SQL - Operators and Clauses

Operators

AND Operator

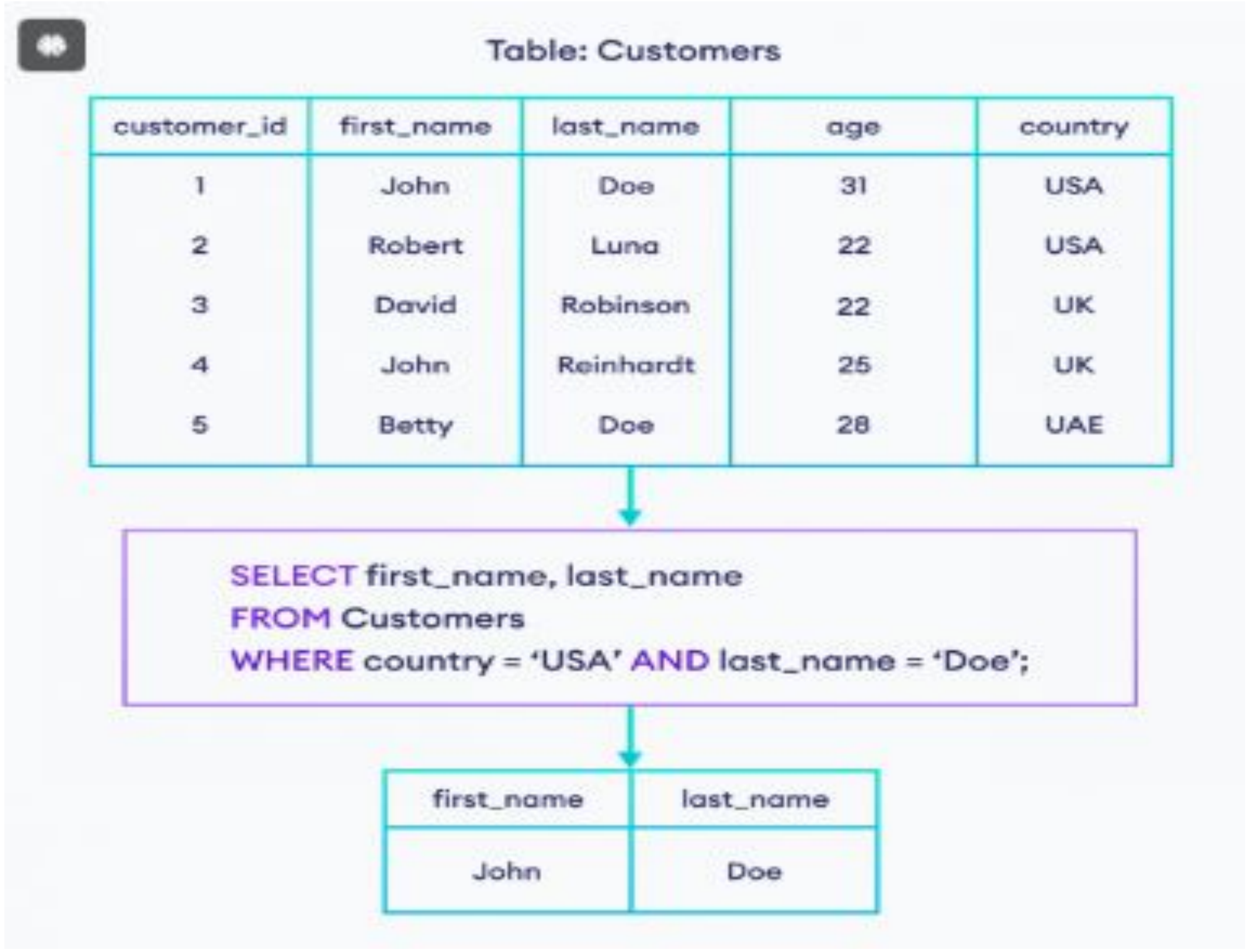


Table: Customers

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

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

first_name	last_name
John	Doe

AND Operator

It is used to display record if all the conditions are separated by AND are TRUE

Syntax:

SELECT *column1*, *column2, ...*
FROM *table_name*
WHERE *condition1* AND *condition2* AND *condition3 ...*;

Display the Details of the Sales made by Department "1" and on 13th August 2010.

```
Select * from sales where Date='2010/08/13' and Dept=1
```

OR Operator

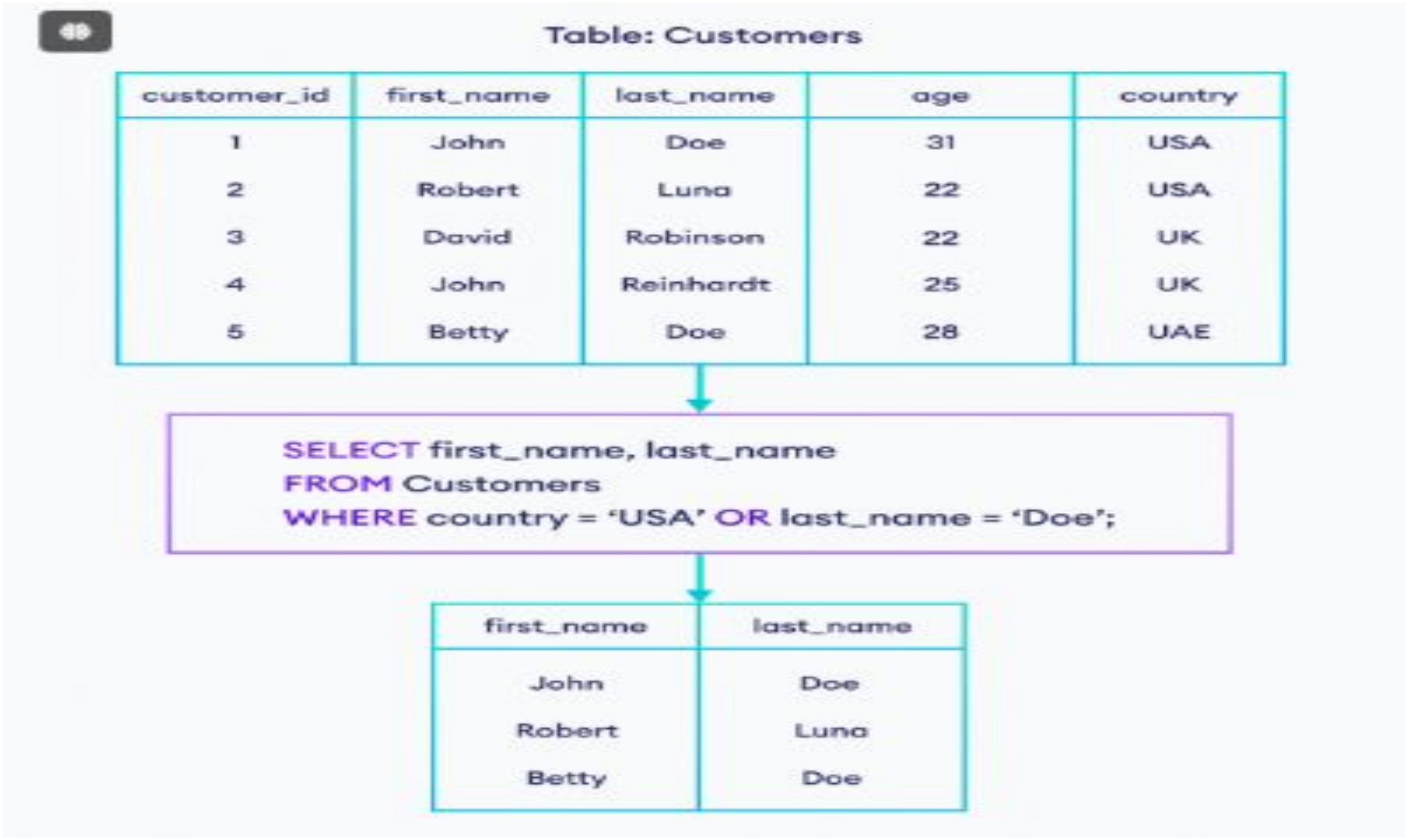


Table: Customers

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

OR Operator

It is used to display record if any of the conditions are separated by OR are TRUE

Syntax:

SELECT *column1*, *column2, ...*

FROM *table_name*

WHERE *condition1* OR *condition2* OR *condition3 ...*;

Display the Details of the Stores where the Type of the Store is "A" or size of the store is more than 100000.

```
Select * from stores where Type='A' OR Size>100000
```

