

# **Database Management System (DBMS)**

**Project Name:**  
Order Management Enterprise

**Author:**  
Tirth Bhimani

## TABLE OF CONTENTS

<b>PROJECT DESCRIPTION .....</b>	<b>1</b>
<i>WHY IS ORDER MANAGEMENT SYSTEM IMPORTANT .....</i>	1
<i>WHY DO WE NEED DATABASE .....</i>	1
<i>BENEFICIARIES OF DATABASE .....</i>	2
<i>REQUIREMENTS TO CHECK BEFORE MAKING DATABASE .....</i>	2
<b>ER MODEL.....</b>	<b>4</b>
<i>ER DIAGRAM.....</i>	4
<i>RELATIONAL SCHEMA.....</i>	5
<b>NORMALIZATION AND SCHEMA REFINEMENT .....</b>	<b>6</b>
<i>FUNCTIONAL DEPENDENCIES.....</i>	6
<i>NORMALIZATION.....</i>	8
<i>FINAL RELATIONAL SCHEMA.....</i>	9
<b>MYSQL QUERIES.....</b>	<b>10</b>
<i>TABLE CREATION .....</i>	10
<i>DATA INSERTION .....</i>	14
<i>SHOW TABLE ENTRIES .....</i>	15
<i>TABLE LENGTH .....</i>	20
<i>SQL QUERIES.....</i>	21

# **PROJECT DESCRIPTION**

An order management system (OMS) is a digital way to manage the lifecycle of an order. It tracks all the information and processes, including order entry, inventory management, fulfilment and after-sales service. An OMS offers visibility to both the business and the buyer. Organizations can have near real-time insight into inventories and customers can check when an order will arrive. Order management starts when a customer places an order and ends once they receive their package or service. It allows a business to coordinate the entire fulfilment process.

## **Why is Order Management Important?**

Order management touches virtually every system and process in the supply chain. Most companies no longer contain order management within their organization. They involve multiple partners, such as parts and components suppliers, assembly and packaging services or distribution centres — making it easy to lose visibility and control of an order. This results in costly manual processes to complete and deliver the order without errors. An OMS can help control costs and generate revenue by automating manual processes and reducing errors.

## **Why do we need database?**

With a high volume of orders frequently coming in from many channels, there would be a high error margin if traditional order processing methods are used. Task automation and the centralization of warehouse and inventory data not only takes care of such errors but also reduces chances of data security and loss. Therefore, it can be concluded that usage of the manual maintenance there are a number of difficulties and drawbacks exist in the system.

## **Cons of Existing System (Pen-Paper system or File System):**

- Large volumes of Orders, Products, Customers etc. can not be stored in one place only.
- Multiple Users should not be able to modify data at the same time.
- Manual maintenance is time consuming and can cause many errors.
- While access to text files or spreadsheets can be secured, once someone has access to a file, they have access to all data within that file.

Whenever using a hand-written register or book for entering a particular data, it can have many difficult consequences which cannot be solved practically or if solved then it could be much expensive or time consuming. Thinking from supply chain perspective, if a local shopkeeper at Delhi wants to access a particular data from database of the other branch situated at Mumbai, then if the system uses hand-written registers, then it would be impractical to send the registers from Delhi to Mumbai via any mode of transportation.

The ordering industry mainly works on trust. Whenever a person buys his or her product to any location via the internet then he/she mainly worries whether the purchased product have been confirmed or the size/colour of product to which they are buying by the help of the merchant website whether it would be successful and enjoyable or would turn into ashes. So, with the help of our database, we would like to assure the customers by adding some attributes which ensure whether the products they are buying would be safely delivered to them. We would also think of the host company which is providing the products to the customer that has available storage for the same product and they have completed the payments or not.

## **Beneficiaries of Database**

Customers expect to be informed every step of the way across the buying journey with regular email updates, tracking information, and delivery status alerts. Without order management solutions in place to compile customer information and filter data into one place, there are more opportunities for error.

From buy online/pickup in-store (BOPIS) to same-day shipping, customers expect as much choice in delivery as they do in channels of engagement. Shoppers expect to be able to have an order fulfilled in a way, time, and place that is most convenient to them. Each touchpoint in the journey presents an opportunity to provide great customer experience and boost retention and revenue.

These days it's about complexity, not about size. If you have multiple stores, warehouses, or sales channels and websites in various countries, and you want to try and bring them all together, order management makes sense. With order management, you're able to pull all of those different platforms for different countries together – in one place.

## **What requirements must be checked before making a database?**

For safety requirements, the system shall save the data if the server goes down or any other harmful things happen to the server for the sake of safety. For security purposes, the application will be password protected or we can have an OTP system, i.e., when the user enters the application, the One Time Password (OTP) will be sent to the respective user. The user should be strictly advised that he/she should not share the login details and the same should be applied with the manager and developer. The planning and the logical structure of the database should not be shared with anyone except the trustful. Similarly, the applications made using the database must be accessible using trusted platforms only and not by any third-party apps. The complexity of the system should be kept in such an

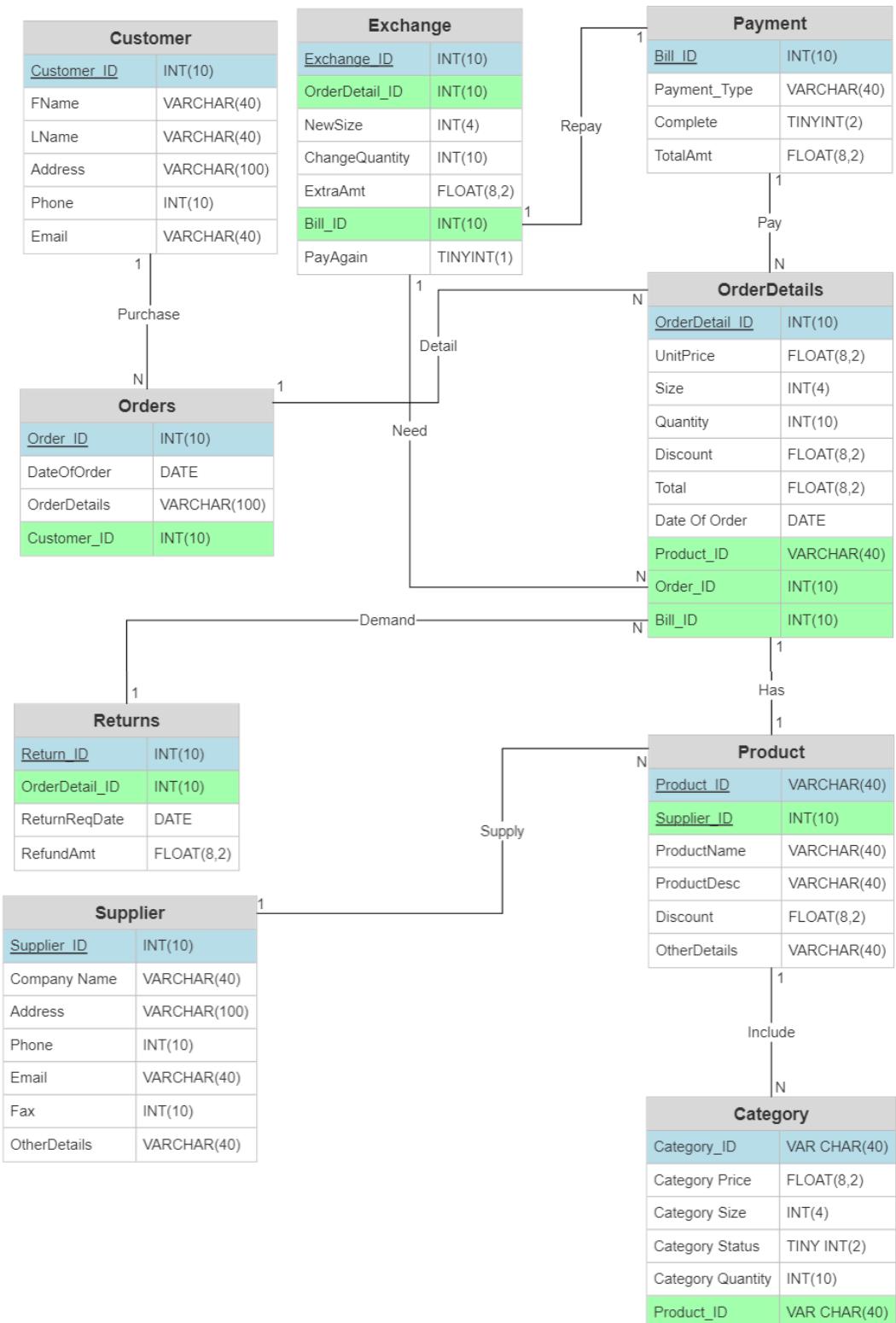
order that whenever we need to manipulate any entity or table data it should be done easily. If we want to insert new data/attributes to a table then it should not create data redundancy or inconsistency in the database. So, the system needs to be made such that it can be maintained whenever needed.

The application will be easily accessible on any OS-based system. For different classes of users, Data requirements will be different. Customers who want to purchase various products will be required to submit one of the id proofs and need to complete their user profile. For seller: Licence of their company will be required. Using Data integrity, users know what data is being collected and who has the right to access it and how they are able to interact with it and so all that information will be inaccessible to unauthorized users and privacy of data is at its fullest. Database should be error-free by storing accurate and consistent data in a database.

# ER MODEL

## ER Diagram

Link of ER Diagram - [Link](#)



Conventions followed in ER diagram:

- Table name is highlighted in the centre of first row in bold.
- Name of attributes are written in the table following the table name.
- Data type of attributes are shown in same row.
- Primary key for the table is underlined and highlighted with blue colour
- Foreign keys for the table are highlighted with green colour
- Relationship between 2 tables is showed along the edge. Type of Relationships:
  - One-to-many Relation (1-N): Customer-Payment, Orders-Orderdetails, Payment-Orderdetails, Exchange-Orderdetails, Supplier-Product, Product-Category, Returns-Orderdetails
  - One-to-One Relation (1-1): Payment-Exchange
  - Many-to-Many Relation (N-M): None
- Name of the relationship is shown in the middle of the edge

## Relational Schema

- Customer (**Customer\_ID**, FName, LName, Address, Phone, Email)
- Orders (**Order\_ID**, DateOfOrder, *Customer\_ID*, OrderDetails)
- OrderDetails (**OrderDetail\_ID**, UnitPrice, Color, Quantity, Discount, Total, Date, *Category\_ID*, *Order\_ID*, *Bill\_ID*)
- Payment (**Bill\_ID**, PaymentType, Complete, TotalAmt)
- Exchange (**Exchange\_ID**, *OrderDetail\_ID*, ChangeQuantity, CategoryColor, NewAmt, *Bill\_ID*, PayAgain, Category\_ID)
- Product (**Product\_ID**, ProductName, ProductDesc, Discount, *Supplier\_ID*, OtherDetails)
- Category (**Category\_ID**, CategoryPrice, CategoryColor, CategoryStatus, CategoryQuantity, *Product\_ID*)
- Supplier (**Supplier\_ID**, CompanyName, Address, Phone, Email, Fax, OtherDetails)
- Returns (**Return\_ID**, *OrderDetail\_ID*, ReturnReqDate)

# NORMALIZATION AND SCHEMA REFINEMENT

## Functional Dependencies

Entity Name	Dependencies
Customer	$\text{Customer\_ID} \rightarrow \text{FName}$ $\text{Customer\_ID} \rightarrow \text{LName}$ $\text{Customer\_ID} \rightarrow \text{Address}$ $\text{Customer\_ID} \rightarrow \text{Phone}$ $\text{Customer\_ID} \rightarrow \text{Email}$  $\text{Phone} \rightarrow \text{Customer\_ID}$ $\text{Phone} \rightarrow \text{FName}$ $\text{Phone} \rightarrow \text{LName}$ $\text{Phone} \rightarrow \text{Address}$ $\text{Phone} \rightarrow \text{Email}$  $\text{Email} \rightarrow \text{Customer\_ID}$ $\text{Email} \rightarrow \text{FName}$ $\text{Email} \rightarrow \text{LName}$ $\text{Email} \rightarrow \text{Address}$ $\text{Email} \rightarrow \text{Phone}$
Order	$\text{Order\_ID} \rightarrow \text{DateOfOrder}$ $\text{Order\_ID} \rightarrow \text{OrderDetails}$ $\text{Order\_ID} \rightarrow \text{Customer\_ID}$
OrderDetails	$\text{OrderDetail\_ID} \rightarrow \text{UnitPrice}$ $\text{OrderDetail\_ID} \rightarrow \text{Color}$ $\text{OrderDetail\_ID} \rightarrow \text{Quantity}$ $\text{OrderDetail\_ID} \rightarrow \text{Discount}$ $\text{OrderDetail\_ID} \rightarrow \text{Total}$ $\text{OrderDetail\_ID} \rightarrow \text{Date}$ $\text{OrderDetail\_ID} \rightarrow \text{Category\_ID}$ $\text{OrderDetail\_ID} \rightarrow \text{Order\_ID}$ $\text{OrderDetail\_ID} \rightarrow \text{Bill\_ID}$  $\text{Category\_ID} \rightarrow \text{Discount}$  $\text{Order\_ID} \rightarrow \text{Date}$ $\text{Order\_ID} \rightarrow \text{Bill\_ID}$
Payment	$\text{Bill\_ID} \rightarrow \text{Payment\_Type}$ $\text{Bill\_ID} \rightarrow \text{Complete}$ $\text{Bill\_ID} \rightarrow \text{TotalAmt}$

