SQL Operators

SQL Arithmetic Operators

Operator	Description
+	Add
-	Subtract
*	Multiply
/	Divide
%	Modulo

SQL Comparison Operators

Operator	Description
=	Equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
<>	Not equal to
!=	Not equal

SQL Logical Operators

- These operators return a Boolean value which can be either a TRUE or FALSE.
- Logical operators available in SQL are,
  - ANY and ALL
  - AND, OR and NOT
  - BETWEEN
  - EXISTS
  - IN
  - LIKE
  - IS NULL

ANY Operator

- SQL ANY compares a value of the first table with all values of the second table and returns the row if there is a match with any value.
- It has the following syntax:

- SELECT column FROM table1 WHERE column OPERATOR ANY (SELECT column FROM table2);

- SELECT * FROM Teachers WHERE age = ANY (SELECT age FROM Students);

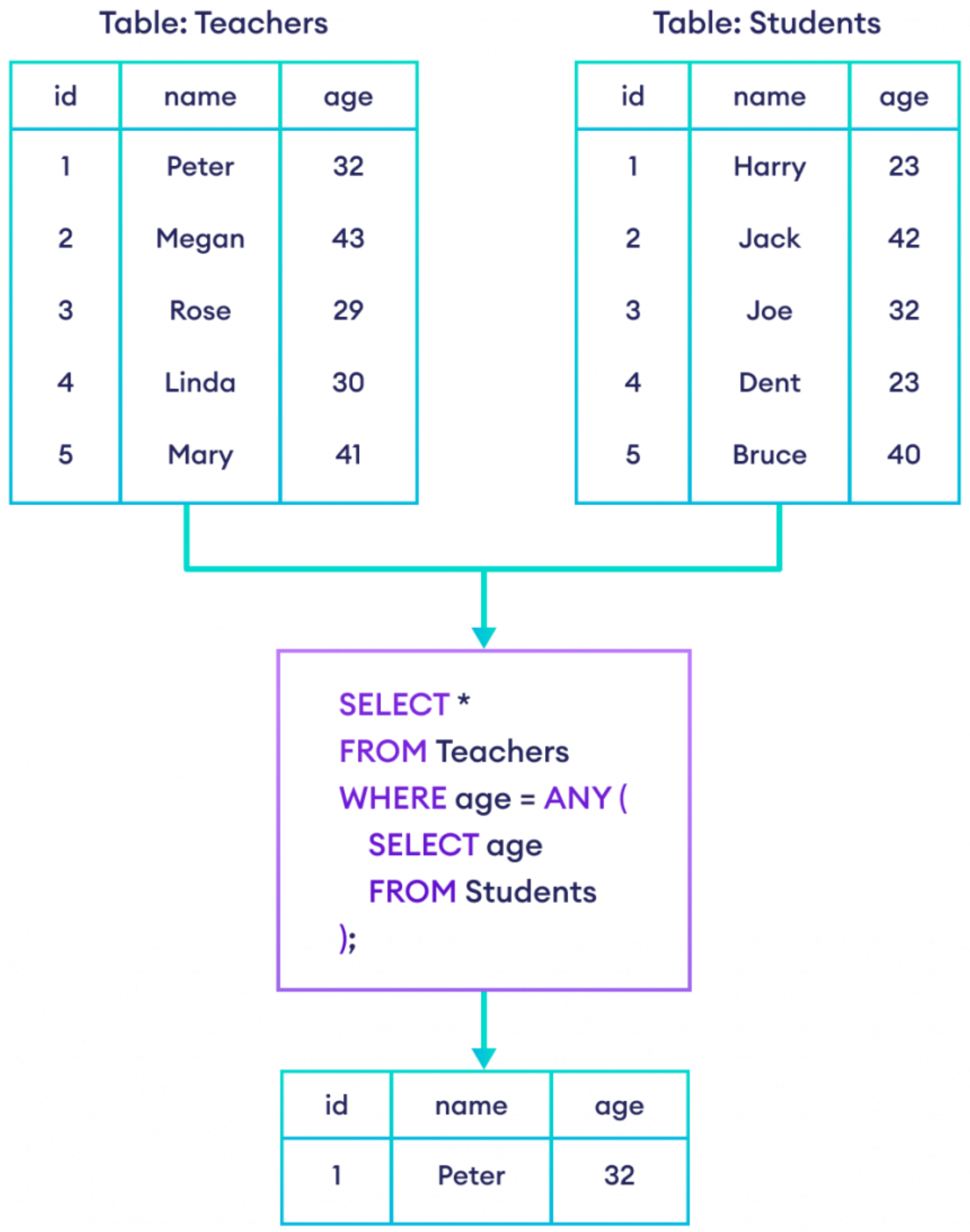


Table: Teachers

id	name	age
1	Peter	32
2	Megan	43
3	Rose	29
4	Linda	30
5	Mary	41

Table: Students

id	name	age
1	Harry	23
2	Jack	42
3	Joe	32
4	Dent	23
5	Bruce	40

```sql
SELECT *
FROM Teachers
WHERE age = ANY (
  SELECT age
  FROM Students
);
```

id	name	age
1	Peter	32

ALL Operator

- SQL ALL compares a value of the first table with all values of the second table and returns the row if there is a match with all values.

- It has the following syntax:
- *SELECT column FROM table1 WHERE column OPERATOR ALL (SELECT column FROM table2);*

- SELECT * FROM Teachers WHERE age > ALL (SELECT age FROM Students);

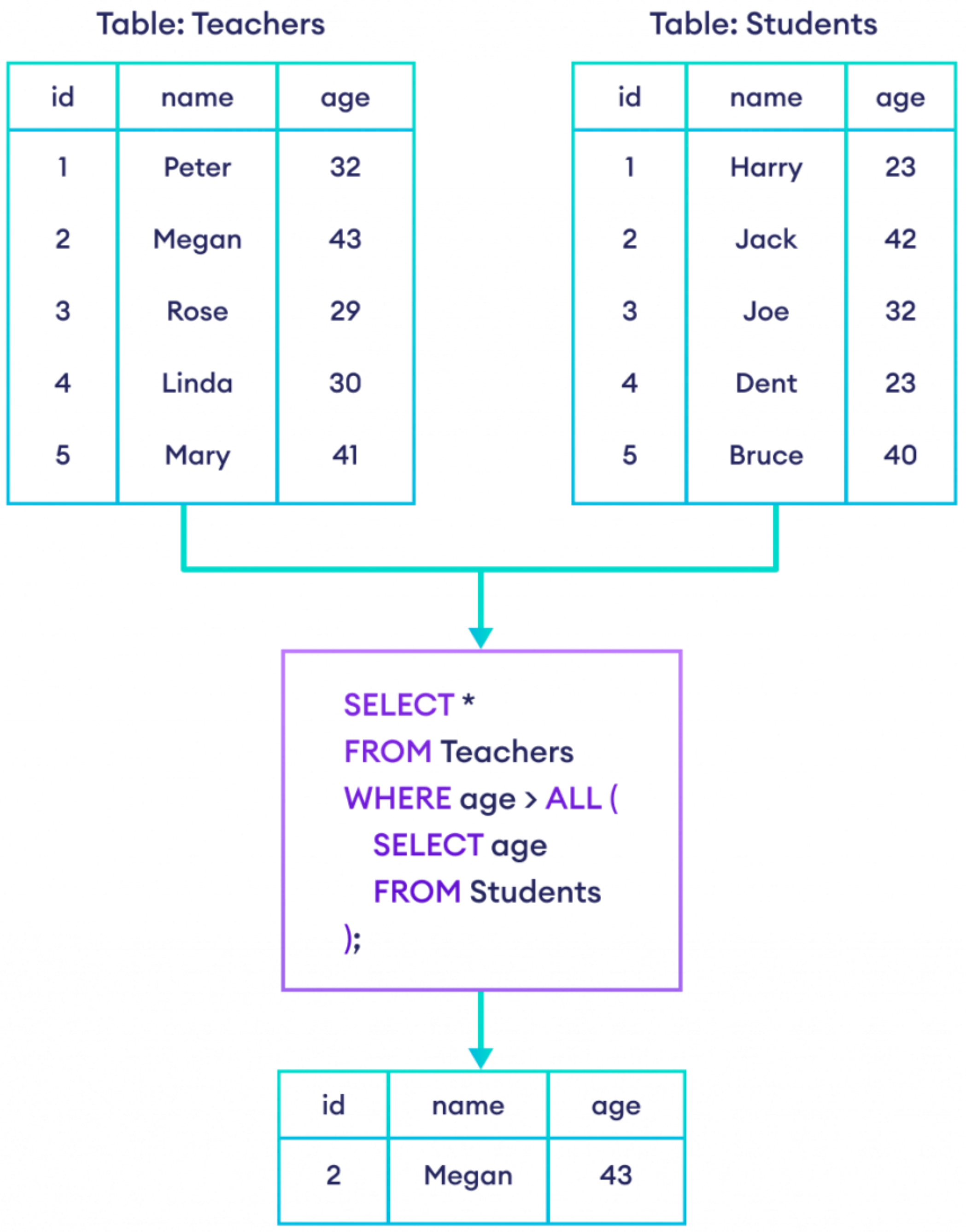


Table: Teachers

id	name	age
1	Peter	32
2	Megan	43
3	Rose	29
4	Linda	30
5	Mary	41

Table: Students

id	name	age
1	Harry	23
2	Jack	42
3	Joe	32
4	Dent	23
5	Bruce	40

```sql
SELECT *
FROM Teachers
WHERE age > ALL (
  SELECT age
  FROM Students
);
```

id	name	age
2	Megan	43

AND Operator

- The SQL AND operator selects data if all conditions are TRUE

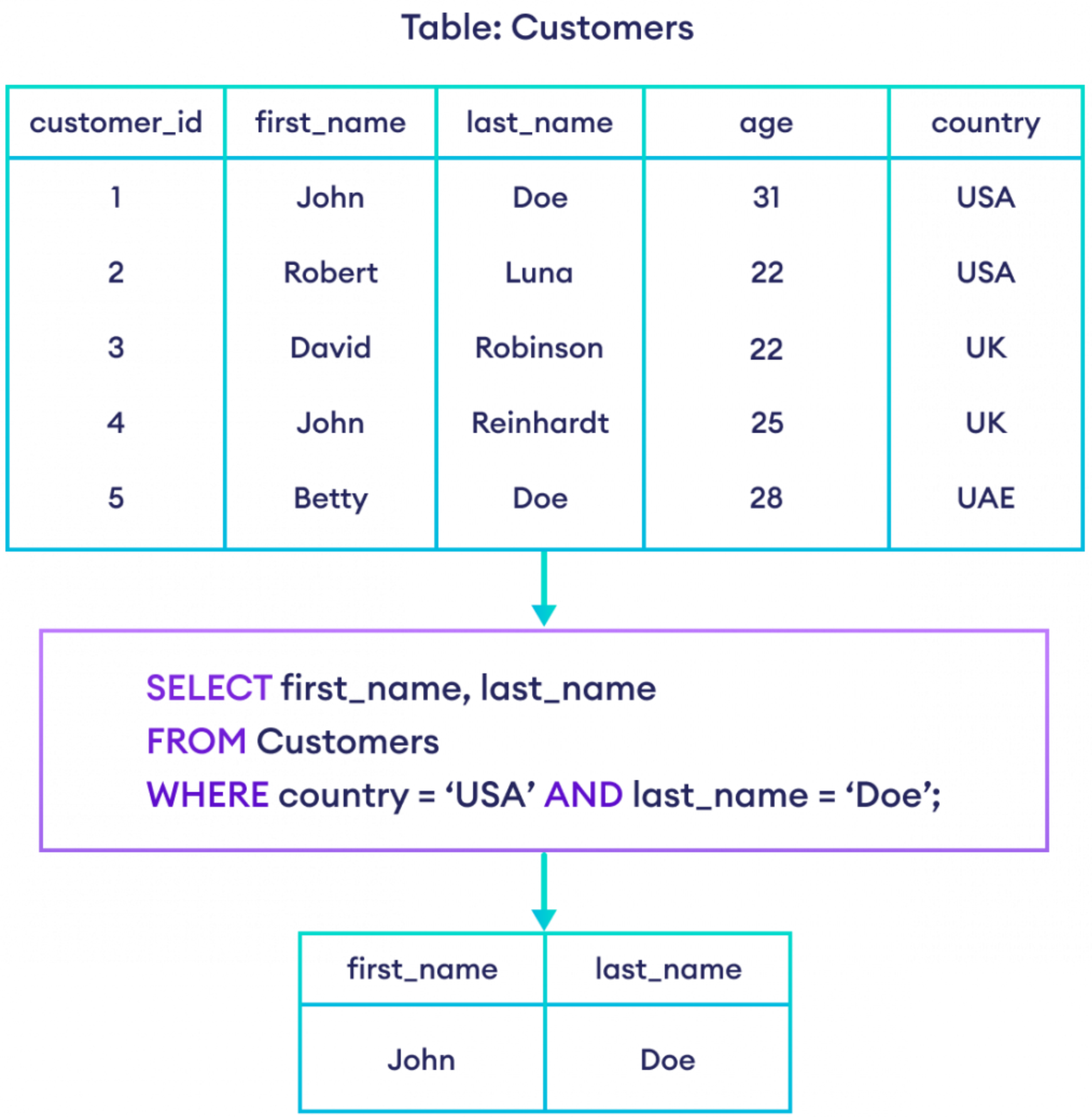


Table: Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

```
SELECT first_name, last_name
FROM Customers
WHERE country = ‘USA’ AND last_name = ‘Doe’;
```

first_name	last_name
John	Doe

OR Operator

- The SQL OR operator selects data if any one condition is TRUE.