NOT Operator

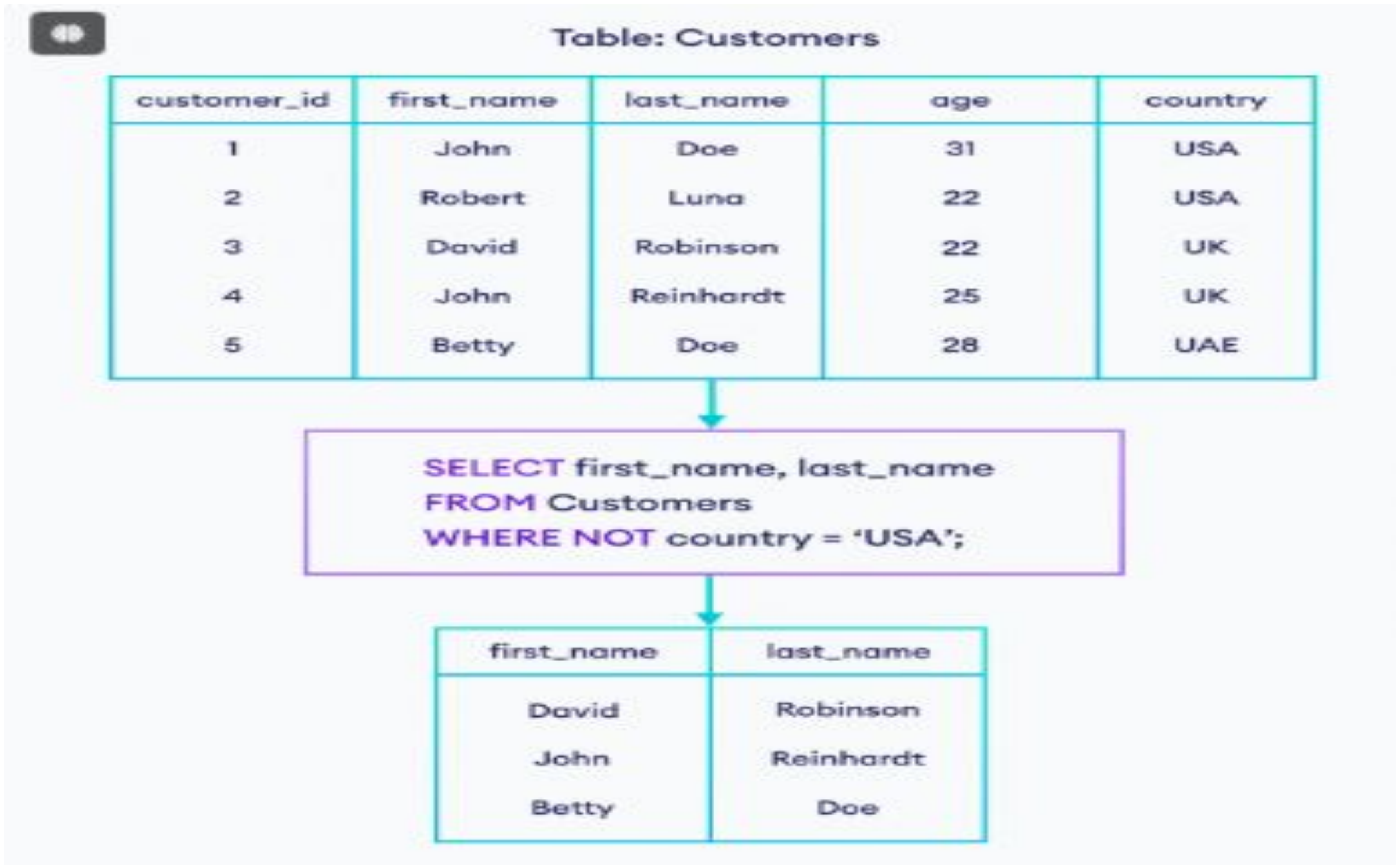


Table: Customers

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

```sql
SELECT first_name, last_name
FROM Customers
WHERE NOT country = 'USA';
```

first_name	last_name
David	Robinson
John	Reinhardt
Betty	Doe

NOT Operator

It is used to display record if the specified condition in the query is FALSE, the SQL NOT operator will show the data.

Syntax:

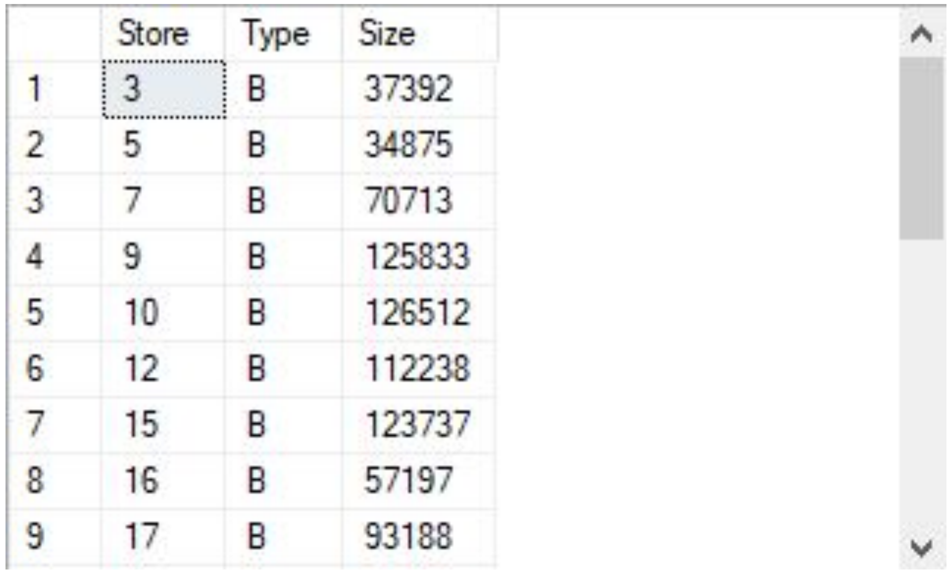
```
SELECT column1, column2, ...
FROM table_name
WHERE NOT condition;
```

Display all Detail of the Store where "A" type store is not present.

```
Select * from stores where NOT Type ='A'
```

NOT Operator

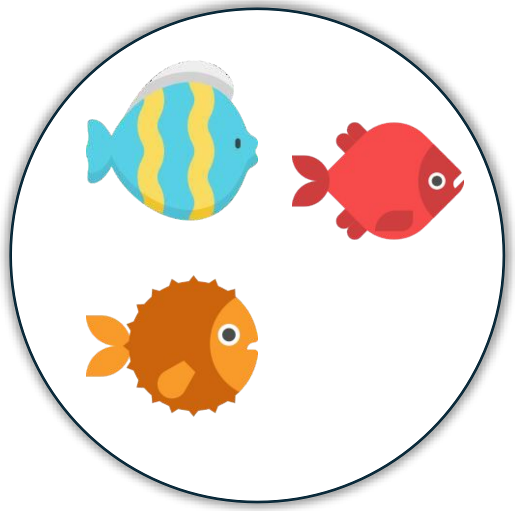
Output



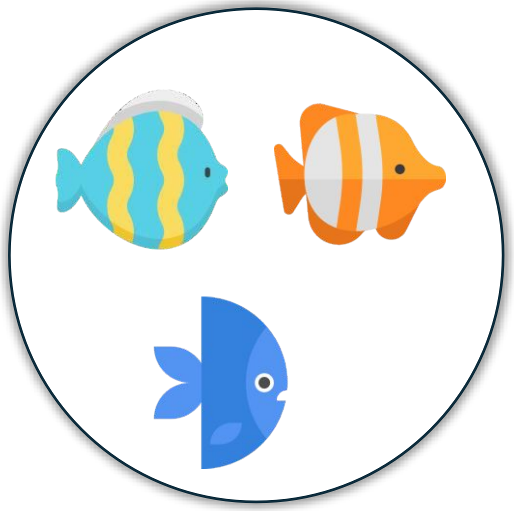
	Store	Type	Size
1	3	B	37392
2	5	B	34875
3	7	B	70713
4	9	B	125833
5	10	B	126512
6	12	B	112238
7	15	B	123737
8	16	B	57197
9	17	B	93188

Union Operator

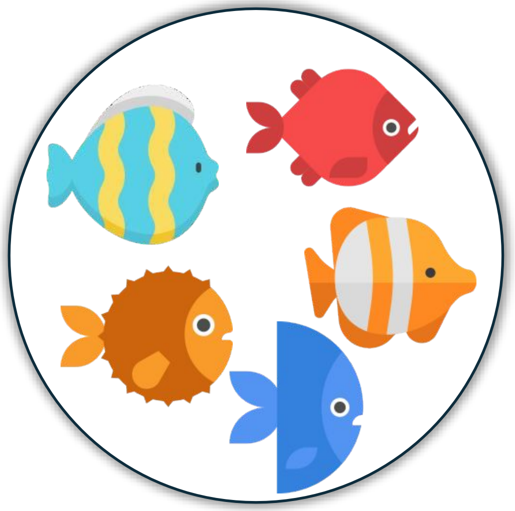
Union operator is used to combine the result set of two or more SELECT statements.



A



B



A U B

UNION

It is used to merge the result-set of two or more SELECT operations.

