UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Foundations of Databases, A.Y. 2020/2021
Master Degree in Computer Engineering
Master Degree in ICT for Internet and Multimedia
Homework 3 – Physical Design

Deadline December 16, 2020

Group FEDA	Project Sartoria FEDA	
Last Name Cecchinato	First Name Fabio	Student Number 1233602
Last Name Garbelotto	First Name Davide	Student Number 1234371
Last Name Grimaldi	First Name Alberto	Student Number 2026704
Last Name Lanza	First Name Enrico	Student Number 1232225

Variations to the Relational Schema

1. The cardinality of the entities 'Employee' and 'Tailor_shop' with the relationship 'Work' is now (0,N) instead of (1,N)

2. The cardinality of the entity 'Product' with 'Manage' is now (1,N) instead of (0,N), meaning that every product is managed by at least one employee

3. The meaning of 'Product' and 'Product_T' has changed: now the latter represents the catalogue, while the first one represents its physical realization

4. The relationship 'Request_quote' is now connected to 'Product_T' instead of 'Product' and has an attribute 'Quote'. One product can have more than one different quote

5. The 'Is' relationship has been renamed 'Has'

6. The 'Purchase' has been renamed 'Place'

7. The 'Order_supplier' entity is no longer storing any information on the raw material, which we assume not relevant for our purposes

8. Now, every customer is tracked in the system, even if it places his/her order in the physical shop. Therefore, the cardinality of 'Order_customer' in 'Place' is now (1,1)

9. The cardinality of ‘Product’ in ‘Contain’ is now (0,1) instead of (0,N), because we assume that each product is unique. Therefore, there is no attribute ‘Quantity’ in the relationship ‘Contain’ anymore

