Introduction to Databases

Tutorial 7

Dr Paolo Guagliardo

Fall 2020 (week 10)

Problem 1 (mandatory). Consider a relation schema over attributes A, B, C, D, E, F with the following set of FDs: $\{ EF \to BC,\ A \to D,\ B \to AE,\ BD \to C \}$. Note that this is the same schema of Problem 1 in Tutorial 6.

(a) List all of the FDs that violate the requirements of 3NF.

(b) Apply the 3NF synthesis algorithm to obtain a lossless, dependency-preserving 3NF schema.

Solution. Call Σ the given set of FDs and U the given set of attributes (that is, $ABCDEF$).

(a) From the previous tutorial, we know that the FDs $A \to D$, $B \to AE$ and $BD \to C$ violate the BCNF requirements (they are all non-trivial and their left-hand sides are not keys). We also know that the candidate keys are BF and EF, so the prime attributes are B, E and F. Therefore, $A \to D$, $B \to AE$ and $BD \to C$ violate the 3NF requirements because, in addition to violating the BCNF ones, their right-hand sides do not consist only of prime attributes.

(b) To synthesize a 3NF schema we need a minimal cover of Σ.

1. We put the FDs in standard form and we get:

$$\Sigma_1 = \{EF \to B,\ EF \to C,\ A \to D,\ B \to A,\ B \to E,\ BD \to C\}$$

2. We cannot remove any attribute from the l.h.s. of $EF \to B$ and $EF \to C$ because EF is a candidate key. But we can remove D from the l.h.s. of $BD \to C$ because $C \in C_{\Sigma_1}(B) = BADCE$. So we get:

$$\Sigma_2 = \{EF \to B,\ EF \to C,\ A \to D,\ B \to A,\ B \to E,\ B \to C\}$$

3. We can remove the FD $EF \to C$ because it can be derived by transitivity from $EF \to B$ and $B \to C$:

$$\Sigma_3 = \{EF \to B,\ A \to D,\ B \to A,\ B \to E,\ B \to C\}$$

At this point we cannot remove any other FD:

- $\Gamma_1 = \{A \to D,\ B \to A,\ B \to E,\ B \to C\} \not\models EF \to B$ because $\{B\} \not\subseteq C_{\Gamma_1}(EF) = EF$;
- $\Gamma_2 = \{EF \to B,\ B \to A,\ B \to E,\ B \to C\} \not\models A \to D$ because $\{D\} \not\subseteq C_{\Gamma_2}(A) = A$;
- $\Gamma_3 = \{EF \to B,\ A \to D,\ B \to E,\ B \to C\} \not\models B \to A$ because $\{A\} \not\subseteq C_{\Gamma_3}(B) = BEC$;
- $\Gamma_4 = \{EF \to B,\ A \to D,\ B \to A,\ B \to C\} \not\models B \to E$ because $\{A\} \not\subseteq C_{\Gamma_4}(B) = BACD$;
- $\Gamma_5 = \{EF \to B,\ A \to D,\ B \to A,\ B \to E\} \not\models B \to C$ because $\{A\} \not\subseteq C_{\Gamma_5}(B) = BAED$.

So Σ_3 is a minimal cover of Σ. We can now apply the 3NF synthesis algorithm:

1. We group together the FDs with the same l.h.s., obtaining
$$\Sigma_4 = \{EF \to B,\ A \to D,\ B \to AEC\}$$
2. For each FD in Σ_4 we create a relation schema as follows:
$$(EFB, \{EF \to B\}),\ (AD, \{A \to D\}),\ (BAEC, \{B \to AEC\})$$
3. As there is already a relation schema, namely $(EFB, \{EF \to B\})$, whose set of attributes is a key for the *original schema* S (EFB contains a candidate key for S), we do not need to add one.
4. No relation schema can be removed, so the final lossless, dependency-preserving, 3NF schema is given by:
$$(EFB, \{EF \to B\}),\ (AD, \{A \to D\}),\ (BAEC, \{B \to AEC\})$$

Problem 2 (optional). Consider a relation schema over attributes A, B, C, D, E, F with the following set of FDs: $\Sigma = \{\, D \to A,\ F \to B,\ DF \to E,\ B \to C \,\}$. Note that this is the same schema of Problem 2 in Tutorial 6.

(a) List all of the FDs that violate the requirements of 3NF.

(b) Apply the 3NF synthesis algorithm to obtain a lossless, dependency-preserving 3NF schema.

