Practice Problem

21\_AIE\_304

Big Data Analysis– SEM-V

Professor – SanjanaSri Mam

Submitted By: Vikhyat Bansal [CB.EN.U4AIE21076]



Practice Problem - 1

Refer to product.csv and perform the following.

1. Count the total number of transactions in the dataset.

select count(\*) from product\_tbl;

OR

select count(transaction\_id) from product\_tbl;

OR

select count(\*) number\_of\_transactions from product\_tbl;

2. Calculate the total revenue generated from all transactions.

select sum(price\*quantity) Total\_Revenue from product\_tbl;

3. Find Product with highest price.

select product\_name,price from product\_tbl where price= (select max(price) from product\_tbl);

4. List the distinct products that were sold in the "Electronics" category.

select distinct(product\_name) from product\_tbl where category='Electronics';

5. Calculate the average price of products in each category .

select category,avg(price) from product\_tbl group by category;

6. Calculate the total quantity sold for each product.

select product\_name,sum(quantity) from product\_tbl group by product\_name;

7. Calculate the revenue generated for each month.

select (month(transaction\_date)),sum(price\*quantity) from product\_tbl group by month(transaction\_date);

8. List the top 5 customers who spent the most.

select customer\_id,sum(price\*quantity) as money\_spent from product\_tbl group by customer\_id order by money\_spent desc limit 0,5;

9. Find the products purchased by a specific customer.

select product\_name from product\_tbl where customer\_id = 1001;

OR

Change 1001 to any number XXXX to get a specific customer

10. List the quantity of products sold in each category.

select category,sum(quantity) from product\_tbl group by category;

11. Calculate the average quantity of products sold per transaction.

select avg(quantity) Quantity\_sold\_per\_transac from product\_tbl;

OR

select sum(quantity)/count(transaction\_id) from product\_tbl;

12. List products that were sold at least twice along with their total quantity sold.

select product\_name,category,quantity from product\_tbl where quantity > 1;

OR

select product\_name,sum(quantity) from product\_tbl group by product\_name having sum(quantity)>1;

13. Find customers who made purchases in both the "Electronics" and "Clothing" categories.

select customer\_id from product\_tbl where category = 'Electronics'

intersect

select customer\_id from product\_tbl where category = 'Clothing';

14. List products sold on a specific transaction date along with their quantity sold.

select product\_name,category,quantity from product\_tbl where transaction\_date = 20230701;

15. List the top N categories by total revenue generated.

select category,sum(quantity\*price) as revenue from product\_tbl group by category order by revenue desc limit 0,3;

Instead of 3 we can write N.

16. Find customers who have made more than one transaction along with the number of transactions.

select customer\_id,count(transaction\_id) as number\_transactions from product\_tbl group by customer\_id having number\_transactions > 1;

17. List products sold between a specific date range along with the quantity sold.

select product\_name,quantity from product\_tbl where transaction\_date between '20230701' and '20230713';

where dates can be changed before and after the BETWEEN keyword

18. Calculate the total revenue generated by each customer.

select customer\_id,sum(quantity\*price) as Revenue\_customerwise from product\_tbl group by customer\_id order by Revenue\_customerwise ;  
  
19. Calculate the percentage of revenue contributed by each category to the total revenue.

select category,(sum(quantity\*price\*100)/(select sum(quantity\*price) from product\_tbl)) as percentage\_contri from product\_tbl group by category;

20. Calculate the total quantity sold and total revenue for products in the low, medium, and high price ranges.

select sum(quantity) as Total\_quantity,sum(quantity\*price) as Total\_revenue, case when price between 0 and 100 then 'Low Range' when price between 99 and 300 then 'Medium Range' else 'High Range' end as Price\_ranges from product\_tbl group by Price\_Ranges;

{For getting the total quantity sold and total revenue from sold product in a price range}

