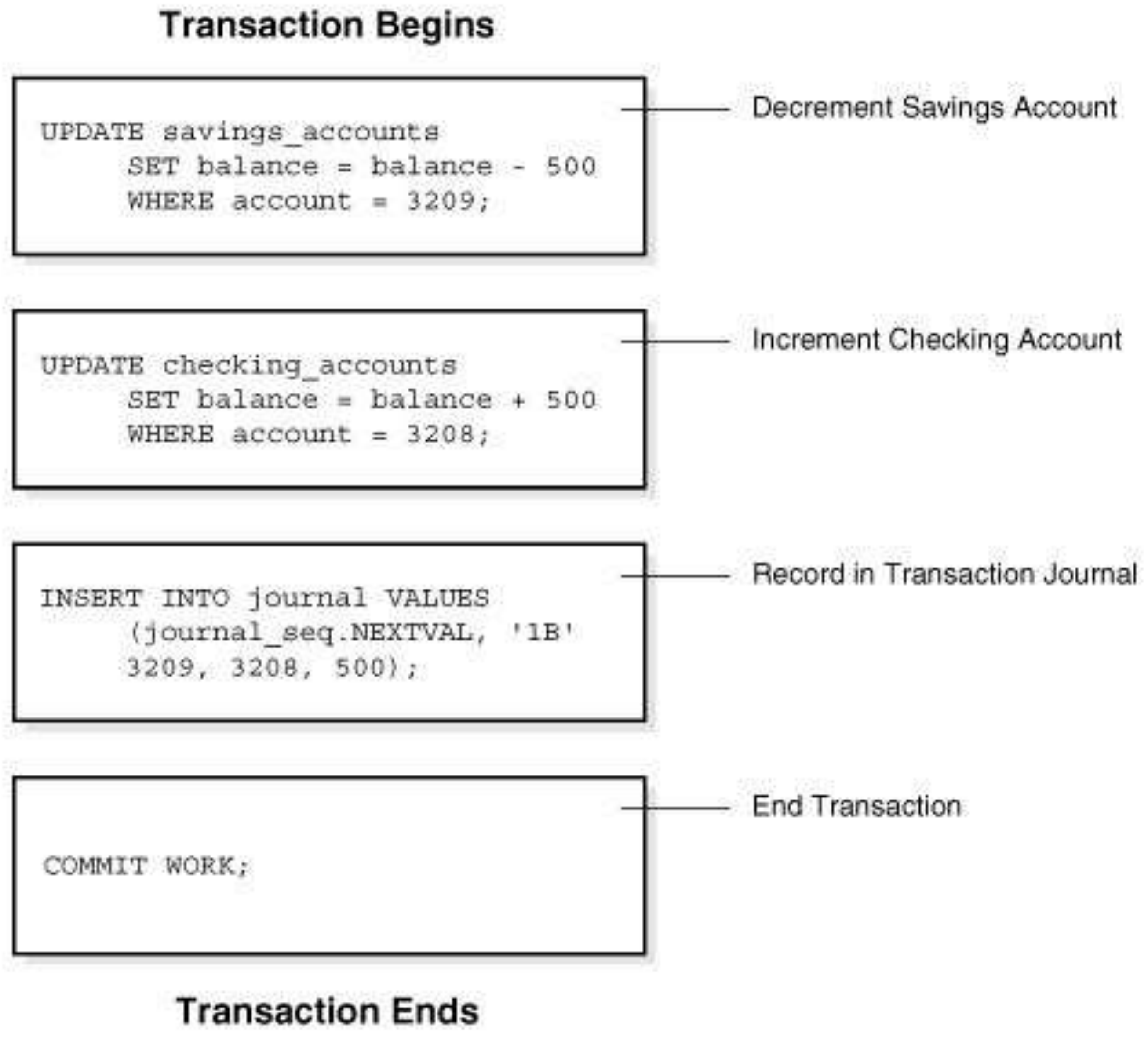
CHAPTER 5

DATABASE TRANSACTION MANAGEMENT

INTRODUCTION TO TRANSACTIONS

- A **transaction** is a logical unit of work that contains one or more SQL statements.
- A transaction is an 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)
- A transaction begins with the first executable SQL statement. A transaction ends when it is committed or rolled back, either explicitly with a COMMIT or ROLLBACK statement or implicitly when a DDL statement is issued.

