Database Management Systems (COP 5725)

(Spring 2009)

Instructor: Dr. Markus Schneider

TA:
Young Namkoong
Xiao Li

Exam 3 Solutions

Name:	
UFID:	
Email Address:	

Pledge (Must be signed according to UF Honor Code)

On my honor, I have neither given nor received unauthorized aid in doing this assignment.

_____Signature

For scoring use only:

	Maximum	Received
Exercise 1	30	
Exercise 2	20	
Exercise 3	20	
Exercise 4	30	
Total	100	

Exercise 1 (Functional Dependencies) [30 points]

(a) [10 points] Given the set $F = \{A\text{->}B, B\text{->}C, A \cup B\text{->}C\}$ of FDs where A, B, and C are attribute sets. Compute a minimum cover (canonical cover) F_c of F and document the single steps.

SOLUTION

Step 1 (left reduction): The FD $A \cup B \rightarrow C$ is replaced by $A \rightarrow C$, because B on the left side is extraneous (C is already functionally dependent from A by the first two FDs).
Step 2 (right reduction): The FD $A \rightarrow C$ is replaced by $A \rightarrow \emptyset$, because C on the right side is extraneous. This results from the fact that $C \subseteq AttrClosure(\{A \rightarrow B, B \rightarrow C, A \rightarrow \emptyset\}, A)$.
Step 3 (elimination of FDs with the empty set on their right side): The FD $A \rightarrow \emptyset$ is removed.
Step 4 (apply union rule): Nothing has to be done.
We obtain as a result $F_c = \{A \rightarrow B, B \rightarrow C\}$.

(b) [10 points] Consider the relational schema $R = A \cup B \cup C \cup D := ABCD$ and the two sets $F = \{AC \rightarrow B, A \rightarrow C, D \rightarrow A\}$ and $G = \{A \rightarrow B, A \rightarrow C, D \rightarrow A, D \rightarrow B\}$ of FDs. Use Armstrong's axioms to prove that these two sets are equivalent.

SOLUTION

We have to show that every FD in G is entailed by F, and vice versa. The FDs $A \rightarrow C$ and $D \rightarrow A$ belong to both F and G; thus, the derivation is trivial.

We show that $A \rightarrow B \in G$ is implied by F. We begin with $A \rightarrow C \in F$. Applying Armstrong's augmentation axiom leads to $A \rightarrow AC$. Now, in addition, we take $AC \rightarrow B \in F$ and apply Armstrong's transitivity axiom. We obtain A→B.

Next, we show that $D \rightarrow B \in G$ is implied by F. We begin with $D \rightarrow A \in F$ and additionally consider $A \rightarrow B$ derived in the previous step. By applying Armstrong's transitivity axiom, we obtain $D \rightarrow B$.

Finally, we show that AC→B ∈ F is implied by G. We begin with A → B ∈ G. Applying Armstrong's augmentation axiom leads to AC→BC. Armstrong's decomposition axiom yields AC→B.

(c) [10 points] Consider the relational schema $R = A \cup B \cup C \cup D := ABCD$ and the set $F = \{AC \rightarrow B, A \rightarrow C, D \rightarrow A\}$ of FDs. Does R have a key? If so, show that your assumption is indeed a key. How many candidate keys are there? If not, explain your statement.

SOLUTION

Attribute D is the only key of R. We have to show that $D \rightarrow ABCD = R$. We directly have (i) $D \rightarrow A$. Additionally considering $A \rightarrow C$ and applying Armstrong's transitivity axiom, we obtain (ii) $D \rightarrow C$. We now apply the union axiom to (i) and (ii) and imply (iii) $D \rightarrow AC$. The transitivity rule on (iii) and $AC \rightarrow B$ yields (iv) $D \rightarrow B$. Further we have the trivial FD (v) D→D. Finally, the union rule applied to (iii), (iv), and (v) provides us with $D \rightarrow ABCD = R$.

There is only one key since attribute D cannot be functionally determined by any attribute combination from ABC.

