

# **SQL Project – Supermarket Company**

## **BUAN6320.32**

**Sindhu Parthasardi Reddy**  
**Shraddha Gururaj Kodancha**

<b>Activity</b>	<b>Sindhu Parthasardi Reddy</b>	<b>Shraddha Gururaj Kodancha</b>
Prepared Data Model and Created Physical DB	x	x
Loaded Data into Database	x	x
Wrote SQL Queries	x	x
Prepared Report	x	x
Reviewed Report	x	x

## Contents

Relational Data Model .....	4
Assumptions/Notes About Data Entities and Relationships .....	4
Entity-Relationship Diagram .....	5
Physical MySQL Database .....	6
Assumptions/Notes About Data Set .....	6
Screen shot of Physical Database objects.....	6
Data in the Database.....	9
SQL Queries.....	13
SQL Query 1 .....	13
Question:.....	13
Notes/Comments About SQL Query and Results (Include # of Rows in Result).....	13
Translation .....	13
Screen Shot of SQL Query and Results.....	13
SQL Query 2 .....	15
Question.....	15
Notes/Comments About SQL Query and Results (Include # of Rows in Result).....	15
Translation .....	15
Screen Shot of SQL Query and Results.....	15
SQL Query 4 .....	17
Question.....	17
Notes/Comments About SQL Query and Results (Include # of Rows in Result).....	17
Translation .....	17
Screen Shot of SQL Query and Results.....	17
SQL Query 5 .....	19
Question.....	19
Notes/Comments About SQL Query and Results (Include # of Rows in Result).....	19
Translation .....	19
Screen Shot of SQL Query and Results.....	19
SQL Query 7 .....	21
Question.....	21
Notes/Comments About SQL Query and Results (Include # of Rows in Result).....	21
Translation .....	21

Screen Shot of SQL Query and Results.....21

## Relational Data Model

### Assumptions/Notes About Data Entities and Relationships

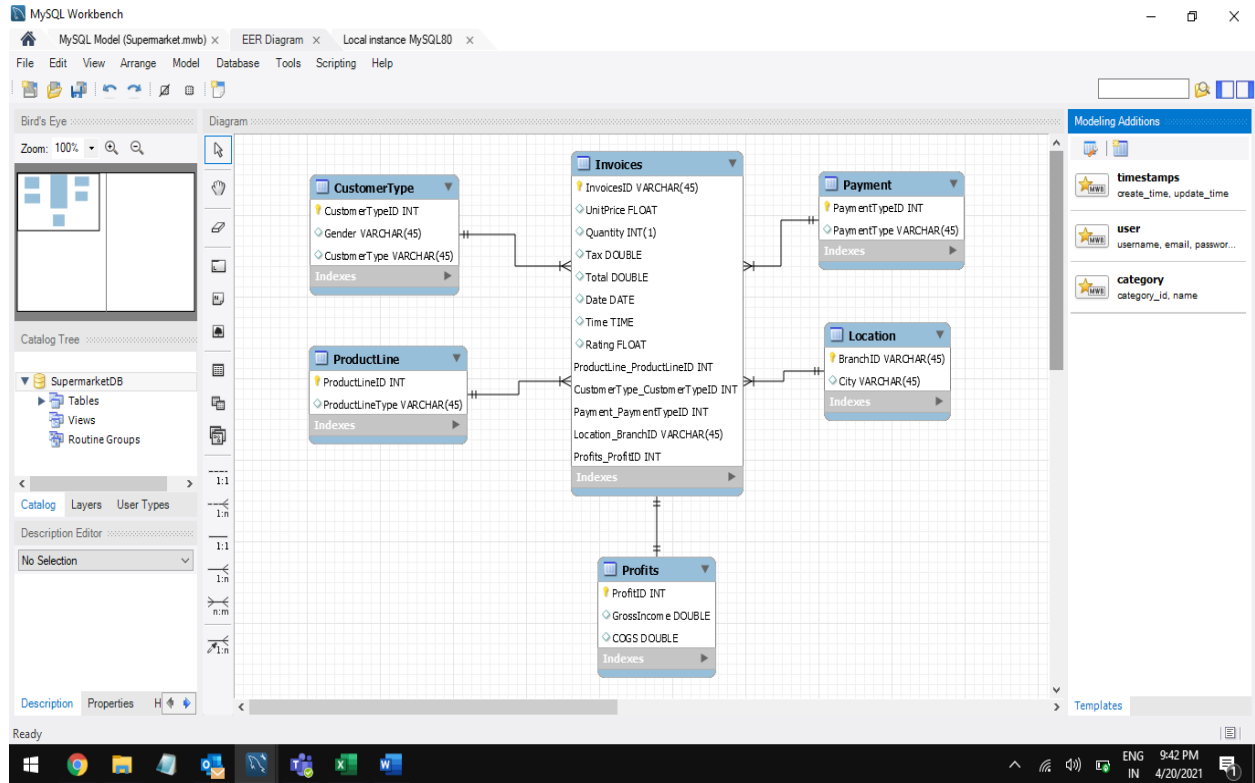
Project assumption:

1. One invoices ID can be associated with only one payment type ID and one payment type ID can be associated with multiple invoice ID. Hence, it is a one-to-many relationship(1:N)
2. One invoices ID can be associated with only one Branch ID and one Branch ID can be associated with multiple invoices ID. Hence, it is a one-to-many relationship(1:N)
3. One invoices ID can be associated with only one product line ID and one product line ID can be associated with multiple invoices ID. Hence, it is a one-to-many relationship(1:N)
4. One invoices ID can be associated with only one customer type ID and one customer type ID can be associated with multiple invoices ID. Hence, it is a one-to-many relationship(1:N)
5. Gross margin percentage column is not included in the data model as it is a constant.

