SQL commands:

1. BASIC DDL commands:

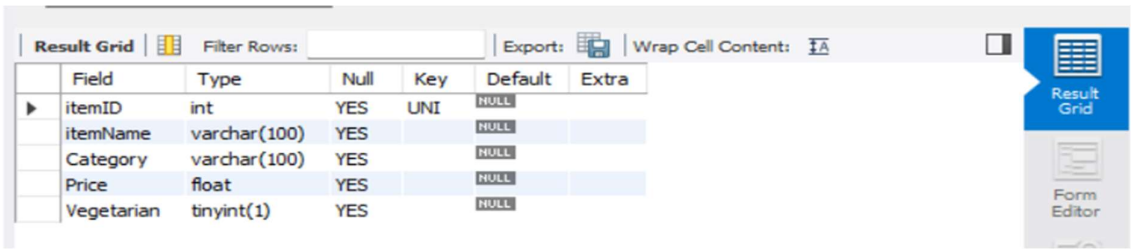
Data Definition Language (DDL) is a subset of SQL commands used to define and modify database schema. Here are the key DDL commands

1. CREATE COMMAND:

Table: MenuItems

ItemID	ItemName	Category	Price	Vegetarian
1	Food Item 1	Beverage	6.35	False
2	Food Item 2	Entree	13.87	True
3	Food Item 3	Beverage	3.92	True
4	Food Item 4	Beverage	12.53	False
5	Food Item 5	Dessert	2.65	True
...	...	...	...	...

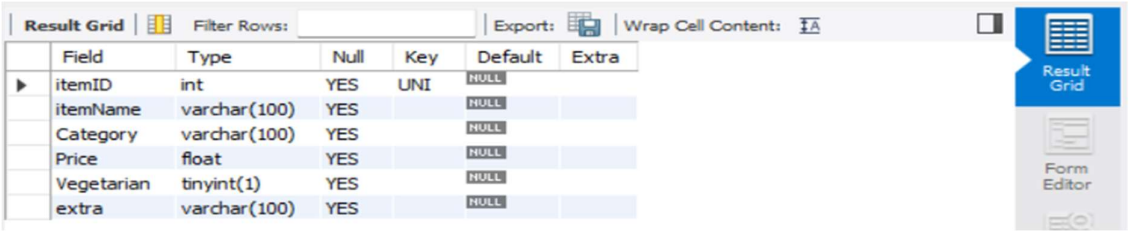
```
CREATE DATABASE practice ;
USE practice ;
CREATE TABLE table1 (itemID int UNIQUE, itemName varchar(100), Category varchar(100), Price
float, Vegetarian bool) ;
DESC table1;
```



ALTER COMMAND:

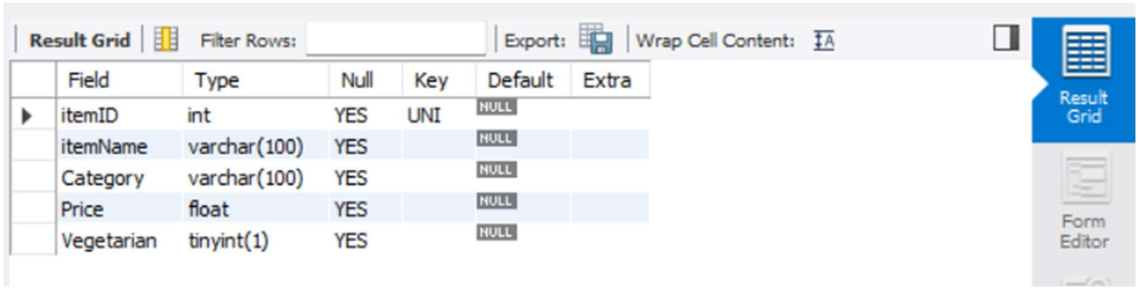
i. ADD COLUMN:

```
ALTER TABLE table1 ADD COLUMN extra varchar(100);
DESC table1;
```



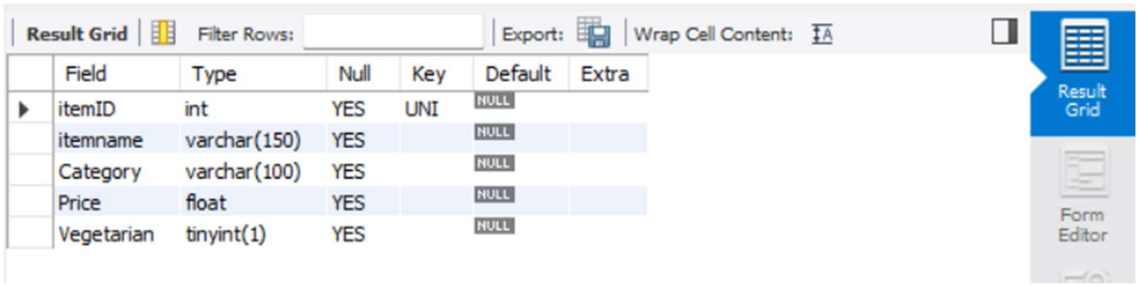
ii. DROP COLUMN:

```
ALTER TABLE table1 DROP COLUMN extra ;
DESC table1;
```



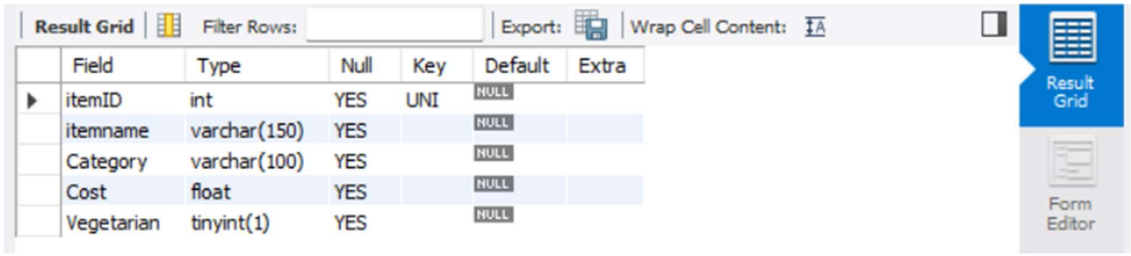
iii. MODIFY COLUMN: (DATATYPE)

```
ALTER TABLE table1 MODIFY COLUMN itemname varchar(150) ;
DESC table1;
```



