Exercise Sheet 4 (WS 2020)

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 at `https://tuwel.tuwien.ac.at/mod/assign/view.php?id=994171`.

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 11.01.	**12:00 Uhr**	Upload your solutions to TUWEL
bis 11.01.	**12:00 Uhr**	Register for a discussion of your solutions in TUWEL

Solution discussion

The solution discussion is performed this semester via Zoom. At the beginning of the discussion you will be asked to turn on your camera and present your student id, such that your examiner is able to verify your identity. Afterwards, the actual solution discussion starts, where 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 Zoom solution discussion. Otherwise we cannot guarantee that your full solution can be graded in your assigned time slot. Remember to have your student id ready for the Zoom solution discussion. It is not possible to score your solution without an id.

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.

book	BID, titel, pages
nonFiction	BID: *book.BID*
novel	BID: *book.BID*, genre
poetry	BID: *book.BID*
about	nonFiction: *nonFiction.BID*, novel: *novel.BID*
hasWritten	author: *author.AID*, book: *book.BID*, startDate
author	AID, name, DOB
edition	book: *book.BID*, EDNR, year, publisherName: *publisher.name*
printing	book: *edition.book*, EDNR: *edition.EDNR*, PNR, printer
specialEdition	book: *edition.book*, EDNR: *edition.EDNR*, reason
publisher	name, budget
department	publisherName: *publisher.name*, area, city
advertises	publisherName: *publisher.name*, channel: *marketingChannel.chName*, targetGroup: *targetGroup.description*, date
marketingChannel	chName, costs
social	channel: *marketingChannel.chName*, platform
newspaper	channel: *marketingChannel.chName*, book: *book.BID*, circ
targetGroup	description, age
uses	targetGroup: *targetGroup.description*, socialChannel: *social.chName*
interestedIn	targetGroup: *targetGroup.description*, book: *book.BID*
loves	targetGroup: *targetGroup.description*, author: *author.AID*
hates	targetGroup: *targetGroup.description*, author: *author.AID*
aka	alias: *author.AID*, aliasOf: *author.AID*
basedOn	new: *Novel.BID*, old: *Novel.BID*

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:

- Each publisher has a main department.
- Each edition has a first printing.

You can implement these changes directly in the CREATE statements.

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

(c) Realize the numbering of the key attribute `AID` in the table `author` using a sequence. The sequence shall start at 10,000 and increment in steps of 7.

(d) The attribute `age` in the relation `targetGroup` can only take the values `'young'`, `'middle-aged'`, and `'old'`. Implement this using an ENUM type.

(e) Represent the budget of publishers as NUMERIC with two decimal digits (the budget is measured in euros).

(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 edition, the year must be at least 1400 (since only in the middle of the 15th century printing was invented).
  - An area code of a department (represented by the attribute "area") must consist of exactly 5 symbols, which start with 2 alphanumeric characters followed by 3 digits.
  - The value of the budget of each publisher must be between 10€ and 1,000,000€ (including 10€ and 1,000,000€).
  - A publisher cannot advertise a target group multiple times at the same date.

(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* authors, books, publishers, etc.. Instead, you can use simple names like "Author 1", "Author 2", "Book 1", "Book 2", 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 `NumberOfLargeBooks` that shows the number of large books per author. The number of large books of an author is defined as the number of all books that have been written by the author and that contain more than 200 pages.

(b) Create a view `AllAliases` that extends the `aka` Table in such a way that if author A is an alias of author B and B is an alias of author C, then A is an alias of C is also shown in the view. Note that this definition includes arbitrary long chains of author aliases (and not only the three steps laid out above). E.g., if in addition to the example above, X is an alias of A, then X is also an alias of B and C.

(c) Create a view `NovelDistance` that contains all tuples (X, Z, G, S), such that novel X is connected to novel Z over S "hops" of novels of the same genre G as X and Z. As an example: If the novels X, Y und Z have the same genre G and if furthermore X is based on Y and Y is based on Z, then X is based on Z via 2 "hops" of novels of the same genre. Yet, if in the previous example Y did not have the same genre G as X and Z (but if Y had a different genre G_Y with $G_Y \neq G$) then X would not be based on Z via novels of the same genre.

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 when adding a `hasWritten` relationship that the `start` date of the `hasWritten` relationship does take place before the date of birth (`DOP`) of the author. If this condition is violated, the relationship shall not be added.

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

- If the `costs` attribute is changed for a tuple in the table, then the `budget` attribute of the corresponding publishers shall be adapted by the difference of the new and old value of `costs`. For example, if the `costs` for a marketing channel increase, then also the `budget` of all publishers that use this marketing channel for advertisements for target groups needs to increase. In addition, using `RAISE NOTICE`, a message containing the new budget shall be output.
- If an `UPDATE` operation sets the value of the `costs` attribute of a marketing channel 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 target group Z in the relation `loves` for an author A that implements the following functionalities:

- If A is an alias of other authors A' then corresponding tuples shall be added to the table `loves`, stating that authors A' are loved by the same target groups Z that love alias A.
- If `loves` already contains an entry stating that one of these (successor) alias authors is loved by target group Z, then this information shall **not** be added a second time.

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

(d) Create a procedure `CreateAuthor` that automatically adds a new author. The procedure has three arguments: A number which indicates the number of `books` of the author and the `name` and `date of birth` of the new author. We assume that the new author is neither loved nor hated by anyone so far.