Entity Name	Dependencies
Exchange	Exchange_ID -> OrderDetail_ID Exchange_ID -> NewColor Exchange_ID -> ChangeQuantity Exchange_ID -> NewAmt Exchange_ID -> Bill_ID Exchange_ID -> PayAgain  NewAmt -> PayAgain
Product	Product_ID -> ProductName Product_ID -> ProductDesc Product_ID -> Discount Product_ID -> Supplier_ID Product_ID -> OtherDetails  ProductName -> Supplier_ID  ProductDesc -> ProductName ProductDesc -> Discount ProductDesc -> Supplier_ID
Category	Category_ID -> CategoryPrice Category_ID -> CategoryColor Category_ID -> CategoryStatus Category_ID -> CategoryQuantity Category_ID -> Product_ID  CategoryQuantity -> CategoryStatus
Supplier	Supplier_ID -> Name Supplier_ID -> Address Supplier_ID -> Phone Supplier_ID -> Email Supplier_ID -> Fax Supplier_ID -> OtherDetails  Email -> Name Email -> Address Email -> Phone Email -> Fax Email -> OtherDetails Email -> Supplier_ID  Phone -> Name Phone -> Address Phone -> Email

Entity Name	Dependencies
	Phone -> Fax Phone -> OtherDetails Phone -> Supplier_ID  Fax -> Name Fax -> Address Fax -> Phone Fax -> Email Fax -> OtherDetails Fax -> Supplier_ID
Return	Return_ID -> OrderDetail_ID Return_ID -> ReturnReqDate Return_ID -> RefundAmt

## Normalization

### 1. First Normal Form(1NF):

In order table, idea of Orderdetails attribute was to contain list of all orderdetails in a string format which leads to making the variable multivalued. For that reason, orderdetails attribute is dropped off. Remaining Tables satisfy the First Normal Form as they do not contain any multivalued attributes within the table.

### 2. Second Normal Form (2NF):

Second Normal Form (2NF) is based on the concept of functional dependency. It applies to a relation in which composite keys are present i.e., relations with a primary key composed of two or more attributes. In above mentioned schema each and every relation has single attribute primary key and hence every relation is in 2NF. Every Relation has no partial dependency i.e., no non - prime attributes are dependent on any proper subset of any candidate key of the table.

### 3. Redundancies:

Data redundancy occurs when the same piece of data exists in multiple places, whereas data inconsistency is when the same data exists in different formats in multiple tables. By analysing functional dependencies, some redundancies can be found. For Example,

- In Category table, it is clear that Category Quantity can itself verify CategoryStatus. If CategoryQuantity has value greater than zero then it is clear that item is available for shopping. That means CategoryStatus can be dropped from Category table to reduce redundancy.
- In OrderDetails table, it can be observed that TotalAmt can be derived from (UnitPrice – Discount) \* Quantity. So, TotalAmt can be dropped from the table.

- In Exchange table, CategoryColor is dependent variable on Category\_ID. As by Category\_ID, CategoryColor can simply be removed.
- In OrderDetails table, Color attribute is dependent on Category\_ID. Color can be dropped.
- In Exchange table, PayAgain is dependent on NewAmt which can be removed.

#### 4. 3rd Normal Form (3NF):

A given relation is said to be in its third normal form when it is in 2NF but has no transitive partial dependency. Meaning, when no transitive dependency exists for the attributes that are non-prime, then the relation can be said to be in 3NF. For given functional dependencies. It can be said that after applying 2<sup>nd</sup> Normal form it will automatically get converted into 3<sup>rd</sup> normal form.

## Final Relational Schema

- Customer (**Customer\_ID**, FName, LName, Address, Phone, Email)
- Orders (**Order\_ID**, DateOfOrder, *Customer\_ID*, **OrderDetails**)
- OrderDetails (**OrderDetail\_ID**, UnitPrice, **Color**, Quantity, Discount, **Total**, Date, *Category\_ID*, *Order\_ID*, *Bill\_ID*)
- Payment (**Bill\_ID**, PaymentType, Complete, TotalAmt)
- Exchange (**Exchange\_ID**, *OrderDetail\_ID*, ChangeQuantity, **CategoryColor**, NewAmt, *Bill\_ID*, PayAgain, *Category\_ID*)
- Product (**Product\_ID**, ProductName, ProductDesc, Discount, *Supplier\_ID*, OtherDetails)
- Category (**Category\_ID**, CategoryPrice, CategoryColor, **CategoryStatus**, CategoryQuantity, *Product\_ID*)
- Supplier (**Supplier\_ID**, CompanyName, Address, Phone, Email, Fax, OtherDetails)
- Returns (**Return\_ID**, *OrderDetail\_ID*, ReturnReqDate)

# MySQL QUERIES

## Table Creation

```
create table if not exists Customer(
```

```
    Customer_ID int not null auto_increment,  
    FName varchar(40) not null,  
    LName varchar(40) not null,  
    Address varchar(100) not null,  
    Email varchar(40) not null,  
    Phone varchar(20) not null,  
    primary key (Customer_ID),  
    unique (Email),  
    unique (Phone)
```

```
);
```

```
create table if not exists Supplier(
```

```
    Supplier_ID int not null auto_increment,  
    CompanyName varchar(40) not null,  
    Address varchar(100) not null,  
    Phone varchar(20) not null,  
    Email varchar(40) not null,  
    Fax varchar(8),  
    OtherDetails varchar(100),  
    primary key (Supplier_ID),  
    unique (Phone),  
    unique (Email),  
    unique (Fax)
```

```
);
```

```
create table if not exists Product(  
    Product_ID int not null auto_increment,  
    ProductName varchar(100) not null,  
    ProductDesc varchar(40) not null,  
    Discount float not null,  
    OtherDetails varchar(100),  
    Supplier_ID int not null,  
    primary key (Product_ID),  
    foreign key (Supplier_ID) references Supplier(Supplier_ID) match simple  
    on update cascade  
    on delete cascade  
);
```

```
create table if not exists Category(  
    Category_ID int not null auto_increment,  
    CategoryPrice float not null,  
    CategoryColor varchar(10) not null,  
    CategoryQuantity int not null,  
    Product_ID int not null,  
    primary key (Category_ID),  
    foreign key (Product_ID) references Product(Product_ID) match simple  
    on update cascade  
    on delete cascade  
);
```

```
create table if not exists Orders(  
    Order_ID int not null auto_increment,  
    DateOfOrder date not null,  
    Customer_ID int not null,  
    primary key (Order_ID),  
    foreign key (Customer_ID) references Customer(Customer_ID) match simple  
    on update cascade  
    on delete cascade  
);
```

```
create table if not exists Payment(  
    Bill_ID int not null auto_increment,  
    PaymentType varchar(40) not null,  
    Complete tinyint default(0),  
    TotalAmt float not null,  
    primary key (Bill_ID)  
);
```

```
create table if not exists OrderDetails(
    OrderDetail_ID int not null auto_increment,
    UnitPrice float not null,
    Quantity int not null,
    Discount float not null,
    DateofOrder date default(CURRENT_DATE),
    Category_ID int not null,
    Order_ID int not null,
    Bill_ID int not null,
    primary key (OrderDetail_ID),
    foreign key (Category_ID) references Category(Category_ID) match simple
        on update cascade
        on delete cascade,
    foreign key (Order_ID) references Orders(Order_ID) match simple
        on update cascade
        on delete cascade,
    foreign key (Bill_ID) references Payment(Bill_ID) match simple
        on update cascade
        on delete cascade
);
```

```
create table if not exists Returns(
    Return_ID int not null auto_increment,
    OrderDetail_ID int not null,
    ReturnReqDate date not null,
    primary key (Return_ID),
    foreign key (OrderDetail_ID) references OrderDetails(OrderDetail_ID) match simple
        on update cascade
        on delete cascade
);
```

```
create table if not exists Exchange(
    Exchange_ID int not null auto_increment,
    OrderDetail_ID int not null,
    Category_ID int not null,
    ChangeQuantity int not null,
    NewAmt float not null,
    Bill_ID int not null,
    primary key (Exchange_ID),
    foreign key (OrderDetail_ID) references OrderDetails(OrderDetail_ID) match simple
    on update cascade
    on delete cascade,
    foreign key (Bill_ID) references Payment(Bill_ID) match simple
    on update cascade
    on delete cascade,
    foreign key (Category_ID) references Category(Category_ID) match simple
    on update cascade
    on delete cascade
);
```

## Data Insertion

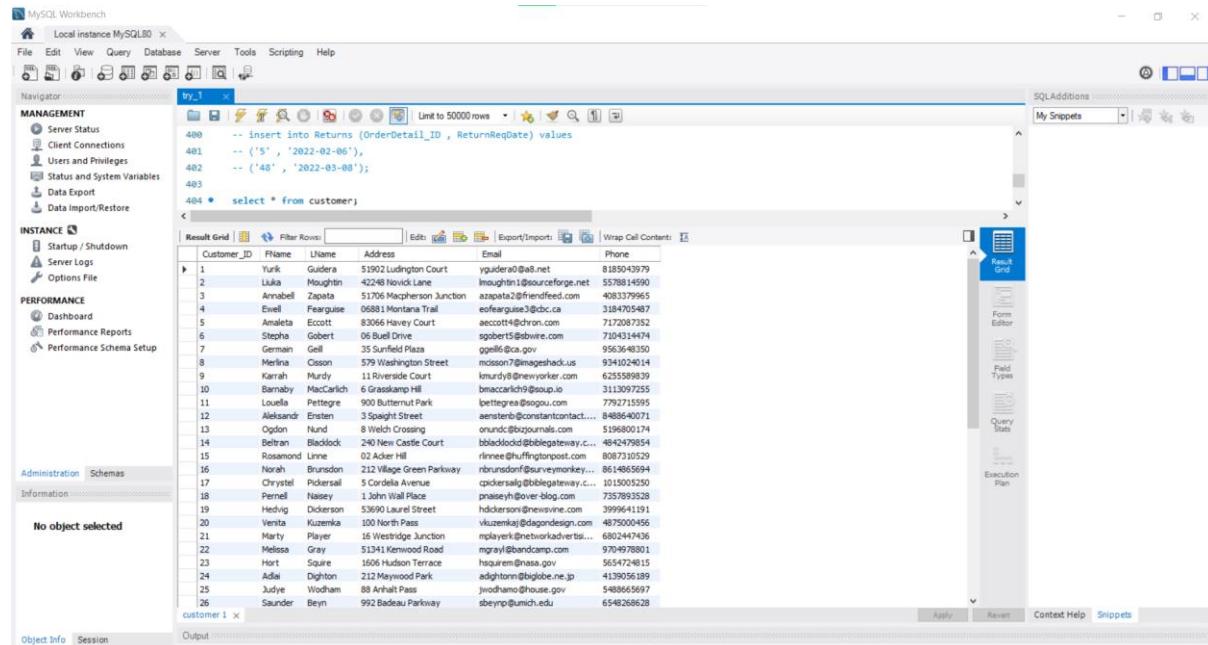
Link of Website Used for Random Data Generation - [Link](#)

Inserted data is generated randomly. Logical part is added with Python script in data generation to derive some meaningful data from random data.

