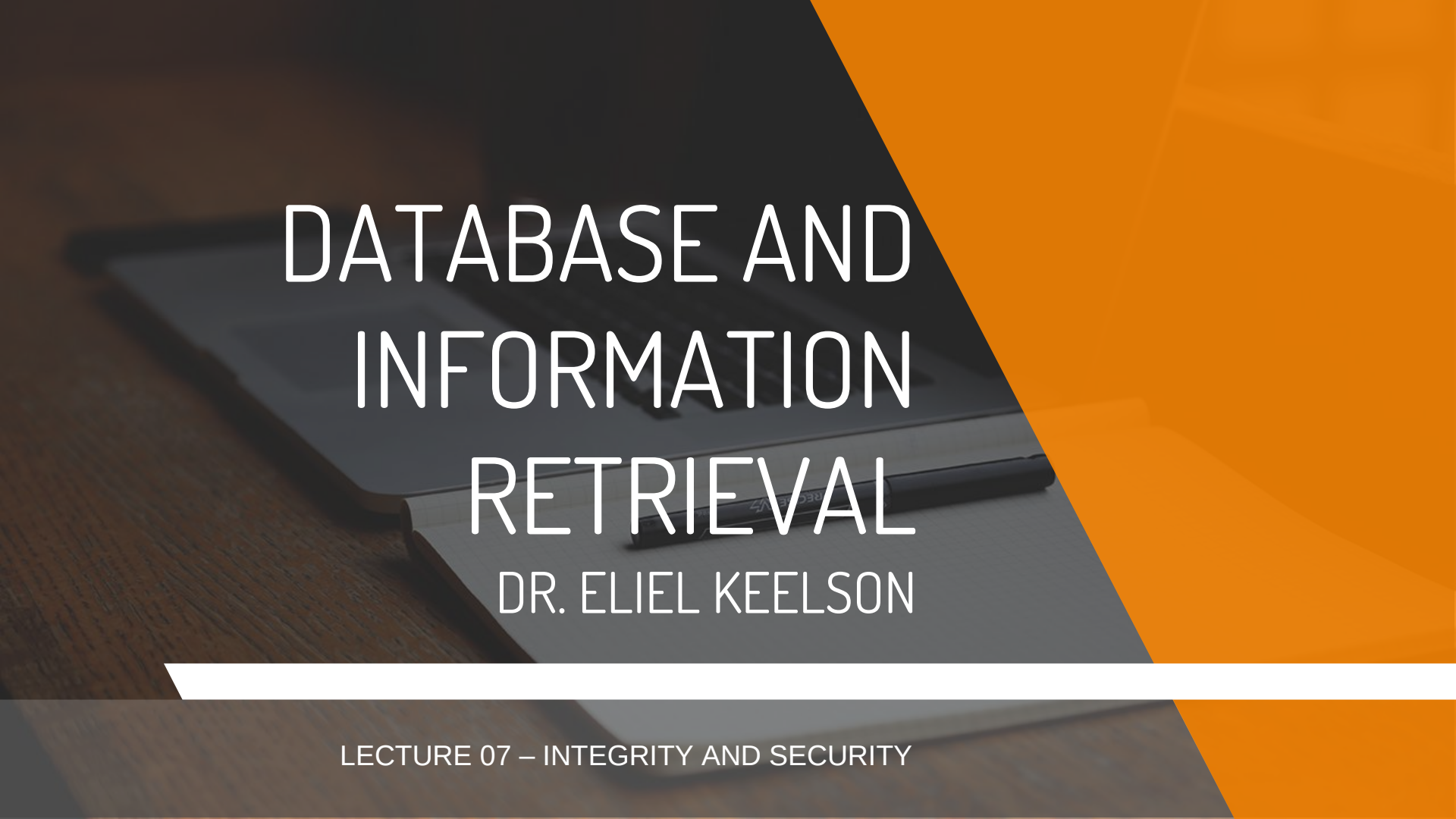


DATABASE AND INFORMATION RETRIEVAL

DR. ELIEL KEELSON

LECTURE 07 – INTEGRITY AND SECURITY

Integrity And Security

- A database is intended to hold data that accurately represents values in the application domain of the system.
- The term 'constraint' is used to refer to a set of rules and restrictions that define the admissible content of the database.
- Some constraints are based on the domain of the data values and other natural limitations, for instance, an employee's name cannot be a numeric value or their age a negative value.

