UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Database Management Systems, A.Y. 2018/2019
Master Degree in Computer Engineering
Master Degree in Telecommunication Engineering

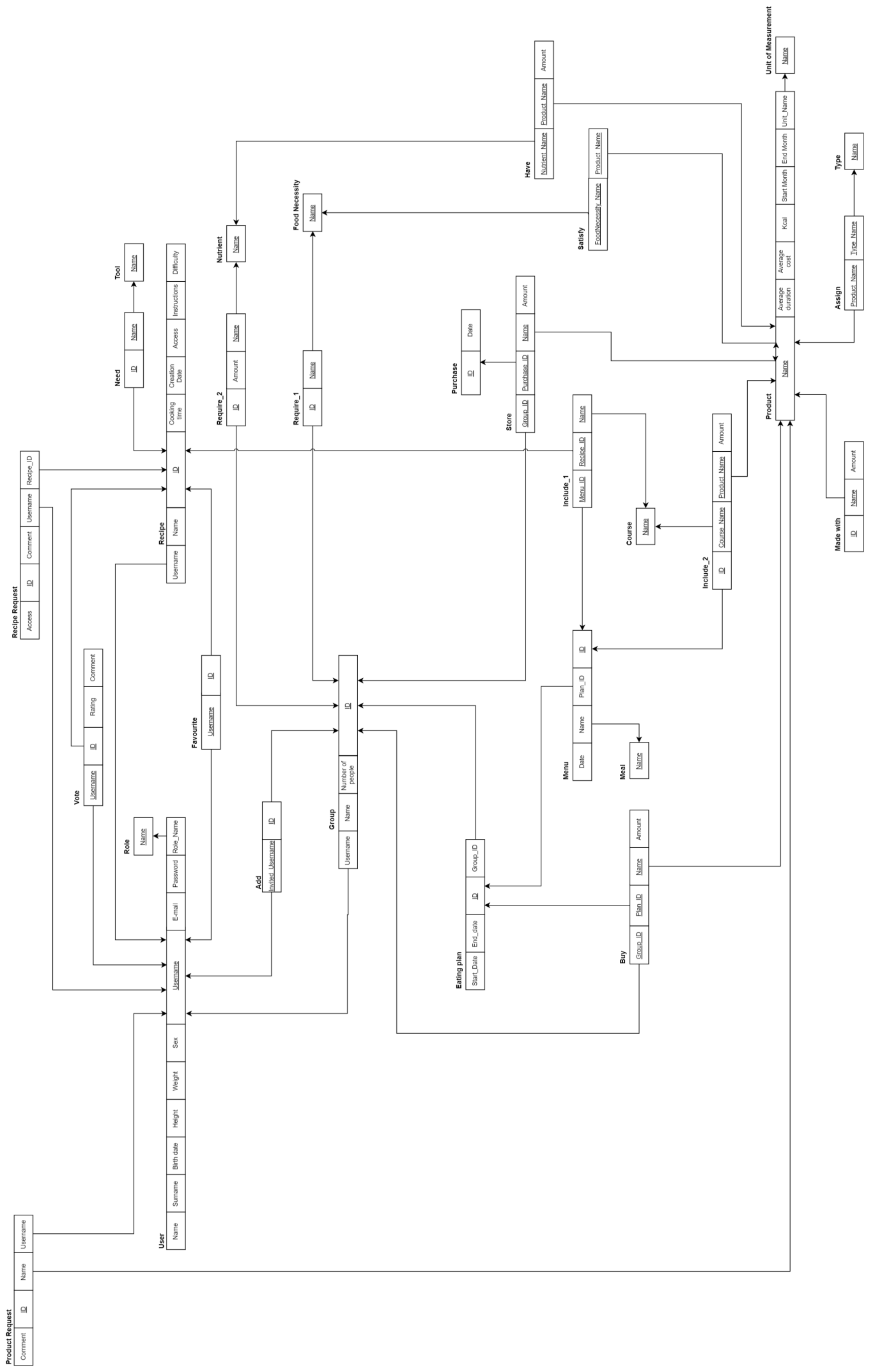
Homework 4 – Physical Design

Deadline: May 22, 2019

Group Four++	Project AutoChef	
Last Name	**First Name**	**Student Number**
Bordignon	Benvenuto	1210175
Di Nardo Di Maio	Raffaele	1204879
Fabris	Cristina	1205722
Perin	Matteo	1205718
Pistilli	Davide Dravindran	1204880

Variations to the Relational Schema

Figure 1 shows the variations made on the relational schema. The attribute "amount" was added to the relation 'Buy', because the user needs to know how many products they need to buy. We also added the attribute "amount" in relation 'Include2' to set how much of a product that the user wants to eat. Moreover, we removed "Creator_Username" inside relation 'Add', because the attribute was redundant and could be derived by the table 'Group'.



Product Request
Comment
ID
Name
Username
User
Name
Surname
Birth date
Height
Weight
Sex
Username
E-mail
Password
Role_Name
Role
Name
Recipe Request
Access
ID
Comment
Username
Recipe_ID
Vote
Username
ID
Rating
Comment
Recipe
Username
Name
ID
Cooking time
Creation Date
Access
Instructions
Difficulty
Favourite
Username
ID
Need
ID
Name
Tool
Name
Add
Invited_Username
ID
Group
Username
Name
Number of people
ID
Require_2
ID
Amount
Name
Nutrient
Name
Require_1
ID
Name
Food Necessity
Name
Eating plan
Start_Date
End_date
ID
Group_ID
Menu
Date
Name
Plan_ID
ID
Meal
Name
Buy
Group_ID
Plan_ID
Name
Amount
Include_1
Menu_ID
Recipe_ID
Name
Course
Name
Include_2
ID
Course_Name
Product_Name
Amount
Purchase
ID
Date
Store
Group_ID
Purchase_ID
Name
Amount
Have
Nutrient_Name
Product_Name
Amount
Satisfy
FoodNecessity_Name
Product_Name
Product
Name
Average duration
Average cost
Kcal
Start Month
End Month
Unit_Name
Unit of Measurement
Name
Assign
Product_Name
Type_Name
Type
Name
Made with
ID
Name
Amount

