

DBMS PROJECT

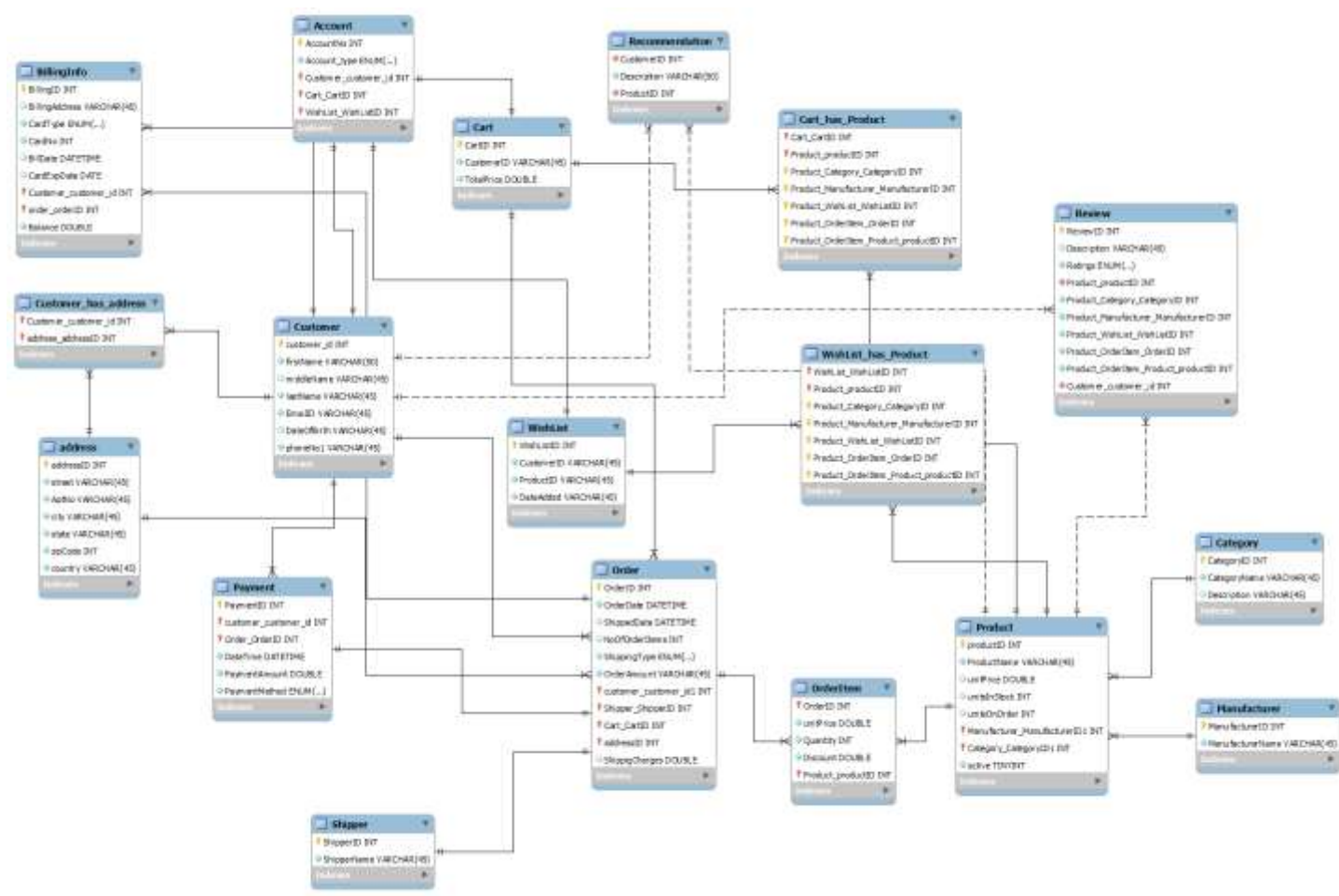
ANN SARA SAJEE – 001813733

AMAZON DATABASE

PROJECT SUMMARY

E-commerce databases are quite interesting and hence I choose an E-commerce database-AMAZON to work on. The purpose was to deliver an E-commerce database that is used for e-commerce operations like check out products, place an order, do payment, review products, find for recommendations but also for data analysis to extract valuable insights from these data for business betterment. I am using queries, joins, views, stored procedures, triggers and a transaction and a few analytical queries to show the working of this project.

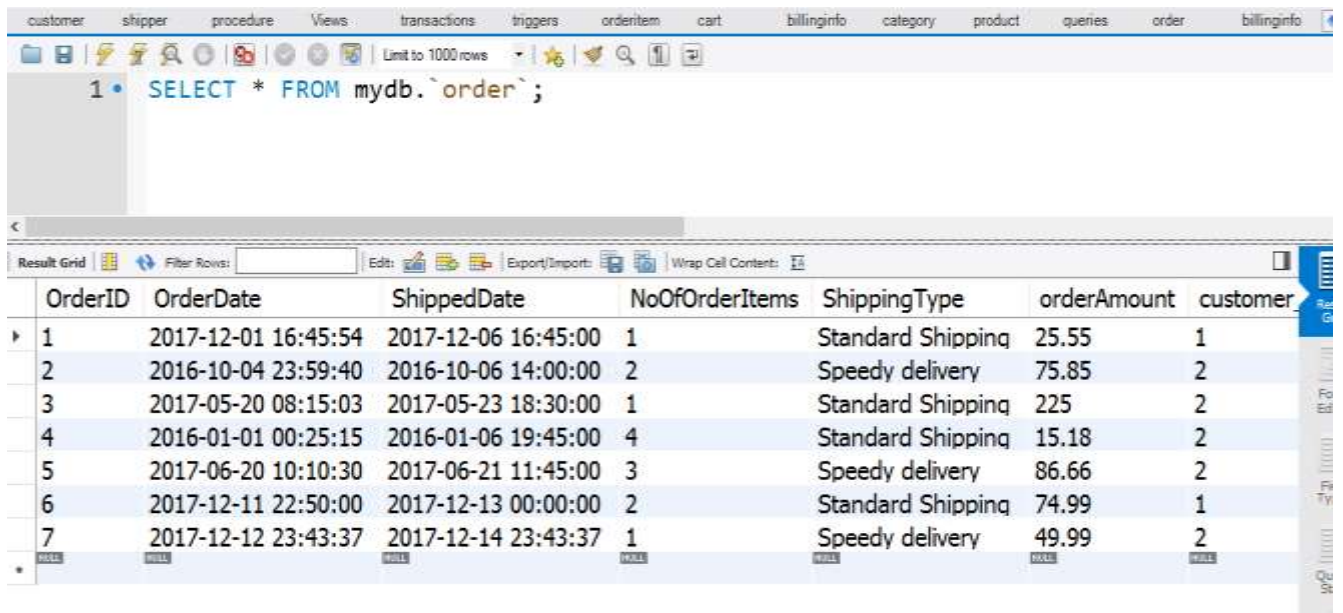
EER DIAGRAM



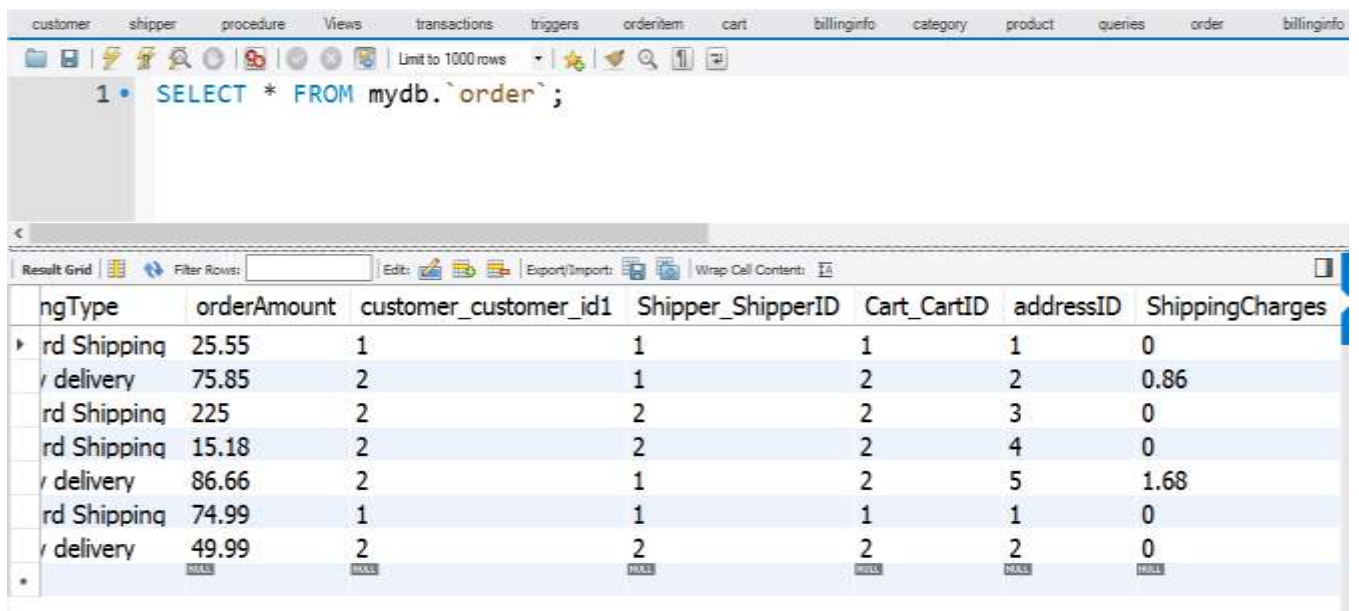
TABLES AND DATA

The following tables are included in my project:

1. **ORDERS:** The table contains the following columns:

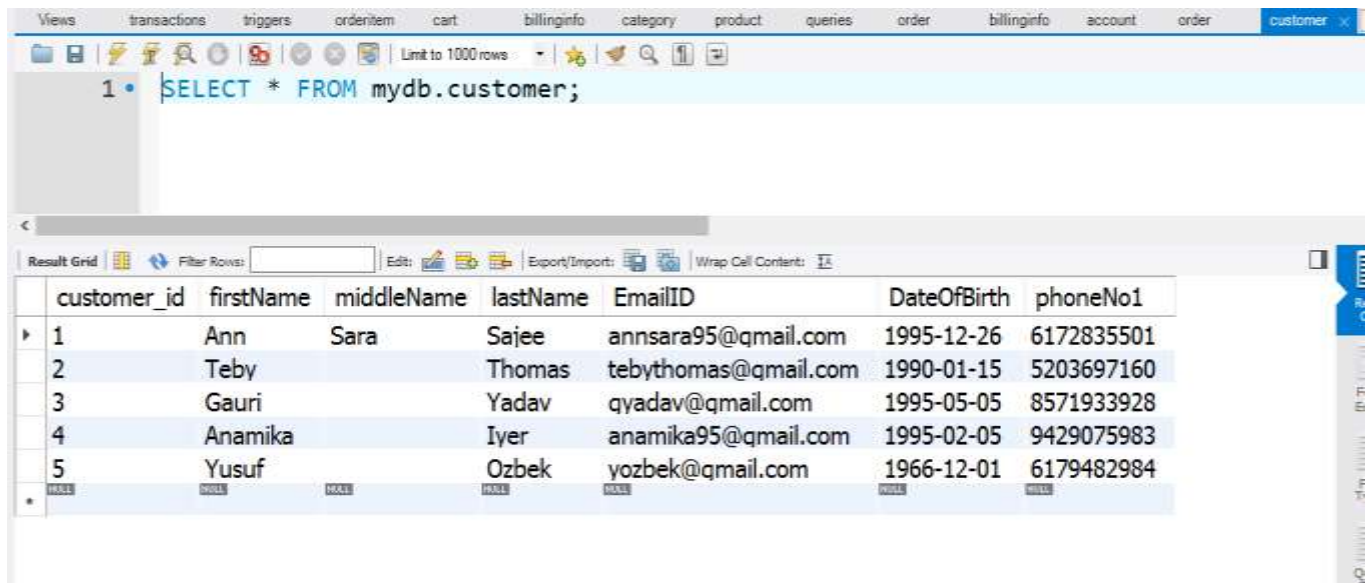


OrderID	OrderDate	ShippedDate	NoOfOrderItems	ShippingType	orderAmount	customer
1	2017-12-01 16:45:54	2017-12-06 16:45:00	1	Standard Shipping	25.55	1
2	2016-10-04 23:59:40	2016-10-06 14:00:00	2	Speedy delivery	75.85	2
3	2017-05-20 08:15:03	2017-05-23 18:30:00	1	Standard Shipping	225	2
4	2016-01-01 00:25:15	2016-01-06 19:45:00	4	Standard Shipping	15.18	2
5	2017-06-20 10:10:30	2017-06-21 11:45:00	3	Speedy delivery	86.66	2
6	2017-12-11 22:50:00	2017-12-13 00:00:00	2	Standard Shipping	74.99	1
7	2017-12-12 23:43:37	2017-12-14 23:43:37	1	Speedy delivery	49.99	2



ShippingType	orderAmount	customer_customer_id1	Shipper_ShipperID	Cart_CartID	addressID	ShippingCharges
Standard Shipping	25.55	1	1	1	1	0
Speedy delivery	75.85	2	1	2	2	0.86
Standard Shipping	225	2	2	2	3	0
Standard Shipping	15.18	2	2	2	4	0
Speedy delivery	86.66	2	1	2	5	1.68
Standard Shipping	74.99	1	1	1	1	0
Speedy delivery	49.99	2	2	2	2	0