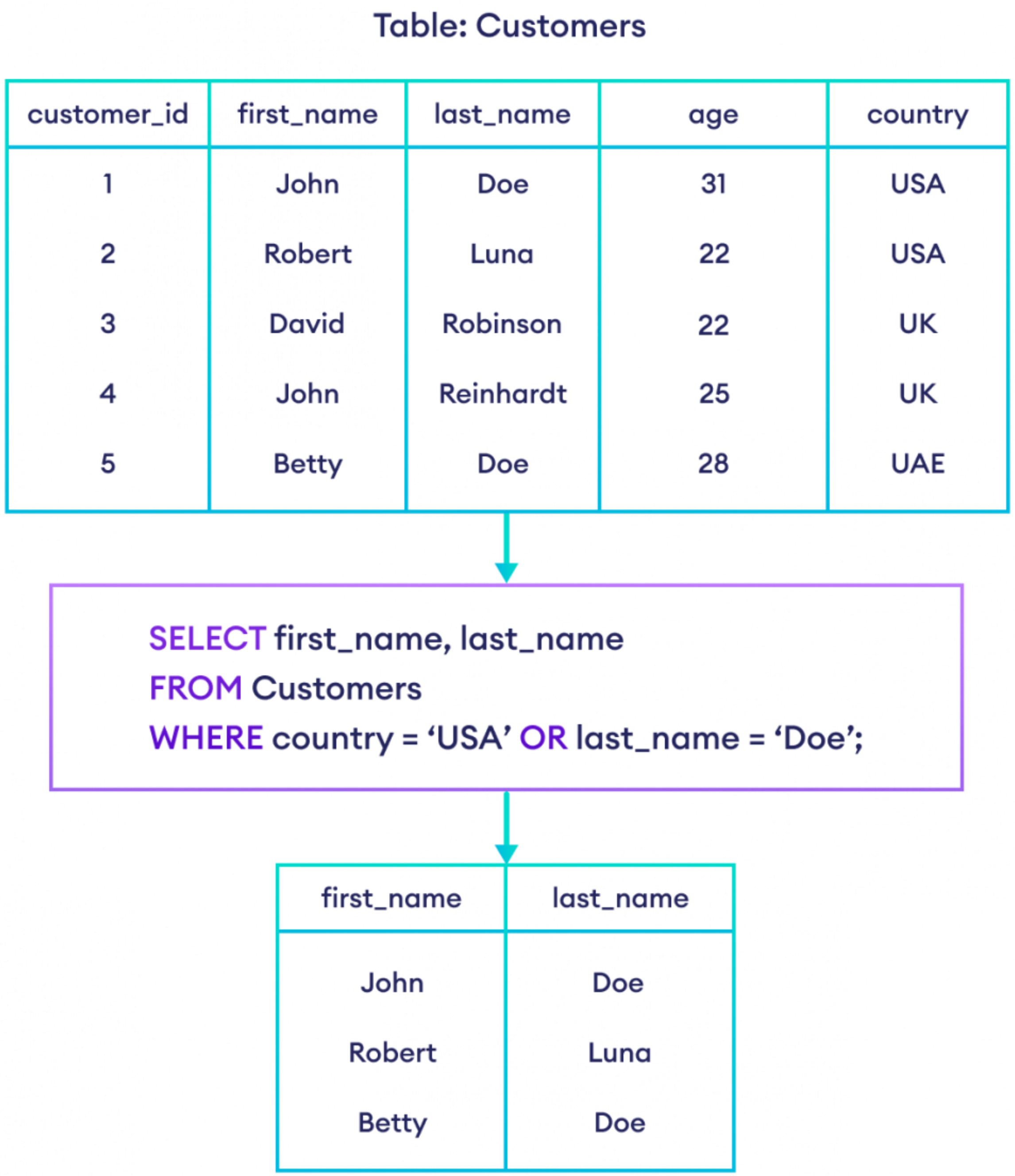


Table: Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

```
SELECT first_name, last_name
FROM Customers
WHERE country = ‘USA’ OR last_name = ‘Doe’;
```

first_name	last_name
John	Doe
Robert	Luna
Betty	Doe

NOT Operator

- The SQL NOT operator selects data if the given condition is FALSE.

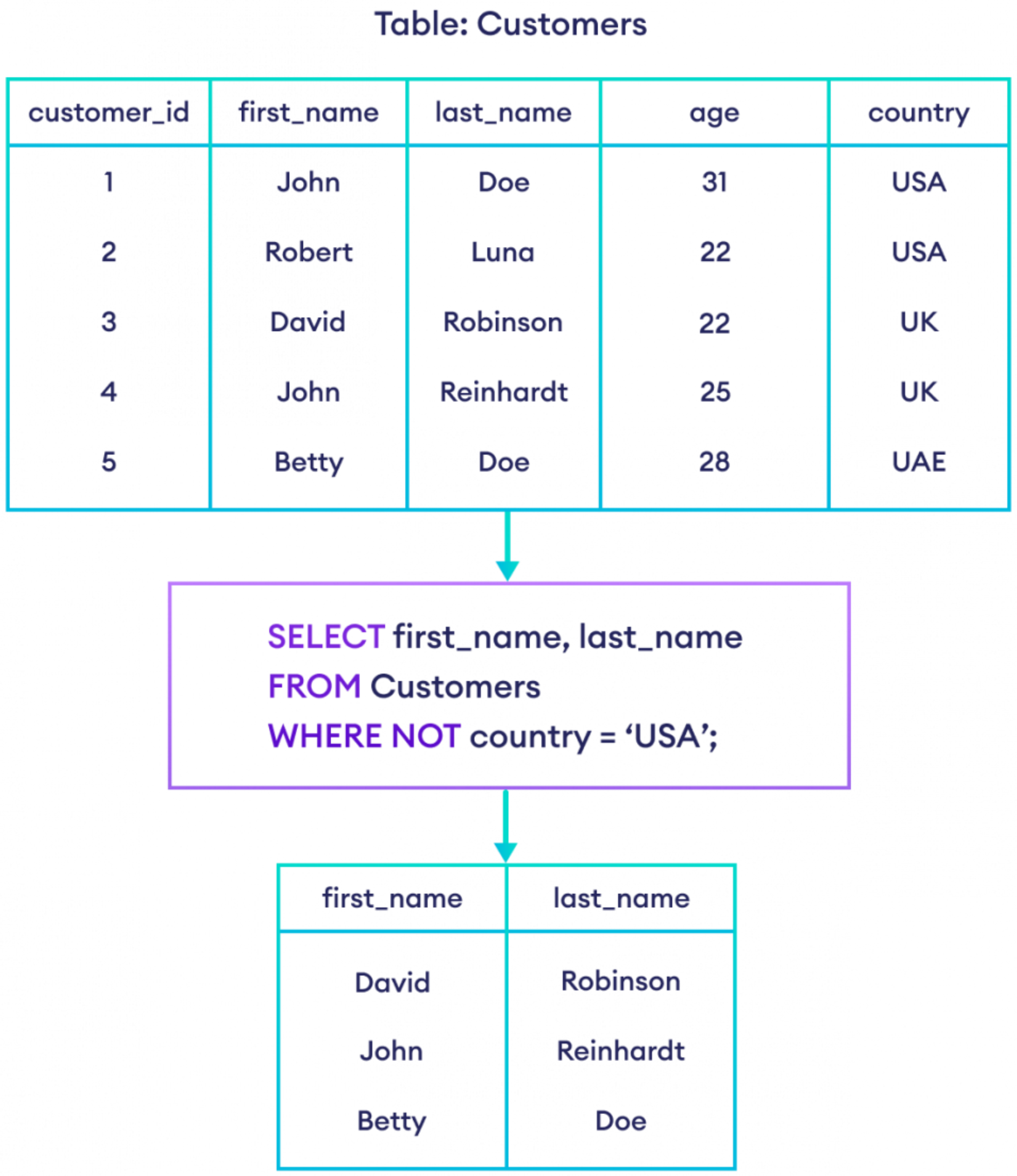


Table: Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

```
SELECT first_name, last_name
FROM Customers
WHERE NOT country = ‘USA’;
```

first_name	last_name
David	Robinson
John	Reinhardt
Betty	Doe

BETWEEN Operator

- In SQL, the BETWEEN operator with the WHERE clause selects values within a given range.

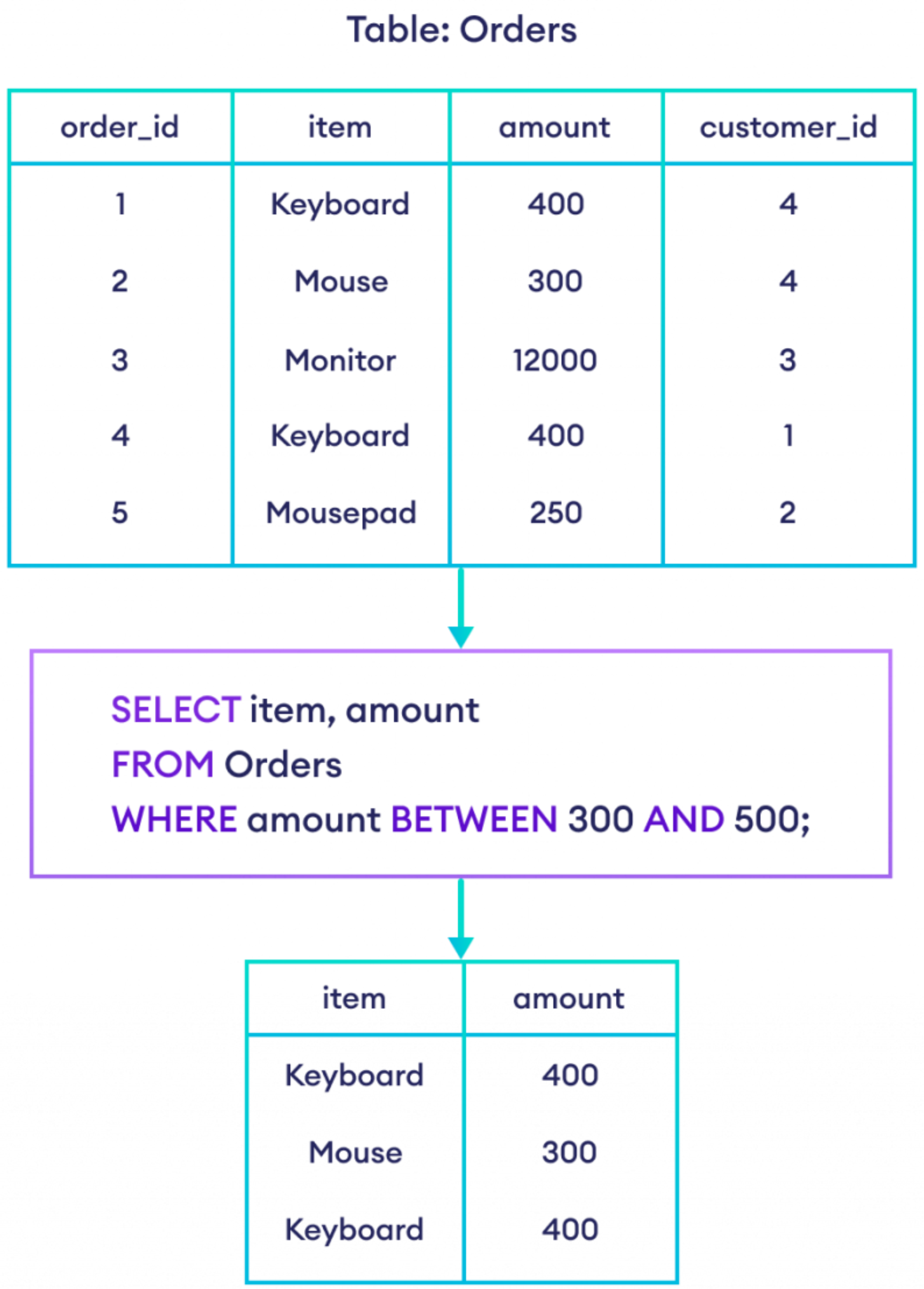


Table: Orders

order_id	item	amount	customer_id
1	Keyboard	400	4
2	Mouse	300	4
3	Monitor	12000	3
4	Keyboard	400	1
5	Mousepad	250	2

```
SELECT item, amount
FROM Orders
WHERE amount BETWEEN 300 AND 500;
```

item	amount
Keyboard	400
Mouse	300
Keyboard	400

IN Operators

- We use the IN operator with the WHERE clause to match values in a list.
- *SELECT column1, column2 FROM table WHERE column IN (value1, value2, ...);*

