Database Design: 3NF & Transaction Management

Abdu Alawini

University of Illinois at Urbana-Champaign

CS411: Database Systems

October 15, 2018

Announcements

- HW 2 is due TODAY (23:59)

- **Midterm: 10/29 in class 11-12:15 pm**

- I'll announce the midterm review session once I reserve a room.

Outline

- Third Normal forms (3NF)
- Transactions and ACID properties: the dangers in concurrent executions (Ch. 6.6)
- Transactions and SQL: isolation levels (Ch. 18.1-18.4)

Normal Forms

First Normal Form = all attributes are atomic

Second Normal Form (2NF) = old and obsolete

Boyce Codd Normal Form (BCNF)

Third Normal Form (3NF)

Fourth Normal Form (4NF)

Others...

3NF: A Problem with BCNF

Phone	Address	Name

FD's: Phone -> Address; Address, Name -> Phone

So, there is a BCNF violation (Phone ➔ Address), and we decompose.

Phone	Address

Phone -> Address

Phone	Name

No FDs

So where's the problem?

Phone	Address
1234	10 Downing
5678	10 Downing

Phone	Name
1234	John
5678	John

FD's: Phone-> Address; Address, Name -> Phone

No problem so far. All *local* FD's are satisfied.

Let's put all the data into a single table:

Phone	Address	Name
1234	10 Downing	John
5678	10 Downing	John

Violates the dependency: Address, Name ➔ Phone

Preserving FDs

- Thus, if the X and Y of a FD X->Y do not both end up in the same decomposed relation:
  - Such a decomposition is not "dependency-preserving."
  - No way to force BCNF to preserve dependencies
- Thus, while BCNF gives us **lossless join** and **less redundancy**, it doesn't give us **dependency preservation**

An alternative: 3rd Normal Form (3NF)

A simple condition for removing anomalies from relations:

A relation R is in 3rd normal form if:

Whenever there is a nontrivial dependency $A_1, A_2, ..., A_n \rightarrow B$ for R, then $\{A_1, A_2, ..., A_n\}$ is a super-key for R,
or B is part of a key.

- Prevents the "Phone ➔ Address" FD from causing a decomposition
- Textbook uses rule with many B_i on the RHS, if so, then each one must be part of some key.

3NF vs. BCNF

- R is in BCNF if whenever X->A holds, then X is a superkey.
  - Slightly stricter than 3NF.
    - Doesn't let R get away with it if A is part of some key
  - Thus, BCNF "more aggressive" in splitting
- Example: R(A,B,C) with {A,B}->C, C->A
  - 3NF but not BCNF

Decomposing R into 3NF

- Some preliminaries first: the “minimal basis”

Minimal basis

- Given a set of FDs: S.
- Say the set S' is *equivalent* to S, in the sense that S' can be inferred from S and v. versa.
  - Any such S' is said to be a *basis* for S.
- "Minimal basis"
  - A basis with all RHS singletons, where any modifications lead to no longer a basis, including:
    - Dropping attribute from LHS of a rule: compact rules
    - Dropping a rule: small # of rules

Example of minimal basis

- R(A, B, C) with FDs:
  - A -> B,C; B -> A,C; C -> A,B
- A basis:
  - A->B; A->C; B->A; B->C; C->A; C->B
- One minimal basis:
  - A->B
  - B->C
  - C->A
- Check this.

Conversion into minimal basis

- "Minimal basis" Condition
  - A basis with all RHS singletons, where any modifications lead to no longer a basis, including:
    - Dropping an attribute from the LHS of a rule
    - Dropping a rule

Algorithm for converting S to a minimal basis

- R = S with all RHS singletons
- Repeat until convergence:
  - If a rule minus an attribute from LHS is inferred from S, replace rule with rule minus attribute from LHS
  - If a rule is inferred from rest, drop it

Minimal basis example

Given R (u v w x y) and F = { U->X, ~~VW->UX~~, W->V, Y->U, Y->X}

Find F', the minimal basis for F.