A Banking Transaction


Transaction Begins
Decrement Savings Account
Increment Checking Account
Record in Transaction Journal
End Transaction
Transaction Ends

```
UPDATE savings_accounts
     SET balance = balance - 500
     WHERE account = 3209;
```

```
UPDATE checking_accounts
     SET balance = balance + 500
     WHERE account = 3208;
```

```
INSERT INTO journal VALUES
     (journal_seq.NEXTVAL, '1B'
     3209, 3208, 500);
```

```
COMMIT WORK;
```

Fundamental Principles - A C I D

- Atomicity
  To outside world, transaction happens indivisibly
- Consistency
  Transaction preserves system invariants
- Isolated
  Transactions do not interfere with each other
- Durable
  Once a transaction “commits,” the changes are permanent

MANAGE DATABASE

- Properties of DB transaction: (ACID)
  - **Atomic :**
    - Transactions are atomic – they don’t have parts (conceptually)
    - Atomic can’t be executed partially; it should not be detectable that they interleave with another transaction

MANAGE DATABASE

- Properties of DB transaction: (ACID)
  - **Atomic :**
    - Examples
      - **Withdrawing money from your account**
        - A transaction that happens completely or not at all.
        - No partial results
        - Cash machine hands you cash and deducts amount from your account.
      - **Making an airline reservation**
        - Airline confirms your reservation and – Reduces number of free seats.
        - Charges your credit card
        - increases number of meals loaded on flight (sometimes)

MANAGE DATABASE

- Properties of DB transaction: (ACID)
  - Consistency
    - Data is in a consistent state when a transaction starts and when it ends.
    - For example, in an application that transfers funds from one account to another, the consistency property ensures that the total value of funds in both the accounts is the same at the start and end of each transaction.

MANAGE DATABASE

- Properties of DB transaction: (ACID)
  - Isolation
    - The intermediate state of a transaction is invisible to other transactions. As a result, transactions that run concurrently appear to be serialized.
    - For example, in an application that transfers funds from one account to another, the isolation property ensures that another transaction sees the transferred funds in one account or the other, but not in both, nor in neither.

MANAGE DATABASE

- Properties of DB transaction: (ACID)
  - Durability
    - After a transaction successfully completes, changes to data persist and are not undone, even in the event of a system failure.
    - For example, in an application that transfers funds from one account to another, the durability property ensures that the changes made to each account will not be reversed.

PSEUDOCODE OR CODE

- Before writing a routine, you describe what that routine should do in plain text.
- So if we set out to write an error handling lookup routine, we'd first write it in **pseudocode**:

RATIONALE OF PSEUDOCODE

- Pseudocode makes reviews easier.
- Pseudocode supports the idea of iterative refinement.
- Pseudocode makes changes easier
- Pseudocode minimizes commenting effort.
- Pseudocode is easier to maintain than other forms of design documentation.

START TRANSACTION AND COMMIT STATEMENTS

Pseudocode for an ATM Transaction

```
START TRANSACTION
    Display greeting
    Get account number, pin, type, and amount
    SELECT account number, type and balance
    If balance is sufficient then
            UPDATE account by posting debit
            UPDATE account by posting credit
            INSERT history record
            Display final message and issue cash
    Else
            Write error message
    End If
    On error : ROLLBACK
COMMIT
```

START TRANSACTION AND COMMIT STATEMENTS

Pseudocode for an Airline Reservation Transaction

```
START TRANSACTION
    Display greeting
    Get reservation preferences
    SELECT departure and return flight records
    If reservation is acceptable then
        UPDATE seats remaining of departure flight record
        UPDATE seats remaining of return flight record
        INSERT reservation record
        Print ticket if requested
    End If
    On error : ROLLBACK
COMMIT
```

