

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
```

The screenshot shows a MySQL Workbench window titled "Task 36A-Insert and Import Data by Kelechi Immedah uwa...". The query editor contains the following SQL code:

```

1 SELECT *
2 FROM Sales_data

```

The results grid displays 1000 rows of data from the "Sales_data" table. The columns and their data types are:

	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 precision
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	132.8
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

Total rows: 9994 Query complete 00:00:00.334 CRLF Ln 2, Col 16

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:

```

Task 36A-Insert and Import Data by Kelechi Immeldah uwa...
No limit
Query History
1 SELECT *
2 FROM Sales_data

Data Output Messages Notifications
Showing rows: 1 to 1000 Page No: 1 of 10
order_line order_id order_date ship_date ship_mode customer_id product_id sales quantity discount profit
1 1 CA-2016-152156 2016-11-08 2016-11-11 Second Class CG-12520 FUR-BO-100017... 261.96 2 0 41.5
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 6.6
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 2.5
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 1.5
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

Total rows: 9994 Query complete 00:00:00.334 CRLF Ln 2, Col 16

```

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;
```

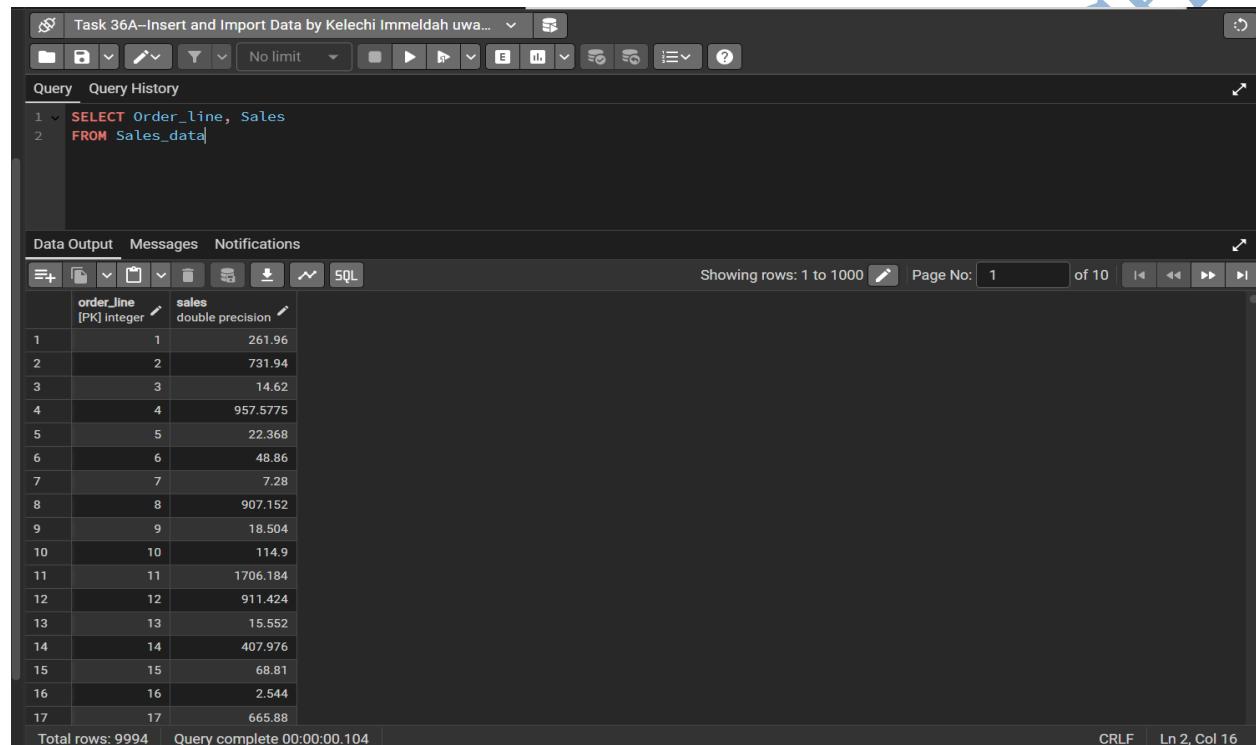
...

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;
```



The screenshot shows a MySQL Workbench interface with a query editor and a results grid. The query editor contains the following SQL code:

```
1 SELECT Order_line, Sales  
2 FROM Sales_data
```

The results grid displays two columns: 'order_line' and 'sales'. The data is as follows:

order_line	sales
1	261.96
2	731.94
3	14.62
4	957.5775
5	22.368
6	48.86
7	7.28
8	907.152
9	18.504
10	114.9
11	1706.184
12	911.424
13	15.552
14	407.976
15	68.81
16	2.544
17	665.88

Total rows: 9994 Query complete 00:00:00.104 CRLF Ln 2, Col 16

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:

Task 36A–Insert and Import Data by Kelechi Immeldah uwa...

```
1+ SELECT *
2   FROM Customer_data
```

Data Output Messages Notifications

	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
16	MA-17560	Matt Abelman	Home Office	19	United States	Houston	Texas	77095	Central
17	GH-14485	Gene Hale	Corporate	28	United States	Richardson	Texas	75080	Central

Total rows: 793 | Query complete 00:00:00.112 | CRLF | Ln 2, Col 19

Example:

Select all the different Segment from the "Customer Data" table:

```
SELECT DISTINCT Segment
FROM Customer_data;
```

Task 36A–Insert and Import Data by Kelechi Immeldah uwa...

```
1+ SELECT DISTINCT Segment
2   FROM Customer_data
```

Data Output Messages Notifications

	segment character varying
1	Consumer
2	Corporate
3	Home Office

Showing rows: 1 to 3 | Page No: 1 of 1 | CRLF | Ln 2, Col 19

Total rows: 3 | Query complete 00:00:00.087 | CRLF | Ln 2, Col 19

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;
```

The screenshot shows a SQL query window with the following content:

```
Task 36A--Insert and Import Data by Kelechi Immeldah uwa...
No limit
Query History
1 ~ SELECT Segment
2   FROM Customer_data
```

Below the query window is a results grid titled "Data Output". The grid has one column labeled "segment" with the data type "character varying". The data consists of 793 rows, each containing one of the following values:

segment
Consumer
Corporate
Consumer
Consumer
Consumer
Consumer
Home Office
Consumer
Consumer
Consumer
Consumer
Corporate
Consumer
Consumer
Consumer
Home Office
Corporate

At the bottom of the results grid, it says "Total rows: 793 Query complete 00:00:00.071 CRLF Ln 2, Col 19".

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 screenshot shows a SQL query window with the following content:

```
Task 36A--Insert and Import Data by Kelechi Immeldah uwa...
No limit
Query History
1 ~ SELECT Count (distinct Segment)
2   FROM Customer_data
```

Below the query window is a results grid titled "Data Output". The grid has one column labeled "count" with the data type "bigint". The data consists of 1 row, containing the value 3.

count
3

At the bottom of the results grid, it says "Total rows: 1 Query complete 00:00:00.090 CRLF Ln 2, Col 9".

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';
```

The screenshot shows the SQL Server Management Studio interface. In the top pane, a query window contains the following SQL code:

```
1 SELECT *\n2 FROM Customer_data\n3 WHERE City = 'Los Angeles'
```

In the bottom pane, the results of the query are displayed in a grid. The columns are: customer_id, customer_name, segment, age, country, city, state, postal_code, and region. The results show 58 rows where the city is 'Los Angeles'. The data includes various names like Darrin Van Huff, Brosina Hoffman, Eric Hoffmann, etc., and ages ranging from 20 to 66.

customer_id	customer_name	segment	age	country	city	state	postal_code	region
DV-13045	Darrin Van Huff	Corporate	31	United States	Los Angeles	California	90036	West
BH-11710	Brosina Hoffman	Consumer	20	United States	Los Angeles	California	90032	West
EH-13945	Eric Hoffmann	Consumer	21	United States	Los Angeles	California	90049	West
RA-19885	Ruben Ausman	Corporate	51	United States	Los Angeles	California	90049	West
KM-16720	Kunst Miller	Consumer	69	United States	Los Angeles	California	90004	West
JS-15685	Jim Sink	Corporate	33	United States	Los Angeles	California	90036	West
LS-16975	Lindsay Shagari	Home Office	66	United States	Los Angeles	California	90004	West
TT-21070	Ted Trevino	Consumer	67	United States	Los Angeles	California	90045	West
CS-12130	Chad Sievert	Consumer	51	United States	Los Angeles	California	90004	West
FM-14290	Frank Mervin	Home Office	45	United States	Los Angeles	California	90032	West
JH-15910	Jonathan Howell	Consumer	40	United States	Los Angeles	California	90032	West
JO-15280	Jas O'Carroll	Consumer	39	United States	Los Angeles	California	90004	West
DB-13615	Doug Bickford	Consumer	19	United States	Los Angeles	California	90045	West
PB-19150	Philip Brown	Consumer	30	United States	Los Angeles	California	90004	West
MT-18070	Michelle Tran	Home Office	69	United States	Los Angeles	California	90045	West
BN-11515	Bradley Nguyen	Consumer	63	United States	Los Angeles	California	90049	West
KB-16240	Karen Bern	Corporate	42	United States	Los Angeles	California	90036	West

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 *\nFROM customer_data\nWHERE age = '31'
```

The screenshot shows the SQL Server Management Studio interface. In the top pane, a query window contains the following SQL code:

```
1 SELECT *\n2 FROM Customer_data\n3 WHERE age = '31'
```

In the bottom pane, the results of the query are displayed in a grid. The columns are: customer_id, customer_name, segment, age, country, city, state, postal_code, and region. The results show 22 rows where the age is 31. The data includes various names like Darrin Van Huff, Linda Cazamias, Paul Stevenson, etc., and cities like Los Angeles, Naperville, Chicago, Orlando Park, Roseville, New York City, etc.

customer_id	customer_name	segment	age	country	city	state	postal_code	region
DV-13045	Darrin Van Huff	Corporate	31	United States	Los Angeles	California	90036	West
LC-16930	Linda Cazamias	Corporate	31	United States	Naperville	Illinois	60540	Central
PS-18970	Paul Stevenson	Home Office	31	United States	Chicago	Illinois	60610	Central
PA-19060	Pete Armstrong	Home Office	31	United States	Orland Park	Illinois	60462	Central
LC-16885	Lena Creighton	Consumer	31	United States	Roseville	California	95661	West
JK-15640	Jim Kriz	Home Office	31	United States	New York City	New York	10009	East
AS-10225	Alan Schoenberger	Corporate	31	United States	New York City	New York	10024	East
KC-16675	Kimberly Carter	Corporate	31	United States	Seattle	Washington	98105	West
CW-19105	Carl Weiss	Home Office	31	United States	Huntsville	Texas	77340	Central
MD-17350	Maribeth Dona	Consumer	31	United States	Fayetteville	Arkansas	72701	South
BN-11785	Bryan Mills	Consumer	31	United States	Columbus	Ohio	43229	East
MG-17890	Michael Granlund	Home Office	31	United States	Clinton	Maryland	20735	East
YS-21880	Yana Sorensen	Corporate	31	United States	Burlington	North Carolina	27217	South
BW-12600	Ben Wallace	Consumer	31	United States	Boynton Beach	Florida	33437	South
TM-21010	Tamara Manning	Consumer	31	United States	San Francisco	California	94122	West
RR-19315	Ralph Ritter	Consumer	31	United States	San Francisco	California	94110	West
TC-21295	Toby Carlisle	Consumer	31	United States	San Diego	California	92024	West

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;
```

The screenshot shows the Oracle SQL Developer interface. The query window contains the following code:

```
1 SELECT *
2 FROM Customer_data
3 WHERE age < '31'
```

The results window displays a table with 182 rows, showing customer details like name, age, and location. The columns are:

	customer_id	customer_name	segment	age	country	city	state	postal_code	region
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 Molinar	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

Total rows: 182 Query complete 00:00:00.097 CRLF Ln 3, Col 17

The following operators can be used in the WHERE clause:

- = Equal
- > Greater than
- < Less than
- >= Greater than or equal
- <= Less than or equal
- Not equal. Note: In some versions of SQL this operator may be written as (! =)
- BETWEEN --- Used between a certain range
- LIKE --- Search for a pattern
- 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.
```

Task 36A—Insert and Import Data by Kelechi Immeldah uwa... ▾

No limit ▾

Query History

```
1 ✓ SELECT *
2 FROM product_data
3 ORDER BY Category
```

Data Output Messages Notifications

Showing rows: 1 to 1000 | Page No: 1 of 2 | ↺ ↻

product_id	category	sub_category	product_name
FUR-FU-10000672	Furniture	Furnishings	Executive Impressions 10 Spectator Wall Clock
FUR-FU-10004351	Furniture	Furnishings	Staple-based wall hangings
FUR-BO-10003450	Furniture	Bookcases	Bush Westfield Collection Bookcases Dark Cherry Finish
FUR-TA-10002041	Furniture	Tables	Bevis Round Conference Table Top X-Base
FUR-TA-10002228	Furniture	Tables	Bevis Traditional Conference Table Top Plinth Base
FUR-BO-10003894	Furniture	Bookcases	Safco Value Mate Steel Bookcase Baked Enamel Finish on Steel Black
FUR-CH-10001190	Furniture	Chairs	Global Deluxe High-Back Office Chair in Storm
FUR-BO-10001619	Furniture	Bookcases	OSullivan Cherrywood Estates Traditional Bookcase
FUR-TA-10002607	Furniture	Tables	KI Conference Tables
FUR-FU-10002107	Furniture	Furnishings	Eldon Pizzaz Desk Accessories
FUR-BO-10001972	Furniture	Bookcases	OSullivan 4-Shelf Bookcase in Odessa Pine
FUR-CH-10003956	Furniture	Chairs	Novimex High-Tech Fabric Mesh Task Chair
FUR-FU-10002505	Furniture	Furnishings	Eldon 100 Class Desk Accessories
FUR-FU-10003095	Furniture	Furnishings	Linden 12 Wall Clock With Oak Frame
FUR-TA-10000849	Furniture	Tables	Bevis Rectangular Conference Tables
FUR-CH-10004886	Furniture	Chairs	Bevis Steel Folding Chairs
FUR-FU-10003577	Furniture	Furnishings	Nu-Dell Leatherette Frames

Total rows: 1862 | Query complete 00:00:00.144 | CRLF | Ln 3, Col 18

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;
```

Task 36A—Insert and Import Data by Kelechi Immeldah uwa... ▾

No limit ▾

Query History

```
1 ✓ SELECT *
2 FROM sales_data
3 ORDER BY sales DESC
```

Data Output Messages Notifications

Showing rows: 1 to 1000 | Page No: 1 of 10 | ↺ ↻

order_line_id	order_id	order_date	ship_date	ship_mode	customer_id	product_id	sales	quantity	discount	profit
2698	CA-2014-145317	2014-03-18	2014-03-23	Standard Class	SM-20320	TEC-MA-100024...	22638.48	6	0.5	-1811.0
6827	CA-2016-118689	2016-10-02	2016-10-09	Standard Class	TC-20980	TEC-CO-100047...	17499.95	5	0	8399
8154	CA-2017-140151	2017-03-23	2017-03-25	First Class	RB-19360	TEC-CO-100047...	13999.96	4	0	6719.5
2624	CA-2017-127180	2017-10-22	2017-10-24	First Class	TA-21385	TEC-CO-100047...	11199.968	4	0.2	3919.5
4191	CA-2017-166709	2017-11-17	2017-11-22	Standard Class	HL-15040	TEC-CO-100047...	10499.97	3	0	5039.5
9040	CA-2016-117121	2016-12-17	2016-12-21	Standard Class	AB-10105	OFF-BI-10000545	9892.74	13	0	494
4099	CA-2014-116904	2014-09-23	2014-09-28	Standard Class	SC-20095	OFF-BI-10001120	9449.95	5	0	4630.4
4278	US-2016-107440	2016-04-16	2016-04-20	Standard Class	BS-11365	TEC-MA-100010...	9099.93	7	0	2365.5
8489	CA-2016-158841	2016-02-02	2016-02-04	Second Class	SE-20110	TEC-MA-100011...	8749.95	5	0	2799
6426	CA-2016-143714	2016-05-23	2016-05-27	Standard Class	CC-12370	TEC-CO-100047...	8399.976	4	0.4	1119.5
2506	CA-2014-143917	2014-07-25	2014-07-27	Second Class	KL-16645	OFF-SU-10000151	8187.65	5	0	327
166	CA-2014-139892	2014-09-08	2014-09-12	Standard Class	BM-11140	TEC-MA-100008...	8159.952	8	0.4	-1359
684	US-2017-168116	2017-11-04	2017-11-04	Same Day	GT-14635	TEC-MA-100041...	7999.98	4	0.5	3839.5
6627	CA-2014-145541	2014-12-14	2014-12-21	Standard Class	TB-21400	TEC-MA-100011...	6999.96	4	0	2239.5
510	CA-2015-145352	2015-03-16	2015-03-22	Standard Class	CM-12385	OFF-BI-10003527	6354.95	5	0	3177
6521	CA-2017-138289	2017-01-16	2017-01-18	Second Class	AR-10540	OFF-BI-10004995	5443.96	4	0	2504.2

Total rows: 9994 | Query complete 00:00:00.174 | CRLF | Ln 3, Col 20

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;
```

