Introduction to databases: Individual assignment

Jens Kristian R. Nielsen
201303862

16. marts 2019

1 Exercise I

We wish to make a relation $\mathcal{R}$ that describes a Person with a primary key Name, and two 1-1 relation describing a persons parents and also allowing each person to have a number of children. Consider

```
CREATE TABLE Person(
name VARCHAR(32) PRIMARY KEY,
mother VARCHAR(32),
father VARCHAR(32),
FOREIGN KEY(mother)
    REFERENCES Person (name),
FOREIGN KEY(father)
    REFERENCES Person (name)
);
```

We cannot always satisfy the (1,1) relationship between Person and Mother, since this would mean that we have to add a Persons family tree all the way back to the beginning, i.e. Adam and Eve. This could be fixed by adding a table of Parents who don't have parent values themselves, but the exercise asked for as few tables as possible. We use foreign key constraints to allow each person to have a mother and a father, we could add a gender and a trigger that checks if the parent is of the proper gender for the role, of course this might upset the LGBTQ+ community, so perhaps we should not. This table does not restrict how many persons a person will be set as a parent to, so the desired 0-n relation from parent to children is satisfied.

2 Exercise II

Consider the schema:

Object(object, description)

Property(object, property, value)

Relationship(object1, relationship, object2)

Question 1.
We wish to find descriptions of objects in relationship with Cynthia.

```
SELECT o.description
FROM object o, relationship r
WHERE r.object1 = o.object AND r.object2 = "Cynthia"
```

Recall that objects o in relationship with Cynthia are those for which there exists a relation r such that r.object1 = 0 and r.object2 = "Cynthia"

Question 2.
Find all pairs of different objects, such that both objects in a pair have some relationship with object 'Cynthia' (remember 'Cynthia' is object2 in a relationship); print the different pairs. Do not print symmetric answers such as ('Tom', 'Sam') and ('Sam', 'Tom')

```
SELECT o1.object, o2.object
FROM object o1, object o2, relationship r1, relationship r2
WHERE r1.object1 = o1.object AND r1.object2 = "Cynthia"
      AND r2.object1 = o2.object AND r2.object2 = "Cynthia"
      AND o1.object < o2.object
```

We find the objects that are in relationship with Cynthia as described in Question 1. Then we print them side by side, and demand that the first one in a 2-uple has object value strictly smaller than the second. The strictness ensures that we avoid duplicates, and the inequality ensures that we select ('Sam', 'Tom'), but not ('Tom', 'Sam') since Sam < Tom, sorry Sam.

Question 3.
Print a list of different descriptions that contain the keyword 'electronic' and the number of objects having each such description. Limit your result only to those descriptions that are shared by more than 10 objects.

```
SELECT o.description,count(description)
  FROM object o
    where o.description LIKE "%electronic%"
      GROUP BY description
      Having count(description)>10;
```

We find pairs of descriptions including the keyword electronic that at least 10 objects share and the exact number of objects that share these descriptions. This by using count and group by, to get those that are equal, LIKE to exclude those that do not contain the keyword, and having count>10 to filter out small occurrences.

Question 4. find those objects that are in relationship with all other objects

```
SELECT o.object
FROM Object o
WHERE NOT EXISTS(SELECT *
                 FROM object p
                 WHERE NOT EXISTS(SELECT *
                                  FROM relationship r
                                  WHERE r.object1 = o.object
                                  AND r.object2 = p.object));
```

We find those objects o that no object p does not have a relationship from o to p.
Question 5. Express the meaning of the following query in plain but very precise English.

```
SELECT O.object
FROM object O
WHERE O.object NOT IN (SELECT R.object2 FROM relationship R)
```

We print the object value of those objects that no object stands in relation to. **Question 6.** Which of the following queries returns the titles of the most expensive books?

```
a) SELECT B1.title
   FROM book B1
   WHERE NOT EXISTS (SELECT B2.title
                     FROM book B2
                     WHERE B1.price > B2.price)
b) SELECT B1.title
   FROM book B1
   WHERE NOT EXISTS (SELECT *
                     FROM book B2
                     WHERE B1.price < B2.price)
c) (SELECT B.title
    FROM book B)
    EXCEPT(SELECT B1.title
           FROM book B1, book B2
           WHERE B1.price >= B2.price)
d) All of the above
e) None of the above
```

a) We take the books where there exists no book that is strictly cheaper, each of these is the cheapest book or a book that is tied for cheapest book.
b) We take the books where there exists no book that is strictly more expensive, each of these is the most expensive book o or one that is tied for most expensive book.

c) let b be a book in our database, then in the query of the EXCEPT statement we get a tuple B1=b=B2, and clearly B1.price = b.price >= b.price = B2.price, and as such b is in the query. This proves that

```
SELECT B1.title
FROM book B1, book B2
WHERE B1.price >= B2.price
```

Returns the titles of ALL books from our database. Now when we except the titles of ALL our books from the outer query we get nothing. Which is not the title of the most expensive book, unless our database is empty.
d) Cannot be true since a) is false.
e) Cannot be true since b) is true.

