

DATABASE

STORE

Local instance MySQL80 x

File Edit View Query Database Server Tools Scripting Help

SCHEMAS

Filter objects

Query 1

Limit to 1000 rows

12

13

14

15

16

17

18

19

20

INSERT INTO Store (storeId, sName, sAddress, sPostCode)

VALUES

('Store01', 'Shoes Williams', '277 Queen, BrisbaneCity', '1234'),

('Store02', 'Vono', '116 Queen, BrsCity', '5678');

SELECT * FROM Store;

Result Grid

Filter Rows

Wrap Cell Content

storeId	sName	sAddress	sPostCode
Store01	Shoes Williams	277 Queen, BrisbaneCity	1234
Store02	Vono	116 Queen, BrsCity	5678

Administration

Schemas

Information

No object selected

SQL Additions

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

Store 1 x

Apply Revert Context Help Snippets

Output

Action Output

#	Time	Action	Message	Duration / Fetch
10	23:31:37	CREATE TABLE Store (storeId VARCHAR(7), sName TEXT NOT NULL, sAddress TEXT, sPostCode VARCHAR(10))	0 row(s) affected	0.016 sec
11	23:31:37	INSERT INTO Store (storeId, sName, sAddress, sPostCode) VALUES ('Store01', 'Shoes Williams', '277 Queen, BrisbaneCity', '1234'), ('Store02', 'Vono', '116 Queen, BrsCity', '5678');	2 row(s) affected Records: 2 Duplicates: 0 Warnings: 0	0.000 sec
12	23:31:37	SELECT * FROM Store LIMIT 0, 1000	2 row(s) returned	0.000 sec / 0.000 sec

1 | Page

StoreEmployee

The screenshot shows the MySQL Workbench interface for a local instance of MySQL 8.0. The 'Query' window contains the following SQL script:

```
18 CREATE TABLE StoreEmployee (  
19     empId VARCHAR(7),  
20     eName TEXT NOT NULL,  
21     eAddress TEXT,  
22     ePostCode VARCHAR(4),  
23     eEmail TEXT,  
24     eMobph TEXT,  
25     eStarDate TEXT,  
26     storeId VARCHAR(7),  
27     CONSTRAINT StoreEmployee_PK PRIMARY KEY(empId),  
28     CONSTRAINT StoreEmployee_Store_FK FOREIGN KEY (storeId) REFERENCES Store(storeId)  
29 );  
30  
31 INSERT INTO StoreEmployee (empId, eName, eAddress, ePostCode, eEmail, eMobph, eStarDate, storeId)  
32 VALUES  
33 ('em1', 'kishore', '71 Gold Coast', '4567', 'kishore3456@gmail.com', '3456-756-446', '23-09-199', 'store01'),  
34 ('em2', 'Aswath', '78 Gold Coast', '5674', 'Aswath567@gmail.com', '5678-789-673', '17-10-1999', 'store02');
```

The 'Result Grid' shows the data inserted into the table:

empId	eName	eAddress	ePostCode	eEmail	eMobph	eStarDate	storeId
em1	kishore	71 Gold Coast	4567	kishore3456@gmail.com	3456-756-446	23-09-199	store01
em2	Aswath	78 Gold Coast	5674	Aswath567@gmail.com	5678-789-673	17-10-1999	store02

The 'Output' window shows the execution results:

#	Time	Action	Message	Duration / Fetch
24	00:03:53	INSERT INTO StoreEmployee (empId, eName, eAddress, ePostCode, eEmail, eMobph, eStarDate, storeId) VA...	2 row(s) affected Records: 2 Duplicates: 0 Warnings: 0	0.000 sec
25	00:03:53	SELECT * FROM StoreEmployee LIMIT 0, 1000	2 row(s) returned	0.000 sec / 0.000 sec

CUSTOMER

The screenshot shows the MySQL Workbench interface for a local instance of MySQL 8.0. The 'Query' window contains the following SQL script:

```
36 ('em2', 'Aswath', '78 Gold Coast', '5674', 'Aswath567@gmail.com', '5678-789-673', '17-10-1999', 'store02');  
37 SELECT * FROM StoreEmployee;  
38  
39 CREATE TABLE Customer (  
40     CustId VARCHAR(7),  
41     cName TEXT NOT NULL,  
42     cMobilePh TEXT,  
43     cEmail TEXT,  
44     cBirthDate TEXT,  
45     CONSTRAINT Customer_PK PRIMARY KEY (custId)  
46 );  
47 INSERT INTO Customer (CustId, cName, cMobilePh, cEmail, cBirthDate)  
48 VALUES  
49 ('Cust1', 'peter', '8960-567-566', 'peter3456@gmail.com', '23-09-199'),  
50 ('Cust2', 'Bose', '56788-543-666', 'Bose@gmail.com', '17-10-1999');  
51 SELECT * FROM Customer;  
52
```

The 'Result Grid' shows the data inserted into the table:

CustId	cName	cMobilePh	cEmail	cBirthDate
Cust1	peter	8960-567-566	peter3456@gmail.com	23-09-199
Cust2	Bose	56788-543-666	Bose@gmail.com	17-10-1999

The 'Output' window shows the execution results:

#	Time	Action	Message	Duration / Fetch
40	00:25:24	INSERT INTO Customer (CustId, cName, cMobilePh, cEmail, cBirthDate) VALUES ('Cust1', 'peter', '8960-567...	2 row(s) affected Records: 2 Duplicates: 0 Warnings: 0	0.000 sec
41	00:25:24	SELECT * FROM Customer LIMIT 0, 1000	2 row(s) returned	0.000 sec / 0.000 sec