Integrity And Security

- Some constraints are determined by the application system designers as being rules and restrictions to which the application system and the database must conform.
- Such constraints are often called **business** or **enterprise rules**.

Threats to the Database

- User Errors : The database users can accidentally (or otherwise) enter erroneous values, amend data incorrectly or delete valid data.
- Software Errors: Programming errors in the database or in the application system software.
- Hardware Failure: Breakdown of computer equipment, physical damage such as flooding, loss of power, etc.

Threats to the Database

- Malicious damage: A database could be corrupted by authorised or unauthorised users.
- Breach of confidentiality: Unauthorised persons may get access to the database.
- Concurrency errors

Database Integrity

- *Consistency* implies that the data held in the various tables of the database is consistent with the concept of the relational model.

- Consistency is expressed in terms of two characteristics:
  - Entity Integrity
  - Referential Integrity

Database Integrity

- Entity integrity is concerned with ensuring that each row of a table has a unique and non-null primary key value. This is the same as saying that each row in a table represents a single instance of the entity type modelled by the table.

- Referential integrity is concerned with the relationships *between* the tables of a database; i.e. that the data of one table does not contradict the data of another table.

Database Integrity

- *Correctness* implies that data captured for entry into the computer does in fact correctly represent the ‘the real world’ data that it is supposed to.

- This involves taking care with the capturing and handling of data at all stages prior to and during data entry to the database.

- Common forms of error in data entry are transposition of numerical digits, misspelling of names, repetition of characters, etc.

- The primary defence against invalid data is data input validation

Data Validation

- Data validation refers to the process of checking data entering the database to ensure that the data is sensible and meaningful to the database application system, as far as this can be accomplished.

Data Validation

There is a range of techniques that can be used:

- Type checking – data-types provide a general defence against errors

- Validation checks – refers to checks for correctness defined by the database designer and applied at various stages of database use.

- Assertions and Triggers - An **assertion** is a term used to describe methods that impose general controls on the content of a database. A **trigger** is a term used in many systems for event-handling routines.

Data Validation

SQL provides some features for data validation:

- NOT NULL clause in a CREATE command; forces a non-null entry in the field.
- Primary Key constraint: defines a column as a primary key.
- Foreign Key constraint: defines the column as a foreign key and references corresponding primary key. Enforces referential integrity.
- CHECK specification: provides a general validation specification within a CREATE table definition

Data Validation – Assertion

- An assertion is a named constraint that may relate to the content of individual rows of a table, to the entire contents of a table, or to a state required to exist among a number of tables.

- Assertions are similar to **CHECK** constraints, but unlike **CHECK** constraints they are not defined on table or column level but are defined on schema level. (i.e., assertions are database objects of their own right and are not defined within a create table or alter table statement.)

Data Validation – Assertion

- The SQL syntax for create assertion is:

CREATE ASSERTION <constraint name> CHECK (<search condition>)

- For Example:

CREATE ASSERTION AverageSal
CHECK ((SELECT AVG(Salary) FROM Employee) < 2000) ;

- This forces the system to verify that the given condition persists as the database is updated.

Data Validation – Triggers

- A **trigger** is a term used in many systems (including Oracle) for event-handling routines.

- Events are associated with tables, fields, forms etc. and cause execution of code routines such as PL/SQL in various DBMS'.

Data Validation – Triggers

- Triggers can be defined in SQL as:

```
CREATE TRIGGER Trig1
        BEFORE
            INSERT or UPDATE
            ON Employee
         BEGIN
            …
          END;
```

- This trigger would 'fire' before any row inserts or update on the Employee table

Backups and Recovery

- Backing up is the most fundamental way of protecting the data against a wide range of failures and involves taking copies of the data at intervals.

- The database can be re-built in the event of loss of data.

Backups and Recovery

- The frequency of taking backups is dictated by a number of factors including:
    1. the rate of transactions applied to the database,
    2. the level of availability of service demanded by the application.
    3. the balance of time required to perform the backup compared with the potential delay in recovering.

Transaction Log

- A transaction log or journal is used to record the effect of all changes made to the database by transactions generated by application systems.

- New rows added to the database are recorded in the log;

- Deleted rows are recorded as at the time of deletion

Transaction Log

- Values of amended rows are recorded *before* and *after* the update (called the before and after images).

