

Transaction Processing

CS 301

“Goal of the chapter”

“Learn about the concept of a transaction that represents a logical unit of DB processing and that must be completed in its entirety to ensure correctness.”

1. Introduce the concept needed to ensure the correct executions of transactions.
2. Discuss about concurrency control and transaction failure.
3. Present ACID properties in transaction processing systems.
4. Look at some commands that support the transaction concept in SQL.

Background

- The concept of *transaction* provides a mechanism for describing logical units of DB processing.
- *Transaction processing systems* have large DBs and hundreds of concurrent users executing DB transactions. Example are airline reservations, banking, credit card processing, online retail purchasing, etc.
- Typically, these systems require high availability and fast response time for hundreds of concurrent users.

Transaction processing

- One criterion for classifying a DB system is according to the number of users who can use the system concurrently.
- Concept of single-user (restricted to personal computer systems) and multiuser in DBMS (example: airline reservation system, banks, etc.).
- Multiple users can access DBs simultaneously because of multiprogramming, which allows the OS to execute multiple processes at the same time.
- Multiprogramming OS execute some commands from one process, then suspends that process and execute some commands from the next process, and so on.
- A process is resumed at the point where it was suspended whenever it gets its turn to use the CPU again.

Transaction processing

- Therefore, concurrent execution of processes is actually interleaved.
- Figure below shows two processes, A and B, executing concurrently in an interleaved fashion.
- Interleaving keeps the CPU busy when a process requires an input or output (I/O) operation, such as reading a block from disk.

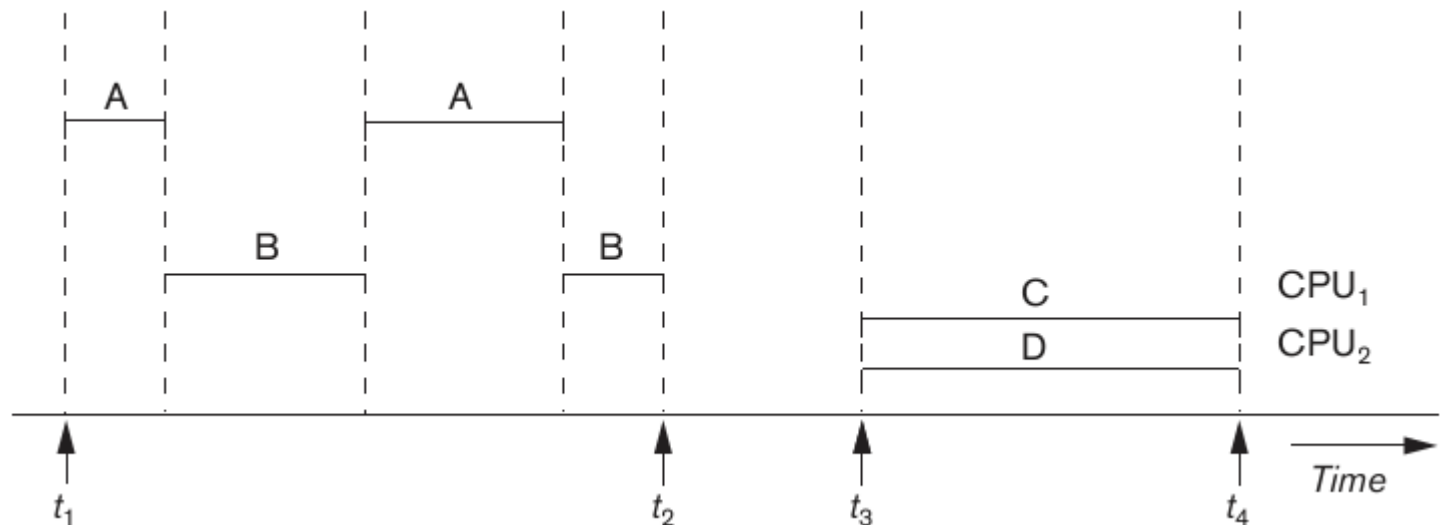


Figure: Interleaved processing (A and B) versus parallel processing (C and D) of concurrent transactions.

- The CPU is switched to execute another process rather than remaining idle during I/O time. So, interleaving also prevents a long process from delaying other processes.
 - With multiple CPUs, parallel processing of multiple processes is possible as shown by processes C and D (see figure in previous slide).
- Concurrency control is based on *interleaved concurrency*.
- In a multiuser DBMS, the stored data items are the primary resources that may be accessed concurrently by interactive users or application programs, which are constantly retrieving information from and modifying the DB.

