Review of Last Lecture

- BCNF exercises
- How to decompose a table with redundancy into BCNF
  - Attribute closure
- BCNF decomposition exercises
- Problem with BCNF
  - Lossless and minimal redundancy
  - But not FD preserving
- The need for 3NF

Today's Plan

- Transactions
  - ACID properties
  - 5 states of transaction
- Transactions in SQL
- 4 Isolation Levels in SQL

TRANSACTIONS

CHAPTER 09

Collin Roberts

University of Waterloo

Suggested Readings

- Textbook Chapter 17

The Setting

- Database systems are normally being accessed by many users or processes at the same time.
  - Both inquiries and modifications.
- Unlike operating systems, which *support* interaction of processes, a DBMS needs to keep processes from troublesome interactions.

Example: Bad Interaction

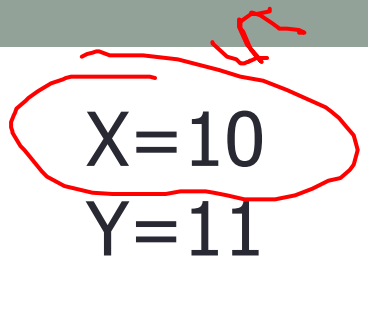
- Shared account between you and your mom with a starting Balance of $500.
- You and your Mom each withdraw $100 from different ATMs at the EXACT same time.
- At your ATM: Balance = Balance – $100 = $400
- At your mom's ATM: Balance = Balance – $100 = $400
- Final Balance = $400!!!!

- The DBMS better make sure one account deduction doesn't get lost in this situation.

Transaction Concept

- A **transaction** is a *unit* of program execution that accesses and possibly updates various data items.
  - A transaction must produce a consistent database.
  - During transaction execution, the database may be temporarily inconsistent.
  - When the transaction completes successfully (is committed), the database must again be consistent.
  - After a transaction commits, the changes it has made to the database persist, even if there are system failures.
- Multiple transactions can execute in parallel.
- Two main issues to handle:
  1. Concurrent execution of multiple transactions
  2. Failures of various kinds, such as hardware failures and system crashes

