

Course – DBMS

Course Instructor

Dr. K. SUBRAHMANYAM, professor
Department of CSE, KL University

Topics Covered in Todays Class

Unit 1:

- Characteristics of the Database Approach
- Advantages of Using the Database Approach
- When not to use a DBMS

Main Characteristics of the Database Approach

- 1. Self-describing nature** of a database system.
- 2. Insulation** between programs and data, and data manipulation.
- 3. Support of multiple views** of the data.
- 4. Sharing** of data and **multi-user** transaction processing

Main Characteristics of the Database Approach

1. Self-describing nature of a database system

What will be your description of the data stored in the following tables ?

1BM14CS001	Aditya	3	WP
1BM14CS002	Bharath	3	DS

1BM14CS001	1000
1BM14CS002	1000

1BM14CS001	S
1BM14CS002	B

Main Characteristics of the Database Approach

1. Self-describing nature of a database system.

Student_Details

USN	Name	Sem	Sub
1BM14CS001	Aditya	3	WP
1BM14CS002	Bharath	3	DS

Student_ExamFee_Details

USN	Amount
1BM14CS001	1000
1BM14CS002	1000

Student_Grade_Details

USN	Grade
1BM14CS001	S
1BM14CS002	B

Main Characteristics of the Database Approach

1. Self-describing nature of a database system.

Student_Details

USN	Name	Sem	Sub
1BM14CS001	Aditya	3	WP
1BM14CS002	Bharath	3	DS

Student_ExamFee_Details

USN	Amount
1BM14CS001	1000
1BM14CS002	1000

Student_Grade_Details

USN	Grade
1BM14CS001	S
1BM14CS002	B

Column_name	Data_Type	Belongs_to	Catalog
USN	Char(10)	Student_Details	Information stored in catalog is called
Name	Char(30)	Student_Details	meta-data which
Sem	Integer	Student_Details	Describes the structure
Sub	Char(2)	Student_Details	Of primary
Amount	Float	Student_ExamFee_Details	database
Grade	Char(1)	Student_Grade_Details	

Main Characteristics of the Database Approach

2. Insulation between programs and data, and data manipulation.

File Approach

student.txt

```
1BM14CS001 Aditya LA,Java,DBMS,OS,DC  
1BM14CS002 Baharth DBMS,OS,DC
```

```
fp=fopen("student.txt","r");  
while(fscanf(fp,"%s %s %s",USN,name,subjects)!=EOF)  
{  
printf("USN: %s Name: %s Subjects: %s",USN,name,subjects);  
}
```

What will be the Output of the above program statements ?

Main Characteristics of the Database Approach

2. Insulation between programs and data, and data manipulation.

File Approach

student.txt

```
Aditya 1BM14CS001 LA,Java,DBMS,OS,DC  
Baharth 1BM14CS002 DBMS,OS,DC
```

```
fp=fopen("student.txt","r");  
while(fscanf(fp,"%s %s %s",USN,name,subjects)!=EOF)  
{  
printf("USN: %s Name: %s Subjects: %s",USN,name,subjects);  
}
```

What will be the Output of the above program statements ?

Main Characteristics of the Database Approach

2. Insulation between programs and data, and data manipulation.

DBMS Approach

student

USN	Name	Subjects
1BM14CS001	Aditya	LA, Java, DBMS,OS,DC
1BM14CS002	Bharath	DBMS, OS, DC

```
select USN, Name, Subjects from student;
```



SQL **query** to retrieve
and display table
information

```
1BM14CS001 Aditya LA,Java,DBMS,OS,DC  
1BM14CS002 Bharath DBMS,OS,DC
```

Main Characteristics of the Database Approach

2. Insulation between programs and data, and data manipulation.

DBMS Approach

student

Name	USN	Subjects
Aditya	1BM14CS001	LA, Java, DBMS,OS,DC
Bharath	1BM14CS002	DBMS, OS, DC

select USN, Name, Subjects from student;

SQL **query** to retrieve and display table information



1BM14CS001 Aditya LA,Java,DBMS,OS,DC
1BM14CS002 Bharath DBMS,OS,DC

Main Characteristics of the Database Approach

3. Support of multiple views of the data

Student database			
Student_Details			
USN	Name	Sem	Sub
1BM14CS001	Aditya	3	WP
1BM14CS002	Bharath	3	DS