Figure 1

Physical Schema

In order to build the database according to the schema presented in **Figure 1** we need the following SQL instructions:

```
-- Database Creation
CREATE DATABASE autochef OWNER POSTGRES ENCODING = 'UTF8';

-- Connect to autochef db to create data for its 'public' schema
\c autochef

-- Create new domains
-- Correct password format
CREATE DOMAIN pwd AS character varying(254)
       CONSTRAINT properpassword CHECK (((VALUE)::text ~* '[A-Za-z0-9._%-
]{5,}'::text));

-- Correct mail format
CREATE DOMAIN mail AS character varying(254)
  CONSTRAINT propermail CHECK (((VALUE)::text ~* '[A-Za-z0-9._%-]+@[A-Za-z0-
9._%]+$'::text));

--Create new data type
CREATE TYPE gendertype AS ENUM (
    'Male',
    'Female'
);


-- Create tables

CREATE TABLE Role(
    Name VARCHAR,

    PRIMARY KEY (Name)
);

CREATE TABLE Tool(
    Name VARCHAR,
    PRIMARY KEY (Name)
);

CREATE TABLE Nutrient(
    Name VARCHAR,

    PRIMARY KEY (Name)
);

CREATE TABLE FoodNecessity(
    Name VARCHAR,

    PRIMARY KEY (Name)
);

CREATE TABLE FoodType(
    Name VARCHAR,

    PRIMARY KEY (Name)
);

CREATE TABLE Meal(
    Name VARCHAR,

    PRIMARY KEY (Name)
);
```

```sql
CREATE TABLE UnitOfMeasurement(
    Name VARCHAR,

    PRIMARY KEY (Name)
);

CREATE TABLE Course(
    Name VARCHAR,

    PRIMARY KEY (Name)
);

--User
CREATE TABLE AutoChefUser (
    Email MAIL,
    Birthdate DATE NOT NULL,
    Height INTEGER NOT NULL,
    Weight REAL NOT NULL,
    Sex GENDERTYPE NOT NULL,
    Name VARCHAR NOT NULL,
    Surname VARCHAR NOT NULL,
    Username VARCHAR,
    Password PWD,
    RoleName VARCHAR NOT NULL,
    CONSTRAINT Physical CHECK ((Height > 0) AND (Weight > 0)),

    PRIMARY KEY (Username),
    FOREIGN KEY(RoleName) REFERENCES Role(Name)
);

CREATE TABLE Recipe (
    Id UUID,
    Name VARCHAR NOT NULL,
    Access BOOLEAN NOT NULL, --TRUE if public FALSE if private
    Username VARCHAR NOT NULL,
    CookingTime INTEGER NOT NULL,
    CreationDate DATE NOT NULL,
    Difficulty INTEGER NOT NULL,
    Instructions TEXT NOT NULL,
    CONSTRAINT Details CHECK ((CookingTime > 0) AND (Difficulty > 0 AND Difficulty <
6)),

    PRIMARY KEY (Id),
    FOREIGN KEY (Username) REFERENCES AutoChefUser(Username)
);

CREATE TABLE RecipeRequest (
    Id UUID,
    Comment TEXT NOT NULL,
    Access BOOLEAN NOT NULL, --TRUE if public FALSE if private
    Username VARCHAR NOT NULL,
    RecipeId UUID NOT NULL,

    PRIMARY KEY (Id),
    FOREIGN KEY (Username) REFERENCES AutoChefUser(Username),
    FOREIGN KEY (RecipeId) REFERENCES Recipe(Id)
);


CREATE TABLE Need (
    Id UUID,
    Name VARCHAR,

    PRIMARY KEY (Id,Name),
    FOREIGN KEY (Id) REFERENCES Recipe(Id),
    FOREIGN KEY (Name) REFERENCES Tool(Name)
```

```sql
);

CREATE TABLE Favourite (
    Username VARCHAR,
    Id UUID,

    PRIMARY KEY (Id,Username),
    FOREIGN KEY (Id) REFERENCES Recipe(Id),
    FOREIGN KEY (Username) REFERENCES AutoChefUser(Username)
);

CREATE TABLE Purchase (
    Id UUID,
    PurchaseDate DATE NOT NULL,

    PRIMARY KEY (Id)
);

--Group
CREATE TABLE UserGroup (
    Id UUID,
    Username VARCHAR NOT NULL,
    Name VARCHAR NOT NULL,
    NumberOfPeople INTEGER NOT NULL DEFAULT 1 CHECK (NumberOfPeople > 0),

    PRIMARY KEY (Id),
    FOREIGN KEY (Username) REFERENCES AutoChefUser(Username)
);

CREATE TABLE EatingPlan (
    Id UUID,
    GroupId UUID NOT NULL,
    StartDate DATE NOT NULL,
    EndDate DATE NOT NULL CHECK (EndDate - StartDate > 0),

    PRIMARY KEY (Id),
    FOREIGN KEY (GroupId) REFERENCES UserGroup(Id)
);

CREATE TABLE Product (
    Name VARCHAR,
    AverageDuration INTEGER NOT NULL CHECK (AverageDuration > 0),
    AverageCost NUMERIC(1000,2) NOT NULL CHECK (AverageCost > 0),
    Kcal REAL NOT NULL CHECK (Kcal > 0),
    StartMonth INTEGER NOT NULL CHECK (StartMonth >= 0 AND StartMonth <= 12),
