Department of UIC

# Database Management Systems

Pawandeep Sharma
Assistant Prof., UIC

# Syllabus

**Unit-I**

**Introduction:** Overview of Database Management System: Various views of data Models, Schemes and Introduction to database Languages & Environments, Advantages of DBMS over file processing systems, Responsibility of Database Administrator. Three level architecture of Database Systems: Introduction to client/Server architecture.

# Syllabus

## Unit-II

Data Models: E-R Diagram (Entity Relationship), mapping Constraints, keys, Reduction of E-R diagram into tables. Network & Hierarchical Models, File Organization: Sequential File, index sequential files, direct files, Hashing, B-trees Index files, Inverted Lists., Relational Models, Relational Algebra & various operations (set operations, select, project, join, division), Order, Relational calculus: Domain, Tuple, Well Formed Formula, specification, quantifiers, Introduction to Query Language, QBE.

# Syllabus

## Unit-III

Integrity constrains, functional dependencies & Normalization, 1st, 2nd, 3rd and BCNF.

Introduction to Distributed Data processing, Concurrency control: Transactions, Time stamping, Lock-based Protocols.

# Reference Books

- Fundamentals of Database Systems by R.Elmasri and S.B.Navathe, 3rd Edition, Pearson Education, New Delhi.
- An Introduction to Database Systems by C.J. Date, 7th Edition, Pearson Education, New Delhi.
- A Guide to the SQL Standard, Data, C. and Darwen, H.3rd Edition, Reading, Addison-Wesley Publications, New Delhi.
- Introduction to Database Management system by Bipin Desai, Galgotia Pub, New Delhi.
- Database System Concepts by A. Silberschatz, H.F.Korth and S.Sudarshan, 3rd Edition, McGraw-Hill, International Edition.
- SQL / PL/SQL, by Ivan Bayross, BPB Publications.

## *Functional Dependency & Normalization*

## *Topics to be covered*

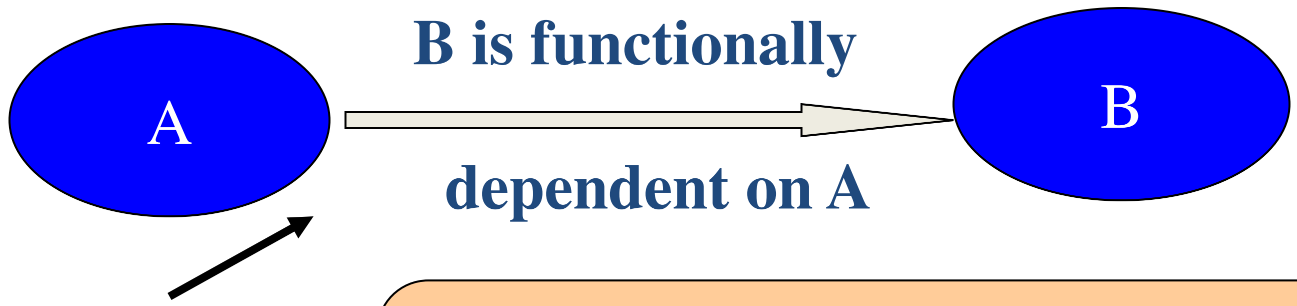
- Functional Dependency (FD)
- Normalization
- The Need of Normalization
- The Purpose of Normalization
- The Process of Normalization
- The Drawback of Normalization
- Denormalization

# Functional Dependency

Functional dependency describes the relationship between attributes in a relation.

For example, if $A$ and $B$ are attributes of relation $R$, and $B$ is functionally dependent on $A$ ( denoted $A \rightarrow B$), if each value of $A$ is associated with exactly one value of $B$. ( $A$ and $B$ may each consist of one or more attributes.)

# ***Functional Dependency***



Determinant

Refers to the attribute or group of attributes on the left-hand side of the arrow of a functional dependency

# Functional Dependencies

1. Functional dependency is a property of the meaning or semantics of the attributes in a relation. When a functional dependency is present, the dependency is specified as a constraint between the attributes.

2. Have a one-to-one relationship between attribute(s) on the left- and right- hand side of a dependency;

3. hold for all time;

# Functional Dependencies

## ❑Inference Rules

**Armstrong's axioms**

1. **Relfexivity:** If B is a subset of A, them A → B
2. **Augmentation:** If A → B, then AC → BC
3. **Transitivity:** If A → B and B → C, then A→ C
4. **Decomposition:** If A → B,C then A → B and A→ C
5. **Union:** If A → B and A → C, then A→ B,C
6. **Pseudo transitive**: if A→B,CB→D ,then CA→D

# Functional Dependencies

## ❑Minimal Sets of Functional Dependencies

A set of functional dependencies X is minimal if it satisfies the following condition:

❖ Every dependency in X has a single attribute on its right-hand side.

❖ We cannot replace any dependency A → B in X with dependency C →B, where C is a proper subset of A.

❖ We cannot remove any dependency from X.

## Types of functional dependency

- Full functional dependency indicates that if A and B are attributes of a relation, B is fully functionally dependent on A if B is functionally dependent on A, but not on any proper subset of A.

- Partial Functinal dependency A functional dependency A→B is partially dependent if there is some attributes that can be removed from A and the dependency still holds.

# Equivalence of sets of FD

- Two sets of Functional Dependency E and F are equivalent if $E^+ = F^+$ , Hence Equivalence means that every FD in E can be inferred from F and every FD in F can be inferred from E.

