

DELIMITER $$

CREATE PROCEDURE `myProcedure`(IN userID varchar(5))

BEGIN

select

OrderID,

case

when Amount < 0

then "Loss"

when Amount > 0

then "Profit"

when Amount = 0

then "Equivalet" end as "Profit/Loss" ,

abs(Amount) as Amount

from

(select

`order details`.OrderID,

ProductID as p,

(select sum(Quantity\*(select avg(Unitprice) from `order details` where (ProductID = p) group by ProductID)(1 - Discount) - Quantity\*UnitPrice(1 - Discount))) as Amount

from

`order details`

join

orders

on

CustomerID = userID && orders.orderid = `order details`.orderid

group by

`order details`.OrderID ) as ariel

order by cast(Amount as signed)

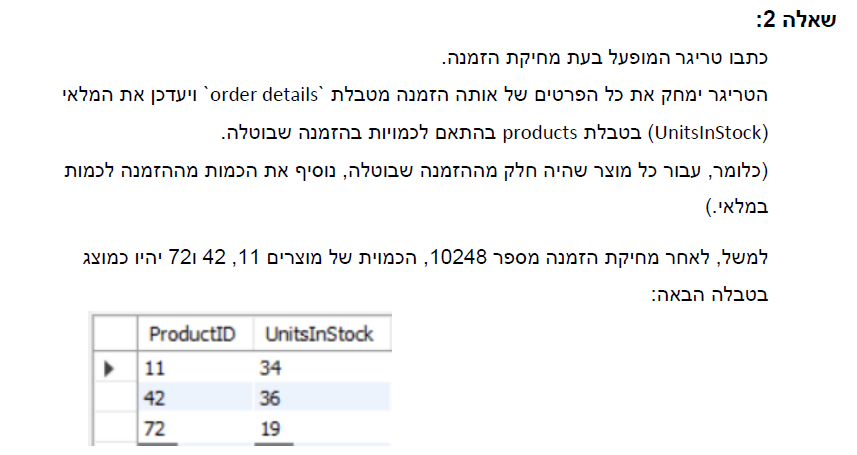
END $$

DELIMITER ;

1. א.

ב.

import java.sql.Connection;  
import java.sql.DriverManager;  
import java.sql.SQLException;  
import java.sql.ResultSet;  
import java.sql.Statement;  
import java.util.Scanner;  
  
public class main {  
 final static String *query* = "CALL `myProcedure`(param)";  
 final static String *Driver* = "com.mysql.cj.jdbc.Driver";  
 final static String *path\_DB* = "jdbc:mysql://localhost/northwind?serverTimezone=UTC";  
 final static String *userName* = "root";  
 final static String *passUser* = "123456789";  
 static String *ColNames* = "OrderID\t\t|\t\tProfit/Loss\t\t|\t\tAmount\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_";  
 Connection con = null;  
 static Statement *stmt* = null;  
 static int *numOfColumns*;  
  
  
 */\*\*  
 \* fixString - Fixing the string to the float variable  
 \** ***@param:*** *val - A value that represents the difference between the price paid and the average  
 \** ***@return:*** *str - A string that corrects the missing spaces for printing  
 \* \*/* private static String fixString(float val) {  
 String str = "" + val;  
 if(str.length() < 9){  
 for (int i = 0; i < (9-str.length()); i++) {  
 str += " ";  
 }  
 }  
 return str;  
 }  
 */\*\*  
 \* addValueToQuery - Add a variable into the query  
 \** ***@param:*** *val - A string representing the value of userID  
 \** ***@return:*** *str - A string representing the query  
 \* \*/* private static String addValueToQuery(String val){  
 return "CALL `myProcedure`('" +val+ "');";  
 }  
  
 */\*\*  
 \* getQuery - Receiving the query and receiving input from the user  
 \** ***@return:*** *str - A string representing the query including the thread of the values  
 \* \*/* private static String getQuery(){  
 System.*out*.println("Enter a CustomerID:");  
 Scanner sc = new Scanner(System.*in*);  
 return *addValueToQuery*(sc.nextLine());  
 }  
  
 */\*\*  
 \* connectAndRunQueryInDB - The function connects to a database and then calls a query,  
 \* and prints the returned results from the database  
 \* \*/* private static void connectAndRunQueryInDB() {  
 try {  
 Class.*forName*(*Driver*);  
 try (Connection con = DriverManager.*getConnection*(*path\_DB*, *userName*, *passUser*)) {  
 *stmt* = con.createStatement();  
 ResultSet rs = *stmt*.executeQuery(*getQuery*());  
 System.*out*.println("\n" + *ColNames*);  
 while (rs.next()) {  
 System.*out*.println(rs.getInt(1) + "\t\t|\t\t" + rs.getString(2) + "\t\t\t|\t\t" + *fixString*(rs.getFloat(3)) + "\t\t|\t\t");  
 }  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
 public static void main(String[] args) {  
 *connectAndRunQueryInDB*();  
 }  
}



