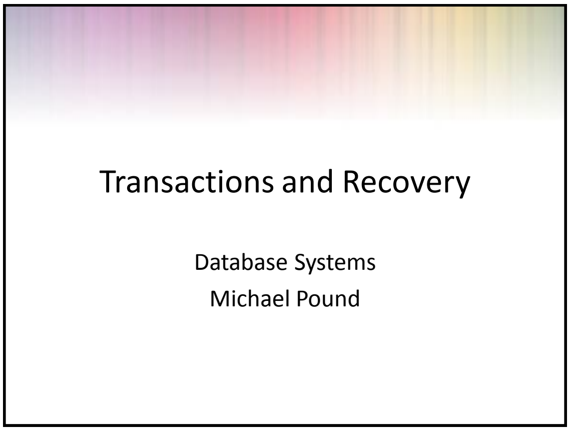
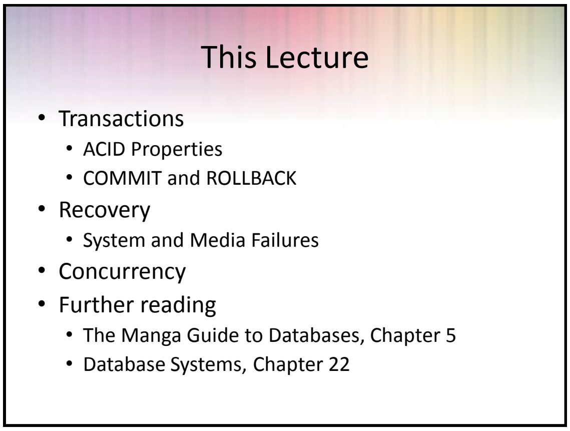


Transactions and Recovery

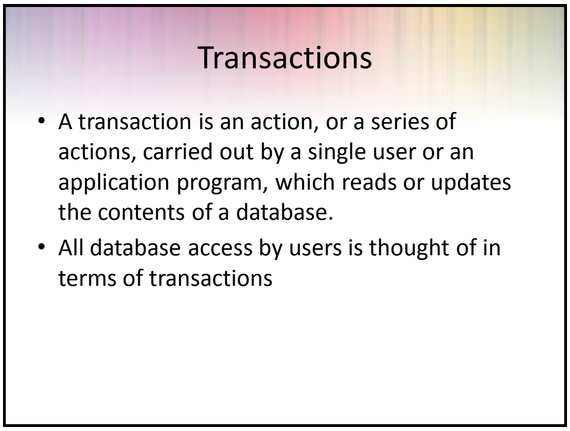
Database Systems

Michael Pound



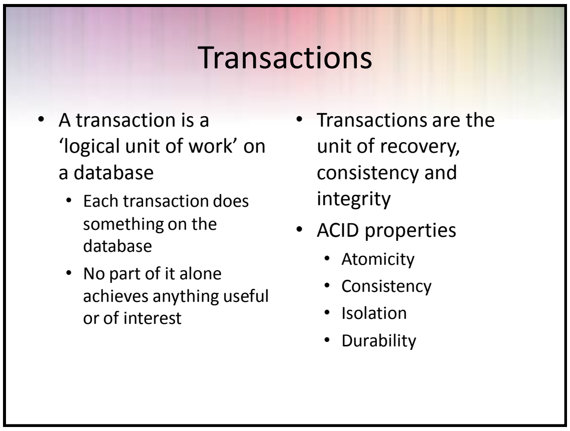
This Lecture

- Transactions
  - ACID Properties
  - COMMIT and ROLLBACK
- Recovery
  - System and Media Failures
- Concurrency
- Further reading
  - The Manga Guide to Databases, Chapter 5
  - Database Systems, Chapter 22



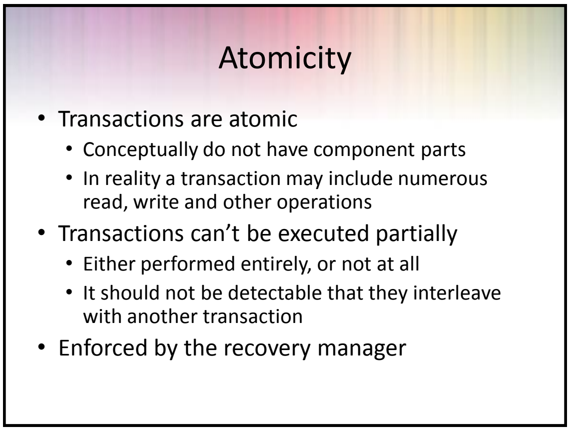
Transactions

- A transaction is an action, or a series of actions, carried out by a single user or an application program, which reads or updates the contents of a database.
- All database access by users is thought of in terms of transactions