ORDER

The screenshot shows the MySQL Workbench interface with a query editor containing SQL code to create an 'order' table and insert two records. The table has columns: orderId (VARCHAR(7)), oDate (TEXT), total (INTEGER), GST (INTEGER), deliveryAddress (TEXT), orderStatus (TEXT), and custId (VARCHAR(7)). It includes a primary key constraint on orderId and a foreign key constraint on custId referencing the Customer table. The result grid shows two rows of data.

```
53 CREATE TABLE `order` (  
54   orderId VARCHAR(7),  
55   oDate TEXT,  
56   total INTEGER,  
57   GST INTEGER,  
58   deliveryAddress TEXT,  
59   orderStatus TEXT,  
60   custId VARCHAR(7),  
61   CONSTRAINT order_PK PRIMARY KEY (orderId),  
62   CONSTRAINT order_Customer_FK FOREIGN KEY (custId) REFERENCES Customer (CustId)  
63 );  
64 INSERT INTO `order` (orderId, oDate, total, GST, deliveryAddress, orderStatus, custId)  
65 VALUES  
66 ('order1', '23-05-2023', '300', '20', '03 parkcentral, Brisbane city', 'processing', 'Cust1'),  
67 ('order2', '15-06-2023', '250', '30', '04 Taranga, Brisbane city', 'shipped', 'Cust2');  
68 SELECT * FROM `order`;  
69
```

orderId	oDate	total	GST	deliveryAddress	orderStatus	custId
order1	23-05-2023	300	20	03 parkcentral, Brisbane city	processing	Cust1
order2	15-06-2023	250	30	04 Taranga, Brisbane city	shipped	Cust2

Output:

#	Time	Action	Message	Duration / Fetch
61	01:01:44	INSERT INTO 'order' (orderId, oDate, total, GST, deliveryAddress, orderStatus, custId) VALUES ('order1', '23-05-2023', '300', '20', '03 parkcentral, Brisbane city', 'processing', 'Cust1')	2 row(s) affected Records: 2 Duplicates: 0 Warnings: 0	0.000 sec
62	01:01:44	SELECT * FROM 'order' LIMIT 0, 1000	2 row(s) returned	0.000 sec / 0.000 sec

ORDERDETAILS

The screenshot shows the MySQL Workbench interface with a query editor containing SQL code to create an 'orderDetail' table and insert two records. The table has columns: orderId (VARCHAR(7)), prodId (VARCHAR(7)), quantity (INTEGER), and retailPrice (INTEGER). It includes a primary key constraint on (orderId, prodId) and foreign key constraints on orderId and prodId referencing the 'order' and 'product' tables respectively. The result grid shows two rows of data.

```
1 CREATE TABLE orderDetail (  
2   orderId VARCHAR(7),  
3   prodId VARCHAR(7),  
4   quantity INTEGER,  
5   retailPrice INTEGER,  
6   CONSTRAINT ORDERDETAILS_PK PRIMARY KEY (orderId, prodId),  
7   CONSTRAINT ORDERDETAILS_Order_FK FOREIGN KEY (orderId) REFERENCES `order` (orderId),  
8   CONSTRAINT ORDERDETAILS_product_FK FOREIGN KEY (prodId) REFERENCES product (prodId)  
9 );  
10 INSERT INTO orderDetail (orderId, prodId, quantity, retailPrice )  
11 VALUES  
12 ('order1', '56789', '4', '200' ),  
13 ('order2', '7869', '5', '250');  
14 SELECT * FROM orderDetails;
```

orderId	prodId	quantity	retailPrice
order1	56789	400	45
order2	7869	500	50

Output:

#	Time	Action	Message	Duration / Fetch
102	02:18:43	DROP TABLE 'walkthruway'.orderDetail	0 row(s) affected	0.031 sec
103	02:18:50	CREATE TABLE orderDetail (orderId VARCHAR(7), prodId VARCHAR(7), quantity INTEGER, retailPrice INTE...	0 row(s) affected	0.094 sec
104	02:18:50	INSERT INTO orderDetail (orderId, prodId, quantity, retailPrice) VALUES ('order1', '56789', '400', '45'), ('order2', '7869', '500', '50')	2 row(s) affected Records: 2 Duplicates: 0 Warnings: 0	0.016 sec

PAYMENT

The screenshot shows the MySQL Workbench interface with the 'payment' table being created and populated. The SQL editor contains the following code:

```

118 CREATE TABLE payment(
119     payId VARCHAR(7),
120     type TEXT,
121     amount INTEGER,
122     pDate TEXT,
123     bankTransactNo TEXT,
124     orderId VARCHAR(7),
125     CONSTRAINT payment_PK PRIMARY KEY (payId),
126     CONSTRAINT Payment_order_FK FOREIGN KEY (orderId) REFERENCES `order` (orderId)
127 );
128 INSERT INTO payment (payId, type, amount, pDate, bankTransactNo, orderId)
129 VALUES
130 ('pay1', 'debit', '300', '12-3-2023', 'Rfd4567', 'order1'),
131 ('pay2', 'credit', '200', '16-3-2023', 'Rfd7872', 'order2');
132 SELECT * FROM payment;
133
134 CREATE TABLE productSupplier(

```

The 'Result Grid' shows the data inserted into the 'payment' table:

payId	type	amount	pDate	bankTransactNo	orderId
pay1	debit	300	12-3-2023	Rfd4567	order1
pay2	credit	200	16-3-2023	Rfd7872	order2

The 'Action Output' pane shows the execution of the SQL statements:

#	Time	Action	Message	Duration / Fetch
5	14:35:27	INSERT INTO payment (payId, type, amount, pDate, bankTransactNo, orderId) VALUES ('pay1','debit','300',...	2 row(s) affected Records: 2 Duplicates: 0 Warnings: 0	0.000 sec
6	14:35:27	SELECT * FROM payment LIMIT 0.1000	2 row(s) returned	0.000 sec / 0.000 sec

Query Completed

PRODUCT

The screenshot shows the MySQL Workbench interface with the 'product' table being created and populated. The SQL editor contains the following code:

```

2 CREATE TABLE Product(
3     prodId VARCHAR(7),
4     size TEXT,
5     colour TEXT,
6     style TEXT,
7     qtyOnHand INTEGER,
8     reorderQty INTEGER,
9     retailPrice INTEGER,
10    supId VARCHAR(7),
11    CONSTRAINT product_PK PRIMARY KEY (prodId),
12    CONSTRAINT product_Supplier_FK FOREIGN KEY (supId) REFERENCES supplier(supId)
13 );
14 INSERT INTO product (prodId, size, colour, style, qtyOnHand, reorderQty, retailPrice, supId)
15 VALUES
16 ('56789', '7', 'Blue', 'canvas', '200', '25', '45', 'sup1'),
17 ('7869', '6', 'Black', 'Boat', '250', '35', '50', 'sup2');
18 SELECT * FROM product;