- In the event of a system failure, it is possible to recover the database by restoring the most recent backup and then automatically re-executing the transactions recorded in the transaction log.

Checkpoints

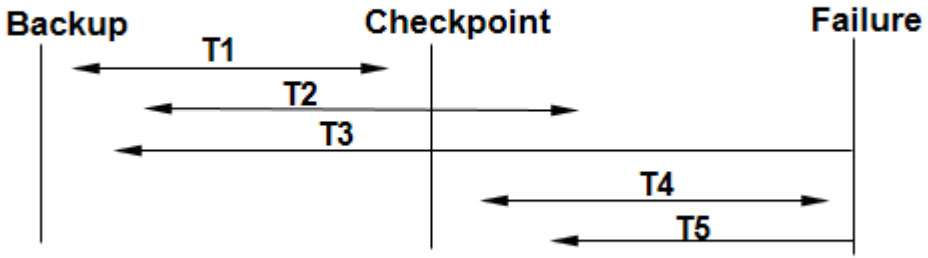
- Checkpoints are transaction status records taken at intervals during normal processing of transactions.

- Typically, a checkpoint would be taken at frequent intervals. At the checkpoint time, the following actions take place:
  1. initiation of new transactions is temporarily suspended.
  2. all memory buffers are flushed to disk. This ensures that all committed transactions have indeed been actioned on the physical database.
  3. all currently active transactions are noted and recorded in the transaction log.

Checkpoints

- This enables faster recovery from a database failure since the database state can be corrected by application of the transactions since the checkpoint.

Checkpoints



Checkpoints

- Transaction t1 begins and ends before the checkpoint (it will not appear on the checkpoint record) and hence will be correctly implemented on the database.

- Transaction t2 started before the checkpoint and would be flagged as a current transaction by the checkpoint. Although the transaction log would show that it had committed, there would be an element of doubt as to whether the in-memory buffers were written to disk. To be certain, the ‘after-image’ from the transaction log would be written to disk. A similar argument applies to T4.

Checkpoints

- Transactions T3 and T5 were incomplete at the time of failure and hence may be partially written to disk. To restore the database to a known consistent condition, the before images from the transaction log would be applied. These transactions would then need to be re-executed.

Privileges and Permissions

- In multi-user database systems, it is necessary to control the data that is accessible and/or modifiable by each user and class of user.

- This is necessary to ensure that data can only be read or changed by users who are entitled to do so.

- Additionally, it is necessary to control who has system permissions such as the right to create, alter or drop a table, to grant privileges to others, etc.

Privileges and Permissions

- Rather than identify the access permissions of each individual user it is more convenient to classify users into workgroups and to assign rights to these groups.

- Systems will vary in the way access rights are specified and in the granularity of ‘objects’ that can be controlled.

Privileges and Permissions

- Access rights are specified in SQL by the GRANT and REVOKE commands.

GRANT <privilege list> ON <object> TO <grantee list> [WITH GRANT OPTION];

REVOKE <privilege list> ON <object> FROM <grantee list>;

Privileges and Permissions

- The GRANT command shown below gives the users Joe and Mary the right to view and update data in a table called Orders :

```
GRANT select, update ON Orders TO joe, mary;
```

- The following REVOKE command removes Joe's right to update the table:

```
REVOKE update ON Orders TO joe;
```

TRANSACTION MANAGEMENT

Transaction Management

- A transaction is a discrete series of actions that must be either completely processed or not processed at all.
- Even though transactions are a group of SQL statements they are to be seen as one logical atomic unit.
- The effects of all the SQL statements in a transaction can be either all **committed** (applied to the database) or all **rolled back** (undone from the database).

Transaction Management

- To illustrate the concept of a transaction, consider a banking database. When a bank customer transfers money from a savings account to a checking account, the transaction can consist of three separate operations:

  - Decrement the savings account
  - Increment the checking account
  - Record the transaction in the transaction journal

