DI, FCT-NOVA **June 29, 2020**

Sistemas de Bases de Dados

Exam, 2019/20

Duration: 3 Hours

Group 1

Consider a database for managing an App for tracking proximity contacts of COVID-19 infected patients. This database includes the following tables (where the attributes that make the primary key are underlined in each table):

municipalities(codMun,name)
persons(idP, nameP,sex,age,codMun)
phones(numberPh,model,idP)
tests(idP,dateT,result)
tokens(tokenStr,numberPh,date)
contacts(tokenStr1,tokenStr2,when)

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

For each person, the database stores her id, name, sex age and the code of the municipality in which she lives. The names of the municipalities are stored in a separate table. Each person can make more than one test, but never more than one in a single day. Each test has a result which can be positive ('+') or negative ('-'). Persons have phones and each phone has a unique number (stored in the *phones* table). From time to time phones are given unique tokens (stored in the *tokens* table). When two phones are close together, they communicate their most recent tokens, and a tuple is stored in the *contacts* table with the most recent tokens of both phones and the date and time (in *when*) of when that occurred.

The database management system (DBMS) uses 4KB blocks and, for the exercises below, consider that the memory can have up to 100 blocks. Tuples of all these tables are of variable size and, on average, a tuple of the municipalities' table has 50 bytes, and a tuple of each of the other tables has 100 bytes.

Furthermore, at a given moment the *municipalities* table has 308 tuples, the *persons* table has 10.000.000 tuples, the *phones* table has 5.000.000 tuples, the *tests* 1.000.000 tuples, the *tokens* 20.000.000 tuples, and *contacts* table has 1.000.000.000 (one billion) tuples. Note that, as such, the contacts table has approximately 100GB.

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

1 a) Show two execution plans for the following query (returning the phone numbers of persons who contacted with other persons who tested positive) and justify which one has the lowest cost.

```
select distinct C.numberPh
from tests natural inner join phones natural inner join tokens T
     contacts, tokens C
where result = '+' and (T.tokenStr = tokenStr1 or T.tokenStr = tokenStr2)
                   and (C.tokenStr = tokenStr1 or C.tokenStr = tokenStr2)
```

1 b) By using this database over time, most of the tuples in the *contacts* table become irrelevant for warning those who have contacted persons with positive tests. In fact, for that purpose the only relevant contacts are the ones made in the 2 weeks before the positive test.
Most DBMS have a mechanism of organisation of data that may significantly improve the efficiency in these situations. What is that mechanism, and how could it be used in this specific case (i.e. for getting the phones of those who, in the last 2 weeks, contacted persons who tested positive)?

1 c) The *tests* table has a primary key composed by the concatenation of 2 attributes: *idP* and *dateT*. As such, the clustering index can be created taking into account the concatenation *(idP,dateT)* or the concatenation *(dateT,idP)*. Which one would you prefer for this particular database (given its expected usage), and why?

1 d) Assume that the DBMS only has the "nested loop join" algorithm for joins, eventually using index files when available. For each of the join queries below, which is the best join order?

municipalities ⋈ ***persons*** ***tests*** ⋈***persons***

1 e) Provide an example of a join of two of the above tables for which the merge-join algorithm is the most adequate.

1 f) Present a schedule with 2 concurrent transactions over this database that would cause a deadlock.

1 g) Consider the following concurrent transactions:

```
begin transaction                    begin transaction
select * from persons;               select * from persons;
insert into persons values (1, ...); insert into persons values (2, ...);
select * from persons;               select * from persons;
commit;                              commit;
```

Present a schedule of the operations in the transactions for which the result differs whether the transactions are both executed in Read uncommited, Read commited or Serializable isolation mode, stating what the result would be for which of the 3 modes (to simplify the presentation of results, assume that initially the table *persons* is empty).

1 h) To save some space in the central server, and as a step towards some privacy concerns, the *contacts* table has been horizontally fragmented so that the contacts made by each phone are stored directly on the phone.
Assuming that phones can execute database queries (i.e. that the phones together with the central serve form a distributed database), explain how the query below (returning names and ages of persons living in Almada who were close to a phone with token "aaaaa") could be efficiently executed:

```
select distinct nameP, age
from municipalities natural inner join persons natural inner join phones
                    natural inner join tokens, contacts
where name = 'Almada' and tokenStr = tokenStr1 and tokenStr2 = 'aaaaaa'
```

Group 2

Note: The answer to each of the questions below **cannot** exceed one page.

2 a) *Indices speed query processing, but it is usually a bad idea to create indices on every attribute, and every combination of attributes, that are potential search keys.*
Explain why.

2 b) *Foreign keys impose restrictions on the contents of tables in order to guarantee some consistency in the data. But this is not the only advantage of declaring foreign key – the declaration of foreign keys can, in some cases, be used by the database management system to improve the efficiency of queries.*
Explain how the declaration of foreign keys can be used to improve efficiency, and for which types of queries.

2 c) *Lock-based concurrency control protocols are usually called pessimistic while timestamp-based protocols are called optimistic.*
In which sense is the former pessimistic and the latter optimistic? In your explanation provide an example showing the pessimistic and optimistic behaviour of these protocols.