ACID Properties

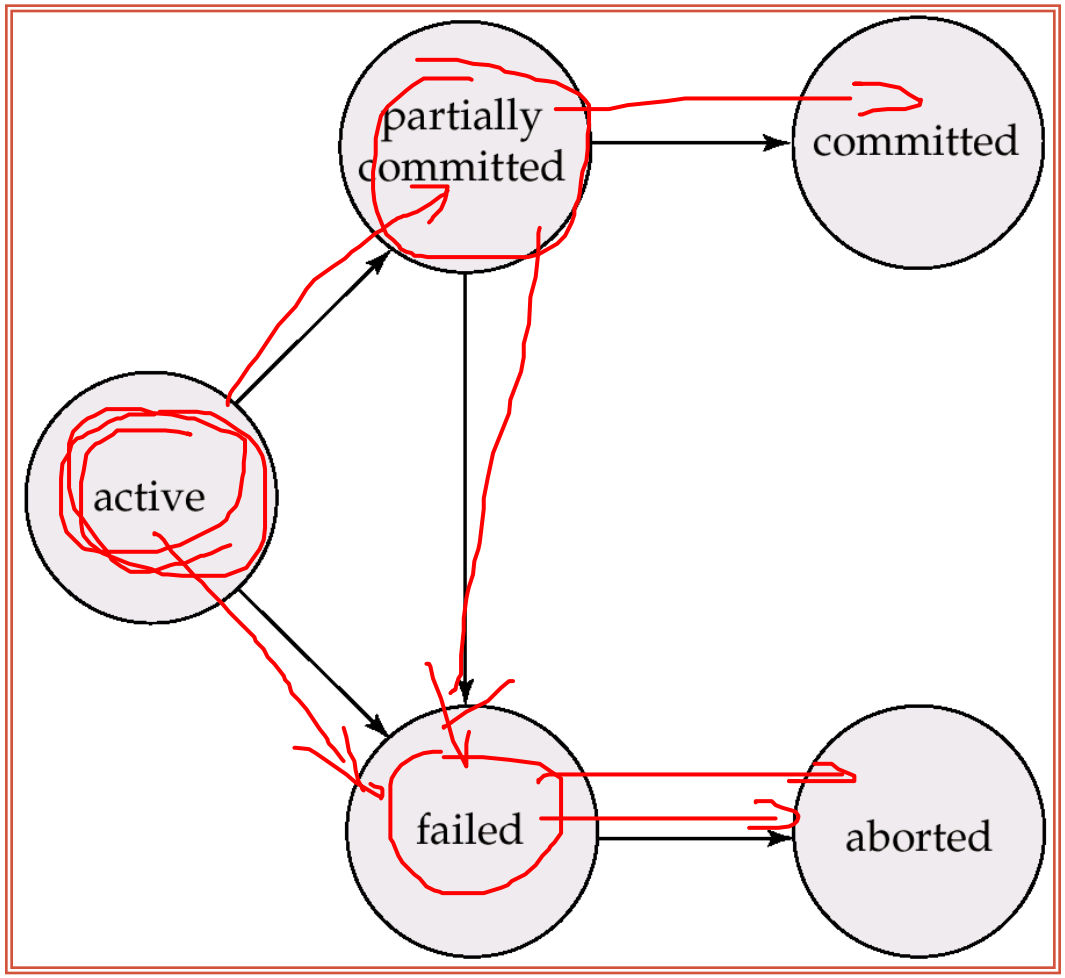


- **Atomicity**. Either all operations of the transaction are properly reflected in the database, or none is.
- **Consistency**. Database constraints are preserved.
- **Isolation**. Although multiple transactions may execute concurrently, each transaction must be unaware of other concurrently executing transactions. Intermediate transaction results must be hidden from other concurrently executed transactions.
  - That is, for every pair of transactions T_i and T_j *(i <>j),* it appears to T_i that either T_j, finished execution before T_i started, or T_j started execution after T_i finished.
- **Durability**. After a transaction completes successfully, the changes it has made to the database persist, even if there are system failures.

Transaction States

- **Active** – the initial state; the transaction stays in this state while it is executing
- **Partially committed** – after the final statement has been executed.
- **Failed** -- after the discovery that normal execution can no longer proceed.
- **Aborted** – after the transaction has been rolled back and the database restored to its state prior to the start of the transaction. Two options after it has been aborted:
  - restart the transaction; can be done only if no internal logical error
  - kill the transaction
- **Committed** – after successful completion.

Transaction State Diagram



Transactions in SQL

- SQL supports transactions, often behind the scenes.
  - Each statement issued at the generic query interface is a transaction unto itself.
  - In programming interfaces like JDBC or PSM, a transaction begins the first time a SQL statement is executed and ends with the statement or an explicit transaction-end.
    - That is, one statement is one transaction by default

Example: Explicit Transaction in SQL

- Defined by the user

```
BEGIN TRANSACTION;
UPDATE Account SET balance = 1.06 * balance
COMMIT;
```

COMMIT

- The SQL statement COMMIT causes a transaction to complete.
  - Its database modifications are then permanent in the database.

ROLLBACK

- The SQL statement ROLLBACK also causes the transaction to end, but instead by *aborting*.
  - No effects remain permanent on the database.
- Failures like division by 0 or a constraint violation can also cause rollback, even if the programmer does not explicitly request it.

Example: Interacting Processes

- Assume the usual Sells(bar,beer,price) relation, and suppose that Joe's Bar sells only Bud for $2.50 and Miller for $3.00.
- Joe decides to stop selling Bud and Miller, but to sell only Heineken at $3.50.
- Sally is querying the Sells for the highest and lowest price Joe charges.

Sally's Program

- Sally executes the following two SQL statements, which we call (min) and (max), to help remember what they do.

(max) SELECT MAX(price) FROM Sells
WHERE bar = 'Joe''s Bar';

(min) SELECT MIN(price) FROM Sells
WHERE bar = 'Joe''s Bar';

Joe's Program

- At about the same time, Joe executes the following steps, which have the mnemonic names (del) and (ins).

```
(del)   DELETE FROM Sells
        WHERE bar = 'Joe''s Bar';
(ins)   INSERT INTO Sells
        VALUES('Joe''s Bar', 'Heineken', 3.50);
```

Interleaving of Statements

- Although (max) must come before (min), and (del) must come before (ins), there are no other constraints on the order of these statements, unless we group Sally's and/or Joe's statements into transactions.
- (max)(del)(min)(ins)
- (max)(del)(ins)(min)
- … …

Example: Strange Interleaving

- Suppose the steps execute in the order (max)(del)(ins)(min).

Joe's Prices:	2.50, 3.00	2.50, 3.00		3.50
Statement:	(max)	(del)	(ins)	(min)
Result:	3.00			3.50

- Sally sees MAX < MIN!