Task 36A-Insert and Import Data by Kelechi Immeldah uwa... ▾

Query History

```
1 SELECT *
2 FROM sales_data
3 ORDER BY sales DESC, quantity;
```

Data Output Messages Notifications

	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 precision
1	2698	CA-2014-145317	2014-03-18	2014-03-23	Standard Class	SM-20320	TEC-MA-100024...	22638.48	6	0.5	-1811.0
2	6827	CA-2016-118689	2016-10-02	2016-10-09	Standard Class	TC-20980	TEC-CO-100047...	17499.95	5	0	8399
3	8154	CA-2017-140151	2017-03-23	2017-03-25	First Class	RB-19360	TEC-CO-100047...	13999.96	4	0	6719.5
4	2624	CA-2017-127180	2017-10-22	2017-10-24	First Class	TA-21385	TEC-CO-100047...	11199.968	4	0.2	3919.5
5	4191	CA-2017-166709	2017-11-17	2017-11-22	Standard Class	HL-15040	TEC-CO-100047...	10499.97	3	0	5039.5
6	9040	CA-2016-117121	2016-12-17	2016-12-21	Standard Class	AB-10105	OFF-BI-10000545	9892.74	13	0	494
7	4099	CA-2014-116904	2014-09-23	2014-09-28	Standard Class	SC-20095	OFF-BI-10001120	9449.95	5	0	4630.4
8	4278	US-2016-107440	2016-04-16	2016-04-20	Standard Class	BS-11365	TEC-MA-100010...	9099.93	7	0	2365.5
9	8489	CA-2016-158841	2016-02-02	2016-02-04	Second Class	SE-20110	TEC-MA-100011...	8749.95	5	0	2799
10	6426	CA-2016-143714	2016-05-23	2016-05-27	Standard Class	CC-12370	TEC-CO-100047...	8399.976	4	0.4	1119.5
11	2506	CA-2014-143917	2014-07-25	2014-07-27	Second Class	KL-16645	OFF-SU-10000151	8187.65	5	0	327
12	166	CA-2014-139892	2014-09-08	2014-09-12	Standard Class	BM-11140	TEC-MA-100008...	8159.952	8	0.4	-1359
13	684	US-2017-168116	2017-11-04	2017-11-04	Same Day	GT-14635	TEC-MA-100041...	7999.98	4	0.5	-3839.5
14	6627	CA-2014-145541	2014-12-14	2014-12-21	Standard Class	TB-21400	TEC-MA-100011...	6999.96	4	0	2239.5
15	510	CA-2015-145352	2015-03-16	2015-03-22	Standard Class	CM-12385	OFF-BI-10003527	6354.95	5	0	3177
16	6521	CA-2017-138289	2017-01-16	2017-01-18	Second Class	AR-10540	OFF-BI-10004995	5443.96	4	0	2504.5

Total rows: 9994 Query complete 00:00:00.132 CRLF Ln 3, Col 31

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%';
```

Task 36A-Insert and Import Data by Kelechi Immeldah uwa... ▾

Query History

```
1 SELECT *
2 FROM customer_data
3 WHERE state = 'Washington' AND City LIKE 'S%'
```

Data Output Messages Notifications

	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	IM-15070	Irene Maddox	Consumer	66	United States	Seattle	Washington	98103	West
2	DK-13090	Dave Kipp	Consumer	24	United States	Seattle	Washington	98103	West
3	NK-18490	Neil Knudson	Home Office	61	United States	Seattle	Washington	98105	West
4	DB-13060	Dave Brooks	Consumer	50	United States	Seattle	Washington	98115	West
5	MJ-17740	Max Jones	Consumer	40	United States	Seattle	Washington	98115	West
6	KC-16675	Kimberly Carter	Corporate	31	United States	Seattle	Washington	98105	West
7	SS-20590	Sonia Sunley	Consumer	51	United States	Seattle	Washington	98115	West
8	JF-15490	Jeremy Farry	Consumer	38	United States	Seattle	Washington	98105	West
9	EB-13840	Ellis Ballard	Corporate	25	United States	Seattle	Washington	98105	West
10	AT-10735	Annie Thurman	Consumer	30	United States	Seattle	Washington	98115	West
11	DS-13030	Darrin Sayre	Home Office	62	United States	Seattle	Washington	98115	West
12	JS-15880	John Stevenson	Consumer	60	United States	Seattle	Washington	98103	West
13	DW-13540	Don Weiss	Consumer	60	United States	Seattle	Washington	98105	West
14	HR-14830	Harold Ryan	Corporate	69	United States	Seattle	Washington	98103	West
15	GM-14680	Greg Matthias	Consumer	69	United States	Seattle	Washington	98105	West
16	TB-21400	Tom Boeckenhauer	Consumer	37	United States	Seattle	Washington	98105	West
17	EB-13750	Edward Becker	Corporate	20	United States	Seattle	Washington	98103	West

Total rows: 32 Query complete 00:00:00.261 CRLF Ln 3, Col 46

AND VS OR

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

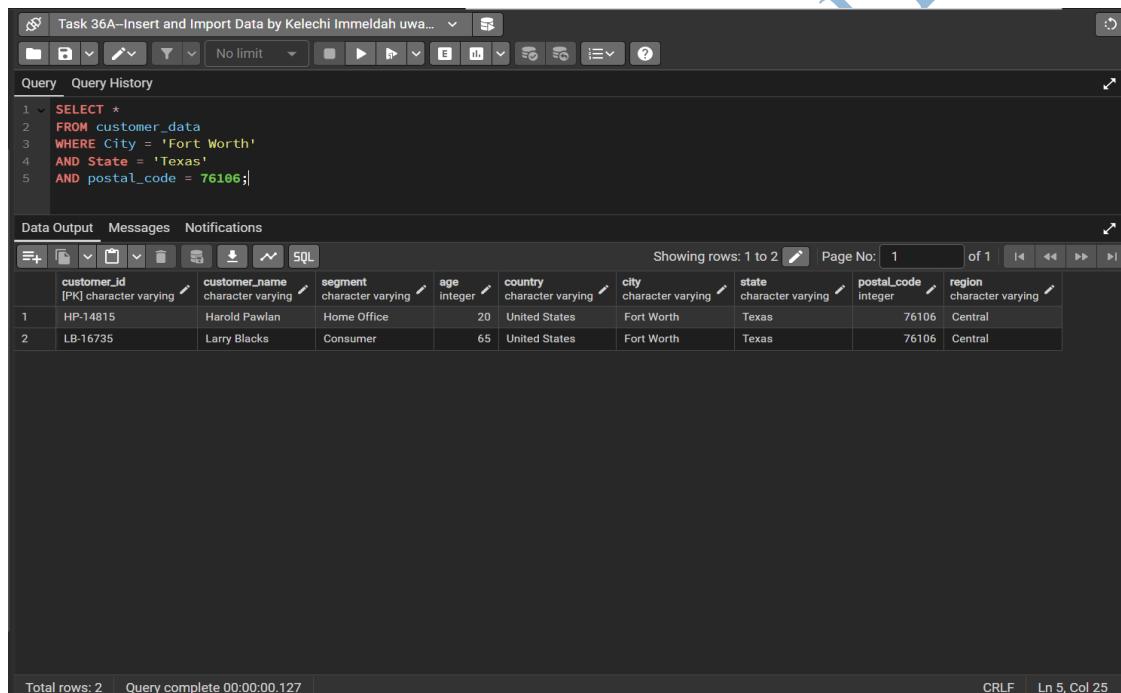
The OR operator displays a record of whether any of the conditions are TRUE.

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;
```



The screenshot shows the MySQL Workbench interface with a query editor and a results grid. The query editor contains the following SQL code:

```
1 ✓ SELECT *
2   FROM customer_data
3  WHERE City = 'Fort Worth'
4    AND State = 'Texas'
5    AND postal_code = 76106;
```

The results grid displays two rows of data from the customer_data table:

	customer_id	customer_name	segment	age	country	city	state	postal_code	region
1	HP-14815	Harold Pawlan	Home Office	20	United States	Fort Worth	Texas	76106	Central
2	LB-16735	Larry Blacks	Consumer	65	United States	Fort Worth	Texas	76106	Central

Total rows: 2 | Query complete 00:00:00.127 | CRLF | Ln 5, Col 25

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%');
```

```
Task 36A-Insert and Import Data by Kelechi Immediah uwa... ▾
Query History
Query
SELECT *
FROM customer_data
WHERE City = 'Fort Worth'
AND (Customer_name LIKE 'H%'
OR Customer_name LIKE 'A%')

Data Output Messages Notifications
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 HP-14815 Harold Pawlan Home Office 20 United States Fort Worth Texas 76106 Central

Showing rows: 1 to 1 Page No: 1 of 1
Total rows: 1 Query complete 00:00:00.070
CRLF Ln 5, Col 28
```

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.

The screenshot shows a SQL query window with the following code:

```

1 SELECT *
2 FROM customer_data
3 WHERE City = 'Los Angeles' OR City = 'Fort Worth'

```

The results grid displays data from the 'customer_data' table, showing 58 rows. The columns are:

	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	DV-13045	Darrin Van Huff	Corporate	31	United States	Los Angeles	California	90036	West
2	BH-11710	Brosina Hoffman	Consumer	20	United States	Los Angeles	California	90032	West
3	EH-13945	Eric Hoffmann	Consumer	21	United States	Los Angeles	California	90049	West
4	RA-19885	Ruben Ausman	Corporate	51	United States	Los Angeles	California	90049	West
5	KM-16720	Kunst Miller	Consumer	69	United States	Los Angeles	California	90004	West
6	JS-15685	Jim Sink	Corporate	33	United States	Los Angeles	California	90036	West
7	LS-16975	Lindsay Shagari	Home Office	66	United States	Los Angeles	California	90004	West
8	TT-21070	Ted Trevino	Consumer	67	United States	Los Angeles	California	90045	West
9	CS-12130	Chad Sievert	Consumer	51	United States	Los Angeles	California	90004	West
10	FM-14290	Frank Merwin	Home Office	45	United States	Los Angeles	California	90032	West
11	JH-15910	Jonathan Howell	Consumer	40	United States	Los Angeles	California	90032	West
12	JO-15280	Jas O'Carroll	Consumer	39	United States	Los Angeles	California	90004	West
13	DB-13615	Doug Bickford	Consumer	19	United States	Los Angeles	California	90045	West
14	PB-19150	Philip Brown	Consumer	30	United States	Los Angeles	California	90004	West
15	MT-18070	Michelle Tran	Home Office	69	United States	Los Angeles	California	90045	West
16	BN-11515	Bradley Nguyen	Consumer	63	United States	Los Angeles	California	90049	West
17	KB-16240	Karen Bern	Corporate	42	United States	Los Angeles	California	90036	West

Total rows: 58 Query complete 00:00:00.090 CRLF Ln 3, Col 51

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 = 'Los Angeles' 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.`

The screenshot shows a SQL query window with the following content:

```

1 SELECT *
2 FROM customer_data
3 WHERE NOT Segment = 'Consumer'

```

Below the query, the results grid displays data from the customer_data table, filtered by Segment not equal to 'Consumer'. The columns are:

customer_id	customer_name	segment	age	country	city	state	postal_code	region
DV-13045	Darrin Van Huff	Corporate	31	United States	Los Angeles	California	90036	West
HP-14815	Harold Pawlan	Home Office	20	United States	Fort Worth	Texas	76106	Central
KB-16585	Ken Black	Corporate	67	United States	Fremont	Nebraska	68025	Central
MA-17560	Matt Abelman	Home Office	19	United States	Houston	Texas	77095	Central
GH-14485	Gene Hale	Corporate	28	United States	Richardson	Texas	75080	Central
SN-20710	Steve Nguyen	Home Office	46	United States	Houston	Texas	77041	Central
LC-16930	Linda Cazamias	Corporate	31	United States	Naperville	Illinois	60540	Central
RA-19885	Ruben Ausman	Corporate	51	United States	Los Angeles	California	90049	West
ES-14080	Erin Smith	Corporate	20	United States	Melbourne	Florida	32935	South
ON-18715	Odella Nelson	Corporate	27	United States	Eagan	Minnesota	55122	Central
JM-15265	Janet Molinari	Corporate	23	United States	New York City	New York	10024	East
PS-18970	Paul Stevenson	Home Office	31	United States	Chicago	Illinois	60610	Central
BS-11590	Brendan Sweet	Corporate	45	United States	Gilbert	Arizona	85234	West
KB-16600	Ken Brennan	Corporate	29	United States	Houston	Texas	77041	Central
SC-20770	Stewart Carmichael	Corporate	18	United States	Decatur	Alabama	35601	South
JC-16105	Julie Creighton	Corporate	56	United States	Durham	North Carolina	27707	South
CS-12400	Christopher Schild	Home Office	33	United States	Chicago	Illinois	60623	Central

Total rows: 384 | Query complete 00:00:00.106 | CRLF | Ln 3, Col 31

NOT LIKE

Example

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

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

The screenshot shows a SQL query window with the following content:

```

1 SELECT *
2 FROM customer_data
3 WHERE customer_name NOT LIKE 'A%'

```

Below the query, the results grid displays data from the customer_data table, filtered by Customer_name not starting with 'A'. The columns are:

customer_id	customer_name	segment	age	country	city	state	postal_code	region
CG-17520	Claire Gute	Consumer	67	United States	Henderson	Kentucky	42420	South
DV-13045	Darrin Van Huff	Corporate	31	United States	Los Angeles	California	90036	West
SO-20335	Sean O'Donnell	Consumer	65	United States	Fort Lauderdale	Florida	33311	South
BH-11710	Brosina Hoffman	Consumer	20	United States	Los Angeles	California	90032	West
IM-15070	Irene Maddox	Consumer	66	United States	Seattle	Washington	98103	West
HP-14815	Harold Pawlan	Home Office	20	United States	Fort Worth	Texas	76106	Central
PK-19075	Pete Kriz	Consumer	46	United States	Madison	Wisconsin	53711	Central
ZD-21925	Zuschuss Donatelli	Consumer	66	United States	San Francisco	California	94109	West
KB-16585	Ken Black	Corporate	67	United States	Fremont	Nebraska	68025	Central
SF-20065	Sandra Flanagan	Consumer	41	United States	Philadelphia	Pennsylvania	19140	East
EB-13870	Emily Burns	Consumer	34	United States	Orem	Utah	84057	West
EH-13945	Eric Hoffmann	Consumer	21	United States	Los Angeles	California	90049	West
TB-21520	Tracy Blumstein	Consumer	48	United States	Philadelphia	Pennsylvania	19140	East
MA-17560	Matt Abelman	Home Office	19	United States	Houston	Texas	77095	Central
GH-14485	Gene Hale	Corporate	28	United States	Richardson	Texas	75080	Central
SN-20710	Steve Nguyen	Home Office	46	United States	Houston	Texas	77041	Central
LC-16930	Linda Cazamias	Corporate	31	United States	Naperville	Illinois	60540	Central

Total rows: 729 | Query complete 00:00:00.329 | CRLF | Ln 3, Col 34

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.
```

The screenshot shows a MySQL Workbench interface with a query editor and a results grid. The query is:

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

The results grid displays 1000 rows of data from the sales_data table, filtered by the WHERE clause. The columns include order_line, order_id, order_date, ship_date, ship_mode, customer_id, product_id, sales, quantity, discount, and profit. The data shows various transactions with different order numbers, dates, and product details.

order_line	order_id	order_date	ship_date	ship_mode	customer_id	product_id	sales	quantity	discount	profit
1	CA-2016-1521...	2016-11-08	2016-11-11	Second Class	CG-12520	FUR-BO-100017...	261.96	2	0	41.5
2	CA-2014-1158...	2014-06-09	2014-06-14	Standard Class	BH-11710	FUR-TA-100015...	1706.184	9	0.2	85.3
3	CA-2014-1158...	2014-06-09	2014-06-14	Standard Class	BH-11710	TEC-PH-100020...	911.424	4	0.2	68.3
4	CA-2017-1144...	2017-04-15	2017-04-20	Standard Class	AA-10480	OFF-PA-10002365	15.552	3	0.2	5.4
5	CA-2016-1613...	2016-12-05	2016-12-10	Standard Class	IM-15070	OFF-BI-10003656	407.976	3	0.2	132.6
6	US-2015-11898...	2015-11-22	2015-11-26	Standard Class	HP-14815	OFF-AP-100023...	68.81	5	0.8	-123
7	US-2015-118983	2015-11-22	2015-11-26	Standard Class	HP-14815	OFF-BI-10000756	2.544	3	0.8	-3
8	CA-2014-1058...	2014-11-11	2014-11-18	Standard Class	PK-19075	OFF-ST-10004186	665.88	6	0	13.5
9	CA-2014-1671...	2014-05-13	2014-05-15	Second Class	AG-10270	OFF-ST-10000107	55.5	2	0	1
10	CA-2014-1433...	2014-08-27	2014-09-01	Second Class	ZD-21925	OFF-AR-100030...	8.56	2	0	2.4
11	CA-2014-1433...	2014-08-27	2014-09-01	Second Class	ZD-21925	TEC-PH-100019...	213.48	3	0.2	16
12	CA-2014-1433...	2014-08-27	2014-09-01	Second Class	ZD-21925	OFF-BI-10002215	22.72	4	0.2	7
13	CA-2016-1373...	2016-12-09	2016-12-13	Standard Class	KB-16585	OFF-AR-100002...	19.46	7	0	5.0
14	CA-2016-1373...	2016-12-09	2016-12-13	Standard Class	KB-16585	OFF-AP-100014...	60.34	7	0	15.6
15	US-2017-156909	2017-07-16	2017-07-18	Second Class	SF-20065	FUR-CH-100027...	71.372	2	0.3	-1.0
16	CA-2015-1063...	2015-09-25	2015-09-30	Standard Class	EB-13870	FUR-TA-100005...	1044.63	3	0	240.2