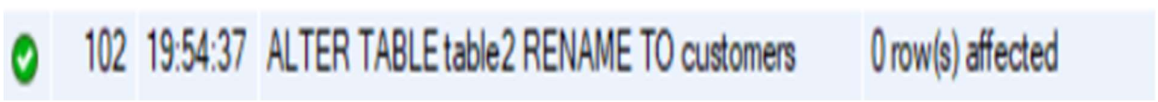
iv. RENAME COLUMN:

```
ALTER TABLE table1 CHANGE Price Cost float ;
# ALTER TABLE table1 CHANGE old_col_name new_col_name new_data_type
DESC table1 ;
```



v. CHANGE TABLE NAME:

```
ALTER TABLE table2 RENAME TO customers ;
```



DROP COMMAND:

```
DROP TABLE table1
DROP DATABASE practice1
```

✓	17	12:31:54	DROP TABLE table1	0 row(s) affected
✓	18	12:32:03	DROP DATABASE practice1	0 row(s) affected

TRUNCATE :

✓	29	12:41:18	INSERT INTO table1 (itemID,itemName,Category,...	1 row(s) affected
✓	30	12:42:06	SELECT * FROM table1 LIMIT 0, 1000	2 row(s) returned

BASIC DML COMMANDS:

1. INSERT:

```
INSERT INTO table1 VALUES (1, 'Food Item1', 'Beverage' , 6.35 , False);
INSERT INTO table1 (itemID,itemName,Category,Cost,Vegetarian) VALUES (2, 'Food Item2', 'Entree'
, 13.84 , TRUE) ;
```

2. SELECT:

```
SELECT * FROM table1 ;
SELECT * FROM table1 WHERE itemname = 'Food Item1' ;
SELECT itemID, Category FROM table1 ;
SELECT itemID, Category FROM table1 WHERE itemname = 'Food Item2' ;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Cont

itemID	itemname	Category	Cost	Vegetarian
1	Food Item1	Beverage	6.35	0
2	Food Item2	Entree	13.84	1

Result Grid | Filter Rows: | Export: | Wrap Cell Cont

itemID	itemname	Category	Cost	Vegetarian
1	Food Item1	Beverage	6.35	0

Result Grid | Filter Rows: | Export: | Wrap Cell Cont

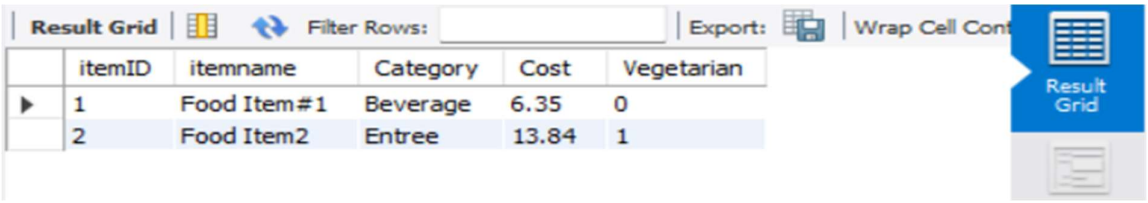
itemID	Category
1	Beverage
2	Entree

Result Grid | Filter Rows: | Export: | Wrap Cell Cont

itemID	Category
2	Entree

3. UPDATE:

```
UPDATE table1 SET itemname = 'Food Item#1' WHERE  itemname = 'Food Item1'  ;
SELECT * FROM table1
```



itemID	itemname	Category	Cost	Vegetarian
1	Food Item#1	Beverage	6.35	0
2	Food Item2	Entree	13.84	1

4. DELETE COMMAND

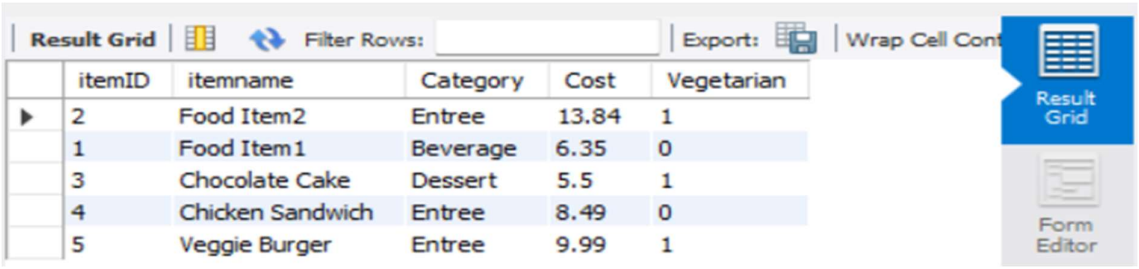
```
DELETE FROM table1 WHERE itemname = 'Food Item#1';
SELECT * FROM table1 ;
```

itemID	itemname	Category	Cost	Vegetarian
2	Food Item2	Entree	13.84	1

VIEWS:

1. CREATE VIEW:

```
CREATE VIEW v1 AS
SELECT * FROM table1 ;
```



Result Grid | Filter Rows: | Export: | Wrap Cell Cont

itemID	itemname	Category	Cost	Vegetarian
2	Food Item2	Entree	13.84	1
1	Food Item1	Beverage	6.35	0
3	Chocolate Cake	Dessert	5.5	1
4	Chicken Sandwich	Entree	8.49	0
5	Veggie Burger	Entree	9.99	1

Result Grid | Form Editor

```
CREATE VIEW v2 AS
SELECT itemName, Category FROM table1 ;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Cont

itemName	Category
Food Item2	Entree
Food Item1	Beverage
Chocolate Cake	Dessert
Chicken Sandwich	Entree
Veggie Burger	Entree

Result Grid | Form Editor

```
CREATE VIEW v3 AS
SELECT itemName, Category FROM table1 WHERE itemname = 'Veggie Burger';
```

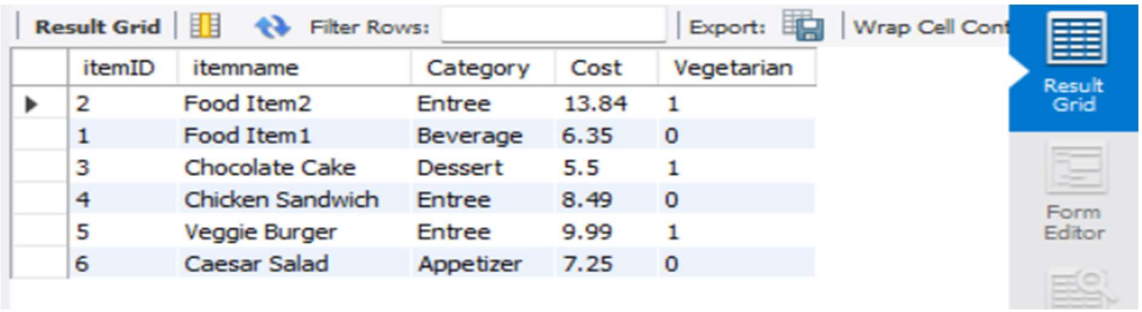
Result Grid | Filter Rows: | Export: | Wrap Cell Cont

