DI, FCT-NOVA

Sistemas de Bases de Dados

2nd 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. Moreover, **all** your answers must include a brief **justification**.

1 a) Present two execution plans for the following query (that returns all persons over 70 that travelled from Lisbon to Madrid on May 10), and justify which one should have the lowest cost:

select distinct *nameP*
from *persons* **natural inner join** *travels* **natural inner join** *travelLegs*
where *age > 70* **and** *Origin = 'Lisbon'* **and** *Destination = 'Madrid'* **and** *Date = 10/05/2022*

1 b) Assume that the DBMS only has the "block 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?

1. $travels \bowtie travelLegs$
2. $routes \bowtie companies \bowtie travelLegs$

1 c) In general the merge-join algorithm does not allow the use of pipelining between the join and other operations. However, there are situation in which that is not the case.

Give an example of an SQL query and a corresponding execution plan for which the merge-join algorithm could be used with pipelining without any problems.

1 d) Give an example of an SQL query whose execution would be more efficient if there were foreign keys in the database above, also specifying which foreign keys would have to be defined for that example.

1 e) Consider now that the system implements all the algorithms that we've studied during the course, and that there is a foreign key declaration on attribute *IdP* in *travels* referencing *IdP* in *persons*.

What is the best execution plan for the following query (that returns the names of persons over 70 that flew on May 10 on a same route in which the passenger with id 1111 also flew)?

with *flightP1111* **as**
(select *NumR, Date* **from** *travelLegs* **natural inner join** *travels* **natural inner join** *persons*
where *IdP = 1111***)**

select *nameP*
from *travelLegs* **natural inner join** *travels* **natural inner join** *persons*
where *person.Age* > 70 **and**
exists (select * from *flightP1111* **where** *travelLegs.Date = 10/05/2022* **and**
travelLegs.Date = flightP1111.Date **and**
travelLegs.NumR = flightP1111.NumR **)**

Group 2

2 a) *"The external sort merge algorithm makes a first pass on the whole table, where individual runs are independently sorted, followed by one **or more** passes where the several runs are merged. However, in practice, it is extremely rare that more than one merge pass is needed. As such, some DBMSs don't even include the implementation for several merge passes."*

Justify that in practice these situations are indeed quite unlikely (e.g. showing, in concrete databases and computers, from which sizes more than one pass is required).

2 b) *Although in general the efficiency of query execution is greatly improved by cost-based optimisation (where the cost of all possible execution plans is estimated before one is chosen for execution) this is not always the case. There are situations, and specific queries, for which cost-based optimisation is unaffordable, and for which its computational cost largely exceeds its potential benefits for the execution of the query.*

Explain in which situations cost-based optimisation is not desirable, giving a concrete example of a database and query where this is the case.

2 c) Give an example of a query-processing algorithm that can take a big advantage of parallelism. Briefly explain the algorithm, focusing on the characteristics that make it especially fit for intra-operation parallelism.