SE3DB3 TUTORIAL

Hongfeng Liang

Nov 21/28, 2014

Outline

- Functional Dependencies
- Lossless Join
- Dependency preservation
- BCNF & 3NF
- Armstrong's Axioms
- Types of Schedules
  - Two Phase Locking Protocol (2PL)
  - Strict Two Phase Locking (Strict 2PL)
- Example
- Deadlock
  - Deadlock Detection
  - Deadlock Prevention

Last Tutorial Next Week

- Nov 28/Dec 1 will be the last tutorial
- Will cover final exam preparation
- Q&A

Functional dependencies(FDs)

- FDs are **constraints** that are derived from the meaning and interrelationships of the data attributes
- X->Y holds if whenever two tuples have the same value for X, they must have the same value for Y
  - If t1[X]=t2[X], then t1[Y]=t2[Y]

FD example

A	B	C	D
a1	b1	c1	d1
a1	b1	c1	d2
a1	b2	c2	d1
a2	b1	c3	d2

FD: AB->C
Does it hold or not

Yes.

What if we insert (a1, b1, c2, d1) into the table

Not hold. Since (a1,b1)->c1 and (a1,b1)->c2.

Armstrong's Axioms

X, Y, Z are sets of attributes

- Reflexivity: If $Y \subseteq X$, then $X \rightarrow Y$
- Augmentation: If $X \rightarrow Y$, then $XZ \rightarrow YZ$ for any Z
- Transitivity: If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$
- Union: If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$
- Decomposition: If $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$

Lossless Join

- When we join decomposed relations we should get ***exactly*** what we started with.
- If we get more tuples when joining several decomposed relations when comparing to the original relation, there may be a information loss.
  - Note: More tuples do not give us more information.

Lossy decomposition example

R

Model Name	Price	Category
a11	100	Canon
s20	200	Nikon
a70	150	Canon

R1

Model Name	Category
a11	Canon
s20	Nikon
a70	Canon

R2

Price	Category
100	Canon
200	Nikon
150	Canon

Lossy decomposition example

R1 ⋈ R2

Model Name	Price	Category
a11	100	Canon
a11	150	Canon
s20	200	Nikon
a70	100	Canon
a70	150	Canon

R

Model Name	Price	Category
a11	100	Canon
s20	200	Nikon
a70	150	Canon

Testing for losslessness

- A (binary) decomposition of **R** = (*R*, ***F***) into **R**1 = (*R1*, ***F1***) and **R**2 = (*R2*, ***F2***) is lossless *if and only if* :
  - either the FD (*R1* ∩ *R2*)a *R1* is in ***F***+
  - or the FD (*R1* ∩ *R2*)a *R2* is in ***F***+

Example

- Schema: R (A, B, C), FDs: A->B, A->C
- Decomposition:

R1 (A, B), R2 (A, C) is lossless,

since $R1 \cap R2 = A$, and A->AB (or A->AC).

R3 (A, B), R4 (B, C) is not lossless,

since $R3 \cap R4 = B$, B cannot imply AB or BC.

Dependency preservation

- Dependency preservation: when decompose a relation into several small relations, all original functional dependencies will be satisfied.

BCNF

- BCNF: Boyce-Codd Normal Form
- We say a relation *R* is in *BCNF* if whenever *X* ->*Y* is a nontrivial FD that holds in *R*, *X* is a super key. *nontrivial* means *Y* is not contained in *X*.
- BCNF provides:
  - *Lossless Join* :✓
  - *No anomalies* :✓
  - *Dependency Preservation* : ✗

BCNF example

- Schema: R (A, B, C)

FD: AB -> C, C -> B

Keys: AB, AC

Does R satisfy BCNF

Answer: No.

(1) For AB -> C:

C not in AB, nontrivial, AB is a super key. ✓

(2) For C -> B:

B not in C, nontrivial, but C is not a super key. ✗

So C -> B violates BCNF, then R violates BCNF.

BCNF decomposition

- Schema: R (A, B, C)

FD: AB -> C, C -> B

Keys: AB, AC

Since C -> B does not satisfy BCNF:

- (1) compute C+: C+ = {C, B}
- (2) R1= C+ = (C, B), R2 = R – C+ + {C} = (A, C)
- (3) So we decompose R into R1(C, B), R2 (A, C).

  But we lost the FD: AB -> C

3NF

- 3NF: Third Normal Form
- Prime: an attribute is *prime* if it is a member of any key.
- *FD X ->A* violates 3NF if and only if *X* is not a super key, and also *A* is not prime.
- 3NF provides:
  - *Lossless Join* : ✓
  - *No anomalies* : ✗
  - *Dependency Preservation* : ✓

3NF example

- Schema: R (A, B, C)

  FD: AB -> C, C -> B

  Keys: AB, AC

  Does R satisfy 3NF

Answer: Yes.

C is prime since C is a member of key AC; B is prime since B is a member of AB. If the right side is prime, whether the left side is a super key or not, it satisfies 3NF.

