Exercise Sheet 4 (WS 2019)

3.0 VU Datenmodellierung 2 / 6.0 VU Datenbanksysteme

Information

General Information

The fourth exercise sheet covers more advanced features of SQL and database systems. You will practice the creation of database schemas, defining and validating data integrity constraints and building complex SQL queries (including procedural programs).

In this exercise you will hand in a single zip file (max. 5 MB). This zip file contains all necessary SQL files for creating and testing your database, see exercise 6. For testing your solutions we are providing you with a PostgreSQL server (version 11.5). You can connect via SSH at `bordo.dbai.tuwie.ac.at` and access the server via `psql`. You can find your login information in TUWEL.

Please carefully check the explanations and requirements stated at each task.

Important! Make sure that your submission contains (syntactically) correct SQL and that the SQL commands you hand in execute on the server `bordo.dbai.tuwien.ac.at` without producing any syntax error. If this is not the case (should there be for example a syntax error when trying to run the files), you will receive **no points** for the corresponding file.

This exercise sheet contains 6 exercises for which you can receive up to 15 points in total.

Deadlines

until 07.01.	**12:00 Uhr**	Upload your solutions to TUWEL
bis 07.01.	**12:00 Uhr**	Register for a discussion of your solutions in TUWEL

Solution discussion

At the solution discussion the correctness of your solution as well as your understanding of the underlying concepts will be assessed. This exercise is intended to improve your practical problem solving skills as well as your theoretical understanding of DBMS. Therefore you will be asked to demonstrate your knowledge of the topics from the lecture related to the exercises of this sheet in addition to explaining your submitted solution.

The scoring of the sheet is primarily based on your performance at the solution discussion. Therefore it is possible (in extreme cases) to get 0 points even though the submitted solution was technically correct. In particular, exercises that were not solved independently will always receive 0 points.

Note: We would like to remind you once again, that your solution has to work on the reference server (`bordo.dbai.tuwien.ac.at`) in order to receive any points. It does not matter if you are able to resolve the syntax errors immediately during the discussion, the *submitted* version must run.

Please be on time for your solution discussion. Otherwise we cannot guarantee that your full solution can be graded in your assigned time slot. Remember to bring your student id to the solution discussion. It is not possible to score your solution without an id.

Consultation Hours (optional)

In the week before the deadline there are consultation hours held by tutors. If you have problems understanding the topics of the exercise sheet or questions about the exercises, you are welcome to just drop in at these consultation hours. The tutors will gladly answer your questions and help you understand the subjects.

The goal of these consultation hours is to help you with **understanding the topics and specific tasks** of the exercise sheet. The tutors will not solve your exercises or check your answers before you hand them in.

Participation is completely voluntary — dates and locations of the consultation hours can be found in TUWEL.

Exercise: Database with PostgreSQL

The following tasks are based on the database you know from exercise sheet 1. The relational schema is repeated here for your. You can find the respectiv EER diagram in Figure 1 on the last page of this document.

Coffee	name, aroma, acidity, body
Lungo	name: *Coffee.name*
Ristretto	name: *Coffee.name*, roasttemp
Espresso	name: *Coffee.name*, pressure
Product	modelno, price, producedBy: *Manufacturer.brand*
CoffeeMachine	modelno: *Product.modelno*, name, pressure, numbcups
BasedOn	old: *CoffeeMachine.modelno*, new: *CoffeeMachine.modelno*
Supports	machine: *CoffeeMachine.modelno*, typeID: *CoffeePodType.ptid*, typesize: *CoffeePodType.size*
CanPrepare	machine: *CoffeeMachine.modelno*, coffee: *Coffee.name*
CoffeePodType	ptid, size, capacity
Compatible	fromID: *CoffeePodType.ptid*, fromSize: *CoffeePodType.size*, toID: *CoffeePodType.ptid*, toSize: *CoffeePodType.size*
CoffeePod	modelno: *Product.modelno*, material, typeID: *CoffeePodType.ptid*, typSize: *CoffeePodType.size*, contains: *coffee:name*
Manufacturer	brand, taxc
Review	manufacturer: *Manufacturer.brand*, id, txt
Licenses	machine:CoffeeMachine.modelno, typeID: *CoffeePodType.ptid*, typeSize: *CoffeePodType.size*, manufacturer: *Manufacturer.brand*, fee
Distributor	name, webseite
Sells	distributor: *Distributor.name*, product: *Product.modelno*
BulkDiscount	offeredBy: *Distributor.name*, label, discount
ConsistsOf	discountOf: *BulkDiscount.offeredBy*, discountLabel: *BulkDiscount.Label*, discountDiscount: *BulkDiscount.discount*, product: *Product.modelno*, quantity
PromotionCode	discountOf: *BulkDiscount.offeredBy*, discountLabel: *BulkDiscount.Label*, discountDiscount: *BulkDiscount.discount*, code