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');
```

Task 36A--Insert and Import Data by Kelechi immeldah uwa...

No limit

```

1 ✓ SELECT *
2   FROM customer_data
3 WHERE City NOT IN ('Fort Worth', 'Los Angeles')

```

Data Output Messages Notifications

Showing rows: 1 to 733 | Page No: 1 of 1 | < > << >>

	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	SO-20335	Sean O'Donnell	Consumer	65	United States	Fort Lauderdale	Florida	33311	South
3	AA-10480	Andrew Allen	Consumer	50	United States	Concord	North Carolina	28027	South
4	IM-15070	Irene Maddox	Consumer	66	United States	Seattle	Washington	98103	West
5	PK-19075	Pete Kriz	Consumer	46	United States	Madison	Wisconsin	53711	Central
6	AG-10270	Alejandro Grove	Consumer	18	United States	West Jordan	Utah	84084	West
7	ZD-21925	Zuschuss Donatelli	Consumer	66	United States	San Francisco	California	94109	West
8	KB-16585	Ken Black	Corporate	67	United States	Fremont	Nebraska	68025	Central
9	SF-20065	Sandra Flanagan	Consumer	41	United States	Philadelphia	Pennsylvania	19140	East
10	EB-13870	Emily Burns	Consumer	34	United States	Orem	Utah	84057	West
11	TB-21520	Tracy Blumstein	Consumer	48	United States	Philadelphia	Pennsylvania	19140	East
12	MA-17560	Matt Abelman	Home Office	19	United States	Houston	Texas	77095	Central
13	GH-14485	Gene Hale	Corporate	28	United States	Richardson	Texas	75080	Central
14	SN-20710	Steve Nguyen	Home Office	46	United States	Houston	Texas	77041	Central
15	LC-16930	Linda Cazamias	Corporate	31	United States	Naperville	Illinois	60540	Central
16	ES-14080	Erin Smith	Corporate	20	United States	Melbourne	Florida	32935	South
17	ON-18715	Odella Nelson	Corporate	27	United States	Eagan	Minnesota	55122	Central

Total rows: 733 Query complete 00:00:00.339 CRLF Ln 3, Col 48

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.

Task 36A--Insert and Import Data by Kelechi immeldah uwa...

No limit

```

1 ✓ SELECT *
2   FROM customer_data
3 WHERE NOT age > 40.

```

Data Output Messages Notifications

Showing rows: 1 to 332 | Page No: 1 of 1 | < > << >>

	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	DV-13045	Darrin Van Huff	Corporate	31	United States	Los Angeles	California	90036	West
2	BH-11710	Brosina Hoffman	Consumer	20	United States	Los Angeles	California	90032	West
3	HP-14815	Harold Pavlan	Home Office	20	United States	Fort Worth	Texas	76106	Central
4	AG-10270	Alejandro Grove	Consumer	18	United States	West Jordan	Utah	84084	West
5	EB-13870	Emily Burns	Consumer	34	United States	Orem	Utah	84057	West
6	EH-13945	Eric Hoffmann	Consumer	21	United States	Los Angeles	California	90049	West
7	MA-17560	Matt Abelman	Home Office	19	United States	Houston	Texas	77095	Central
8	GH-14485	Gene Hale	Corporate	28	United States	Richardson	Texas	75080	Central
9	LC-16930	Linda Cazamias	Corporate	31	United States	Naperville	Illinois	60540	Central
10	ES-14080	Erin Smith	Corporate	20	United States	Melbourne	Florida	32935	South
11	ON-18715	Odella Nelson	Corporate	27	United States	Eagan	Minnesota	55122	Central
12	DP-13000	Darren Powers	Consumer	40	United States	New Albany	Indiana	47150	Central
13	JM-15265	Janet Mollnar	Corporate	23	United States	New York City	New York	10024	East
14	TB-21055	Ted Butterfield	Consumer	34	United States	Troy	New York	12180	East
15	PS-18970	Paul Stevenson	Home Office	31	United States	Chicago	Illinois	60610	Central
16	HM-14980	Henry MacAllister	Consumer	35	United States	New York City	New York	10009	East
17	JE-15745	Joel Eaton	Consumer	25	United States	Memphis	Tennessee	38109	South

Total rows: 332 Query complete 00:00:00.283 CRLF Ln 3, Col 19

NOT LESS THAN

Example

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

```

SELECT * FROM Customers
WHERE NOT age < 50.

```

The screenshot shows a SQL query results grid from a database interface. The query is:

```

1 SELECT *
2 FROM customer_data
3 WHERE NOT age < 50

```

The results grid displays data from the 'customer_data' table, showing 320 rows. The columns are:

	customer_id	customer_name	segment	age	country	city	state	postal_code	region
1	CG-12520	Claire Gute	Consumer	67	United States	Henderson	Kentucky	42420	South
2	SO-20335	Sean O'Donnell	Consumer	65	United States	Fort Lauderdale	Florida	33311	South
3	AA-10480	Andrew Allen	Consumer	50	United States	Concord	North Carolina	28027	South
4	IM-15070	Irene Maddox	Consumer	66	United States	Seattle	Washington	98103	West
5	ZD-21925	Zuschuss Donatelli	Consumer	66	United States	San Francisco	California	94109	West
6	KB-16585	Ken Black	Corporate	67	United States	Fremont	Nebraska	68025	Central
7	RA-19885	Ruben Ausman	Corporate	51	United States	Los Angeles	California	90049	West
8	PO-18865	Patrick O'Donnell	Consumer	64	United States	Westland	Michigan	48185	Central
9	LH-16900	Lena Hernandez	Consumer	66	United States	Dover	Delaware	19901	East
10	KM-16720	Kunst Miller	Consumer	69	United States	Los Angeles	California	90004	West
11	KD-16270	Karen Daniels	Consumer	59	United States	Springfield	Virginia	22153	South
12	JC-16105	Julie Creighton	Corporate	56	United States	Durham	North Carolina	27707	South
13	PG-18895	Paul Gonzalez	Consumer	64	United States	Rochester	Minnesota	55901	Central
14	GM-14455	Gary Mitchum	Home Office	64	United States	Houston	Texas	77095	Central
15	KB-16315	Karl Braun	Consumer	70	United States	Minneapolis	Minnesota	55407	Central
16	RB-19705	Roger Barco	Home Office	63	United States	Portland	Oregon	97206	West
17	ER-13855	Elpidia Rittenbach	Corporate	56	United States	Saint Paul	Minnesota	55106	Central

Total rows: 320 Query complete 00:00:00.122 CRLF Ln 3, Col 19

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:

- Specify both the column names and the values to be inserted:

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

- 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, ...).
```

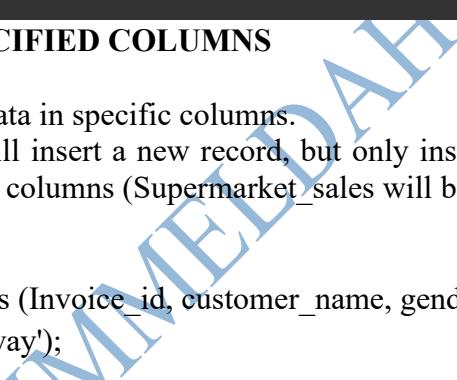
INSERT INTO EXAMPLE

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:



Task 36A--Insert and Import Data by Kelechi Immeldah uwa...

```

13 FROM Supermarket_sales
14
15 --Insert some details into the Table
16 --- INSERT keyword allows us to input details into an existing table
17
18 INSERT INTO Supermarket_sales
19     VALUES (45, 'Joel JOY',      'F',      'Livestock',      3000),
20             (56, 'James Jane',    'M',      'Fruits',        1000),
21             (64, 'Frank Eni',     'M',      'Vegetables',    2000),
22             (76, 'Chi Ernest',   'F',      'Meat',          4000),
23             (13, 'Esther Presh', 'F',      'Cream',         5000),
24             (15, 'Eden Joe',     'M',      'Perfume',       6000)
25
26 SELECT *
27 FROM Supermarket_sales
28
29

```

Data Output Messages Notifications

Invoice_Id	customer_name	gender	product_type	unit_price	
1	45	Joel JOY	Female	Livestock	3000
2	56	James Jane	Male	Fruits	1000
3	64	Frank Eni	Male	Vegetables	2000
4	45	Joel JOY	F	Livestock	3000
5	56	James Jane	M	Fruits	1000
6	64	Frank Eni	M	Vegetables	2000
7	76	Chi Ernest	F	Meat	4000
8	13	Esther Presh	F	Cream	5000
9	15	Eden Joe	M	Perfume	6000

Total rows: 9 Query complete 00:00:00.073 CRLF Ln 30, Col 1

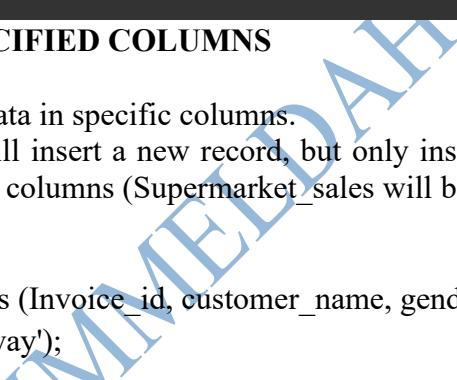
INSERT DATA ONLY IN SPECIFIED COLUMNS

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');
```



Task 36A--Insert and Import Data by Kelechi Immeldah uwa...

```

28
29 INSERT INTO supermarket_sales
30     VALUES ('60', 'Stavanger', 'Norway')
31
32
33 SELECT *
34 FROM Supermarket_sales
35
36
37
38
39
40
41
42
43 --Let's Create an Employee Details Table
44

```

Data Output Messages Notifications

Invoice_Id	customer_name	gender	product_type	unit_price	
3	64	Frank Eni	Male	Vegetables	2000
4	45	Joel JOY	F	Livestock	3000
5	56	James Jane	M	Fruits	1000
6	64	Frank Eni	M	Vegetables	2000
7	76	Chi Ernest	F	Meat	4000
8	13	Esther Presh	F	Cream	5000
9	15	Eden Joe	M	Perfume	6000
10	60	Stavanger	Norway	[null]	[null]

Total rows: 10 Query complete 00:00:00.063 CRLF Ln 33, Col 9

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)

The screenshot shows the SQL Server Management Studio interface. In the Query pane, a script is run to insert data into the 'Supermarket_sales' table. The script includes comments explaining the purpose of the INSERT keyword. The results pane displays the inserted data, showing 9 rows of customer details and their corresponding product type and unit price.

```
13 FROM Supermarket_sales
14
15 --Insert some details into the Table
16 --- INSERT keyword allows us to input details into an existing table
17
18 INSERT INTO Supermarket_sales
19     VALUES (45, 'Joel JOY', 'F', 'Livestock', 3000),
20             (56, 'James Jane', 'M', 'Fruits', 1000),
21             (64, 'Frank Eni', 'M', 'Vegetables', 2000),
22             (76, 'Chi Ernest', 'F', 'Meat', 4000),
23             (13, 'Esther Presh', 'F', 'Cream', 5000),
24             (15, 'Eden Joe', 'M', 'Perfume', 6000)
25
26 SELECT *
27 FROM Supermarket_sales
28
29
```

	Invoice_Id	customer_name	gender	product_type	unit_price
1	45	Joel JOY	Female	Livestock	3000
2	56	James Jane	Male	Fruits	1000
3	64	Frank Eni	Male	Vegetables	2000
4	45	Joel JOY	F	Livestock	3000
5	56	James Jane	M	Fruits	1000
6	64	Frank Eni	M	Vegetables	2000
7	76	Chi Ernest	F	Meat	4000
8	13	Esther Presh	F	Cream	5000
9	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 screenshot shows a MySQL Workbench interface. The query editor contains the following SQL code:

```

1 SELECT *
2 FROM supermarket_sales
3
4 SELECT product_type, gender, customer_name
5 FROM supermarket_sales
6 WHERE product_type IS NULL
7

```

The results pane shows one row:

product_type	gender	customer_name
[null]	Norway	Stavanger

Showing rows: 1 to 1 | Page No: 1 of 1 | < << >> >

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

```

The screenshot shows a MySQL Workbench interface. The query editor contains the following SQL code:

```

1 SELECT *
2 FROM supermarket_sales
3
4 SELECT product_type, gender, customer_name
5 FROM supermarket_sales
6 WHERE product_type IS NULL
7
8 SELECT Product_type, Customer_name, gender
9 FROM supermarket_sales
10 WHERE product_type IS NOT NULL
11

```

The results pane shows 9 rows:

product_type	customer_name	gender
Livestock	Joel JOY	Female
Fruits	James Jane	Male
Vegetables	Frank Eni	Male
Livestock	Joel JOY	F
Fruits	James Jane	M
Vegetables	Frank Eni	M
Meat	Chi Ernest	F
Cream	Esther Presh	F
Perfume	Eden Joe	M

Total rows: 9 | Query complete 00:00:00.269 | CRLF | Ln 11, Col 1

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:

The screenshot shows a SQL query editor window titled "Task 36A-Insert and Import Data by Kelechi Immeldah uwa...". The "Query" tab is selected, displaying a script with numbered lines. The script includes various SQL statements such as SELECT, UPDATE, and WHERE clauses. Below the script is a table output showing data from the Employee_details table. The table has columns: customer, customer_id, company, country, income, and department. The data shows 28 rows of employee information. The bottom status bar indicates "Total rows: 28" and "Query complete 00:00:00.178".

	customer	customer_id	company	country	income	department
1	Richard Elliot	4706	Voltage	USA	400,000	Finance
2	Robert Spear	4083	Inky	USA	100,000	Accounting
3	Roger Mun	4093	Sleeps	USA	200,000	Data
4	Paul Garza	4364	Kind Ape	USA	400,000	Networking
5	Robert Marquez	4882	Pet Feed	USA	100,000	Finance
6	Natalie Porter	4592	Right And Learning	USA	200,000	Accounting

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
```

Task 36A--Insert and Import Data by Kelechi Immeldah uwa...

Query History

```

21  FROM Employee_details
22  WHERE country = 'Austria'
23
24  --UPDATE--This is used to update things in the table
25
26  SELECT *
27  FROM Employee_details
28
29  --Lets Update the table
30
31  UPDATE Employee_details
32  SET Customer = 'Kelechi Uwanaka'
33  WHERE Customer_id = 4731
34
35  --Invoke the Update
36
37  SELECT *
38  FROM Employee_details
39  WHERE customer_id = '4731'
40
41  --DELETE--This is used to delete from the table

```

Data Output

	customer	customer_id	company	country	income	department
1	Kelechi Uwanaka	4731	WenCaL US	USA	100,000	Data

Total rows: 1 Query complete 00:00:00.057 CRLF Ln 38, Col 22

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';

Task 36A--Insert and Import Data by Kelechi Immeldah uwa...

Query History