10. Every customer must be subscribed to a Tailor shop, hence the cardinality is (1,1)

11. Only primary keys in the ER schema are now underlined

12. Some of the ‘NOT NULL’ constraints for the attributes have changed

13. Since the ‘Review’ entity is linked to ‘Product’, only one review can be made for that specific product

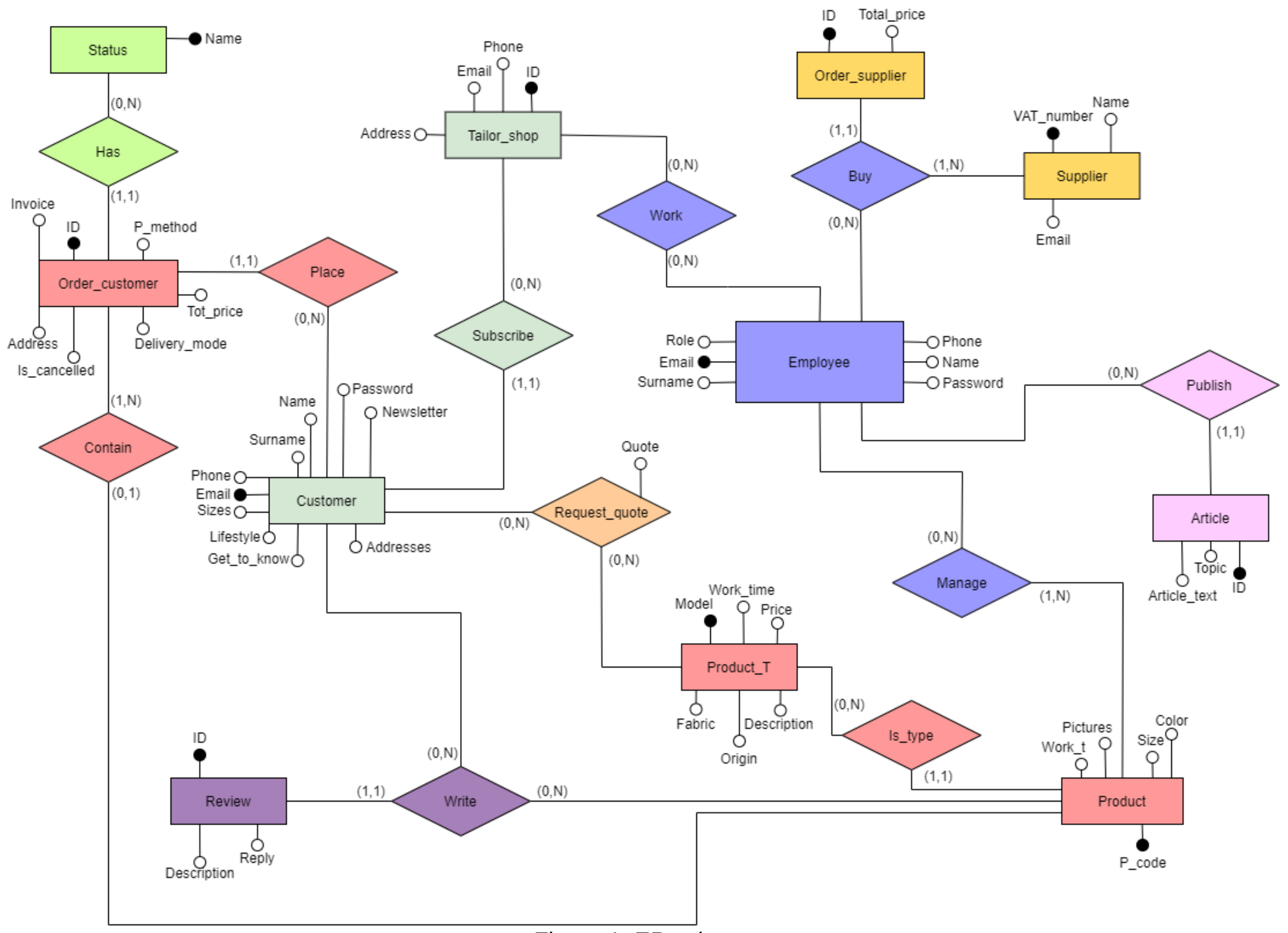


Figure 1: ER schema

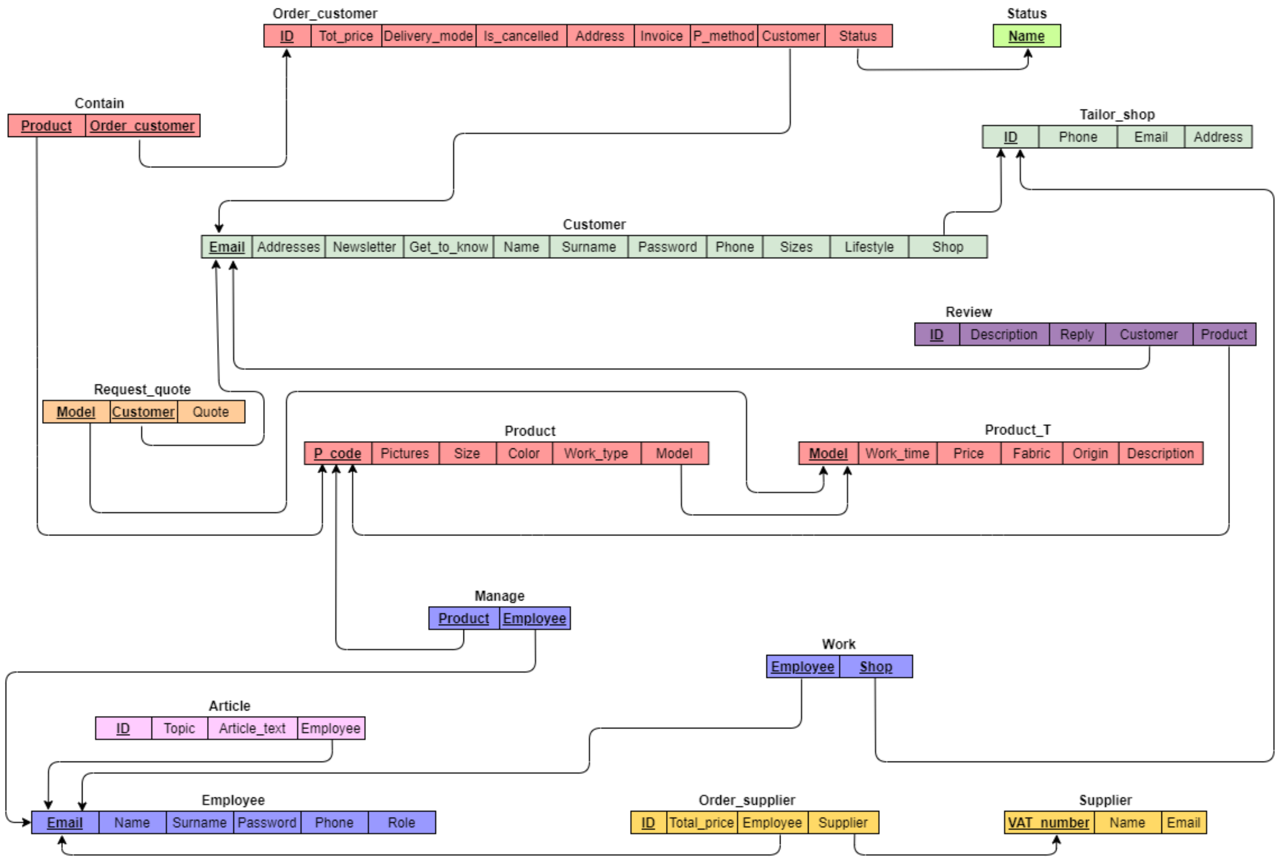


Figure 2: Relational schema

Physical Schema

In the following, the instructions to create the database are reported. They are reported also in the file `HW3-Feda_creation.sql`.

```
-- Create the Database
CREATE DATABASE feda_db OWNER postgres ENCODING = 'UTF8';

-- Connect to the new db
\c feda_db

-- Extension UUID
CREATE EXTENSION IF NOT EXISTS "uuid-ossp";

-- Extention Cryptography
CREATE EXTENSION IF NOT EXISTS pgcrypto;

-- Create new schema
DROP SCHEMA IF EXISTS Tailor_feda CASCADE;
CREATE SCHEMA Tailor_feda;

-- Create new domains
CREATE DOMAIN Tailor_feda.passwd AS VARCHAR(256)
        CONSTRAINT properpassword CHECK (((VALUE)::text ~* '[A-Za-z0-
9._%!]{8,}'::text));
```

```
CREATE DOMAIN Tailor_feda.emailaddress AS VARCHAR(256)
        CONSTRAINT properemail CHECK (((VALUE)::text ~* '^[A-Za-z0-9._%]+@[A-Za-
z0-9.]+[.][A-Za-z]+$'::text));

CREATE DOMAIN Tailor_feda.color AS CHAR(7)
        CONSTRAINT propercolor CHECK (((VALUE)::text ~* '[#][a-fA-F0-
9]{6}'::text));

CREATE DOMAIN Tailor_feda.vatnumber AS VARCHAR(15)
        CONSTRAINT properVAT CHECK (((VALUE)::text ~* '[A-Za-z0-9]{4,}'::text));

-- Create new data types
CREATE TYPE Tailor_feda.orderstatus AS ENUM (
        'Pending',
        'Accepted',
        'Processing',
        'Delivering',
        'Completed',
        'Refunded'
);

CREATE TYPE Tailor_feda.paymentmethod AS ENUM (
        'Cash',
        'Credit/debit card',
        'Bank transfer',
        'PayPal'
);

CREATE TYPE Tailor_feda.deliverymode AS ENUM (
        'Home',
        'Shop'
);

CREATE TYPE Tailor_feda.productorigin AS ENUM (
        'Customer',
        'Tailor'
);

CREATE TYPE Tailor_feda.worktype AS ENUM (
        'From scratch',
        'Tailor',
        'Online shop'
);

CREATE TYPE Tailor_feda.employeerole AS ENUM(
    'Employee',
    'Manager'
);

-- Create the tables

-- Tailor's shop
CREATE TABLE Tailor_feda.tailor_shop (
    id UUID DEFAULT uuid_generate_v4(),
    phone VARCHAR(15) NOT NULL,
    email Tailor_feda.EMAILADDRESS NOT NULL,
    address VARCHAR(128) NOT NULL,
    PRIMARY KEY (id)
);
```

```
-- Customer
CREATE TABLE Tailor_feda.customer (
    email Tailor_feda.EMAILADDRESS,
    name VARCHAR(64) NOT NULL,
    surname VARCHAR(64) NOT NULL,
    password Tailor_feda.PASSWD,
    phone VARCHAR(15) NOT NULL,
    addresses VARCHAR(128) ARRAY,
    newsletter BOOLEAN DEFAULT FALSE NOT NULL,
    get_to_know VARCHAR(64),
    sizes VARCHAR(64),
    lifestyle VARCHAR(64),
    shop UUID NOT NULL,
    PRIMARY KEY (email),
    FOREIGN KEY (shop) REFERENCES Tailor_feda.tailor_shop (id)
);

-- Status
CREATE TABLE Tailor_feda.status (
        name Tailor_feda.ORDERSTATUS,
        PRIMARY KEY (name)
);

-- Order_customer
CREATE TABLE Tailor_feda.order_customer (
        id UUID DEFAULT uuid_generate_v4(),
        tot_price NUMERIC(8,2) NOT NULL,
        address VARCHAR(128) NOT NULL,
        p_method Tailor_feda.PAYMENTMETHOD NOT NULL,
        delivery_mode Tailor_feda.DELIVERYMODE NOT NULL,
        is_cancelled BOOLEAN DEFAULT FALSE NOT NULL,
        invoice VARCHAR NOT NULL UNIQUE,
        customer Tailor_feda.EMAILADDRESS NOT NULL,
        status Tailor_feda.ORDERSTATUS NOT NULL,