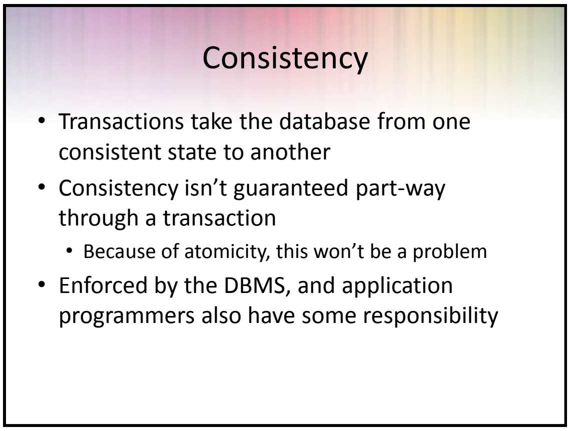
Transactions

- A transaction is a 'logical unit of work' on a database
  - Each transaction does something on the database
  - No part of it alone achieves anything useful or of interest
- Transactions are the unit of recovery, consistency and integrity
- ACID properties
  - Atomicity
  - Consistency
  - Isolation
  - Durability



Atomicity

- Transactions are atomic
  - Conceptually do not have component parts
  - In reality a transaction may include numerous read, write and other operations
- Transactions can't be executed partially
  - Either performed entirely, or not at all
  - It should not be detectable that they interleave with another transaction
- Enforced by the recovery manager



Consistency

- Transactions take the database from one consistent state to another
- Consistency isn't guaranteed part-way through a transaction
  - Because of atomicity, this won't be a problem
- Enforced by the DBMS, and application programmers also have some responsibility

Isolation

- All transactions execute independently of one another
- The effects of a transaction are invisible to other transactions until it has been completed
- Enforced by the scheduler

Durability

- Once a transaction has completed, it's changes are made permanent
- If the database system crashes, completed transactions must remain complete
- Enforced by the recovery manager

Transaction Example

- Transfer £50 from bank account A to account B

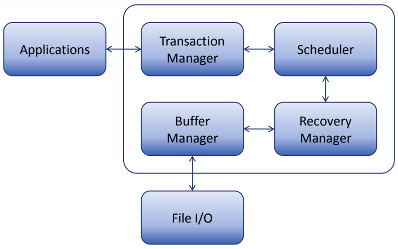
$$\left.\begin{array}{l}\text{Read}(A)\\ A = A - 50\\ \text{Write}(A)\\ \text{Read}(B)\\ B = B + 50\\ \text{Write}(B)\end{array}\right\}\text{Transaction}$$

- **Atomicity** – Shouldn't take money from A without giving it to B
- **Consistency** – Money isn't lost or gained overall
- **Isolation** – Other queries shouldn't see A or B change until completion
- **Durability** – The money does not return to A, even after a system crash

Transaction Subsystem

- The transaction subsystem enforces the ACID properties
  - Schedules the operations of all transactions
  - Uses COMMIT and ROLLBACK to ensure atomicity
  - Locks and/or timestamps are used to ensure consistency and isolation (next lectures)
  - A log is kept to ensure durability

Transaction Subsystem



Database Systems, Connolly & Begg, p574

COMMIT and ROLLBACK

- COMMIT is used to signal the successful end of a transaction
  - Any changes that have been made to the database should be made permanent
  - These changes are now available to other transactions
- ROLLBACK is used to signal the unsuccessful end of a transaction
  - Any changes that have been made to the database should be undone
  - It is now as if the transaction never happened, it can now be reattempted if necessary

Recovery

- Transactions must be durable, but some failures will be unavoidable
  - System crashes
  - Power failures
  - Disk crashes
  - User mistakes
  - Sabotage
  - etc
- Prevention is better than a cure
  - Reliable OS
  - Security
  - UPS and surge protectors
  - RAID arrays
- Can't protect against everything, system recovery will be necessary