Student_ExamFee_Details	
USN	Amount
1BM14CS001	1000
1BM14CS002	1000

Student_Grade_Details	
USN	Grade
1BM14CS001	S
1BM14CS002	B



Accounts Section

1BM14CS001 Aditya 3 WP 1000
1BM14CS002 Bahart 3 DS 1000

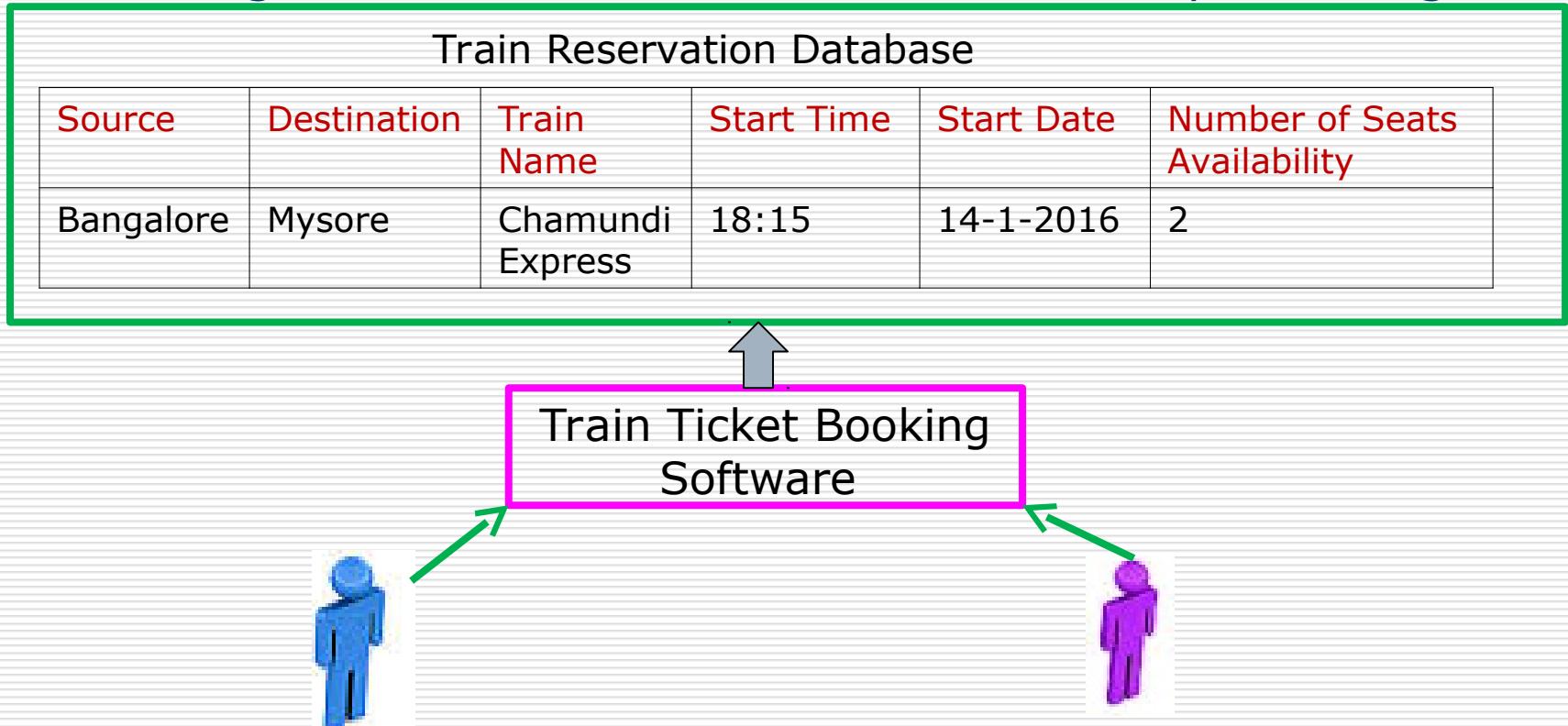


Examination Section

1BM14CS001 Aditya 3 WP S
1BM14CS002 Bahart 3 DS B

Main Characteristics of the Database Approach

4. Sharing of data and **multi-user** transaction processing



Main Characteristics of the Database Approach

- **Self-describing nature of a database system:**
 - A DBMS **catalog** stores the description of a particular database (e.g. data structures, types, and constraints)
 - The description is called **meta-data**.
 - This allows the DBMS software to work with different database applications.
- **Insulation between programs and data, and data manipulation:**
 - Called **program-data independence**.
 - Allows changing data structures and storage organization without having to change the DBMS access programs.
 - A **data model** is used to hide storage details and present the users with a conceptual view of the database.
 - Programs refer to the data model constructs rather than data storage details

Main Characteristics of the Database Approach

- ❑ **Support of multiple views of the data:**
 - ❑ Each user may see a different view of the database, which describes **only** the data of interest to that user.
- ❑ **Sharing of data and multi-user transaction processing:**
 - Allowing a set of **concurrent users** to retrieve from and to update the database.
 - *Concurrency control* within the DBMS guarantees that each **transaction** is correctly executed or aborted
 - *Recovery* subsystem ensures each completed transaction has its effect permanently recorded in the database
 - **OLTP** (Online Transaction Processing) is a major part of database applications. This allows hundreds of concurrent transactions to execute per second.