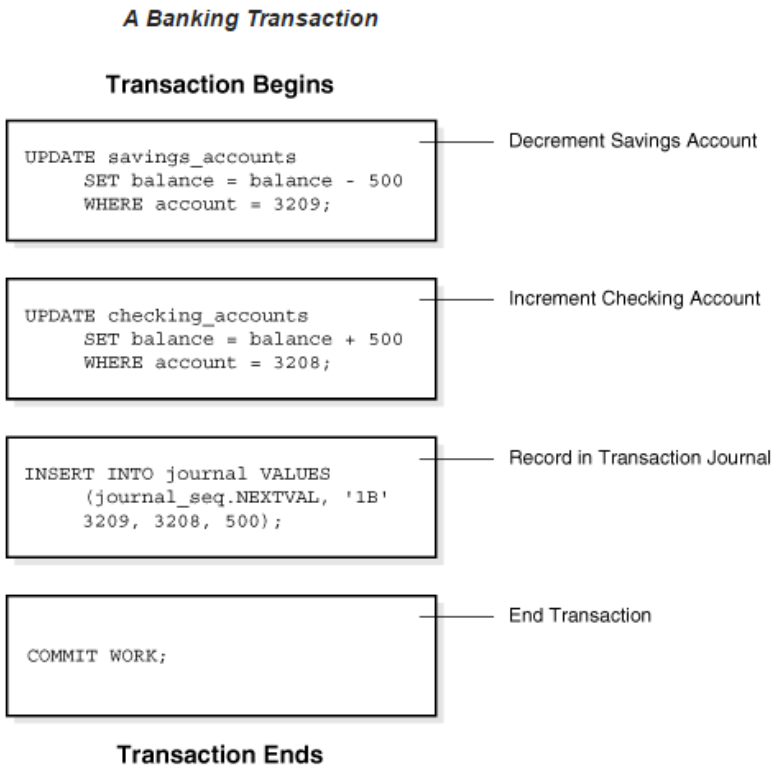
Transaction Management

- The DBMS in use must allow for two situations.

1. If all three SQL statements can be performed to maintain the accounts in proper balance, the effects of the transaction can be applied to the database.

2. However, if a problem such as insufficient funds, invalid account number, or a hardware failure prevents one or two of the statements in the transaction from completing, the entire transaction must be rolled back so that the balance of all accounts is correct.

Transaction Management



Transaction Management –ACID Properties

- A transaction is a very small unit of a program and it may contain several low level tasks.
- A transaction in a database system must maintain **A**tomicity, **C**onsistency, **I**solation, and **D**urability. These together are commonly known as **ACID** properties.
- They are implemented in order to ensure accuracy, completeness, and data integrity.

Transaction Management – Atomicity

- This property states that a transaction must be treated as an atomic unit, that is, either all of its operations are executed or none.
- It must remain whole. That is, it must completely succeed or completely fail.
- There must be no state in a database where a transaction is left partially completed.
- When it succeeds, all changes that were made by the transaction must be preserved by the system.
- Should a transaction fail or be aborted, all changes that were made by it must be completely undone.

Transaction Management – Consistency

- A transaction should transform the database from one consistent state to another.
- The database must remain in a consistent state after any transaction.
- No transaction should have any adverse effect on the data residing in the database.
- If the database was in a consistent state before the execution of a transaction, it must remain consistent after the execution of the transaction as well.

Transaction Management – Isolation

- In a database system where more than one transaction are being executed simultaneously and in parallel, the property of isolation states that all the transactions will be carried out and executed as if it is the only transaction in the system.
- Each transaction should carry out its work independent of any other transaction that might occur at the same time.
- No transaction will affect the existence of any other transaction.

Transaction Management – Durability

- Changes made by completed transactions should remain permanent, even after a subsequent shutdown or failure of the database or other critical system component.
- In other words, the database should be durable enough to hold all its latest updates even if the system fails or restarts.
- If a transaction updates a chunk of data in a database and commits, then the database will hold the modified data.
- If a transaction commits but the system fails before the data could be written on to the disk, then that data will be updated once the system springs back into action.

SERIALIZABILITY

Serializability – Serial Schedule

- A **Schedule** can be defined as a chronological execution sequence of a transaction.
- A **schedule** can have many transactions in it, each comprising of a number of instructions/tasks.
- A **Serial Schedule** is a schedule in which transactions are aligned in such a way that one transaction is executed first. When the first transaction completes its cycle, then the next transaction is executed. They are executed in a serial manner.

Serializability

- When multiple transactions are being executed by the operating system in a multiprogramming environment, there are possibilities that instructions of one transactions are interleave/interspersed with some other transactions.
- In a multi-transaction environment, serial schedules are considered as a benchmark.