```

32  SET Customer = 'Kelechi Uwanaka'
33  WHERE Customer_id = 4731
34
35
36  --Invoke the Update
37
38  SELECT *
39  FROM Employee_details
40  WHERE customer_id = '4731'
41
42  --DELETE--This is used to delete from the table
43  UPDATE Employee_details
44  SET Customer = 'Kelechi Uwanaka'
45  WHERE department = ' Data '
46
47
48  --Invoke the Update
49
50  SELECT *
51  FROM Employee_details
52  WHERE department = ' Data '

```

Data Output

	customer	customer_id	company	country	income	department
1	Kelechi Uwanaka	4093	Sleeps	USA	200,000	Data
2	Kelechi Uwanaka	4991	Right App Play	USA	400,000	Data
3	Kelechi Uwanaka	4993	Rehire	Austria	100,000	Data
4	Kelechi Uwanaka	4073	Perino	UK	200,000	Data
5	Kelechi Uwanaka	4619	Baden Paper	Germany	400,000	Data
6	Kelechi Uwanaka	4826	WenCaL UK	UK	100,000	Data

Total rows: 8 Query complete 00:00:00.062 CRLF Ln 49, Col 1

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

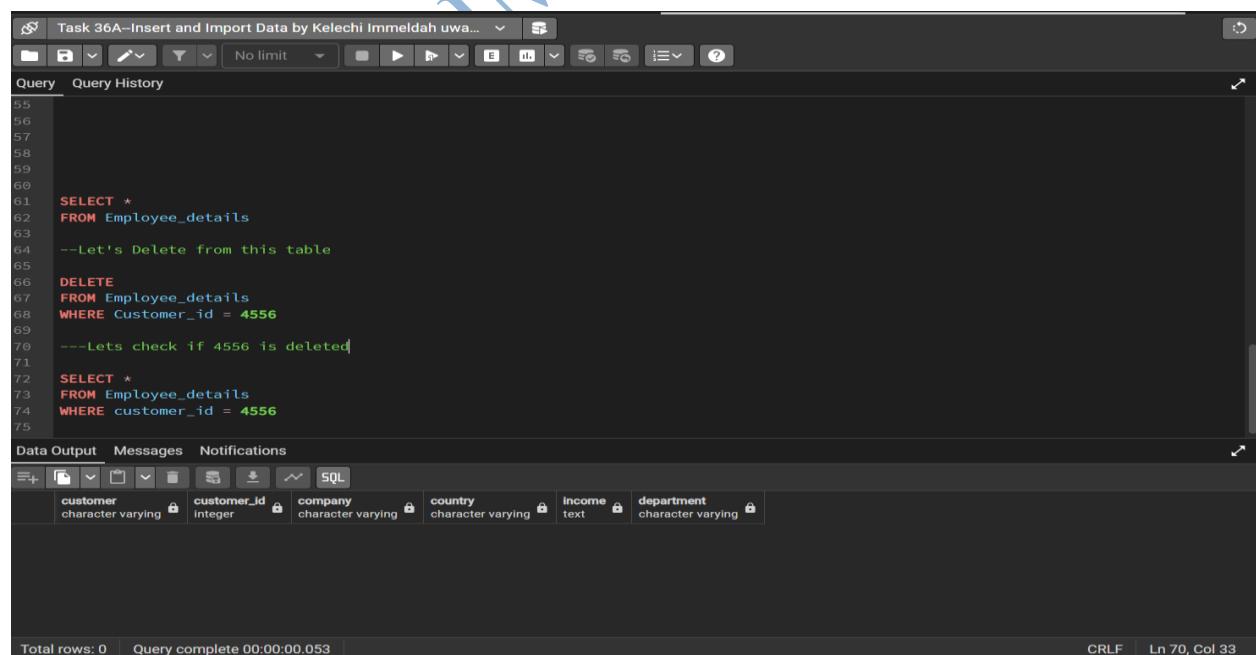
It is possible to delete all rows in a table without deleting the table. This means that the table structure, attributes, and indexes will be intact:

`DELETE FROM table_name;`

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

Example

`DELETE FROM Customers.`



A screenshot of a SQL query editor window titled "Task 36A - Insert and Import Data by Kelechi Immeldah uwa...". The main pane shows the following SQL code:

```
55  
56  
57  
58  
59  
60  
61 SELECT *  
62 FROM Employee_details  
63  
--Let's Delete from this table  
64  
65 DELETE  
66 FROM Employee_details  
67 WHERE Customer_id = 4556  
68  
---Lets check if 4556 is deleted|  
69  
70  
71  
72  
73 SELECT *  
74 FROM Employee_details  
75 WHERE customer_id = 4556  
76
```

The status bar at the bottom indicates "Total rows: 0" and "Query complete 00:00:00.053".

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;
```

The screenshot shows the MySQL Workbench interface. The query editor window contains the following SQL code:

```
59
60
61 SELECT *
62 FROM Employee_details
63
64 --Let's Delete from this table
65
66 DELETE
67 FROM Employee_details
68 WHERE Customer_id = 4556
69
70 ---Lets check if 4556 is deleted
71
72 SELECT *
73 FROM Employee_details
74 WHERE customer_id = 4556
75
76
77 SELECT *
78 FROM Employee_details
79 LIMIT 3;
```

The results pane displays the output of the last query, showing three rows of data from the Employee_details table:

customer character varying	customer_id integer	company character varying	country character varying	income text	department character varying
Robert Spear	4083	Inky	USA	100,000	Accounting
Paul Garza	4364	Kind Ape	USA	400,000	Networking
Robert Marquez	4882	Pet Feed	USA	100,000	Finance

Total rows: 3 | Query complete 00:00:00.075 | CRLF | Ln 79, Col 9

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;
```

The screenshot shows the Oracle SQL Developer interface. The query editor window displays the following SQL code:

```
80
81
82
83 SELECT *
84 FROM Employee_details
85 FETCH FIRST 3 ROWS ONLY
86
87
88
```

The results pane shows a table with three rows of data:

	customer	customer_id	company	country	income	department
1	Robert Spear	4083	Inky	USA	100,000	Accounting
2	Paul Garza	4364	Kind Ape	USA	400,000	Networking
3	Robert Marquez	4882	Pet Feed	USA	100,000	Finance

The status bar at the bottom indicates "Total rows: 3 Query complete 00:00:00.086".

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.
```

```

82
83   SELECT *
84     FROM Employee_details
85   FETCH FIRST 3 ROWS ONLY
86
87   SELECT *
88     FROM Employee_details
89   FETCH FIRST 50 PERCENT ROWS ONLY;
90

```

The screenshot shows the Oracle SQL Developer interface. The SQL tab contains the provided SQL code. The Data Output tab displays the results of the second query, which retrieves the first 50% of rows from the Employee_details table. The results are as follows:

	customer character varying	customer_id integer	company character varying	country character varying	income text	department character varying
1	Robert Spear	4083	Inky	USA	100,000	Accounting
2	Paul Garza	4364	Kind Ape	USA	400,000	Networking
3	Robert Marquez	4882	Pet Feed	USA	100,000	Finance
4	Natalie Porter	4593	Right App Learning	USA	200,000	Accounting
5	Stevie Bridge	4956	Hackr	USA	100,000	Networking
6	Crystal Doyle	4761	Silvr	Switzerland	200,000	Finance
7	Robert Musser	4027	Dasring	Austria	400,000	Accounting
8	Ann Withers	4236	Didactic	Switzerland	200,000	Networking
9	Paul Hill	4354	Fightrr	Switzerland	400,000	Finance
10	Corinna Schmidt	4510	Kryptis	Austria	100,000	Accounting
11	Walter Miller	4314	Twistr Clothes	Austria	400,000	Networking
12	Paul Wells	4370	Hackr Europe	UK	100,000	Finance
13	Betina Bauer	4831	Pes	Austria	200,000	Accounting
14	Dan Ziegler	4594	Baden Packaging	Germany	100,000	Networking
15	Peter Ramsey	4287	deRamblr	Germany	200,000	Finance

Total rows: 28 Query complete 00:00:00.071 CRLF Ln 89, Col 24

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.

```

```

88 FROM Employee_details
89 FETCH FIRST 30 PERCENT ROWS ONLY;
90
91
92 SELECT *
93 FROM Employee_details
94 WHERE Country = 'USA'
95 FETCH FIRST 3 ROWS ONLY
96

```

Total rows: 3 Query complete 00:00:00.085 CRLF Ln 96, Col 1

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;

```

```

94 WHERE Country = 'USA'
95   FETCH FIRST 3 ROWS ONLY
96
97
98   SELECT *
99     FROM Employee_details
100    ORDER BY Customer DESC
101   LIMIT 3;
102

```

	customer	customer_id	company	country	income	department
1	Wolfgang Ramljak	4299	Arcade	Germany	400,000	Accounting
2	Walter Miller	4314	Twistr Clothes	Austria	400,000	Networking
3	Stevie Bridge	4956	Hackrr	USA	100,000	Networking

Total rows: 3 Query complete 00:00:00.081 CRLF Ln 101, Col 9

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
```

The screenshot shows a SQL query editor window titled "Task 36A - Insert and Import Data by Kelechi Immeldah uwa...". The query history pane contains the following code:

```
28 ---Min and Max--
29
30
31 SELECT *
32 FROM Sales_data
33
34 --3. Let's use Minimum and Maximum function
35
36 SELECT Min(quantity) AS Minimum_order_quantity, Max (quantity) AS Maximum_order_quantity
37 FROM Sales_data
38
39 ----AVERAGE--
40
41 SELECT *
42 FROM Sales_data
43
44 --4. Let's use the Average Function (helps look at the average of a value) and it's written as "Avg"
45
46 SELECT Avg(sales) AS avg_number_of_sales
47 FROM Sales_data
48
49 ---AVERAGE 2nd example--
```

The results pane shows a table with two columns: "minimum_order_quantity" and "maximum_order_quantity". The data row has values 1 and 14 respectively.

Total rows: 1 Query complete 00:00:00.063 CRLF Ln 42, Col 16

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
```

A screenshot of SQL Server Management Studio (SSMS) showing a query results grid. The query is:

```
1 SELECT MIN(sales) AS Smallest_Price, Customer_id
2 FROM Sales_data
3 GROUP BY Customer_id
```

The results show the smallest price and customer ID for each customer. The data is as follows:

	smallest_price	customer_id
1	53.94	LT-16765
2	6.672	SC-20845
3	3.576	AC-10615
4	28.9	BT-11485
5	7.632	JP-15620
6	13.68	CM-11815
7	1.988	LB-16735
8	16.74	JD-16130
9	14.7	JG-15805
10	2.588	TS-21610
11	5.214	RF-19345
12	4.448	AZ-10750
13	11.76	CG-12040
14	8.26	TS-21430
15	2.808	TP-21130
16	1.234	KD-16270
17	7.62	EM-13870

Total rows: 793 Query complete 00:00:00.111

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.
```

A screenshot of SQL Server Management Studio (SSMS) showing a query results grid. The query is:

```
1 SELECT *  
2 FROM Sales_data  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12
```

The results show the total number of customers:

	total_customers
1	9994

Total rows: 1 Query complete 00:00:00.072

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
```

The screenshot shows a SQL query editor window titled "Task 36A—Insert and Import Data by Kelechi Immeldah uwa...". The query pane contains the following code:

```
7  
8  
9  
10 FROM Sales_data  
11  
12  
13 SELECT COUNT(Customer_id) AS Total_customers  
14 FROM Sales_data  
15 WHERE quantity > 12  
16
```

The results pane shows a single row of data:

	total_customer
1	56

Below the results, the status bar indicates "Total rows: 1" and "Query complete 00:00:00.063".

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
```

The screenshot shows a SQL developer interface with the following details:

- Query Editor:** Contains the SQL code from the previous block.
- Data Output:** Shows the results of the query:

total_customer_id	ship_mode
5968	Standard Class
1945	Second Class
543	Same Day
1538	First Class
- Messages:** Shows "Showing rows: 1 to 4" and "Page No: 1 of 1".
- Bottom Status:** "Total rows: 4" and "Query complete 00:00:00.087".
- Bottom Right:** "CRLF" and "Ln 17, Col 1".

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
```

```

16 GROUP BY Ship_Mode
17
18 SELECT COUNT (Customer_ID) AS Total_Customer
19 FROM Sales_Data
20 WHERE Quantity > 12
21
22
23 SELECT SUM(Quantity) AS Total_Quantity
24 FROM Sales_Data
25

```

Data Output Messages Notifications

total_quantity	bigint
1	37873

Showing rows: 1 to 1 Page No: 1 of 1 CRLF Ln 25, Col 1

Total rows: 1 Query complete 00:00:00.120

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

```

```

21
22
23 SELECT SUM(Quantity) AS Total_Quantity
24 FROM Sales_Data
25
26
27 SELECT Customer_ID, SUM(Quantity) AS Total_Quantity
28 FROM Sales_Data
29 GROUP BY Customer_ID
30

```

Data Output Messages Notifications

Customer_ID	total_quantity
LT-16765	8
SC-20845	13
AC-10615	59
BT-11485	11
JP-15520	63
CM-11815	38
LB-16735	12
JD-16150	60
JG-15805	46
TS-21610	44
RF-19345	67
AZ-10750	40
CG-12040	37
TS-21430	31

Showing rows: 1 to 793 Page No: 1 of 1 CRLF Ln 30, Col 1

Total rows: 793 Query complete 00:00:00.095

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

```
Task 36A--Insert and Import Data by Kelechi immeldah uwa... ▾
Query History
Query
26
27
28
29
30
31
32
33
34
SELECT customer_id, SUM(Quantity) AS Total_Quantity
FROM Sales_data
GROUP BY customer_id
35
36
37
38
39
SELECT SUM (Quantity * 10) AS Multiplied_quantity
FROM Sales_data
Data Output Messages Notifications
Showing rows: 1 to 1 Page No: 1 of 1 < << > >>
+-----+-----+
| multiplied_quantity | bigint |
+-----+-----+
| 378730 |          |
+-----+-----+
1
Total rows: 1 | Query complete 00:00:00.077 | CRLF | Ln 33, Col 16
```

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 SOL_AVG_0 FUNCTION

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

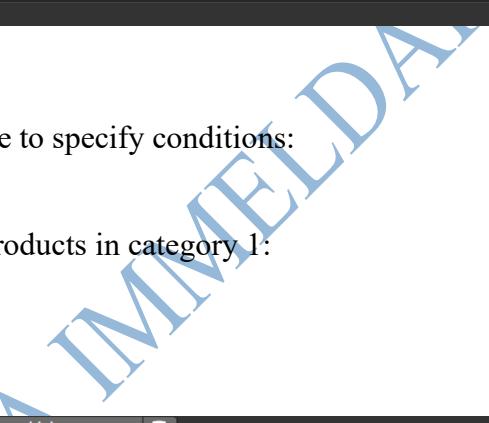
Example

Find the average price of all products:

SELECT AVG (sales)

FROM sales data

Note: NULL values are ignored.



```
Task 36A--Insert and Import Data by Kelechi Immeldah uwa... ▾
```

Query History

```
29 GROUP BY customer_id
30
31
32 SELECT SUM (Quantity * 10) As Multiplied_quantity
33 FROM Sales_data
34
35
36 SELECT AVG(sales)
37 FROM sales_data
38
```

Data Output Messages Notifications

Showing rows: 1 to 1 | Page No: 1 | of 1 | < << > >> |

	avg	double precision
1	229.8580008304938	

Total rows: 1 | Query complete 00:00:00.193 | CRLF | Ln 33, Col 16

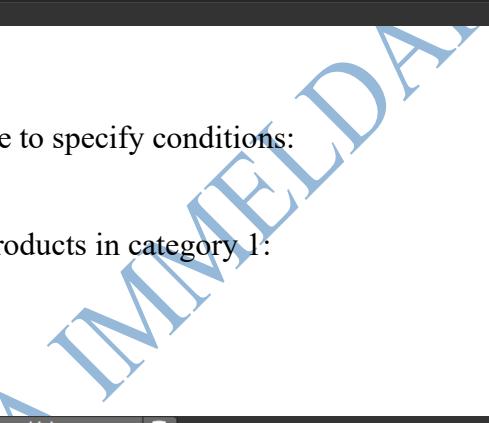
ADD A WHERE CLAUSE

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
```



```
Task 36A--Insert and Import Data by Kelechi Immeldah uwa... ▾
```

Query History

```
34
35
36 SELECT AVG(sales)
37 FROM sales_data
38
39
40 SELECT AVG (sales)
41 FROM Sales_data
42 WHERE order_line = 1
43
```

Data Output Messages Notifications

Showing rows: 1 to 1 | Page No: 1 | of 1 | < << > >> |

	avg	double precision
1	261.96	

Total rows: 1 | Query complete 00:00:00.182 | CRLF | Ln 42, Col 21

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
```

The screenshot shows a SQL query window titled 'Task 36A-Insert and Import Data by Kelechi Immeldah uwa...'. The query is:

```
37  
38  
39  
40    SELECT AVG (sales)  
41    FROM Sales_data  
42    WHERE order_line = 1  
43  
44    SELECT AVG (sales) AS Average_sales  
45    FROM Sales_data  
46
```

The results pane shows a single row of data:

	average_sales
1	229.8580008304938

Total rows: 1 | Query complete 00:00:00.319 | CRLF | Ln 46, Col 1

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
```

```

41   FROM Sales_data
42 WHERE order_line = 1
43
44 SELECT AVG(sales) AS Average_sales
45 FROM Sales_data
46
47 SELECT AVG(sales) AS Average_sales, Customer_id
48 FROM Sales_data
49 GROUP BY Customer_id
50

```

Data Output Messages Notifications

Showing rows: 1 to 793 Page No: 1 of 1

	average_sales	customer_id
	double precision	character varying
1	109.95866666666666	LT-16765
2	70.158	SC-20845
3	140.98255555555556	AC-10615
4	138.39866666666666	BT-11485
5	227.22462500000003	JP-15520
6	167.3888	CM-11815
7	16.72933333333333	LB-16735
8	519.2959117647058	JD-16150
9	167.44644444444447	JG-15805
10	256.4021818181819	TS-21610
11	206.82187500000003	RF-19345
12	135.16816666666668	AZ-10750
13	100.49644444444444	CG-12040
14	293.4485714285714	TS-21430

Total rows: 793 Query complete 00:00:00.472 CRLF Ln 49, Col 21

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%'
```

Task 36A—Insert and Import Data by Kelechi Immeldah uwa...

