DATABASES

ADIT

Lab Compendium – BryanAir Project

Institutionen för datavetenskap (IDA), Linköpings universitet

Table of Contents

Project; BrianAir Database

Objectives

The objectives of this lab are to learn how to implement a database, get some hands on experience of MySQL procedures and more experience of basic MySQL queries learnt in previous labs.

Background Reading

For this lab, you need to be familiar with EER-modelling, translation of EER into relational tables, MySQL queries, normalisation, stored procedures and transactions, that is, almost the whole course.

Estimated Project Time

This is a large project, reaching over six lab-occasions and one mandatory seminar. Note however that these lab-occasions are only expected to cover the MySQL-implementation-part of your project. The EER-modelling and translation into relational tables (i.e. the first part of the project) needs to be handled outside the scheduled lab-time. Check the homepage or ask your lab-assistant if you have further questions about the schedule, deadlines, etc.

Exercise Background

A friend of yours has decided to get into the travel business and has just started up a low price airline-company called BrianAir. He is now designing the java-script based booking web page and has realized that he needs a database in the back-end. He knows you are taking a database-course and has hence asked you to design and implement the flight and booking database. In other words you are supposed to plan and set up the database he will use for his company. You think the company is a terrible idea but as a good friend you have agreed to help him.

The main idea of the website is that a customer should be able to enter it, search for flights and create bookings on these flights. The booking procedure should be fairly simple where the customer creates a reservation for a number of people on a flight and get a reservation-number. The customer should then be able to add passengers to the reservation as well as some contact details. Finally the customer should be able to add payment details and pay for the reservation and thereby ensure that the passengers have seats on the chosen plane. When this happens the reservation is said to become a booking. All of this can be done within one session at the web page, but the customer should also be able to create a reservation, check the price, and finish the booking procedure at a later date using the given reservation number.

Part 1, Modelling the Database

1. (at home) Draw an EER-diagram for the BrianAir's database. Your friend has not really thought this through and will let you decide how to design the database as long as the following requirements and specifications are met. Note that you must read through the entire project (including questions 2-10) to get a proper understanding of what it should be able to handle and what attributes are required for the entities.

a. BrianAir uses only one type of airplane that takes 40 passengers. You need not to model different airplane types.

b. BrianAir only flies between the airports Lillby and Smallville (and returns) but they will soon expand. Hence the database should be constructed so that it is easy to add new destinations and routes later.

c. BrianAir operates on a strict weekly schedule. There are no exceptions for holidays. The weekly schedule is valid for one year and may be changed on every January 1st. Of course, there can be several flights per day.

d. Reservations should be possible to make on any route added to the system. A single reservation should only be for one route, i.e. one direct flight, but may contain several passengers. The reservation is confirmed by issuing a unique, unguessable, reservation number that is needed to finish the booking (i.e. paying) at a later date.

e. The flight pricing depends on

- the start and stop destination which (together) has a route price,
- the day of the week. BrianAir has the same weekday pricing factor for all flights regardless of destination, e.g. factor 4.7 on Fridays and Sundays, factor 1 on Tuesdays, etc.
- the number of already confirmed/booked passengers on the flight. The more passengers are booked the more expensive the flight becomes.
- what profit BrianAir wants to make on the flights. This factor is the same for all flights.

The total price is thus calculated as:

$$TotalPrice = Routeprice_{to,from} \cdot Weekdayfactor_{day} \cdot (\#BookedPassengers_{flight} + 1)/40 \cdot profitfactor$$

Pricing factors (including profitfactor) and route prices can change when the schedule changes, once per year!

f. The actual price for a booking is calculated at payment time and should be saved for future reference. Moreover, each person in one booking has the same seat price (calculated as the first person in the booking).

g. Information about each passenger participating in any BryanAir flight must be stored in the database as detailed by the flight safety standard. This information includes passport number as well as the passengers' full name.

h. Each booking should contain one passenger that is the *contact* of that booking and must supply phone number and e-mail address. Note that this contact has to be added before the booking is payed.

i. Only credit cards can be used to pay for the flights and the necessary credit card information, such as card number and credit card holder, should be stored. It may be that the one who pays for the flight (the credit card holder) does not participate in the flight.

j. The payment of the booking is confirmed by issuing a unique, unguessable, ticket number per passenger that the passenger needs to bring to the airport instead of a paper ticket.

k. Overbookings are not allowed but overreservations are. This means that there can exist more reserved seats on a plane than the actual number of seats, but that one only can reserve seats, and pay previously made reservations, if there exist enough unpaid seats on the flight. If there are not enough unpaid seats, the whole booking-process is aborted and the reservation should be removed from the system.

