3rd normal form (3NF)

- Non-prime attr. depend *only* on candidate keys
  - Consider FD X -> A
  - Either X is a superkey OR A is *prime* (part of a key)
- Counter-example:
  - studio -> studioAddr

    (*studioAddr* depends on *studio* which is not a candidate key)

Title	Year	Studio	StudioAddr
Star Wars	1977	Lucasfilm	1 Lucas Way
Patriot Games	1992	Paramount	Cloud 9
Last Crusade	1989	Lucasfilm	1 Lucas Way

3NF, dependencies, and join loss

- Theorem: always possible to convert a schema to lossless join, dependency-preserving 3NF
- Caveat: always *possible* to create schemas in 3NF for which these properties do not hold
- FD loss example 1:
  - MovieInfo(title, year, studioName)
  - StudioAddress(title, year, studioAddress)

  => Cannot enforce studioName -> studioAddress
- Join loss example 2:
  - Movies(title, year, star)
  - StarSalary(star, salary)

  => Movies ⋈ StarSalary yields additional tuples

Boyce-Codd normal form (BCNF)

- One additional restriction over 3NF
  - All non-trivial FDs have superkey LHS
- Counterexample
  - CanadianAddress(<u>street</u>, <u>city</u>, <u>province</u>, postalCode)
  - Candidate keys: {street, postalCode}, {street, city, province}
  - FD: postalCode -> city, province
  - Satisfies 3NF: city, province both prime
  - Violates BCNF: postalCode is not a superkey

  => Possible anomalies involving postalCode

Boyce-Codd Normal Form

- We say a relation R is in *BCNF* if whenever $X \rightarrow A$ is a nontrivial FD that holds in R, X is a superkey.
  - Remember: *nontrivial* means A is not contained in X.

Example: a relation not in BCNF

88

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

FD's: name->addr, favBeer, beersLiked->manf

- Only key is {name, beersLiked}.
- In each FD, the left side is **not** a superkey.
- Any one of these FDs shows *Drinkers* is not in BCNF

Another Example

89

Beers(name, manf, manfAddr)

FD's: name->manf, manf->manfAddr

- Beers w.r.t. name->manf does not violate BCNF, but manf->manfAddr does.

In other words, BCNF requires that:
the only FDs that hold are the result of key(s).
Why does that help?

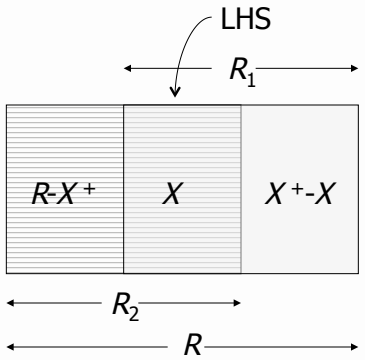
Decomposition into BCNF

90

- Given: relation *R* with FDs *F*
- Look among the given FDs for a BCNF violation $X \rightarrow Y$ (i.e., X is not a superkey)
- Compute X^+.
  - Find $X^+ \neq X \neq$ all attributes, (o.w. X is a superkey)
- Replace *R* by relations with:
  - $R_1 = X^+$.
  - $R_2 = R - (X^+ - X) = R - X^+ \cup X$
- Continue to recursively decompose the two new relations
- *Project* given FDs *F* onto the two new relations.

Decomposition on X →Y

91



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 -- Continued

- 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 -- Continued

- 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 -- Concluded

- 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.

What we want from a decomposition

96

- *Lossless Join* : it should be possible to project the original relations onto the decomposed schema, and then reconstruct the original,
  i.e., get back exactly the original tuples.
- *No anomalies*
- *Dependency Preservation* : All the original FDs should be satisfied.

What we get from a BCNF decomposition

97

- *Lossless Join* : ✓
- *No anomalies* : ✓
- *Dependency Preservation* : ✗

Example: Failure to preserve dependencies

98

- Suppose we start with R(*A,B,C*) and FDs
  - *AB* ->C and C ->*B*.
- There are two keys, {*A,B* } and {*A,C* }.
- C ->*B* is a BCNF violation, so we must decompose into *AC*, *BC*.

