SCHOOL OF DATA SCIENCE

ON

COMPREHENSIVE REPORT ON SQL

BY

KELECHI IMMELDAH UWANAKA

March, 2025

INTRODUCTION

SQL is a standard language for accessing and manipulating databases. It is widely used for tasks like; Querying data, manipulating data, defining structure, and Controlling access to a database. SQL is highly versatile and forms the backbone of many database systems such as MYSQL, PostgreSQL, Microsoft SQL Server, and SQLite. It's an essential tool for developers and data analysts alike.

What is SQL
SQL stands for Structured Query Language; it lets you access and manipulate databases. SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987.
Although SQL is an ANSI/ISO standard, there are different versions of the SQL language. However, to be compliant with the ANSI standard, they all support at least the major commands (Such as SELECT, UPDATE, DELETE, INSERT, WHERE) in a similar manner.

What Can SQL do?

- SQL can execute queries against a database
- SQL can retrieve data from a database
- SQL can insert records in a database
- SQL can update records in a database
- SQL can delete records from a database
- SQL can create new databases
- SQL can create new tables in a database
- SQL can create stored procedures in a database
- SQL can create views in a database
- SQL can set permissions on tables, procedures, and views

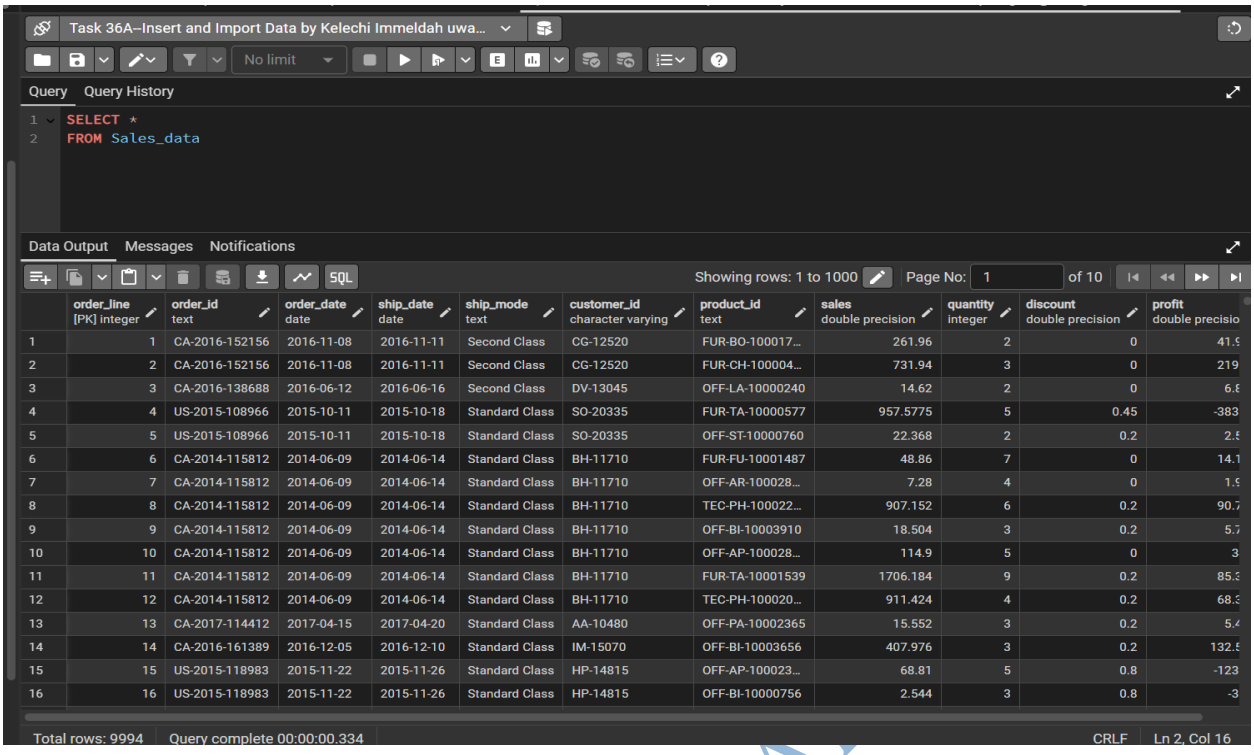
RDBMS

RDBMS stands for Relational Database Management System. RDBMS is the basis for SQL and for all modern database systems such as MS SQL Server, IBM DB2, Oracle, MySQL, and Microsoft Access. The data in RDBMS is stored in database objects called tables.

TABLES IN SQL
A table is a collection of related data entries and it consists of columns and rows.

Example:

```
SELECT *
FROM Sales_data
```



	order_line [PK] integer	order_id text	order_date date	ship_date date	ship_mode text	customer_id character varying	product_id text	sales double precision	quantity integer	discount double precision	profit double precisio
1	1	CA-2016-152156	2016-11-08	2016-11-11	Second Class	CG-12520	FUR-BO-100017...	261.96	2	0	41.9
2	2	CA-2016-152156	2016-11-08	2016-11-11	Second Class	CG-12520	FUR-CH-100004...	731.94	3	0	219
3	3	CA-2016-138688	2016-06-12	2016-06-16	Second Class	DV-13045	OFF-LA-10000240	14.62	2	0	6.8
4	4	US-2015-108966	2015-10-11	2015-10-18	Standard Class	SO-20335	FUR-TA-10000577	957.5775	5	0.45	-383
5	5	US-2015-108966	2015-10-11	2015-10-18	Standard Class	SO-20335	OFF-ST-10000760	22.368	2	0.2	2.8
6	6	CA-2014-115812	2014-06-09	2014-06-14	Standard Class	BH-11710	FUR-FU-10001487	48.86	7	0	14.1
7	7	CA-2014-115812	2014-06-09	2014-06-14	Standard Class	BH-11710	OFF-AR-100028...	7.28	4	0	1.9
8	8	CA-2014-115812	2014-06-09	2014-06-14	Standard Class	BH-11710	TEC-PH-100022...	907.152	6	0.2	90.7
9	9	CA-2014-115812	2014-06-09	2014-06-14	Standard Class	BH-11710	OFF-BI-10003910	18.504	3	0.2	5.7
10	10	CA-2014-115812	2014-06-09	2014-06-14	Standard Class	BH-11710	OFF-AP-100028...	114.9	5	0	3
11	11	CA-2014-115812	2014-06-09	2014-06-14	Standard Class	BH-11710	FUR-TA-10001539	1706.184	9	0.2	85.3
12	12	CA-2014-115812	2014-06-09	2014-06-14	Standard Class	BH-11710	TEC-PH-100020...	911.424	4	0.2	68.3
13	13	CA-2017-114412	2017-04-15	2017-04-20	Standard Class	AA-10480	OFF-PA-10002365	15.552	3	0.2	5.4
14	14	CA-2016-161389	2016-12-05	2016-12-10	Standard Class	IM-15070	OFF-BI-10003656	407.976	3	0.2	132.5
15	15	US-2015-118983	2015-11-22	2015-11-26	Standard Class	HP-14815	OFF-AP-100023...	68.81	5	0.8	-123
16	16	US-2015-118983	2015-11-22	2015-11-26	Standard Class	HP-14815	OFF-BI-10000756	2.544	3	0.8	-3

Every table is broken up into smaller entities called fields. The fields in the sale's data consist of order_id, order_date, ship_date, ship_mode, customer_id, Product_id, sales, Quantity, discount and profit. A field is a column in a table that is designed to maintain specific information about every record in the table.

A record, also called a row, is each individual entry in a table. For example, the above sales data has multiple records. A record is a horizontal entity in a table. A column is a vertical entity in a table that contains all information associated with a specific field in a table.

SQL SYNTAX

SQL Statements

Most of the actions you need to perform on a database are done with SQL statements. SQL statements consist of keywords that are easy to understand. The following SQL statement returns all records from a table named "sales_data":

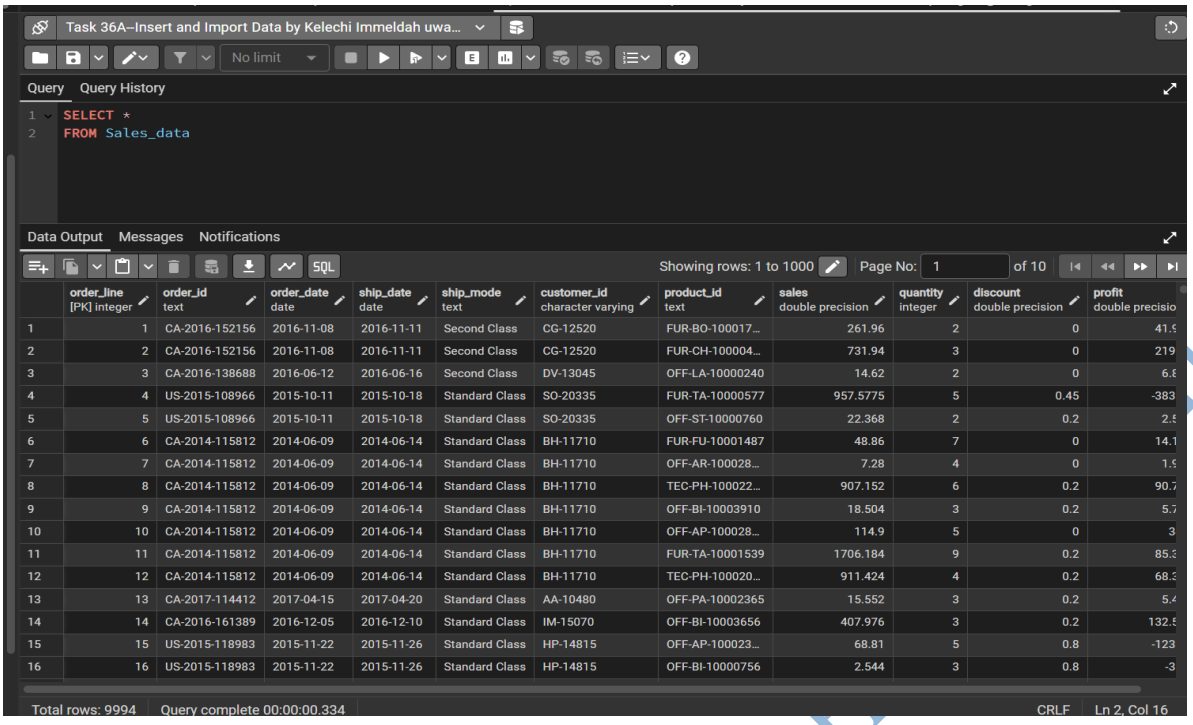
Example:

To select all records from the sales_data table:

SELECT * FROM Sales_data;

DATABASE TABLES

A database most often contains one or more tables. Each table is identified by a name (e.g. "Sales data" or "Checkout") and contains records (rows) with data. Below is a selection from the sales table used in the examples:



The table above contains multiple records (one for each book) and five columns (Order_line, order_id, order_date, ship_date, ship_mode, customer_id, Product_id, sales, Quantity, discount, and profit.).
Note that SQL keywords are NOT case sensitive: select is the same as SELECT, but uppercase is used for convention and readability purposes.

SEMICOLON AFTER SQL STATEMENTS
Some database systems require a semicolon at the end of each SQL statement. A semicolon is the standard way to separate each SQL statement in database systems that allow more than one SQL statement to be executed in the same call to the server.

SOME OF THE MOST IMPORTANT SQL COMMANDS

- SELECT - extracts data from a database
- UPDATE - updates data in a database
- DELETE - deletes data from a database
- INSERT INTO - inserts new data into a database
- CREATE DATABASE - creates a new database
- ALTER DATABASE - modifies a database
- CREATE TABLE - creates a new table
- ALTER TABLE - modifies a table
- DROP TABLE - deletes a table
- CREATE INDEX - creates an index (search key)
- DROP INDEX - deletes an index

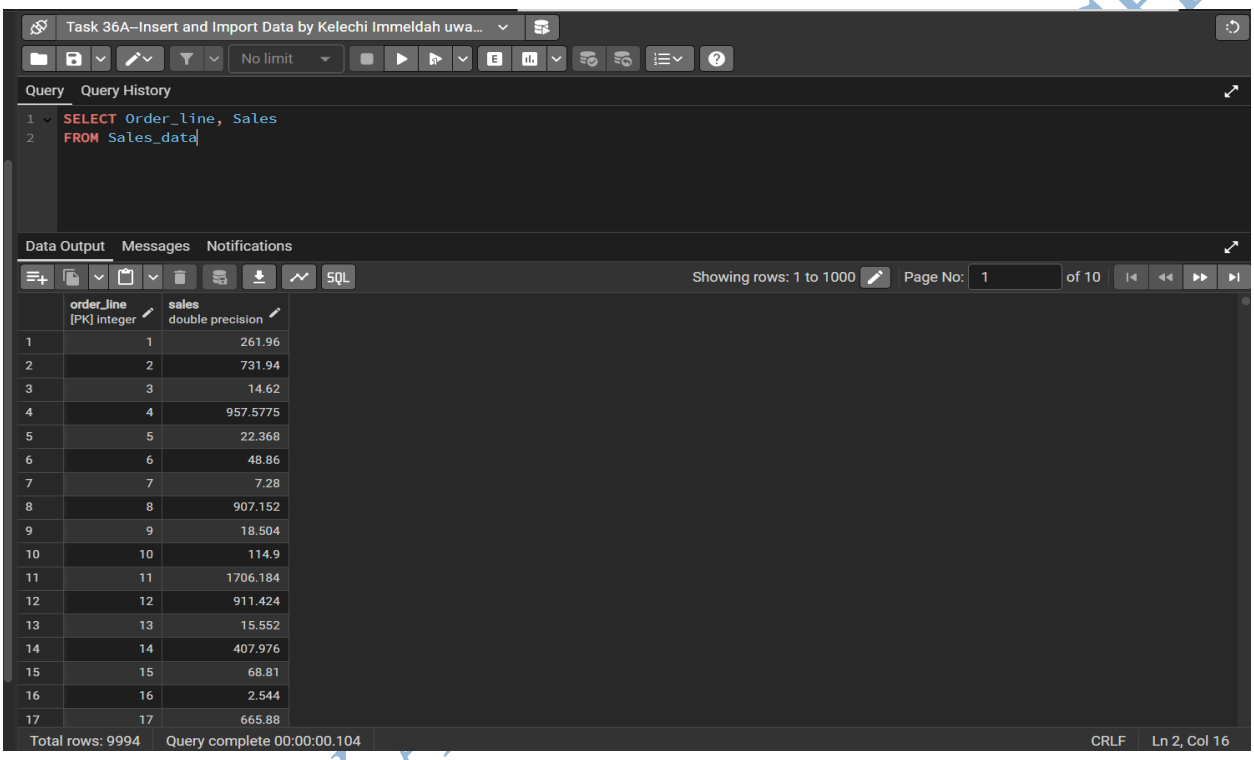
THE SQL SELECT STATEMENT

The SELECT statement is used to select data from a database. Syntax

SELECT *column1*, *column2,* ...
FROM *table_name*;

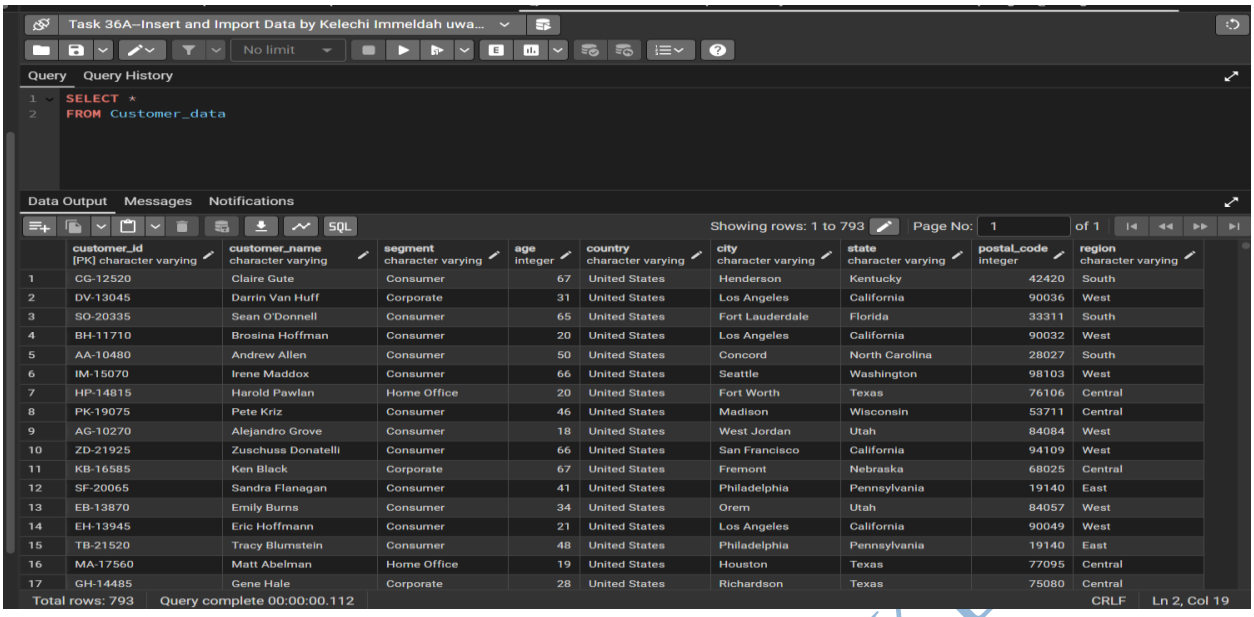
Here, column1, column2, ... are the *field names* of the table you want to select data from.
The table_name represents the name of the *table* you want to select data from.

Example:
Return data from the Sales_data table:
SELECT Order_id, sales FROM Sales_data;

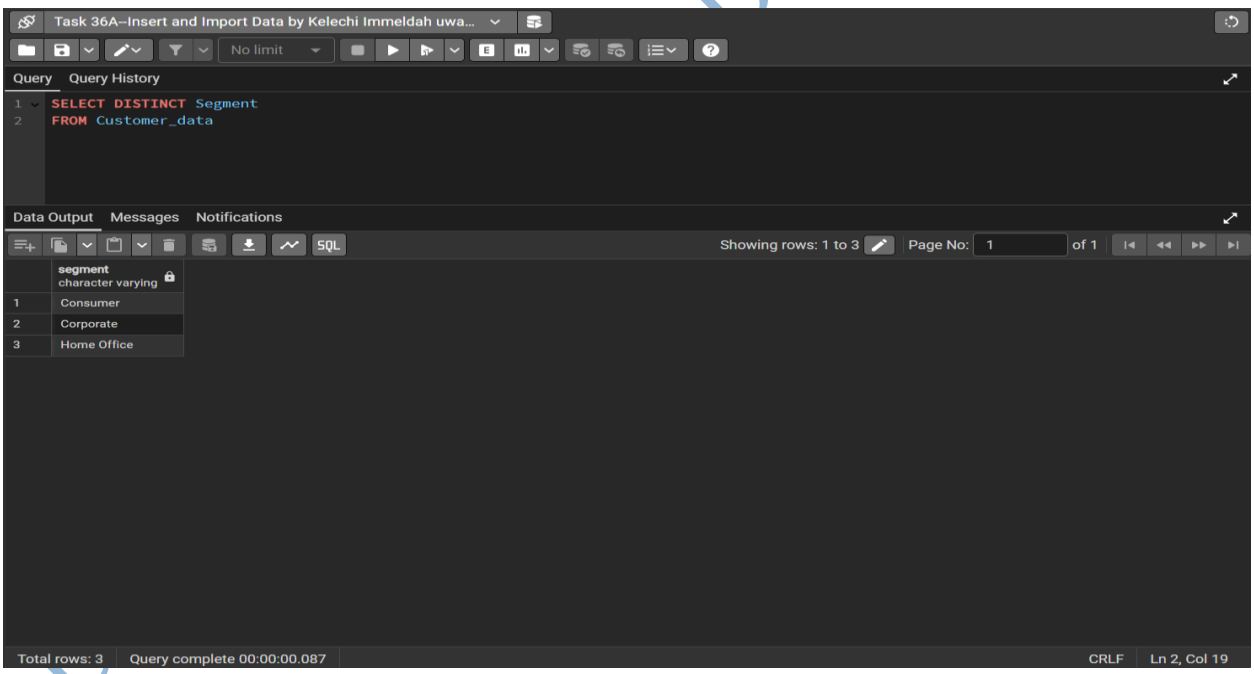


If you want to return all columns, without specifying every column name, you can use the SELECT * FROM Sales_data;

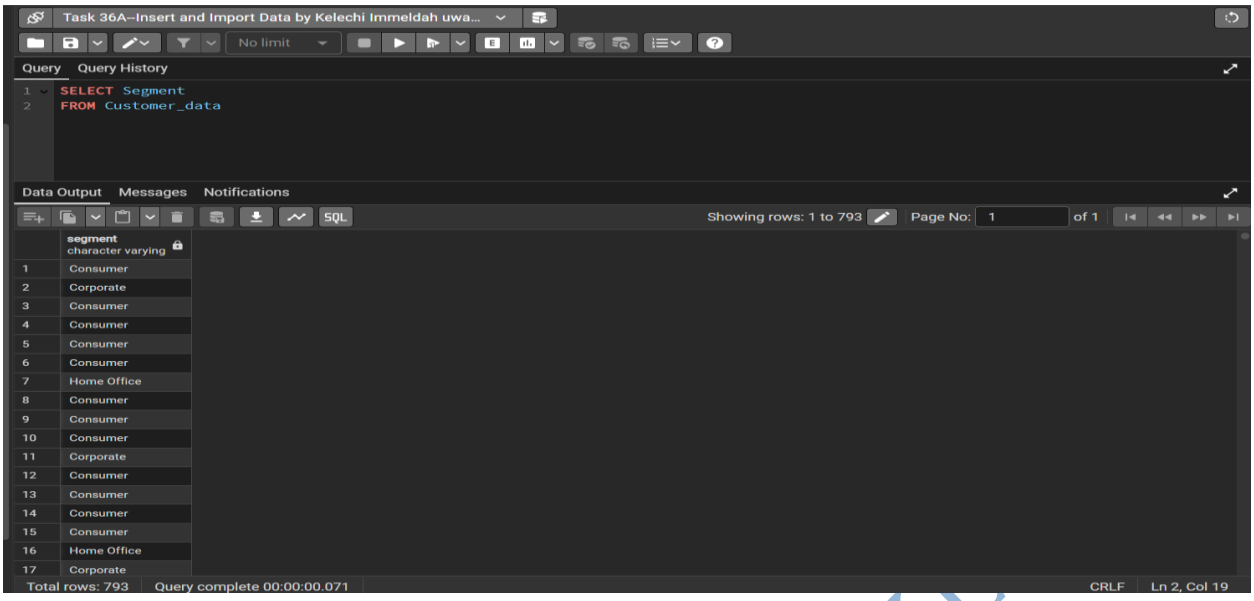
THE SQL SELECT DISTINCT STATEMENT
The SELECT DISTINCT statement is used to return only distinct (different) values.
Syntax
SELECT DISTINCT *column1*, *column2, ...*
FROM *table_name*;
For example, we have this customer data table below:



Example:
Select all the different Segment from the "Customer Data" table:
SELECT DISTINCT Segment
FROM Customer_data;



If you omit the DISTINCT keyword, the SQL statement returns the "segment" value from all the records of the "Customer Data" table:
Example
SELECT state FROM Customer_data;

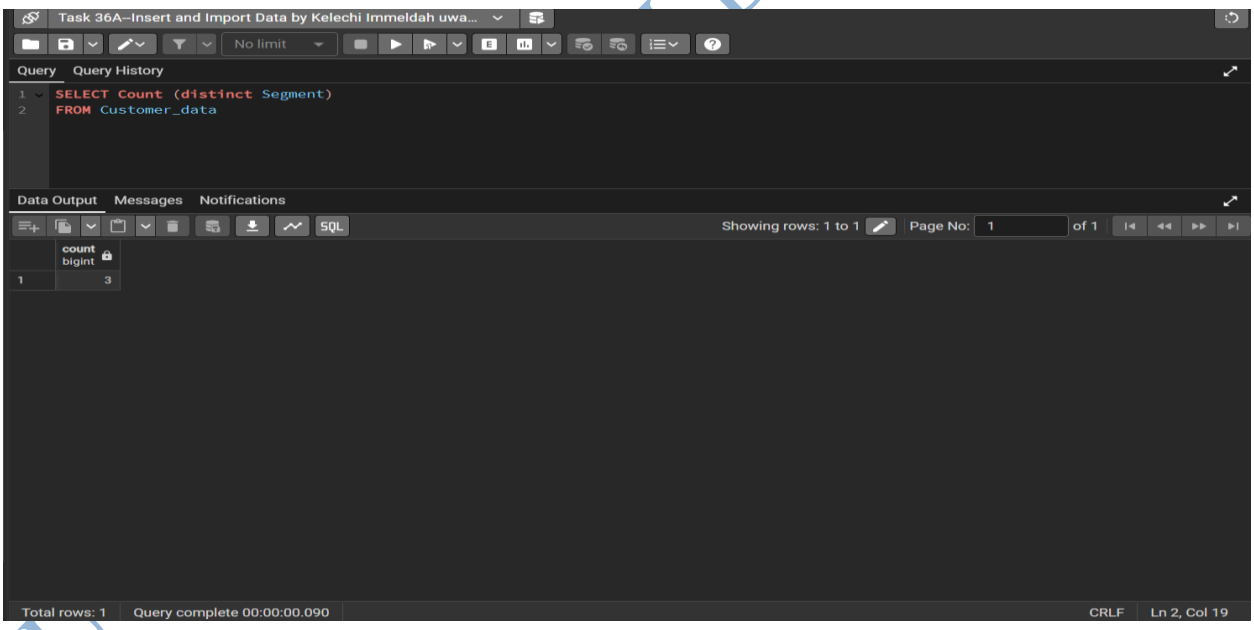


COUNT DISTINCT

By using the DISTINCT keyword in a function called COUNT, we can return the number of different countries.

Example:

SELECT COUNT (DISTINCT segment) FROM Customer data;



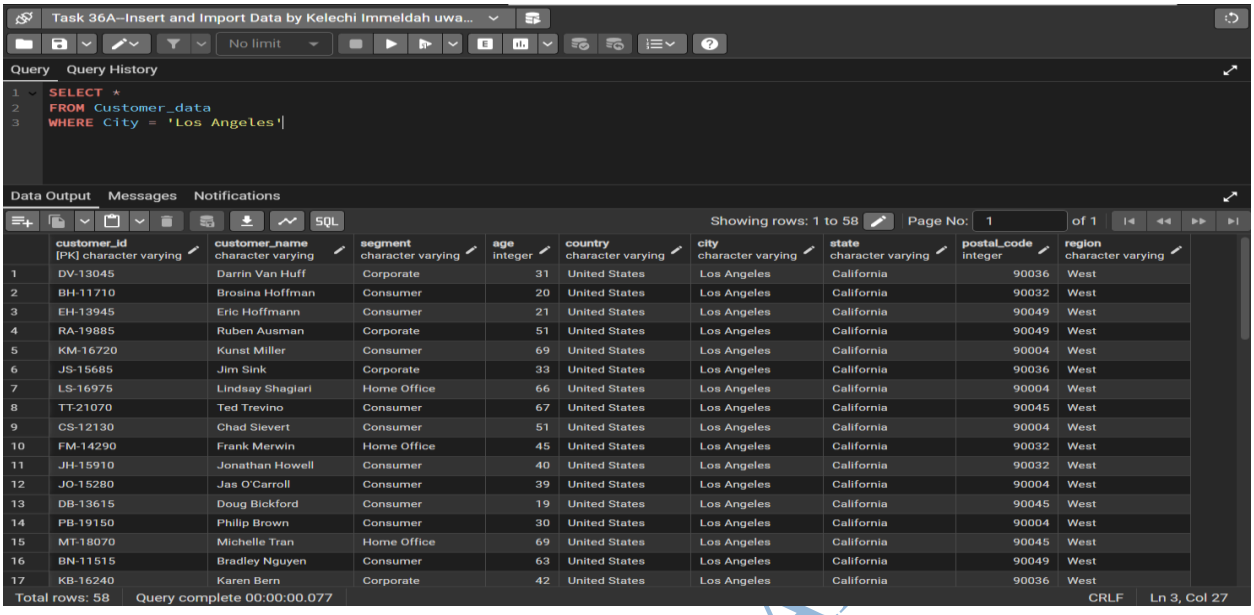
THE SQL WHERE CLAUSE

The WHERE clause is used to filter records. It is used to extract only those records that fulfill a specified condition.

Syntax

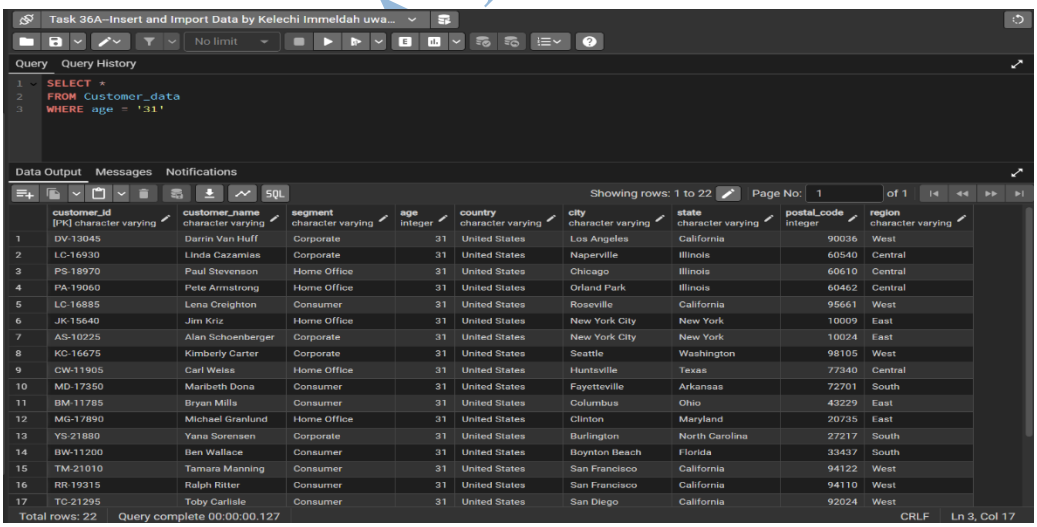
SELECT *column1*, *column2, ...*
FROM *table_name*
WHERE *condition*;

Example: Select all customers from California:
SELECT * FROM Customer_data
WHERE City = 'Los Angeles';



Text Fields vs. Numeric Fields
SQL requires single quotes around text values (most database systems will also allow double quotes). However, numeric fields should not be enclosed in quotes:

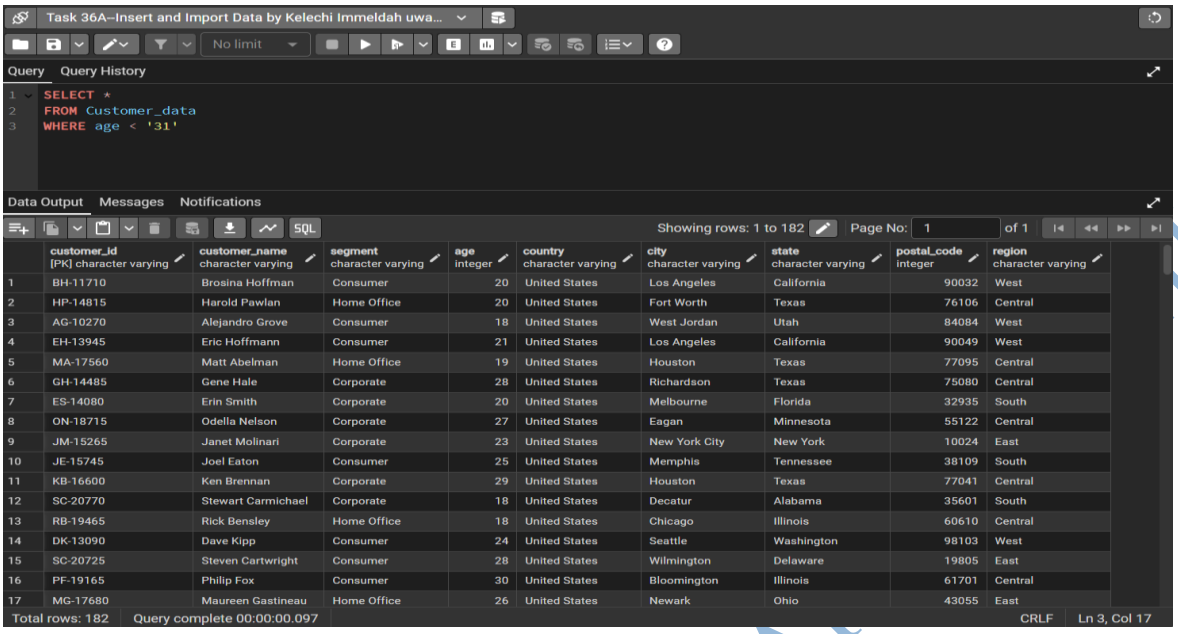
Example:
SELECT *
FROM customer_data
WHERE age = '31'



Operators in the WHERE Clause
You can use other operators than the = operator to filter the search.

Example: Select all customers with an age greater than 40:

SELECT * FROM Customers
WHERE age < 31;



	customer_id [PK] character varying	customer_name character varying	segment character varying	age integer	country character varying	city character varying	state character varying	postal_code integer	region character varying
1	BH-11710	Brosina Hoffman	Consumer	20	United States	Los Angeles	California	90032	West
2	HP-14815	Harold Pawlan	Home Office	20	United States	Fort Worth	Texas	76106	Central
3	AG-10270	Alejandro Grove	Consumer	18	United States	West Jordan	Utah	84084	West
4	EH-13945	Eric Hoffmann	Consumer	21	United States	Los Angeles	California	90049	West
5	MA-17560	Matt Abelman	Home Office	19	United States	Houston	Texas	77095	Central
6	GH-14485	Gene Hale	Corporate	28	United States	Richardson	Texas	75080	Central
7	ES-14080	Erin Smith	Corporate	20	United States	Melbourne	Florida	32935	South
8	ON-18715	Odella Nelson	Corporate	27	United States	Eagan	Minnesota	55122	Central
9	JM-15265	Janet Molinari	Corporate	23	United States	New York City	New York	10024	East
10	JE-15745	Joel Eaton	Consumer	25	United States	Memphis	Tennessee	38109	South
11	KB-16600	Ken Brennan	Corporate	29	United States	Houston	Texas	77041	Central
12	SC-20770	Stewart Carmichael	Corporate	18	United States	Decatur	Alabama	35601	South
13	RB-19465	Rick Bensley	Home Office	18	United States	Chicago	Illinois	60610	Central
14	DK-13090	Dave Kipp	Consumer	24	United States	Seattle	Washington	98103	West
15	SC-20725	Steven Cartwright	Consumer	28	United States	Wilmington	Delaware	19805	East
16	PF-19165	Philip Fox	Consumer	30	United States	Bloomington	Illinois	61701	Central
17	MG-17680	Maureen Gastineau	Home Office	26	United States	Newark	Ohio	43055	East

The following operators can be used in the WHERE clause:

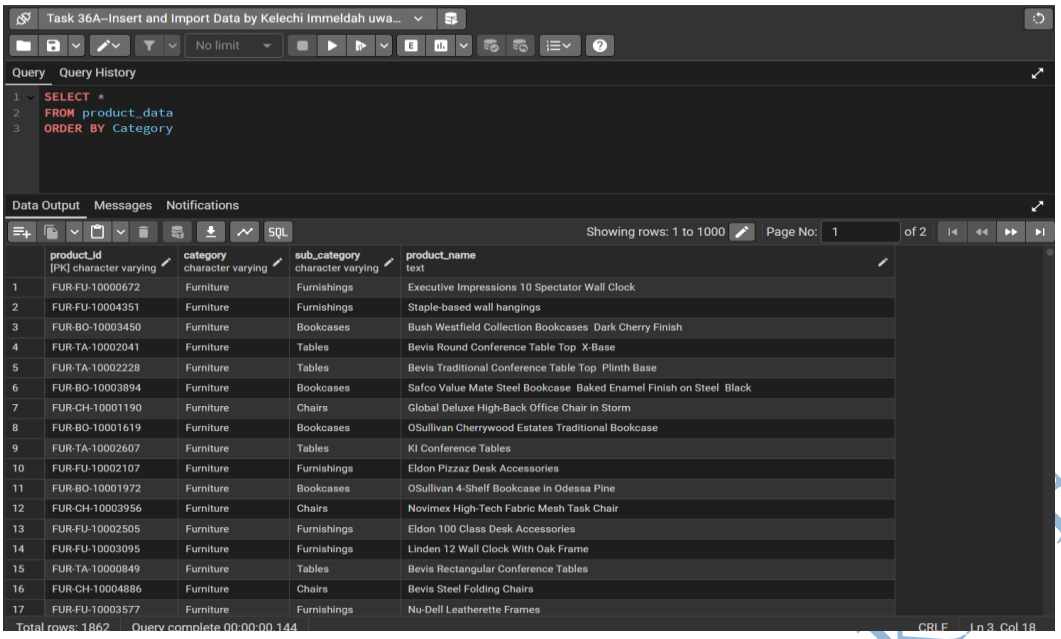
a. = Equal
b. Greater than
c. < Less than
d. >= Greater than or equal
e. <= Less than or equal
f. Not equal. Note: In some versions of SQL this operator may be written as (! =)
g. BETWEEN ---- Used between a certain range
h. LIKE --- Search for a pattern
i. IN ----- To specify multiple possible values for a column.

THE SQL ORDER BY

The ORDER BY keyword is used to sort the result set in ascending or descending order.

Syntax
SELECT *column1*, *column2, ...*
FROM *table_name*
ORDER BY *column1, column2, ...* ASC|DESC;

Example: Sort the product data by category of products, using the product data table.
SELECT * FROM Product_data
ORDER BY Category.