        PRIMARY KEY (id),
        FOREIGN KEY (customer) REFERENCES Tailor_feda.customer(email),
        FOREIGN KEY (status) REFERENCES Tailor_feda.status(name)
);

-- Product type
CREATE TABLE Tailor_feda.product_t (
        model VARCHAR(128),
        work_time SMALLINT CHECK (work_time>0),
        price NUMERIC(8,2) CHECK(price>=0),
        fabric VARCHAR(32),
        origin Tailor_feda.PRODUCTORIGIN NOT NULL,
        description TEXT NOT NULL,
        PRIMARY KEY (model)
);

-- Product
CREATE TABLE Tailor_feda.product (
        p_code UUID DEFAULT uuid_generate_v4(),
        pictures BYTEA ARRAY,
        size VARCHAR(64) NOT NULL,
        color Tailor_feda.COLOR,
        work_type Tailor_feda.WORKTYPE NOT NULL,
        model VARCHAR(128) NOT NULL,
        PRIMARY KEY (p_code),
        FOREIGN KEY (model) REFERENCES Tailor_feda.product_t(model)
);
```

```sql
-- Request quote
CREATE TABLE Tailor_feda.request_quote (
    model VARCHAR(128),
    customer Tailor_feda.EMAILADDRESS,
    quote NUMERIC(8,2) CHECK (quote>=0),
    PRIMARY KEY (model, customer),
    FOREIGN KEY (model) REFERENCES Tailor_feda.product_t(model),
    FOREIGN KEY (customer) REFERENCES Tailor_feda.customer(email)
);

-- Contain
CREATE TABLE Tailor_feda.contain (
        order_customer UUID,
        product UUID,
        PRIMARY KEY (order_customer, product),
        FOREIGN KEY (order_customer) REFERENCES Tailor_feda.order_customer(id),
        FOREIGN KEY (product) REFERENCES Tailor_feda.product(p_code)
);

-- Employee
CREATE TABLE Tailor_feda.employee (
    email Tailor_feda.emailaddress,
    name VARCHAR(64) NOT NULL,
    surname VARCHAR(64) NOT NULL,
    password Tailor_feda.PASSWD NOT NULL,
    phone VARCHAR(15) NOT NULL,
    role Tailor_feda.employeerole NOT NULL,
    PRIMARY KEY (email)
);

-- Article
CREATE TABLE Tailor_feda.article (
    id UUID DEFAULT uuid_generate_v4(),
    topic TEXT NOT NULL,
    article_text TEXT NOT NULL,
    employee Tailor_feda.emailaddress NOT NULL,
    PRIMARY KEY (id),
    FOREIGN KEY (employee) REFERENCES Tailor_feda.employee(email)
);

-- Manage
CREATE TABLE Tailor_feda.manage (
    product UUID,
    employee Tailor_feda.emailaddress,
    PRIMARY KEY (product, employee),
    FOREIGN KEY (product) REFERENCES Tailor_feda.product(p_code),
    FOREIGN KEY (employee) REFERENCES Tailor_feda.employee(email)
);

-- Work
CREATE TABLE Tailor_feda.work (
    employee Tailor_feda.emailaddress,
    shop UUID,
    PRIMARY KEY (employee, shop),
    FOREIGN KEY (employee) REFERENCES Tailor_feda.employee(email),
    FOREIGN KEY (shop) REFERENCES Tailor_feda.tailor_shop(id)
);
```

```
-- Supplier
CREATE TABLE Tailor_feda.supplier (
    VAT_number Tailor_feda.VATNUMBER ,
    name VARCHAR(64) NOT NULL,
    email Tailor_feda.EMAILADDRESS NOT NULL,
    PRIMARY KEY(VAT_number)
);

-- Order supplier
CREATE TABLE Tailor_feda.order_supplier (
    id UUID DEFAULT uuid_generate_v4(),
    total_price NUMERIC(8,2) NOT NULL CHECK (total_price>=0),
    employee Tailor_feda.EMAILADDRESS NOT NULL,
    supplier Tailor_feda.VATNUMBER NOT NULL,
    PRIMARY KEY(id),
    FOREIGN KEY(employee) REFERENCES Tailor_feda.employee(email),
    FOREIGN KEY(supplier) REFERENCES Tailor_feda.supplier(VAT_number)
);

-- Review
CREATE TABLE Tailor_feda.review (
    id UUID DEFAULT uuid_generate_v4(),
    description TEXT NOT NULL,
    reply TEXT,
    customer Tailor_feda.EMAILADDRESS NOT NULL,
    product UUID UNIQUE NOT NULL,
    PRIMARY KEY(id),
    FOREIGN KEY(customer) REFERENCES Tailor_feda.customer(email),
    FOREIGN KEY(product) REFERENCES Tailor_feda.product(p_code)
);
```

