CS157A: Introduction to Database Management Systems

Transactions

Suneuy Kim

Motivations

- Serializablity of concurrent access to database
  - A DBMS typically allows many different transactions to access the database. This may result in data inconsistency.
  - Multiusers, multi-programming, parallel processing
- Recoverability from failures in DBMS
  - Media failure (e.g. faulty hard drive)
  - System failure (e.g. power outage)

Terminology

- Transaction
  - An executing program that forms a logical unit of database processing.
  - It consists of one or more database operations such as insert, delete, update , and select.
- Transaction processing system
  - Systems with large databases and hundreds of concurrent users executing database transactions.

Database Operation Details

- Read_item(X) reads a database item named X
  - Find the address of the disk block that contains X
  - Copy that disk block into a buffer
  - Copy X from the buffer to the program
- Write_item(X) writes a database item X into the database
  - Find the address of the disk block that contains X
  - Copy that disk block into a buffer in main memory
  - Copy item X into the correct location in buffer.
  - Store the updated block from the buffer back to disk.

Transaction Pseudocode

```
Start TRANSACTION

database operations here !

IF no error THEN COMMIT
ELSE ROLLBACK
```

COMMIT vs ROLLBACK

- COMMIT – Successful end of a transaction

  Changes made by database operations are installed permanently in the database.
- ROLLBACK – Abnormal end of a transaction

  Any changes made by database operations are undone.

Transaction ACID properties

- Atomicity

  Transactions are atomic (all or nothing).

- Consistency

  Transaction transforms the DB from one valid state to another valid state.

- Isolation

  Transactions are isolated from each other.

- Durability

  Once a transaction commits, it remains so even in the event of system or media failures.

Atomicity

To transfer money from 123 to 456.

```
UPDATE Accounts
SET balance = balance – 100
WHERE acctNo = 123;

UPDATE Accounts
SET balance = balance + 100
WHERE acctNo = 456;
```

Atomicity requires that both of the steps, or neither, be completed: all or none

Consistency

Any data written to the database must be valid according to all defined rules, including but not limited to constraints and triggers, and any combination thereof.

Examples:

- Columns only store values of a particular type (int columns store only integers, etc…)
- Primary keys and unique keys are unique
- Check constraints are satisfied
- Foreign key constraints are satisfied
- In an application that transfers funds from one account to another, the consistency property ensures that the total value of funds in both the accounts is the same at the start and end of each transaction.

Isolation

- Deals with behavior of a transaction with respect to other concurrent transactions.
- Ensures **serializability** of concurrent execution of transactions - Operations may be interleaved, but execution must be equivalent to *some* serial order of all transactions.
- Providing isolation is the main goal of concurrency control

Isolation

```
SELECT seatNo
FROM Flights
WHERE fltNo = 123 AND
fltDate = DATE '2013-12-25'
AND seatStatus  = 'available';
```

```
UPDATE Flights
SET seatStatus = 'occupied'
WHERE fltNo = 123 AND
fltDate = DATE '2013-12-25'
AND seatNO  = '22A';
```