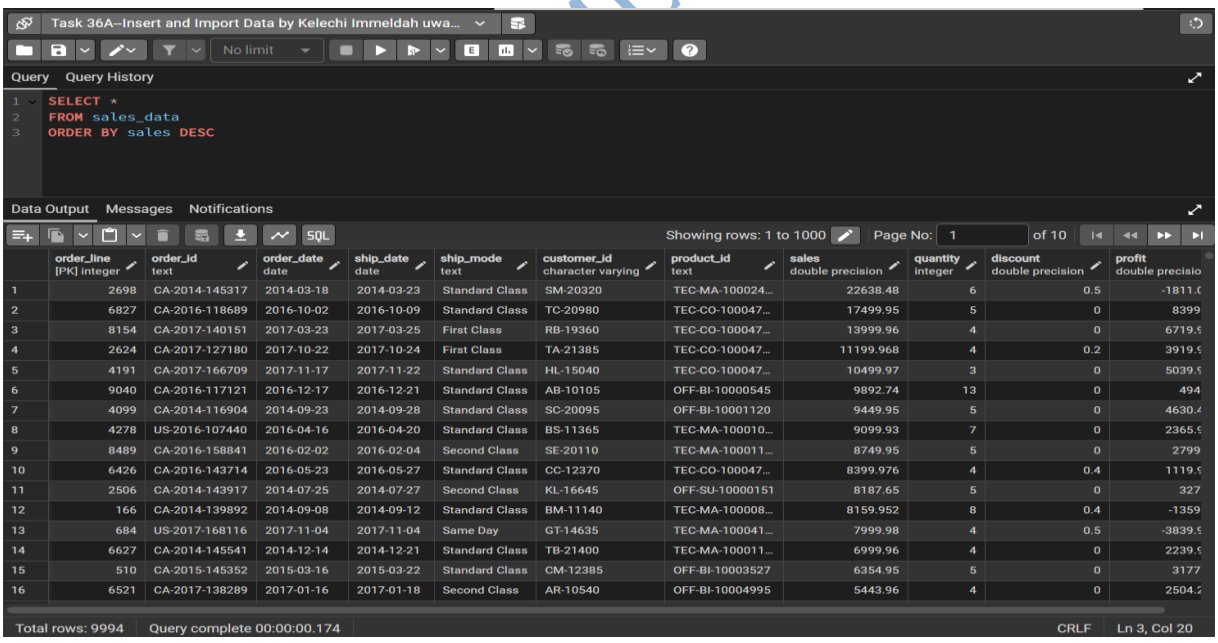
ASC AND DESC

The ORDER BY keyword sorts the records in ascending order by default, that is from A-Z. To sort the records in descending order Z-A, use the DESC keyword.

Example: sort the sales from highest to lowest.

SELECT * FROM sales_data

ORDER BY sales DESC;

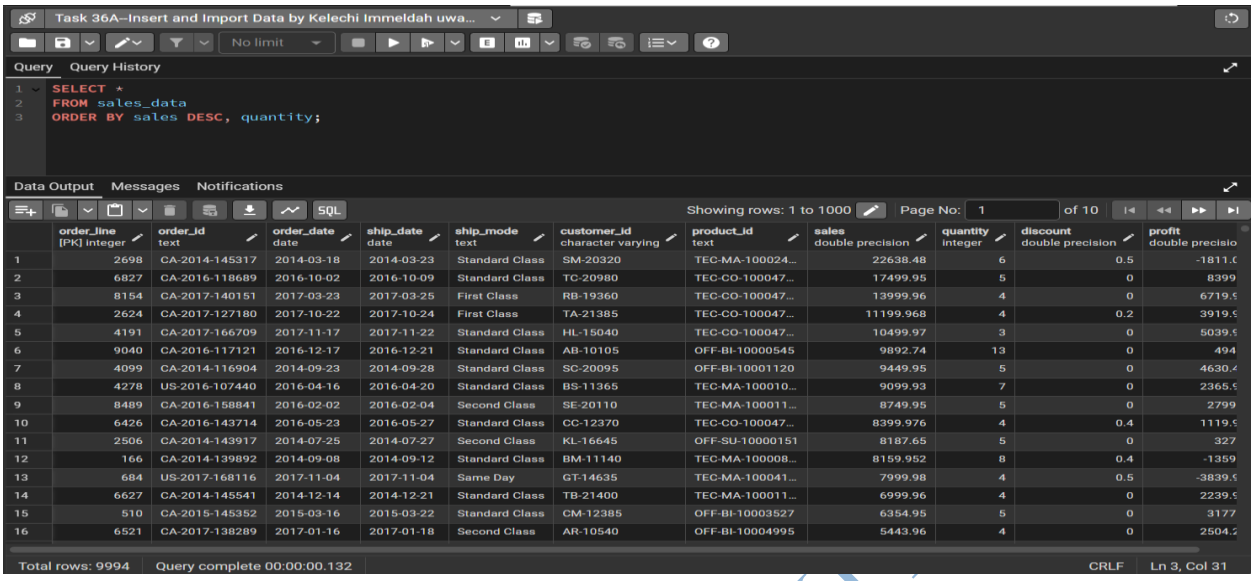


ORDER BY SEVERAL COLUMNS

When sorting by several columns in SQL, you can include multiple columns in the ORDER BY clause, separated by commas.

Example: The query below selects all fields from sales data and ORDER BY sales in descending order and quantity by default order.

SELECT * FROM sales_data
ORDER BY sales DESC, quantity;



SQL AND OPERATOR

The WHERE clause can contain one or many AND operators.

The AND operator is used to filter records based on more than one condition, like if you want to return all customers from the city that start with the letter 'S':

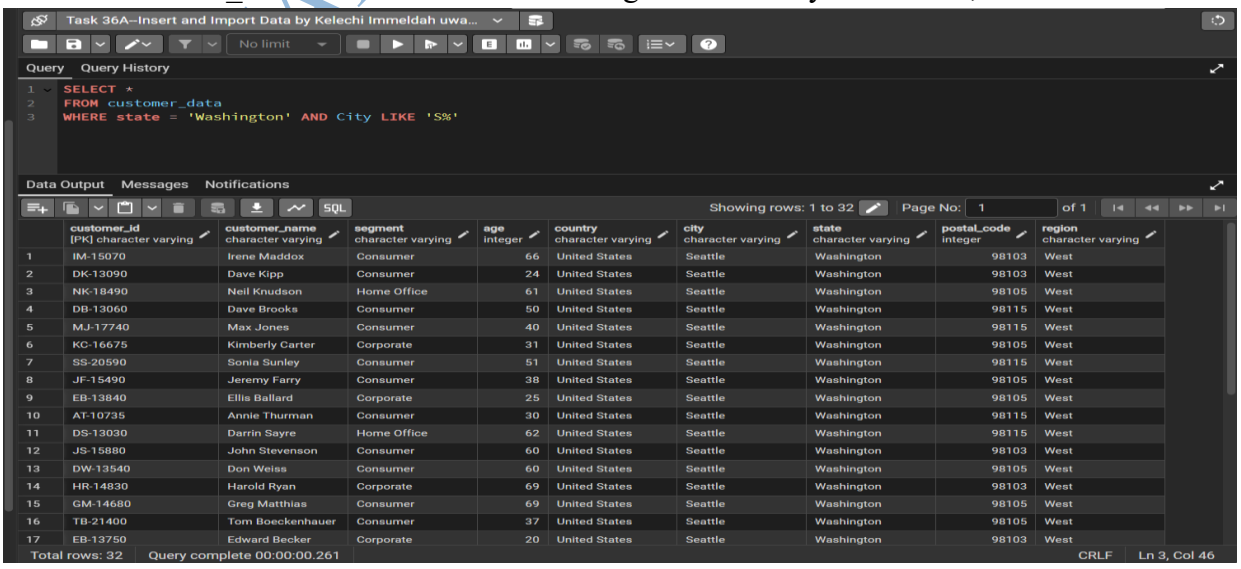
SYNTAX

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

EXAMPLE

Select all customers from the city that start with the letter 'S':

SELECT *
FROM Customer_data WHERE State = 'Washington' AND City LIKE 'S%';



AND VS OR

The AND operator displays a record of whether all the conditions are TRUE.

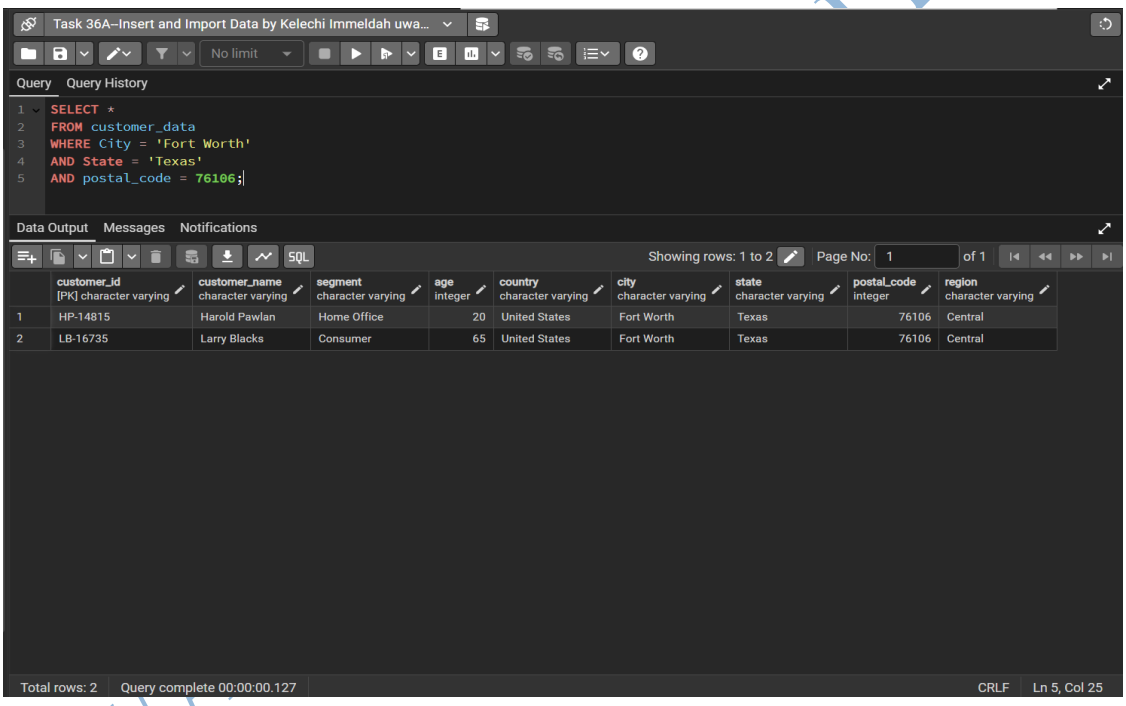
The OR operator displays a record of whether any of the conditions are TRUE.

ALL CONDITIONS MUST BE TRUE

The following SQL statement selects all fields from Customer_data where City is "Fort Worth" AND State is "Texas" AND Postal_Code is equal to 76106:

EXAMPLE

```
SELECT *
FROM Customer_data
WHERE City = 'Fort Worth'
AND State = 'Texas'
AND Postal_code = 76106;
```



COMBINING 'AND' AND 'OR'

You can combine the AND and OR operators.

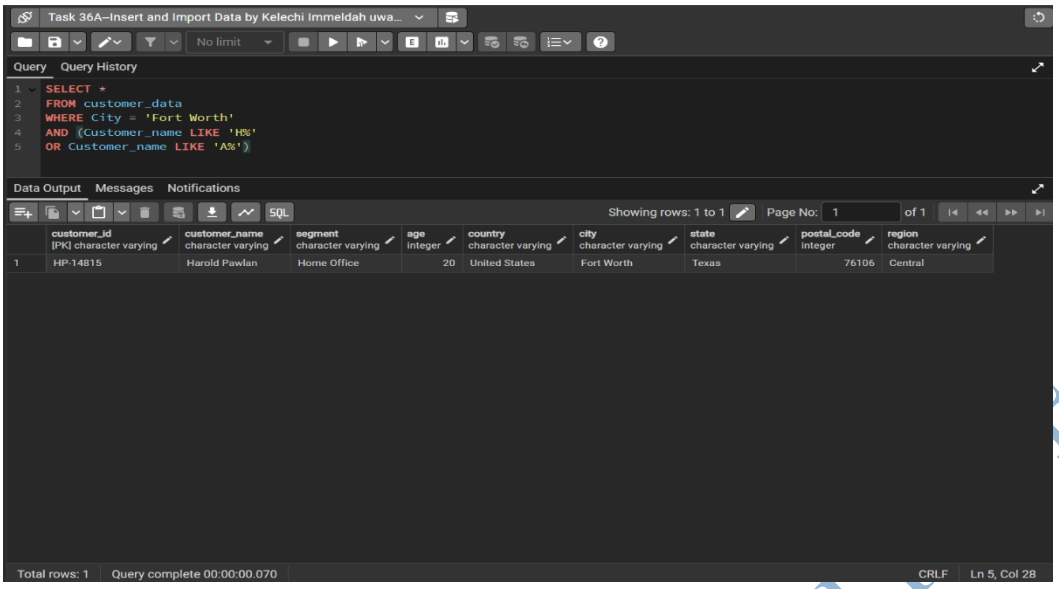
The following SQL statement selects all customers from the city that start with a "G" or an "R".

Make sure you use parenthesis to get the correct result.

Example
Select all Cities that start with either "H" or "A":

```
SELECT * FROM Customer_data
```

WHERE City = ‘Fort Worth' AND (Customer_name LIKE 'H%' OR Customer_name LIKE 'A%');



Without parenthesis, the select statement will return all customers from Fort Worth that starts with an "H", plus all customers that start with an "A", regardless of the country value:

SQL OR OPERATOR

SQL OR OPERATOR

The WHERE clause can contain one or more OR operators.
The OR operator is used to filter records based on more than one condition, like if you want to return all customers from Germany but also those from Spain:

EXAMPLE
Select all customers from Los Angeles or Fort Worth:

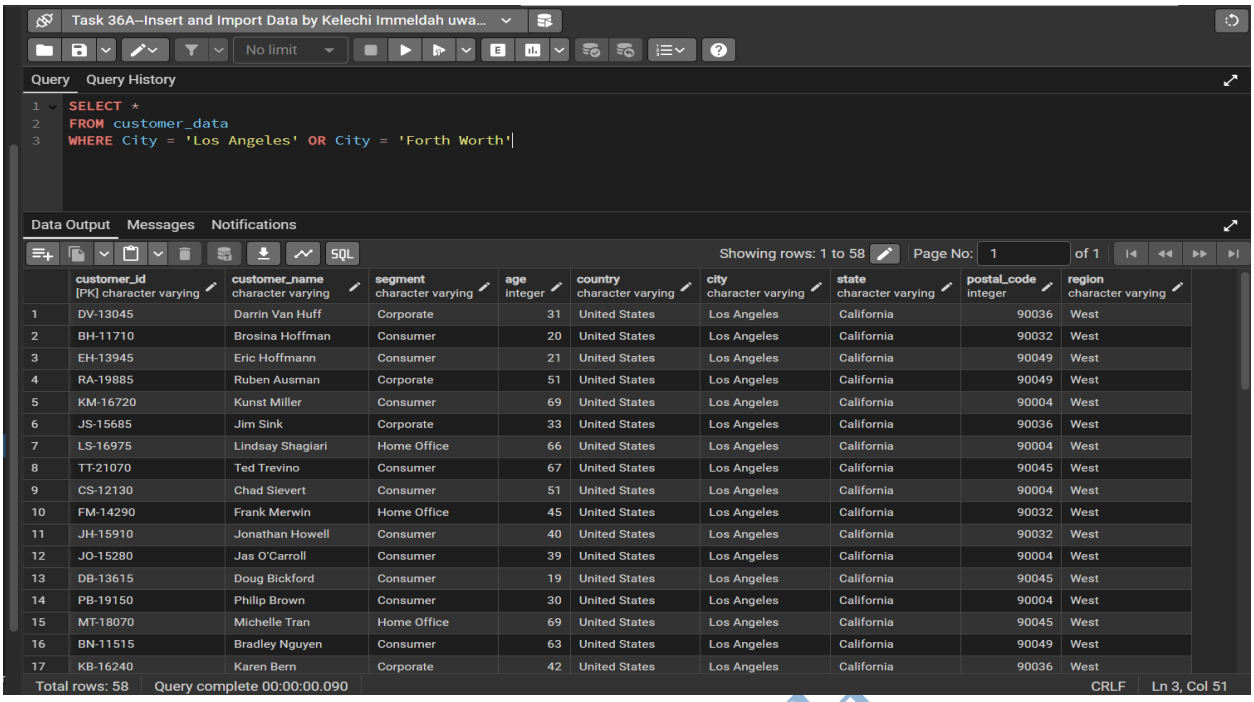
SELECT *
FROM Customer_data
WHERE City = 'Los Angeles' OR City = 'Fort Worth'.

SYNTAX

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

OR vs AND

The OR operator displays a record if *any* of the conditions are TRUE.
The AND operator displays a record if *all* the conditions are TRUE.



AT LEAST ONE CONDITION MUST BE TRUE

The following SQL statement selects all fields from Customers where either the city is "Los Angeles", the Customer Name starts with the letter "H" or the city is "Fort Worth":
Example
SELECT * FROM Customers
WHERE City = 'Los Angeles' OR Customer Name LIKE 'A%' OR City = 'Fort Worth';

COMBINING 'AND' AND 'OR'

You can combine the AND and OR operators.
The following SQL statement selects all customers from the city that start with an "L" or an "F". Make sure you use parenthesis to get the correct result.

Example
Select all customers that start with either "H" or "A":
SELECT * FROM Customers
WHERE City= 'LosAngeles' AND (Customer_Name LIKE 'H%' OR Customer Name LIKE 'A%')

SQL NOT OPERATOR

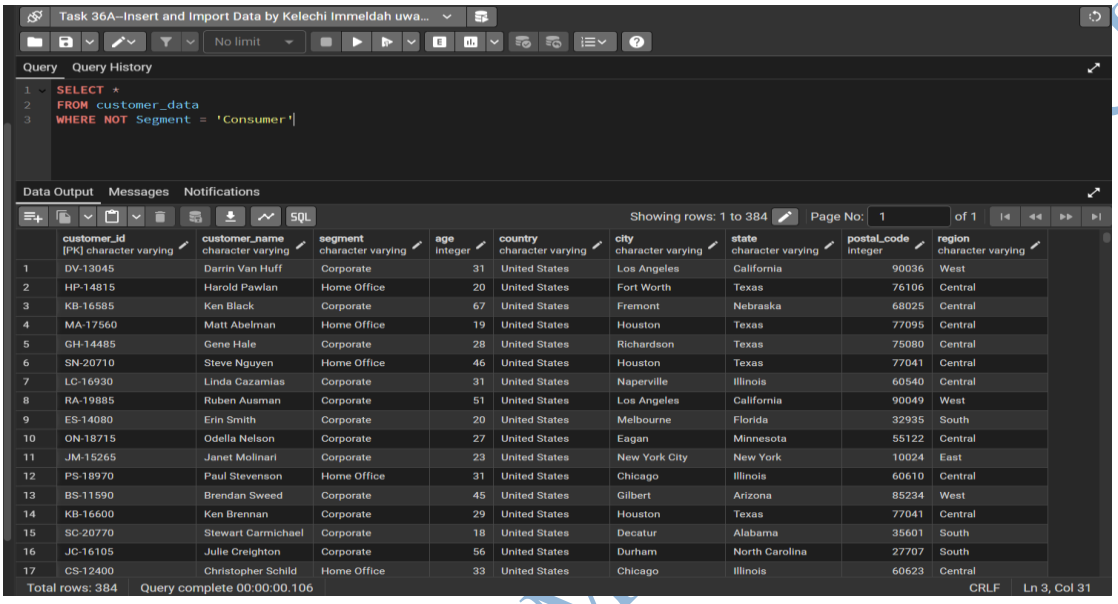
THE NOT OPERATOR

The NOT operator is used in combination with other operators to give the opposite result, also called the negative result.
In the select statement below we want to return all segments that are NOT from Consumer:

Example
Select only the Segments that are NOT from consumers:

SELECT * FROM Customer_data
WHERE NOT Segment = 'Consumer'.
In the example above, the NOT operator is used in combination with the = operator, but it can be used in combination with other comparison and/or logical operators. See examples below.

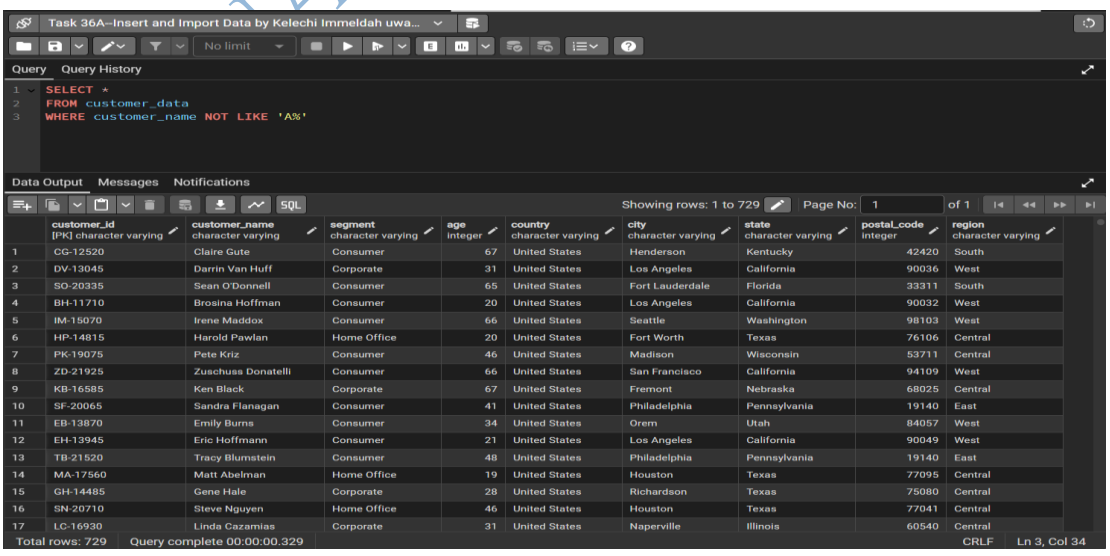
SYNTAX
SELECT column1, column2, ...
FROM table_name
WHERE NOT condition.



NOT LIKE

Example
Select customers that do not start with the letter 'A':

SELECT * FROM Customer_data
WHERE Customer_name NOT LIKE 'A%';

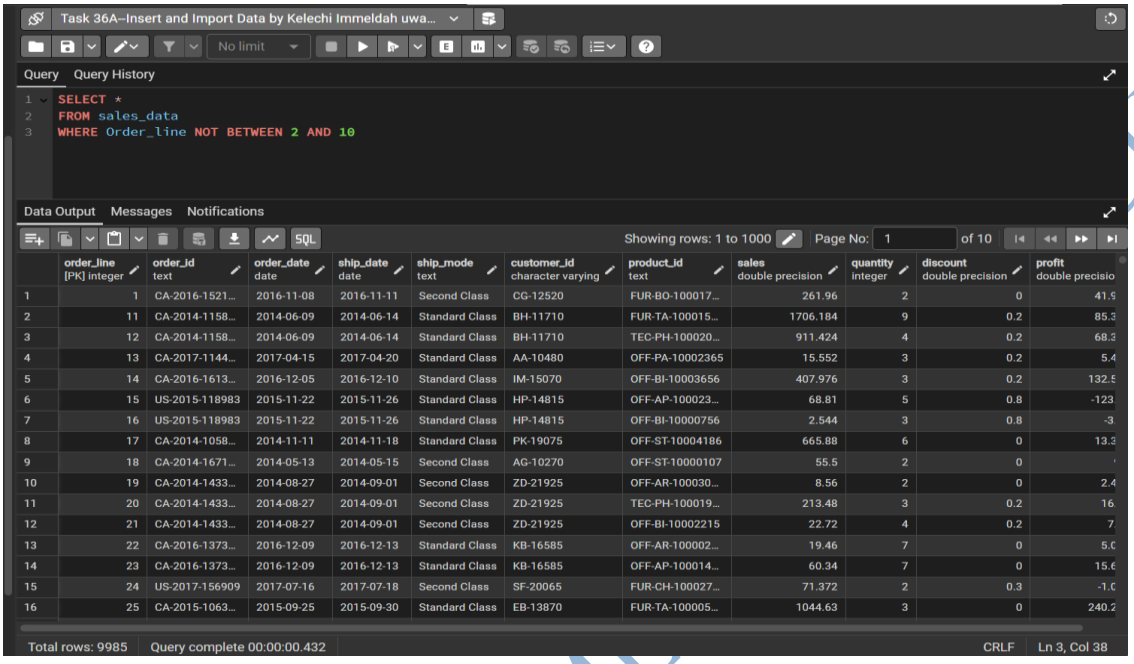


NOT BETWEEN

Example

Select customers with order_line not between 2 and 10:

```
SELECT * FROM sales_data
WHERE Order_line NOT BETWEEN 2 AND 10.
```

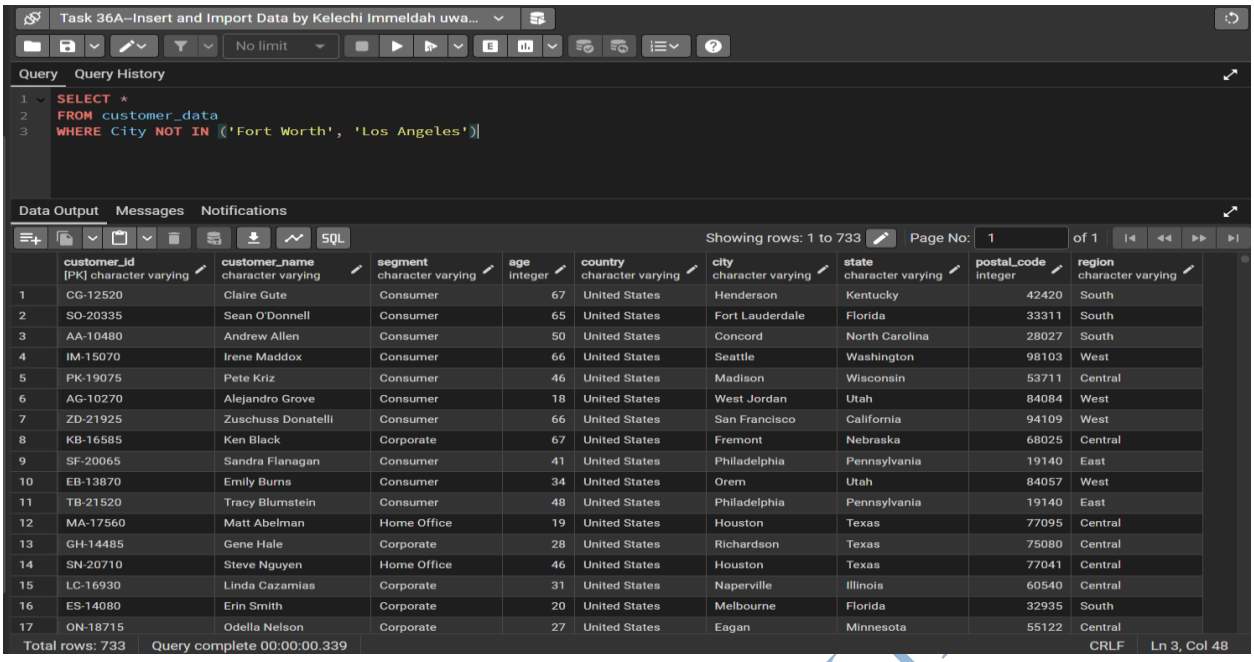


NOT IN

Example

Select customers that are not from Fort Worth or Los Angeles:

```
SELECT * FROM Customer_data
WHERE City NOT IN ('Fort Worth', 'Los Angeles');
```



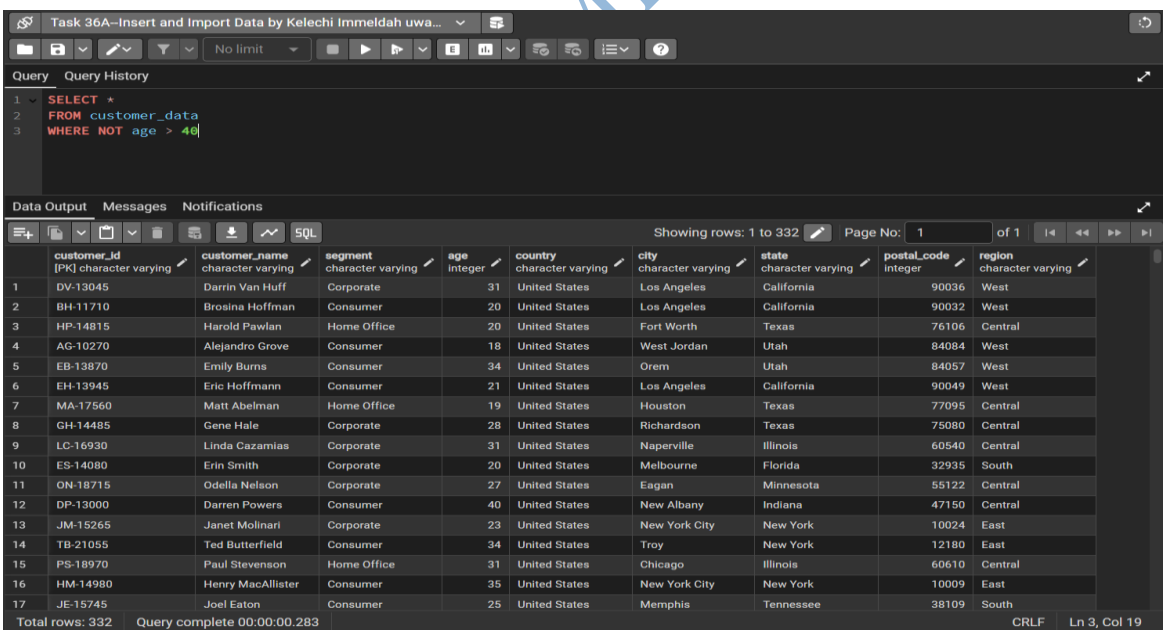
NOT GREATER THAN

Example

Selecting customers with an age not greater than 40:

SELECT * FROM Customer_data
WHERE NOT age > 40.

Note: There is a not-greater-than operator:> that would give you the same result.

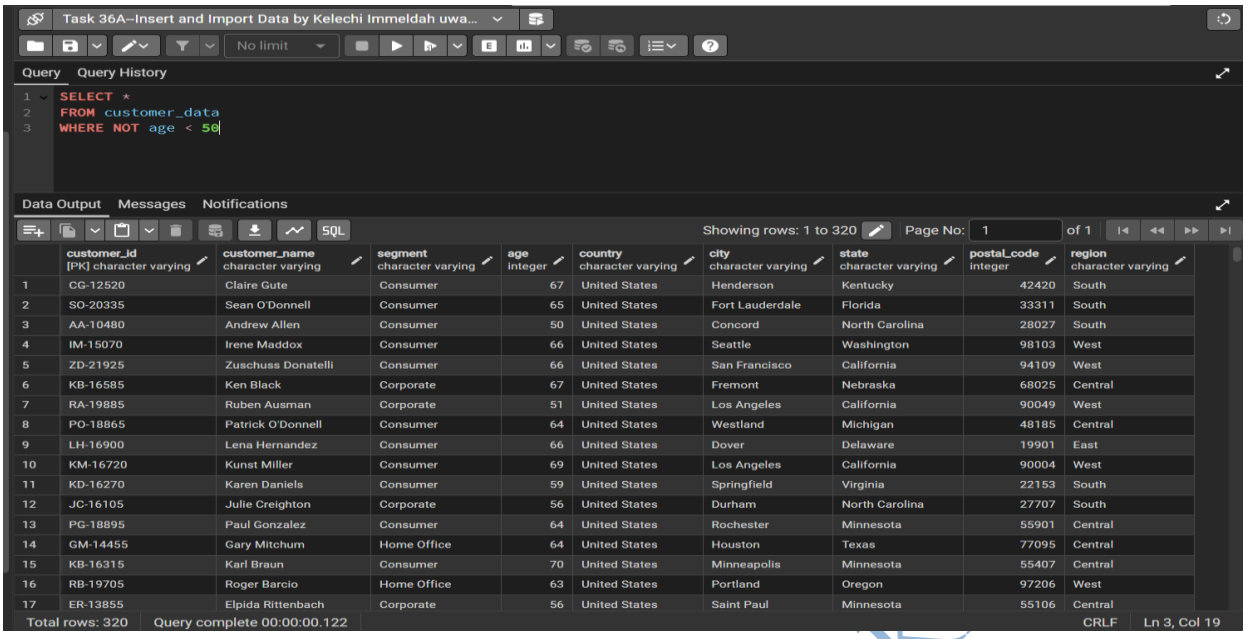


NOT LESS THAN

Example

Select customers with an age that is not less than 50:

SELECT * FROM Customers
WHERE NOT age < 50.



SQL INSERT INTO STATEMENT

THE SQL INSERT INTO STATEMENT

The INSERT INTO statement is used to insert new records in a table.

INSERT INTO SYNTAX

It is possible to write the INSERT INTO statement in two ways:

1. Specify both the column names and the values to be inserted:

INSERT INTO *table_name* (*column1*, *column2*, *column3*...)
VALUES (*value1*, *value2*, *value3*, ...).

2. If you are adding values for all the columns of the table, you do not need to specify the column names in the SQL query. However, make sure the order of the values is in the same order as the columns in the table. Here, the INSERT INTO syntax would be as follows:

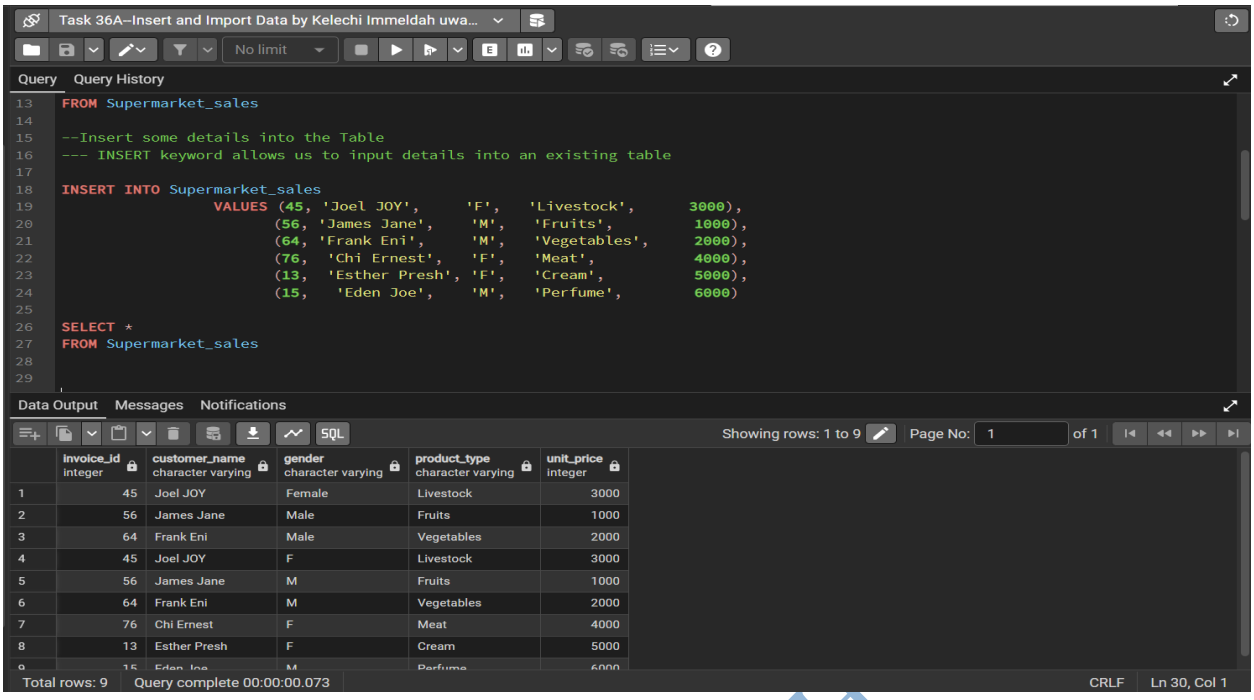
INSERT INTO *table_name*
VALUES (*value1*, *value2*, *value3*, ...).

