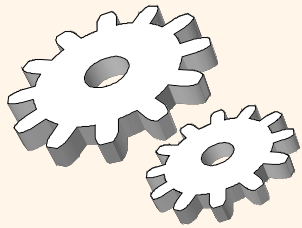
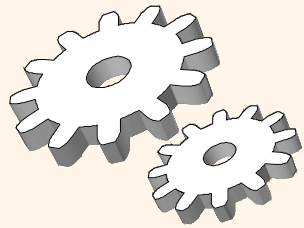
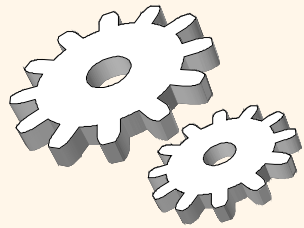


Transaction Management



Motivation

- What is a transaction and why we need them?
- Example: building a banking application
- Transfer $20 from checking to saving account
- We can write an app with two SQL statements to do this
  - one statement subtracts $20 from checking
  - another statement add $20 to saving
- What if the app crashes in between two statements?
- We would have an "inconsistent" state

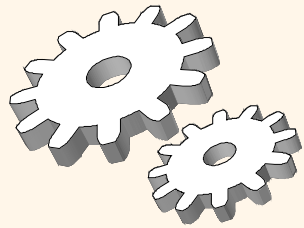


- Accounts(id, checking, saving)

- UPDATE Accounts
  SET checking = checking – 20
  WHERE id = 123

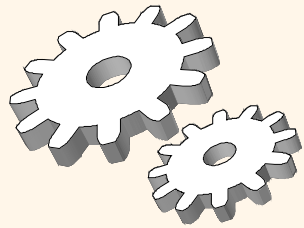
- UPDATE Accounts
  SET saving = saving + 20
  Where id = 123

Inconsistency



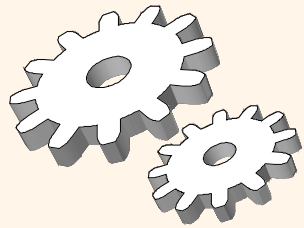
- When you write an application, you do have a notion of what it means for the app to be consistent, and when things may look inconsistent ("doesn't make sense")
  - "inconsistency" is subjective, depending on the business logic of the app
- Want to make sure app will never be in an inconsistent state
- Do this using the notion of transaction

Transaction



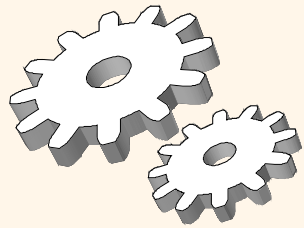
- A transaction is a sequence of SQL statements that you want to execute as a single "atomic" unit
  - either all statements in the transaction would be executed, or none would be executed
  - if the app is consistent before a transaction, we assume that it will also be consistent after the transaction
  - you don't want to execute a transaction half way; that can leave app in an inconsistent state
- When the app runs, transactions will be executed, one after another

What do we want the DB to do regarding transactions?

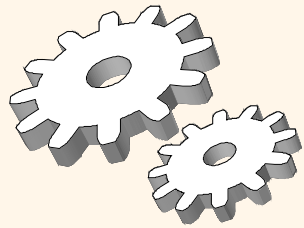


- ACID properties
- Atomic, consistent, isolation, durable
- Execute each transaction "atomically"
  - if a transaction crashes half way, then remove its effect
- Isolation: if two users run transactions concurrently, they should not interfere with each other
  - e.g., moving $20 from checking to saving AND return the balance (the sum of checking + saving)

What do we want the DB to do regarding transactions?



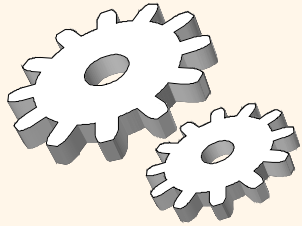
- Isolation: if two users run transactions concurrently, they should not interfere with each other
  - e.g., moving $20 from checking to saving AND return the balance (the sum of checking + saving)
  - what do we want in this case? sequential execution of the two transactions
- Durable: if a transaction has been executed, its effect is persisted in the database
- What about "consistency"?



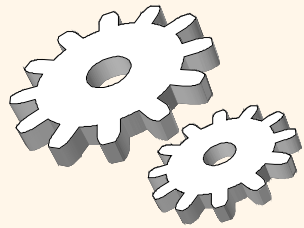
- Accounts(id, checking, saving)
- UPDATE Accounts
  SET checking = checking – 20
  WHERE id = 123
- UPDATE Accounts
  SET saving = saving + 20
  Where id = 123

- SELECT (checking + saving)
  FROM Accounts
  WHERE id = 123

How does the DB do this?



- Using locks and crash recovery



Summary

- Notion of transaction
- Each user app must be structured as executing transactions on a database
- Multiple users can execute multiple transactions concurrently
- We want ACID
- DB system ensures this using locks and crash recovery