Other requirements are up to your assumptions. State them in the EER-diagram.

Some Hints and Further requirements:

A simple solution would be to model flights as an entity with attributes such as cities of departure and arrival, day of the year and time of departure, etc. However, this is not acceptable because this table may contain duplicated information. Hence we require you to include the following entities instead (note that you will need additional entities outside these):

- *Route*, which contains all the routes the company flies. A route is characterized by the cities of departure and arrival.
- *Weekly schedule*, which contains the one week schedule of flights, that is the same for all weeks, for a given year. A weekly flight is characterized by its id, a route, a year, a day of the week and the time of departure.
- *Flight*, which contains the instantiations of the weekly flight schedule for each week. A flight is characterized by its unique flightnumber, ~~a weekly flight~~ and a week.

Handing In

For the first hand-in of the project, before the tutor session, the following should be handed in:

- EER-diagram.
- Translation of the EER-diagram into a relational model (tables)
- For each table we also want you to identify **functional dependencies**, **candidate keys** and **primary key** as well as the **normal form** of the table. If the table is not in BCNF, motivate why you have not normalized it into this form.

Make sure that your EER diagram and relational model is approved before you start the implementation in part 2 of the project!

Part 2, Implementation of the Database

Once your EER-diagram and relational model are approved, it is time for the implementation. Your friend has no opinions of how this should be done as long as the database contains a set of stored procedures that works as an interface to the front-end.
Please **read through all the questions, hints, etc.**, **before starting with the implementation** to avoid unnecessary work.

Hint

For this lab it is strongly recommended that you write your queries in a script to allow you to easily recreate the database from scratch. This is simply done by writing all queries sequentially in a text-file and save it in your home directory. All queries in the text-file can then be run using the query "*SOURCE name_of_text_file.txt*". In the beginning of the text file you should drop all tables and procedures in the database before you "recreate" them (using the commands *"DROP TABLE IF EXISTS table_name" resp. "DROP PROCEDURE IF EXISTS proc_name").*

Testing and Expected Output

In the lab-section on the course-page there are a number of scripts that allow you to test your implementation to see that it gives the correct output given different input. These scripts also contain additional information about the input-output relation, such as error messages etc., that are not given in the questions. **Note that your implementation should give the correct output for all scripts before hand-in.** Also note that these scripts are by no means complete and it is not enough to only give the correct output without using the correct way of "finding" it.

Questions:

2. Create your tables and foreign keys in the database using the CREATE TABLE and if necessary the ALTER TABLE queries. Once you are done the database should have the same structure as shown in your relational model. Also, read up on how attributes can be automatically incremented and implement where appropriate.
For the database to properly work with the front-end the attributes should be of the following types:

Variable	**Type**
Year	INTEGER
Day	VARCHAR(10)
Airport code	VARCHAR(3)
Airport name	VARCHAR(30)
Country	VARCHAR(30)
Departure time	TIME
Profitfactor	DOUBLE
Routeprice	DOUBLE
Weekdayfactor	DOUBLE
Flightnumber	INTEGER
Reservation number	INTEGER
Name	VARCHAR(30)
Passport number	INTEGER
Email	VARCHAR(30)
Phone number	BIGINT
Creditcard number	BIGINT

3. Write **procedures** for filling the database with flights, etc. These procedures will work as an interface with the front-end.

a) Insert a year: Procedure call: *addYear(year, factor);*
b) Insert a day: Procedure call: *addDay(year, day, factor);*
c) Insert a destination: Procedure call: *addDestination(airport_code, name, country);*
d) Insert a route: Procedure call: *addRoute(departure_airport_code, arrival_airport_code, year, routeprice);*
e) Insert a weekly flight: Procedure call: *addFlight(departure_airport_code, arrival_airport_code, year, day, departure_time);* Note that this procedure should add information in both weeklyflights and flights (you can assume there are 52 weeks each year).

