**Project Work**

**FALL 2023**

**Pramodh Reddy Savasani**

**BANL-6430-04**

**DATABASE MANAGEMENT SYSTEM FOR BUSINESS ANALYTICS**

I am excited to collaborate on a real-time application, leveraging the knowledge gained throughout my course. For my final project, I have selected the Amazon chain system. Moving forward into the database design phase of this application, I have focused on practical considerations related to Amazon orders, selecting the following key tables.

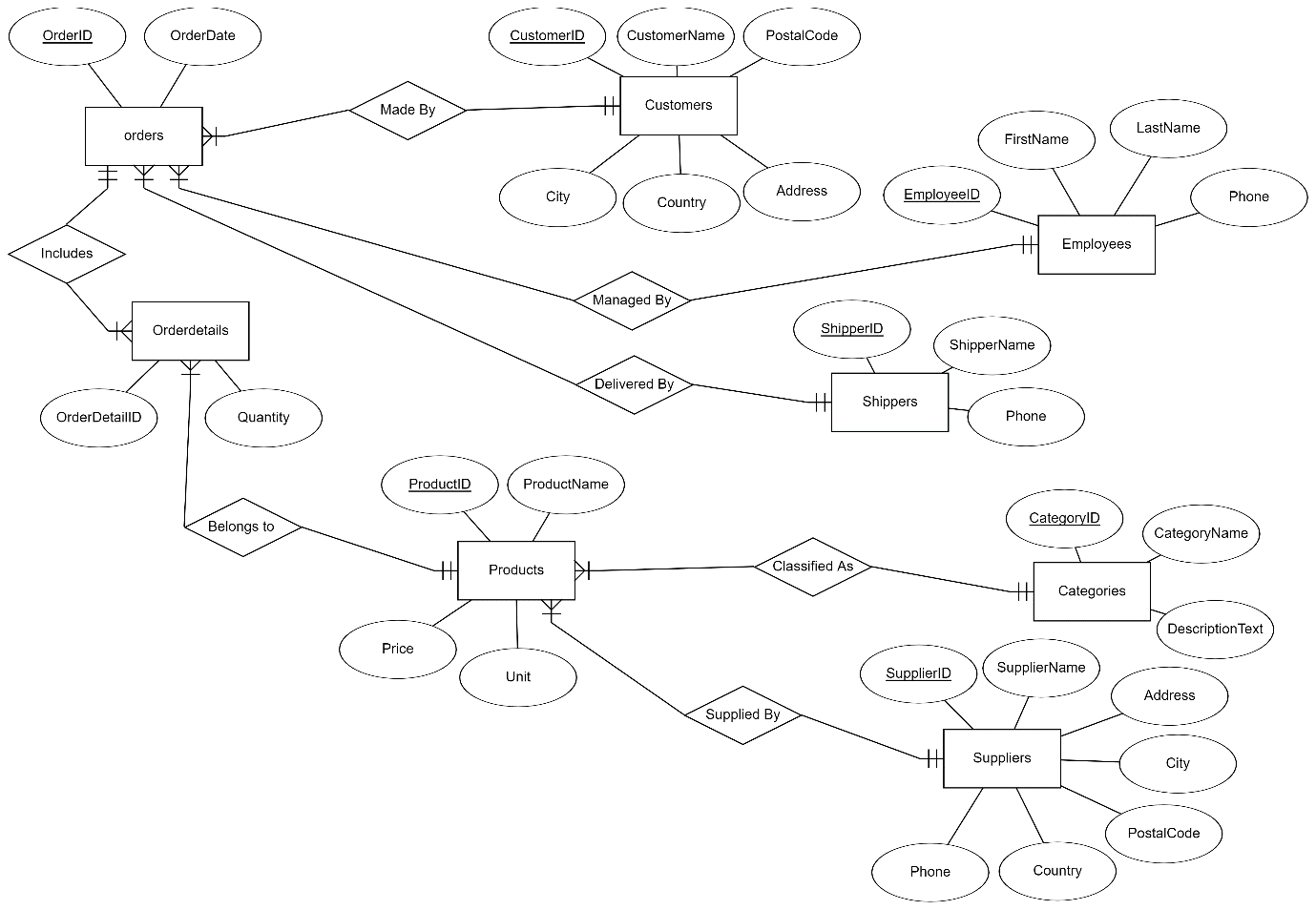
1. Orders
2. Order Details
3. Customers
4. Products
5. Employees
6. Shippers
7. Categories
8. Suppliers