## Normalization

Normalization of data is a process of analyzing the given relation schemas based on their Functional Dependancy(FD) and Primary Key to achieve the desirable properties:

1. Minimizing Redundancy
2. Minimizing the Anomalies.

# Redundant Values in Tuples

EMP_DEPT

Redundancy

| ENAME | SSN | BDATE | DNUM | DNAME | DMSSN |
|---|---|---|---|---|---|
| John | 1234 | 1-10-1987 | 2 | Research | 9876 |
| Mark | 2345 | 2-12-1984 | 6 | Research | 8765 |
| Smith | 3456 | 3-2-1983 | 8 | Aministration | 7654 |
| Vilis | 4567 | 10-5-1982 | 7 | Headquater | 6543 |

## Update Anomalies

- Insertion Anomalies
- Deletion Anomalies
- Modification Anomalies

## *Concept of Null Values in Tuples*

- The Attribute does not apply to this tuple.
- The Attribute value for this tuple is unknown.
- The value is known but absent; It has not been recorded yet.

**EMP_DEPT**

| E_No | ENAME | BASIC |
|---|---|---|
| 1 | John | 12000 |
| 2 | tom | |
| 3 | | |
| 4 | Mark | 10050 |

## The Purpose of Normalization

Normalization is a technique for producing a set of relations with desirable properties, given the data requirements of an enterprise.

# The Process of Normalization

- Normalization is often executed as a series of steps. Each step corresponds to a specific normal form that has known properties.

- As normalization proceeds, the relations become progressively more restricted in format, and also less vulnerable to update anomalies.

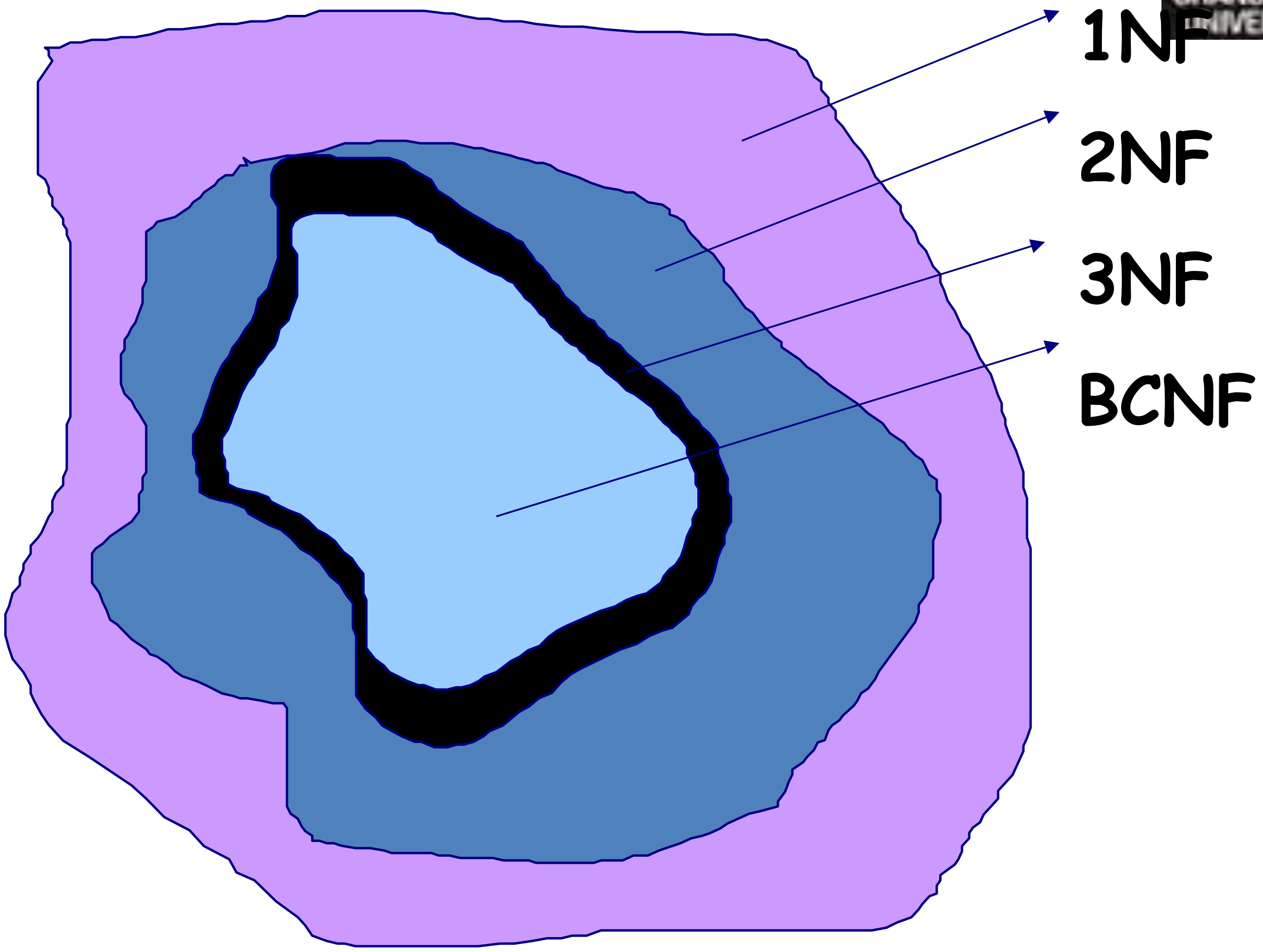
- For the relational data model, it is important to recognize that it is only first normal form (1NF) that is critical in creating relations. All the subsequent normal forms are optional.

- The Normal Form of a Relation refers to highest normal form condition that it meets, and hence indicates the degree to which it has been normalized.

Normal form includes two properties:

- ❖ Lossless Join Or Non-additive Join Property
- ❖ Dependency preservation property

# Overview of Normal Forms



## Definition of 1NF

First Normal Form is a relation in which the intersection of each row and column contains one and only one value.