Transactions, DB items, read and write operations

- A transaction is an executing program that forms a logical unit of DB processing and includes DB access operations such as insertion, deletion, modification (update) or retrieval.
- The DB operations that form a transaction is within an application program or specified via SQL.
- Transaction boundaries are specified by explicit begin transaction and end transaction statements in application programs and all DB access operations between the two are considered “one transaction”.
- A single application program may contain more than one transaction if it contains several transaction boundaries.
- If the DB operations in a transaction do not update the DB but only retrieve data, the transaction is called a read-only otherwise it is read-write.

- Using the simplified DB model, the basic DB access operations that a transaction can include are as follows:
 - **read_item(*X*)**. Reads a database item named *X* into a program variable. To simplify our notation, we assume that *the program variable is also named X*.
 - **write_item(*X*)**. Writes the value of program variable *X* into the database item named *X*.

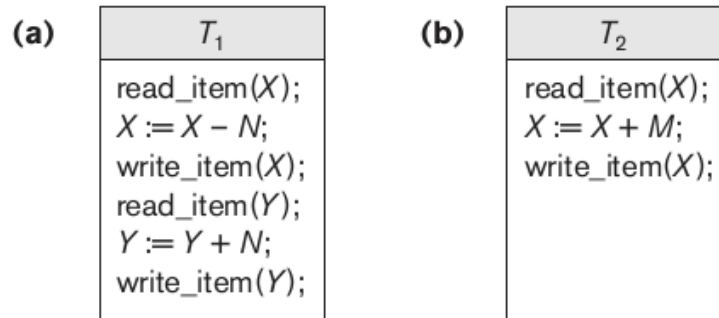
- Executing a **read_item(*X*)** command includes the following steps:
 1. Find the address of the disk block that contains item *X*.
 2. Copy that disk block into a buffer in main memory (if that disk block is not already in some main memory buffer). The size of the buffer is the same as the disk block size.
 3. Copy item *X* from the buffer to the program variable named *X*.

- Executing a `write_item(X)` command includes the following steps:
 1. Find the address of the disk block that contains item X .
 2. Copy that disk block into a buffer in main memory (if that disk block is not already in some main memory buffer).
 3. Copy item X from the program variable named X into its correct location in the buffer.
 4. Store the updated disk block from the buffer back to disk (either immediately or at some later point in time).
- Usually, the decision about when to store a modified disk block whose contents are in a main memory buffer is handled by the recovery manager of the DBMS in coordination with the OS (operating system).

- The DBMS maintains in the DB cache a number of data buffers in main memory where each buffer typically holds the contents of one DB disk block, which contains some of the DB items being processed.
 - When these buffers are all occupied, and additional DB disk blocks must be copied into memory, some buffer replacement policy is used to choose which of the current occupied buffers is to be replaced such as using LRU (least recently used) policies.
 - If the chosen buffer has been modified, it must be written back to disk before it is reused.
- It is important to remember that concurrency control and recovery mechanisms are mainly concerned with the DB commands in a transaction.

Why concurrency control is required?

- Consider airline reservations DB in which a record is stored for each airline flight. Each record includes the *number of reserved seats* on that flight as a named data item.



- T_1 and T_2 are specific executions of the program that refer to the specific flights whose numbers of seats are stored in data items X and Y in the DB. Types of problem encountered with two simple concurrent transactions are:
 - The lost update problem – This problem occurs when two transactions that access the same DB items have their operations interleaved in a way that makes the value of some DB items incorrect.