I have intentionally limited my choice to eight tables. Through collaborative efforts, I have established the following relationships between these tables:

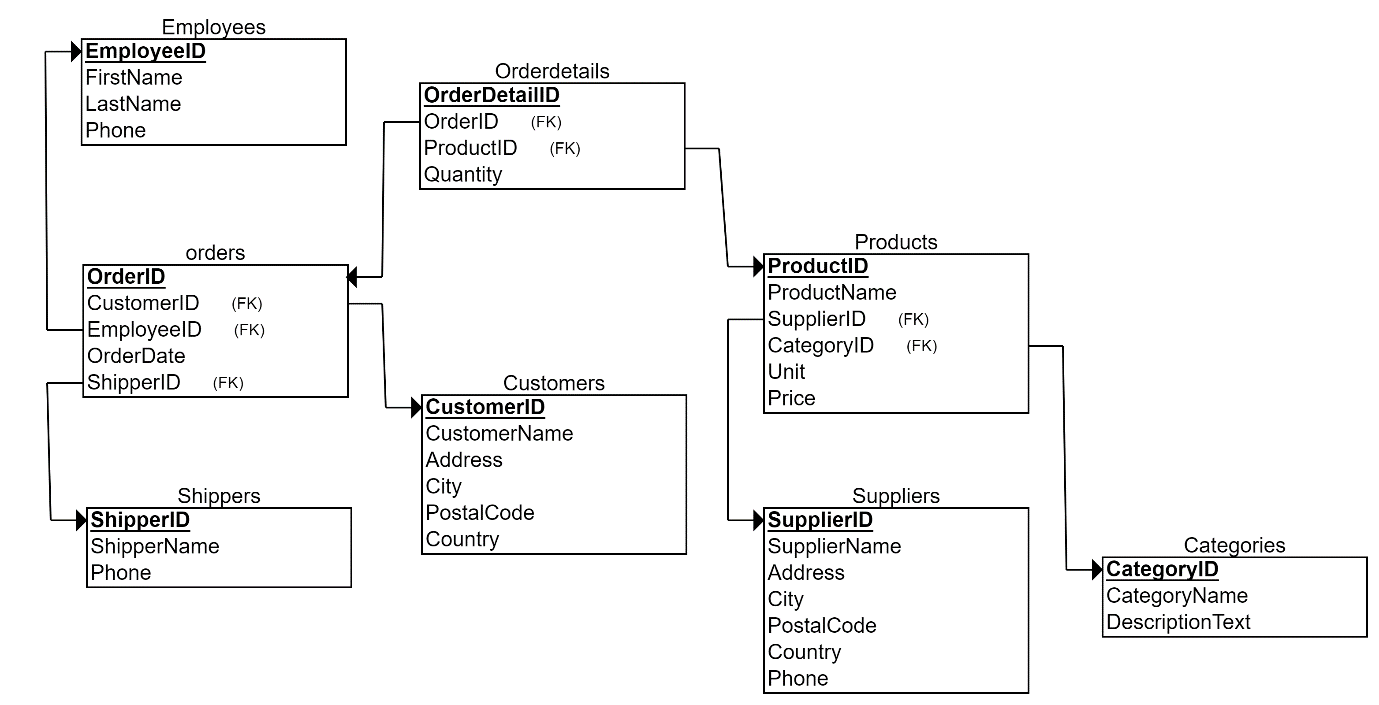
1. A customer can place many orders.
2. An order can only be placed by one customer.
3. An order can be processed by one employee.
4. An order is shipped by one shipper.
5. A shipper can ship many orders.
6. A product belongs to one category.
7. A category can have many products.
8. A product is supplied by one supplier.
9. A supplier can supply many products.