INSERT INTO EXAMPLE

The following SQL statement inserts a new record in the "supermarket-sales" table:
EXAMPLE

```
INSERT INTO Customers  (Invoice_id,  Customer_name,  gender,  product_type,  Unit_price)
VALUES                  (45, 'Joel JOY',    'F',  'Livestock',    3000),
                        (56, 'James Jane',  'M',  'Fruits',       1000),
                        (64, 'Frank Eni',   'M',  'Vegetables',   2000),
                        (76, 'Chi Ernest',  'F',  'Meat',         4000),
```

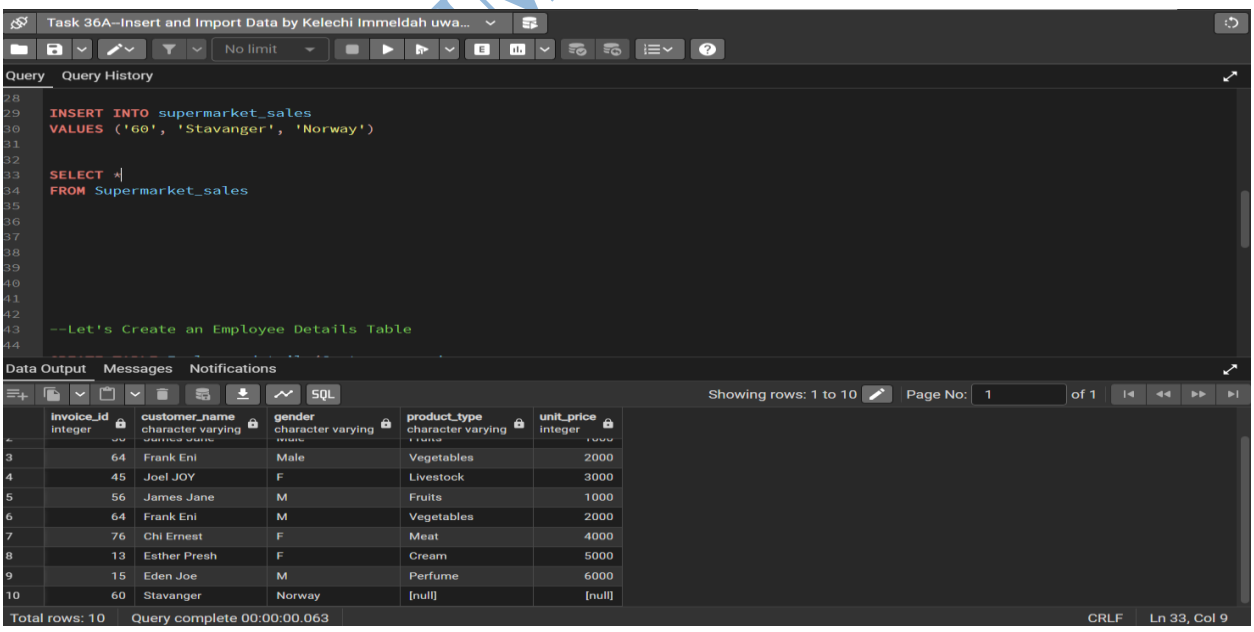
The selection from the "Customers" table will now look like this:



INSERT DATA ONLY IN SPECIFIED COLUMNS

It is also possible to only insert data in specific columns.
The following SQL statement will insert a new record, but only insert data in the "invoice_id", "Customer_name", and "Gender" columns (Supermarket_sales will be updated automatically):

Example
INSERT INTO supermarket_sales (Invoice_id, customer_name, gender,)
VALUES ('60', 'Stavanger', 'Norway');



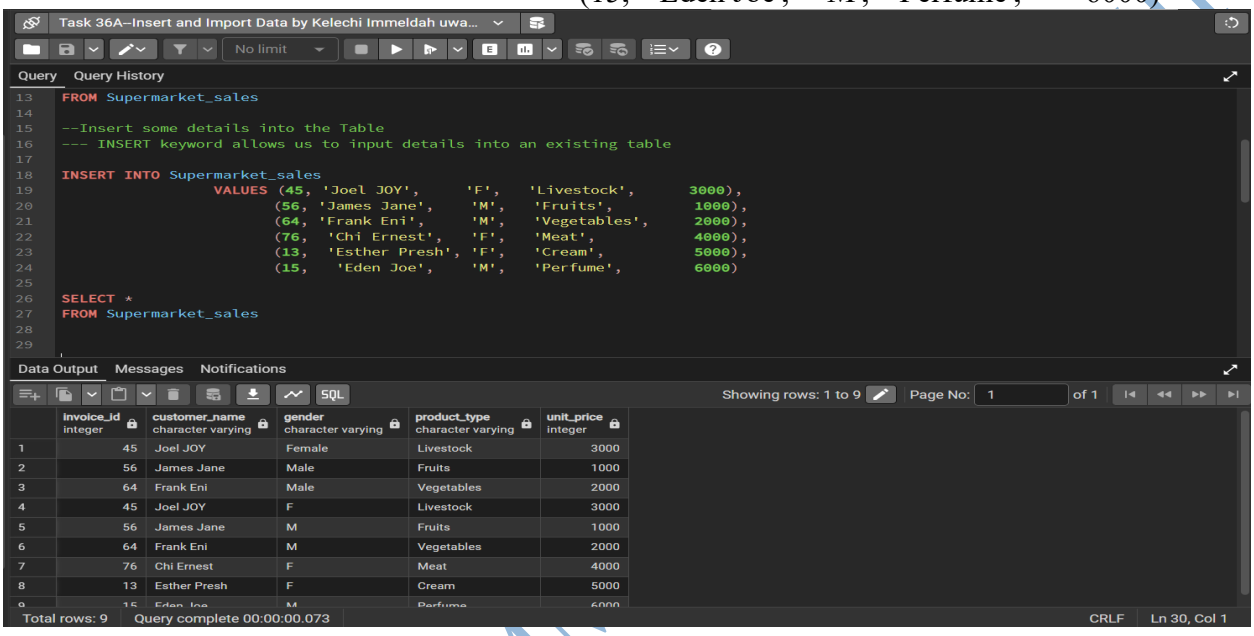
INSERT MULTIPLE ROWS

It is also possible to insert multiple rows in one statement.

To insert multiple rows of data, we use the same INSERT INTO statement, but with multiple values:

Example

```
INSERT INTO Supermarket_sales
        VALUES      (45, 'Joel JOY',     'F',   'Livestock',      3000),
                    (56, 'James Jane',   'M',   'Fruits',         1000),
                    (64, 'Frank Eni',    'M',   'Vegetables',     2000),
                    (76, 'Chi Ernest',   'F',   'Meat',           4000),
                    (13, 'Esther Presh', 'F',   'Cream',          5000),
                    (15,  'Eden Joe',    'M',   'Perfume',        6000)
```



SQL NULL VALUES

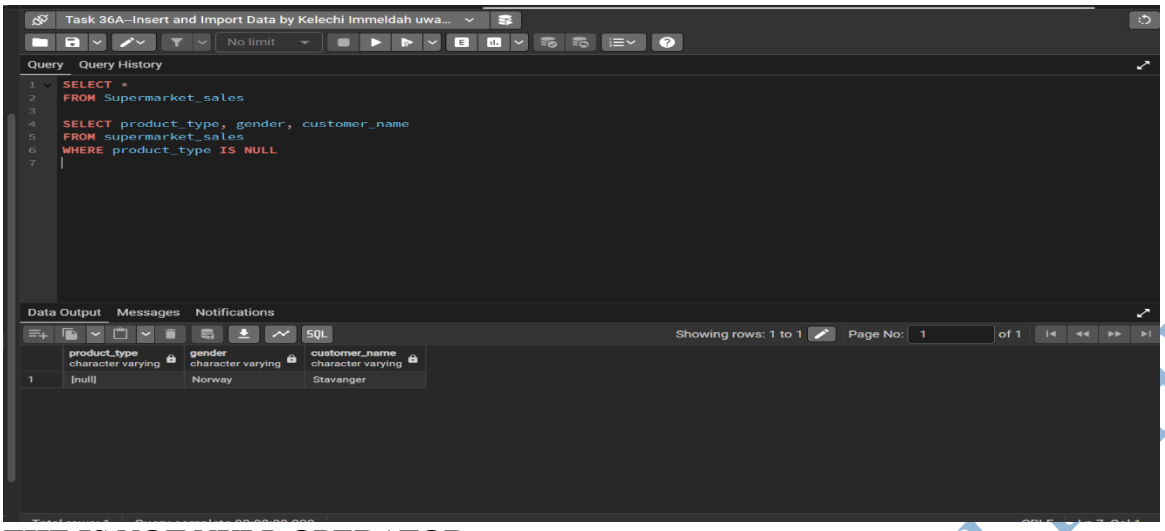
WHAT IS NULL VALUE?

A field with a NULL value is a field with no value.
If a field in a table is optional, it is possible to insert a new record or update a record without adding value to this field. Then, the field will be saved with NULL value.

Note: A NULL value is different from a zero value or a field that contains spaces. A field with a NULL value has been left blank during record creation!

Example

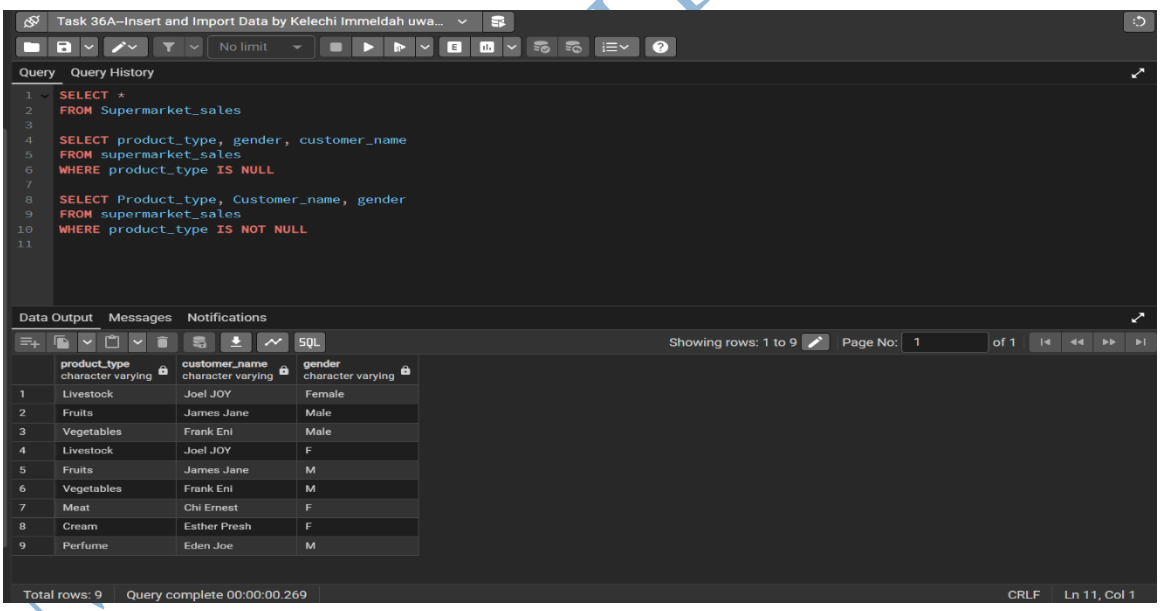
```
SELECT product_type, gender, customer_name
FROM supermarket_sales
WHERE product_type IS NULL.
```



THE IS NOT NULL OPERATOR

The IS NOT NULL operator is used to test for non-empty values (NOT NULL values).
The following SQL lists all customers with value in the "product-type" field:

Example
SELECT Product_type, Customer_name, gender
FROM supermarket_sales
WHERE product_type IS NOT NULL



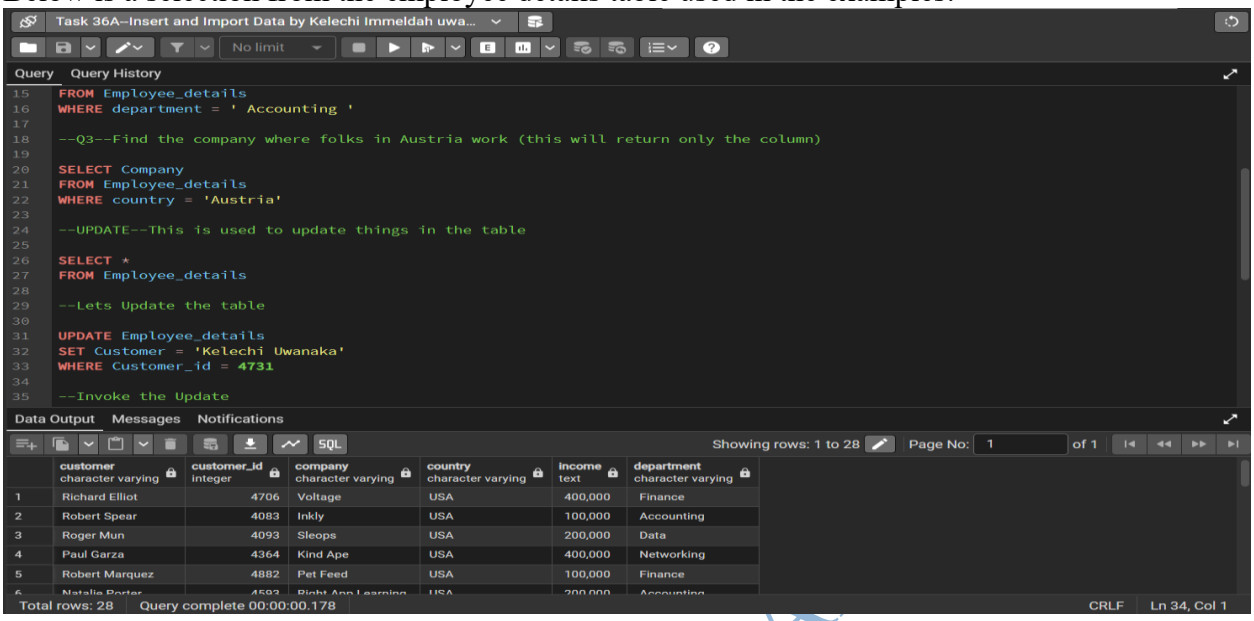
SQL UPDATE STATEMENT

THE SQL UPDATE STATEMENT

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

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

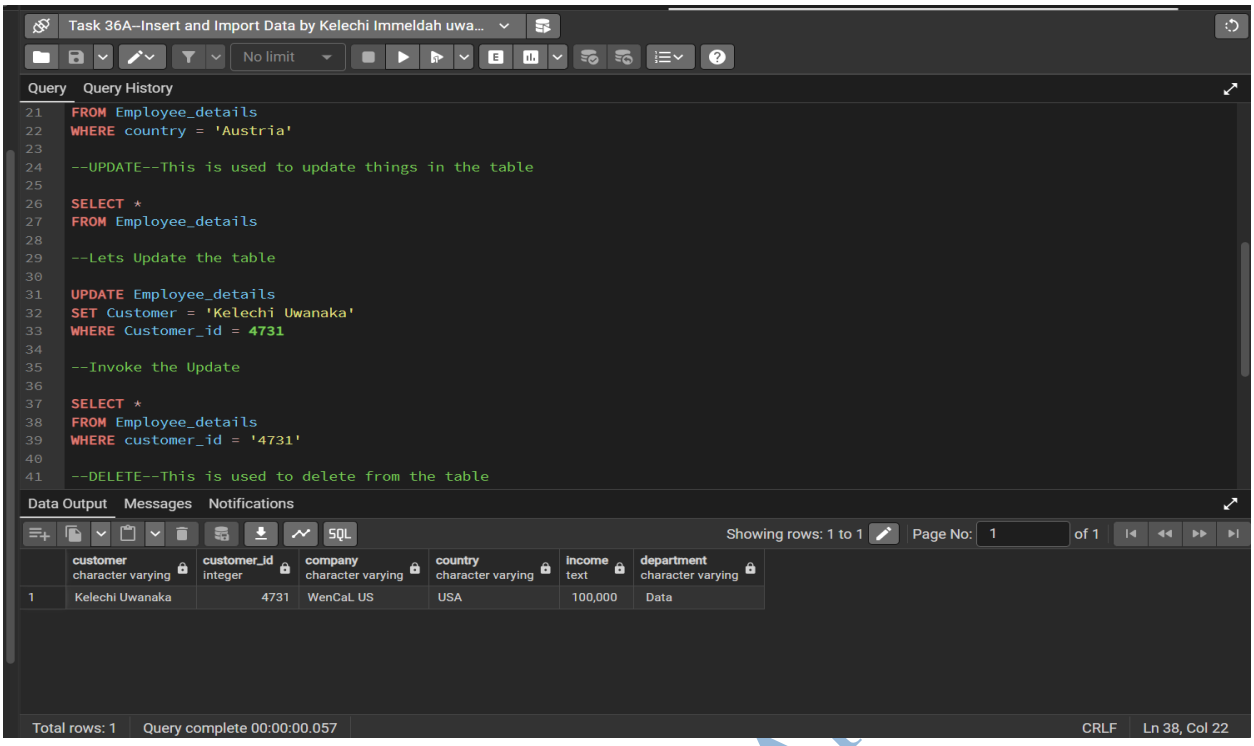
Note: Be careful when updating records in a table! Notice the WHERE clause in the UPDATE statement. The WHERE clause specifies which record(s) that should be updated. If you omit the WHERE clause, all records in the table will be updated.
Below is a selection from the employee details table used in the examples:



UPDATE TABLE

The following SQL statement updates the first customer (Customer id = 4731) with a new contact person.

EXAMPLE
UPDATE Employee_details
SET Customer = 'Kelechi Uwanaka'
WHERE Customer_id = 4731



UPDATE MULTIPLE RECORDS

It is the WHERE clause that determines how many records will be updated.

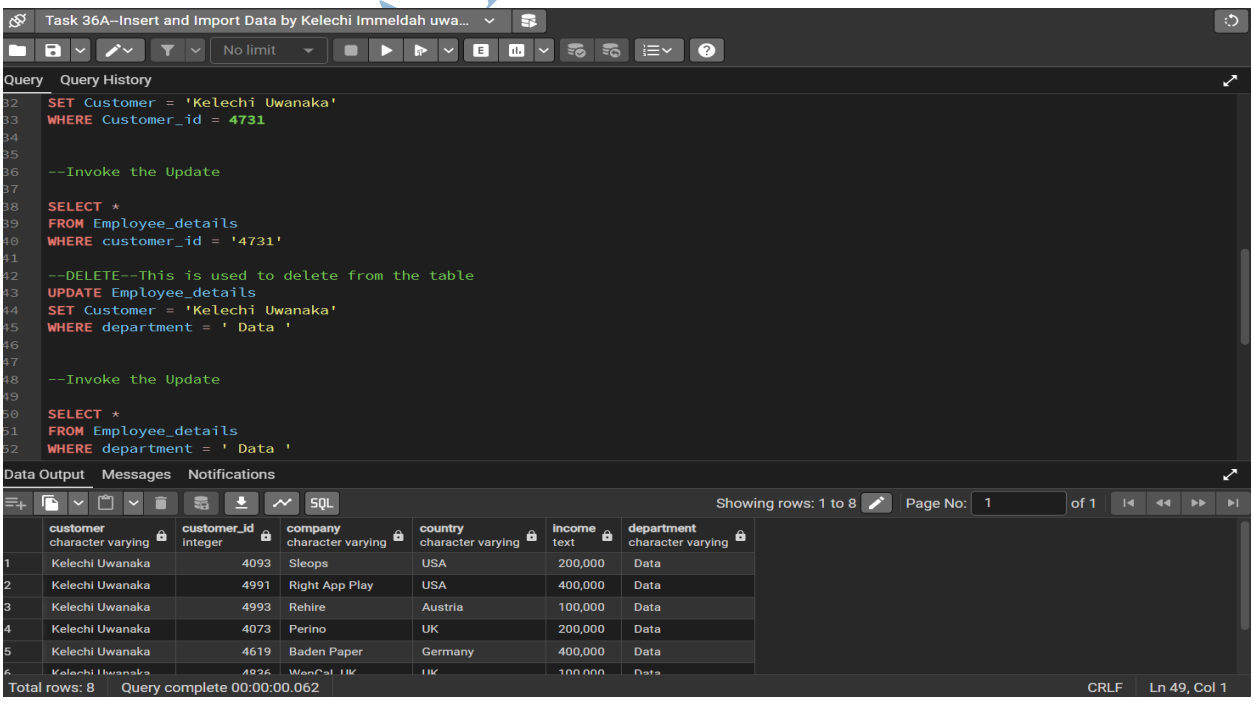
The following SQL statement will update the Customer to "Kelechi Uwanaka" for all records where the department is "Data".

EXAMPLE

UPDATE Customers

SET Customer='Kelechi Uwanaka'

WHERE Department='Data';



SQL DELETE STATEMENT

THE SQL DELETE STATEMENT

The DELETE statement is used to delete existing records in a table.

DELETE SYNTAX

DELETE FROM *table_name*
WHERE *condition*.

Note: Be careful when deleting records in a table! Notice the WHERE clause in the DELETE statement. The WHERE clause specifies which record(s) should be deleted. If you omit the WHERE clause, all records in the table will be deleted.

SQL DELETE EXAMPLE

The following SQL statement deletes the customer ID "4556" from the "Employee Details" table:
EXAMPLE
DELETE FROM Employee_details WHERE Customer_id='4556'

DELETE ALL RECORDS

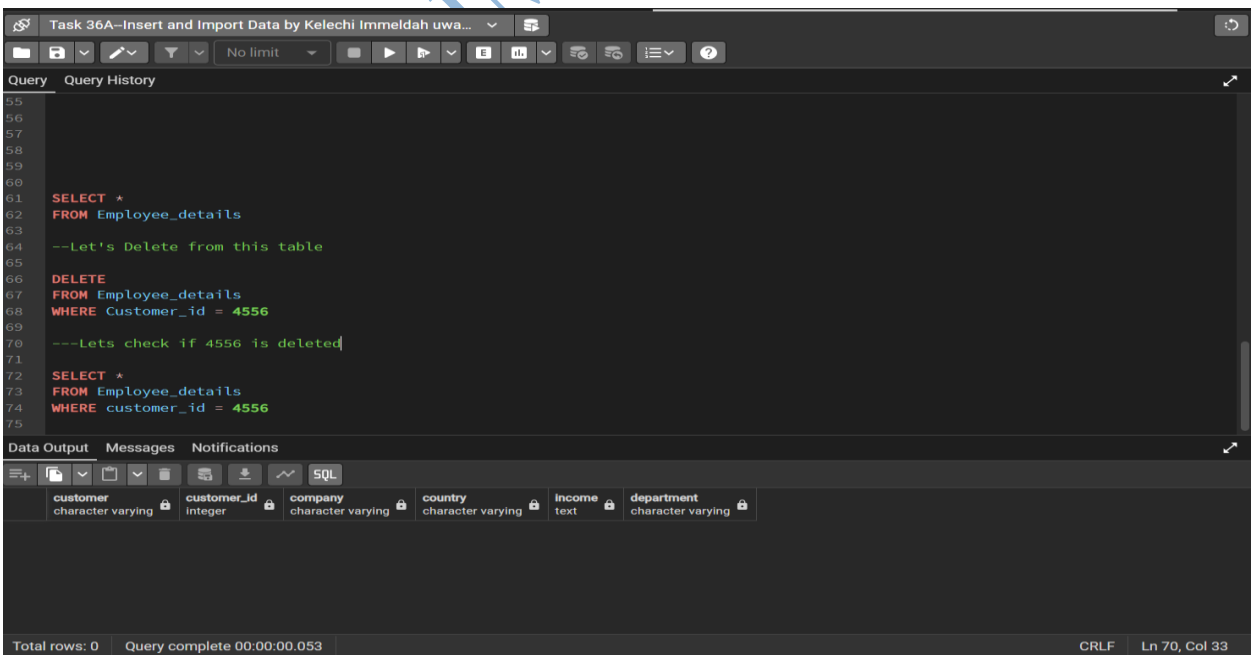
It is possible to delete all rows in a table without deleting the table. This means that the table structure, attributes, and indexes will be intact:

DELETE FROM *table_name*;

The following SQL statement deletes all rows in the "Customers" table, without deleting the table:

Example

DELETE FROM Customers.



SQL TOP, LIMIT, FETCH FIRST OR ROWNUM CLAUSE

THE SQL SELECT TOP CLAUSE

The SELECT TOP clause is used to specify the number of records to return.
The SELECT TOP clause is useful on large tables with thousands of records. Returning a large number of records can impact performance.

Example
Select only the first 3 records of the Employee details table:

```
SELECT TOP 3 *
 FROM Employee_details
```

Note: Not all database systems support the SELECT TOP clause. MySQL supports the LIMIT clause to select a limited number of records, while Oracle uses FETCH FIRST ROWS ONLY and ROWNUM.

SQL SERVER / MS ACCESS SYNTAX:

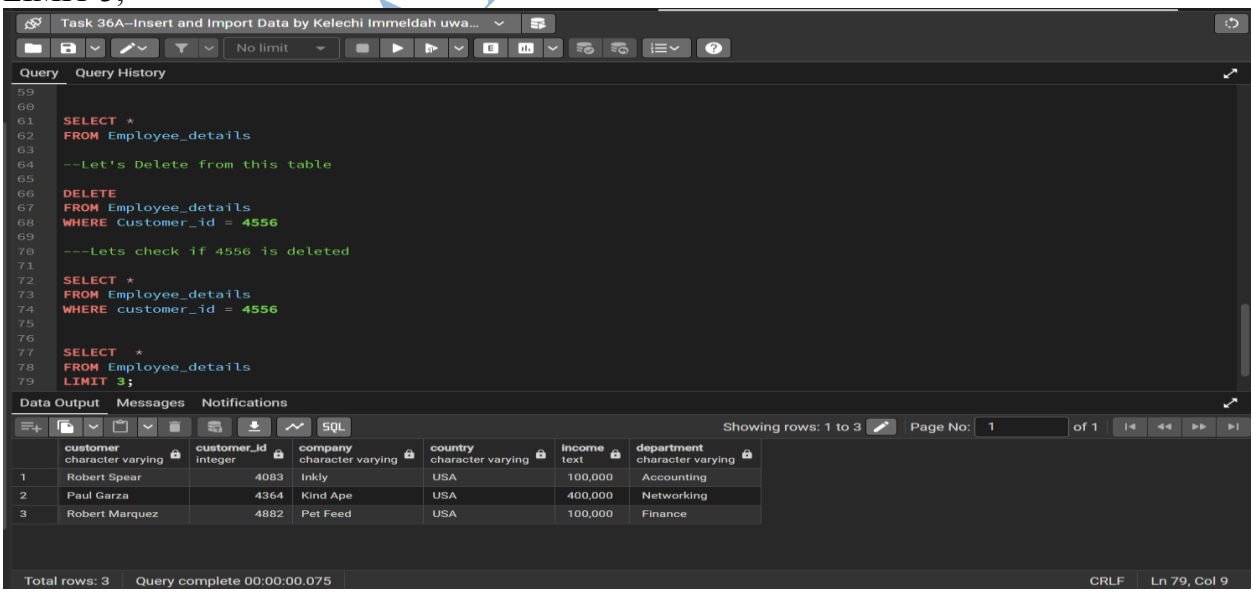
```
SELECT TOP number|percent column_name(s)
FROM table_name
WHERE condition
```

LIMIT

The following SQL statement shows the equivalent example for MySQL:

Example
Select the first 3 records of the Employee details table:

```
SELECT *
FROM Employee_details
LIMIT 3;
```

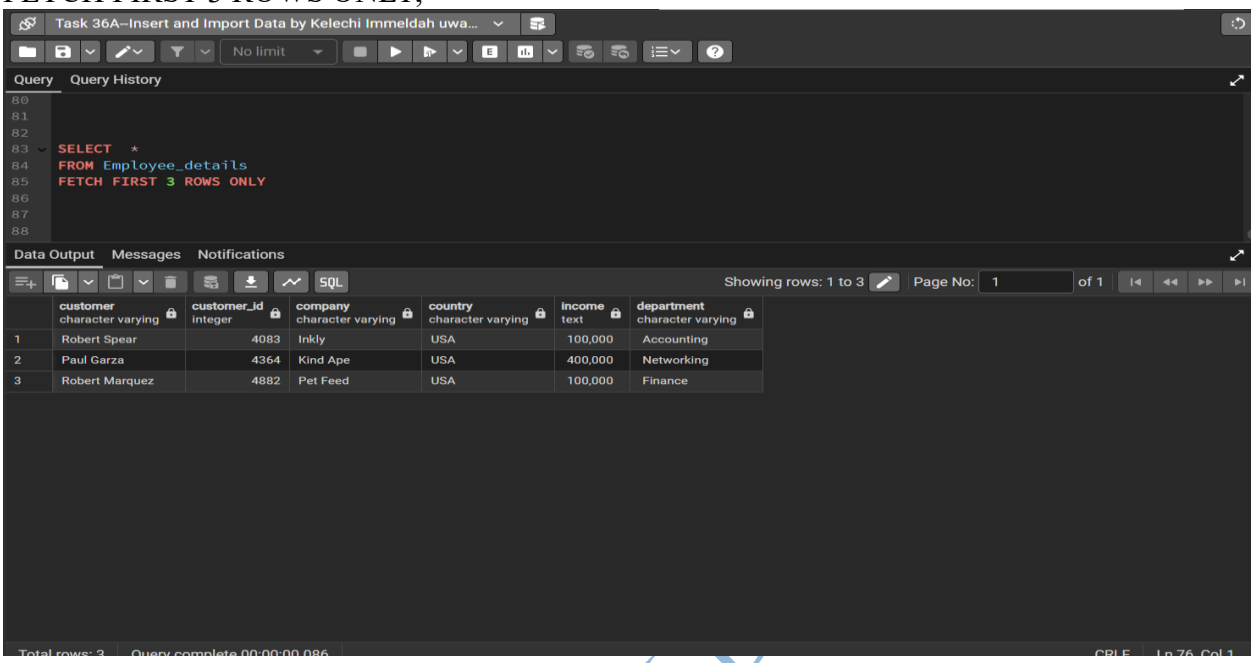


FETCH FIRST

The following SQL statement shows the equivalent example for Oracle:

Example
Select the first 3 records of the Employee details table:

```
SELECT *
FROM Employee_details
FETCH FIRST 3 ROWS ONLY;
```



SQL TOP PERCENT EXAMPLE

The following SQL statement selects the first 50% of the records from the "Employee Details" table (for SQL Server/MS Access):

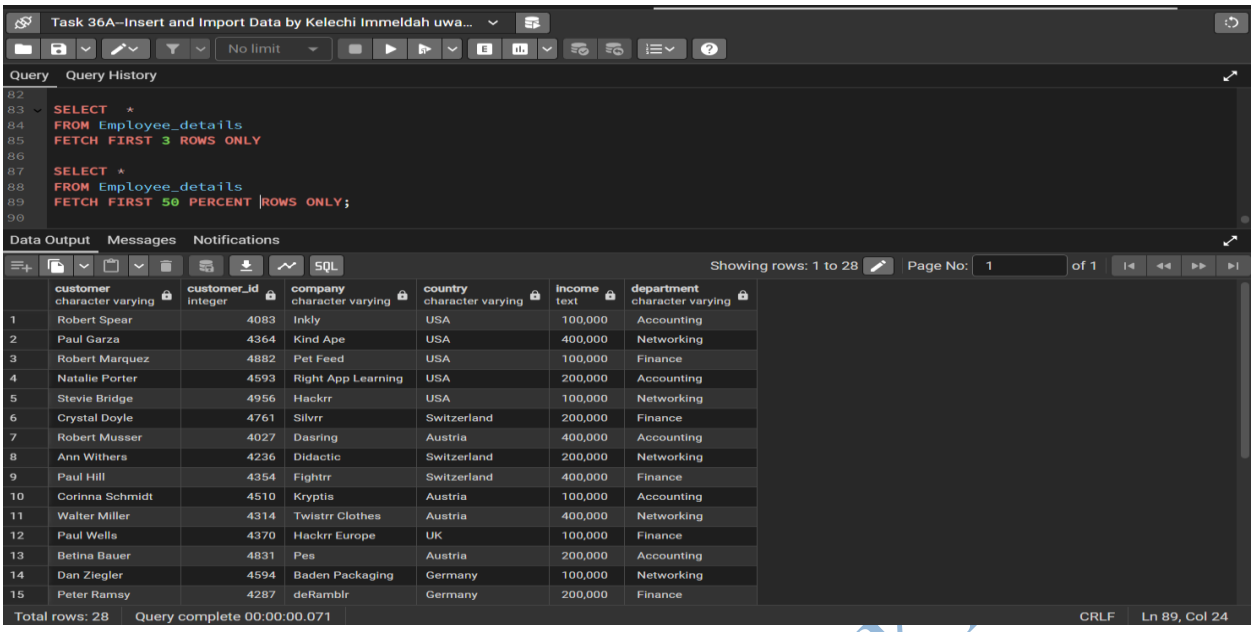
Example

```
SELECT TOP 50 PERCENT * FROM Employee_details;
```

The following SQL statement shows the equivalent example for Oracle:

Example

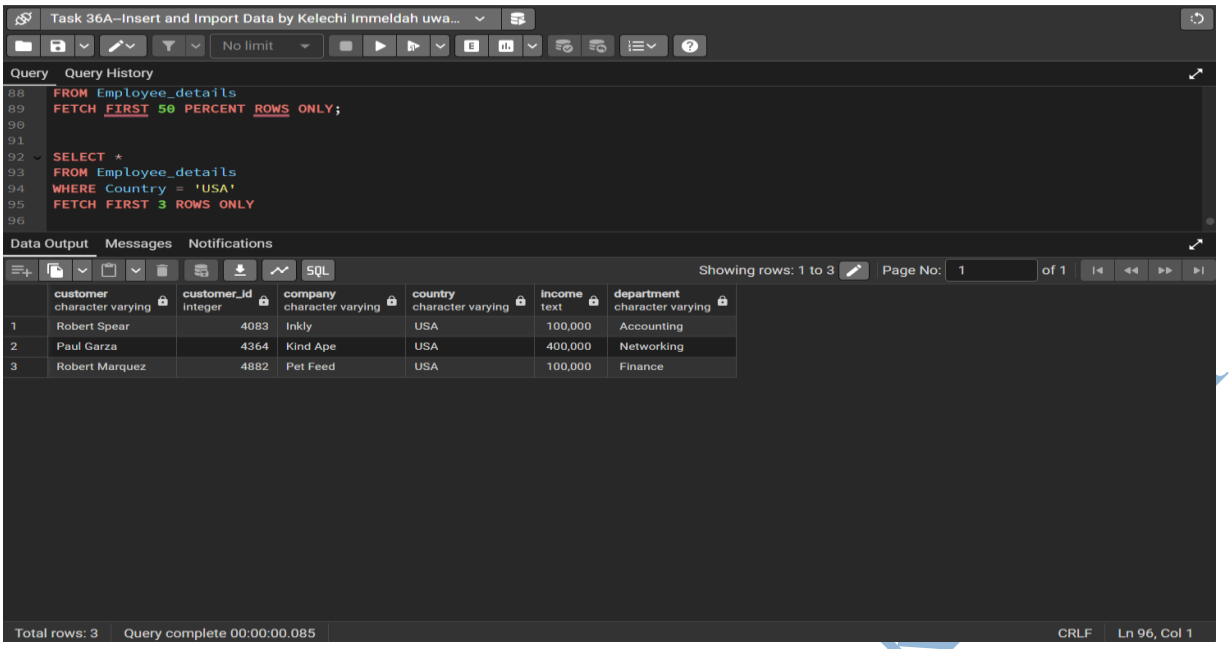
```
SELECT *
FROM Employee_details
FETCH THE FIRST 50 PERCENT ROWS ONLY.
```



ADD A WHERE CLAUSE

The following SQL statement selects the first three records from the "Customers" table, where the country is "Germany" (for SQL Server/MS Access):

The following SQL statement shows the equivalent example for MySQL:
Example
SELECT *
FROM Employee_ details
WHERE Country = 'USA'
FETCH FIRST 3 ROWS ONLY.



ADD THE ORDER BY KEYWORD

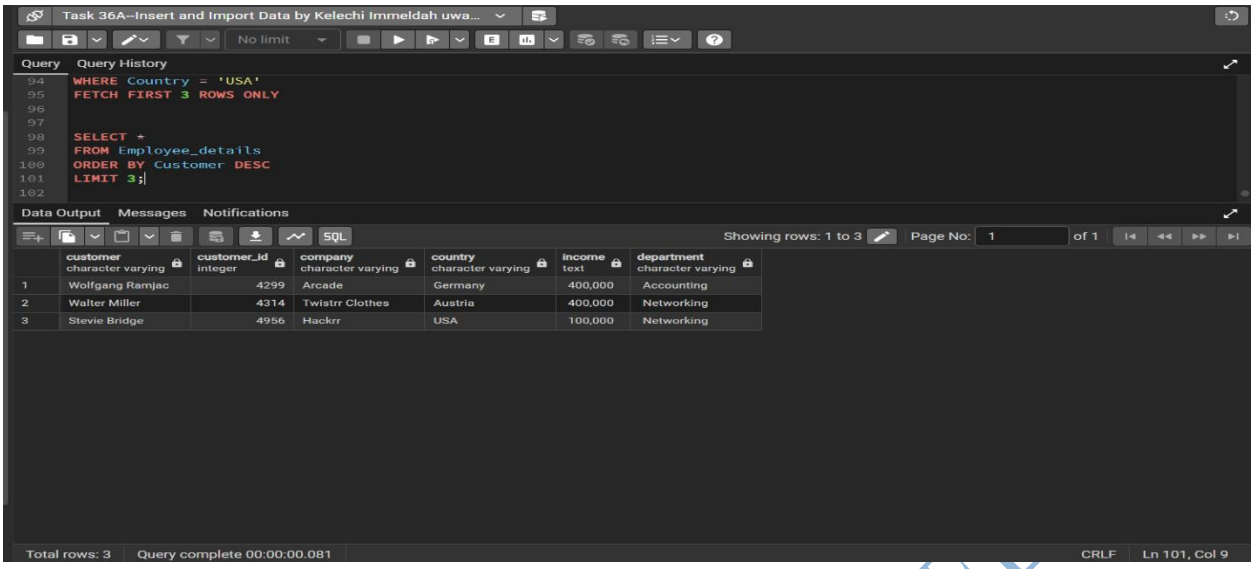
Add the ORDER BY keyword when you want to sort the result and return the first 3 records of the sorted result.
For SQL Server and MS Access:

Example
Sort the result reversed alphabetically by Customer, and return the first 3 records:

```
SELECT TOP 3 *
FROM Customers
ORDER BY Customer DESC;
```

The following SQL statement shows the equivalent example for MySQL:
Example

```
SELECT *
FROM Employee_details
ORDER BY Customer DESC
LIMIT 3;
```



SQL AGGREGATE FUNCTIONS

SQL AGGREGATE FUNCTIONS

An aggregate function is a function that performs a calculation on a set of values and returns a single value.
Aggregate functions are often used with the GROUP BY clause of the SELECT statement. The GROUP BY clause splits the result set into groups of values and the aggregate function can be used to return a single value 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

Aggregate functions ignore null values (except for COUNT ()).
We will go through the aggregate functions above in the next chapters.

SQL MIN () AND MAX () FUNCTIONS

THE SQL MIN () AND MAX () FUNCTIONS

The MIN () function returns the smallest value of the selected column.
The MAX () function returns the largest value of the selected column.