Serializability

- The execution sequence of an instruction in a transaction cannot be changed, but two transactions can have their instructions executed in a random fashion.
- This execution does no harm if two transactions are **mutually independent** and working on different segments of data; but in case these two transactions are working on the same data, then the results may vary.
- This ever-varying result may bring the database to an inconsistent state.
- To resolve this problem, we allow parallel execution of a transaction schedule, if its transactions are either **serializable** or have some equivalence relation among them.

Serializability

- **Serializability** is the often described as a classical concurrency scheme.
- It ensures that a schedule for executing concurrent transactions is equivalent to one that executes the transactions serially in some order.
- It assumes that all accesses to the database are done using read and write operations.

Serializability

- A schedule is called ``correct'' if we can find a serial schedule that is ``equivalent'' to it.

- Given a set of transactions T1...Tn, two schedules S1 and S2 of these transactions are equivalent if the following conditions are satisfied:

1. ***Read-Write Synchronization*:** If a transaction reads a value written by another transaction in one schedule, then it also does so in the other schedule.
2. ***Write-Write Synchronization*:** If a transaction overwrites the value of another transaction in one schedule, it also does so in the other schedule.

Serializability

Serial Schedule	
T1	T2
R1(A)	
W1(A)	
R1(B)	
R1(B)	
C1	
	R2(A)
	W2(A)
	R2(B)
	W2(B)
	C2

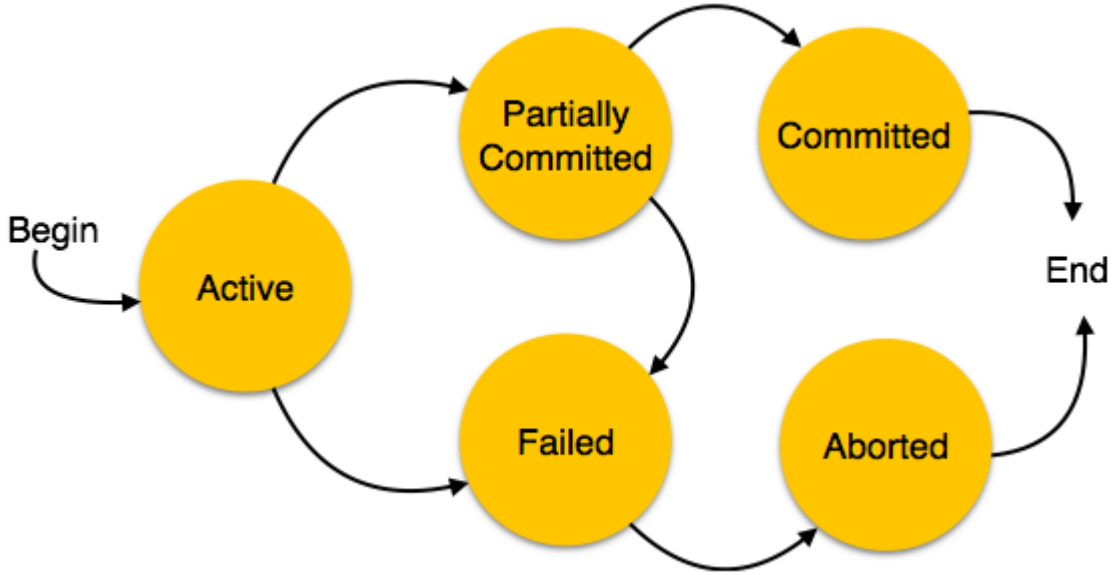
schedule-1

Interleaved schedule	
T1	T2
R1(A)	
W1(A)	
	R2(A)
	W2(A)
R1(B)	
R1(B)	
C1	
	R2(B)
	W2(B)
	C2

schedule-2

States of Transactions

A transaction in a database can be in one of the following states



States of Transactions

- **Active** – In this state, the transaction is being executed. This is the initial state of every transaction.
- **Partially Committed** – When a transaction executes its final operation, it is said to be in a partially committed state.
- **Failed** – A transaction is said to be in a failed state if any of the checks made by the database recovery system fails. A failed transaction can no longer proceed further.

States of Transactions

- **Aborted** − If any of the checks fails and the transaction has reached a failed state, then the recovery manager rolls back all its write operations on the database to bring the database back to its original state where it was prior to the execution of the transaction.

