DAT151
Database and Unix System Management

Spring 2024

Assignment 7

Obligatory assignment. Deadline: Sunday 26.04.2024.

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 it's learning.

The assignment should be accomplished in groups of two or three students. Signing up for a group will be closed on Friday 19.04.

You must select a group "Lab7 N" when delivering to see all the comments that you will get on this assignment.

***Observe**: Before you modify an existing configuration file, save a copy of the original somewhere on the computer.*

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

Last changed, 03.04.2024 (BK)

Task 1: Normal Forms

An logical database schema is given on the next page, and DDL for the database is found in the attached file *model.sql*.

For the model on the next page, answer the following (explain why, not just yes/no):

1. Is this schema in 1NF?
2. Is it in 2NF?
3. Is it in 3NF?

The database schema is shown on the next page, where primary keys are underlined and candidate keys are marked with waves above the column names. The primary key *fee* of entity **Fee** is a surrogate key.

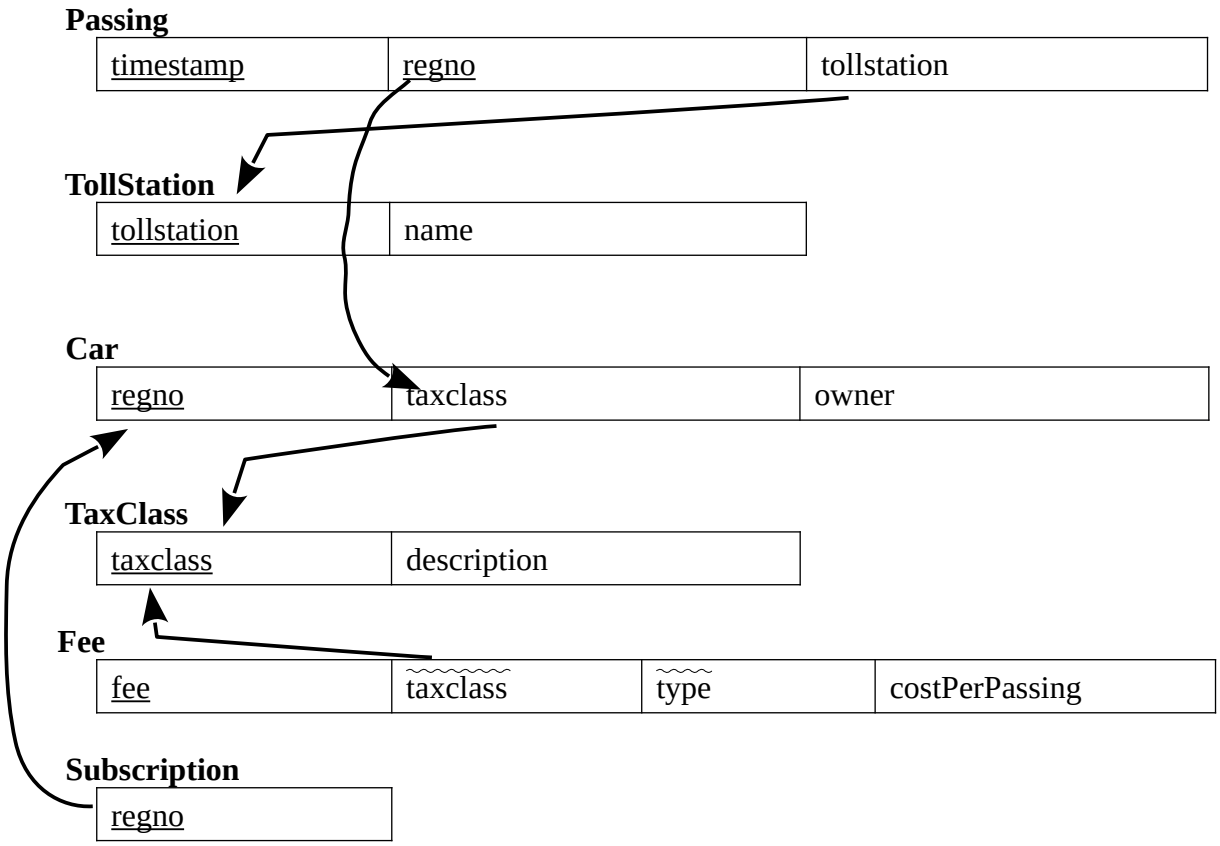

Illustration 1: Logical database schema

An EER diagram of the model is shown below, made by Dbeaver, a tool that has support for most major DBMSs.

