
```

GROUP BY



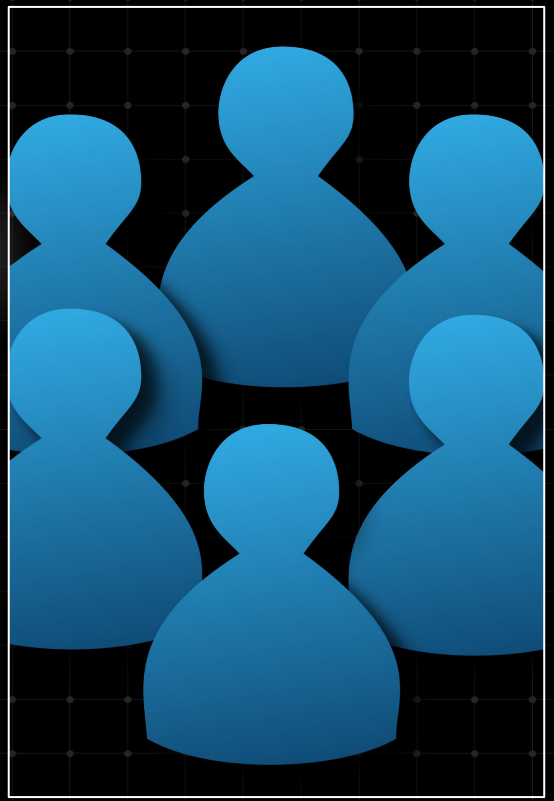
GROUP BY

The GROUP BY clause is used in SQL to arrange identical data into groups. This is typically used in conjunction with aggregate functions to produce summary reports.



Basic Syntax

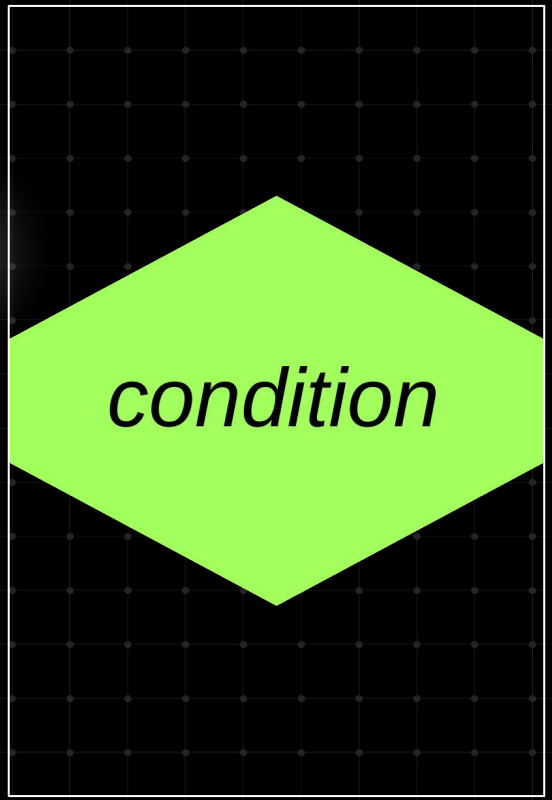
```
SELECT column1,
AGGREGATE_FUNCTION(column2)

FROM table_name

GROUP BY column1;
```



Example

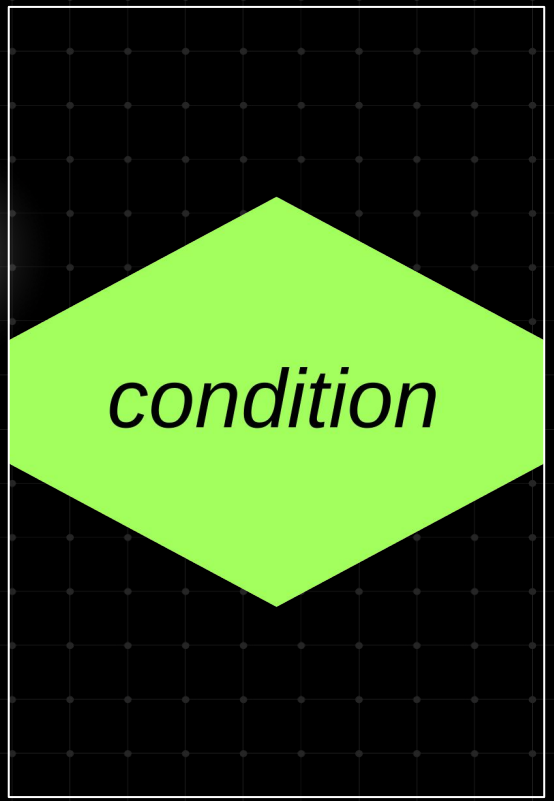
```
SELECT DepartmentID, COUNT(*) AS
NumberOfEmployees

FROM Employees

GROUP BY DepartmentID;
```

HAVING



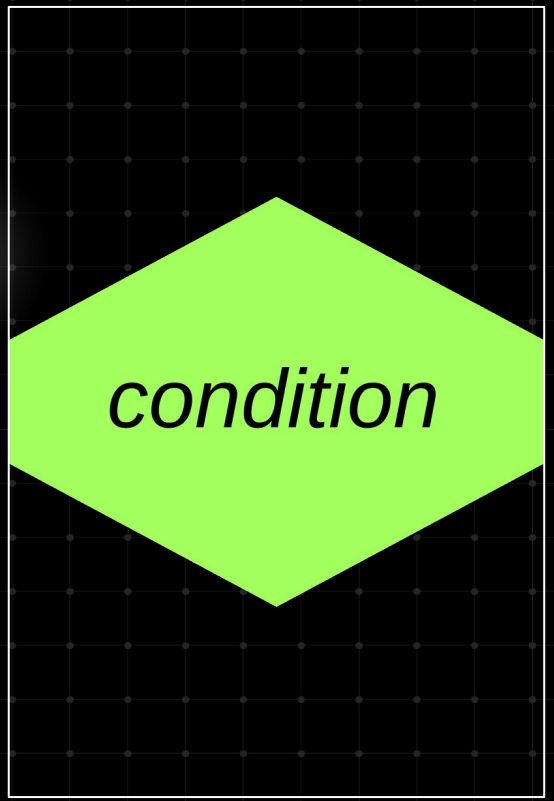
HAVING

The HAVING clause is used to filter records that work with GROUP BY. It is similar to the WHERE clause, but it is used to filter grouped rows after aggregation has been performed.



Basic Syntax

```
SELECT column1,
AGGREGATE_FUNCTION(column2)