```

The 'Result Grid' shows the data inserted into the 'product' table:

prodId	size	colour	style	qtyOnHand	reorderQty	retailPrice	supId
56789	7	Blue	canvas	200	25	45	sup1
7869	6	Black	Boat	250	35	50	sup2

The 'Action Output' pane shows the execution of the SQL statements:

#	Time	Action	Message	Duration / Fetch
1	14:31:59	SELECT * FROM walkthruway product LIMIT 0.1000	2 row(s) returned	0.000 sec / 0.000 sec

PRODUCTSUPPLIER

The screenshot shows the MySQL Workbench interface with a query editor containing SQL code to create and populate the `productSupplier` table. The code includes a `CREATE TABLE` statement with columns `prodId` and `supId`, both of type `VARCHAR(7)`. It also includes a `PRIMARY KEY` constraint on `prodId` and `supId`, and two `FOREIGN KEY` constraints: `ProductSupplier_product_FK` referencing `product(prodId)` and `ProductSupplier_supplier_FK` referencing `supplier(supId)`. The `INSERT INTO` statement adds two rows: `(56789, 'sup1')` and `(7869, 'sup2')`. The `SELECT * FROM productSupplier;` statement is used to verify the data.

The output window shows the execution results of the `INSERT INTO` and `SELECT` statements. The `INSERT INTO` statement successfully inserted 2 rows. The `SELECT` statement returned 2 rows.

prodId	supId
56789	sup1
7869	sup2

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

SUPPLIER

The screenshot shows the MySQL Workbench interface with a query editor containing SQL code to create and populate the `Supplier` table. The code includes a `CREATE TABLE` statement with columns `supId`, `supName`, `supAddress`, `supPostcode`, `supEmail`, and `supMobph`. The `supId` column is of type `VARCHAR(7)` and is the `PRIMARY KEY`. The other columns are of type `TEXT`. The `INSERT INTO` statement adds three rows: `(sup1, 'handmade', '456 Thomas street, Taranga', '7689', 'handmade@gmail.com', '4567-567-6454')`, `(sup2, 'lock', '653 Queen street, Brisbane city', '4567', 'lock@gmail.com', '3678-786-567')`, and `(sup3, 'lock', '653 Queen street, Brisbane city', '4567', 'lock@gmail.com', '3678-786-567')`. The `SELECT * FROM Supplier;` statement is used to verify the data.

The output window shows the execution results of the `INSERT INTO` and `SELECT` statements. The `INSERT INTO` statement successfully inserted 3 rows. The `SELECT` statement returned 3 rows.

supId	supName	supAddress	supPostcode	supEmail	supMobph
sup1	handmade	456 Thomas street, Taranga	7689	handmade@gmail.com	4567-567-6454
sup2	lock	653 Queen street, Brisbane city	4567	lock@gmail.com	3678-786-567
sup3	lock	653 Queen street, Brisbane city	4567	lock@gmail.com	3678-786-567

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

Write an SQL script for querying data for the following information.

1. List the hotelNo, type and price of each room that is a suite or family with price more than \$90. Order the result by type, hotelNo, and price .

```
USE IFN554Hoteldb;
SELECT hotelNo, roomtype AS type, price
FROM Room
WHERE roomtype in ('Suite', 'Family') and price > 90
ORDER BY roomtype, hotelNo, price;
```

The screenshot displays the SQL Server Enterprise Manager interface. The top pane shows the SQL script being executed. The 'Result Grid' pane below it shows the query results. The results are as follows:

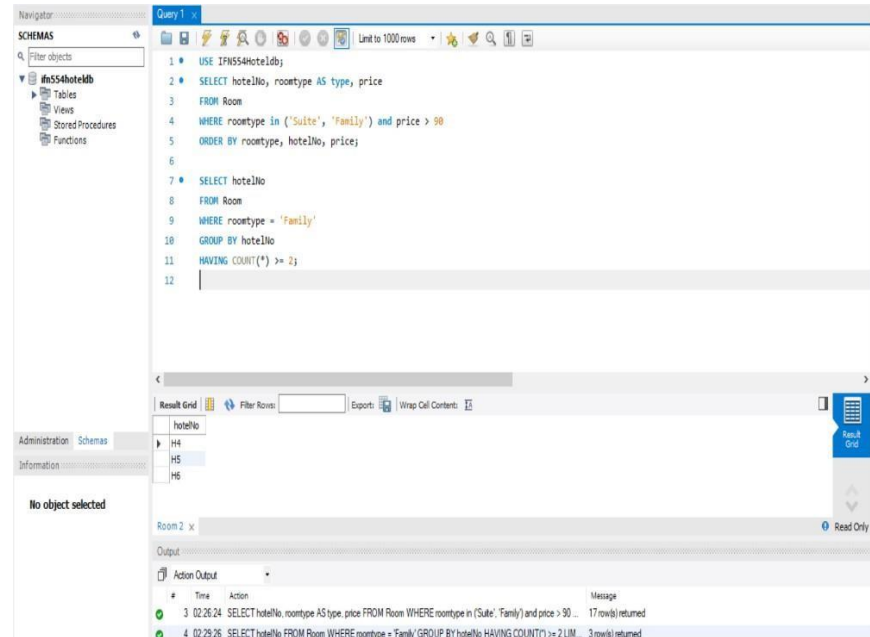
hotelNo	type	price
H1	Family	95.00
H2	Family	105.00
H3	Family	115.00
H4	Family	115.00
H5	Family	105.00
H5	Family	105.00
H5	Family	115.00
H6	Family	105.00
H6	Family	105.00
H7	Family	115.00
H1	Suite	95.00
H2	Suite	95.00
H3	Suite	95.00
H4	Suite	95.00
H5	Suite	95.00
H7	Suite	95.00

The bottom pane shows the 'Output' window with the following message:

#	Time	Action	Message
2	02:26:24	USE IFN554Hoteldb	0 row(s) affected
3	02:26:24	SELECT hotelNo, roomtype AS type, price FROM Room WHERE roomtype in ('Suite', 'Family') and price > 90 ...	17 row(s) returned

2. List the hotelNo which has 2 or more family rooms .

```
SELECT hotelNo
FROM Room
WHERE roomtype = 'Family'
GROUP BY hotelNo
HAVING COUNT(*) >= 2;
```



The screenshot shows the SQL Server Enterprise Manager interface. The query window displays the following SQL code:

```
1 USE IFMS54HotelDb;
2 SELECT hotelNo, roomtype AS type, price
3 FROM Room
4 WHERE roomtype in ('Suite', 'Family') and price > 90
5 ORDER BY roomtype, hotelNo, price;
6
7 SELECT hotelNo
8 FROM Room
9 WHERE roomtype = 'Family'
10 GROUP BY hotelNo
11 HAVING COUNT(*) >= 2;
```

The Results grid shows the following data:

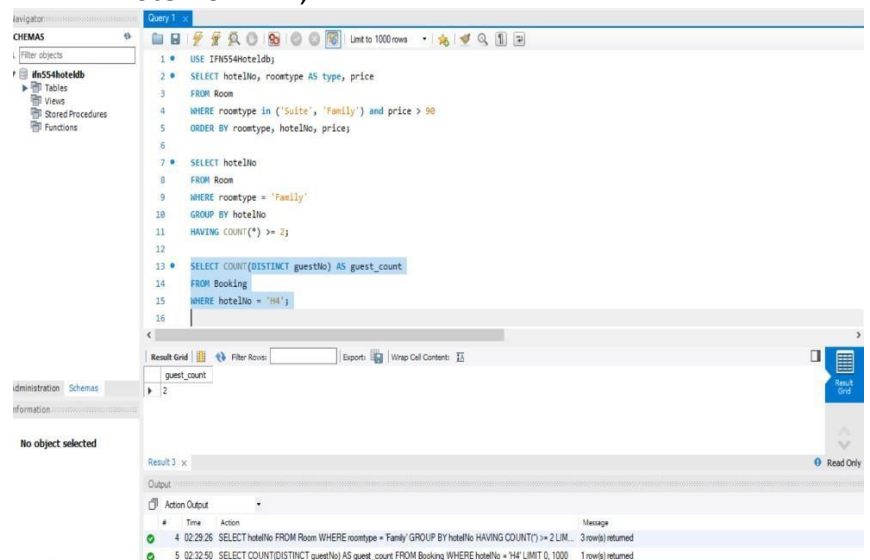
hotelNo
H4
H5
H6

The Action Output pane shows the following messages:

```
4 02:26:24 SELECT hotelNo, roomtype AS type, price FROM Room WHERE roomtype in ('Suite', 'Family') and price > 90 ... 17 row(s) returned
4 02:29:26 SELECT hotelNo FROM Room WHERE roomtype = 'Family' GROUP BY hotelNo HAVING COUNT(*) >= 2 LIM... 3 row(s) returned
```

3. How many different guests visited the Accor Hotel?

```
SELECT COUNT(DISTINCT guestNo) AS guest_count
FROM Booking
WHERE hotelNo = 'H4';
```



The screenshot shows the SQL Server Enterprise Manager interface. The query window displays the following SQL code:

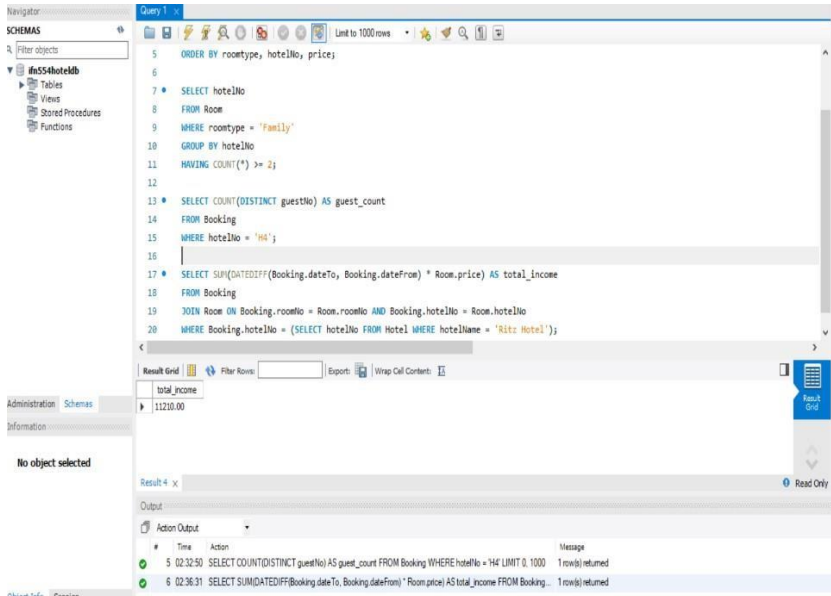
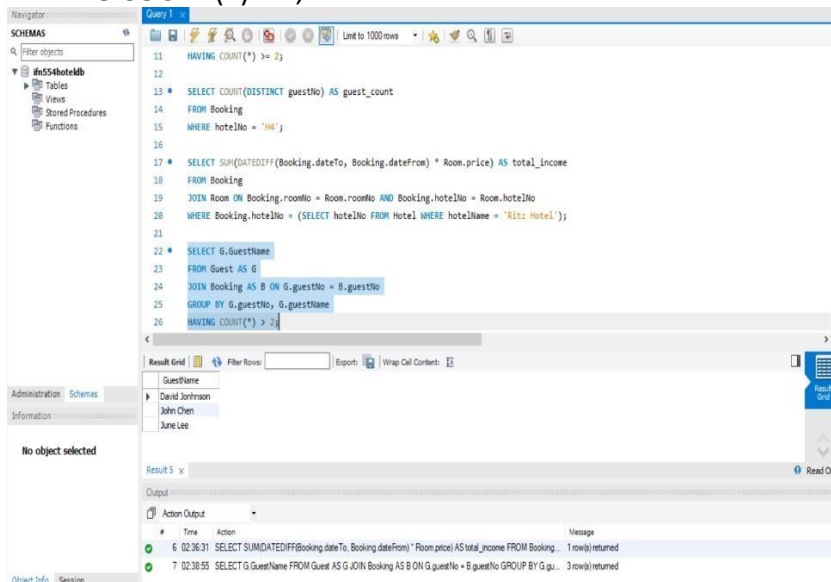
```
1 USE IFMS54HotelDb;
2 SELECT hotelNo, roomtype AS type, price
3 FROM Room
4 WHERE roomtype in ('Suite', 'Family') and price > 90
5 ORDER BY roomtype, hotelNo, price;
6
7 SELECT hotelNo
8 FROM Room
9 WHERE roomtype = 'Family'
10 GROUP BY hotelNo
11 HAVING COUNT(*) >= 2;
```