Make sure the following requirements are met:

- The new author has to have written at least 3 books. If the values in the corresponding parameter is smaller, produce a suitable error message.
- Use the corresponding parameters of the procedure to set the name and date of birth parameter of the new author.
- In addition, an alias needs to be generated for each author. Create an alias by extracting the first letter of the names of the author (e.g. the alias of “Cleo Virginia” would be “C.V.”). You are allowed to assume that an author has only a single first and last name (no middle name). All further attributes of the alias can be chosen arbitrarily. Do not forget to link the created alias author to the real author using the aka relationship.
- Now the specified number of books needs to be created. The procedure shall contain a counter, indicating how many books have already been generated. Based on the divisibility of the current state of this counter by 3 a novel, poetry or non-fiction book shall be generated. As an example, if the counter’s state is 0 a novel, if it is 1 a poetry, if it is 2 a non-fiction book, if it is 3 a novel and so on shall be generated. We suggest that you use the modulo operator for this task.

  Use the following titles for the generated books: “Novel of $aka”, “Poetry of $aka” and “Non-fiction of $aka”, where $aka represents the alias of the author, which has been

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

generated in the previous step (e.g. "C.V."). Do not forget to respectively create an entry in the novel, poetry and non-fiction table. Choose any further attributes of books and novels to your liking.

- Finally, enter tuples into the "hasWritten" relation that represent that the generated author has written under their alias the books which were generated in the previous step.

Nota bene: For all the authors you create (including all aliases), 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.

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., inserting an entry into `hasWritten` with a start date that is before and one that is after the date of birth of the respective author.

(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-Diagram

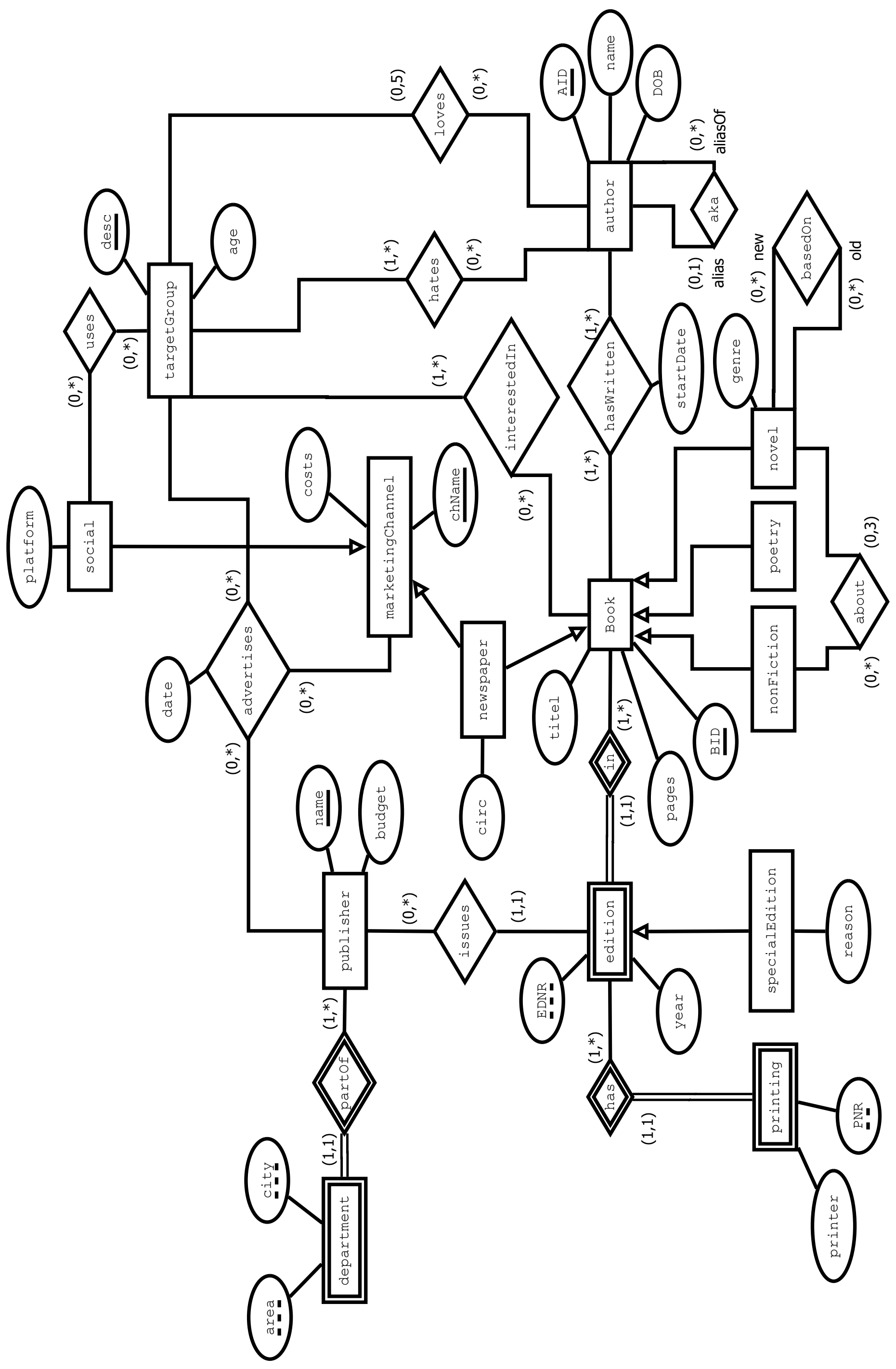

Figure 1: EER-Diagram