Choosing a dataset is a crucial decision in any project, and there could be various reasons for selecting the Amazon chain system as my dataset for the project. Here are some potential reasons for choosing this dataset:

1. Richness of Data: Amazon is a massive e-commerce platform with a diverse range of products, customers, and transactions. The dataset may provide a rich and varied set of data, allowing for comprehensive analysis and modeling.
2. Real-world Application: Analyzing an e-commerce system like Amazon is highly relevant in today’s business landscape. It provides insights into customer behavior, sales patterns, and inventory management, which are crucial for business analytics.
3. Complex Relationships: The relationships between entities like orders, customers, products, and suppliers in an e-commerce system can be intricate and dynamic. Designing a database that effectively models and manages these relationships is a challenging but valuable exercise.
4. Practical Experience: Working with a real-world dataset, especially from a well-known company like Amazon, can offer practical experience and insights into the challenges of managing large-scale databases for business analytics.
5. **The ERD Diagram for the above application is shown below:**



2**. The Schema for Amazon Sales along with field types and data type size is mentioned below:**



3) **SQL Code generated from Relational Schema:**

**CREATE** **TABLE** Customers

(

PostalCode **VARCHAR**(10) **NOT** **NULL**,

CustomerID **CHAR**(4) **NOT** **NULL**,

CustomerName **VARCHAR**(50) **NOT** **NULL**,

City **VARCHAR**(20) **NOT** **NULL**,

Address **VARCHAR**(80) **NOT** **NULL**,

Country **VARCHAR**(20) **NOT** **NULL**,

**PRIMARY** **KEY** (CustomerID)

);

**CREATE** **TABLE** Employees

(

LastName **VARCHAR**(40) **NOT** **NULL**,

FirstName **VARCHAR**(40) **NOT** **NULL**,

Phone **VARCHAR**(14) **NOT** **NULL**,

EmployeeID **CHAR**(4) **NOT** **NULL**,

**PRIMARY** **KEY** (EmployeeID)

);

**CREATE** **TABLE** Shippers

(

ShipperID **CHAR**(4) **NOT** **NULL**,

ShipperName **VARCHAR**(30) **NOT** **NULL**,

Phone **VARCHAR**(14) **NOT** **NULL**,

**PRIMARY** **KEY** (ShipperID)

);

**CREATE** **TABLE** Categories

(

CategoryID **CHAR**(4) **NOT** **NULL**,

CategoryName **VARCHAR**(20) **NOT** **NULL**,

DescriptionText **VARCHAR**(100) **NOT** **NULL**,

**PRIMARY** **KEY** (CategoryID)

);

**CREATE** **TABLE** Suppliers

(

SupplierID **CHAR**(4) **NOT** **NULL**,

SupplierName **VARCHAR**(80) **NOT** **NULL**,

Address **VARCHAR**(80) **NOT** **NULL**,

City **VARCHAR**(20) **NOT** **NULL**,

PostalCode **VARCHAR**(10) **NOT** **NULL**,

Country **VARCHAR**(20) **NOT** **NULL**,

Phone **VARCHAR**(14) **NOT** **NULL**,

**PRIMARY** **KEY** (SupplierID)

);

**CREATE** **TABLE** Products

(

ProductID **CHAR**(4) **NOT** **NULL**,

ProductName **VARCHAR**(80) **NOT** **NULL**,

Unit **VARCHAR**(50) **NOT** **NULL**,

Price **FLOAT** **NOT** **NULL**,

SupplierID **CHAR**(4) **NOT** **NULL**,

CategoryID **CHAR**(4) **NOT** **NULL**,

**PRIMARY** **KEY** (ProductID),

**FOREIGN** **KEY** (SupplierID) **REFERENCES** Suppliers(SupplierID),

**FOREIGN** **KEY** (CategoryID) **REFERENCES** Categories(CategoryID)

);

**CREATE** **TABLE** orders

(

OrderID **CHAR**(4) **NOT** **NULL**,

OrderDate **DATE** **NOT** **NULL**,

CustomerID **CHAR**(4) **NOT** **NULL**,

EmployeeID **CHAR**(4) **NOT** **NULL**,

ShipperID **CHAR**(4) **NOT** **NULL**,

**PRIMARY** **KEY** (OrderID),

**FOREIGN** **KEY** (CustomerID) **REFERENCES** Customers(CustomerID),

**FOREIGN** **KEY** (EmployeeID) **REFERENCES** Employees(EmployeeID),

**FOREIGN** **KEY** (ShipperID) **REFERENCES** Shippers(ShipperID)

);