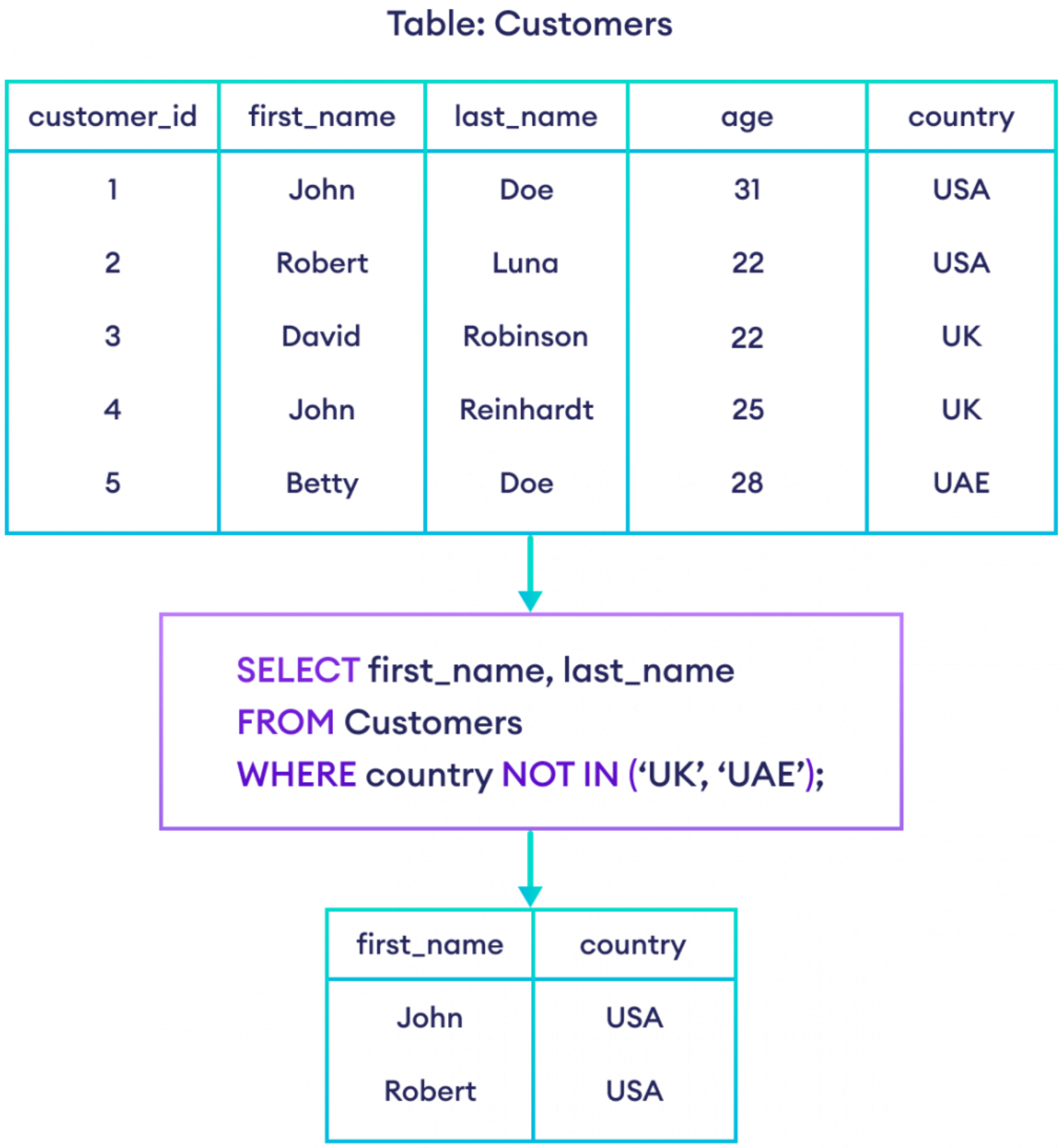


Table: Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

```
SELECT first_name, last_name
FROM Customers
WHERE country NOT IN (‘UK’, ‘UAE’);
```

first_name	country
John	USA
Robert	USA

LIKE Operator

- We use the SQL LIKE operator with the WHERE clause to get a result set that matches the given string pattern.
- Syntax
- *SELECT column1, column2 FROM table_name WHERE columnN LIKE pattern;*

- There are two wildcards often used in conjunction with the LIKE operator:
- The **percent sign %** represents zero, one, or multiple characters
- The **underscore sign _** represents one, single character

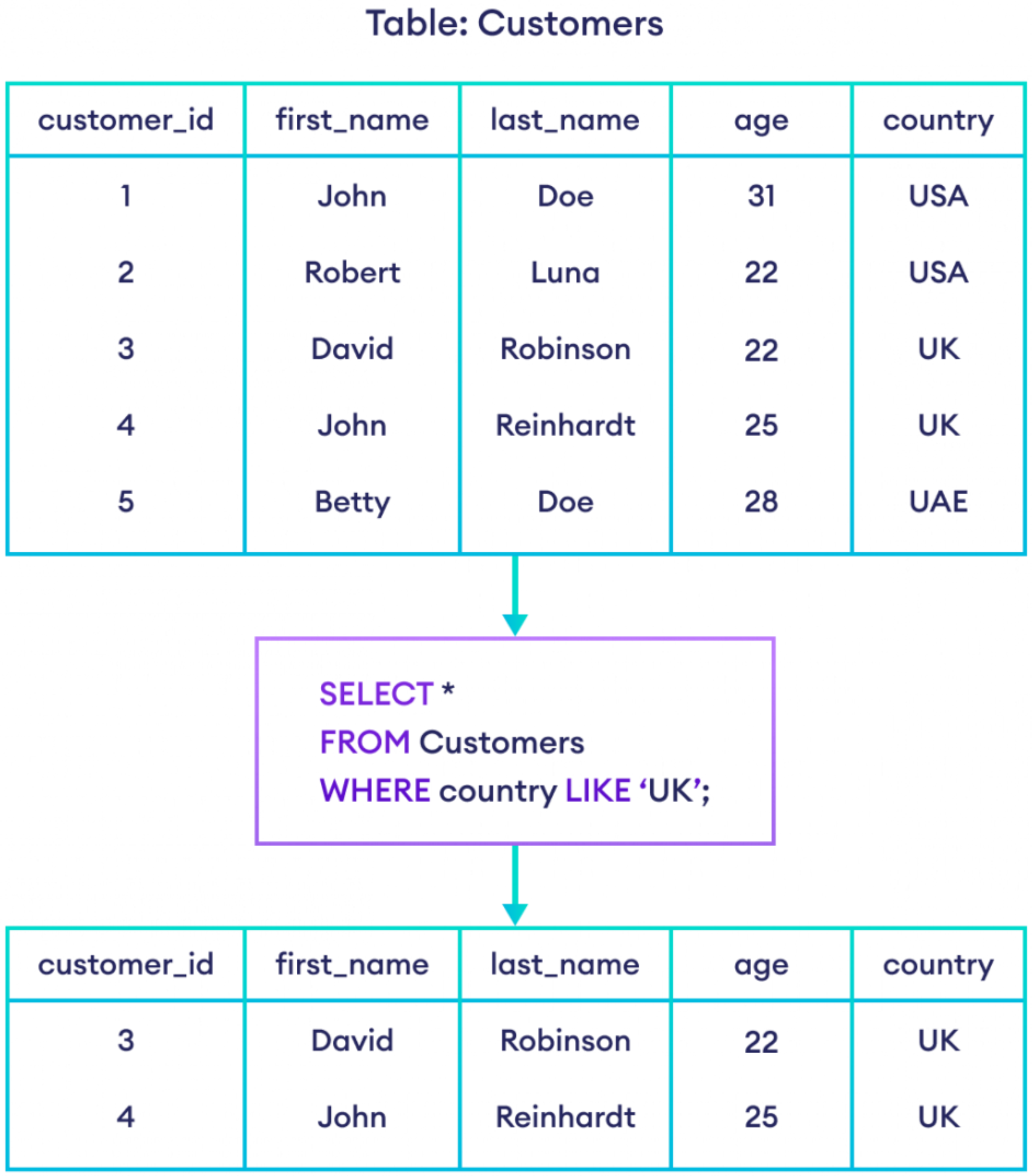


Table: Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

```
SELECT *
FROM Customers
WHERE country LIKE 'UK';
```

customer_id	first_name	last_name	age	country
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK

- **SQL LIKE With the % Wildcard**
- The SQL LIKE query is often used with the % wildcard to match a pattern of a string.

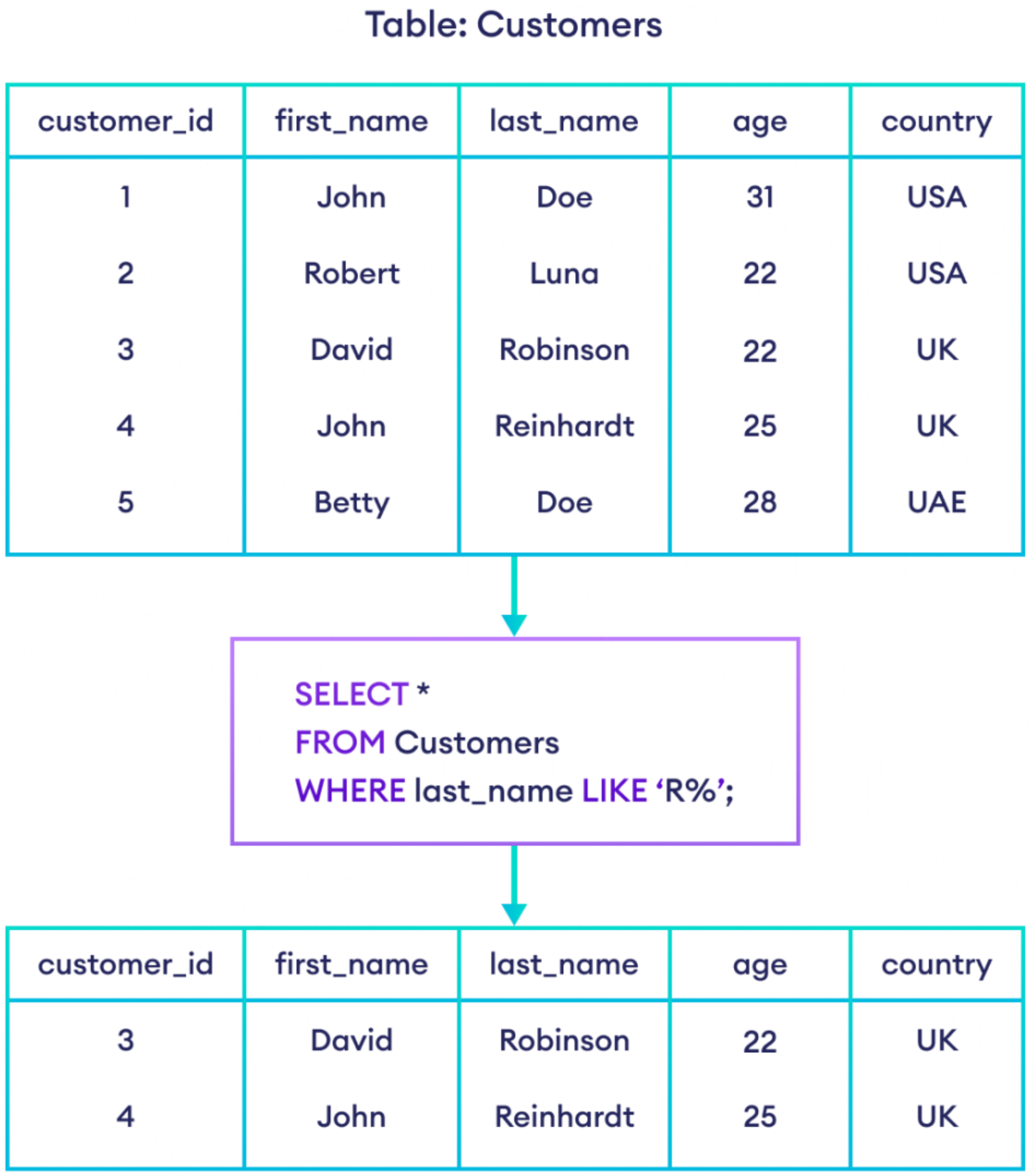


Table: Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

```
SELECT *
FROM Customers
WHERE last_name LIKE 'R%';
```

customer_id	first_name	last_name	age	country
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK

Starts With

- To return records that starts with a specific letter or phrase, add the % at the end of the letter or phrase.
- Example1
- Return all customers that starts with 'La':
- *SELECT * FROM Customers WHERE CustomerName LIKE 'La%’;*

- Example2
- Return all customers that starts with 'a' or starts with 'b':
- *SELECT * FROM Customers WHERE CustomerName LIKE 'a%' OR CustomerName LIKE 'b%';*

Ends With

- To return records that ends with a specific letter or phrase, add the % at the beginning of the letter or phrase.
- Example
- Return all customers that ends with 'a':
- *SELECT * FROM Customers WHERE CustomerName LIKE '%a';*

Contains

- To return records that contains a specific letter or phrase, add the % both before and after the letter or phrase.
- Example
- Return all customers that contains the phrase 'or'
- *SELECT * FROM Customers WHERE CustomerName LIKE '%or%';*

Example

- Return all customers that starts with "b" and ends with "s":

- *SELECT * FROM Customers WHERE CustomerName LIKE 'b%s';*

Combine Wildcards

- Any wildcard, like % and _ , can be used in combination with other wildcards.
- Example
- Return all customers that starts with "a" and are at least 3 characters in length:

- *SELECT * FROM Customers WHERE CustomerName LIKE 'a__%';*

DISTINCT Statement

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

  FROM table_name;
- **Example**
- Select all the different countries from the "Customers" table:
- *SELECT DISTINCT Country FROM Customers;*

ORDER BY

- The ORDER BY clause in SQL is a powerful feature used to sort query results in either ascending or descending order based on one or more columns.
- The ORDER BY command sorts the result set in ascending order by default. To sort the records in descending order, use the DESC keyword.
- The DESC command is used to sort the data returned in descending order.