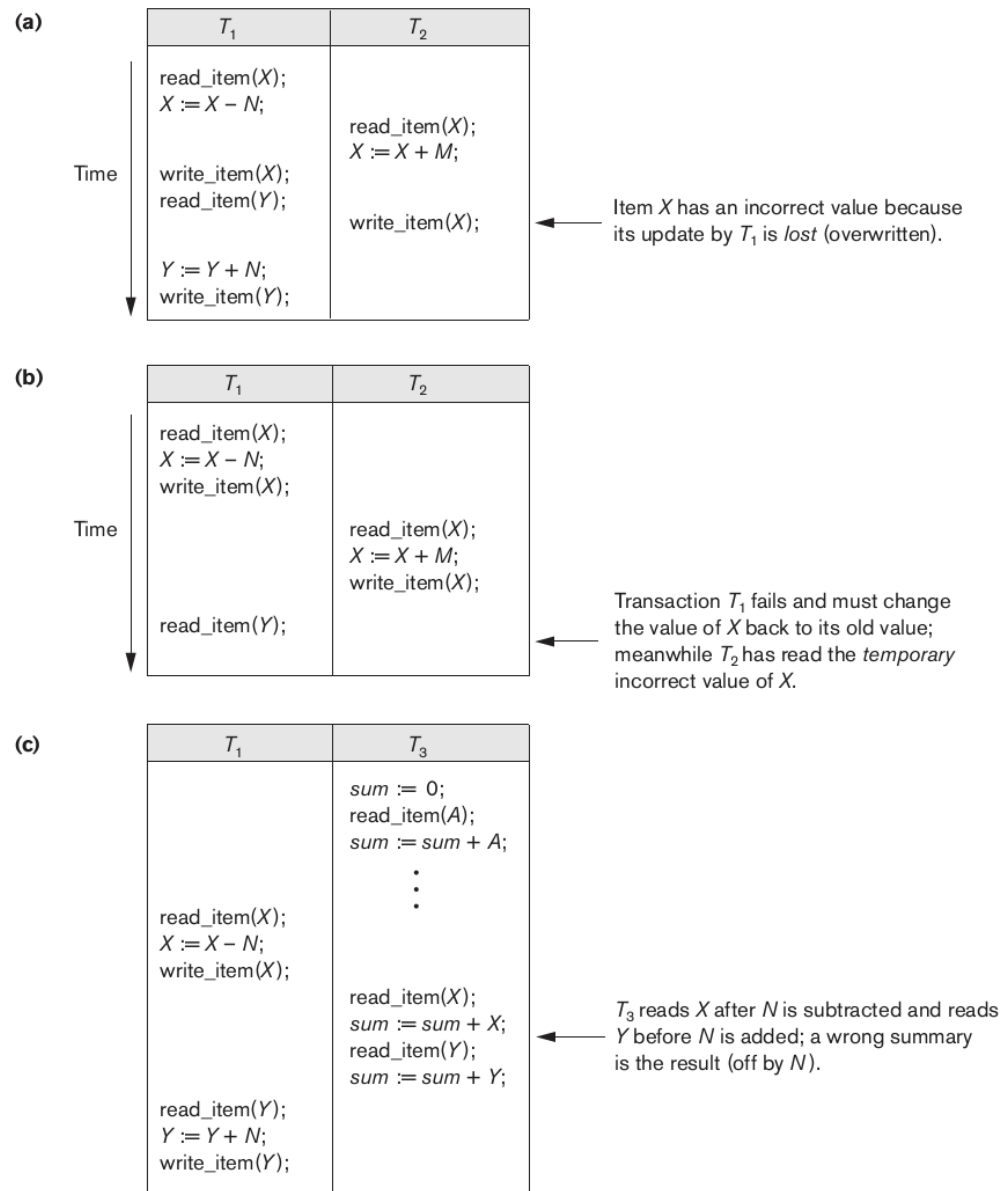



Figure: Problems that occur when concurrent execution is uncontrolled.
 (a) The lost update problem, (b) The temporary update problem, (c) The incorrect summary problem.

- 
2. The temporary update (or dirty read) problem – This problem occurs when one transaction updates a DB item and then the transaction fails for some reason. Meanwhile, the updated item is accessed (read) by another transaction before it is changed back (or rolled back) to its original value.
 3. The incorrect summary problem – This happens if one transaction is calculating an aggregate summary function on a number of DB items while other transactions are updating some of these items, the aggregate function may calculate some values before they are updated and others after they are updated.
 4. The unrepeatable read problem – This happens when a transaction T reads the same item twice and the item is changed by another transaction T' between the two reads.

Why recovery is required?

- Whenever a transaction is submitted to a DBMS for execution, either all the operations in the transaction are completed successfully and recorded in the DB (known as committed) or the transaction does not have any effect on the DB or any other transactions (known as aborted).
- If a transaction fails after executing some of its operations but before executing all of them, the operations already executed must be undone and have no lasting effect.

Why recovery is required?

