Database Tutorial 6: Relational Algebra and Transactions

1. Consider the following schema:

```
Airports(code, city)
Flights(code, airlineName)
FlightLegs(from, to, departureTime, arrivalTime, code )
    from → Airports.code
    to → 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';
```

$$\delta(\pi_{served}(\sigma_{airlineName='Lufthansa' \text{ OR } airlineName='SAS'}(\rho_{D(served,\ airlineName)}($$
$$\pi_{from,\ airlineName}(Flights \bowtie_{FlightLegs.code=Flights.code} FlightLegs)$$
$$\cup$$
$$\pi_{from,\ airlineName}(Flights \bowtie_{FlightLegs.code=Flights.code} FlightLegs)$$
$$))))$$

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

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

```
SELECT First.departureTime, Second.arrivalTime
FROM FlightLegs AS First JOIN FlightLegs AS Second
ON First.to = Second.from;
```

c. Write a relational algebra expression for finding the code of all flights that have more than two flight-legs

$$\pi_{code}(\sigma_{n >= 2}(\gamma_{code, COUNT(*) \rightarrow n}(FlightLegs)))$$

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)

$$\text{Froms} = \pi_{\text{FlightLegs.code}}(\sigma_{\text{city = 'Gothenburg'}} (\text{FlightLegs} \bowtie_{\text{from=Airports.code}} \text{Airports}))$$
$$\text{Tos} = \pi_{\text{FlightLegs.code}} (\sigma_{\text{city = 'London'}} (\text{FlightLegs} \bowtie_{\text{to=Airports.code}} \text{Airports}))$$
$$R = \text{Froms} \cap \text{Tos}$$

2. Authorization, 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?

Yes this code contains an SQL injection vulnerability. The vulnerability can be removed by either correctly sanitizing or escaping the data in the *userinput* variable. A better solution is to use a PreparedStatement with placeholder:

```
…
String query = "SELECT * FROM UserStatus WHERE id = ? ";
PreparedStatement stmt = conn.preparedStatement(query);
Stmt.setString(1, userinput);
ResultSet rs = stmt.executeQuery();
…
```

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();
    }
}
```

The two problems in the code are:
1. A commit is performed at the end of each iteration, starting a new transaction. The commit needs to be moved outside the loop.
2. The rollback does not terminate the loop, so it keeps going in a new transaction. Triggering the error needs to terminate the loop using break/return/throw.

Correct code:

```
for(int user : blockList){
    update.setInt(1,user);
    insert.setInt(1,user);
    int res = update.executeUpdate();
    if (res == 0){
      System.err.println("error, missing id: "+user);
      conn.rollback();
      break;
    }
    insert.executeUpdate();
}
conn.commit();
```