Syntax:

- *SELECT column1, column2, ...*

  FROM table_name

  ORDER BY column1, column2, ... ASC|DESC;

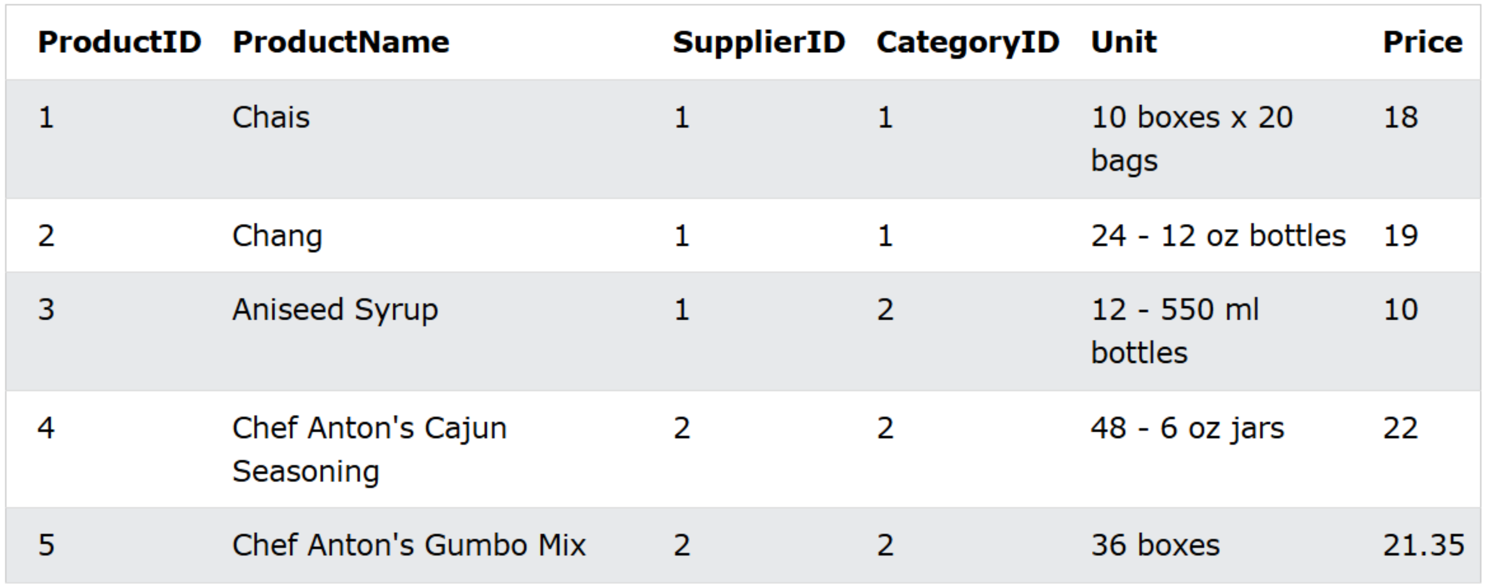
Example:

1.Sort the products by price:

```
SELECT * FROM Products
ORDER BY Price;
```

2.Sort the products from highest to lowest price:

```
SELECT * FROM Products
ORDER BY Price DESC;
```



ProductID	ProductName	SupplierID	CategoryID	Unit	Price
1	Chais	1	1	10 boxes x 20 bags	18
2	Chang	1	1	24 - 12 oz bottles	19
3	Aniseed Syrup	1	2	12 - 550 ml bottles	10
4	Chef Anton's Cajun Seasoning	2	2	48 - 6 oz jars	22
5	Chef Anton's Gumbo Mix	2	2	36 boxes	21.35

Order Alphabetically

- For string values the ORDER BY keyword will order alphabetically:

Example

1. Sort the products alphabetically by ProductName:

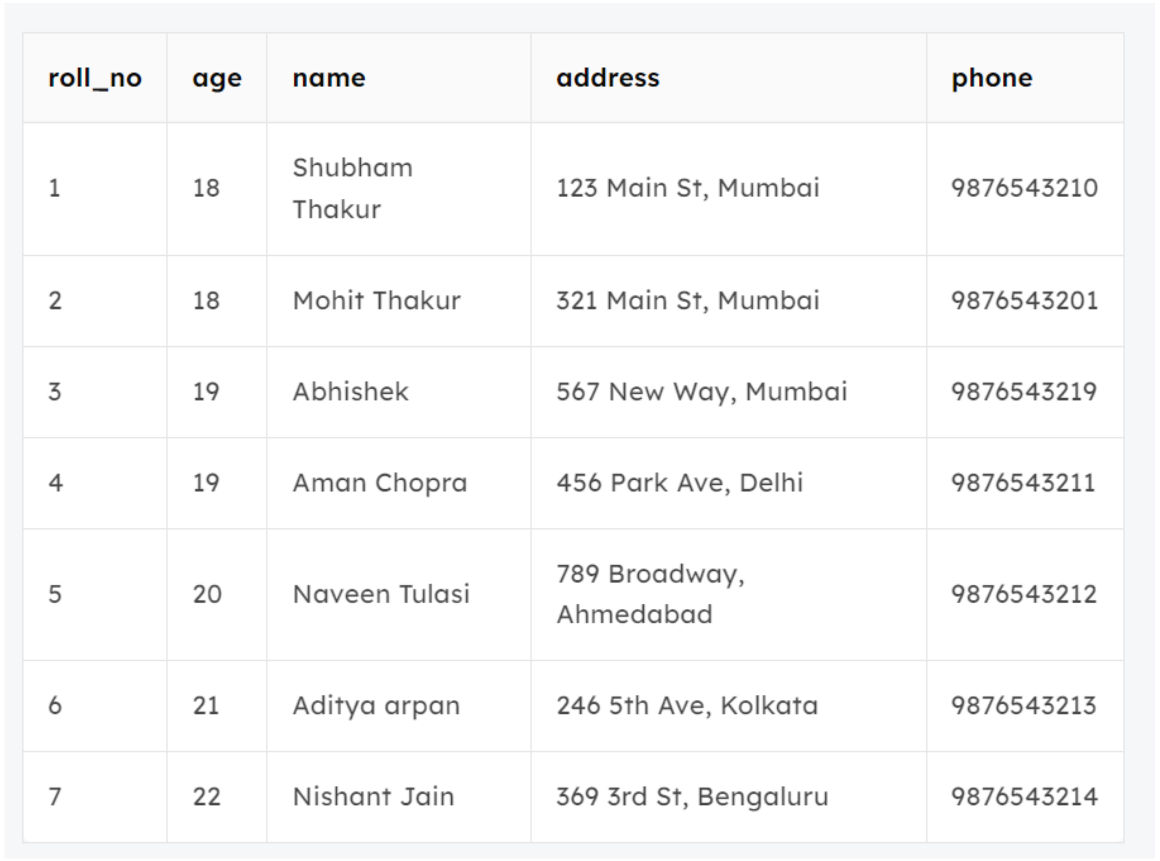
*SELECT * FROM Products*

ORDER BY ProductName;

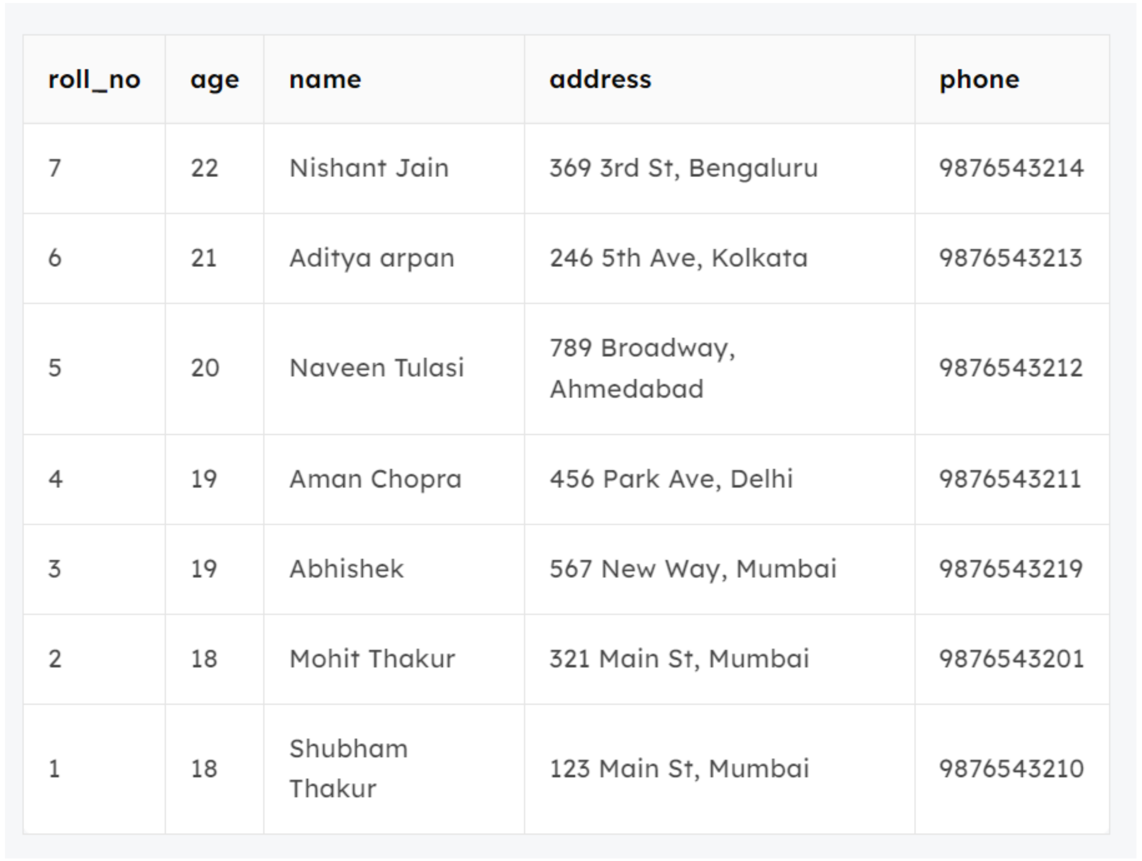
2. Sort the products by ProductName in reverse order:

*SELECT * FROM Products*

ORDER BY ProductName DESC;



roll_no	age	name	address	phone
1	18	Shubham Thakur	123 Main St, Mumbai	9876543210
2	18	Mohit Thakur	321 Main St, Mumbai	9876543201
3	19	Abhishek	567 New Way, Mumbai	9876543219
4	19	Aman Chopra	456 Park Ave, Delhi	9876543211
5	20	Naveen Tulasi	789 Broadway, Ahmedabad	9876543212
6	21	Aditya arpan	246 5th Ave, Kolkata	9876543213
7	22	Nishant Jain	369 3rd St, Bengaluru	9876543214