The Results grid shows the following data:

guest_count
2

The Action Output pane shows the following messages:

```
4 02:29:26 SELECT hotelNo FROM Room WHERE roomtype = 'Family' GROUP BY hotelNo HAVING COUNT(*) >= 2 LIM... 3 row(s) returned
5 02:32:50 SELECT COUNT(DISTINCT guestNo) AS guest_count FROM Booking WHERE hotelNo = 'H4' LIMIT 0, 1000 1 row(s) returned
```


<p>4. What is the total income from bookings for the Ritz Hotel?</p>	<pre> SELECT SUM(DATEDIFF(Booking.dateTo, Booking.dateFrom) * Room.price) AS total_income FROM Booking JOIN Room ON Booking.roomNo = Room.roomNo AND Booking.hotelNo = Room.hotelNo WHERE Booking.hotelNo = (SELECT hotelNo FROM Hotel WHERE hotelName = 'Ritz Hotel'); </pre> 
<p>5. List all the guests' names who have visited more than 2 times</p>	<pre> SELECT G.GuestName FROM Guest AS G JOIN Booking AS B ON G.guestNo = B.guestNo GROUP BY G.guestNo, G.guestName HAVING COUNT(*) > 2; </pre> 
<p>6. List all the guests' names who have visited more than 2</p>	<pre> SELECT G.GuestName FROM Guest AS G JOIN Booking AS B ON G.guestNo = B.guestNo GROUP BY G.guestNo, G.guestName </pre>

times

HAVING COUNT(*) > 2;

The screenshot displays the SQL Server Enterprise Manager interface. The left pane shows the 'SCHEMAS' tree with 'AdventureWorks2008' expanded. The central pane shows a query window with the following SQL code:



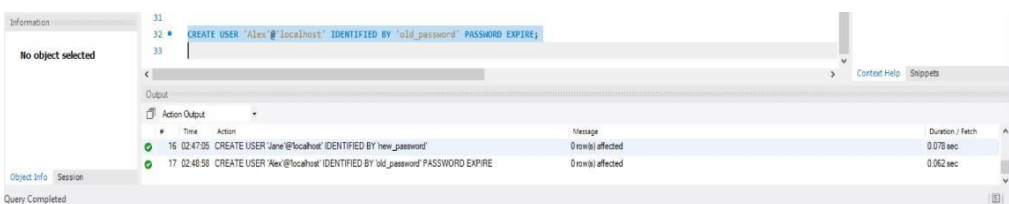
```
11 HAVING COUNT(*) >= 2;
12
13 SELECT COUNT(DISTINCT guestNo) AS guest_count
14 FROM Booking
15 WHERE hotelNo = '104';
16
17 SELECT SUM(DATEDIFF(Booking.dateTo, Booking.dateFrom) * Room.price) AS total_income
18 FROM Booking
19 JOIN Room ON Booking.roomNo = Room.roomNo AND Booking.hotelNo = Room.hotelNo
20 WHERE Booking.hotelNo = (SELECT hotelNo FROM Hotel WHERE hotelName = 'Ritz Hotel');
21
22 SELECT G.guestName
23 FROM Guest AS G
24 JOIN Booking AS B ON G.guestNo = B.guestNo
25 GROUP BY G.guestNo, G.guestName
26 HAVING COUNT(*) > 2;
```


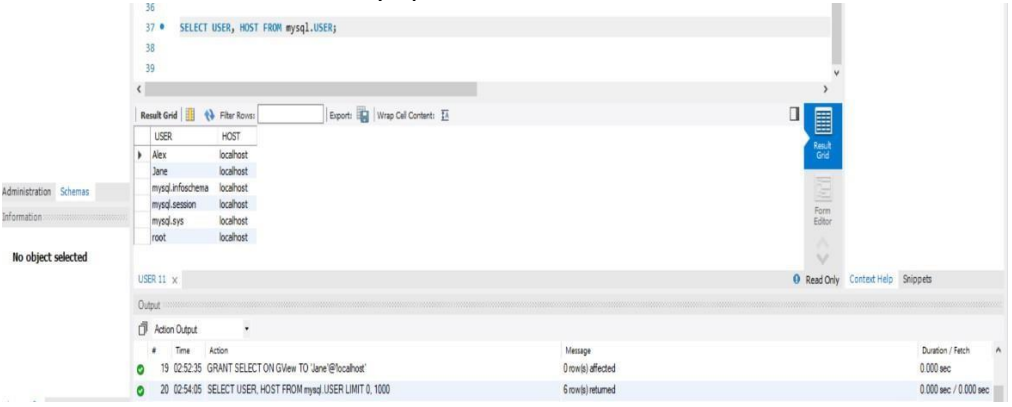
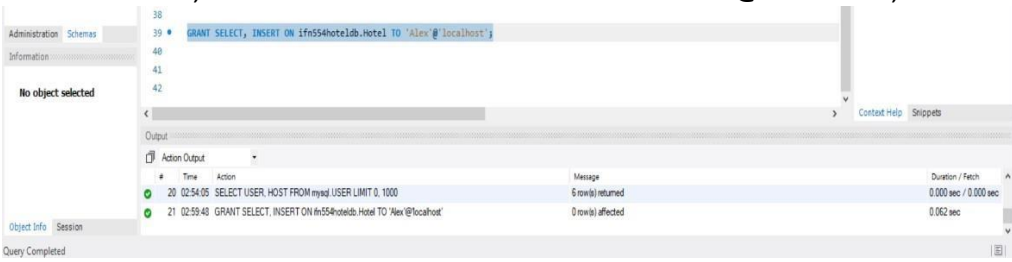
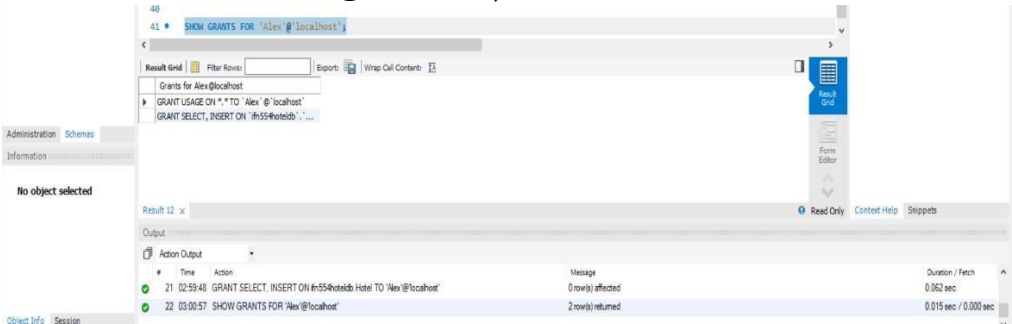
The right pane shows the 'Result Grid' with the following data:

GuestName
David Johnson
John Chen
June Lee

Below the result grid, the 'Output' pane shows the execution plan and results:

#	Time	Action	Message
6	02:36:31	SELECT SUM(DATEDIFF(Booking.dateTo, Booking.dateFrom) * Room.price) AS total_income FROM Booking...	1 row(s) returned
7	02:36:55	SELECT G.guestName FROM Guest AS G JOIN Booking AS B ON G.guestNo = B.guestNo GROUP BY G.gu...	3 row(s) returned

<p>1. Write a command to create an index on guestName of the Guest table.</p>	<pre>CREATE INDEX Indexes_guestName ON Guest(guestName);</pre> 
<p>2. Create a user with the name “Jane” @ local host with password new_password</p>	<pre>CREATE USER 'Jane'@'localhost' IDENTIFIED BY 'new_password';</pre> 
<p>3. Create a user with the name “Alex” @ local host with password old_password. Mark the password expired so that the user must choose a new one at the first connection to the server.</p>	<pre>CREATE USER 'Alex'@'localhost' IDENTIFIED BY 'old_password' PASSWORD EXPIRE;</pre> 
<p>4. Create a view called GView showing Guest No and Guest Name and Grant permission to</p>	<pre>CREATE VIEW GView AS SELECT guestNo, guestName FROM Guest; GRANT SELECT ON GView TO 'Jane'@'localhost';</pre>

<p>Jane to select this view</p> <p>Note: Both parts are necessary for a correct answer. No part marks.</p>	
<p>5. Display a list of users</p>	<p>SELECT USER, HOST FROM mysql.USER;</p> 
<p>6. Grant Select, Insert to Alex on table Hotel</p>	<p>GRANT SELECT, INSERT ON ifn554hoteldb.Hotel TO 'Alex'@'localhost';</p> 
<p>7. Show Grants on table hotel to Alex</p>	<p>SHOW GRANTS FOR 'Alex'@'localhost';</p> 

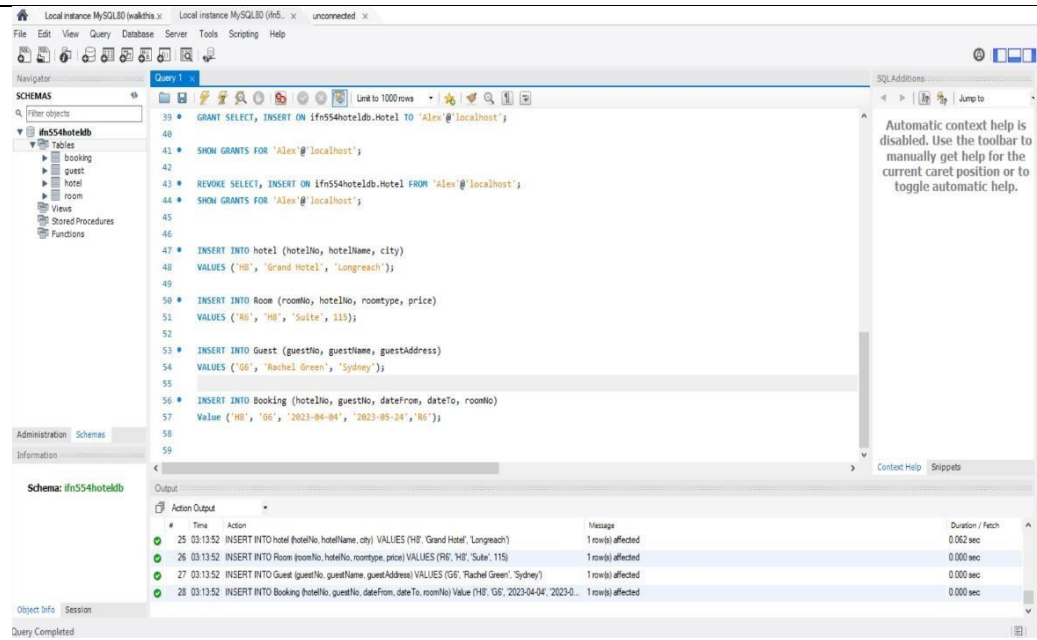
8. Remove Select and Insert privileges on hotel from Alex and Show grants

REVOKE SELECT, INSERT ON ifn554hoteldb.Hotel FROM 'Alex'@'localhost';
SHOW GRANTS FOR 'Alex'@'localhost';