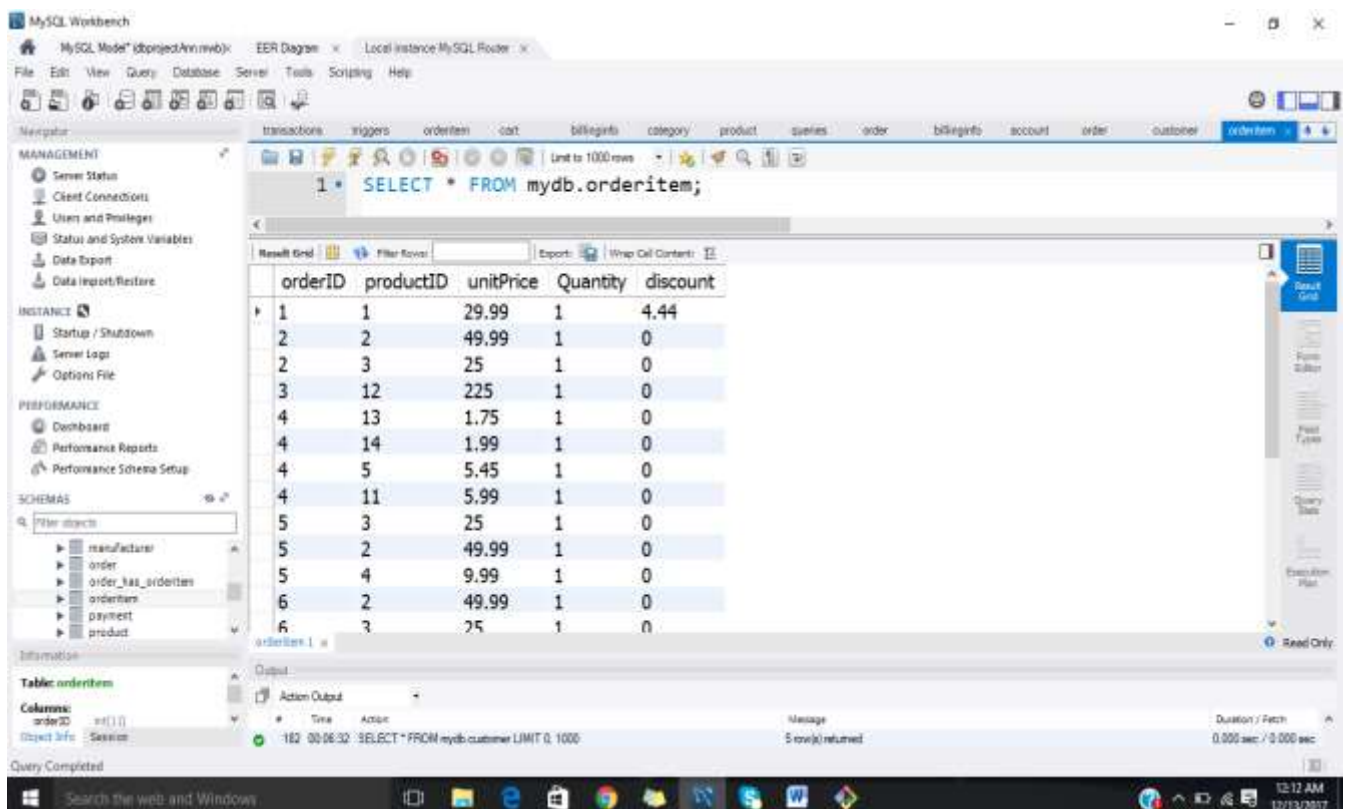
2. CUSTOMERS: This table contains basic information about the customers



The screenshot shows the MySQL Workbench interface with the 'customer' table selected in the top tab. The query editor contains the SQL statement: `SELECT * FROM mydb.customer;`. The 'Result Grid' displays the following data:

customer_id	firstName	middleName	lastName	EmailID	DateOfBirth	phoneNo1
1	Ann	Sara	Sajee	annsara95@gmail.com	1995-12-26	6172835501
2	Teby		Thomas	tebythomas@gmail.com	1990-01-15	5203697160
3	Gauri		Yadav	qyadav@gmail.com	1995-05-05	8571933928
4	Anamika		Iyer	anamika95@gmail.com	1995-02-05	9429075983
5	Yusuf		Ozbek	yozbek@gmail.com	1966-12-01	6179482984

3. ORDER ITEMS: An order contains a number of order items. It's basically a product with a specific unit price, quantity and discounts. It has the following attributes :

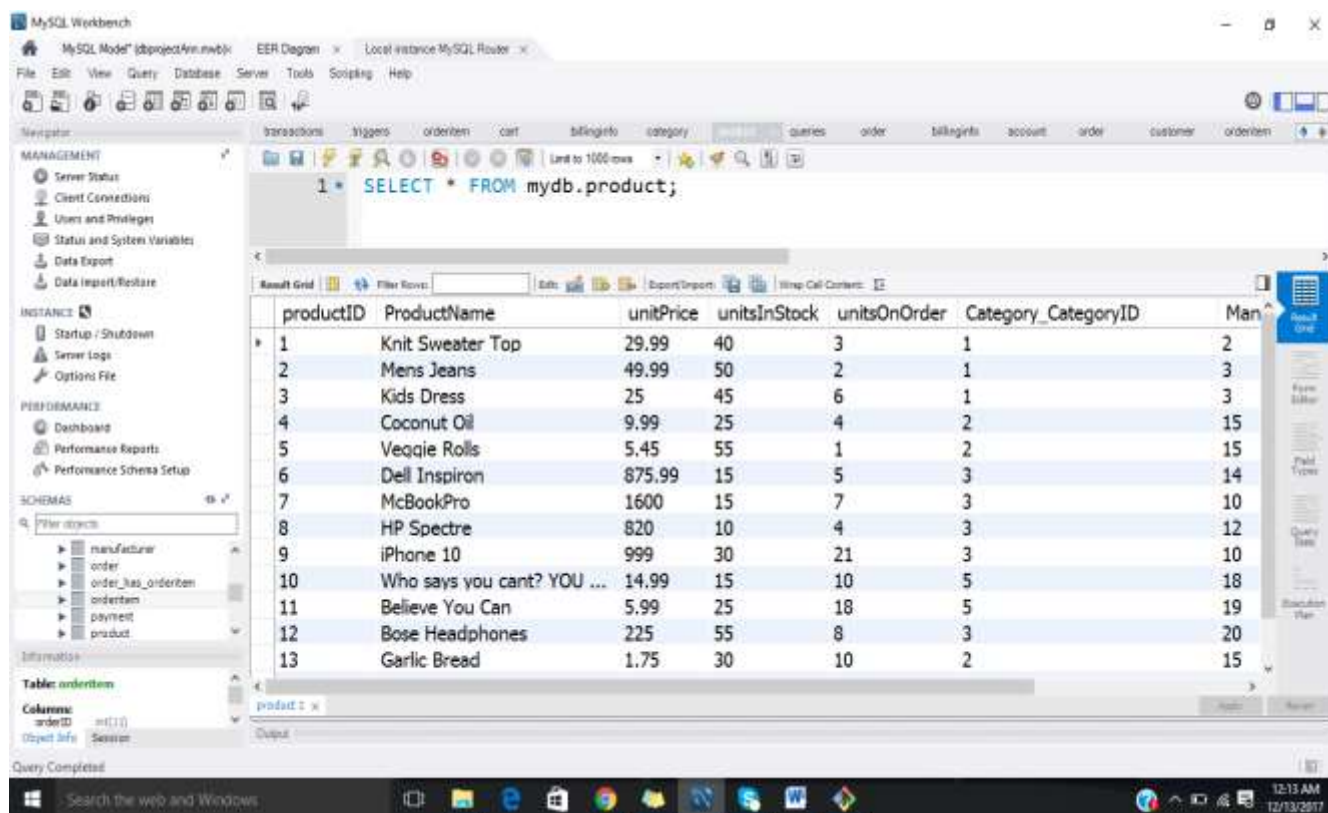


The screenshot shows the MySQL Workbench interface with the 'orderitem' table selected in the top tab. The query editor contains the SQL statement: `SELECT * FROM mydb.orderitem;`. The 'Result Grid' displays the following data:

orderID	productID	unitPrice	Quantity	discount
1	1	29.99	1	4.44
2	2	49.99	1	0
2	3	25	1	0
3	12	225	1	0
4	13	1.75	1	0
4	14	1.99	1	0
4	5	5.45	1	0
4	11	5.99	1	0
5	3	25	1	0
5	2	49.99	1	0
5	4	9.99	1	0
6	2	49.99	1	0
6	3	25	1	0

At the bottom of the screenshot, a status bar indicates: `182 00:06:32 SELECT * FROM mydb.customer LIMIT 0, 1000` with a message `5 row(s) returned` and a duration of `0.000 sec / 0.000 sec`.

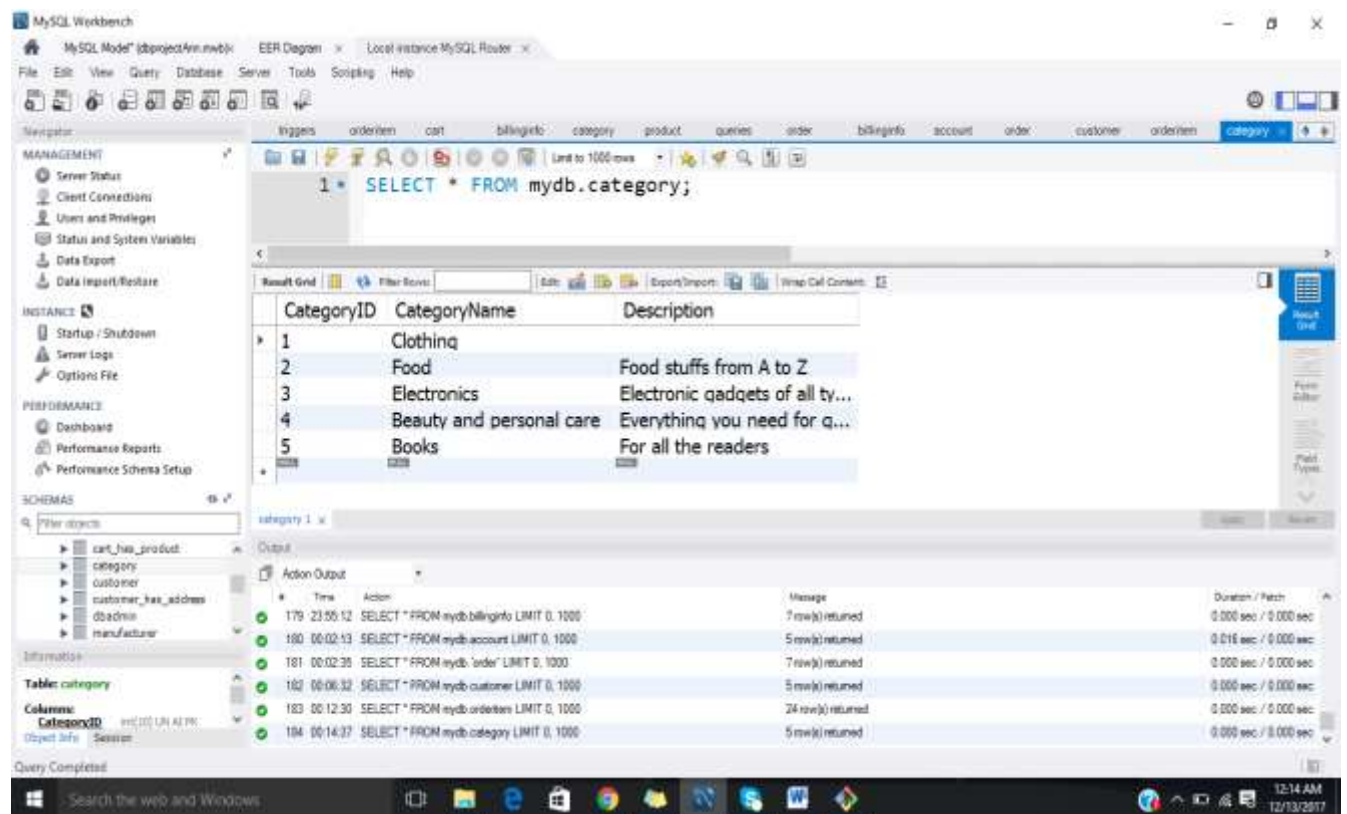
4. PRODUCTS: Product is an item that is delivered to the customers. It has the following attributes :



The screenshot shows the MySQL Workbench interface. The query editor contains the SQL statement: `SELECT * FROM mydb.product;`. The result grid displays 13 rows of product data. The columns are: productID, ProductName, unitPrice, unitsInStock, unitsOnOrder, Category_CategoryID, and ManufacturerID. The left sidebar shows the database schema with tables like manufacturer, order, order_has_orderitem, orderitem, payment, and product. The bottom status bar indicates the query is completed.

productID	ProductName	unitPrice	unitsInStock	unitsOnOrder	Category_CategoryID	ManufacturerID
1	Knit Sweater Top	29.99	40	3	1	2
2	Mens Jeans	49.99	50	2	1	3
3	Kids Dress	25	45	6	1	3
4	Coconut Oil	9.99	25	4	2	15
5	Veggie Rolls	5.45	55	1	2	15
6	Dell Inspiron	875.99	15	5	3	14
7	McBookPro	1600	15	7	3	10
8	HP Spectre	820	10	4	3	12
9	iPhone 10	999	30	21	3	10
10	Who says you cant? YOU ...	14.99	15	10	5	18
11	Believe You Can	5.99	25	18	5	19
12	Bose Headphones	225	55	8	3	20
13	Garlic Bread	1.75	30	10	2	15

5. **CATEGORY:** Every product belongs to a certain category. Its attributes are as follows:

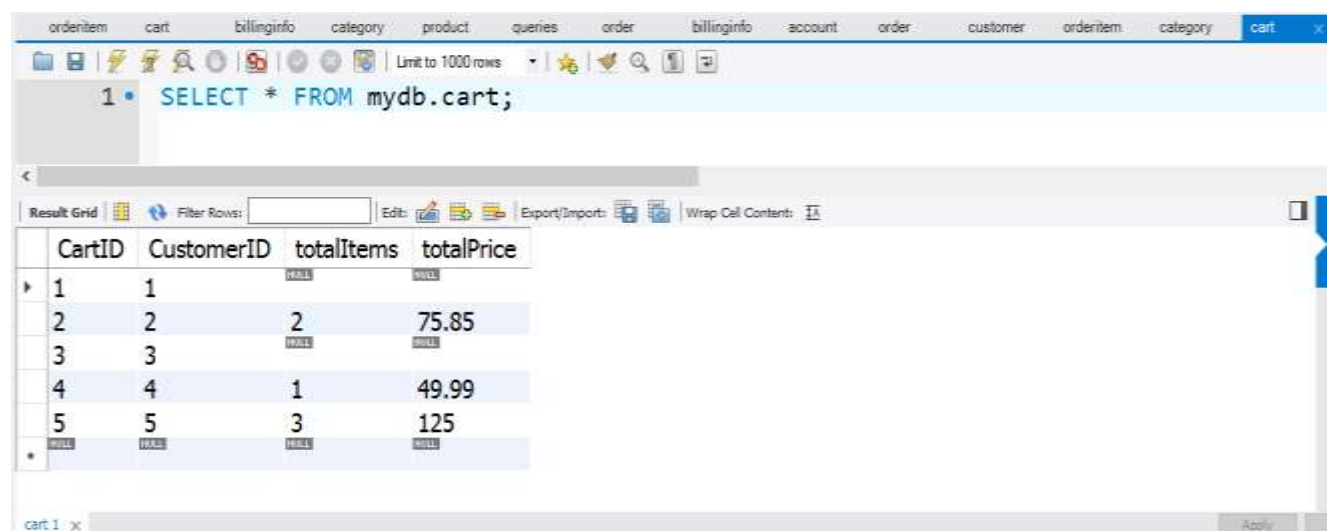


The screenshot shows the MySQL Workbench interface with the 'category' table selected in the 'SHEMAS' pane. The 'Result Grid' displays the following data:

CategoryID	CategoryName	Description
1	Clothing	
2	Food	Food stuffs from A to Z
3	Electronics	Electronic gadqets of all ty...
4	Beauty and personal care	Everything you need for q...
5	Books	For all the readers

The 'Action Output' pane at the bottom shows the execution of the query 'SELECT * FROM mydb.category;' with 5 rows returned.

6. **CART:** A temporary list of items the customer wants to purchase. It has the following attributes:

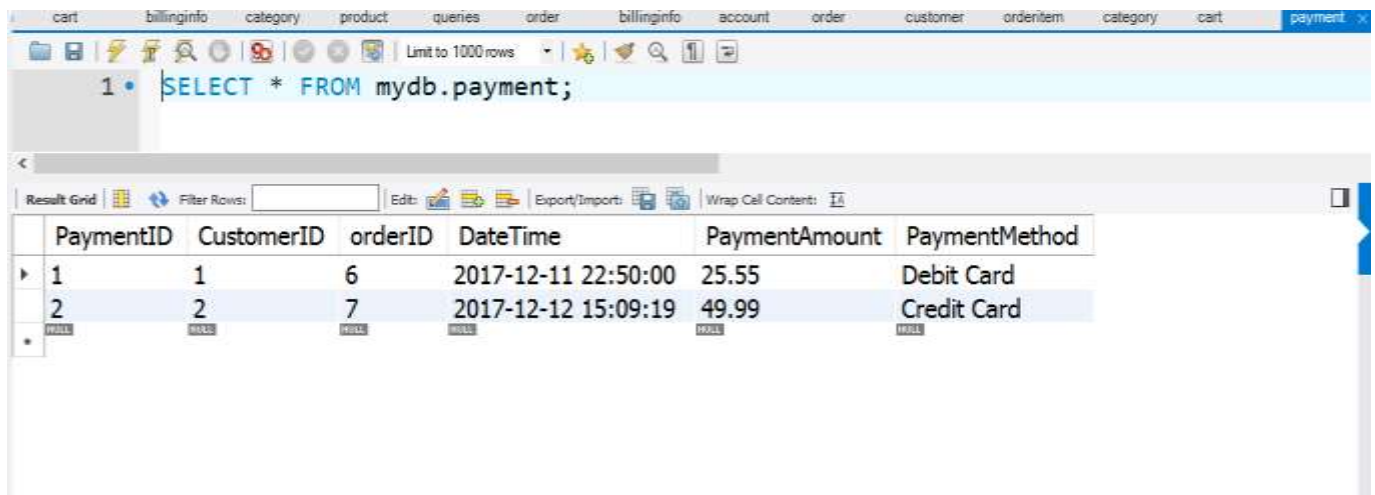


The screenshot shows the MySQL Workbench interface with the 'cart' table selected in the 'SHEMAS' pane. The 'Result Grid' displays the following data:

CartID	CustomerID	totalItems	totalPrice
1	1	1	75.85
2	2	2	75.85
3	3	1	49.99
4	4	1	125
5	5	3	125

The 'Action Output' pane at the bottom shows the execution of the query 'SELECT * FROM mydb.cart;' with 5 rows returned.

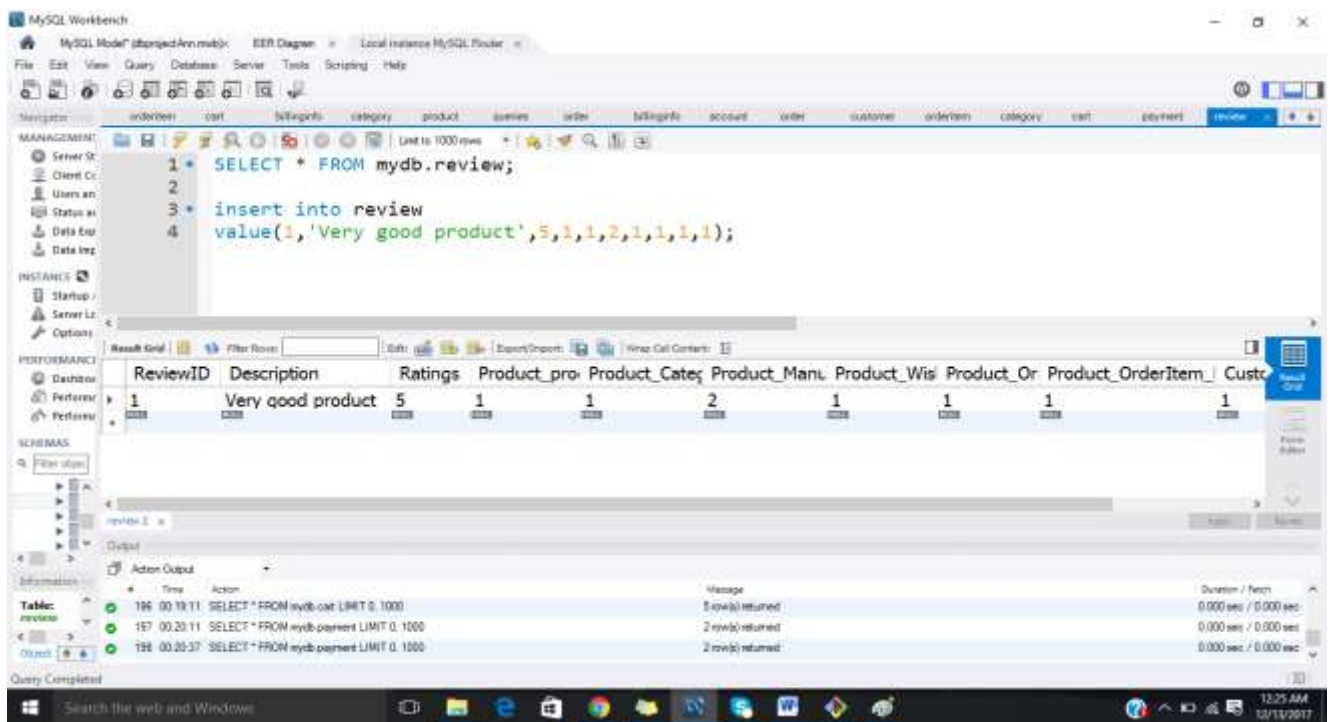
7. PAYMENT: When a customer purchases items, a payment is made. This entity has the following attributes:



The screenshot shows the MySQL Workbench interface with a query window containing the SQL statement: `SELECT * FROM mydb.payment;`. The result grid displays two rows of data from the payment table.

PaymentID	CustomerID	orderID	DateTime	PaymentAmount	PaymentMethod
1	1	6	2017-12-11 22:50:00	25.55	Debit Card
2	2	7	2017-12-12 15:09:19	49.99	Credit Card

8. REVIEWS: A customer can give reviews for the product he bought

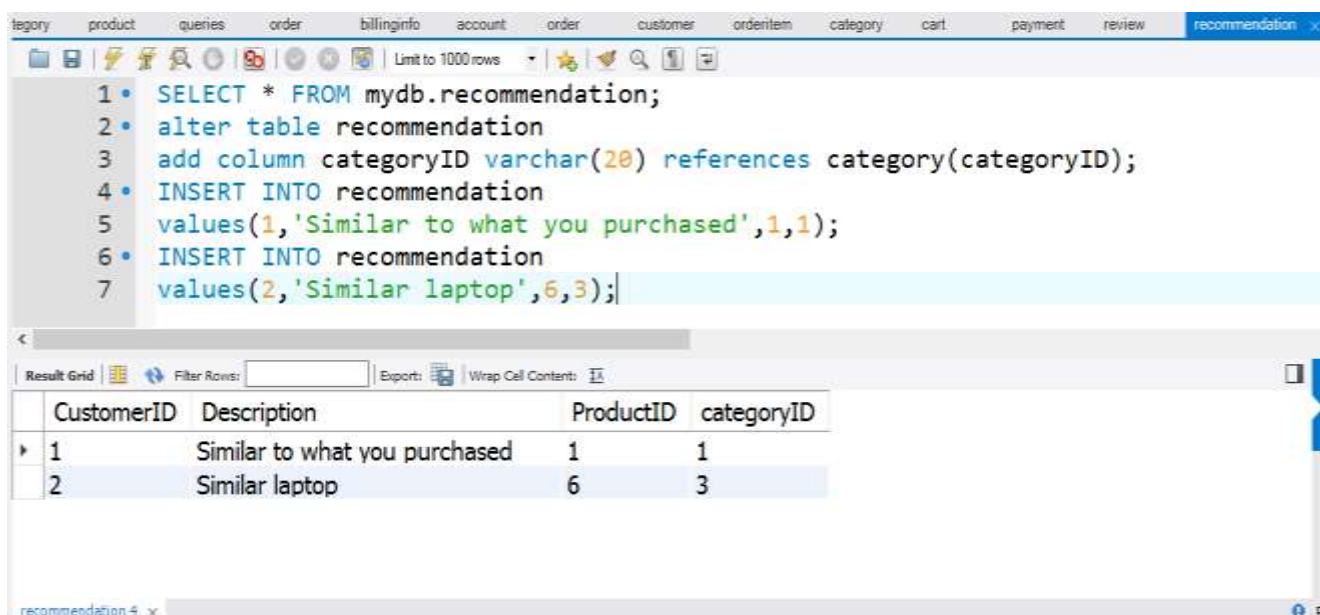


The screenshot shows the MySQL Workbench interface with a query window containing the SQL statements: `SELECT * FROM mydb.review;` and `insert into review value(1,'Very good product',5,1,1,2,1,1,1,1);`. The result grid displays one row of data from the review table. Below the result grid, the Action Log shows the execution of three queries.

ReviewID	Description	Ratings	Product_pro	Product_Cate	Product_Manu	Product_Wis	Product_Or	Product_OrderItem	Custo
1	Very good product	5	1	1	2	1	1	1	1

Time	Action	Message	Duration / Fetch
196 00:19:11	SELECT * FROM mydb.cart LIMIT 0.1000	5 row(s) returned	0.000 sec / 0.000 sec
197 00:20:11	SELECT * FROM mydb.payment LIMIT 0.1000	2 row(s) returned	0.000 sec / 0.000 sec
198 00:20:37	SELECT * FROM mydb.payment LIMIT 0.1000	2 row(s) returned	0.000 sec / 0.000 sec