itemName	Category
Veggie Burger	Entree

Result Grid

UPDATING TABLE

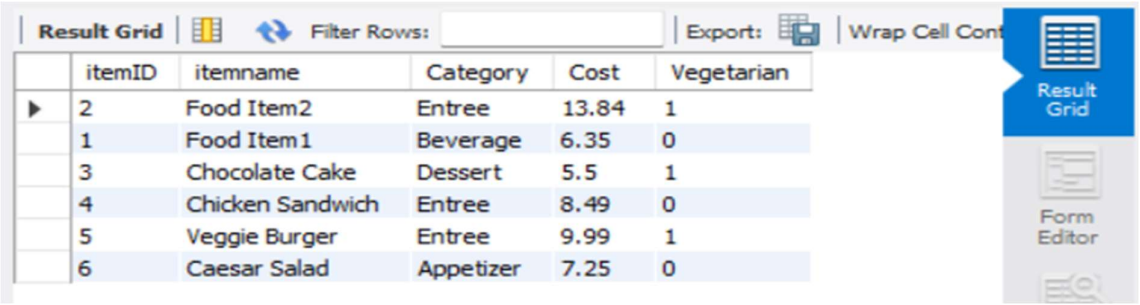
```
INSERT INTO table1 VALUES (6, 'Caesar Salad', 'Appetizer', 7.25, FALSE);
SELECT * FROM table1 ;
SELECT * FROM v1 ;
```



Result Grid | Filter Rows: | Export: | Wrap Cell Cont

itemID	itemname	Category	Cost	Vegetarian
2	Food Item2	Entree	13.84	1
1	Food Item1	Beverage	6.35	0
3	Chocolate Cake	Dessert	5.5	1
4	Chicken Sandwich	Entree	8.49	0
5	Veggie Burger	Entree	9.99	1
6	Caesar Salad	Appetizer	7.25	0

Result Grid | Form Editor



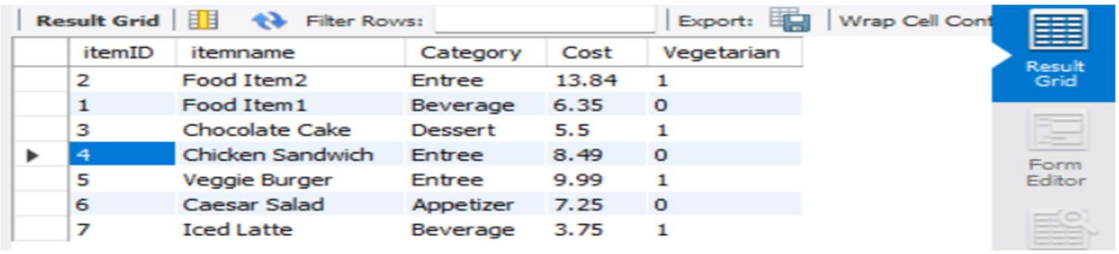
Result Grid | Filter Rows: | Export: | Wrap Cell Cont

itemID	itemname	Category	Cost	Vegetarian
2	Food Item2	Entree	13.84	1
1	Food Item1	Beverage	6.35	0
3	Chocolate Cake	Dessert	5.5	1
4	Chicken Sandwich	Entree	8.49	0
5	Veggie Burger	Entree	9.99	1
6	Caesar Salad	Appetizer	7.25	0

Result Grid | Form Editor

UPDATING VIEW:

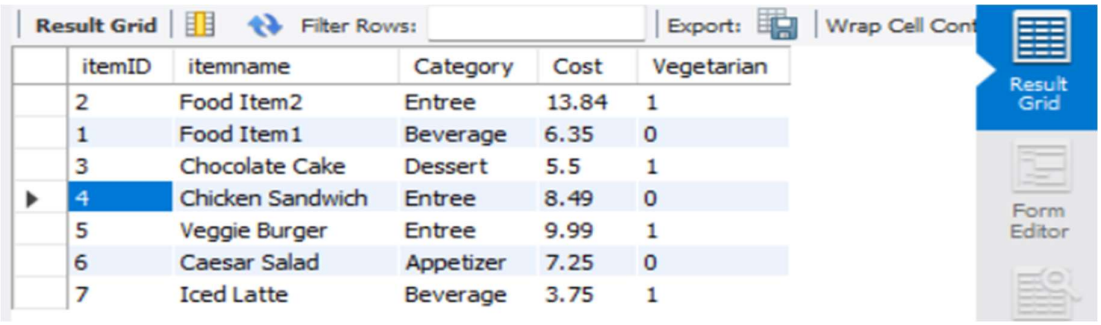
```
INSERT INTO v1 VALUES (7, 'Iced Latte', 'Beverage', 3.75, TRUE);
SELECT * FROM table1 ;
SELECT * FROM v1 ;
```



Result Grid | Filter Rows: | Export: | Wrap Cell Cont

itemID	itemname	Category	Cost	Vegetarian
2	Food Item2	Entree	13.84	1
1	Food Item1	Beverage	6.35	0
3	Chocolate Cake	Dessert	5.5	1
4	Chicken Sandwich	Entree	8.49	0
5	Veggie Burger	Entree	9.99	1
6	Caesar Salad	Appetizer	7.25	0
7	Iced Latte	Beverage	3.75	1

Result Grid | Form Editor



Result Grid | Filter Rows: | Export: | Wrap Cell Cont

itemID	itemname	Category	Cost	Vegetarian
2	Food Item2	Entree	13.84	1
1	Food Item1	Beverage	6.35	0
3	Chocolate Cake	Dessert	5.5	1
4	Chicken Sandwich	Entree	8.49	0
5	Veggie Burger	Entree	9.99	1
6	Caesar Salad	Appetizer	7.25	0
7	Iced Latte	Beverage	3.75	1

Result Grid | Form Editor

INSERTING INTO EITHER TABLE OR VIEW UPDATES BOTH

INSERTING INTO PARTIAL COLUMNS:

```
INSERT INTO v2 VALUES('Roll' , 'Appetizer') ;
SELECT * FROM v2 ;
SELECT * FROM table1 ;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Cont

itemName	Category
Food Item2	Entree
Food Item1	Beverage
Chocolate Cake	Dessert
Chicken Sandwich	Entree
Veggie Burger	Entree
Caesar Salad	Appetizer
Iced Latte	Beverage
Roll	Appetizer

Result Grid | Filter Rows: | Export: | Wrap Cell Cont

itemID	itemname	Category	Cost	Vegetarian
2	Food Item2	Entree	13.84	1
1	Food Item1	Beverage	6.35	0
3	Chocolate Cake	Dessert	5.5	1
4	Chicken Sandwich	Entree	8.49	0
5	Veggie Burger	Entree	9.99	1
6	Caesar Salad	Appetizer	7.25	0
7	Iced Latte	Beverage	3.75	1
NULL	Roll	Appetizer	NULL	NULL

AGAIN INSERTING INTO EITHER TABLE OR VIEW UPDATES BOTH

UPDATE:

```
UPDATE table1 SET itemname = 'Veg Roll' WHERE itemname = 'Roll';
SELECT * FROM table1;
SELECT * FROM v2 ;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Cont

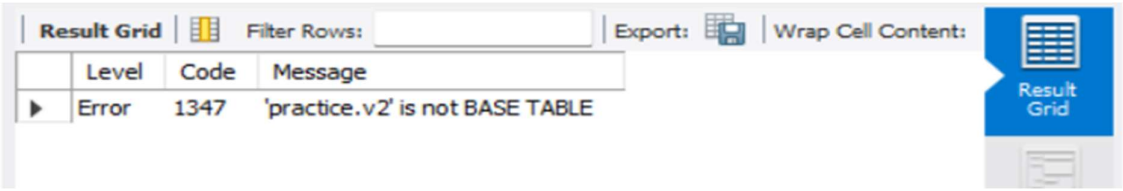
itemID	itemname	Category	Cost	Vegetarian
2	Food Item2	Entree	13.84	1
1	Food Item1	Beverage	6.35	0
3	Chocolate Cake	Dessert	5.5	1
4	Chicken Sandwich	Entree	8.49	0
5	Veggie Burger	Entree	9.99	1
6	Caesar Salad	Appetizer	7.25	0
7	Iced Latte	Beverage	3.75	1
NULL	Veg Roll	Appetizer	NULL	NULL

Result Grid | Filter Rows: | Export: | Wrap Cell Cont

itemName	Category
Food Item2	Entree
Food Item1	Beverage
Chocolate Cake	Dessert
Chicken Sandwich	Entree
Veggie Burger	Entree
Caesar Salad	Appetizer
Iced Latte	Beverage
Veg Roll	Appetizer

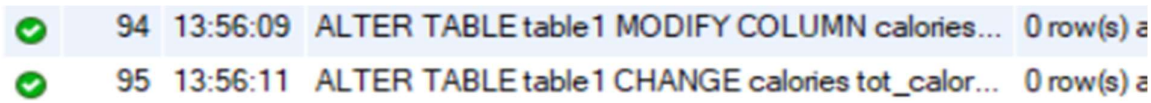
ALTERING VIEW IS NOT ALLOWED:

```
ALTER TABLE v1 ADD COLUMN calories int;
ALTER TABLE v1 MODIFY COLUMN calories int;
ALTER TABLE v2 CHANGE calories tot_calories int ;
ALTER TABLE v1 DROP COLUMN calories ;
SHOW ERRORS ;
```



ALTERING BASE TABLE WILL ALTER THE VIEW AS WELL:

```
ALTER TABLE table1 ADD COLUMN calories int;
ALTER TABLE table1 MODIFY COLUMN calories int;
ALTER TABLE table1 CHANGE calories tot_calories int ;
ALTER TABLE table1 DROP COLUMN tot_calories ;
```



DATABASE CONSTRAINTS:

Table 1: Customers

CustomerID	Name	Email
1	John Doe	john.doe@example.com
2	Jane Smith	jane.smith@example.com
3	Alice Johnson	alice.johnson@example.com

Table 2: Orders

OrderID	CustomerID	OrderDate	TotalAmount
101	1	2023-01-15	150.0
102	2	2023-02-20	200.0
103	1	2023-03-05	250.0

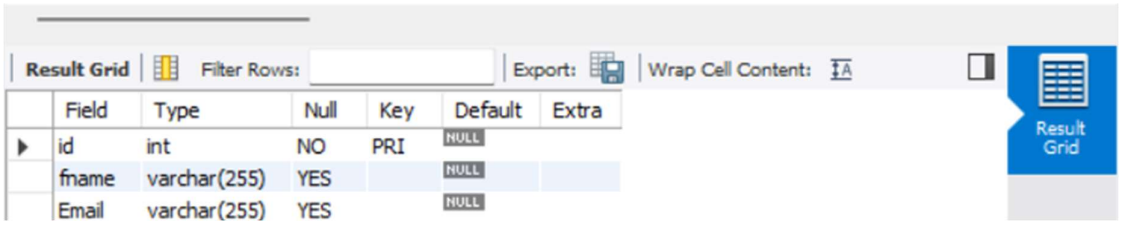
```
CREATE TABLE Customer(
id int , fname varchar(255), Email varchar(255),
PRIMARY KEY (id)) ;
```

Or

```
CREATE TABLE Customer(
id int PRIMARY KEY, fname varchar(255), Email varchar(255)) ;
```

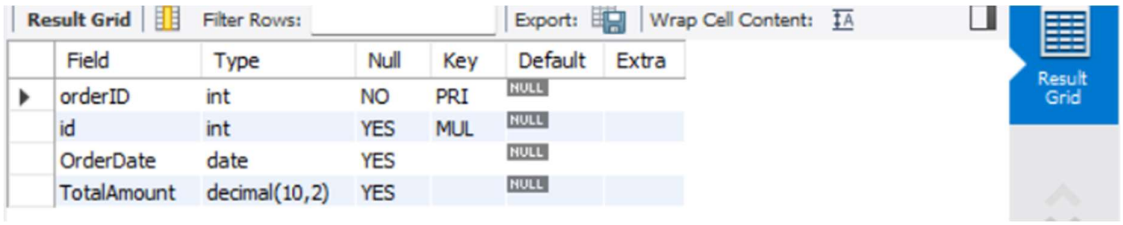
Or

```
CREATE TABLE Customer1(
id int, fname varchar(255), Email varchar(255)) ;
ALTER TABLE Customer1 ADD PRIMARY KEY (ID) ;
```



CREATE CHILD TABLE:

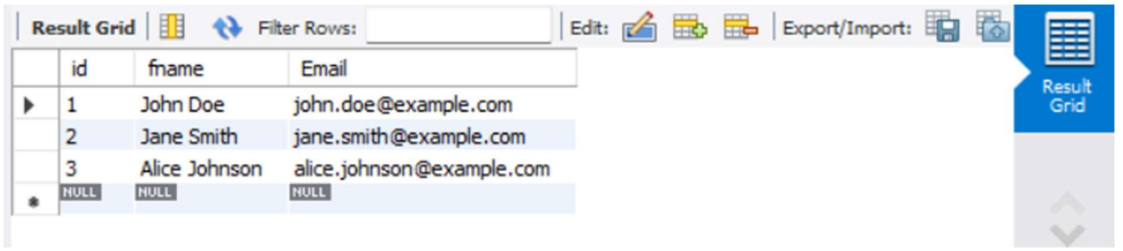
```
CREATE TABLE Orders(orderID int PRIMARY KEY,
 id int , OrderDate date ,
 TotalAmount decimal(10, 2),
 FOREIGN KEY (id) REFERENCES Customer(id)) ;
 DESC Orders ;
```