Populate the Database: Example

In the following, the instructions to populate only one row of the tables are reported. Please, find the other ones in the attached file `HW3-Feda_population.sql`.

```
-- Connect to the database
\c feda_db

-- Populate the tables
-- Tailor's shop
INSERT INTO Tailor_feda.tailor_shop (id, phone, email, address) VALUES
    ('6c4dbb99-ab18-4891-a190-3bf125324177', '3485678792', 'roma@feda.it',
'Piazza Navona 5, 12345 Roma (Italia)');

-- Status
INSERT INTO Tailor_feda.status (name) VALUES
    ('Pending');

-- Product_t
INSERT INTO Tailor_feda.product_t (model, work_time, price, fabric, origin,
description) VALUES
    ('T-shirt summer 2020', 10, 14.99, 'Cotton', 'Tailor', 'A simple and cheap
T-shirt');
```

```sql
-- Product
INSERT INTO Tailor_feda.product (p_code, pictures, size, color, work_type,
model) VALUES
    ('15796786-df9c-41e7-b78f-44c78a7a4e7a', NULL, 'M', '#FF671A', 'Online
shop', 'T-shirt summer 2020');
```

Please, insert the correct path for the image tshirt.jpg!

```sql
-- Insert one image for a specific product which already exists
UPDATE Tailor_feda.product
SET pictures[0] = pg_read_binary_file('tshirt.jpg')::bytea
WHERE p_code = '15796786-df9c-41e7-b78f-44c78a7a4e7a';
```

Passwords are crypted before being saved.