9. RECOMMENDATIONS: Every customer gets a recommendation based on his/her purchases



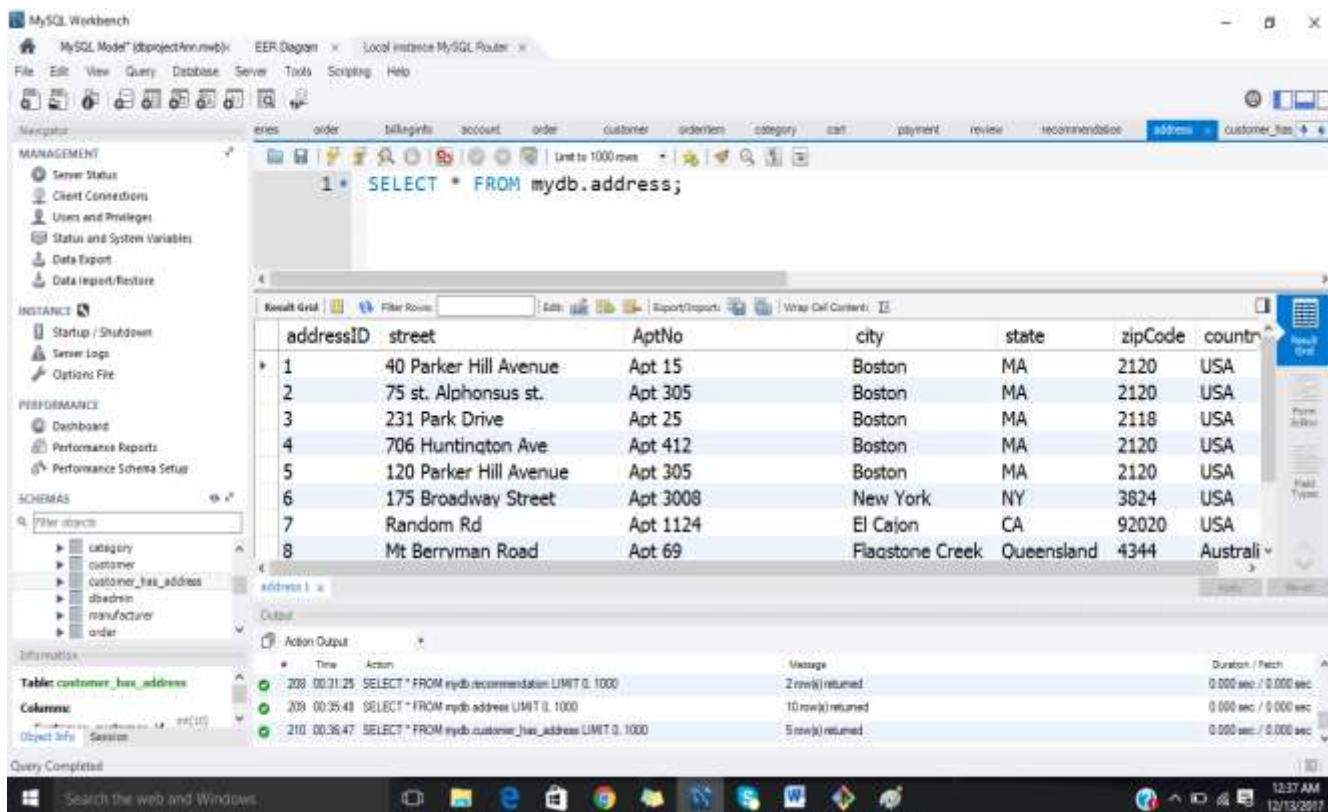
The screenshot shows the MySQL Workbench interface with the 'recommendation' table selected in the top menu. The SQL editor contains the following queries:

```
1 • SELECT * FROM mydb.recommendation;  
2 • alter table recommendation  
3 add column categoryID varchar(20) references category(categoryID);  
4 • INSERT INTO recommendation  
5 values(1,'Similar to what you purchased',1,1);  
6 • INSERT INTO recommendation  
7 values(2,'Similar laptop',6,3);
```

The 'Result Grid' shows the data from the 'recommendation' table:

CustomerID	Description	ProductID	categoryID
1	Similar to what you purchased	1	1
2	Similar laptop	6	3

10. ADDRESS: Customer addresses are placed in separate column for normalization of data



The screenshot shows the MySQL Workbench interface with the 'address' table selected in the top menu. The SQL editor contains the query:

```
1 • SELECT * FROM mydb.address;
```

The 'Result Grid' shows the data from the 'address' table:

addressID	street	AptNo	city	state	zipCode	country
1	40 Parker Hill Avenue	Apt 15	Boston	MA	2120	USA
2	75 st. Alphonsus st.	Apt 305	Boston	MA	2120	USA
3	231 Park Drive	Apt 25	Boston	MA	2118	USA
4	706 Huntington Ave	Apt 412	Boston	MA	2120	USA
5	120 Parker Hill Avenue	Apt 305	Boston	MA	2120	USA
6	175 Broadway Street	Apt 3008	New York	NY	3824	USA
7	Random Rd	Apt 1124	El Cajon	CA	92020	USA
8	Mt Berryman Road	Apt 69	Flagstone Creek	Queensland	4344	Australi

The 'Action Output' pane at the bottom shows the execution of the query:

Time	Action	Message	Duration / Fetch
208 00:31:25	SELECT * FROM mydb.recommendation LIMIT 0, 1000	2 row(s) returned	0.000 sec / 0.000 sec
209 00:35:43	SELECT * FROM mydb.address LIMIT 0, 1000	10 row(s) returned	0.000 sec / 0.000 sec
210 00:36:47	SELECT * FROM mydb.customer_has_address LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec

11. Customer_has_address: Bridge table to connect the address and customer tables

The screenshot shows the MySQL Workbench interface. The query editor contains the query: `1 * SELECT * FROM mydb.customer_has_address;`. The result grid displays the following data:

Customer_customer_id	address_addressID
1	1
2	2
3	3
4	4
5	5

The left sidebar shows the SCHEMAS pane with the database structure. The table `customer_has_address` is selected. The bottom pane shows the Action Output with the following log:

Time	Action	Message	Duration / Fetch
208 00:21:25	SELECT * FROM mydb.recommendation LIMIT 0, 1000	2 row(s) returned	0.000 sec / 0.000 sec
209 00:25:40	SELECT * FROM mydb.address LIMIT 0, 1000	10 row(s) returned	0.000 sec / 0.000 sec
210 00:36:47	SELECT * FROM mydb.customer_has_address LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec

12. MANUFACTURER: A list of all manufacturers. It has the following attributes:

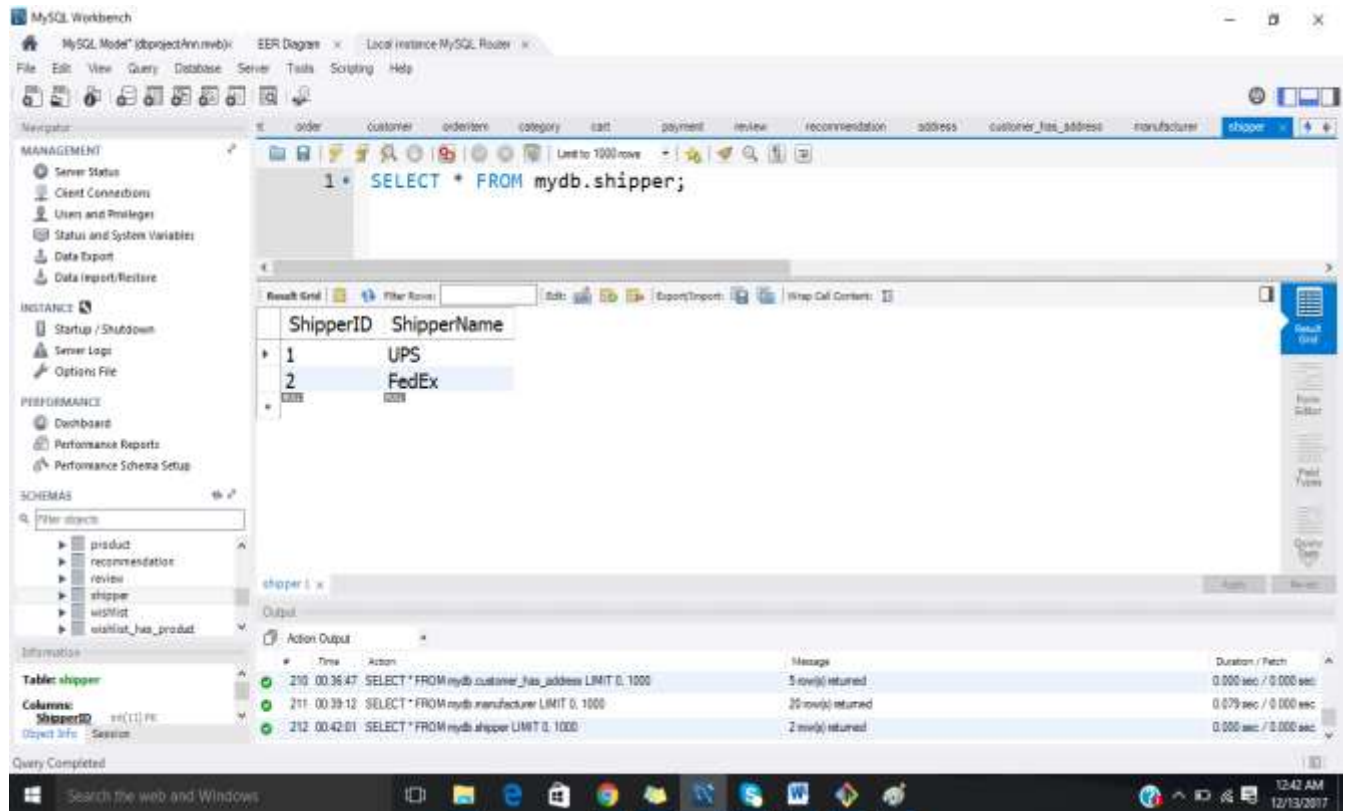
The screenshot shows the MySQL Workbench interface. The query editor contains the query: `1 * SELECT * FROM mydb.manufacturer;`. The result grid displays the following data:

ManufacturerID	ManufacturerName	additionalInfo	category_id
1	Michael Kors	www.michaelkors.com	1
2	Forever 21	www.forever21.com	1
3	H & M	www.hm.com	1
4	Zara	www.zara.com	1
5	L'Oreal	Beauty care	4
6	OLAY	Beauty care	4
7	Maybelline New York	Beauty care	4
8	Garnier Frutis	Beauty care	4
9	Dove	Beauty care	4
10	Apple	Mobile phones	3

The left sidebar shows the SCHEMAS pane with the database structure. The table `manufacturer` is selected. The bottom pane shows the Action Output with the following log:

Time	Action	Message	Duration / Fetch
209 00:35:40	SELECT * FROM mydb.address LIMIT 0, 1000	10 row(s) returned	0.000 sec / 0.000 sec
210 00:36:47	SELECT * FROM mydb.customer_has_address LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec
211 00:39:12	SELECT * FROM mydb.manufacturer LIMIT 0, 1000	20 row(s) returned	0.079 sec / 0.000 sec

13. SHIPPER: A shipper ships the customer its ordered product.

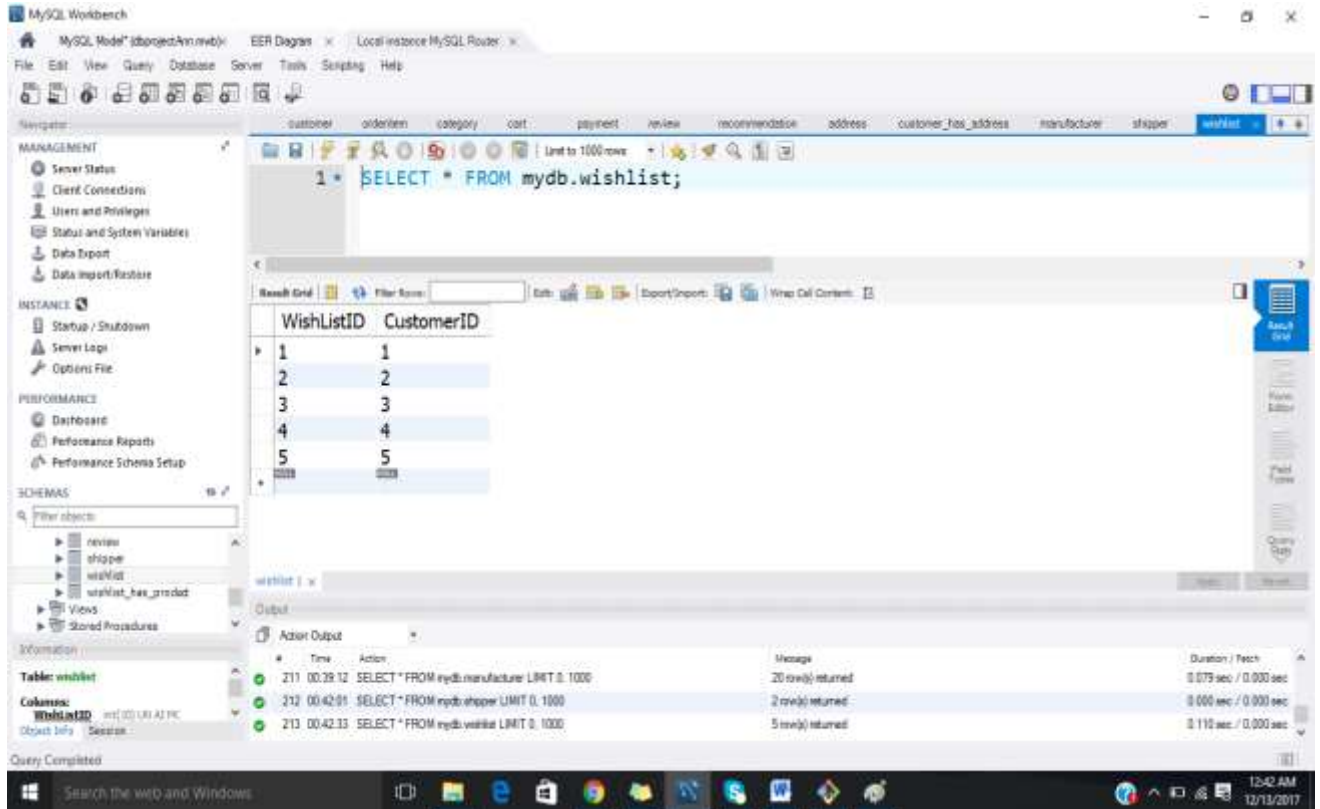


The screenshot shows the MySQL Workbench interface. The query editor at the top contains the SQL statement: `SELECT * FROM mydb.shipper;`. The query has been executed, and the results are displayed in a table with two columns: `ShipperID` and `ShipperName`. The results show two rows: `1` for `UPS` and `2` for `FedEx`. The left sidebar shows the database schema, with the `mydb` database selected. The bottom status bar indicates that the query was completed successfully.