4. Write two help-**functions** that do some of the calculations necessary for the booking procedure:

a) Calculate the number of available seats for a certain flight: Function call: *calculateFreeSeats(flightnumber);* where the output is the number of free (unpaid) seats (i.e. an integer) on that certain flight.
b) Calculate the price of the next seat on a flight: Function call: *calculatePrice(flightnumber);* where the output is the price (i.e. a double) of the next seat calculated as shown in 1e.

5. Create a **trigger** that issues unique unguessable ticket-numbers (of type integer) for each passenger on a reservation once it is paid. An appropriate MySQL function to find unguessable numbers is *rand()*.

6. It is now time to write the stored procedures necessary for creating and handling a reservation from the front-end. In addition to the input and output detailed below, see the test-files for appropriate error-messages to return in case of unsuccessful payments etc.

a) Create a reservation on a specific flight. Procedure call: *addReservation(departure_airport_code, arrival_airport_code, year, week, day, time, number_of_passengers, output_reservation_nr);* where the number_of_passengers is the number of passengers the reservation is for (and only used to check that enough unpaid seats are available) and output_reservation_nr is an output-variable and should contain the assigned reservation number.
b) Add a passenger to a reservation: Procedure call to handle: a*ddPassenger(reservation_nr, passport_number, name);*
c) Add a contact: Procedure call to handle: *addContact(reservation_nr, passport_number, email, phone);* where the contact already must be added as a passenger to the reservation.
d) Add a payment: Procedure call to handle: *addPayment (reservation_nr, cardholder_name, credit_card_number);* This procedure should, if the reservation has a contact and there are enough unpaid seats on the plane, add payment information to the reservation and save the amount to be drawn from the credit card in the database. If the conditions above are not fulfilled the appropriate error message should be shown.

7. Create a view *allFlights* containing all flights in your database with the following information: *departure_city_name, destination_city_name, departure_time, departure_day, departure_week, departure_year, nr_of_free_seats, current_price_per_seat.* See the testcode for an example of how it can look like.

8. Answer the following theoretical questions:

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

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 web-page)?

In the next two questions you will see how MySQL handles concurrency. As default MySQL commits every single query as a transaction meaning that as soon as a query is executed it is also committed to the database. However, MySQL also supports multiple queries to be bundled up in a single transaction as described during the lecture.

To test this, open two terminals in the same computer, both connecting to the same database, then use the query ***START TRANSACTION*** to start a transaction that is finished by either **COMMIT** or **ROLLBACK**. Note that once the transaction is finished the database goes back to the default setting of committing every query unless a new transaction is started. The main queries to handle transactions are *START TRANSACTION, COMMIT, ROLLBACK, LOCK TABLES, UNLOCK TABLES, SAVEPOINT* and *SELECT…FOR UPDATE*. Please check the MySQL manual for further description and examples of the queries.

Play around with the transaction queries above in the two terminals and try to anticipate what will happen. One important difference between MySQL and what is described during the lecture is that MySQL has the transaction isolation level *Repeatable-Read* as default to maintain consistency of the database for the transactions. This means that MySQL implicitly adds a write lock on each tuple that is inserted, or updated, in any table. It also adds read locks on the tuples when they are used in for example IF-statements. For further motivation of this, how it works, and how it can be changed, check the MySQL manual regarding *transaction isolation level*.

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.

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

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.

10. Is your BryanAir 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 testscripts 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?

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.

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.

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.

Note that depending on how you have implemented the project it might be very hard to block the overbooking due to how transactions and locks are implemented in MySQL. If you have a good idea of how it should be solved but are stuck on getting the queries right, talk to your lab-assistant and he or she might help you get it right or allow you to hand in the exercise with pseudocode and a theoretical explanation.

Handing in

For the final hand-in of the project the following should be handed in:

- *Approved* EER-diagram and relational model. If minor changes have been made during the implementation these should be reflected here.
- Answers (including SQL-code) for questions 2 to 10. Note that the code should have been tested on the available test scripts and give the correct output before hand-in.
- Identify one case where a secondary index would be useful. Design the index, describe and motivate your design. (Do not implement this.)

Note that the final report should be handed in both to your lab-assistant and sent to the Urkund address as detailed on the course page. For the lab-assistant the hand-in is handled just like any other lab-report. For the Urkund address please put everything together into one big .pdf or .txt file and attach it to the email.

If the lab assistant does not understand your report, you will be contacted for a question and answer session and a possible demonstration of your project.