FROM table_name

GROUP BY column1

HAVING condition;
```



Example

```
SELECT DepartmentID, COUNT(*) AS
NumberOfEmployees

FROM Employees

GROUP BY DepartmentID

HAVING COUNT(*) > 5;
```

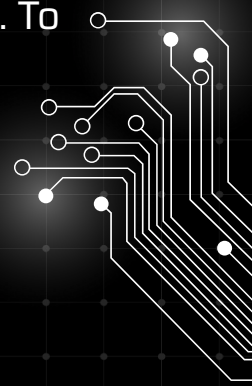
Practical Examples and Exercises

EXERCISE 01

01) Calculate Total Sales by Product

- Suppose you have a Sales table with columns ProductID, Quantity, and Price. To calculate the total sales for each product, you can use the following query:

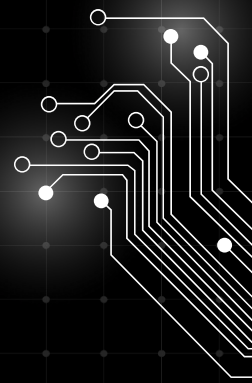
```
SELECT ProductID, SUM(Quantity * Price) AS TotalSales
FROM Sales
GROUP BY ProductID;
```



01) Calculate Total Sales by Product

Exercise 1:

Task: Write a query to find the average salary for each department.

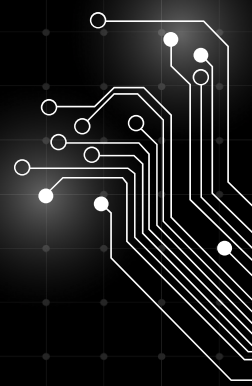


EXERCISE 02

02) Find Departments with High Average Salaries

- To find departments where the average salary is greater than $60,000

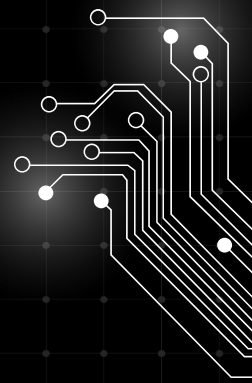
```
SELECT DepartmentID, AVG(Salary) AS AverageSalary
FROM Employees
GROUP BY DepartmentID
HAVING AVG(Salary) > 60000;
```



02) Find Departments with High Average Salaries

Exercise 2:

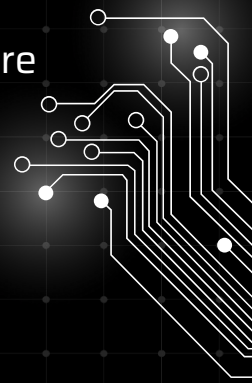
Task: Write a query to count how many products have been sold more than 100 times.



EXERCISE 03

03) Find the Min & Max Hire Dates by Department

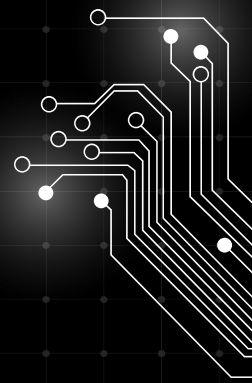
```
SELECT DepartmentID, MIN(HireDate) AS EarliestHire, MAX(HireDate) AS LatestHire
FROM Employees
GROUP BY DepartmentID;
```



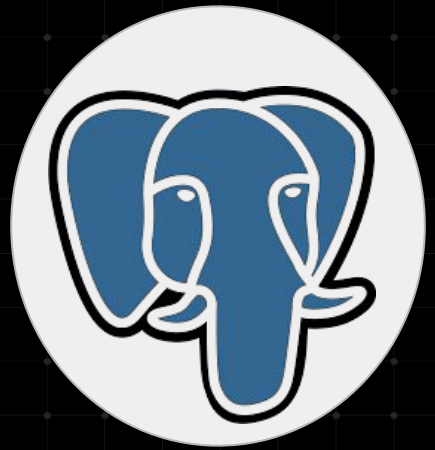
03) Find the Min & Max Hire Dates by Department

Exercise 3:

Task: Write a query to find the total number of employees in departments where the total number of employees is less than 10.



END OF DAY 3!



04

Join & Subqueries

Day 4

1. Introduction to JOIN operations: INNER JOIN, LEFT JOIN, RIGHT JOIN, FULL JOIN.
2. Working with multiple tables in a query.
3. Using subqueries in SELECT, FROM, WHERE clauses.
4. Practical exercises to combine data from different tables.

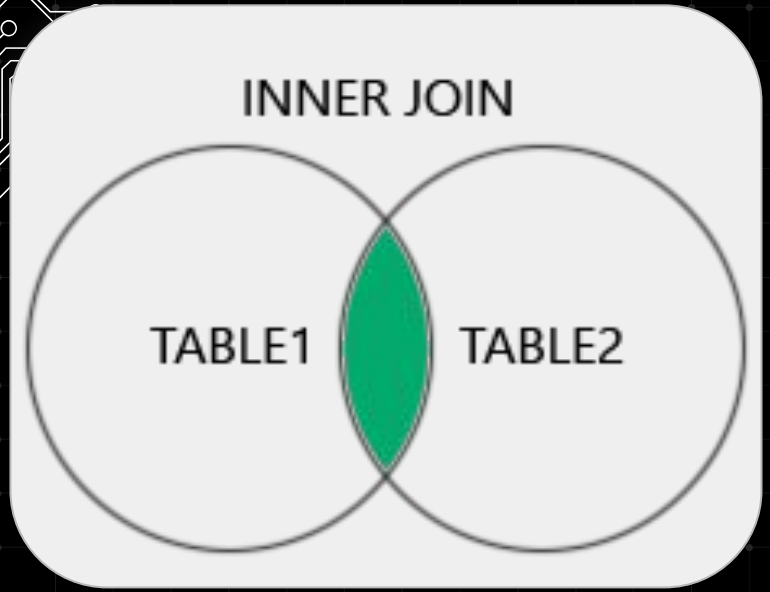
JOIN Operations

JOIN operations in SQL are used to combine rows from two or more tables based on a related column between them. Each type of JOIN determines how records from each table are matched and returned in the result set.

W3 School

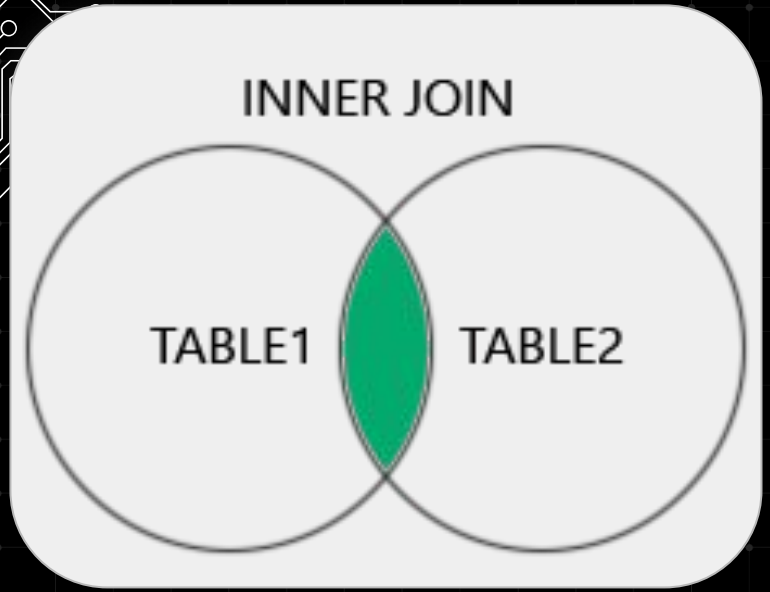
01

INNER JOIN



INNER JOIN

An INNER JOIN returns only the rows that have matching values in both tables. If a row in one table does not have a corresponding row in the other table, it will not be included in the result.

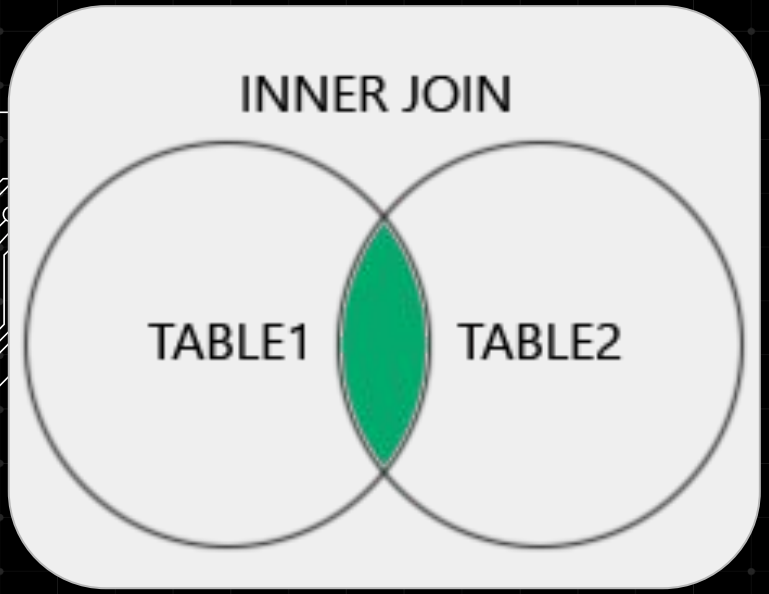


SYNTAX

SELECT columns

FROM table1

INNER JOIN table2 ON table1.column = table2.column;



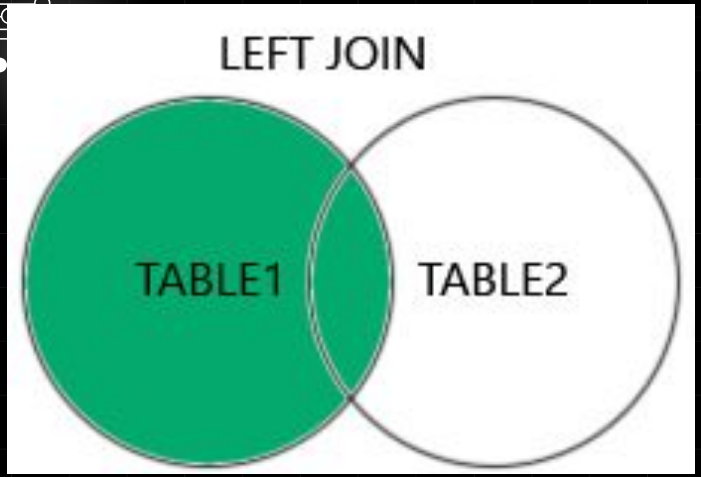
This query returns only the orders that have a matching customer in the Customers table.

EXAMPLE