Syntax:

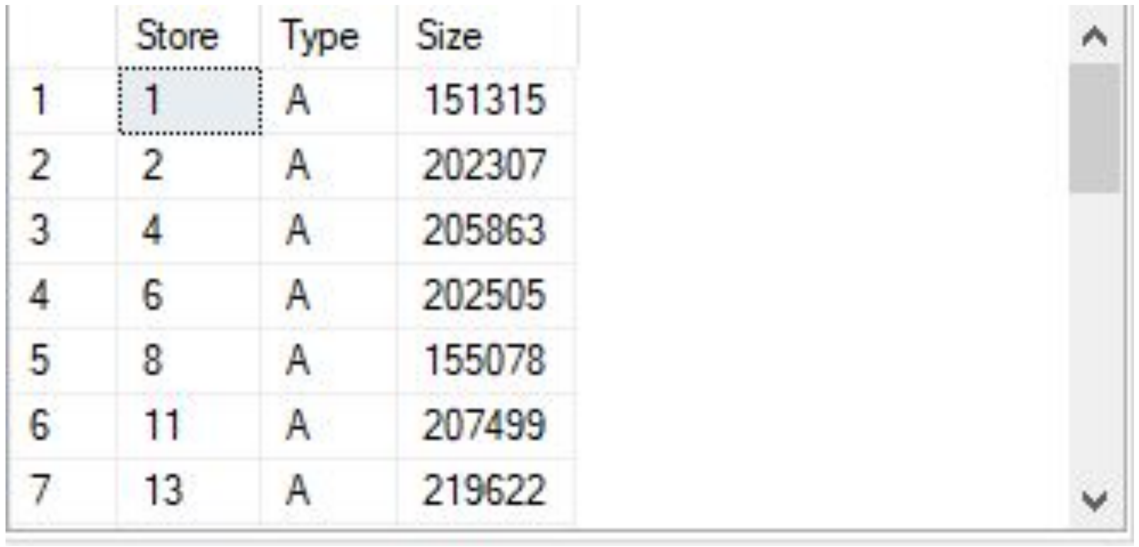
```
Select Column_Names From Table_Name1
Where Condition;
UNION
Select Column_Names From
Table_Name2 Where Condition;
```

Create a view for sales and store where weekly sales is less than 25000 and store is less than 15. Perform a operation in store and store view and display the unique data where store type is a and arrange it according to store.

UNION

```
Select * from stores where type='A‘ Union Select * from Store_View where type='A‘
 ORDER BY STORE
```

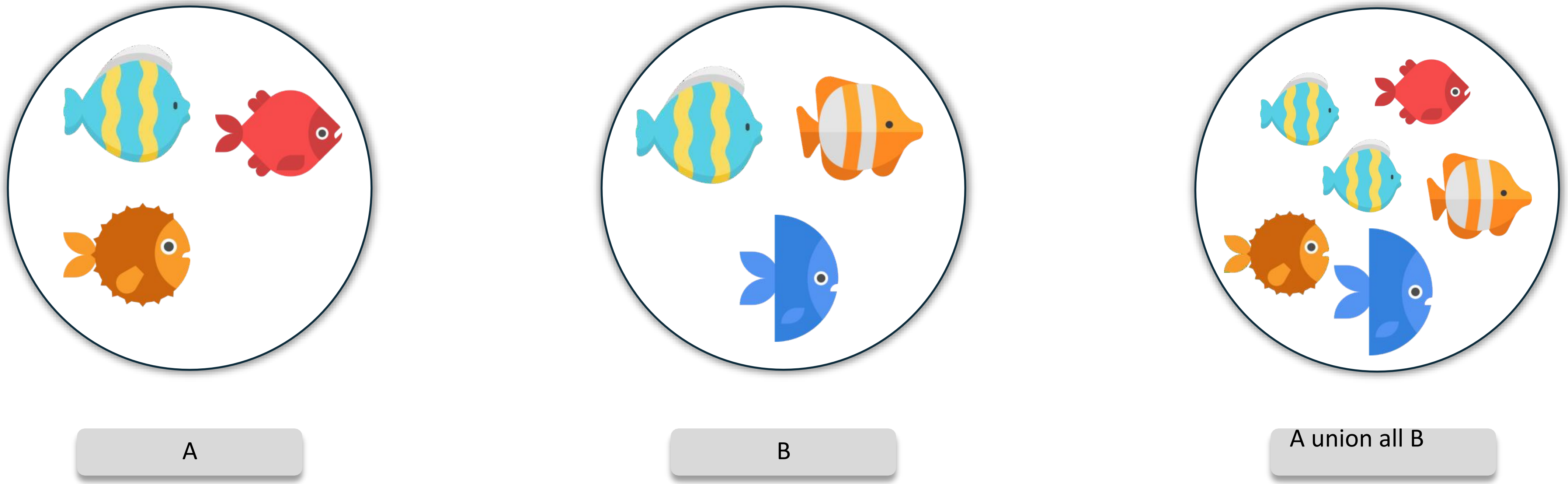
Output



	Store	Type	Size
1	1	A	151315
2	2	A	202307
3	4	A	205863
4	6	A	202505
5	8	A	155078
6	11	A	207499
7	13	A	219622

Union All Operator

Union All operator gives all rows from both tables including the duplicates.



UNION ALL

It is used to merge the result set of two or more SELECT statements. It allows duplicate values.

Syntax:

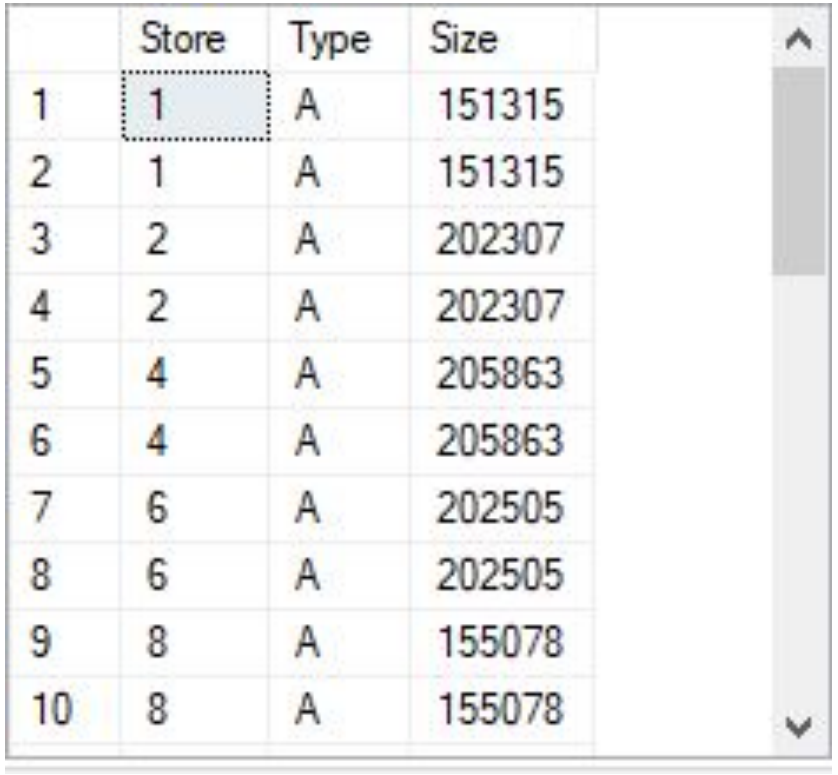
```
Select Column_Names From
Table_Name1 Where Condition;
UNION ALL
Select Column_Names From
Table_Name2 Where Condition;
```

Perform a operation in store and store view and display the unique data where store type is a and arrange it according to store.

UNION ALL

```
Select * from stores where type='A' Union all Select * from Store_View where type='A'
 ORDER BY STORE
```

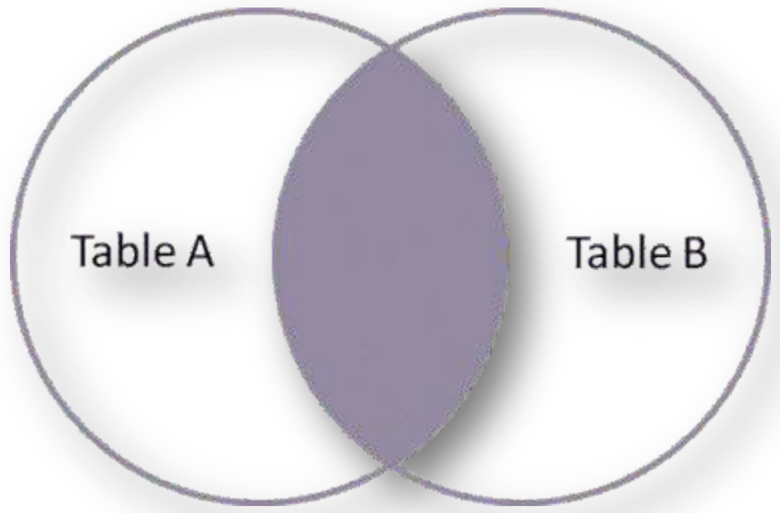
Output



	Store	Type	Size
1	1	A	151315
2	1	A	151315
3	2	A	202307
4	2	A	202307
5	4	A	205863
6	4	A	205863
7	6	A	202505
8	6	A	202505
9	8	A	155078
10	8	A	155078

Intersect Operator

Intersect Operator helps to combine two select statements and returns the records which are common to both the select statements.



