# CS 235: Introduction to Databases

Svetlozar Nestorov

*Lecture Notes #19*

## Transaction Management

- Manage many queries/updates running simultaneously.
  - Airline reservations, auctions, ATMs.
- Atomicity – all or nothing principle.
- Serializability – the effect of transactions as if they occurred one at a time.

## Transaction Control

- Items – units of data to be controlled:
  - fine-grained – small items, e.g. tuples.
  - coarse-grained – large items, e.g. relations.
- Controlling access by locks.
  - Read – sharable with other readers.
  - Write – not sharable with anyone else.
- Model – (item, locktype, transactionID).

## Transactions

- A transaction is a unit of work that must be:

1. *Atomic* = either all work is done, or none of it.
2. *Consistent* = relationships among values maintained.
3. *Isolated* = appear to have been executed when no other DB operations were being performed.
   - Often called *serializable* behavior.
4. *Durable* = effects are permanent even if system crashes.

## Commit or Abort

- Each transaction ends with either:

1. *Commit* = the work of the transaction is installed in the database; previously its changes may be invisible to other transactions.
2. *Abort* = no changes by the transaction appear in the database; it is as if the transaction never occurred.
   - ROLLBACK is the term used in SQL and MySQL.

## Transaction Boundaries

- In the ad-hoc query interface (*e.g.*, mysql client), every query or modification statement is a transaction.
- You can disable this mode by:
  SET AUTOCOMMIT = 0;
  - A COMMIT or ROLLBACK ends the previous transaction and starts a new one. Exiting mysql forces implicit COMMIT.
- You can also start transaction explicitly:
  START TRANSACTION;
- Transactions work with InnoDB tables but not with MyISAM tables.

## Example

- Spoon sells Bud for $2.50 and Miller for $3.00.
- Sally is querying the database for the highest and lowest price Spoon charges:
- At the same time, Spoon has decided to replace Miller and Bud by Heineken at $3.50:
- Can Sally find that the cheapest beer sold at Spoon is more expensive than the most expensive one?!
- Fix the problem by grouping Sally's two statements into one transaction, *e.g.*, with one SQL statement.

## Problem With Rollback

- Suppose Spoon inserts Heineken, but then, during the transaction thinks better of it and issues a ROLLBACK statement.
- If Sally is allowed to execute finding the max price just before the rollback, she gets the answer $3.50, even though Spoon doesn't sell any beer for $3.50.
- Fix by making the insert a transaction, or part of a transaction, so its effects cannot be seen by Sally unless there is a COMMIT action.

## SQL Isolation Levels

- *Isolation levels* determine what a transaction is allowed to see. The declaration, valid for one transaction, is:
  SET TRANSACTION ISOLATION LEVEL *X*;
- X can be:
  - SERIALIZABLE
  - READ COMMITTED
  - REPEATABLE READ
  - READ UNCOMMITTED
- In MySQL (with InnoDB) REPEATABLE READ is the default.

## Serializable Example

- The transaction must execute as if at a point in time, where all other transactions occurred either completely before or completely after.
- Sally's queries are one transaction and Spoon updates are another transaction. If Sally's transaction runs at isolation level SERIALIZABLE, she would see the Sells relation either before or after the updates ran, but not in the middle.

## Read-Committed Example

- The transaction can read only committed data.
- If transactions are as before, Sally could see the original Sells for statement 1 and the completely changed Sells for statement 2.

## Repeatable-Read Example

- If a transaction reads data twice, then what it saw the first time, it will see the second time (it may see more the second time).
- If find max is executed before, then it must see the Bud and Miller tuples when it computes the min, even if it executes after their deletion. But if find max executes between the deletion and insertion, then find min may see the Heineken tuple.

## Read-Uncommitted Example

- No constraint, even on reading data written and then removed by a rollback.
- Sally's two queries could see Heineken, even if Spoon rolled back the transaction.

## Another Example

| T1 | T2 | start with A = 5 | | |
|---|---|---|---|---|
| Read A | | A on disk | A in T1 | A in T2 |
| | Read A | 5 | 5 | 5 |
| A:= A + 1 | | | | |
| | A:= 2* A | | | |
| | Write A | | | |
| Write A | | | | |

## Locks

RLOCK A
WLOCK A
UNLOCK A

| T2 | T1: NO | T1: R | T1: W |
|---|---|---|---|
| NO | OK | OK | OK |
| R | OK | OK | bad |
| W | OK | bad | bad |

RLOCK –> UNLOCK can enclose a read
WLOCK –> UNLOCK can enclose a write or read

## Example with Locks

| T1 | T2 |
|---|---|
| WLOCK A | |
| Read A | |
| | WLOCK A |
| A:= A+1 | waits |
| Write A | |
| UNLOCK A | |
| | granted |
| | Read A |
| | A:=2*A |
| | Write A |
| | UNLOCK A |

## Deadlock!

| T1 | T2 | |
|---|---|---|
| RLOCK A | | |
| Read A | | |
| | RLOCK A | |
| | Read A | |
| A:= A+1 | | |
| | A:= 2*A | |
| | WLOCK A | upgrade lock request |
| WLOCK A | | upgrade lock request |
| wait | wait | |
| Deadlock! | | |

## Another Deadlock

| T1 | T2 | |
|---|---|---|
| WLOCK A | | |
| | WLOCK B | |
| WLOCK B | | |
| wait | WLOCK A | |
| UNLOCK A | wait | deadlock! |
| | UNLOCK B | |
| UNLOCK B | | |
| | UNLOCK A | |

## Deadlock Conditions

1. Hold some locks while you wait for others.
2. Circular chain of waiters wait-for graph.
3. No pre-emption.

- We can avoid deadlock by doing at least ONE of:

1. Get all your locks at once
2. Apply an ordering to acquiring locks
3. Allow preemption (for example, use timeout on waits)

## Authorization in SQL

- File systems identify certain access privileges on files, *e.g.*, read, write, execute.
- In partial analogy, SQL identifies six access privileges on relations, of which the most important are:

1. SELECT = the right to query the relation.

## More Privileges

2. INSERT: the right to insert tuples into the relation – may refer to one attribute, in which case the privilege is to specify only one column of the inserted tuple.
   - MySQL does not support attribute-level privileges.
3. DELETE: the right to delete tuples from the relation.
4. UPDATE: the right to update tuples of the relation – may refer to one attribute.

## Granting Privileges

- You have all possible privileges to the relations you create.
- You may grant privileges to any user if you have those privileges "with grant option."
  - You have this option to your own relations.

## Example

1. Here, Sally can query and update Sells, but cannot pass on this power:

```
GRANT SELECT ON Sells,
UPDATE  ON Sells
TO sally;
```

2. Here, Sally can also pass these privileges to whom she chooses:

```
GRANT SELECT ON Sells,
UPDATE ON Sells
TO sally
WITH GRANT OPTION;
```

## Revoking Privileges

- Your privileges can be revoked.
- Syntax is like granting, but REVOKE ... FROM instead of GRANT ... TO.
- Determining whether or not you have a privilege is tricky, involving "grant diagrams" as in text. However, the basic principles are:
  - If you have been given a privilege by several different people, then all of them have to revoke in order for you to lose the privilege.
  - Revocation is transitive. if *A* granted *P* to *B*, who then granted *P* to *C*, and then *A* revokes *P* from *B*, it is as if *B* also revoked *P* from *C*.