```
SELECT Orders.OrderID,
Customers.CustomerName

FROM Orders

INNER JOIN Customers ON
Orders.CustomerID =
Customers.CustomerID;
```

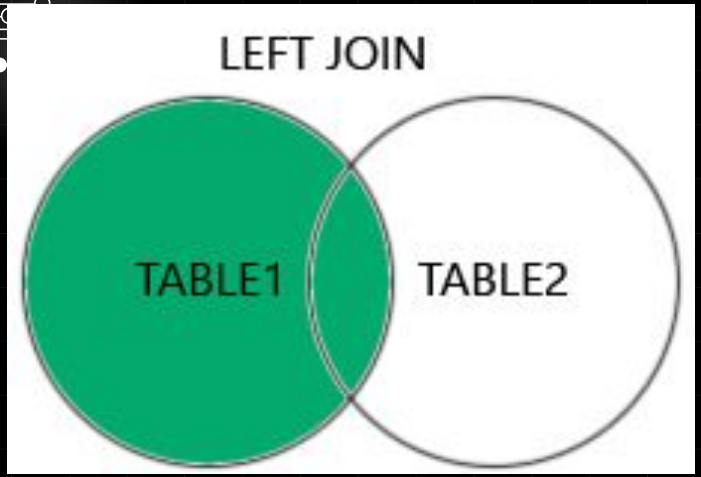
02

LEFT OUTER JOIN



LEFT JOIN

A LEFT JOIN returns all rows from the left table and the matching rows from the right table. If there is no match, the result is NULL on the side of the right table.

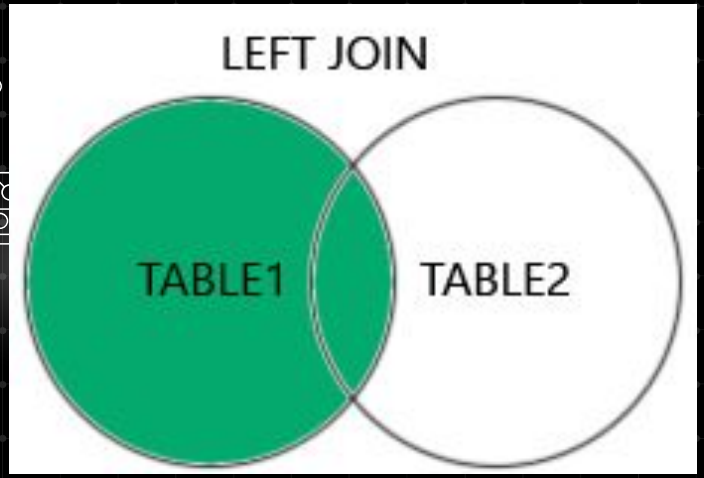


SYNTAX

SELECT columns

FROM table1

LEFT JOIN table2 ON table1.column = table2.column;



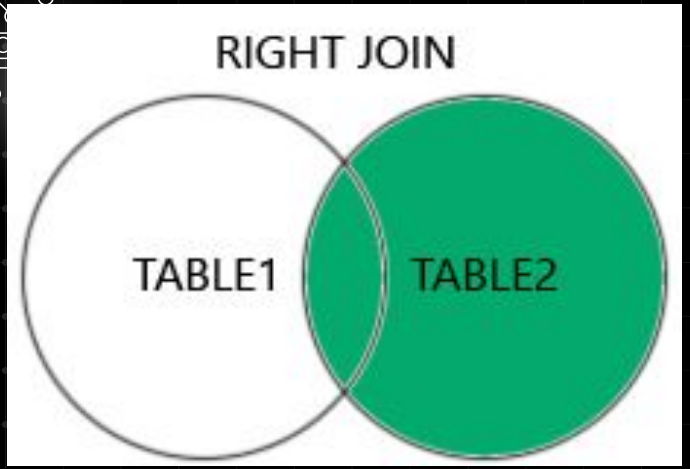
This query returns all orders, including those that do not have a corresponding customer, with NULL in the CustomerName column where there is no match.

EXAMPLE