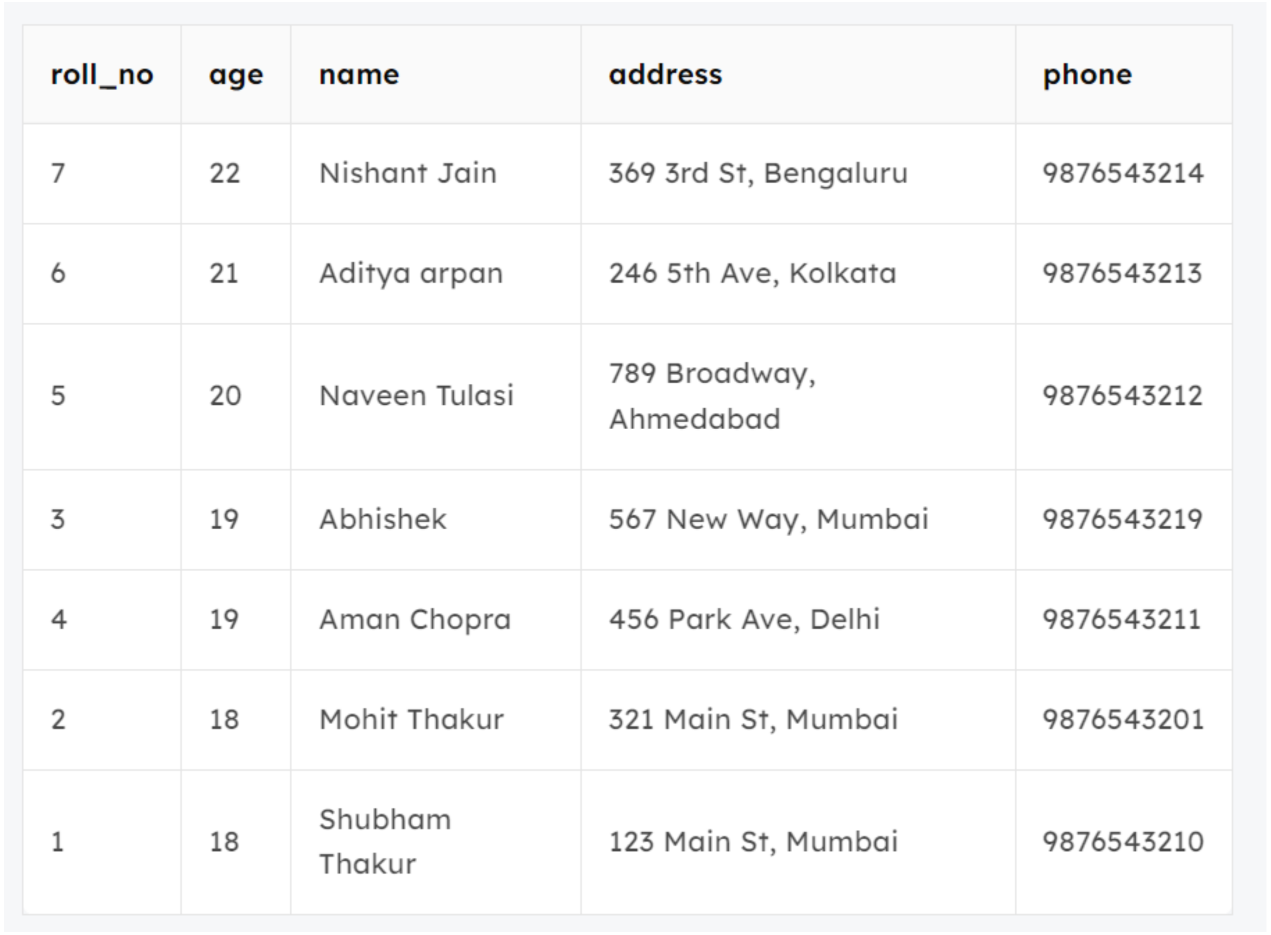


roll_no	age	name	address	phone
7	22	Nishant Jain	369 3rd St, Bengaluru	9876543214
6	21	Aditya arpan	246 5th Ave, Kolkata	9876543213
5	20	Naveen Tulasi	789 Broadway, Ahmedabad	9876543212
4	19	Aman Chopra	456 Park Ave, Delhi	9876543211
3	19	Abhishek	567 New Way, Mumbai	9876543219
2	18	Mohit Thakur	321 Main St, Mumbai	9876543201
1	18	Shubham Thakur	123 Main St, Mumbai	9876543210

Query:

```
SELECT * FROM students ORDER BY ROLL_NO DESC;
```

- SELECT * FROM students ORDER BY age DESC , name ASC;
- Output:



roll_no	age	name	address	phone
7	22	Nishant Jain	369 3rd St, Bengaluru	9876543214
6	21	Aditya arpan	246 5th Ave, Kolkata	9876543213
5	20	Naveen Tulasi	789 Broadway, Ahmedabad	9876543212
3	19	Abhishek	567 New Way, Mumbai	9876543219
4	19	Aman Chopra	456 Park Ave, Delhi	9876543211
2	18	Mohit Thakur	321 Main St, Mumbai	9876543201
1	18	Shubham Thakur	123 Main St, Mumbai	9876543210

In the above output, we can see that first the result is sorted in descending order according to Age. There are multiple rows of having the same Age. Now, sorting further this result-set according to name will sort the rows with the same Age according to name in ascending order.

DDL (Data Definition Language)

The DDL commands in SQL are:

- CREATE: To add a new object to the database.
- ALTER: To change the structure of the database.
- DROP: To remove an existing object from the database. Check out our guide on how to delete a column in SQL.
- TRUNCATE: To remove all records from a table, including the space allocated to store this data.

ALTER command in SQL

- The ALTER TABLE statement in SQL is used to modify an existing table structure in a database without losing any data.
- It allows you to add, remove, or modify columns, change data types, or apply constraints to improve data integrity and ensure that the table meets evolving business requirements.
- It allows for structural changes like adding new columns, modifying existing ones, deleting columns, and renaming columns within a table.
- Syntax:
- *ALTER TABLE table_name*

  [ADD | DROP | MODIFY] column_name datatype;

1. ADD – To add a new column to the table

- The ADD clause is used to add a new column to an existing table. You must specify the name of the new column and its data type.
- Query:
- *ALTER TABLE table_name*

  ADD column_name datatype;

2. MODIFY/ALTER – To change the data type of an existing column

- The MODIFY (or ALTER COLUMN in some databases like SQL Server) clause is used to modify the definition of an existing column, such as changing its data type or size.
- Query:
- *ALTER TABLE table_name*

  MODIFY COLUMN column_name datatype;

3. DROP – To delete an existing column from the table

- The DROP clause allows you to remove a column from a table. Be cautious when using this command as it will permanently remove the column and its data.
- Query:
- *ALTER TABLE table_name*

  DROP COLUMN column_name;

4. RENAME COLUMN – To rename an existing column

- We can rename an existing column using the RENAME COLUMN clause. This allows you to change the name of a column while preserving its data type and content.
- Query:
- *ALTER TABLE table_name*

  RENAME COLUMN old_name TO new_name;

5. RENAME TO – To rename the table itself

- We can rename an entire table using the RENAME TO clause. This changes the name of the table while preserving its structure and data.
- *ALTER TABLE table_name*

  RENAME TO new_table_name;

DROP and TRUNCATE in SQL

- The DROP and TRUNCATE commands in SQL are used to remove data from a table, but they work differently.
- In SQL, the DROP command is used to permanently remove an object from a database, such as a table, database, index, or view. When we DROP a table, both the data and the structure of the object are permanently removed from the database leaving no trace of the object.

1. DROP Table

- To delete an entire table, including its data and structure:
- Syntax:
- *DROP TABLE table_name;*

- TRUNCATE eemoves all rows but preserves the table structure. It deletes only the data, not the table structure.
- Is a faster operation compared to the DROP statement.
- *TRUNCATE TABLE table_name;*

Data Manipulation Language (DML) commands

Commands for inserting, deleting, updating, and merging data in Snowflake tables:

- INSERT
- MERGE
- UPDATE
- DELETE
- TRUNCATE TABLE

SQL INSERT INTO Statement

- The INSERT INTO statement in SQL is used to add new rows of data to a table in a database.
- There are two main ways to use the INSERT INTO statement by specifying the columns and values explicitly or by inserting values for all columns without specifying them.
- There are two primary syntaxes of INSERT INTO statements depending on the requirements. The two syntaxes are:
- Syntax 1. Only Values
- The first method is to specify only the value of data to be inserted without the column names.
- *INSERT INTO table_name*

  VALUES (value1, value2, value);

- Syntax 2. Column Names And Values Both
- In the second method we will specify both the columns which we want to fill and their corresponding values as shown below:
- *INSERT INTO table_name (column1, column2, column3)*

  VALUES (value1, value2, value);

UPDATE Statement

- The UPDATE statement in SQL is used to update the data of an existing table in the database.
- We can update single columns as well as multiple columns using the UPDATE statement as per our requirement.
- Syntax:
- The syntax for SQL UPDATE Statement is :
- *UPDATE table_name SET column1 = value1, column2 = value2,…*

  WHERE condition;

Customer

CustomerID	CustomerName	LastName	Country	Age	Phone
1	Shubham	Thakur	India	23	xxxxxxxxxx
2	Aman	Chopra	Australia	21	xxxxxxxxxx
3	Naveen	Tulasi	Sri lanka	24	xxxxxxxxxx
4	Aditya	Arpan	Austria	21	xxxxxxxxxx
5	Nishant. Salchichas S.A.	Jain	Spain	22	xxxxxxxxxx

Query:

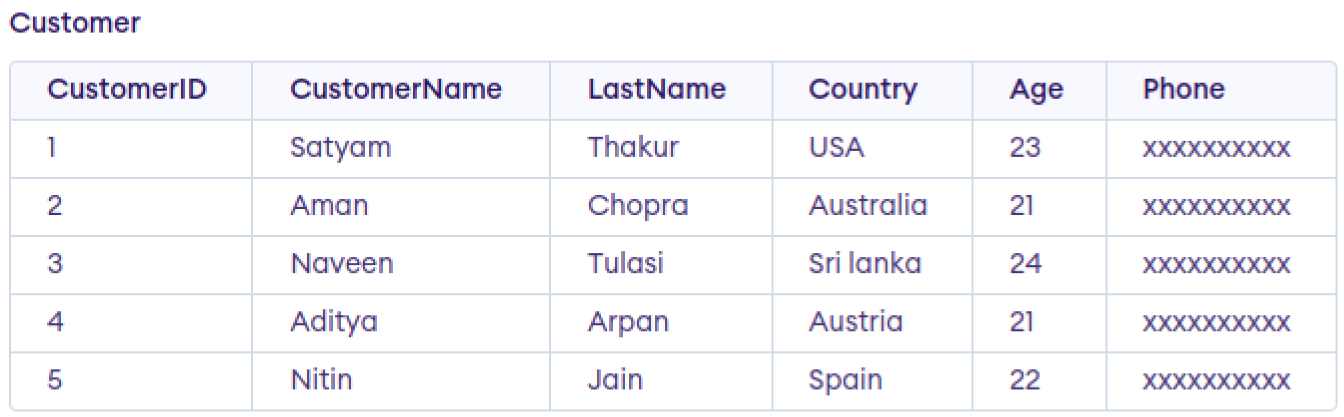
UPDATE Customer
SET CustomerName = 'Nitin'
WHERE Age = 22;

Customer

CustomerID	CustomerName	LastName	Country	Age	Phone
1	Shubham	Thakur	India	23	xxxxxxxxxx
2	Aman	Chopra	Australia	21	xxxxxxxxxx
3	Naveen	Tulasi	Sri lanka	24	xxxxxxxxxx
4	Aditya	Arpan	Austria	21	xxxxxxxxxx
5	Nitin	Jain	Spain	22	xxxxxxxxxx

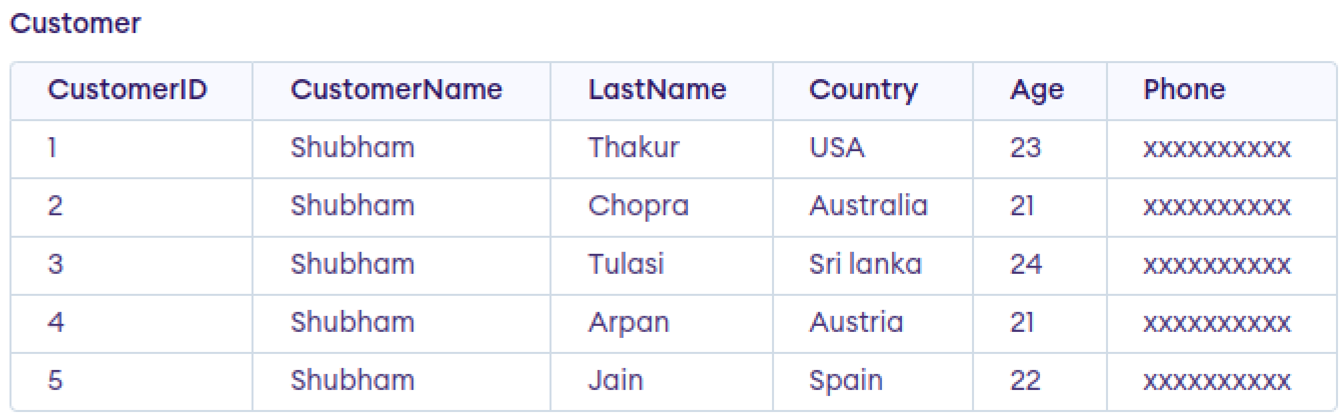
Omitting WHERE Clause in UPDATE Statement

- If we omit the WHERE clause from the update query then all of the rows will get updated.
- Query:
- *UPDATE Customer SET CustomerName = 'Shubham';*