**CREATE** **TABLE** Orderdetails

(

OrderDetailID **CHAR**(4) **NOT** **NULL**,

Quantity **INT** **NOT** **NULL**,

OrderID **CHAR**(4) **NOT** **NULL**,

ProductID **CHAR**(4) **NOT** **NULL**,

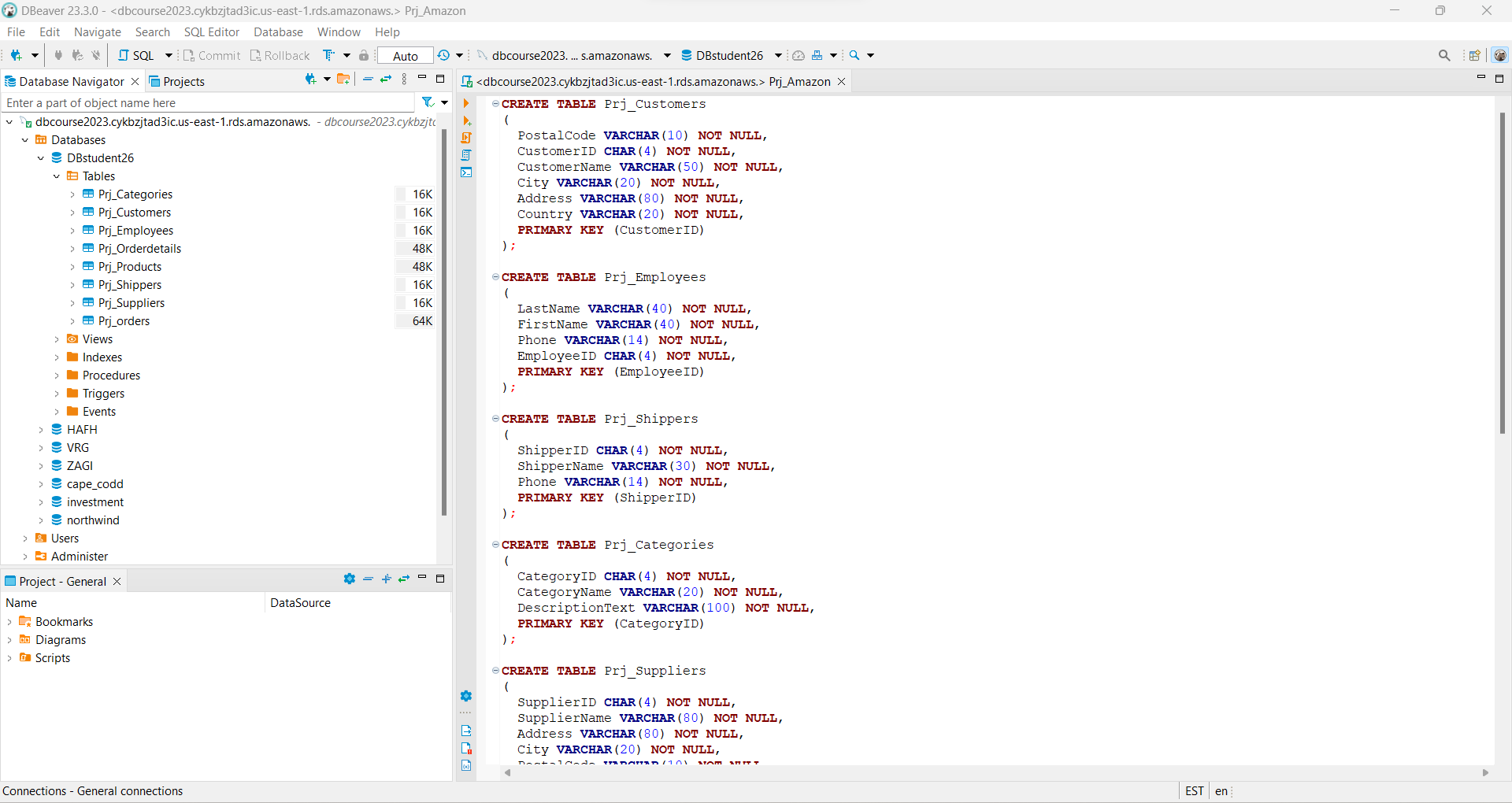
**PRIMARY** **KEY** (OrderDetailID),

**FOREIGN** **KEY** (OrderID) **REFERENCES** orders(OrderID),

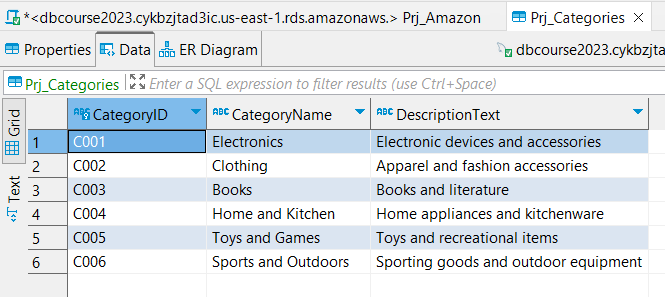
