DAT151
Database and Unix System Management

Spring 2024

Assignment 4

Obligatory assignment. Deadline: Sunday 10.03.2024.

Your code should be easy to read:

- *Use indentation in your code.*
- *DDL for CREATE TABLE should have each column definition on its own line.*
- *Do not use screen dumps to show code. Code should be normal text, formatted as code.*

The report should include all necessary commands to complete the tasks, printout from the system, explanation of what is done, the result and explanation of the result.

Remember to include the names of the group members on the front page of the report. The report can be in English or Norwegian. The report should be handed in via Canvas.

The assignment should be accomplished in groups of one to three students. Signing up for a group will be closed on Thursday 07.03.

You must select a group "Lab4 N" when delivering to see all the comments that you will get on this assignment.

Lecturers:
Bjarte Kileng <Bjarte.Kileng@hvl.no>, room E418
Haakon André Reme-Ness <Haakon.Andre.Reme-Ness@hvl.no>, Fabrikkgaten 5, R235-02

Task 1: Triggers

As a normal database user (i.e. not root), create triggers on table **TheTable** so that all changes of **TheTable** will be logged in **LogTable**.

Make sure all values before and after a change are logged.

- For DELETE, you must log the action DELETE, and the row before values.
- For INSERT, you must log the action INSERT, and the row after values.
- For UPDATE, you must log the action UPDATE, and both the row before and after values.
  - The LogTable must clearly identify the old and the new values.

Your delivery must include:

- The DDL of the triggers,

- the DDL of the tables,
- output that demonstrates the working of the triggers.

Task 2: Temporal database

Create a table **AnotherTable** as a *System-Versioned* table. Insert some data, then do some DELETEs, UPDATEs and INSERTs.

After all modifications are done, use queries with "*FOR SYSTEM_TIME AS OF*" to list the before and after values after each modification of the table.

Task 3: Integrity Constraints

As a normal database user (i.e. not root), create a teacher table, with numeric attributes for salary, bonus and total. Implement the following integrity constraints:

1. salary must be between 1 000 and 100 000.
2. total must be the sum of salary and bonus, and calculated AUTOMATICALLY.

Insert some test data to verify whether your solution works. Document the output.

Is the table 3NF?

Task 4: Order of triggers

1. Test the following, and give necessary explanations:
   a) Create three tables T1, T2, and T3.
   b) Create a trigger tr12 before insert on T1, insert into T2 a row.
   c) Create a trigger tr23 after insert on T2, insert into T3 a row.
   d) Create a trigger tr13 after insert on T1, insert into T3 a row.
   e) Insert a row into T1, in which order are the triggers fired? Why?
2. Can there be any unexpected result or deadlock regarding the order of triggers?

Task 5: Pendant DELETE

Implement the following as a normal database user (i.e. not root):

1. Create two tables with a one-to-many relation, and implement referential integrity. (Create foreign keys on the "many"-table).
2. Add some rows to each table. The child table must have several rows referencing each row of the parent
3. Use triggers to implement this rule which is known as pendant DELETE (Mullins page 435): A row in the parent table should be deleted if no rows in the child table refer to it any more (when you delete the last row in the child table referencing it).

Explain the point of doing this and why it is difficult compared with other rules on p.435.

Task 6: Concurrency

An application uses a DBMS to store data about events and participants. Events can have many participants, but the number of spaces should still be limited.

All work against the database must be done as a normal database user (i.e. not root).

The table below shows examples of data:

eventId	**Name of event**	**Date and time**	**Tot. Spaces**	**pId**	**Sure name**	**Given names**
13	Stolzekleiven opp	2024-09-12 10:00:00	50000	1357859	Helland	Unto
9	Knarvikmila	2024-06-23 08:30:00	53000	4346917	Norheim	Marit
7	Led Zeppelin at Koengen	2024-07-12 19:30:00	25000	4234474	Langerud	Walther
2	The 7-mountain hike	2024-05-23 09:00:00	32000	1639433	Bremnes	Victoria
9	Knarvikmila	2024-06-23 08:30:00	53000	5042431	Rogne	Irina

The fields are:

- **eventId**: Unique id of event.
- **Name of event**.
- **Date and time**: Date and time when the event starts.
- **Tot. Spaces**: Total number of available spaces at this event.
- **pId**: Unique id of participant.
- **Sure name**: Last name (family name) of participant.
- **Given name**: First name(s) (sure name) of participant.

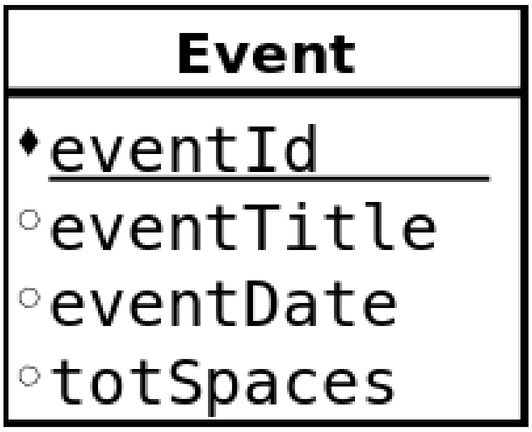
Observe that the character set for the text data is UTF-8.