MIN EXAMPLE
Find the lowest price in the Price column:

```
SELECT MIN(Price)
FROM Sales_data;
```

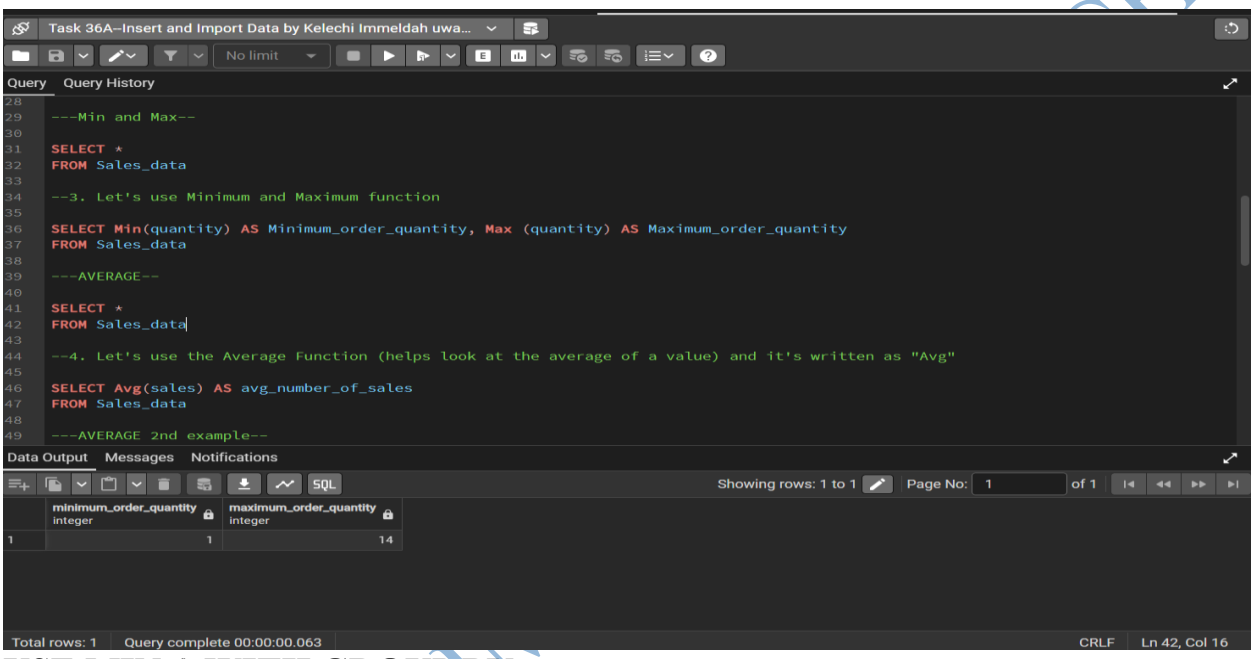
MAX EXAMPLE
Find the highest price in the Price column:

SELECT MAX(Price)
FROM Sales_data;

SET COLUMN NAME (ALIAS)

When you use MIN () or MAX (), the returned column will not have a descriptive name. To give the column a descriptive name, use the AS keyword:

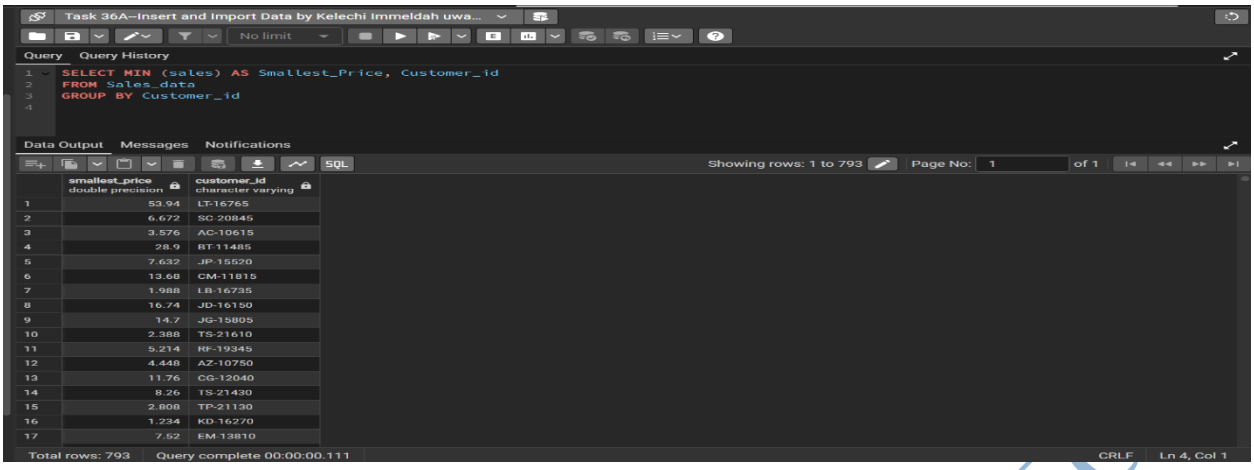
Example
SELECT MIN (sales) AS Smallest_Price
FROM sales_data



USE MIN () WITH GROUP BY

Here we use the MIN () function and the GROUP BY clause, to return the smallest price for each category in the Sales_data table:

Example
SELECT MIN (sales) AS Smallest_Price, Customer_id
FROM Sales_data
GROUP BY Customer_id



SQL COUNT () FUNCTION

THE SQL COUNT () FUNCTION

The COUNT () function returns the number of rows that match a specified criterion.

Example
Find the total number of rows in the Sales_data table:

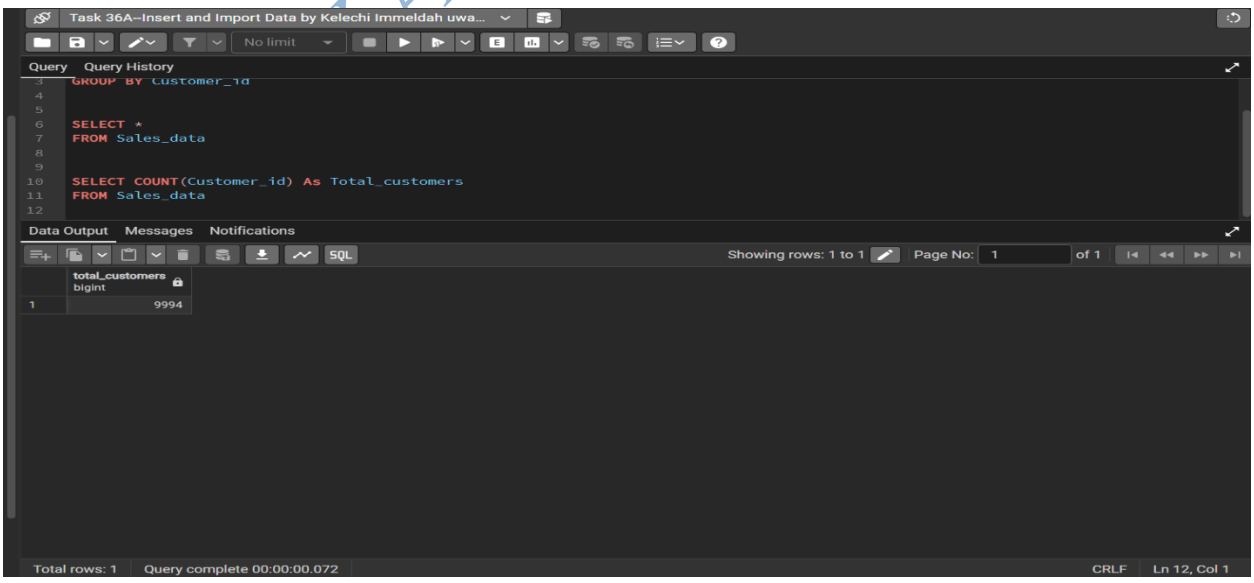
```
SELECT COUNT (*)
FROM Sales_data.
```

SPECIFY COLUMN

You can specify a column name instead of the asterisk symbol (*).
If you specify a column name instead of (*), NULL values will not be counted.

Example
Find the number of Customers where the Customer_id is not null:

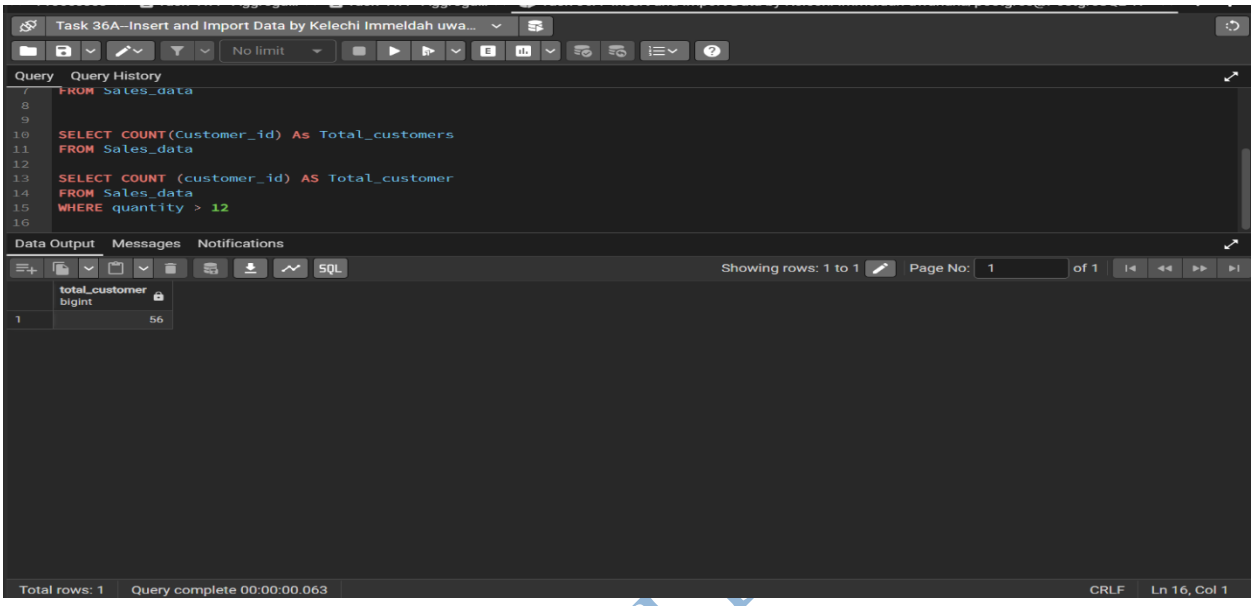
```
SELECT COUNT(Customer_id) As Total_Customers
FROM Sales_data.
```



ADD A WHERE CLAUSE

You can add a WHERE clause to specify conditions:

Example
Find the number of Customers where the quantity is higher than 12:
SELECT COUNT (customer_id) AS Total_Customer
FROM Sales_data
WHERE quantity > 12



IGNORE DUPLICATES

You can ignore duplicates by using the DISTINCT keyword in the COUNT () function.
If DISTINCT is specified, rows with the same value for the specified column will be counted as one.
Example
How many *different* prices are there in the sales_data table:
SELECT COUNT (DISTINCT quantity)
FROM sales_data

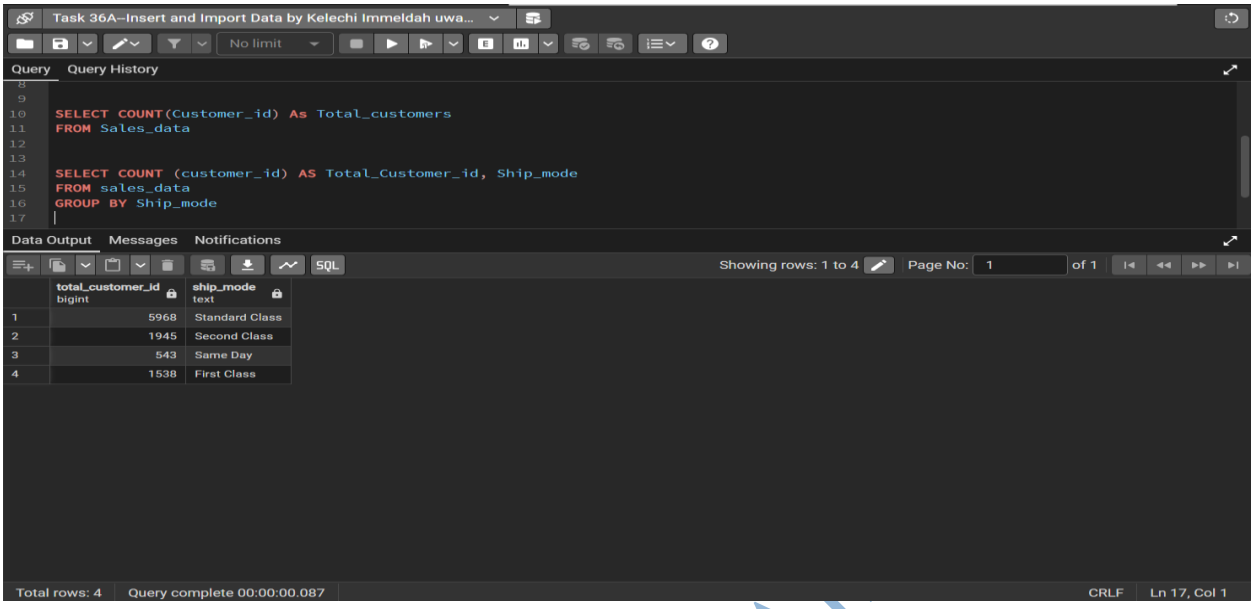
USE AN ALIAS

Give the counted column a name by using the AS keyword.
Example
Name the column "Number of records":
SELECT COUNT (*) AS [Number of records]
FROM sales_data

USE COUNT () WITH GROUP BY

Here we use the COUNT () function and the GROUP BY clause, to return the number of records for each category in the Products table:
Example

SELECT COUNT (customer_id) AS Total_Customer_id, Ship_mode
FROM sales_data
GROUP BY Ship_mode



SQL SUM () FUNCTION

THE SQL SUM () FUNCTION

The SUM () function returns the total sum of a numeric column.

Example
Return the sum of all Quantity fields in the sales_data table:
SELECT SUM(Quantity)
FROM sales_data

ADD A WHERE CLAUSE

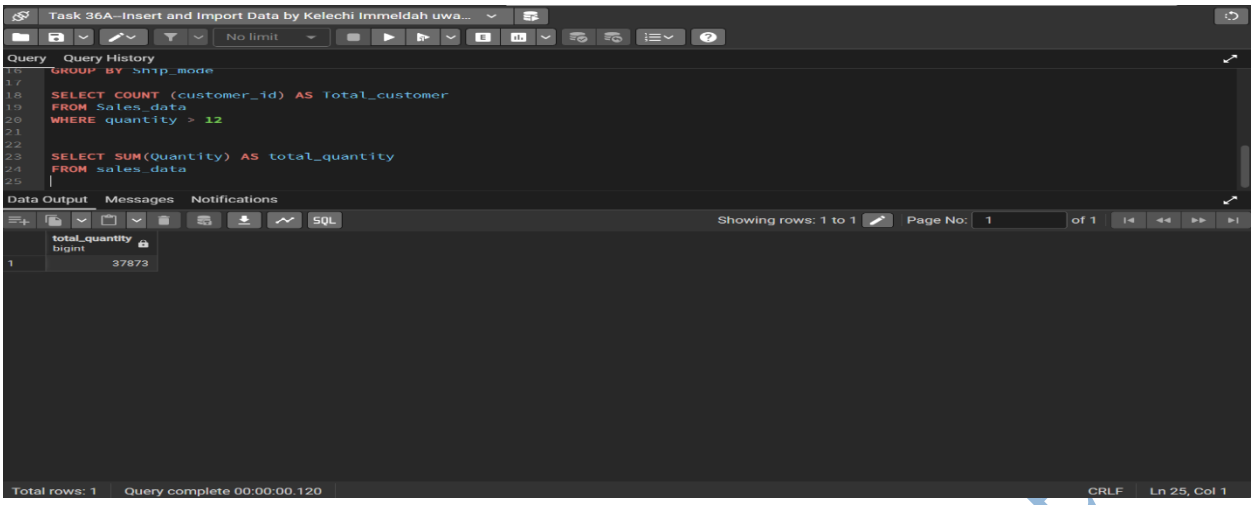
You can add a WHERE clause to specify conditions:

Example
Return the sum of the Quantity field for the customer with customer ID 11:
SELECT SUM(Quantity)
FROM sales_data
WHERE order_line = 11

USE AN ALIAS

Give the summarized column a name by using the AS keyword.

Example
Name the column "total":
SELECT SUM(Quantity) AS total_quantity
FROM sales_data

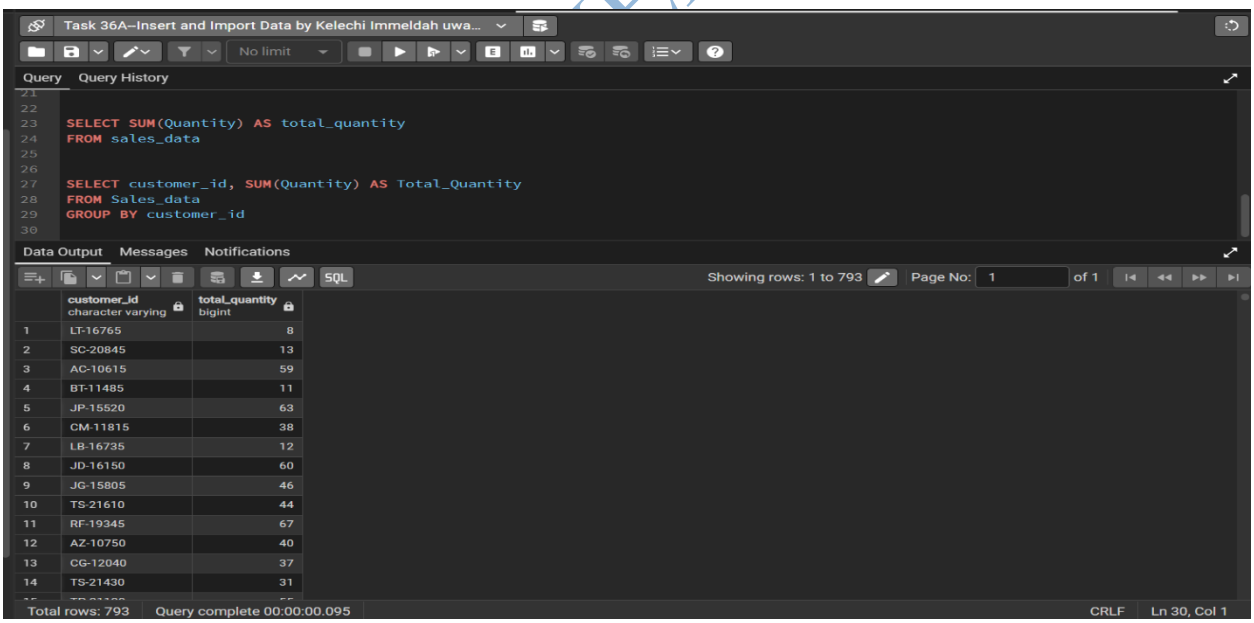


USE SUM () WITH GROUP BY

Here we use the SUM () function and the GROUP BY clause, to return the Quantity for each customer ID in the Sales_data table:

Example

```
SELECT customer_id, SUM(Quantity) AS Total_Quantity
FROM Sales_data
GROUP BY customer_id
```

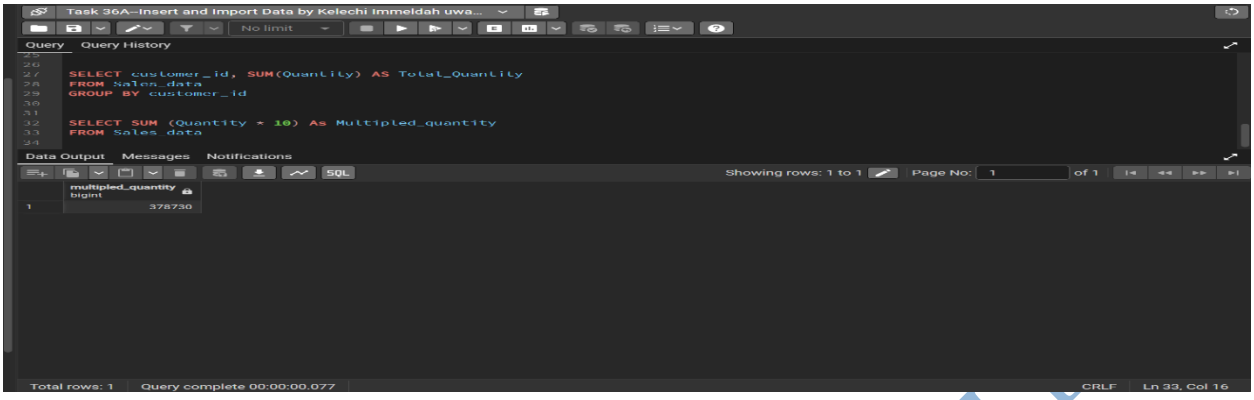


SUM () WITH AN EXPRESSION

The parameter inside the SUM () function can also be an expression.
If we assume that each sale in the sales data column costs 10 dollars, we can find the total earnings in dollars by multiplying each quantity by 10:

Example

Use an expression inside the SUM () function:
SELECT SUM (Quantity * 10)
FROM Sales_data



We can also join the Sales data table to the Products table to find the actual amount, instead of assuming it is 10 dollars:

Example
Join sales with Profit, and use SUM () to find the total amount:
SELECT SUM (sales * profit)
FROM Sales_data
LEFT JOIN sales ON Sales. Profit = sales. customer_id

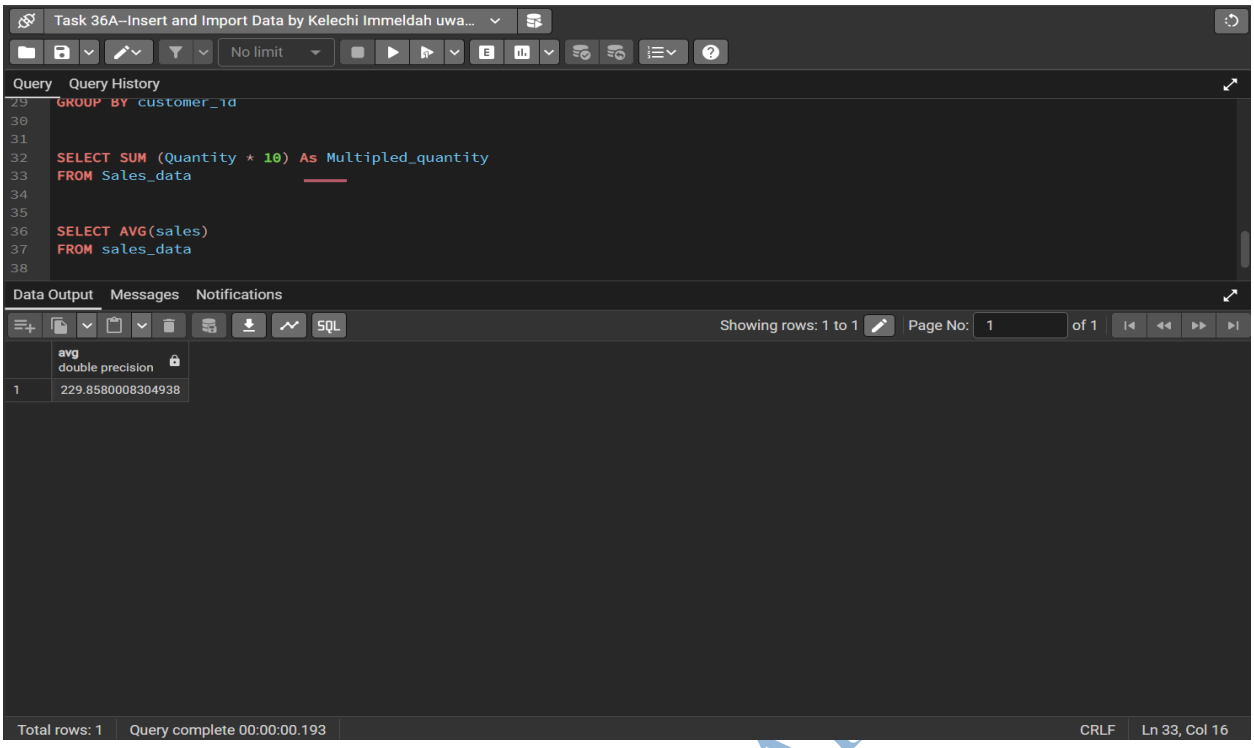
SQL AVG () FUNCTION

THE SQL AVG () FUNCTION

The AVG () function returns the average value of a numeric column.

Example
Find the average price of all products:
SELECT AVG (sales)
FROM sales_data

Note: NULL values are ignored.

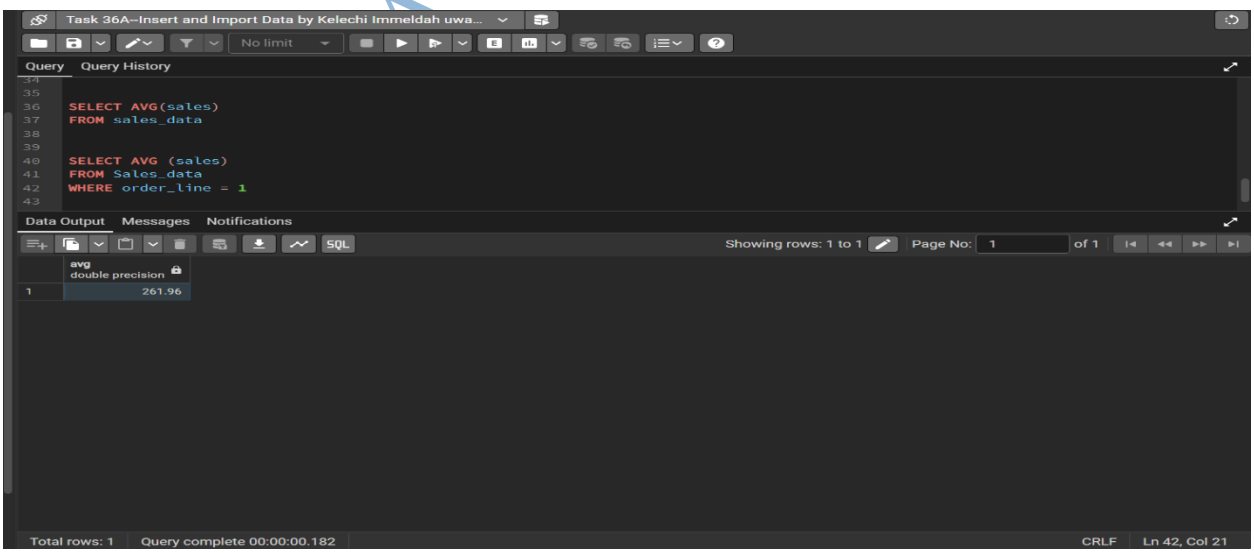


ADD A WHERE CLAUSE

You can add a WHERE clause to specify conditions:

Example
Return the average price of products in category 1:

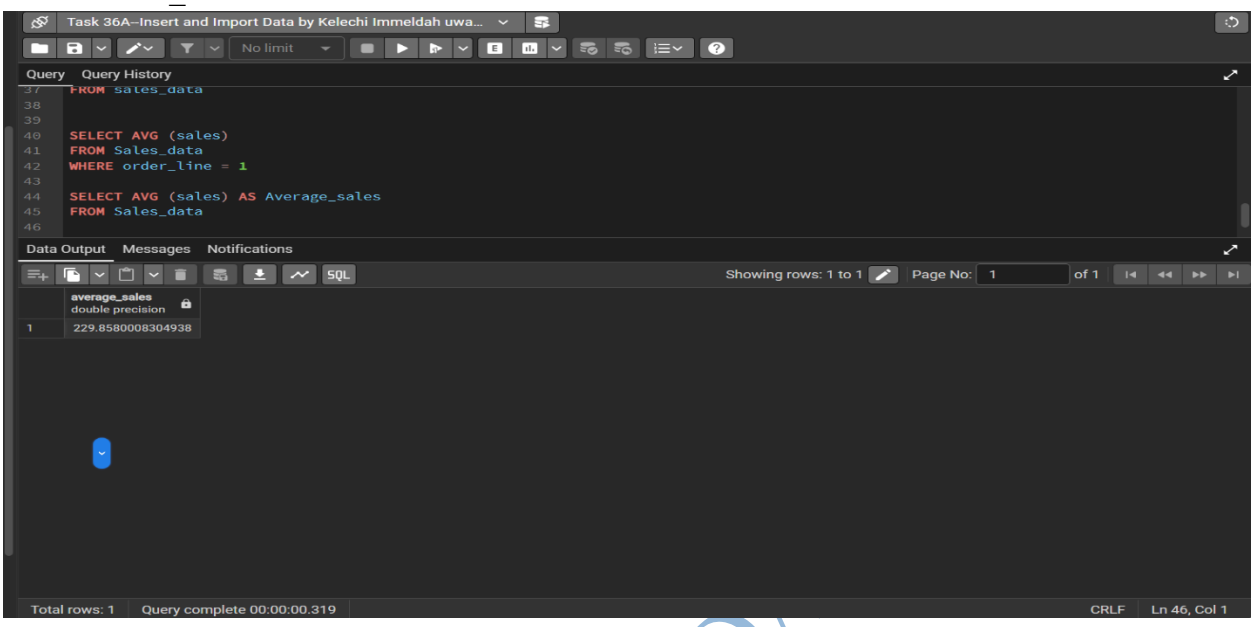
```
SELECT AVG (sales)
FROM Sales_data
WHERE order_line = 1
```



USE AN ALIAS

Give the AVG column a name by using the AS keyword.

Example
Name the column "average price":
SELECT AVG (sales) AS Average_sales
FROM Sales_data



HIGHER THAN AVERAGE

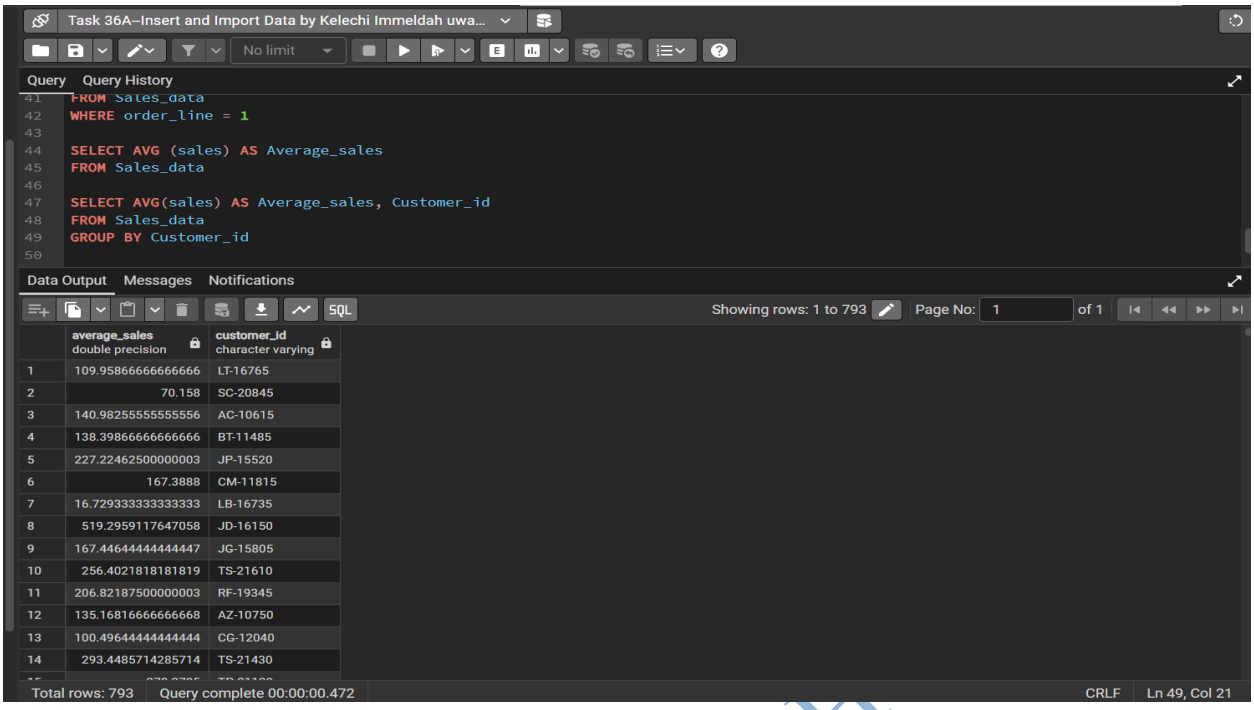
To list all records with a higher price than average, we can use the AVG () function in a sub-query

Example
Return all products with a higher price than the average price:
SELECT *
FROM Sales_data
WHERE Sales > (SELECT AVG (sales) FROM sales)

USE AVG () WITH GROUP BY

Here we use the AVG () function and the GROUP BY clause, to return the average price for each category in the Products table:

Example
SELECT AVG (sales) AS Average_sales, Customer_id
FROM Sales_data
GROUP BY Customer_id



SQL LIKE OPERATOR

THE SQL-LIKE OPERATOR

The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

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

You will learn more about

Example
Select all customers that start with the letter "a":
SELECT *
FROM Customer_data
WHERE Customer_name LIKE 'a%'

THE _ WILDCARD

The _ wildcard represents a single character.
It can be any character or number, but each _ represents one, and only one, character.
Example
Return all customers from a city that starts with 'L' followed by one wildcard character, then 'nd' and then two wildcard characters:

SELECT *

```
FROM Customer_data
WHERE city LIKE 'L_nd__'
```

THE % WILDCARD

The % wildcard represents any number of characters, even zero characters.
Example
Return all customers from a city that *contains* the letter 'L':

```
SELECT *
FROM Customer_data
WHERE city LIKE '%L%
```

STARTS WITH

To return records that start with a specific letter or phrase, add the % at the end of the letter or phrase.

Example
Return to all customers that start with 'La':

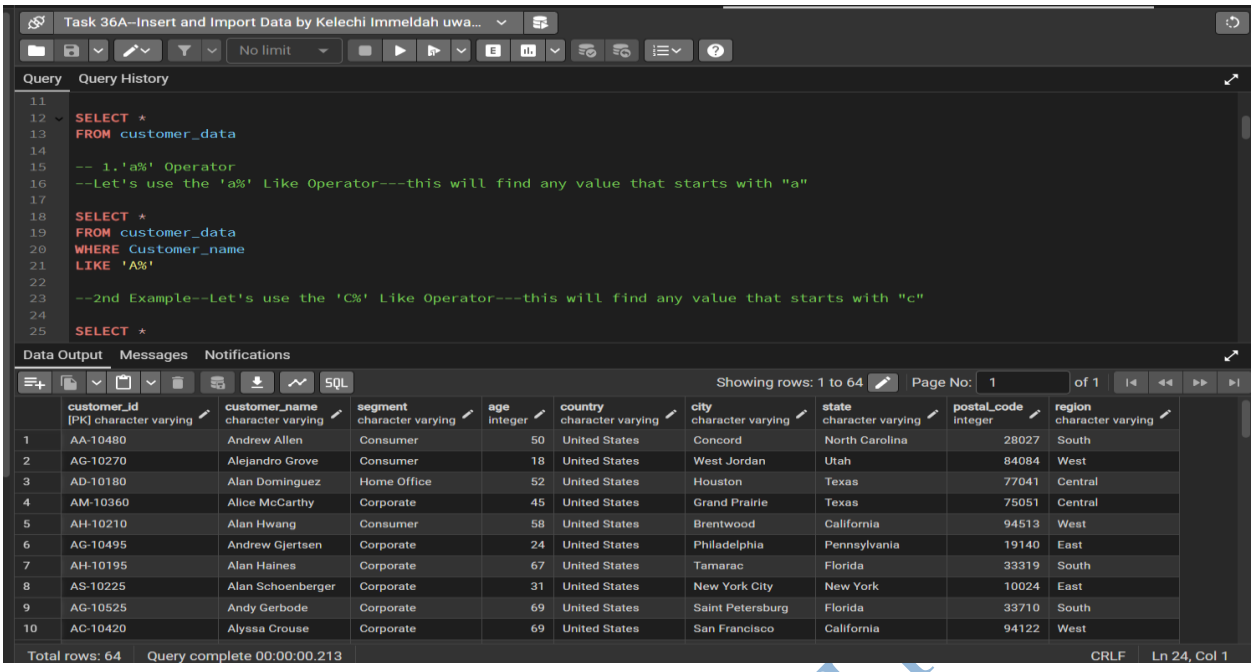
```
SELECT *

FROM Customer_data
WHERE Customer_Name LIKE 'La%';
```

Tip: You can also combine any conditions using AND or OR operators.