```

11
12  SELECT *
13  FROM customer_data
14
15  -- 1.'a%' Operator
16  --Let's use the '%a' Like Operator---this will find any value that starts with "a"
17
18  SELECT *
19  FROM customer_data
20  WHERE Customer_name
21  LIKE 'A%'
22
23  --2nd Example--Let's use the 'C%' Like Operator---this will find any value that starts with "c"
24
25  SELECT *

```

Data Output Messages Notifications

	customer_id	customer_name	segment	age	country	city	state	postal_code	region
1	AA-10480	Andrew Allen	Consumer	50	United States	Concord	North Carolina	28027	South
2	AG-10270	Alejandro Grove	Consumer	18	United States	West Jordan	Utah	84084	West
3	AD-10180	Alan Dominguez	Home Office	52	United States	Houston	Texas	77041	Central
4	AM-10360	Alice McCarthy	Corporate	45	United States	Grand Prairie	Texas	75051	Central
5	AH-10210	Alan Hwang	Consumer	58	United States	Brentwood	California	94513	West
6	AG-10495	Andrew Gjertsen	Corporate	24	United States	Philadelphia	Pennsylvania	19140	East
7	AH-10195	Alan Haines	Corporate	67	United States	Tamarac	Florida	33319	South
8	AS-10225	Alan Schoenberger	Corporate	31	United States	New York City	New York	10024	East
9	AG-10625	Andy Gerbode	Corporate	69	United States	Saint Petersburg	Florida	33710	South
10	AC-10420	Alyssa Crouse	Corporate	69	United States	San Francisco	California	94122	West

Total rows: 64 Query complete 00:00:00.213 CRLF | Ln 24, Col 1

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":

Task 36A—Insert and Import Data by Kelechi immeldah uwa...

```

28
29  LIKE 'C$'
30
31  -- 2.'%a' Operator
32  --Let's use the '%h' Like Operator---this will find any value that ends with "h"
33
34  SELECT *
35  FROM customer_data
36  WHERE region
37  LIKE '%h'
38
39  --2nd example --Let's use the '%R' Like Operator---this will find any value that ends with "R"
40
41  SELECT *
42  FROM customer_data
43  WHERE segment

```

Data Output Messages Notifications

	customer_id	customer_name	segment	age	country	city	state	postal_code	region
1	CG-12520	Claire Gute	Consumer	67	United States	Henderson	Kentucky	42420	South
2	SO-20335	Sean O'Donnell	Consumer	65	United States	Fort Lauderdale	Florida	33311	South
3	AA-10480	Andrew Allen	Consumer	50	United States	Concord	North Carolina	28027	South
4	ES-14080	Erin Smith	Corporate	20	United States	Melbourne	Florida	32935	South
5	KD-16270	Karen Daniels	Consumer	59	United States	Springfield	Virginia	22153	South
6	JE-15745	Joel Eaton	Consumer	25	United States	Memphis	Tennessee	38109	South
7	SC-20770	Stewart Carmichael	Corporate	18	United States	Decatur	Alabama	35601	South
8	JC-16105	Julie Creighton	Corporate	56	United States	Durham	North Carolina	27707	South
9	JM-15250	Janet Martin	Consumer	62	United States	Charlotte	North Carolina	28205	South
10	GG-14650	Greg Guthrie	Corporate	51	United States	Bristol	Tennessee	37620	South

Total rows: 134 Query complete 00:00:00.116 CRLF | Ln 41, Col 19

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%'
```

The screenshot shows the MySQL Workbench interface. In the Query Editor, there is a multi-line SQL script. The first part of the script demonstrates the use of the OR operator with wildcards (%). The second part shows the use of the LIKE operator with wildcards (%). The third part shows another example of using wildcards. Below the editor is a results grid displaying customer data from the 'Customer_data' table. The columns include customer_id, customer_name, segment, age, country, city, state, postal_code, and region. The results show 339 rows of data.

customer_id	customer_name	segment	age	country	city	state	postal_code	region
DV-13045	Darrin Van Huff	Corporate	31	United States	Los Angeles	California	90036	West
SO-20335	Sean O'Donnell	Consumer	65	United States	Fort Lauderdale	Florida	33311	South
BH-11710	Brosina Hoffman	Consumer	20	United States	Los Angeles	California	90032	West
AA-10480	Andrew Allen	Consumer	50	United States	Concord	North Carolina	28027	South
ZD-21925	Zuschuss Donatelli	Consumer	66	United States	San Francisco	California	94109	West
EH-13945	Eric Hoffmann	Consumer	21	United States	Los Angeles	California	90049	West
RA-19885	Ruben Ausman	Corporate	51	United States	Los Angeles	California	90049	West
ES-14080	Erin Smith	Corporate	20	United States	Melbourne	Florida	32935	South
JM-15265	Janet Molinari	Corporate	23	United States	New York City	New York	10024	East
TB-21055	Ted Butterfield	Consumer	34	United States	Troy	New York	12180	East

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__%'
```

The screenshot shows the MySQL Workbench interface. In the Query Editor, there is a multi-line SQL script. The first part of the script demonstrates the use of the _r% operator. The second part shows the use of the _e% operator. The third part shows another example of using wildcards. Below the editor is a results grid displaying customer data from the 'Customer_data' table. The columns include customer_id, customer_name, segment, age, country, city, state, postal_code, and region. The results show 79 rows of data.

customer_id	customer_name	segment	age	country	city	state	postal_code	region
BH-11710	Brosina Hoffman	Consumer	20	United States	Los Angeles	California	90032	West
IM-15070	Irene Maddox	Consumer	66	United States	Seattle	Washington	98103	West
EH-13945	Eric Hoffmann	Consumer	21	United States	Los Angeles	California	90049	West
TB-21520	Tracy Blumstein	Consumer	48	United States	Philadelphia	Pennsylvania	19140	East
ES-14080	Erin Smith	Corporate	20	United States	Melbourne	Florida	32935	South
BS-11590	Brendan Sweed	Corporate	45	United States	Gilbert	Arizona	85234	West
GG-14650	Greg Guthrie	Corporate	51	United States	Bristol	Tennessee	37620	South
TS-21610	Troy Stabel	Consumer	33	United States	Phoenix	Arizona	85023	West
EM-13960	Eric Murdock	Consumer	46	United States	Philadelphia	Pennsylvania	19134	East
FM-14290	Frank Merwin	Home Office	45	United States	Los Angeles	California	90032	West

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'
```

A screenshot of a SQL query editor showing the results of a query. The query is:

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

The results table has the following columns and data:

customer_id	customer_name	segment	age	country	city	state	postal_code	region
CG-12520	Claire Gute	Consumer	67	United States	Henderson	Kentucky	42420	South
SO-20335	Sean O'Donnell	Consumer	65	United States	Fort Lauderdale	Florida	33311	South
BH-11710	Brosina Hoffman	Consumer	20	United States	Los Angeles	California	90032	West
AA-10480	Andrew Allen	Consumer	50	United States	Concord	North Carolina	28027	South
IM-15070	Irene Maddox	Consumer	66	United States	Seattle	Washington	98103	West
PK-19075	Pete Kriz	Consumer	46	United States	Madison	Wisconsin	53711	Central
AG-10270	Alejandro Grove	Consumer	18	United States	West Jordan	Utah	84084	West
ZD-21925	Zuschuss Donatelli	Consumer	66	United States	San Francisco	California	94109	West
SF-20065	Sandra Flanagan	Consumer	41	United States	Philadelphia	Pennsylvania	19140	East
EB-13870	Emily Burns	Consumer	34	United States	Orem	Utah	84057	West
EH-13945	Eric Hoffmann	Consumer	21	United States	Los Angeles	California	90049	West
TB-21520	Tracy Blumstein	Consumer	48	United States	Philadelphia	Pennsylvania	19140	East
PO-18865	Patrick O'Donnell	Consumer	64	United States	Westland	Michigan	48185	Central
LH-16900	Lena Hernandez	Consumer	66	United States	Dover	Delaware	19901	East

Total rows: 409 Query complete 00:00:00.279

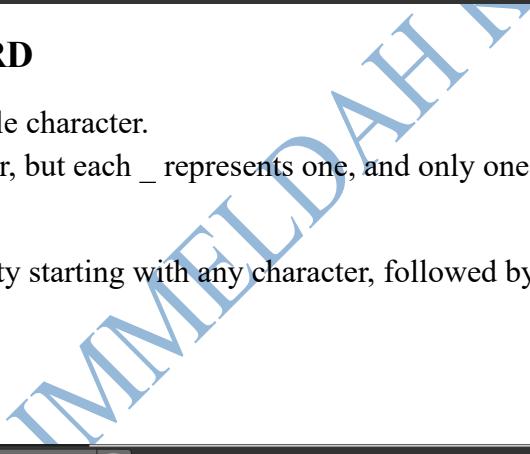
Example

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

```
SELECT *
```

```
FROM Customer_data
```

```
WHERE Customer_id LIKE '%DV%'
```



Task 36A--Insert and Import Data by Kelechi Immeldah uwa...

Query History

```

40 SELECT *
41 FROM customer_data
42 WHERE segment
43 LIKE '%r'
44
45 -- 3.%or% Operator
46 --Let's use the '%or%' Like Operator---this will find any value that has 'or' in any position
47
48 SELECT *
49 FROM customer_data
50 WHERE state
51 LIKE '%or%'
52
53 --2nd Example--Let's use the '%dv%' Like Operator---this will find any value that has 'dv' in any position
54
55 SELECT *
56 FROM customer_data
57 WHERE customer_id
58 LIKE '%DV%'
59
60
61 -- 4._r% Operator
62 --Let's use the '_r%' Like Operator---this will find any value that has 'r' in the second position

```

Data Output

customer_id	customer_name	segment	age	country	city	state	postal_code	region
1 DV-13045	Darrin Van Huff	Corporate	31	United States	Los Angeles	California	90036	West
2 DV-13465	Dianna Vittorini	Consumer	60	United States	Denver	Colorado	80219	West

Total rows: 2 Query complete 00:00:00.237 CRLF | Ln 61, Col 20

USING THE _ WILDCARD

The _ wildcard represents a single character.

It can be any character or number, but each _ represents one, and only one, character.

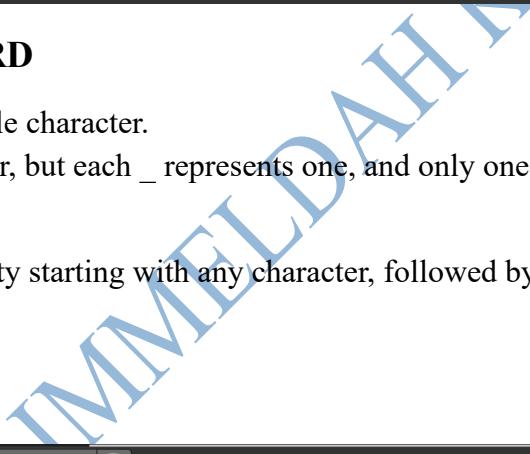
Example

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

```

SELECT *
FROM Customer_data
WHERE City LIKE '_oncord'

```



Task 36A--Insert and Import Data by Kelechi Immeldah uwa...

Query History

```

1 SELECT *
2 FROM Customer_data
3 WHERE City LIKE '_oncord'
4

```

Data Output

customer_id	customer_name	segment	age	country	city	state	postal_code	region
1 AA-10480	Andrew Allen	Consumer	50	United States	Concord	North Carolina	28027	South
2 SC-20095	Sanjit Chand	Consumer	27	United States	Concord	California	94521	West
3 SB-20185	Sarah Brown	Consumer	31	United States	Concord	New Hampshire	3301	East

Total rows: 3 Query complete 00:00:00.272 CRLF | Ln 4, Col 1

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

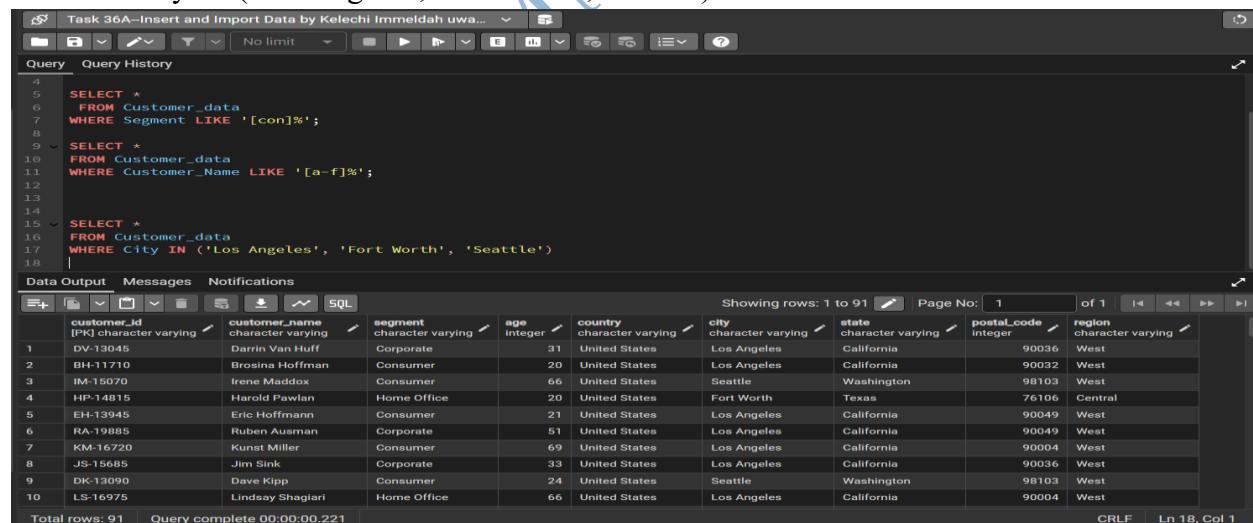
THE SQL IN OPERATOR

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')
```



The screenshot shows a SQL IDE interface with three distinct code snippets demonstrating the use of the IN operator:

- Query 1: SELECT * FROM Customer_data WHERE Segment LIKE '[con]%' ;
- Query 2: SELECT * FROM Customer_data WHERE Customer_Name LIKE '[a-f]%' ;
- Query 3: SELECT * FROM Customer_data WHERE City IN ('Los Angeles', 'Fort Worth', 'Seattle')

Below the queries, the Data Output tab displays a table with 91 rows of customer data. The columns include customer_id, customer_name, segment, age, country, city, state, postal_code, and region. The data shows various customers from different cities like Los Angeles, Fort Worth, and Seattle, each associated with a specific segment and region.

customer_id	customer_name	segment	age	country	city	state	postal_code	region
DV-13045	Darrin Van Huff	Corporate	31	United States	Los Angeles	California	90036	West
BH-17710	Brosina Hoffman	Consumer	20	United States	Los Angeles	California	90032	West
IM-15070	Irene Maddox	Consumer	66	United States	Seattle	Washington	98103	West
HP-14815	Harold Pawlan	Home Office	20	United States	Fort Worth	Texas	76106	Central
EH-13945	Eric Hoffmann	Consumer	21	United States	Los Angeles	California	90049	West
RA-19885	Ruben Ausman	Corporate	51	United States	Los Angeles	California	90049	West
KM-16720	Kunst Miller	Consumer	69	United States	Los Angeles	California	90004	West
JS-15685	Jim Sink	Corporate	33	United States	Los Angeles	California	90036	West
DK-13090	Dave Kipp	Consumer	24	United States	Seattle	Washington	98103	West
LS-16975	Lindsay Shagari	Home Office	66	United States	Los Angeles	California	90004	West

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')
```

```
Task 36A-Insert and Import Data by Kelechi Immeldah uwa... ▾
```

```

9   SELECT *
10  FROM Customer_data
11  WHERE Customer_Name LIKE '[a-f]%' ;
12
13
14
15  SELECT *
16  FROM Customer_data
17  WHERE City IN ('Los Angeles', 'Fort Worth', 'Seattle')
18
19
20  SELECT *
21  FROM Customer_data
22  WHERE City NOT IN ('Los Angeles', 'Fort Worth', 'Seattle')
23