- Types of failure – Failures are generally classified as transactions, system, and media failures. There are several possible reasons for a transaction to fail in the middle of execution:
 1. **A computer failure (system crash)** – A hardware, software, or network error occurs in the computer system during transaction execution.
 2. **A transaction or system error** – Some operation in the transaction may cause it to fail, such as integer overflow or division by zero or because of erroneous parameter values or because of a logical programming error.

Why recovery is required?

3. **Local errors or exception conditions detected by the transaction** – During transaction execution, certain conditions may occur that necessitate cancellation of the transaction. Example: data for the transaction may not be found such as insufficient account balance in a banking transaction which may cause a transaction such as a fund withdrawal to be cancelled.
4. **Concurrency control enforcement** – The concurrency control method may abort one or more transactions to resolve a state of deadlock among several transactions.
5. **Disk failure** – Some disk blocks may lose their data because of a read or write malfunction or because of a disk read/write head crash happening during a read or a write operation of a transaction.
6. **Physical problems and catastrophes** – This may include problems such as fire, theft, overwriting disks or tapes by mistake, etc.

Transaction and system concepts

- **Transaction states and additional operations** – A transaction is an atomic unit of work that should either be completed in its entirety or not done at all.
- For recovery purposes, the system needs to keep track of when each transaction starts, terminates, and commits, or aborts. The recovery manager of the DBMS needs to keep track of the following operations:
 - **BEGIN_TRANSACTION** – This marks the beginning of transaction execution.
 - **READ** or **WRITE** – These specify read or write operations on the DB items that are executed as part of a transaction.

- **END_TRANSACTION** – This specifies that **READ** and **WRITE** transaction operations have ended and marks the end of transaction execution. It is necessary to check whether the changes introduced by the transaction can be permanently applied to the DB (committed) or whether the transaction has to be aborted.
- **COMMIT_TRANSACTION** – This signals a successful end of the transaction so that any changes (updates) executed by the transaction can be safely committed to the DB and will not be undone.
- **ROLLBACK** (or **ABORT**) – It signals that the transaction has ended unsuccessfully, so that any changes or effects that the transaction may have applied to the DB must be undone.

- Figure below shows how a transaction moves through its execution states.

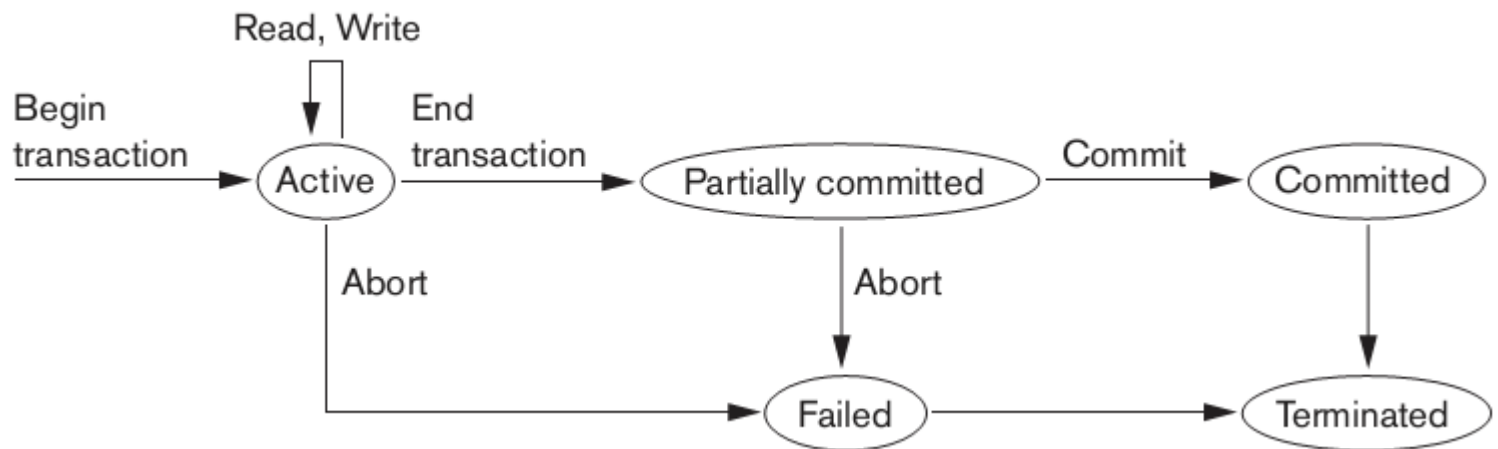


Figure: State transition diagram illustrating the states for transaction execution.