Reasons why the data model is in 3NF:

1. All the rows in the tables have primary keys with unique and non-null values.
2. Every attribute in every table has atomic values (no multi value, no multi part values).
3. Every attribute that is not a part of primary key is functionally dependent on complete primary key.
4. Every non-key column is independent of each other and is only functionally dependent on the entire primary key and not on each other.

## Entity-Relationship Diagram

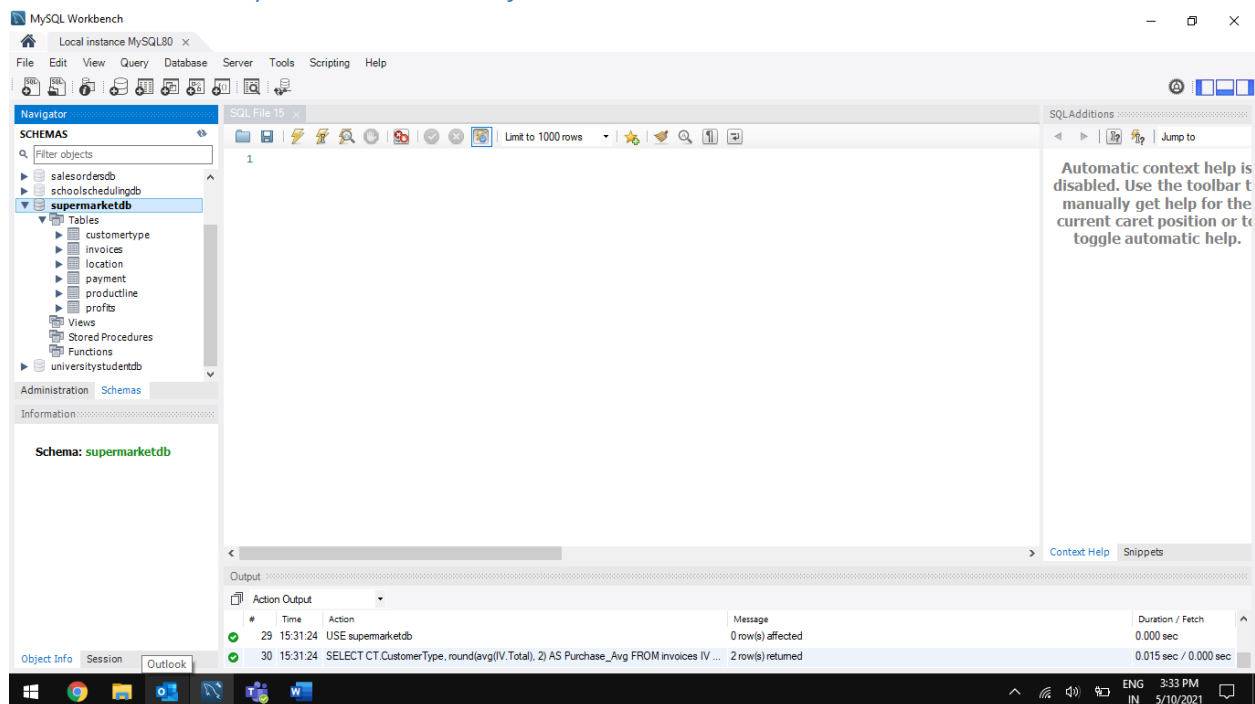


## Physical MySQL Database

### Assumptions/Notes About Data Set

1. The customer Type table has four combinations of customer type with their primary keys defined as below:
  - Member, female = 1001
  - Normal, female = 1002
  - Member, male = 1003
  - Normal, male = 1004
2. The payment Type table has four combinations of payment type with their primary keys defined as below:
  - Ewallet = 101
  - Cash = 102
  - Credit card = 103
3. The product line Type table has four combinations of product line type with their primary keys defined as below:
  - Health and beauty = 1
  - Electronic accessories = 2
  - Home and lifestyle = 3
  - Sports and travel = 4
  - Food and beverages = 5
  - Fashion accessories = 6

### Screen shot of Physical Database objects



## Location Table

The screenshot shows the MySQL Workbench interface. On the left, the 'SCHEMAS' pane lists various databases, with 'supermarketdb' expanded to show its tables: 'customertype', 'invoices', 'location', 'payment', 'productline', and 'profits'. The 'location' table is selected. The main query editor contains the query: `SELECT * FROM supermarketdb.location;`. Below the query editor, the 'Result Grid' shows the table structure with columns 'BranchID' and 'City'. The 'Output' pane at the bottom shows the execution log with two entries: a successful query execution at 21:00:55 and a successful query execution at 21:01:44, both returning 0 rows.

BranchID	City
----------	------

location 1 x

Output

Action Output

#	Time	Action	Message	Duration / Fetch
2	21:00:55	SELECT * FROM supermarketdb.invoices LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec
3	21:01:44	SELECT * FROM supermarketdb.location LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec

## Customer Type Table

The screenshot shows the MySQL Workbench interface. On the left, the 'SCHEMAS' pane lists various databases, with 'supermarketdb' expanded to show its tables: 'customertype', 'invoices', 'location', 'payment', 'productline', and 'profits'. The 'customertype' table is selected. The main query editor contains the query: `SELECT * FROM supermarketdb.customertype;`. Below the query editor, the 'Result Grid' shows the table structure with columns 'CustomerTypeID', 'Gender', and 'CustomerType'. The 'Output' pane at the bottom shows the execution log with one entry: a successful query execution at 21:00:11, returning 0 rows.