```

Data Output Messages Notifications

customer_id	customer_name	segment	age	country	city	state	postal_code	region
CG-12520	Claire Gute	Consumer	67	United States	Henderson	Kentucky	42420	South
SO-20335	Sean O'Donnell	Consumer	65	United States	Fort Lauderdale	Florida	33311	South
AA-10480	Andrew Allen	Consumer	50	United States	Concord	North Carolina	28027	South
PK-19075	Pete Kriz	Consumer	46	United States	Madison	Wisconsin	53711	Central
AG-10270	Alejandro Grove	Consumer	18	United States	West Jordan	Utah	84084	West
ZD-21925	Zuschuss Donatelli	Consumer	66	United States	San Francisco	California	94109	West
KB-16585	Ken Black	Corporate	67	United States	Fremont	Nebraska	68025	Central
SF-20065	Sandra Flanagan	Consumer	41	United States	Philadelphia	Pennsylvania	19140	East
EB-13870	Emily Burns	Consumer	34	United States	Orem	Utah	84057	West
TB-21520	Tracy Blumstein	Consumer	48	United States	Philadelphia	Pennsylvania	19140	East

Total rows: 702 Query complete 00:00:00.359 CRLF Ln 22, Col 59

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)
```

```
Task 36A-Insert and Import Data by Kelechi Immeldah uwa... ▾
```

```

15  SELECT *
16  FROM Customer_data
17  WHERE City IN ('Los Angeles', 'Fort Worth', 'Seattle')
18
19
20
21  SELECT *
22  FROM Customer_data
23  WHERE City NOT IN ('Los Angeles', 'Fort Worth', 'Seattle')
24
25
26  SELECT *
27  FROM Customer_data
28  WHERE Customer_ID IN (SELECT Customer_ID FROM Customer_data)
29

```

Data Output Messages Notifications

customer_id	customer_name	segment	age	country	city	state	postal_code	region
CG-12520	Claire Gute	Consumer	67	United States	Henderson	Kentucky	42420	South
DV-13045	Darrin Van Huff	Corporate	31	United States	Los Angeles	California	90036	West
SO-20335	Sean O'Donnell	Consumer	65	United States	Fort Lauderdale	Florida	33311	South
BH-11710	Brosina Hoffman	Consumer	20	United States	Los Angeles	California	90032	West
AA-10480	Andrew Allen	Consumer	50	United States	Concord	North Carolina	28027	South
IM-15070	Irene Maddox	Consumer	66	United States	Seattle	Washington	98103	West
HP-14815	Harold Pawlan	Home Office	20	United States	Fort Worth	Texas	76106	Central
PK-19075	Pete Kriz	Consumer	46	United States	Madison	Wisconsin	53711	Central
AG-10270	Alejandro Grove	Consumer	18	United States	West Jordan	Utah	84084	West
ZD-21925	Zuschuss Donatelli	Consumer	66	United States	San Francisco	California	94109	West

Total rows: 793 Query complete 00:00:00.293 CRLF Ln 28, Col 62

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
```

The screenshot shows a SQL query editor window with the following details:

- Query History:** Task 36A--Insert and Import Data by Kelechi Immeldah uwa...
- Code:**

```
34
35
36
37 SELECT *
38 FROM Customer_data
39 WHERE Customer_ID NOT IN (SELECT Customer_ID FROM Customer_data);
40
41
42
43
44
45 SELECT *
46 FROM sales_data
47 WHERE Quantity BETWEEN 10 AND 15
```
- Data Output:** A table showing 170 rows of product data. The columns include:
 - 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 precision
- Table Data Preview:** The table contains 170 rows of data, with the first few rows shown below:

order_line	order_id	order_date	ship_date	ship_mode	customer_id	product_id	sales	quantity	discount	profit	
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-10002...	384.45	11	0	103.6
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	8.0
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	411	CA-2017-117457	2017-12-08	2017-12-12	Standard Class	KH-16510	EUR-RO-100019	1336.829	12	0.15	31.4
- Total rows:** 170 **Query complete:** 00:00:00.235
- Bottom Status:** CRLF | Ln 47, Col 33

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
```

```

44
45 SELECT *
46 FROM sales_data
47 WHERE Quantity BETWEEN 10 AND 15
48
49
50
51
52
53
54
55 SELECT *
56 FROM sales_data
57 WHERE Quantity NOT BETWEEN 10 AND 15;
58

```

Data Output Messages Notifications

Showing rows: 1 to 1000 Page No: 1 of 10

order_line	order_id	order_date	ship_date	ship_mode	customer_id	product_id	sales	quantity	discount	profit	
[PK] integer	text	date	text	character varying	text	text	double precision	integer	double precision	double precision	
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-100002...	14.62	2	0	6.8
4	4	US-2015-108966	2015-10-11	2015-10-18	Standard Class	SO-20335	FUR-TA-100005...	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	2.5
6	6	CA-2014-115812	2014-06-09	2014-06-14	Standard Class	BH-11710	FUR-FU-100014...	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	22.

Total rows: 9824 Query complete 0:00:00.423 CRLF Ln 57, Col 38

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

```

Task 36A-Insert and Import Data by Kelechi Immeldah uwa...

```

1 SELECT *
2 FROM Customer_data
3
4 SELECT *
5 FROM Customer_data
6
7 SELECT *
8 FROM Customer_data
9 WHERE City BETWEEN 'Henderson' AND 'Newark'
10 ORDER BY city
11

```

Data Output Messages Notifications

Showing rows: 1 to 288 Page No: 1 of 1

	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	DK-12985	Darren Koutras	Consumer	58	United States	Henderson	Kentucky	42420	South
2	RD-19585	Rob Dowd	Consumer	60	United States	Henderson	Kentucky	42420	South
3	CG-12520	Claire Gute	Consumer	67	United States	Henderson	Kentucky	42420	South
4	GK-14620	Grace Kelly	Corporate	32	United States	Hesperia	California	92345	West
5	SC-20695	Steve Chapman	Corporate	46	United States	Hialeah	Florida	33012	South
6	LB-16795	Laurel Beltran	Home Office	18	United States	Highland Park	Illinois	60035	Central
7	CA-11965	Carol Adams	Corporate	19	United States	Hoover	Alabama	35244	South
8	DD-13570	Dorothy Dickinson	Consumer	32	United States	Houston	Texas	77041	Central
9	GT-14710	Greg Tran	Consumer	67	United States	Houston	Texas	77070	Central
10	HL-15040	Hunter Lopez	Consumer	65	United States	Houston	Texas	77095	Central

Total rows: 288 Query complete 00:00:00.119 CRLF Ln 11, Col 1

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

```

Task 36A-Insert and Import Data by Kelechi Immeldah uwa...

```

17
18
19
20
21
22
23
24
25
26
27
    SELECT *
    FROM Customer_data
    WHERE Customer_name NOT BETWEEN 'Claire Gute' AND 'Susan Mackendrick'
    ORDER BY Customer_name

```

Data Output Messages Notifications

Showing rows: 1 to 234 Page No: 1 of 1

	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	AB-10015	Aaron Bergman	Consumer	66	United States	Seattle	Washington	98103	West
2	AH-10030	Aaron Hawkins	Corporate	60	United States	Philadelphia	Pennsylvania	19134	East
3	AS-10045	Aaron Smayling	Corporate	42	United States	Jacksonville	North Carolina	28540	South
4	AB-10060	Adam Bellavance	Home Office	25	United States	New York City	New York	10009	East
5	AH-10075	Adam Hart	Corporate	21	United States	New York City	New York	10011	East
6	AS-10090	Adam Shillingsburg	Consumer	46	United States	Charlottesville	Virginia	22901	South
7	AB-10105	Adrian Barton	Consumer	63	United States	Phoenix	Arizona	85023	West
8	AH-10120	Adrian Hane	Home Office	27	United States	Tucson	Arizona	85705	West
9	AS-10135	Adrian Shami	Home Office	65	United States	New York City	New York	10035	East
10	AB-10150	Aimee Bixby	Consumer	65	United States	Long Beach	New York	11561	East
11	AH-10165	Alan Barnes	Consumer	22	United States	Los Angeles	California	90036	West
12	AD-10180	Alan Dominguez	Home Office	52	United States	Houston	Texas	77041	Central
13	AH-10195	Alan Haines	Corporate	67	United States	Tamarac	Florida	33319	South
14	AH-10210	Alan Weiss	Consumer	59	United States	Brentwood	California	94513	West

Total rows: 234 Query complete 00:00:00.139 CRLF Ln 27, Col 1

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'.

```

The screenshot shows a MySQL Workbench interface with a query editor and a results grid. The query editor contains the following SQL code:

```

23
24
25
26
27
28 SELECT *
29 FROM sales_data
30 WHERE Order_date
31 BETWEEN '2016-12-01'
32 AND
33 '2016-12-31'

```

The results grid displays 352 rows of data from the sales_data table, with columns including order_line, order_id, order_date, ship_date, ship_mode, customer_id, product_id, sales, quantity, discount, and profit.

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 precision	
1	14	CA-2016-1613...	2016-12-05	2016-12-10	Standard Class	IM-15070	OFF-BI-10003656	407.976	3	0.2	132.5
2	22	CA-2016-1373...	2016-12-09	2016-12-13	Standard Class	KB-16585	OFF-AR-100002...	19.46	7	0	5.0
3	23	CA-2016-1373...	2016-12-09	2016-12-13	Standard Class	KB-16585	OFF-AP-100014...	60.34	7	0	15.6
4	36	CA-2016-1175...	2016-12-08	2016-12-10	First Class	GH-14485	TEC-PH-100049...	1097.544	7	0.2	123.4
5	37	CA-2016-1175...	2016-12-08	2016-12-10	First Class	GH-14485	FUR-FU-100036...	190.92	5	0.6	-147.1
6	54	CA-2016-1058...	2016-12-11	2016-12-17	Standard Class	JM-15265	OFF-FA-100003...	15.26	7	0	6.2
7	55	CA-2016-1058...	2016-12-11	2016-12-17	Standard Class	JM-15265	TEC-PH-100024...	1029.95	5	0	298.6
8	103	CA-2016-1299...	2016-12-01	2016-12-04	Second Class	GZ-14470	OFF-PA-100040...	23.92	4	0	11.7
9	148	CA-2016-1144...	2016-12-05	2016-12-09	Standard Class	JE-16165	TEC-PH-100002...	384.45	11	0	103.8
10	149	CA-2016-1144...	2016-12-05	2016-12-09	Standard Class	JE-16165	TEC-PH-100014...	149.97	3	0	5.9
11	150	CA-2016-1144...	2016-12-05	2016-12-09	Standard Class	JE-16165	FUR-CH-100004...	1951.84	8	0	585.1
12	151	CA-2016-1144...	2016-12-05	2016-12-09	Standard Class	JE-16165	OFF-BI-10002735	171.55	5	0	80.6
13	250	CA-2016-1547...	2016-12-10	2016-12-15	Second Class	LH-17155	FUR-CH-100029...	321.568	2	0.2	28.1

Total rows: 352 Query complete 00:00:00.104 CRLF Ln 33, Col 13

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

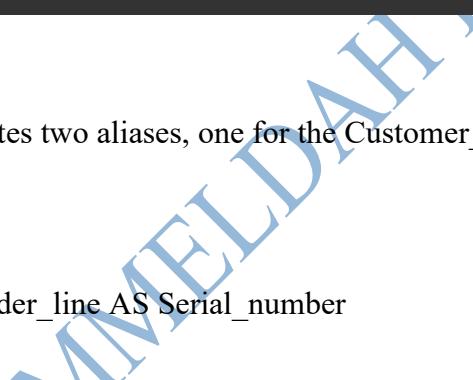
Actually, in most database languages, you can skip the AS keyword and get the same result:

Example

```

SELECT Customer_ID ID
FROM Sales_data

```



Screenshot of a SQL query editor showing the results of a SELECT statement. The query selects all columns from the sales_data table where the Order_date is between '2016-12-01' and '2016-12-31', and then selects Customer_id AS id from the same table.

```

27
28     SELECT *
29     FROM sales_data
30     WHERE Order_date
31     BETWEEN '2016-12-01'
32     AND
33     '2016-12-31'
34
35
36     SELECT Customer_id AS id
37     FROM sales_data

```

The resulting data table has one column named 'id' containing the following values:

id
CG-12520
CG-12520
DV-13045
SO-20335
SO-20335
BH-11710
AA-10480
IM-15070

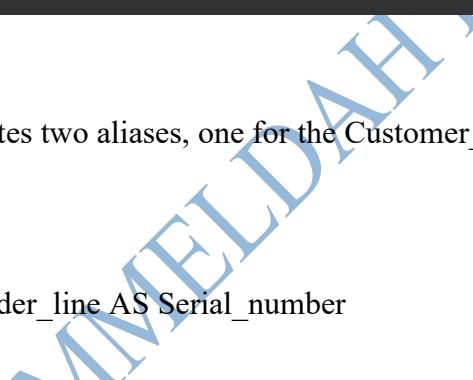
Total rows: 9994 | Query complete 00:00:00.082 | CRLF | Ln 37, Col 16

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
```



Screenshot of a SQL query editor showing the results of a SELECT statement. The query selects Customer_ID AS ID and Order_line AS Serial_number from the Sales_data table.

```

34
35
36     SELECT Customer_id AS id
37     FROM sales_data
38
39
40
41
42     SELECT Customer_ID AS ID, Order_line AS Serial_number
43     FROM Sales_data
44

```

The resulting data table has two columns: 'id' and 'serial_number'. The 'id' column contains the same values as the previous screenshot, and the 'serial_number' column contains the following values:

id	serial_number
CG-12520	1
CG-12520	2
DV-13045	3
SO-20335	4
SO-20335	5
BH-11710	6
BH-11710	7
BH-11710	8
BH-11710	9
BH-11710	10
BH-11710	11
BH-11710	12
AA-10480	13
IM-15070	14

Total rows: 9994 | Query complete 00:00:00.097 | CRLF | Ln 44, Col 1

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
```

The screenshot shows a MySQL Workbench interface. The query editor window contains the following SQL code:

```
39  
40  
41  
42 SELECT Customer_ID AS ID, Order_line AS Serial_number  
43 FROM Sales_data  
44  
45  
46  
47 SELECT segment AS [My_Product_category]  
48 FROM Customer_data  
49
```

The results pane displays a table with one column named 'my_product_category' containing the values: Consumer, Corporate, Consumer, Consumer, Consumer, Home Office, Consumer, Consumer, Consumer, Consumer, Corporate, Consumer, Consumer, Consumer. The table has 14 rows.

At the bottom of the interface, it says "Total rows: 793" and "Query complete 00:00:00.125".

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
```

```

48 SELECT segment AS [My_Product_category]
49 FROM Customer_data
50
51
52
53
54 SELECT Customer_Name, CONCAT(Segment, ', ', Postal_Code, ', ', City, ', ', State) AS Product_Address
55 FROM Customer_data
56
57

```

	customer_name	product_address
1	Claire Gute	Consumer, 42420, Henderson, Kentucky
2	Darrin Van Huff	Corporate, 90036, Los Angeles, California
3	Sean O'Donnell	Consumer, 33311, Fort Lauderdale, Florida
4	Brosina Hoffman	Consumer, 90032, Los Angeles, California
5	Andrew Allen	Consumer, 28027, Concord, North Carolina
6	Irene Maddox	Consumer, 98103, Seattle, Washington
7	Harold Pawlan	Home Office, 76106, Fort Worth, Texas
8	Pete Kriz	Consumer, 53711, Madison, Wisconsin
9	Alejandro Grove	Consumer, 84084, West Jordan, Utah
10	Zuschuss Donatelli	Consumer, 94109, San Francisco, California
11	Ken Black	Corporate, 68025, Fremont, Nebraska
12	Sandra Flanagan	Consumer, 19140, Philadelphia, Pennsylvania
13	Emily Burns	Consumer, 84057, Orem, Utah
14	Eric Hoffmann	Consumer, 90049, Los Angeles, California

Total rows: 793 Query complete 00:00:00.075 CRLF Ln 54, Col 47

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

```

```

58
59
60
61
62
63
64
65
66 SELECT *
67 FROM Customer_data AS Customer_Info_data
68

```

	customer_id	customer_name	segment	age	country	city	state	postal_code	region
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	EU-12045	Eric Hoffmann	Consumer	21	United States	Los Angeles	California	90049	West