## Show Table Entries

1. Show the all details of Customer  

```
select * from customer;
```



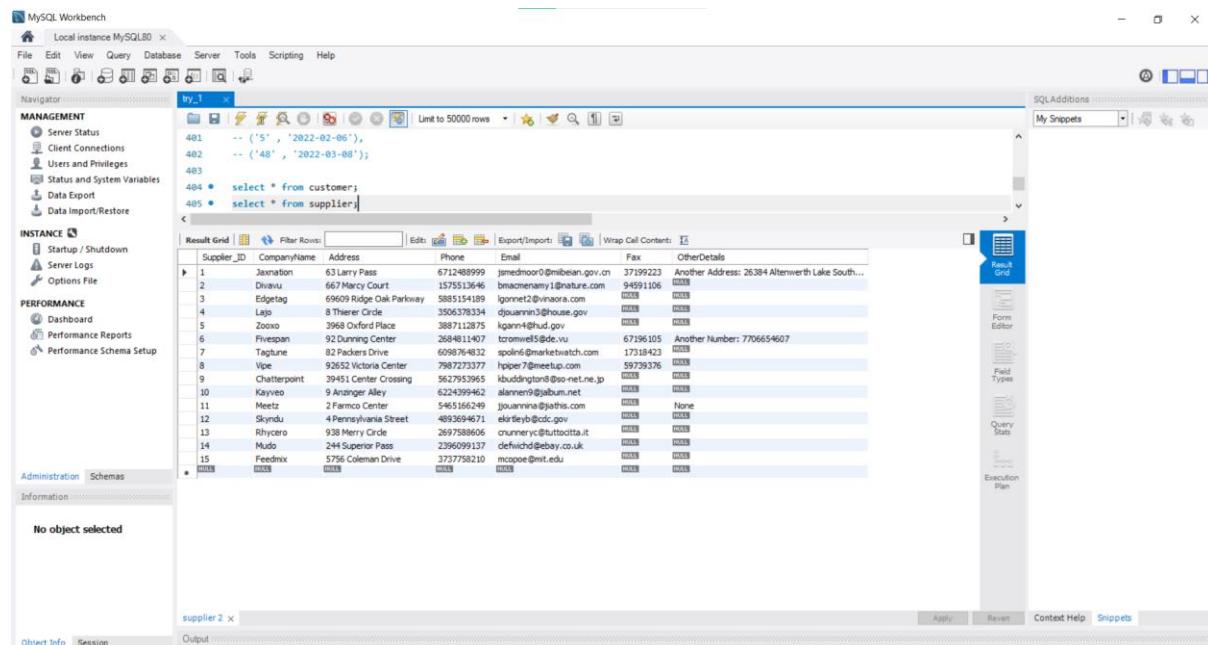
The screenshot shows the MySQL Workbench interface with the following details:

- File Bar:** File, Edit, View, Query, Database, Server, Tools, Scripting, Help.
- Navigator:** MANAGEMENT, PERFORMANCE, Administration, Schemas, Information.
- SQL Editor:** The query `select * from customer;` is entered.
- Result Grid:** The results show 26 rows of customer data with columns: Customer\_ID, Fname, Lname, Address, Email, Phone.
- Output Tab:** Shows the raw SQL query and the results grid.
- Toolbars and Panels:** Standard Windows-style toolbar, SQLAdditions panel with 'My Snippets' tab.

Customer_ID	Fname	Lname	Address	Email	Phone
1	Yurk	Guidera	51902 Ludington Court	yguidera@ais.net	8185043979
2	Luka	Moughlin	42248 Novick Lane	lmoughlin@sourceforge.net	5578814590
3	Anabel	Zapata	51706 Macpherson Junction	azapata2@feedfeed.com	4083379965
4	Ewell	Feargus	06831 Montana Trail	efeargus3@cbc.ca	3184705487
5	Amelita	Ecott	83066 Hayey Court	aecott14@chron.com	7172067352
6	Stepha	Gobert	06 Buell Drive	sgobert15@dwire.com	7104314474
7	German	Gell	35 Sunfield Plaza	ggell6@cs.gov	9563648350
8	Merlina	Cisson	579 Washington Street	mcsson7@mageshadow.com	9341024014
9	Karrah	Murdy	11 Riverside Court	kmurdy8@newyorker.com	6255598939
10	Barnaby	MacCarlich	6 Grasskamp Hill	bmaccarlich9@sophia.io	3113097255
11	Louella	Pettegrew	900 Butternut Park	lpettegrew@soogoo.com	7792715595
12	Aleksander	Ersten	3 Spright Street	aersten10@constantcontact.com	8488640071
13	Ogdon	Nunn	8 Welch Crossing	onunn11@bzjournals.com	5196800174
14	Beltran	Blacklock	240 New Castle Court	bbblacklock12@bubbleteway.com	4842479854
15	Rosemond	Linn	02 Acker Hill	rlinn13@ruffingtonpost.com	8087310529
16	Norah	Brundson	212 Village Green Parkway	rbrundson14@sourcemonkey.com	8614865694
17	Christy	Pidkiss	420 Villa Avenue	c.pidkiss15@outlook.com	1015005020
18	Reinell	Neumeyer	2 John W Place	rneumeyer16@wpaper-blog.com	9109003528
19	Hedwig	Dickerson	53690 Laur Street	hdickerson17@newswise.com	3999641981
20	Venita	Kuzemka	100 North Park	v.kuzemka18@spindesign.cz	487500456
21	Marty	Player	16 Westridge Junction	mplayer19@newswiseadvertisers.com	6802474746
22	Melissa	Gray	51341 Kenwood Road	mgray1@bandcamp.com	979497801
23	Hort	Soure	1609 Hudson Terrace	hsouren1@nsa.gov	5654724815
24	Adai	Dighton	212 Maywood Park	adighton@biglobe.ne.jp	4139056189
25	Judye	Wodham	88 Anhalt Pass	jwodham@house.gov	5488665697
26	Saundar	Beyn	992 Badeau Parkway	sbeynp@umich.edu	6548268628

2. Show the all details of Supplier

```
select * from supplier;
```



The screenshot shows the MySQL Workbench interface with the following details:

- File Bar:** File, Edit, View, Query, Database, Server, Tools, Scripting, Help.
- Navigator:** MANAGEMENT, PERFORMANCE, Administration, Schemas, Information.
- SQL Editor:** The queries `select * from customer;` and `select * from supplier;` are entered.
- Result Grid:** The results show 15 rows of supplier data with columns: Supplier\_ID, CompanyName, Address, Phone, Email, Fax, OtherDetails.
- Output Tab:** Shows the raw SQL queries and the results grid.
- Toolbars and Panels:** Standard Windows-style toolbar, SQLAdditions panel with 'My Snippets' tab.

Supplier_ID	CompanyName	Address	Phone	Email	Fax	OtherDetails
1	Janatton	63 Larry Pass	5712488999	jmedmoor0@mitbean.gov.cn	37192923	Another Address: 26384 Alterneworth Lake South...
2	Bilavu	6574964 Court	1575813244	l.bilavu1@nature.com	34991106	
3	Edgetag	69609 Ridge Oak Parkway	5885154189	lgovet2@innowe.com		
4	Laps	8 Therier Circle	3506178334	d.pouann3@house.gov		
5	Zoxio	3968 Oxford Place	3887112875	k.parn4@hul.gov		
6	FiveSpan	92 Dunning Center	264811407	tcroneveld5@de.vu	67196105	Another Number: 7076654607
7	Tagtune	82 Packers Drive	6098764832	spolin6@marketwatch.com	17318423	
8	Vipe	92852 Victoria Center	7987273377	hpiper7@metabu.com	59729376	
9	Chatterpoint	39451 Center Crossing	5627953965	kbounding9@osu-net.ne.jp		
10	Kaytee	9 Anzinger Alley	6224399462	alanner10@abun.net		
11	Meetz	2 Farmco Center	5465166249	jjoanna@jaithis.com		None
12	Skyndy	4 Pennsylvania Street	4893694671	ekirleyb@cdc.gov		
13	Rhynero	938 Merry Circle	2697588606	cnunmeryc@tutotatta.it		
14	Mudo	24 Superior Pass	2396099137	cliefield12@ebay.co.uk		
15	Feedmix	5756 Coleman Drive	3737758210	mcopoe@mit.edu		

### 3. Show the all details of Product

`select * from product;`

The screenshot shows the MySQL Workbench interface with a query editor window titled 'try\_1'. The SQL code is:

```

482 -- ('48', '2022-03-08');
483
484 • select * from customer;
485 • select * from supplier;
486 • select * from product;

```

The results grid displays 26 rows of data from the 'product' table:

Product_ID	ProductName	ProductDesc	Discount	OtherDetails	Supplier_ID
1	Mazda6	speed - 179, fuel - 9	0.77	Inspired from Formula-1 cars	15
2	CR-V	speed - 151, fuel - 7	4.71		1
3	CL-Class	speed - 114, fuel - 9	4.99		9
4	Vantage	speed - 183, fuel - 9	2.66		9
5	Nano	speed - 168, fuel - 5	1.32		14
6	Crown Victoria	speed - 109, fuel - 5	1.71		13
7	Tahoe	speed - 162, fuel - 7	1		9
8	Caprice	speed - 191, fuel - 10	4.43		11
9	Tiburon	speed - 171, fuel - 6	3.15		5
10	Continental Flying Spur	speed - 177, fuel - 10	3.35		12
11	G37	speed - 151, fuel - 6	3.88		1
12	Savana 2500	speed - 138, fuel - 6	1.31		9
13	Aero S	speed - 184, fuel - 10	3.48	SuperFast Speed coverage	9
14	Defender 90	speed - 149, fuel - 10	5.52		10
15	Grand Am	speed - 146, fuel - 5	1.42		5
16	Avalanche	speed - 130, fuel - 6	1.91		11
17	Galant	speed - 174, fuel - 8	1.96		4
18	Skylark	speed - 146, fuel - 5	3.18		5
19	Century	speed - 158, fuel - 6	4.61		6
20	M6	speed - 160, fuel - 10	2.12		2
21	Mits	speed - 119, fuel - 6	0.84		3
22	Patero	speed - 170, fuel - 6	2.22		1
23	Avenger	speed - 157, fuel - 6	2.15		1
24	Riviera	speed - 167, fuel - 10	4.96	Book today. Cars for you	9
25	Avalon	speed - 111, fuel - 5	4.03		14
26	Escape	speed - 165, fuel - 5	1.75		10

### 4. Show the all details of Category

`select * from category;`

The screenshot shows the MySQL Workbench interface with a query editor window titled 'try\_1'. The SQL code is:

```

483
484 • select * from customer;
485 • select * from supplier;
486 • select * from products;
487 • select * from category;

```

The results grid displays 26 rows of data from the 'category' table:

Category_ID	CategoryName	CategoryColor	CategoryQuantity	Product_ID
1	3580140	Turquoise	406	23
2	1000000	Red	247	24
3	312790	Red	276	4
4	3204790	Aquamarine	403	
5	302460	Fuchsia	347	19
6	2002199	Khaki	56	27
7	3143640	Mauv	410	6
8	4714510	Teal	380	2
9	2343590	Aquamarine	136	15
10	1308640	Orange	416	12
11	3054990	Indigo	284	26
12	4345150	Violet	142	12
13	640788	Violet	447	21
14	1034840	Turquoise	457	27
15	928127	Puce	477	20
16	1011650	Orange	59	6
17	2768050	Orange	320	6
18	1632970	Goldennod	267	16
19	844571	Crimson	433	25
20	2274880	Orange	270	14
21	2973990	Pink	388	6
22	2804600	Aquamarine	339	8
23	2394600	Aquamarine	263	7
24	867983	Pink	340	19
25	4118680	Crimson	457	23
26	4927320	Mauv	155	11

## 5. Show the all details of Payment

```
select * from payment;
```

The screenshot shows the MySQL Workbench interface with the following details:

- Navigator:** Shows the database structure with tables: customer, supplier, products, category, and payment.
- INSTANCE:** Shows startup/shutdown, server logs, and options file.
- PERFORMANCE:** Shows dashboard, performance reports, and performance schema setup.
- Administration:** Shows schemas and information.
- Information:** Shows no object selected.
- Result Grid:** Displays the results of the query "select \* from payment;". The columns are BILL\_ID, PaymentType, Complete, and TotalAmt. The data includes rows for various payment types like Cash On Delivery, UPI, DebitCard, Banking, etc., with their respective counts and amounts.
- SQLAdditions:** Shows snippets like "My Snippets".
- Toolbars:** Includes standard MySQL Workbench tools for file operations, search, and export.
- Status Bar:** Shows "Limit to 50000 rows", "Edit", "File", "Export/Import", "Wrap Cell Content", and "Result Grid" button.
- Bottom Buttons:** Includes "Copy", "Revert", "Context Help", and "Snippets".

## 6. Show the all details of Order

```
select * from orders;
```

The screenshot shows the MySQL Workbench interface with the following details:

- Navigator:** Shows the database structure with tables: supplier, product, category, payment, and orders.
- INSTANCE:** Shows startup/shutdown, server logs, and options file.
- PERFORMANCE:** Shows dashboard, performance reports, and performance schema setup.
- Administration:** Shows schemas and information.
- Information:** Shows no object selected.
- Result Grid:** Displays the results of the query "select \* from orders;". The columns are Order\_ID, DateOfOrder, and Customer\_ID. The data includes rows for various dates from 2022-02-24 to 2022-05-23, with their respective order IDs and customer IDs.
- SQLAdditions:** Shows snippets like "My Snippets".
- Toolbars:** Includes standard MySQL Workbench tools for file operations, search, and export.
- Status Bar:** Shows "Limit to 50000 rows", "Edit", "File", "Export/Import", "Wrap Cell Content", and "Result Grid" button.
- Bottom Buttons:** Includes "Copy", "Revert", "Context Help", and "Snippets".