CustomerTypeID	Gender	CustomerType
----------------	--------	--------------

customertype 1 x

Output

Action Output

#	Time	Action	Message	Duration / Fetch
1	21:00:11	SELECT * FROM supermarketdb.customertype LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec



## Invoices Table

The screenshot shows the MySQL Workbench interface. The 'Navigator' pane on the left displays the 'supermarketdb' schema with tables: customertype, invoices, location, payment, productline, and profits. The 'Invoices' table is selected. The 'Query' pane shows the query: `SELECT * FROM supermarketdb.invoices;`. The 'Result Grid' shows the table structure with columns: InvoicesID, UnitPrice, Quantity, Tax, Total, Date, Time, Rating, ProductLine\_ProductLineID, CustomerType\_CustomerTypeID, Payment\_PaymentTypeID, and Location\_ID. The 'Output' pane shows the query execution results.

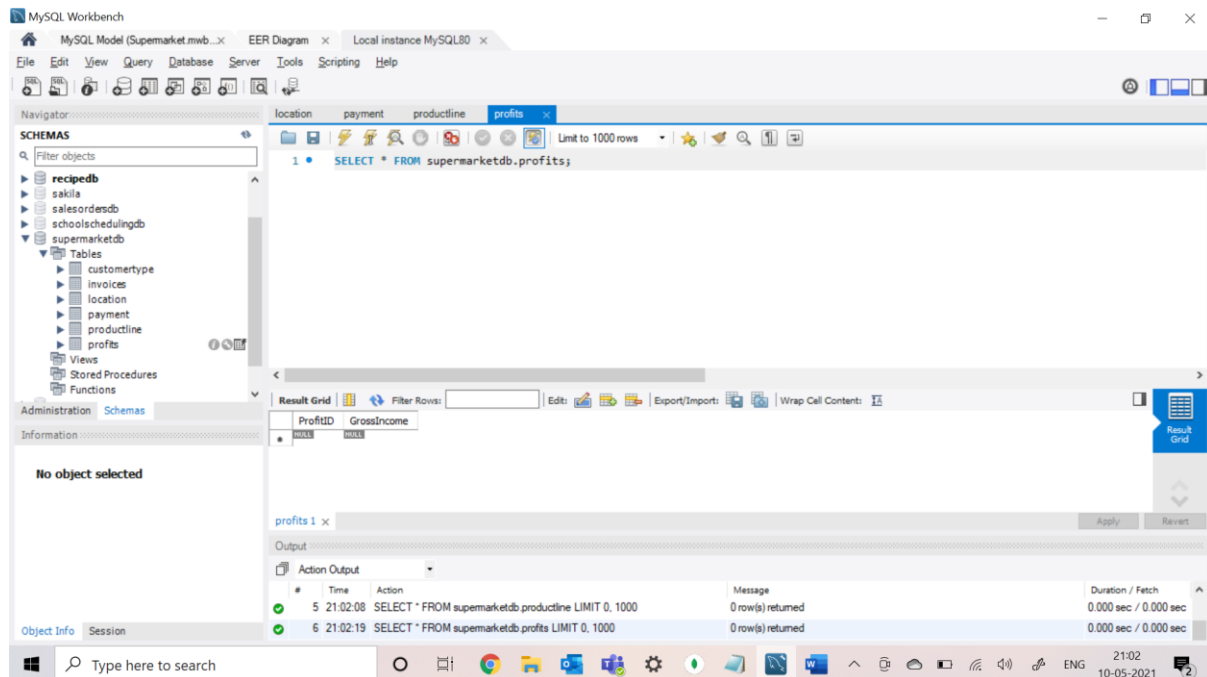
#	Time	Action	Message	Duration / Fetch
1	21:00:11	SELECT * FROM supermarketdb.customertype LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec
2	21:00:55	SELECT * FROM supermarketdb.invoices LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec

## Product Line Details Table

The screenshot shows the MySQL Workbench interface. The 'Navigator' pane on the left displays the 'supermarketdb' schema with tables: customertype, invoices, location, payment, productline, and profits. The 'productline' table is selected. The 'Query' pane shows the query: `SELECT * FROM supermarketdb.productline;`. The 'Result Grid' shows the table structure with columns: ProductLineID and ProductLineType. The 'Output' pane shows the query execution results.

#	Time	Action	Message	Duration / Fetch
4	21:01:55	SELECT * FROM supermarketdb.payment LIMIT 0, 1000	0 row(s) returned	0.016 sec / 0.000 sec
5	21:02:08	SELECT * FROM supermarketdb.productline LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec

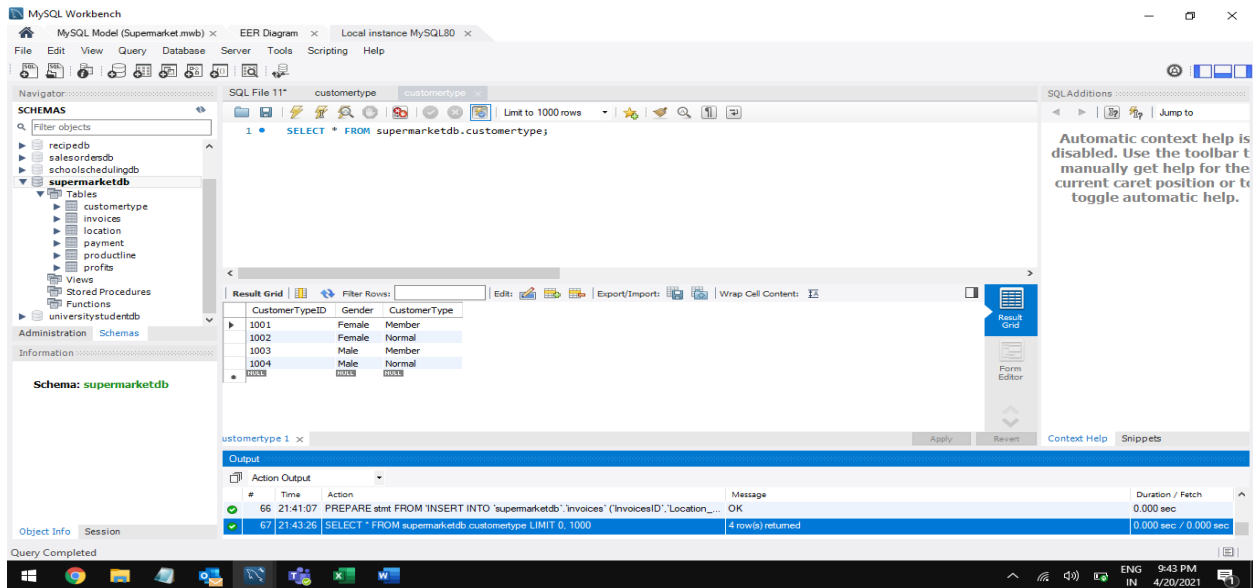
## Profits Table



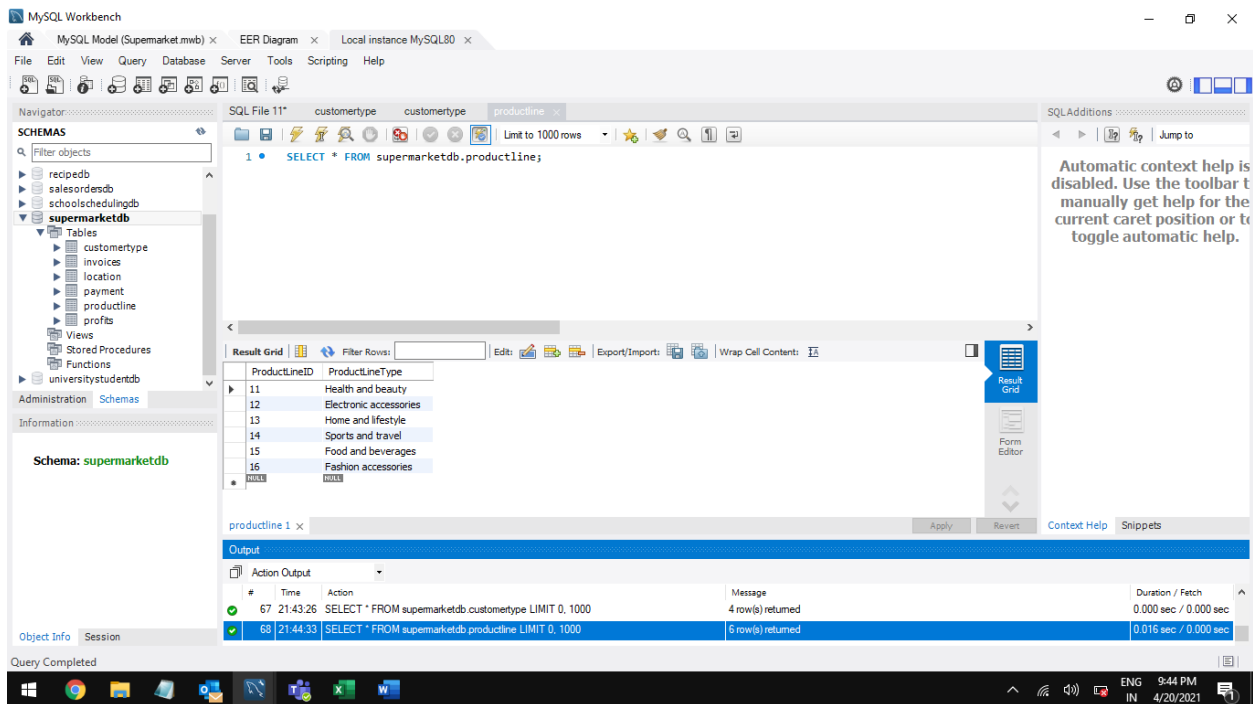
## Data in the Database

Table Name	Primary Key	Foreign Key	# of Rows in Table
CustomerType	CustomerTypeID	-	4
ProductLine	ProductLineID	-	6
Invoices	InvoicesID	ProductLine_ProductLineID CustomerType_CustomerTypeID Payment_PaymentTypeID Location_BranchID Profits_ProfitID	1000
Payment	PaymentTypeID	-	3
Profits	ProfitID	-	1000
Location	BranchID	-	3

CustomerType table loaded in the database:



ProductLine table loaded in the database:



## Invoices table loaded in the database:

MySQL Workbench interface showing the Invoices table loaded in the database. The SQL query is: `SELECT * FROM supermarketdb.invoices;`

The Result Grid displays the following data:

InvoicesID	UnitPrice	Quantity	Tax	Total	Date	Time	Rating	ProductLine_ProductLineID	CustomerType_CustomerType
101-17-6199	45.79	7	16.0265	336.5565	2019-03-13	19:44:00	7	15	1004
101-81-4070	62.82	2	6.282	131.922	2019-01-17	12:36:00	4.9	11	1001
102-06-2002	25.25	5	6.3125	132.5625	2019-03-20	17:52:00	6.1	14	1003
102-77-2261	65.31	7	22.8585	480.0285	2019-03-05	18:02:00	4.2	11	1003
105-10-6182	21.48	2	2.148	45.108	2019-02-27	12:22:00	6.6	16	1003
105-31-1824	69.52	7	24.332	510.972	2019-02-01	15:10:00	8.5	14	1003
106-35-6779	44.34	2	4.434	93.114	2019-03-27	11:26:00	5.8	13	1003
109-28-2512	97.61	6	29.283	614.943	2019-01-07	15:01:00	9.9	16	1001

The Output pane shows the following action output:

#	Time	Action	Message	Duration / Fetch
78	22:07:59	DEALLOCATE PREPARE stmt	OK	0.000 sec
79	22:08:29	SELECT * FROM supermarketdb.invoices LIMIT 0, 1000	1000 row(s) returned	0.031 sec / 0.000 sec

