Solution and marking notes, INF3100-V19 exam

Evgenij Thorstensen

1, Intro

Attached to each question are marking notes to help consistent marking.

2, Normal forms and decompositions

In this section, we will work with the following relation, which stores information about products and the parts needed to make them:

Requirement(ProdID, PartID, Quantity, ProdName, AssemblyTime, Category)

Some analysis reveals the following functional dependencies (FDs):

ProdID -> ProdName
ProdID -> AssemblyTime
ProdID -> Category
ProdID, PartID -> Quantity

2.1, Candidate keys, 3p

Question: Find the candidate keys of this relation. Explain how you found them.

Solution: Starting with the attributes that do not occur on any right-hand side (they must be part of every candidate key), we take their closure. The attributes are ProdID and PartID. Their closure is the entire set of attributes, and therefore { ProdID, PartID } is the only candidate key.

Marking notes: No specific guidance.

2.2, Normal form, 7p

Question: What normal form does this relation satisfy? Explain your answer.

Solution: We need to check the FDs for NF violations. Starting with ProdID -> ProdName, it violates BCNF, as ProdID is not a candidate key. It also violates 3NF, since ProdName is not a prime attribute (not part of any candidate key). It also violates 2NF, since ProdID is part of a candidate key. We are left with 1NF, which is the lowest we consider. We therefore do not need to check any other FDs.

Marking notes: Students who do more work than needed by starting with a different FD should not lose points if the answer is otherwise correct. 2 point loss formissing explanation of why we do not need to check all FDs if we reach 1NF. Up to 5 point loss for inadequate explanation in general.

2.3, Decomposition, 11p

Question: Decompose this relation into BCNF. Explain your answer, and determine whether your decomposition is FD-preserving.

Solution: Decompositions are not unique. However, as long as closure of attributes is used, most if not all end up FD-preserving. One decomposition below. To make life easier, we can group up the right-hand sides of FDs with the same left-hand side:

ProdID -> ProdName, AssemblyTime, Category
ProdID, PartID -> Quantity

Take the top FD, it violates BCNF since ProdID is not a superkey. The closure of { ProdID } is { ProdName, AssemblyTime, Category }. The remaining attributes are { PartID, Quantity }. We thus get the new relations

A(ProdID, ProdName, AssemblyTime, Category)
B(ProdID, PartID, Quantity)

We now need to check the new relations for BCNF violations. A is clearly in BCNF, since everything directly depends on ProdID and nothing else.

B is also in BCNF, since ProdID and PartID do not occur in any right-hand side, and Quantity directly depends on them.

This decomposition is FD-preserving, since all FDs apply to only one relation, and can be checked on their relation.

Marking notes: Quality of explanation is important. Up to 2 points loss for missing FD-preserving explanation.

3, SQL and relational algebra

We have a product database with the following tables. Primary keys are underlined.

Requirements(<u>ProductID, DependencyID</u>, Quantity)
Product(<u>ProductID</u>, Name, AssemblyTime, Category)

Order(OrderID, ProductID, Quantity, PricePerUnit)

The Product table has IDs (int), Names (string), and Categories (int) of a product, as well as AssemblyTime in minutes (int) of a product from the components required. This value does not include the time required to assemble those components (see recursive SQL question).

The Order table tracks orders, with OrderID (int), ProductID (int, references Product), Quantity (int) ordered, and the price per unit (int).

3.1, SQL, join, 5p

Question: Find the name and id of all products in category 3 that have an order asking for at least 20 units and an order (same or different) where the cost per unit is at least 200.

Solution: Many variations possible, here is one

```
SELECT ProductID, Name FROM Product JOIN Order ON
    Product.ProductID = Order.ProductID
WHERE Category = 3 AND Quantity > 20
INTERSECT
SELECT ProductID, Name FROM Product JOIN Order ON
    Product.ProductID = Order.ProductID
WHERE Category = 3 AND CostPerUnit > 200;
```

Marking notes: The fact that it can be the same order or a different one is significant. Up to 3 points loss for mistakes on this point.

3.2, SQL, aggregates, 6p

Question: Write a query to find all orders where it would take more than 24 hours to assemble all products ordered given the direct required products (so no recursion) sorted by the total price of the order descending. Output the OrderID, total price, and the time to assemble.

```
SELECT Order.OrderID, SUM(Product.AssemblyTime)
   as totTime,
SUM(PricePerUnit*Quantity) as totPrice FROM Order
    JOIN Product ON Order.ProductID = Product.
   ProductID
GROUP BY OrderID
HAVING totTime > 24*60
ORDER BY totPrice DESC;
```

Marking notes: This questions tests aggregation, so only having the join correct gets 1 to 0 points. Missing HAVING or using WHERE instead loses 2 points. Missing order by loses 1. Correct solutions involving views dealing with the total price and total time separately are perfectly fine.

