Answers to questions for assignment 4, BrianAir project

Question 8:
a) How can you protect the credit card information in the database from hackers?

Answer:
There are multiple ways you can protect that information, first you could limit the access by assigning special access privileges.
You could also encrypt the information or use hashing on sensitive and vulnerable credit card data.

b) Give three advantages of using stored procedures in the database (and thereby execute them on the server) instead of writing the same functions in the system's frontend (in for example JavaScript on a Web page)?

Answer:

1. The procedure calls are quick and efficient since they are only stored and compiled once. The executable code is also automatically cached, therefore it will lower the memory requirements.

2. Updating a stored procedure in the database automatically will apply changes to all applications using it.
Which will avoid the need to redeploy front-end code.

3. One statement can trigger several statements inside the procedure, resulting in more efficient queries.

Question 9:
a) In session A, add a new reservation.
b) Is this reservation visible in session B? Why? Why not?

Answer:
Since we started a transaction, changes are not visible in the second terminal window that is session B, because each session is locked.
We need to first commit the transaction to make the changes visible and update the database.

c) What happens if you try to modify the reservation from A in B? Explain what happens and why this happens and how this relates to the concept of isolation of transactions.

Answer:
We can't perform actions in window B due to the active transaction in window A, since session B is waiting for session A to commit.

We tried to delete the reservation from session B, but nothing happened for a while and then ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction, occured.
We then performed commit from session A, and session B could thereafter modify the reservation by deleting the last one added.

Question 10:
a) Did overbooking occur when the scripts were executed? If so, why? If not, why not?

Answer:
Overbooking did not occur when the scripts were executed since on of the scripts showed the error, "The reservation has no contact yet".
Which indicates by itself that it handled the concurrency.

b) Can an overbooking theoretically occur? If an overbooking is possible, in what order must the lines of code in your procedures/functions be executed.

Answer:
A overbooking could theoretically occur if the check for free seats is done in both scripts before a booking is done.

c) Try to make the theoretical case occur in reality by simulating that multiple sessions call the procedure at the same time. To specify the order in which the lines of code are executed use the MySQL query SELECT sleep(5); which makes the session sleep for 5 seconds. Note that it is not always possible to make the theoretical case occur, if not, motivate why.

Answer:
Could not make the theoretical case work with sleep.

d) *Modify the testscripts so that overbookings are no longer possible using (some of) the commands START TRANSACTION, COMMIT, LOCK TABLES, UNLOCK TABLES, ROLLBACK, SAVEPOINT, and SELECT…FOR UPDATE. Motivate why your solution solves the issue, and test that this also is the case using the sleep implemented in 10c. Note that it is not ok that one of the sessions ends up in a deadlock scenario. Also, try to hold locks on the common resources for as short time as possible to allow multiple sessions to be active at the same time.*

Answer:
We address this issue by encapsulating all database calls within a single transaction and applying locks to tables accessed or affected by the addPayment() function. Write locks are used for tables requiring write access, while read locks are applied to those needing only read access. Locks are acquired immediately before addPayment() and released immediately after. This approach minimizes lock contention while effectively safeguarding against race conditions and deadlocks.

Since we couldn´t make the theoretical case above work, we couldn´t test that this solution would solve 4c.

Secondary Index:
It would be useful to use a secondary index to improve the speed of finding passengers by name on the passenger table. Currently, finding a passenger requires a sequential scan of the entire table, which can be slow in the worst case (O(n) in the worst case). By creating a secondary index on the passenger_name field as it was explained in the lectures covering topic 8, we can store names in a sorted order along with pointers to the corresponding rows. This allows us to use binary search, which would speed up queries. Even when multiple passengers share the same name, the index can quickly narrow down the results, improving efficiency compared to a full table scan. This would help by reducing query time to O(log n) for lookups.