## Payment table loaded in the database:

MySQL Workbench interface showing the Payment table loaded in the database. The SQL query is: `SELECT * FROM supermarketdb.payment;`

The Result Grid displays the following data:

PaymentTypeID	PaymentType
101	Wallet
102	Cash
103	Credit card

The Output pane shows the following action output:

#	Time	Action	Message	Duration / Fetch
68	21:44:33	SELECT * FROM supermarketdb.productline LIMIT 0, 1000	6 row(s) returned	0.016 sec / 0.000 sec
69	21:45:07	SELECT * FROM supermarketdb.payment LIMIT 0, 1000	3 row(s) returned	0.000 sec / 0.000 sec

## Location table loaded in the database:

The screenshot shows the MySQL Workbench interface with the 'location' table selected in the 'supermarketdb' schema. The SQL editor contains the query: `SELECT * FROM supermarketdb.location;`. The 'Result Grid' displays the following data:

BranchID	City
A	Yangon
B	Mandalay
C	Naypyitaw

The 'Output' pane shows the execution log with the following entries:

#	Time	Action	Message	Duration / Fetch
69	21:45:07	SELECT * FROM supermarketdb.payment LIMIT 0, 1000	3 row(s) returned	0.000 sec / 0.000 sec
70	21:45:52	SELECT * FROM supermarketdb.location LIMIT 0, 1000	3 row(s) returned	0.000 sec / 0.000 sec

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

## Profits table loaded in the database:

The screenshot shows the MySQL Workbench interface with the 'profits' table selected in the 'supermarketdb' schema. The SQL editor contains the query: `SELECT * FROM supermarketdb.profits;`. The 'Result Grid' displays the following data:

ProfitID	GrossIncome	COGS
1	26.1415	522.83
2	3.82	76.4
3	16.2155	324.31
4	23.288	465.76
5	30.2085	604.17
6	29.8865	597.73
7	20.652	413.04
8	36.78	735.6
9	3.626	72.52

The 'Output' pane shows the execution log with the following entries:

#	Time	Action	Message	Duration / Fetch
70	21:45:52	SELECT * FROM supermarketdb.location LIMIT 0, 1000	3 row(s) returned	0.000 sec / 0.000 sec
71	21:46:28	SELECT * FROM supermarketdb.profits LIMIT 0, 1000	1000 row(s) returned	0.047 sec / 0.015 sec

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

## SQL Queries

### SQL Query 1

Question:

1. Some retailers believe that there is more money to be made in selling fashion accessories to men than sports and travel to women. Is this true?

Notes/Comments About SQL Query and Results (Include # of Rows in Result)

2. As per the requirement we need three tables to be joined, that is, the invoice table, product line and customer type table.
3. The SQL query returns two rows consisting of total that is made by "male" in "fashion and accessories" and "female" in "sports and travel"
4. From the output, it is clearly seen that total for male in fashion and accessories is 23868.49 and the total for female in sports and travel is 28574.72.
5. Therefore, we conclude that whatever the retailers believe is true and that there is more money to be made from male in fashion and accessories than female in sports and travel.

### Translation

**Translation:** Select the sum of total, gender and production line type from invoices table joining the production line table joining customer type table where gender is Male and product line type is Fashion accessories union with Select the purchase total, gender and production line type from invoices table joining the production line table joining customer type table where gender is female and product line type is Sports and travel

**Clean up:** Select sum of Total as Purchase\_Total, Gender and ProductLineType from invoices join productline on ProductLine\_ProductLineID = ProductLineID join customertype on CustomerType\_CustomerTypeID = CustomerTypeID where Gender = Male and ProductLineType = Fashion accessories union Select sum of Total as Purchase\_Total, Gender and ProductLineType from invoices join productline on ProductLine\_ProductLineID = ProductLineID join customertype on CustomerType\_CustomerTypeID = CustomerTypeID where Gender = Female and ProductLineType = Sports and travel

### Screen Shot of SQL Query and Results

SQL Query:

```
USE supermarketdb;
```

```
SELECT sum(IV.Total) AS Purchase_Total, CT.Gender, PL.ProductLineType
```

```
FROM invoices IV
```

```
JOIN productline PL ON IV.ProductLine_ProductLineID = PL.ProductLineID
```

```
JOIN customertype CT ON IV.CustomerType_CustomerTypeID = CT.CustomerTypeID
```

```
WHERE CT.Gender = "Male" AND PL.ProductLineType = "Fashion accessories"
```

```
UNION
```

SELECT sum(IV.Total) AS Purchase\_Total, CT.Gender, PL.ProductLineType

FROM invoices IV

JOIN productline PL ON IV.ProductLine\_ProductLineID = PL.ProductLineID

JOIN customertype CT ON IV.CustomerType\_CustomerTypeID = CT.CustomerTypeID

WHERE CT.Gender = "Female" AND PL.ProductLineType = "Sports and travel";

The screenshot shows the MySQL Workbench interface. The main window displays a SQL query with the following text:

```
2 USE supermarketdb;
3 SELECT sum(IV.Total) AS Purchase_Total, CT.Gender, PL.ProductLineType
4 FROM invoices IV
5 JOIN productline PL ON IV.ProductLine_ProductLineID = PL.ProductLineID
6 JOIN customertype CT ON IV.CustomerType_CustomerTypeID = CT.CustomerTypeID
7 WHERE CT.Gender = "Male" AND PL.ProductLineType = "Fashion accessories"
8 UNION
9 SELECT sum(IV.Total) AS Purchase_Total, CT.Gender, PL.ProductLineType
10 FROM invoices IV
11 JOIN productline PL ON IV.ProductLine_ProductLineID = PL.ProductLineID
12 JOIN customertype CT ON IV.CustomerType_CustomerTypeID = CT.CustomerTypeID
13 WHERE CT.Gender = "Female" AND PL.ProductLineType = "Sports and travel";
```

The results are displayed in the 'Result Grid' tab, showing two rows of data:

Purchase_Total	Gender	ProductLineType
23868.495000000003	Male	Fashion accessories
28574.720999999998	Female	Sports and travel

The bottom status bar indicates 'Query Completed' and shows the execution time as 0.000 sec / 0.000 sec.