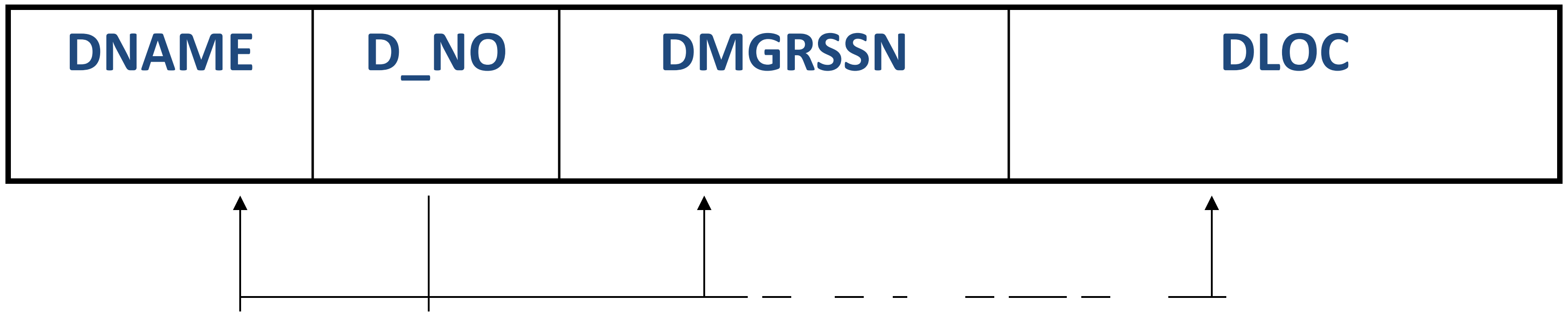
There are two approaches to removing repeating groups from unnormalized tables:

1. Removes the repeating groups by entering appropriate data in the empty columns of rows containing the repeating data.

2. Removes the repeating group by placing the repeating data, along with a copy of the original key attribute(s), in a separate relation. A primary key is identified for the new relation.

# First Normal Form (1NF)

DEPT



**(Example 1)**

# First Normal Form (1NF)

DEPARTMENT

(a)

| DNAME | D_No | DMGSSN | D_LOC |
|---|---|---|---|
| Research | 1 | 1234 | Victoria, Sidny |
| Administration | 2 | 2345 | Highton |
| Headquater | 3 | 3456 | Sidny |

DEPARTMENT

(b)

| DNAME | D_No | DMGSSN | D_LOC |
|---|---|---|---|
| Research | 1 | 1234 | Victoria |
| Research | 1 | 1234 | Sidny |
| Administration | 2 | 2345 | Highton |
| Headquater | 3 | 3456 | Sidny |

# Example of Nested Relations

**EMP_PROJ**

(a)

| SSN | ENAME | PROJS | |
|---|---|---|---|
| | | P_NO | HOURS |

**EMP_PROJ1**

| <u>SSN</u> | ENAME |
|---|---|
| | |

(b)

**EMP_PROJ2**

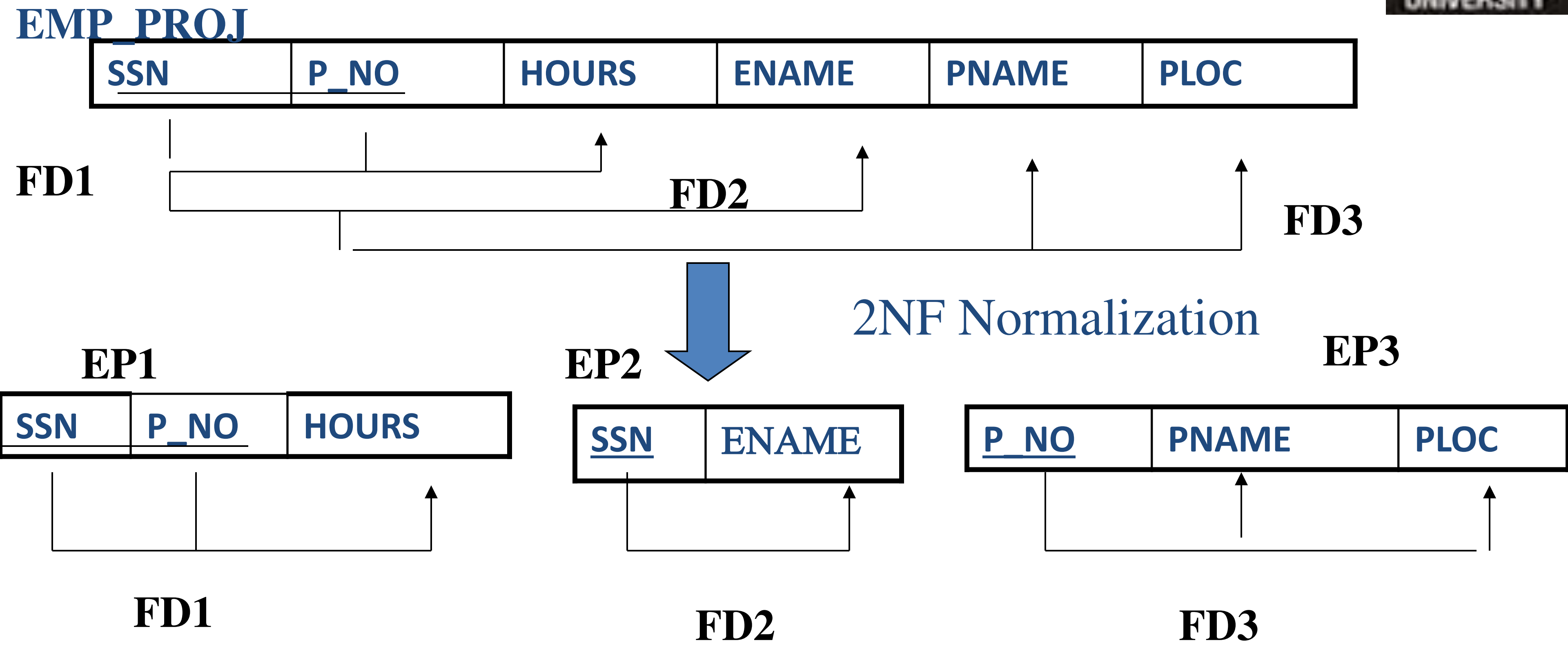
| <u>SSN</u> | <u>P_NO</u> | HOURS |
|---|---|---|
| | | |

(c)

## Second Normal Form (2NF)

Second normal form (2NF) is a relation that is in first normal form and every non-primary-key attribute is fully functionally dependent on the primary key.