Algorithm:
1- Only singleton in RHS
2- Remove unnecessary att. from LHS
3- Remove FDs that can be inferred from the rest

Decomposing R into 3NF

1. Get a "minimal basis" G of given FDs (Section 3.2.7)
2. For each FD $A \rightarrow B$ in the minimal basis G, use AB as the schema of a new relation.
3. If none of the schemas from Step 2 is a superkey, add another relation whose schema is a key for the original relation.

Result will be lossless, will be dependency-preserving, 3NF; might not be BCNF

Decomposing R into 3NF

1. Get a "minimal basis" G of given FDs (Section 3.2.7)
2. For each FD $A \rightarrow B$ in the minimal basis G, use AB as the schema of a new relation.
3. If none of the schemas from Step 2 is a superkey, add another relation whose schema is a key for the original relation. *Implicitly this is connecting all the LHSs with the remaining attributes*

Result will be lossless, will be dependency-preserving

Basically every minimal FD is preserved somewhere

Example

- R(A, B, C) with FDs:
  - A -> B,C; B -> A,C; C -> A,B

Minimal Basis: A->B; B->C; C->A

So, first cut:

R(A, B), R(B, C), R(C, A)

Any attributes left? Nope ➔ done

Example

- R(A, B, C, D, E) with FDs:
  - A -> B; CD -> B; DA -> C

  BCNF Decomp:

  (AB), (ACD), (ADE) or:
  (BCD), (ACD), (ADE)

  Which FDs do each of these not preserve?

Minimal Basis:

A -> B; CD -> B; DA -> C

3NF Decomp: (AB), (BCD), (ACD), (ADE)

Desirable Properties of Schema Refinement

1) minimize redundancy
✔ **2) avoid info loss**
✔ **3) preserve dependency**
4) ensure good query performance

3NF

Fact of life...

Finding a decomposition which is both lossless and dependency-preserving is not always possible.

Guideline: Aim for BCNF and settle for 3NF

Multi-valued Dependencies and 4NF

- we will not cover this.

Caveat

- Normalization is not the be-all and end-all of DB design
- Example: suppose attributes A and B are always used together, but normalization theory says they should be in different tables.
  - decomposition might produce unacceptable performance loss (extra disk reads)

Outline

- ✓ Third Normal forms (3NF)
- Transactions and ACID properties: the dangers in concurrent executions (Ch. 6.6)
- Transactions and SQL: isolation levels (Ch. 18.1-18.4)

Transaction Management

Ch 6.6.1 – 6.6.3

and 18.1-18.4

Motivating "Transactions"

- We've learned how to interact with DB using SQL.
- We assumed that:
  - each operation (e.g., UPDATE … SET … WHERE) is executed one at a time. One operation executes, perhaps changes the DB state, then next operation executes. (ISOLATION)
  - each operation is executed in entirety or not executed at all. (ATOMIC)
- Complications arise if these assumptions are violated
  - multiple operations acting on the same table simultaneously?
  - system crash in the middle of an operation (e.g., half the tuples have been updated the others not)

Example 1: flight seat selection

- Program:
  - Check if seat is available : SFW (A)
  - Book seat (change availability to "occupied"): NSW (B)

Two simultaneous runs

A1

A2

B1

B2

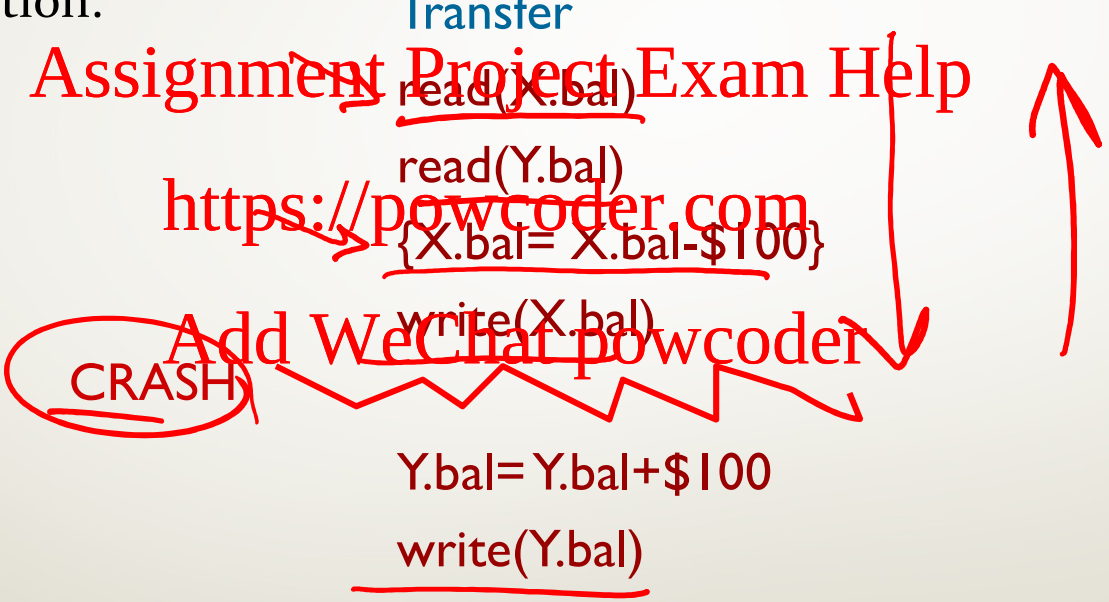
- Two executions of the same "UPDATE … SET … WHERE" leads to seat being double-booked