```
SELECT Orders.OrderID,
Customers.CustomerName

FROM Orders

LEFT JOIN Customers ON
Orders.CustomerID =
Customers.CustomerID;
```

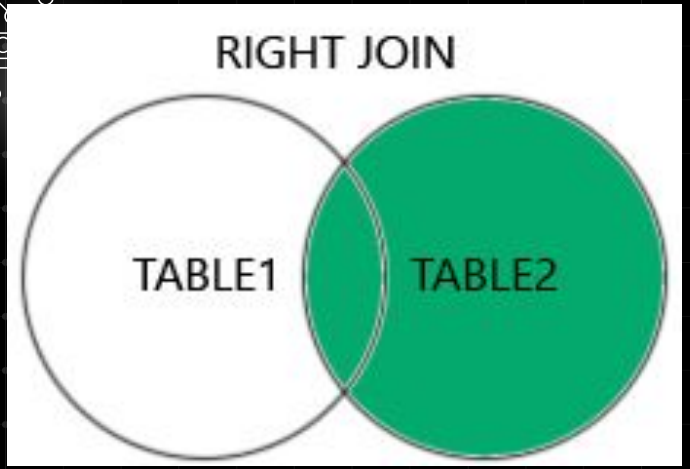
03

RIGHT OUTER JOIN



RIGHT JOIN

A RIGHT JOIN returns all rows from the right table and the matching rows from the left table. If there is no match, the result is NULL on the side of the left table.

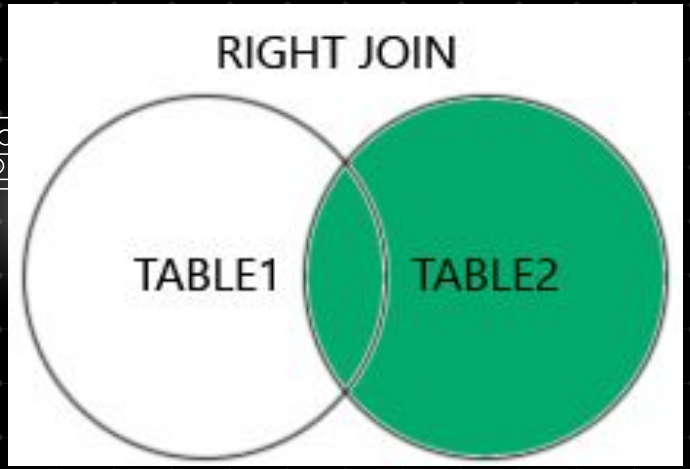


SYNTAX

SELECT columns

FROM table1

RIGHT JOIN table2 ON table1.column = table2.column;



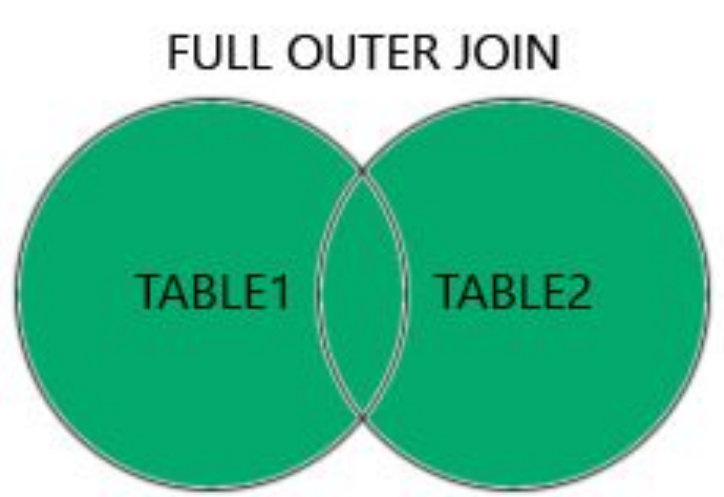
This query returns all customers, including those that have no corresponding orders, with NULL in the OrderID column where there is no match.

EXAMPLE