Solution. Call Σ the given set of FDs.

(a) From the previous tutorial, we know that three of the FDs in Σ, namely $D \to A$, $F \to B$ and $B \to C$, violate BCNF (they are non-trivial and their left-hand sides are not keys). We also know that the only candidate key is DF and so the prime attributes of the schema are D and F. Thus, all of the FDs $D \to A$, $F \to B$ and $B \to C$ violate 3NF, because they are non-trivial, their left-hand sides are not keys, and their right-hand sides do not consist solely of prime attributes.

(b) To synthesize a 3NF schema we need a minimal cover of Σ.

1. The FDs in Σ are already in standard form (only one attribute in the r.h.s.).
2. The only l.h.s. we could minimize is that of $DF \to E$, but DF is a candidate key so we cannot remove any attribute without compromising equivalence to Σ.
3. It is easy to see that we cannot remove any FD:
   - $\{F \to B, DF \to E, B \to C\} \not\models D \to A$
   - $\{D \to A, DF \to E, B \to C\} \not\models F \to B$
   - $\{D \to A, F \to B, B \to C\} \not\models DF \to E$
   - $\{D \to A, F \to B, DF \to E\} \not\models B \to C$

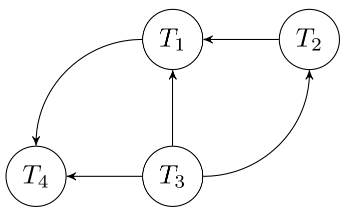
So the given set of FDs Σ is already a minimal cover. We now apply the 3NF synthesis algorithm:

1. There are no FDs in Σ with the same l.h.s. that we could group into one.
2. For each FD in Σ we create a relation schema as follows:
$$(DA, \{D \to A\}), \quad (FB, \{F \to B\}), \quad (DFE, \{DF \to E\}), \quad (BC, \{B \to C\})$$
3. Since there is already a relation schema, namely $(DEF, \{DF \to E\})$, whose set of attributes is a key for the *original schema* S, we don't need to add one.
4. None of the above relation schemas have a set of attributes contained in the set of attributes of another, so we are done.

Problem 3 (mandatory). Is the following schedule conflict serializable? Justify your answer. If it is, give an equivalent serial schedule.

	T_1	T_2	T_3	T_4
1			**r**(A)	
2		**w**(A)		
3	**r**(A)			
4			**r**(B)	
5			**r**(C)	
6		**w**(B)		
7	**w**(C)			
8				**r**(C)
9				**w**(C)

Solution. The precedence graph is as follows:



$T_3 \rightarrow T_2$ because T_3 reads A in step 1 before T_2 writes it in step 2 (T_3 also reads B in step 4 before T_2 writes it in step 6);

$T_2 \rightarrow T_1$ because T_2 writes A in step 2 before T_1 reads it in step 3;

$T_3 \rightarrow T_1$ because T_3 reads C in step 5 before T_1 writes it in step 7;

$T_3 \rightarrow T_4$ because T_3 reads C in step 5 before T_4 writes it in step 9;

$T_1 \rightarrow T_4$ because T_1 writes C in step 7 before T_4 reads it in step 8.

As there are no cycles, the given schedule is conflict serializable. The precedence graph tells us that T_3 must come before all other transactions, T_2 must come before T_1 and T_1 must come before T_4. Thus, the only equivalent serial schedule is: T_3, T_2, T_1, T_4.

Problem 4 (optional). Say whether each of the following schedules is conflict serializable. Justify your answer. If it is, give an equivalent serial schedule.

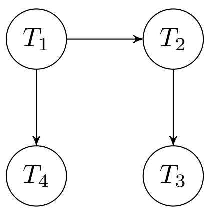
	T_1	T_2	T_3	T_4
1	**s**(A)			
2		**x**(C)		
3	**x**(B)			
4	**u**(A)			
5		**x**(A)		
6				**s**(B)
7				**u**(B)
8	**u**(B)			
9		**u**(C)		
10			**s**(C)	
11			**u**(C)	
12		**u**(A)		

	T_1	T_2	T_3	T_4
1	**s**(A)			
2		**x**(C)		
3	**x**(B)			
4	**u**(A)			
5	**u**(B)			
6		**x**(A)		
7				**s**(B)
8				**u**(B)
9		**u**(C)		
10			**s**(C)	
11			**u**(C)	
12		**u**(A)		

[*First check whether each schedule satisfies the basic rules of lock-based scheduling*]

Solution. The schedule on the left is not valid, because it violates the basic rules of lock-based scheduling: T_4 cannot get a shared lock on B in step 6 because T_1 has an exclusive lock on B (obtained in step 3) which it has not yet released.

