DI, FCT-NOVA

Sistemas de Bases de Dados

3rd Test – 2021/22

Duration: 1 Hour and 30 Minutes

Group 1

Consider a (simplified) database for managing a flying-passenger locator system of a national health authority (something that is quite common since the COVID pandemic), with the following tables (where attributes that form the primary keys are underlined):

routes(NumR,IdC,Origin,Time,Destination)
persons(IdP,NameP,Address,Zip,Sex,Age,CertID)
flights(NumR,Date)
companies(IdC,NameC,Country,…)
travels(Code,IdP,Origin,Destination)
travelLegs(Code,NumR,Date,SeatNo)
vaccinationCert(CertID,IdP,DateV,Doses,Lab,issuedBy)

Each of these tables has a B+ tree clustered index over the primary key.

For each passenger, the database stores, in the *persons* table, her id, name, address (including zip code) sex, age, the id of her certificate of vaccination (with a **null** value if the person has no vaccination certificate). Each vaccination certificate has, besides a unique certificate identifier, the identifier of the person, the date of the (last) vaccination shot, number of doses taken, the name of the Lab that made the vaccine, and of the authority that issued the certificate.

The health authority uses this database to track which flights each person is taking, passing through which airports. Each flight has a route number and a date. Route numbers refer to routes that may take place everyday, or once a week, etc. E.g., Route number TP943 refers to daily TAP flights leaving at 1pm from Genève to Lisbon.

A travel is a sequence of flights that a passenger may take. Each one is identified by a reservation code, refers to a passenger, and has and origin and a final destination. Each step (or travel leg) in a travel, refers to a flight (identified by the route number and date), and the seat number reserved for the corresponding passenger in that flight.

Furthermore, at a given moment the *companies* table has 100 tuples, the *routes* table has 2.000 tuples, the *flights* table has 200.000 tuples, the *travels* table 400.000 tuple, *vaccinationCert* with 800.000 tuples, *persons* with 1.000.000 tuples, and *travelLegs* with 20.000.000 tuples. Tuples of all these tables are of variable size and, on average, each tuple of *flights* has 25 bytes, each tuple of either *routes*, *companies*, *travels* or *travelLegs* has 50 bytes, each tuple of *vaccinationCert* has 100 bytes, and each tuple of *persons* has 200 bytes.

The database is implemented in a system using 4KB blocks and a memory of 1,000 blocks.

Note: In this group, whenever an example is asked for, the example **must** be in terms of the database above, and **all examples must be given in SQL**. Moreover, **all** your answers must include a brief **justification**.

1 a) Assume that the database management system uses a rigorous *2-phase lock protocol* for concurrency control. Give an example of a schedule (**in SQL**) that would cause a deadlock.

1 b) Consider the following concurrent transactions (without assuming, for now, any particular order between the operations between the transaction):

1. begin transaction
2. update *persons*
set *age* = *age* * 2
where *IdP*=12345678**;**
3. update *persons*
set *age* = *age* * 3
where *IdP* =12345678
4. commit;

A. begin transaction
B. update *persons*
set *age* = *age* + 1
where *IdP* =12345678;
C. commit;

Assume that before the transactions' execution the *age* of the person with *IdP* =12345678 is 1, and that no other transaction is executing at the same time.

What are the possible *age*s of that person after the execution of both transactions if:
(1) both transactions execute in **serializable** mode?
(2) both transactions execute in **read committed** mode?

For each possible final result, say which schedule generates that value of the *age*.

[You can write a schedule as a sequence of numbers and letters identifying the order in which the operations above are executed, e.g. 1, 2, A, B, 3, 4, C]

1 c) Some database systems, such as Oracle, use multi-version concurrency control protocols, where lock granularity is done at tuple level, and with *"snapshot isolation"*. Although these protocols guarantee some weak form of isolation, they do not guarantee that the concurrent execution is always equivalent to one of the sequential executions of the concurrent transactions.

Present a schedule of **SQL** actions over two concurrent transactions which illustrates just that. I.e., a schedule that, if executed e.g., in Oracle, both transactions would successfully end but with a different result from the one obtained by first completely executing one transaction and then the other (or vice-versa).

1 d) Present **one** schedule of two concurrent transactions (in **SQL**) that, if the DBMS uses a 2-phase lock protocol then a deadlock would occur, but if the DBMS uses a protocol based on timestamps, then both transactions would run successfully until the end (i.e., none of them would have to rollback) without the need for suspending any action, let alone a deadlock.

1 e) Suppose now that the database is fragmented, with a central server with the complete tables of *persons*, *vaccinationCert* and *travels* (i.e., all the tables with information that does not depend on companies), plus the table *companies*, and then there is a server in each company with the routes, flights, and travelLegs of that company.

What is the best execution plan for the following query (what are the names of the persons who flew at least once from Lisbon to Madrid with TAP), that is asked in the central server, assuming that the number of persons that flew from Lisbon to Madrid with TAP is much less than the total number of persons (as it is obviously the case, in practice)?

select *persons.NameP*
from *persons* **natural inner join** *travels* **natural inner join** *travelLegs*
inner join *routes* **using (***NumR***)**
where *routes.Origin = 'Lisbon'* **and** *routes.Destination = 'Madrid'* **and** *idC = 'TAP'*

Group 2

2 a) *In ACID transactions, the Consistency property only imposes the verification of integrity constraints at the end of the transaction, rather than imposing the verification after each action inside the transaction.*

Why isn't it important to impose the integrity of the database after each action of a transaction? And which operations would be impossible if integrity would be imposed after each action? Give an example of a database, and an operation over it, that would be impossible if integrity was imposed after each action.

2 b) *For dealing with recovery and thus guarantying the atomicity and durability ACID properties, a database management system may either implement a deferred or an immediate modification schema.*

Briefly explain the differences between these two schemas, mentioning also how the logs in each of the cases differ.

2 c) Which problem in distributed databases does the *2-phase commit protocol* address? In your answer mention which of the ACID properties are relevant to that problem.