A ∩ B

INTERSECT

It is used to return only those entries that are present in both SELECT statements.

Syntax:

```
Select Column_Names From Table_Name1
Where Condition;
Intersect
Select Column_Names From
Table_Name2 Where Condition;
```

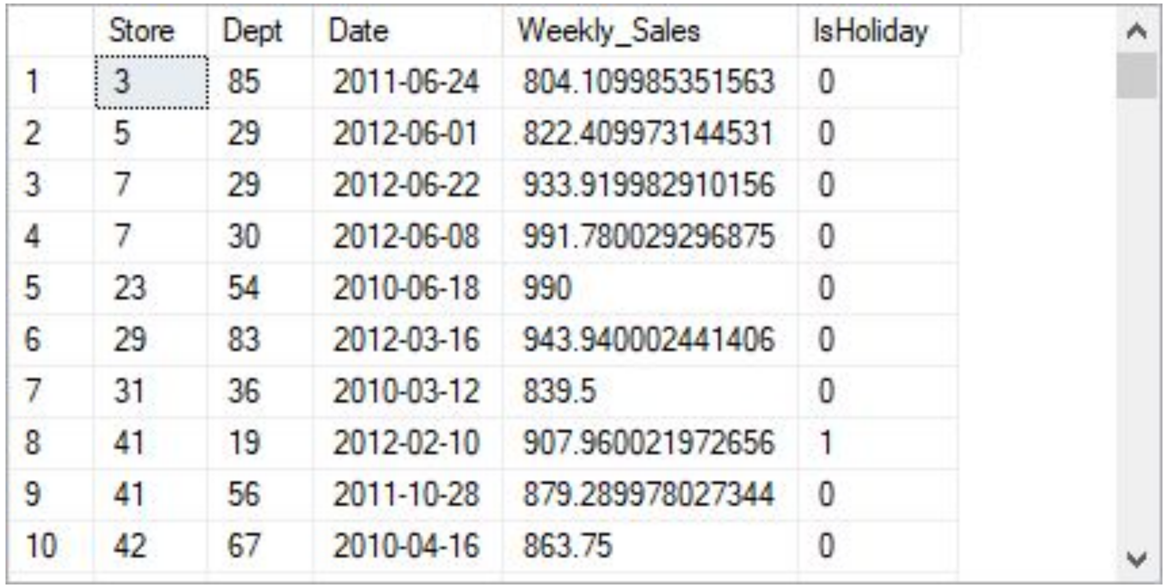
Perform a operation in sales and sales view and display the common data where weekly_sales of sales table is less than 1000 and weekly_sales of sales view table is greater than 800.

```
Select * from stores where type='A‘ Union Select * from Store_View where
type='A‘ ORDER BY STORE
```

INTERSECT

```
SELECT * FROM sales WHERE Weekly_Sales<1000 INTERSECT SELECT * FROM
Sales_View WHERE WEEKLY_SALES>800
```

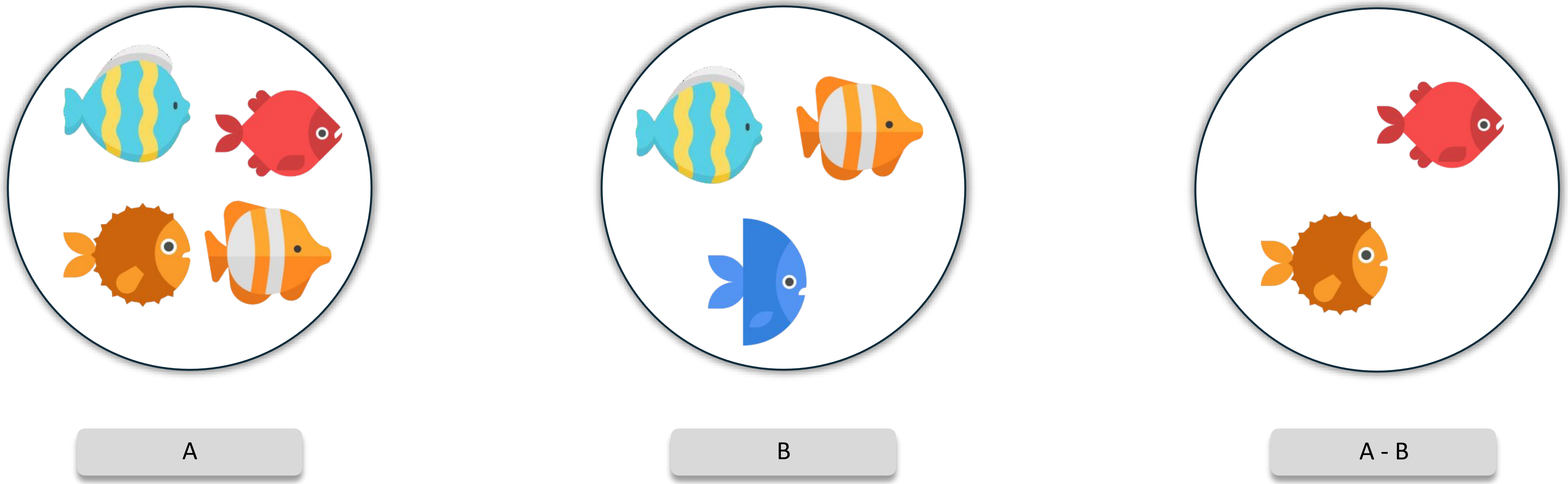
Output



	Store	Dept	Date	Weekly_Sales	IsHoliday
1	3	85	2011-06-24	804.109985351563	0
2	5	29	2012-06-01	822.409973144531	0
3	7	29	2012-06-22	933.919982910156	0
4	7	30	2012-06-08	991.780029296875	0
5	23	54	2010-06-18	990	0
6	29	83	2012-03-16	943.940002441406	0
7	31	36	2010-03-12	839.5	0
8	41	19	2012-02-10	907.960021972656	1
9	41	56	2011-10-28	879.289978027344	0
10	42	67	2010-04-16	863.75	0

EXCEPT

Except Operator combines two select statements and returns unique records from the left query which are not part of the right query.



EXCEPT

It is used to combine two SELECT statements and returns rows from the first SELECT statement that are not returned by the second SELECT statement. In other terms, EXCEPT returns only rows, which are not available in the second SELECT statement.

Syntax:

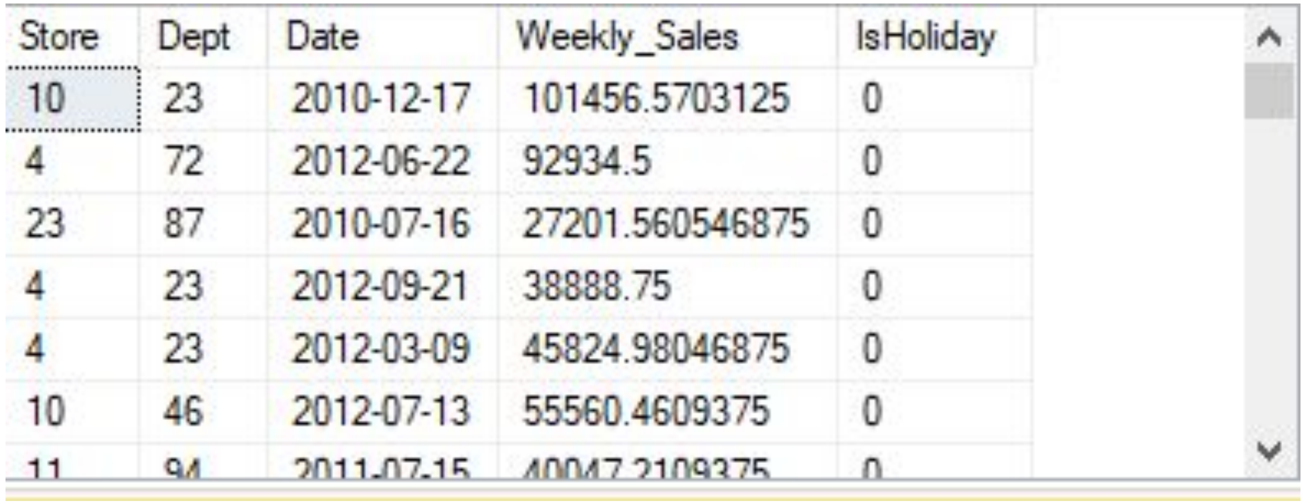
```
Select Column_Names From
Table_Name1 Where Condition;
EXCEPT
Select Column_Names From Table_Name2
Where Condition;
```