- A transaction goes into an **active state** immediately after it starts execution, where it can execute its READ and WRITE operations.

- When the transaction ends, it moves to the **partially committed state**, where some types of concurrency control protocols may do additional checks to see if the transaction can be committed or not.
- Also, some recovery protocols need to ensure that a system failure will not result in an ability to record the changes of the transaction permanently.

- If these checks are successful, the transaction is said to have reached its commit point and enters the **committed state**. After commit, a transaction has concluded its execution successfully and all its changes must be recorded permanently in the DB, even if a system failure occurs.

- However, a transaction can go to the **failed state** if one of the checks fails or if the transaction is aborted during its active state. The transaction may then have to be rolled back to undo the effect of its WRITE operations on the DB.

- The **terminated state** corresponds to the transaction leaving the system.
- The transaction information that is maintained in system tables (while the transaction has been running) is removed when the transaction terminates.
- Failed or aborted transactions may be *restarted* later either automatically or after being resubmitted by the user – as brand new transactions.

- **The system log** – To be able to recover from failures that affect transactions, the system maintains a “log” to keep track of all transaction operations that affect the values of DB items, as well as other transaction information that may be needed to permit recovery from failures.

- The log is a sequential, append only file that is kept on disk, so it is not affected by any type of failure.

- Typically, one (or more) main memory buffers, called the log buffers, hold that last part of the log file, so that log entries are first added to the log main memory buffer.
- When the log buffer is filled, it is appended to the end of the log file on disk. Additionally, the log file from disk is periodically backed up to archival storage (tape).

- “Log records” that are written to the log file and the corresponding action for each log record are as listed below.
- T refers to a unique transaction-id that is generated automatically by the system for each transaction and is used to identify each transaction:
 1. [**start_transaction**, T]. Indicates that transaction T has started execution.
 2. [**write_item**, T , X , *old_value*, *new_value*]. Indicates that transaction T has changed the value of database item X from *old_value* to *new_value*.
 3. [**read_item**, T , X]. Indicates that transaction T has read the value of database item X .
 4. [**commit**, T]. Indicates that transaction T has completed successfully, and affirms that its effect can be committed (recorded permanently) to the database.
 5. [**abort**, T]. Indicates that transaction T has been aborted.

- All permanent changes to the DB occur within transactions, so the notion of recovery from a transaction failure amounts to either undoing or redoing transaction operations from the log.

- If the system crashes, we can recover to a consistent DB state by examining the log.

- Because the log contains a record of every WRITE operation that changes the value of some DB item, it is possible to “undo” the effect of these WRITE operations of a transaction T by tracking backwards through the log and resetting all items changed by a WRITE operation of T to their *old_values*.
- “Redo” of an operation may also be necessary if a transaction has its updates recorded in the log but a failure occurs before the system can be sure that all these *new_values* have been written to the actual DB on disk from the main memory buffers.

- **Commit point of a transaction** – A transaction T reaches its commit point when all its operations that access that DB have been executed successfully and the effect of all the transaction operations on the DB have been recorded in the log.
- Beyond the commit point, the transaction is said to be committed, and its effect must be *permanently recorded* in the DB. The transaction then writes a commit record $[\text{commit}, T]$ into the log.
- If a system failure occurs, we can search back in the log for all transactions T that have written a $[\text{start_transaction}, T]$ record into the log but have not written their $[\text{commit}, T]$ record yet which may have to be *rolled back to undo their effect* of the DB during a recovery process.
- Transactions that have written their commit record in the log must also have recorded all their WRITE operations in the log, so their effect on the DB can be “*redone*” from the log records.

- **DBMS-specific buffer replacement policies** – If all the buffers in the DBMS cache are occupied and new disk pages are required to be loaded into main memory from disk, *a page replacement policy is used to select the particular buffers to be replaced*. Example:

1. Domain separation (DS) method – In this method, the DBMS cache is divided into separate domains (set of buffers).

- Each domain handles one type of disk pages (such as index page, data file pages, log file pages, etc.), and page replacements within each domain are handled via the basic LRU (least recently used) page replacement.
- This is a static algorithm, and so does not adapt to the dynamically changing loads because the number of available buffers of each domain is predetermined.

