

CREATE A VIRTUAL TABLE TO SUMMARIZE DATA:

Creating a virtual table called “OrdersView” that focuses on “OrderID”, Quantity and Cost columns within the Orders table for all orders with a quantity greater than 2.

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

```
1 use littlelemondb;
2 CREATE VIEW OrdersView AS
3 SELECT orderID, Quantity, totalCost AS Cost
4 FROM Orders
5 WHERE Quantity > 2;
6 select * from ordersView;
```

The result grid shows the output of the query:

orderID	Quantity	Cost
3	7	61.32
5	9	28.00

The bottom pane shows the execution log with the following messages:

#	Time	Action	Message	Duration / Fetch
34	11:17:53	use littlelemondb	0 row(s) affected	0.016 sec
35	11:17:55	select * from orders LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec
36	11:18:58	CREATE VIEW OrdersView AS SELECT orderID, Quantity, totalCost AS Cost FROM Orders WHERE Quantity > 2	Error Code: 1050. Table 'OrdersView' already exists	0.000 sec
37	11:19:20	drop view ordersView	0 row(s) affected	0.000 sec
38	11:19:23	CREATE VIEW OrdersView AS SELECT orderID, Quantity, totalCost AS Cost FROM Orders WHERE Quantity > 2	0 row(s) affected	0.016 sec
39	11:19:43	select * from ordersView LIMIT 0, 1000	2 row(s) returned	0.000 sec / 0.000 sec

Summary from all four tables on all customers with orders that cost more than \$150.

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

```
1 use littlelemondb;
2 SELECT c.customerID, c.fullName, o.orderID, o.totalCost AS Cost, m.menuName, m_i.courseName, m_i.starterName
3 FROM Orders o
4 JOIN Customers c
5 ON o.customerID = c.customerID
6 JOIN Menus m
7 ON o.menuID = m.menuID
8 JOIN MenuItem m_i
9 ON m.menuItemsID = m_i.menuItemsID
10 WHERE o.totalCost > 150
11 ORDER BY o.totalCost ASC;
```

The result grid shows the output of the query:

customerID	fullName	orderID	Cost	menuName	courseName	starterName
2	mikal jane	2	250.00	finnish fries	rice	mexican fish Soup
4	james carter	4	320.00	potuguese fries	pizza	italian onion

The bottom pane shows the execution log with the following messages:

#	Time	Action	Message	Duration / Fetch
97	13:43:18	delete from OrderCancellations where id = 2	1 row(s) affected	0.000 sec
98	13:44:10	drop procedure CancelOrder	0 row(s) affected	0.016 sec
99	13:44:20	CREATE PROCEDURE CancelOrder(IN id INT) BEGIN DELETE FROM Orders WHERE orderID = id; END	0 row(s) affected	0.000 sec
100	13:44:22	Call CancelOrder(5)	1 row(s) returned	0.016 sec / 0.000 sec
101	14:02:40	SELECT c.customerID, c.fullName, o.orderID, o.totalCost AS Cost, m.menuName, m_i.courseName, m_i.starterName	2 row(s) returned	0.015 sec / 0.000 sec
102	14:04:11	SELECT c.customerID, c.fullName, o.orderID, o.totalCost AS Cost, m.menuName, m_i.courseName, m_i.starterName	2 row(s) returned	0.000 sec / 0.000 sec

Find all menu items for which more than 2 orders have been placed.

The screenshot shows the MySQL Workbench interface. The 'Query' tab is active, displaying the following SQL query:

```
1 use littlelemondb;
2 SELECT menuName FROM Menu
3 WHERE
4 menuID = ANY (SELECT menuID FROM Orders
5 GROUP BY menuID
6 HAVING COUNT(orderID) > 2);
7
8
```

The 'Navigator' pane on the left shows the database structure, including tables like 'customers', 'menu', 'orders', and 'superstore_model'. The 'Output' pane at the bottom shows the execution results of the query, including the time taken and the number of rows returned.

#	Time	Action	Message	Duration / Fetch
54	11:58:54	SELECT menuName FROM Menu WHERE menuID = ANY (SELECT menuID	0 row(s) returned	0.000 sec / 0.000 sec
55	12:00:14	select * from orders LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec
56	12:01:43	SELECT menuID FROM Orders GROUP BY menuID HAVING	0 row(s) returned	0.000 sec / 0.000 sec
57	12:02:18	SELECT menuID FROM Orders GROUP BY menuID HAVING	0 row(s) returned	0.000 sec / 0.000 sec
58	12:02:58	SELECT menuName FROM Menu WHERE menuID = ANY (SELECT menuID	0 row(s) returned	0.000 sec / 0.000 sec
59	12:05:02	SELECT menuName FROM Menu WHERE menuID = ANY (SELECT menuID FROM Orders GROUP BY m...	0 row(s) returned	0.000 sec / 0.000 sec

MYSQL QUERY OPTIMIZATION WITH STORED PROCEDURES AND PREPARED STATEMENTS:

CREATING OPTIMIZED QUERIES TO MANAGE AND ANALYZE DATA:

Create a procedure that displays the maximum ordered quantity in the Orders table.