Use sales and sales_view table. Perform a except operation on sales view

EXCEPT

```
SELECT * FROM sales EXCEPT SELECT * FROM SALES_VIEW
```

Output



Store	Dept	Date	Weekly_Sales	IsHoliday
10	23	2010-12-17	101456.5703125	0
4	72	2012-06-22	92934.5	0
23	87	2010-07-16	27201.560546875	0
4	23	2012-09-21	38888.75	0
4	23	2012-03-09	45824.98046875	0
10	46	2012-07-13	55560.4609375	0
11	94	2011-07-15	40047.2109375	0

LIKE

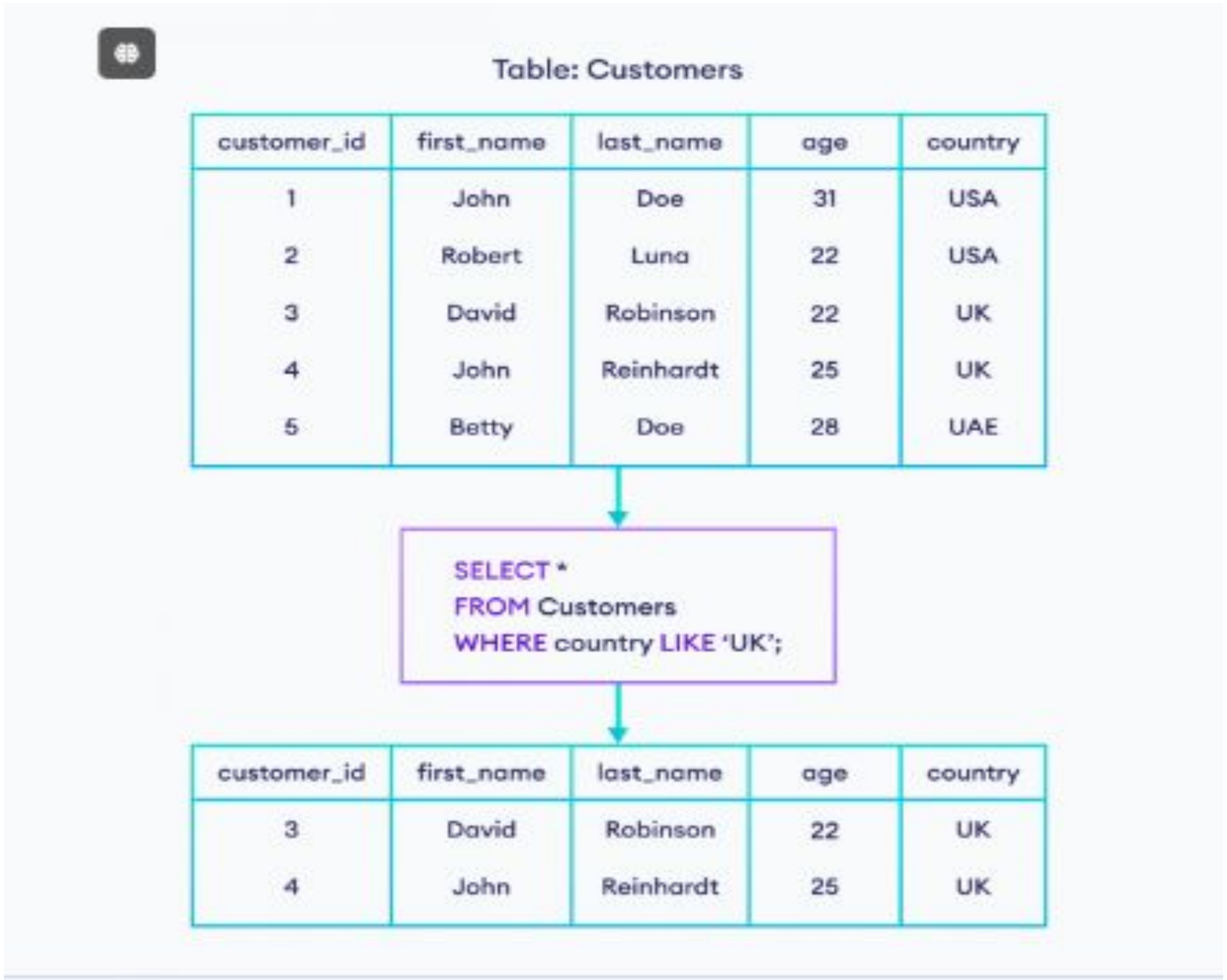


Table: Customers

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

LIKE

It is used to search a specified pattern in a column
% - Represents zero, one, or multiple characters
_ - Represents a single character

Syntax:
Select Column_Names from Table_Name where Column_name Like '%%'

Display the store, department, weekly_sales detail where the weekly sales starts with 45.

```
SELECT Store, Dept, Weekly_Sales FROM sales WHERE Weekly_Sales LIKE'45%'
```

BETWEEN

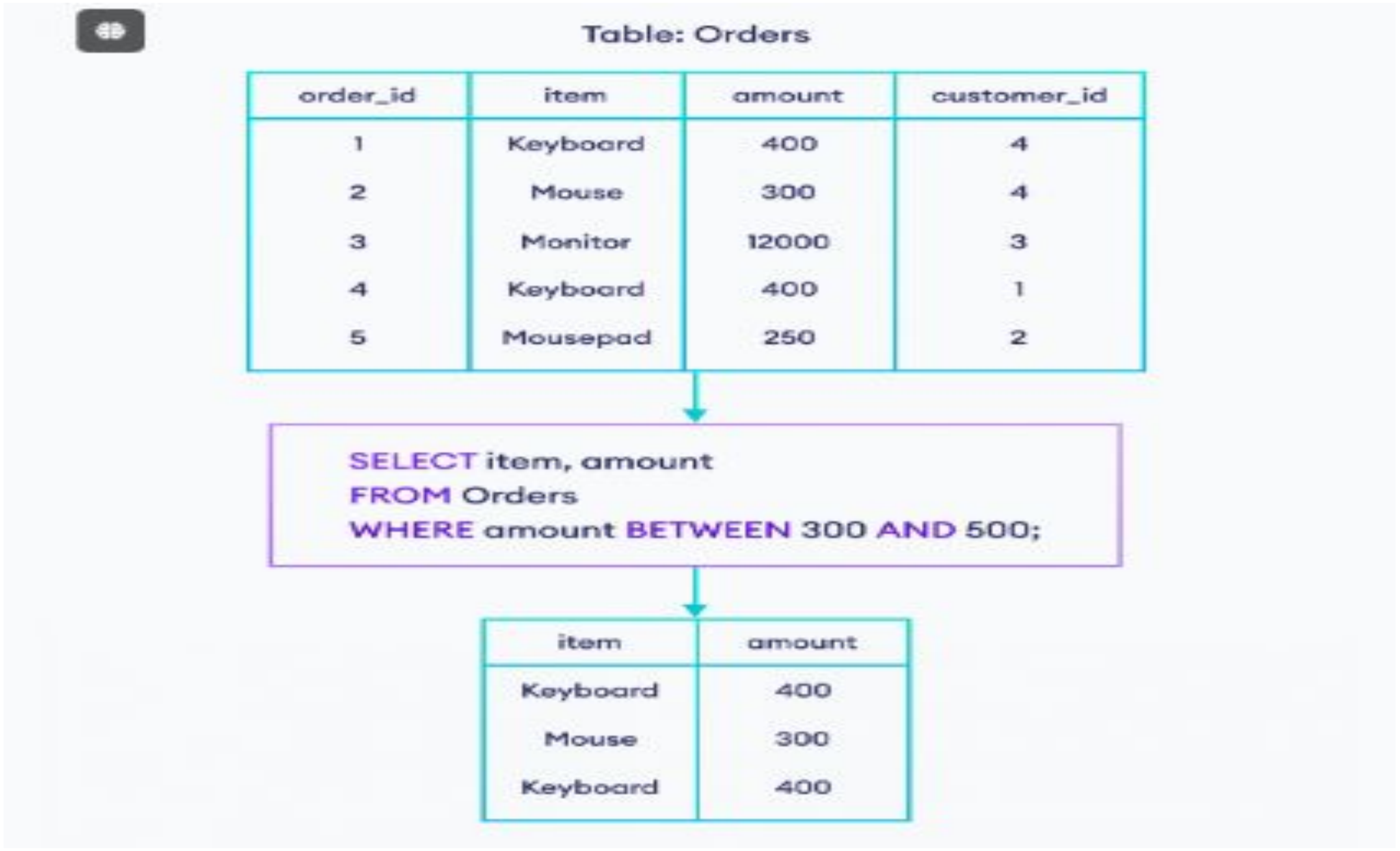


Table: Orders

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

BETWEEN

It is used to select values within a given range.