CONCURRENCY CONTROL

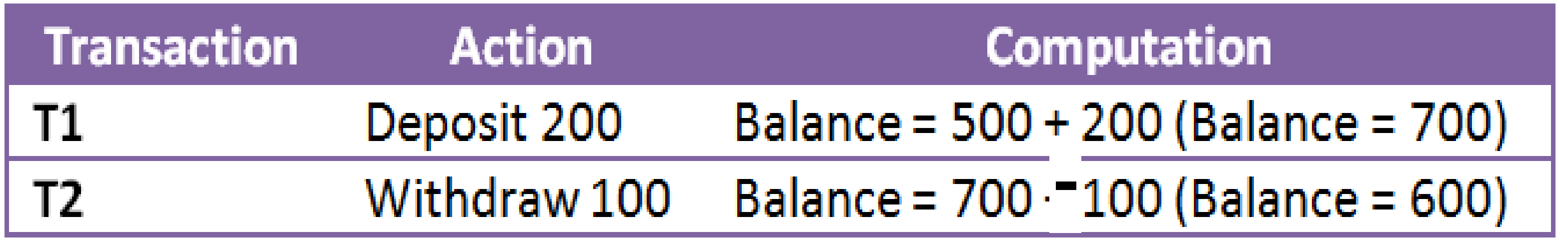
- Concurrency control refers to the coordination (synchronization) of execution of multiple transactions in a multi-user database environment.
- The DBMS must provide concurrency control to prevent problems associated with concurrent processing of transactions.
- The objective of concurrency control is to maximize transaction throughput while preventing interference among multiple users.

CONCURRENCY CONTROL

- Transaction throughput , the number of transactions processed per time unit, is a measure of the amount of work performed by a DBMS.
- There are three problems that can result because of simultaneous access to a database:
  - Lost update
  - Uncommitted dependency
  - Inconsistent retrieval

LOST UPDATE

- A concurrency problem in which one user’s update overwrites another user’s update.
- To illustrate lost updates let’s consider an account table where one of the attributes is the customer’s balance.
- Let’s assume the customer’s balance is RM500. Suppose if now two concurrent transactions T1 and T2 attempt to update the balance at the same time.
- Let’s assume T1 want to add RM200 and T2 wants to withdraw RM100 as shown below:

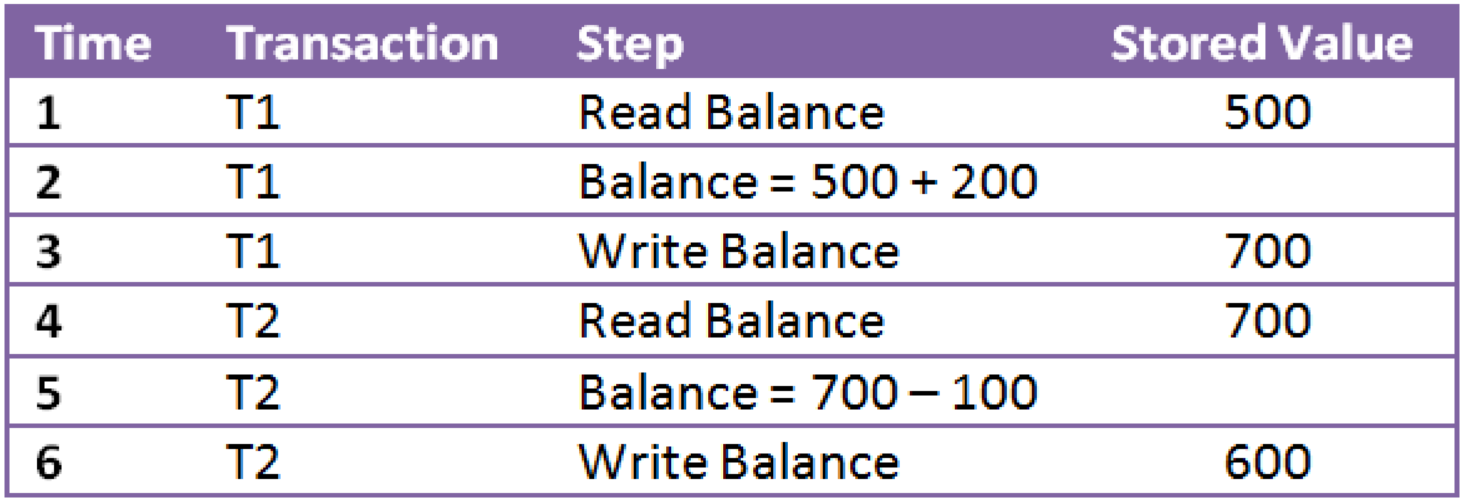


Transaction	Action	Computation
T1	Deposit 200	Balance = 500 + 200 (Balance = 700)
T2	Withdraw 100	Balance = 700 - 100 (Balance = 600)

CONT..

- A transaction requires several steps such as reading, modifying and writing.
- And these steps must be done in the correct sequence or order if two (or more) concurrent transactions attempt to update the same data item (i.e., at the same time)

CORRECT SEQUENCE



Time	Transaction	Step	Stored Value
1	T1	Read Balance	500
2	T1	Balance = 500 + 200	
3	T1	Write Balance	700
4	T2	Read Balance	700
5	T2	Balance = 700 – 100	
6	T2	Write Balance	600

But what if scheduler schedules these steps in a different sequence, such as in the following sequence..

INCORRECT SEQUENCE LEADING TO LOST UPDATE

Time	Transaction	Step	Stored Value
1	T1	Read Balance	500
2	T2	Read Balance	500
3	T1	Balance = 500 + 200	700
4	T2	Balance = 500 – 100	400
5	T1	Write Balance (Lost update)	700
6	T2	Write Balance	400

WRONG RESULT!!

❑This is because transaction T1 has not been committed before the execution of transaction T2.

❑Transaction T2 still uses the old value 500 and its abstraction yields 400.

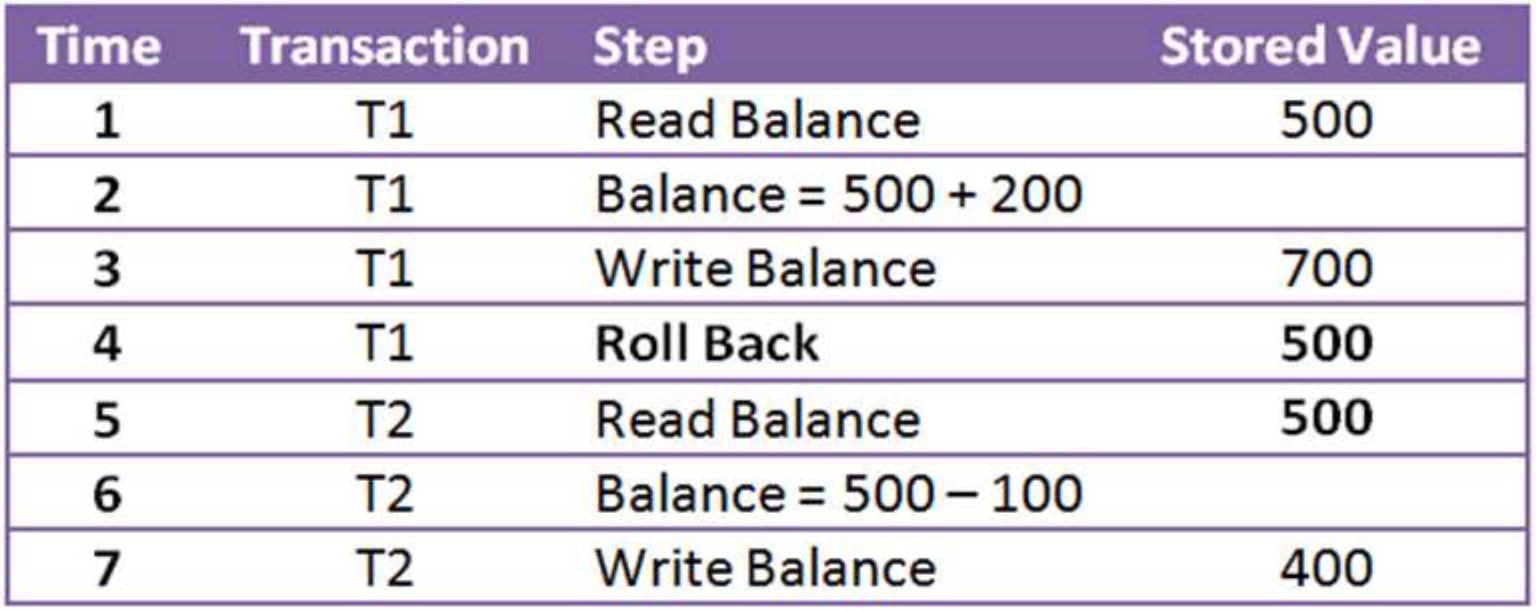
❑In the meantime, T1 writes the value 700 to the disk, which transaction T2 immediately overwrites.