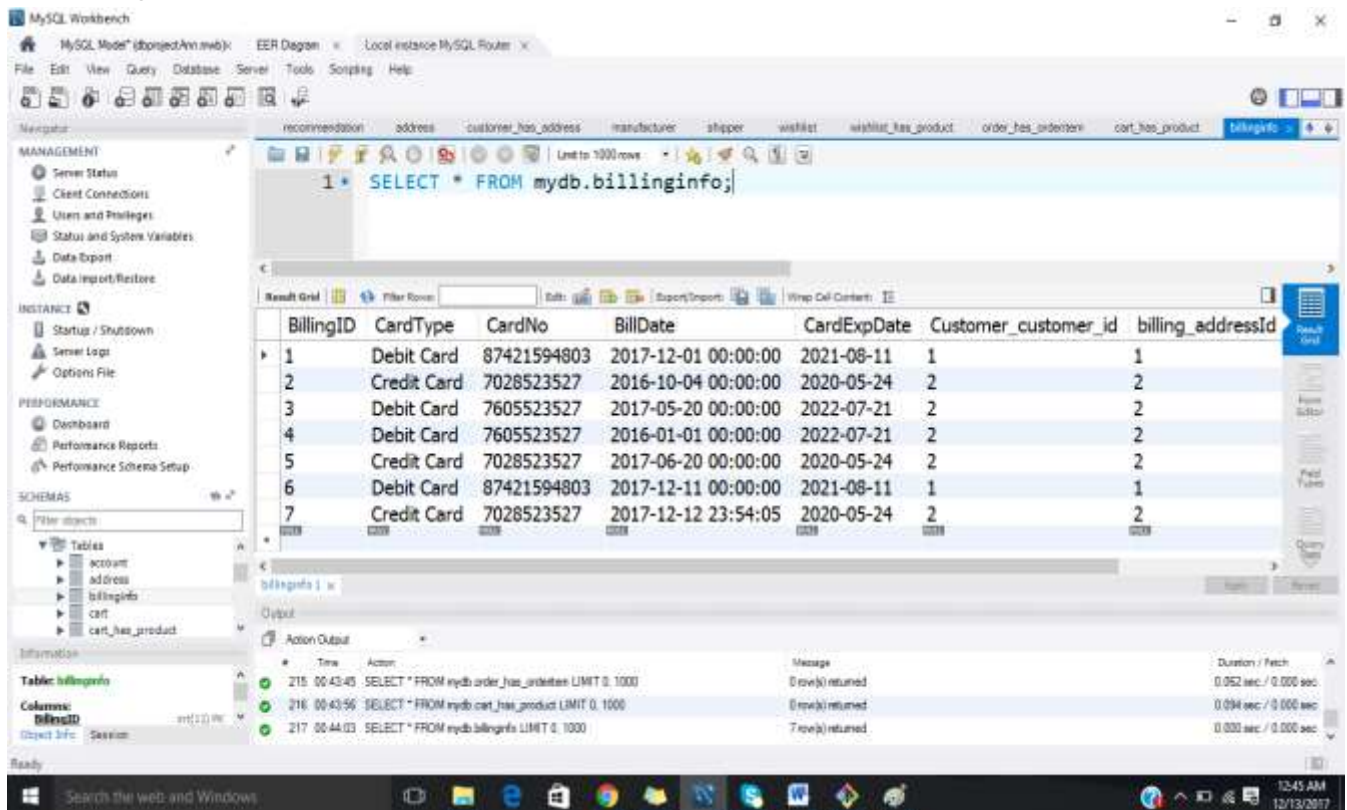
ShipperID	ShipperName
1	UPS
2	FedEx

Query Completed

14. WISHLIST: Every customer has a wishlist where he/she can add products



15. BILLINGINFO: There is a separate table for billingInfo about each customer. It has the following attributes:



ANALYTICAL QUERIES

1. Highest order placed

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

```
1 /*Top order*/
2 select mydb.order.OrderID,
3 concat_ws(' ', customer.firstName,customer.lastName) as CustomerName,
4 mydb.order.OrderAmount as OrderAmount
5 from mydb.order inner join customer
6 where customer.customer_id = mydb.order.customer_customer_id1
7 group by OrderID
8 order by OrderAmount desc
9 limit 1;
```

The results grid displays the following data:

OrderID	CustomerName	OrderAmount
3	Teby Thomas	225

The status bar at the bottom indicates "Query Completed".

2. Top orders

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

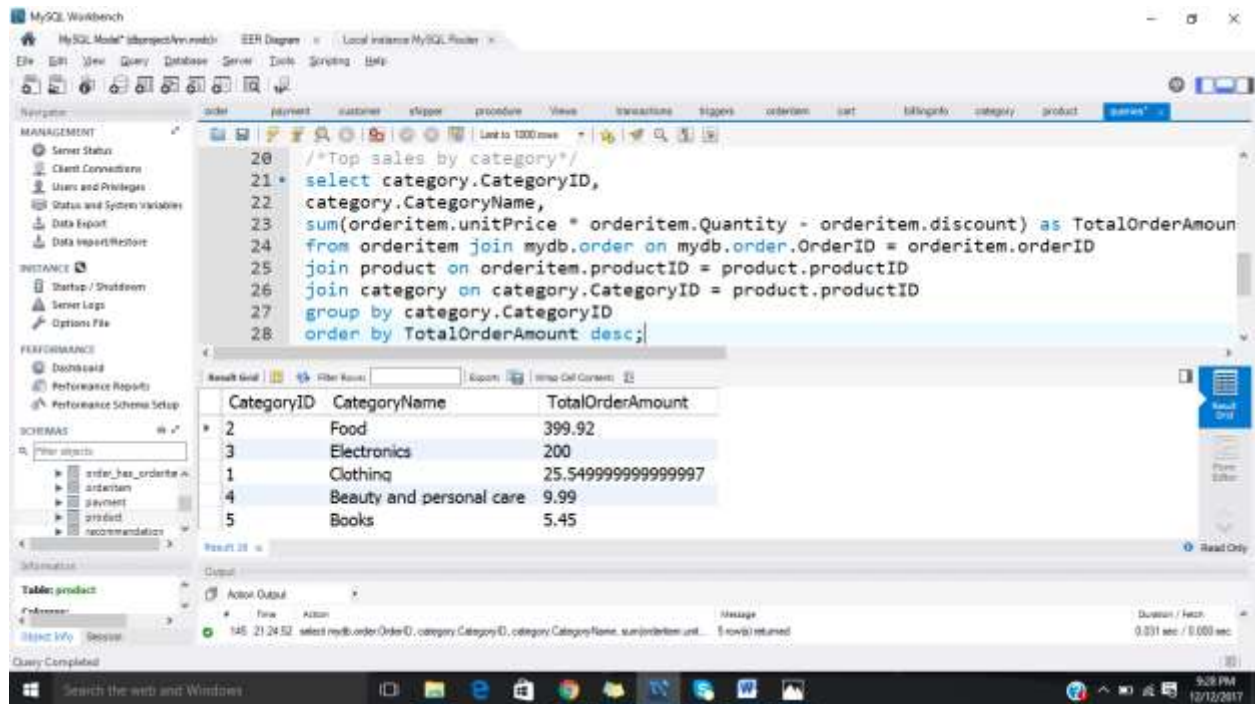
```
7 /*Top 10 orders*/
8 select mydb.order.OrderID,
9 concat_ws(' ', customer.firstName,customer.lastName) as CustomerName,
10 mydb.order.OrderAmount as OrderAmount
11 from mydb.order inner join customer
12 where customer.customer_id = mydb.order.customer_customer_id1
13 group by OrderID
14 order by OrderAmount desc;
```

The results grid displays the following data:

OrderID	CustomerName	OrderAmount
3	Teby Thomas	225
5	Teby Thomas	86.66
2	Teby Thomas	75.85
6	Ann Sajee	74.99
1	Ann Sajee	25.55
4	Teby Thomas	15.18

The status bar at the bottom indicates "Query Completed".

3. Top sales by category



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

```
20 /*Top sales by category*/
21 select category.CategoryID,
22        category.CategoryName,
23        sum(orderitem.unitPrice * orderitem.Quantity - orderitem.discount) as TotalOrderAmount
24 from orderitem join mydb.order on mydb.order.OrderID = orderitem.orderID
25 join product on orderitem.productID = product.productID
26 join category on category.CategoryID = product.productID
27 group by category.CategoryID
28 order by TotalOrderAmount desc;
```

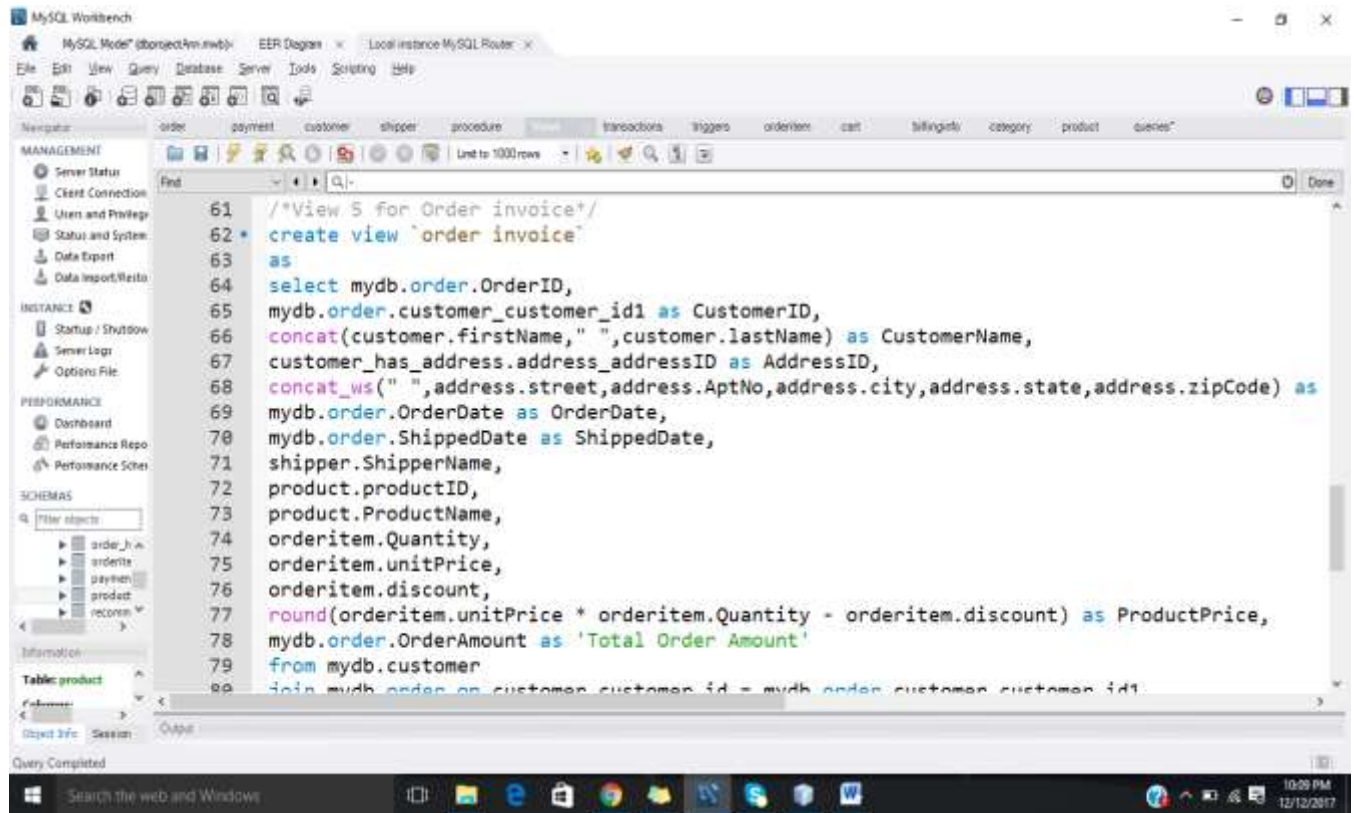
The results pane displays the following data:

CategoryID	CategoryName	TotalOrderAmount
2	Food	399.92
3	Electronics	200
1	Clothing	25.549999999999997
4	Beauty and personal care	9.99
5	Books	5.45

The status bar at the bottom indicates that the query was completed successfully, returning 5 rows in 0.031 seconds.