User 1 finds seat(22A) empty

User 2 finds seat empty

User 1 sets seat 22A occupied

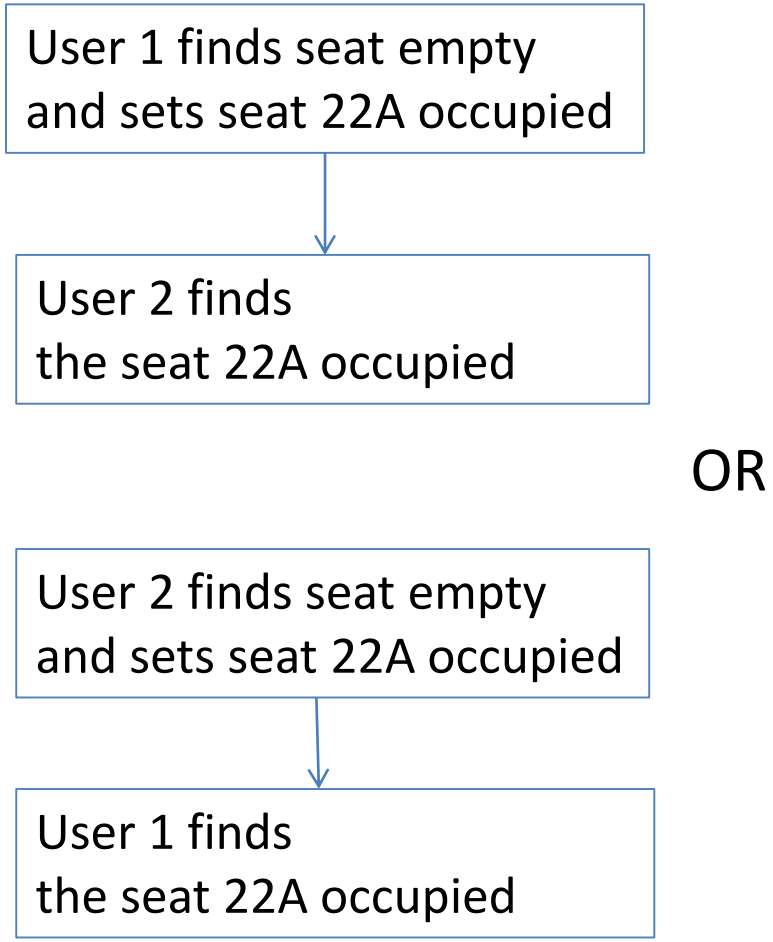
User2 sets seat 22A occupied

time

Isolation

```
Lock flight;
SELECT @seat = min(seatNo)
FROM flights
WHERE fltNo = 123 AND
fltDate = DATE '2013-12-25'
AND seatStatus  = 'available';

UPDATE Flights
SET seatStatus = 'occupied'
WHERE fltNo = 123 AND
fltDate = DATE '2013-12-25'
AND seatNO  = @seat;
Unlock flight;
```



Durability

- Guarantees that transactions that have committed will survive any subsequent mal-functions.
- Example: If a flight booking reports that a seat has successfully been booked, then the seat will remain booked even if the system crashes.

Durability

- Write-Ahead Transaction Log: First write changes to a transaction log and then write the changes to the database.

Durability

Media failure (e.g. a faulty disk drive)

- Data loss
- Databases are recovered by using backups and transaction logs.
- A DBA has to
  - Take backups regularly
  - Keep your transaction logs and your main database files on different hard disks
  - Backup the tail of the log (the log that has not been backup)

Durability

System failures (e.g. system crashes, power outages)

- Half-performed transactions that were interrupted and not yet committed – roll back using the transaction log
- Committed transactions may not have their changes written to disk – if the conditions are right and there's enough info in the transaction log, replay them

Read Phenomena

Users

id	name	age
1	Joe	20
2	Jill	25

Suppose T1 is a reader and T2 is a writer.

T1:
SELECT age FROM Users WHERE id = 1; // Q1
SELECT age FROM Users WHERE id = 1; // Q1 again

T2:
UPDATE User SET age = 21 WHERE id = 1; // Q2

Read Phenomena

- Dirty reads
- Non-repeatable reads
- Phantom reads

Examples are from Wikipedia

Read Phenomena: Dirty Reads

- A dirty read occurs when a transaction is allowed to read data from a row that has been modified by another running transaction and not yet committed.

Read Phenomena: Dirty reads

T1

T2

SELECT age FROM Users
WHERE id = 1 ; // 20

UPDATE User SET age = 21
WHERE id = 1; // 21

SELECT age FROM Users
WHERE id = 1; //21

Rollback

A user with id = 1 and age = 21 does not exist !

Read phenomena: Non-repeatable reads

- A non-repeatable read occurs, when during the course of a transaction, a row is retrieved twice and the values within the row differ between reads.
- Sometimes non-repeatable reads might be completely desirable. Some applications may want to know the absolute, real-time value, whereas other types of transactions might need to read the same value multiple times. → Set the isolation level according to the need of application.

Read phenomena: Non-repeatable reads

T1

```
SELECT age FROM Users
WHERE id = 1 ; // 20
```

T2

```
UPDATE User SET age = 21
WHERE id = 1;  // 21
COMMIT;
```

T1

```
SELECT age FROM Users
WHERE id = 1; //21
```

Transaction 1 has already seen a different value for *age* in that row !