❑The result is lost update – the deposit transaction is lost.

UNCOMMITTED DEPENDENCY

- An uncommitted dependency occurs when one transaction reads data written by another transaction before the other transaction commits.
- Data are not committed when two transactions T1 and T2 are executed concurrently, and the first transaction (T1) is rolled back after the second transaction (T2) has already accessed the uncommitted data, thus violating the isolation property of transactions.

CORRECT SEQUENCE



Time	Transaction	Step	Stored Value
1	T1	Read Balance	500
2	T1	Balance = 500 + 200	
3	T1	Write Balance	700
4	T1	**Roll Back**	**500**
5	T2	Read Balance	**500**
6	T2	Balance = 500 – 100	
7	T2	Write Balance	400

INCORRECT SEQUENCE LEADING TO UNCOMMITTED DEPENDENCY

Time	Transaction	Step	Stored Value
1	T1	Read Balance	500
2	T1	Balance = 500 + 200	
3	T1	Write Balance	700
4	T2	Read Balance (Uncommitted Data)	700
5	T1	Roll Back	500
6	T2	Balance = 700 – 100	
7	T2	Write Balance (Lost update)	600

INCONSISTENT RETRIEVAL

- An inconsistent retrieval occurs when a transaction calculates an aggregate or summary function (e.g., SUM) over a set of data, which the other transactions are updating.
- Occurs because the transaction may read some data *before* they are changed and read other data *after* they are changed.

Time Period 1-Transaction 1

```
SELECT SUM (BALANCE)
FROM ACCOUNT;
```

Time Period 2- Transaction T2

```
UPDATE ACCOUNT
   SET BALANCE = BALANCE -100
   WHERE ACCOUNT NUMBER = ‘777’;
UPDATE ACCOUNT
   SET BALANCE = BALANCE + 200
   WHERE ACCOUNT NUMBER = ‘888’;
COMMIT;
```

Account Number	Balance (Before T2)	Balance (After T2)
555	700	700
666	300	300
777	500	400(500-100)
888	600	800 (600+200)
999	900	900
SUM	3000	3100

- ❑Since the transaction T1 and T2 are processed at time periods, there is no inconsistent retrieval.
- ❑ However, the difference processing sequence could result in inconsistent retrieval as shown:

Time	Transaction	Step	Stored Value	Sum
1	T1	Read Balance for ‘555’	700	700
2	T1	Read Balance for ‘666’	300	1000
3	T2	Read Balance for ‘777’	500	
4	T2	Balance = 500 – 100		
5	T2	Write Balance for ‘777’	400	
6	T1	Read Balance for ‘777’	400	1400
7	T1	Read Balance for ‘888’	600	2000
8	T2	Read Balance for ‘888’	600	
9	T2	Balance = 600 + 200		
10	T2	Write Balance for ‘888’	800	
11	T1	Read Balance for ‘999’	900	2900

WRONG RESULT!!

CONCURRENCY CONTROL TOOLS

- Tools used by DBMS to prevent the interference problems:
  - Locks
  - Two-Phase Locking Protocol

LOCKS

- In locking, if one user updates the data, all other users are denied access until the update are completed.
- Locking Scenario….(next slide)
- The question arises:
  - What exactly should be locked if multiple users attempt to access data at same time – the entire db, a table, a page, a row, or a field?