**FOREIGN** **KEY** (ProductID) **REFERENCES** Products(ProductID)

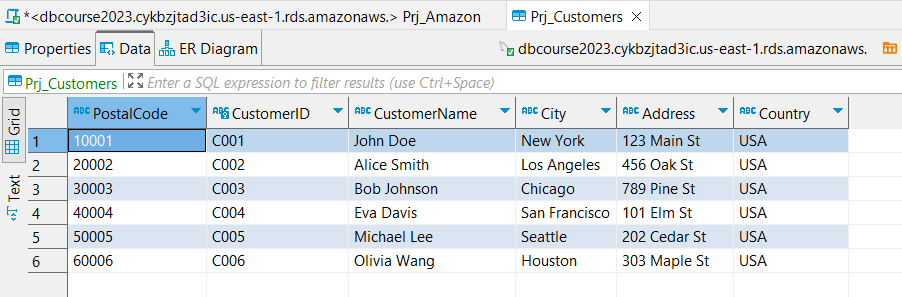
);

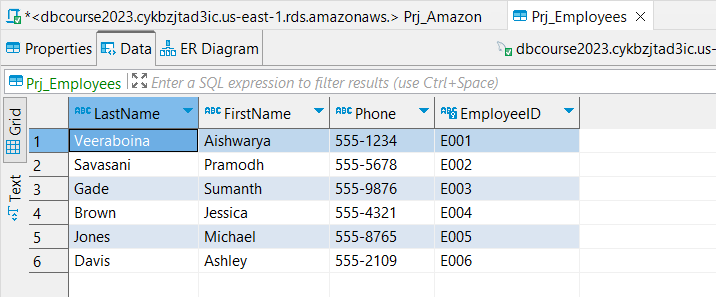
4) **After executing the SQL code by mentioning prj\_ as prefix for each table name, the database DBstudent26 is reflected with the tables as shown below:**

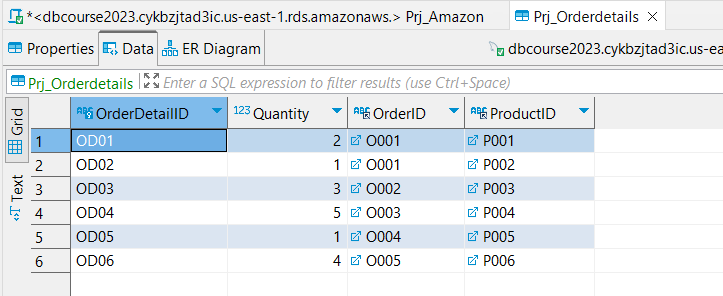


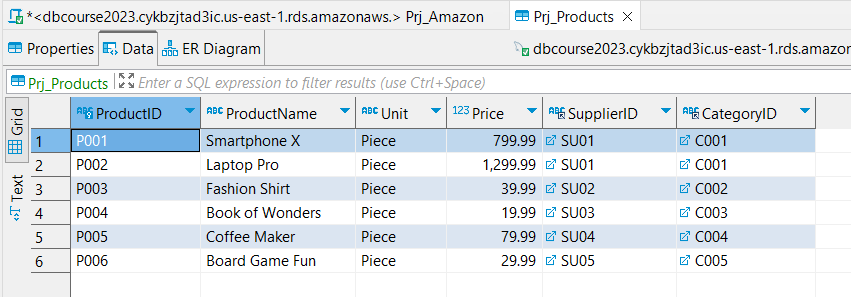
**5) After entering random values into the tables, they can be viewed as below:**