Example
Return to all customers that start with 'a' or start with 'b':

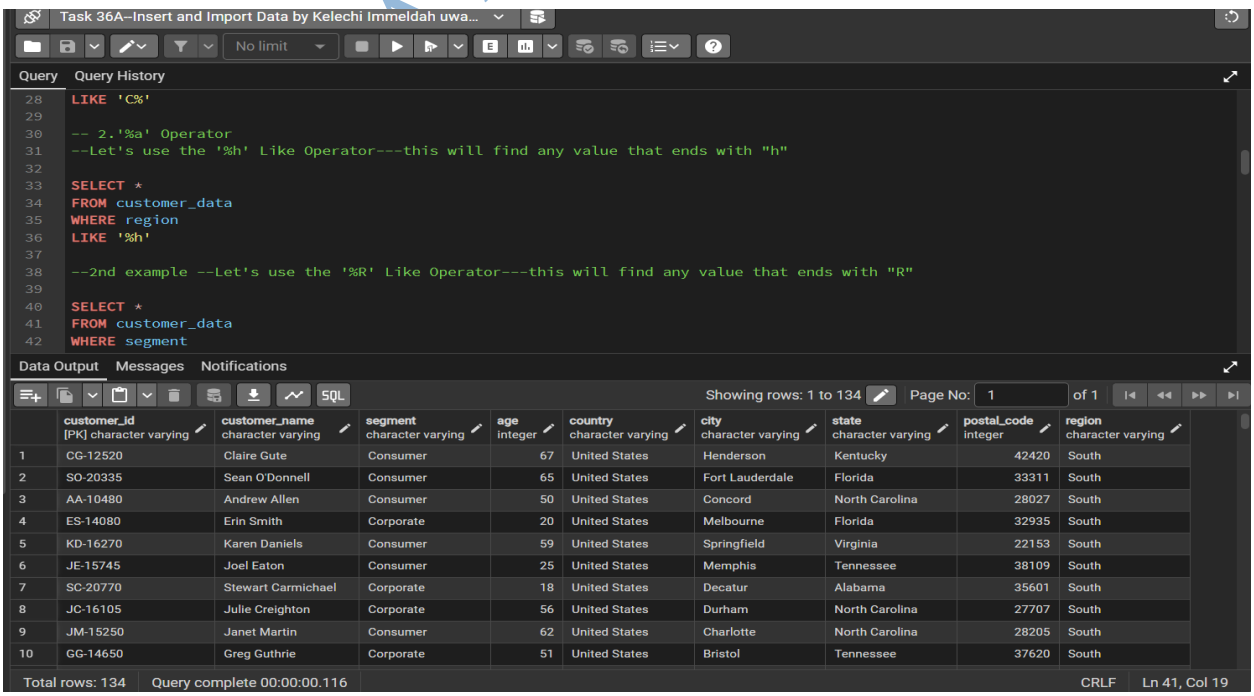
```
SELECT *
FROM Customer_data
WHERE Customer_Name LIKE 'a%' OR Customer_Name LIKE 'b%'
```



ENDS WITH

To return records that end with a specific letter or phrase, add the % at the beginning of the letter or phrase.

Example
Return all customers that end with 'a':
SELECT *
FROM Customer_data
WHERE Region LIKE '%h'
Tip: You can also combine "starts with" and "ends with":



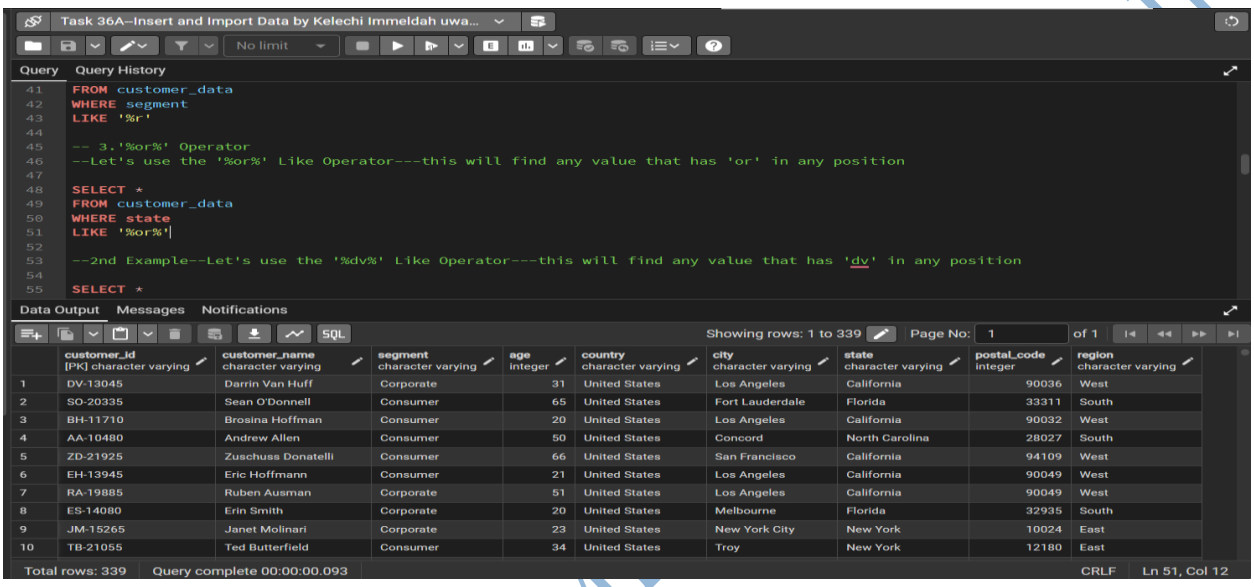
CONTAINS

To return records that contain a specific letter or phrase, add the % both before and after the letter or phrase.

Example

Return all customers that contain the phrase 'or'

```
SELECT *
FROM Customer_data
WHERE State LIKE '%or%'
```



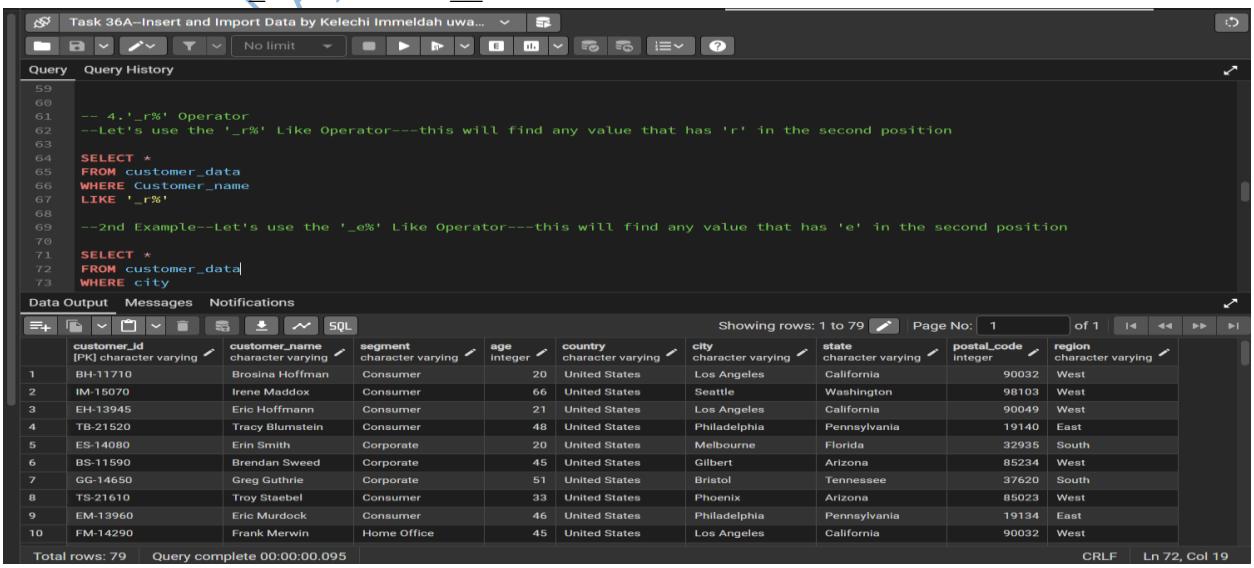
COMBINE WILDCARDS

Any wildcard, like % and _, can be used in combination with other wildcards.

Example

Return all customers that start with "a" and are at least 3 characters in length:

```
SELECT *
FROM Customer_data
WHERE Customer_Name LIKE 'a__%'
```



WITHOUT WILDCARD

If no wildcard is specified, the phrase has to have an exact match to return a result.

Example
Return all customers from Spain:
SELECT *
FROM Customer_data
WHERE City LIKE 'Los Angeles';

SQL WILDCARD

SQL WILDCARD CHARACTERS

A wildcard character is used to substitute one or more characters in a string.
Wildcard characters are used with the LIKE operator. The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

Example
Return all customers that start with the letter 'a'
SELECT *
FROM Customer_data
WHERE Customer_Name LIKE 'a%'

WILDCARD CHARACTERS

Symbol	Description
%	Represents zero or more characters
_	Represents a single character
[]	Represents any single character within the brackets *
^	Represents any character not in the brackets *
-	Represents any single character within the specified range *
{}	Represents any escaped character **

* Not supported in PostgreSQL and MySQL databases.
** Supported only in Oracle databases.

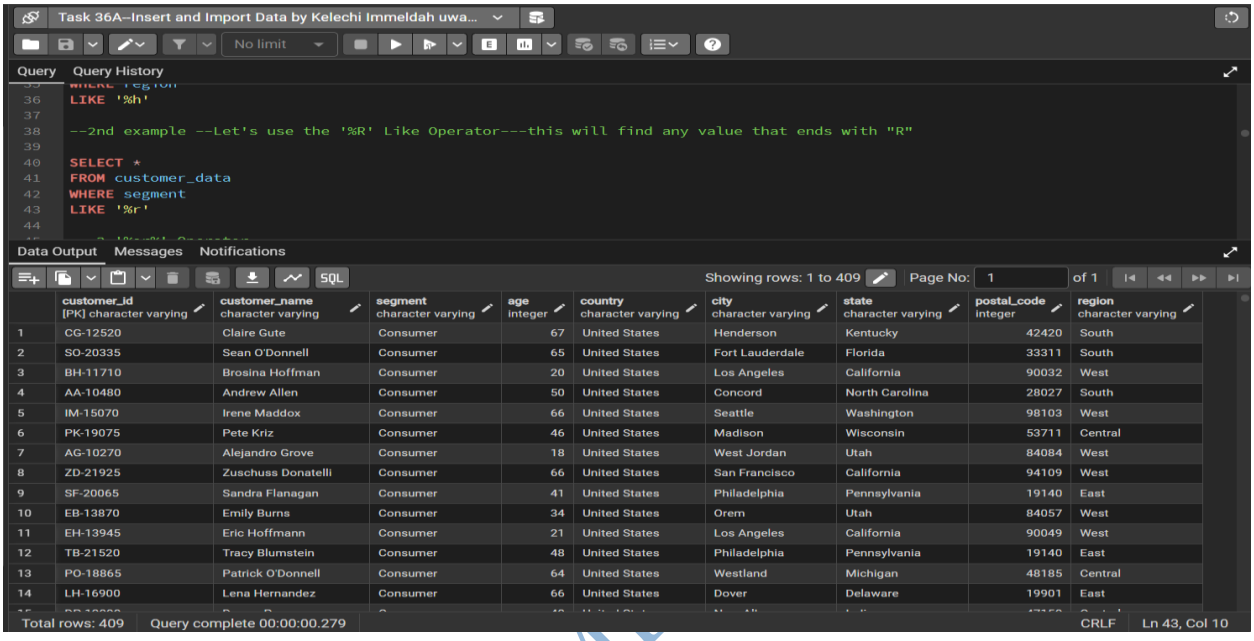
USING THE % WILDCARD

The % wildcard represents any number of characters, even zero characters.

Example

Return all customers that end with the pattern 'r':

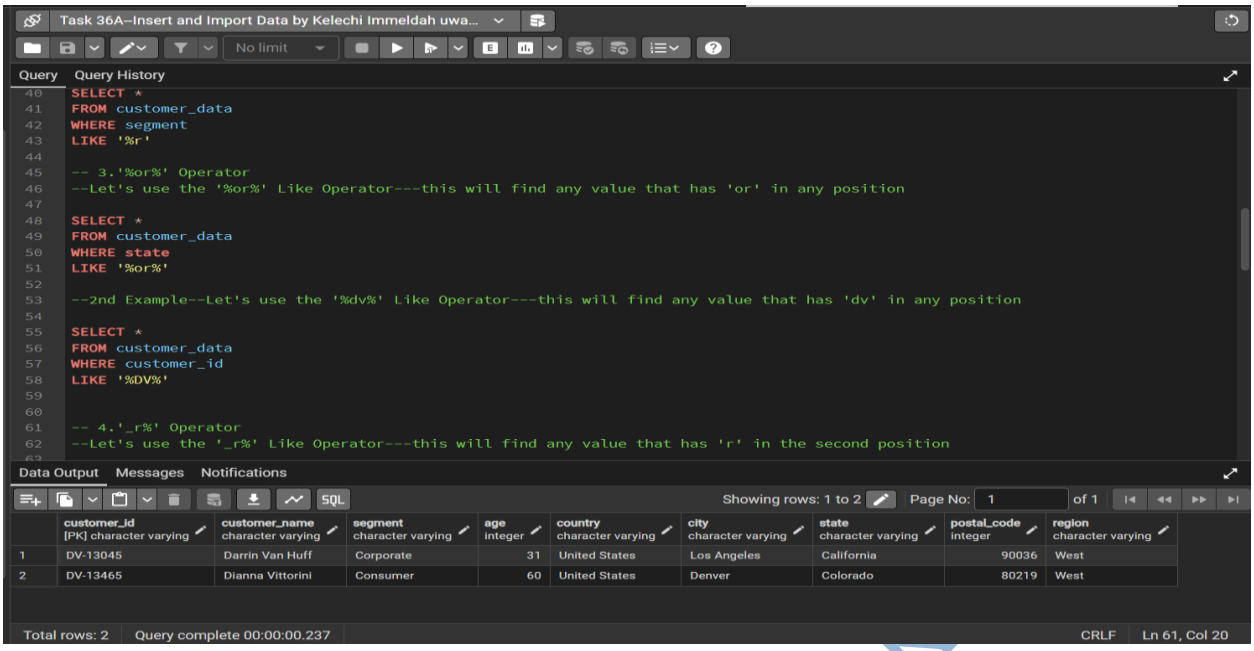
```
SELECT *
FROM Customer_data
WHERE Segment LIKE '%r'
```



Example

Return all customers that *contain* the pattern 'DV':

```
SELECT *
FROM Customer_data
WHERE Customer_id LIKE '%DV%'
```

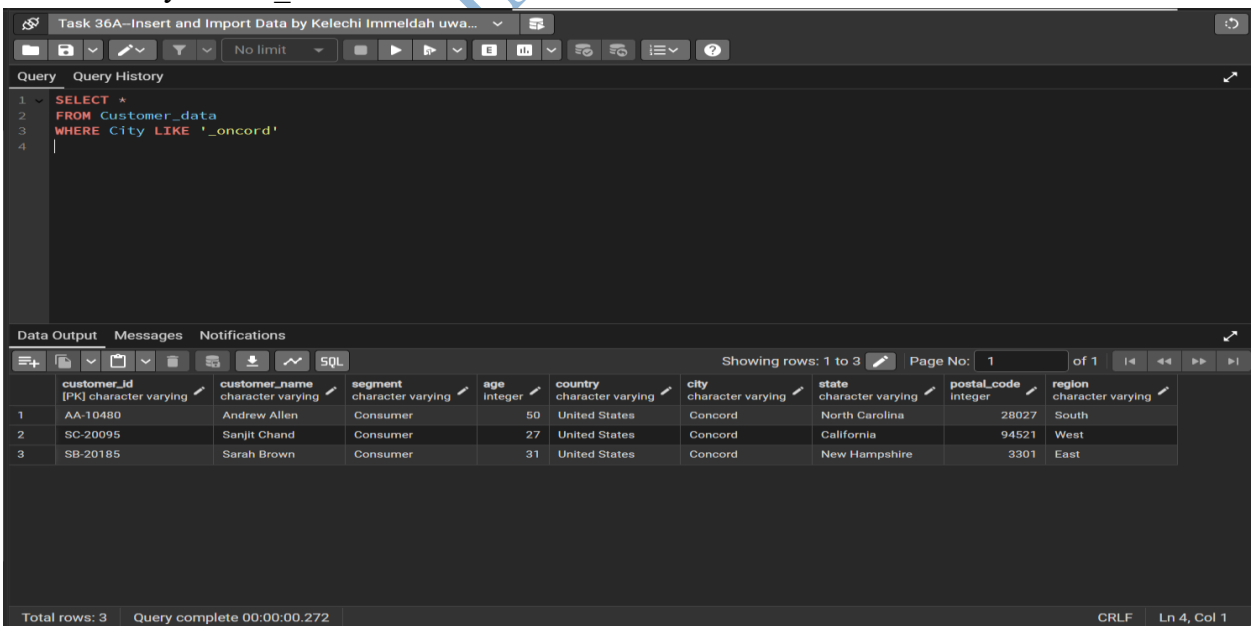


USING THE _ WILDCARD

The _ wildcard represents a single character.
It can be any character or number, but each _ represents one, and only one, character.
Example

Return to all customers with a city starting with any character, followed by "concord":

SELECT *
FROM Customer_data
WHERE City LIKE '_oncord'



COMBINE WILDCARDS

Any wildcard, like % and _, can be used in combination with other wildcards.

Example
Return all customers that have "r" in the second position:
SELECT *
FROM Customer_data
WHERE Customer_Name LIKE '_r%'

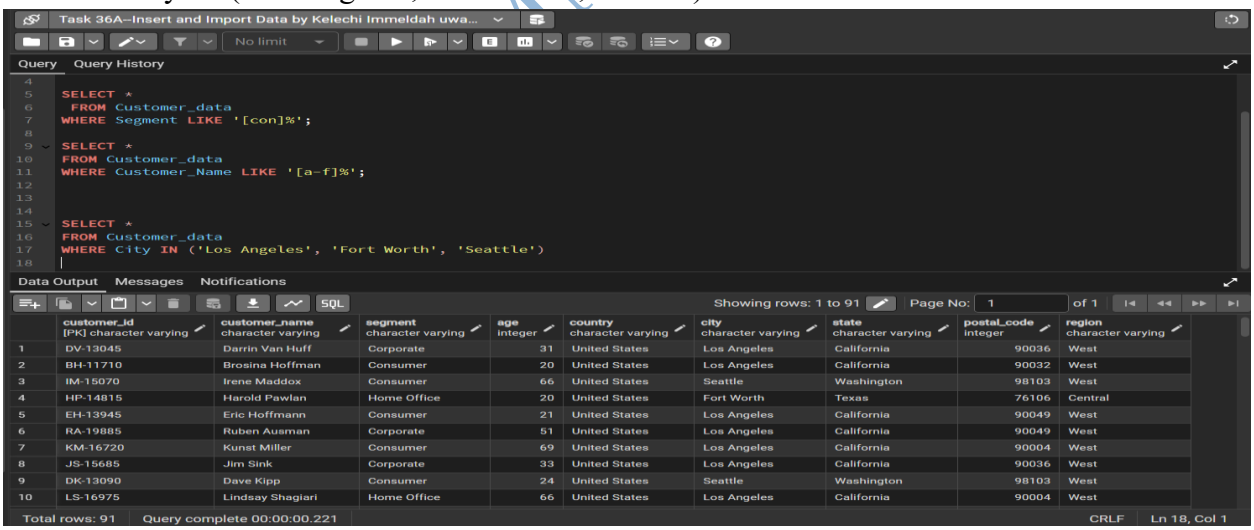
WITHOUT WILDCARD

If no wildcard is specified, the phrase has to have an exact match to return a result.
Example
Return all customers from Customer data:
SELECT *
FROM Customer_data
WHERE Country LIKE 'Los Angeles'

SQL IN OPERATOR

THE SQL IN OPERATOR

The IN operator allows you to specify multiple values in a WHERE clause.
Example
Return all customers from 'Los Angeles', 'Fort Worth', 'And Seattle'
SELECT *
FROM Customer_data
WHERE City IN ('Los Angeles', 'Fort Worth', 'Seattle')

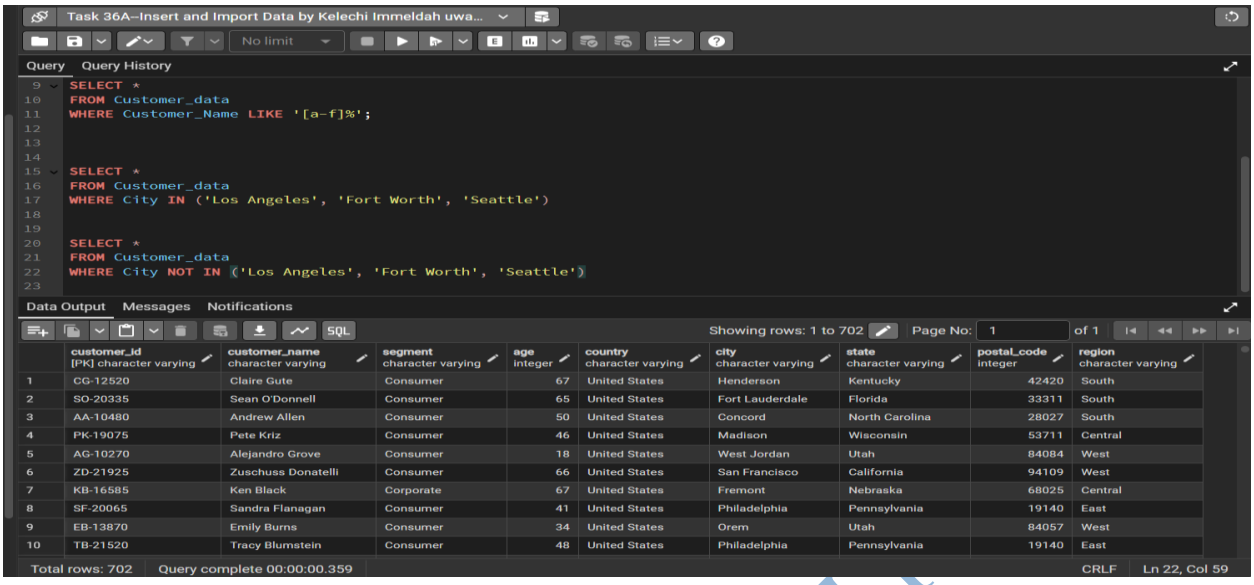


NOT IN

By using the NOT keyword in front of the IN operator, you return all records that are NOT any of the values in the list.
Example

Return to all customers that are NOT from 'Los Angeles', 'Fort Worth', or 'Seattle':SELECT *

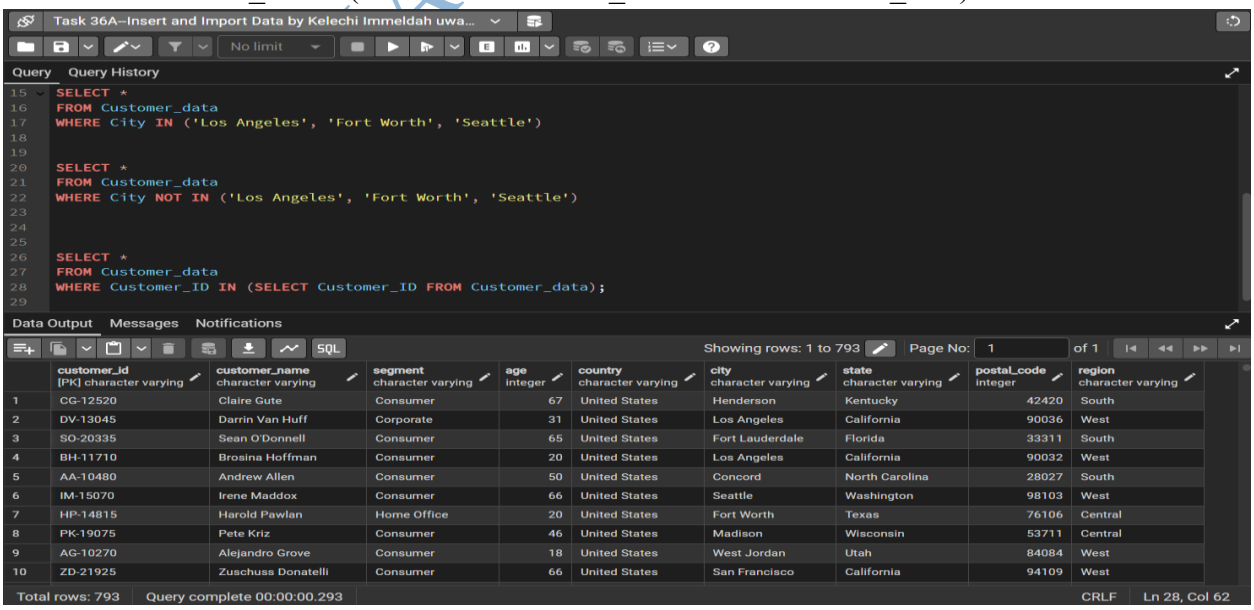
FROM Customer_data
WHERE City NOT IN ('Los Angeles', 'Fort Worth', 'Seattle')



IN (SELECT)

You can also use IN with a subquery in the WHERE clause.
With a subquery, you can return all records from the main query that are present in the result of the subquery.

Example
Return all customers that have an order in the Customer data table:
SELECT *
FROM Customer_data
WHERE Customer_ID IN (SELECT Customer_ID FROM Customer_data)



NOT IN (SELECT)

The result in the example above returned 74 records, which means that 17 customers haven't placed any orders.
Let us check if that is correct, by using the NOT IN operator.

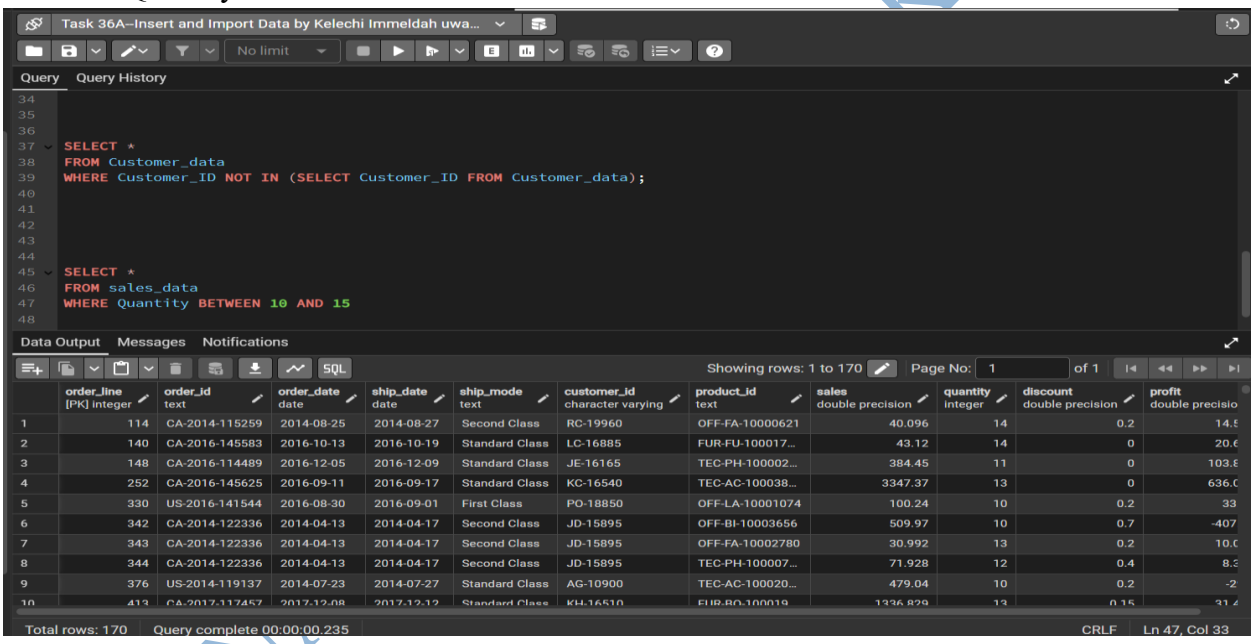
SQL BETWEEN OPERATORS

THE SQL BETWEEN OPERATORS

The BETWEEN operator selects values within a given range. The values can be numbers, text, or dates.
The BETWEEN operator is inclusive: begin and end values are included.

Example
Selects all products with a price of between 10 and 20:

```
SELECT *
FROM sales_data
WHERE Quantity BETWEEN 10 AND 15
```

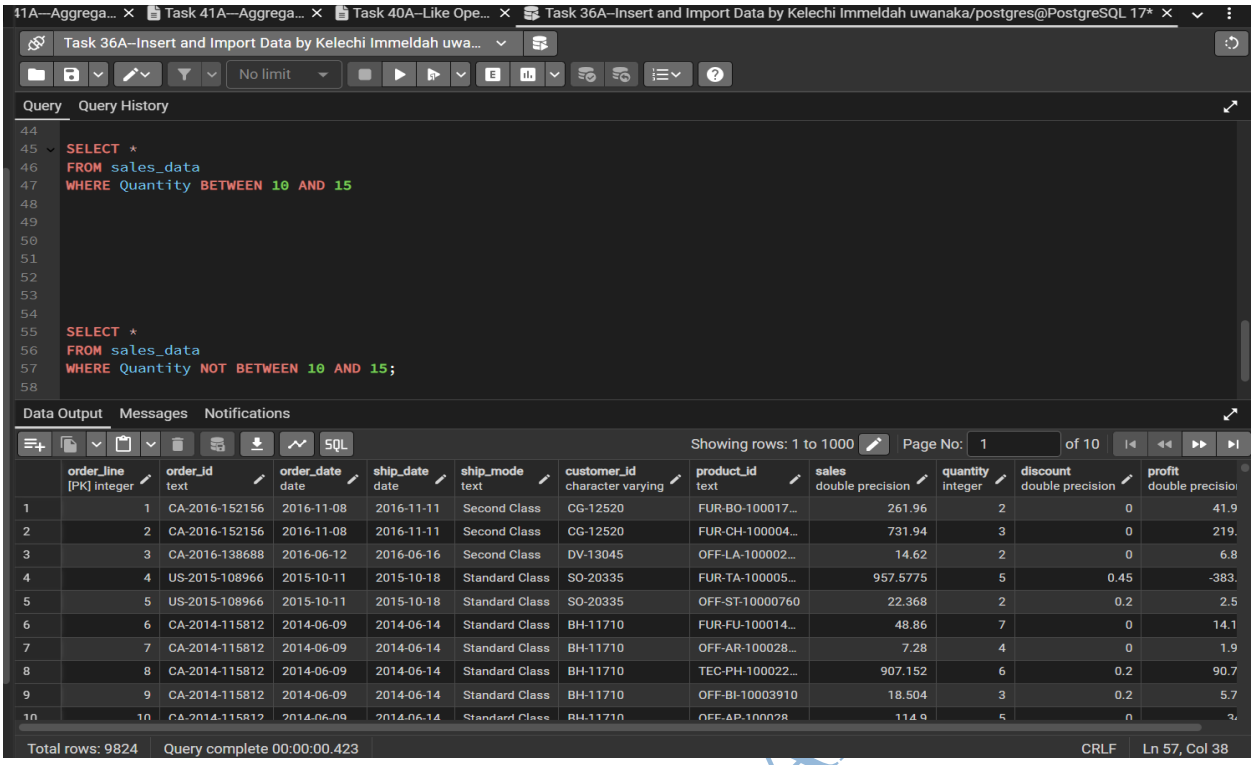


	order_line [PK] integer	order_id text	order_date date	ship_date date	ship_mode text	customer_id character varying	product_id text	sales double precision	quantity integer	discount double precision	profit double precisio
1	114	CA-2014-115259	2014-08-25	2014-08-27	Second Class	RC-19960	OFF-FA-10000621	40.096	14	0.2	14.5
2	140	CA-2016-145583	2016-10-13	2016-10-19	Standard Class	LC-16885	FUR-FU-100017...	43.12	14	0	20.6
3	148	CA-2016-114489	2016-12-05	2016-12-09	Standard Class	JE-16165	TEC-PH-100002...	384.45	11	0	103.8
4	252	CA-2016-145625	2016-09-11	2016-09-17	Standard Class	KC-16540	TEC-AC-100038...	3347.37	13	0	636.0
5	330	US-2016-141544	2016-08-30	2016-09-01	First Class	PO-18850	OFF-LA-10001074	100.24	10	0.2	33
6	342	CA-2014-122336	2014-04-13	2014-04-17	Second Class	JD-15895	OFF-BI-10003656	509.97	10	0.7	-407
7	343	CA-2014-122336	2014-04-13	2014-04-17	Second Class	JD-15895	OFF-FA-10002780	30.992	13	0.2	10.0
8	344	CA-2014-122336	2014-04-13	2014-04-17	Second Class	JD-15895	TEC-PH-100007...	71.928	12	0.4	8.3
9	376	US-2014-119137	2014-07-23	2014-07-27	Standard Class	AG-10900	TEC-AC-100020...	479.04	10	0.2	-2
10	413	CA-2017-117457	2017-12-08	2017-12-12	Standard Class	KH-16510	FUR-BO-100019...	1336.829	13	0.15	31.4

NOT BETWEEN

To display the products outside the range of the previous example, use NOT BETWEEN:
Example

```
SELECT *
FROM sales_data
WHERE Quantity NOT BETWEEN 10 AND 15
```



BETWEEN WITHIN

The following SQL statement selects all customers with a quantity between 10 and 15. In addition, the order_line must be either 1,2, or 3:

Example

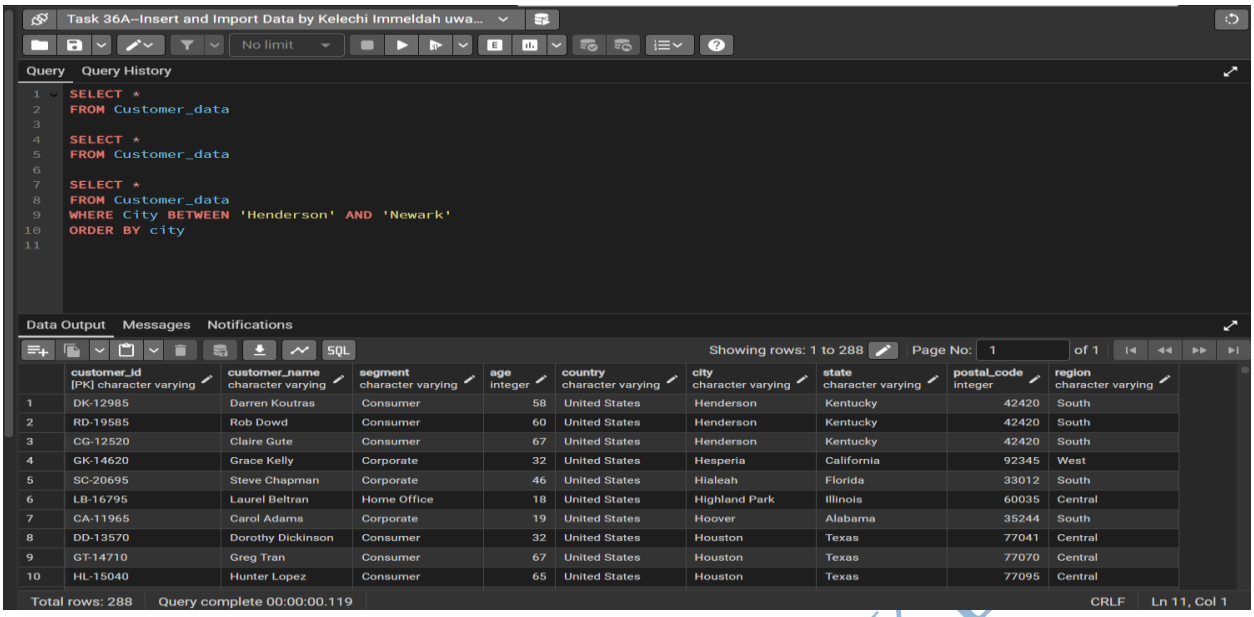
```
SELECT *
FROM Sales_data
WHERE Quantity BETWEEN 10 AND 15
AND Order_line IN (1,2,3)
```

BETWEEN TEXT VALUES

The following SQL statement selects all Customers with Customer_Name alphabetically between Claire Gute and Andrew Allen.

Example

```
SELECT *
FROM Customer_data
WHERE City BETWEEN 'Henderson' AND 'Newark'
ORDER BY City
```

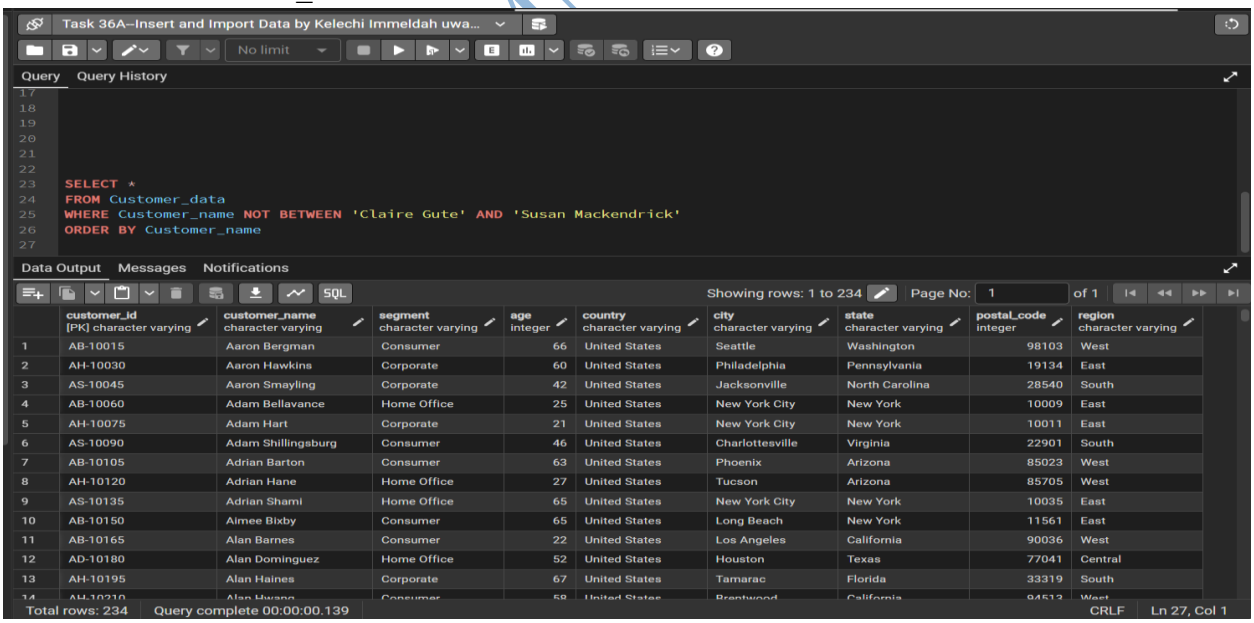


NOT BETWEEN TEXT VALUES

The following SQL statement selects all Customers with a Customer Name, not between Claire Gute and Susan Mackendrick.