```
SELECT Orders.OrderID,
Customers.CustomerName

FROM Orders

RIGHT JOIN Customers ON
Orders.CustomerID =
Customers.CustomerID;
```

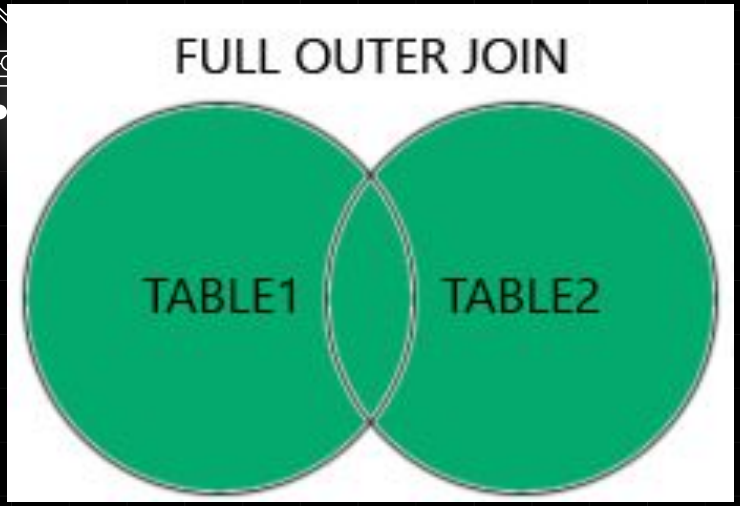
04

FULL JOIN



FULL JOIN

A FULL JOIN returns all rows when there is a match in either the left or right table. Rows without a match in one of the tables will have NULL in the columns from the table without the match.

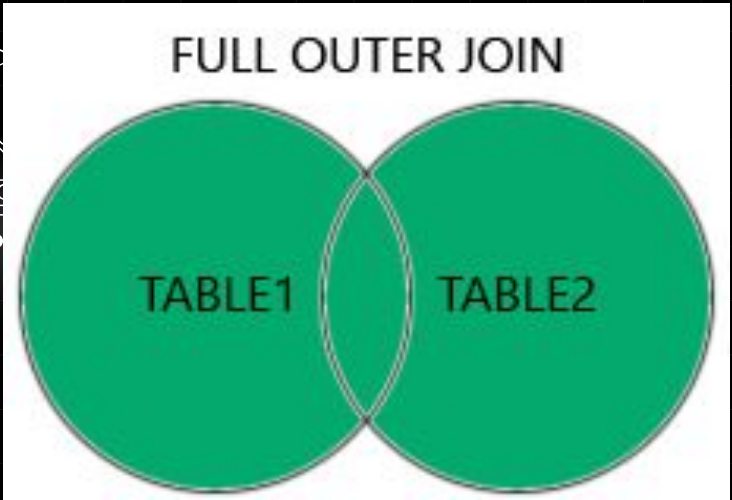


SYNTAX

SELECT columns

FROM table1

FULL JOIN table2 ON table1.column = table2.column;



This query returns all orders and customers, with NULL in the corresponding columns where there is no match.

EXAMPLE

```
SELECT Orders.OrderID,
Customers.CustomerName

FROM Orders

FULL JOIN Customers ON
Orders.CustomerID =
Customers.CustomerID;
```

Working with Multiple Tables in a Query

JOIN operations allow you to retrieve data from multiple tables in a single query. You can use multiple JOINs to combine data from more than two tables.

Example

Scenario

Suppose you have three tables: Orders, Customers, and Products. To get a list of orders, the customers who placed them, and the products ordered,
This query retrieves data from all three tables based on their relationships.

SQL Code

```
SELECT Orders.OrderID,
Customers.CustomerName,
Products.ProductName
FROM Orders
INNER JOIN Customers ON
Orders.CustomerID =
Customers.CustomerID
INNER JOIN Products ON
Orders.ProductID =
Products.ProductID;
```

Using Subqueries in SELECT, FROM, WHERE Clauses

Subquery

A subquery is a query within another query. Subqueries can be used in various parts of an SQL statement, such as in the SELECT, FROM, or WHERE clauses, to perform more complex operations.

Subquery in SELECT

You can use a subquery in the SELECT clause to return a value that will be used as a column in the outer query.

Subquery in SELECT

You can use a subquery in the SELECT clause to return a value that will be used as a column in the outer query.

Example

```
SELECT CustomerName, (SELECT
COUNT(*) FROM Orders WHERE
Orders.CustomerID =
Customers.CustomerID) AS OrderCount

FROM Customers;
```

This query returns each customer and the number of orders they have placed.

Subquery in FROM

A subquery in the FROM clause acts as a temporary table.

Example

```
SELECT AVG(OrderTotal)

FROM (SELECT SUM(Price) AS
OrderTotal FROM Orders GROUP BY
CustomerID) AS OrderSums;

FROM Customers;
```

This query calculates the average total order amount across all customers.

Subquery in WHERE

Subqueries are often used in the WHERE clause to filter results based on the output of another query.

Example

```
SELECT ProductName

FROM Products

WHERE ProductID IN (SELECT ProductID
FROM Orders WHERE CustomerID = 1);
```

This query returns the names of products ordered by the customer with CustomerID = 1.

Practical Exercises to Combine Data from Different Tables

Exercise 1: Retrieve Order and Customer Information

Write a query to list all orders, the customers who placed them, and the date of each order using an INNER JOIN.
Expected Output:

OrderID, CustomerName, OrderDate.

Exercise 2: List All Customers and Their Orders

Use a LEFT JOIN to list all customers and any orders they have placed.
Show NULL for customers without orders.
Expected Output:

CustomerName, OrderID.

Exercise 3: Products Not Ordered by Any Customer

Use a LEFT JOIN and WHERE clause to find products that have not been ordered by any customer.
Expected Output:

ProductName.

Exercise 4: Orders and Total Amount Spent

Write a query to list each order and the total amount spent on that order using a JOIN between Orders and OrderDetails (assuming an OrderDetails table exists).

Expected Output:

OrderID, TotalAmount.

Exercise 5: Aggregate Data with Subqueries

Write a query that lists customers and their average order total using a subquery in the SELECT clause.
Expected Output:

CustomerName, AverageOrderTotal.

ANSWERS FOR EXERCISE

Exercise 01

```
SELECT Orders.OrderID, Customers.CustomerName,
Orders.OrderDate
FROM Orders
INNER JOIN Customers ON Orders.CustomerID =
Customers.CustomerID;
```

Exercise 05

```
SELECT Customers.CustomerName,
(SELECT AVG(SUM(OD.Quantity * OD.Price))
FROM OrderDetails OD
INNER JOIN Orders O ON OD.OrderID = O.OrderID
WHERE O.CustomerID = Customers.CustomerID
GROUP BY O.OrderID) AS AverageOrderTotal
FROM Customers;
```

Exercise 02

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
LEFT JOIN Orders ON Customers.CustomerID =
Orders.CustomerID;
```

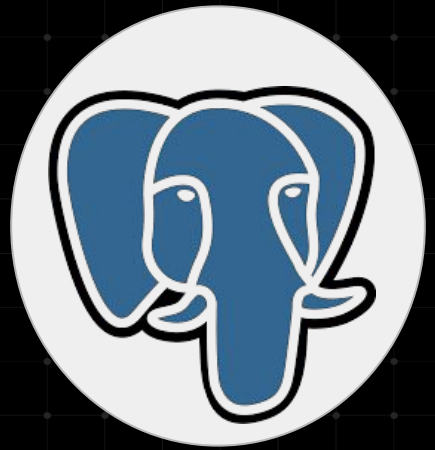
Exercise 03

```
SELECT Products.ProductName
FROM Products
LEFT JOIN Orders ON Products.ProductID =
Orders.ProductID
WHERE Orders.ProductID IS NULL;
```

Exercise 04

```
SELECT Orders.OrderID, SUM(OrderDetails.Quantity *
OrderDetails.Price) AS TotalAmount
FROM Orders
INNER JOIN OrderDetails ON Orders.OrderID =
OrderDetails.OrderID
GROUP BY Orders.OrderID;
```

END OF DAY 4!