## 7. Show the all details of OrderDetails

```
select * from orderdetails;
```

The screenshot shows the MySQL Workbench interface with a query editor window titled 'try\_1'. The SQL pane contains the following code:

```
486 • select * from products;
487 • select * from category;
488 • select * from payment;
489 • select * from orders;
410 • select * from orderdetails;
```

The Results pane displays the data from the 'orderdetails' table:

OrderDetail_ID	UnitPrice	Quantity	Discount	DateofOrder	Category_ID	Order_ID	Bill_ID
1	2093670	2	1.20	2022-02-24	49	1	1
2	3580140	3	2.15	2022-02-24	1	1	1
3	1632970	3	1.91	2021-12-14	18	2	2
4	2768050	3	1.71	2022-02-03	17	3	3
5	3093670	1	1.32	2022-02-03	49	3	3
6	4869640	3	3.15	2022-01-19	33	4	4
7	705635	2	2.95	2022-06-20	46	5	5
8	844371	3	4.03	2022-06-20	19	5	5
9	640789	1	0.84	2022-06-08	13	6	6
10	4649340	2	2.12	2022-06-08	35	6	6
11	4003720	3	4.96	2022-06-08	50	6	6
12	3343640	3	1.71	2022-06-08	7	6	6
13	2811550	2	2.15	2022-06-08	34	6	31
14	4899980	1	2.66	2022-03-12	29	7	7
15	2973970	3	1.71	2022-03-12	21	7	7
16	1811070	2	3.88	2022-04-28	44	8	8
17	640789	2	0.84	2021-08-09	13	9	32
18	4003720	2	2.12	2021-08-09	6	9	10
19	705635	1	2.95	2022-06-03	48	10	10
20	4714510	1	6.71	2021-12-28	8	11	11
21	3418610	3	2.12	2022-02-21	43	12	12
22	2343590	1	1.42	2021-09-03	9	13	13
23	4950930	3	4.61	2022-02-04	30	14	14
24	3204790	2	3.15	2022-02-04	4	14	14
25	1957150	1	2.12	2022-02-04	42	14	14
26	2768050	3	1.71	2021-12-14	17	15	15

## 8. Show the all details of Return

```
select * from returns;
```

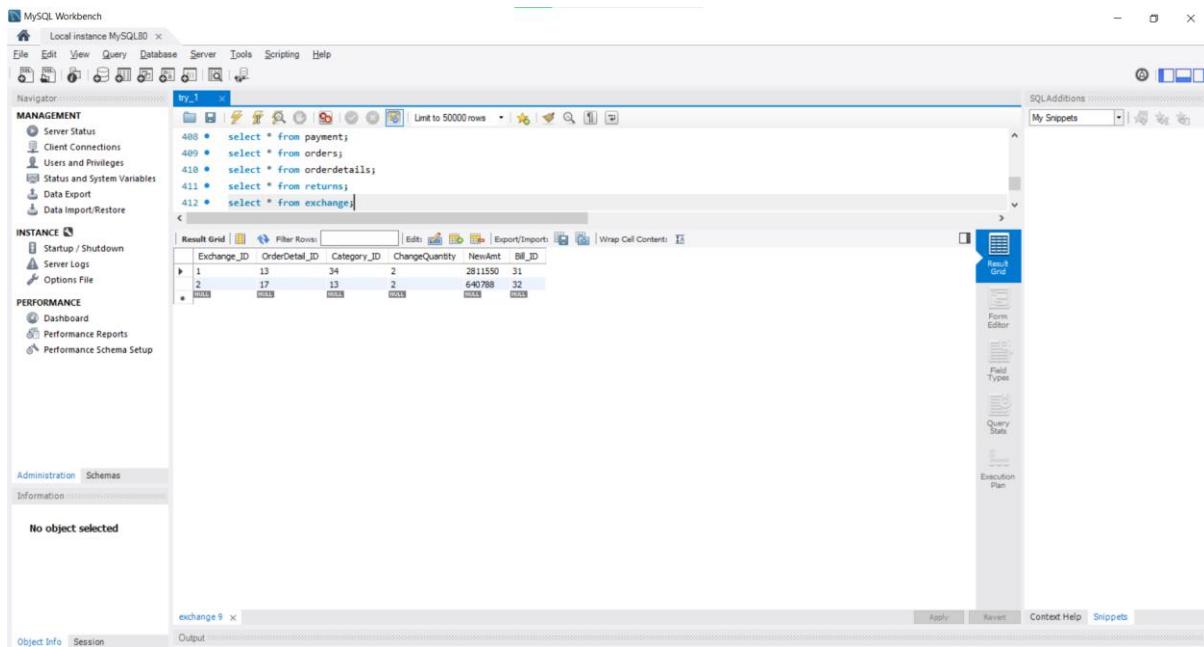
The screenshot shows the MySQL Workbench interface with a query editor window titled 'try\_1'. The SQL pane contains the following code:

```
487 • select * from category;
488 • select * from payment;
489 • select * from orders;
410 • select * from orderdetails;
411 • select * from returns;
```

The Results pane displays the data from the 'returns' table:

Return_ID	OrderDetail_ID	ReturnReqDate
1	5	2022-02-06
2	48	2022-03-08
*	NULL	NULL

9. Show the all details of Exchange  
select \* from exchange;



The screenshot shows the MySQL Workbench interface. In the top navigation bar, 'File', 'Edit', 'View', 'Query', 'Database', 'Server', 'Tools', 'Scripting', and 'Help' are visible. Below the menu is a toolbar with various icons. The main area contains a query editor window titled 'try\_1' with the following SQL code:

```
408 • select * from payments;
409 • select * from orders;
410 • select * from orderdetails;
411 • select * from returns;
412 • select * from exchange;
```

Below the code is a result grid titled 'Result Grid' showing the data from the 'exchange' table:

Exchange_ID	OrderDetail_ID	Category_ID	ChangeQuantity	NewAmt	Bill_ID
1	13	34	2	2811550	31
2	17	13	2	640788	32

The bottom of the interface shows tabs for 'Object Info' and 'Session' under 'exchange 9'.

## Length of Tables

Query – select count(\*) from Table\_Name

Table Name	Length query response
Customer	<input type="text" value="count(*)"/> <input type="button" value="▶"/> 50
Supplier	<input type="text" value="count(*)"/> <input type="button" value="▶"/> 15
Order	<input type="text" value="count(*)"/> <input type="button" value="▶"/> 30
OrderDetails	<input type="text" value="count(*)"/> <input type="button" value="▶"/> 50
Product	<input type="text" value="count(*)"/> <input type="button" value="▶"/> 30
Category	<input type="text" value="count(*)"/> <input type="button" value="▶"/> 50
Payment	<input type="text" value="count(*)"/> <input type="button" value="▶"/> 32
Return	<input type="text" value="count(*)"/> <input type="button" value="▶"/> 2
Exchange	<input type="text" value="count(*)"/> <input type="button" value="▶"/> 2

## SQL Queries

### Select ... from queries

1. Show the sum of all payments made till now.

```
select sum(totalamt) from payment;
```

The screenshot shows the MySQL Workbench interface with a query editor window titled 'try\_1'. The query is:

```
420 • select count(*) from payment;
421 • select count(*) from returns;
422 • select count(*) from exchange;
423
424 • select sum(totalamt) from payment;
```

The results grid shows one row with the value '194891940' under the column 'sum(totalamt)'. The right sidebar contains tabs for 'Result Grid', 'Form Editor', 'Field Types', 'Query Stats', and 'Execution Plan'.

Tuple count = 1

2. Show different type of payments made.

```
select distinct(paymenttype) from payment;
```

The screenshot shows the MySQL Workbench interface with a query editor window titled 'try\_1'. The query is:

```
422 • select count(*) from exchange;
423
424 • select sum(totalamt) from payment;
425
426 • select distinct(paymenttype) from payment;
```

The results grid shows five rows with the values 'paymenttype', 'Cash On Delivery', 'UPI', 'DebitCard', 'Banking', and 'CreditCard'. The right sidebar contains tabs for 'Result Grid', 'Form Editor', 'Field Types', 'Query Stats', and 'Execution Plan'.

Tuple count = 5

## Select ... from ... where queries

3. Show all the payments made with debit card.

`select * from payment where paymenttype = 'DebitCard'`

The screenshot shows the MySQL Workbench interface with a query editor window titled 'try\_1'. The SQL code is:

```

424 • select sum(totalamt) from payment;
425
426 • select distinct(paymenttype) from payment;
427
428 • select * from payment where paymenttype = 'DebitCard';
    
```

The results grid shows the following data:

Bill_ID	PaymentType	Complete	Totalamt
5	DebitCard	1	5773540
13	DebitCard	1	2751100
15	DebitCard	1	2692770
22	DebitCard	1	2310310
25	DebitCard	1	9742160
27	DebitCard	1	7328400

Tuple count = 6

4. Show the details of product in which otherdetails field is not empty.

`select * from product where otherdetails is not NULL;`

The screenshot shows the MySQL Workbench interface with a query editor window titled 'try\_1'. The SQL code is:

```

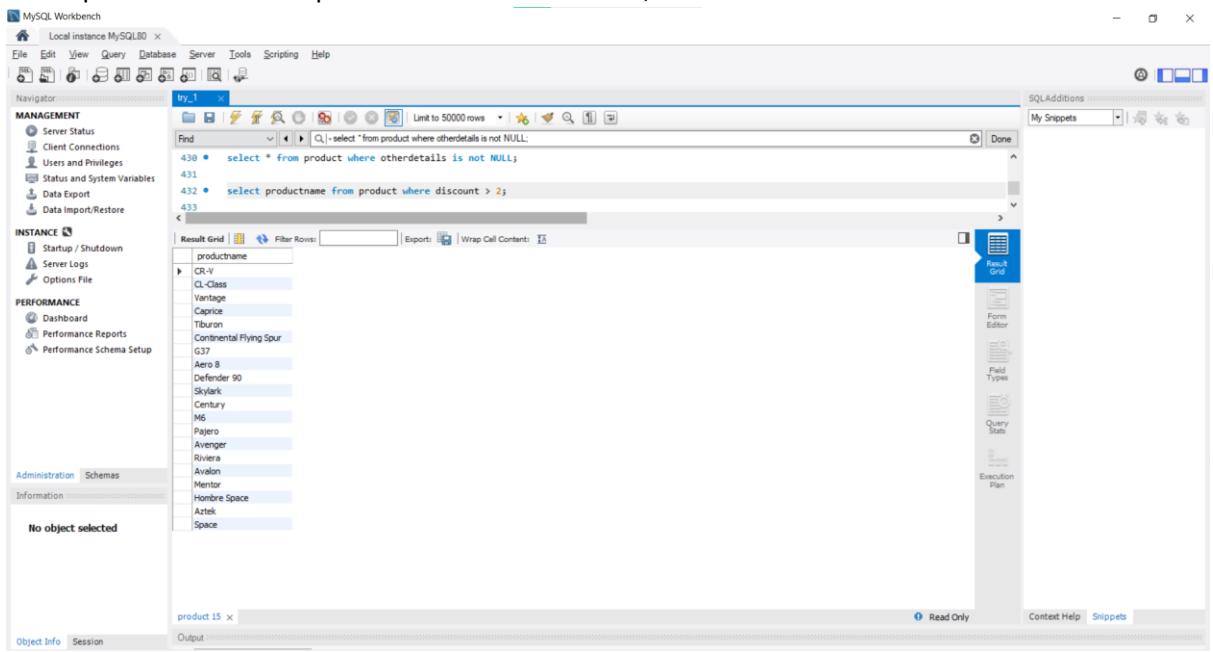
Find: Q -> select * from product where otherdetails is not NULL;
428 • select * from payment where paymenttype = 'DebitCard';
429
430 • select * from product where otherdetails is not NULL;
431
    
```

The results grid shows the following data:

Product_ID	ProductName	ProductDesc	Discount	OtherDetails	Supplier_ID
1	Mazda6	speed - 179, fuel - 9	0.77	Inspired from Formula 1 cars	15
13	Aero B	speed - 184, fuel - 10	3.48	SuperFast Speed coverage	9
24	Riviera	speed - 167, fuel - 10	4.96	Book today, Cars for you	9

Tuple Count = 3

5. Show all product names with discount greater than 2%  
 select productname from product where discount > 2;



The screenshot shows the MySQL Workbench interface with a query editor window titled 'try\_1'. The query is:

```

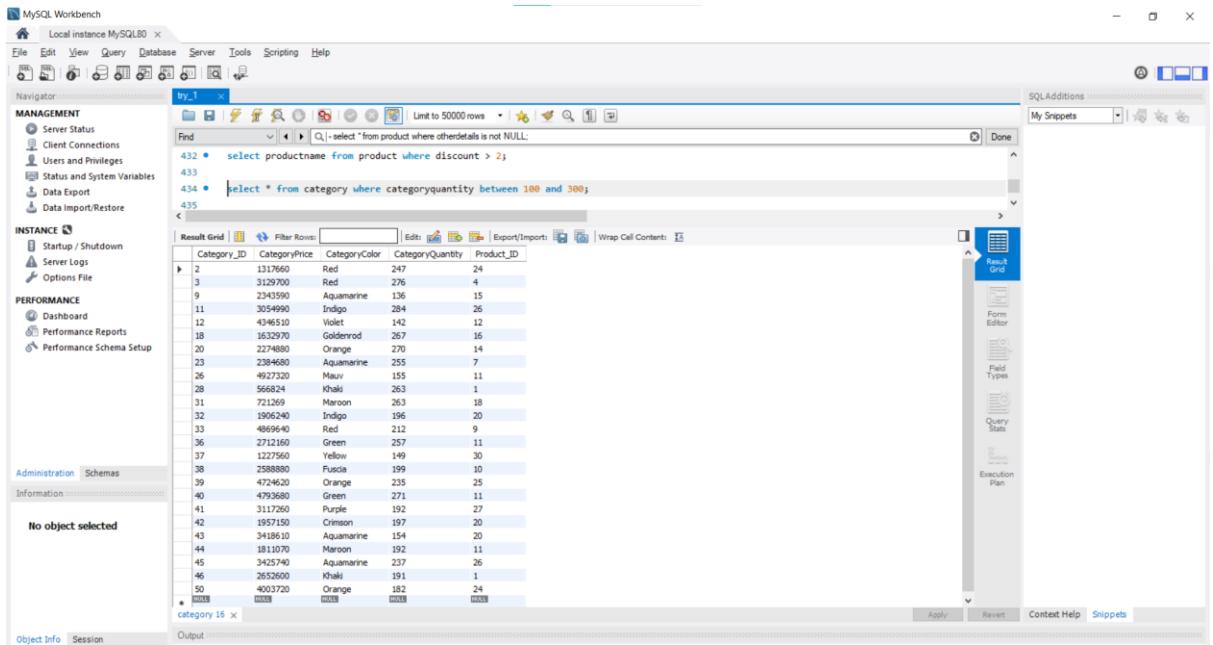
430 • select * from product where otherdetails is not NULL;
431
432 • select productname from product where discount > 2;
433
434
  