OR

select price,sum(quantity) as Total\_quantity,sum(quantity\*price) as Total\_revenue, case when price between 0 and 100 then 'Low Range' when price between 99 and 300 then 'Medium Range' else 'High Range' end as Price\_ranges from product\_tbl group by price order by price;

{For getting each product price tag in a range with quantity and revenue earned from it.}

21. Count the number of products in each category from the products table

select category,count(product\_id) as Number\_of\_products from product\_tbl group by category;

22. Calculate the total sales for each product category from the sales table.

select category,sum(quantity) as Number\_of\_products from product\_tbl group by category;

23. Retrieve products from the products table whose names contain the word 'Laptop'.

select product\_id,category,price from product\_tbl where product\_name like %Laptop%;

OR

select \* from product\_tbl where product\_name like %Laptop%;

(For all the details regarding the product containing name Laptop).

24. Retrieve orders from the orders table placed between January 1, 2022, and December 31, 2022.

select \* from product\_tbl where transaction\_date between '20220101' and '20221231';

Refer to employee2.csv and perform the following.

1. Calculate the average salary of employees in each department with a salary greater than $40,000 from the employees table.

select dpt\_name,avg(salary) from emp\_tbl where salary>40000 group by dpt\_name;

2. Retrieve employees from the employees table who have a salary greater than the average salary.

select emp\_id,concat(emp\_fname,' ',emp\_lname) as emp\_name from emp\_tbl where salary>(select avg(salary) from emp\_tbl);

3. Retrieve employees from the employees table with a salary greater than $60,000 and job title is 'Manager'.

select emp\_id from emp\_tbl where salary>50000 intersect select emp\_id from emp\_tbl where job\_desc like '%Manager%' group by emp\_id;

4. Retrieve employees from the employees table whose last name starts with 'S'.

select \* from emp\_tbl where emp\_lname like 'S%';

5. Update the salary of an employee with employee\_id 101 to $55,000 in the employees table.

update emp\_tbl set salary=55000 where emp\_id = 101;

6. Retrieve employees from the employees table who are in the 'Sales' department and have a salary between $40,000 and $50,000.

select \* from emp\_tbl where dpt\_name = 'Sales' and salary between 40000 and 50000;

7. Retrieve employees from the employees table who have not been assigned to any department (department is NULL).

select \* from emp\_tbl where dpt\_name = '';

8. Retrieve employees' full names and a calculated column for their annual bonus (10% of salary) from the employees table.

select emp\_id,concat(emp\_fname, ' ',emp\_lname) as emp\_fullname,(salary + ((salary\*10)/100))

as bonus from emp\_tbl;

9. Retrieve distinct job titles from the employees table.

select distinct(dpt\_name) from emp\_tbl;

10. Retrieve employees from the employees table who are in the 'Sales' or 'Marketing' departments.

select \* from emp\_tbl where dpt\_name = 'Sales'

union

select \* from emp\_tbl where dpt\_name = 'Marketing';

11. Retrieve employees from the employees table who are in the 'Sales' and 'Marketing' departments.

select emp\_id,concat(emp\_fname,' ',emp\_lname) as emp\_fullname from emp\_tbl where dpt\_name = 'Marketing' intersect select emp\_id,concat(emp\_fname,' ',emp\_lname) as emp\_fullname from emp\_tbl where dpt\_name = 'Sales';

12. Retrieve employees from the employees table who are not in the 'Sales' department.

select \* from emp\_tbl where dpt\_name != 'Sales';

13. Retrieve employees from the employees table, ordered first by department in ascending order, and then by salary in descending order.

select \* from emp\_tbl order by dpt\_name,salary desc;

14. Retrieve employees' full names and a column indicating whether their salary is above $60,000 in the employees table.

select emp\_id,concat(emp\_fname,' ',emp\_lname) as emp\_fullname,salary, case when salary>60000 then 'Yes!!Salary is above threshold' else 'No!Salary is less' end as 'Threshold' from emp\_tbl;

15. You want to return values in multiple columns as one column. For example, you would like to produce this result set from a query against the EMP table:

CLARK WORKS IN IT

KING WORKS in HR

select concat(emp\_fname,' ',emp\_lname,' ','works in',' ',dpt\_name) as emp\_details from emp\_tbl;

16. Get the random records from but limit the size to .

Select \* from emp\_tbl order by rand() limit 5;

Inside rand parenthesis goes the seed for the rand function.