- Transactions in this state are called aborted. The database recovery module can select one of the two operations after a transaction aborts −
  1. Re-start the transaction
  2. Kill the transaction

States of Transactions

- **Committed** – If a transaction executes all its operations successfully, it is said to be committed. All its effects are now permanently established on the database system.

CONCURRENCY CONTROL

Concurrency Control

- In a multiprogramming environment where multiple transactions can be executed simultaneously, it is highly important to control the concurrency of transactions.

- We have concurrency control protocols to ensure atomicity, isolation, and serializability of concurrent transactions.

- Concurrency control protocols can be broadly divided into two categories

1. Lock based protocols
2. Time stamp based protocols

Concurrency Control – Lock-based Protocols

- Database systems equipped with lock-based protocols use a mechanism by which any transaction cannot read or write data until it acquires an appropriate lock on it. Locks are of two kinds –

1. Binary Locks
2. Shared/Exclusive Locks

Concurrency Control - Lock-based Protocols

- **Binary Locks** – A lock on a data item can be in two states; it is either locked or unlocked.

- **Shared/exclusive** – This type of locking mechanism differentiates the locks based on their uses. If a lock is acquired on a data item to perform a write operation, it is an exclusive lock. Allowing more than one transaction to write on the same data item would lead the database into an inconsistent state. Read locks are shared because no data value is being changed.

Concurrency Control – Lock-based Protocols

- There are four types of lock protocols available –

1. Simplistic Lock Protocol
2. Pre-claiming Lock Protocol
3. Two-Phase Locking 2PL
4. Strict Two-Phase Locking

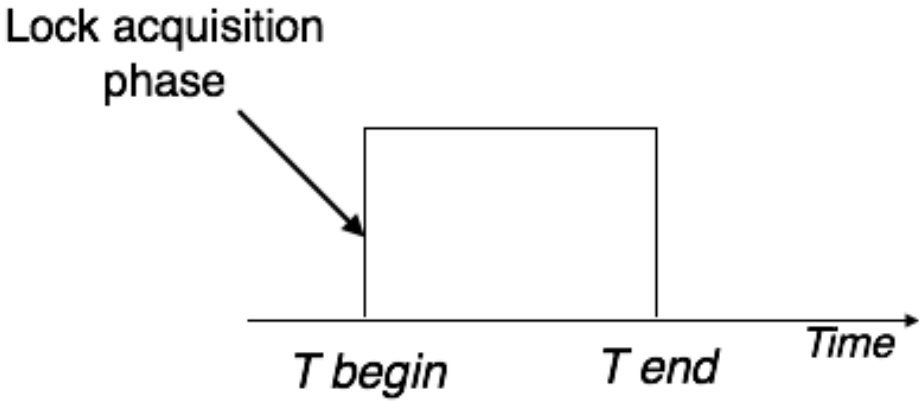
Lock-based Protocols – Simplistic Lock Protocol

- Simplistic lock-based protocols allow transactions to obtain a lock on every object before a 'write' operation is performed.
- Transactions may unlock the data item after completing the ‘write’ operation.

Lock-based Protocols – Pre-claiming Lock Protocol

- Pre-claiming protocols evaluate their operations and create a list of data items on which they need locks.
- Before initiating an execution, the transaction requests the system for all the locks it needs beforehand.
- If all the locks are granted, the transaction executes and releases all the locks when all its operations are over.
- If all the locks are not granted, the transaction rolls back and waits until all the locks are granted.

Lock-based Protocols – Pre-claiming Lock Protocol

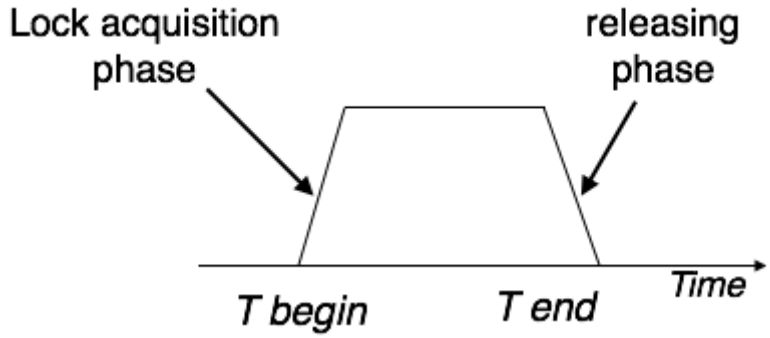


Lock-based Protocols - Two-Phase Locking (2PL)

- This locking protocol divides the execution phase of a transaction into three parts.
- In the first part, when the transaction starts executing, it seeks permission for the locks it requires.
- The second part is where the transaction acquires all the locks.
- As soon as the transaction releases its first lock, the third phase starts. In this phase, the transaction cannot demand any new locks; it only releases the acquired locks.