2. Hot set method – This algorithm is useful in queries that have to scan a set of pages repeatedly, such as “join operation using the nested-loop method”.


- If the inner loop file is loaded completely into main memory buffers without replacement (the hot set), the join will be performed efficiently because each page in the outer loop file will have to scan all the records in the inner loop file to find join matches.
- Therefore, this method determines for each DB processing algorithm the set of disk pages that will be accessed repeatedly, and it does not replace them until their processing is completed.

3. The DBMIN method – This method uses QLSM (query locality set model) which predetermines the pattern of page references for each algorithm for a particular type of DB operation (example SELECT and JOIN).

- Depending on the type of access method, the file characteristics, and the algorithms used, the QLSM will estimate the number of main memory buffers needed for each file involved in the operation.

Desirable properties of transactions

- Transactions should possess ACID properties:
 1. **Atomicity** – A transaction is an atomic unit of processing; it should either be performed in its entirety or not performed at all.
 2. **Consistency preservation** – A transaction should be consistency preserving, meaning that if it is completely executed from beginning to end without interference from other transactions, it should take the DB from one consistent state to another.
 3. **Isolation** – A transaction should appear as though it is being executed in isolation from other transactions, even though many transactions are executed concurrently. That is, the execution of a transaction should not be interfered with by any other transactions executing concurrently.
 4. **Durability or permanency** – The changes applied to the DB by a committed transaction must persist in the DB. These changes must not be lost because of any failure.

- 
- The *atomicity property* requires that we execute a transaction to completion. It is the responsibility of the *transaction recovery subsystem* of a DBMS to ensure atomicity.
 - If a transaction fails to complete for some reason, such as a system crash during execution, the recovery technique must “undo” any effects of the transaction on the DB.
 - On the other hand, write operations of a committed transaction must be eventually written to disk.

Transaction support in SQL

- In SQL, a transaction is a logical unit of work and is guaranteed to be atomic. A single SQL statement is always considered to be atomic – either it completes execution without an error or it fails and leaves the DB unchanged.
- There is no explicit `Begin_Transaction` statement and is done implicitly in SQL statements. However, every transaction must have an explicit “end statement” which is either a `COMMIT` or a `ROLLBACK`.
- Every transaction has certain characteristics specified by `SET TRANSACTION` statement in SQL such as *access mode*, the *diagnostic area size*, and the *isolation level*.

- Access mode – It can be specified as READ ONLY or READ WRITE (which is default unless the isolation level of READ UNCOMMITTED is specified).
 - READ WRITE allows select, update, insert, delete and create commands to be executed. READ ONLY is simply for data retrieval.
- The diagnostic area size option, DIAGNOSTIC SIZE specifies an integer value (n), which indicates the number of conditions that can be held simultaneously in the diagnostic area. These conditions supply feedback information (errors or exceptions) to the user or program on the “ n ” most recently executed SQL statement.
- The isolation level – it is specified using the statement ISOLATION LEVEL <isolation>, where its values can be READ UNCOMMITTED*, READ COMMITTED*, REPEATABLE READ*, etc.

* See next slide.

- **Read Uncommitted** is the weakest isolation level because it can read the data – which are acquired exclusive lock to the resources by the other transactions. So, it might help to avoid locks and deadlock problems for the data reading operations.
- **Read Committed** is the default isolation level. When a transaction runs on this isolation level, a SELECT query sees only data committed before the query began and never sees either uncommitted data or changes committed during query execution by concurrent transactions.
- **Repeatable Read** is a higher isolation level, that in addition to the guarantees of the read committed level, it also guarantees that any data read cannot change, if the transaction reads the same data again, it will find the previously read data in place, unchanged, and available to read.

Summary

We discussed

- DBMS concepts for transaction processing.
- Single-user and multi-user system.
- Uncontrolled execution of concurrent transactions.
- Failures during transaction execution.
- States of transactions and concepts of recovery and concurrency control.
- System log and committed transactions.
- ACID properties.
- Transaction concepts within SQL.



Practice questions (check yourself)

1. What is a transaction?
2. Discuss lost update, temporary update, incorrect summary and unrepeatable read problems.
3. What is committed state of a transaction?
4. Explain system log, log buffer and log records.
5. Explain domain separation method.
6. Define hot set method.
7. Discuss ACID properties of transactions.