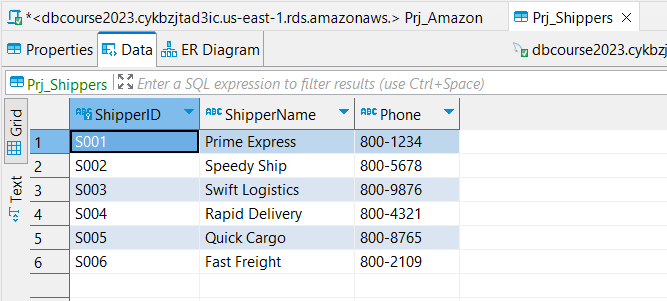


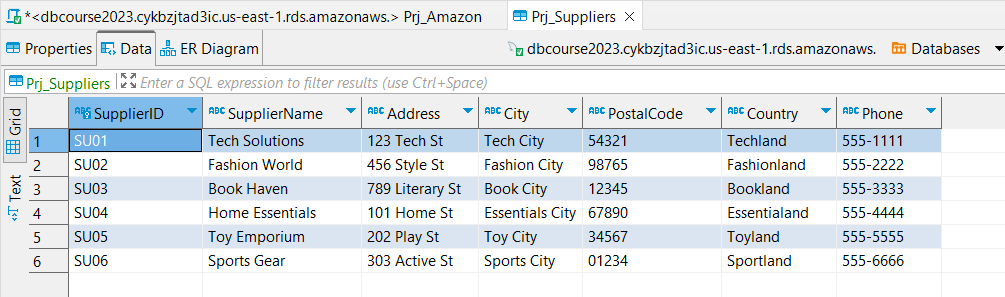


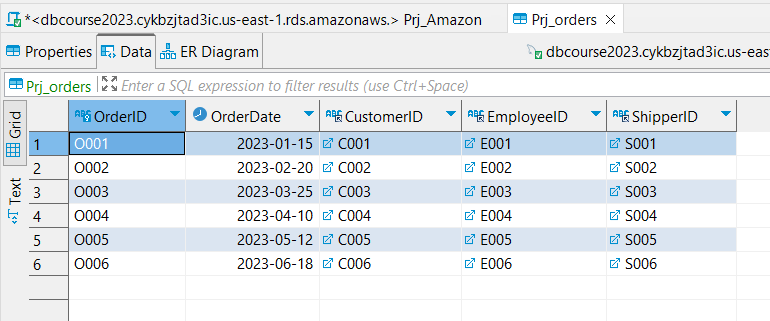












6) I have used the SUM aggregate function along with GROUP BY and ORDER BY functions. We have considered printing the total no of order items by inner joining **Order Details** with the **Products** table and again inner joining with the **Categories** table.

**SELECT**

PC.CategoryID,

PC.CategoryName,

**SUM**(POD.Quantity) **AS** TotalQuantityOrdered

**FROM**

Prj\_Orderdetails POD

**INNER** **JOIN** Prj\_Products PP **ON** POD.ProductID = PP.ProductID

**INNER** **JOIN** Prj\_Categories PC **ON** PP.CategoryID = PC.CategoryID

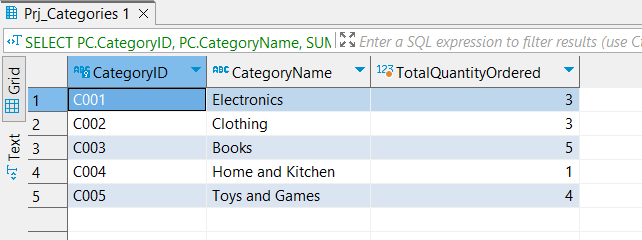
**GROUP** **BY**

PC.CategoryID, PC.CategoryName

**ORDER** **BY**

PC.CategoryID ;

This Query provides a list of unique product categories along with their IDs and names, showing the total quantity of items ordered for each category. The result set is organized by category ID in ascending order.



**7) The same question we tried to answer using R code, below is the R code:**

The below code creates a table in DBeaver as Prj\_Result by connecting to the SQL server using a user ID and password.