Task 1 (Create Sequences and Tables) [2 points]

Create a file `create.sql` containing the CREATE statements necessary to realize the given relations with SQL.

Consider the following:

(a) Change the relational schema in order to implement the following facts:
  - Every manufacturer has a main product.
  - Each coffee has one coffee pod that serves as "reference pod" for this coffee.

  You can implement these changes directly in the CREATE statements.

(b) Realize the consecutive numbering of the attribute `ModelNo` in the relation `Product` using a sequence. The sequence shall start at 1 and increment in steps of 1.

(c) Realize the numbering of the key attribute `PTId` in the table `CoffePodType` using a sequence. The sequence shall start at 7000 and increment in steps of 3.

(d) The attribute `Aroma` in the relation `Coffee` can only take the values `'Mild'`, `'Normal'`, and `'Strong'`. Implement this using an ENUM type.

(e) Represent the pressure of coffee machines as NUMERIC with two decimal digits (the pressure is measured in Bar).

(f) If two tables have cyclic FOREIGN KEY relations, then make sure that checking the FOREIGN KEYS occurs only at the time entries are COMMITed and not earlier.

(g) Do not use ä, ö, ü and similar characters in the names of relations, attributes, etc. .

(h) Make sure to ensure the following requirements satisfied using appropriate constraints:
  - For an Espresso, the pressure must be at least 7 (Bar).
  - A PromotionCode must consist of at least 10 characters and must contain at least one upper case latter.
  - The value of NumbCups for each CoffeMachine must be between 1 and 20 (including 1 and 20).
  - A CoffeePodType cannot be licensed more than once by a manufacturer.

(i) Make plausible assumptions for missing specifications (e.g.: types of attributes). Avoid NULL values in your tables, i.e., all attributes have to be set.

(j) You need not take care of/consider the (min,max) notation in the EER-diagram in Figure 1.

Task 2 (Inserting Test Data) [1 point]

Create a file `insert.sql` which contains INSERT commands with your test data for the tables you created in Task 1. Every table shall contain at least three rows. You can choose names, etc. as simple as you want, i.e., you don't have to fill your database with *real, existing* coffee machines, producers, resellers, etc.. Instead, you can use simple names like "coffee machine 1", "coffee machine 2", "Producer I", "Producer II", etc.. You are allowed to use the triggers and procedures of Task 4 to create the test data.

Task 3 (SQL Queries) [4 points]

Create a file `queries.sql` that contains the code for the following views.

(a) Create a view `MaxCapacity` that shows the maximal capacity of a coffee machine. The maximal capacity of a coffee machine is the maximal capacity over all types of coffee pods supported by the coffee machine.

(b) Create a view `AllBasedOn` that extends the `BasedOn` Table in such a way that if A is based on B and B is based on C, then A is based on C is also shown in the view. Note that this definition includes arbitrary long chains of coffee machines (and not only the three steps laid out above). E.g., if in addition to the above example, X is based on A, then X is also based on B and C.

(c) Create a view `CompatibleDistance`, that shows all coffee pods compatible with the pod type `ptid` 7000 and `size` 2, and over how many "hops" this compatibility is based on. As an example: If pod A is compatible with pod B and pod B is compatible with pod C, then A is compatible with C by two "hops" (or "steps").

Hint: For solving the latter two subtasks you need a recursive query. Make sure that your queries can deal with cyclic relationships and terminate also if such relationships are present. We also suggest that in your `insert.sql` file you provide suitable entries for the tables such that the recursive queries can be reasonable tested.