Fixing the Problem by Using Transactions

- If we group Sally's statements (max)(min) into one transaction, then she cannot see this inconsistency.
  - That is, max and min treated as one and nothing can be changed in the middle.
- She sees Joe's prices at some fixed time.
  - Either before or after he changes prices, or in the middle, but the MAX and MIN are computed from the same prices.
  - Only the following sequences would Sally observe.
    - (max)(min)(del)(ins)
    - (del)(max)(min)(ins)
    - (del)(ins)(max)(min)

Another Problem: Rollback

- Suppose Joe executes (del)(ins), not as a transaction, but after executing these statements, thinks better of it and issues a ROLLBACK statement.
- If Sally executes her statements after (ins) but before the rollback, then she sees a value, 3.50, that never existed in the database.
- (del)(ins)(max)(min)(ROLLBACK)

Solution

- If Joe executes (del)(ins) as a transaction, its effect cannot be seen by others until the transaction executes COMMIT.
  - If the transaction executes ROLLBACK instead, then its effects can *never* be seen.

Isolation Levels

- SQL defines four *isolation levels* = choices about what interactions are allowed by transactions that execute at about the same time.
- How a DBMS implements these isolation levels is highly complex, and a typical DBMS provides its own options.

Choosing the Isolation Level

Within a transaction, we can say:

SET TRANSACTION ISOLATION LEVEL *X*

where *X* =

1. SERIALIZABLE
2. REPEATABLE READ
3. READ COMMITTED
4. READ UNCOMMITTED

Serializable Transactions

- If Sally = (max)(min) and Joe = (del)(ins) are each transactions, and Sally runs with isolation level SERIALIZABLE, then she will see the database either before or after Joe runs, but not in the middle.
- It is up to the DBMS vendor to figure out how to do that, e.g.:
  - True isolation in time.
  - Keep Joe's old prices around to answer Sally's queries.

Read-Committed Transactions

- If Sally runs with isolation level READ COMMITTED, then she can see only committed data, but not necessarily the same data each time.
- Example: Under READ COMMITTED, the interleaving (max)(del)(ins)(min) is allowed, as long as Joe commits.
  - Sally sees MAX < MIN.

Repeatable-Read Transactions

- Requirement is like read-committed, plus: if data is read again, then everything seen the first time will be seen the second time, the third time … …
  - But the second and subsequent reads may see *more* tuples as well.

Example: Repeatable Read

- Suppose Sally runs under REPEATABLE READ, and the order of execution is (max)(del)(ins)(min).
  - (max) sees prices 2.50 and 3.00.
  - (min) can see 3.50, but must also see 2.50 and 3.00, because they were seen on the earlier read by (max).

Read Uncommitted

- A transaction running under READ UNCOMMITTED can see data in the database, even if it was written by a transaction that has not committed (and may never commit).
- Example: (del)(ins)(max)(min).
  - If Sally runs under READ UNCOMMITTED, she could see a price 3.50 even if Joe later aborts.

Dirty Read: Example

- A data item is "dirty" if it has been written by an uncommitted transaction.

```
Client 1 - BEGIN TRANSACTION;
        ...
        UPDATE Student SET SAT = .99 * SAT
        ...
        COMMIT;

Client 2 - BEGIN TRANSACTION;
        ...
        SELECT AVG(SAT) FROM Student
        ...
        COMMIT;
```

SID	SAT
S1	1000
S2	1000

1000
990
995

Client 2 may only care about approximate average - dirty reads okay. Use:

- "SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED"

Isolation Level Is Personal Choice

- Your choice, e.g., run serializable, affects only how *you* see the database, not how others see it.
- Example: If Joe Runs serializable, but Sally doesn't, then Sally might see no prices for Joe's Bar.
  - i.e., it looks to Sally as if she ran in the middle of Joe's transaction.

Summary: 4 Level of Isolation in SQL

- Serializable — default
- Repeatable read — only committed records to be read, repeated reads of same record must return same value. However, a transaction may not be serializable – it may find some records inserted by a transaction but not find others.
  - SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
- Read committed — only committed records can be read, but successive reads of record may return different (but committed) values.
  - SET TRANSACTION ISOLATION LEVEL READ COMMITTED
- Read uncommitted — even uncommitted records may be read: dirty reads
  - SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED

Lower degrees of consistency useful for gathering approximate information about the database, e.g., statistics for query optimizer.

Summary of Today's Lecture

- Transactions
  - ACID properties
  - 5 states of transaction
- Transactions in SQL
- 4 Isolation Levels in SQL