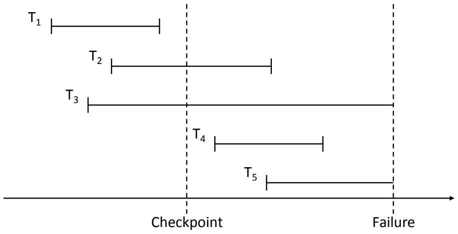
The Transaction Log

- The transaction log records details of all transactions
  - Any changes the transaction makes to the database
  - How to undo these changes
  - When transactions complete and how
- The log is stored on disk, not in memory
  - If the system crashes, the log is preserved
- Write ahead log rule
  - The entry in the log must be made before COMMIT processing can complete

System Failures

- A system failure effects all running transactions
  - Software crash
  - Power failure
- The physical media (disks) are not damaged
- At various times a DBMS takes a checkpoint
  - All transactions are written to disk
  - A record is made (on disk) of all transactions that are currently running

Transaction Timeline

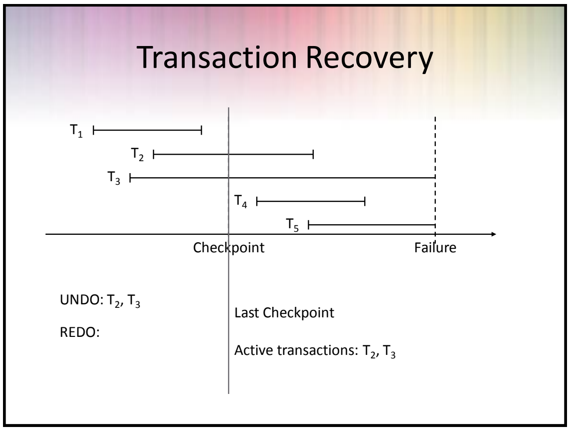


System Recovery

- Any transaction that was running at the time of failure needs to be undone and possibly restarted
- Any transactions that committed since the last checkpoint need to be redone
- Transactions of type T_1 need no recovery
- Transactions of type T_3 or T_5 need to be undone
- Transactions of type T_2 or T_4 need to be redone

Transaction Recovery

- Create two lists of transactions: UNDO and REDO
  - UNDO – all transactions running at the last checkpoint
  - REDO – empty
- For every entry in the log since the last checkpoint, until the failure:
  1. If a BEGIN TRANSACTION entry is found for T, Add T to UNDO
  2. If a COMMIT entry is found for T, Move T From UNDO to REDO



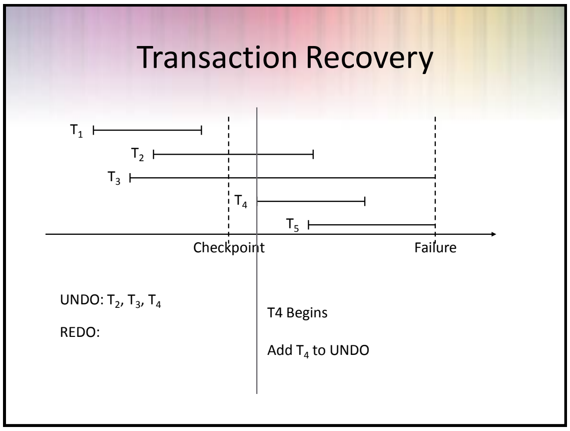
Transaction Recovery

UNDO: T_2, T_3

REDO:

Last Checkpoint

Active transactions: T_2, T_3



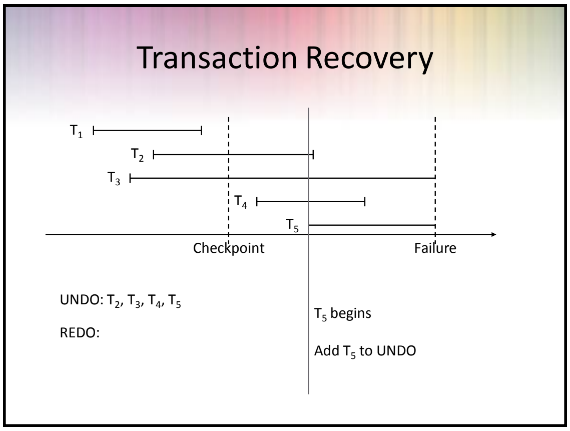
Transaction Recovery

UNDO: T_2, T_3, T_4

REDO:

T4 Begins

Add T_4 to UNDO



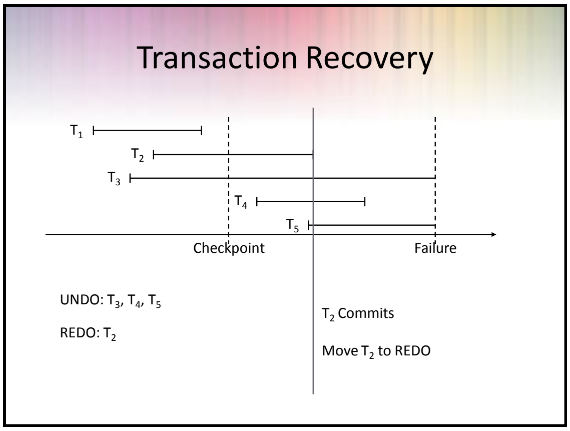
Transaction Recovery

UNDO: T_2, T_3, T_4, T_5

REDO:

T_5 begins

Add T_5 to UNDO



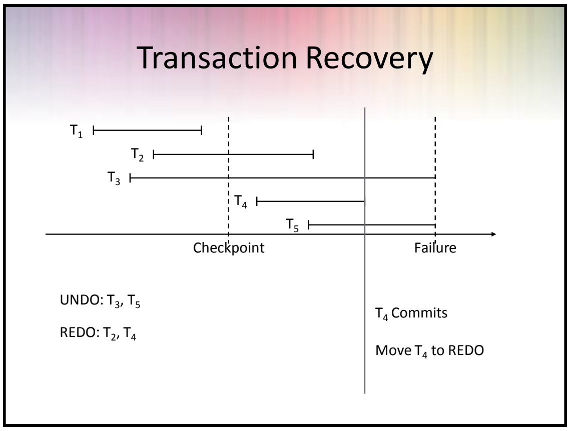
Transaction Recovery

UNDO: T_3, T_4, T_5

REDO: T_2

T_2 Commits

Move T_2 to REDO



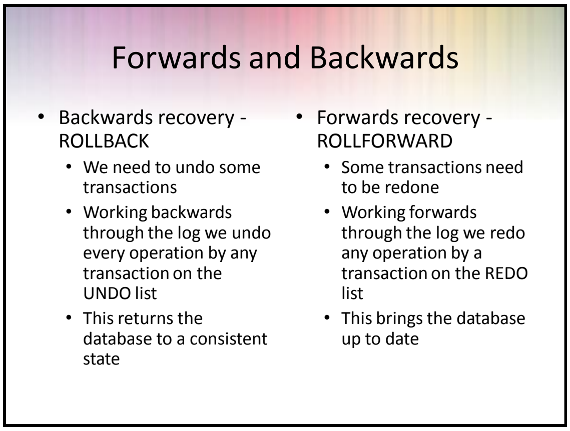
Transaction Recovery

UNDO: T_3, T_5

REDO: T_2, T_4

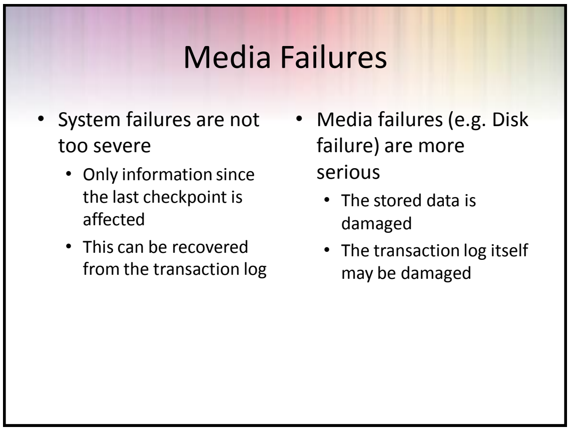
T_4 Commits

Move T_4 to REDO



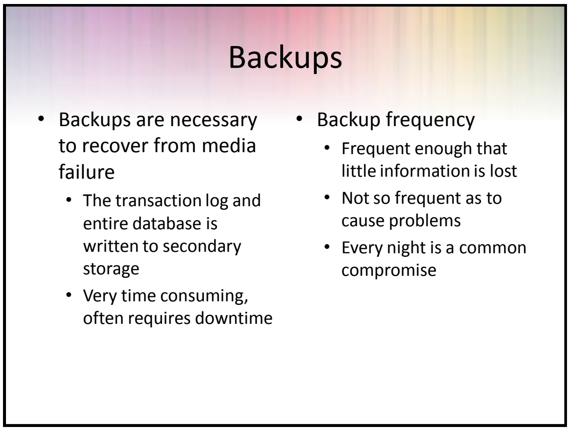
Forwards and Backwards

- Backwards recovery - ROLLBACK
  - We need to undo some transactions
  - Working backwards through the log we undo every operation by any transaction on the UNDO list
  - This returns the database to a consistent state
- Forwards recovery - ROLLFORWARD
  - Some transactions need to be redone
  - Working forwards through the log we redo any operation by a transaction on the REDO list
  - This brings the database up to date



