Transactions

Overview

- Transactions
  - Concept
  - ACID properties
  - Examples and counter-examples
- Implementation techniques
- Weak isolation issues

Definition

- A transaction is a collection of one or more operations on one or more databases, which reflects a single real-world transition
  - In the real world, this happened (completely) or it didn't happen at all (Atomicity)
- Commerce examples
  - Transfer money between accounts
  - Purchase a group of products
- Student record system
  - Register for a class (either waitlist or allocated)

Coding a transaction

- Typically a computer-based system doing OLTP has a collection of *application programs*
- Each program is written in a high-level language, which calls DBMS to perform individual SQL statements
  - Either through embedded SQL converted by preprocessor
  - Or through Call Level Interface where application constructs appropriate string and passes it to DBMS

Why write programs?

- Why not just write a SQL statement to express “what you want”?
- An individual SQL statement can’t do enough
  - It can’t update multiple tables
  - It can’t perform complicated logic (conditionals, looping, etc)

COMMIT

- As app program is executing, it is “in a transaction”
- Program can execute COMMIT
  - SQL command to finish the transaction successfully
  - The next SQL statement will automatically start a new transaction

Warning

- The idea of a transaction is hard to see when interacting directly with DBMS, instead of from an app program
- Using an interactive query interface to DBMS, by default each SQL statement is treated as a separate transaction (with implicit COMMIT at end) unless you explicitly say “START TRANSACTION”

ROLLBACK

- If the app gets to a place where it can't complete the transaction successfully, it can execute ROLLBACK
- This causes the system to "abort" the transaction
  - The database returns to the state without any of the previous changes made by activity of the transaction

Reasons for Rollback

- User changes their mind (“ctl-C”/cancel)
- App program finds a problem
  - Eg qty on hand < qty being sold
- System-initiated abort
  - System crash
  - Housekeeping
    - Eg due to timeouts

Atomicity

- Two possible outcomes for a transaction
  - It *commits*: all the changes are made
  - It *aborts*: no changes are made
- That is, transaction's activities are all or nothing

Integrity

- A real world state is reflected by collections of values in the tables of the DBMS
- But not every collection of values in a table makes sense in the real world
- The state of the tables is restricted by integrity constraints
- Eg account number is unique
- Eg stock amount can’t be negative

Integrity (ctd)

- Many constraints are explicitly declared in the schema
  - So the DBMS will enforce them
  - Especially: primary key (some column's values are non null, and different in every row)
  - And referential integrity: value of foreign key column is actually found in another "referenced" table
- Some constraints are not declared
  - They are business rules that are supposed to hold

Consistency

- Each transaction can be written on the assumption that all integrity constraints hold in the data, before the transaction runs
- It must make sure that its changes leave the integrity constraints still holding
  - However, there are allowed to be intermediate states where the constraints do not hold
- A transaction that does this, is called consistent
- This is an obligation on the programmer
  - Usually the organization has a testing/checking and sign-off mechanism before an application program is allowed to get installed in the production system

Example - Tables

- System for managing inventory
- InStore(prodID, storeID, qty)
- Product(prodID, desc, mnfr, …, WarehouseQty)
- Order(orderNo, prodID, qty, rcvd, ….)
  - Rows never deleted!
  - Until goods received, rcvd is null
- Also Store, Staff, etc etc

Example - Constraints

- Primary keys
  - InStore: (prodID, storeID)
  - Product: prodID
  - Order: orderId
  - etc
- Foreign keys
  - Instore.prodID references Product.prodID
  - etc

Example - Constraints

- Data values
  - Instore.qty >= 0
  - Order.rcvd <= current_date or Order.rcvd is null
- Business rules
  - for each p, (Sum of qty for product p among all stores and warehouse) >= 50
  - for each p, (Sum of qty for product p among all stores and warehouse) >= 70 or there is an outstanding order of product p

Example - transactions

- MakeSale(store, product, qty)
- AcceptReturn(store, product, qty)
- RcvOrder(order)
- Restock(store, product, qty)
  - // move from warehouse to store
- ClearOut(store, product)
  - // move all held from store to warehouse
- Transfer(from, to, product, qty)
  - // move goods between stores

Example - ClearOut

- Validate Input (appropriate product, store)
- SELECT qty INTO :tmp
  FROM InStore
  WHERE StoreID = :store AND prodID = :product
- UPDATE Product
  SET WarehouseQty = WarehouseQty + :tmp
  WHERE prodID = :product
- UPDATE InStore
  SET Qty = 0
  WHERE prodID = :product
- COMMIT

Example - Restock

- Input validation
  - Valid product, store, qty
  - Amount of product in warehouse >= qty
- UPDATE Product
  SET WarehouseQty = WarehouseQty - :qty
  WHERE prodID = :product
- If no record yet for product in store
  INSERT INTO InStore (:product, :store, :qty)
- Else, UPDATE InStore
  SET qty = qty + :qty
  WHERE prodID = :product and storeID = :store
- COMMIT

Example - Consistency

- How to write the app to keep integrity holding?
- MakeSale logic:
  - Reduce Instore.qty
  - Calculate sum over all stores and warehouse
  - If sum < 50, then ROLLBACK // Sale fails
  - If sum < 70, check for order where date is null
    - If none found, insert new order for say 25

Threats to data integrity

- Need for application rollback
- System crash
- Concurrent activity

- The system has mechanisms to handle these

Application rollback

- A transaction may have made changes to the data before discovering that these aren't appropriate
  - the data is in state where integrity constraints are false
  - Application executes ROLLBACK
- System must somehow return to earlier state
  - Where integrity constraints hold
- So aborted transaction has no effect at all

Example

- While running MakeSale, app changes InStore to reduce qty, then checks new sum
- If the new sum is below 50, txn aborts
- System must change InStore to restore previous value of qty
  - Somewhere, system must remember what the previous value was!

System crash

- At time of crash, an application program may be part-way through (and the data may not meet integrity constraints)
- Also, buffering can cause problems
  - Note that system crash loses all buffered data, restart has only disk state
  - Effects of a committed txn may be only in buffer, not yet recorded in disk state
  - Lack of coordination between flushes of different buffered pages, so even if current state satisfies constraints, the disk state may not

Example

- Suppose crash occurs after
  - MakeSale has reduced InStore.qty
  - found that new sum is 65
  - found there is no unfilled order
  - // but before it has inserted new order
- At time of crash, integrity constraint did not hold
- Restart process must clean this up (effectively aborting the txn that was in progress when the crash happened)

Concurrency

- When operations of concurrent threads are interleaved, the effect on shared state can be unexpected
- Well known issue in operating systems, thread programming
  - see OS textbooks on critical section
  - Java use of synchronized keyword

ACID

- Atomic
  - State shows either all the effects of txn, or none of them
- Consistent
  - Txn moves from a state where integrity holds, to another where integrity holds
- Isolated
  - Effect of txns is the same as txns running one after another (ie looks like batch mode)
- Durable
  - Once a txn has committed, its effects remain in the database

Big Picture

- If programmer writes applications so each txn is consistent
- And DBMS provides atomic, isolated, durable execution
  - Ie actual execution has same effect as some serial execution of those txns that committed (but not those that aborted)
- Then the final state will satisfy all the integrity constraints

NB true even though system does not know all integrity constraints!