    EndMonth INTEGER NOT NULL CHECK (EndMonth >= 0 AND EndMonth <= 12),
    UnitName VARCHAR NOT NULL,

    PRIMARY KEY (Name),
    FOREIGN KEY (UnitName) REFERENCES UnitOfMeasurement(Name)
);

CREATE TABLE Buy (
    GroupId UUID,
    PlanId UUID,
    Name VARCHAR,
    Amount INTEGER NOT NULL CHECK (Amount > 0),
    PRIMARY KEY(GroupId, PlanId, Name),
    FOREIGN KEY(GroupId) REFERENCES UserGroup(Id),
    FOREIGN KEY(PlanID) REFERENCES EatingPlan(Id),
    FOREIGN KEY(Name) REFERENCES Product(Name)
);

CREATE TABLE MadeWith (
    Id UUID,
    Name VARCHAR,
    Amount INTEGER NOT NULL,
    CONSTRAINT AmountConstraint CHECK (Amount > 0),

    PRIMARY KEY (Id, Name),
    FOREIGN KEY (Id) REFERENCES Recipe(Id),
    FOREIGN KEY (Name) REFERENCES Product(Name)
);

CREATE TABLE ProductRequest (
    Id UUID,
    Comment TEXT NOT NULL,
    Name VARCHAR NOT NULL,
    Username VARCHAR NOT NULL,

    PRIMARY KEY (Id),
    FOREIGN KEY (Username) REFERENCES AutoChefUser(Username)
);

CREATE TABLE Have (
    NutrientName VARCHAR,
    ProductName VARCHAR,
    Amount INTEGER NOT NULL CHECK (Amount > 0),

    PRIMARY KEY (NutrientName, ProductName),
    FOREIGN KEY (NutrientName) REFERENCES Nutrient(Name),
    FOREIGN KEY (ProductName) REFERENCES Product(Name)
);

CREATE TABLE Assign (
    TypeName VARCHAR,
    ProductName VARCHAR,

    PRIMARY KEY (TypeName, ProductName),
    FOREIGN KEY (TypeName) REFERENCES FoodType(Name),
    FOREIGN KEY (ProductName) REFERENCES Product(Name)
);

CREATE TABLE Store (
    PurchaseId UUID,
    Name VARCHAR,
    GroupId UUID,
    Amount INTEGER NOT NULL CHECK (Amount > 0),

    PRIMARY KEY (PurchaseId, Name, GroupId),
    FOREIGN KEY (PurchaseId) REFERENCES Purchase(Id),
    FOREIGN KEY (Name) REFERENCES Product(Name),
    FOREIGN KEY (GroupId) REFERENCES UserGroup(Id)
);

CREATE TABLE Satisfy (
    FoodNecessityName VARCHAR,
    ProductName VARCHAR,

    PRIMARY KEY (FoodNecessityName, ProductName),
    FOREIGN KEY (FoodNecessityName) REFERENCES FoodNecessity(Name),
    FOREIGN KEY (ProductName) REFERENCES Product(Name)
);

CREATE TABLE Vote (
    Rating INTEGER NOT NULL,
    Username VARCHAR NOT NULL,
    Id UUID,
    Comment TEXT NOT NULL,
    CONSTRAINT Vote CHECK (Rating>0 AND Rating<6),

    PRIMARY KEY (Username, Id),
    FOREIGN KEY (Username) REFERENCES AutoChefUser(Username),
    FOREIGN KEY (Id) REFERENCES Recipe(Id)
);

CREATE TABLE Add (
    InvitedUsername VARCHAR,
    Id UUID,

    PRIMARY KEY (InvitedUsername, Id),
    FOREIGN KEY (InvitedUsername) REFERENCES AutoChefUser(Username),
    FOREIGN KEY (Id) REFERENCES UserGroup(Id)
);

CREATE TABLE Menu (
    MenuDate DATE NOT NULL,
    Name VARCHAR NOT NULL,
    PlanId UUID NOT NULL,
    Id UUID,

    PRIMARY KEY (Id),
    FOREIGN KEY (PlanId) REFERENCES EatingPlan(Id),
    FOREIGN KEY (Name) REFERENCES Meal(Name)
);

CREATE TABLE Include1 (
    MenuId UUID,
    RecipeId UUID,
    Name VARCHAR,

    PRIMARY KEY (MenuId, RecipeId, Name),
    FOREIGN KEY (MenuId) REFERENCES Menu(Id),
    FOREIGN KEY (RecipeId) REFERENCES Recipe(Id),
    FOREIGN KEY (Name) REFERENCES Course(Name)
);

CREATE TABLE Include2(
    Id UUID,
    CourseName VARCHAR,
    ProductName VARCHAR,
    Amount INTEGER NOT NULL CHECK (Amount > 0),

    PRIMARY KEY (Id, CourseName, ProductName),
    FOREIGN KEY (Id) REFERENCES Menu(Id),
    FOREIGN KEY (ProductName) REFERENCES Product(Name),
    FOREIGN KEY (CourseName) REFERENCES Course(Name)
);

CREATE TABLE Require1(
    Id UUID,
    Name VARCHAR,

    PRIMARY KEY (Id, Name),
    FOREIGN KEY (Id) REFERENCES UserGroup(Id),
    FOREIGN KEY (Name) REFERENCES FoodNecessity(Name)
);

CREATE TABLE Require2(
    Id UUID,
    Name VARCHAR,
    Amount INTEGER NOT NULL,

    PRIMARY KEY (Id, Name),
    FOREIGN KEY (Id) REFERENCES UserGroup(Id),
    CONSTRAINT Quantity CHECK (Amount>0),
    FOREIGN KEY (Name) REFERENCES Nutrient(Name)
);
```