Exercise 2 (Decomposition) [20 points]

Suppose we have a schema, Lending, given as

Lending = (bname, bcity, assets, cname, loan#, amount)

And suppose an instance of this relation is

bname	bcity	assets	cname	loan#	amount
Campus	Gainesville	20M	Schneider	L-10	10K
Campus	Gainesville	20M	Young	L-20	15K
Downtown	Miami	80M	Schneider	L-50	50K

A tuple t in the new relation has the following attributes:

- t[assets] is the assets for t[bname]
- t[bcity] is the city for t[bname]
- t[loan#] is the loan number made by branch t[bname] to customer t[cname].
- t[amount] is the amount of the loan for t[loan#]

The set of functional dependencies we required to hold on this schema was:

- bname → assets bcity
- loan# → amount bname

1) In the schema **Lending**, we are repeating the assets and branch city information for every loan. Now assuming you are an expert, please tell us three different kinds of **anomalies** that might be caused in this relation by the information repetition. Give examples by using the instances of **this schema**. (6 points)

SOLUTION

A) Update anomalies

The bcity/assets of a branch can be changed in one of its tuples but remains unchanged in another tuple (→ inconsistency)

B) Insertion anomalies

A bcity's address cannot be inserted without a loan#.

C) Deletion anomalies

The deletion of the last loan information leads to a loss of the branch's bcity and assets.

2) (a) List a sufficient condition for the losslessness of a decomposition. (3 points)
(b) Is it a necessary condition? (1 point)
(c) List at least two lossless-join decompositions in the relation **Lending**. (4 points)

SOLUTION

(a) Condition:

Let R be a relation schema and FR the set of FDs. A decomposition of R in R1 and R2 is lossless, if

$(R1 \cap R2) \rightarrow R1 \in FR$ or $(R1 \cap R2) \rightarrow R2 \in FR$

i.e., $R1 \cap R2$ is a superkey for R1 or R2

Alternative formulation:

Let $R = A \cup B \cup C$, $R1 = A \cup B$ and $R2 = A \cup C$ with pairwise disjoint attribute sets A, B, C. Then:
B □ AttrClosure(FR, A) or C □ AttrClosure(FR, A) must hold.

(b) Both are sufficient, but not necessary conditions for losslessness

(c) Lossless-join decomposition 1:

Branch-schema = (bname, bcity, assets)
Loan-info-schema = (bname, cname, loan#, amount)

Lossless-join decomposition 2:

Branch-schema = (bname, bcity, assets)
Loan-schema = (bname, loan#, amount)
Borrow-schema = (cname, loan#)

Lossless-join decomposition 3:

Loan-schema = (bname, loan#, amount)
X-schema = (bcity, assets, loan#, cname)

3) Consider a design where Lending is decomposed into two schemas

Branch-customer = (bname, bcity, assets, cname)
Customer-loan = (cname, loan#, amount)

(a) Xiao thinks this is a good design because we can reconstruct the **lending** relation by performing a natural join on the two new schemas. Now you have to show Xiao at least two reasons why this decomposition is awful. (Hints: consider the correctness criteria for such a decomposition taught in class) (4 points)
(b) Use the data stored in the above table **Lending** to tell Xiao that this decomposition will make Schneider very angry. ☺ (2 points)

SOLUTION

a) Two main reasons:
1. lossy-join: The intersection of the two schemas is cname, so the natural join is made on the basis of equality in the cname. Although we have more tuples in the join, we have less information. Because of this, we call this a lossy or lossy-join decomposition.
2. dependency-loss: The FD loan# → bname is lost after this decomposition.

b) If we decompose the schema Lending into two schemas Branch-customer and Customer-loan, these two schema will be like:

bname	bcity	assets	cname

Campus	Gainesville	20M	Schneider
Campus	Gainesville	20M	Young
Downtown	Miami	80M	Schneider

cname	loan#	amount
Schneider	L-10	10K
Young	L-20	15K
Schneider	L-50	50K

Since two lendings are for Schneider, there will be four(!!!) tuples in the natural join. That is to say, the total amount of Schneider is doubled, and thus… Obviously, two of these tuples will be spurious - they will not appear in the original lending relation, and should not appear in the database.

bname	bcity	assets	cname	loan#	amount
Campus	Gainesville	20M	Schneider	L-10	10K
Campus	Gainesville	20M	Schneider	L-50	50K
Campus	Gainesville	20M	Young	L-20	15K
Downtown	Miami	80M	Schneider	L-10	10K
Downtown	Miami	80M	Schneider	L-50	50K

Exercise 3 (BCNF, 3NF) [20 points]

Consider R = {A, B, D, E, G, H, I} := ABCDEGHI and the following set F of functional dependencies:
H → GD
E → D
HD → CE
BD → A
a) Find a join loss-less, dependency preserving and **3NF** decomposition of R. (10 points)

SOLUTION
We first find a minimal cover of the FDs, as shown below.

Right reduced	Left reduced	Minimal cover
H → G	H → G	H → CDEG
H → D	H → D	E → D
E → D	E → D	BD → A
HD → C	H → C	
HD → E	H → E	
BD → A	BD → A	

Step 2 of the synthesis algorithm yields R1 = CDEGH and R2 = ABD.