05

Data Manipulation

Day 5

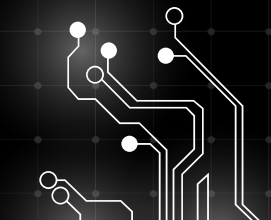
1. Using UNION and INTERSECT to combine results.
2. Modifying data with UPDATE and DELETE.
3. Inserting data using INSERT INTO ... SELECT.

Using UNION and INTERSECT to combine results.

How to use UNION and Intersect

Introduction

SQL provides operators like UNION and INTERSECT to combine the results of multiple SELECT queries. These operators allow you to perform set operations, making it easier to combine and compare data from different queries.



01

UNION

UNION

The UNION operator is used to combine the results of two or more SELECT statements into a single result set. It returns all unique rows from the combined result sets. If you want to include duplicates, you can use UNION ALL.

Syntax

```
SELECT column1, column2, ...
FROM table1
UNION
SELECT column1, column2, ...
FROM table2;
```

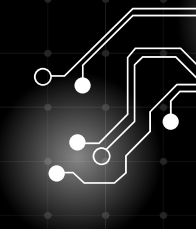
Example

Example

Suppose you have two tables, Employees and Contractors, and you want to list all the people who work for your company, whether they are employees or contractors:

SQL

```
SELECT FirstName, LastName
FROM Employees
UNION
SELECT FirstName, LastName
FROM Contractors;
```



02

INTERSECT

INTERSECT

The INTERSECT operator is used to return only the rows that are common to the result sets of two or more SELECT statements. It provides the intersection of the results.

Syntax

```
SELECT column1, column2, ...
FROM table1
INTERSECT
SELECT column1, column2, ...
FROM table2;
```

Example

Example

If you want to find the people who are both employees and contractors:

SQL

```
SELECT FirstName, LastName
FROM Employees
INTERSECT
SELECT FirstName, LastName
FROM Contractors;
```

Modifying Data with UPDATE and DELETE

How to use UPDATE and DELETE

01

UPDATE

UPDATE

The UPDATE statement is used to modify existing records in a table. You can update one or more columns in one or more rows.

Syntax

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

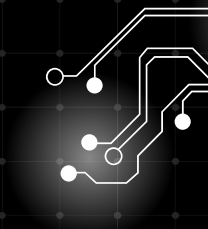
Example

Example

Suppose you want to update the salary of an employee with EmployeeID = 101:

SQL

```
UPDATE Employees
SET Salary = 60000
WHERE EmployeeID = 101;
```



02

DELETE

DELETE

The DELETE statement is used to remove rows from a table. Be cautious with this operation, especially without a WHERE clause, as it will delete all rows from the table.

Syntax

```
DELETE FROM table_name
WHERE condition;
```

Example

Example

If you want to delete a customer with CustomerID = 202 from the Customers table:

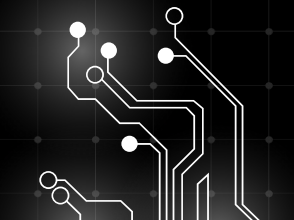
SQL

```
DELETE FROM Customers
WHERE CustomerID = 202;
```

Inserting Data Using INSERT INTO ... SELECT

Introduction

The INSERT INTO ... SELECT statement is used to insert data into a table based on the result of a SELECT query. This is particularly useful for copying data from one table to another.



Syntax

```
INSERT INTO target_table (column1,
column2, ...)
SELECT column1, column2, ...
FROM source_table
WHERE condition;
```

Example

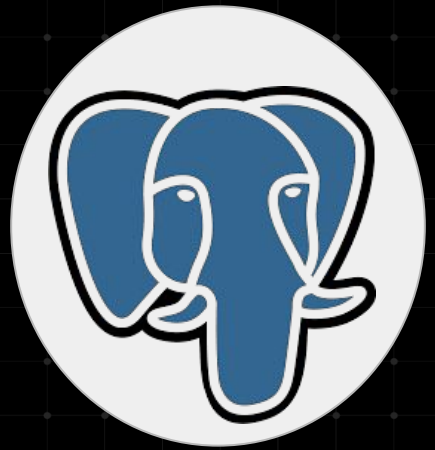
Example

Suppose you want to insert all customers from the Customers table who live in 'New York' into a NewYorkCustomers table:

SQL

```
INSERT INTO
NewYorkCustomers
(CustomerID, CustomerName,
City)
SELECT CustomerID,
CustomerName, City
FROM Customers
WHERE City = 'New York';
```

END OF DAY 5!



06

Real-World Scenarios

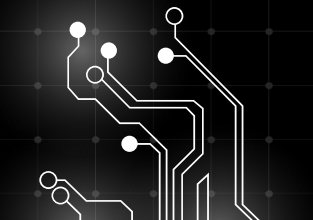
Day 6

1. Hands-on practice with complex queries.
2. Real-world scenarios: sales data analysis, customer segmentation, etc.
3. Q&A and troubleshooting common SQL challenges.

Understanding Complex Queries

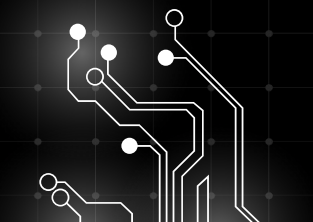
01 - Start with the Edges (Understand the Basics)

Explanation: Just like when you start a puzzle by finding the edge pieces, begin by understanding the basic parts of the query. Look at the SELECT, FROM, WHERE, and JOIN parts. These are like the edges of your puzzle, giving you a clear boundary.