## SQL Query 2

### Question

Some retailers believe that revenue in food and beverages can be increased amongst women by focusing on Ewallets, while others believe eWallets are more popular with men buying electronic accessories. Who is right?

### Notes/Comments About SQL Query and Results (Include # of Rows in Result)

1. As per the requirement we need four tables to be joined, that is, the invoice table, product line type, payment and customer type table.
2. The SQL query returns two rows consisting of total that is made by "female" on "food and beverages" using "Ewallet" payment type and "male" in "Electronic accessories" using "Ewallet" payment type.
3. From the output, it is clearly seen that total for male in electronic accessories using ewallet is 11364.6435 and the total for female in food and beverages using ewallet is 9335.0549.
4. Therefore, we conclude that Ewallets are more popular with men buying electronic accessories than female buying food and beverages.

### Translation

**Translation:** Select the sum of total, gender, payment type and production line type from invoices table joining the production line table joining customer type table joining payment table where gender is Male and product line type is Electronic accessories and payment type is EWallet union with Select the purchase total, gender, payment type and production line type from invoices table joining the production line table joining customer type table joining the payments table where gender is female and product line type is Food and beverages and payment type is EWallet

**Clean up:** Select sum(IV.Total) as Purchase\_Total, Gender, ProductLineType and PaymentType from invoices join productline on ProductLine\_ProductLineID = ProductLineID customertype on CustomerType\_CustomerTypeID = CustomerTypeID join payment on payment\_PaymentTypeID = PaymentTypeID where Gender = Male AND ProductLineType = Electronic accessories AND PaymentType = Ewallet union select sum(IV.Total) as Purchase\_Total, Gender, ProductLineType, PaymentType from invoices join productline on ProductLine\_ProductLineID = ProductLineID join customertype on CustomerType\_CustomerTypeID = CustomerTypeID join payment on Payment\_PaymentTypeID = PaymentTypeID where Gender = Female and ProductLineType = Food and beverages and PaymentType = Ewallet

### Screen Shot of SQL Query and Results

#### SQL Query:

```
USE supermarketdb;
```

```
SELECT sum(IV.Total) AS Purchase_Total, CT.Gender, PL.ProductLineType, PY.PaymentType
```

```
FROM invoices IV
```

```
JOIN productline PL ON IV.ProductLine_ProductLineID = PL.ProductLineID
```



JOIN customertype CT ON IV.CustomerType\_CustomerTypeID = CT.CustomerTypeID

JOIN payment PY ON IV.Payment\_PaymentTypeID = PY.PaymentTypeID

WHERE CT.Gender = "Male" AND PL.ProductLineType = "Electronic accessories" AND PY.PaymentType="Ewallet"

UNION

SELECT sum(IV.Total) AS Purchase\_Total, CT.Gender, PL.ProductLineType, PY.PaymentType

FROM invoices IV

JOIN productline PL ON IV.ProductLine\_ProductLineID = PL.ProductLineID

JOIN customertype CT ON IV.CustomerType\_CustomerTypeID = CT.CustomerTypeID

JOIN payment PY ON IV.Payment\_PaymentTypeID = PY.PaymentTypeID

WHERE CT.Gender = "Female" AND PL.ProductLineType = "Food and beverages" AND PY.PaymentType = "Ewallet";

The screenshot shows the MySQL Workbench interface. The SQL editor contains a query that uses a UNION to combine results for male and female customers. The query is as follows:

```
1 -- 2. Some retailers believe that revenue in food and beverages can be increased amongst women by focusing on Ewallet
2 USE supermarketdb;
3 SELECT sum(IV.Total) AS Purchase_Total, CT.Gender, PL.ProductLineType, PY.PaymentType
4 FROM invoices IV
5 JOIN productline PL ON IV.ProductLine_ProductLineID = PL.ProductLineID
6 JOIN customertype CT ON IV.CustomerType_CustomerTypeID = CT.CustomerTypeID
7 JOIN payment PY ON IV.Payment_PaymentTypeID = PY.PaymentTypeID
8 WHERE CT.Gender = "Male" AND PL.ProductLineType = "Electronic accessories" AND PY.PaymentType="Ewallet"
9 UNION
10 SELECT sum(IV.Total) AS Purchase_Total, CT.Gender, PL.ProductLineType, PY.PaymentType
11 FROM invoices IV
12 JOIN productline PL ON IV.ProductLine_ProductLineID = PL.ProductLineID
13 JOIN customertype CT ON IV.CustomerType_CustomerTypeID = CT.CustomerTypeID
14 WHERE CT.Gender = "Female" AND PL.ProductLineType = "Food and beverages" AND PY.PaymentType = "Ewallet";
```

The Results window shows the output of the query:

Purchase_Total	Gender	ProductLineType	PaymentType
11364.6435	Male	Electronic accessories	Ewallet
9335.045999999997	Female	Food and beverages	Ewallet

The Output window shows the execution details:

#	Time	Action	Message	Duration / Fetch
50	17:43:45	USE supermarketdb	0 row(s) affected	0.000 sec
51	17:43:45	SELECT sum(IV.Total) AS Purchase_Total, CT.Gender, PL.ProductLineType, PY.PaymentType	2 row(s) returned	0.000 sec / 0.000 sec

## SQL Query 4

### Question

Some retailers believe that their members are spending more per purchase while members believe they are spending less per purchase. Who is right?