Total rows: 793 Query complete 00:00:00.140 CRLF Ln 65, Col 1

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

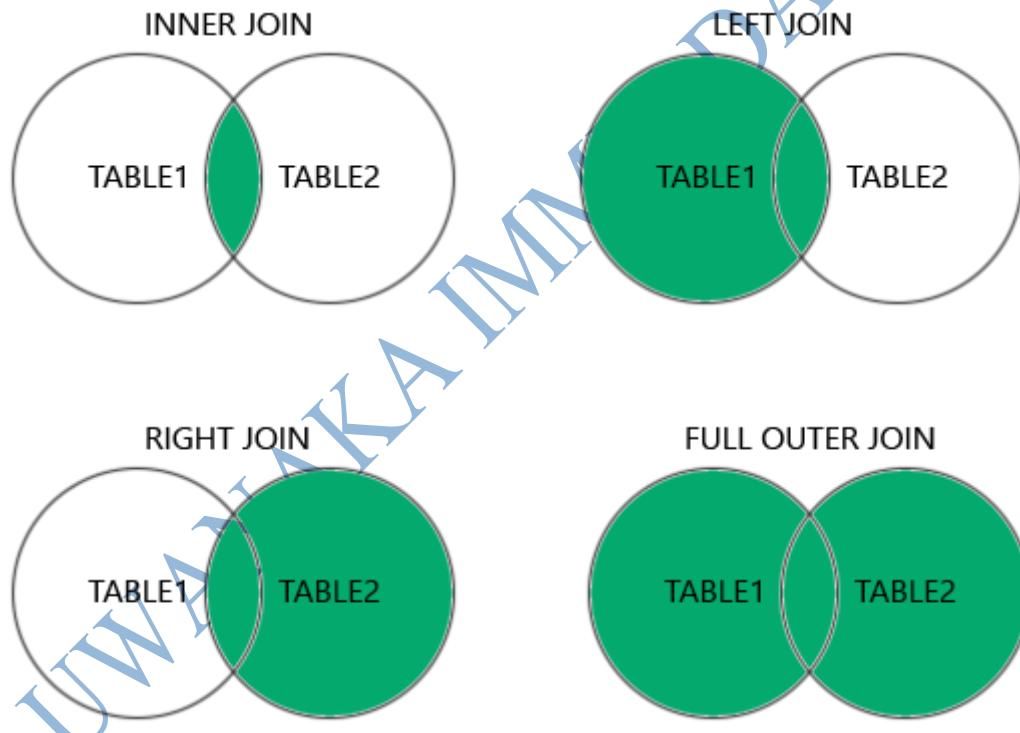
SQL JOIN

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:

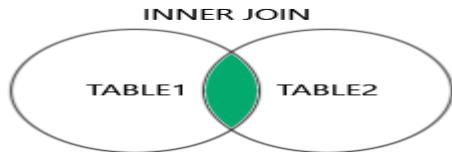
- **INNER JOIN:** Returns records that have matching values in both tables
- **LEFT (OUTER) JOIN:** Returns all records from the left table, and the matched records from the right table
- **RIGHT (OUTER) JOIN:** Returns all records from the right table, and the matched records from the left table
- **FULL (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
```

	customer_name	city	sales
1	Darrin Van Huff	Los Angeles	14.62
2	Brosina Hoffman	Los Angeles	48.86
3	Brosina Hoffman	Los Angeles	7.28
4	Brosina Hoffman	Los Angeles	907.152
5	Brosina Hoffman	Los Angeles	18.504
6	Brosina Hoffman	Los Angeles	114.9
7	Brosina Hoffman	Los Angeles	1706.184
8	Brosina Hoffman	Los Angeles	911.424
9	Irene Maddox	Seattle	407.976
10	Alejandro Grove	West Jordan	55.5
11	Zuschuss Donatelli	San Francisco	8.56

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

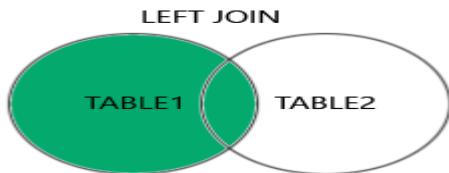
JOIN and INNER JOIN will return the same result.

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;
```

The screenshot shows a SQL query editor interface. The top bar includes a title 'Task 36A-Insert and Import Data by Kelechi Immeldah uwa...', a dropdown menu, and various toolbar icons. The main area is divided into 'Query' and 'Query History' tabs, with the 'Query' tab selected. The query code is listed in the 'Query' tab, starting from line 22 and ending at line 31. The results are displayed in a table below, with 14 rows of data. The table has columns: customer_name, age, segment, sales, and quantity. The bottom status bar indicates 'Total rows: 9994' and 'Query complete 00:00:00.303'.

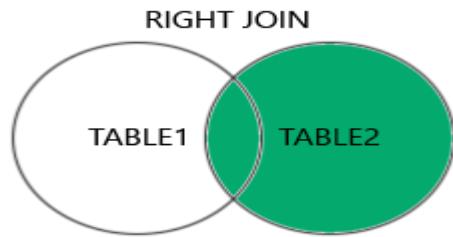
	customer_name	age	segment	sales	quantity
1	Claire Gute	67	Consumer	261.96	2
2	Claire Gute	67	Consumer	731.94	3
3	Darrin Van Huff	31	Corporate	14.62	2
4	Sean O'Donnell	65	Consumer	957.5775	5
5	Sean O'Donnell	65	Consumer	22.368	2
6	Brosina Hoffman	20	Consumer	48.86	7
7	Brosina Hoffman	20	Consumer	7.28	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
```

Screenshot of a SQL query editor showing the execution of a RIGHT JOIN query. The query selects customer_id, customer_name, state, and city from Customer_data and Western_customers respectively, using a RIGHT JOIN and matching customer_id. The results show 793 rows, with many entries having null values for state and city.

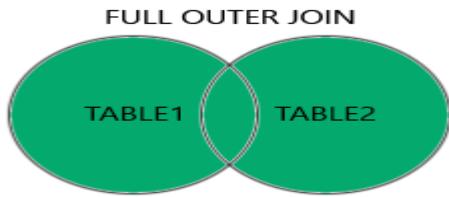
customer_id	customer_name	state	city
CG-12520	Claire Gute	[null]	[null]
DV-13045	Darrin Van Huff	California	Los Angeles
SO-20335	Sean O'Donnell	[null]	[null]
BH-11710	Brosina Hoffman	California	Los Angeles
AA-10480	Andrew Allen	[null]	[null]
IM-15070	Irene Maddox	Washington	Seattle
HP-14815	Harold Pawlan	[null]	[null]
PK-19075	Pete Kriz	[null]	[null]
AG-10270	Alejandro Grove	Utah	West Jordan
ZD-21925	Zuschuss Donatelli	California	San Francisco
KB-16585	Ken Black	[null]	[null]
SF-20065	Sandra Flanagan	[null]	[null]
FR-13870	Emily Burns	Utah	Orem

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
```

The screenshot shows a SQL query editor interface with the following details:

- Query History:** Task 36A-Insert and Import Data by Kelechi Immeldah uwa...
- Query:** The SQL code for a Full Outer Join between Western_customers and Sales_data.
- Data Output:** A table showing the results of the query. The columns are customer_name, city, and sales.
- Results:** The table contains 15 rows of data, including rows where either customer_name or city is null, demonstrating the nature of a Full Outer Join.
- Statistics:** Total rows: 9994, Query complete 00:00:00.387, CRLF, Ln 45, Col 36.

	customer_name	city	sales
1	[null]	[null]	261.96
2	[null]	[null]	731.94
3	Darrin Van Huff	Los Angeles	14.62
4	[null]	[null]	957.5775
5	[null]	[null]	22.368
6	Brosina Hoffman	Los Angeles	48.86
7	Brosina Hoffman	Los Angeles	7.28
8	Brosina Hoffman	Los Angeles	907.152
9	Brosina Hoffman	Los Angeles	18.504
10	Brosina Hoffman	Los Angeles	114.9
11	Brosina Hoffman	Los Angeles	1706.184
12	Brosina Hoffman	Los Angeles	911.424
13	[null]	[null]	15.552
14	Irene Maddox	Seattle	407.976
15	[null]	[null]	68.81

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 "Western 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.
```

The screenshot shows a SQL development environment with the following details:

- Query Editor:** The code area contains the SQL query for a self-join on the "Customer_data" table.
- Data Output:** The results are displayed in a grid table with three columns: "customer_id", "customer_name", and "customer_id". The data shows 793 rows where two different customer IDs map to the same name.
- Information:** At the bottom, it says "Total rows: 793" and "Query complete 00:00:00.337".
- Status:** In the bottom right, it shows "CRLF | Ln 49, Col 1".

	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;
```

The screenshot shows a SQL query editor interface with a dark theme. At the top, there's a toolbar with various icons. Below it is a 'Query History' pane showing the executed SQL code. The main area is a 'Data Output' grid displaying the results of the query. The grid has a header row 'city character varying' with a lock icon. Below are 15 rows of city names, each preceded by a number from 1 to 15. The bottom of the screen shows the total rows (252), a 'Query complete' message, and some status information.

	city character varying
1	Akron
2	Albuquerque
3	Allen
4	Amarillo
5	Apple Valley
6	Arlington
7	Arlington Heights
8	Arvada
9	Asheville
10	Atlanta
11	Auburn
12	Aurora
13	Austin
14	Baltimore
15	Belleville

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;
```

The screenshot shows a SQL query editor interface with the following details:

- Title Bar:** Task 36A—Insert and Import Data by Kelechi Immeldah uwa...
- Toolbar:** Includes icons for file, edit, search, and database navigation.
- Query History:** Shows the history of queries run.
- Query Editor:** Contains the following SQL code:

```
70
71
72
73     SELECT City
74     FROM Customer_data
75   UNION ALL
76     SELECT City |
77     FROM Western_customers
78   ORDER BY City;
```
- Data Output:** A table showing the results of the query, with columns labeled "city" and "character varying". The data includes:

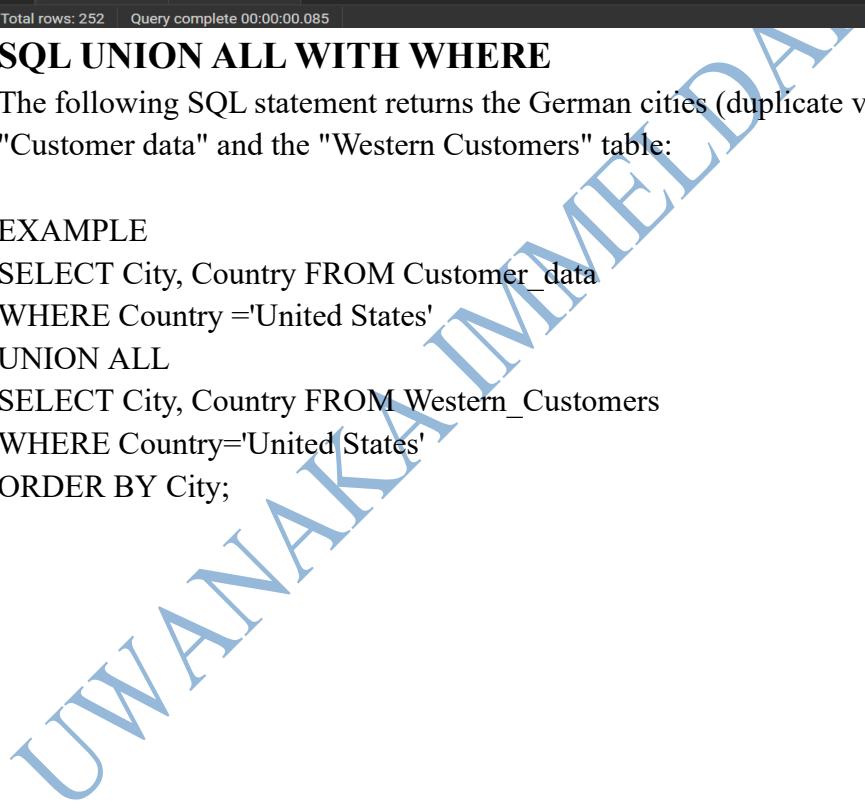
	city	character varying
1	Akron	
2	Akron	
3	Albuquerque	
4	Albuquerque	
5	Albuquerque	
6	Albuquerque	
7	Allen	
8	Amarillo	
9	Apple Valley	
10	Apple Valley	
11	Arlington	
12	Arlington	
13	Arlington	
14	Arlington	
15	Arlington	
- Status Bar:** Total rows: 1048 | Query complete 00:00:00.073 | CRLF | Ln 76, Col 13

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;
```



Task 36A--Insert and Import Data by Kelechi Immeldah uwa...

No limit

Query History

```
30
31
32 SELECT City, Country FROM Customer_data
33 WHERE Country = 'United States'
34 UNION
35 SELECT City, Country FROM Western_customers
36 WHERE Country= 'United States'
37 ORDER BY City;
```

Data Output Messages Notifications

Showing rows: 1 to 252 Page No: 1 of 1

	city character varying	country character varying
1	Akron	United States
2	Albuquerque	United States
3	Allen	United States
4	Amarillo	United States
5	Apple Valley	United States
6	Arlington	United States
7	Arlington Heights	United States
8	Arvada	United States
9	Asheville	United States
10	Atlanta	United States
11	Auburn	United States
12	Aurora	United States
13	Austin	United States
14	Baltimore	United States
15	Belleville	United States

Total rows: 252 Query complete 00:00:00.085 CRLF Ln 80, Col 1

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;
```

Task 36A-Insert and Import Data by Kelechi Immeldah uwa... ▾

No limit ▾ E ▾

Query History

90

91

92 ▾ `SELECT City, Country FROM Customer_data`

93 `WHERE Country = 'United States'`

94 `UNION ALL`

95 `SELECT City, Country FROM Western_Customers`

96 `WHERE Country='United States'`

97 `ORDER BY City;`

98

Data Output Messages Notifications

SQL

Showing rows: 1 to 1000 Page No: 1 of 2

	city character varying	country character varying
1	Akron	United States
2	Akron	United States
3	Albuquerque	United States
4	Albuquerque	United States
5	Albuquerque	United States
6	Albuquerque	United States
7	Allen	United States
8	Amarillo	United States
9	Apple Valley	United States
10	Apple Valley	United States
11	Arlington	United States
12	Arlington	United States
13	Arlington	United States
14	Arlington	United States
15	Arlington	United States

Total rows: 1048 Query complete 00:00:00.094 CRLF Ln 97, Col 15

ANOTHER UNION EXAMPLE

The following SQL statement lists all customers and suppliers:

EXAMPLE

```
SELECT 'Customer_data' AS Type, Customer_name, City, Country  
FROM Customer
```

FROM Customer_data
WHERE

UNION

```
SELECT 'Western_Customers', Customer_Name, City, Country
```

FROM Western_Customers;

The screenshot shows a SQL query window in SSMS. The query is:

```

101  FROM Customer_data
102
103
104  SELECT 'Customer_data' AS Type, Customer_name, City, Country
105  FROM Customer_data
106  UNION
107  SELECT 'Western_Customers', Customer_Name, City, Country
108  FROM Western_Customers;
109

```

The results grid displays data from two tables: Customer_data and Western_Customers. The columns are type, customer_name, city, and country. The data includes rows for various customers across different cities and countries.

	type	customer_name	city	country
1	Customer_data	Carl Jackson	Philadelphia	United States
2	Customer_data	Nora Pelletier	Niagara Falls	United States
3	Customer_data	Erica Smith	San Francisco	United States
4	Customer_data	Christine Kargatis	Orem	United States
5	Customer_data	Rick Bensley	Chicago	United States
6	Customer_data	David Flashing	Vallejo	United States
7	Customer_data	Beth Paige	Evanston	United States
8	Customer_data	Corinna Mitchell	Los Angeles	United States
9	Customer_data	Art Foster	Louisville	United States
10	Customer_data	Anthony Rawles	Vancouver	United States
11	Western_Customers	Kean Takahito	Los Angeles	United States
12	Customer_data	Christine Sundaresam	Roswell	United States
13	Customer_data	Ryan Crowe	Columbus	United States
14	Customer_data	Toby Swindell	Houston	United States
15	Customer_data	Anemone Ratner	Columbus	United States

Total rows: 1048 Query complete 00:00:00.089 CRLF Ln 109, Col 1

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

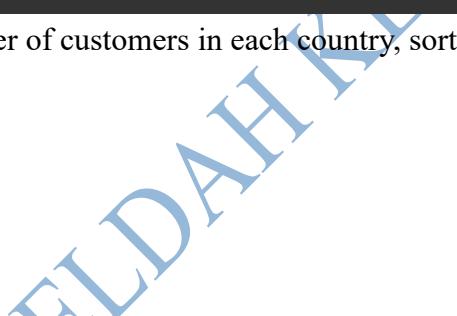
The following SQL statement lists the number of customers in each region:

Example

```

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

```



```
Task 36A--Insert and Import Data by Kelechi Immeldah uwa... ▾
```

Query History

```
108 FROM Western_Customers;
109
110
111 | SELECT COUNT(Customer_ID), Region
112 | FROM Customer_data
113 | GROUP BY region;
114
```

Data Output Messages Notifications

	count	region
1	134	South
2	255	West
3	220	East
4	184	Central

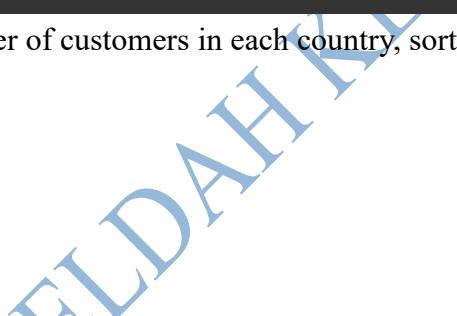
Showing rows: 1 to 4 | Page No: 1 of 1 | < > << >> | SQL

Total rows: 4 | Query complete 00:00:00.105 | CRLF | Ln 110, Col 1

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;
```



```
Task 36A--Insert and Import Data by Kelechi Immeldah uwa... ▾
```