```

The results grid displays the following data:

productname
CR-Y
CL-Class
Vantage
Caprice
Tiburon
Continental Flying Spur
G37
Aero 8
Defender 90
Skylark
Century
M6
Pajero
Avenger
Riviera
Avalon
Hombre Space
Mentor
Hombre Space
Atztek
Space

Tuple count = 20

6. Show category of product available with between 100 to 300.  
 select \* from category where categoryquantity between 100 and 300;



The screenshot shows the MySQL Workbench interface with a query editor window titled 'try\_1'. The query is:

```

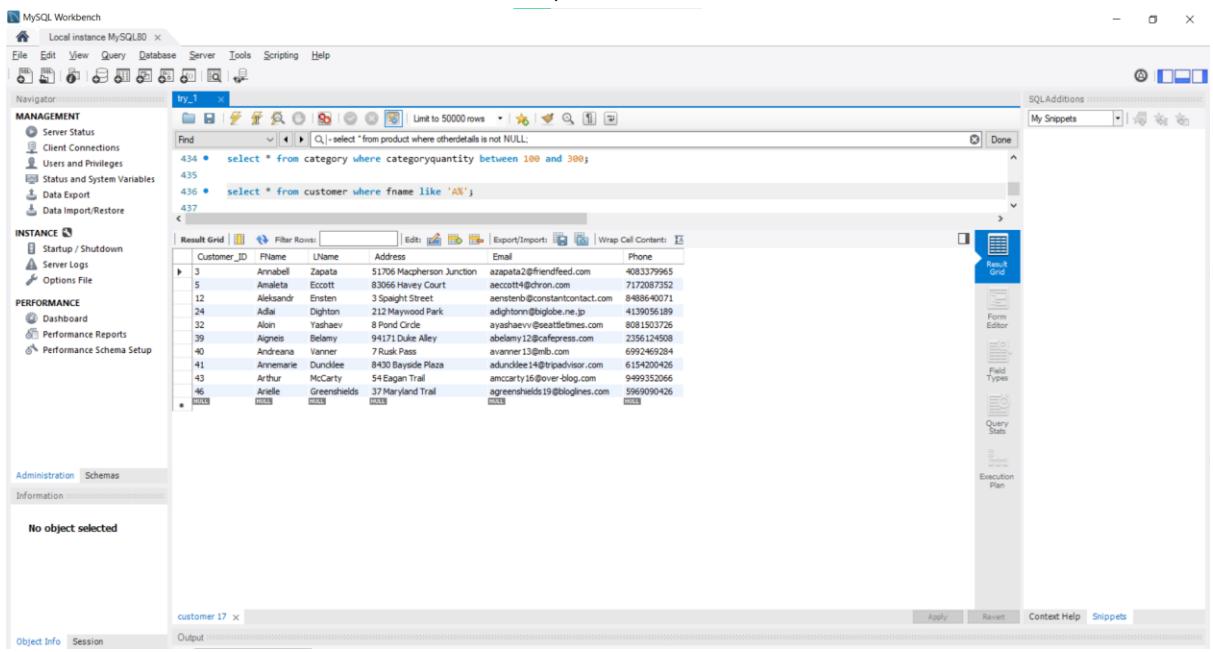
432 • select productname from product where discount > 2;
433
434 • select * from category where categoryquantity between 100 and 300;
435
  
```

The results grid displays the following data:

Category_ID	CategoryPrice	CategoryColor	CategoryQuantity	Product_ID
2	1317660	Red	247	24
3	3129700	Red	276	4
9	2343590	Aquamarine	136	15
11	3054990	Indigo	284	26
12	4346510	Violet	142	12
18	1632970	Goldenrod	267	16
20	2274880	Orange	270	14
23	2384680	Aquamarine	255	7
26	4927320	Mauv	155	11
28	566824	Khaki	263	1
31	721269	Maroon	263	18
32	1906240	Indigo	196	20
33	4869640	Red	212	9
36	2712160	Green	257	11
37	1227560	Yellow	149	30
38	2588880	Fuscia	199	10
39	4724620	Orange	235	25
40	4793580	Green	271	11
41	311260	Purple	192	27
42	1897150	Cream	197	20
43	3418610	Aquamarine	154	20
44	1811070	Marsion	192	11
45	3425740	Aquamarine	237	26
46	2652600	Khaki	191	1
50	4003720	Orange	182	24

Tuple count = 25

7. Show details of customer with first name starting from A.  
select \* from customer where fname like 'A%';



```

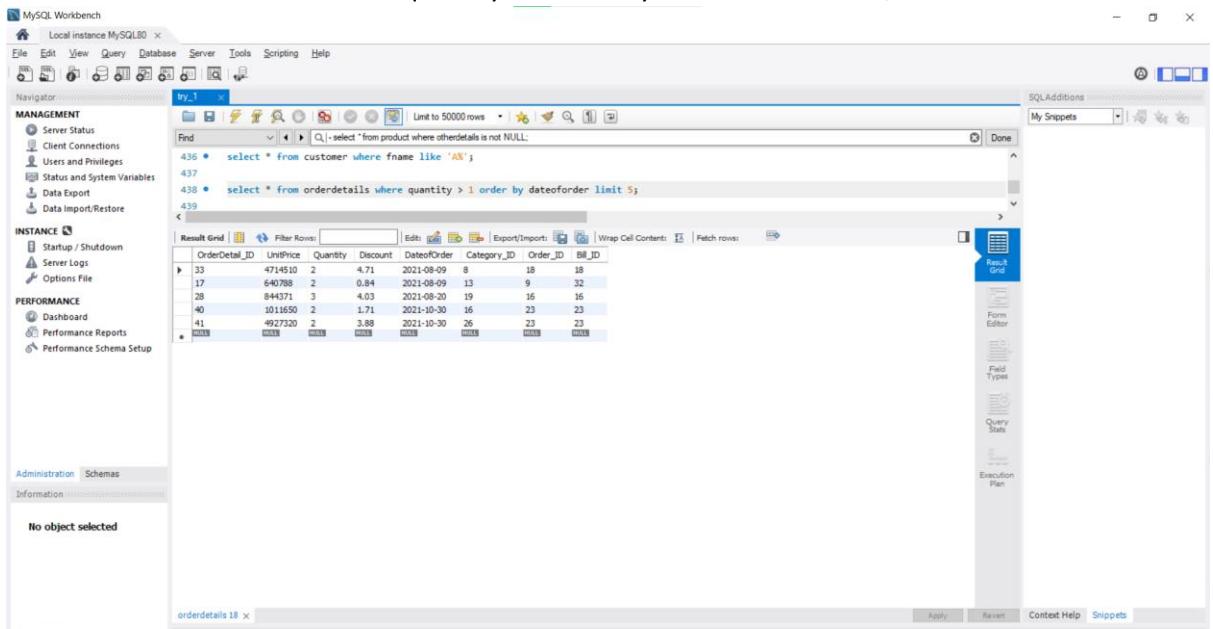
434 • select * from category where categoryquantity between 100 and 300;
435
436 • select * from customer where fname like 'A%' ;
437

```

Customer_ID	FName	LName	Address	Email	Phone
3	Anabel	Zapata	51706 Macpherson Junction	azapata2@friendfeed.com	4083379965
5	Amalita	Eccott	83066 Hayven Court	aeccott4@chron.com	7172087352
12	Aleksandr	Ersten	3 Spaight Street	aerstenb@constantcontact.com	8488640071
24	Adai	Dighton	212 Maywood Park	adighton@biglobe.ne.jp	4139056189
32	Alon	Yashaev	8 Pond Circle	ayashaev@seattletimes.com	8081503726
39	Aignis	Belamy	94171 Duke Alley	abelamy12@cafexpress.com	2356124508
40	Andreae	Vanne	7 Rusk Pass	avanner13@nbn.com	6992469284
41	Annerarie	Dundee	8430 Bayside Plaza	adundee14@tpadvisor.com	6154200426
43	Arthur	McCarty	54 Eagan Trail	amccarty16@over-blog.com	9499352066
46	Arielle	Greenshields	37 Maryland Trail	agreenshields19@bloglines.com	5969090426

Tuple count = 10

8. Show details of first 5 orders in OrderDetails with quantity greater than 1.  
select \* from orderdetails where quantity > 1 order by dateoforder limit 5;



```

436 • select * from customer where fname like 'A%' ;
437
438 • select * from orderdetails where quantity > 1 order by dateoforder limit 5;
439

```

OrderDetail_ID	UnitPrice	Quantity	Discount	DateofOrder	Category_ID	Order_ID	BIL_ID
33	47.14510	2	4.71	2021-08-09	8	18	18
17	649788	2	0.84	2021-08-09	13	9	32
28	844371	3	4.03	2021-08-20	19	16	16
40	1011650	2	1.71	2021-10-30	16	23	23
41	49.27220	2	3.88	2021-10-30	26	23	23

Tuple count = 5

9. Calculate average amount paid via Banking mode.

```
select avg(totalamt) from payment where paymenttype = 'Banking';
```

avg(totalamt)
6849044.142857143

Tuple count = 1

10. Show details of category of product with quantity between 200 to 300 and price more than 200000.

```
select * from category where categoryquantity  
between 200 and 300 and categoryprice > 100000;
```

Category_ID	CategoryPrice	CategoryColor	CategoryQuantity	Product_ID
2	1317660	Red	247	24
3	3129700	Red	276	4
11	3054990	Indigo	284	26
18	1632970	Goldengrod	267	16
20	2273680	Orange	270	14
23	2384680	Aquamarine	255	7
28	566824	Khaki	263	1
31	721269	Maron	263	18
33	4866460	Red	212	9
36	2712160	Green	257	11
39	4724620	Orange	235	25
40	4793680	Green	271	11
45	3425740	Aquamarine	237	26
*	NULL	NULL	NULL	NULL

Tuple count = 13

## Select ... from ... orderby/groupby queries

11. Show the different ways of payment with average amount of payment in that type.

```
select paymenttype , avg(totalamt) from payment
group by paymenttype
order by paymenttype;
```

paymenttype	avg(totalamt)
Banking	684041.142857143
Cash On Delivery	3185021
CreditCard	556049
DebitCard	9933047.833333333
UPI	8351477.125

Tuple count = 5

12. Show different type of payment and count of it sorted from lower to higher count.

```
select paymenttype , count(paymenttype) as count from payment
group by paymenttype order by count;
```

paymenttype	count
CreditCard	4
DebitCard	6
Cash On Delivery	7
Banking	7
UPI	8

Tuple count = 5

13. Show orderdetails of orders made in 2022 year in sorted manner.

```
select * from orderdetails
where dateoforder >= '2022-01-01'
order by dateoforder;
```

OrderDetail_ID	UnitPrice	Quantity	Discount	DatedOrder	Category_ID	Order_ID	Bill_ID
6	4899640	3	3.15	2022-01-15	33	4	4
43	1034840	2	2.95	2022-01-25	14	24	24
4	2768050	3	1.71	2022-02-03	17	3	3
5	3093670	1	1.32	2022-02-03	49	3	3
25	1957150	2	2.12	2022-02-04	42	14	14
24	3204760	2	3.15	2022-02-04	4	14	14
23	4910450	3	4.31	2022-02-04	20	14	14
47	9745100	2	4.71	2022-02-09	8	28	28
21	2418610	3	2.12	2022-02-21	43	12	12
2	3580190	3	2.15	2022-02-24	1	1	1
1	3093670	2	1.32	2022-02-24	49	1	1
30	4714510	1	4.71	2022-02-25	8	17	17
31	2806460	1	2.66	2022-02-25	27	17	17
32	2598890	3	3.35	2022-02-25	38	17	17
48	3093670	3	1.32	2022-03-01	49	29	29
15	2973990	3	1.71	2022-03-12	21	7	7
14	4899990	1	2.66	2022-03-12	29	7	7
44	2343590	1	1.42	2022-04-20	9	25	25
16	1811070	2	3.88	2022-04-28	44	8	8
45	4703680	1	3.88	2022-05-23	40	26	26
19	705635	1	2.95	2022-06-03	48	10	10
18	2002190	2	2.95	2022-06-03	6	10	10
9	640788	1	0.84	2022-06-08	13	6	6
10	4649340	2	2.12	2022-06-08	35	6	6
11	4003720	3	4.96	2022-06-08	50	6	6
12	3343640	3	1.71	2022-06-08	7	6	6