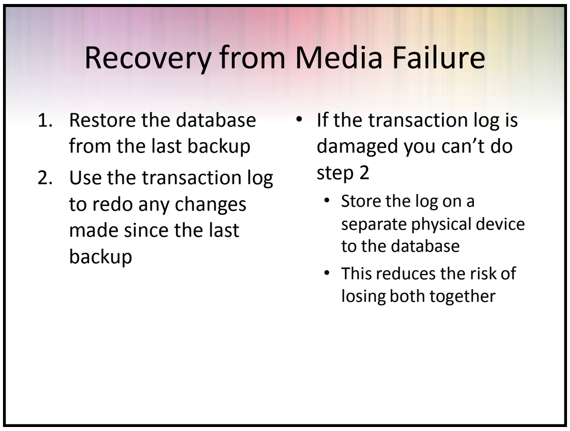
Media Failures

- System failures are not too severe
  - Only information since the last checkpoint is affected
  - This can be recovered from the transaction log
- Media failures (e.g. Disk failure) are more serious
  - The stored data is damaged
  - The transaction log itself may be damaged



Backups

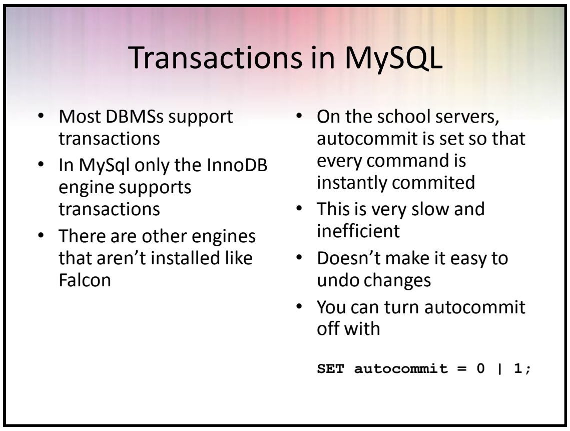
- Backups are necessary to recover from media failure
  - The transaction log and entire database is written to secondary storage
  - Very time consuming, often requires downtime
- Backup frequency
  - Frequent enough that little information is lost
  - Not so frequent as to cause problems
  - Every night is a common compromise



Recovery from Media Failure

1. Restore the database from the last backup
2. Use the transaction log to redo any changes made since the last backup

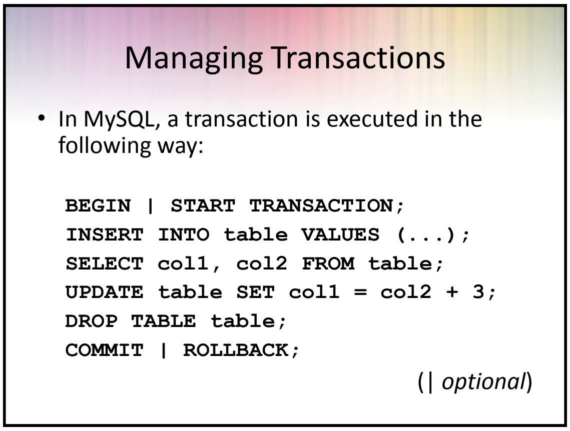
- If the transaction log is damaged you can't do step 2
  - Store the log on a separate physical device to the database
  - This reduces the risk of losing both together



Transactions in MySQL

- Most DBMSs support transactions
- In MySql only the InnoDB engine supports transactions
- There are other engines that aren't installed like Falcon
- On the school servers, autocommit is set so that every command is instantly commited
- This is very slow and inefficient
- Doesn't make it easy to undo changes
- You can turn autocommit off with

```
SET autocommit = 0 | 1;
```