# Load the necessary libraries  
library(dplyr)

library(DBI)  
library(RMySQL)  
  
# Create a connection to the MySQL database  
con <- dbConnect(RMySQL::MySQL(),   
 user = "student26",   
 password = "deputy",   
 dbname = "DBstudent26",   
 host = "dbcourse2023.cykbzjtad3ic.us-east-1.rds.amazonaws.com")  
  
POD = tbl(con, "Prj\_Orderdetails")  
PP = tbl(con, "Prj\_Products")  
PC= tbl(con, "Prj\_Categories")  
  
# Perform the inner joins and aggregation in the database  
  
dbExecute(con, "  
 CREATE TABLE Prj\_Result AS  
 SELECT PC.CategoryID, PC.CategoryName, SUM(POD.Quantity) AS TotalQuantityOrdered  
 FROM Prj\_Orderdetails POD  
 INNER JOIN Prj\_Products PP ON POD.ProductID = PP.ProductID  
 INNER JOIN Prj\_Categories PC ON PP.CategoryID = PC.CategoryID  
 GROUP BY PC.CategoryID, PC.CategoryName  
")

## [1] 5

# Print the result  
result <- dbGetQuery(con, "SELECT \* FROM Prj\_Result")

## Warning in dbSendQuery(conn, statement, ...): Decimal MySQL column 2 imported  
## as numeric

print(result)

## CategoryID CategoryName TotalQuantityOrdered  
## 1 C001 Electronics 3  
## 2 C002 Clothing 3  
## 3 C003 Books 5  
## 4 C004 Home and Kitchen 1  
## 5 C005 Toys and Games 4

# Close the database connection  
dbDisconnect(con)

## [1] TRUE

The same 7th question can be answered in other ways the 2nd way to answer the 7th question using R studio is:

# Install and load the dplyr package  
  
library(dplyr)

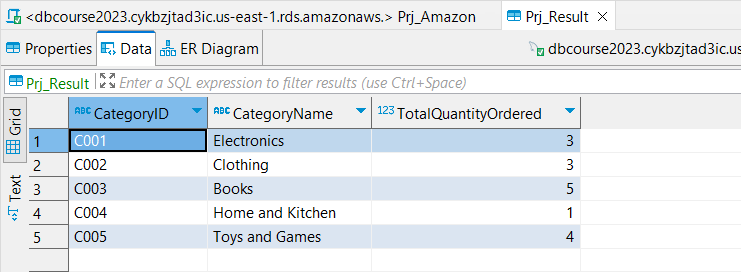
# Perform the inner joins and aggregation  
result <- Prj\_Orderdetails %>%  
 inner\_join(Prj\_Products, by = "ProductID") %>%  
 inner\_join(Prj\_Categories, by = "CategoryID") %>%  
 group\_by(CategoryID, CategoryName) %>%  
 summarise(TotalQuantityOrdered = sum(Quantity))

## `summarise()` has grouped output by 'CategoryID'. You can override using the  
## `.groups` argument.

# Print the result  
print(result)

## # A tibble: 5 × 3  
## # Groups: CategoryID [5]  
## CategoryID CategoryName TotalQuantityOrdered  
## <chr> <chr> <dbl>  
## 1 C001 Electronics 3  
## 2 C002 Clothing 3  
## 3 C003 Books 5  
## 4 C004 Home and Kitchen 1  
## 5 C005 Toys and Games 4

As the output of both codes is the same, we can either use any one of these.



**8) We have created a VIEW for top5ItemsSold by considering Products and Order Details Tables as shown below:**

-- Create a view of the top 5 products sold

**CREATE** **VIEW** Top5ProductsSold **AS**

**SELECT**

PP.ProductID,

PP.ProductName,

PP.Unit,

PP.Price,

**SUM**(POD.Quantity) **AS** TotalQuantitySold

**FROM**

Prj\_Orderdetails POD

**INNER** **JOIN** Prj\_Products PP **ON** POD.ProductID = PP.ProductID

**GROUP** **BY**

PP.ProductID, PP.ProductName, PP.Unit, PP.Price

**ORDER** **BY**

TotalQuantitySold **DESC**

**LIMIT** 5;

The view “Top 5 Products sold” is designed to provide a concise list of the top 5 best-selling products, including their ID, name, unit, price, and the total quantity sold, with the result set ordered by the total quantity sold in descending order.