17. Return employee names and departments from the table employee and sort by the last two characters in the name field.

select concat(emp\_fname,' ',emp\_lname) as emp\_fullname,dpt\_name from emp\_tbl order by substring(emp\_fullname,-2);

OR

select concat(emp\_fname,' ',emp\_lname) as emp\_fullname,dpt\_name from emp\_tbl order by right(emp\_fullname,2);

18. Display the Full Name of the employee whose salary is maximum.

select concat(emp\_fname,' ',emp\_lname) as emp\_fullname,salary from emp\_tbl order by salary

desc limit 0,1;

OR

select concat(emp\_fname,' ',emp\_lname) as emp\_fullname,salary from emp\_tbl where salary = (select max(salary) from emp\_tbl);

19. Select an attribute of your choice and make a constraint in table if a value is missing by default, the attribute should be filled with a default value and not NULL.

alter table emp\_tbl alter salary set default 30000;

insert into emp\_tbl values(91,’lol’,’hmm’,’IT’,1,2,default,23);

Remember there is a diff between missing value and NULL value.

20. Get the products with invalid product\_id.

Select \* from product\_tbl where product\_id like “[0-9]….”  
  
Whatever type of pattern matching you want to do.

Practice Problem – 2

1. Create table that establish many to many relationship and explain the same.

CREATE TABLE Employee (

EmployeeID INT NOT NULL AUTO\_INCREMENT,

EmployeeName VARCHAR(100) NOT NULL,

PRIMARY KEY (EmployeeID)

);

CREATE TABLE SkillDescription (

SkillID INT NOT NULL AUTO\_INCREMENT,

SkillName VARCHAR(100) NOT NULL,

PRIMARY KEY (SkillID)

);

CREATE TABLE EmployeeSkill (

EmployeeID INT NOT NULL,

SkillID INT NOT NULL,

PRIMARY KEY (EmployeeID, SkillID),

FOREIGN KEY (EmployeeID) REFERENCES Employee(EmployeeID),

FOREIGN KEY (SkillID) REFERENCES SkillDescription(SkillID)

);

2. You are working for an online bookstore, and your manager has asked you to create a report that displays the discount amount for each book in the inventory based on its price. Books priced differently will have different discount rates applied to them.

Here are the requirements:

If the book's price is less than $20, apply a 10% discount.

If the book's price is between $20 and $50 (inclusive), apply a 20% discount.

If the book's price is greater than $50, apply a 30% discount.

Write an SQL query to generate a report that includes the book's title, original price, and the discounted price based on the criteria above.

SELECT title, price, CASE WHEN price < 20 THEN price \* 0.9 WHEN price >= 20 AND price <= 50 THEN price \* 0.8 ELSE price \* 0.7 END AS discounted\_price FROM books;

3. You are managing a student course registration system. Create a table called **Registrations** with the following attributes:

**student\_id (not unique, same id given to students of different semester,year and section)**

**course\_id**

**semester**

**year**

**section**

Perform the following

•Identify the super key(s) for the Registrations table.

Super key is a set of attributes that uniquely identifies each row in a table. In this case, the super key is the combination of student\_id, course\_id, semester, year, and section.

•Determine the candidate key(s) among the attributes.

Candidate key is a minimal set of attributes that can uniquely identify each row in a table. In this case, the candidate key is the combination of student\_id, course\_id, semester, year, and section.

•Specify which attribute(s) serve as the primary key.