MySQL Workbench

sales_analysis - Warning - not... MySQL Model reviewedModel.m... EER Diagram

File Edit View Query Database Server Tools Scripting Help

Navigator

SCHMAS

Filter objects

hr

little_lemon

little_lemon_db

littlelemondb

Tables

customers

menus

orders

Views

Stored Procedures

Functions

littlelemondb

mangata_galo

mg_schema

sales

storage

superstore_model

Tables

Views

Stored Procedures

Administration Schemas

Information

No object selected

Query 1

```

1 use littlelemondb;
2 DELIMITER //
3
4 CREATE PROCEDURE GetMaxQuantity()
5 BEGIN
6 SELECT MAX(Quantity) AS "Max Quantity In Order"
7 FROM Orders;
8 END //
9
10 DELIMITER ;
11 CALL GetMaxQuantity();
12

```

Result Grid Filter Rows: Export: Wrap Cell Contents: 15

Max Quantity In Order

9

Result 29

Output

Action Output

#	Time	Action	Message	Duration / Fetch
68	13:17:14	DROP procedure GetMaxQuantity;	0 row(s) affected	0.000 sec
69	13:17:27	CREATE PROCEDURE GetMaxQuantity() BEGIN SELECT MAX(Quantity) AS "Max Quantity In Order" FROM ...	0 row(s) affected	0.000 sec
70	13:17:31	CALL GetMaxQuantity();	1 row(s) returned	0.000 sec / 0.000 sec
71	13:17:49	DROP procedure GetMaxQuantity;	0 row(s) affected	0.016 sec
72	13:18:10	CREATE PROCEDURE GetMaxQuantity() BEGIN SELECT MAX(Quantity) AS "Max Quantity In Order" FROM ...	0 row(s) affected	0.000 sec
73	13:18:16	CALL GetMaxQuantity();	1 row(s) returned	0.000 sec / 0.000 sec

Query Completed

Object Info Session

EUR/USD +0.27%

ENG 1:18 PM 6/24/2024

Create a prepared statement called *GetOrderDetail*. This prepared statement will help to reduce the parsing time of queries. It will also help to secure the database from SQL injections.

MySQL Workbench

sales_analysis - Warning - not... MySQL Model reviewedModel.m... EER Diagram

File Edit View Query Database Server Tools Scripting Help

Navigator

SCHMAS

Filter objects

hr

little_lemon

little_lemon_db

littlelemondb

Tables

customers

menus

orders

Views

Stored Procedures

Functions

littlelemondb

mangata_galo

mg_schema

sales

storage

superstore_model

Tables

Views

Stored Procedures

Administration Schemas

Information

No object selected

Query 1

```

1 use littlelemondb;
2
3 select * from orders;
4 PREPARE GetOrderDetail FROM
5 'SELECT orderID, Quantity, totalCost
6 FROM Orders
7 WHERE customerID = ?';
8
9 SET @CustomerID = 2;
10 EXECUTE GetOrderDetail USING @CustomerID ;
11

```

Result Grid Filter Rows: Export: Wrap Cell Contents: 15

orderID Quantity totalCost

2 2 250.00

Result 41

Output

Action Output

#	Time	Action	Message	Duration / Fetch
102	14:04:11	SELECT c.customerID,c.fullName,o.orderID,o.totalCost AS Cost,m.menuName,m.j_courseName,m.j_startDate...	2 row(s) returned	0.000 sec / 0.000 sec
103	14:12:17	select * from orders LIMIT 0, 1000	4 row(s) returned	0.000 sec / 0.000 sec
104	14:13:07	insert into orders values(5,5.2,250.00)	1 row(s) affected	0.016 sec
105	14:14:08	PREPARE GetOrderDetail FROM 'SELECT orderID, Quantity, totalCost FROM Orders WHERE customerID ...	0 row(s) affected	0.000 sec
106	14:14:11	SET @CustomerID = 2	0 row(s) affected	0.000 sec
107	14:14:13	EXECUTE GetOrderDetail USING @CustomerID	1 row(s) returned	0.000 sec / 0.000 sec

Query Completed

Object Info Session

17°C

ENG 2:14 PM 6/24/2024

Create a stored procedure called CancelOrder to delete an order record based on the user input of the order id.

The screenshot displays the MySQL Workbench interface. The central editor shows the following SQL code:

```
1 use littlelemondb;
2 CREATE TABLE OrderCancellations (
3   id INT AUTO_INCREMENT PRIMARY KEY,
4   orderID INT,
5   confirmation VARCHAR(255)
6 );
7 DELIMITER //
8 CREATE PROCEDURE CancelOrder(IN id INT)
9 BEGIN
10  DELETE FROM Orders WHERE orderID = id;
11  INSERT INTO OrderCancellations (orderID, confirmation) VALUES (id, CONCAT('Order ', id, ' is cancelled'));
12  SELECT * FROM OrderCancellations;
13 END //
14 DELIMITER ;
15 Call CancelOrder(5);
```

The Output window at the bottom shows the execution results:

#	Time	Action	Message	Duration / Fetch
92	13:39:53	CREATE PROCEDURE CancelOrder(IN id INT) BEGIN	DELETE FROM Orders WHERE orderID = id; IN...	0 row(s) affected / 0.000 sec
93	13:39:57	Call CancelOrder(5)	2 row(s) returned	0.016 sec / 0.000 sec
94	13:42:27	delete from OrderCancellations;	Error Code: 1175. You are using safe update mode and you tried to update a table without a WHERE that uses...	0.000 sec
95	13:43:01	select * from OrderCancellations LIMIT 0, 1000 ;	2 row(s) returned	0.000 sec / 0.000 sec
96	13:43:15	delete from OrderCancellations where id = 1;	1 row(s) affected	0.000 sec
97	13:43:18	delete from OrderCancellations where id = 2;	1 row(s) affected	0.000 sec

The interface also includes a Navigator on the left showing the database schema, a right-hand pane with context help, and a status bar at the bottom indicating the query is completed.