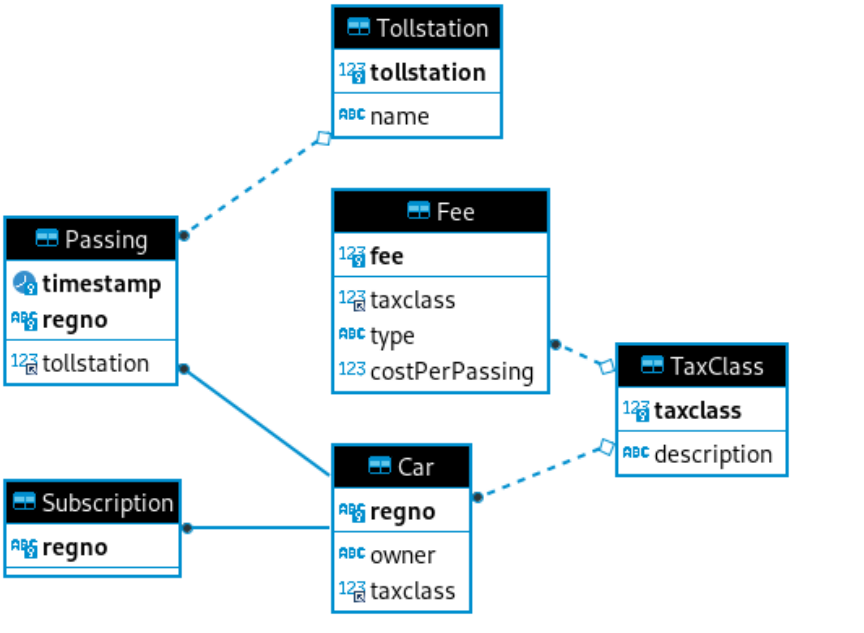

Illustration 2: EER diagram of the model

Task 2: Create a physical schema and a test environment

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

You will now implement the schema from the previous task in your MariaDB database and fill it with test data before you in the next task will try to optimize the performance.

Fill the database with data from the file *carpassingdb.txt*:

```
wget https://eple.hvl.no/fag/dat151/v2024/carpassingdb.txt
```

The fields are separated with ";" with each row on a separate line. The fields are:

1. Data and time for passing a toll station
2. Car registration number
3. Id of toll station
4. Name of toll station
5. Name of car owner
6. Id of car tax-class
7. Description of tax class
8. Whether the car has a subscription, 'yes' or 'no'
9. Fee for this car when passing the toll station

The file *carpassingdb.txt* is large and contains a lot of data. The fastest, and probably also easiest solution 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.

You should probably adjust parameters in your MariaDB setup for acceptable performance with big databases, e.g. increase the size of the data cache (named *buffer pool* in MariaDB).

Observe: If loading the data takes much time, increase the size of the data cache!

Check your setup with the program *mysqltuner*. Adjust the parameters **before** you start to load data from the auxiliary table into the normalized model. Filling the normalized model will take a long time if the parameters are wrong.

The program *mysqltuner* needs some statistics before giving good advices. Therefore, make some queries against the temporary table, e.g.:

```
SELECT COUNT(*) FROM <auxiliary table>;
SELECT * FROM <auxiliary table> ORDER BY timestamp DESC LIMIT 20;
SELECT * FROM <auxiliary table> ORDER BY timestamp ASC LIMIT 20;
```

The first query should return:

```
+----------+
| COUNT(*) |
+----------+
|  4154196 |
+----------+
```

Then, run *mysqltuner* and follow the advices. You install the program from the repository *epel*.