Result Grid | Filter Rows: | Export: | Wrap Cell Content:

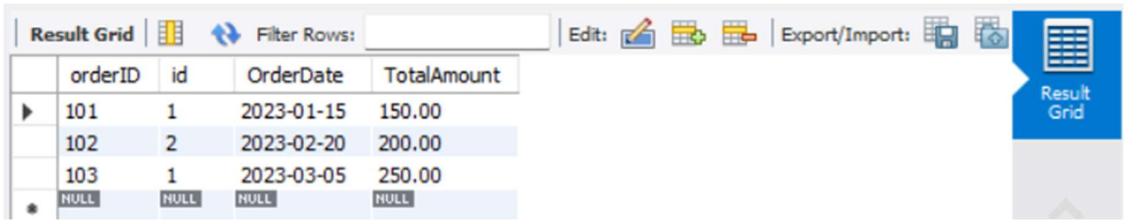
Field	Type	Null	Key	Default	Extra
orderID	int	NO	PRI	NULL	
id	int	YES	MUL	NULL	
OrderDate	date	YES		NULL	
TotalAmount	decimal(10,2)	YES		NULL	

```
INSERT INTO Customer  VALUES
(1, 'John Doe', 'john.doe@example.com'),
(2, 'Jane Smith', 'jane.smith@example.com'),
(3, 'Alice Johnson', 'alice.johnson@example.com');
INSERT INTO Orders VALUES
(101, 1, '2023-01-15', 150.00),
(102, 2, '2023-02-20', 200.00),
(103, 1, '2023-03-05', 250.00);
```



Result Grid | Filter Rows: | Edit: | Export/Import:

id	fname	Email
1	John Doe	john.doe@example.com
2	Jane Smith	jane.smith@example.com
3	Alice Johnson	alice.johnson@example.com
NULL	NULL	NULL



Result Grid | Filter Rows: | Edit: | Export/Import:

orderID	id	OrderDate	TotalAmount
101	1	2023-01-15	150.00
102	2	2023-02-20	200.00
103	1	2023-03-05	250.00
NULL	NULL	NULL	NULL

INTEGRITY CONTRAINTS THAT CAN BE VIOLATED:

1. Primary Key constraint:

Adding duplicate:

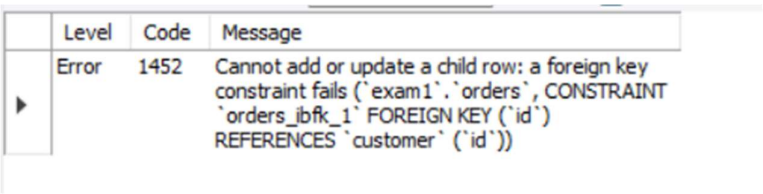
```
INSERT INTO Customer  VALUES
(1, 'John Doe', 'john.doe@example.com');
```

	Level	Code	Message
▶	Error	1062	Duplicate entry '1' for key 'customer.PRIMARY'

2. Foreign Key Constraint:

```
INSERT INTO Orders VALUES
(111, 5, '2023-01-15', 150.00) ;
```

Id=5 doesn't exist in the parent table

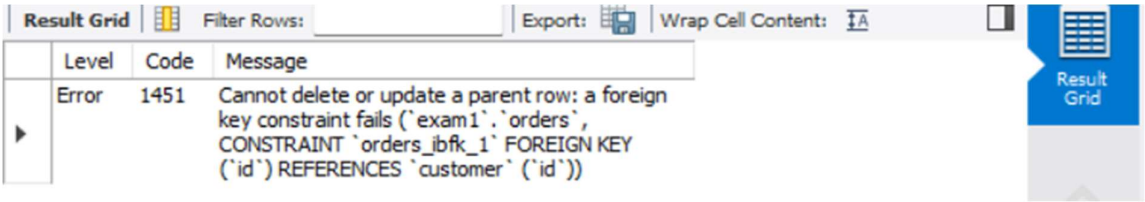


	Level	Code	Message
▶	Error	1452	Cannot add or update a child row: a foreign key constraint fails (`exam1`.`orders`, CONSTRAINT `orders_ibfk_1` FOREIGN KEY (`id`) REFERENCES `customer` (`id`))

3. Referential Integrity constraint:

Can't delete from primary table while it is being referenced.

```
DELETE FROM Customer WHERE id = 1 ;
```



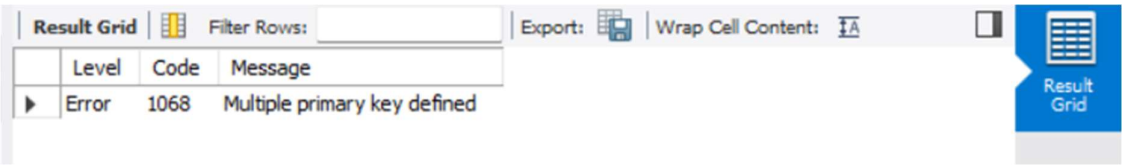
	Level	Code	Message
▶	Error	1451	Cannot delete or update a parent row: a foreign key constraint fails (`exam1`.`orders`, CONSTRAINT `orders_ibfk_1` FOREIGN KEY (`id`) REFERENCES `customer` (`id`))

REMOVE PRIMARY KEY:

```
ALTER TABLE Customer1 DROP PRIMARY KEY ;
```

MULTIPLE PRIMARY KEYS NOT ALLOWED

```
ALTER TABLE Customer ADD PRIMARY KEY(Name) ;
```

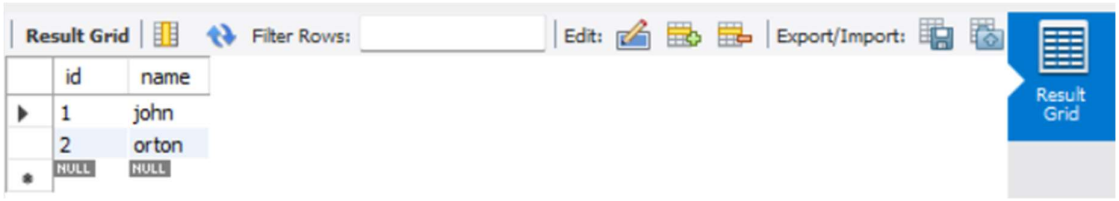


	Level	Code	Message
▶	Error	1068	Multiple primary key defined

AUTO INCREMENT:

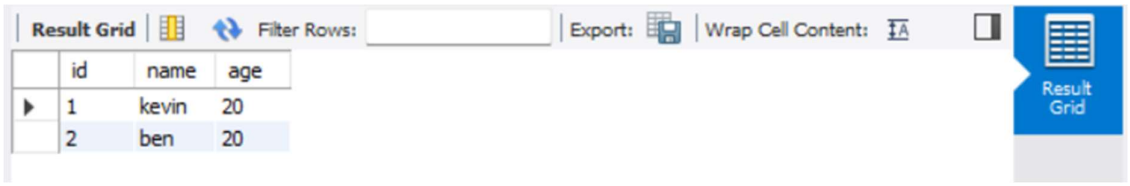
```
CREATE TABLE student (
    id int AUTO_INCREMENT PRIMARY KEY,
    name varchar(255)
);
INSERT INTO student (name) values ('john'), ('orton') ;
SELECT * FROM student ;
```

Not being primary key is not allowed



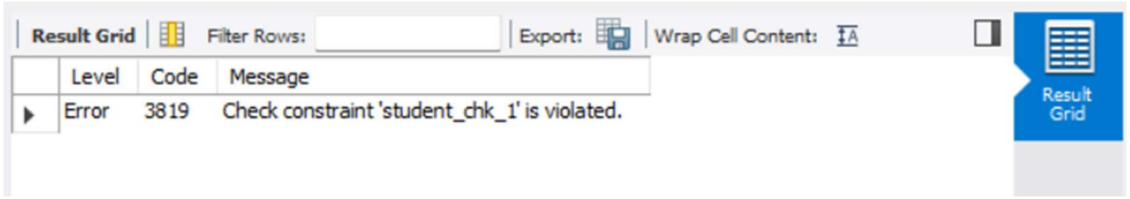
DEFAULT:

```
CREATE TABLE student (
    id int,
    name varchar(255),
    age int default 20
);
INSERT INTO student (id, name) VALUES (1, 'kevin') , (2 , 'ben');
SELECT * FROM STUDENT ;
```



CHECK:

```
CREATE TABLE student (
    id int,
    name varchar(255),
    age int CHECK (AGE >20)
);
INSERT INTO student (id, name, age) VALUES (1, 'kevin',19);
SHOW ERRORS ;
```



UNIQUE:

```
CREATE TABLE student (
    id int UNIQUE,
    name varchar(255)
);
INSERT INTO student (id, name) VALUES (1, 'kevin');
INSERT INTO student (id, name) VALUES (1, 'ben');
SHOW ERRORS ;
```

JOINS:

Table 1: Departments

DepartmentID	DepartmentName
1	Human Resources
2	Finance
3	IT
4	Marketing

Table 2: Employees

EmployeeID	Name	DepartmentID	Position	Salary
1	John Doe	3	Software Developer	70000
2	Jane Smith	2	Financial Analyst	65000
3	Alice Johnson	1	HR Manager	75000
4	Bob Brown	4	Marketing Coordinator	60000

CARTESIAN JOIN:

Random combination of pairs of rows, not that useful actually

```
SELECT * FROM Departments, Employees ;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

DepartmentID	DepartmentName	EmployeeID	Name	DepartmentID	Position	Salary
4	Marketing	1	John Doe	3	Software Developer	70000.00
3	IT	1	John Doe	3	Software Developer	70000.00
2	Finance	1	John Doe	3	Software Developer	70000.00
1	Human Resources	1	John Doe	3	Software Developer	70000.00
4	Marketing	2	Jane Smith	2	Financial Analyst	65000.00
3	IT	2	Jane Smith	2	Financial Analyst	65000.00
2	Finance	2	Jane Smith	2	Financial Analyst	65000.00
1	Human Resources	2	Jane Smith	2	Financial Analyst	65000.00
4	Marketing	3	Alice Johnson	1	HR Manager	75000.00
3	IT	3	Alice Johnson	1	HR Manager	75000.00
2	Finance	3	Alice Johnson	1	HR Manager	75000.00

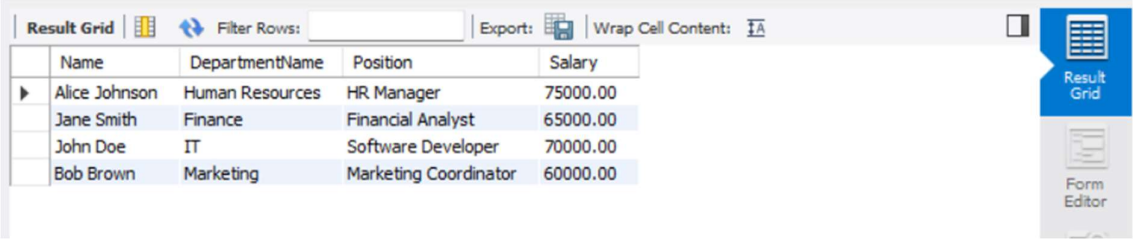
Result Grid
Form Editor
Field Types

INNER JOIN:

```
SELECT Name,DepartmentName ,Employees.Position, Employees.Salary FROM Departments INNER JOIN
Employees ON Departments.DepartmentID = Employees.DepartmentID ;
```

primary function is to combine rows from two or more tables based on a related column between them, An inner join requires a common column (or columns) in both tables.

Within the query the column names from first columns be written directly but the col names from 2nd columns need to be written as table_name.column_name

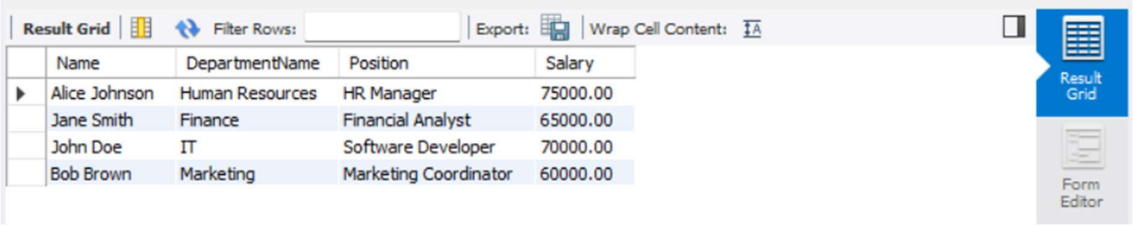


Name	DepartmentName	Position	Salary
Alice Johnson	Human Resources	HR Manager	75000.00
Jane Smith	Finance	Financial Analyst	65000.00
John Doe	IT	Software Developer	70000.00
Bob Brown	Marketing	Marketing Coordinator	60000.00

LEFT JOIN

In a left join, one table is considered the primary (or left) table. All rows from this table will appear in the result set, The other table (or tables) in the join is considered the secondary (or right) table. Only common from this table will come in the result