Syntax:
SELECT *column_name(s)*
FROM *table_name*
WHERE *column_name* BETWEEN *value1* AND *value2;*

Display the detail of the store along with fuel price where the consumer price index between 210 and 211.

Select store temperature, fuel_price , cpi from features where cpi between 210 and 211

CLAUSE

WHERE

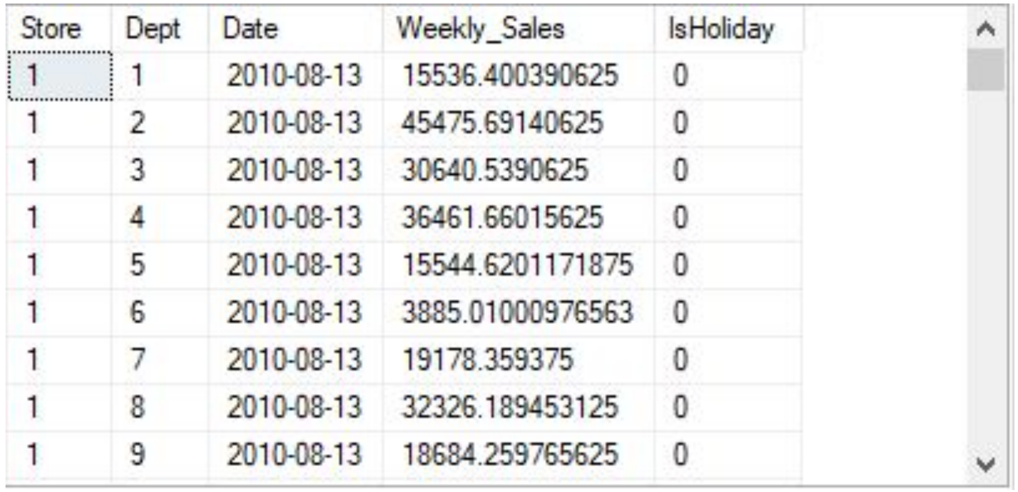
The `WHERE` clause is used to filter records.

WHERE

It is used to give some condition in SQL Query
Syntax - Select Column_Names From Table_Name where Condition;

Display the Detail of the Sales done on 13th August 2010.

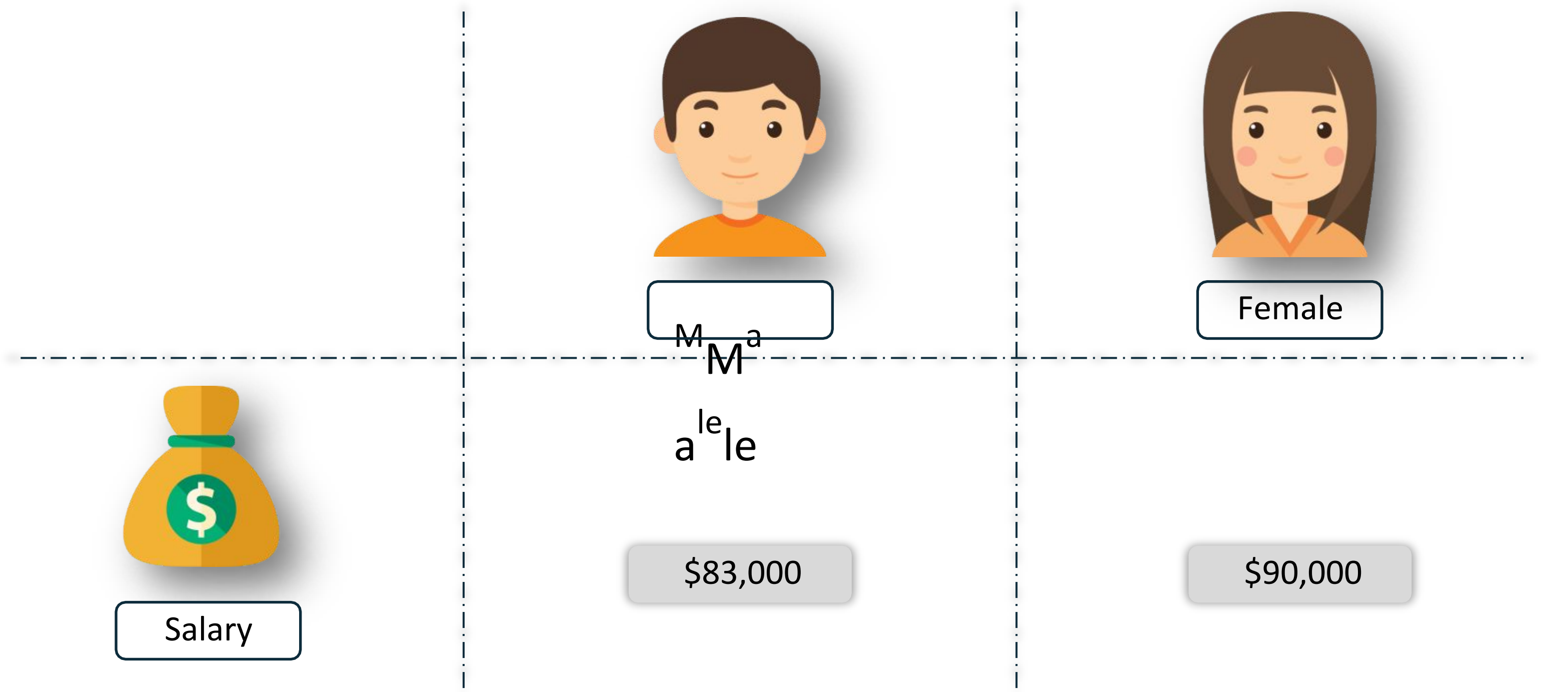
```
Select * from sales where Date= '2010-08-13'
```



Store	Dept	Date	Weekly_Sales	IsHoliday
1	1	2010-08-13	15536.400390625	0
1	2	2010-08-13	45475.69140625	0
1	3	2010-08-13	30640.5390625	0
1	4	2010-08-13	36461.66015625	0
1	5	2010-08-13	15544.6201171875	0
1	6	2010-08-13	3885.01000976563	0
1	7	2010-08-13	19178.359375	0
1	8	2010-08-13	32326.189453125	0
1	9	2010-08-13	18684.259765625	0

GROUP BY

Group By is used to get an aggregate result with respect to a group.



GROUP BY

It groups the selected rows based on identical values in a column or expression.

Syntax:

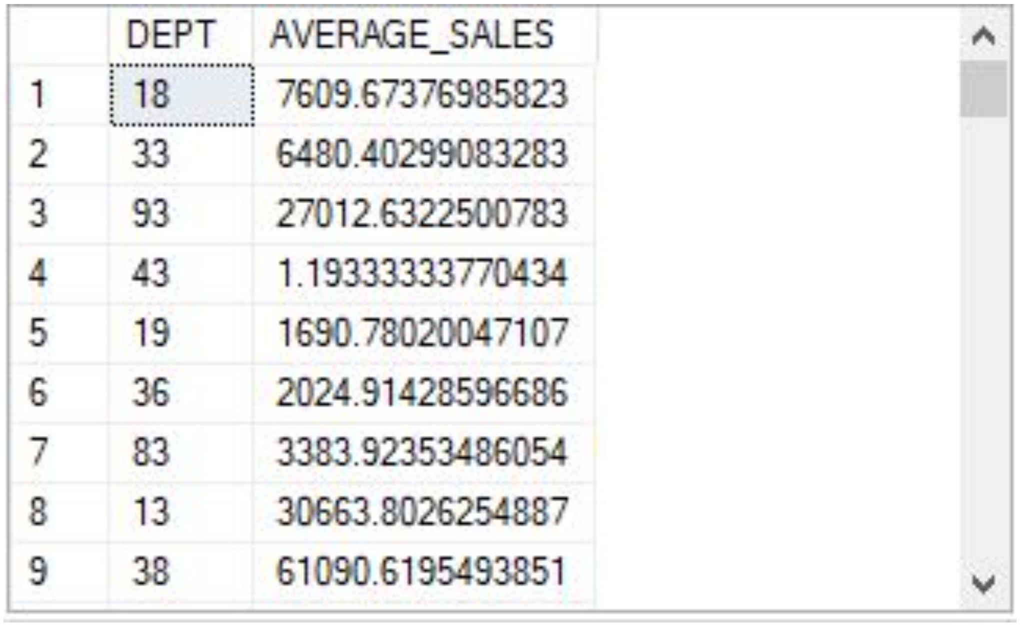
SELECT *column_names*
FROM *table_name*
WHERE *condition*
GROUP BY *column_names;*

Display the average weekly_sales of each department from sales

```
SELECT DEPT,AVG(WEEKLY_SALES) AS AVERAGE_SALES FROM SALES GROUP BY DEPT
```

GROUP BY

Output



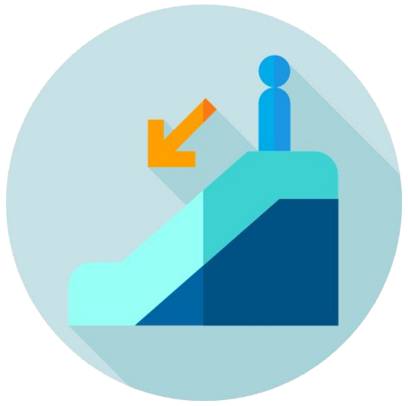
	DEPT	AVERAGE_SALES
1	18	7609.67376985823
2	33	6480.40299083283
3	93	27012.6322500783
4	43	1.19333333770434
5	19	1690.78020047107
6	36	2024.91428596686
7	83	3383.92353486054
8	13	30663.8026254887
9	38	61090.6195493851

ORDER BY

It is used to sort the result-set in ascending or descending order.