# Second Normal Form (2NF)

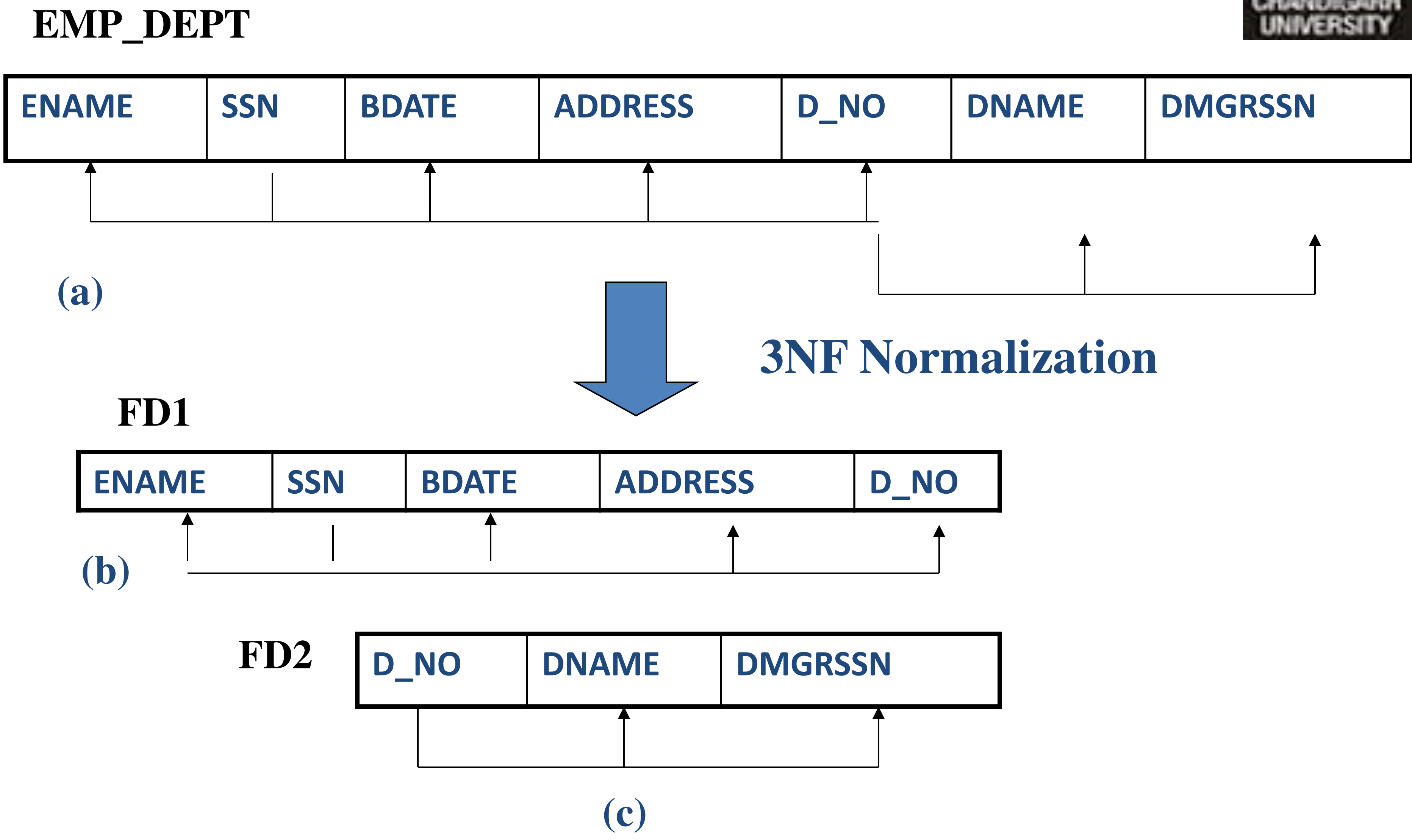


**Normalizing EMP_PROJ into 2NF Relation**

## Third Normal Form (3NF)

**Transitive dependency** A condition where A, B, and C are attributes of a relation such that if A → B and B → C, then C is transitively dependent on A via B.

