**Transaction**: updates must be entirely completed, or not at all.  
so as to keep data integrity  
  
reach a: commit | rollback | end of program |program absnormally terminated

A transaction log is beneficial so as to not lose consistency.   
each transaction component (each sql execution)  
-type operation (update, delete, insert)  
- Name of objects affected (entities – attributes)  
- Before and After values for updated fields  
- pointers to previous and next transaction log entries for the same transaction  
Lase execution/pointer is the COMMIT/ROLLBACK transaction  
**Disadvantage:** extra processing power required  
  
transaction(ID,Num,Prev\_Pointer,Next\_Pointer,operation, table, rowID, attribute, beforeVal, AfterVal)  
  
PrevPtr: Null… 1,2,3  
NextPtr: 1,2,3… Null  
Operation: Start … Update or delete or insert … commit or rollback  
Table: Start transaction … TableName … End transaction  
TransactionNum is automatically assigned by DBMS  
  
  
**Database Recovery**restores data from a given state, usually inconsistent, to a previous state.  
levels of backup:  
-full backup: entire database  
-differential backup: only last modifications of database are copied  
-transaction log backup: only transaction log operations that are not reflected in a previous backup copy of the database are copied.  
ie saves the initial state of DB, and the transactions  
  
Database failures: Software, hardware, programming exemptions, Transactions, external factors   
  
**We need to know: what is a transaction log, how do we use a transaction log, identify, why use, levels of backup, how to recover from failure, potential risks and how to recover from.**

**Concurrency control (will be in exam)**  
  
A method to ensure that serilizability of transactions in a multiuser DB environment  
-ensures convrrent execution of transactions yields consistent results  
  
Several problems that can occur:  
- Lost updates: slide show good example. When one transaction reads before the previous transaction has been completed  
-uncommitted Data: simultaneous transactions running: when a rollback oocurs in T1, but T2 reads data before rollback occurs  
  
How to solve the above?  
The scheduler: establishes the order of operations in which concurrent transactions are executed.  
Locking: (**NB must be able to write in exam)**Locking: Guarantees exclusive use of the data item for a current transaction  
prevents another transaction from reading inconsistent data  
Lock is **acquired** by a transaction **prior** to data access, and **released** when transaction is completed.  
  
Lock Granularity: levels of lock use.  
-database  
-table  
-page  
-row  
-field  
each lock only allows 1 transaction to occur at a time.  
PROBLEM with database lock, only 1 user can access at a time  
page: A portion of disc space… fixed size… tables can span several pages

Lock types:  
-Binary lock: is locked or not locked  
  
Lock:  
does not guarantee serializability  
schedules may create deadlocks: when 2 transactions wait for each other to unlock/lock data.  
  
2 phase Lock  
ensures serializability   
rules:  
-2 T cannot have confilicting locks  
-No unlock operation can proceed a lock operation of the same T  
-  
  
  
  
  
  
Deadlock:  
 T1 reserved by R1  
T1 wants to reserve R2  
R2 reserves T2  
T2 wants to reserve R1  
  
Deadlock prevention: T requesting new lock is **aborted** if **possibility** of deadlock can occur. Then reschedules execution  
detection **periodically tests** db for deadlocks, if found, one T is aborted  
Avoidance: T **must obtain all locks needed before it can be executed  
(WILL BE ASKED TO EXPLAIN A DEADLOCK)**  
Next 3,4 slides wont be in test, but might be in exam (concurrency control with time stamping methods)  
  
Optimistic approach: nothing will go wrong  
  
Importance: look at rec.