Task 3: Performance of the physical model

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

To give a fair comparison between the queries, start all comparisons from an empty cache.

To clear the Linux file system cache, as root:

```
sync
echo 3 > /proc/sys/vm/drop_caches
```

To clear the MariaDB caches:

```
sudo systemctl restart mariadb
```

Consider the SQL queries below:

a) Query one:

```
SELECT C.owner, P.timestamp
    FROM Car C JOIN Passing P USING(regno)
    WHERE YEAR(P.timestamp)=2024 AND MONTH(P.timestamp)=2
    AND DAYOFMONTH(P.timestamp)=1 AND HOUR(P.timestamp)=23;
```

b) Query two:

```
SELECT C.regno AS regno, Sum(F.costPerPassing) AS totalfee
    FROM Car C JOIN Passing P USING(regno)
        JOIN TaxClass T USING(taxclass)
        JOIN Fee F USING(taxclass)
        JOIN Subscription S USING(regno)
        WHERE F.type='withsubscription'
        GROUP BY C.regno HAVING totalfee > 5000;
```

c) Query three uses a subselect to find the same result as **b)**:

```
SELECT C.regno AS regno, Sum(F.costPerPassing) AS totalfee
   FROM Car C JOIN Passing P USING(regno)
       JOIN TaxClass T USING(taxclass)
       JOIN Fee F USING(taxclass)
       WHERE F.type='withsubscription'
       AND C.regno IN (SELECT regno FROM Subscription)
       GROUP BY C.regno HAVING totalfee > 5000;
```

d) Query four: Reformulate the query of **b)** and ask for cars without a subscription.

e) Query five: Reformulate the query of **c)** and ask for cars without a subscription.

First, explain with your own words what is returned by the queries above. Then carry out the following and document the input and output:

- Turn on profiling, run each query three times to get data into the buffer pool, and use SHOW PROFILE(S) to see the statistics.
- Use EXPLAIN to see how indexes, JOINs, and subqueries are handled by the optimizer.

Now, for each of the query, try to improve performance by the following (restore to the original schema after each test):

1. Denormalization:

   Describe the denormalization you do, and explain what normal forms it violates. You will

need to implement mechanisms to make sure that any redundancy is maintained correctly without any risk of loss of data integrity.

2. Using indexes:

   Create indexes and specify index hints (use index, ignore index, force index, ...), if necessary, to influence the execution of the queries.

3. If an index is not being used, discuss whether a rewrite of the WHERE clause can make the DBMS use the index.

For **each and every query**, the report must document:

A) The query that is discussed.

B) Profiling results of the original query

C) Denormalisation approaches, and for each approach, you must restart MariaDB, clear the caches and show the profiling results.

D) For each index that is tested, use EXPLAIN to check if the index is used. You must show the EXPLAIN output. If the index is used, clear the caches and show the profiling results. Observe, sorting can benefit from an index even if EXPLAIN does not show that the index is used.

E) If an index is not used, try to rewrite the WHERE clause and see if that can make the DBMS use the index. You must show the EXPLAIN output. If the index is used, clear the caches and show the profiling results. Observe, sorting can benefit from an index even if EXPLAIN does not show index use.

For each query a) to f), and each of B) to E) the following steps must be repeated and documented in the report:

- Restart MariaDB and clear the caches.
- Show profiling results:

```
MariaDB [DAT151_LAB7_V2024]> SHOW PROFILES;
+----------+-----------+------- ... +
| Query_ID | Duration  | Query  ... |
+----------+-----------+------- ... +
|        1 | ......... | SELECT ... |
|        2 | ......... | SELECT ... |
|        3 | ......... | SELECT ... |
+----------+-----------+------- ... +
```

There are 24 combinations of a) to f) with B) to E), but some of the combinations will result in the same denomalization approach, and rewrites for indexes may not be needed for all combinations. As a minimum, the report **must** include at least 15 profiling tests.

For at least one of the query, you should find a denormalization approach that improves the speed with at least a factor of 1000. And, you should find some rewrites of WHERE clauses that are good for indexes.