Customer

CustomerID	CustomerName	LastName	Country	Age	Phone
1	Satyam	Thakur	USA	23	xxxxxxxxxx
2	Aman	Chopra	Australia	21	xxxxxxxxxx
3	Naveen	Tulasi	Sri lanka	24	xxxxxxxxxx
4	Aditya	Arpan	Austria	21	xxxxxxxxxx
5	Nitin	Jain	Spain	22	xxxxxxxxxx



Customer

CustomerID	CustomerName	LastName	Country	Age	Phone
1	Shubham	Thakur	USA	23	xxxxxxxxxx
2	Shubham	Chopra	Australia	21	xxxxxxxxxx
3	Shubham	Tulasi	Sri lanka	24	xxxxxxxxxx
4	Shubham	Arpan	Austria	21	xxxxxxxxxx
5	Shubham	Jain	Spain	22	xxxxxxxxxx

DELETE Statement

- The SQL DELETE statement removes one or more rows from a database table based on a condition specified in the WHERE clause.
- Syntax:
- *DELETE FROM table_name*

  WHERE some_condition;

Aggregate functions

- SQL Aggregate Functions are used to perform calculations on a set of rows and return a single value. They are used to perform mathematical calculations on data. They usually return a single value as output.
- They are often used with the GROUP BY clause in SQL to summarize data for each group.
- The most commonly used SQL aggregate functions are:
  - MIN() - returns the smallest value within the selected column
  - MAX() - returns the largest value within the selected column
  - COUNT() - returns the number of rows in a set
  - SUM() - returns the total sum of a numerical column
  - AVG() - returns the average value of a numerical column

ProductID	ProductName	SupplierID	CategoryID	Unit	Price
1	Chais	1	1	10 boxes x 20 bags	18
2	Chang	1	1	24 - 12 oz bottles	19
3	Aniseed Syrup	1	2	12 - 550 ml bottles	10
4	Chef Anton's Cajun Seasoning	2	2	48 - 6 oz jars	22
5	Chef Anton's Gumbo Mix	2	2	36 boxes	21.35

- **Syntax**
- *SELECT **MIN**(column_name)*
  FROM table_name
  WHERE condition;

- *SELECT **MAX**(column_name)*
  FROM table_name
  WHERE condition;

MIN Example:

- Find the lowest price in the Price column:
- *SELECT MIN(Price)*
  FROM Products;

MAX Example:

- *Find the highest price in the Price column:*
- *SELECT MAX(Price)*
  FROM Products;

- **Syntax**
- *SELECT* ***COUNT****(column_name)*

  FROM table_name

  WHERE condition;

- **Example**
- Find the number of products where Price is higher than 20:
- *SELECT COUNT(ProductID)*

  FROM Products

  WHERE Price > 20;

- **Example**
- Find the total number of rows in the Products table:
- *SELECT COUNT(*)*

  FROM Products;

- **Example**
- Find the number of products where the ProductName is not null:
- *SELECT COUNT(ProductName)*

  FROM Products;

- **Syntax**
- *SELECT **SUM**(column_name)*

  FROM table_name

  WHERE condition;

- **Example**
- Return the sum of the Quantity field for the product with ProductID 11:
- *SELECT SUM(Quantity)*

  FROM OrderDetails

  WHERE ProductId = 11;

- **Example**
- Return the sum of all Quantity fields in the OrderDetails table:
- *SELECT SUM(Quantity)*

  FROM OrderDetails;

- **Example**
- Use an expression inside the SUM() function:
- *SELECT SUM(Quantity * 10)*

  FROM OrderDetails;

- **Syntax**
- *SELECT* ***AVG****(column_name)*

  FROM table_name

  WHERE condition;

- **Example**
- Find the average price of all products:
- *SELECT AVG(Price)*

  FROM Products;

- **Example**
- Return the average price of products in category 1:
- *SELECT AVG(Price)*

  FROM Products

  WHERE CategoryID = 1;

- **Example**
- Return all products with a higher price than the average price:
- *SELECT * FROM Products*

  WHERE price > (SELECT AVG(price)
  FROM Products);

GROUP BY Clause

- The GROUP BY statement in SQL is used for organizing and summarizing data based on identical values in specified columns.
- By using the GROUP BY clause, users can apply aggregate functions like SUM, COUNT, AVG, MIN, and MAX to each group, making it easier to perform detailed data analysis.
- The GROUP BY statement in SQL is used to arrange identical data into groups based on specified columns.
- If a particular column has the same values in multiple rows, the GROUP BY clause will group these rows together.
- **Key Points About GROUP BY:**
- GROUP BY clause is used with the SELECT statement.
- In the query, the GROUP BY clause is placed after the WHERE clause.
- In the query, the GROUP BY clause is placed before the ORDER BY clause if used.
- In the query, the Group BY clause is placed before the Having clause.
- Place condition in the having clause.

- **Syntax:**
- *SELECT column1, function_name(column2)*

  FROM table_name

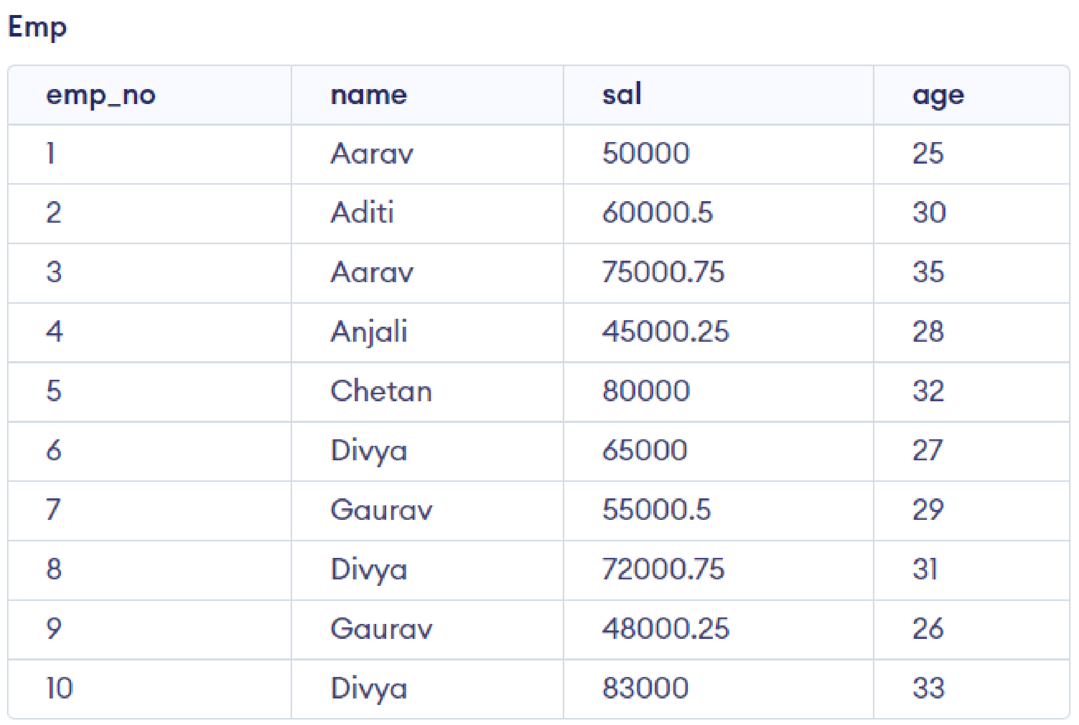
  GROUP BY column1, column2

Example 1 : Group By Single Column

Query:

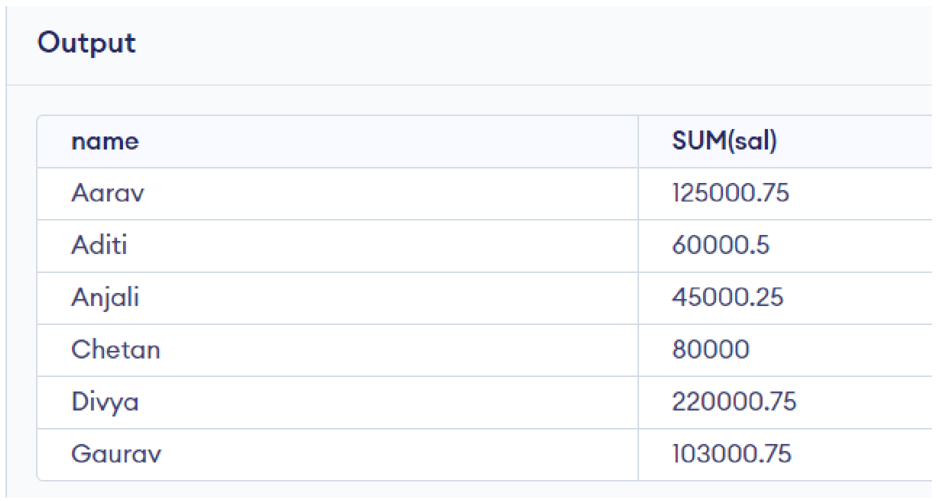
SELECT name, SUM(sal) FROM emp

GROUP BY name;



Emp

emp_no	name	sal	age
1	Aarav	50000	25
2	Aditi	60000.5	30
3	Aarav	75000.75	35
4	Anjali	45000.25	28
5	Chetan	80000	32
6	Divya	65000	27
7	Gaurav	55000.5	29
8	Divya	72000.75	31
9	Gaurav	48000.25	26
10	Divya	83000	33



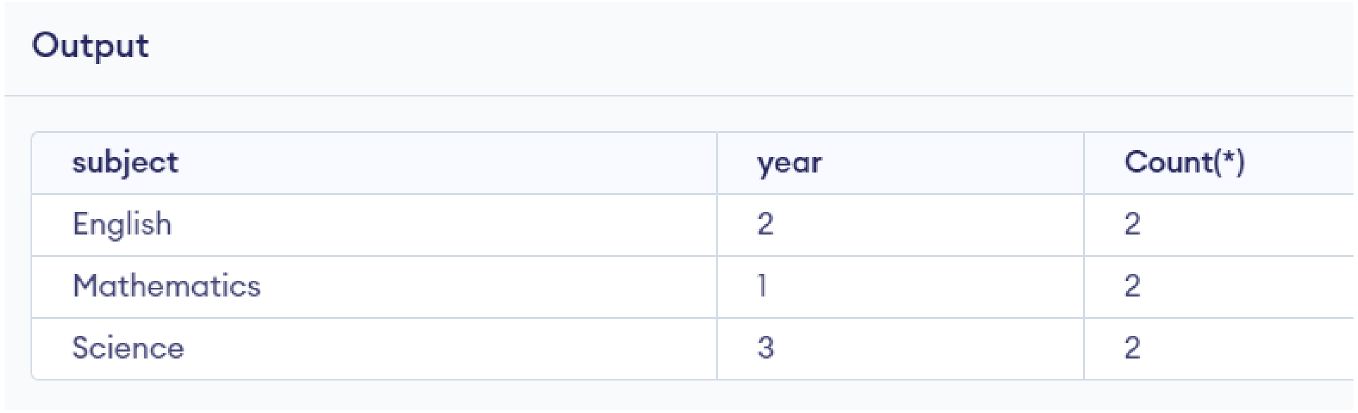
Output

name	SUM(sal)
Aarav	125000.75
Aditi	60000.5
Anjali	45000.25
Chetan	80000
Divya	220000.75
Gaurav	103000.75

- Example 2 : Group By Multiple Columns
- Group by multiple columns is say, for example, GROUP BY column1, column2. This means placing all the rows with the same values of columns column 1 and column 2 in one group. Consider the below query:
- **Query:**
- *SELECT SUBJECT, YEAR, Count(*)*

  FROM Student

  GROUP BY SUBJECT, YEAR;


Output

subject	year	Count(*)
English	2	2
Mathematics	1	2
Science	3	2

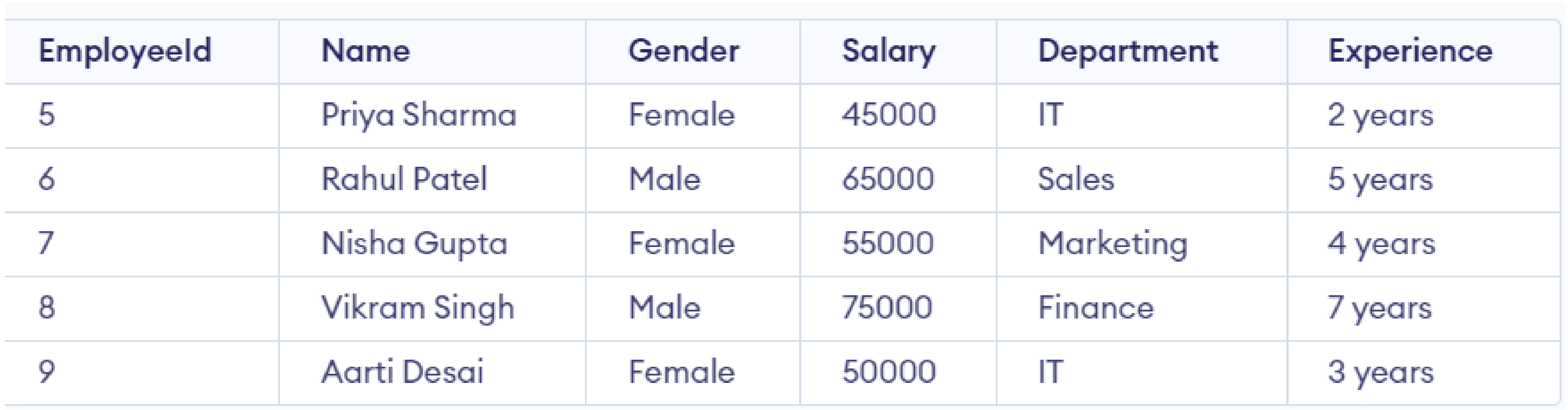
As we can see in the above output the students with both the same SUBJECT and YEAR are placed in the same group. And those whose only SUBJECT is the same but not YEAR belong to different groups. So here we have grouped the table according to two columns or more than one column. The Grouped subject and years are (English,2) , (Mathematics,1) and (Science,3). The above-mentioned all groups and years are repeated twice.

