

Database Tutorial 6: Relational Algebra and Transactions

1. Consider the following schema:

```
Airports(code, city)
Flights(code, airlineName)
FlightLegs(from, to, departureTime, arrivalTime, code )
    departureAirport → Airports.code
    destinationAirport → Airports.code
    code → Flights.code
    (to, code) UNIQUE
```

Each flight-leg represents one take-off and landing, and a flight might have multiple flight-legs.

- a. Below is an SQL query that finds all airports that have departures or arrivals (or both) of flights operated by Lufthansa or SAS (or both). Express this query as a relational algebra expression.

```
SELECT DISTINCT served
FROM
((SELECT to AS served, airlineName
  FROM Flights JOIN FlightLegs ON FlightLegs.code = Flights.code)
UNION
 (SELECT from AS served, airlineName
  FROM Flights JOIN FlightLegs ON FlightLegs.code = Flights.code)) AS D
WHERE D.airlineName = 'Lufthansa' OR D.airlineName = 'SAS';
```

- b. Translate the following relational algebra expression to an SQL query:

```
 $\pi_{\text{First.departureTime, Second.arrivalTime}}$ 
(( $\rho_{\text{First}}(\text{FlightLegs})$ )  $\bowtie_{\text{First.to=Second.from}}$  ( $\rho_{\text{Second}}(\text{FlightLegs})$ ))
```

- c. Write a relational algebra expression for finding the code of all flights that have more than two flight-legs
- d. Write a relational algebra expression for finding the code of all flights that departs from Gothenburg and land in London (you do not have to list the flight-legs)

2. SQL Injection, Transactions. Consider an existing database with the following database definition in a PostgreSQL DBMS:

```
CREATE TABLE Users (  
    id INTEGER PRIMARY KEY,  
    name TEXT,  
    password TEXT  
);  
  
CREATE TABLE UserStatus (  
    id INTEGER PRIMARY KEY REFERENCES Users,  
    loggedin BOOLEAN NOT NULL  
);  
  
CREATE TABLE Logbook (  
    id INTEGER REFERENCES Users,  
    timestamp INTEGER,  
    name TEXT,  
    PRIMARY KEY (id, timestamp)  
);
```

- a. Users of a web application are allowed to query this database for a certain user id. This function implemented in JDBC using the following code fragment:

```
...  
String query=  
    "SELECT * FROM UserStatus WHERE id = " + userInput + "";  
PreparedStatement stmt = conn.prepareStatement(query);  
ResultSet rs = stmt.executeQuery();  
...
```

Does this code contain an SQL injection vulnerability? If it does not, why not? If it does, how would you correct the code?

- b. The JDBC code below is supposed to run a batch job in the following way: Loop through an array of users, set each users password hash to a certain value and add a logbook message for that user. If any of the users do not exist, the whole batch job should be rolled back.

The queries seem to work fine, and if there is an invalid id in the array the error message is printed, but all the other ids in the array are still changed! What goes wrong and how can it be fixed (Hint: two separate things need to be fixed, think about what happens both before and after the error).

```
int[] blockList = {11,42,55}; // Ids to be locked
PreparedStatement update = conn.prepareStatement(
    "UPDATE Users SET password='qiyh4XPJGsOZ2MEAy' WHERE id=?");
PreparedStatement insert = conn.prepareStatement(
    "INSERT INTO Logbook VALUES (?,now(),'Account locked')");
conn.setAutoCommit(false); // Enable transactions

for(int user : blockList){ // For each user in blocklist ...
    update.setInt(1,user);
    insert.setInt(1,user);
    int res = update.executeUpdate();
    if (res == 0){ // 0 rows affected - user does not exist!
        System.err.println("Error, missing id: "+user);
        conn.rollback();
    } else{
        insert.executeUpdate();
        conn.commit();
    }
}
```