Tutorial 10

CSC343
Fall 2019

Oluwaseun Cardoso (OLUWASEUN.CARDOSO@MAIL.UTORONTO.CA)

Saihiel Bakshi (SAIHIEL.BAKSHI@MAIL.UTORONTO.CA)

Recall: Definitions

$BCNF \subseteq 3NF \subseteq 2NF \subseteq 1NF$

- 1NF: No multi-valued attributes allowed.
- 2NF: **Non-key** attributes depend on **candidate keys.**
  - If A is a non-key attribute, then ∃ X s.t. X -> A, and X is a candidate key.
- 3NF: **Non-prime** attributes depend only on **candidate keys.**
- BCNF: All non-trivial FDs have superkey LHS.

Example: BCNF Decomposition

Drinkers(name, addr, beersLiked, manf, favBeer)

F = name⟶addr, name⟶favBeer, beersLiked⟶manf

Key = name, beersLiked

- Pick BCNF violation name->addr.
- Closure: $\{name\}^+$ = {name, addr, favBeer}.
- Decomposed relations:
    - Drinkers1(name, addr, favBeer)
    - Drinkers2(name, beersLiked, manf)

Example: BCNF Decomposition

- We are not done; we need to check Drinkers1 and Drinkers2 for BCNF.
- Projecting FDs is easy here.
- For Drinkers1(name, addr, favBeer), relevant FDs are name→addr and name→favBeer.
  - Thus, {name} is the only key and Drinkers1 is in BCNF.

Example: BCNF Decomposition

- For Drinkers2(name, beersLiked, manf), the only FD is beersLiked→manf, and the only key is {name, beersLiked}.
  - Violation of BCNF.

- beersLiked$^+$ = {beersLiked, manf}, so we decompose *Drinkers2* into:
  - Drinkers3(beersLiked, manf)
  - Drinkers4(name, beersLiked)

Example: BCNF Decomposition

- The resulting decomposition of *Drinkers* :
  - Drinkers1(name, addr, favBeer)
  - Drinkers3(beersLiked, manf)
  - Drinkers4(name, beersLiked)

- Notice: *Drinkers1* tells us about drinkers, *Drinkers3* tells us about beers, and *Drinkers4* tells us the relationship between drinkers and the beers they like.

Checkpoint

Complete the BCNF decomposition from the worksheet on quercus.

Introduction to Transactions & Concurrency

Transactions

- A sequence of many actions which are considered to be one unit of work.
  - Example:

    T1: R(A) R(B) W(B) W(A) Commit

- R(A): Read database object A
- W(A): Writing (to) an object A

- Commit: Committing transaction
- Abort: Aborting transaction

Schedules

- A list of actions from a set of transactions in a specific order

Example:

- **T1:** R(A) R(B) W(B) W(A) Commit
- **T2:** R(B) W(A) Commit
- **S:** R_1(A) R_1(B) R_2(B) W_2(A) W_1(B) W_1(A) $Commit_1$ $Commit_2$

S	**T1**	R(A)	R(B)			W(B)	W(A)	Commit	
	T2			R(B)	W(A)				Commit

Conflict Operations

Two operations in a schedule are said to be conflict if they satisfy all three of the following conditions:

1. they belong to different transactions;
2. they access the same item A; and
3. at least one of the operations is a write(A).

Example in Sa: R1(A), R2(A), W1(A), W2(A), A1, C2

- R1(A),W2(A) conflict, so do R2(A),W1(A),
- R1(A), W1(A) do not conflict because they belong to the same transaction,
- R1(A),R2(A) do not conflict because they are both read operations

Write Read Conflict

1. they belong to different transactions? ✓
2. they access the same item?
3. at least one of the operatio

S	**T1**	R(A)	W(A)				R(B)	W(B)	Abort
	T2			R(A)	W(A)	Commit			
	Action	X = A	A = X + 200	Y = A	A = Y * 1.05				

This is a **W**rite**R**ead conflict. (dirty read

The problem is we are reading data ed (and later aborted).

Read Write Conflict

1. they belong to different transactions? ✓
2. they access the same it A)
3. at least one of the ope a write? ✓

This is a **R**$_{ead}$**W**$_{write}$ conflict.

The problem is that after T1 ha T2 runs a es A and then T1 continues to

S							
	T1	R(A)				W(A)	Commit
	T2		R(A)	W(A)	Commit		
	Action	X = 5	Y = 5	A = Y - 1		A = X -1	

Write Write Conflict

1. they belong to different transactions? ✓
2. they access the same i ✓
3. at least one of the ope

S	**T1**	W(A)				W(B)	Commit
	T2		W(A)	W(B)	Commit		
	Action	A = 1000	A = 2000	B = 2000		B = 1000	

This is a **W**$_{\text{rite}}$**W**$_{\text{write}}$ conflict.

T2 runs during T1 running and a that T1 wrote into A. The objective of T1 is to set both A and B to 1000, and the objective of T2 is to set both A and B to 2000. At the end of this schedule we have an inconsistent state.

Serializable

- A schedule is **serializable** if the results of executing that schedule is identical to executing the transactions in the schedule in some serial order.

S	**T1**	R(A)	W(A)			R(B)	W(B)		
	T2			R(A)	W(A)			R(B)	W(B)

- S is serializable because it is equivalent to running, **T1; T2;**
- **T1**'s read and write of B (shaded in grey) is not affected by **T2** in S (because R(A) W(A) do not affect B).

Any Questions?

- Do you have any questions?

  1. Check piazza
  2. Post the question on piazza
     (unless it's a personal question then email one of the TAs)

- If you have any content that you would like to be added in a Tutorial, please let me know by Friday!

- Email requests to:
  - saihiel.bakshi@mail.utoronto.ca