### Notes/Comments About SQL Query and Results (Include # of Rows in Result)

1. As per the requirement we need two tables to be joined, that is, the invoice table, customer type table.
2. The SQL query returns two rows consisting of Member and Normal customer type with their respective purchase average.
3. From the output, it is clearly seen that purchase average for member customer type is 327.79 and purchase average for normal customer type is 318.12
4. Therefore, we conclude that retailers who believe that members are spending more per purchase are right in making the statement.

### Translation

**Translation:** Select CustomerType, Purchase average from invoices table join customer type table and group by customer type

**Cleanup:** Select CustomerType, round(avg(IV.Total), 2) as Purchase\_Avg from invoices join customertype on CustomerType\_CustomerTypeID = CustomerTypeID group by CT.CustomerType

### Screen Shot of SQL Query and Results

```
USE supermarketdb;
SELECT CT.CustomerType, round(avg(IV.Total), 2) AS Purchase_Avg
FROM invoices IV
JOIN customertype CT ON IV.CustomerType_CustomerTypeID = CT.CustomerTypeID
GROUP BY CT.CustomerType;
```

MySQL Workbench

Local instance MySQL80

File Edit View Query Database Server Tools Scripting Help

Navigator

1 2 4 5 7

Limit to 1000 rows

SQL Additions

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

SCHEMAS

Filter objects

salesordersdb

schoolscheduledb

supermarketdb

customerType

invoices

location

payment

productline

profits

Views

Stored Procedures

Functions

universitystudentdb

Administration Schemas

Information

Schema: supermarketdb

Result Grid

CustomerType	Purchase_Avg
Member	327.79
Normal	318.12

Result 3

Output

Action Output

#	Time	Action	Message	Duration / Fetch
52	17:54:03	USE supermarketdb	0 row(s) affected	0.000 sec
53	17:54:03	SELECT CT.CustomerType, round(avg(IV.Total), 2) AS Purchase_Avg FROM invoices IV JOIN customerType CT ON IV.CustomerType_CustomerTypeID = CT.CustomerTypeID GROUP BY CT.CustomerType;	2 row(s) returned	0.016 sec / 0.000 sec

Query Completed

Object Info Session

Windows Taskbar: ENG IN 5:54 PM 5/10/2021

## SQL Query 5

### Question

Some retailers believe that their male members are bringing in more overall revenue per purchase while others believe female non-members are bringing in more revenue per purchase of fashion accessories. Who is right?

### Notes/Comments About SQL Query and Results (Include # of Rows in Result)

1. As per the requirement we need three tables to be joined, that is, the invoice table, customer type and product line table.
2. The SQL query returns two rows consisting of Member male on fashion accessories and Normal female on fashion accessories with their respective purchase average.
3. From the output, it is clearly seen that purchase average for member male in fashion accessories is 287.21296 and purchase average for normal female in fashion accessories is 312.5457
4. Therefore, we conclude that female non-members are bringing in more revenue per purchase of fashion accessories than male members in fashion accessories. Hence, retailers who believe female non-members are bringing in more revenue per purchase of fashion accessories is right.

### Translation

**Translation:** Select average of total, gender, product line, customer type from invoices table join product line table join customer type table where customer type id is 1003 and product line type is Fashion accessories union select purchase average, gender, product line, customer type from invoices table join product line table join customer type table where customer type id is 1002 and product line type is Fashion accessories.

**Cleanup:** Select avg(IV.Total) AS Purchase\_Avg, Gender, ProductLineType, CustomerType from invoices IV Join productline on ProductLine\_ProductLineID = ProductLineID join customertype on CustomerType\_CustomerTypeID = CT.CustomerTypeID where CustomerTypeID = 1003 and ProductLineType = "Fashion accessories" union select avg(IV.Total) AS Purchase\_Avg, Gender, ProductLineType, CustomerType from invoices join productline on ProductLine\_ProductLineID = ProductLineID join customertype CustomerType\_CustomerTypeID = CustomerTypeID where CustomerTypeID = 1002 and ProductLineType = "Fashion accessories";

### Screen Shot of SQL Query and Results

#### SQL Query:

```
USE supermarketdb;
```

```
SELECT avg(IV.Total) AS Purchase_Avg, CT.Gender, PL.ProductLineType, CT.CustomerType
```

```
FROM invoices IV
```

```
JOIN productline PL ON IV.ProductLine_ProductLineID = PL.ProductLineID
```

```
JOIN customertype CT ON IV.CustomerType_CustomerTypeID = CT.CustomerTypeID
```

```
WHERE CT.CustomerTypeID = 1003 AND PL.ProductLineType = "Fashion accessories"
```

```
UNION
```

```
SELECT avg(IV.Total) AS Purchase_Avg, CT.Gender, PL.ProductLineType, CT.CustomerType
```

FROM invoices IV

JOIN productline PL ON IV.ProductLine\_ProductLineID = PL.ProductLineID

JOIN customertype CT ON IV.CustomerType\_CustomerTypeID = CT.CustomerTypeID

WHERE CT.CustomerTypeID = 1002 AND PL.ProductLineType = "Fashion accessories";

The screenshot shows the MySQL Workbench interface. The left sidebar displays the 'SCHEMAS' tree with 'supermarketdb' selected. The main editor contains a SQL query with comments and a UNION statement. The 'Result Grid' shows two rows of data. The 'Output' pane at the bottom shows the execution progress.