LOCKING' SCENARIO

Time	Faizal's transaction	Amely's transaction
1	*Lock account balance*	
2	Read account balance (Balance = RM1000)	
3		Request account balance (Access is denied)
4	Withdraw RM600 (Balance = RM400)	
5	Write account balance (Balance = RM400)	
6	*Unlock account balance*	
7		*Lock account balance*
8		Read account balance (Balance = RM400)
9		Withdraw RM100 (Balance RM300)
10		Write account balance (Balance = RM300)
11		*Unlock account balance*

LOCKING LEVEL

- An important consideration in implementing concurrency control is choosing the locking level.
- This is called **lock granularity.**
- The LG specifies the extent of resources that will be locked when multiple users attempt to access the db.
- Locking Levels
  - Database Level
  - Table Level
  - Page Level
  - Row Level
  - Field Level

A.DATABASE LEVEL

- The entire db is locked and becomes unavailable to other users.
- This level may be used during backup or in batch systems.
- But for online, multi-user applications, this approach cannot be tolerated.
- It is too restrictive and will db performance adversely.

B.TABLE LEVEL

- The entire table containing the requested record is locked.
- This level may be used when the entire table or when or most of the records need to be updated such as when you want to reduce the price of all items by 12%.
- If the application requires access only to a single record, this level will also be very restricted.

C.PAGE LEVEL

- The operating system loads pages into memory as and when they are needed.
- A page has a fixed size such as 4K (or multiple of it).
- The number of records that can fit into a page will depend on the size of the record.
- A table typically spans several pages, and a page spans several records (rows).
- When a page is locked, other users are denied access to the records in that page.
- This is currently most widely used locking method in a multi-user db environment.

D.ROW LEVEL

- In this method, only the requested record is locked.
- All other records are available to other users.
- Although this level gives better performance, it requires high overhead since it requires a lock for *each* row in the table.

E.FIELD LEVEL

- In this method, the system allows concurrent access to the same row as long as they require the use of different fields in the same row.
- Although this level gives good performance, it also requires high overhead.