3 Exercise III

Consider the relational scheme R with the set of functional dependencies F.

$$R = \{A, B, C, D\}$$
$$F = \{\{A\} \to \{C\}, \{B, C\} \to \{D\}, \{A, B\} \to \{C\}\}$$

Question 1. We wish to determine candidate keys, we use algorithm 15.2(a)
We remove one attribute at a time and compute the closure with regard to F.

$$\{A, B, C\}+ = \{A, B, C, D\} \qquad \{B, C, D\}+ = \{B, C, D\}$$
$$\{A, B, D\}+ = \{A, B, C, D\} \qquad \{A, C, D\}+ = \{A, C, D\}$$

We continue the algorithm in the two relevant cases.
Case 1: ABC

$$\{A, B\}+ = \{A, B, C, D\} \qquad \{B, C\}+ = \{B, C, D\} \qquad \{A, C\}+ = \{A, C\}$$

Case 2: ABD

$$\{A, B\}+ = \{A, B, C, D\} \qquad \{B, D\}+ = \{B, D\} \qquad \{A, D\}+ = \{A, C, D\}$$

We continue the algorithm in the one relevant case.
Case 3: AB

$$\{A\}+ = \{A, C\} \qquad \{B\}+ = \{B\}$$

In conclusion we get super keys: $\{A, B, C, D\}, \{A, B, C\}, \{A, B, D\}, \{A, B\}$ where $\{A, B\}$ is our sole candidate key, since it is "irreducible" as shown in case 3.
To obtain $\{A, B\}+ = \{A, B, C, D\}$ we used the functional dependency $\{A\} \to \{C\}$ that grants access to $\{B, C\} \to \{D\}$.
Notice that no FD generates A or B, so these must be included in a super key.

Question 2: We wish to compute a minimal cover of F.
Step 1 is to split out right hand sides, but these are already atomic so no work is needed.
Step 2 we see that $\{A, B\} \to \{C\}$ can be reduced to $\{A\} \to \{C\}$, but this FD is already in F so we simply remove $\{A, B\} \to \{C\}$ and get $F' = \{\{A\} \to \{C\}, \{B, C\} \to \{D\}\}$.

Step 3 we wish to see if we can remove one of the remaining FD's.

$$\{A\}+ = \{A, C\} \qquad \{A\}+' = \{A\}$$
$$\{B, C\}+ = \{B, C, D\} \qquad \{B, C\}+' = \{B, C\}$$

Here $+'$ means closure where we removed the appropriate FD, i.e. $\{A\}+'$ is computed in $F' \setminus \{\{A\} \to \{C\}\}$ and $\{B, C\}+'$ is computed in $F' \setminus \{\{B, C\} \to \{D\}\}$.
In conclusion we end out with the minimal cover $\mathcal{F} = \{\{A\} \to \{C\}, \{B, C\} \to \{D\}\}$.

Question 3: We wish to make a 3NF synthesis of R with F.
Step 1: In question 2 we computed the minimal cover, we reuse this here.
Step 2 and 3: We create a relation for each FD in the minimal cover:

$\mathcal{R}_1 = \{A, C\}$	local key $\{A\}$
$\mathcal{R}_2 = \{B, C, D\}$	local key $\{B, C\}$
$\mathcal{R}_3 = \{A, B\}$	candidate key table

Here $\mathcal{R}_1$ and $\mathcal{R}_2$ are formed to maintain the 2 FD's from $\mathcal{F}$ and $\mathcal{R}_3$ to contain our candidate key $\{A, B\}$. Merging
This decomposition is lossless since we can reforge our data on A and B, first we can get the C values from $\mathcal{R}_1$ with A, then we can get the D values from $\mathcal{R}_2$ with B and C. More schematically we merge $\mathcal{R}_3$ with $\mathcal{R}_1$ on A which is lossless since A generates $\mathcal{R}_1$, then this new table say $\mathcal{R}_4 = \{A, B, C\}$ can be combined with $\mathcal{R}_2$ on B and C giving us our original R.
The decomposition is on 3NF because values are atomic and depend upon local candidate keys within their new structures (1NF), we have no partial dependencies (at all) to non key attributes (2NF), and no transitive dependencies (at all) to non key attributes (3NF).