Primary key is a candidate key that is chosen to uniquely identify each row in a table. In this case, the primary key is the combination of student\_id, course\_id, semester, year, and section.

• Explain the rationale behind your choices for candidate and primary keys.

The combination of student\_id, course\_id, semester, year, and section is the primary key because it uniquely identifies each row in the table. The student\_id alone is not sufficient to uniquely identify each row because the same student can register for the same course in different semesters, years, and sections.

4. The client wants you to create an efficient registration database. The client has come up with attributes, student\_id, semester, section,course,instructor name, department, first name, last name.

Hint: Perform normalisation over this, find keys and create table . Also explain your choice.

Let us assume that Primary Key for the above data is only student\_id taking that student\_id is unique for each student.

Now performing normalization on the above data, we get the following tables:  
From the above given attributes we knew that if the data is already in 1 NF, then we will look for 2 NF where attributes like instructor name,department,course are not dependent on primary key, so that is why we create separate table such as Course Table,Instructor Table,Registration Table, then finally if there exists some kind of transitive dependency we will look for it and if it doesn’t then we have our attributes in 3NF.

Student Table: student\_id, first\_name, last\_name

Primary Key of Student Table: student\_id

Foreign Key of Student Table: student\_id in Registration Table

Course Table: course\_id, course\_name, department

Primary Key of Course Table: course\_id

Foreign Key of Course Table: course\_id in Registration Table

Instructor Table: instructor\_id, instructor\_name, department

Primary Key of Instructor Table: instructor\_id

Foreign Key of Instructor Table: instructor\_id in Registration Table

Registration Table: student\_id, course\_id,instructor\_id

Primary Key of Registration Table: student\_id

Foreign Key of Registration Table: student\_id in Student Table, course\_id in Course Table, instructor\_id in Instructor Table

Student Class Detail Table: student\_id, semester, section

Primary Key of Student Class Detail Table: student\_id, semester, section

Foreign Key of Student Class Detail Table: student\_id in Student Table

5. Consider the following table representing a company's employee information:

**EmployeeID –unique to each employee, EmployeeName Department ManagerID, ManagerName (manager is also one of the employee) Salary HireDate.**

Normalize the table to at least 3NF. Pay special attention to the hierarchical relationship between employees and managers.

Employee Table: EmployeeID, EmployeeName, Department, ManagerID, ManagerName,Salary, HireDate

***In the above table the primary key is EmployeeID. The ManagerID is the foreign key referencing the EmployeeID in the same table. The ManagerName is the derived attribute.***

***The above table is in 1NF because all the attributes are atomic.***

***The above table is not in 2 NF because manager name is a derived attribute and it is dependent on the ManagerID. So, we need to remove the ManagerName attribute from the above table and create a new table.***

Employee Table: EmployeeID, EmployeeName, Department, ManagerID, Salary, HireDate

Manager Table: ManagerID, ManagerName

***The above tables are in 2NF because the ManagerName is dependent on the ManagerID and the ManagerID is the primary key of the Manager Table.***

***The above tables are not in 3NF because the Department attribute is dependent on the EmployeeID. So, we need to remove the Department attribute from the Employee Table and create a new table.***

Employee Table: EmployeeID, EmployeeName, ManagerID, Salary, HireDate

Manager Table: ManagerID, ManagerName

Department Table: EmployeeID, Department

***The above tables are in 3NF because the Department attribute is dependent on the EmployeeID and the EmployeeID is the primary key of the Employee Table.***

6. Imagine a university database that keeps track of courses, instructors, and student enrollments. The original table is as follows:

CREATE TABLE University ( CourseID INT, CourseName VARCHAR(50), InstructorID INT, InstructorName VARCHAR(50), StudentID INT, StudentName VARCHAR(50), Grade CHAR(1), PRIMARY KEY (CourseID, InstructorID, StudentID) );

Normalize the table to at least 3NF.

The primary key of the above table is (CourseID, InstructorID, StudentID).

The above tables are in 1NF because all the attributes are atomic.

