**מסדי נתונים - מטלה 2**

מאיה חייט - 322515669

אביה אורן - 322273301

**QUESTION 1**

1. DELIMITER $$

CREATE DEFINER=`root`@`localhost` PROCEDURE `myPrecuder`(in cusID VARCHAR(5))

BEGIN

select b.OrderID ,case

when sum(amount)>0 then "Loss"

else "Profit"

end as `Profit/Loss` , ABS(sum(amount)) as Amount

from

(select OrderID , ((UnitPrice\*Quantity)-avgPrice\*Quantity) as amount from `order details`

join

(select ProductID , avg(UnitPrice) as avgPrice from `order details` group by ProductID) a

where a.ProductID = `order details`.ProductID

) b join orders

where b.OrderID = orders.OrderID and CustomerID = cusID

group by OrderID

order by OrderID DESC;

END $$

DELIMITER ;

1. package Task2;

import java.sql.\*;

import java.util.Scanner;

public class ex1 {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Set up the database connection

connection conn = null;

CallableStatement cstmt = null;

try {

Class.forName("com.mysql.cj.jdbc.Driver");

}

catch (Exception ex){ex.printStackTrace();}

try {

conn=DriverManager.getConnection("jdbc:mysql://localhost:3306/northwind", "root", "<password>");

// Get the customer ID from the user

System.out.println("write your ID:");

String cusID = scanner.nextLine();

// Call the stored procedure

cstmt = conn.prepareCall("{CALL `myPrecuder`(?)}");

cstmt.setString(1, cusID);

ResultSet rs = cstmt.executeQuery();

int numOfColumns = rs.getMetaData().getColumnCount();

// Print the results to the user

while (rs.next()) {

for (int col = 1; col <= numOfColumns; col++){

System.out.print(rs.getString(col) + " ");

}

System.out.println();

}

} catch (SQLException e) {

e.printStackTrace();

} finally {

try {

if (cstmt != null) cstmt.close();

if (conn != null) conn.close();

} catch (SQLException e) { e.printStackTrace(); }

}

scanner.close();

}

}

**QUESTION 2**

1. ALTER TABLE products

ADD COLUMN minUnitPrice DECIMAL(10, 4),

ADD COLUMN maxUnitPrice DECIMAL(10, 4);

b.

UPDATE products

SET minUnitPrice = (SELECT MIN(UnitPrice)  
 FROM `order details`  
 WHERE `order details`.ProductID = products.ProductID) ,

maxUnitPrice = (SELECT MAX(UnitPrice)   
FROM `order details`   
WHERE `order details`.ProductID = products.ProductID);

c.

DELIMITER $$

CREATE TRIGGER new\_product\_price

AFTER INSERT ON `order details`

FOR EACH ROW

BEGIN

IF NEW.UnitPrice< COALESCE((SELECT minUnitPrice FROM products WHERE productID = NEW.productID), NEW.UnitPrice+1 ) THEN

UPDATE products

SET minUnitPrice = NEW.UnitPrice

WHERE productID = NEW.productID;

ELSEIF NEW.UnitPrice > COALESCE((SELECT maxUnitPrice FROM products WHERE productID = NEW.productID), 0) THEN

UPDATE products

SET maxUnitPrice = NEW.UnitPrice

WHERE productID = NEW.productID;

END IF;

END $$

DELIMITER ;

**QUESTION 3**

π CustomerID, CompanyName ((π CustomerID, ProductID (orders ⋈ `order details`) ÷ π ProductID (products)) ⋈ customers)