Since neither R1 nor R2 contains a candidate key, we have to add R3 = BHI.

Hence, we obtain the database schema D = {R1, R2, R3} = {CDEGH, ABD, BHI}.
Now D is a join loss-less, dependency preserving and 3NF decomposition of R.

b) Indicate whether your database schema is in BCNF with respect to F. **Explain**. (6 points)

SOLUTION

D is not in BCNF since R1 is not in BCNF. In R1, E → D is not trivial and D is not superkey of R1.

c) In designing a relational database schema, **why** might we choose a non-BCNF design? (4 points)

SOLUTION

This is because in some cases, there exists no database schema that is both BCNF and dependence preserving. It is always possible to obtain a 3NF design without sacrificing dependency-preservation. If we cannot check for dependency preservation efficiently, we either pay a high price in system performance or risk the integrity of the data. The limited amount of redundancy in 3NF is then a lesser problem.

To summarize, our goal for a relational database design is
- BCNF.
- Lossless-join.
- Dependency-preservation.

If we cannot achieve this, we accept
- 3NF
- Lossless-join
- Dependency-preservation.

Exercise 4 (Data Integration) [30 points]

a) Give a real world example to present the importance of recursive constraints. (3 points)

SOLUTION

In a relation TrainConnection each station should be connected with each other station.

b) Compare the concepts "assertions" and "triggers" from the perspectives of purposes, costs and effects. (6 points)

SOLUTION

1) Both assertions and triggers are used to specify integrity constraints (Astrahan et al. ACM TODS 1 No 2, June 1976).
2) Assertions must be checked at any change to the mentioned. Assertions are powerful, but the DBMS often can't tell when they need to be checked.

Triggers let the user decide when to check for a powerful condition instead of simply preventing some action.

3) Triggers allow actions that may violate constraints, but assertions do not allow such actions.

c) List at least three critical problems when the application of triggers is considered. (6 points)

SOLUTION

1) User must control that triggers do not contradict each other.
2) A trigger can activate another trigger. Cycles should be avoided.
3) Termination of events
4) If a consistency condition can be formulated by an integrity constraint, triggers should not be used.

d) Use QBE to give three examples for representing the three kinds of different constraints below. **Explain** why. (6 points)

1. Key constraints

SOLUTION

Example 1: Checking key integrity when inserting a data record

order	cname	product	amount
I.	Smith	Pen	20

Example 2: Specifying key IC when creating a schema

I. order	I. cname	product	amount
KEY	I.	Y	N	N
TYPE	I. CHAR	CHAR	FIXED

2. Domain constraints

SOLUTION

Example 1: Checking domain constraints when inserting a data record

order	cname	product	amount
I.	Smith	Pen	20

Example 2: Specifying domain constraints when creating a schema

I. order	I. cname	product	amount
KEY	I.	Y	N	N
TYPE	I. CHAR	CHAR	FIXED

3. Dynamic integrity constraints

SOLUTION

supplier	sname	saddr	product	price

I.CONSTR(U.).I.	_N	bread	≤ _P	(new values)
	_N	bread	_P	(old values)

e) Consider the following SQL declarations:

```
Create table Employee
(ID integer primary key,
 salary integer,
 dept_num integer references Department(number));

Create table Department
(number integer primary key,
salaryCap integer);

Create assertion Policy check(
     Not exists (select *
                 From Employee, Department
                 Where Employee.dept_num  = Department.number
                 And Employee.salary > Department.salaryCap))
```

Hints: A salary cap, sometimes called a wage cap, is a limit on the amount of money a department can spend on salary per employee.

1. State in English the policy enforced by the assertion Policy. (3 points)

SOLUTION

The above assertion states that after any insertion or update on the Employee or the Department table, there should not be any employee having a salary greater than the salary cap in the corresponding department of the employee.

2. Rewrite the above table declarations to use **tuple constraints** instead of **referential constraints**. Your constraints should be defined so that under no circumstances the policy can be violated. Remember you will be graded for simplicity as well as correctness. *Please do not ask TAs about tuple constraints and referential constraints that have been taught in class.* (6 points)

SOLUTION

In the employee table define the following tuple based check constraint:

```
CHECK ( salary <=  (Select salaryCap  from Department
                    where Department.number=dept_num) )
```

In the department table add the following check constraint:

```
CHECK ( salaryCap >= ALL (Select salary from Employee
                                  where  Employee.dept_num=number) )
```