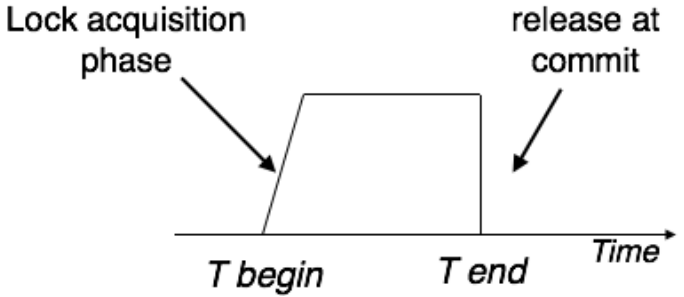
Lock-based Protocols – Two-Phase Locking (2PL)

- Two-phase locking has two phases, one is **growing**, where all the locks are being acquired by the transaction; and the second phase is shrinking, where the locks held by the transaction are being released.
- To claim an exclusive (write) lock, a transaction must first acquire a shared (read) lock and then upgrade it to an exclusive lock.



Lock-based Protocols –Strict Two-Phase Locking

- The first phase of Strict-2PL is same as 2PL. After acquiring all the locks in the first phase, the transaction continues to execute normally.
- But in contrast to 2PL, Strict-2PL does not release a lock after using it. Strict-2PL holds all the locks until the commit point and releases all the locks at a time. Thus Strict-2PL does not have cascading abort as 2PL does.



Concurrency Control – Timestamp-based Protocols

- The most commonly used concurrency protocol is the timestamp based protocol.
- This protocol uses either system time or logical counter as a timestamp.

Concurrency Control - Timestamp-based Protocols

- Lock-based protocols manage the order between the conflicting pairs among transactions at the time of execution, whereas timestamp-based protocols start working as soon as a transaction is created.

- Every transaction has a timestamp associated with it, and the ordering is determined by the age of the transaction. A transaction created at 0002 clock time would be older than all other transactions that come after it. For example, any transaction 'y' entering the system at 0004 is two seconds younger and the priority would be given to the older one.

Concurrency Control – Timestamp-based Protocols

- In addition, every data item is given the latest read and write-timestamp. This lets the system know when the last ‘read and write’ operation was performed on the data item.

Concurrency Control – Timestamp Ordering Protocol

- The timestamp-ordering protocol ensures serializability among transactions in their conflicting read and write operations.

- This is the responsibility of the protocol system that the conflicting pair of tasks should be executed according to the timestamp values of the transactions.

Concurrency Control – Timestamp Ordering Protocol

- In the following slide we would use the notation described below:

1. The timestamp of transaction Ti is denoted as $TS(T_i)$.
2. Read time-stamp of data-item X is denoted by R-timestamp(X).
3. Write time-stamp of data-item X is denoted by W-timestamp(X).

Concurrency Control – Timestamp Ordering Protocol

- Timestamp ordering protocol works as follows:

If a transaction T_i issues a read(X) operation

- If $TS(T_i) < W\text{-timestamp}(X)$
  Operation rejected.
- If $TS(T_i) >= W\text{-timestamp}(X)$
  Operation executed.
- All data-item timestamps updated.

Concurrency Control – Timestamp Ordering Protocol

- Timestamp ordering protocol works as follows:

If a transaction T_i issues a write(X) operation −

- If $TS(T_i) < R\text{-timestamp}(X)$
  Operation rejected.
- If $TS(T_i) < W\text{-timestamp}(X)$
  Operation rejected and T_i rolled back.
- Otherwise, operation executed.

Concurrency Control – Thomas' Write Rule

- This rule states if $TS(T_i) < W\text{-timestamp}(X)$, then the operation is rejected and T_i is rolled back.
- Time-stamp ordering rules can be modified to make the schedule view serializable.
- Instead of making T_i rolled back, the 'write' operation itself is ignored.

END

THANKS!

Any questions?

You can find me at elielkeelson@gmail.com &
ekeelson@knust.edu.gh