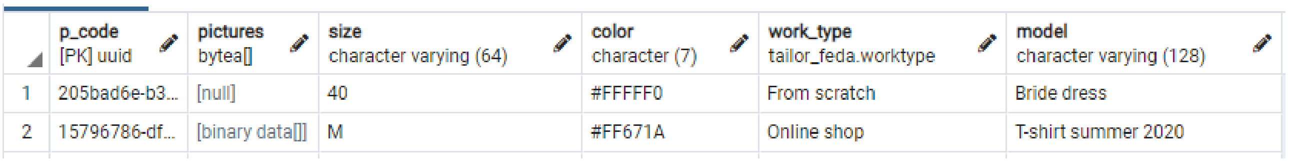
```sql
-- Customer
INSERT INTO Tailor_feda.customer (email, name, surname, password, phone,
addresses, newsletter, get_to_know, sizes, lifestyle, shop) VALUES
    ('paolorossi@gmail.com', 'Paolo', 'Rossi',
crypt(('pwd56789')::Tailor_feda.PASSWD, gen_salt('bf', 4)), '3368495859', '{"Via
Roma 1, 12345 Roma (Italia)", "Viale Garibaldi 25, 54321 Milano (Italia)"}',
FALSE, NULL, NULL, NULL, '6c4dbb99-ab18-4891-a190-3bf125324177');

-- Order_customer
INSERT INTO Tailor_feda.order_customer (id, tot_price, address, p_method,
delivery_mode, is_cancelled, invoice, customer, status) VALUES
    ('f2571408-08d5-440e-bb16-283fe1c4c09b', 3014.99, 'Via Roma 1, 12345 Roma
(Italia)', 'Bank transfer', 'Home', false, 'ABC123-ZZZ1',
'paolorossi@gmail.com', 'Accepted');

-- Contain
INSERT INTO Tailor_feda.contain (order_customer, product) VALUES
    ('f2571408-08d5-440e-bb16-283fe1c4c09b', '205bad6e-b3ea-44c0-8719-
abb8a5e80b4e');

-- Employee
INSERT INTO Tailor_feda.employee (email, name, surname, password, phone, role)
VALUES
    ('mariorossi@feda.it', 'Mario', 'Rossi',
crypt(('pwd12345')::Tailor_feda.PASSWD, gen_salt('bf', 4)), '3473837363',
'Manager');

-- Article
INSERT INTO Tailor_feda.article (topic, article_text, employee) VALUES
    ('Fashion Week', 'The former fashion writer and contributor to major online
luxury fashion destinations ever heard of Net-a-Porter and Matchesfashion has
undoubtedly become the face of cool, French girl style.', 'mariorossi@feda.it');

-- Manage
INSERT INTO Tailor_feda.manage (product, employee) VALUES
    ('15796786-df9c-41e7-b78f-44c78a7a4e7a', 'mariorossi@feda.it');

-- Work
INSERT INTO Tailor_feda.work (employee, shop) VALUES
    ('mariorossi@feda.it', '6c4dbb99-ab18-4891-a190-3bf125324177');

-- Supplier
INSERT INTO Tailor_feda.supplier (VAT_number,name, email) VALUES
    ('0344352075A','Stoffe&Tessuti', 'sandt@gmail.com');
```

```sql
-- Order_supplier
INSERT INTO Tailor_feda.order_supplier (total_price, employee, supplier) VALUES
    (138.50, 'mariorossi@feda.it', '0344352075A');

-- Review
INSERT INTO Tailor_feda.review (description, reply, customer, product) VALUES
    ('Fits very well and very affordable. Great job by tailor shop','Thank you
very much!', 'paolorossi@gmail.com','15796786-df9c-41e7-b78f-44c78a7a4e7a');

-- Request_quote
INSERT INTO Tailor_feda.request_quote (model, customer, quote) VALUES
    ('Shorts', 'paolorossi@gmail.com', 4.99);
```

Principal Queries

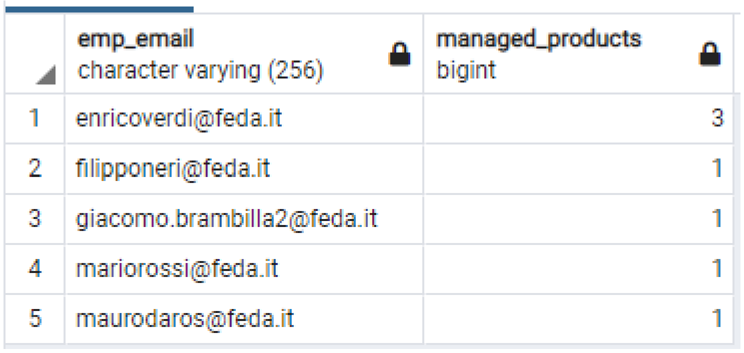
In the following, some queries to extract meaningful information from the database are reported. The same queries are reported also in the file `HW3-Feda_queries.sql`.

```sql
-- Select the features of the products related to a specific order by the
customer Paolo Rossi
SELECT P_CODE, PICTURES, SIZE, COLOR, WORK_TYPE, MODEL FROM
TAILOR_FEDA.PRODUCT AS P
  INNER JOIN TAILOR_FEDA.CONTAIN AS C
ON C.PRODUCT=P.P_CODE
  INNER JOIN TAILOR_FEDA.ORDER_CUSTOMER AS O
ON O.ID=C.ORDER_CUSTOMER
WHERE O.CUSTOMER='paolorossi@gmail.com' AND O.ID='f2571408-08d5-440e-bb16-
283fe1c4c09b';
```



