BrianAir Theoretical questions

Jakob Berggren, jakbe841
Max Wallhem, maxwa237

———————————————————————————————————

Question 8

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

The credit card information should be stored as hashed values in the database using some sort of strong encryption. Moreover, it is important to protect the database against SQL-injection. (This applies to the database as o whole. Not only regarding credit card information).

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 front- end of the system (in for example java-script on a webpage)?

1. Stored procedures offer higher performance due to the procedures only having to compile once. These can then be executed more quickly and efficiently compared to front-end functions, which increases response time.
2. Stored procedures also offer better scalability since everything is done on server.
3. Furthermore, stored procedures offer easier maintainability since everything is stored in one location, instead of being stored on different clients.

Question 9

Open two MySQL sessions. We call one of them A and the other one B. Write START TRANSACTION; in both terminals

a) In session A, add a new reservation.

```
-- SELECT "Step2, add a bunch of bookings to the flights" AS "Message";
-- CALL addReservation("MIT","HOB",2010,1,"Monday","09:00:00",3,@a);
```

b)
Is this reservation visible in session B? Why? Why not?

No, it is not visable since querys in Session A is done in its own transaction. This is not visible in Session B until it has been commited. Once commited it will be visable.

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. If we try to modify the reservation from A in B the query stalls. From B we know that we cannot see the tuple we want to modify, since this is done in its own transaction. This also implies that we can't modify it. It's only when transaction A has been committed, we can see and modify the tuple through Session B. This is due to the principle of isolation.

Question 10

Is your BrianAir implementation safe when handling multiple concurrent transactions? Let two customers try to simultaneously book more seats than what are available on a flight and see what happens. This is tested by executing the test scripts available on the course-page using two different MySQL sessions. Note that you should not use explicit transaction control unless this is your solution on 10c.

a)
Did overbooking occur when the scripts were executed? If so, why? If not, why not?
No, no overbooking occurred. One of the sessions was started slightly ahead and therefore one of the sessions succeeded and the other one got the message that there were not enough seats.

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.

Yes this could theoretically occur in addPayment if both sessions read the IF-statement that checks free seats
before the booking table is updated.

```
CODE:

IF (nr_of_passengers_in_reservation <= calculateFreeSeats(current_flight_nr))
THEN
INSERT INTO Credit_card(Card_number, Card_holder, Reservation_number)
VALUES (credit_card_number, cardholder_name, reservation_nr);

INSERT INTO Booking(Booking_number, Credit_card_number, Price)
VALUES (reservation_nr, credit_card_number,
calculatePrice(current_flight_nr));
```

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.

```
CODE:

IF (nr_of_passengers_in_reservation <= calculateFreeSeats(current_flight_nr))
THEN
INSERT INTO Credit_card(Card_number, Card_holder, Reservation_number)
VALUES (credit_card_number, cardholder_name, reservation_nr);

SELECT SLEEP(5);

INSERT INTO Booking(Booking_number, Credit_card_number, Price)
VALUES (reservation_nr, credit_card_number
calculatePrice(current_flight_nr));
```

By implementing a sleep as the code shows above, the theoretical case canoccur.

d)
Modify the test scripts 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.

```
CODE:

CALL addContact(@a,00000001,"saruman@magic.mail",080667989);
SELECT SLEEP(5);
SELECT "Making payment, supposed to work for one session and be denied for
the other" as "Message";

START TRANSACTION;
LOCK TABLES
Flight READ,
Reservation READ,
Credit_card WRITE,
Passenger_reservation READ,
Booking WRITE,
Ticket WRITE,
Route READ,
Weekly_schedule READ,
Day_of_week READ,
Year READ;

CALL addPayment (@a, "Sauron",7878787877);
```

The above code locks the necessary tables which addPayment either writes to or reads from. This allows the first session to complete before the second session can execute and therefore preventing overbooking.

Secondary Index

If you could have flight_number as a secondary index in the reservation table it would speed up the search for the correct flight.