Question 4: A short and precise explanation of why R with F is not on BCNF
The FD $\{B, C\} \to \{D\}$ is a BCNF violation since the left hand side was proved NOT to be a super key, and on BCNF this must always be the case! cf. chapter 14.5. Note also that $\{A\} \to \{C\}$ is a BCNF violation, by the same reasoning.

Question 5: Claim $(\{A, B\} \to \{C\}) \in F+$
Proof: Recall that the definition of $F+$ is that it is all FD's that can be produced from F by Armstrongs Axioms. Consider then:

$$(\{A\} \to \{C\}) \in F \tag{3.1}$$
$$\{A, B\} = \{A\} \cup \{B\} \xrightarrow{\text{Augmentation of (3.1) by } \{B\}} \{C\} \cup \{B\} = \{C, B\} \tag{3.2}$$
$$(\{C, B\} \to \{D\}) \in F \tag{3.3}$$
$$\{A, B\} \xrightarrow{\text{Transitivity of (3.2) and (3.3)}} \{D\} \tag{3.4}$$

So $\{A, B\} \to \{C\}$ is in $F+$. □

Question 6: We wish to find the BCNF decomposition that **is not** dependency preserving
We use the minimal cover we previously found, both FD's within it are BCNF violations in R with $\mathcal{F}$, so we choose to decompose with regards to $\{A\} \rightarrow \{C\}$, we then compute $A+ = \{A, C\}$ and decompose

$$\mathcal{R}_1 = \{A, C\} \qquad \mathcal{R}_2 = \{A, B, D\}$$
$$\mathcal{F}_1 = \{\{A\} \rightarrow \{C\}\} \qquad \mathcal{F}_2 = \emptyset$$

Now $\mathcal{R}_2$ has no non trivial FD's, and we have lost the FD $\{B, C\} \rightarrow \{D\}$, making this BCNF decomposition non FD preserving.

Question 7: We wish to find the BCNF decomposition that **is** dependency preserving
Consider now the other choice we had for our decomposition in question 6, namely decomposing with regards to $\{B, C\} \rightarrow \{D\}$, compute $\{B, C\}+ = \{B, C, D\}$ so we decompose into

$$\mathcal{R}_1 = \{A, C\} \qquad \mathcal{R}_2 = \{B, C, D\}$$
$$\mathcal{F}_1 = \{\{A\} \rightarrow \{C\}\} \qquad \mathcal{F}_2 = \{\{B, C\} \rightarrow \{D\}\}$$

Note that this is almost the same as our 3NF decomposition from question 3. And that it is clearly dependency preserving.

4 Exercise IV

Question 1: How many tuples are there in Book for each category value?
We know that there are 6 categories, and 1200 books uniformly distributed amongst non-key values. This means that we can expect 1200/6=200 books in each category. However it is worth noting that statistics isn't an exact science so some standard deviation is expected.

Question 2: How many disk blocks are needed to store the Book tuples for each category value?
When 1200 books take up 600 disk blocks we can fit 2 books in a block, and since we have 200 books in a category it will take up 100 blocks of space to store the book tuples with a common category.

Question 3: How many tuples in Purchasing qualify the query's condition on Month?
It is stated that our database contains 10 unique values of month, and since month is a non key attribute it is uniformly distributed so we have $10800/10 = 1080$ tuples representing purchases made in february. Note that we do not know if february is one of the months that we do not have any purchasing information on, in which case we get 0 tuples matching the search.

Question 4: How many tuples of Purchasing are there in one disk block?
10800 tuples take up 900 blocks, so $10800/900 = 12$ purchases pr block.

Question 5: How many disk blocks should we reach to obtain all tuples in Purchasing that qualify the query's condition on Month?
We should reach all books of category "cs"and all purchases made in february, this amounts to $1080/12 = 90$ blocks. Unless february was one of the dead months, in which case we need to reach 0 blocks.

Question 6: Assume we use the B+ tree index on Book to process the query's condition on a book's Category. How many disk blocks have to be read for that purpose?
We know that the B+-tree that stores the index of category can be used to access the first tuple of any category in 2 disk accesses, and after that we can reach the remaining tuples with the same category can be reached with disk accesses equal to the storage space they take. In total 102 disk accesses.