The above tables are not in 2NF because the Grade attribute is dependent on the StudentID. So, we need to remove the Grade attribute from the above table and create a new table.

University Table: CourseID, CourseName, InstructorID, InstructorName, StudentID, StudentName

Grade Table: StudentID, Grade

The above tables are not in 3NF because the InstructorName attribute is dependent on the InstructorID. So, we need to remove the InstructorName attribute from the University Table and create a new table.

University Table: CourseID, CourseName, InstructorID, StudentID, StudentName

Instructor Table: InstructorID, InstructorName

Grade Table: StudentID, Grade

CREATE TABLE Instructor ( InstructorID INT, InstructorName VARCHAR(50), PRIMARY KEY (InstructorID) );

CREATE TABLE Grade ( StudentID INT, Grade CHAR(1), PRIMARY KEY (StudentID) );

CREATE TABLE University ( CourseID INT, CourseName VARCHAR(50), InstructorID INT, StudentID INT, StudentName VARCHAR(50), PRIMARY KEY (CourseID, InstructorID, StudentID), FOREIGN KEY (InstructorID) REFERENCES Instructor(InstructorID), FOREIGN KEY (StudentID) REFERENCES Grade(StudentID) );

7. You have a table representing library transactions:

CREATE TABLE LibraryTransactions (

TransactionID INT PRIMARY KEY,

BookID INT,

BookTitle VARCHAR(100), AuthorName VARCHAR(50),

MemberID INT,

MemberName VARCHAR(50),

CheckOutDate DATE,

ReturnDate DATE );

Check whether the table is normalized. If not, normalize the table to at least 3NF.

The primary key of the above table is TransactionID.

The above table is in 1NF because all the attributes are atomic i.e containing atmost one value.

The above table is not in 2NF because the BookTitle and AuthorName attributes are dependent on the BookID. So, we need to remove the BookTitle and AuthorName attributes from the above table and create a new table.

LibraryTransactions Table: TransactionID, BookID, MemberID, MemberName, CheckOutDate, ReturnDate

Book Table: BookID, BookTitle, AuthorName

The above tables are not in 3NF because the MemberName attribute is dependent on the MemberID. So, we need to remove the MemberName attribute from the LibraryTransactions Table and create a new table.

LibraryTransactions Table: TransactionID, BookID, MemberID, CheckOutDate, ReturnDate

Book Table: BookID, BookTitle, AuthorName

Member Table: MemberID, MemberName

Creating above table with appropriate foreign key constraints:

CREATE TABLE Book ( BookID INT, BookTitle VARCHAR(100), AuthorName VARCHAR(50), PRIMARY KEY (BookID) );

CREATE TABLE Member ( MemberID INT, MemberName VARCHAR(50), PRIMARY KEY (MemberID) );

CREATE TABLE LibraryTransactions ( TransactionID INT PRIMARY KEY, BookID INT, MemberID INT, CheckOutDate DATE, ReturnDate DATE, FOREIGN KEY (BookID) REFERENCES Book(BookID), FOREIGN KEY (MemberID) REFERENCES Member(MemberID) );

Practice Problem – 3

1. A) Extract the domain from the page\_url.

val data = Seq( (1, "https://example.com/spark/page1"),

(2, "https://example.com/spark/page2"),

(1, https://example.com/spark/page3),

(3, "https://example.com/hadoop/page1"),

(2, https://example.com/spark/page4),

(3, "https://example.com/spark/page5"),

(1, https://anotherdomain.com/page6),

(2, "https://anotherdomain.com/page7") )

// Create dataframe from the data

val df = data.toDF("user\_id", "page\_url")

// Extract the domain from the page\_url.

// Hint: Use the regexp\_extract function.

val df2 = df.withColumn("domain", regexp\_extract($"page\_url", "https?://([^/]+).\*", 1));

B) Count the number of visits for each user on each domain.

val df3 = df2.groupBy("user\_id", "domain").count()

C) Find the top domain for each user based on the number of visits.