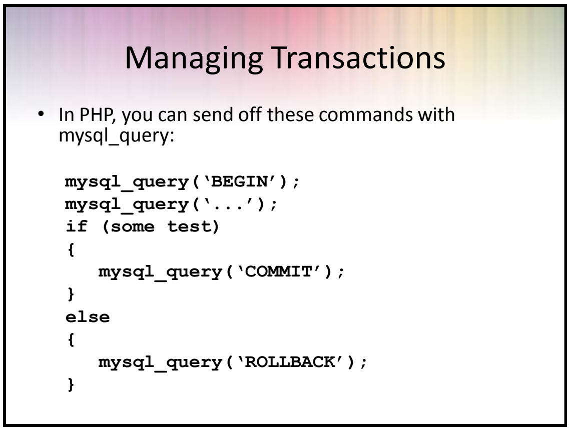


Managing Transactions

- In MySQL, a transaction is executed in the following way:

```
BEGIN | START TRANSACTION;
INSERT INTO table VALUES (...);
SELECT col1, col2 FROM table;
UPDATE table SET col1 = col2 + 3;
DROP TABLE table;
COMMIT | ROLLBACK;
```

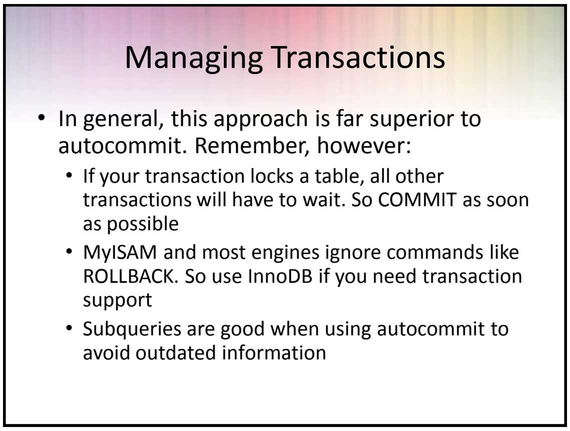
(| *optional*)



Managing Transactions

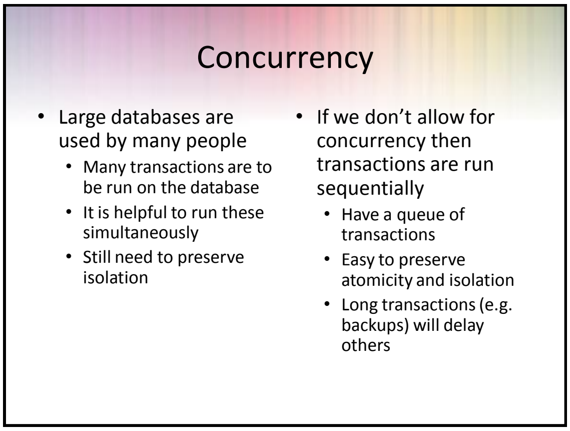
- In PHP, you can send off these commands with mysql_query:

```
mysql_query('BEGIN');
mysql_query('...');
if (some test)
{
   mysql_query('COMMIT');
}
else
{
   mysql_query('ROLLBACK');
}
```



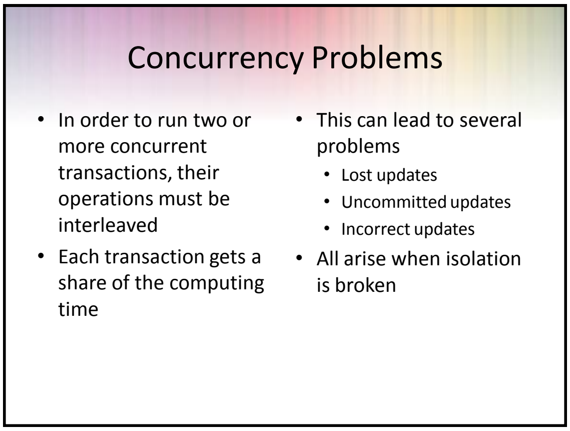
Managing Transactions

- In general, this approach is far superior to autocommit. Remember, however:
  - If your transaction locks a table, all other transactions will have to wait. So COMMIT as soon as possible
  - MyISAM and most engines ignore commands like ROLLBACK. So use InnoDB if you need transaction support
  - Subqueries are good when using autocommit to avoid outdated information