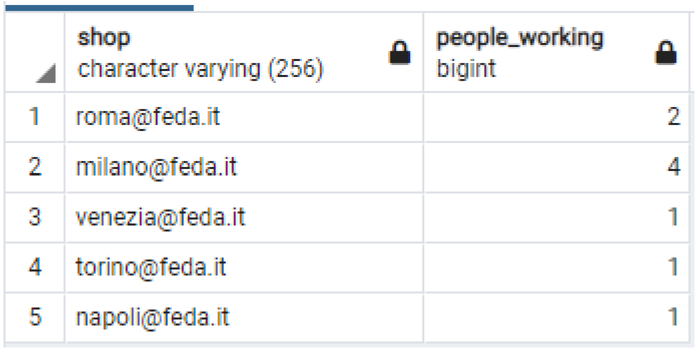
	p_code [PK] uuid	pictures bytea[]	size character varying (64)	color character (7)	work_type tailor_feda.worktype	model character varying (128)
1	205bad6e-b3...	[null]	40	#FFFFF0	From scratch	Bride dress
2	15796786-df...	[binary data[]]	M	#FF671A	Online shop	T-shirt summer 2020

```sql
-- Count how many products are managed by each employee, with employees in
alphabetic order
SELECT email AS emp_email, COUNT(email) AS managed_products
FROM Tailor_feda.product AS PROD
  INNER JOIN Tailor_feda.manage AS MAN
ON PROD.p_code = MAN.product
  INNER JOIN Tailor_feda.employee AS EMP
ON MAN.employee = EMP.email
GROUP BY email
ORDER BY email;
```



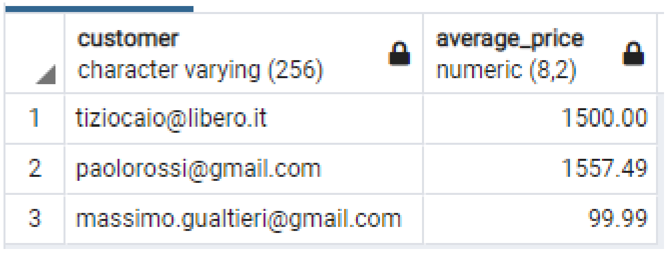
	emp_email character varying (256)	managed_products bigint
1	enricoverdi@feda.it	3
2	filipponeri@feda.it	1
3	giacomo.brambilla2@feda.it	1
4	mariorossi@feda.it	1
5	maurodaros@feda.it	1

```
-- Count the employees working on at least one product in each shop
SELECT email AS shop, people_working FROM Tailor_feda.tailor_shop AS sh
INNER JOIN (SELECT shop, COUNT(*) AS people_working
FROM Tailor_feda.manage AS m
  INNER JOIN Tailor_feda.work AS w
ON m.employee = w.employee
GROUP BY shop) AS counting ON sh.id = counting.shop;
```



	shop character varying (256)	people_working bigint
1	roma@feda.it	2
2	milano@feda.it	4
3	venezia@feda.it	1
4	torino@feda.it	1
5	napoli@feda.it	1

```
-- Compute the average cost of an order by customers who wrote at least one
review
SELECT O.customer, CAST(AVG(Tot_price) AS NUMERIC(8,2)) AS average_price
FROM Tailor_feda.order_customer AS O
  INNER JOIN Tailor_feda.customer AS C
ON C.email= O.customer
  INNER JOIN Tailor_feda.review AS R
ON C.Email = R.customer
GROUP BY O.customer;
```