Example

```
SELECT *
FROM Customer_data
WHERE Customer_name NOT BETWEEN 'Claire Gute' AND 'Susan Mackendrick'
ORDER BY Customer_name
```

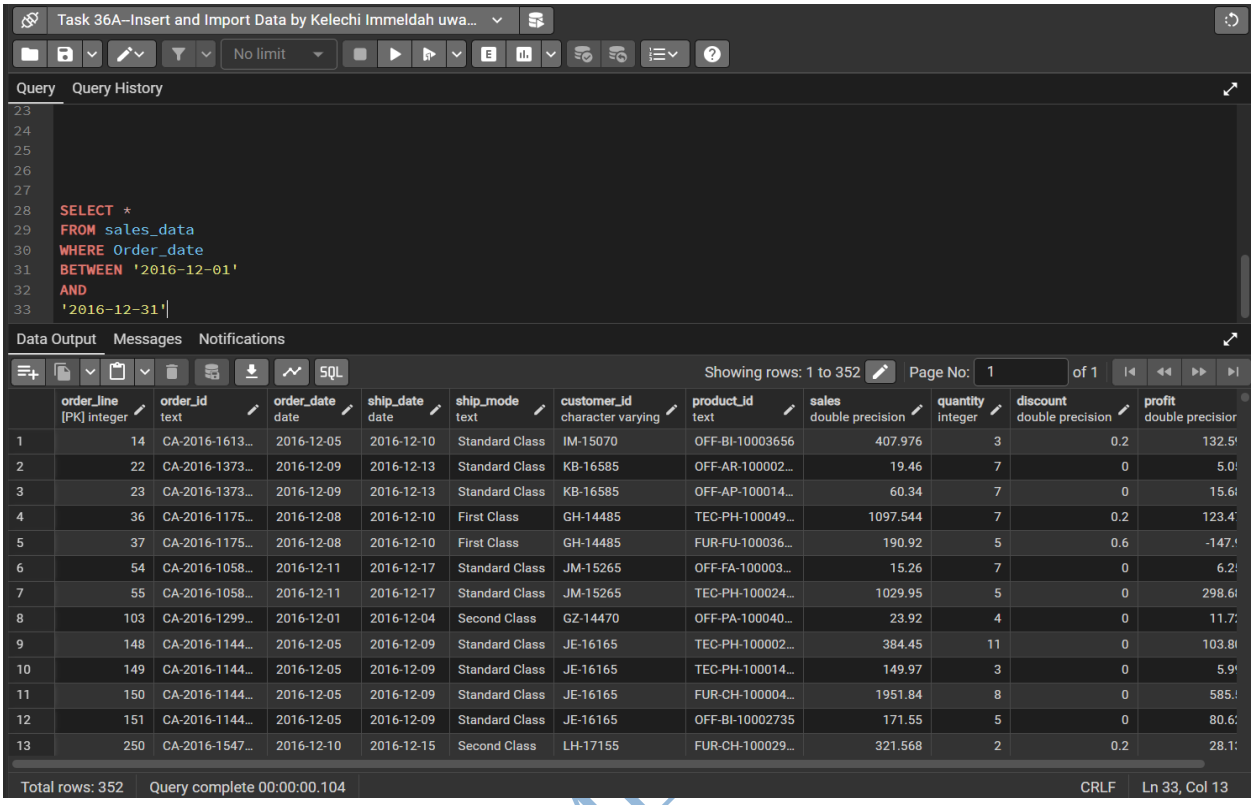


BETWEEN DATES

The following SQL statement selects all orders with an Order_Date between '2016-12-01' and '2016-12-31'.

Example

SELECT *
FROM sales_data
WHERE Order_Date BETWEEN '2016-12-01' AND '2016-12-31'.



SQL ALIASES

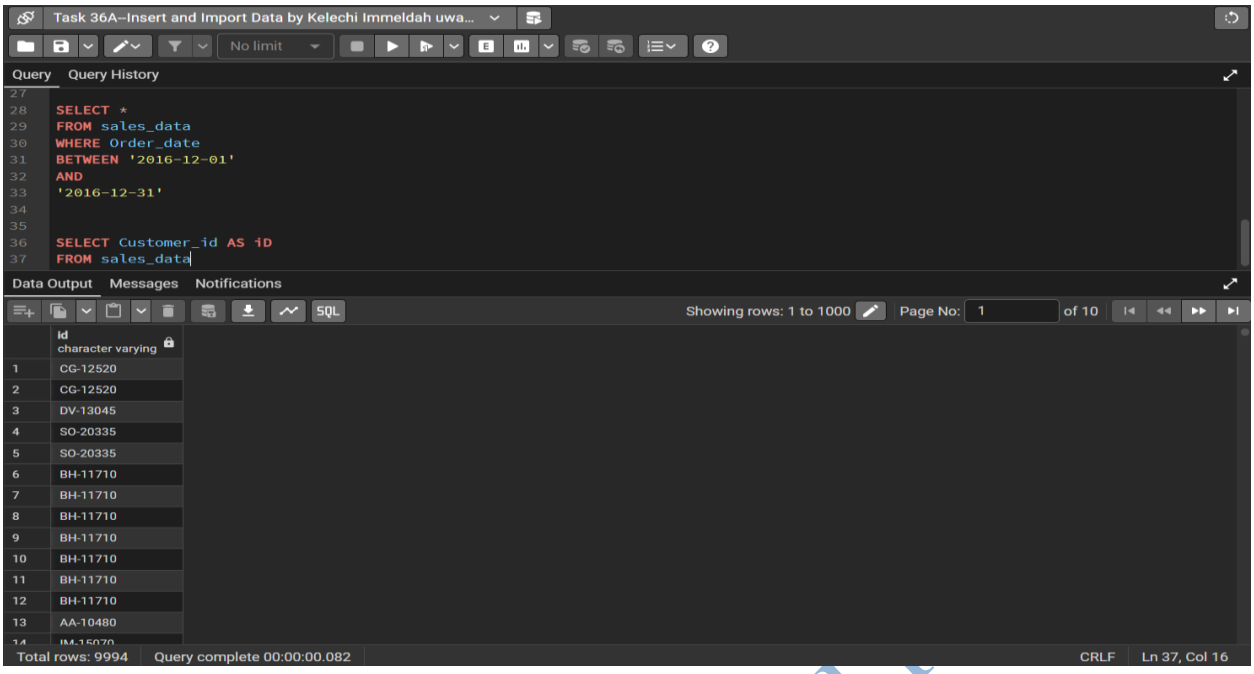
SQL ALIASES

SQL aliases are used to give a table, or a column in a table, a temporary name.
Aliases are often used to make column names more readable.
An alias only exists for the duration of that query.
An alias is created with the AS keyword.
Example

SELECT Customer_ID AS IDs
FROM Sales_data

AS IS OPTIONAL

Actually, in most database languages, you can skip the AS keyword and get the same result:
Example
SELECT Customer_ID ID
FROM Sales_data

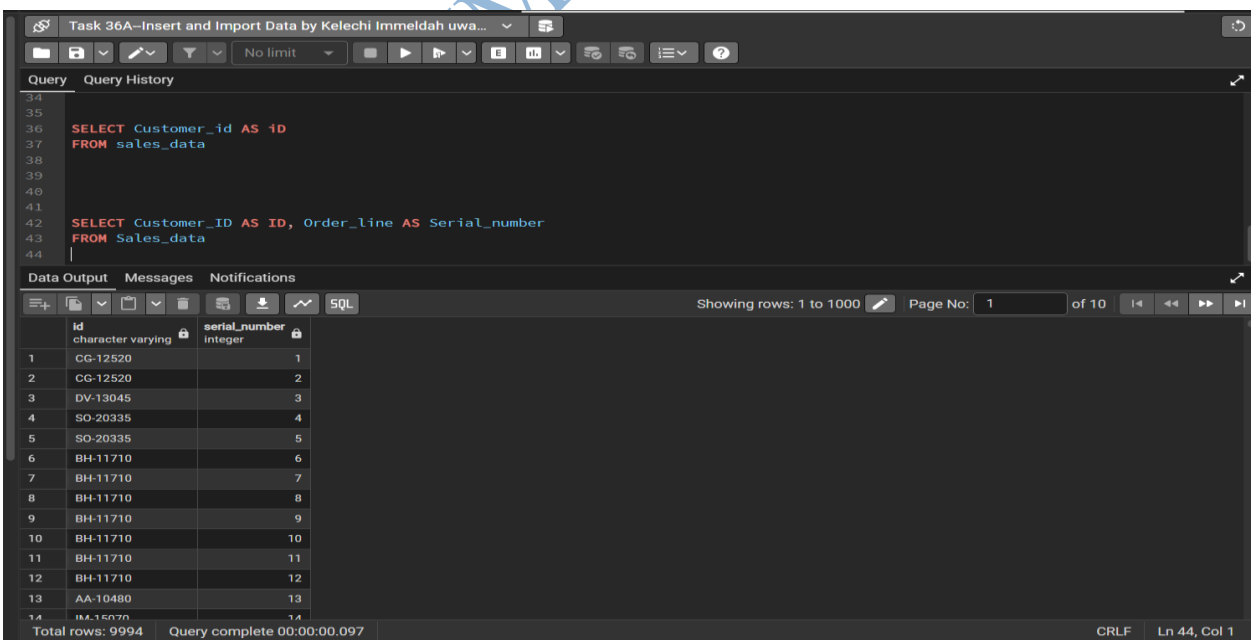


ALIAS FOR COLUMNS

The following SQL statement creates two aliases, one for the Customer_ID column and one for the Customer_Name column:

Example

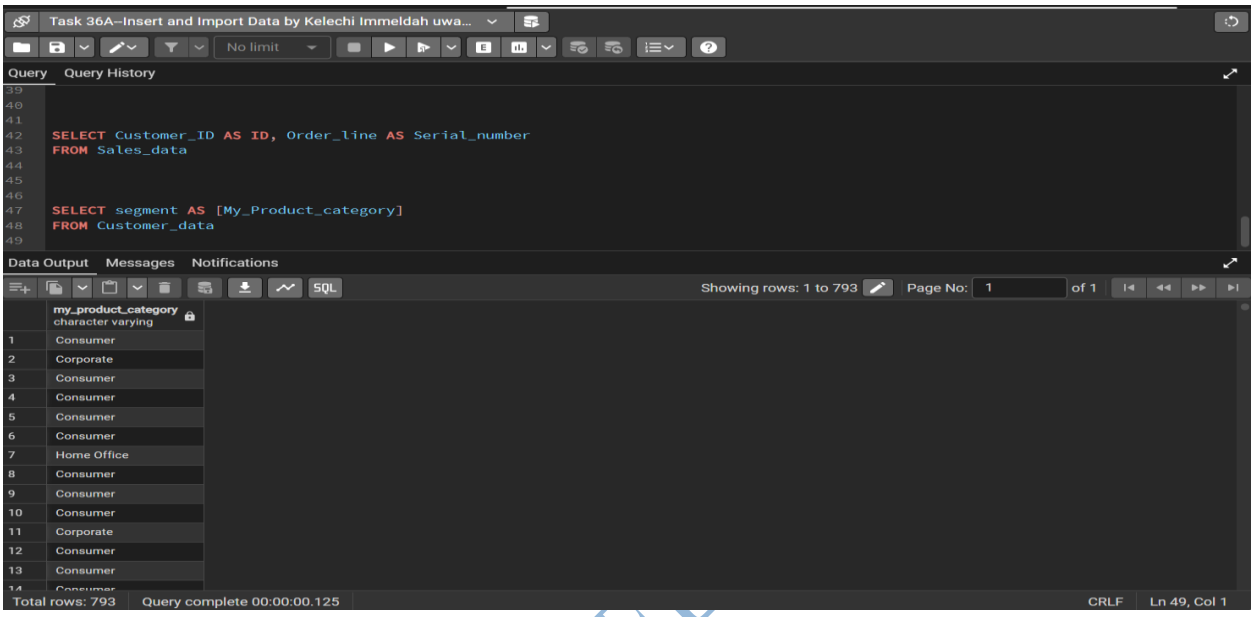
SELECT Customer_ID AS ID, Order_line AS Serial_number
FROM Sales_data



USING ALIASES WITH A SPACE CHARACTER

If you want your alias to contain one or more spaces, like " My Product category", surround your alias with square brackets or double quotes.

Example
Using "double quotes" for aliases with space characters:
SELECT segment AS ‘My Products Category
FROM Customer_data



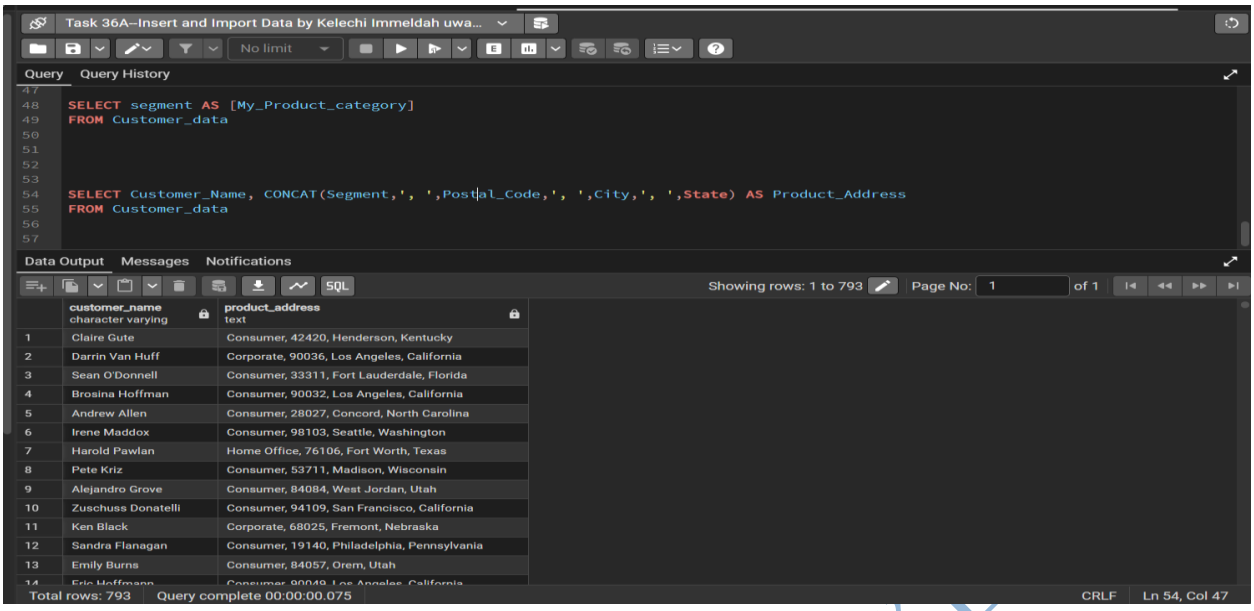
Note: Some database systems allow both [] and ", and some only allow one of them.

CONCATENATE COLUMNS

The following SQL statement creates an alias named "Address" that combines four columns (Segment, Postal_Code, City, and state):

MYSQL EXAMPLE

SELECT Customer_Name, CONCAT (Segment,', ', Postal_Code,', ‘, City,', ",
State) AS Product_Address
FROM Customer_data

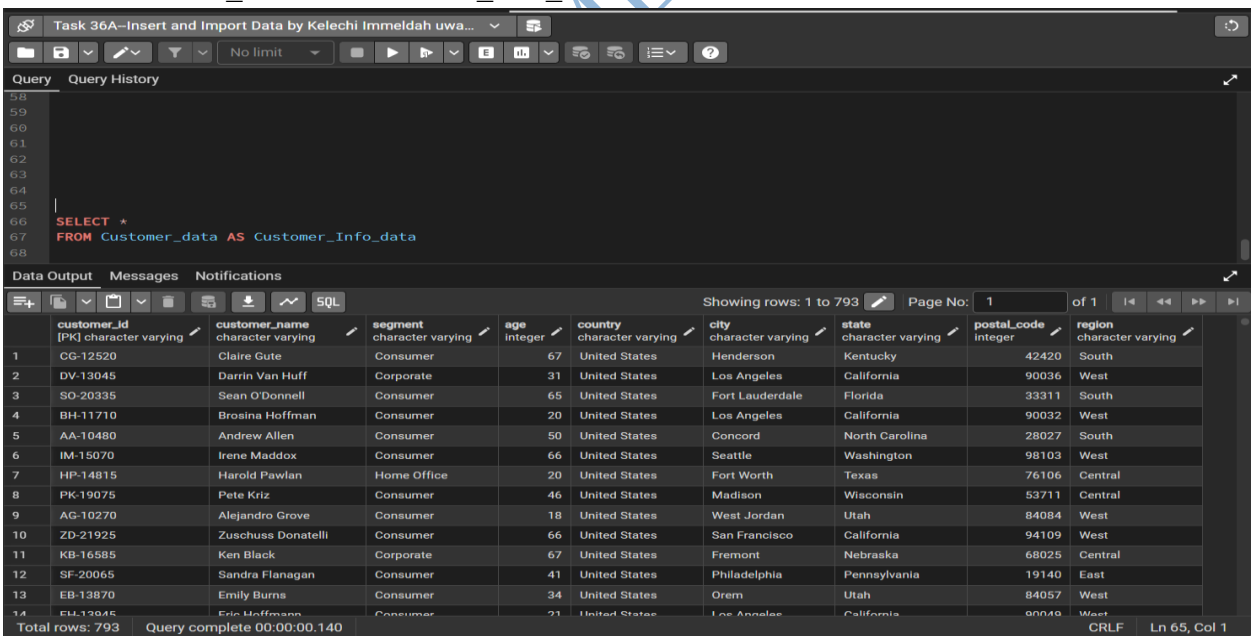


ALIAS FOR TABLES

The same rules apply when you want to use an alias for a table.
Example

Refer to the Customers data table as Customer Info data instead:

SELECT *
FROM Customer_data AS Customer_Info_data



It might seem useless to use aliases on tables, but when you are using more than one table in your queries, it can make the SQL statements shorter.

The following SQL statement selects all the orders from the customer with Customer_ID=4 (Around the Horn). We use the "Customers" and "Orders" tables, and give them the table aliases of "c" and "o" respectively (Here we use aliases to make the SQL shorter):

ALIASES CAN BE USEFUL WHEN:

- There is more than one table involved in a query
- Functions are used in the query
- Column names are big or not very readable
- Two or more columns are combined

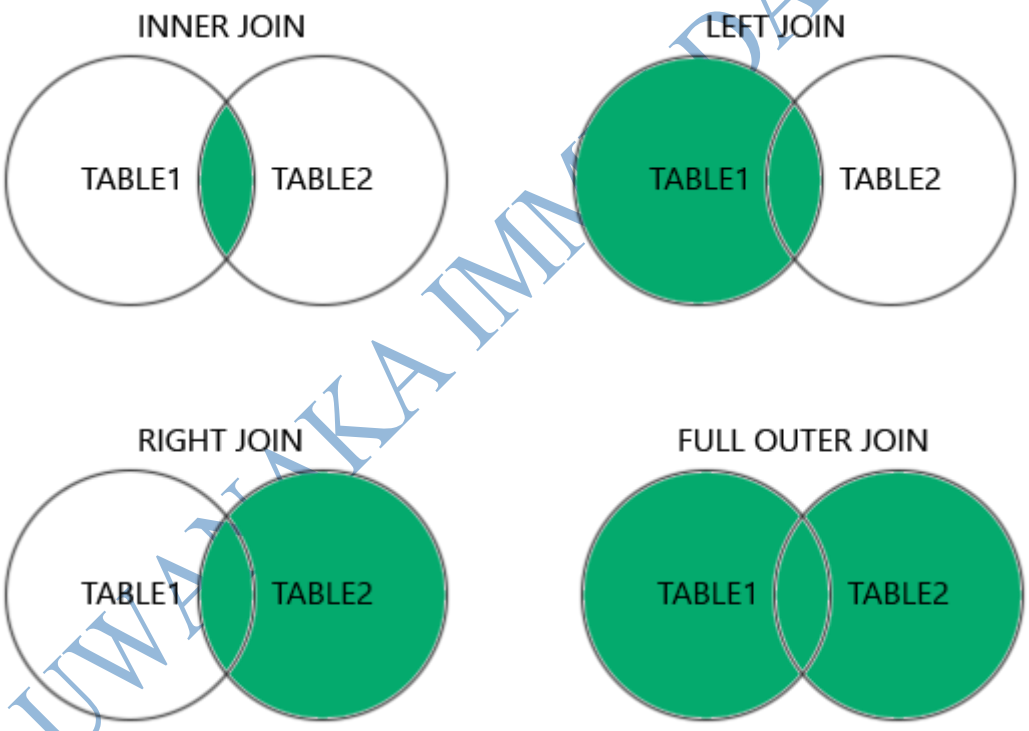
SQL JOIN

A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

DIFFERENT TYPES OF SQL JOINS

Here are the different types of JOINs in SQL:

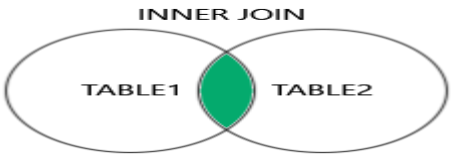
- **INNER JOIN:** Returns records that have matching values in both tables
- **LEFT (OUTER) JOIN:** Returns all records from the left table, and the matched records from the right table
- **RIGHT (OUTER) JOIN:** Returns all records from the right table, and the matched records from the left table
- **FULL (OUTER) JOIN:** Returns all records when there is a match in either left or right table



SQL INNER JOIN

THE INNER JOIN keyword selects records that have matching values in both tables.

We will join the Products table with the Categories table, by using the Category ID field from both tables:



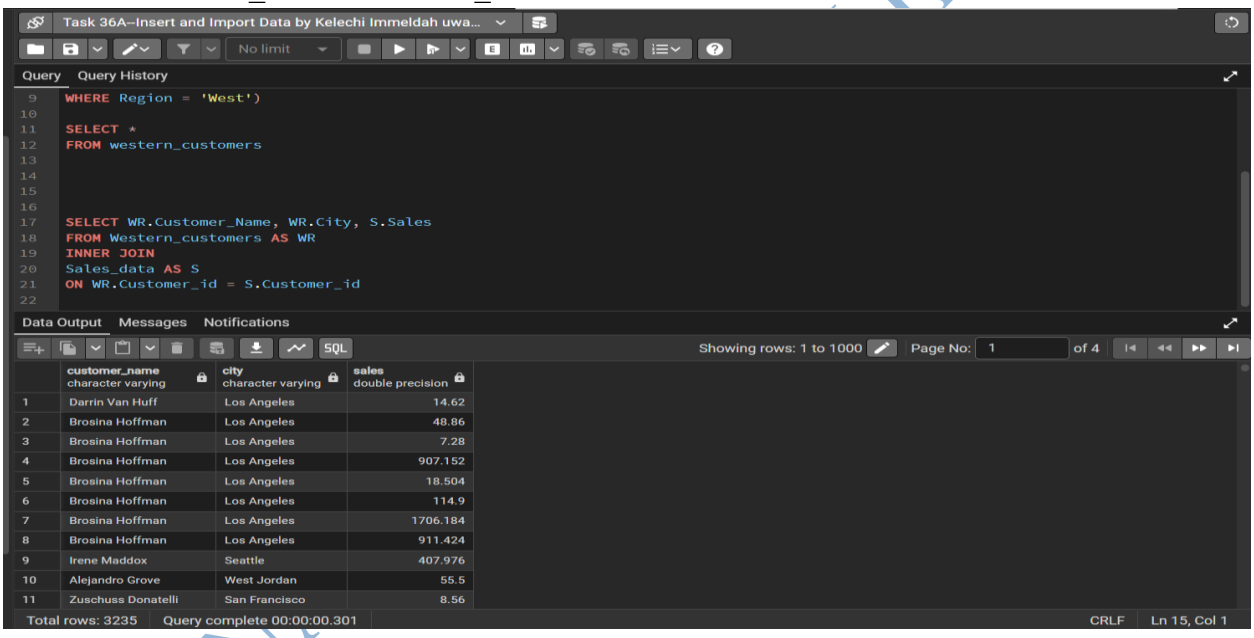
Note: The INNER JOIN keyword returns only rows with a match on both tables. If you have a product with no Customer ID or a Customer ID that is not present on the sales table, that record will not be returned as a result.

NAMING THE COLUMNS

It is good practice to include the table name when specifying columns in the SQL statement.

Example

```
SELECT WR. Customer_Name, WR. City, S.Sales
FROM Western_customers AS WR
INNER JOIN
Sales_data AS S
ON WR. Customer_id = S.Customer_id
```



The example above works without specifying table names, because none of the specified column names are present in both tables. If you try to include Customer_id in the SELECT statement, you will get an error if you do not specify the table name (because Customer_id is present in both tables.

JOIN or INNER JOIN

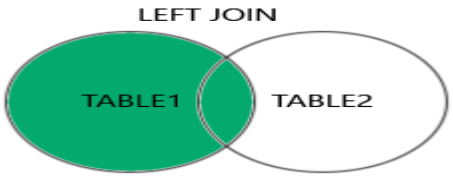
JOIN and INNER JOIN will return the same result.

INNER is the default join type for JOIN, so when you write JOIN the parser writes INNER JOIN.

SQL LEFT JOIN KEYWORD

THE LEFT JOIN keyword returns all records from the left table (table1), and the matching records from the right table (table2). The result is 0 records from the right side if there is no match.

Note: In some databases LEFT JOIN is called LEFT OUTER JOIN.

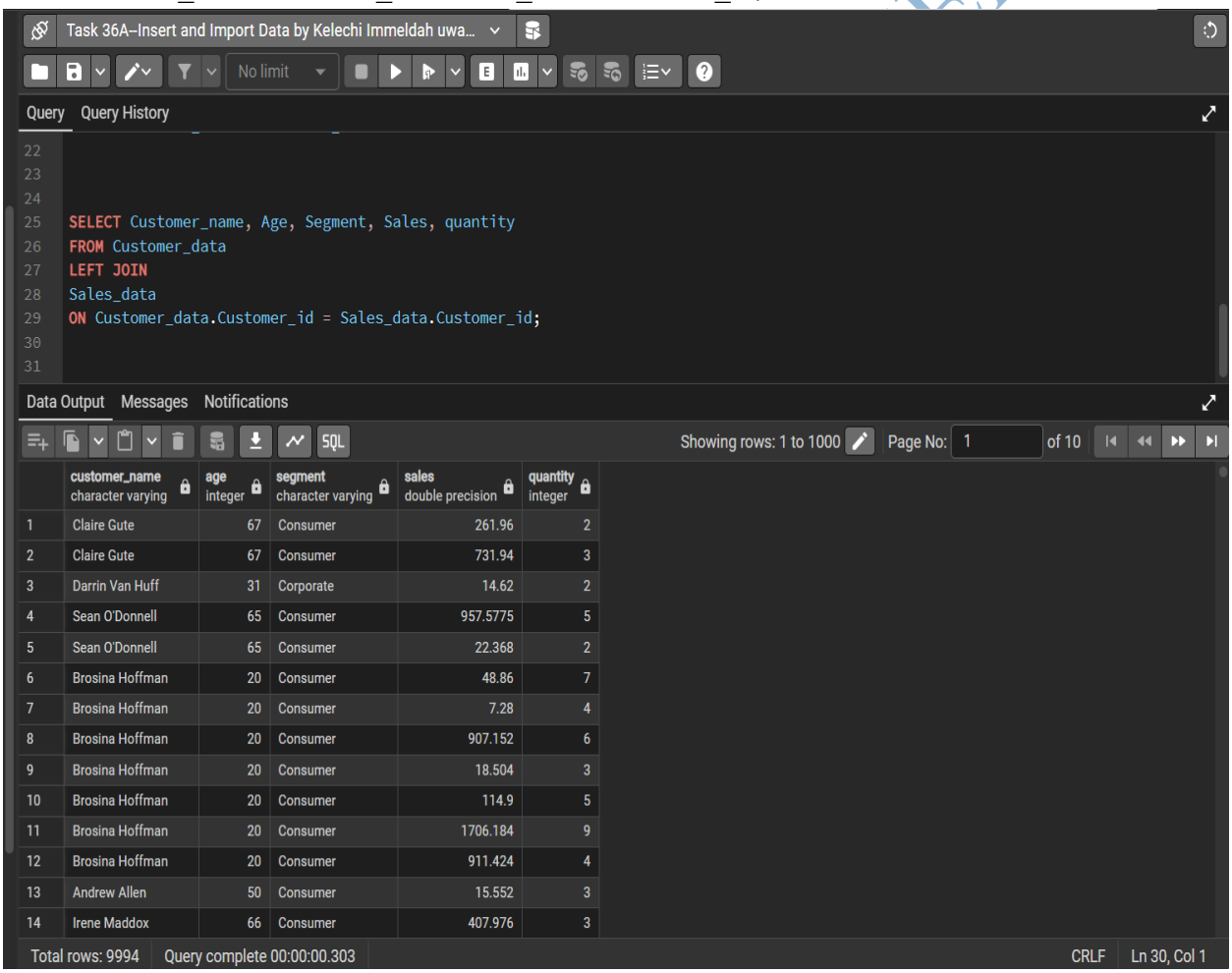


SQL LEFT JOIN EXAMPLE

The following SQL statement will select all customers and any orders they might have.

Example

```
SELECT Customer_name, Age, Segment, Sales, quantity
FROM Customer_data
LEFT JOIN
Sales_data
ON Customer_data.Customer_id = Sales_data.Customer_id;
```

	customer_name character varying	age integer	segment character varying	sales double precision	quantity integer
1	Claire Gute	67	Consumer	261.96	2
2	Claire Gute	67	Consumer	731.94	3
3	Darrin Van Huff	31	Corporate	14.62	2
4	Sean O'Donnell	65	Consumer	957.5775	5
5	Sean O'Donnell	65	Consumer	22.368	2
6	Brosina Hoffman	20	Consumer	48.86	7
7	Brosina Hoffman	20	Consumer	7.28	4
8	Brosina Hoffman	20	Consumer	907.152	6
9	Brosina Hoffman	20	Consumer	18.504	3
10	Brosina Hoffman	20	Consumer	114.9	5
11	Brosina Hoffman	20	Consumer	1706.184	9
12	Brosina Hoffman	20	Consumer	911.424	4
13	Andrew Allen	50	Consumer	15.552	3
14	Irene Maddox	66	Consumer	407.976	3

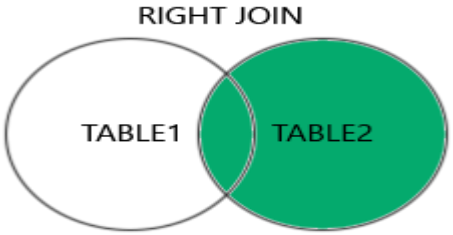
Note: The LEFT JOIN keyword returns all records from the left table (Customer-data), even if there are no matches in the right table (Sales_data).

SQL RIGHT JOIN KEYWORD

THE RIGHT JOIN keyword returns all records from the right table (table2), and the matching records from the left table (table1). The result is 0 records from the left side if there is no match.

RIGHT JOIN

Note: In some databases RIGHT JOIN is called RIGHT OUTER JOIN.

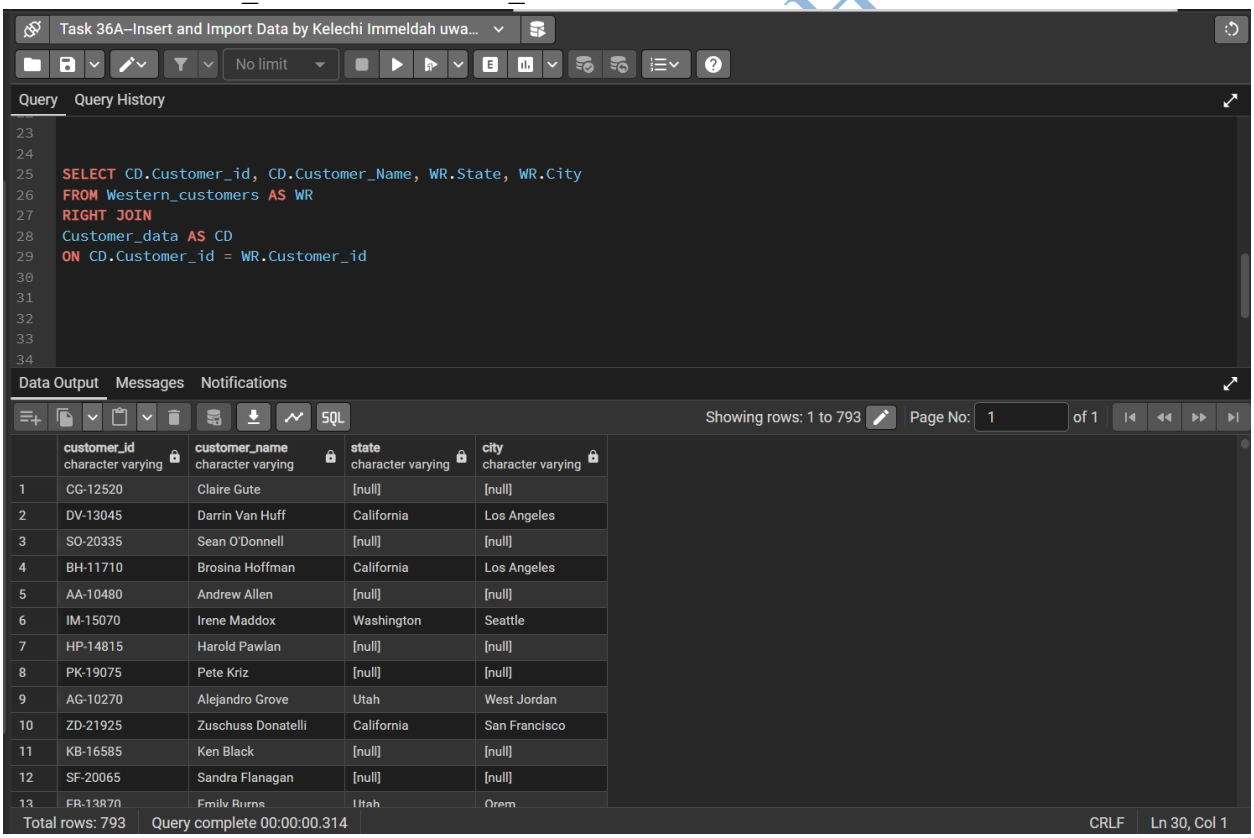


SQL RIGHT JOIN EXAMPLE

The following SQL statement will be returned to all customers, and any orders they might have placed.

Example

```
SELECT CD. Customer_id, CD. Customer_Name, WR. State, WR. City
FROM Western_customers AS WR
RIGHT JOIN
Customer_data AS CD
ON CD. Customer_id = WR. Customer_id
```

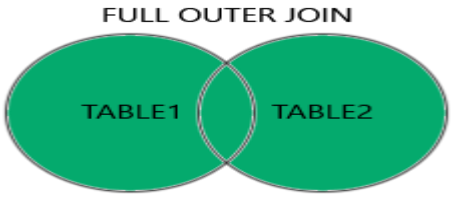



Note: The RIGHT JOIN keyword returns all records from the right table (Western Customers), even if there are no matches in the left table (Customer data).

SQL FULL OUTER JOIN KEYWORD

THE FULL OUTER JOIN keyword returns all records when there is a match on left (table1) or right (table2) table records.

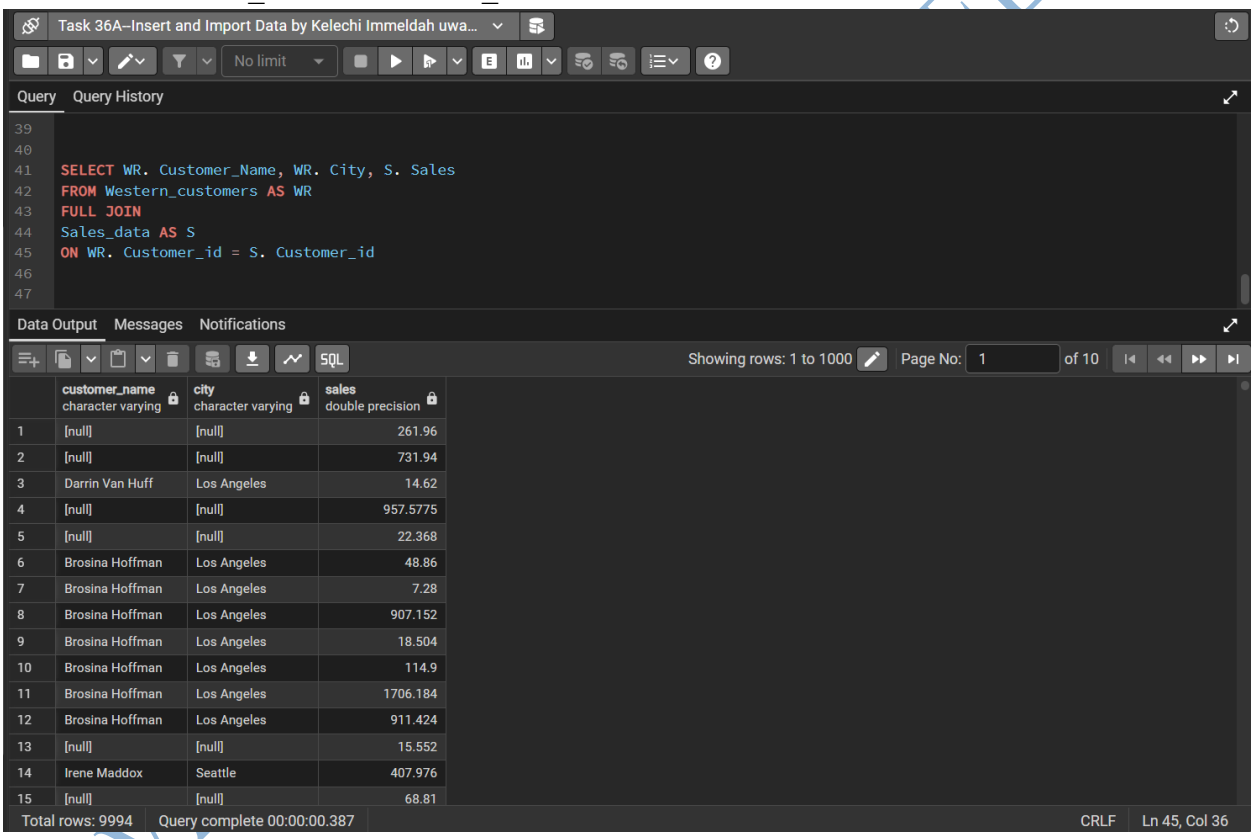
Tip: FULL OUTER JOIN and FULL JOIN are the same.