In the schedule on the right, each lock is acquired only after all conflicting locks have been released, so the schedule satisfies the rules of lock-based scheduling. Moreover, all of the transactions follow the 2PL protocol, so we can conclude that the schedule is conflict serializable. However, to find an equivalent serial schedule we still need to construct the precedence graph. For lock-based schedules, this is constructed by examining each lock, finding the corresponding unlock operation by the same transaction, say T_i, and looking for a conflicting lock subsequently obtained by a different transaction T_j: if that is the case, there is an edge from T_i to T_j. The precedence graph for the schedule on the right is as follows:



$T_1 \to T_2$ because T_1 releases a shared lock on A at 4 and T_2 acquires an exclusive lock on A at 6;

$T_2 \to T_3$ because T_2 releases an exclusive lock on C at 9 and T_3 acquires a shared lock on C at 10;

$T_1 \to T_4$ because T_1 releases an exclusive lock on B at 5 and T_4 acquires a shared lock on B at 7.

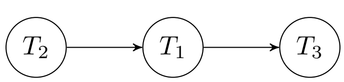
As we expected, there are no cycles. The graph tells us that T_1 must come before T_1 and T_4, and that T_2 must come before T_3. Thus, the possible equivalent serial schedules are:

- T_1, T_2, T_3, T_4
- T_1, T_2, T_4, T_3
- T_1, T_4, T_2, T_3

Problem 5 (optional). Is the following schedule conflict serializable? Justify your answer. If it is, indicate all of the serial schedules equivalent to it.

	T_1	T_2	T_3
1		**s**(B)	
2	**s**(C)		
3		**s**(A)	
4		**u**(B)	
5			**s**(A)
6		**u**(A)	
7	**x**(B)		
8			**u**(A)
9	**u**(C)		
10			**x**(C)
11			**u**(C)
12	**u**(B)		

Solution. The schedule is valid (each lock is acquired after all conflicting locks are released) but it does not satisfy 2PL (because T_3 gets a lock after releasing a lock), so we cannot directly conclude that the schedule is conflict serializable. We need to construct the precedence graph, which is as follows:



$T_2 \rightarrow T_1$ because T_2 releases a shared lock on B at 4 and T_1 acquires an exclusive lock on B at 7;

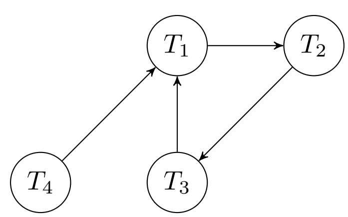
$T_1 \rightarrow T_3$ because T_1 releases a shared lock on C at 9 and T_3 acquires an exclusive lock on C at 10.

There are no cycles, so the given schedule is conflict serializable. The precedence graph tells us that T_2 must come before T_1 and T_1 must come before T_3, so the only equivalent serial schedule is: T_2, T_1, T_3.

Problem 6 (optional). In the following schedule, consider each line as a request for a lock. Indicate which requests can be granted and whether there are deadlocks. If a deadlock is found, suggest a proper way to resolve it.

	T_1	T_2	T_3	T_4
1	**s**(A)			
2	**x**(B)			
3		**x**(C)		
4				**s**(B)
5			**s**(D)	
6			**x**(A)	
7		**x**(D)		
8	**s**(C)			

Solution. The request in step 1 can be granted because there are no existing locks on A; similarly for the requests in steps 2 and 3. The request in step 4 cannot be granted and T_4 must wait for T_1 to release the exclusive lock on B. The request in step 5 can be granted because there is no lock on D. The request in step 6 cannot be granted and T_3 must wait for T_1 to release the shared lock on A. The request in step 7 cannot be granted and T_2 must wait for T_3 to release the exclusive lock on D. The request in step 8 cannot be granted and T_1 must wait for T_2 to release the exclusive lock on C. The *wait-for* graph is as follows:



The cycle $T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_1$ is a deadlock. This can be resolved by rolling back any one of T_1, T_2, T_3. Rolling back T_1 is preferable because two other transactions (T_3 and T_4) are waiting for it.