# Third Normal Form (3NF)

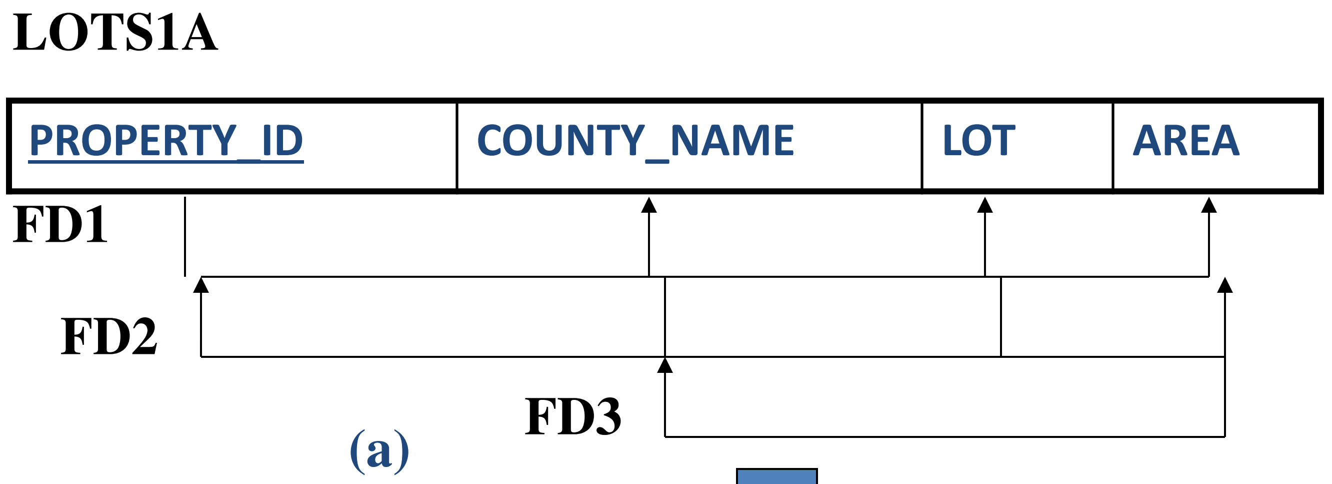


# Boyce-Codd Normal Form (BCNF)

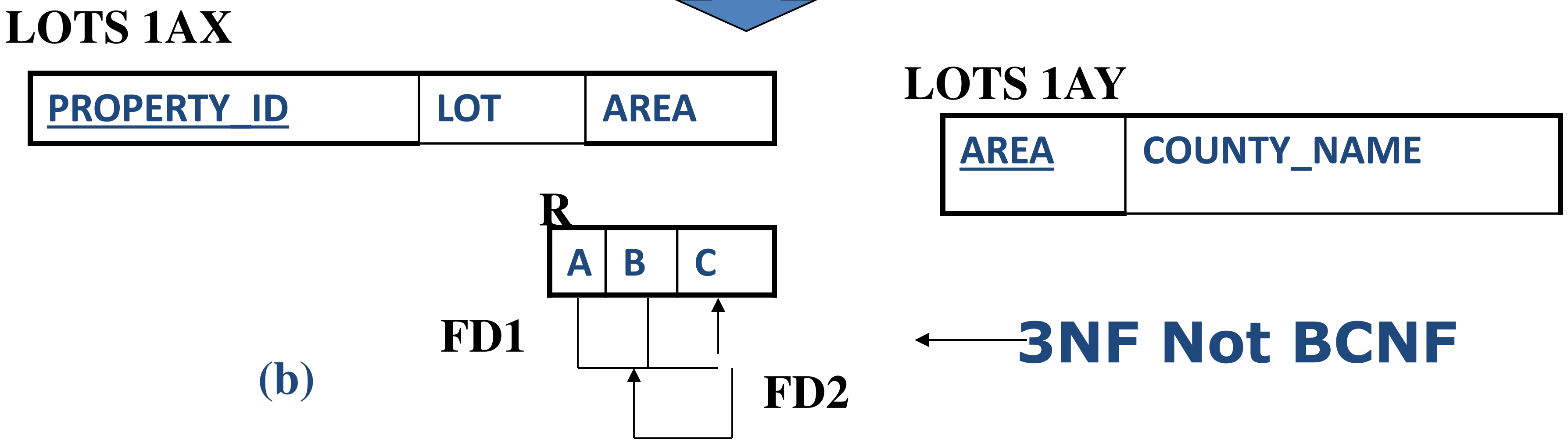
Boyce-Codd normal form (BCNF)A relation is in BCNF, if and only if, every determinant is a candidate key.

The difference between 3NF and BCNF is that for a functional dependency A → B, 3NF allows this dependency in a relation  if B is a primary-key attribute and A is not a candidate key, whereas BCNF insists that for this dependency to remain in a relation, A must be a candidate key.

# Boyce-Codd Normal Form (BCNF)



BCNF Normalization



3NF Not BCNF

QUERIES ??

## THANK YOU

Department of UIC

# Database Management Systems

Pawandeep Sharma
Assistant Prof., UIC

# Syllabus

**Unit-I**

**Introduction:** Overview of Database Management System: Various views of data Models, Schemes and Introduction to database Languages & Environments, Advantages of DBMS over file processing systems, Responsibility of Database Administrator. Three level architecture of Database Systems: Introduction to client/Server architecture.

# Syllabus

## Unit-II

Data Models: E-R Diagram (Entity Relationship), mapping Constraints, keys, Reduction of E-R diagram into tables. Network & Hierarchical Models, File Organization: Sequential File, index sequential files, direct files, Hashing, B-trees Index files, Inverted Lists., Relational Models, Relational Algebra & various operations (set operations, select, project, join, division), Order, Relational calculus: Domain, Tuple, Well Formed Formula, specification, quantifiers, Introduction to Query Language, QBE.

# Syllabus

## Unit-III

Integrity constrains, functional dependencies & Normalization, 1st, 2nd, 3rd and BCNF.

Introduction to Distributed Data processing, Concurrency control: Transactions, Time stamping, Lock-based Protocols.

## Reference Books

- Fundamentals of Database Systems by R.Elmasri and S.B.Navathe, 3rd Edition, Pearson Education, New Delhi.
- An Introduction to Database Systems by C.J. Date, 7th Edition, Pearson Education, New Delhi.
- A Guide to the SQL Standard, Data, C. and Darwen, H.3rd Edition, Reading, Addison-Wesley Publications, New Delhi.
- Introduction to Database Management system by Bipin Desai, Galgotia Pub, New Delhi.
- Database System Concepts by A. Silberschatz, H.F.Korth and S.Sudarshan, 3rd Edition, McGraw-Hill, International Edition.
- SQL / PL/SQL, by Ivan Bayross, BPB Publications.