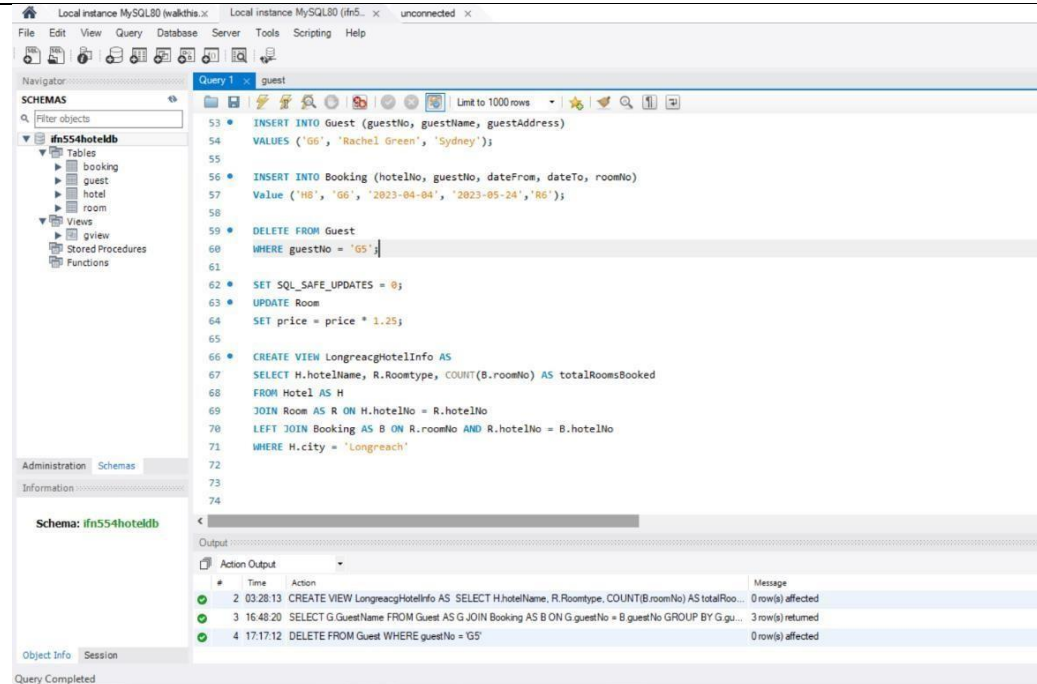
The screenshot shows the SQL Server Enterprise Manager interface. The left pane displays the 'Administration' tree with 'Schemas' selected. The right pane shows the 'Query Editor' with two SQL statements: 'REVOKE SELECT, INSERT ON ifn554hoteldb.Hotel FROM 'Alex'@'localhost';' and 'SHOW GRANTS FOR 'Alex'@'localhost';'. Below the editor, the 'Results' pane shows the output of the 'SHOW GRANTS' command, displaying a single row: 'GRANT USAGE ON *.* TO 'Alex'@'localhost''.

#	Time	Action	Message	Duration / Fetch
23	03/03/12	REVOKE SELECT, INSERT ON ifn554hoteldb.Hotel FROM 'Alex'@'localhost'	0 row(s) affected	0.063 sec
24	03/03/12	SHOW GRANTS FOR 'Alex'@'localhost'	1 row(s) returned	0.000 sec / 0.000 sec

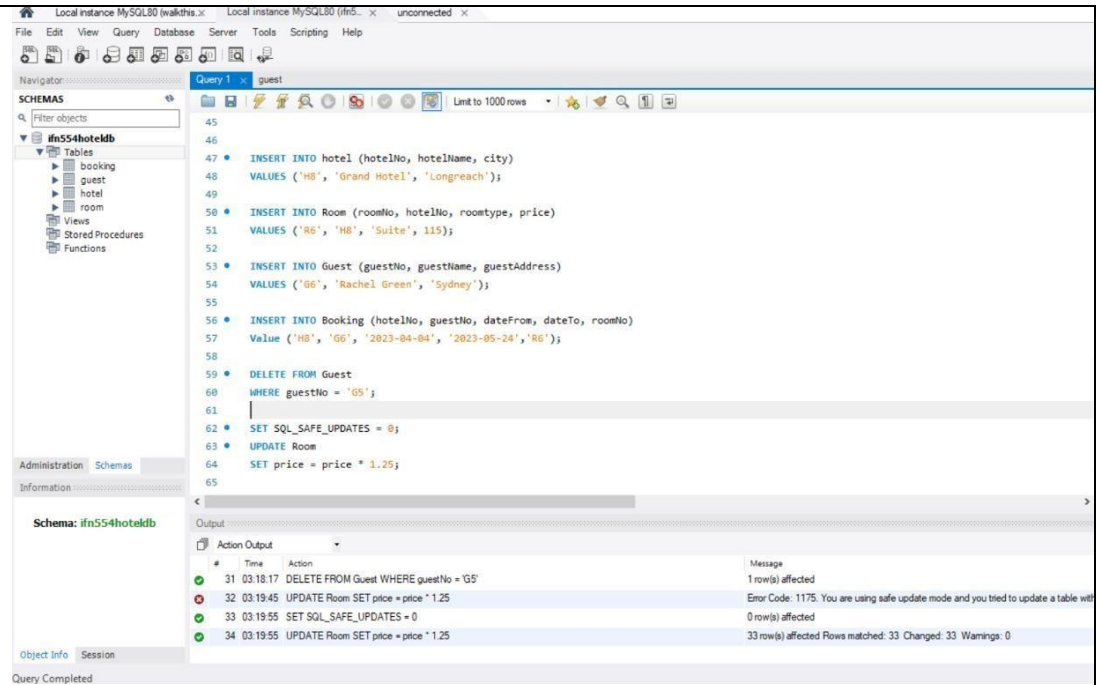
1. Write commands to insert rows in each of the Hotel database tables.