Query History

```
116
117 | SELECT COUNT(Customer_ID), region
118 | FROM Customer_data
119 | GROUP BY region
120 | ORDER BY COUNT(Customer_ID) DESC;
121
```

Data Output Messages Notifications

	count	region
1	255	West
2	220	East
3	184	Central
4	134	South

Showing rows: 1 to 4 | Page No: 1 of 1 | < > << >> | SQL

Total rows: 4 | Query complete 00:00:00.098 | CRLF | Ln 121, Col 1

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 screenshot shows the SQL Server Management Studio interface. The query window contains the following code:

```
1.1 / SELECT COUNT(Customer_ID), region
1.1.1 FROM Customer_data
1.1.2 GROUP BY region
1.1.3 ORDER BY COUNT(Customer_ID) DESC;
1.2.1
1.2.2
1.2.3
1.2.4 SELECT COUNT(Customer_ID), Country
1.2.5 FROM Customer_data
1.2.6 GROUP BY Country
1.2.7 HAVING COUNT(Customer_ID) > 5;
1.2.8
```

The results pane shows a single row of data:

count	country
793	United States

Total rows: 1 Query complete 00:00:00.108 CRLF Ln 128, Col 1

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;
```

The screenshot shows the SQL Server Management Studio interface. The query window contains the following code:

```
1.2.3
1.2.4 SELECT COUNT(Customer_ID), Country
1.2.5 FROM Customer_data
1.2.6 GROUP BY Country
1.2.7 HAVING COUNT(Customer_ID) > 5;
1.2.8
1.2.9
1.2.10 SELECT COUNT(Customer_ID), Country
1.2.11 FROM Customer_data
1.2.12 GROUP BY Country
1.2.13 HAVING COUNT(Customer_ID) > 5
1.2.14 ORDER BY COUNT(Customer_ID) DESC;
1.2.15
```

The results pane shows a single row of data:

count	country
793	United States

Total rows: 1 Query complete 00:00:00.077 CRLF Ln 135, Col 1

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_da  
ta.Customer_ID = Western_Customers.Customer_ID AND Age < 20);
```

The screenshot shows a SQL query editor window titled "Task 36A-Insert and Import Data by Kelechi Immeldah uwa...". The query pane contains the following SQL code:

```
134 ORDER BY COUNT(Customer_ID) DESC;  
135  
136 SELECT*  
137 FROM Customer_data  
138  
139 SELECT Customer_Name  
140 FROM Customer_data  
141 WHERE EXISTS (SELECT Customer_Name  
142 FROM Western_customers  
143 WHERE Customer_data.Customer_ID = Western_Customers.Customer_ID  
144 AND age < 20);  
145
```

The results pane shows a table with one column "customer_name" containing 9 rows of data:

customer_name
Alejandro Grove
Stephanie Phelps
Doug Bickford
Gary McGarr
Alyssa Tate
Ben Ferrer
Michelle Moray
Benjamin Patterson
Justin MacKendrick

Total rows: 9 | Query complete 00:00:00.078 | CRLF | Ln 145, Col 1

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);
```

The screenshot shows a SQL query editor interface. The top pane displays a multi-line code editor with numbered lines 145 through 154. Line 147 contains a complex WHERE clause using the EXISTS operator to check if there are any rows in the Western_customers table where Customer_data.Customer_ID matches Western_Customers.Customer_ID and age is 30. The bottom pane shows the results of the query in a table format. The table has one column labeled 'customer_name' with data: Philip Brown, Annie Thurman, and Fred McMath. The status bar at the bottom indicates 'Total rows: 3' and 'Query complete 00:00:00.108'.

```

145
146
147   SELECT Customer_Name
148   FROM Customer_data
149   WHERE EXISTS (SELECT Customer_Name
150   FROM Western_customers
151   WHERE Customer_data.Customer_ID = Western_Customers.Customer_ID
152   AND age =30);
153
154

```

	customer_name
1	Philip Brown
2	Annie Thurman
3	Fred McMath

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 screenshot shows a SQL query editor interface with the following details:

- Title Bar:** Task 36A-Insert and Import Data by Kelechi Immeldah uwa...
- Toolbar:** Includes icons for file operations, search, and navigation.
- Query Tab:** Active tab, showing the executed SQL code.
- Data Output Tab:** Active tab, displaying the results of the query.
- Results:** A table with one column labeled "ship_mode" containing 14 rows of data:

ship_mode
text
Second Class
Standard Class
Standard Class
Second Class
Second Class
Second Class
Standard Class
Standard Class
Standard Class
Standard Class
Standard Class
Standard Class
- Page Footer:** Showing rows: 1 to 849 | Page No: 1 of 1

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);
```

Screenshot of a SQL query editor showing a query and its results.

```

163
164
165
166 SELECT Segment, Customer_name
167 FROM Customer_data
168 WHERE Customer_ID = ANY
169   (SELECT Customer_ID
170    FROM Customer_data
171    WHERE age > 40);
172

```

The results table has two columns: segment and customer_name.

	segment	customer_name
1	Consumer	Claire Gute
2	Consumer	Sean O'Donnell
3	Consumer	Andrew Allen
4	Consumer	Irene Maddox
5	Consumer	Pete Kriz
6	Consumer	Zuschuss Donatelli
7	Corporate	Ken Black
8	Consumer	Sandra Flanagan
9	Consumer	Tracy Blumstein
10	Home Office	Steve Nguyen
11	Corporate	Ruben Ausman
12	Consumer	Patrick O'Donnell
13	Consumer	Lena Hernandez
14	Consumer	Kunst Miller

Total rows: 461 Query complete 00:00:00.071

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);

```

The screenshot shows a SQL query editor interface with the following details:

- Query Text:**

```
163
164
165
166 SELECT Segment, Customer_name
167 FROM Customer_data
168 WHERE Customer_ID = ANY
169 (SELECT Customer_ID
170 FROM Customer_data
171 WHERE age > 40);
```
- Data Output:** A table showing the results of the query. The columns are "segment" and "customer_name". The data consists of 14 rows, each with a row number from 1 to 14.
- Table Data:**

	segment	customer_name
1	Consumer	Claire Gute
2	Consumer	Sean O'Donnell
3	Consumer	Andrew Allen
4	Consumer	Irene Maddox
5	Consumer	Pete Kriz
6	Consumer	Zuschuss Donatelli
7	Corporate	Ken Black
8	Consumer	Sandra Flanagan
9	Consumer	Tracy Blumstein
10	Home Office	Steve Nguyen
11	Corporate	Ruben Ausman
12	Consumer	Patrick O'Donnell
13	Consumer	Lena Hernandez
14	Consumer	Kunst Miller
- Message Bar:** Shows "Total rows: 461" and "Query complete 00:00:00.071".
- Status Bar:** Shows "CRLF | Ln 172, Col 1".

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
```

Task 36A—Insert and Import Data by Kelechi Immeldah uwa...

```

Query History
178   FROM Sales_data
179 WHERE Quantity = 10
180
181
182
183   SELECT * INTO Customer_Backup_2017
184   FROM Customer_data
185
186   SELECT *
187   FROM Customer_backup_2017

```

Data Output Messages Notifications

	customer_id	customer_name	segment	age	country	city	state	postal_code	region
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

Total rows: 793 Query complete 00:00:00.110 CRLF Ln 187, Col 26

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

```

Task 36A—Insert and Import Data by Kelechi Immeldah uwa...

```

187   FROM Customer_backup_2017
188
189
190
191   SELECT Customer_Name, Customer_id INTO Customers_Backup_2017
192   FROM Customer_data
193
194
195   SELECT *
196   FROM Customers_backup_2017

```

Data Output Messages Notifications

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

Successfully run. Total query runtime: 88 msec. 793 rows affected.

Total rows: 793 Query complete 00:00:00.088 CRLF Ln 196, Col 27

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

Task 36A–Insert and Import Data by Kelechi Immeldah uwa... ▾

No limit ▾

Query History

```

28 ---QUESTIONS ONE---
29 ---1. Classify the age of the customers
30 ---2. People between the age of 0 - 20-- classify them as "Young"
31 ---3. People between the age of 21 - 40-- classify them as "Adults"
32 ---4. People between the age of 41 - 60-- classify them as "Older"
33 ---5. People between the ages of 61 and above-- classify them as "Aged"
34
35
36 SELECT *,
37   CASE
38     WHEN age <= 20 THEN 'Young'
39     WHEN age <= 40 THEN 'Adults'
40     WHEN age <= 60 THEN 'Older'
41   ELSE 'Aged'
42 END AS Classification_of_age
43 FROM Customer_data
44

```

Data Output Messages Notifications

Showing rows: 1 to 793 | Page No: 1 of 1 | < << > >>

customer_id	customer_name	segment	age	country	city	state	postal_code	region	classification
520	Claire Gute	Consumer	67	United States	Henderson	Kentucky	42420	South	Aged
445	Darrin Van Huff	Corporate	31	United States	Los Angeles	California	90036	West	Adults
335	Sean O'Donnell	Consumer	65	United States	Fort Lauderdale	Florida	33311	South	Aged
710	Brosina Hoffman	Consumer	20	United States	Los Angeles	California	90032	West	Young
480	Andrew Allen	Consumer	50	United States	Concord	North Carolina	28027	South	Older
170	Irene Maddox	Consumer	66	United States	Seattle	Washington	98103	West	Aged
315	Harold Pawlan	Home Office	20	United States	Fort Worth	Texas	76106	Central	Young
775	Pete Kriz	Consumer	46	United States	Madison	Wisconsin	53711	Central	Older

Total rows: 793 | Query complete 00:00:00.105 | CRLF | Ln 32, Col 66

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

Task 36A–Insert and Import Data by Kelechi Immeldah uwa... ▾

No limit ▾

Query History

```

1 SELECT Customer_id, Quantity * (Sales + Profit)
2 FROM Sales_data
3

```

Data Output Messages Notifications

Showing rows: 1 to 1000 | Page No: 1 of 10 | < << > >>

customer_id	?column?
CG-12520	607.7472
CG-12520	2854.5660000000003
DV-13045	42.9828
SO-20335	2872.7324999999996
SO-20335	49.7688
BH-11710	441.2057999999995
BH-11710	36.9824
BH-11710	5987.2032
BH-11710	72.8595
BH-11710	746.85
BH-11710	16123.4388
BH-11710	3919.1232
AA-10480	62.98560000000005
IM-15070	1621.7045999999998
HP-14815	-275.24
HP-14815	-3.815999999999994
PK-19075	4075.1856
AG-10270	130.98
ZD-21925	22.0848

Total rows: 9994 | Query complete 00:00:00.084 | CRLF | Ln 3, Col 1

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).

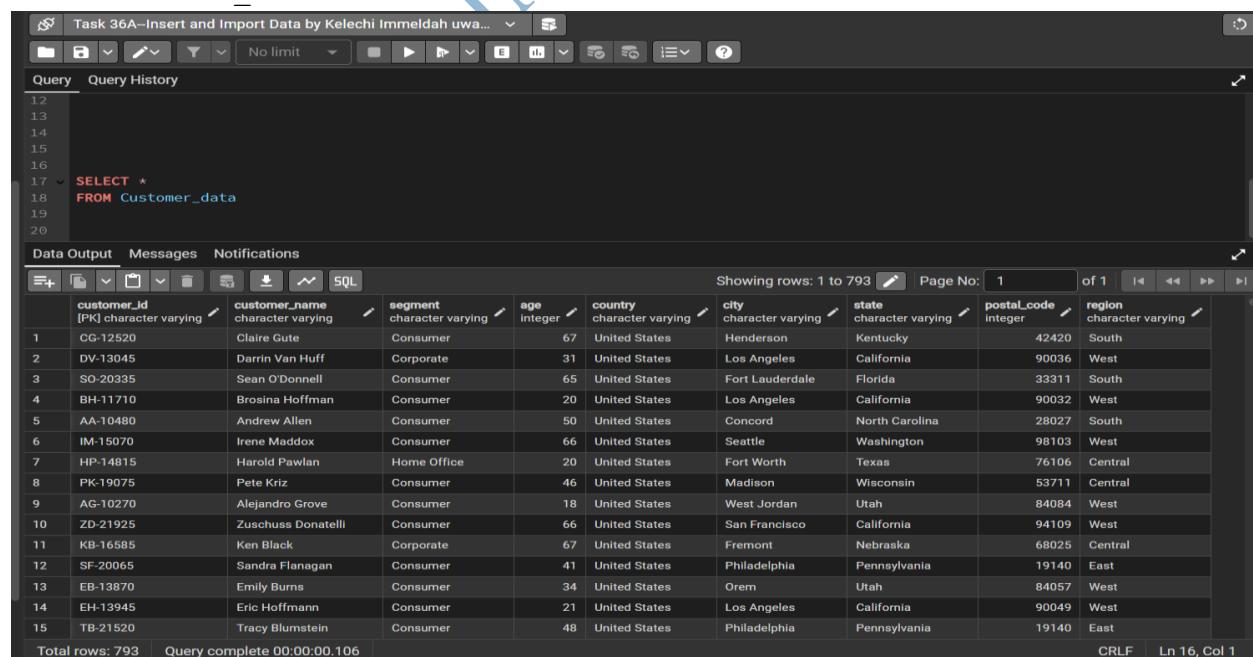
The following example uses a single-line comment as an explanation:

EXAMPLE

-- Select all:

SELECT *

FROM Customer_data



The screenshot shows a SQL query editor interface. The query window contains the following code:

```
12  
13  
14  
15  
16  
17 -- SELECT *  
18 FROM Customer_data  
19  
20
```

The results pane displays a table with 793 rows of data from the Customer_data table. The columns are: customer_id, customer_name, segment, age, country, city, state, postal_code, and region. The data includes various customers like Claire Gute, Darrin Van Huff, Sean O'Donnell, etc., from different cities and states across the United States, with postal codes ranging from 42420 to 91940 and regions South, West, Central, and East.

customer_id	customer_name	segment	age	country	city	state	postal_code	region
CG-12520	Claire Gute	Consumer	67	United States	Henderson	Kentucky	42420	South
DV-13045	Darrin Van Huff	Corporate	31	United States	Los Angeles	California	90036	West
SO-20355	Sean O'Donnell	Consumer	65	United States	Fort Lauderdale	Florida	33311	South
BH-11710	Brosina Hoffman	Consumer	20	United States	Los Angeles	California	90032	West
AA-10480	Andrew Allen	Consumer	50	United States	Concord	North Carolina	28027	South
IM-15070	Irene Maddox	Consumer	66	United States	Seattle	Washington	98103	West
HP-14815	Harold Pawlan	Home Office	20	United States	Fort Worth	Texas	76106	Central
PK-19075	Pete Kriz	Consumer	46	United States	Madison	Wisconsin	53711	Central
AG-10270	Alejandro Grove	Consumer	18	United States	West Jordan	Utah	84084	West
ZD-21925	Zuschuss Donatelli	Consumer	66	United States	San Francisco	California	94109	West
KB-16585	Ken Black	Corporate	67	United States	Fremont	Nebraska	68025	Central
SF-20065	Sandra Flanagan	Consumer	41	United States	Philadelphia	Pennsylvania	19140	East
EB-13870	Emily Burns	Consumer	34	United States	Orem	Utah	84057	West
EH-13945	Eric Hoffmann	Consumer	21	United States	Los Angeles	California	90049	West
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'

```

	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	DV-13045	Darrin Van Huff	Corporate	31	United States	Los Angeles	California	90036	West
2	BH-11710	Brosina Hoffman	Consumer	20	United States	Los Angeles	California	90032	West
3	EH-13945	Eric Hoffmann	Consumer	21	United States	Los Angeles	California	90049	West
4	RA-19885	Ruben Ausman	Corporate	51	United States	Los Angeles	California	90049	West
5	KM-16720	Kunst Miller	Consumer	69	United States	Los Angeles	California	90004	West
6	JS-15685	Jim Sink	Corporate	33	United States	Los Angeles	California	90036	West
7	LS-16975	Lindsay Shagiari	Home Office	66	United States	Los Angeles	California	90004	West
8	TT-21070	Ted Trevino	Consumer	67	United States	Los Angeles	California	90045	West
9	CS-12130	Chad Sievert	Consumer	51	United States	Los Angeles	California	90004	West
10	FM-14290	Frank Merwin	Home Office	45	United States	Los Angeles	California	90032	West
11	JH-15910	Jonathan Howell	Consumer	40	United States	Los Angeles	California	90032	West
12	JO-15280	Jas O'Carroll	Consumer	39	United States	Los Angeles	California	90004	West
13	DB-13615	Doug Bickford	Consumer	19	United States	Los Angeles	California	90045	West
14	PB-19150	Philip Brown	Consumer	30	United States	Los Angeles	California	90004	West
15	MT-18070	Michelle Tran	Home Office	69	United States	Los Angeles	California	90045	West

Total rows: 58 | Query complete 00:00:00.079 | CRLF | Ln 26, Col 1

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
\diamond	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

UWANAKA IMM