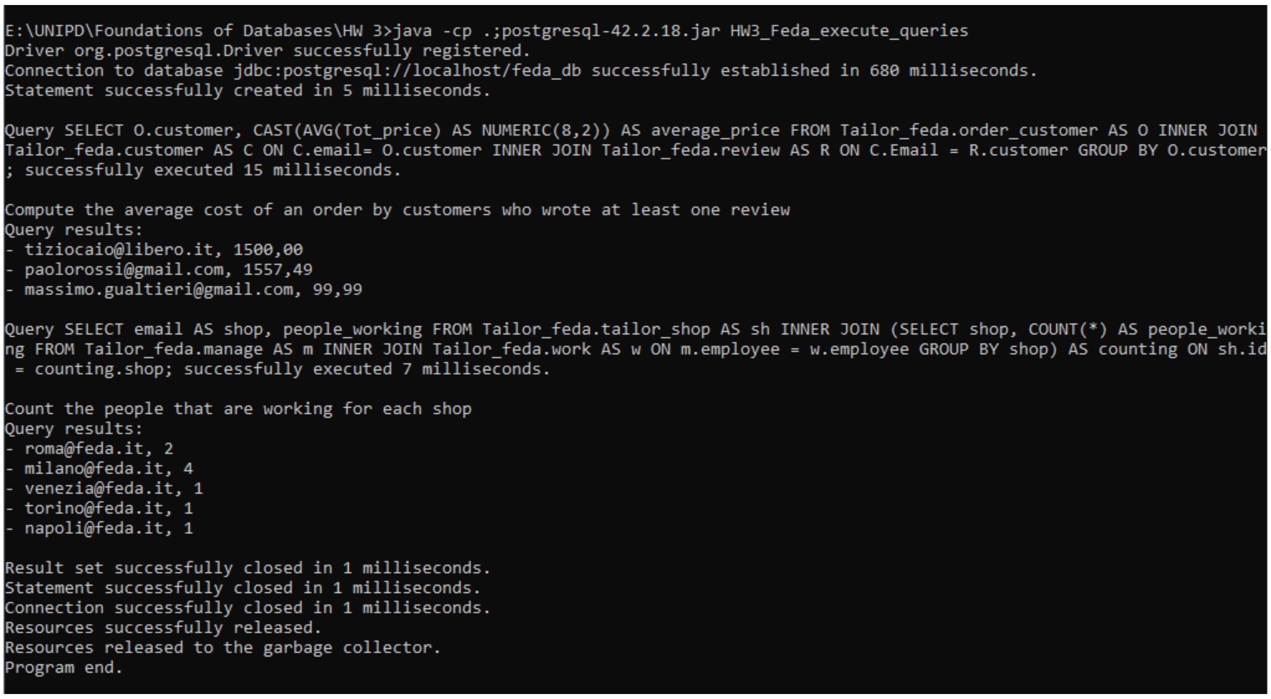
	customer character varying (256)	average_price numeric (8,2)
1	tiziocaio@libero.it	1500.00
2	paolorossi@gmail.com	1557.49
3	massimo.gualtieri@gmail.com	99.99