Tuple count = 34

Select ... from ... where ... like queries

14. Show paymenttypes with average amount paid less than 6000000.

```
select paymenttype from payment
group by paymenttype
having avg(totalamt) <= 6000000;
```

paymenttype
Cash On Delivery
DebitCard
CreditCard

Tuple Count = 3

15. Show customer details of those customers whose number start with 8.

```
select * from customer where phone like '8%';
```

Customer_ID	Fname	LName	Address	Email	Phone
32	Alain	Yashvev	8 Pond Circle	ayashvev@seattletimes.com	8081503726
15	Rosamond	Linne	02 Acker Hill	rlinnee@huffingtonpost.com	8087310529
1	Yurk	Gudera	51902 Ludington Court	ygudera@alt.net	8185043979
12	Aleksandr	Ersten	3 Spaight Street	aerstenb@constantcontact.com	8488640071
16	Norah	Brundson	212 Village Green Parkway	nbrundson@surveymonkey.com	8614865694
37	Chelsey	Johanning	2804 Northland Alley	cjohanning10@desdev.on	8628471245
28	Hasheem	Kingsroad	59670 Blane Hill	Hkingsoad@cmu.edu	8761285130
34	Holly	Keggan	98737 Burrows Court	Hkeggan@utexas.edu	8849249127

Tuple count = 8

16. Show third most costly product category.

```
select * from
```

```
(select * from category order by categoryprice desc limit 3) as cat
order by categoryprice limit 1;
```

Category_ID	CategoryPrice	CategoryColor	CategoryQuantity	Product_ID
29	4899980	Fuscia	488	4

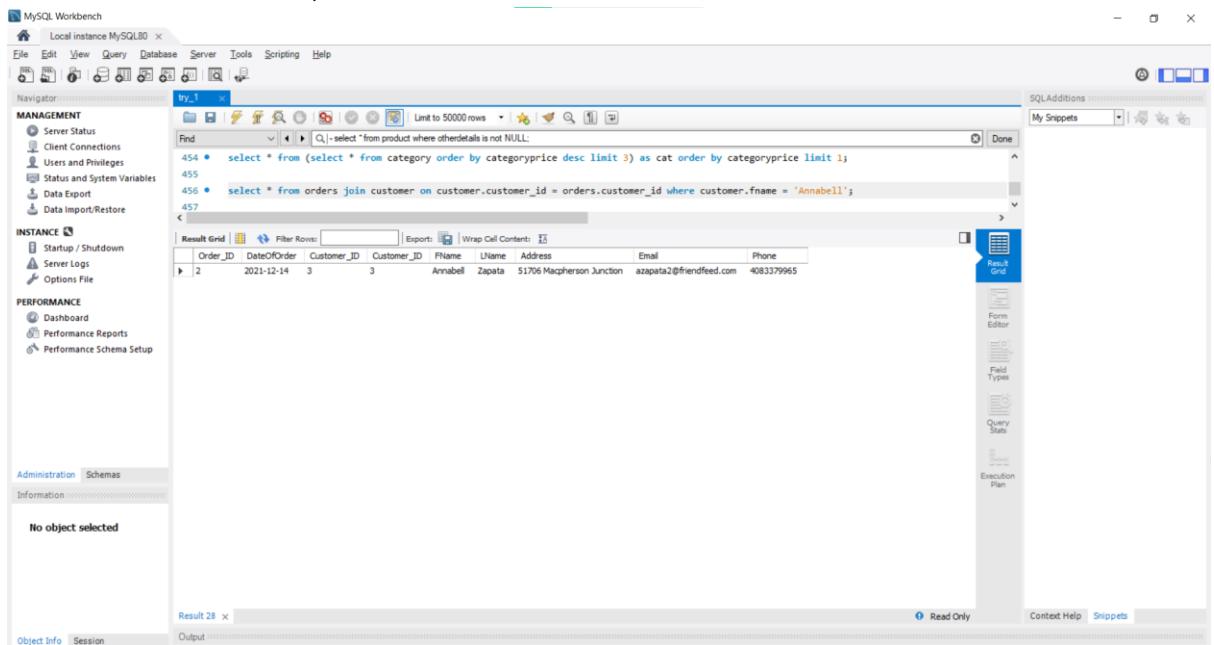
Tuple count = 1

## Select ... from ... join ... where queries

17. Show all the orders made by customer Annabell as first name.

```
select * from orders
```

```
join customer on customer.customer_id = orders.customer_id  
where fname = 'Annabell';
```



The screenshot shows the MySQL Workbench interface with a query editor and a result grid. The query editor contains the following code:

```
454 • select * from (select * from category order by categoryprice desc limit 3) as cat order by categoryprice limit 1;  
455 • 456 • select * from orders join customer on customer.customer_id = orders.customer_id where customer.fname = 'Annabell';  
457
```

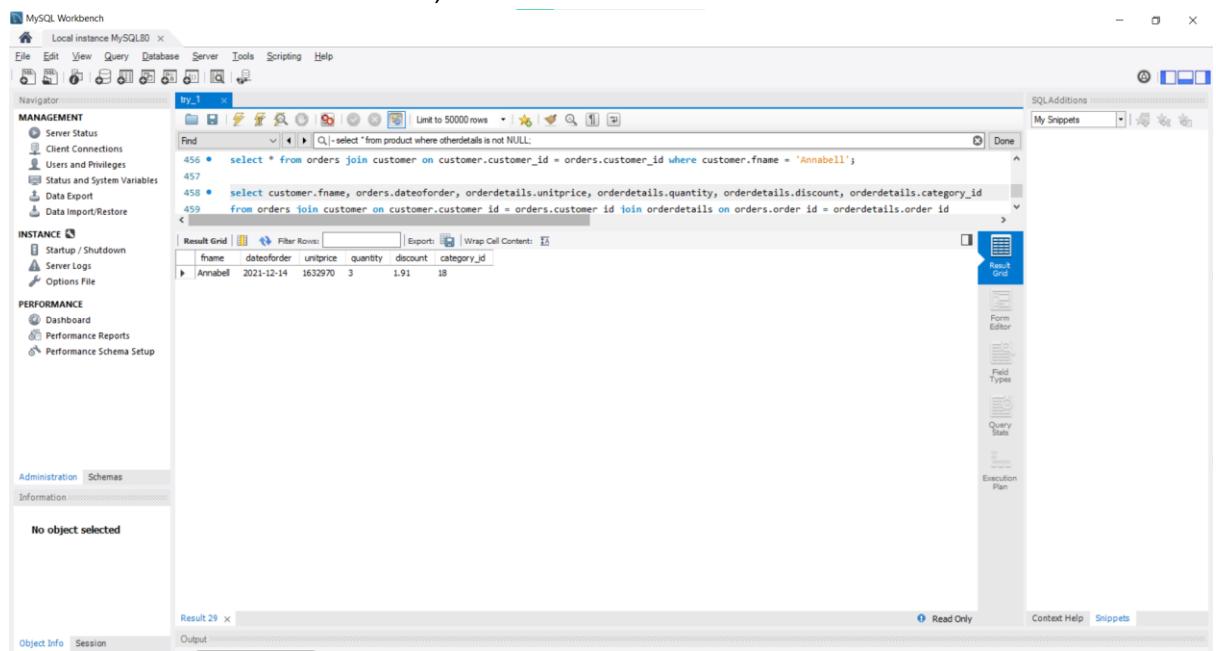
The result grid displays one row of data:

Order_ID	DateOfOrder	Customer_ID	Customer_ID	Fname	Lname	Address	Email	Phone
2	2021-12-14	3	3	Annabell	Zapata	51706 Macpherson Junction	azapata2@friendfeed.com	4083379965

Tuple count = 1

18. Show order history of Annabell with order details like name, dateoforder, category\_id, price, discount and quantity.

```
select customer.fname, orders.dateoforder, orderdetails.unitprice, orderdetails.quantity,
orderdetails.discount, orderdetails.category_id
from orders join customer on customer.customer_id = orders.customer_id
join orderdetails on orders.order_id = orderdetails.order_id
where customer.fname = 'Annabell';
```



The screenshot shows the MySQL Workbench interface with a query editor window titled 'try\_1'. The query is:

```
456 • select * from orders join customer on customer.customer_id = orders.customer_id where customer.fname = 'Annabell';
457
458 • select customer.fname, orders.dateoforder, orderdetails.unitprice, orderdetails.quantity, orderdetails.discount, orderdetails.category_id
459 from orders join customer on customer.customer_id = orders.customer_id join orderdetails on orders.order_id = orderdetails.order_id
<
```

The result grid shows one row of data:

fname	dateoforder	unitprice	quantity	discount	category_id
Annabell	2021-12-14	1632970	3	1.91	18

Below the result grid, the status bar displays 'Tuple count = 1'.

19. Show orders and their orderdetails in sorted order of order\_id.

```
select * from orderdetails
join orders on orders.order_id = orderdetails.order_id
order by orders.order_id;
```

OrderDetail_ID	UnitPrice	Quantity	Discount	DatedOrder	Category_ID	Order_ID	Bill_ID	Order_ID	DateOfOrder	Customer_ID
1	30936.70	2	1.32	2022-02-24	49	1	1	1	2022-02-24	50
2	35801.40	3	2.15	2022-02-24	1	1	1	1	2022-02-24	50
3	16329.70	3	1.91	2021-12-14	18	2	2	2	2021-12-14	3
4	27680.50	3	1.71	2022-02-03	17	3	3	3	2022-02-03	37
5	30936.70	1	1.32	2022-02-03	49	3	3	3	2022-02-03	37
6	48694.40	3	3.15	2022-01-19	33	4	4	4	2022-01-19	22
7	70250.00	2	2.95	2022-01-19	46	5	5	5	2022-01-19	26
8	844457.91	3	4.83	2022-06-20	19	5	5	5	2022-06-20	26
9	649788	1	0.84	2022-06-08	13	6	6	6	2022-06-08	47
10	464934.90	2	2.12	2022-06-08	35	6	6	6	2022-06-08	47
11	400372.0	3	4.96	2022-06-08	50	6	6	6	2022-06-08	47
12	334364.40	3	1.71	2022-06-08	7	6	6	6	2022-06-08	47
13	281155.0	2	2.15	2022-06-08	34	6	31	6	2022-06-08	47
14	489999.0	1	2.66	2022-03-12	29	7	7	7	2022-03-12	15
15	297399.0	3	1.71	2022-03-12	21	7	7	7	2022-03-12	15
16	181107.0	2	3.88	2022-04-28	44	8	8	8	2022-04-28	7
17	649788	2	0.84	2021-08-09	13	9	32	9	2021-08-09	25
18	200210.0	2	2.95	2022-06-03	6	10	10	10	2022-06-03	22
19	705635	1	2.95	2022-06-03	48	10	10	10	2022-06-03	22
20	474151.0	1	4.71	2021-12-28	8	11	11	11	2021-12-28	44
21	34186.10	3	2.12	2022-02-21	43	12	12	12	2022-02-21	45
22	234395.0	1	1.42	2021-09-03	9	13	13	13	2021-09-03	31
23	495093.0	3	4.61	2022-02-04	30	14	14	14	2022-02-04	22
24	326476.0	2	3.15	2022-02-04	4	14	14	14	2022-02-04	22
25	195715.0	1	2.12	2022-02-04	42	14	14	14	2022-02-04	22
26	276805.0	3	1.71	2021-12-14	17	15	15	15	2021-12-14	36