Syntax:
SELECT *column_names*
FROM *table_name*
ORDER BY *column_names;*

Ascendin
g



Descending

ORDER BY

It is used to sort the result-set in ascending or descending order.

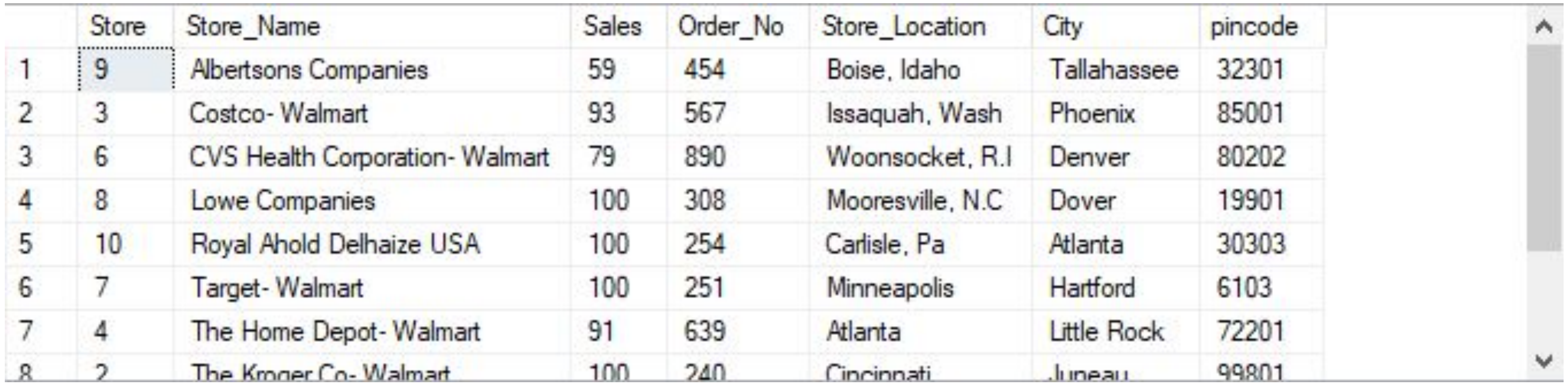
Syntax:
SELECT *column_names*
FROM *table_name*
ORDER BY *column_names;*

Display store name in alphabetical order from the table Store_details

```
select * from store_details order by store_name
```

ORDER BY

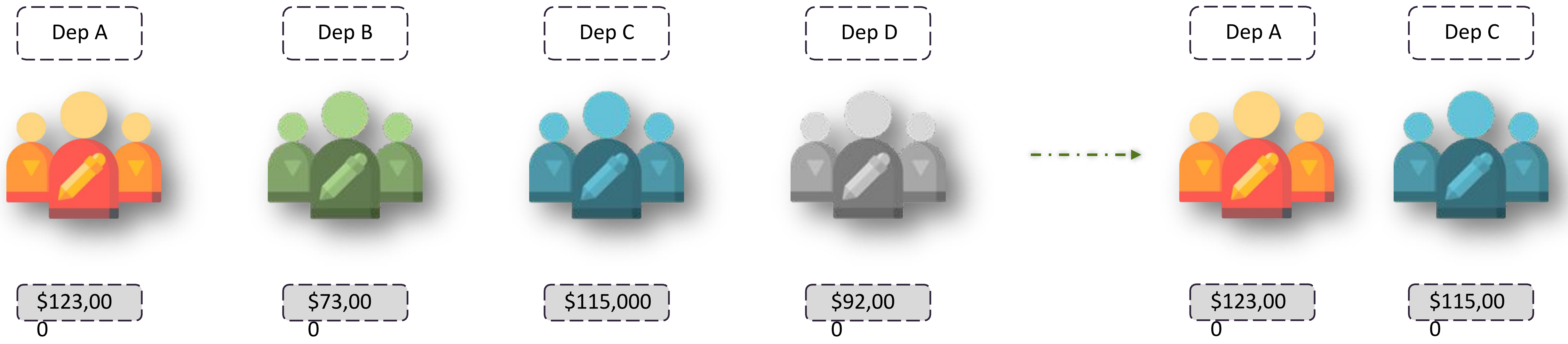
Output



	Store	Store_Name	Sales	Order_No	Store_Location	City	pincode
1	9	Albertsons Companies	59	454	Boise, Idaho	Tallahassee	32301
2	3	Costco- Walmart	93	567	Issaquah, Wash	Phoenix	85001
3	6	CVS Health Corporation- Walmart	79	890	Woonsocket, R.I	Denver	80202
4	8	Lowe Companies	100	308	Mooresville, N.C	Dover	19901
5	10	Royal Ahold Delhaize USA	100	254	Carlisle, Pa	Atlanta	30303
6	7	Target- Walmart	100	251	Minneapolis	Hartford	6103
7	4	The Home Depot- Walmart	91	639	Atlanta	Little Rock	72201
8	2	The Kroger Co- Walmart	100	240	Cincinnati	Juneau	99801

HAVING

Having clause is used in combination with Group By to impose conditions on groups.



HAVING

The HAVING clause in SQL is utilized to filter the outcome set in view of aggregate functions like MIN() and MAX(), SUM() and AVG() and COUNT().

Syntax:

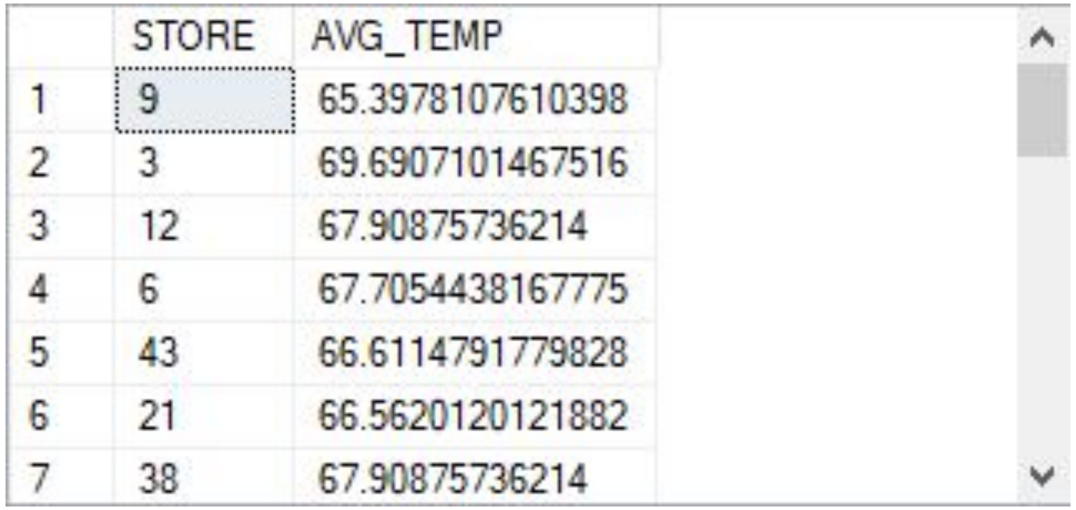
```
Select Column_Names
From Table_Name
Where Condition
Group By Column_name
Having Condition ;
```

Display the average temperature of the store where average of temperature should be less than 55 fahrenheit

```
SELECT STORE,AVG(TEMPERATURE) AS AVG_TEMP FROM [dbo].[Features] GROUP BY
 Store HAVING AVG(Temperature)>55
```