## Lecture Contents

- Transaction and System Concepts
- Desirable Properties of Transactions
- Introduction to Transaction Processing
- Characterizing Schedules based on Recoverability
- Characterizing Schedules based on Serializability

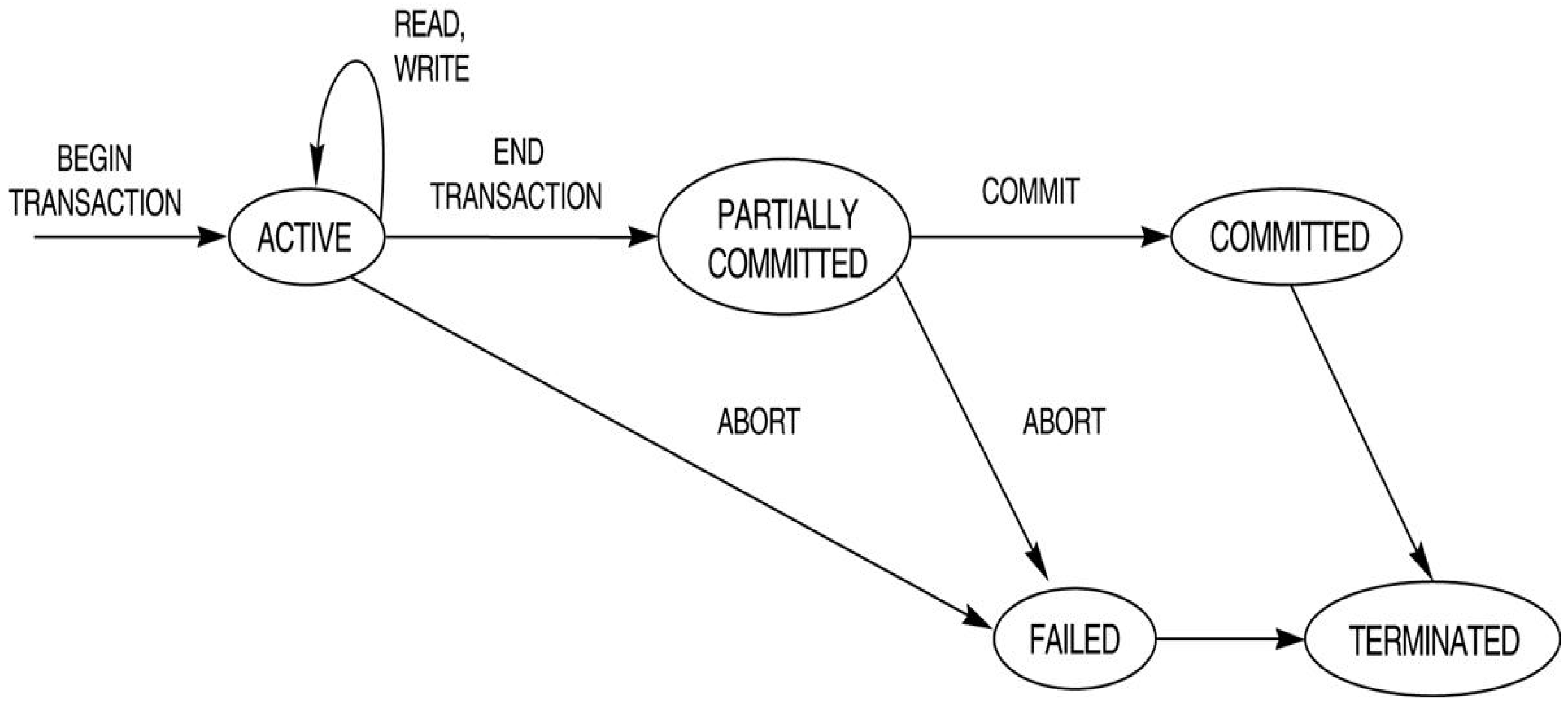
## Transaction Concepts

- A Transaction is a set of changes that must all be done together.
- A transaction is executed as a single unit of work.
- The consistency of system/database should remain unchanged before & after the transaction.

# Transaction states

- **Active state :** The Initial state. The transaction stays in this state while it is executing.
- **Partially committed state :**After the final statement has been executed.
- **Committed state:** After Successful completion.
- **Failed (aborted) state:** When the normal Execution can no longer proceed.
- **Terminated State:** After the transaction has been rolled back & database has been restored to its sate prior to the start of transaction.

## State transition diagram illustrating the states for transaction execution



Ms.Pawandeep Sharma,Assistant Professor
University Institute of computing
Uic.pawandeepsharma@gmail.com

## Transaction and System Concepts

**Commit Point of a Transaction:**

- **Definition:** A transaction T reaches its **commit point** when all its operations that access the database have been executed successfully *and* the effect of all the transaction operations on the database has been recorded in the log. Beyond the commit point, the transaction is said to be **committed,** and its effect is assumed to be *permanently recorded* in the database. The transaction then writes an entry [commit,T] into the log.
- **Roll Back of transactions:** Needed for transactions that have a [start_transaction,T] entry into the log but no commit entry [commit,T] into the log.

Ms.Pawandeep Sharma,Assistant Professor
University Institute of computing
Uic.pawandeepsharma@gmail.com

## Desirable Properties of Transactions

- *A - Atomicity*
- *C - Consistency Preservation*
- *I - Isolation*
- *D - Durability or Permanency*

## ACID properties:

- **Atomicity**: A transaction is an atomic unit of processing; it is either performed in its entirety or not performed at all. All or nothing

- **Consistency preservation**: A correct execution of the transaction must take the database from one consistent state to another. NO violation of integrity constraints

Ms.Pawandeep Sharma,Assistant Professor
University Institute of computing
Uic.pawandeepsharma@gmail.com

## ACID properties (cont.):

- **Isolation**: A transaction should not make its updates visible to other transactions until it is committed. Concurrent changes invisible
- **Durability or permanency**: Once a transaction changes the database and the changes are committed, these changes must never be lost because of subsequent failure. Committed update persists .