Tuple Count = 50

20. Show full name and date of return request for the customers who have returned their orders.

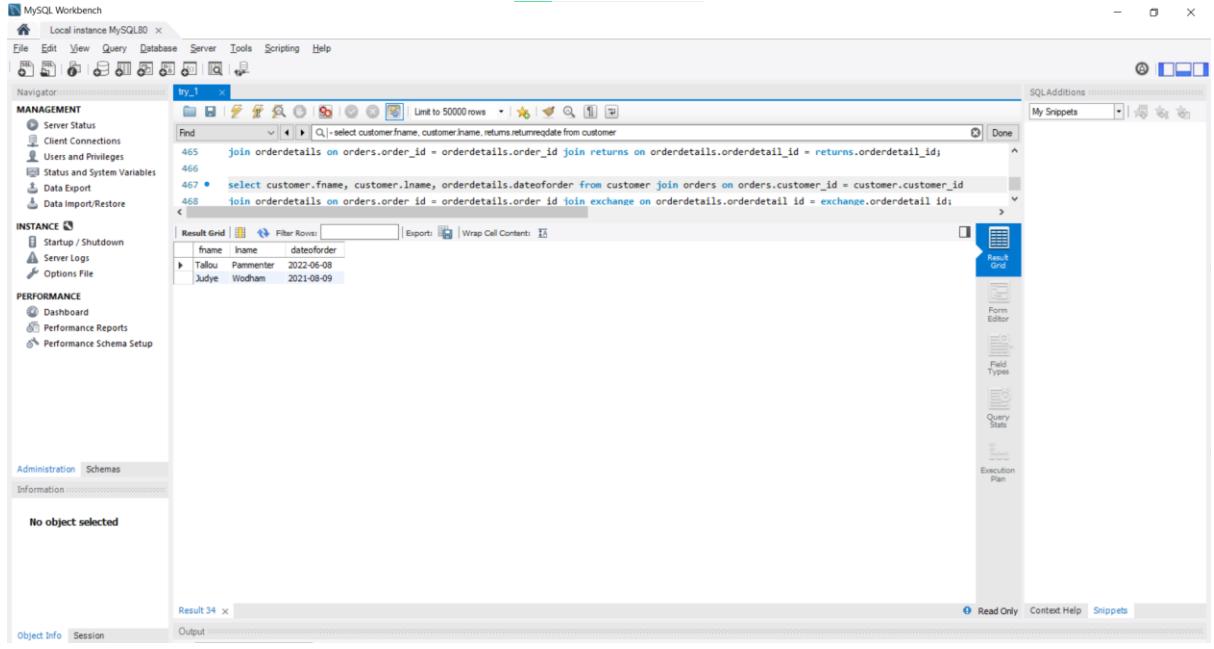
```
select customer.fname, customer.lname, returns.returnreqdate from customer
join orders on orders.customer_id = customer.customer_id
join orderdetails on orders.order_id = orderdetails.order_id
join returns on orderdetails.orderdetail_id = returns.orderdetail_id;
```

frame	lname	returnreqdate
Chesley	Johanning	2022-02-06
Hasheem	Kingsroad	2022-03-08

Tuple count = 2

21. Show name and date of exchange for the customers who have exchanged order.

```
select customer.fname, customer.lname, orderdetails.dateoforder from customer  
join orders on orders.customer_id = customer.customer_id  
join orderdetails on orders.order_id = orderdetails.order_id  
join exchange on orderdetails.orderdetail_id = exchange.orderdetail_id;
```

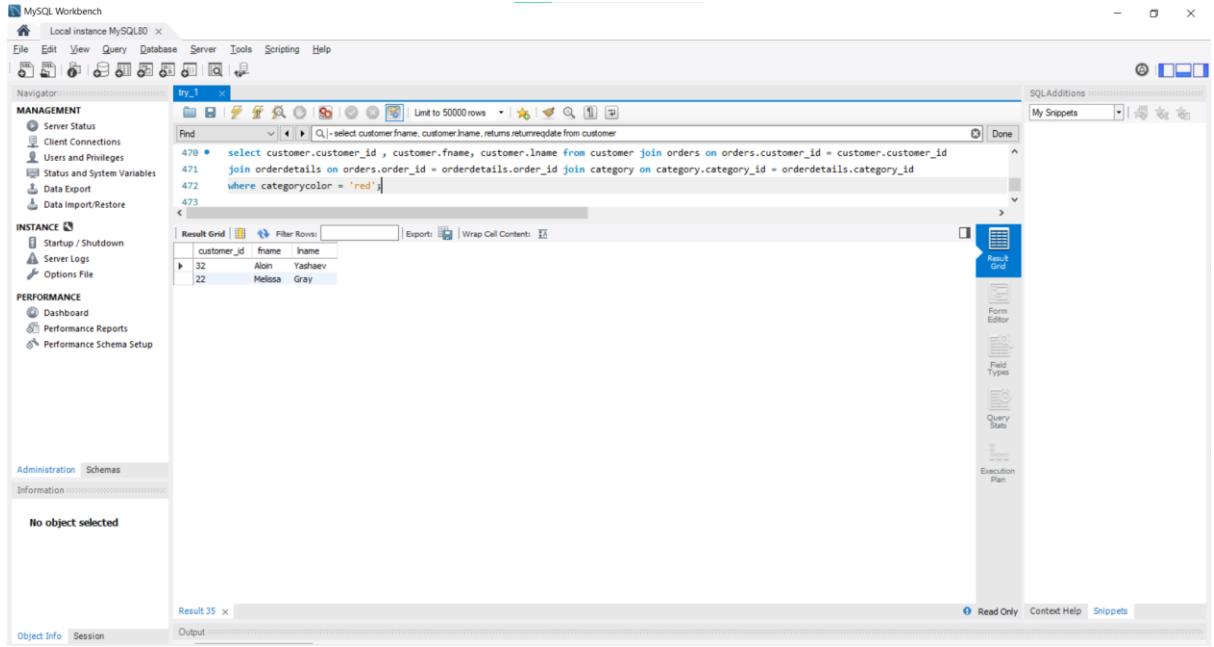


fname	lname	dateoforder
Talou	Pammenter	2022-06-08
Judy	Wodhan	2021-08-09

Tuple count = 2

22. Show the name and id of customers who have bought any product of red color.

```
select customer.customer_id , customer.fname, customer.lname from customer  
join orders on orders.customer_id = customer.customer_id  
join orderdetails on orders.order_id = orderdetails.order_id  
join category on category.category_id = orderdetails.category_id  
where categorycolor = 'red';
```



customer_id	fname	lname
32	Alon	Yashaev
22	Melissa	Gray

Tuple count = 2

23. Show product name which has been returned by customer.

```
select distinct(product.productname) from product  
join category on category.product_id = product.product_id  
join orderdetails on orderdetails.category_id = category.category_id  
join returns on returns.orderdetail_id = orderdetails.orderdetail_id;
```

The screenshot shows the MySQL Workbench interface. In the top-left pane, the Navigator displays various management, instance, and performance metrics. The central pane contains a SQL editor window titled 'wy\_1' with the following query:

```
473  
474 * select distinct(product.productname) from product join category on category.product_id = product.product_id  
475 join orderdetails on orderdetails.category_id = category.category_id join returns on returns.orderdetail_id = orderdetails.orderdetail_id;  
476
```

The 'Result Grid' tab is selected, showing the output:

productname
Nano

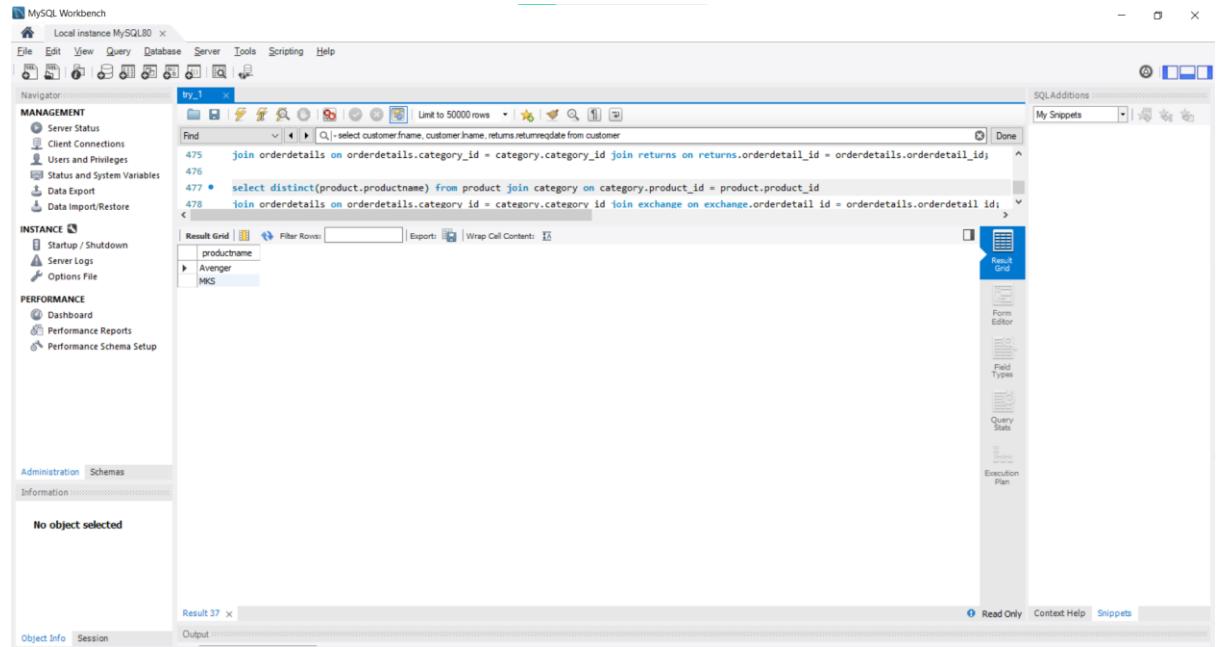
The bottom pane shows the result set count as 'Result 36 x'. The status bar at the bottom right indicates 'Read Only' and 'Context Help'.

Tuple Count = 1

This result is quite an amazing observation. Though data was created in random orders (Though some logic part was added) it can be seen that both the returned orders are on product name 'Nano'.

24. Show product name which has been exchanged by customer.

```
select distinct(product.productname) from product
join category on category.product_id = product.product_id
join orderdetails on orderdetails.category_id = category.category_id
join exchange on exchange.orderdetail_id = orderdetails.orderdetail_id;
```



```
MySQL Workbench - Local instance MySQL80
```

try\_1

```
475 join orderdetails on orderdetails.category_id = category.category_id join returns on returns.orderdetail_id = orderdetails.orderdetail_id;
476
477 • select distinct(product.productname) from product join category on category.product_id = product.product_id
478 join orderdetails on orderdetails.category_id = category.category_id join exchange on exchange.orderdetail_id = orderdetails.orderdetail_id;
```

productname
Avenger
MKS

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

No object selected

Result 37 x

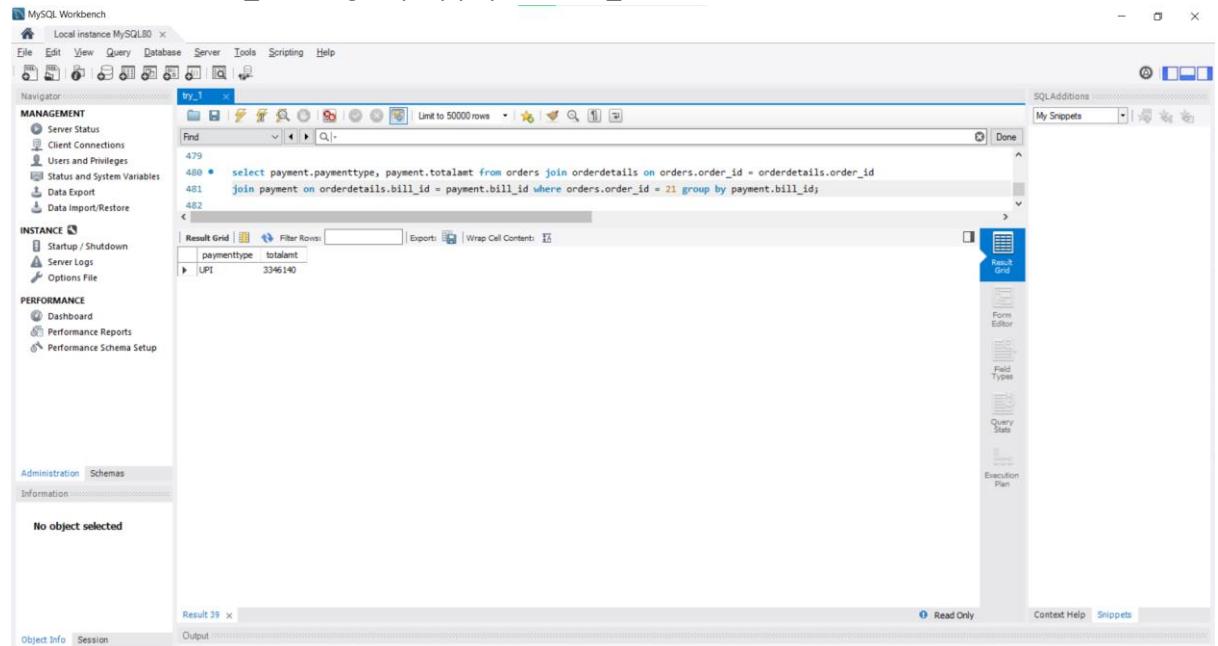
Object Info Session Output

Read Only Context Help Snippets

Tuple Count = 2

25. Show the paymenttype and total amt paid for order number 21.

```
select payment.paymenttype, payment.totalamt from orders
join orderdetails on orders.order_id = orderdetails.order_id
join payment on orderdetails.bill_id = payment.bill_id
where orders.order_id = 21 group by payment.bill_id;
```



```
MySQL Workbench - Local instance MySQL80
```

try\_1

```
479
480 • select payment.paymenttype, payment.totalamt from orders join orderdetails on orders.order_id = orderdetails.order_id
481 join payment on orderdetails.bill_id = payment.bill_id where orders.order_id = 21 group by payment.bill_id;
```