VIEWS

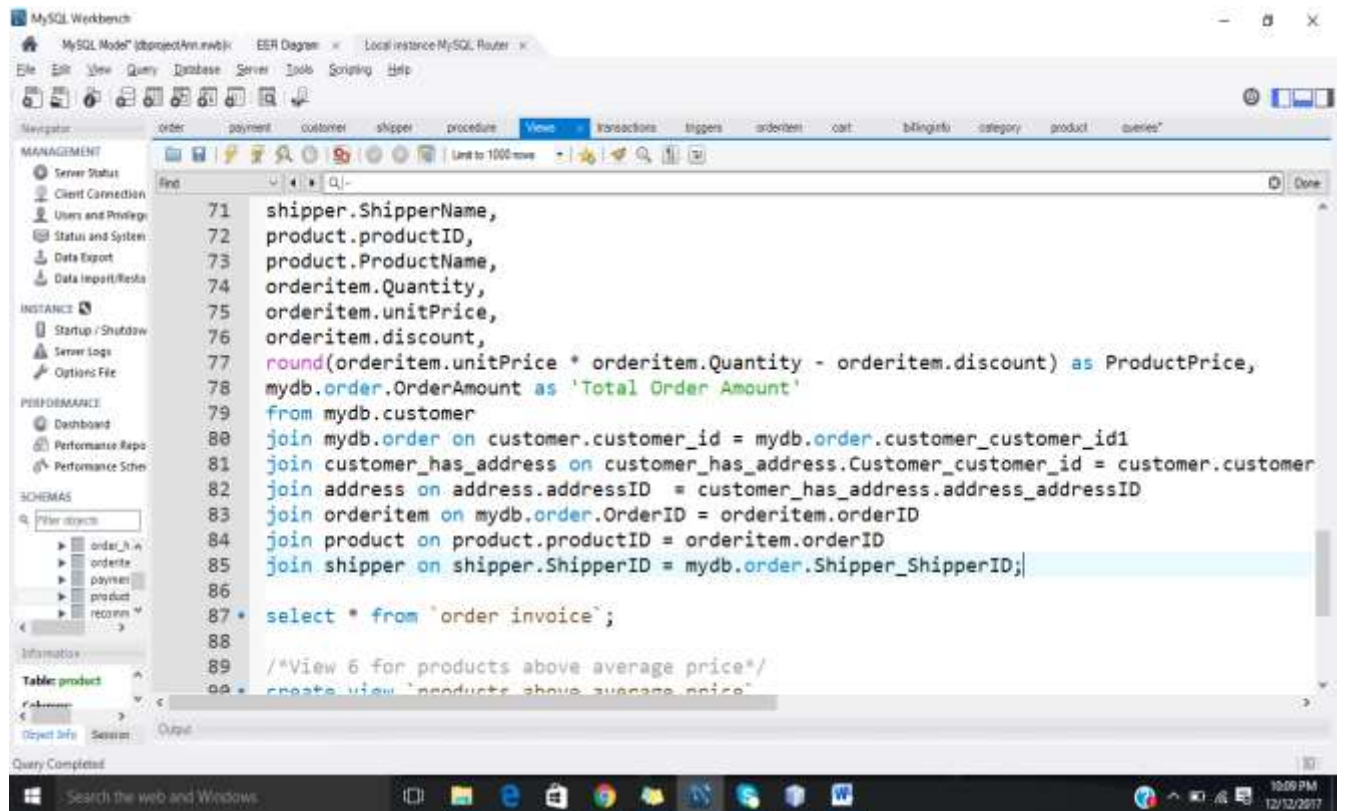
1. Order Invoice



The screenshot shows the MySQL Workbench interface. The 'Query' tab is active, displaying a SQL script to create a view named 'order invoice'. The script is as follows:

```
61 /*View 5 for Order invoice*/
62 create view `order invoice`
63 as
64 select mydb.order.OrderID,
65 mydb.order.customer_customer_id1 as CustomerID,
66 concat(customer.firstName, " ", customer.lastName) as CustomerName,
67 customer_has_address.address_addressID as AddressID,
68 concat_ws(" ", address.street, address.AptNo, address.city, address.state, address.zipCode) as
69 mydb.order.OrderDate as OrderDate,
70 mydb.order.ShippedDate as ShippedDate,
71 shipper.ShipperName,
72 product.productID,
73 product.ProductName,
74 orderitem.Quantity,
75 orderitem.unitPrice,
76 orderitem.discount,
77 round(orderitem.unitPrice * orderitem.Quantity - orderitem.discount) as ProductPrice,
78 mydb.order.OrderAmount as 'Total Order Amount'
79 from mydb.customer
80 join mydb.order on customer.customer_id = mydb.order.customer_customer_id1;
```

The 'Schemas' pane on the left shows the 'mydb' database selected. The 'Query' pane at the bottom shows the status 'Query Completed'.



The screenshot shows the MySQL Workbench interface. The 'Query' tab is active, displaying a SQL script to create a view named 'products above average price'. The script is as follows:

```
71 shipper.ShipperName,
72 product.productID,
73 product.ProductName,
74 orderitem.Quantity,
75 orderitem.unitPrice,
76 orderitem.discount,
77 round(orderitem.unitPrice * orderitem.Quantity - orderitem.discount) as ProductPrice,
78 mydb.order.OrderAmount as 'Total Order Amount'
79 from mydb.customer
80 join mydb.order on customer.customer_id = mydb.order.customer_customer_id1
81 join customer_has_address on customer_has_address.Customer_customer_id = customer.customer
82 join address on address.addressID = customer_has_address.address_addressID
83 join orderitem on mydb.order.OrderID = orderitem.orderID
84 join product on product.productID = orderitem.orderID
85 join shipper on shipper.ShipperID = mydb.order.Shipper_ShipperID;
86
87 select * from `order invoice`;
88
89 /*View 6 for products above average price*/
90 create view `products above average price`
```

The 'Schemas' pane on the left shows the 'mydb' database selected. The 'Query' pane at the bottom shows the status 'Query Completed'.

OUTPUT:

MySQL Workbench interface showing a query result for 'order invoice'. The query is: `select * from order invoice`. The result set contains 11 rows of data.

OrderID	CustomerID	CustomerName	AddressID	CustomerAddress	OrderDate	ShippedDate
1	1	Ann Sajee	1	40 Parker Hill Avenue Apt ...	2017-12-01 16:45:54	2017-12-06 16:45
2	2	Teby Thomas	2	75 st. Alphonsus st. Apt 3...	2016-10-04 23:59:40	2016-10-06 14:00
2	2	Teby Thomas	2	75 st. Alphonsus st. Apt 3...	2016-10-04 23:59:40	2016-10-06 14:00
3	2	Teby Thomas	2	75 st. Alphonsus st. Apt 3...	2017-05-20 08:15:03	2017-05-23 18:30
4	2	Teby Thomas	2	75 st. Alphonsus st. Apt 3...	2016-01-01 00:25:15	2016-01-06 19:45
4	2	Teby Thomas	2	75 st. Alphonsus st. Apt 3...	2016-01-01 00:25:15	2016-01-06 19:45
4	2	Teby Thomas	2	75 st. Alphonsus st. Apt 3...	2016-01-01 00:25:15	2016-01-06 19:45
4	2	Teby Thomas	2	75 st. Alphonsus st. Apt 3...	2016-01-01 00:25:15	2016-01-06 19:45
5	2	Teby Thomas	2	75 st. Alphonsus st. Apt 3...	2017-06-20 10:10:30	2017-06-21 11:45
5	2	Teby Thomas	2	75 st. Alphonsus st. Apt 3...	2017-06-20 10:10:30	2017-06-21 11:45
5	2	Teby Thomas	2	75 st. Alphonsus st. Apt 3...	2017-06-20 10:10:30	2017-06-21 11:45

MySQL Workbench interface showing a query result for 'order invoice'. The query is: `select * from order invoice`. The result set contains 11 rows of data, including product details.

ShipperName	productID	ProductName	Quantity	unitPrice	discount	ProductPrice	Total Order Amount
UPS	1	Knit Sweater Top	1	29.99	4.44	26	25.55
UPS	2	Mens Jeans	1	49.99	0	50	75.85
UPS	2	Mens Jeans	1	25	0	25	75.85
FedEx	3	Kids Dress	1	225	0	225	225
FedEx	4	Coconut Oil	1	1.75	0	2	15.18
FedEx	4	Coconut Oil	1	1.99	0	2	15.18
FedEx	4	Coconut Oil	1	5.45	0	5	15.18
FedEx	4	Coconut Oil	1	5.99	0	6	15.18
UPS	5	Veggie Rolls	1	25	0	25	86.66
UPS	5	Veggie Rolls	1	49.99	0	50	86.66
UPS	5	Veggie Rolls	1	9.99	0	10	86.66

2. List of active product

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

```
1 /*View 1 for Active Product*/
2 create view `Active Product`
3 as
4 select product.productID, product.ProductName
5 from product
6 where active =1;
7
8 select * from `Active Product`;
```

The Results grid displays the output of the second query, showing a list of active products:

productID	ProductName
1	Knit Sweater Top
2	Mens Jeans
3	Kids Dress
4	Coconut Oil
5	Veggie Rolls

The bottom status bar indicates the query was completed successfully.

3. Products by category

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

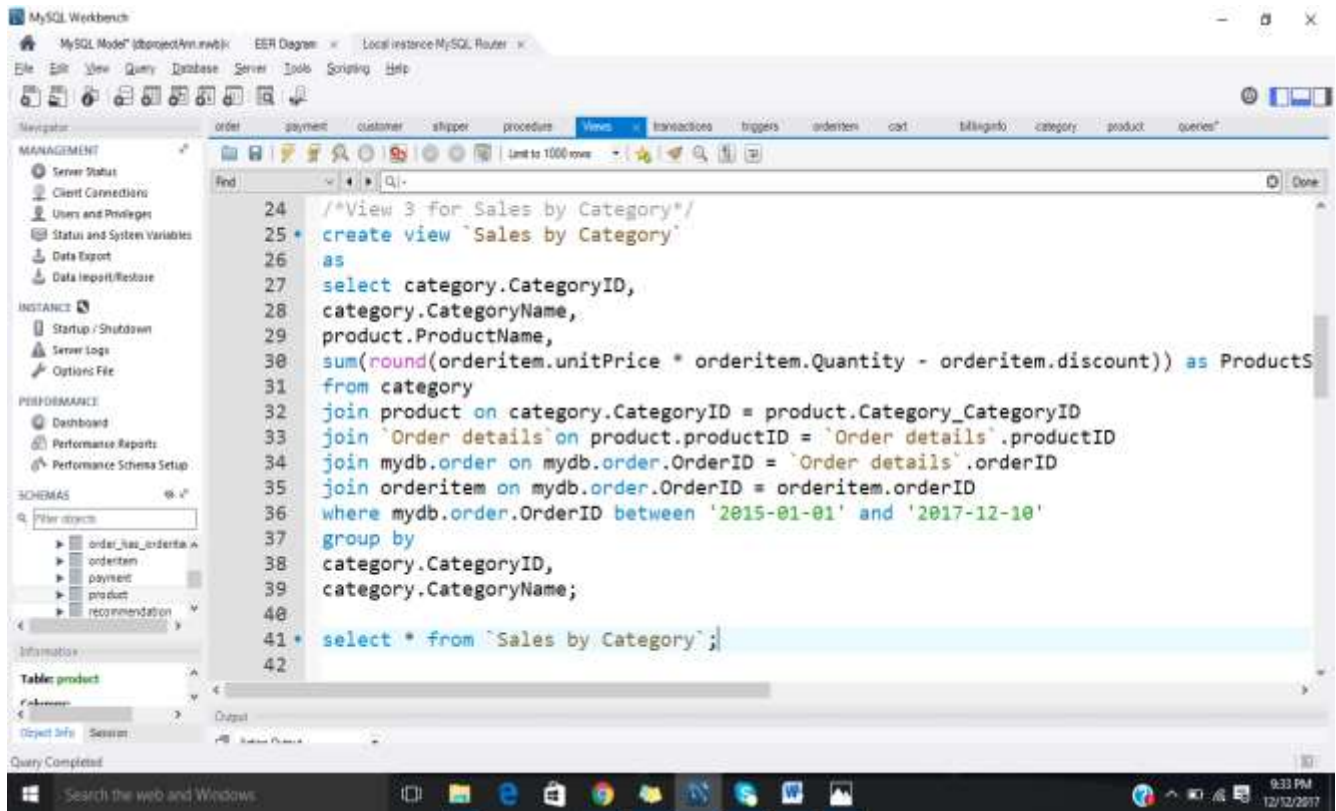
```
10 /*View 2 for Products by Category*/
11 create view `Products by Category`
12 as
13 select category.CategoryName,
14 product.ProductName,
15 product.unitPrice,
16 product.unitsInStock
17 from category inner join product
18 on category.CategoryID = product.Category_CategoryID
19 where product.active = 1;
```

The Results grid displays the output of the second query, showing a list of products by category:

CategoryName	ProductName	unitPrice	unitsInStock
Clothing	Knit Sweater Top	29.99	40
Clothing	Mens Jeans	49.99	50
Clothing	Kids Dress	25	45
Food	Coconut Oil	9.99	25
Food	Veggie Rolls	5.45	55

The bottom status bar indicates the query was completed successfully.

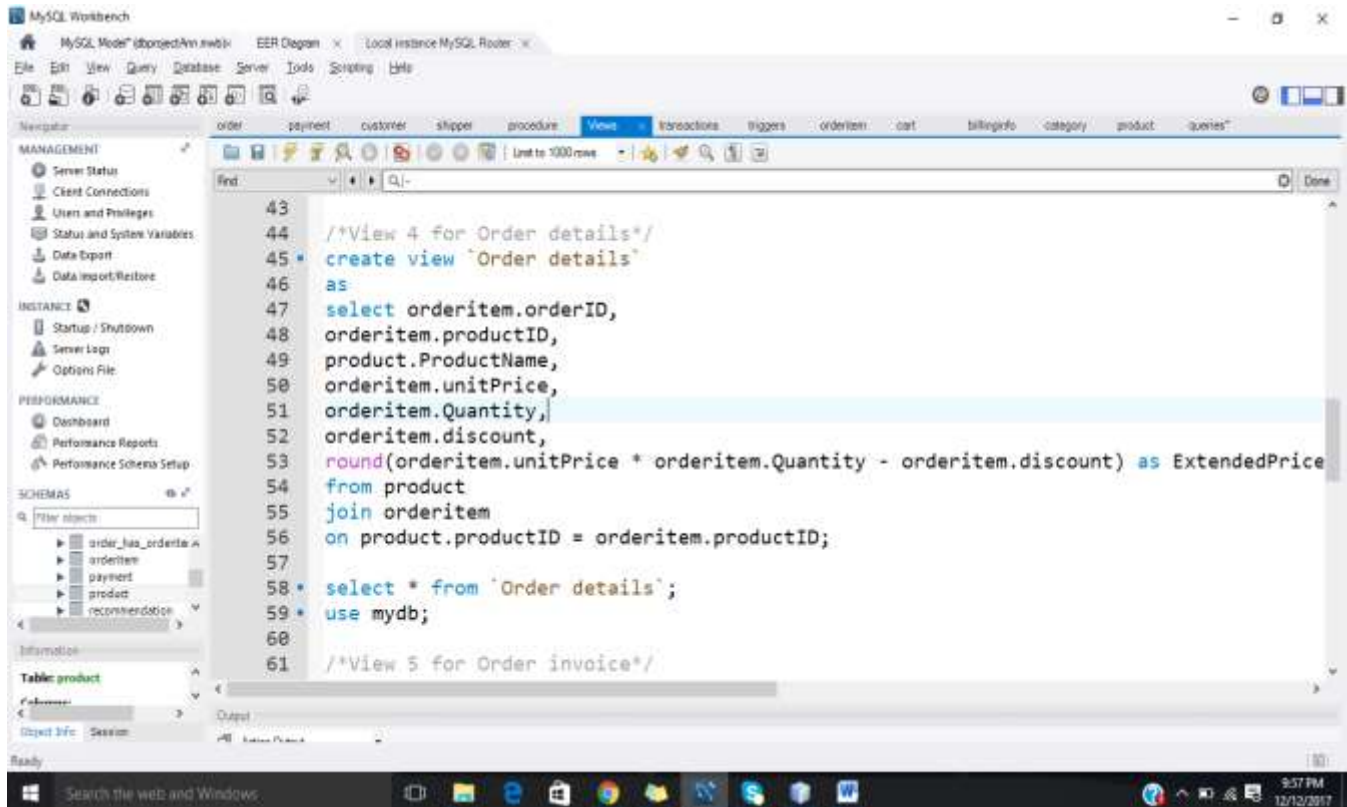
4. Sales by category



The screenshot shows the MySQL Workbench interface. The left sidebar contains the 'MANAGEMENT' and 'SCHEMAS' panels. The 'SCHEMAS' panel shows a tree view of databases, with 'mydb' selected. The main editor window displays a SQL script for creating a view named 'Sales by Category'. The script includes a comment, a 'create view' statement, a 'select' statement with multiple joins and a sum calculation, and a 'select * from' statement to view the data. The status bar at the bottom indicates 'Query Completed'.

```
24 /*View 3 for Sales by Category*/
25 create view `Sales by Category`
26 as
27 select category.CategoryID,
28 category.CategoryName,
29 product.ProductName,
30 sum(round(orderitem.unitPrice * orderitem.Quantity - orderitem.discount)) as Products
31 from category
32 join product on category.CategoryID = product.Category_CategoryID
33 join `Order details` on product.productID = `Order details`.productID
34 join mydb.order on mydb.order.OrderID = `Order details`.orderID
35 join orderitem on mydb.order.OrderID = orderitem.orderID
36 where mydb.order.OrderID between '2015-01-01' and '2017-12-10'
37 group by
38 category.CategoryID,
39 category.CategoryName;
40
41 select * from `Sales by Category`;
42
```