Trigger Function

When a user tries to insert a new product in a shopping list (represented by the table ‘Buy’), this trigger calls a procedure to check if the product is already present in the table and, in this case, it deletes the corresponding row. After this check the specified row is inserted into the table.

```
-- Connect to autochef database
\c autochef

--Update the Buy table.
CREATE OR REPLACE FUNCTION update_shopping_list() RETURNS TRIGGER AS $$
BEGIN
        --Check whether the product needed is already present in the shopping list.
        PERFORM GroupId, PlanId, Name
        FROM Buy
        WHERE (Buy.GroupId = NEW.GroupId AND Buy.PlanId = NEW.PlanId AND Buy.Name =
NEW.Name);
        --If it is, then delete the previous row.
        IF FOUND THEN
                DELETE FROM Buy
                WHERE (Buy.GroupId = NEW.GroupId AND Buy.PlanId = NEW.PlanId AND
Buy.Name = NEW.Name);
        END IF;

        RETURN NEW;
END
$$ LANGUAGE PLPGSQL;

--Trigger that activates before any insert is executed on Buy.
CREATE TRIGGER shopping_list_trigger BEFORE INSERT
ON Buy
FOR EACH ROW
EXECUTE PROCEDURE update_shopping_list();
```

Populate the Database: Example

In the following, there are some examples of SQL instructions to create a new Group and add users to it:

1. When a user creates a group, a new row in the table UserGroup is added. Its values contain: the assigned GroupID, the creator’s username, the desired name for the group and the number of participants (both specified by the user).

   ```
   INSERT INTO UserGroup VALUES ('3f151df9-187c-4c31-9f65-ce7feac98c7b',
   'jsimonato3', 'Family', 3);
   ```

2. Each time the group creator adds other users to the group a row is added in table Add. In this case 3 users have been invited in the previously created group.

   ```
   INSERT INTO Add VALUES ('rwitherupf', '3f151df9-187c-4c31-9f65-ce7feac98c7b');
   INSERT INTO Add VALUES ('ovanyukovk', '3f151df9-187c-4c31-9f65-ce7feac98c7b');
   INSERT INTO Add VALUES ('cmenichino2', '3f151df9-187c-4c31-9f65-ce7feac98c7b');
   ```

In the following, there are some examples of SQL instructions to insert a new Eating Plan populated with menus for a given group. The process goes as follows:

1. A new row is created in the table Eating Plan and it is linked to the group creating it by storing its GroupID (in this case we consider group '3f151df9-187c-4c31-9f65-ce7feac98c7b'). Each EatingPlan is given its own ID, in this case: '90be8e4e-f923-4ac1-9193-bfe129e0fea0'. The last two values are the user defined start and end dates for this specific plan.

   ```
   INSERT INTO EatingPlan VALUES ('2af5de30-2d93-441b-8d5a-3b6e0dfd51c2', 'dced3581-
   3853-4ed5-9cd3-ecce4bb95969', '12/05/2019', '19/05/2019');
   ```

2. An Eating Plan can then be populated with menus. Each menu is assigned to a certain date and a certain time of the day in which it will be consumed. It is then associated to the eating plan currently being modified and assigned a specific MenuID.

```sql
INSERT INTO Menu VALUES ('13/05/2019', 'Lunch', '2af5de30-2d93-441b-8d5a-3b6e0dfd51c2', '0e9056ae-8dac-4579-bd8a-f3aa492bf6f3');
```

```sql
INSERT INTO Menu VALUES ('14/05/2019', 'Dinner', '2af5de30-2d93-441b-8d5a-3b6e0dfd51c2', '0e9056ae-8dac-4579-bd8a-f3aa422bf6f3');
```

3. Each Menu can then be populated with both recipes and products.

   The table Include1 deals with recipe insertion, in this case we want to add two recipes: Salmon with feta and cucumbers (RecipeID: '64acc5fb-4d58-48ac-85d1-3b9697597951') as a second course and Apple and cinnamon oatmeal cookies (RecipeID: 'cdb95208-2e45-4a31-bff3-9de5a9d412c6') as dessert in the 13/05/2019 dinner (MenuID: '6b606a67-ecb1-40e7-9996-664ecb26f95c').

```sql
INSERT INTO Include1 VALUES ('6b606a67-ecb1-40e7-9996-664ecb26f95c', '64acc5fb-4d58-48ac-85d1-3b9697597951', 'Second Course');
INSERT INTO Include1 VALUES ('6b606a67-ecb1-40e7-9996-664ecb26f95c', 'cdb95208-2e45-4a31-bff3-9de5a9d412c6', 'Dessert');
```

   The table Include2 deals with product insertion, in this instance if we wanted to add Chicken Wings as a second course in the same meal we would write:

```sql
INSERT INTO Include2 VALUES ('6b606a67-ecb1-40e7-9996-664ecb26f95c', 'Second Course', 'Chicken Wings', 100);
```

   In this instance we also had to specify the amount to be consumed, as products are not linked to a specific serving size as recipes do.

Main Queries

In the following there are four queries to navigate the database:

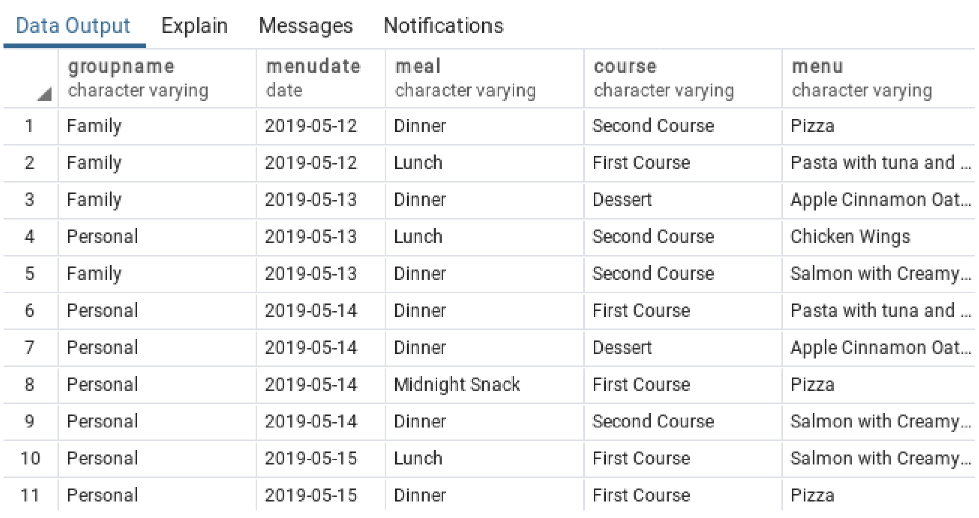
1. Take all groups linked to a certain user and print the resulting eating plan;
2. Print the list of recipes and the number of eating plans in which they are included;
3. Print the list of products and the number of recipes in which they are contained;
4. Print the list of recipes with their author and average rating.