HAVING

Output

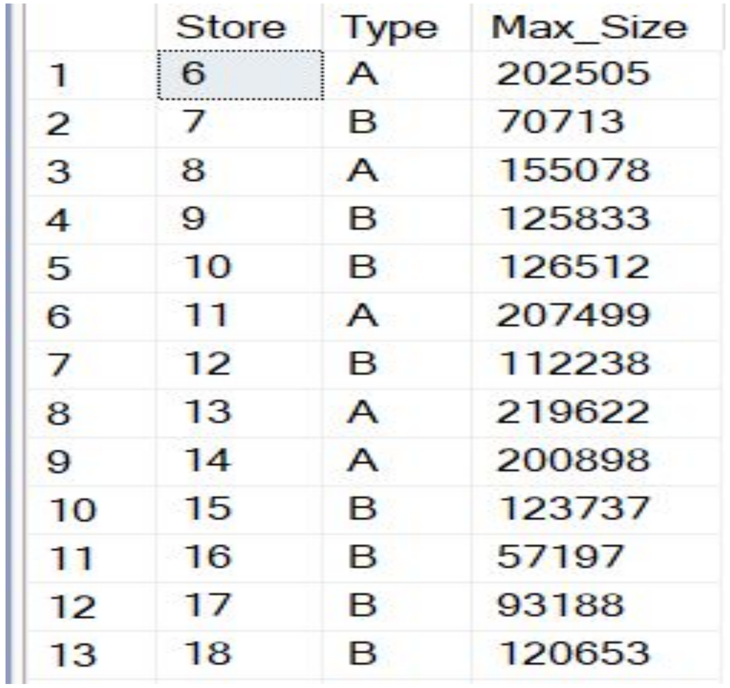


	STORE	AVG_TEMP
1	9	65.3978107610398
2	3	69.6907101467516
3	12	67.90875736214
4	6	67.7054438167775
5	43	66.6114791779828
6	21	66.5620120121882
7	38	67.90875736214

HAVING CLAUSE WITH GROUP BY & ORDER BY

Find the maximum size of store where stores are not repeated and store value should be greater than 5
and group it by Store and Type columns where maximum store size should be greater than 50000.

```
SELECT Distinct Store,Type,
MAX(Size) FROM stores
 WHERE Store>5
 group by
 Store,Type
 having MAX(Size)> 50000
 order by Store
```

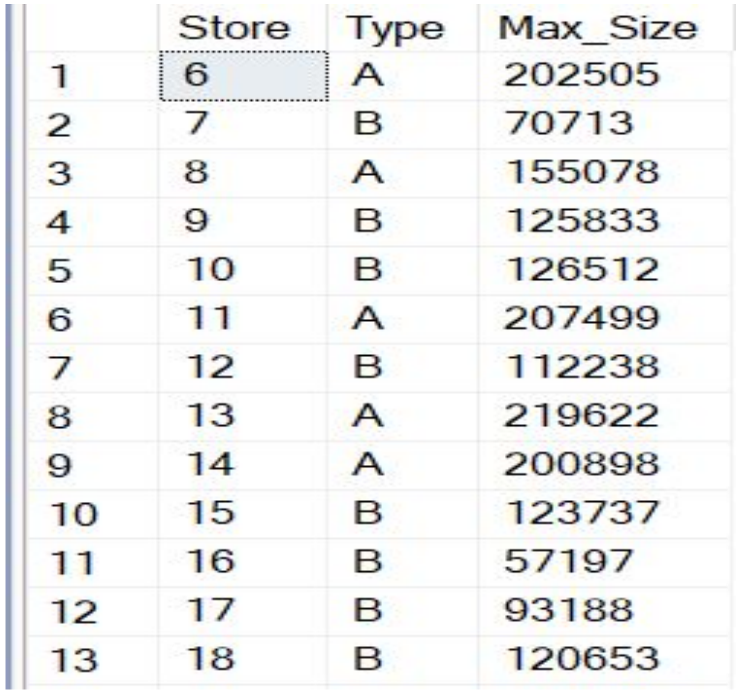


	Store	Type	Max_Size
1	6	A	202505
2	7	B	70713
3	8	A	155078
4	9	B	125833
5	10	B	126512
6	11	A	207499
7	12	B	112238
8	13	A	219622
9	14	A	200898
10	15	B	123737
11	16	B	57197
12	17	B	93188
13	18	B	120653

HAVING CLAUSE WITH GROUP BY & ORDER BY

Display the Sales data with the Department, Year of date, and the number of stores and Group it by Department.

```
  SELECT Distinct Store,Type,
MAX(Size)
  FROM
  stores
   WHERE Store>5
   group by
   Store,Type
  having MAX(Size)> 50000
   order by Store
```



	Store	Type	Max_Size
1	6	A	202505
2	7	B	70713
3	8	A	155078
4	9	B	125833
5	10	B	126512
6	11	A	207499
7	12	B	112238
8	13	A	219622
9	14	A	200898
10	15	B	123737
11	16	B	57197
12	17	B	93188
13	18	B	120653

TEMPORARY TABLE

Temporary Tables are most likely as Permanent Tables.

Temporary Tables are Created in TempDB and are automatically deleted as soon as the last connection is terminated.

SYNTAX

```
CREATE TEMPORARY TABLE
CUSTOMERS( ID INT NOT NULL,
NAME VARCHAR (20) NOT NULL,
AGE INT NOT NULL, ADDRESS
CHAR (25) , SALARY DECIMAL (18,
2), PRIMARY KEY (ID) );
```

TEMPORARY TABLE

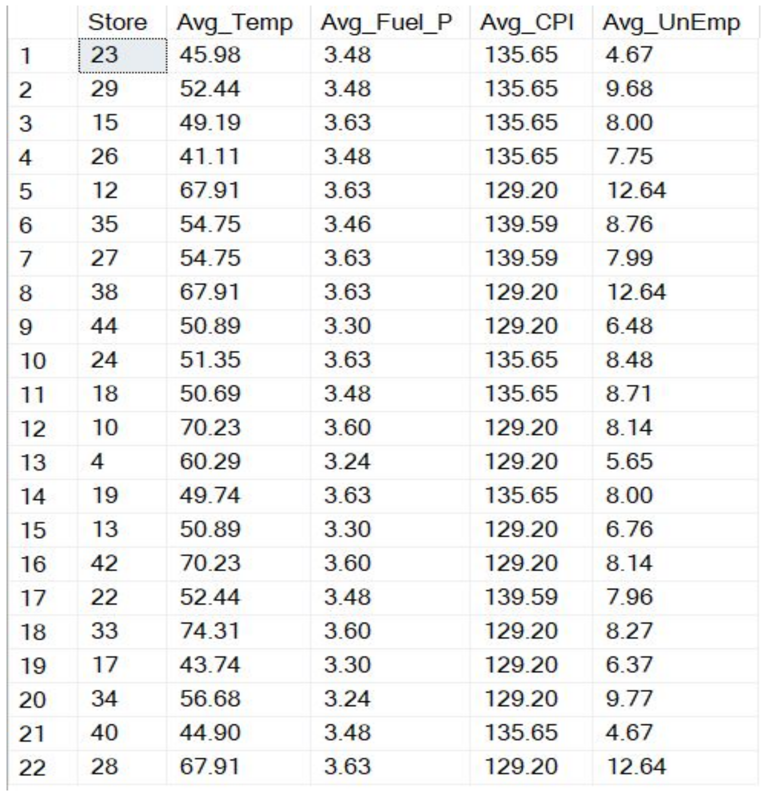
Temporary tables are tables that exist temporarily on the SQL Server. The temporary tables are useful for storing the immediate result sets that are accessed multiple times.

Create a Temp table containing the Average of Temperature, Fuel_price, CPI, and Unemployment of each Store where avg CPI is less than 150

```
CREATE TABLE #Avg_F(Store INT, Avg_Temp DECIMAL(18,2),Avg_Fuel_P
DECIMAL(18,2),Avg_CPI DECIMAL(18,2),Avg_UnEmp DECIMAL(18,2))
INSERT INTO #Avg_F(Store, Avg_Temp, Avg_Fuel_P, Avg_CPI, Avg_UnEmp)
SELECT Store,AVG(Temperature),AVG(Fuel_Price),AVG(CPI), AVG(Unemployment) FROM Features GROUP
BY Store having AVG(CPI)<150 order by
Store SELECT * FROM #Avg_F
```

TEMPORARY TABLE

Output



	Store	Avg_Temp	Avg_Fuel_P	Avg_CPI	Avg_UnEmp
1	23	45.98	3.48	135.65	4.67
2	29	52.44	3.48	135.65	9.68
3	15	49.19	3.63	135.65	8.00
4	26	41.11	3.48	135.65	7.75
5	12	67.91	3.63	129.20	12.64
6	35	54.75	3.46	139.59	8.76
7	27	54.75	3.63	139.59	7.99
8	38	67.91	3.63	129.20	12.64
9	44	50.89	3.30	129.20	6.48
10	24	51.35	3.63	135.65	8.48
11	18	50.69	3.48	135.65	8.71
12	10	70.23	3.60	129.20	8.14
13	4	60.29	3.24	129.20	5.65
14	19	49.74	3.63	135.65	8.00
15	13	50.89	3.30	129.20	6.76
16	42	70.23	3.60	129.20	8.14
17	22	52.44	3.48	139.59	7.96
18	33	74.31	3.60	129.20	8.27
19	17	43.74	3.30	129.20	6.37
20	34	56.68	3.24	129.20	9.77
21	40	44.90	3.48	135.65	4.67
22	28	67.91	3.63	129.20	12.64