Read phenomena: Phantom reads

- A phantom read occurs when, in the course of a transaction, two identical queries are executed, and the collection of rows returned by the second query is different from the first.

Read phenomena: Phantom reads

T1

```
SELECT age FROM Users
WHERE age >10 and age < 30;
```

T2

```
INSERT INTO Users
VALUES  (3, 'BOB', 27);
```

```
SELECT age FROM Users
WHERE age >10 and age < 30;
```

Transaction 1 gets a different set of rows for the second time.

Transaction Isolation Levels

- Controls the degree of locking that occurs when selecting data.
- Isolation Levels
  - Read uncommitted (the lowest)
  - Read committed
  - Repeatable reads
  - Serializable (the highest)

Transaction Isolation Levels

- A higher isolation level lowers the risk of concurrency problems but also lowers the average performance due to locking overhead and loss of parallelism.
- With a relaxed isolation level – application programmer must ensure not to cause any software bugs

Isolation level: Read uncommitted

- In this level, dirty reads are allowed so one transaction may see not-yet-committed changes made by other transactions.
- Non-repeatable reads and phantom reads are allowed.

Isolation level: Read committed

- Keeps write locks (acquired on selected data) until the end of the transaction – forbids reading uncommitted data)
- But, read locks are released as soon as the SELECT operation is performed (so the non-repeatable reads phenomenon can occur.)
- Range-locks are not managed.
- No dirty reads
- Non-repeatable reads and phantom reads are allowed.

Isolation level: Repeatable reads

- Keeps read and write locks (acquired on selected data) until the end of the transaction.
- However, range-locks are not managed, so the phantom reads phenomenon can occur.
- No dirty reads, no unrepeatable reads
- Phantom reads are allowed.

Isolation level: Serializable

- Keeps read and write locks (acquired on selected data) until the end of the transaction.
- Also *range-locks* must be acquired when a SELECT query uses a ranged *WHERE* clause, especially to avoid the *phantom reads* phenomenon.
- No dirty reads, no unrepeatable reads, no phantom reads

Read Only vs. Read Write

- READ WRITE is a default assumption.
- Exception

  READ ONLY is the default with ISOLATION LEVEL READ UNCOMMITTED, and therefore, if your transaction is not-read only and at the level of READ UNCOMMITTED, you must explicitly SET TRANSACTION READ WRITE.

Isolation Levels vs. Read Phenomena

	Dirty Reads	Non-repeatable Reads	Phantom Reads
Read Uncommitted	Allowed	Allowed	Allowed
Read Committed	Not Allowed	Allowed	Allowed
Repeatable Read	Not Allowed	Not Allowed	Allowed
Serializable	Not Allowed	Not Allowed	Not Allowed

MySQL: Transactions

- START TRANSACTION ; // disable auto-commit until commit or rollback
- COMMIT
- ROLLBACK
- Note:
  - BEGIN and BEGIN WORK are aliases for START TRANSACTION which standard syntax.
  - BEGIN and BEGIN…END are different: The first one is to initiate a transaction and the second one is to form a compound statement.

Transaction and Stored Procedure

```
Drop table if exists t1;
CREATE TABLE T1(A INT PRIMARY KEY, B INT) ;
INSERT INTO T1 VALUES (1,2), (3,4);

Drop procedure if exists CommitTest;
delimiter //
create procedure CommitTest()
begin
delete from t1 where a = 1;end; //
delimiter ;

START TRANSACTION;
delete from t1 where a = 3;
CALL CommitTest;
rollback;

select * from t1; # (1,2), (3,4)
```

Transaction and Trigger

```
Drop table if exists t1;
CREATE TABLE T1(A INT PRIMARY KEY, B INT) ;
INSERT INTO T1 VALUES (1,2), (3,4);
Drop table if exists t2;
CREATE TABLE T2(X INT PRIMARY KEY, Y INT) ;

drop trigger if exists deleteFromTransaction;
delimiter //
create trigger deleteFromTransaction
after delete on T1
for each row
begin insert into t2 values (Old.A,Old.B);end; //delimiter ;

START TRANSACTION;  delete from t1 where a = 3;
rollback;
select * from t1; # (1,2), (3,4)
select * from t2; # empty
```