Ms.Pawandeep Sharma,Assistant Professor
University Institute of computing
Uic.pawandeepsharma@gmail.com

## The Transaction Processing

- **Single-User System:** At most one user at a time can use the system.
- **Multiuser System**: Many users can access the system concurrently.
- **Concurrency can be as :**
  - **Interleaved processing**: concurrent execution of processes is interleaved in a single CPU

    OR

  - **Parallel processing**: processes are concurrently executed in multiple CPUs.

## Transaction Concepts

The transaction is executed as series of READ & WRITE operations.

**READ OPERATION:**

To read a database object it is first brought into main memory from disk and then its value is copied into program variable.

**WRITE OPERATION :**

To write a database object, an in-memory copy of the object is first modified and then written to disk.

Ms.Pawandeep Sharma,Assistant Professor
University Institute of computing
Uic.pawandeepsharma@gmail.com

## Transaction Processing

For Example :

- **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.

## Two sample transactions. (a) Transaction $T_1$. (b) Transaction $T_2$.

(a) $T_1$

```
read_item (X);
X:=X-N;
write_item (X);
read_item (Y);
Y:=Y+N;
write_item (Y);
```

(b) $T_2$

```
read_item (X);
X:=X+M;
write_item (X);
```

## Concurrency Control

## Concurrency Control

- The concurrency control is the set of operations performed to control the execution of various concurrent processes so that execution of one process does not affect the other process occurring at some concurrent time.

## Why Concurrency Control is needed?

**The Lost Update Problem.**

This occurs when two transactions that access the same database items have their operations interleaved in a way that makes the value of some database item incorrect.

- **The Temporary Update (or Dirty Read) Problem.**

  This occurs when one transaction updates a database item and then the transaction fails for some reason . The updated item is accessed by another transaction before it is changed back to its original value.

- **The Incorrect Summary Problem .**

  If one transaction is calculating an aggregate summary function on a number of records while other transactions are updating some of these records, the aggregate function may calculate some values before they are updated and others after they are updated.

## ***CONCURRENCY CONTROL SCHEMES***

## 2 CONCURRENCY CONTROL TECHNIQUES ARE:

- ✓LOCKING
- ✓TIMESTAMPING

## Concurrency Control: LOCKS

- **Lock?**
  - A variable associated with a data item in database
  - describes the status of an item w.r.t. possible operations (e.g., read/write)
- **Two types of Locks**
  - Binary Locks
  - Shared/Exclusive Locks

## LOCKS IMPLEMENTATION

- **Lock can be implemented as a record with following fields**
  - Data Item
  - Lock variable
  - Locking Transactions
- **Lock Table**
  - Maintains the records of those items that are currently locked

- **Lock Manager**

  DBMS module that keep track of locks and queues.

## Binary Locks

- **Binary Locks**
  - Lock having two states (or values)
    - Lock(X)=1 (i.e., Item is not free)
    - Lock(X)=0 (i.e., item is free)
  - Uses two atomic operations to access the items
    - Lock_item(x)
    - Unlock_item(x)

## Lock_item(X):

```
A: if Lock(X)=0
  then Lock(X)=1
else begin
      wait (until Lock(X)=0 and
            the lock manager wakes up the transaction);
      go to A
      end;
```

## Unlock_item(X):

```
Lock(X)=0
if any  transactions are waiting then wakeup one of the waiting
transactions;
```

## Rules for binary locks

(1) Lock before first *read*(*X*) / *write*(*X*).

(2) Unlock after all *read*(*X*) / *write*(*X*).

(3) No *lock*(*X*) if *X* is already locked.

(4) *Unlock*(*X*) allowed only for the holder of *lock*(*X*)

## Problem with Binary Lock

- Binary locks are too restrictive
  - Only one Transaction can have a lock at any time

## Multiple-mode (Shared/Exclusive) Locks

- **Read-lock = shared lock**
  Any number of transactions may hold.
- **Write-lock = exclusive lock**
  Only a single *write-lock*(*X*) is allowed at a time; not even a simultaneous *read_lock*(*X*).
- **Unlocked**.

## read_lock (X):

```
          B:if lock(X)="unlocked"
          then begin lock(X)="read-locked";
                    no_of_reads(X)=1
          end
                    else begin wait(until lock(X)="unlocked" and
                    the lock manager wakes up the transaction);
          go to B
end;
```

## write_lock(X):

```
B: if lock(X)="unlocked"
        then lock(X)="write_locked"
else begin
        wait (until lock(X)="unlocked" and
                the lock manager wakes up the transaction);
go to B
end;
```

## unlock(X)

```
        if lock(X)="write_locked"
                then begin lock(X)="unlocked
                wakeup one of the waiting transactions, if any
                        end
if lock(X)="write_locked"
                then begin lock(X)="unlocked
                wakeup one of the waiting transactions, if any
                        end


                        end;
```

## Rules for multiple-mode locks

(1) ***Unlock***(*X*) after all ***read***(*X*) / ***write***(*X*).

(2) No ***read-lock***(*X*) if transaction already holds a ***read-lock***(*X*) or ***write-lock***(*X*).

(3) No ***write-lock***(*X*) if transaction already holds a ***read-lock***(*X*) or ***write-lock***(*X*).

(4) ***Unlock***(*X*) only if transaction holds a ***read-lock***(*X*) or ***write-lock***(*X*).