Note: FULL OUTER JOIN can potentially return very large result sets.

SQL FULL OUTER JOIN EXAMPLE

```
SELECT WR. Customer_Name, WR. City, S. Sales
FROM Western_customers AS WR
FULL JOIN
Sales_data AS S
ON WR. Customer_id = S. Customer_id
```



A selection from the result set may look like this:

Note: The FULL OUTER JOIN keyword returns all matching records from both tables whether the other table matches or not. So, if there are rows in "Wester Customers" that do not have matches in "Sales data", or if there are rows in "Sales data" that do not have matches in "Western Customers", those rows will be listed as well.

SQL SELF JOIN

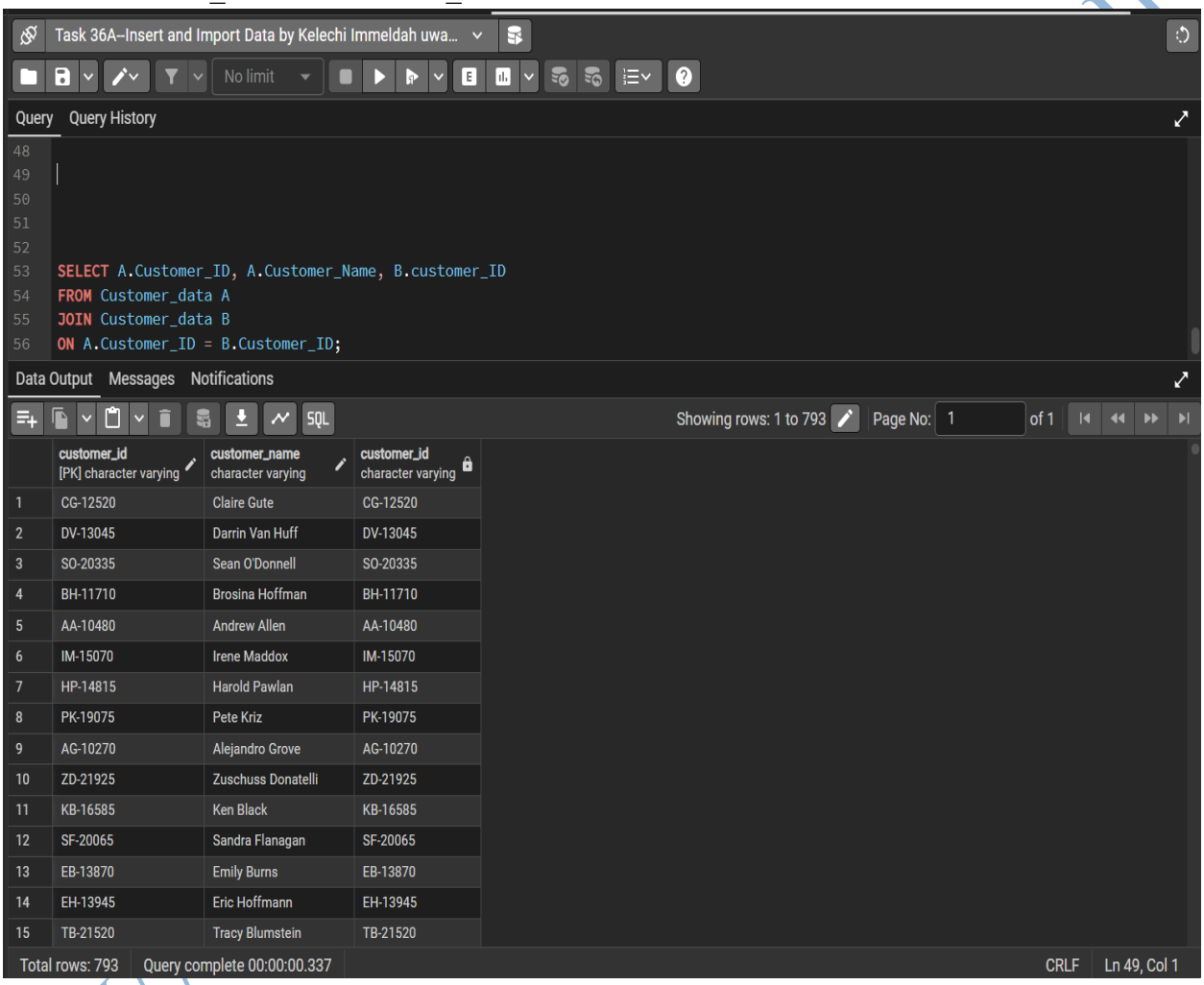
A SELF JOIN is a regular join, but the table is joined with itself.

SQL SELF-JOIN EXAMPLE

The following SQL statement matches customers who are from the same city.

Example

```
SELECT A. Customer_ID, A. Customer_Name, B. customer_ID
FROM Customer_data A
JOIN Customer_data B
ON A. Customer_ID = B. Customer_ID.
```



	customer_id [PK] character varying	customer_name character varying	customer_id character varying
1	CG-12520	Claire Gute	CG-12520
2	DV-13045	Darrin Van Huff	DV-13045
3	SO-20335	Sean O'Donnell	SO-20335
4	BH-11710	Brosina Hoffman	BH-11710
5	AA-10480	Andrew Allen	AA-10480
6	IM-15070	Irene Maddox	IM-15070
7	HP-14815	Harold Pawlan	HP-14815
8	PK-19075	Pete Kriz	PK-19075
9	AG-10270	Alejandro Grove	AG-10270
10	ZD-21925	Zuschuss Donatelli	ZD-21925
11	KB-16585	Ken Black	KB-16585
12	SF-20065	Sandra Flanagan	SF-20065
13	EB-13870	Emily Burns	EB-13870
14	EH-13945	Eric Hoffmann	EH-13945
15	TB-21520	Tracy Blumstein	TB-21520

SQL UNION OPERATOR

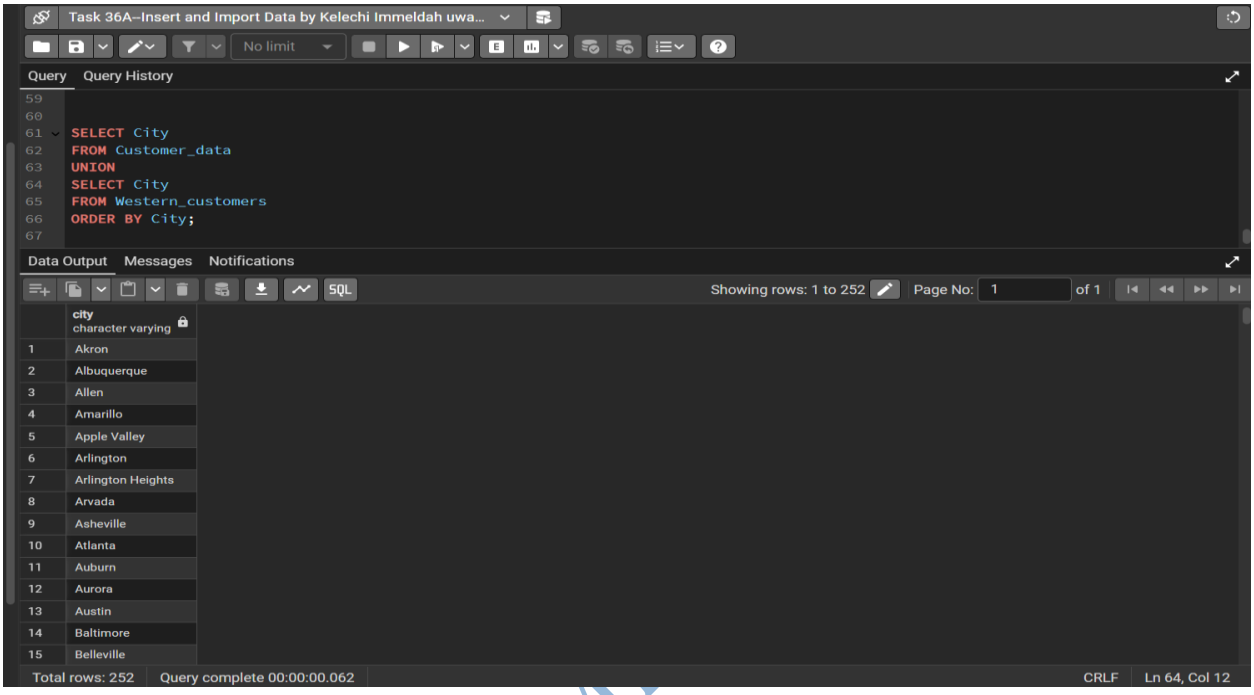
THE UNION OPERATOR combines the result set of two or more SELECT statements.

- Every SELECT statement within UNION must have the same number of columns
- The columns must also have similar data types
- The columns in every SELECT statement must also be in the same order

SQL UNION EXAMPLE

The following SQL statement returns the cities (only distinct values) from both the "Customer data" and the "Suppliers" table.

EXAMPLE
SELECT City FROM Customer_data
UNION
SELECT City FROM Sales_data
ORDER BY City;.

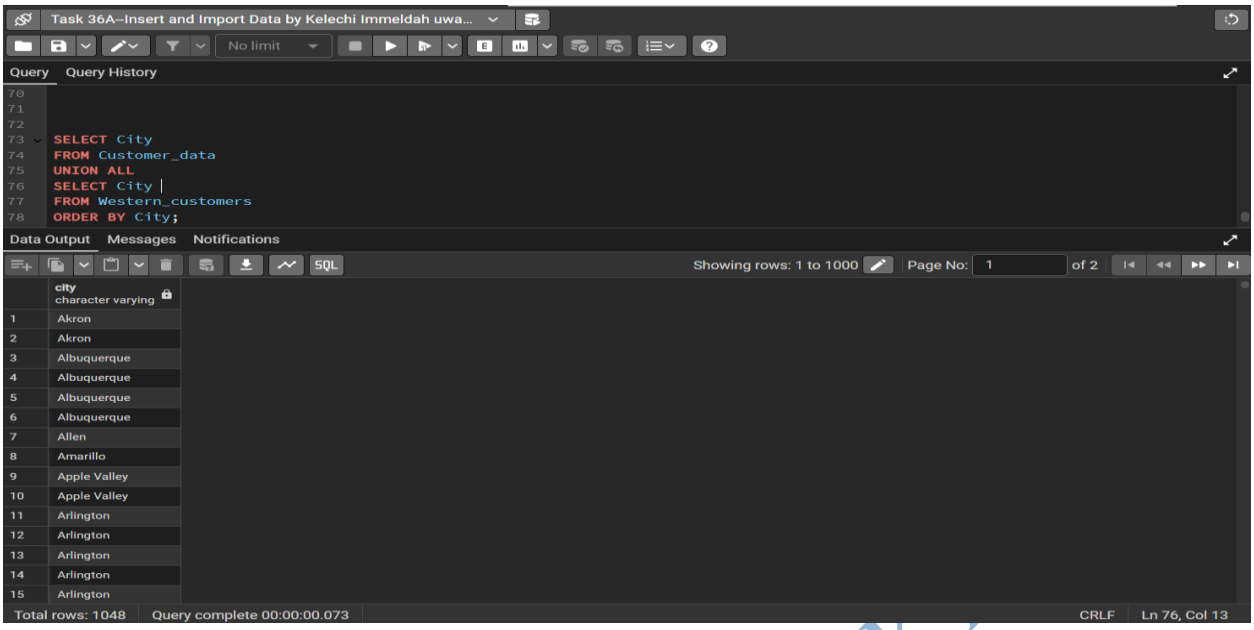


Note: If some customer_data or western_customers have the same city, each city will only be listed once, because UNION selects only distinct values. Use UNION ALL also to select duplicate values.

SQL UNION ALL EXAMPLE

The following SQL statement returns the cities (duplicate values also) from both the "Customer data" and the "Western Customers" table:

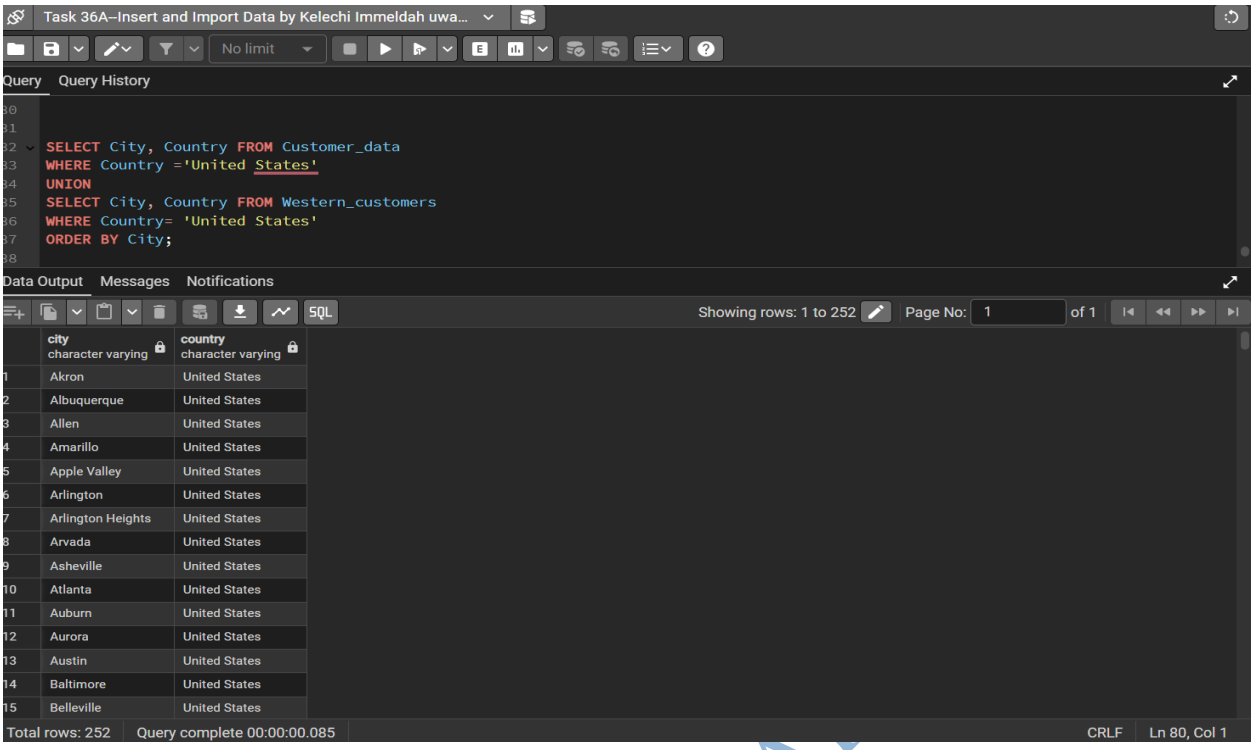
EXAMPLE
SELECT City FROM Customer_data
UNION ALL
SELECT City FROM Western_customers
ORDER BY City;



SQL UNION WITH WHERE

The following SQL statement returns the German cities (only distinct values) from both the "Customer data" and the "Western customer" table:

EXAMPLE
SELECT City, Country FROM Customer_data
WHERE Country ='United States'
UNION
SELECT City, Country FROM Western_customer
WHERE Country= 'United States'
ORDER BY City;

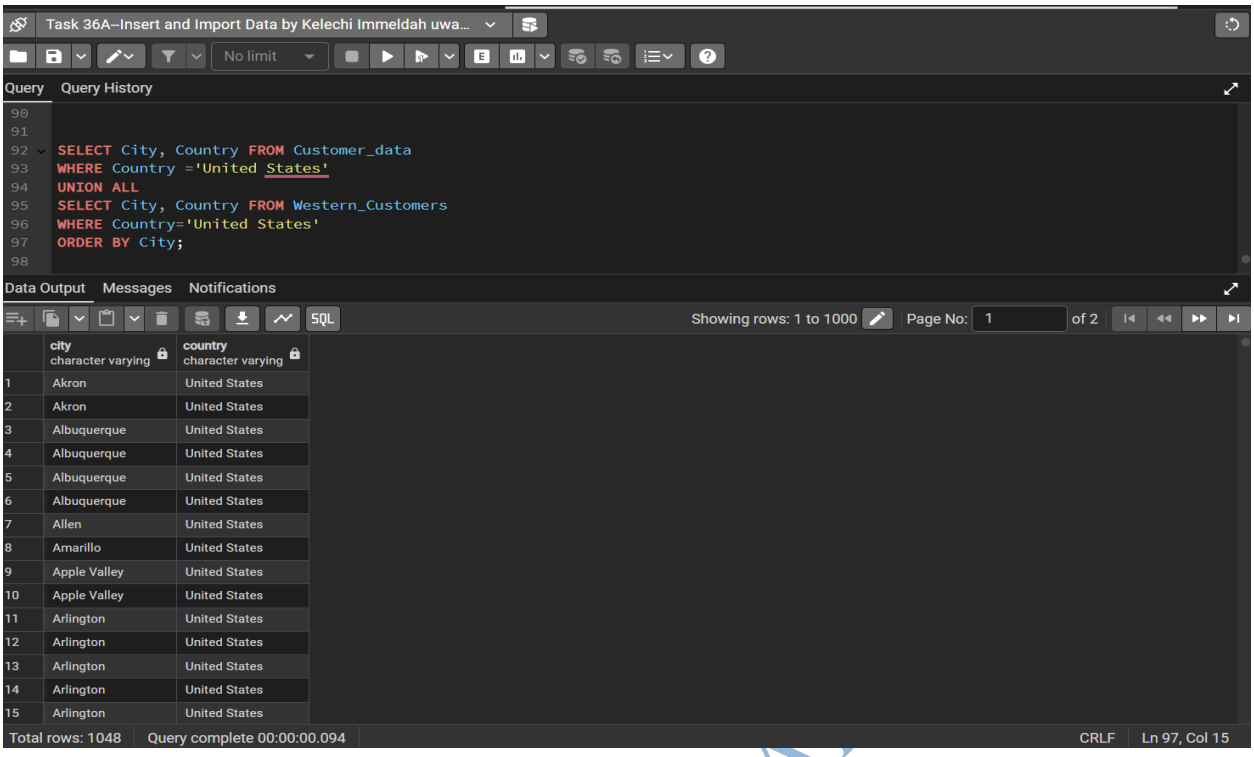


SQL UNION ALL WITH WHERE

The following SQL statement returns the German cities (duplicate values also) from both the "Customer data" and the "Western Customers" table:

EXAMPLE

```
SELECT City, Country FROM Customer_data
WHERE Country ='United States'
UNION ALL
SELECT City, Country FROM Western_Customers
WHERE Country='United States'
ORDER BY City;
```

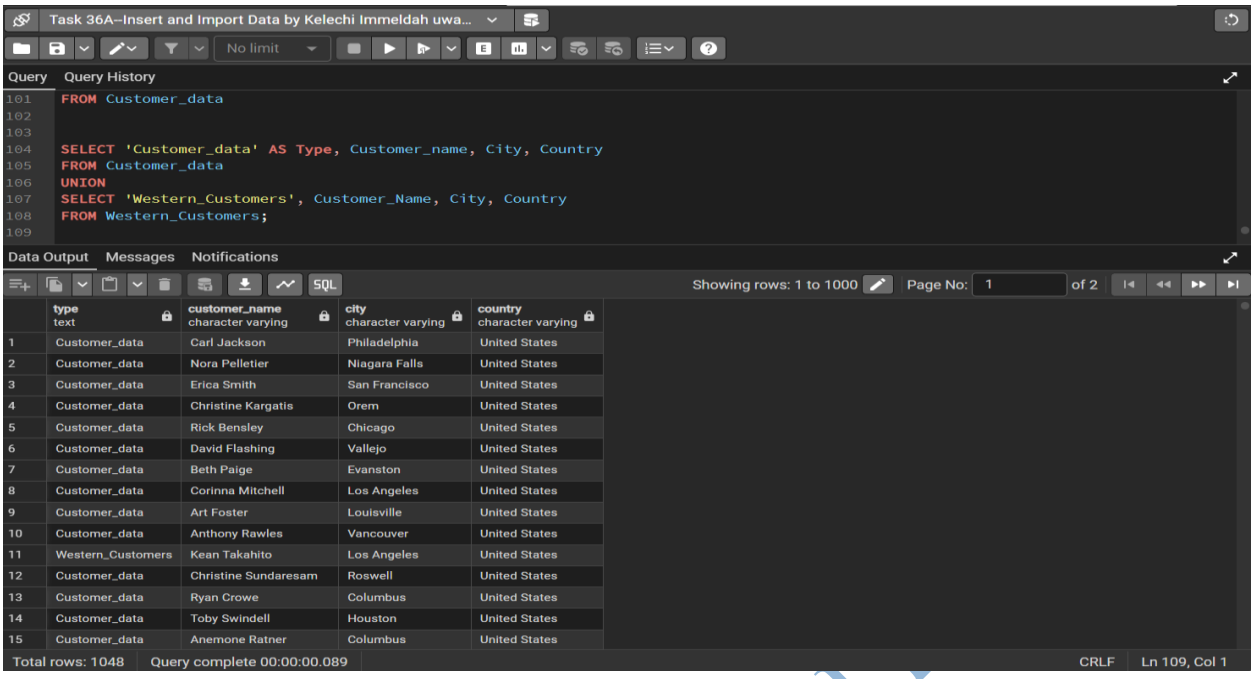


ANOTHER UNION EXAMPLE

The following SQL statement lists all customers and suppliers:

EXAMPLE

```
SELECT 'Customer_data' AS Type, Customer_name, City, Country
FROM Customer_data
UNION
SELECT 'Western_Customers', Customer_Name, City, Country
FROM Western_Customers;
```



SQL GROUP BY STATEMENT

The GROUP BY statement groups rows that have the same values into summary rows, like "find the number of customers in each country".

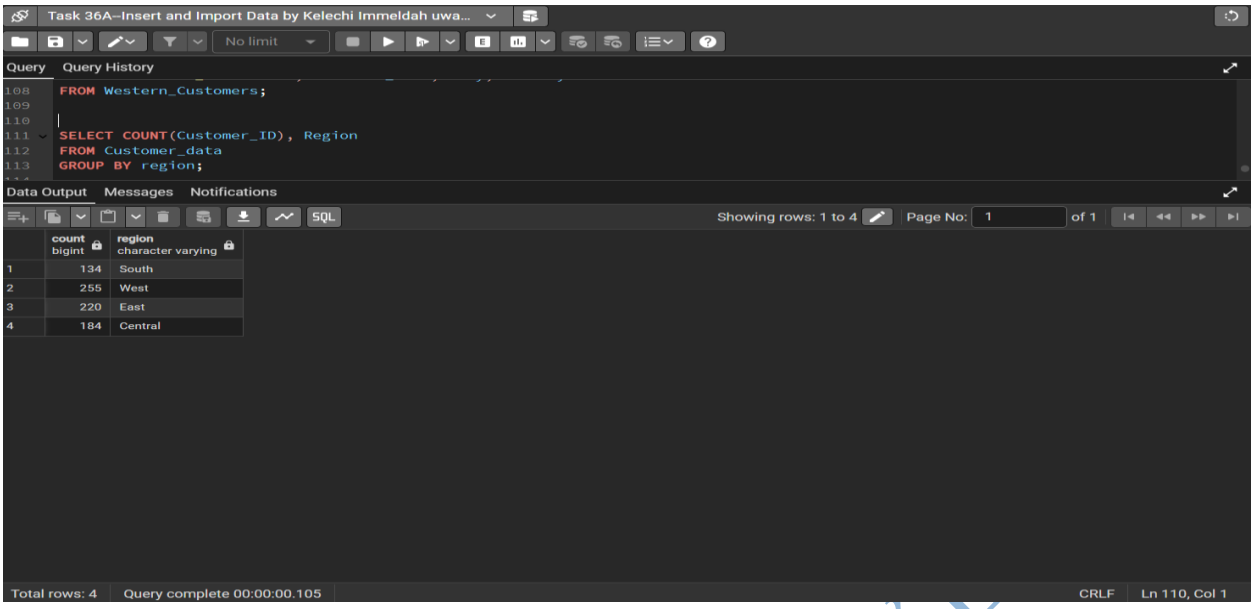
The GROUP BY statement is often used with aggregate functions (COUNT (), MAX (), MIN (), SUM (), AVG ()) to group the result set by one or more columns.

SQL GROUP BY EXAMPLES

The following SQL statement lists the number of customers in each region:

Example

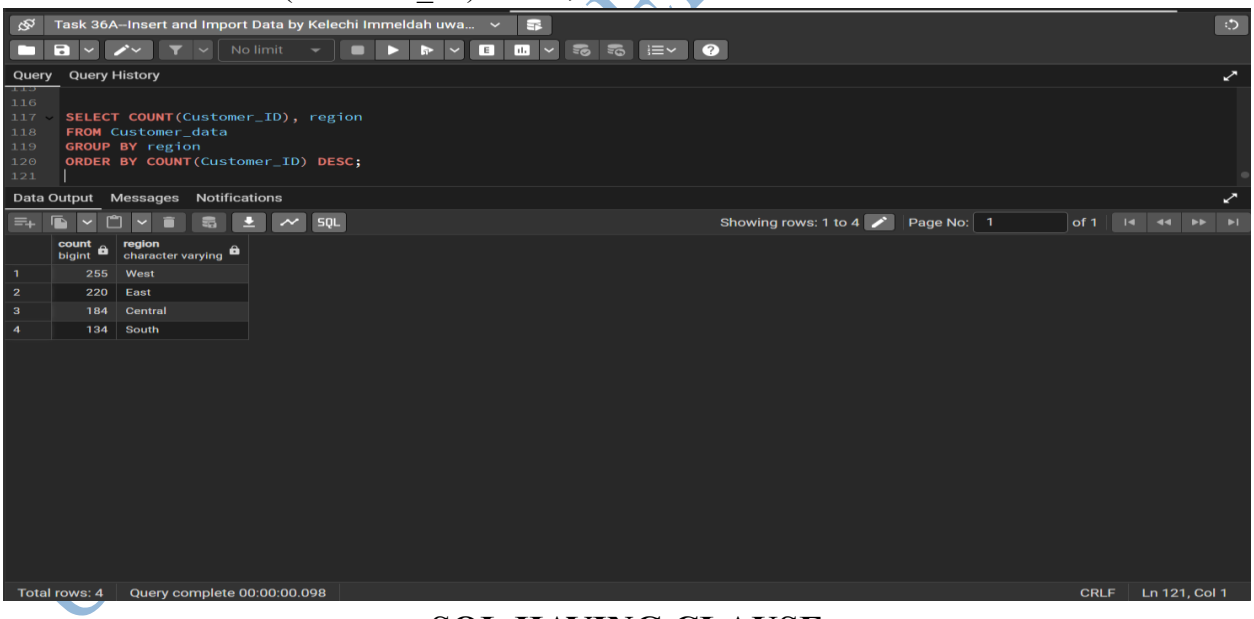
```
SELECT COUNT(Customer_ID), Region
FROM Customer_data
GROUP BY region;
```



The following SQL statement lists the number of customers in each country, sorted from high to low:

EXAMPLE

```
SELECT COUNT(Customer_ID), region
FROM Customer_data
GROUP BY region
ORDER BY COUNT(Customer_ID) DESC;
```



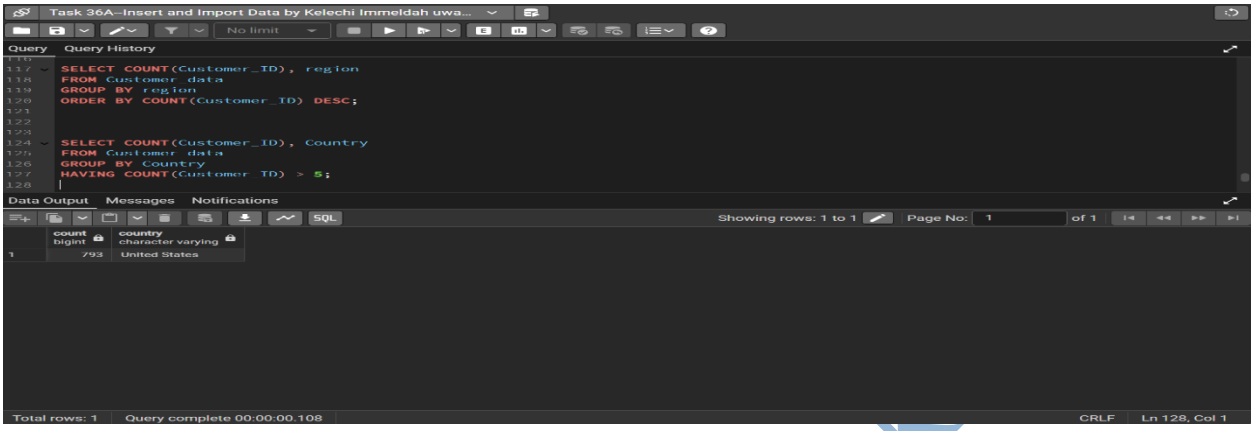
SQL HAVING CLAUSE

THE HAVING CLAUSE was added to SQL because the WHERE keyword cannot be used with aggregate functions.

SQL HAVING EXAMPLES

The following SQL statement lists the number of customers in each country. Only include countries with more than 5 customers:

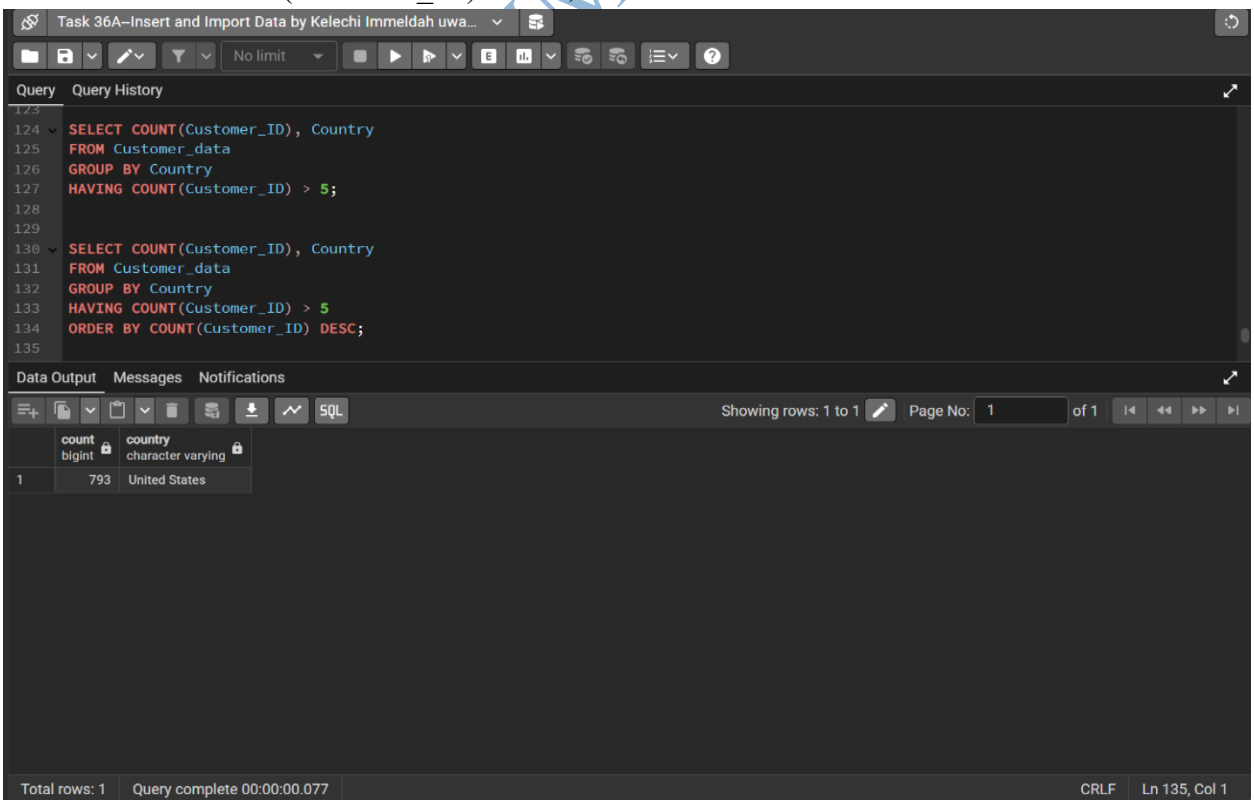
EXAMPLE
SELECT COUNT(Customer_ID), Country
FROM Customer_data
GROUP BY Country
HAVING COUNT(Customer_ID) > 5;



The following SQL statement lists the number of customers in each country, sorted from high to low (Only include countries with more than 5 customers):

EXAMPLE

SELECT COUNT(Customer_ID), Country
FROM Customer_data
GROUP BY Country
HAVING COUNT(Customer_ID) > 5
ORDER BY COUNT(Customer_ID) DESC;



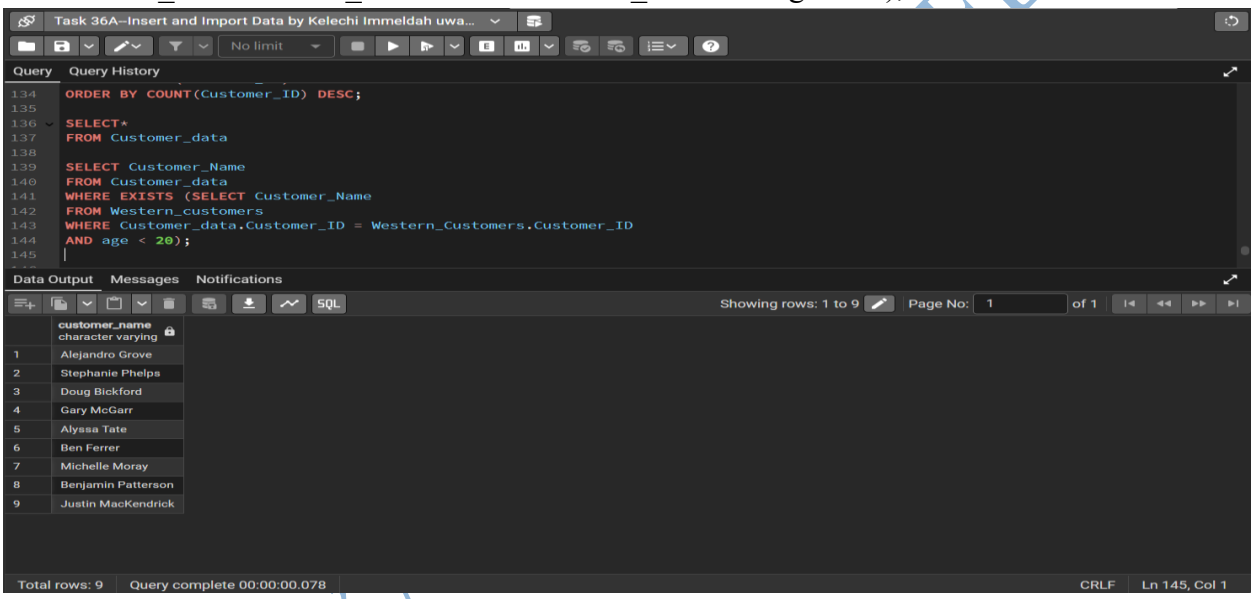
SQL EXISTS OPERATOR

THE EXISTS OPERATOR is used to test for the existence of any record in a subquery.
The EXISTS operator returns TRUE if the subquery returns one or more records.

SQL EXISTS EXAMPLES

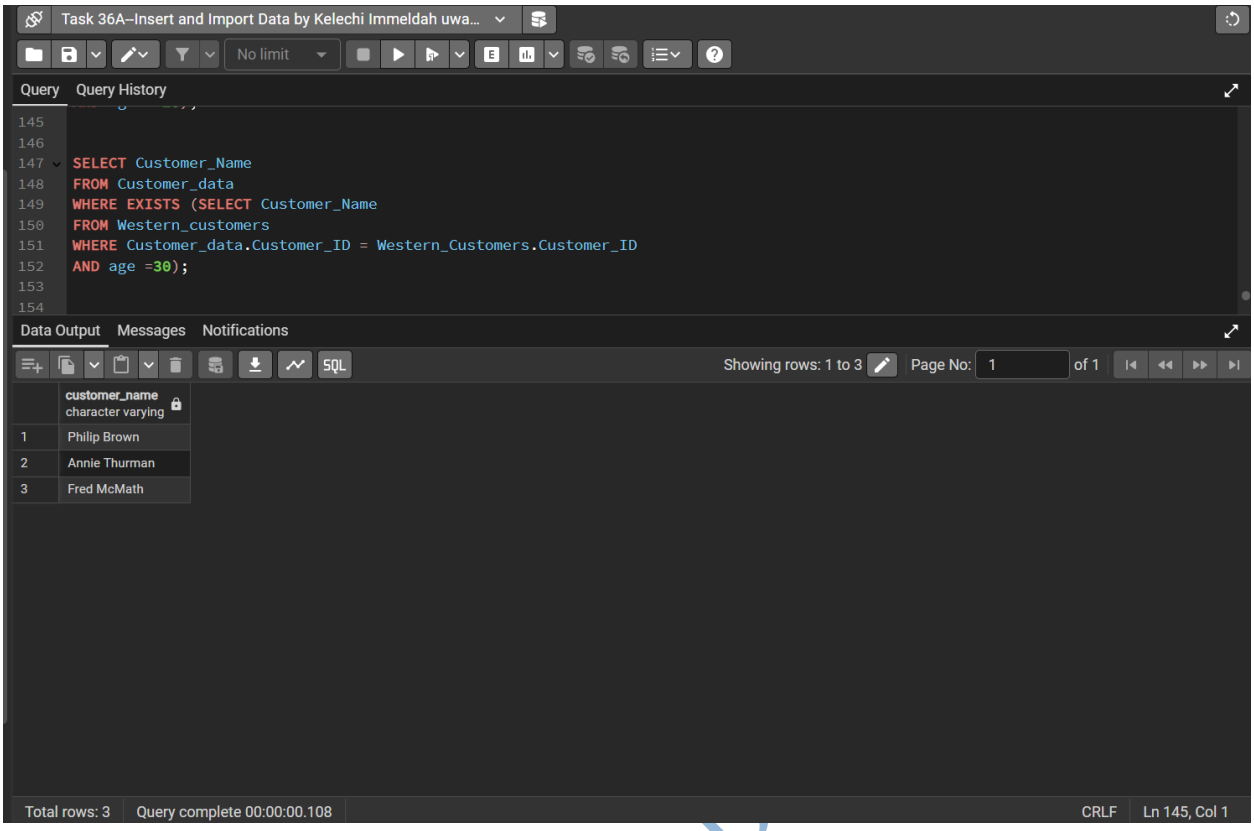
The following SQL statement returns TRUE and lists the Western customers with an Age that is less than 20:

EXAMPLE
SELECT Customer_Name
FROM Customer_data
WHERE EXISTS (SELECT Customer_Name FROM Western_customers WHERE Customer_data.Customer_ID = Western_Customers.Customer_ID AND Age < 20);



The following SQL statement returns TRUE and lists the Customers with an Age that is equal to 22:

EXAMPLE
SELECT Customer_Name
FROM Customer_data
WHERE EXISTS (SELECT Customer_Name
FROM Western_customers
WHERE Customer_data.Customer_ID = Western_Customers.Customer_ID
AND age =30);



SQL ANY AND ALL OPERATORS

THE ANY AND ALL OPERATORS allow you to perform a comparison between a single column value and a range of other values.