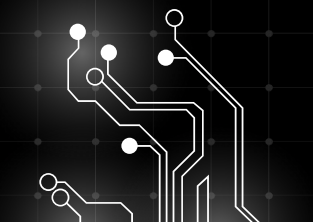
01 - Start with the Edges (Understand the Basics)

Example: In a query, SELECT is like saying, "I want to pick these pieces of information," and FROM tells you where to find them.



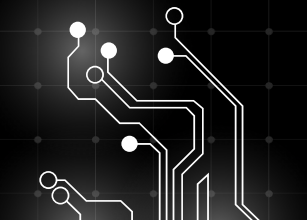
02 - Look for Small Sections (Break Down the Query)

Explanation: Complex queries can be long and confusing, but if you break them into smaller parts, it’s easier to see what each part does. Focus on one piece at a time.



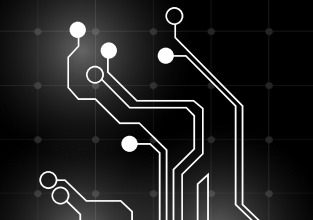
02 - Look for Small Sections (Break Down the Query)

Example: If there's a JOIN in the query, look at just that part and figure out how two tables are being connected, like putting together a small part of the puzzle with similar colors.



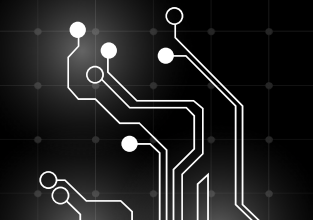
03 - Find the Picture (Understand the Goal)

Explanation: Just like a puzzle has a picture on the box that shows what it should look like, try to understand the goal of the query. What is the query trying to find or show? Knowing this helps you see how all the parts fit together.



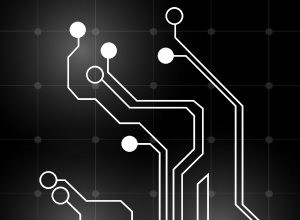
03 - Find the Picture (Understand the Goal)

Example: If the query is trying to find out which customers bought the most, you can think of that as the “picture” you’re trying to create.



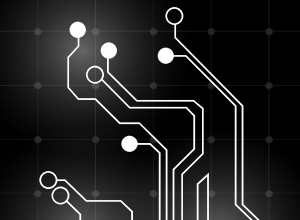
04 - Check for Clues (Look at the Keywords)

Explanation: Some words in SQL are like clues that tell you what's happening. Words like GROUP BY, ORDER BY, or HAVING give you hints about how the data is being grouped, sorted, or filtered.



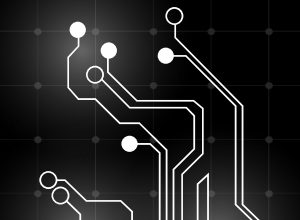
04 - Check for Clues (Look at the Keywords)

Example: ORDER BY is like putting puzzle pieces in a row from smallest to biggest. GROUP BY is like putting all similar pieces together.



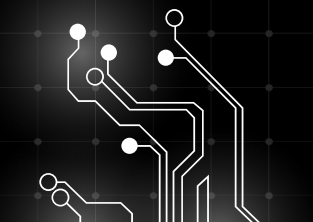
05 - Put It All Together (Assemble the Parts)

Explanation: After you've understood the small parts, try to put them together and see the full query. How do all these pieces work together to solve the problem?



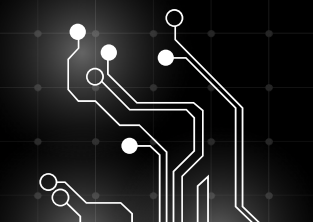
05 - Put It All Together (Assemble the Parts)

Example: Once you know what each piece does, you can see how the whole query works to answer the question, just like seeing the whole picture in the puzzle.



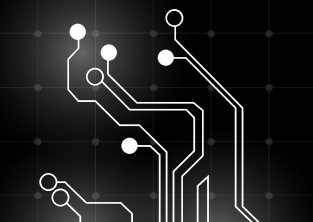
06 - Ask for Help (Work Together)

Explanation: If you're stuck, don't be afraid to ask for help or look for examples. Just like how you might ask someone to help you with a tricky part of a puzzle, it's okay to get guidance on understanding a complex query.



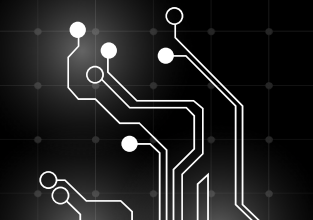
06 - Ask for Help (Work Together)

Example: Sometimes, reading examples or asking someone who’s done it before can make things clearer.



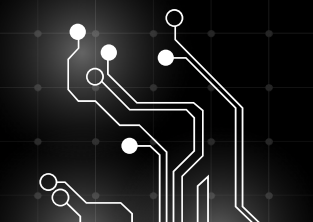
07 - Practice Makes Perfect (Try Different Puzzles)

Explanation: The more puzzles you solve, the better you get. The same goes for SQL queries. Practice with different queries, and soon, even the complex ones will start to make sense.



07 - Practice Makes Perfect (Try Different Puzzles)

Example: Start with easier puzzles (simple queries) and work your way up to harder ones (complex queries).



Example

Suppose you have three tables: Sales, Customers, and Products. You want to find the total sales for each product category by customers who live in 'New York' and have spent more than $1,000 in total.

Answer

```
SELECT P.Category, SUM(S.Amount) AS TotalSales
FROM Sales S
INNER JOIN Customers C ON S.CustomerID = C.CustomerID
INNER JOIN Products P ON S.ProductID = P.ProductID
WHERE C.City = 'New York'
AND C.CustomerID IN (
SELECT CustomerID
FROM Sales
GROUP BY CustomerID
HAVING SUM(Amount) > 1000
)
GROUP BY P.Category;
```

Real-World Scenarios: Sales Data Analysis, Customer Segmentation

01

Sales Data Analysis

Sales data analysis involves examining sales transactions to understand trends, identify top-performing products, and monitor overall business health. SQL can be used to calculate metrics like total sales, average order value, and growth rates.

Example

Example

You want to identify the top 5 products by total sales in the last quarter

This query calculates total sales for each product in the last quarter and lists the top 5 products.

SQL

```
SELECT P.ProductName, SUM(S.Amount)
AS TotalSales
FROM Sales S
INNER JOIN Products P ON S.ProductID =
P.ProductID
WHERE S.SaleDate BETWEEN
'2024-04-01' AND '2024-06-30'
GROUP BY P.ProductName
ORDER BY TotalSales DESC
LIMIT 5;
```

02

Customer Segmentation

Customer segmentation involves dividing customers into distinct groups based on their behaviors, preferences, or demographics. This helps in targeting marketing efforts more effectively.