Task 4 (Creating and Testing Triggers) [6 points]

Create a file `plpgsql.sql` containing the code for the following triggers and procedures.

(a) Create a trigger that ensures that when adding a `CanPrepare` relationship, in case the coffee is an Espresso, the `pressure` of the coffee machine is not less than the `pressure` of the Espresso. If this condition is violated, the relationship shall not be added.

(b) Create a trigger that implements the following behavior on changes on the content of the table `License`:

- If the value of the `fee` attribute is increased for a tuple in the table, then the price of the corresponding coffee machines shall be increased by the difference between the new and old value of `fee`. In addition, using `RAISE NOTICE`, a message containing the new price of the machine shall be output.

- If an `UPDATE` operation sets the value of the `fee` attribute to the same value it already has, a warning shall be output using `RAISE`.

  Hint: More information on how to output messages using `RAISE` can be found in the PostgreSQL online documentation[1].

(c) Create a trigger on adding a coffee C in the relation `CanPrepare` for a coffee machine M that implements the following functionalities:

- For all coffee machines M' that are based on M, a corresponding tuple shall be added to the table `CanPrepare` stating that M' can prepare C.
- If `CanPrepare` already contains an entry stating that one of these (successor) machines can prepare C, then this information shall **not** be added a second time.

Note that you need not take care of the recursive relationships between the coffee machines, but that trigger can trigger trigger again.

(d) Create a procedure `CreateManufacturer` that automatically adds a new manufacturer. The procedure has three arguments: A number which indicates the number of `Products` for the manufacturer to be created, a country and the `brand` of the new manufacturer. We assume that for the new manufacturer no license agreements exist and no reviews have been written yet.

Make sure the following requirements are met:

- The new manufacturer must manufacture at least 3 products. If the values in the corresponding parameter is smaller, produce a suitable error message.
- Use the “country” parameter for the attribute `TaxC`, stating the country in which the manufacturer pays its taxes.
- The first new product must be a coffee machine. For the name of the coffee machine use “Coffee Machine of $Manufacturer”, where the name of the new manufacturer shall be used for the parameter $Manufacturer. Set the remaining attributes to your liking. All further products must be coffee pods.
- Create three types of coffee: “Ristretto of $Manufacturer”’, “Espresso of $Manufacturer” and “Lungo of $Manufacturer” (choose all additional attributes to your liking).
- Now create as many coffee pods as indicated by the parameter (minus the coffee machine already produced). Pick any arbitrary string as the material. You also need to maintain a counter, which indicates how many coffe pods have already been produced.

  Based on the divisibility of the current state of this counter by 3, the pod shall contain either the newly added Ristretto, Lungo, or Espresso (the ones described in the previous bullet point). As an example, 0 is a pod with a Ristretto, 1 a pod with a Lungo, 2 for an Espresso, 3 for a Ristretto, and so on). We suggest that you use the modulo operator for this task.

Nota bene: For all the entities you create (products and coffees), your procedure shall abort with an error message in case the database already contains entries with the same name. In the case of such an abort, the procedure must undo all its changes to the database.

[1] `https://www.postgresql.org/docs/11/static/plpgsql-errors-and-messages.html`

Task 5 (Cleaning Up) [1 point]

Create a file `drop.sql`, with the necessary DROP commands to remove all objects (tables, sequences, triggers, etc.) that were created as part of the previous exercises. You are **not** allowed to use the keyword CASCADE.

Task 6 (Testing your Database and Creating the Submission Archive) [1 point]

(a) Create the file `test.sql`. Think of a sensible test coverage for the requirementes of the previous exercises (i.e. the `CREATE` statements, the `VIEWS`, and the PL/pgSQL parts. Try to cover all cases, positive and negative ones. E.g., adding an Espresso to `CanPrepare` with both, a pressure that is ok and one that is too high.

(b) Create listings file with the name `listing.txt` that you created by executing your SQL files. It is recommended that you create the listing on the provided server at `bordo.dbai.tuwien.ac.at`. There start `psql`. With "`\o listing.txt`" you can redirect output to the file `listing.txt`. Then execute your files (when present) in the following order with "`\i xxx.sql`":

(1) `create.sql`

(4) `insert.sql`

(2) `queries.sql`

(3) `plpgsql.sql`

(5) `test.sql`

(6) `drop.sql`

Combine all files (i.e. the 6 files mentioned above an the `listing.txt`) to a zip archive `blatt4.zip` and upload it in TUWEL.