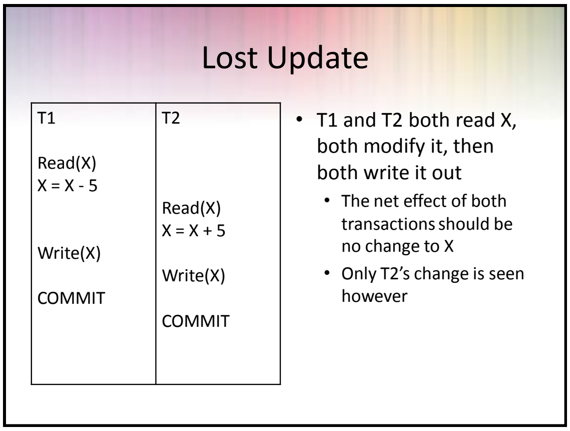
Concurrency

- Large databases are used by many people
  - Many transactions are to be run on the database
  - It is helpful to run these simultaneously
  - Still need to preserve isolation
- If we don't allow for concurrency then transactions are run sequentially
  - Have a queue of transactions
  - Easy to preserve atomicity and isolation
  - Long transactions (e.g. backups) will delay others



Concurrency Problems

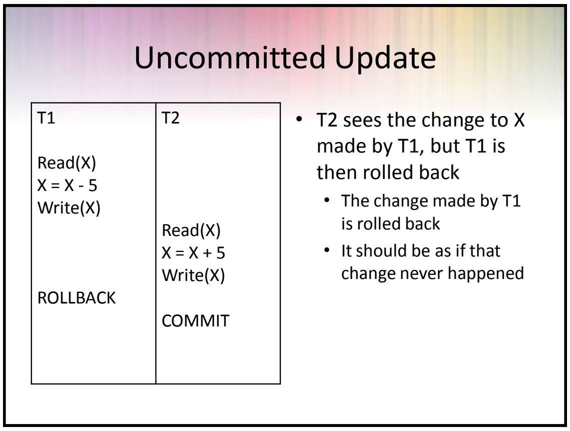
- In order to run two or more concurrent transactions, their operations must be interleaved
- Each transaction gets a share of the computing time
- This can lead to several problems
  - Lost updates
  - Uncommitted updates
  - Incorrect updates
- All arise when isolation is broken



Lost Update

T1	T2
Read(X)	
X = X - 5	
	Read(X)
	X = X + 5
Write(X)	
	Write(X)
COMMIT	
	COMMIT

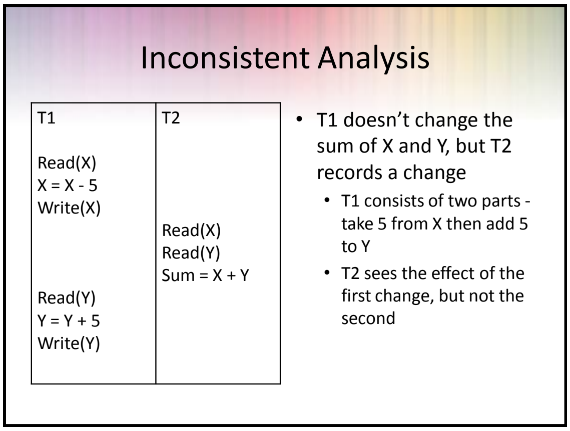
- T1 and T2 both read X, both modify it, then both write it out
  - The net effect of both transactions should be no change to X
  - Only T2's change is seen however



Uncommitted Update

T1	T2
Read(X)	
X = X - 5	
Write(X)	
	Read(X)
	X = X + 5
	Write(X)
ROLLBACK	
	COMMIT

- T2 sees the change to X made by T1, but T1 is then rolled back
  - The change made by T1 is rolled back
  - It should be as if that change never happened



Inconsistent Analysis

T1	T2
Read(X)	
X = X - 5	
Write(X)	
	Read(X)
	Read(Y)
	Sum = X + Y
Read(Y)	
Y = Y + 5	
Write(Y)	

- T1 doesn't change the sum of X and Y, but T2 records a change
  - T1 consists of two parts - take 5 from X then add 5 to Y
  - T2 sees the effect of the first change, but not the second

This Lecture in Exams

Define a transaction in the context of database management

Explain how a DBMS uses a transaction log to recover from a system failure using ROLLBACK and ROLLFORWARD

Explain the difference between a system failure and a media failure

Next Lecture

- Concurrency
  - Locks and Resources
  - Deadlock
- Serialisability
  - Schedules of transactions
  - Serial and serialisable schedules
- Further reading
  - The Manga Guide to Databases, Chapter 5
  - Database Systems, Chapter 22