Example

Example

You want to segment customers based on their total spending

This query segments customers into 'Premium', 'Regular', or 'Basic' based on their total spending.

SQL

```
SELECT C.CustomerID, C.CustomerName,
CASE
WHEN SUM(S.Amount) > 5000 THEN
'Premium'
WHEN SUM(S.Amount) BETWEEN 1000
AND 5000 THEN 'Regular'
ELSE 'Basic'
END AS Segment
FROM Sales S
INNER JOIN Customers C ON S.CustomerID
C.CustomerID
GROUP BY C.CustomerID, C.CustomerName;
```

Q&A and Troubleshooting Common SQL Challenges

01

Common SQL Challenges

Handling NULL Values:
NULL values can cause unexpected results in queries. Use IS NULL or COALESCE() to handle NULLs effectively.

EX:

```
SELECT COALESCE(SUM(Amount), 0) AS
TotalSales
FROM Sales;
```

01

Common SQL Challenges

Performance Issues:
Large datasets can lead to slow query performance. Use indexing, proper JOIN techniques, and avoid unnecessary columns in SELECT statements to improve performance.

01

Common SQL Challenges

Data Type Mismatches:
Ensure that columns being compared in JOIN or WHERE clauses have compatible data types.

01

Common SQL Challenges

Syntax Errors:
Missing commas, incorrect keywords, or misplaced clauses can lead to syntax errors. Always review your query structure.

02

Troubleshooting SQL Queries

Understanding Error Messages:
SQL error messages often provide clues about what went wrong. Carefully read error messages to identify the issue.

02

Troubleshooting SQL Queries

Using EXPLAIN:
The EXPLAIN command can help you understand how a query is being executed and identify potential performance bottlenecks.

EX:
EXPLAIN SELECT * FROM Sales WHERE Amount > 1000;

02

Troubleshooting SQL Queries

Testing Queries Step-by-Step:
Break down complex queries into smaller parts and test each part individually to identify where the issue lies.

Q&A Session

Q1

How do I combine data from multiple tables with different structures?

A: Use JOIN operations, and ensure you understand the relationships between the tables. If tables are unrelated, consider using UNION to combine similar columns.

Q2

How can I improve the performance of my queries?

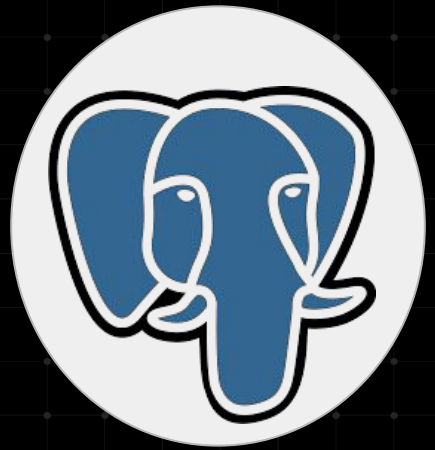
A: Focus on indexing key columns, reducing the number of columns in the SELECT clause, and optimizing JOIN conditions.

Q3

What should I do if my query returns incorrect or unexpected results?

A: Review the logic in your WHERE, JOIN, and GROUP BY clauses. Test the query with smaller datasets to verify each step.

END OF DAY 6!



Tables and SQL

1. Employees Table

EmployeeID	FirstName	LastName	Department	Salary
101	John	Doe	HR	50000
102	Jane	Smith	IT	60000
103	Mike	Johnson	Finance	55000
104	Emily	Davis	Marketing	45000

1. Employees Table

```
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    Department VARCHAR(50),
    Salary DECIMAL(10, 2)
);

INSERT INTO Employees (EmployeeID, FirstName, LastName, Department, Salary)
VALUES
(101, 'John', 'Doe', 'HR', 50000),
(102, 'Jane', 'Smith', 'IT', 60000),
(103, 'Mike', 'Johnson', 'Finance', 55000),
(104, 'Emily', 'Davis', 'Marketing', 45000);
```

2. Customers Table

CustomerID	CustomerName	City	TotalSpent
201	Alice Johnson	New York	3000
202	Bob Smith	Los Angeles	5000
203	Charlie Brown	New York	12000
204	Diana Prince	Chicago	4000

2. Customers Table

```
CREATE TABLE Customers (
    CustomerID INT PRIMARY KEY,
    CustomerName VARCHAR(100),
    City VARCHAR(50),
    TotalSpent DECIMAL(10, 2)
);

INSERT INTO Customers (CustomerID, CustomerName, City, TotalSpent) VALUES
(201, 'Alice Johnson', 'New York', 3000),
(202, 'Bob Smith', 'Los Angeles', 5000),
(203, 'Charlie Brown', 'New York', 12000),
(204, 'Diana Prince', 'Chicago', 4000);
```

3. Products Table

ProductID	ProductName	Category	Price
301	Laptop	Electronics	1200
302	Smartphone	Electronics	800
303	Desk Chair	Furniture	150
304	Coffee Maker	Appliances	80

3. Products Table

```
CREATE TABLE Products (
    ProductID INT PRIMARY KEY,
    ProductName VARCHAR(100),
    Category VARCHAR(50),
    Price DECIMAL(10, 2)
);

INSERT INTO Products (ProductID, ProductName, Category, Price) VALUES
(301, 'Laptop', 'Electronics', 1200),
(302, 'Smartphone', 'Electronics', 800),
(303, 'Desk Chair', 'Furniture', 150),
(304, 'Coffee Maker', 'Appliances', 80);
```

4. Orders Table

OrderID	CustomerID	ProductID	OrderDate	Amount
401	201	301	2024-07-01	1200
402	203	302	2024-07-05	800
403	202	303	2024-07-07	150
404	204	304	2024-07-10	80

4. Orders Table

```
CREATE TABLE Orders (
    OrderID INT PRIMARY KEY,
    CustomerID INT,
    ProductID INT,
    OrderDate DATE,
    Amount DECIMAL(10, 2),
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID),
    FOREIGN KEY (ProductID) REFERENCES Products(ProductID)
);

INSERT INTO Orders (OrderID, CustomerID, ProductID, OrderDate, Amount) VALUES
(401, 201, 301, '2024-07-01', 1200),
(402, 203, 302, '2024-07-05', 800),
(403, 202, 303, '2024-07-07', 150),
(404, 204, 304, '2024-07-10', 80);
```

Thanks!

Do you have any questions?

mayamnaizel2013@gmail.com

The Tech Stuff