The problem is that if we use *AC* and *BC* as our database schema, we cannot enforce the FD *AB* →C in these decomposed relations.

3NF Let's Us Avoid This Problem

99

- *3rd Normal Form* (3NF) modifies the BCNF condition so we do not have to decompose in this problem situation.
- An attribute is *prime* if it is a member of any key.
- *X* →*A* violates 3NF if and only if *X* is not a superkey, and also *A* is not prime.
- i.e., it's ok if *X* is not a superkey, as long as *A* is prime.

Example: 3NF

- In our problem situation with FDs *AB* ->C and C ->*B*, we have keys *AB* and *AC*.
- Thus *A*, *B*, and C are each prime.
- Although C ->*B* violates BCNF, it does not violate 3NF.

What we get from a 3NF decomposition

- *Lossless Join* : ✓
- *No anomalies* : ✗
- *Dependency Preservation* : ✓

Why?

Unfortunately, neither BCNF nor 3NF can guarantee all three properties we want.

3NF Synthesis Algorithm

- We can always construct a decomposition into 3NF relations with a lossless join and dependency preservation.
- Need *minimal basis* for the FDs (same as used in projection)
  - Right sides are single attributes.
  - No FD can be removed.
  - No attribute can be removed from a left side.

3NF Synthesis – (2)

- One relation for each FD in the minimal basis.
  - Schema is the union of the left and right sides.
- If no key is contained in an FD, then add one relation whose schema is some key.

Example: 3NF Synthesis

- Relation R = ABCD.
- FDs *A->B* and *A->C*.
- Decomposition: AB and AC from the FDs, plus AD for a key.

Limits of decomposition

- Pick two…
  - Lossless join
  - Dependency preservation
  - Anomaly-free
- 3NF
  - Provides lossless join and dependency preserving
  - May allow some anomalies
- BCNF
  - Anomaly-free, lossless join
  - Sacrifice dependency preservation

Use domain knowledge to choose 3NF vs. BCNF

TRANSACTIONS & CONCURRENCY CONTROL

MICHAEL LIUT (LIUTM@MCMASTER.CA)
DEPARTMENT OF COMPUTING AND SOFTWARE
MCMASTER UNIVERSITY

SE 3DB3 Fall 2016

(Slides adapted from Dr. Fei Chiang, K. Goldberg (UC Berkeley) and Ramakrishnan & Gherke)

Introduction

- Concurrent execution of user programs is essential for good DBMS performance.
- Because disk accesses are frequent, and relatively slow, it is important to keep the CPU going by working on several user programs concurrently.

Transactions

- A user's program may carry out many operations on the data retrieved from the database, but the DBMS is only concerned about what data is read/ written from/to the database.
- A *transaction* is the DBMS's abstract view of a user program: a sequence of reads and writes.

Concurrency

- What is Concurrent Process (CP)?
  - Multiple users access databases and use computer systems simultaneously.
- Example: Airline reservation system.
  - An airline reservation system is used by hundreds of travel agents and reservation clerks concurrently.
  - Banking system: you may be updating your account balances the same time the bank is crediting you interest.

Concurrency

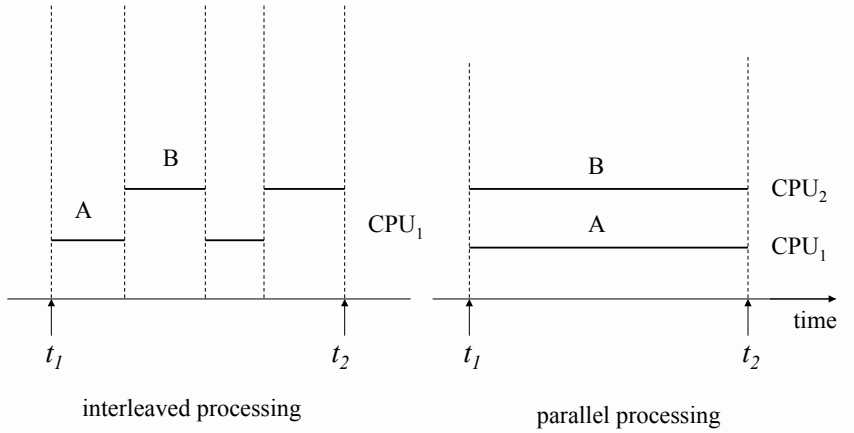
5

- Why Concurrent Process?
- Better transaction throughput and response time
- Better utilization of resource

