DAT151
Database and Unix System Management

Spring 2018

Assignment 3

Obligatory assignment. Deadline: Monday 05.02.2018

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 it's learning.

The assignment should be accomplished in groups of three or four students.

Lecturers:
Bjarte Kileng, Bjarte.Kileng@hib.no, room E418
Maksim Melnik Storetvedt, Maxim.Storetvedt@hvl.no, room E506
Faustin Ahishakiye, Faustin.Ahishakiye@hvl.no, room E506
Violet Ka I Pun, violet@foldr.org, room E415

Task 1: Triggers

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

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

Task 2: 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.

Note that you are free to use the CHECK constraints, but be aware that a CHECK clause is parsed but ignored in MySQL 5.x (and MariaDB 5.x). In this case just make sure what you do is THEORETICALLY correct.

Task 3: Order of triggers

1. Create a table, as a normal database user (i.e. not root), with multiple triggers.
2. 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?
3. Can there be any unexpected result or deadlock regarding the order of triggers?

Task 4: 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 a couple of rows to each table.
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).

This task can be a bit difficult. If you are unable to complete it, I will still accept your answer if you explain the point of doing this and why it is difficult compared with other rules on p.374.

Task 5: Concurrency

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

Unless explicitly required, all work must be done as a normal database user (i.e. not root).

The table below shows examples of data:

arrId	**Name of arrangement**	**Date and time**	**Num. spaces**	**pId**	**Last name**	**First names**
5	Stolzekleiven opp	2017-09-12 10:00:00	50000	1357859	Helland	Unto
9	Knarvikmila	2017-06-23 08:30:00	53000	4346917	Norheim	Marit
7	Led Zeppelin at Koengen	2017-07-12 07:30:00	25000	1639433	Bremnes	Victoria
2	The 7-mountain hike	2017-05-23 09:00:00	32000	5042431	Rogne	Irina
9	Knarvikmila	2017-06-23 08:30:00	53000	4823164	Bergli	Qendrim

The fields are:

- **arrId**: Uniq id of arrangement.

- **Name of arrangement**.
- **Date and time**: Date and time for start of arrangement.
- **Num. Spaces**: Total number of available spaces at this arrangement.
- **pId**: Uniq id of participant.
- **Last name**: Last name (family name) of participant.
- **First names**: First name(s) (sure name) of participant.

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 data file.

   The model should have an attribute showing the total number of spaces at each arrangement. The value should not change when people sign up for an arrangement.

   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 database and afterwards delete the auxiliary table.

   Some tips for loading the data:

   - "LOAD DATA LOCAL INFILE" can load data from a 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 arrangements in the auxiliary table.
2. Write SQL for the following queries against the 3NF model, perform your queries and document the output:

   i. Find the name of all participants that will attend more than one arrangement.

   ii. Find the maximum number of arrangements that a single participant will attend.

   iii. List names of the participants that attend the largest number of arrangements.

   iv. Some persons attend 3 arrangements. List all arrangements that they attend.
3. Make a store procedure *takeSeat* that will book a space for an existing participant at an arrangement. The stored procedure has the following characteristics:

   - If there are available spaces at the arrangement, book a space for the participant.
   - If the arrangement is sold out, do nothing.
   - The procedure should be accessible by concurrent users, i.e. locking is probably required with a 3NF model.
   - The input parameters are:
     - **pId**: Uniq id of an existing user.
     - **arrId**: Uniq id of an existing arrangement.
4. Start profiling and check the time consumption used by the stored procedure when booking a space at an arrangement. What takes most time?

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

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

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

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

   The following information tell how to create and use stored procedures

   - Stored Programs and Views
   - MySQL Compound-Statement Syntax

   The function ROW_COUNT can be useful in *takeSeat* for the denormalised model.

Notes: Triggers in MariaDB and MySQL

Triggers have been supported in MySQL since version 5.0.2, but the implementation is not mature until version 5.1.x. The following sections from the online manual might be helpful:

- 13.1.19. CREATE TRIGGER Syntax
- 13.6. MySQL Compound-Statement Syntax
- 13.7.5.40. SHOW TRIGGERS Syntax
- 13.7.5.13. SHOW CREATE TRIGGER Syntax
- 20.23. The INFORMATION_SCHEMA TRIGGERS Table

The above documentation is valid also for MariaDB.

CHECK constraints are parsed but not implemented in MariaDB 5.x and MySQL 5.x, but we can use triggers instead.

Let us implement a CHECK constraint using triggers. We have several possibilities when a value is out of range:

- We can let the application do the check and ignore the problem in the DBMS.
- We can insert a default value, e.g. “0” or NULL.
- We can insert a default value, e.g. “0” or NULL and issue a warning.
- We can abort the operation and issue an error.

I will show how we can implement the two last scenarios in MariaDB and MySQL.

Using a default value and issue a warning

Let us create a table *Student*:

```
DROP TABLE IF EXISTS Student;
CREATE TABLE Student (
   studentId SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
   Age TINYINT UNSIGNED,
   FirstName VARCHAR(40),
   LastName VARCHAR(40),
   PRIMARY KEY (studentId) )
ENGINE InnoDB;
```

We will include triggers both for inserts and updates. The insert trigger can look like:

```
DROP TRIGGER IF EXISTS CheckAge_Insert;
DELIMITER |
CREATE
  TRIGGER CheckAge_Insert BEFORE INSERT ON Student
  FOR EACH ROW BEGIN
    IF NEW.Age > 120 THEN
      SET NEW.Age = -1; # -1 is an illegal value for the Age column
    END IF;
  END
|
DELIMITER ;
```

The update trigger will be similar.

We do not need to check for negative values for the **Age** column, since it is *UNSIGNED*.

In the example, the maximum age is hard coded into the trigger, but we can also declare a *MaxAge* function. Remember then to declare the return value as *DETERMINISTIC* (see CREATE PROCEDURE and CREATE FUNCTION Syntax). With multiple constraints we can also use a *Constraint* table that is queried by trigger code.

Try to insert some values with some ages out of range. To see the last warning, issue:

SHOW WARNINGS;

Abort the operation and issue an error

Often we would like to abort the operation when a value is out of range. In MariaDB 5.5 and MySQL 5.5 we can raise an error using *SIGNAL* and *RESIGNAL* and specify an error message, see 13.6.7. Condition Handling in the MySQL manual.

Last changed, 17.01.2018 (BK)