1.

```sql
-- Take all groups linked to a certain user and print the resulting eating plan
WITH linked_groups AS(
        SELECT
                UserGroup.Name,
                UserGroup.Id
        FROM AutoChefUser
              INNER JOIN UserGroup ON UserGroup.Username = AutoChefUser.Username
        WHERE UserGroup.Username = 'jsimonato3'

        UNION

        SELECT
                UserGroup.Name,
                UserGroup.Id
        FROM UserGroup
                INNER JOIN Add ON UserGroup.Id = Add.Id
        WHERE Add.InvitedUsername = 'jsimonato3'
), plan_recipes AS(
        SELECT
                linked_groups.Name as groupname,
                Menu.MenuDate,
                Menu.Name as meal,
                Include1.Name as course,
                Recipe.Name as menu
        FROM linked_groups
                INNER JOIN EatingPlan ON linked_groups.Id = EatingPlan.GroupId
                INNER JOIN Menu ON Menu.PlanId = EatingPlan.Id
                INNER JOIN Include1 ON Include1.MenuId = Menu.Id
                INNER JOIN Recipe ON Include1.RecipeId = Recipe.Id
), plan_products AS(
        SELECT
                linked_groups.Name as groupname,
                Menu.MenuDate,
                Menu.Name as meal,
                Include2.CourseName as course,
                Include2.ProductName as menu
        FROM linked_groups
                INNER JOIN EatingPlan ON linked_groups.Id = EatingPlan.GroupId
                INNER JOIN Menu ON Menu.PlanId = EatingPlan.Id
                INNER JOIN Include2 ON Include2.Id = Menu.Id
)
SELECT * FROM plan_recipes UNION SELECT * FROM plan_products
                                ORDER BY MenuDate ASC;
```

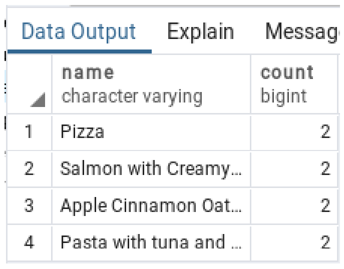


	groupname character varying	menudate date	meal character varying	course character varying	menu character varying
1	Family	2019-05-12	Dinner	Second Course	Pizza
2	Family	2019-05-12	Lunch	First Course	Pasta with tuna and ...
3	Family	2019-05-13	Dinner	Dessert	Apple Cinnamon Oat...
4	Personal	2019-05-13	Lunch	Second Course	Chicken Wings
5	Family	2019-05-13	Dinner	Second Course	Salmon with Creamy...
6	Personal	2019-05-14	Dinner	First Course	Pasta with tuna and ...
7	Personal	2019-05-14	Dinner	Dessert	Apple Cinnamon Oat...
8	Personal	2019-05-14	Midnight Snack	First Course	Pizza
9	Personal	2019-05-14	Dinner	Second Course	Salmon with Creamy...
10	Personal	2019-05-15	Lunch	First Course	Salmon with Creamy...
11	Personal	2019-05-15	Dinner	First Course	Pizza

Figure 2: Results for Query 1

2.

```
--Print the list of recipes and the number of eating plans in which they are included.
SELECT
        PlanRecipe.Name,
        count(PlanRecipe.PlanId)
FROM
(
  SELECT DISTINCT
          Menu.PlanId,
          RecipeMenu.Name
  FROM Menu
    INNER JOIN
    (
      SELECT
              Include1.MenuId,
              Recipe.Name
      FROM Recipe
        INNER JOIN Include1 ON Recipe.id = Include1.RecipeId
    ) AS RecipeMenu
    ON Menu.id = RecipeMenu.MenuId
) AS PlanRecipe
GROUP BY PlanRecipe.Name;
```

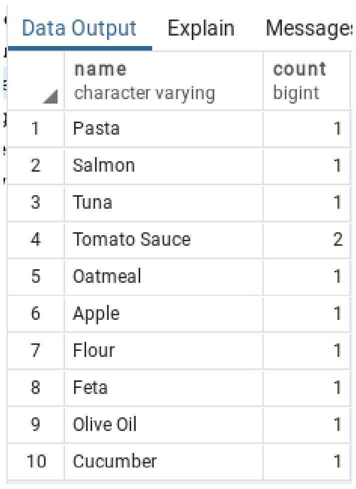

Data Output | Explain | Messag

	name character varying	count bigint
1	Pizza	2
2	Salmon with Creamy...	2
3	Apple Cinnamon Oat...	2
4	Pasta with tuna and ...	2

Figure 3: Results for Query 2

3.

```
--Print the list of products and the number of recipes in which they are contained.
SELECT
        RecipeProduct.Name,
        count(RecipeProduct.Id)
FROM
(
  SELECT
        MadeProduct.Name,
        MadeProduct.Id
  FROM Recipe
    INNER JOIN
    (
      SELECT
                Product.Name,
                MadeWith.Id
      FROM Product
        INNER JOIN MadeWith ON Product.Name = MadeWith.Name
    ) AS MadeProduct
    ON Recipe.Id = MadeProduct.Id
) AS RecipeProduct
GROUP BY RecipeProduct.Name;
```