**SQL Query:**

```
1 -- 5. Some retailers believe that their male members are bringing in more overall revenue per purchase while others  
2 -- are bringing in more revenue per purchase of fashion accessories. Who is right?  
3 USE supermarketdb;  
4 SELECT avg(IV.Total) AS Purchase_Avg, CT.Gender, PL.ProductLineType, CT.CustomerType  
5 FROM invoices IV  
6 JOIN productline PL ON IV.ProductLine_ProductLineID = PL.ProductLineID  
7 JOIN customertype CT ON IV.CustomerType_CustomerTypeID = CT.CustomerTypeID  
8 WHERE CT.CustomerTypeID = 1002 AND PL.ProductLineType = "Fashion accessories"  
9 UNION  
10 SELECT avg(IV.Total) AS Purchase_Avg, CT.Gender, PL.ProductLineType, CT.CustomerType  
11 FROM invoices IV  
12 JOIN productline PL ON IV.ProductLine_ProductLineID = PL.ProductLineID  
13 JOIN customertype CT ON IV.CustomerType_CustomerTypeID = CT.CustomerTypeID
```

**Result Grid:**

Purchase_Avg	Gender	ProductLineType	CustomerType
287.2129615384615	Male	Fashion accessories	Member
312.5457857142858	Female	Fashion accessories	Normal

**Output:**

#	Time	Action	Message	Duration / Fetch
54	17:57:27	USE supermarketdb	0 row(s) affected	0.000 sec
55	17:57:27	SELECT avg(IV.Total) AS Purchase_Avg, CT.Gender, PL.ProductLineType, CT.Customer...	2 row(s) returned	0.000 sec / 0.000 sec

## SQL Query 7

### Question

Yangon calls itself the most eWallet-friendly city for health and beauty while Mandalay calls itself a haven for cash spending. Does the data support their claims?

### Notes/Comments About SQL Query and Results (Include # of Rows in Result)

1. As per the requirement we need three tables to be joined, that is, the invoice table, location, payment and product line table.
2. The SQL query returns six rows consisting of Purchase total using Ewallet payment type in Yangon, Mandalay, Naypyitaw city and Purchase total using Cash payment type in Yangon, Mandalay, Naypyitaw city.
3. Therefore, we conclude that Yangon having a purchase total of 5155.4684 is not the most ewallet friendly city for health and beauty as Mandalay has a higher purchase total of 6054.7725. We also conclude that Mandalay with a purchase total of 35339.4614 is not a haven for cash spending as Naypyitaw has a higher purchase total of 43085.8574.

### Translation

**Translation:** Select city, sum of total, payment type from invoices join product line join location join payment where ProductLineType = Health and beauty and PaymentType=Ewallet group by City union select city, sum of total, payment type from invoices join product line join location join payment where PaymentType=Cash group by City.

**Cleanup:** Select sum(IV.Total) as Purchase\_Total, PaymentType, City from invoices join productline on ProductLine\_ProductLineID = ProductLineID join location on Location\_BranchID = BranchID join payment on Payment\_PaymentTypeID = PaymentTypeID where ProductLineType = Health and beauty" and PaymentType=Ewallet group by City union Select sum(IV.Total) as Purchase\_Total, PaymentType, City from invoices join productline on ProductLine\_ProductLineID = ProductLineID join location on Location\_BranchID = BranchID join payment on Payment\_PaymentTypeID = PaymentTypeID where PaymentType=Cash group by City.

### Screen Shot of SQL Query and Results

SQL Query:

USE supermarketdb;

SELECT sum(IV.Total) AS Purchase\_Total, PY.PaymentType, LT.City

FROM invoices IV

JOIN productline PL ON IV.ProductLine\_ProductLineID = PL.ProductLineID

JOIN location LT ON IV.Location\_BranchID = LT.BranchID

JOIN payment PY ON IV.Payment\_PaymentTypeID = PY.PaymentTypeID

WHERE PL.ProductLineType = "Health and beauty" AND PY.PaymentType="Ewallet"

GROUP BY LT.City

UNION

```

SELECT sum(IV.Total) AS Purchase_Total, PY.PaymentType, LT.City

FROM invoices IV

JOIN productline PL ON IV.ProductLine_ProductLineID = PL.ProductLineID

JOIN location LT ON IV.Location_BranchID = LT.BranchID

JOIN payment PY ON IV.Payment_PaymentTypeID = PY.PaymentTypeID

WHERE PY.PaymentType = "Cash"

GROUP BY LT.City;

```

The screenshot shows the MySQL Workbench interface. The SQL editor contains a query that joins the invoices, productline, location, and payment tables to calculate the total purchase amount for each city, filtered by cash payments. The query is as follows:

```

4 FROM invoices IV
5 JOIN productline PL ON IV.ProductLine_ProductLineID = PL.ProductLineID
6 JOIN location LT ON IV.Location_BranchID = LT.BranchID
7 JOIN payment PY ON IV.Payment_PaymentTypeID = PY.PaymentTypeID
8 WHERE PL.ProductLineType = "Health and beauty" AND PY.PaymentType="Ewallet"
9 GROUP BY LT.City
10 UNION
11 SELECT sum(IV.Total) AS Purchase_Total, PY.PaymentType, LT.City
12 FROM invoices IV
13 JOIN productline PL ON IV.ProductLine_ProductLineID = PL.ProductLineID
14 JOIN location LT ON IV.Location_BranchID = LT.BranchID
15 JOIN payment PY ON IV.Payment_PaymentTypeID = PY.PaymentTypeID
16 WHERE PY.PaymentType = "Cash"

```

The Results tab shows the output of the query, which is a table with three columns: Purchase\_Total, PaymentType, and City. The data is as follows:

Purchase_Total	PaymentType	City
5155.468499999999	Ewallet	Yangon
6054.772500000001	Ewallet	Mandalay
4824.918	Ewallet	Naypyitaw
33781.251	Cash	Yangon
35339.461499999999	Cash	Mandalay

The Output tab shows the execution log, indicating that the query was completed successfully and returned 6 rows.