THE SQL ANY OPERATOR

THE ANY OPERATOR:

- returns a Boolean value as a result
- returns TRUE if ANY of the subquery values meet the condition

ANY means that the condition will be true if the operation is true for any of the values in the range.

Note: The *operator* must be a standard comparison operator (=, <>, !=, >, >=, <, or <=).

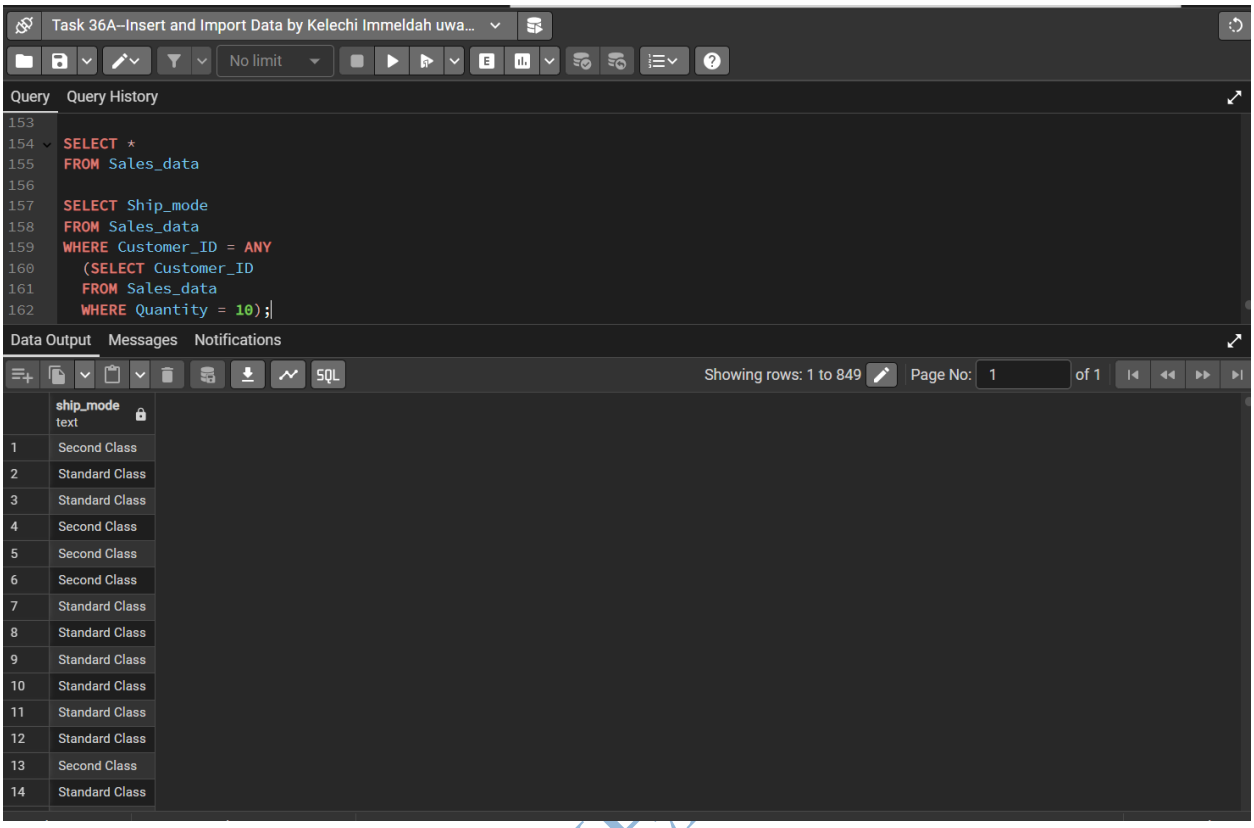
SQL ANY EXAMPLES

The following SQL statement lists the ship_mode if it finds ANY records in the sales data table that have Quantity equal to 10 (this will return TRUE because the Quantity column has some values of 10):

EXAMPLE

```
SELECT Ship_mode
FROM Sales_data
WHERE Customer_ID = ANY
  (SELECT Customer_ID
```

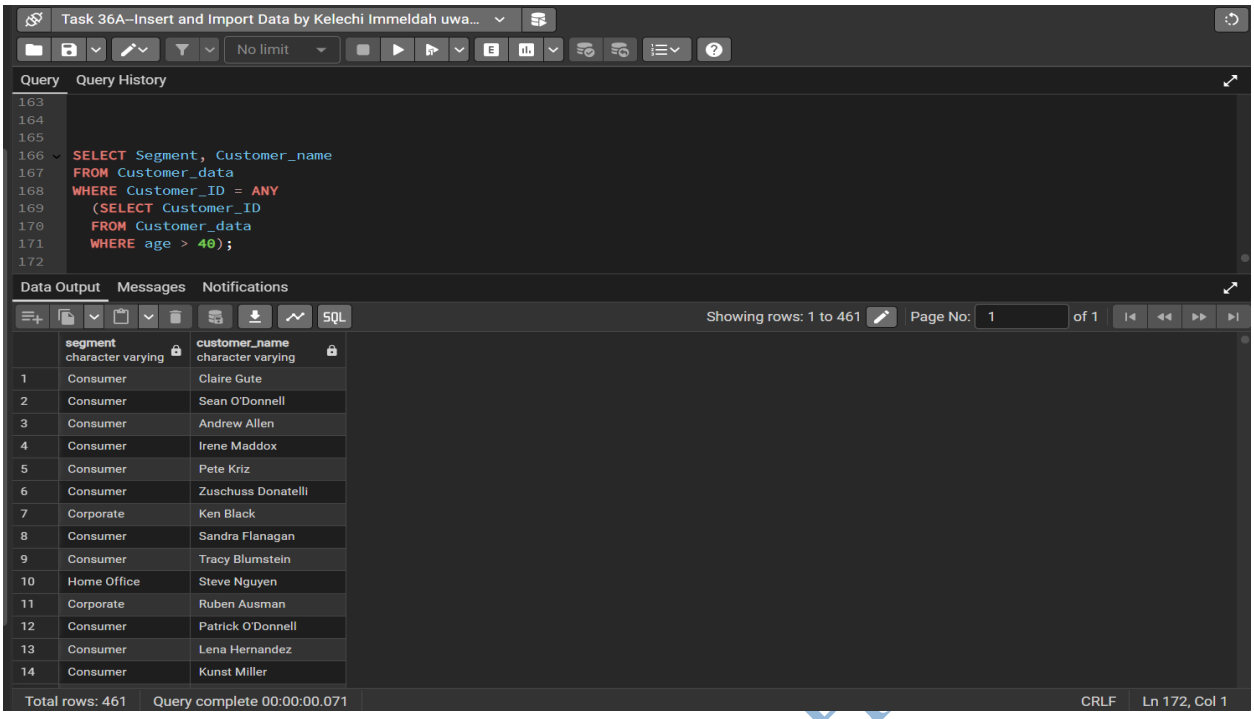
FROM Sales_data
WHERE Quantity = 10);



The following SQL statement lists the segment if it finds ANY records in the customer data table that have an age larger than 40 (this will return TRUE because the Age column has some values larger than 40):

EXAMPLE

SELECT Segment, Customer_name
FROM Customer_data
WHERE Customer_ID = ANY
(SELECT Customer_ID
FROM Customer_data
WHERE age > 40);



THE SQL ALL OPERATOR

THE ALL OPERATOR:

- returns a boolean value as a result
- returns TRUE if ALL of the subquery values meet the condition
- is used with SELECT, WHERE, and HAVING statements

ALL means that the condition will be true only if the operation is true for all values in the range.

Note: The *operator* must be a standard comparison operator (=, <>, !=, >, >=, <, or <=).

SQL ALL EXAMPLES

The following SQL statement lists ALL the customer names:

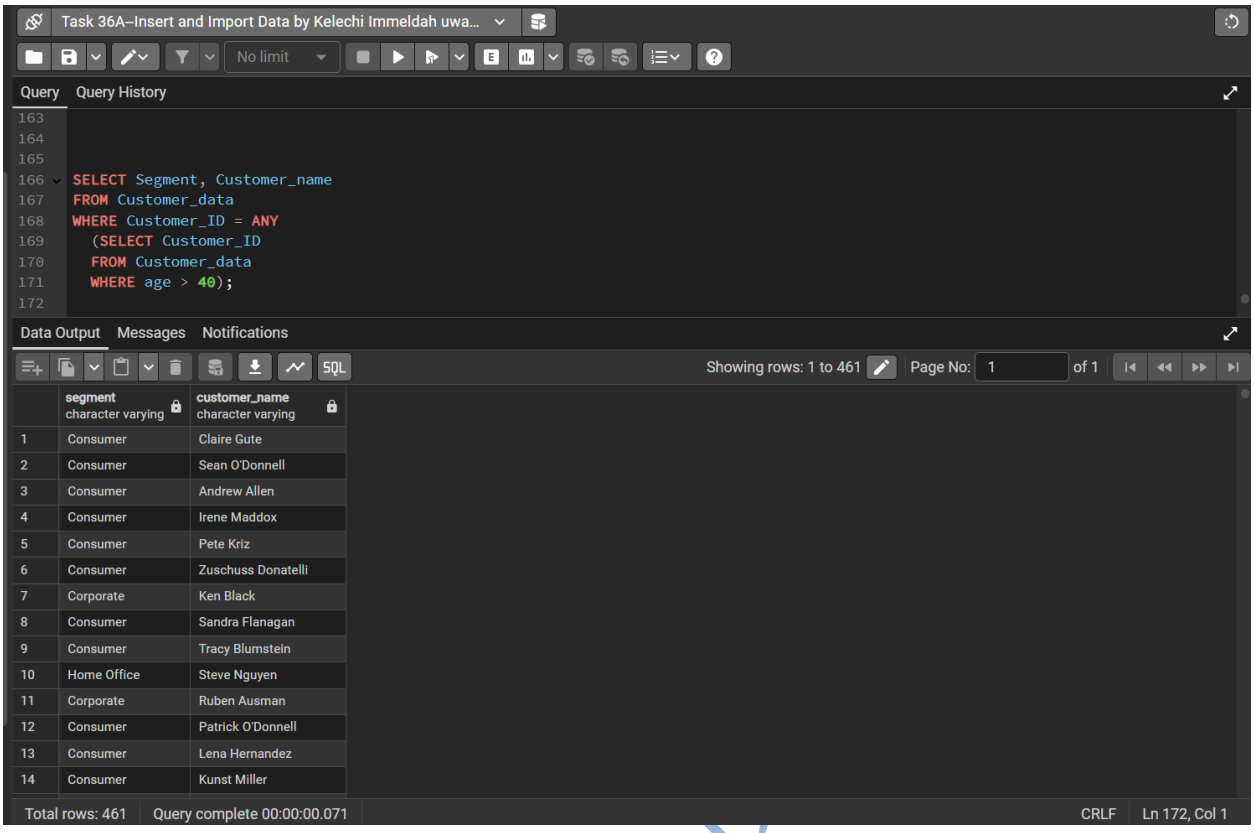
EXAMPLE

```
SELECT ALL Customer_Name
FROM Customer_data
WHERE TRUE;
```

The following SQL statement lists the customer Name if ALL the records in the customer data table have an Age equal to 20. This will of course return FALSE because the age column has many different values (not only the value of 20):

EXAMPLE

```
SELECT Customer_Name
FROM Customer_data
WHERE Customer_ID = ALL
  (SELECT Customer_ID
  FROM Customer_data
  WHERE age = 20);
```



SQL SELECT INTO STATEMENT

THE SELECT INTO STATEMENT copies data from one table into a new table.

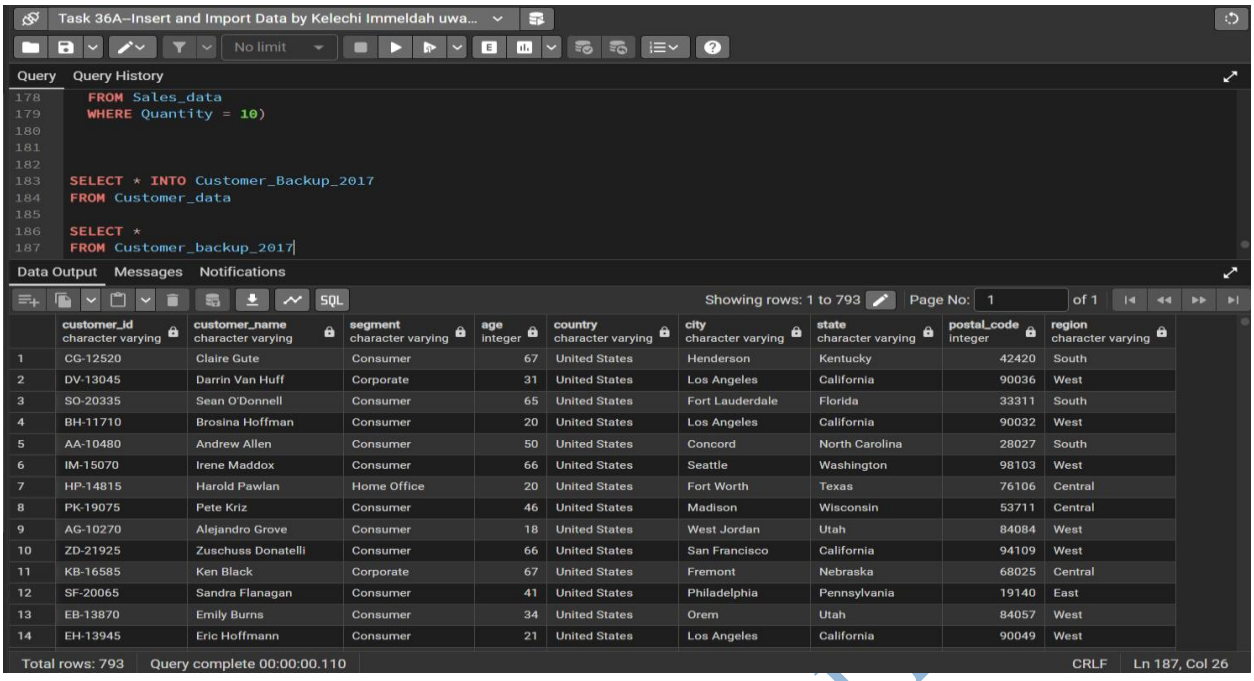
SELECT INTO

The new table will be created with the column names and types as defined in the old table. You can create new column names using the AS clause.

SQL SELECT INTO EXAMPLES

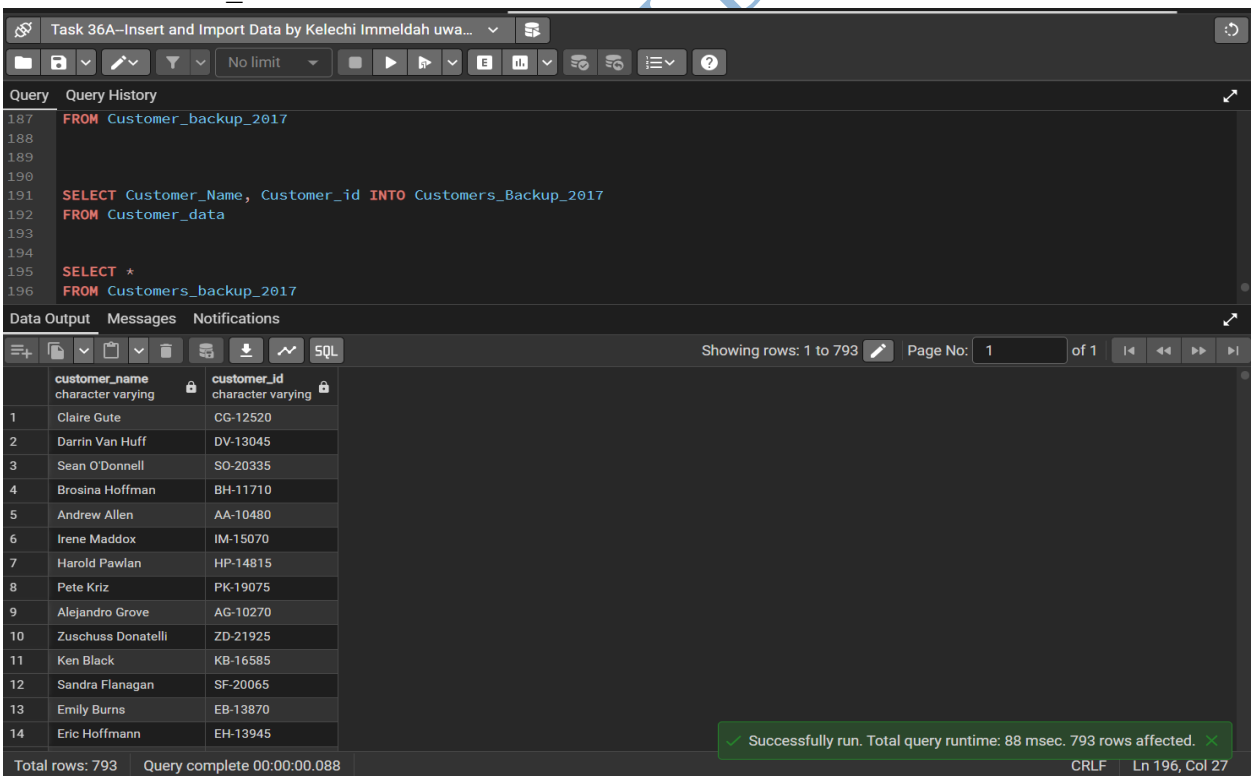
The following SQL statement creates a backup copy of Customers:

```
SELECT * INTO Customer_Backup_2017
FROM Customer_data
```



The following SQL statement copies only a few columns into a new table:

SELECT Customer_Name, Customer_id
INTO Customers_Backup_2017
FROM Customer_data



SQL INSERT INTO SELECT STATEMENT

THE INSERT INTO SELECT STATEMENT copies data from one table and inserts it into another table.

The INSERT INTO SELECT statement requires that the data types in the source and target tables match.

Note: The existing records in the target table are unaffected.

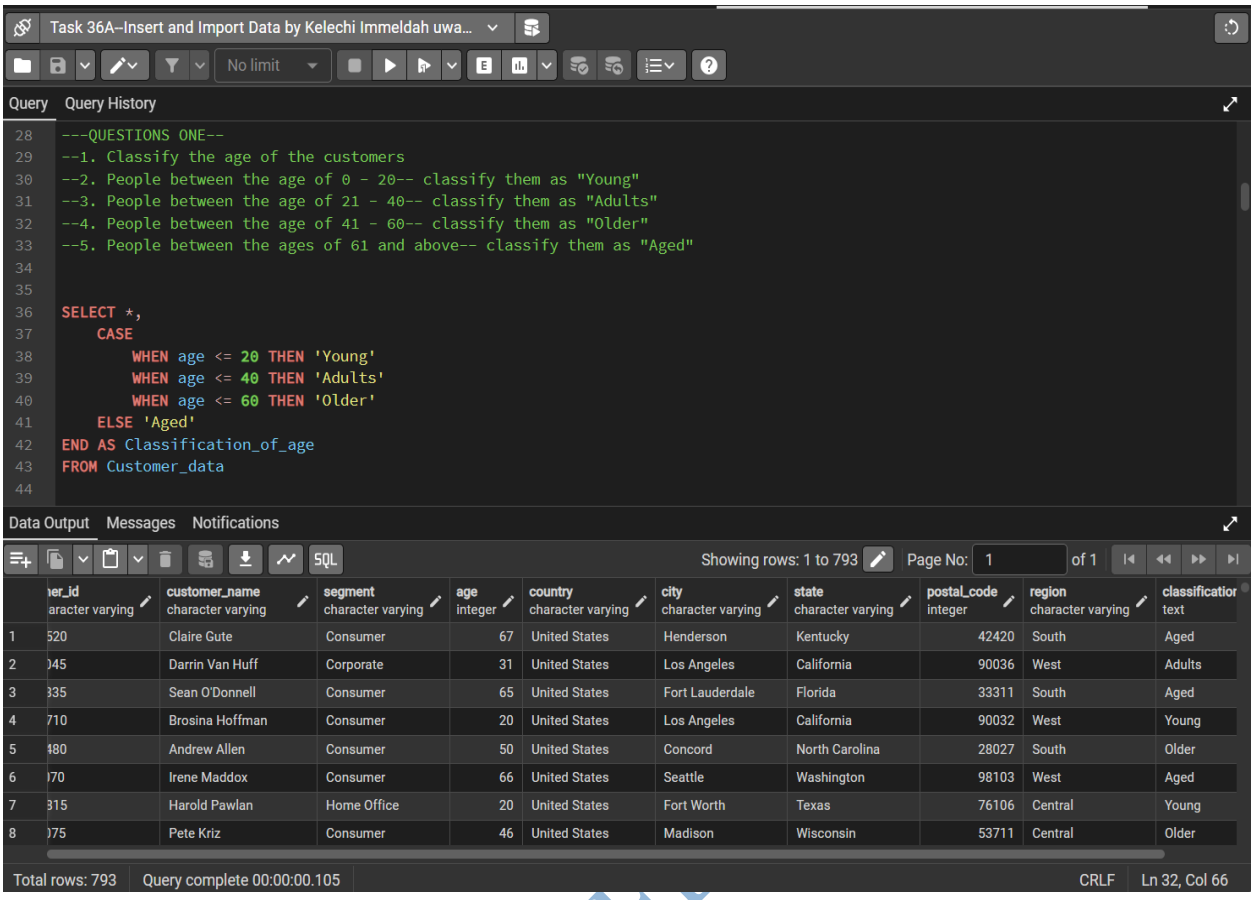
SQL CASE EXPRESSION

The CASE expression goes through conditions and returns a value when the first condition is met (like an if-then-else statement). So, once a condition is true, it will stop reading and return the result. If no conditions are true, it returns the value in the ELSE clause.

If there is no ELSE part and no conditions are true, it returns NULL.

EXAMPLE OF CASE EXPRESSION

```
--1. Classify the age of the customers
--2. People between the age of 0 - 20-- classify them as "Young"
--3. People between the age of 21 - 40-- classify them as "Adults"
--4. People between the age of 41 - 60-- classify them as "Older"
--5. People between the ages of 61 and above-- classify them as "Aged"
SELECT *,
        CASE
                WHEN age <= 20 THEN 'Young'
                WHEN age <= 40 THEN 'Adults'
                WHEN age <= 60 THEN 'Older'
        ELSE 'Aged'
END AS Classification_of_age
FROM Customer_data
```



	customer_name character varying	segment character varying	age integer	country character varying	city character varying	state character varying	postal_code integer	region character varying	classification text
1	Claire Gute	Consumer	67	United States	Henderson	Kentucky	42420	South	Aged
2	Darrin Van Huff	Corporate	31	United States	Los Angeles	California	90036	West	Adults
3	Sean O'Donnell	Consumer	65	United States	Fort Lauderdale	Florida	33311	South	Aged
4	Brosina Hoffman	Consumer	20	United States	Los Angeles	California	90032	West	Young
5	Andrew Allen	Consumer	50	United States	Concord	North Carolina	28027	South	Older
6	Irene Maddox	Consumer	66	United States	Seattle	Washington	98103	West	Aged
7	Harold Pawlan	Home Office	20	United States	Fort Worth	Texas	76106	Central	Young
8	Pete Kriz	Consumer	46	United States	Madison	Wisconsin	53711	Central	Older

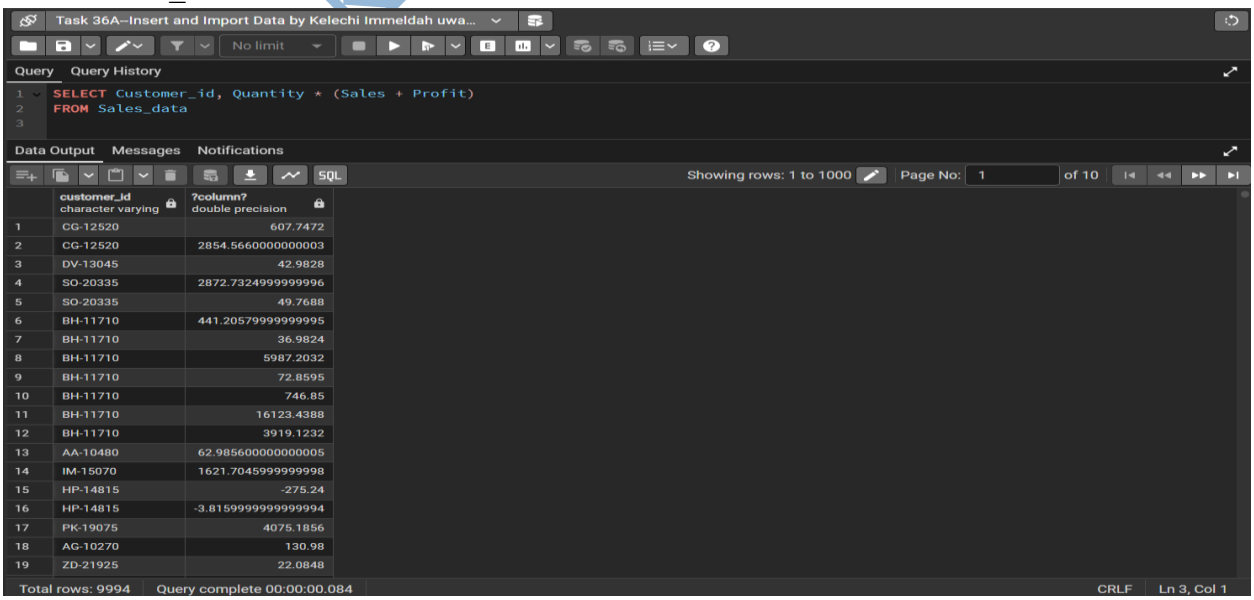
SQL NULL FUNCTIONS

SQL IFNULL (), ISNULL (), COALESCE (), and NVL () Functions

Suppose that the "Profit" column is optional, and may contain NULL values.

Look at the following SELECT statement:

```
SELECT Customer_id, Quantity * (Sales + Profit)
FROM Sales_data
```



	customer_id character varying	?column? double precision
1	CG-12520	607.7472
2	CG-12520	2854.5660000000003
3	DV-13045	42.9828
4	SO-20335	2872.7324999999996
5	SO-20335	49.7688
6	BH-11710	441.20579999999995
7	BH-11710	36.9824
8	BH-11710	5987.2032
9	BH-11710	72.8595
10	BH-11710	746.85
11	BH-11710	16123.4388
12	BH-11710	3919.1232
13	AA-10480	62.985600000000005
14	IM-15070	1621.7045999999998
15	HP-14815	-275.24
16	HP-14815	-3.8159999999999994
17	PK-19075	4075.1856
18	AG-10270	130.98
19	ZD-21925	22.0848

In the example above, if any of the "profit" values are NULL, the result will be NULL.

SQL STORED PROCEDURES FOR SQL SERVER

WHAT IS A STORED PROCEDURE?

A stored procedure is a prepared SQL code that you can save, so the code can be reused over and over again.

So, if you have an SQL query that you write over and over again, save it as a stored procedure, and then just call it to execute it.

You can also pass parameters to a stored procedure so that the stored procedure can act based on the parameter value(s) that is passed.

SQL COMMENTS

COMMENTs are used to explain sections of SQL statements, or to prevent execution of SQL statements.

Note: Comments are not supported in Microsoft Access databases!

SINGLE LINE COMMENTS

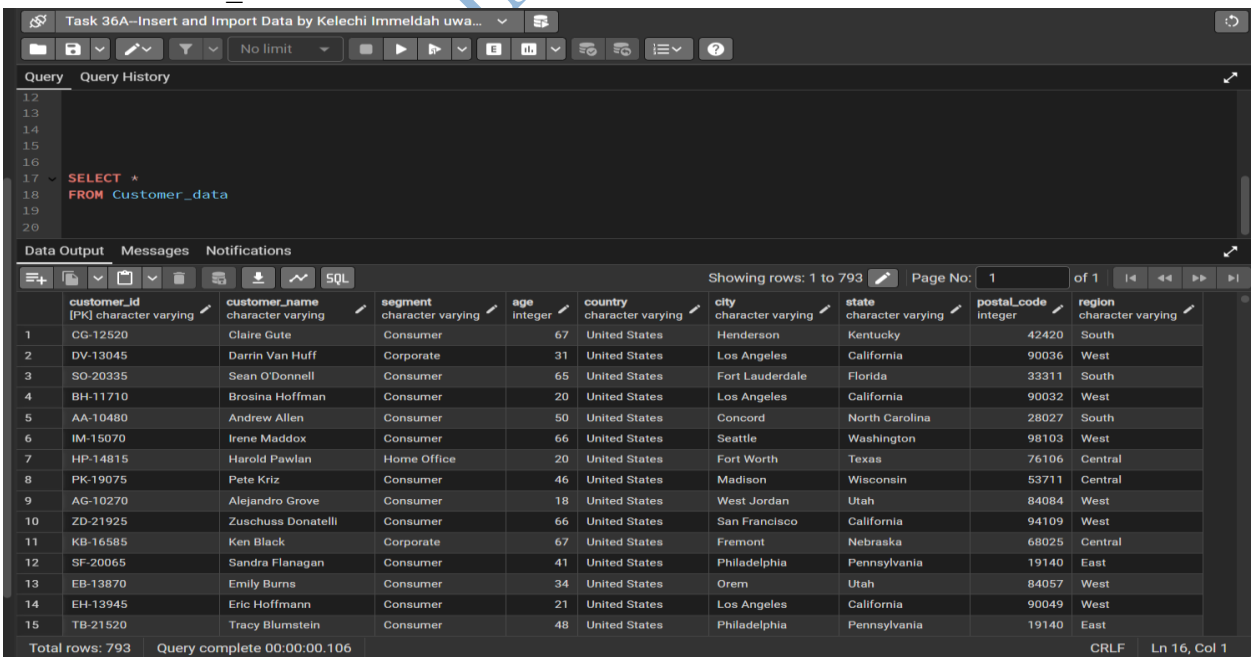
Single-line comments start with --.

Any text between -- and the end of the line will be ignored (will not be executed).

The following example uses a single-line comment as an explanation:

EXAMPLE

```
-- Select all:
SELECT *
FROM Customer_data
```

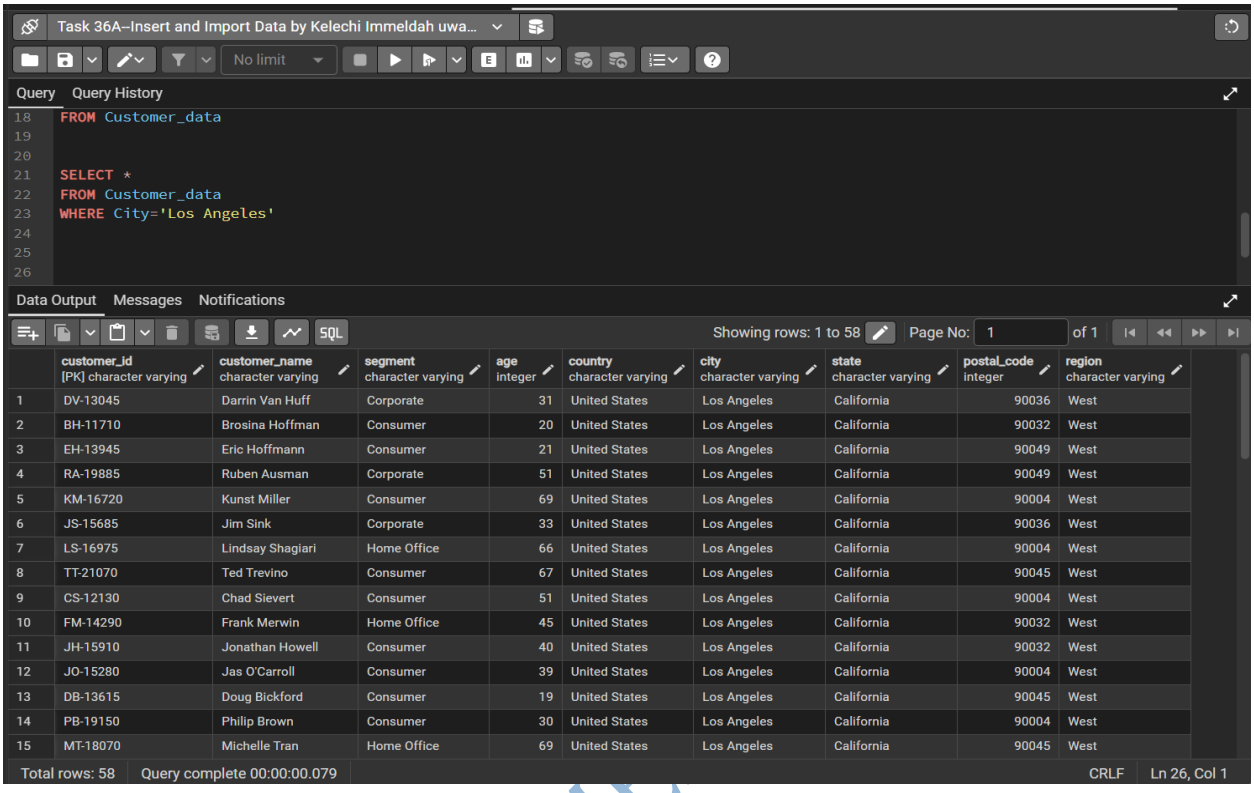


	customer_id [PK] character varying	customer_name character varying	segment character varying	age integer	country character varying	city character varying	state character varying	postal_code integer	region character varying
1	CG-12520	Claire Gute	Consumer	67	United States	Henderson	Kentucky	42420	South
2	DV-13045	Darrin Van Huff	Corporate	31	United States	Los Angeles	California	90036	West
3	SO-20335	Sean O'Donnell	Consumer	65	United States	Fort Lauderdale	Florida	33311	South
4	BH-11710	Brosina Hoffman	Consumer	20	United States	Los Angeles	California	90032	West
5	AA-10480	Andrew Allen	Consumer	50	United States	Concord	North Carolina	28027	South
6	IM-15070	Irene Maddox	Consumer	66	United States	Seattle	Washington	98103	West
7	HP-14815	Harold Pawlan	Home Office	20	United States	Fort Worth	Texas	76106	Central
8	PK-19075	Pete Kriz	Consumer	46	United States	Madison	Wisconsin	53711	Central
9	AG-10270	Alejandro Grove	Consumer	18	United States	West Jordan	Utah	84084	West
10	ZD-21925	Zuschuss Donatelli	Consumer	66	United States	San Francisco	California	94109	West
11	KB-16585	Ken Black	Corporate	67	United States	Fremont	Nebraska	68025	Central
12	SF-20065	Sandra Flanagan	Consumer	41	United States	Philadelphia	Pennsylvania	19140	East
13	EB-13870	Emily Burns	Consumer	34	United States	Orem	Utah	84057	West
14	EH-13945	Eric Hoffmann	Consumer	21	United States	Los Angeles	California	90049	West
15	TB-21520	Tracy Blumstein	Consumer	48	United States	Philadelphia	Pennsylvania	19140	East

The following example uses a single-line comment to ignore the end of a line:

EXAMPLE

```
SELECT *
FROM Customer_data
WHERE City='Los Angeles'
```



SQL OPERATORS

SQL ARITHMETIC OPERATORS

OPERATOR	DESCRIPTION
+	Add
-	Subtract
*	Multiply
/	Divide
%	Modulo

SQL BITWISE OPERATORS

OPERATOR	DESCRIPTION

&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR

SQL COMPARISON OPERATORS

OPERATOR	DESCRIPTION
=	Equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
<>	Not equal to

SQL COMPOUND OPERATORS

OPERATOR	DESCRIPTION
+=	Add equals
-=	Subtract equals
*=	Multiply equals
/=	Divide equals
%=	Modulo equals
&=	Bitwise AND equals
^-=	Bitwise exclusive equals
|*=	Bitwise OR equals

SQL LOGICAL OPERATORS

OPERATOR	DESCRIPTION
ALL	TRUE if all of the subquery values meet the condition
AND	TRUE if all the conditions are separated by AND is TRUE
ANY	TRUE if any of the subquery values meet the condition
BETWEEN	TRUE if the operand is within the range of comparisons
EXISTS	TRUE if the subquery returns one or more records
IN	TRUE if the operand is equal to one of a list of expressions
LIKE	TRUE if the operand matches a pattern
NOT	Displays a record if the condition(s) is NOT TRUE
OR	TRUE if any of the conditions are separated by OR is TRUE
SOME	TRUE if any of the subquery values meet the condition