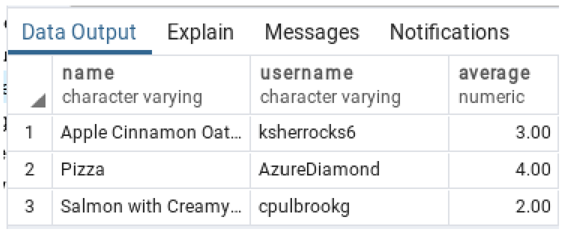

Data Output | Explain | Message

	name character varying	count bigint
1	Pasta	1
2	Salmon	1
3	Tuna	1
4	Tomato Sauce	2
5	Oatmeal	1
6	Apple	1
7	Flour	1
8	Feta	1
9	Olive Oil	1
10	Cucumber	1

Figure 4: Results for Query 3

4.

```
--Print the list of recipes with their author and average rating.
SELECT
        Recipe.Name,
        Recipe.Username,
        RateId.Average
FROM Recipe
  INNER JOIN
  (
    SELECT
              RecipeVote.Id,
              TRUNC(AVG(RecipeVote.Rating),2) AS Average
    FROM
    (
      SELECT
              Recipe.Id,
              Vote.Rating
      FROM Recipe
        INNER JOIN Vote ON Recipe.Id = Vote.Id
    )AS RecipeVote
    GROUP BY RecipeVote.Id
  ) AS RateId
  ON Recipe.Id = RateId.Id;
```



	name character varying	username character varying	average numeric
1	Apple Cinnamon Oat...	ksherrocks6	3.00
2	Pizza	AzureDiamond	4.00
3	Salmon with Creamy...	cpulbrookg	2.00

Figure 5: Results for Query 4

Stored Procedure

In the following are reported three stored procedures that allow to receive some parameters in input from the user. In particular:

1. Compute the shopping list for a given group and an eating plan;
2. Return a shopping list of a given group and eating plan;
3. Return the list of groups for a given user.

1.

```
--Connect to autochef database

\c autochef database

--Compute the shopping list for a given group and eating plan and add it to Buy.
CREATE OR REPLACE FUNCTION initial_shopping_list(param_GroupId UUID, param_PlanId UUID)
RETURNS VOID AS $$
BEGIN
       WITH shopping_list AS (
         --Select all products directly included in the eating plan.
         SELECT
           EatingPlan.GroupId AS UserGroupId,
           EatingPlan.Id AS EatingPlanId,
           Product.Name,
           Include2.Amount
         FROM EatingPlan
           INNER JOIN Menu ON EatingPlan.Id = Menu.PlanId
           INNER JOIN Include2 ON Menu.Id = Include2.Id
           INNER JOIN Product ON Include2.ProductName = Product.Name
         WHERE (EatingPlan.GroupId = param_GroupId
                 AND EatingPlan.Id = param_PlanId)

          UNION ALL --Put together all products required for the eating plan.

          --Select all products needed to prepare recipes included in the current
eating plan.
          SELECT
            EatingPlan.GroupId AS UserGroupId,
            EatingPlan.Id AS EatingPlanId,
            Product.Name,
            MadeWith.Amount
          FROM EatingPlan
            INNER JOIN Menu ON EatingPlan.Id = Menu.PlanId
            INNER JOIN Include1 ON Menu.Id = Include1.MenuId
            INNER JOIN Recipe ON Include1.RecipeId = Recipe.Id
            INNER JOIN MadeWith ON Recipe.Id = MadeWith.Id
            INNER JOIN Product ON MadeWith.Name = Product.Name
          WHERE (EatingPlan.GroupId = param_GroupId
                 AND EatingPlan.Id = param_PlanId)
        ), pantry AS (
           --Select all products already present in the group's pantry.
          SELECT *
          FROM store
          WHERE Store.GroupId = param_GroupId
        ), temp_list AS (
           --Sum the amounts of duplicate products in shopping_list.
          SELECT
            UserGroupId,
            EatingPlanId,
            shopping_list.Name,
            SUM(shopping_list.Amount) AS amount
          FROM shopping_list
          GROUP BY shopping_list.Name, EatingPlanId, UserGroupId
        ), item_list AS (
           --Compute the final shopping list taking into account the group's pantry.
          SELECT
            temp_list.UserGroupId AS GroupId,
            temp_list.EatingPlanId AS EatingPlanId,
            temp_list.Name AS ProductName,
          CASE WHEN pantry.Name IS NOT NULL THEN
              temp_list.Amount - pantry.Amount
          ELSE
              temp_list.Amount
          END AS necessary_amount
          FROM temp_list
          FULL OUTER JOIN pantry ON temp_list.Name = pantry.Name
          WHERE (
            CASE WHEN pantry.Name IS NOT NULL THEN
              temp_list.Amount - pantry.Amount > 0
            ELSE
              temp_list.Amount > 0
            END)
        )
        INSERT INTO Buy SELECT * FROM item_list;
END
$$ LANGUAGE PLPGSQL;
```

2.

```
--Return the most relevant information of the shopping list.
CREATE OR REPLACE FUNCTION print_shopping_list(param_GroupId UUID, param_PlanId UUID)
RETURNS TABLE(Name VARCHAR, Amount INTEGER, UnitName VARCHAR, Cost NUMERIC(1000,2)) AS
$$
BEGIN
```

```
        RETURN QUERY SELECT
          Product.Name,
          Buy.Amount,
          Product.UnitName,
          TRUNC(((Product.AverageCost * Buy.Amount) / 100),2) AS Cost
        FROM Buy
          INNER JOIN Product ON Buy.Name = Product.Name
        WHERE (Buy.GroupId = param_GroupId
                  AND Buy.PlanId = param_PlanId);
END
$$ LANGUAGE PLPGSQL;
```