Question 7: Similarly, assume we use the B+-tree index on Purchasing to process the query's condition on Month. How many disk blocks have to be read for that purpose?
Similarly to question 6, we the B+-tree allows us quick access to the purchases made in february in 2 disk accesses, for a total of 92. Unless february was one of the months with no data, then just 2 to determine that nothing was there to be found

Question 8.a: We allocate 40 pages of memory for the input of Book, 1 page for the input of Purchasing, and 1 page for output. We load the qualifying blocks in Book in iterations (i.e., steps), using the 40 pages of allocated memory (i.e., we load as many as fit in memory in each iteration). In each iteration, we find all qualifying tuples in Purchasing, for each qualifying tuple in Book that is in memory.
Allocating 40 pages to book we can load in 80 of the relevant tuples then run through all the relevant purchases 12 at a time and check if book.bid = purchase.bid, this makes us run through all relevant purchases 3 times for a total of $102 + 3 * 92 = 378$ disk accesses.
Note that we use a look up in the B+-tree at the start of each the 3 iterations, so it takes 92 disk accesses for each iteration to run through the relevant purchases.

Question 8.b: We allocate 40 pages of memory for the input of Purchasing, 1 page for the input of Book, and 1 page for output. We load the qualifying blocks in Purchasing in iterations
In this case we have 90 blocks of relevant purchasing information to compare, this once more takes 3 iterations to cycle through, so in total $92 + 3 * 102 = 398$ disk accesses.
Note that we use a look up in the B+-tree at the start of each the 3 iterations, so it takes 102 disk accesses for each iteration to run through the relevant books.
The query uses B+-trees to find proper place to start and end the iteration in each case, and in this manner saves a great number of disk accesses that we know from the get go wont yield any desirable results.
Note that if February was one of the months without data, these operations would be over in $42 + 2 = 44$ and 2 disk accesses, given that we load data into memory first, and then look for relevant counterparts for the iteration.

Question 9: Which of the two ways suggested above would you prefer?
I prefer option a), since it is quicker.

Question 10: Based on your experience after answering the above questions, write down your thoughts on how to process a join query on two tables that involves a condition on an attribute of each of the two joined tables, when we have an available B+-tree on each of those two attributes of the query's conditions.
Notice that in general we used the formula

$$t = (b + l(B)) + \left\lceil \frac{b}{m} \right\rceil (a + l(A)) \tag{4.1}$$

Where a and b represents the number of disk blocks categories A and B fills, $l(A)$ and $l(B)$ the

disk block accesses of a lookup in the B+-tree, and m the memory available to hold one of the categories in. In general we want to use the procedure that minimizes the above, but

$$(132 + 4 * (83), 83 + 3 * 132) = (464, 479)$$
$$(132 + 4 * (120), 120 + 3 * 132) = (612, 516)$$

Above is an example showing that if we wish to be efficient in general, we have to make the computation each time we make a query to achieve the fastest speed.

5 Exercise V

Given the schedule

$$S : R1(A), W1(B), R1(C), R2(C), R2(B), R2(A), W3(B), W3(A)$$

Subtask 1) We wish to show that its is executed in a way that satisfies 2 phase locking.
Consider the following complete schedule

$$\begin{aligned} S_{complete} : \; & S1[A]; R1[A]; X1[B]; W1[B]; S1[C]; R1[C]; U1[A]; U1[B]; U1[C]; \\ & S2[C]; R2[C]; S2[B]; R2[B]; S2[A]; R2[A]; U2[A]; U2[B]; U2[C]; \\ & X3[B]; W3[B]; X3[A]; W3[A]; U3[A]; U3[B]; \end{aligned}$$

We lock and work in the appropriate sequence and, with a Shared lock ahead of a read and a eXclusive lock ahead of a write. The original schedual is unambitious and just takes care of each transaction one at a time, so the 2 phase lock does not run into any trouble.
Subtask 2) we wish to make a change to the schedual that makes it impossible to execute with 2 phase locking.
Consider the schedual

$$\begin{aligned} S_{no} : & S1[A]; R1[A]; X3[B]; W3[B]; U3[B]; X1[B]; W1[B]; S1[C]; R1[C]; U1[A]; \\ & U1[B]; U1[C]; S2[C]; R2[C]; S2[B]; R2[B]; S2[A]; R2[A]; U2[A]; \\ & U2[B]; U2[C]; X3[A]; W3[A]; U3[A]; \end{aligned}$$

This red part is a big problem, we try to have transaction 3 write to B before transaction 1, which is a problem since in order to have proper concurrency the end result of running transaction 1-3 should be that B has the value T3 writes it to have, and in S_{no} the output has the B value from to T1. It is also in conflict with 2PL since transaction 3 locks unlocks locks and unlocks, which is not allowed in the 2 phase locking system.