DELIMITER $$

CREATE TRIGGER afterDelOrder

BEFORE DELETE

ON orders

FOR EACH ROW

BEGIN

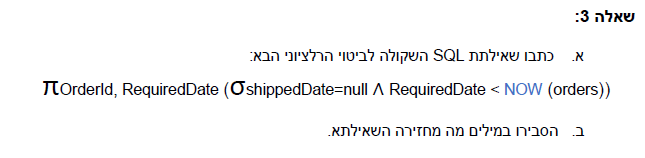
UPDATE products SET products.UnitsInStock = products.UnitsInStock +

(SELECT o.Quantity FROM `order details` o WHERE (o.orderID = OLD.orderID AND products.productID = o.productID));

DELETE FROM `order details` Where orderID = OLD.orderID;

END $$

DELIMITER ;



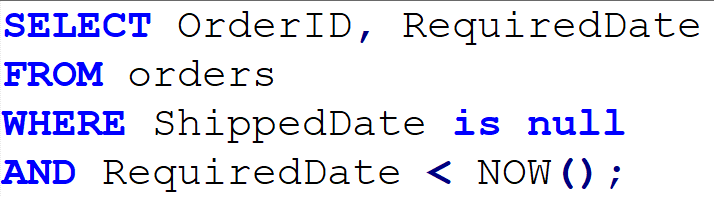
3.

א.

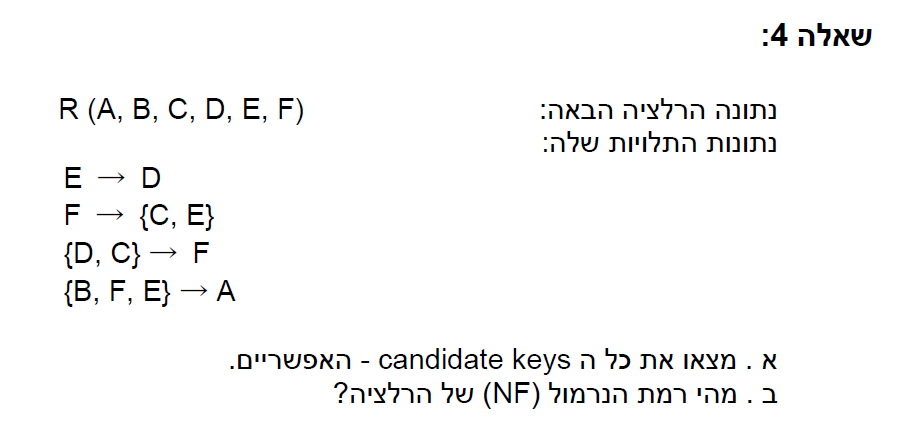
SELECT OrderID, RequiredDate

FROM orders

WHERE ShippedDate is null AND RequiredDate < NOW();



1. השאילתא מחזירה טבלה עם 2 עמודות (OrderID, RequiredDate) מתוך טבלת ההזמנות   
   (orders) איפה שהערך בעמודת תאריך השילוח היא null (ShippedDate is null) וגם עמודת התאריך המבוקש בהזמנה קטן מהתאריך שהשאילתא נשלחה בו (RequiredDate < NOW()).  
   כלומר להביא את כל ההזמנות שבתאריך השילוח קיים ערך null וגם שהתאריך המבוקש עבר.



4.

1. נבדוק קודם מי לא קיים לנו בצד ימין ולפי זה נדע אם הוא יכול להיות מפתח או לא.  
   המפתח חייב להכיל רק ערכים מ- (E,F,D,C,B) נבדוק עכשיו את השילובים שלהם ונראה מי מביא אותנו לכל הערכים:

BF -> C,E,D,A,F

BCE -> D,F,A,C,E

BCD -> F,C,E,D,A

מכאן ה- candidate key: {{BF},{BCE},{BCD}}

1. נניח נכונות עבור NF1.  
   נבדוק עבור NF2 שאומר האם ה-non-prime שלנו שתלויים בתת קבוצה של candidate key, A שהוא non-prime אינו תלוי בתת קבוצה של- candidate key מכיוון שהוא תלוי ב-F והוא לא חלק מה- candidate key ולכן מקיים את NF2.  
   כעת נבדוק האם מקיים את NF3 שאומר שלא קיימים non-prime שתלויים בתת קבוצה של SUPER-KEY? לא ולכן לא מקיים את NF3, ולכן נרד רמה אחת למטה.

**לכן אנחנו ברמת נירמול של NF2.**