## Timestamp-Ordering Protocol

- Assigns a unique timestamp to each transaction
- Timestamp can be created
  - Counter
  - Current Date/time
- Timestamp order equals to serialization order
- Ensure conflict serializability
- Conflicts are detected when
  - $TS(T_i) < TS(T_j)$
- Deadlock free (Good)
- Livelock (Bad)

## The Timestamp Ordering Algorithm

- Order Transactions based on their TS$^s$ known as Timestamp Ordering (TO)
- TO should not violate the Serializability order
- Each data item X have two timestamps
  - Write_TS(X):
    - Refers to the largest timestamps of any transaction that successfully executed write_Item(X)
      - Write_TS(X)= TS(T)
  - Read_TS(X):
    - Refers to the largest timestamps of any transaction that successfully executed read_Item(X)
      - Read_TS(X)= TS(T)

## SERIALIZIBILITY

## Understanding Serializability

- **Serial schedule**: A schedule S is **serial** if, for every transaction T participating in the schedule, all the operations of T are executed consecutively in the schedule. Otherwise, the schedule is called **nonserial schedule.**
- **Serializable schedule**: A schedule S is **serializable** if it is equivalent to some serial schedule of the same n transactions.

Ms.Pawandeep Sharma,Assistant Professor
University Institute of computing
Uic.pawandeepsharma@gmail.com

## Characterizing Schedules based on Serializability

- **Result equivalent**: Two schedules are called result equivalent if they produce the same final state of the database.
- **Conflict equivalent**: Two schedules are said to be conflict equivalent if the order of any two conflicting operations is the same in both schedules.

# Characterizing Schedules based on Serializability

- Being serializable is <u>not</u> the same as being serial

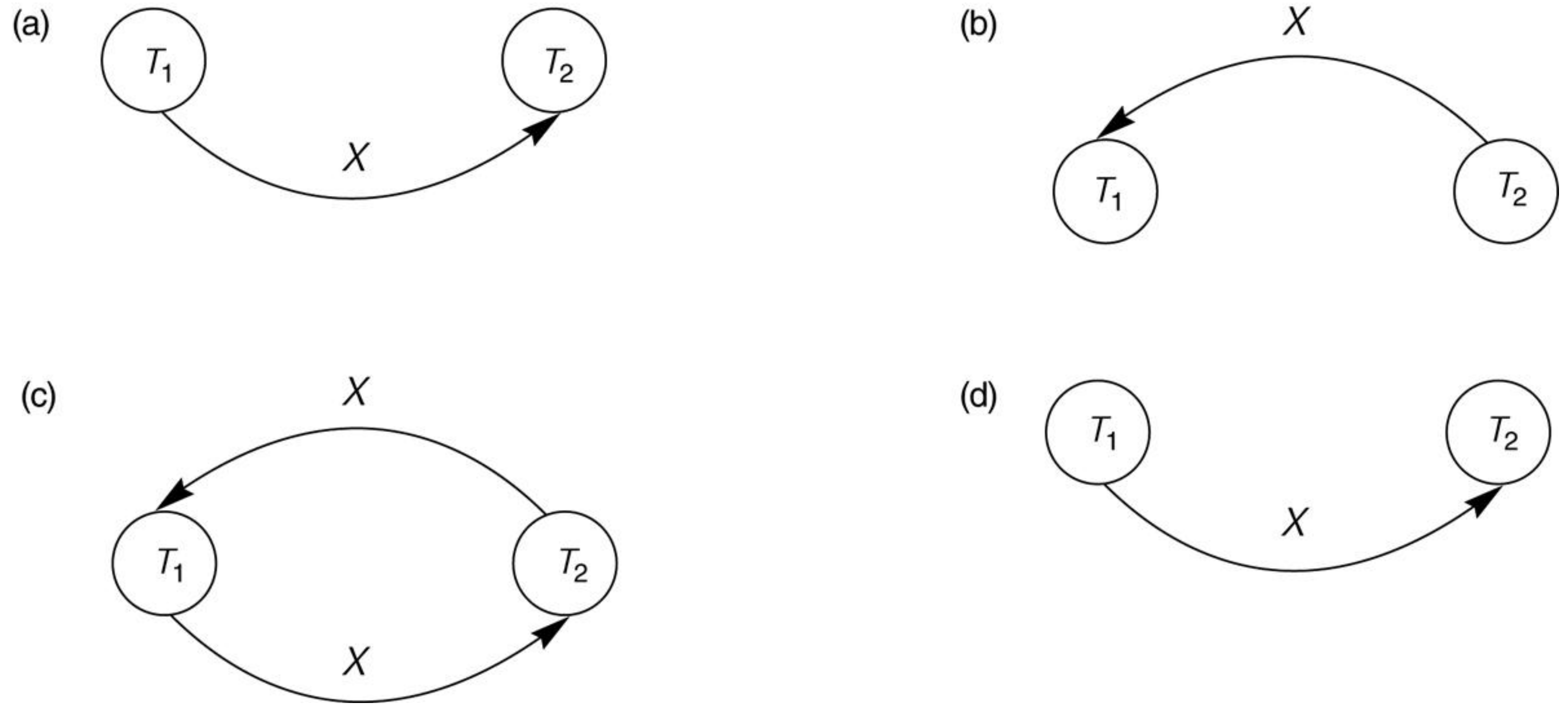
- Being serializable implies that the schedule is a <u>correct</u> schedule.
  - It will leave the database in a consistent state.
  - The interleaving is appropriate and will result in a state as if the transactions were serially executed.

## Testing for conflict serializability

**Algorithm 1.3.1:**

1. Looks at only read_Item (X) and write_Item (X) operations
2. Constructs a precedence graph (serialization graph) - a graph with directed edges
3. An edge is created from $T_i$ to $T_j$ if one of the operations in $T_i$ appears before a conflicting operation in $T_j$
4. The schedule is serializable if and only if the precedence graph has no cycles.

Constructing the precedence graphs for schedules *A* and *D* from Figure 17.5 to test for conflict serializability. (a) Precedence graph for serial schedule *A*. (b) Precedence graph for serial schedule *B*. (c) Precedence graph for schedule *C* (not serializable). (d) Precedence graph for schedule *D* (serializable, equivalent to schedule *A*).



QUERIES ??

## THANK YOU