paymenttype	totalamt
(UPI)	3346140

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

No object selected

Result 39 x

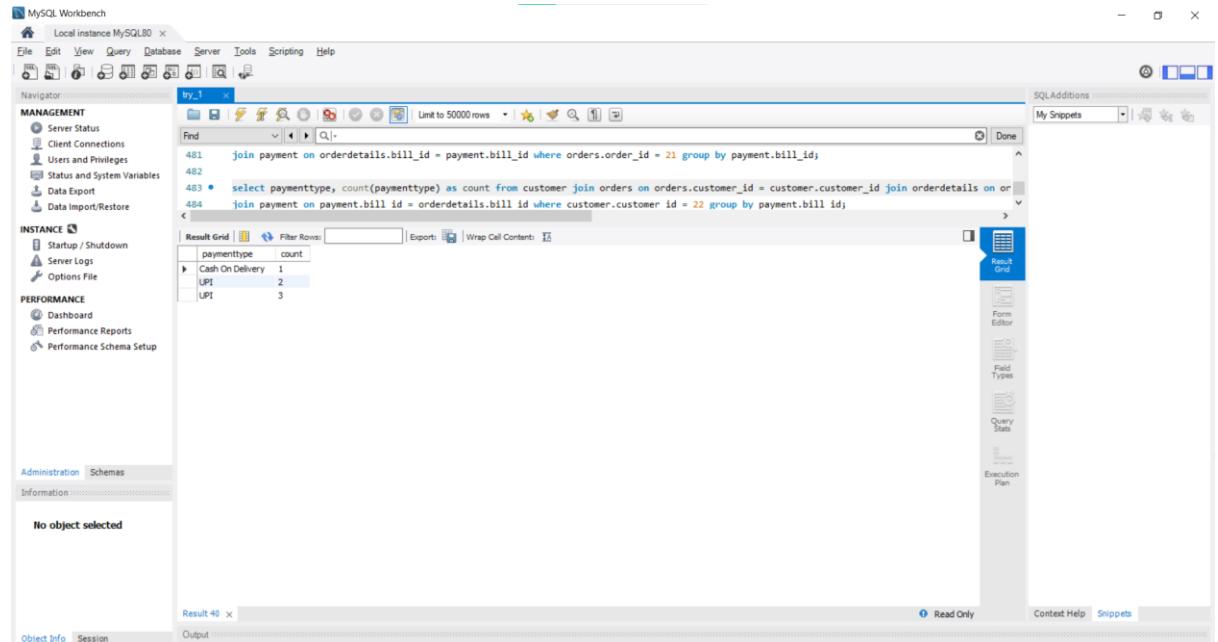
Object Info Session Output

Read Only Context Help Snippets

Tuple Count = 1

26. Show different type of payment with their count made by customer with id 22.

```
select paymenttype, count(paymenttype) as count from customer  
join orders on orders.customer_id = customer.customer_id  
join orderdetails on orders.order_id = orderdetails.order_id  
join payment on payment.bill_id = orderdetails.bill_id  
where customer.customer_id = 22  
group by payment.bill_id;
```



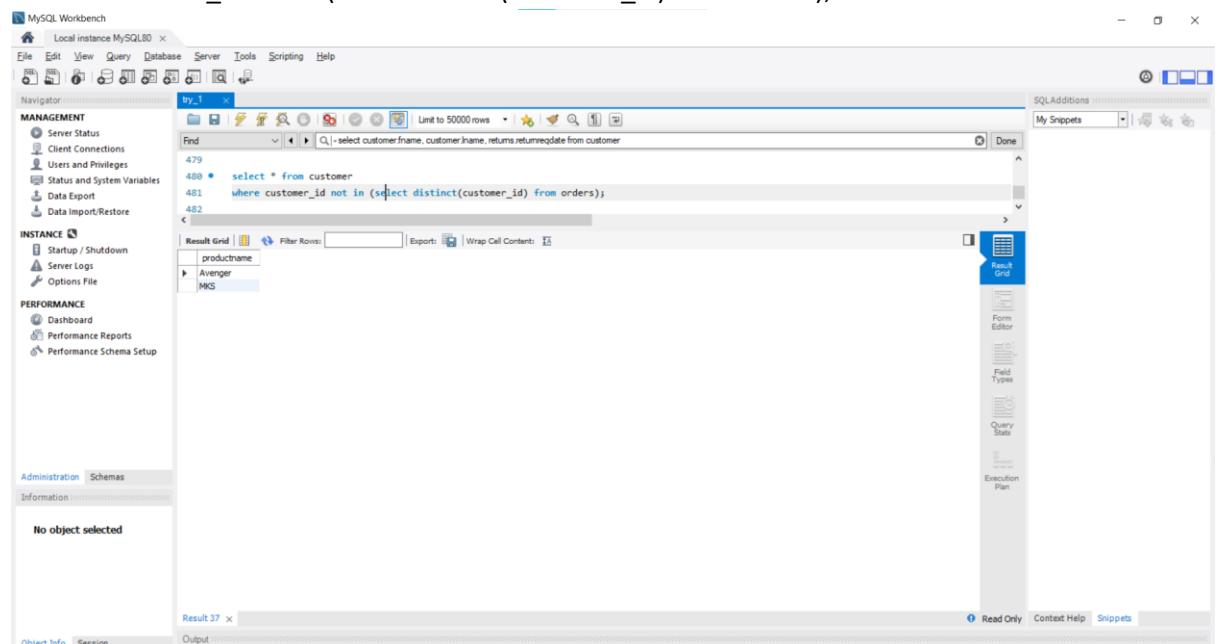
paymenttype	count
Cash On Delivery	1
UPI	2
UPI	3

Tuple Count = 2

Select ... from ... where ... not in queries

27. Show details of customers who have not ordered anything.

```
select * from customer  
where customer_id not in (select distinct(customer_id) from orders);
```



productname
Avenger
MKS

Tuple Count = 26

28. Show details of product which has no categories under it.

`select * from product`

`where product_id not in (select distinct(product_id) from category);`

The screenshot shows the MySQL Workbench interface with a query editor window titled 'try\_1'. The SQL code is:

```
481 where customer_id not in (select distinct(customer_id) from orders);
482
483 • select * from product
484 where product_id not in (select distinct(product_id) from category);
485
```

The 'Result Grid' tab is selected, displaying the following data:

Product_ID	ProductName	ProductDesc	Discount	OtherDetails	Supplier_ID
13	Aero	speed - 184, fuel - 10	3.48	SuperFast Speed coverage	9
17	Galant	speed - 174, fuel - 8	1.96	HULL	4
22	Pajero	speed - 170, fuel - 6	2.22	HULL	1
28	Hombre Space	speed - 123, fuel - 6	3.91	HULL	1
29	Atzak	speed - 128, fuel - 9	3.72	HULL	4
*	HULL	HULL	HULL	HULL	HULL

The 'SQLAdditions' panel on the right shows the following tabs: My Snippets, Result Grid (selected), Form Editor, Field Types, Query Stats, and Execution Plan.

Tuple Count = 5

## View Queries

29. Create and show view for customers and their orders made sorted with customer\_id.

create or replace view OrdersAndCustomers as

```
select customer.customer_id, customer.fname, customer.lname, customer.email,
```

```
customer.phone, orders.order_id, orders.dateoforder from customer
```

```
join orders on customer.customer_id = orders.customer_id
```

```
order by customer_id;
```

```
select * from ordersandcustomers;
```

The screenshot shows the MySQL Workbench interface with a query editor window titled 'try\_1'. The code area contains the SQL statements for creating and selecting from the 'OrdersAndCustomers' view. The results grid displays 50 rows of data, each representing a customer and their corresponding order details. The columns include customer\_id, fname, lname, email, phone, order\_id, and dateoforder. The data spans from March 2021 to April 2022.

customer_id	fname	lname	email	phone	order_id	dateoforder
3	Annabel	Zapata	azapata2@friendfeed.com	4083379965	2	2021-12-14
5	Amelita	Eccott	aeccott4@chron.com	7172087355	25	2022-04-20
7	German	Gest	ggest1@ca.gov	9563648350	8	2022-04-28
13	Ogdon	Nund	onundc@bzjournals.com	3196800174	18	2022-06-09
15	Rosamond	Linne	rlinne5@fingertoppost.com	8087310525	7	2022-03-12
15	Rosamond	Linne	rlinne5@fingertoppost.com	8087310525	20	2022-11-03
16	Norah	Brunndon	rbrunndon@surveymonkey.com	8614865694	30	2022-07-06
21	Marty	Player	mplayer6@networkdriveisling.org	6802447436	27	2021-12-19
22	Melissa	Gray	mgray4@barcamp.com	9704978801	4	2022-01-19
22	Melissa	Gray	mgray4@barcamp.com	9704978801	10	2022-06-03
22	Melissa	Gray	mgray4@barcamp.com	9704978801	14	2022-02-04
24	Adai	Dightor	adighthor@biglobe.ne.jp	4139055189	17	2022-02-25
25	Judy	Wodhar	jwodham@house.gov	5486665697	9	2021-08-09
26	Sauder	Beyn	sbeypn@unich.edu	6548268624	5	2022-06-20
28	Haseem	Kingsroad	hkingsroad1@cmu.edu	8761285130	29	2022-03-01
29	Geralit	Elderton	gelderton5@privacy.gov.au	3317698825	16	2021-08-20
31	Ingram	Ficken	ificken1@stumbleupon.com	6768697080	13	2021-09-03
32	Alon	Yashaev	ayashaev2@seattletimes.com	8081503726	21	2022-07-20
35	Zacharias	Buntn	zbuntny@google.com.au	1015095761	28	2022-02-09
36	Samuele	Wibsy	swibsy2@stanford.edu	3016396194	15	2021-12-14
36	Samuele	Wibsy	swibsy2@stanford.edu	3016396194	19	2021-12-30
36	Samuele	Wibsy	swibsy2@stanford.edu	3016396194	26	2022-05-23
37	Chelsey	Johanning	cjohanning10@desdev.cn	8628471245	3	2022-02-03
42	Brose	Biesty	bniesty12@drupal.org	2089669320	24	2022-01-25
44	Win	Langmuir	wlangmuir17@panzi.net	2365290313	11	2021-12-28
45	Eustace	Mixture	emixture18@usgs.gov	3956868050	12	2022-02-21

Tuple Count = 50

30. Create and show view orders, their orderdetails and payment method with required features.

```
create or replace view OrdersAndPayments as select orderdetails.orderdetail_id,
orders.order_id, orderdetails.dateoforder, orderdetails.category_id, orderdetails.bill_id,
orderdetails.unitprice, orderdetails.discount, orderdetails.quantity, payment.paymenttype,
payment.totalamt from orders
join orderdetails on orderdetails.order_id = orders.order_id
join payment on payment.bill_id = orderdetails.bill_id
order by orderdetail_id;
select * from ordersandpayments;
```

The screenshot shows the MySQL Workbench interface with a query editor window titled 'try\_1'. The code in the editor is:

```
499 payment.totalamt from orders join orderdetails on orderdetails.order_id = orders.order_id join payment
500 on payment.bill_id = orderdetails.bill_id order by orderdetail_id;
501 • select * from ordersandpayments;
502
```

The results grid displays 50 rows of data from the 'ordersandpayments' view. The columns are:

orderdetail_id	order_id	dateoforder	category_id	bill_id	unitprice	discount	quantity	paymenttype	totalamt
1	1	2022-02-24	49	1	3093670	1.32	2	Cash On Delivery	6556000
2	1	2022-02-24	1	1	3580140	2.15	3	Cash On Delivery	6556000
3	2	2021-12-14	18	2	1632970	1.91	3	UPI	6556000
4	3	2022-02-03	17	3	2768050	1.71	3	Cash On Delivery	1601780
5	3	2022-02-03	49	3	3093670	1.32	1	Cash On Delivery	1601780
6	4	2022-01-19	33	4	4869640	3.15	3	Cash On Delivery	5773540
7	5	2022-06-20	48	5	705635	2.95	2	DebitCard	5773540
8	5	2022-06-20	19	5	844371	4.03	3	DebitCard	5773540
9	6	2022-06-08	13	6	640788	0.84	1	Banking	4716250
10	6	2022-06-08	35	6	4649340	2.12	2	Banking	4716250
11	6	2022-06-08	50	6	4003720	4.96	3	Banking	4716250
12	6	2022-06-08	7	6	3343640	1.71	3	Banking	4716250
13	6	2022-06-08	34	31	2811550	2.15	2	UPI	9755620
14	7	2022-03-12	29	7	4899980	2.66	1	Banking	1495160
15	7	2022-03-12	21	7	2973990	1.71	3	Banking	1495160
16	8	2022-04-26	44	8	1811070	3.88	2	Cash On Delivery	1495160
17	9	2021-09-09	13	32	640788	0.84	2	Banking	9755620
18	10	2022-06-06	6	10	2600000	2.95	2	UPI	12277800
19	10	2022-06-03	48	10	705635	2.95	1	UPI	12277800
20	11	2021-12-28	8	11	4741510	1.71	1	UPI	12277800
21	12	2022-02-21	43	12	3418610	2.12	3	Banking	12277800
22	13	2021-09-03	9	13	3343590	1.42	1	DebitCard	3751100
23	14	2022-02-04	30	14	4650930	4.61	3	UPI	7692770
24	14	2022-02-04	4	14	3204760	3.15	2	UPI	7692770
25	14	2022-02-04	42	14	1957150	2.12	1	UPI	7692770
26	15	2021-12-14	17	15	2768050	1.71	3	DebitCard	7692770

Tuple Count = 50