5. Order details



The screenshot shows the MySQL Workbench interface. The left sidebar contains the 'MANAGEMENT' and 'SCHEMAS' panels. The 'SCHEMAS' panel shows a tree view of databases, with 'mydb' selected. The main editor window displays a SQL script for creating a view named 'Order details'. The script includes a comment, a 'create view' statement, a 'select' statement with multiple joins and a sum calculation, and a 'select * from' statement to view the data. The status bar at the bottom indicates 'Ready'.

```
43
44 /*View 4 for Order details*/
45 create view `Order details`
46 as
47 select orderitem.orderID,
48 orderitem.productID,
49 product.ProductName,
50 orderitem.unitPrice,
51 orderitem.Quantity,
52 orderitem.discount,
53 round(orderitem.unitPrice * orderitem.Quantity - orderitem.discount) as ExtendedPrice
54 from product
55 join orderitem
56 on product.productID = orderitem.productID;
57
58 select * from `Order details`;
59 use mydb;
60
61 /*View 5 for Order invoice*/
```

MySQL Workbench

MySQL Model (ibprojectAnn.mwb) EER Diagram Local instance MySQL Router

File Edit View Query Database Server Tools Scripting Help

Navigator order payment customer shipper procedure Views transactions triggers orderitem cart billinginfo category product queries

MANAGEMENT

- Server Status
- Client Connections
- Users and Privileges
- Status and System Variables
- Data Export
- Data Import/Restore

INSTANCE

- Startup / Shutdown
- Server Logs
- Options File

PERFORMANCE

- Dashboard
- Performance Reports
- Performance Schema Setup

SCHEMAS

Filter objects

- order_has_orderitem
- orderitem
- payment
- product
- recommendation

Information

Table: product

Columns

Object Info Session

Query Completed

```

56 on product.productID = orderitem.productID;
57
58 * select * from `Order details`;
59 * use mydb;

```

orderID	productID	ProductName	unitPrice	Quantity	discount	ExtendedPrice
1	1	Knit Sweater Top	29.99	1	4.44	26
2	2	Mens Jeans	49.99	1	0	50
2	3	Kids Dress	25	1	0	25
3	12	Bose Headphones	225	1	0	225
4	13	Garlic Bread	1.75	1	0	2
4	14	Mini Donuts	1.99	1	0	2
4	5	Veggie Rolls	5.45	1	0	5
4	11	Believe You Can	5.99	1	0	6
5	3	Kids Dress	25	1	0	25
5	2	Mens Jeans	49.99	1	0	50
5	4	Coconut Oil	9.99	1	0	10

Order details 10

Output

Query Completed

Search the web and Windows

10:03 PM 12/12/2017

6. Product above average price

MySQL Workbench

MySQL Model (ibprojectAnn.mwb) EER Diagram Local instance MySQL Router

File Edit View Query Database Server Tools Scripting Help

Navigator order payment customer shipper procedure transactions triggers orderitem cart billinginfo category product queries

MANAGEMENT

- Server Status
- Client Connections
- Users and Privileges
- Status and System Variables
- Data Export
- Data Import/Restore

INSTANCE

- Startup / Shutdown
- Server Logs
- Options File

PERFORMANCE

- Dashboard
- Performance Reports
- Performance Schema Setup

SCHEMAS

Filter objects

- order_has_orderitem
- orderitem
- payment
- product
- recommendation
- review

Information

Table: product

Columns

Object Info Session

Query Completed

```

89 /*View 6 for products above average price*/
90 * create view `products above average price`
91 as
92 select product.ProductName,
93 product.unitPrice
94 from product
95 where product.unitsInStock > (select avg(product.unitsInStock) from product);
96 * select * from `products above average price`;

```

ProductName	unitPrice
Knit Sweater Top	29.99
Mens Jeans	49.99
Kids Dress	25
Veggie Rolls	5.45
Bose Headphones	225

products above average price 15

Output

Query Completed

Search the web and Windows

10:14 PM 12/12/2017

STORED PROCEDURES

1. Procedure for Top 10 most expensive products

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

```
1 /*Procedure to get top 10 expensive products*/
2 delimiter $$
3 create procedure `Top 10 most expensive products`()
4 begin
5 select product.ProductName as Top10MostExpensiveProducts,
6 product.unitPrice
7 from product
8 order by product.unitPrice desc
9 limit 10;
10 end
11 $$
12 delimiter ;
13 call `Top 10 most expensive products`();
```

The Results Grid shows the output of the procedure:

Top10MostExpensiveProduct	unitPrice
McBookPro	1600

The Output pane shows the message: "153 22:13:40 select 'from 'products above average price' LIMIT 0, 1000". The status bar indicates "5 row(s) returned" and "0.000 sec / 0.000 sec".

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

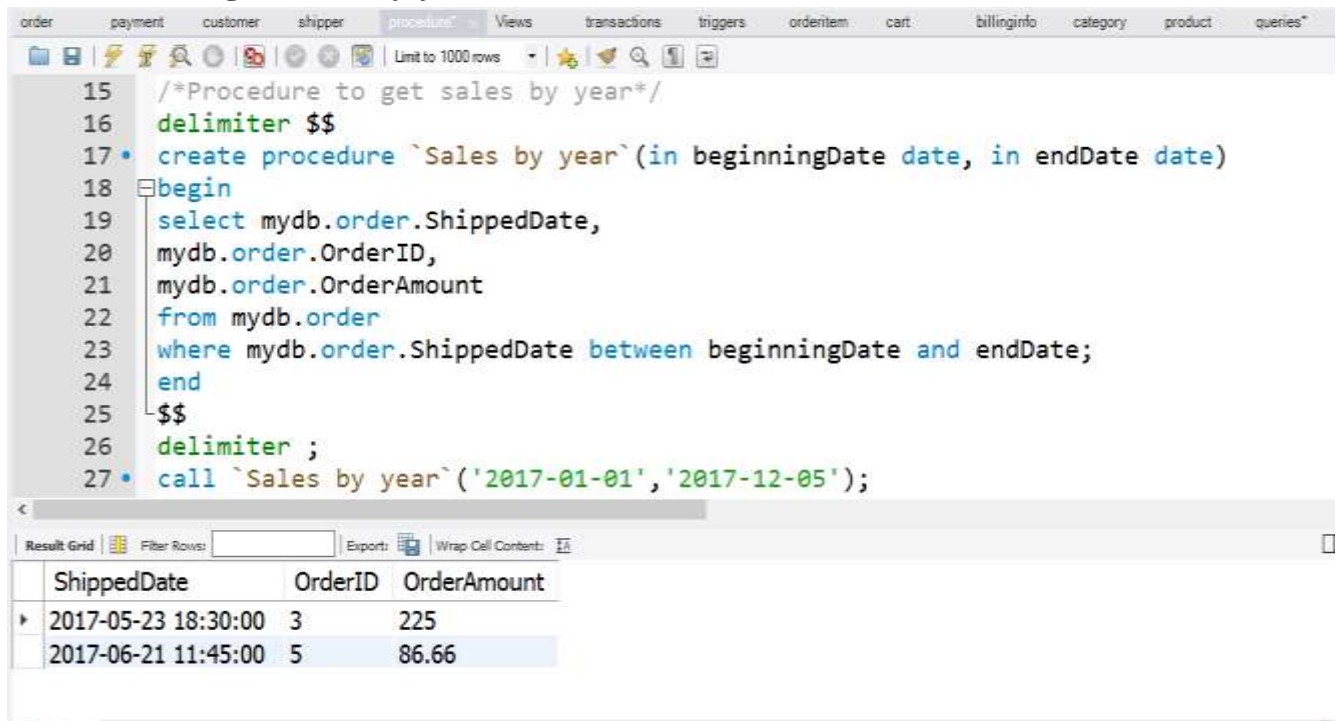
```
10 end
11 $$
12 delimiter ;
13 call `Top 10 most expensive products`();
```

The Results Grid shows the output of the procedure:

Top10MostExpensiveProduct	unitPrice
McBookPro	1600
iPhone 10	999
Dell Inspiron	875.99
HP Spectre	820
Bose Headphones	225
Mens Jeans	49.99
Knit Sweater Top	29.99
Kids Dress	25
Who says you cant? YOU ...	14.99
Coconut Oil	9.99

The Output pane shows the message: "153 22:13:40 select 'from 'products above average price' LIMIT 0, 1000". The status bar indicates "5 row(s) returned" and "0.000 sec / 0.000 sec".

2. Procedure to get Sales by year

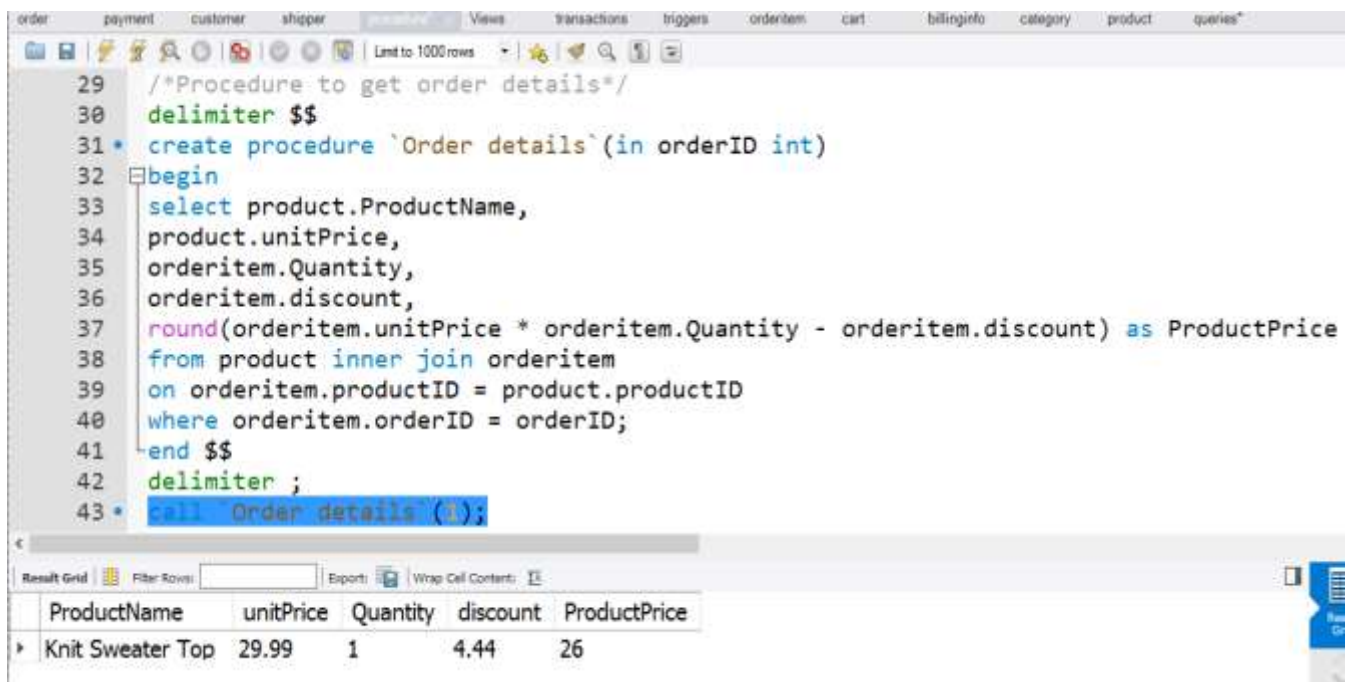


The screenshot shows a SQL IDE with a toolbar at the top containing icons for file operations, search, and execution. Below the toolbar, a list of database objects (order, payment, customer, shipper, procedure, Views, transactions, triggers, orderitem, cart, billinginfo, category, product, queries*) is visible. The main editor displays a SQL script for a procedure named 'Sales by year'. The script starts with a comment, sets a delimiter to '\$\$', and then defines the procedure with two input parameters: 'beginningDate' and 'endDate', both of type 'date'. The procedure body contains a SELECT statement that retrieves 'ShippedDate', 'OrderID', and 'OrderAmount' from the 'mydb.order' table, filtered by a date range. The script ends with a call to the procedure for the dates '2017-01-01' and '2017-12-05'. Below the editor, a 'Result Grid' shows the output of the procedure call, with columns 'ShippedDate', 'OrderID', and 'OrderAmount'. Two rows of data are displayed.

```
15 /*Procedure to get sales by year*/
16 delimiter $$
17 • create procedure `Sales by year`(in beginningDate date, in endDate date)
18 begin
19 select mydb.order.ShippedDate,
20 mydb.order.OrderID,
21 mydb.order.OrderAmount
22 from mydb.order
23 where mydb.order.ShippedDate between beginningDate and endDate;
24 end
25 $$
26 delimiter ;
27 • call `Sales by year`('2017-01-01','2017-12-05');
```