- Concurrency
  - **Interleaved processing:**
    - Concurrent execution of processes is interleaved in a single CPU
  - **Parallel processing:**
    - Processes are concurrently executed in multiple CPUs.

Concurrent Transactions

6



Transactions

7

- What is a transaction?

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

  Basic operations a transaction can include "actions":
  - Reads, writes
  - Special actions: commit, abort

Concurrency (cont'd)

8

- Users submit transactions, and can think of each transaction as executing by itself.
  - Concurrency is achieved by the DBMS, which interleaves actions (reads/writes of DB objects) of various transactions.
  - Each transaction must leave the database in a consistent state if the DB is consistent when the transaction begins.
    - DBMS will enforce some constraints
    - Beyond this, the DBMS does not really understand the semantics of the data. (e.g., it does not understand how the interest on a bank account is computed).
- *Issues:* Effect of *interleaving* transactions, and *crashes.*

Atomicity of Transactions

9

- A transaction might *commit* after completing all its actions, or it could *abort* (or be aborted by the DBMS) after executing some actions.
- A very important property guaranteed by the DBMS for all transactions is that they are *atomic.* That is, a user can think of a Xact as always executing all its actions in one step, or not executing any actions at all.
  - DBMS *logs* all actions so that it can *undo* the actions of aborted transactions.

Transaction Properties (ACID)

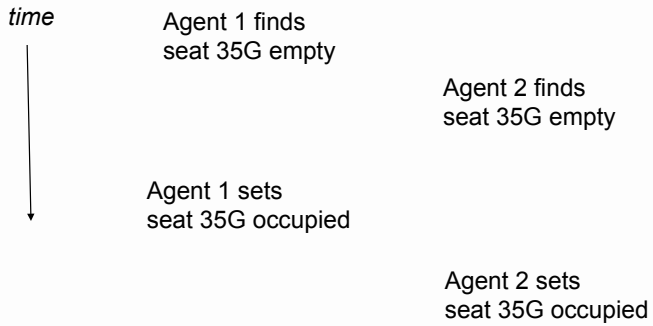
10

- Atomicity
  - Transaction is either performed in its entirety or not performed at all, this should be DBMS' responsibility
- Consistency
  - Transaction must take the database from one consistent state to another.
- Isolation
  - Transaction should appear as though it is being executed in isolation from other transactions
- Durability
  - Changes applied to the database by a committed transaction must persist, even if the system fails before all changes reflected on disk

Oops, Something's Wrong

11

- Reserving a seat for a flight
- In concurrent access to data in DBMS, two users may try to book the same seat simultaneously



Example

12

- Consider two transactions (*Xacts*):

```
T1:   BEGIN  A=A+100,  B=B-100  END
T2:   BEGIN  A=1.06*A,  B=1.06*B  END
```

- Intuitively, the first transaction is transferring $100 from B's account to A's account. The second is crediting both accounts with a 6% interest payment.
- There is no guarantee that T1 will execute before T2 or vice-versa, if both are submitted together. However, the net effect *must* be equivalent to these two transactions running serially in some order.

Example (Contd.)

- Consider a possible interleaving (*schedule*):

T1:	A=A+100,		B=B-100
T2:		A=1.06*A,	B=1.06*B

- This is OK. But what about:

T1:	A=A+100,		B=B-100
T2:		A=1.06*A, B=1.06*B	

- The DBMS's view of the second schedule:

T1:	R(A), W(A),		R(B), W(B)
T2:		R(A), W(A), R(B), W(B)	