SQL HAVING Clause

- The HAVING clause in SQL is used to filter query results based on aggregate functions. Unlike the WHERE clause, which filters individual rows before grouping, the HAVING clause filters groups of data after aggregation.
- The HAVING clause is used to filter the result of the GROUP BY statement based on the specified conditions.
- It allows filtering grouped data using Boolean conditions (AND, OR).
- It was introduced because the WHERE clause cannot be used with aggregate functions.

- **Syntax:**
- *SELECT column_name, AGGREGATE_FUNCTION(column_name)*

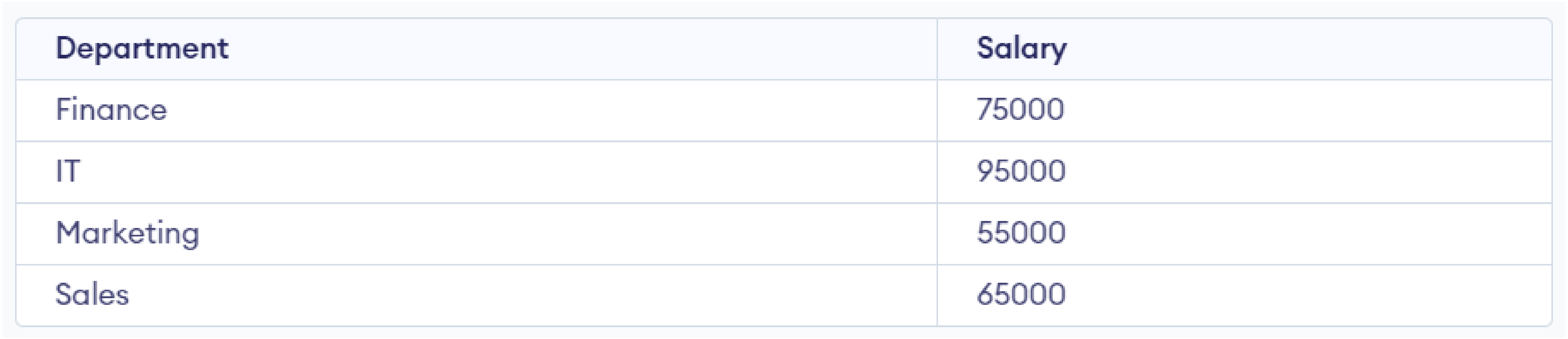
```
SELECT column_name, AGGREGATE_FUNCTION(column_name)
FROM table_name
GROUP BY column_name
HAVING condition;
```



EmployeeId	Name	Gender	Salary	Department	Experience
5	Priya Sharma	Female	45000	IT	2 years
6	Rahul Patel	Male	65000	Sales	5 years
7	Nisha Gupta	Female	55000	Marketing	4 years
8	Vikram Singh	Male	75000	Finance	7 years
9	Aarti Desai	Female	50000	IT	3 years

Query:

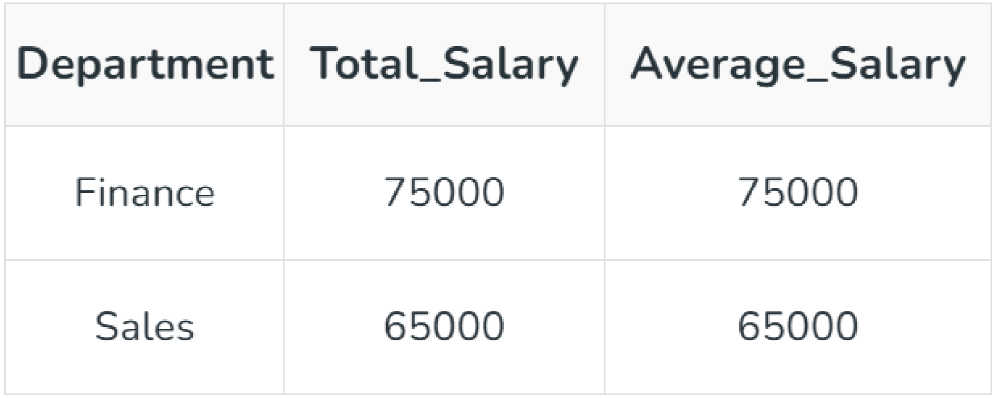
```
SELECT Department, sum(Salary) as Salary
FROM Employee
GROUP BY department;
```



Department	Salary
Finance	75000
IT	95000
Marketing	55000
Sales	65000

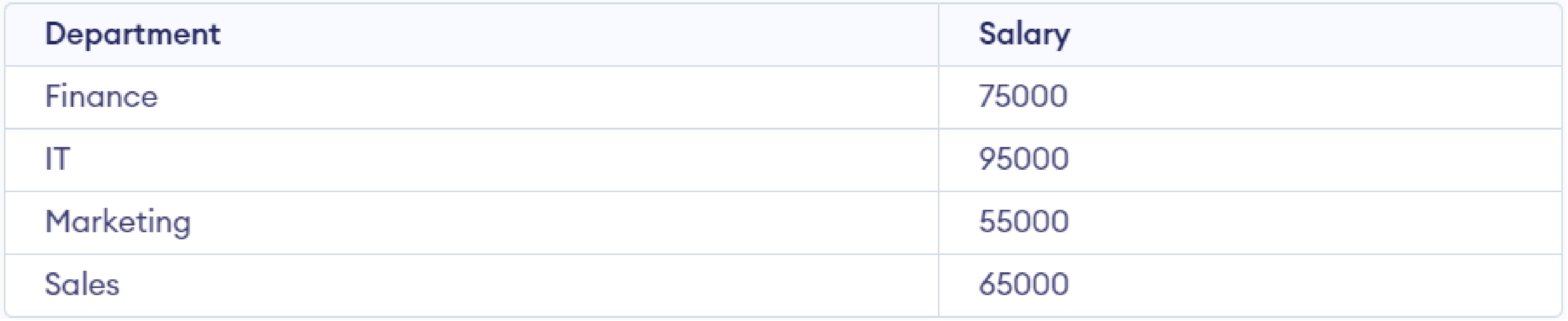
- **Example 2:** Using HAVING with Multiple Conditions
- If we want to find the departments where the total salary is greater than or equal to $50,000, and the average salary is greater than $55,000. We can use the HAVING clause to apply both conditions.
- **Query**

-
  ```
  SELECT Department, SUM(Salary) AS Total_Salary, AVG(Salary) AS
  Average_Salary  FROM Employee
  GROUP BY Department
   HAVING SUM(Salary) >= 50000 AND AVG(Salary) > 55000;
  ```



Department	Total_Salary	Average_Salary
Finance	75000	75000
Sales	65000	65000

- **Example:** Now if we need to display the departments where the sum of salaries is 50,000 or more. In this condition, we will use the HAVING Clause.
- **Query:**
-
```
SELECT Department, sum(Salary) as Salary
FROM Employee
GROUP BY department
HAVING SUM(Salary) >= 50000;
```



Department	Salary
Finance	75000
IT	95000
Marketing	55000
Sales	65000

Scalar Functions

- Scalar functions are also used to return a single value after performing data operation.
- Scalar functions are used to perform operations on individual values in a column and return a single value for each row in the result set.
- These functions can be used to manipulate strings, numbers, and dates, among other data types.
- The most common Scalar Functions are:
- UCASE()   ROUND()
- LCASE()   NOW()
- MID()   FORMAT()
- LENGTH()

1. UCASE()

- The UCASE() or the upper case function changes the case of the string to uppercase. An individual string or a column can be passed as the parameter to the function.
- **Syntax:**
- *SELECT UCASE(string);*

OR

- *SELECT UCASE(column_name) FROM table_name;*
- **Example:**
- *SELECT UCASE('tree');*
- Output: TREE

- Using UCASE(), the case of the string has been changed to uppercase.
- tree -> TREE

2. LCASE()

- The LCASE() or the lowercase function changes the case of the string to lowercase. Like UCASE(), any number of strings can be passed as a parameter to the LCASE() function.
- **Syntax:**
- *SELECT LCASE(string);*

  OR
- *SELECT LCASE(column_name) FROM table_name;*
- **Example:**
- *SELECT LCASE('HeLLOWorld');*
- Output: helloworld
- Explaination: Here, the case of the letters is changed to lowercase using the LCASE() function.
- HeLLOWorld -> helloworld

3. MID()

- The MID() function is used to extract substrings from the string passed as a parameter or from the table's column, which is of string datatype.
- **Syntax:**
- *SELECT MID(string, start, length);*

OR

- *SELECT MID(column_name, start, length) FROM table_name;*
- Where, string/column_name is full of the string from which the substrings need to be extracted. start refers to the integer value that denotes the start position, starting from 1. length is the integer value that denotes the length of the substring to be extracted from the main string starting from start. Length is an optional parameter. If not stated, the whole string after the start position will be extracted as result.

- **Example:**
- *SELECT MID('Worlds of Wonder', 11, 6);*
- **Output:** *Wonder*

- **Explanation:** In the above example, a substring has been extracted from the string.
- start = 11, means the extraction would start from 11th position,i.e., W length = 6, 6 letters would be extracted out.
- As a result, 'Wonder' is returned.

4. LENGTH()

- The LENGTH() function returns the length of the string passed as a parameter.
- **Syntax:**
- *SELECT LENGTH(string);*

OR

- *SELECT LENGTH(column_name) FROM table_name;*
- **Example:**
- SELECT LENGTH('sunshine');
- **Output:** 8
- using LENGTH(), the length of the string provided is returned. Here, the string is 'sunshine', 8 letter word. Thereby, 8 is returned.

5. ROUND()

- The ROUND() function is used to round off the numeric values to the specified decimal value.
- **Syntax:**
- *SELECT ROUND(numeric_value, Decimal);*

OR

- *SELECT ROUND(column_name, Decimal) FROM table_name;*
- Where Decimal is the number of decimals that need to be fetched.
- **Example:**
- SELECT ROUND(3245.8762, 3);
- **Output:** 3245.876
- The ROUND() function has rounded off the value given to 3 decimal places.

6. NOW()

- The NOW() function is used to get the current system's date and time.
- **Syntax:**
- *SELECT NOW();*

  OR
- *SELECT NOW() FROM table_name;*
- **Example:**
- *SELECT NOW();*
- **Output:** 2022-02-12 21:41:48
- Here, the NOW() function has simply returned the current date and time.

7. FORMAT()

- The FORMAT() function is used to modify the display of the column values. There can be different formats that we can use.
- **Syntax:**
- SELECT FORMAT(value/string, FormatRequired);

  OR
- SELECT FORMAT(column_name, FormatRequired) FROM table_name;
- Where, FormatRequired is the format in which the value/string will be displayed.
- **Example:**
- SELECT FORMAT(137/3, 2);
- **Output:** 45.67
- FORMAT() function has returned the value rounded off 2 decimal places after dividing by 3.

String functions

- SQL String Functions are powerful tools that allow us to manipulate, format, and extract specific parts of text data in our database. These functions are essential for tasks like cleaning up data, comparing strings, and combining text fields.
- **Common SQL String Functions**
- String functions are used to perform an operation on input string and return an output string. Following are the string functions defined in SQL:

1. CONCAT(): Concatenate Strings

- The CONCAT() function is used to concatenate (combine) two or more strings into one string. It is useful when we want to merge fields like first and last names into a full name.
- Query:
- SELECT CONCAT('John', ' ', 'Doe') AS FullName;
- Output: John Doe

2. CHAR_LENGTH() / CHARACTER_LENGTH(): Find String Length

- The CHAR_LENGTH() or LENGTH() function returns the length of a string in characters. It’s essential for validating or manipulating text data, especially when you need to know how many characters a string contains.
- Query:
- *SELECT CHAR_LENGTH('Hello') AS StringLength;*
- Output: 5

3. UPPER() and LOWER(): Convert Text Case

- These functions convert the text to uppercase or lowercase, respectively. They are useful for normalizing the case of text in a database.
- Query:
- *SELECT UPPER('hello') AS UpperCase;*
- *SELECT LOWER('HELLO') AS LowerCase;*
- Output:

```
HELLO
hello
```

4. LENGTH(): Length of String in Bytes

- LENGTH() returns the length of a string in bytes. This can be useful for working with multi-byte character sets.