ShippedDate	OrderID	OrderAmount
2017-05-23 18:30:00	3	225
2017-06-21 11:45:00	5	86.66

3. Procedure to get order details



The screenshot shows a SQL IDE with a toolbar at the top. Below the toolbar, a list of database objects (order, payment, customer, shipper, procedure, Views, transactions, triggers, orderitem, cart, billinginfo, category, product, queries*) is visible. The main editor displays a SQL script for a procedure named 'Order details'. The script starts with a comment, sets a delimiter to '\$\$', and then defines the procedure with one input parameter: 'orderID' of type 'int'. The procedure body contains a SELECT statement that retrieves 'productName', 'unitPrice', 'Quantity', 'discount', and 'ProductPrice' from the 'product' and 'orderitem' tables, joined on 'productID'. The 'ProductPrice' is calculated as 'round(orderitem.unitPrice * orderitem.Quantity - orderitem.discount)'. The script ends with a call to the procedure for the orderID '1'. Below the editor, a 'Result Grid' shows the output of the procedure call, with columns 'ProductName', 'unitPrice', 'Quantity', 'discount', and 'ProductPrice'. One row of data is displayed.

```
29 /*Procedure to get order details*/
30 delimiter $$
31 • create procedure `Order details`(in orderID int)
32 begin
33 select product.ProductName,
34 product.unitPrice,
35 orderitem.Quantity,
36 orderitem.discount,
37 round(orderitem.unitPrice * orderitem.Quantity - orderitem.discount) as ProductPrice
38 from product inner join orderitem
39 on orderitem.productID = product.productID
40 where orderitem.orderID = orderID;
41 end $$
42 delimiter ;
43 • call `Order details`(1);
```

ProductName	unitPrice	Quantity	discount	ProductPrice
Knit Sweater Top	29.99	1	4.44	26

4. Procedure to get customer order history

```
order  payment  customer  shipper  procedure* Views  transactions  triggers  orderitem  cart  billinginfo  category  product  queries*
Limit to 1000 rows

44
45  /*Procedure to get Customer Orders history*/
46  delimiter $$
47  • create procedure `Customer Orders history`(in customerID int)
48  begin
49      select mydb.order.OrderID,
50      mydb.order.NoOfOrderItems,
51      mydb.order.OrderDate,
52      mydb.order.ShippedDate,
53      mydb.order.OrderAmount
54      from mydb.order
55      where mydb.order.customer_customer_id1 = customerID
56      order by OrderID;
57  end
58  $$
59  delimiter ;
60  • call `Customer Orders history`(1);
61
```

```
order  payment  customer  shipper  procedure* Views  transactions  triggers  orderitem  cart  billinginfo  category  product  queries*
Limit to 1000 rows

55  where mydb.order.customer_customer_id1 = customerID
56  order by OrderID;
57  end
58  $$
59  delimiter ;
60  • call `Customer Orders history`(1);
61
62
63  /*Procedure to get sales by category*/
```

Result Grid Filter Rows: Exports: Wrap Cell Contents: [\[icon\]](#)

	OrderID	NoOfOrderItems	OrderDate	ShippedDate	OrderAmount
▶	1	1	2017-12-01 16:45:54	2017-12-06 16:45:00	25.55
	6	2	2017-12-11 22:50:00	2017-12-13 00:00:00	74.99

5. Sales by category

```
order  payment  customer  shipper  orderitem  Views  transactions  triggers  orderitem  cart  billinginfo  category  product  queries*
Limit to 1000 rows

63  /*Procedure to get sales by category*/
64  DELIMITER $$
65  * CREATE PROCEDURE `SalesByCategory`(IN AtCategoryName VARCHAR(15), IN AtOrdYear VARCHAR(4))
66  BEGIN
67      SELECT
68          ProductName,
69          ROUND(SUM(orderitem.Quantity * orderitem.unitPrice * orderitem.discount)) AS TotalPurcl
70  FROM orderitem
71      INNER JOIN mydb.order USING (OrderID)
72      INNER JOIN product USING (productID)
73      INNER JOIN category USING (CategoryID)
74  WHERE category.CategoryName = AtCategoryName
75      AND YEAR(mydb.order.OrderDate) = AtOrdYear
76  GROUP BY ProductName
77  ORDER BY ProductName;
78  END $$
79  DELIMITER ;
80  * call `SalesByCategory`('Clothing',2017);
81
```


TRIGGERS

1. Trigger to Update product Quantity

```
order payment customer shipper procedure Views transactions triggers orderitem cart billinginfo category product queries*
Limit to 1000 rows
1  /*trigger 1 for Update Product Quantity*/
2  • create trigger `Update Product Quantity`
3  AFTER Insert
4  ON payment
5  FOR EACH ROW
6  update product JOIN orderitem
7  ON orderitem.productID = product.productID
8  JOIN payment ON payment.orderID = orderitem.orderID
9  set unitsInStock = unitsInStock - orderitem.Quantity;
10
11
12  /*trigger 2 to check Product Quantity available*/
13  DELIMITER $$
14  • CREATE TRIGGER `Check_product_qty_availability`
15  before insert ON `orderitem`
16  FOR EACH ROW
17  BEGIN
```

2. Trigger to check product quantity availability

```
order payment customer shipper procedure Views transactions triggers orderitem cart billinginfo category product queries*
Limit to 1000 rows
12  /*trigger 2 to check Product Quantity available*/
13  DELIMITER $$
14  • CREATE TRIGGER `Check_product_qty_availability`
15  before insert ON `orderitem`
16  FOR EACH ROW
17  BEGIN
18      IF (orderitem.Quantity <= product.unitsInStock)
19      THEN
20          insert into orderitem(orderID,productID,unitPrice,Quantity,discount)
21          values(new.orderID,new.productID,new.unitPrice,new.Quantity,new.discount);
22      END IF;
23  END$$
24  DELIMITER ;
25
26  /*trigger 3 to update Balance before Payment*/
```


3. Trigger to deduct amount from account of customer before payment

```
order  payment  customer  shipper  procedure  Views  transactions  trigger  orderitem  cart  billinginfo  category  product  queries*
Limit to 1000 rows

26  /*trigger 3 to update Balance before Payment*/
27  DELIMITER $$
28  CREATE TRIGGER updateBalancebeforePayment
29  before insert ON `payment`
30  FOR EACH ROW
31  BEGIN
32      if(mydb.order.OrderAmount <= billinginfo.Balance)
33      then
34          update billinginfo join customer
35          on customer.customer_id = billinginfo.Customer_customer_id
36          join mydb.order on customer.customer_id = mydb.order.customer_customer_id1
37          set Balance = Balance - mydb.order.OrderAmount;
38          insert into billinginfo(BillingID,CardType,CardNo,BillDate,CardExpDate,
39          Customer_customer_id,billing_addressId, Balance,orderId)
40          values(7,'Credit Card','7028523527',now(),'2020-05-24',2,2,
41          Balance - mydb.order.OrderAmount,7);
42      end if;
43  END$$
44  DELIMITER ;
45
```

TRANSACTIONS

When the transaction is executed, the insert on order and payment is done if the transaction is committed.

A trigger is used to check availability before insert into order and a trigger is used to update product's unitInStock and unitsOnOrder.

A trigger is also used to update balance of the card of the customer.

```
order  payment  customer  shipper  procedure  Views  transactions  triggers  orderitem  cart  billinginfo  category  product  queries*
Limit to 1000 rows

5
6 • start transaction;
7 • begin;
8 • insert into mydb.order
9 values(7,now(),adddate(now(),interval 2 day),1,'Speedy delivery',49.99,2,2,2,2,0);
10 • insert into payment
11 values(2,2,7,now(),49.99,'Credit Card');
12 • insert into orderitem
13 values(7,2,49.99,1,0);
14 • update cart
15 set totalItems=0 and totalPrice=0
16 where CustomerID=2;
17 • select 'Payment done successfully! Order placed';
18 • commit;
19
```

MySQL Workbench

MySQL Model (dbproject\mydb) - EER Diagram - Local instance MySQL Router

File Edit View Query Database Server Tools Scripting Help

Navigator

- MANAGEMENT
 - Server Status
 - Client Connections
 - Users and Privileges
 - Status and System Variables
 - Data Export
 - Data Import/Restore
- INSTANCE
 - Startup / Shutdown
 - Server Logs
 - Options File
- PERFORMANCE
 - Dashboard
 - Performance Reports
 - Performance Schema Setup
- SCHEMAS
 - Filter objects
 - manufacturer
 - order
 - order_has_orderitem
 - orderitem
 - payment
 - product
- Information
 - Table: order
 - Columns: 11(12)
 - Object Info: Select

1 • SELECT * FROM mydb.`order`;

Result Grid

OrderID	OrderDate	ShippedDate	NoOfOrderItems	ShippingType	orderAmount	customer_cu
1	2017-12-01 16:45:54	2017-12-06 16:45:00	1	Standard Shipping	25.55	1
2	2016-10-04 23:59:40	2016-10-06 14:00:00	2	Speedy delivery	75.85	2
3	2017-05-20 08:15:03	2017-05-23 18:30:00	1	Standard Shipping	225	2
4	2016-01-01 00:25:15	2016-01-06 19:45:00	4	Standard Shipping	15.18	2
5	2017-06-20 10:10:30	2017-06-21 11:45:00	3	Speedy delivery	86.66	2
6	2017-12-11 22:50:00	2017-12-13 00:00:00	2	Standard Shipping	74.99	1
7	2017-12-12 23:43:37	2017-12-14 23:43:37	1	Speedy delivery	49.99	2

Output

Action Output

Message

7 row(s) returned

Duration / Fetch

0.000 sec / 0.000 sec

Query Completed

170 23:45:39 SELECT * FROM mydb.`order` LIMIT 0, 1000

Search the web and Windows

11:46 PM 12/12/2017

MySQL Workbench

MySQL Model* (dbproject/Am.rmb) | EER Diagram | Local instance MySQL Router |

File Edit View Query Database Server Tools Scripting Help

Navigator: order payment customer shipper procedure Views transactions triggers orderitem cart billinginfo category product queries* order

MANAGEMENT

- Server Status
- Client Connections
- Users and Privileges
- Status and System Variables
- Data Export
- Data Import/Restore

INSTANCE

- Startup / Shutdown
- Server Logs
- Options File

PERFORMANCE

- Dashboard
- Performance Reports
- Performance Schema Setup

SCHEMAS

Filter objects

- manufacturer
- order
- order_has_orderitem
- orderitem
- payment
- product

Information

Table: order

Columns: int(10)

Query: SELECT * FROM mydb.payment;

Result Grid

PaymentID	CustomerID	orderID	DateTime	PaymentAmount	PaymentMethod
1	1	6	2017-12-11 22:50:00	25.55	Debit Card
2	2	7	2017-12-12 15:09:19	49.99	Credit Card

Output

Action Output

Time Action Message Duration / Fetch

170 23:45:39 SELECT * FROM mydb.order LIMIT 5, 1000 7 row(s) returned 0.000 sec / 0.000 sec

Query Completed

Search the web and Windows

11:47 PM 12/12/2017

MySQL Workbench

MySQL Model* (dbproject/Am.rmb) | EER Diagram | Local instance MySQL Router |

File Edit View Query Database Server Tools Scripting Help

Navigator: order payment customer shipper procedure Views transactions triggers orderitem cart billinginfo category product queries* order

MANAGEMENT

- Server Status
- Client Connections
- Users and Privileges
- Status and System Variables
- Data Export
- Data Import/Restore

INSTANCE

- Startup / Shutdown
- Server Logs
- Options File

PERFORMANCE

- Dashboard
- Performance Reports
- Performance Schema Setup

SCHEMAS

Filter objects

- manufacturer
- order
- order_has_orderitem
- orderitem
- payment
- product

Information

Table: billinginfo

Columns: int(10)

Query: SELECT * FROM mydb.billinginfo;

Result Grid

BillingID	CardType	CardNo	BillDate	CardExpDate	Customer_customer_id	billing_addressId	Balance	orderID
1	Debit Card	87421594803	2017-12-01 00:00:00	2021-08-11	1	1	10000	1
2	Credit Card	7028523527	2016-10-04 00:00:00	2020-05-24	2	2	4500	2
3	Debit Card	7605523527	2017-05-20 00:00:00	2022-07-21	2	2	3560	3
4	Debit Card	7605523527	2016-01-01 00:00:00	2022-07-21	2	2	2800	4
5	Credit Card	7028523527	2017-06-20 00:00:00	2020-05-24	2	2	1853	5
6	Debit Card	87421594803	2017-12-11 00:00:00	2021-08-11	1	1	8500	6
7	Credit Card	7028523527	2017-12-12 23:54:05	2020-05-24	2	2	1830.1	7

Output

Action Output

Time Action Message Duration / Fetch

178 23:54:31 SELECT * FROM mydb.billinginfo LIMIT 0, 1000 7 row(s) returned 0.000 sec / 0.000 sec

Query Completed

Search the web and Windows

11:55 PM 12/12/2017

Hence, the following has been used in order to perform different operations on the database:

- 1. Analytical queries**
- 2. Joins**
- 3. Views**
- 4. Stored procedures**
- 5. Date functions**
- 6. Triggers**
- 7. Transaction**

Thank you.