3NF decomposition

- What if a schema does not satisfy 3NF? How to decompose it into 3NF
- We compute the minimal cover to decompose a relation into 3NF.
  - Right sides are single attributes.
  - No FD can be removed.
  - No attribute can be removed from a left side.
  - One relation for each FD in the minimal cover.

Locking Protocol

- Lock: a mechanism to control concurrent access to a data object.
- Types of Locks
  - Shared (S) lock: for reading
  - Exclusive (X) lock: for writing, and of course, also for reading
- Use a locking Protocol: a set of rules to be followed by each transaction to ensure serializable schedule.

Two Phase Locking Protocol (2PL)

- Two rules:
  1. Each Xact must obtain a S (shared) lock on object before reading, and an X (exclusive) lock on object before writing.
  2. A transaction can not request additional locks once it releases any lock.

  Note: If an Xact holds an X lock on an object, no other Xact can get a lock (S or X) on that object.
- Each transaction is executed in two phases:
  - Growing Phase: the transaction obtains locks
  - Shrinking phase: the transaction releases locks

Strict Two Phase Locking (Strict 2PL)

- Two rules:
  1. Each Xact must obtain a S (shared) lock on object before reading, and an X (exclusive) loc on object before writing.
  2. All locks held by a transaction are released when the transaction completes.

Example

- $W_1(X), R_2(X), W_2(X), W_1(Y), C_2, C_1$

T1	T2
W1(X)	
	R2(X)
	W2(X)
W1(Y)	
	C2
C1	

Serializable	
Conflict Serializable	
Recoverable	
ACA	
2PL	
Strict 2PL	

Example (Cont.)

- $W_1(X), R_2(X), W_2(X), W_1(Y), C_2, C_1$

Serializable	YES
Conflict Serializable	YES
Recoverable	NO
ACA	NO
2PL	NO
Strict 2PL	NO

Deadlock

- Deadlock: Cycle of transactions waiting for locks to be released by each other.

T1	T2
X(A)	
	X(B)
X(B)	
	X(A)

T1 is waiting for T2 to release its lock;
T2 is waiting for T1 to release its lock.
Such a cycle of transactions is a deadlock.

- Two ways of dealing with deadlocks:
  - Deadlock detection
  - Deadlock prevention

Deadlock Detection

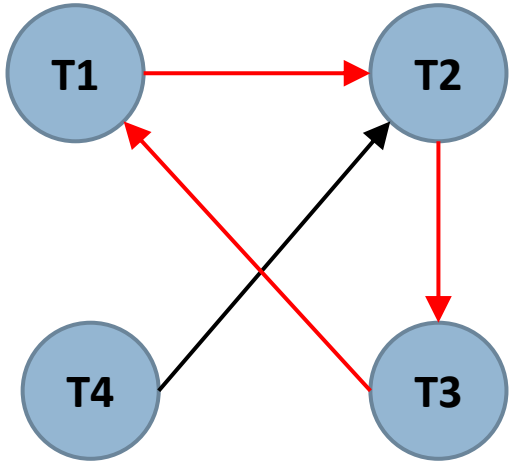
- The lock manager maintains a structure called a **waits-for graph** to detect deadlock cycles.
  - Nodes are transactions
  - There is an edge from Ti to Tj if Ti is waiting for Tj to release a lock
  - Lock manager adds edge when lock request is queued
  - Remove edge when lock request granted

  Note: A deadlock exists if there is a cycle in the waits-for graph.
- Deadlock is resolved by aborting a Xact in the cycle, and releasing its locks.

Deadlock Detection (Cont.)

- Waits-for graph example:

T1	T2	T3	T4
S(A)			
R(A)			
	X(B)		
	W(B)		
S(B)			
		S(C)	
		R(C)	
	X(C)		
			X(B)
		X(A)	



CYCLE → Deadlock

T1 is waiting for T2 to release a lock
T2 is waiting for T3 to release a lock
T4 is waiting for T2 to release a lock
T3 is waiting for T1 to release a lock

Deadlock Prevention

- Prevent deadlocks by giving each Xact a priority.
- Assign priorities based on timestamps.
  - Lower timestamp indicates higher priority
  - i.e., oldest transaction has the highest priority
  - Higher priority Xacts cannot wait for lower priority Xacts (or vice versa)
- Assume Ti wants a lock that Tj holds. Two policies are possible:
  - **Wait-Die**: If Ti has higher priority, Ti waits for Tj; otherwise Ti aborts.
  - **Wound-Wait**: If Ti has higher priority, Tj aborts; otherwise Ti waits.

Deadlock Prevention (Cont.)

- Example: Given the transaction timestamps: ts(T1)=10, ts(T2)=5, ts(T3)=12, and ts(T4)=6, determine the action of both protocols (wait-die and would-wait) for each of the following scenarios.
  - (a) T1 requests an object whose lock is held by T3.
  - (b) T4 requests an object whose lock is held by T2.

Deadlock Prevention (Cont.)

- Solution:
  - Note: Priority T2>T4>T1>T3

Question	Wait-Die	Would-Wait	Reason
(a)	T1 waits	T1 wounds T3 and T3 dies	T1 is older than T3
(b)	T4 dies	T4 waits	T4 is younger than T2