3.

```
--Return the list of groups for a given user.
CREATE OR REPLACE FUNCTION find_user(param_Username VARCHAR)
RETURNS TABLE(GroupId UUID, GroupName VARCHAR) AS $$
BEGIN
        RETURN QUERY SELECT
                UserGroup.Id,
                UserGroup.Name
        FROM AutoChefUser
                INNER JOIN UserGroup ON UserGroup.Username = param_Username

        UNION

        SELECT
                UserGroup.Id,
                UserGroup.Name
        FROM AutoChefUser
                INNER JOIN Add ON Add.InvitedUsername = param_Username
                INNER JOIN UserGroup ON UserGroup.Id = Add.Id;
END
$$ LANGUAGE PLPGSQL;

CREATE OR REPLACE FUNCTION find_plans(param_GroupId UUID)
RETURNS TABLE(Id UUID, StartDate DATE, EndDate DATE) AS $$
BEGIN
        RETURN QUERY SELECT
                EatingPlan.Id,
                EatingPlan.StartDate,
                EatingPlan.EndDate
        FROM EatingPlan
        WHERE EatingPlan.GroupId=param_GroupId;
END
$$ LANGUAGE PLPGSQL;
```

JDBC Implementations of the Principal Queries and Visualization

In this section is reported a java class which performs some test queries on the database.

```
import java.math.BigDecimal;
import java.sql.Connection;
import java.sql.Date;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.Scanner;

public class AutoChef
{
  /* Database connection information. */
  private static final String DRIVER = "org.postgresql.Driver";
  private static final String DATABASE = "jdbc:postgresql://localhost/autochef";
  private static final String USER = "cjm036653";
  private static final String PASSWORD = "postgres";

  /* Queries. */
  private static final String SQL0 = "SELECT Name, Surname, Birthdate, Sex  FROM 
AutoChefUser;";
  private static final String SQL1 = "SELECT PlanRecipe.Name, count(PlanRecipe.PlanID) 
FROM (  SELECT DISTINCT Menu.PlanID, RecipeMenu.Name  FROM Menu INNER JOIN  (  SELECT 
Include1.MenuID, Recipe.Name FROM Recipe INNER JOIN Include1 ON Recipe.id = 
Include1.RecipeID  ) AS RecipeMenu   ON Menu.ID=RecipeMenu.MenuID ) AS PlanRecipe GROUP 
BY PlanRecipe.Name;";

  /**********MAIN**********/
  public static void main(String[] args)
  {
     /* Variable initialization. */
     Connection con = null; //Connection to the database.
     Statement stmt = null; //Statement object needed to execute SQL queries.
     ResultSet rs = null; //Query results.
     /* Time monitoring variables. */
     long start = 0;
     long end = 0;
     /* Variables that represent a user (Query 0)*/
     String Name = null;
     String Surname  = null;
     Date Birthdate = null;
     String Sex = null;
     /* Variables of query 1. */
     String PlanRecipe = null;
     Integer CountRecipe = null;
     /* Variables of query 2/3. */
     String ProductName = null;
     Integer Amount = null;
     String UnitName = null;
     BigDecimal Cost = null;

     /* Driver registration*/
     Try
     {
         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());
        System.exit(-1);
     }

     /* Database connection and statement creation. */
     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 a statement. */
        start = System.currentTimeMillis();
        stmt = con.createStatement();
        end = System.currentTimeMillis();
        System.out.printf("Statement successfully created in %,d milliseconds.%n", end-
start);
      }
      catch (SQLException e)
      {
        System.out.printf("Database access error:%n");
        printErrorMessages(e);
        releaseResources(con, stmt, rs);
        System.exit(-1);
      }

      /* First query: SQL0. */
      try
      {
        start = System.currentTimeMillis();
        rs = stmt.executeQuery(SQL0);
        end = System.currentTimeMillis();

        System.out.printf("Query %s successfully executed %,d milliseconds.%n", SQL0,
end - start);
        System.out.printf("Query results:%n");

        /* Gather and print the query results. */
        System.out.println("\n  Users of Autochef  ");
        System.out.println("---------------------------------------------------------
");

        while (rs.next())
        {
           Name = rs.getString("Name");
           Surname = rs.getString("Surname");
           Birthdate = rs.getDate("Birthdate");
           Sex = rs.getString("Sex");
           System.out.printf("- %12s, %15s, %13s, %9s%n", Name, Surname,
Birthdate.toString(), Sex);
                          }
           System.out.println("---------------------------------------------------------
-");

        }
        catch (SQLException e)
        {
           System.out.printf("Database access error:%n");
           printErrorMessages(e);
           releaseResources(con, stmt, rs);
           System.exit(-1);
        }

        /* Second query: SQL1. */
        try
        {
           start = System.currentTimeMillis();
           rs = stmt.executeQuery(SQL1);// SQL1
           end = System.currentTimeMillis();
           System.out.printf("Query %s successfully executed %,d milliseconds.%n",
SQL1, end - start);
           System.out.printf("Query results:%n");

           /* Gather and print the query results. */
           System.out.println("\n  Number of plans that contains each recipe  ");
           System.out.println("----------------------------------------------");
           while (rs.next())
           {
                PlanRecipe = rs.getString("Name");
                CountRecipe = rs.getInt("count");
                System.out.printf("- %35s, %6s%n", PlanRecipe, CountRecipe.toString());
```

```
            }
            System.out.println("----------------------------------------------\n");
        }
        catch (SQLException e)
        {
            System.out.printf("Database access error:%n");
            printErrorMessages(e);
            releaseResources(con, stmt, rs);
            System.exit(-1);
        }

        /* Creation and display of a shopping list for a specific user. */
        try
        {
            boolean flag= false;
            Scanner input = new Scanner(System.in);

            while(!flag)
            {
                System.out.println("Insert your Autochef Username (or q to exit)");
                String line = input.nextLine();

                //exit condition from the database
                if(line.compareTo("q")==0)
                {
                    flag=true;
                }
                else
                {
                    int i = 0;
                    ArrayList<String> all_GroupIDs=new ArrayList<>();

                    //Search the username in the database
                    start = System.currentTimeMillis();
                    rs = stmt.executeQuery("SELECT * FROM find_user('"+line+"');");
                    end = System.currentTimeMillis();

                    int groupCount = 0;
                    i=1;
                    String output = "";

                    //Print the list of all groups of the user
                    while (rs.next())
                    {
                        groupCount++;
                        all_GroupIDs.add(rs.getString("GroupId"));
                        output += String.format("%2d: %12s%n", i,