```
SELECT Name,DepartmentName ,Employees.Position, Employees.Salary FROM Departments LEFT JOIN
Employees ON Departments.DepartmentID = Employees.DepartmentID ;
```



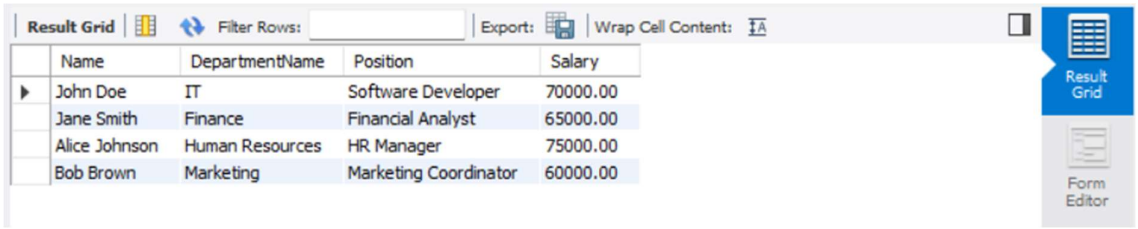
Name	DepartmentName	Position	Salary
Alice Johnson	Human Resources	HR Manager	75000.00
Jane Smith	Finance	Financial Analyst	65000.00
John Doe	IT	Software Developer	70000.00
Bob Brown	Marketing	Marketing Coordinator	60000.00

The table chosen here has all the entries of primary key in the child column hence the results for inner, left and right join would be the same

RIGHT JOIN

In a right join, one table is considered the primary (or right) table. All rows from this table will appear in the result set, The other table (or tables) in the join is considered the secondary (or left) table. Only common from this table will come in the result

```
SELECT Name,DepartmentName ,Employees.Position, Employees.Salary FROM Departments RIGHT JOIN
Employees ON Departments.DepartmentID = Employees.DepartmentID ;
```



Name	DepartmentName	Position	Salary
John Doe	IT	Software Developer	70000.00
Jane Smith	Finance	Financial Analyst	65000.00
Alice Johnson	Human Resources	HR Manager	75000.00
Bob Brown	Marketing	Marketing Coordinator	60000.00

UNION:

The `UNION` command in SQL is used to combine the result sets of two or more `SELECT` statements, blindly combines the columns that are selected, if some common columns are there all entries will be shown under one, and even if not columns are same the output is still the mismatch of the combination of the select queries. NUMBER OF COLUMNS RETURNING IN THE SELECT QUERY BE THE SAME.

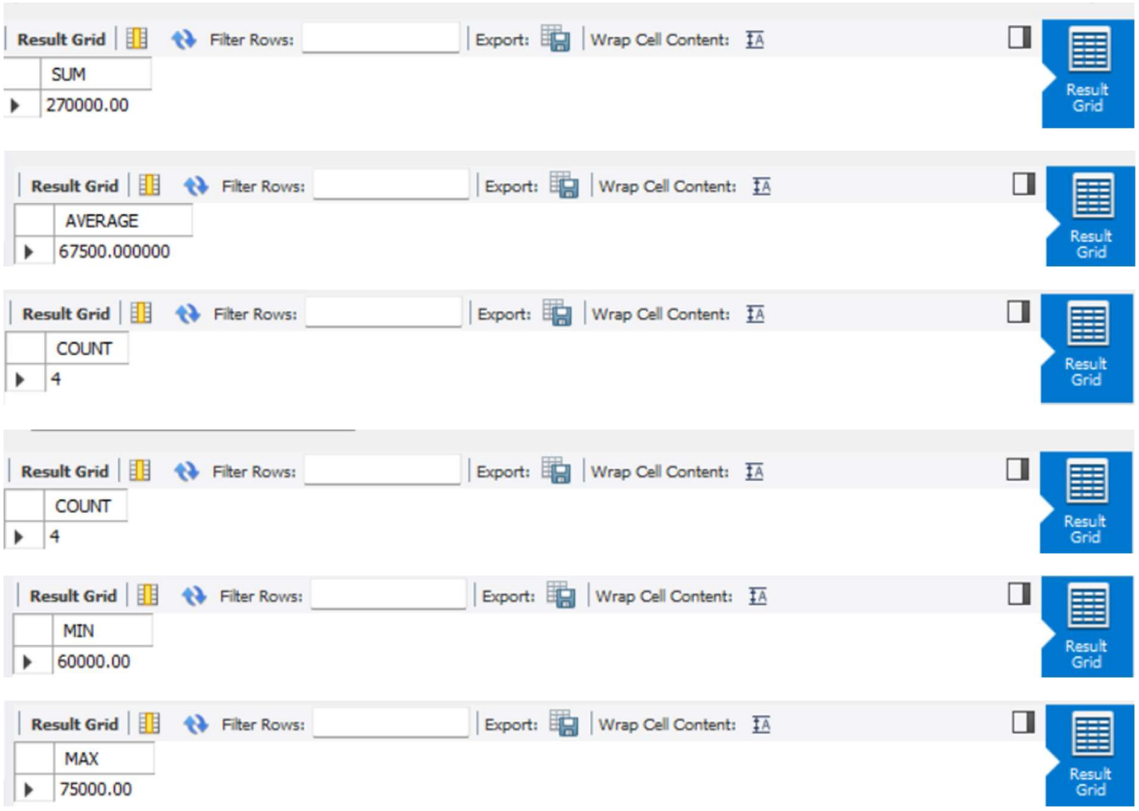
```
SELECT DepartmentName FROM Departments UNION SELECT Salary FROM Employees ;
```

DepartmentName
Finance
IT
Marketing
70000.00
65000.00
75000.00

UNION ALL SHOWS ALL THE ENTRIES WITH DUPLICATES

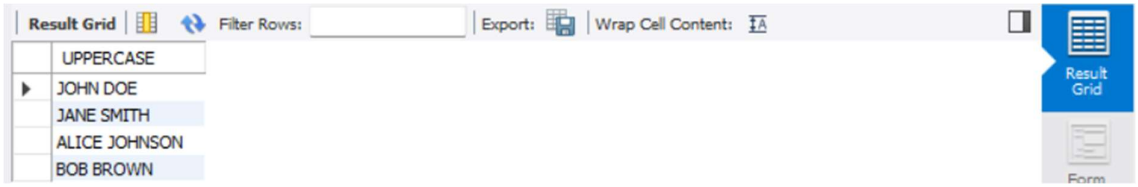
AGGREGATE COMMANDS:

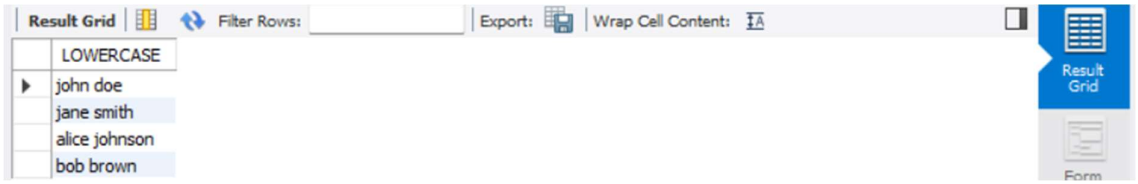
```
SELECT sum(salary) as SUM FROM EMPLOYEES;
SELECT avg(salary) as AVERAGE FROM EMPLOYEES;
SELECT COUNT(salary) as COUNT FROM EMPLOYEES;
SELECT COUNT(*) as COUNT FROM EMPLOYEES;
SELECT min(salary) as MIN FROM EMPLOYEES;
SELECT max(salary) as MAX FROM EMPLOYEES;
```



STRING COMMANDS:

```
SELECT UPPER(name) as UPPERCASE FROM EMPLOYEES ;
SELECT LOWER(name) as UPPERCASE FROM EMPLOYEES ;
```





GROUP BY , ORDER BY and HAVING:

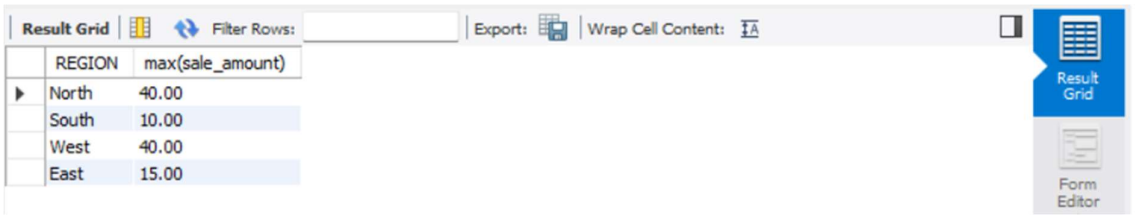
sale_id	product_name	quantity	sale_amount	sale_date	region
1	Widget	3	30.0	2023-01-01	North
2	Widget	1	10.0	2023-01-03	South
3	Gadget	2	40.0	2023-01-02	West
4	Widget	4	40.0	2023-01-01	North
5	Thingamajig	1	15.0	2023-01-04	East

GROUP BY:

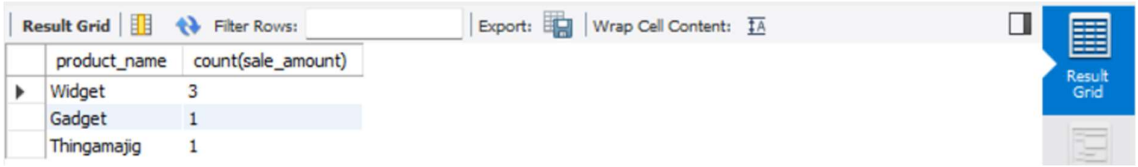
Always an aggregate function is mandatory for group by clause

```
SELECT REGION , max(sale_amount)  FROM sales_data GROUP BY REGION ;
```

The quantity that is being grouped must be included while selecting




```
SELECT product_name , count(sale_amount)  FROM sales_data GROUP BY product_name ;
```




```
SELECT product_name , max(sale_amount)  FROM sales_data GROUP BY product_name ;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

product_name	max(sale_amount)
Widget	40.00
Gadget	40.00
Thingamajig	15.00

HAVING:

Use an alias to make condition

```
SELECT REGION , max(sale_amount) as amt FROM sales_data GROUP BY REGION HAVING amt > 10;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

REGION	amt
North	40.00
West	40.00
East	15.00

```
SELECT product_name , count(sale_amount) as count FROM sales_data GROUP BY product_name having
count>1;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

product_name	count
Widget	3

ORDER BY:

```
SELECT REGION , max(sale_amount) as amt FROM sales_data GROUP BY REGION ORDER BY amt;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

REGION	amt
South	10.00
East	15.00
North	40.00
West	40.00

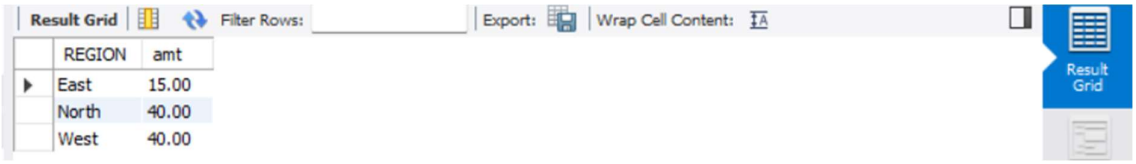
```
SELECT REGION , max(sale_amount) as amt FROM sales_data GROUP BY REGION ORDER BY amt DESC;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

REGION	amt
North	40.00
West	40.00
East	15.00
South	10.00

GROUP BY HAVING ORDER BY:

```
SELECT REGION , max(sale_amount) as amt FROM sales_data GROUP BY REGION HAVING amt>10 ORDER BY
amt;
```



NESTED QUERIES:

Customers Table

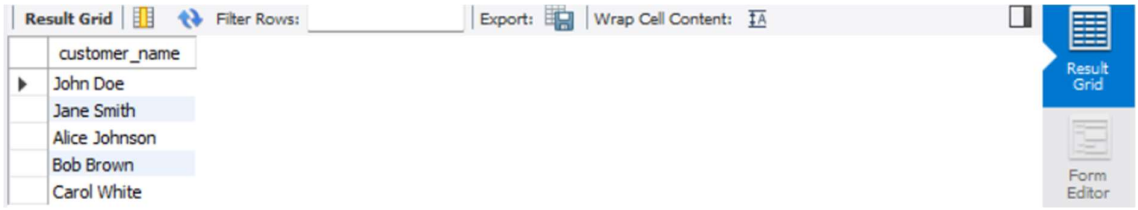
customer_id	customer_name	customer_email
1	John Doe	john.doe@example.com
2	Jane Smith	jane.smith@example.com
3	Alice Johnson	alice.johnson@example.com
4	Bob Brown	bob.brown@example.com
5	Carol White	carol.white@example.com
6	David Green	david.green@example.com
7	Eva Black	eva.black@example.com
8	Frank Gray	frank.gray@example.com
9	Grace Hall	grace.hall@example.com
10	Henry Adams	henry.adams@example.com

Orders Table

order_id	customer_id	order_date	order_amount
101	1	2023-01-01	100.0
102	2	2023-01-02	150.0
103	3	2023-01-03	200.0
104	4	2023-01-04	250.0
105	5	2023-01-05	300.0
106	1	2023-01-06	350.0
107	2	2023-01-07	400.0
108	3	2023-01-08	450.0
109	4	2023-01-09	500.0
110	5	2023-01-10	550.0

Q1. Find the names of all customers who have made orders above $300.

```
SELECT customer_name FROM Customers WHERE customer_id IN
 (SELECT customer_id from Orders WHERE order_amount >300);
```



Q2. **Display the highest order amount for each customer along with their name.**

```
SELECT customer_name,
       (SELECT MAX(order_amount) FROM orders WHERE orders.customer_id = customers.customer_id)
AS highest_order
FROM customers;
```