JDBC Implementations of the Principal Queries and Visualization

In the following, the Java code to perform two of the queries shown above is reported. Eventually, the output of the terminal is displayed. This code is also available in the attached file `HW3_Feda_execute_queries.java`.

```java
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.math.BigDecimal;

/**
 * Lists the products in the catalog
 */
public class HW3_Feda_execute_queries {

    /**
     * The JDBC driver to be used
     */
    private static final String DRIVER = "org.postgresql.Driver";

    /**
     * The URL of the database to be accessed
     */
    private static final String DATABASE = "jdbc:postgresql://localhost/feda_db";

    /**
     * The username for accessing the database
     */
    private static final String USER = "postgres";

    /**
     * The password for accessing the database
     */
    private static final String PASSWORD = "pwd";

    /**
     * The SQL statements to be executed
     */
    private static final String SQL1 = "SELECT O.customer, CAST(AVG(Tot_price) AS NUMERIC(8,2)) AS average_price " +
            "FROM Tailor_feda.order_customer AS O " + "INNER JOIN Tailor_feda.customer AS C " + "ON C.email= O.customer " +
            "INNER JOIN Tailor_feda.review AS R " + "ON C.Email = R.customer " + "GROUP BY O.customer;";

    private static final String SQL2 = "SELECT email AS shop, people_working FROM Tailor_feda.tailor_shop AS sh INNER JOIN (SELECT shop, COUNT(*) AS people_working " +
            "FROM Tailor_feda.manage AS m " + "INNER JOIN Tailor_feda.work AS w " + "ON m.employee = w.employee " + "GROUP BY shop) AS counting ON sh.id = counting.shop;";

    public static void main(String[] args) {

        // the connection to the DBMS
        Connection con = null;

        // the statement to be executed
        Statement stmt = null;

        // the results of the statement execution
        ResultSet rs = null;

        // start time of a statement
        long start;

        // end time of a statement
        long end;

        // "data structures" for the data to be read from the database

        // First query
        double average_price = 0;
        String customer = null;

        // Second query
        String shop = null;
        int count = 0;

        try {
            // register the JDBC driver
            Class.forName(DRIVER);

            System.out.printf("Driver %s successfully registered.%n", DRIVER);
        } catch (ClassNotFoundException e) {
            System.out.printf(
                    "Driver %s not found: %s.%n", DRIVER, e.getMessage());

            // terminate with a generic error code
            System.exit(-1);
        }

        try {

            // connect to the database
            start = System.currentTimeMillis();

            con = DriverManager.getConnection(DATABASE, USER, PASSWORD);

            end = System.currentTimeMillis();

            System.out.printf(
                    "Connection to database %s successfully established in %,d milliseconds.%n",
                    DATABASE, end-start);

            // create the statement to execute the query
            start = System.currentTimeMillis();

            stmt = con.createStatement();

                   end = System.currentTimeMillis();

                   System.out.printf(
                           "Statement successfully created in %,d milliseconds.%n",
                           end-start);

                   // execute the first query
                   start = System.currentTimeMillis();

                   rs = stmt.executeQuery(SQL1);

                   end = System.currentTimeMillis();

                   System.out
                           .printf("%nQuery %s successfully executed %,d milliseconds.%n",
                                   SQL1, end - start);

                   System.out
                           .printf("%nCompute the average cost of an order by customers who wrote at least one review%n");

                   System.out
                           .printf("Query results:%n");

                   // cycle on the query results and print them
                   while (rs.next()) {

                       // read the customer
                       customer = rs.getString("customer");

                       // read the average price
                       average_price = rs.getBigDecimal("average_price").doubleValue();

                       System.out.printf("- %s, %.2f%n",
                               customer, average_price);
```

```java

                }

                // execute the second query
                start = System.currentTimeMillis();

                rs = stmt.executeQuery(SQL2);

                end = System.currentTimeMillis();

                System.out
                        .printf("%nQuery %s successfully executed %,d milliseconds.%n",
                                SQL2, end - start);

                System.out
                        .printf("%nCount the people that are working for each shop%n");

                System.out
                        .printf("Query results:%n");

                // cycle on the query results and print them
                while (rs.next()) {

                    // read the customer's name
                    shop = rs.getString("shop");

                    // read the customer's surname
                    count = rs.getInt("people_working");

                    System.out.printf("- %s, %d%n",
                            shop, count);
                }
            } catch (SQLException e) {
                System.out.printf("Database access error:%n");

                // cycle in the exception chain
                while (e != null) {
                    System.out.printf("- Message: %s%n", e.getMessage());
                    System.out.printf("- SQL status code: %s%n", e.getSQLState());
                    System.out.printf("- SQL error code: %s%n", e.getErrorCode());
                    System.out.printf("%n");
                    e = e.getNextException();
                }
            } finally {
                try {

                    // close the used resources
                    if (rs != null) {

                        start = System.currentTimeMillis();

                        rs.close();

                        end = System.currentTimeMillis();

                        System.out
                        .printf("%nResult set successfully closed in %,d milliseconds.%n",
                                end-start);
                    }

                    if (stmt != null) {

                        start = System.currentTimeMillis();

                        stmt.close();

                        end = System.currentTimeMillis();

                        System.out
                        .printf("Statement successfully closed in %,d milliseconds.%n",
                                end-start);
                    }

                    if (con != null) {

                        start = System.currentTimeMillis();

                        con.close();

                        end = System.currentTimeMillis();

                        System.out
                        .printf("Connection successfully closed in %,d milliseconds.%n",
                                end-start);
                    }

                    System.out.printf("Resources successfully released.%n");

                } catch (SQLException e) {
                    System.out.printf("Error while releasing resources:%n");

                    // cycle in the exception chain
                    while (e != null) {
                        System.out.printf("- Message: %s%n", e.getMessage());
                        System.out.printf("- SQL status code: %s%n", e.getSQLState());
                        System.out.printf("- SQL error code: %s%n", e.getErrorCode());
                        System.out.printf("%n");
                        e = e.getNextException();
                    }

                } finally {

                    // release resources to the garbage collector
                    rs = null;
                    stmt = null;
                    con = null;

                    System.out.printf("Resources released to the garbage collector.%n");
                }
            }

            System.out.printf("Program end.%n");

        }
    }
```


```
E:\UNIPD\Foundations of Databases\HW 3>java -cp .;postgresql-42.2.18.jar HW3_Feda_execute_queries
Driver org.postgresql.Driver successfully registered.
Connection to database jdbc:postgresql://localhost/feda_db successfully established in 680 milliseconds.
Statement successfully created in 5 milliseconds.

Query SELECT O.customer, CAST(AVG(Tot_price) AS NUMERIC(8,2)) AS average_price FROM Tailor_feda.order_customer AS O INNER JOIN
Tailor_feda.customer AS C ON C.email= O.customer INNER JOIN Tailor_feda.review AS R ON C.Email = R.customer GROUP BY O.customer
; successfully executed 15 milliseconds.

Compute the average cost of an order by customers who wrote at least one review
Query results:
- tiziocaio@libero.it, 1500,00
- paolorossi@gmail.com, 1557,49
- massimo.gualtieri@gmail.com, 99,99

Query SELECT email AS shop, people_working FROM Tailor_feda.tailor_shop AS sh INNER JOIN (SELECT shop, COUNT(*) AS people_worki
ng FROM Tailor_feda.manage AS m INNER JOIN Tailor_feda.work AS w ON m.employee = w.employee GROUP BY shop) AS counting ON sh.id
 = counting.shop; successfully executed 7 milliseconds.

Count the people that are working for each shop
Query results:
- roma@feda.it, 2
- milano@feda.it, 4
- venezia@feda.it, 1
- torino@feda.it, 1
- napoli@feda.it, 1

Result set successfully closed in 1 milliseconds.
Statement successfully closed in 1 milliseconds.
Connection successfully closed in 1 milliseconds.
Resources successfully released.
Resources released to the garbage collector.
Program end.
```