rs.getString("GroupName"));
                        i++;
                    }

                    if (groupCount != 0)
                    {
                        //If the username is found
                        System.out.println("\n  Your groups  ");
                        System.out.println("-----------------");
                        System.out.println(output);

                        //Ask to the user which group they want to select
                        System.out.println("-----------------");
                        int choice1=0;
                        while(choice1<1 || choice1>=i)
                        {
                            System.out.println("\nChoose your group number (1-"+(i-
1)+")");
```

```
                                try
                                {
                                   choice1=Integer.parseInt(input.nextLine());
                                }
                                catch(NumberFormatException e)
                                {
                                    System.out.println("\n You must enter an integer");
                                     choice1=0;
                                }
                             }

                             //Look for all eating plans of the chosen group
                             String SQL2 ="SELECT * FROM
find_plans('"+all_GroupIDs.get(choice1-1)+"');";
                             start = System.currentTimeMillis();
                             rs = stmt.executeQuery(SQL2);
                             end = System.currentTimeMillis();
                             System.out.printf("Query %s successfully executed in %,d
milliseconds.%n", SQL2, end - start);
                             ArrayList<String> all_PlanIDs=new ArrayList<>();
                             i=1;
                             System.out.println("\nAvailable plans  ");
                             System.out.println("--------------------------------");

                             while (rs.next())
                             {
                                  //Print the list of all eating plans of the chosen group
                                  all_PlanIDs.add(rs.getString("Id"));
                                  System.out.printf("%2d: %27s%n", i,
rs.getDate("StartDate")+" / "+rs.getDate("EndDate"));
                                  i++;
                             }
                             System.out.println("--------------------------------");

                             //Ask to the user which eating plan they want to select
                             int choice2=0;
                             while(choice2<1 || choice2>=i)
                             {

                                System.out.println("\nChoose your plan number (1-"+(i-
1)+")");

                                try
                                {

                                   choice2=Integer.parseInt(input. nextLine());
                                }
                                catch(NumberFormatException e)
                                {
                                     System.out.println("\n TYou must enter an integer");
                                     choice2=0;
                                }
                             }

                             //Initialization of the shopping list for this eating plan
                             String SQL3 = "SELECT initial_shopping_list('"+
all_GroupIDs.get(choice1-1) +"', '"+ all_PlanIDs.get(choice2-1) +"');";
                             start = System.currentTimeMillis();
                             rs = stmt.executeQuery(SQL3);
                             end = System.currentTimeMillis();
                             System.out.printf("Query %s successfully executed in %,d
milliseconds.%n%n", SQL3, end - start);

                             //Retrieve relevant information for the shopping list
                             String SQL4 = "SELECT * FROM
print_shopping_list('"+all_GroupIDs.get(choice1-1)+"', '"+all_PlanIDs.get(choice2-1)
+"');";
```

```
                       start = System.currentTimeMillis();
                       rs = stmt.executeQuery(SQL4);
                       end = System.currentTimeMillis();
                       System.out.printf("Query %s successfully executed in %,d
milliseconds.%n", SQL4, end - start);
                       System.out.printf("Query results:%n");

                       /* Gather and print the query results. */
                       System.out.println("\n Shopping list");
                       System.out.println("--------------------------------------");
                       while (rs.next())
                       {
                          ProductName = rs.getString("Name");
                          Amount = rs.getInt("Amount");
                          UnitName = rs.getString("UnitName");
                          Cost = rs.getBigDecimal("Cost");
                          System.out.printf("- %15s, %5s, %5s, %5s%n", ProductName,
Amount.toString(), UnitName, Cost.toString());
                       }
                       System.out.println("--------------------------------------");

                  }
                  else
                  {
                       //if the username is not found in table AutoChefUser
                       System.out.println("This username doesn't exist, try again");
                       System.out.println("-----------------");
                  }
               }
           }

           input.close();
        }
        catch (SQLException e)
        {
           System.out.printf("Database access error:%n");
           printErrorMessages(e);
           releaseResources(con, stmt, rs);
           System.exit(-1);
        }

        /* Release the resources and end the program. */
        releaseResources(con, stmt, rs);
        System.out.printf("Program end.%n");

   }

   /* Print error messages related to an SQLException. */
   private static void printErrorMessages(SQLException e)
   {
   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();
   }
}

/* Release the database resources. */
private static void releaseResources(Connection con, Statement stmt, ResultSet rs)
{
   long start = 0;
   long end = 0;
```

```
    try
    {
        /* Close the result set. */
        if (rs != null)
        {
            start = System.currentTimeMillis();
            rs.close();
            end = System.currentTimeMillis();

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

        /* Close the statement. */
        if (stmt != null)
        {
            start = System.currentTimeMillis();
            stmt.close();
            end = System.currentTimeMillis();

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

        /* Close the connection. */
        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");
        printErrorMessages(e);
        }
        finally
        {
            /* Release resources to the garbage collector. */
            rs = null;
            stmt = null;
            con = null;
            System.out.printf("Resources released to the garbage collector.%n");
        }
    }
}
```