Do the following:

1. Make a normalized, 3NF logical for the above data and implement the model in a MariaDB database using InnoDB tables. Fill the database with data from the attached file *data.txt*.

   The model should have an attribute showing the total number of spaces at each event. This number is the number read from the data file.

   The data model should include a table *Event*. Use the following layout for this table:



Figur 1: The Event table

Several approaches can be used to fill the database, e.g. create a program that reads the data file and fills the database. The fastest, and probably also easiest solution though is to create a table that temporarily holds all the data. From this table, load data into the normalized tables and afterwards delete the auxiliary table.

Some tips for loading the data:

- “LOAD DATA LOCAL INFILE” can load data from a local file into a table.
- You will need to use sub queries when loading data from the auxiliary table into the normalized database.
- Use SQL with the DISTINCT keyword when finding people and events in the auxiliary table.

2. Before continuing, you must confirm that your model is working, and that the data got correctly loaded into the database.

   Formulate the queries below and test your database.

   i. Find the maximum number of events that a single participant will attend.

      Your SQL should give the following result:

```
+------------+
| MAX(arr.c) |
+------------+
|          4 |
+------------+
```

   ii. List names of the participants that attend the largest number of events.
      - You should not know in advance the number of participants involved. Using LIMIT to restrict the result set will not be approved.
      - Sorting a result set to find the first is a bad solution. The performance will be bad and the query will be resource demanding. Avoid ORDER BY for this query!

Your SQL should produce a result with three names, including:

```
+---------+-----------+------------+
| pId     | givenNames | surename   |
+---------+-----------+------------+
| 4359525 | Astrid     | Gjerstad   |
| ...                               |
```

iii. What events are attended by Ludvig Rustad?

Your SQL should produce a result with three events, including:

```
+-------------------+
| eventTitle        |
+-------------------+
| Knarvikmila       |
| Lyderhorn opp     |
| ...               |
```

iv. List participants that attend exactly 3 events.

Your SQL should produce a result with 89 participants, including:

```
+---------+-----------+--------------+
| pId     | givenNames | sureName     |
+---------+-----------+--------------+
|   25113 | Zoulikha   | Wilhelmsen   |
|   72717 | Ulf        | Erstad       |
|   ...                               |
```

v. What events do they attend, the participants that attend 3 events?

Your SQL should return all events except the event with *eventId* equal to 5.

- Your query should not refer explicitly to any events, nor *eventId* equal to 5.

```
+---------+------------------------+
| eventId | eventTitle             |
+---------+------------------------+
|       1 | Ulriken down           |
|       2 | The 7-mountain hike    |
|       ...                        |
```

vi. Are there any events that is not attended by anybody that attends three events?

- Your query should not refer explicitly any of the events.

Your SQL should produce a result with one event:

```
+---------+----------------------------+
| eventId | eventTitle                 |
+---------+----------------------------+
|       5 | Guided tour to Løvstakken  |
+---------+----------------------------+
```

vii. Are there any events that is attended only by the participants that attend only one event, i.e. participants attending more than one event, is there an event that none of them attend, but attended by those attending one event?

   - Your query should not refer explicitly any of the events.

   Your SQL should produce a result with one event:

```
+---------+----------------------------+
| eventId | eventTitle                 |
+---------+----------------------------+
|       5 | Guided tour to Løvstakken  |
+---------+----------------------------+
```

3. Make a store procedure *takeSpace* that will book a space for an existing participant at an event.

   The stored procedure has the following characteristics:

   - If there are available spaces at the event, book a space for the participant.
   - If the event is sold out, do nothing.
   - The procedure should be accessible by concurrent users, i.e. locking is required with a 3NF model.
   - The input parameters are:
     - **pId**: Unique id of an existing user.
     - **eventId**: Unique id of an existing event.

   Observe: The stored procedure should not modify column *totSpaces* of the table *Event*.

4. Start profiling and check the time consumption used by the stored procedure when booking a space at an event. What takes most time?

   Discuss how multiple concurrent bookings for the same event would influence the time consumption.

5. Can you advice a denormalisation scheme that would reduce the time consumption?

   Implement the new model, and update *takeSpace* to take the new model into account. Try to avoid the need for explicit locks in the updated version of *takeSpace*.

Use profiling and measure the time consumption when booking a space at an event with the new version *takeSpace*.

The following sections from the online manuals can be helpful:

The following information tell how to create and use stored procedures:

- Stored Procedures
- Programmatic & Compound Statements

The function ROW_COUNT can be useful in *takeSpace* for the denormalised model.Triggers in MariaDB and MySQL.

The following information tell how to create and use triggers:

- CREATE TRIGGER
- Programmatic & Compound Statements
- SHOW TRIGGERS
- SHOW CREATE TRIGGER
- Information Schema TRIGGERS Table

Last changed, 21.02.2024 (BK)