3.3, Recursive SQL, 13p

New for this question: The Requirements table stores, for each product, the products required to build it (its dependencies), and the Quantity (int) of each needed. Both ProductID and DependencyID are foreign keys referencing the Product table. Dependencies are recursive — product A may need 5 units of product B, and B in turn needs 7 units of C and 3 of D. *Important:* Note that Requirements is a many-to-many relation; several products may have e.g. product A as a dependency, and A itself could have many different direct dependencies.

Question: Write a recursive query to find all products with total assembly time (including assembly of dependencies and their dependencies and so on) above 24 hours. Output the time and product id.

Solution: There are two approaches that I can see here. First, a bottom-up version where we start with products that do not have any dependencies.

```
  --ProdTime will be a view that contains total
     time for each product (eventually)
  --Start with all products that have no
     dependencies.
  WITH RECURSIVE ProdTime(pid, totMins) AS (
  SELECT Product.ProductID, Product.AssemblyTime
     FROM Requirements JOIN Product ON Requirements
     .ProductID = Product.ProductID
  WHERE Product.ProductID NOT IN (SELECT
     DependencyID FROM Requirements)
  UNION
  --We know the total time of everything in
     ProdTime
  --Need to now add all products that have any of
     these as dependencies, and calculate their
     totals.
  --Their totals are sum (total for each dependency
      times quantity) + assemblytime for the
     product
  SELECT Requirements.ProductID,
  SUM(Requirements.Quantity*ProdTime.totMins)+
     Product.AssemblyTime FROM
  ProdTime JOIN Requirements ON Requirements.
     DependencyID = ProdDep.pid
  JOIN Product ON Requirements.ProductID = Product.
     ProductID
  GROUP BY Requirements.ProductID
  )
--Now we can select all the ones we need.
SELECT pid, totMins FROM ProdTime WHERE totMins >
   24*60;
```

The other option is to go top-down. We start with the Requirements table, then add dependencies and their dependencies. However, due to the fact that this relationship is many-to-many, we need to keep track of the end product for each dependency we add, as different products can have the same dependency. This requires careful joins.

```
WITH RECURSIVE Dependencies(ProdID, DepID, TotAtime, EndProd) AS (
  --Start by adding all products as their own dependency, to include their
  --assembly time
  SELECT ProductID, ProductID, AssemblyTime, ProductID FROM Product
UNION
  --Now we recursively add dependencies.
  --If P1 has P3 as a dependency, we add the line
  --P1 | P3 | quantity*assembly time for P3 | EndProd
  SELECT Dependencies.DepID, Requirements.DependencyID,
  (Requirements.Quantity*Product.AssemblyTime), Dependencies.EndProd
  FROM Dependencies JOIN Requirements ON Dependencies.DepID = Requirements.ProductID
  JOIN Product ON Product.ProductID = Requirements.DependencyID)
--To get total assembly time for end products, we aggregate on this column
SELECT EndProd, SUM(TotATime) as TotTime FROM Dependencies GROUP BY EndProd HAVING TotTime > 24*60;
```

Marking notes: Simple but incorrect top-down solutions that double-count due to the many-to-many problem are worth up to 8 points if they are otherwise correct and demonstrate recursive SQL skills. Smaller errors in WHERE/arithmetic to find totals should not cost more than 4 points.

Marking notes: Simple but incorrect top-down solutions that double-count due to the many-to-many problem are worth up to 8 points if they are otherwise correct and demonstrate recursive SQL skills. Smaller errors in WHERE/arithmetic to find totals should not cost more than 4 points.

3.4, Relational algebra, 6p

Question: Write an SQL query equivalent to the following relational algebra expression:

$$\gamma_{\{R.id,SUM(S.x)\rightarrow sumx\}}(\sigma_{R.y<300}(R \bowtie_{R.id=S.id} S))$$

Solution: This is a simple aggregation, a direct translation suffices.

```
SELECT R.id, SUM(S.x) AS sumx FROM R JOIN S ON R.
    id = S.id
WHERE R.y < 300 GROUP BY R.id;
```

Marking notes: Mistakes related to the aggregation (γ) operator up to 2 points loss, same for join and filter condition.

1 Transactions

4.1, Serializability, 3p

Question: What does it mean for a transaction schedule to be serializable?

Solution: A serializable transaction schedule is one that is equivalent to (produces the same result as) a serial schedule, i.e. one where transactions are executed one at a time.

Marking notes: No specific notes.

4.2, Conflict serializability, 5p

Question: Consider the following transaction schedule:

$$r_1(A); r_3(B); w_3(B); r_2(A); w_2(A); w_1(A);$$

Is this schedule conflict serializable? Explain your answer.

Solution: The standard approach is to draw a precendence graph. The precendences here are $T_1 \to T_2$ (T_1 reads A before T_2 writes A) and also $T_2 \to T_1$ (T_2 writes A before T_1 writes A). Therefore, the precedence graph has a cycle, and hence this schedule is not conflict serializable.

Marking notes: A good explanation matters here. The graph itself is not required, but the notion of precedence and a cycle in the precedences is. Precedences that are not in the schedule but in the answer should lead to 1-2 points loss.

