Database Systems I

Transaction Processing (1)

Instructor: Ouldooz Baghban Karimi

CMPT 354 - Summer 2019
Transaction

• A **transaction** is a sequence of one or more **operations** (reads or writes) which reflects a **single real-world transition**

• Examples
  • Transfer money between accounts
  • Purchase a group of products
  • Register for a class (waitlist or allocated)
Transactions (ACID)

**Atomic**
The all-or-nothing execution of transactions

**Consistent**
Transactions are expected to preserve the consistency and integrity of the database

**Isolated**
Transactions to be executed as if no other transaction is executing at the same time

**Durable**
Once a transaction has committed, its effects remain in the database
ACID: Atomic

• Transaction activities are atomic (all or nothing)
  • A transaction is something that would either occur completely or not at all

• Two possible outcomes for a transaction
  • It commits: all the changes are made
  • It aborts: no changes are made
Example

Accounts(acctNo, balance)

• Goal: Transferring $100 from the account numbered 123 to the account 456

  Step(1)
  UPDATE Accounts
  SET balance = balance + 100 WHERE acctNo = 456;

  Step(2)
  UPDATE Accounts
  SET balance = balance - 100 WHERE acctNo = 123;

• What happens if there is a failure after Step(1) but before Step(2)?
ACID: Consistent

• The tables must always satisfy user-specified constraints
  • Examples
    • Account number is unique
    • Stock amount cannot be negative
    • Sum of debits and of credits is 0

• How consistency is achieved?
  • Programmer makes sure a transaction takes a consistent state to a consistent state
  • System makes sure that the transaction is atomic
ACID: Isolated

• A transaction executes concurrently with other transactions

• Isolation: the effect is as if each transaction executes in isolation of the others
  • E.g. Should not be able to observe changes from other transactions during the run
Example

Flights(flightNo, flightDate, seatNo, seatStatus)

SELECT seatNo
FROM Flights
WHERE flightNo = 123 AND flightDate = DATE '2008-12-25'
AND seatStatus = 'available';

UPDATE Flights
SET seatStatus = 'occupied'
WHERE flightNo = 123 AND flightDate = DATE '2008-12-25'
AND seatNo = '22A';

• What happens if more than one person has the same request at the same time?
• Serialization!
ACID: Durable

• The effect of a transaction must continue to exist (**persist**) after the transaction
  • And after the whole program has terminated
  • And even if there are power failures, crashes, etc.
  • And ...

• **Means:** Write data to **disk**
ACID

• Many debates over ACID, both historically and currently

• Many newer NoSQL DBMSs relax ACID

• In turn, now NewSQL reintroduces ACID compliance to NoSQL-style DBMSs
Transaction Management

**Definition**

Transaction Management involves managing transactions and their interactions with storage. A transaction is a list of writes and reads. For example, {Read, Read, Write}.

**Two Big Problems**

1. Support multiple transaction at the same time
2. Make sure the data stored is reliable

**Techniques**

1. Concurrency Control
2. Database Recovery

**Properties**

Atomicity, Consistency, Isolation, Durability
Concurrency Control

• The DBMS must handle concurrency such that
  • **Isolation** is maintained
    • Users must be able to execute each transaction *as if they were the only user*
    • DBMS handles the details of interleaving various transactions

• **Consistency** is maintained
  • Transactions must leave the DB in a **consistent state**
  • DBMS handles the details of enforcing integrity constraints
Interleaving

• Need to swap the control of transaction execution between multiple simultaneous transaction
  • Example
    • Execute few instructions from Transaction 1 and few instructions from Transaction 2
    • Again few more instructions from Transaction 1 and Transaction 2 and so on

• This is done for many simultaneous transactions

• This action is called as interleaving of transactions
Interleaving

• Interleaving transactions might lead to anomalous outcomes
  • Why do we do it then?

• Several important reasons: All concern large differences in performance
  • Individual transactions might be slow
    • Avoid blocking other users during slow transactions
  • Transactions waiting for locks
    • Avoid blocking other users while this transaction is also waiting
  • Disk access may be slow
    • Let some transactions use CPUs while others accessing disk
Scheduling

• A **serial schedule** is one that does not interleave the actions of different transactions

• A **serializable schedule** is a schedule that is equivalent to some serial schedule

• Two schedules are **equivalent** if, for any database state, the effect of executing them on database is **identical**
Acknowledgements

I have used materials from the following resources in preparation of this course:

- **Database Systems: The Complete Book**
- Database Systems (Kiefer, Bernstein, Lewis)
- Course offerings
  - W 4111 (Eugene Wu - Columbia): [https://w4111.github.io/](https://w4111.github.io/)
  - CS 186 (Joe Hellerstein - Berkeley): [https://sites.google.com/site/cs186fall17/](https://sites.google.com/site/cs186fall17/)