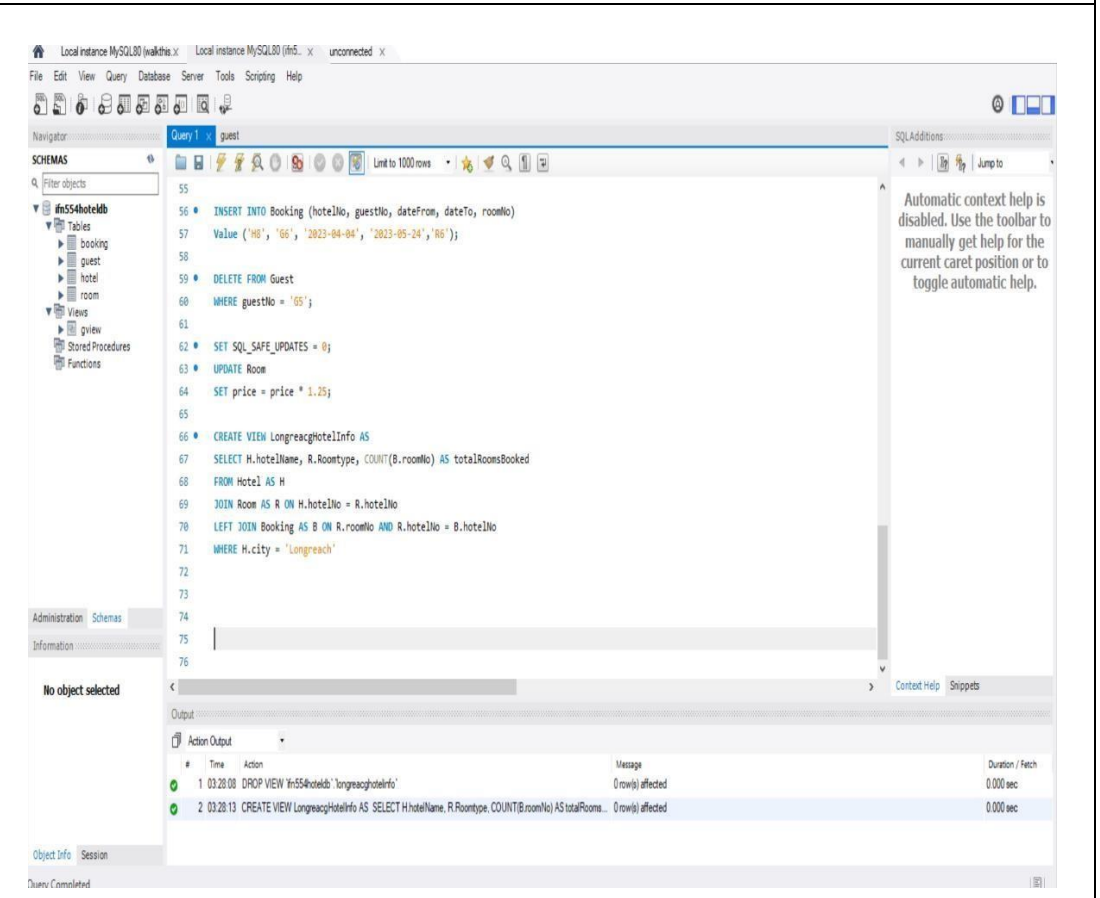
2. Write a command to delete the row you inserted in the table Guest.



3. Write a command to update the price of all rooms by 25%.



4. Write a command to create a view to list the information (hotelName, roomType and the total number of rooms booked) of the hotels which are in the city Longreach



Assumptions:

In the following table, these assumptions can be made:

1. there are no multivalued dependencies.
2. any invoice numbers (invoiceID) may reference more than one product.
3. any given product is supplied by a single supplier, but a supplier can supply many products.

invoiceID	productID	clientID	productDetail	vendorID	quantitySold	productPrice
925547	SA-S4522QW	2312	Sandle Strapped	SH5120	5	\$120.00
925547	BZ- 857932X	2451	Boot Zipped	SH2110	7	\$250.00
958547	SS -995748G	2312	Slides Suede	SH3094	10	\$275.00
937448	SP-S3422QW	2451	Stiletto point spiked	SH2110	12	\$115.00
965149	SP-778345P	2312	Sneaker Pump	SH1574	16	\$95.00
965784	BL-B47WP0	2458	Boots Long Blk	SH2410	2	\$290.00

The functional dependencies are

invoiceID -> productID,->ClientID-> productsDetails-> vendorId ->quantitySold->productPrice->

productID-> vendorID

invoiceID	clientID	productDetail	quantitySold	productPrice
925547	2312	Sandle Strapped	5	\$120.00
925540	2451	Boot Zipped	7	\$250.00
958547	2312	Slides Suede	10	\$275.00
937448	2451	Stiletto point (spiked)	12	\$115.00
965149	2312	Sneaker Pump	16	\$95.00
965784	2458	Boots Long Blk	2	\$290.00

productID	vendorID
SA-S4522QW	SH5120
BZ- 857932X	SH2110
SS -995748G	SH3094
SP-S3422QW	SH2110
SP-778345P	SH1574
BL-B47WP0	SH2410

These data leaks have made me more wary of revealing personal information. I've been more choosy about what information I submit to organizations, particularly those with a poor record of data protection. It has encouraged me to prioritize digital hygiene, such as using reliable, distinctive passwords, establishing two-factor authentication, and monitoring security settings on social media sites on a regular basis.

Gathering of Data and Storage: Businesses should use robust encryption techniques for data at rest and in transit, establish strong access controls, and update their safety measures on a regular basis. Organizations should closely comply to privacy legislation that include GDPR or CCPA to ensure the permissible use of data. This entails gaining informed consent from individuals and only using data for the purposes mentioned. Organizations should provide comprehensive disclosures when obtaining data from clients or consumers, describing precisely how the data will be used, who will have access to it, and giving individuals the option to opt in or out of data sharing. Unauthorized sharing or auction of data, applying it for unrelated objectives, or failing to effectively protect it are all examples of unethical use. Failure to address security access and dissemination can result in unauthorized entry and misuse of personal information, which can have disastrous consequences. Data breaches can cause significant financial losses, damage an organization's brand, result in legal consequences, and lose customer or client trust. Neglecting issues such as security access and data sharing may result in unauthorized access and misuse of sensitive data, exacerbating the severity of the harm. This includes adhering to core concepts such as privacy, confidentiality, and permission.