- Query:
- *SELECT LENGTH('Hello') AS LengthInBytes;*
- Output: 5

5. REPLACE(): Replace Substring in String

- The REPLACE() function replaces occurrences of a substring within a string with another substring. This is useful for cleaning up data, such as replacing invalid characters or formatting errors.
- Query:
- SELECT REPLACE('Hello World', 'World', 'SQL') AS UpdatedString;
- Output: Hello SQL

6. SUBSTRING() / SUBSTR(): Extract Part of a String

- The SUBSTRING() (or SUBSTR()) function is used to extract a substring from a string, starting from a specified position. It is especially useful when we need to extract a specific part of a string, like extracting the domain from an email address.
- Query:
- *SELECT SUBSTRING('Hello World', 1, 5) AS SubStringExample;*
- Output: Hello

7. LEFT() and RIGHT(): Extract Substring from Left or Right

- The LEFT() and RIGHT() functions allow you to extract a specified number of characters from the left or right side of a string, respectively. It is used for truncating strings for display.
- Query:
- *SELECT LEFT('Hello World', 5) AS LeftString;*
- *SELECT RIGHT('Hello World', 5) AS RightString;*
- Output:

```
Hello
World
```

8. INSTR(): Find Position of Substring

- The INSTR() function is used to find the position of the first occurrence of a substring within a string. It returns the position (1-based index) of the substring. If the substring is not found, it returns 0. This function is particularly useful for locating specific characters or substrings in text data.
- Query:
- *SELECT INSTR('Hello World', 'World') AS SubstringPosition;*
- Output: 7

10. REVERSE(): Reverse the String

- The REVERSE() function reverses the characters in a string. It’s useful in situations where we need to process data backward, such as for password validation or certain pattern matching.
- Query:
- *SELECT REVERSE('Hello') AS ReversedString;*
- Output: olleH

Numeric Functions

- SQL Numeric Functions are essential tools for performing mathematical and arithmetic operations on numeric data. These functions allow you to manipulate numbers, perform calculations, and aggregate data for reporting and analysis purposes.
- 1. ABS() – Absolute Value
- The ABS() function returns the absolute value of a number, which is the number without its sign (i.e., it converts negative numbers to positive).
- Syntax:
- *SELECT ABS(number);*
- Example:
- *SELECT ABS(-25);*
- Output: 25

2. CEIL() or CEILING() – Round Number Up

- The CEIL() (or CEILING()) function rounds a number up to the nearest integer, regardless of whether the decimal part is greater than or less than 0.5.
- Syntax:
- *SELECT CEIL(number);*
- Example:
- *SELECT CEIL(12.34)*;
- Output: 13

3. FLOOR() – Round Number Down

- The FLOOR() function rounds a number down to the nearest integer, ignoring the decimal part.
- Syntax:
- *SELECT FLOOR(number);*
- Example:
- *SELECT FLOOR(12.98);*
- Output: 12

4. ROUND() – Round a Number to a Specified Decimal Place

- The ROUND() function rounds a number to a specified number of decimal places. It is very useful for financial calculations or whenever precise rounding is necessary.
- Syntax:
- *SELECT ROUND(number, decimal_places);*
- Example:
- *SELECT ROUND(15.6789, 2);*
- Output: 15.68

**5. TRUNCATE(**) – Remove Decimal Places

- The TRUNCATE() function is used to remove the decimal portion of a number without rounding. It truncates the number to the specified number of decimal places.
- Syntax:
- *SELECT TRUNCATE(number, decimal_places);*
- Example:
- *SELECT TRUNCATE(12.98765, 2);*
- Output: 12.98

6. MOD() – Modulo or Remainder

- The MOD() function returns the remainder of a division operation (i.e., it computes the modulus). This function is useful for tasks like determining even/odd numbers or finding remainders in mathematical operations.
- Syntax:
- *SELECT MOD(dividend, divisor);*
- Example:
- *SELECT MOD(10, 3);*
- Output: 1

7. POWER() – Raise a Number to the Power of Another

- The POWER() function is used to raise a number to the power of another number. It is often used in mathematical calculations like compound interest or growth rate.

- Syntax:
- *SELECT POWER(base, exponent);*
- Example:
- *SELECT POWER(2, 3);*
- Output: 8

8. SQRT() – Square Root

- The SQRT() function returns the square root of a number. This is useful for mathematical calculations involving geometry or statistical analysis.

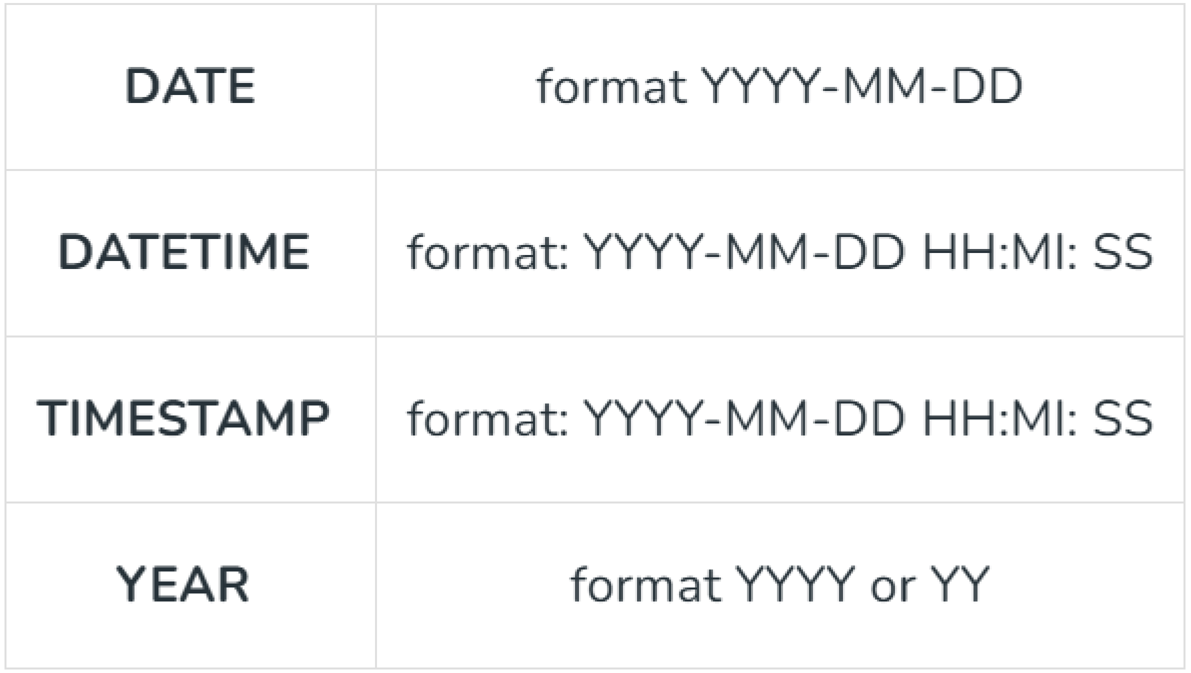
- Syntax:
- SELECT SQRT(number);
- Example:
- SELECT SQRT(16);
- Output: 4

9. LOG() – Logarithm

- The LOG() function returns the natural logarithm (base e) of a number. You can also use LOG(base, number) to calculate the logarithm of a number with a custom base.
- Syntax:
- *SELECT LOG(number);*
- *SELECT LOG(base, number);*
- Example:
- *SELECT LOG(100);*
- Output: 4.605170186

SQL Date and Time Functions

- Handling date and time data in SQL is essential for many database operations. SQL provides a variety of date and time functions that help users work with date values, perform calculations, and format them as needed.



DATE	format YYYY-MM-DD
DATETIME	format: YYYY-MM-DD HH:MI: SS
TIMESTAMP	format: YYYY-MM-DD HH:MI: SS
YEAR	format YYYY or YY

SQL Date Functions

1. NOW()

- The NOW() function retrieves the current date and time in YYYY-MM-DD HH:MI:SS format.
- Query:
- *SELECT NOW();*

2. CURDATE()

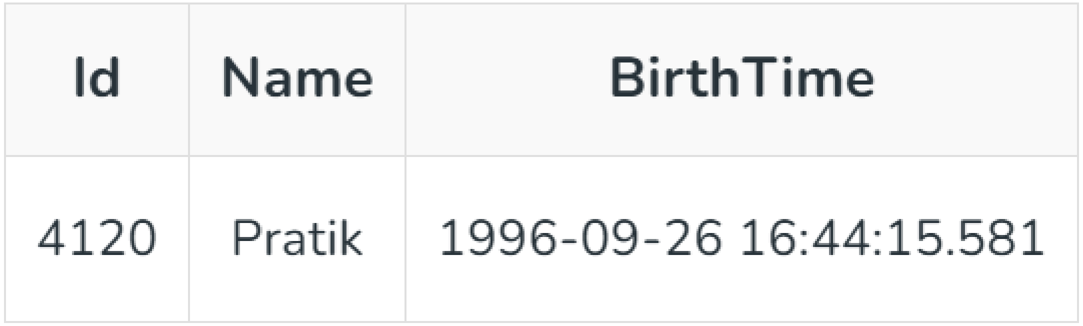
- CURDATE() returns the current date in YYYY-MM-DD format, without the time part.
- Query:
- *SELECT CURDATE();*

3. CURTIME()

- The CURTIME() function returns the current time in HH:MI:SS format, excluding the date.
- Query:
- SELECT CURTIME();

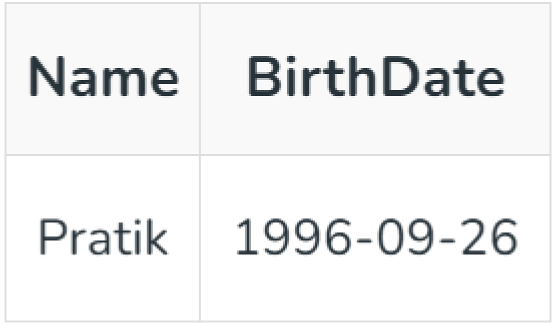
4. DATE()

- Extracts the date part of a date or date/time expression. Example: For the below table named ‘Test’



Id	Name	BirthTime
4120	Pratik	1996-09-26 16:44:15.581

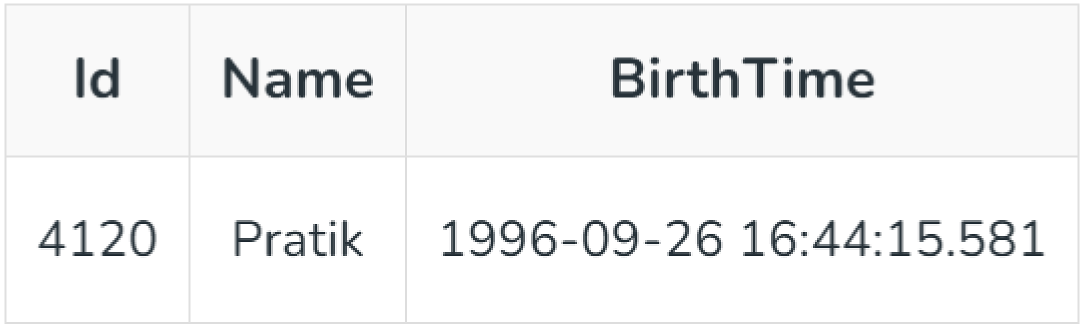
- Query:
- *SELECT Name, DATE(BirthTime) AS BirthDate FROM Test;*
- Output:



Name	BirthDate
Pratik	1996-09-26

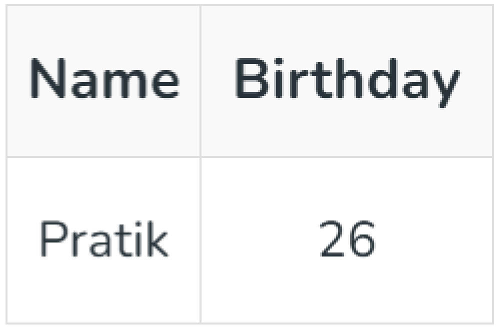
5. EXTRACT()

- The EXTRACT() function retrieves a specific part of a DATE, DATETIME, or TIMESTAMP value, such as the day, year, or second.
- Syntax:
- *EXTRACT(unit FROM date);*
- Several units can be considered but only some are used such as MICROSECOND, SECOND, MINUTE, HOUR, DAY, WEEK, MONTH, QUARTER, YEAR, etc. And ‘date’ is a valid date expression. Example: For the below table named ‘Test’.



Id	Name	BirthTime
4120	Pratik	1996-09-26 16:44:15.581

- Query:
- *SELECT Name, Extract(DAY FROM BirthTime) AS BirthDay FROM Test;*
- Output:



Name	Birthday
Pratik	26