// Without using row\_number(),PartitionBy().

val df4 = df3.groupBy("user\_id").agg(max(struct($"count", $"domain")).as("max")).select($"user\_id", $"max.domain", $"max.count")

// Using Window Partition By

import org.apache.spark.sql.expressions.Window

val df4 = df3.withColumn("rank", row\_number().over(Window.partitionBy("user\_id").orderBy($"count".desc)))

1. Suppose you have a CSV file named **shopping\_data.csv** with the following attributes: **customer\_id,product\_id,quantity,price.** Compute the below

**Total Spending per Customer:** Calculate the total spending for each customer and display the result.

// Create a DataFrame from the CSV file.

val df = spark.read.format("csv").option("header", "true").load("shopping\_data.csv")

; Total Spending per Customer: Calculate the total spending for each customer and display the result.

val df2 = df.groupBy("customer\_id").agg(sum($"quantity" \* $"price").as("total\_spending"))

**Most Purchased Product:** Identify the product that has been purchased the most and display its details.

val df3 = df.groupBy("product\_id").agg(sum($"quantity").as("total\_quantity"))

**Average Price per Product:** Calculate the average price for each product and display the result.

val df4 = df.groupBy("product\_id").agg((sum($"price")/sum($"quantity")).as("avg\_price"));

1. Suppose you have a DataFrame containing information about employees, and you want to add a new column called "performance\_category" based on the "performance\_score" column. The categorization should be as follows:

; 1. If the performance\_score is greater than or equal to 90, the performance\_category is "Excellent".

; 2. If the performance\_score is greater than or equal to 80 and less than 90, the performance\_category is "Good".

; 3. If the performance\_score is greater than or equal to 70 and less than 80, the performance\_category is "Average".

; 4. If the performance\_score is greater than or equal to 60 and less than 70, the performance\_category is "Poor".

val df = data.toDF("employee\_id", "employee\_name", "performance\_score")

val df2 = df.withColumn("performance\_category", when($"performance\_score" >= 90, "Excellent").when($"performance\_score" >= 80, "Good").when($"performance\_score" >= 70, "Average").when($"performance\_score" >= 60, "Poor").otherwise("Poor"))

1. Suppose you have a dataset of marketing campaign results with columns like "campaign\_id," "conversion\_rate," and "cost\_per\_conversion." The goal is to analyze the effectiveness of each campaign and calculate the overall marketing ROI.

; The sample data is

; val data = Seq( ("campaign\_1", 0.1, 50.0), ("campaign\_2", 0.15, 60.0), ("campaign\_3", 0.12, 55.0), ("campaign\_4", 0.2, 70.0), ("campaign\_5", 0.18, 65.0) )

; Create dataframe from the data

val df = data.toDF("campaign\_id", "conversion\_rate", "cost\_per\_conversion")

; Calculate the total number of conversions for each campaign.

val df2 = df.withColumn("campaign\_roi", $"conversion\_rate" \* $"cost\_per\_conversion")

1. Suppose you have a dataset of travel bookings with columns like "booking\_id," "destination," and "travel\_date." The goal is to identify popular travel destinations and analyze booking trends.

The sample data is

val data = Seq( ("booking\_1", "City A", "2023-01-01"), ("booking\_2", "City B", "2023-01-02"), ("booking\_3", "City A", "2023-01-03"), ("booking\_4", "City C", "2023-01-04"), ("booking\_5", "City B", "2023-01-05") )

; Create dataframe from the data

val df = data.toDF("booking\_id", "destination", "travel\_date")

; Find the top 3 most popular destinations.

val df2 = df.groupBy("destination").count().orderBy($"count".desc)

; Find the number of bookings for each destination by month.

val df3 = df.withColumn("month", month($"travel\_date")).groupBy("destination", "month").count().orderBy($"month".asc, $"count".desc).