4.3, 2PL protocol, 6p

Question: Consider the following transaction schedule:

$$r_1(A); r_1(B); r_2(B); w_2(A); w_1(C); c_1; c_2$$

Explain what will happen when this schedule is executed under the two-phase locking (2PL) protocol with shared (S), exclusive (X), and upgrade (U) locks. Which locks are requested and taken when, who has to wait, and so on. The lock matrix is below (the lock martix in question is the pessimistic one, where holding an upgrade lock disallows others to take shared locks).

Solution: T_1 takes shared locks on A and B, and reads them. T_2 requests a shared lock on B; this is granted per lock matrix. Now T_2 requests an exclusive lock on A; this is denied due to T_1 having a shared lock, and T_2 waits. T_1 takes an exclusive lock on C, gets it, and commits. Now T_2 gets the exclusive lock on A, writes, and commits.

Marking notes: The key is the wait. Note that T_1 cannot have released the shared lock on B prior to T_2 acting, since T_1 must first take the lock on C. Mistakes around waiting, lose up to 3 points. Too early lock release (violating the two phase rule), lose up to 3 points.

4.4, 2PL, explanation, 5p

Question: Explain, in your own words, why the two phase locking protocol (2PL) with shared (S) and exclusive (X) locks guarantees serializability.

Solution: The first transaction to write an element gets an exclusive lock on that element, and every other transaction must wait to both read and write that element until this first transaction is done. The resulting outcome is therefore that this transaction goes first, then the rest do their actions. The only concurrent actions that can happen are thus reads, which do not affect the result.

Marking notes: A full proof is not necessary. I consider the explanation above acceptable for full marks.

4.5, UNDO/REDO logging, 5p

Question: Explain how UNDO/REDO logging and recovery works and what advantages and disadvantages it has compared with REDO logging. You do not need to describe how checkpointing works.

Solution: In UNDO/REDO logging, both old and new values of an update are logged, in contrast to REDO, where only the new value is logged. Due to only knowing the new value with REDO logging, no data may be written to tables on disk until a transaction commits or aborts; however, disk writes may be postponed indefinately. In contrast, with U/R logging, data may be written to disk as soon as they are logged, or postponed indefinately. The disadvantage of U/R logging is that twice as much needs to be logged.

Marking notes: No specific guidance.

4.6, Deadlocks, 6p

Question: In protocols that use locking, such as 2PL, it is possible for a deadlock to occur. Define the concept, and describe a way of detecting or preventing deadlock (there are several).

Solution: A deadlock occurs when mutliple transactions (for example two) each have a lock the other wants. In this situation, neither can proceed. A way of detecting deadlocks is by maintaining a waits-for graph, with transactions as nodes and edges showing who waits for whose lock. A cycle in this graph indicates deadlock, and it can be broken by rolling back a transaction.

A different way of preventing deadlock is by using a protocol such as wait-die or wound-wait, where transactions may only wait on each other in sequence. A transaction that violates the sequence has to rollback.

Marking notes: Only one way of prevention or detection is enough for full marks. 2 points for describing deadlock correctly, remaining 4 for a way of dealing with them.

2 Query plans and optimization

5.1, Bitmap index scan, 3p

Question: One of the disk access methods that we have seen is a bitmap index scan. Explain what it is and how it works.

Solution: A bitmap index scan uses an index to find blocks containing the matching tuples. Instead of reading the blocks directly, a bitmap noting which blocks are to be read is created. This allows for a semi-sequential scan of the table, and can also be used to evaluate AND and OR conditions if both referenced columns have an index.

Marking notes: 1 point loss if the AND/OR point is not mentioned.

5.2, Estimation, 5p

Question: Using T(R) for the number of tuples in a relation, and V(R.x) for the number of distinct values in an attribute, write down an estimate for the number of tuples returned by the query

$$\sigma_{R.id=35}(R \bowtie_{R.x=S.y} S)$$

Solution: The estimate for the join is $\frac{T(R)\times T(S)}{\max(V(R.x),V(S.y))}$, and the selectivity factor for constant selection is $\frac{1}{V(R.id)}$. The whole thing is then

$$\frac{T(R)\times T(S)}{\max(V(R.x),V(S.y))}\times\frac{1}{V(R.id)}$$

Marking notes: It is perfectly ok to write "assuming $V(R.x) > V(S.y)$" or similar instead of max. Not including this fact should lose a point.

3 Distributed systems

6.1, CAP theorem, 8p

Question: Explain the CAP theorem in your own words, and what it means for a distributed system to choose CP or AP in the context of all three being impossible.

Solution: The CAP theorem states that a distributed system cannot guarantee all of consistency, availability, and partition tolerance at the same time. Consistency means that nodes do not end up with conflicting information, availability that a node that is up can respond to queries, and partition tolerance that the whole system does not go down due to some nodes being unable to communicate.

A system choosing CP guarantees consistency at the cost of availability — nodes may not accept operations during a partition ("try again later"). A system choosing AP, on the other hand, continues to accept queries even during a partition, but different nodes may end up with different information.

Marking notes: Stating CAP without explaining what the three properties are (loosely is ok) loses up to 3 points. Lack of CP and AP discussion loses up to another 3.