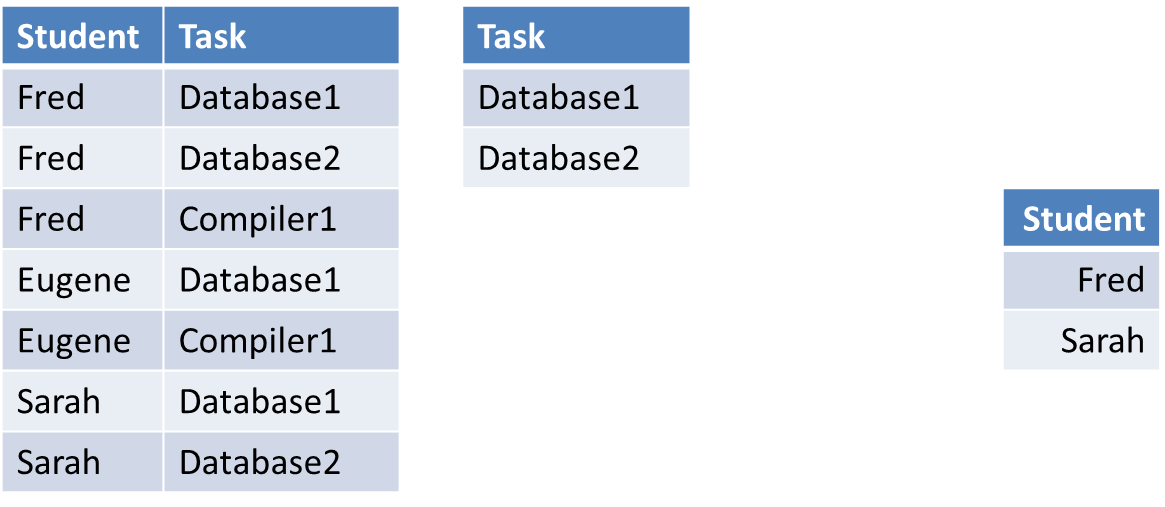
==

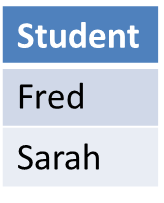
We’ll begin by translating the signs:

π - is called projection and means SELECT

⋈ - natural join and in our words INNER JOIN

÷ - is division for example if we have A÷B is like taking table A and kind of ignoring column, in the following example we have the Big\_Table ÷ Task\_table = Student\_Table





A.

SELECT customers.CustomerID, customers.CompanyName

FROM orders

JOIN `order details` ON orders.OrderID = `order details`.OrderID

JOIN products ON `order details`.ProductID = products.ProductID

JOIN customers ON orders.CustomerID = customers.CustomerID

group by customers.CustomerID, customers.CompanyName

having count (distinct products.ProductID) = (select count(\*) from products);

// the “having” makes :Only customers whose total of all the different products they ordered is equal to the number of all products//

B.

Display the ID number and company name of the customers who ordered all the products in products.

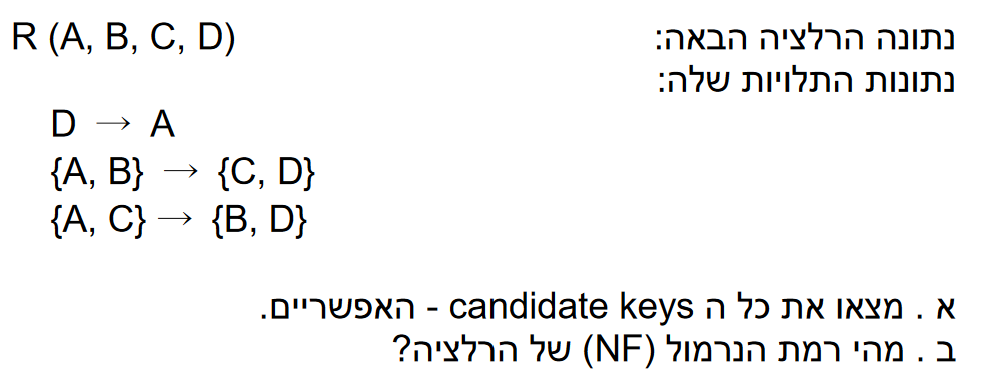
And a bit more detailed:

We begin with the natural (INNER) JOIN between the ‘order’ and ‘order details’ tables as this order is in the most inner brackets (which will use the OrderID column) and displaying the columns CustumerID and ProductID of the join table.

After having done that we divide the given table by the ProductID column from the ‘products’ table (which will result in removing that column leaving us with the CustomerID column), followed by the natural (INNER) join with the Customers table (using the ProductID column).

Next, we Select the CustomerID and CompanyName (columns) and display them. The GROUP BY statement groups the result set by the CustomerID and CompanyName columns from the customers table. The table given is empty as it should display all the customers (and company names) that ordered all kinds of products and no customers like this exist in our database.

**QUESTION 4**



CD gives us AB

BD gives us AC

A gives us D

A.

Candidate keys: { {A,B} , {A,C}, {B,D} , {C,D} }

B.

First we notice that all attributes are Prime

**1NF:**

Assuming it doesn’t have lists in the table 1nf is satisfied, since we don’t have the table, we say 1NF is satisfied.

**2NF:**

Since all attributes are Primes, we don’t have any non Primes therefore 2NF is definitely satisfied.

**3NF:**

Since all attributes are Primes, we don’t have any non Primes therefore 3NF is definitely satisfied.

**3.5NF:**

“Note: If Y is prime, and X→Y, and X is not a super-key, while the table might be in 3NF, it is not in BCNF” - From class,

In our case we have D -> A and we already know that even though A is Prime, D by itself isn’t a super key.

To conclude the Resolution is **3NF.**