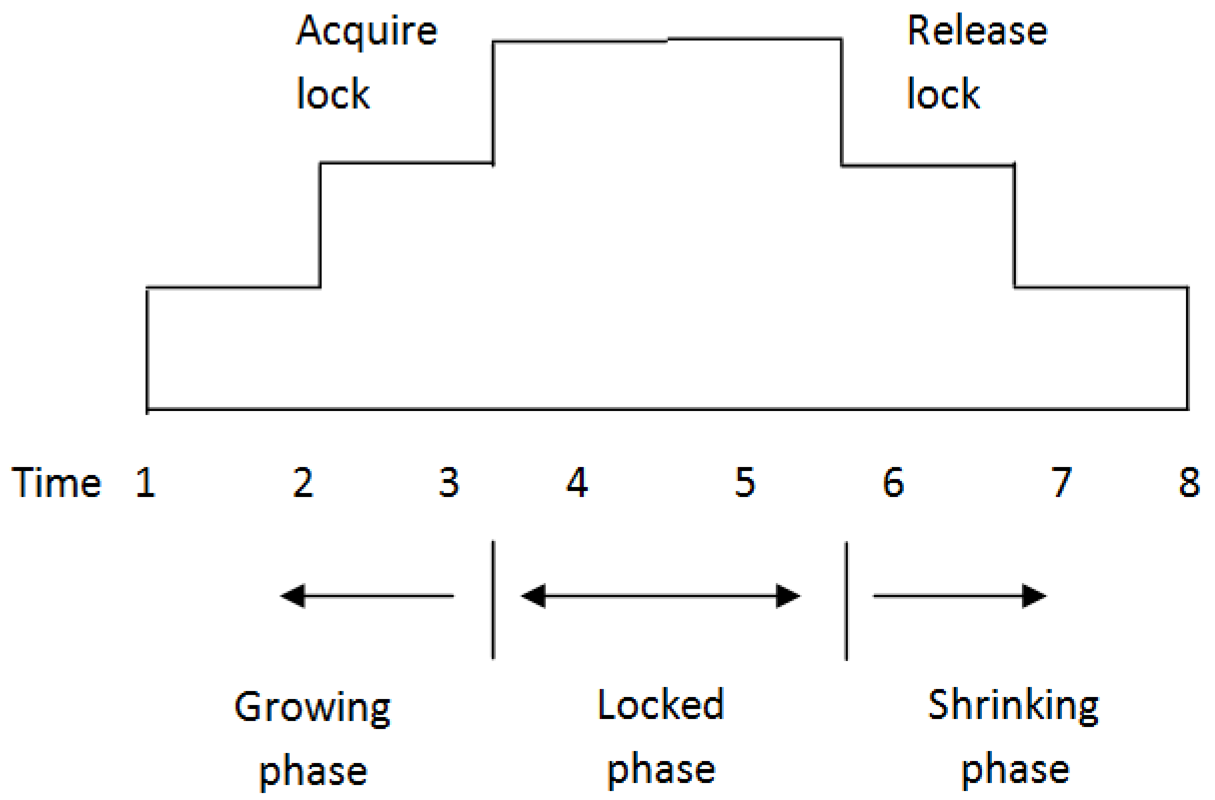
TWO-PHASE LOCKING PROTOCOL

- Definition of 2PL
  - Before reading or writing to a data item, the transaction must acquire the applicable lock to the data item.
  - Wait if a conflicting lock is held on the data item.
  - After releasing a lock, the transaction does not acquire any new locks.

CONT..

- The rules governing the two-phase locking protocol are as follows:
  - A growing phase in which a transaction acquires all the required locks but no unlocks. Once it acquires all the locks, it is in its locked point.
  - A shrinking phase in which a transaction have unlocks but cannot acquire any new locks.

TWO PHASE LOCKING PROTOCOL



CONT..

- Although the two-phase lock guarantees serializability, it may not prevent the occurrence of deadlocks.
- 2PL reduces concurrency because they hold locks longer.
- Both subject to deadlock

DEADLOCKS

- Deadlocks happen when two transactions are each waiting for the other
  - E.g.,each waiting for a lock that the other holds
- 2pl is not deadlock free
- Two possible approaches:
  - Deadlock prevention (don’t let it happen)
  - Deadlock detection (notice when it’s happened and take recovery action)

A DEADLOCK

Time	Faizal's transaction	Amely's transaction
1	*Place an S lock*	
2	Read account balance (Balance = RM1000)	
3		*Place an S lock*
4	*Request an X lock* (Access denied)	
5		Read account balance (Balance = RM1000)
6	(Wait)	*Read an X lock* (Access denied)
7		(Wait)

RECOVERY TOOLS

- Db failures do occur even in the best-designed systems.
- They may be caused by hardware failures, flaws in operating systems,
- Application programs, invalid data, computer viruses or human error.
- There are several approaches one can use to recovery from system failures:
  - Transaction log
  - Checkpoint
  - Database back-up

TRANSACTION LOG

- A transaction log provides a history of database changes.
- Every change to a database is also recorded in the log. The log is a hidden table not available to normal users.
- A typical log contains a unique log sequence number (LSN), a transaction identifier, a database action, a time, a row identifier, a column name and values (old and new).
- The recovery manager can perform two operations on the log : ***undo*** and ***redo*** operation.
- To undo/redo a transaction, the undo/redo operation is applied to all log records of a specified transaction except for the start and commit records.

DATABASE BACK-UP

- The backup facility in a DBMS makes copies of the db periodically.
- The frequency of the backup will depend on the needs of the application.
- Critical applications may need to be backed up every 10 minutes while less critical applications may only need to be backed up every hour or every day.
- It is also not necessary to backup up entire db each time; only the affected or modified parts of db need to be backed up.
- Making copies of the entire db frequently will affect system performance.

CHECKPOINT

- The DBMS uses the checkpoint facility to synchronize all files and journals periodically.
- When a system failure occurs, the DBMS rolls back to the last checkpoint and then uses the checkpoint information to restart the processing.
- To avoid a lot of rework, checkpoint operations should be performed as frequently as possible.
- Although these operations will affect system performance, they are nevertheless necessary to recover from db failures.