Example 1: lesson

- Group the "SELECT … FROM … WHERE" (which retrieved seat availability) and the "UPDATE … SET … WHERE" (which reserved a seat) into one TRANSACTION.
- Transaction is a sequence of statements that are considered a "unit of operation" on a database
- Either user1's transaction executes first and then user2's transaction, or the other way, but not in parallel. *Serializability of transactions.*

Example 2: bank inter-account transfer

Problems can also occur if a crash occurs in the middle of executing a transaction:



Need to guarantee that the write to X does not persist (ABORT)

- Default assumption if a transaction doesn't commit

Example 2: lesson

- Steps 2 and 3 must be done as one unit. *Atomically*.

- Either they both execute, or neither does.

- *Transactions must be atomic.*

Transactions

- Standard notion for updates is a transaction:
  - Sequence of read and write operations on data items that logically functions as one unit of work
  - If it succeeds, the effects of all write operations persist (*commit*); if it fails, no effects persist (*abort*)
  - These guarantees are made despite concurrent activity in the system, and despite failures that may occur

Transaction Manager

- Part of the DBMS
- Its job is to ensure that a transaction is executed as expected.
- **Purpose 1: Ensure that transactions that execute in parallel don't *interfere* with each other.**
- Purpose 2: Talks to "Log Manager", ensures that steps inside a transaction are being "logged".
- Purpose 3: Performs recovery after crashes, using logs.
- We will not cover recovery in this class.

ACID Properties

Atomicity

- either all of the actions of a transaction are executed, or none are.

Consistency

- each transaction executed in isolation keeps the database in a consistent state

Isolation

- Transactions are isolated from the effects of other, concurrently executing, transactions.

Durability

- updates stay in the DBMS!!!

Outline

- ✓ Third Normal forms (3NF)
- ✓ Transactions and ACID properties: the dangers in concurrent executions (Ch. 6.6)
- Transactions and SQL: isolation levels (Ch. 18.1-18.4)

Transactions in SQL

- A transaction begins when any SQL statement that queries the db begins.
- To end a transaction, the user issues a COMMIT or ROLLBACK statement.

Transfer


```
BEGIN
UPDATE Accounts
  SET balance = balance - $100
  WHERE account#= '1234';
UPDATE Accounts
  SET balance = balance + $100
  WHERE account#= '5678';
COMMIT;
```

Read-Only Transactions

- When a transaction only reads information, we have more freedom to let the transaction execute concurrently with other transactions.

- We signal this to the system by stating:


```
SET TRANSACTION READ ONLY;
SELECT * FROM Accounts
WHERE account#= '1234' ;
...
```

Read-Write Transactions

- If we state "read-only", then the transaction cannot perform any updates.

ILLEGAL!

```
SET TRANSACTION READ ONLY;
UPDATE Accounts
SET balance = balance - $100
WHERE account#= '1234' ; ...
```


- Instead, we must specify that the transaction may update (the default):

```
SET TRANSACTION READ WRITE;
update Accounts
set balance = balance - $100
where account#= '1234' ; ...
```