EER-Diagramm

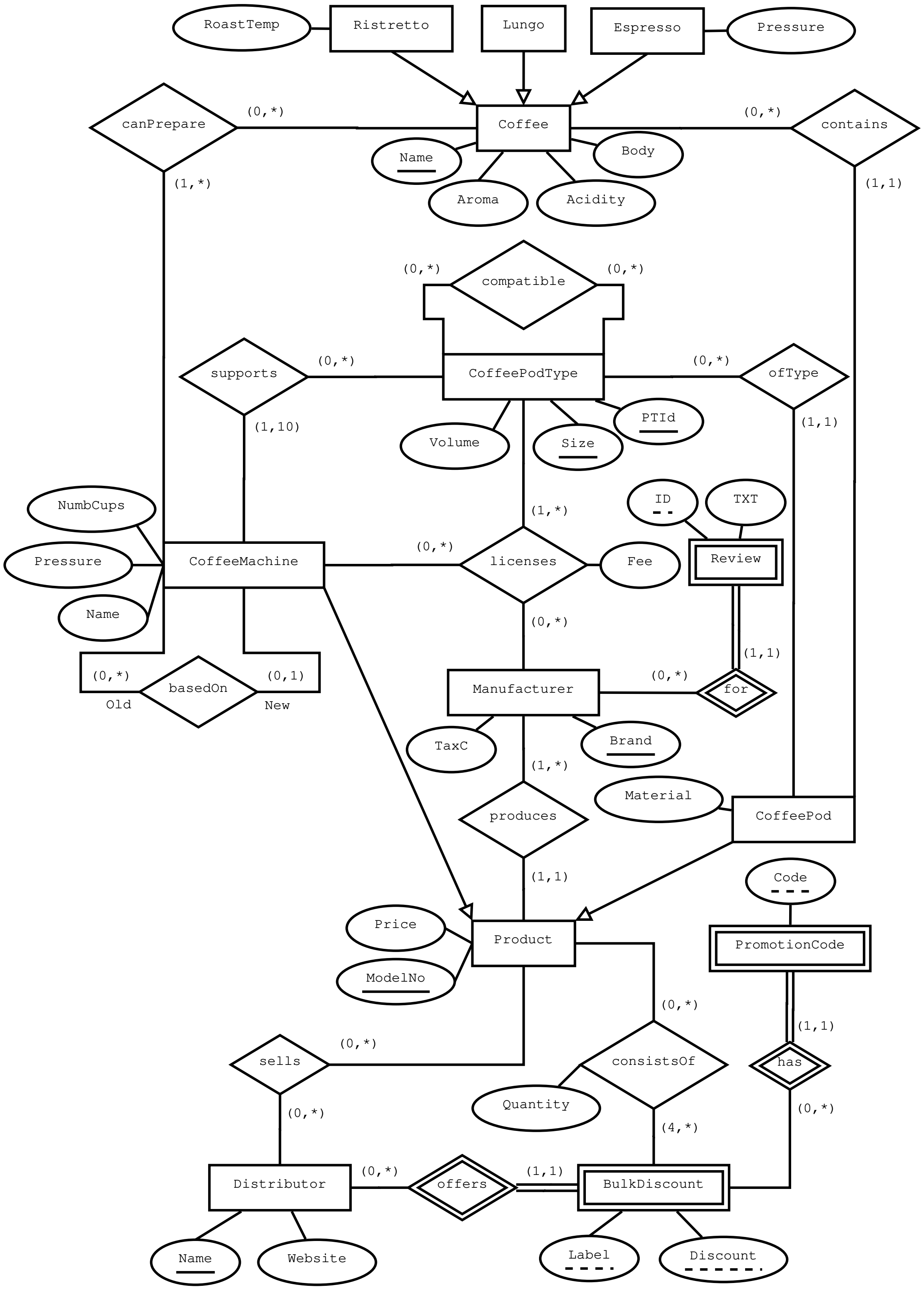

Abbildung 1: EER-Diagram