Advantages of Using the Database Approach

1. **Controlling redundancy** in data storage and in development and maintenance efforts.
2. Restricting **unauthorized** access to data.
3. Providing **persistent storage** for program Objects (data structures provided by DBMS & the programming languages were incompatible)
4. Providing Storage Structures (e.g. indexes) for **efficient Query Processing**
5. Providing **backup** and **recovery** services.
6. Providing **multiple** interfaces to **different classes of users**.
7. Representing **complex relationships** among data.
8. Enforcing **integrity constraints** on the database.
9. Drawing **inferences and actions** from the stored data using deductive and active rules

Understanding **integrity constraints** on the database

Database **Catalog** for following database tables

Column_name	Data_Type	Belongs_to
USN	Char(10)	Student_Details
Name	Char(30)	Student_Details
Sem	Integer	Student_Details
Sub	Char(2)	Student_Details
Amount	Float	Student_ExamFee_Details
Grade	Char(1)	Student_Grade_Details

Which of the following tables data storage is correct as per the above catalog definition ?

Student_Grade_Details

USN	Grade
1BM14CS001	S
1BM14CS002	B

Student_Grade_Details

USN	Grade
1BM14CS001	S
1BM14CS002	8

Understanding **integrity constraints** on the database

Student_Details

USN	Name	Sem	Sub
1BM14CS001	Aditya	3	WP
1BM14CS002	Bharath	3	DS

What is wrong in the following table data as per the **Student_Details** table ?

Student_Grade_Details

USN	Grade
1BM14CS001	S
1BM14CS002	B
1BM14CS003	C

Understanding drawing of “**inferences and actions**” from the stored data

Student_Grade_Details

USN	Grade
1BM14CS001	S
1BM14CS002	B
1BM14CS003	C
1BM14CS004	S
1BM14CS005	F
1BM14CS006	S

Using the above table data, Can you **infer**
How many Students have scored **S grade** and What are their USN's?

Understanding drawing of “**inferences and actions**” from the stored data

Student_Grade_Details

USN	Grade
1BM14CS001	S
1BM14CS002	B
1BM14CS003	C
1BM14CS004	S
1BM14CS005	F
1BM14CS006	S

Catalog

Column_name	Data_Type	Constraint
USN	Char(10)	
Grade	Char(1)	Should not be empty

Using the above **Catalog** information, Can you tell
Whether the following insert action on
Student_Grade_Details table is **Right or Wrong** ?

insert into **Student_Grade_Details** values (1BM14CS007);

When not to use a DBMS ?

When not to use a DBMS

- Main inhibitors (**costs**) of using a DBMS:
 - High initial investment and possible need for additional hardware.
 - Overhead for providing generality, security, concurrency control, recovery, and integrity functions.
- When a DBMS may be unnecessary:
 - If the database and applications are **simple**, well defined, and **not expected to change**.
 - If access to data by **multiple users is not required**.

Activity - Questionnaire

1. What are the four main Characteristics of DBMS ?
2. What is the meaning of "Catalog" w.r.t DBMS ?

Topics Covered in Todays Class

Unit 1: Database System Concepts and Architecture

- Data Models, Schemas and instances
- Three Schema Architecture and Data Independence
- Database language and interfaces
- The Database System Environment

Objective of todays class

Understanding the basic terminologies and definitions involved in **building Architecture of Database Systems.**

In this regard, First we will understand
Data Models, Schemas and Instances

Data Models

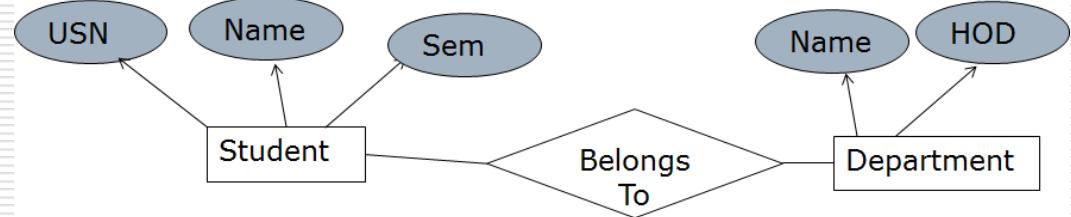
- **Data Abstraction** generally refers to the suppression of details of data organization and storage and the highlighting of the essential features for improved understanding of data.
- **Data Model:**
 - A set of concepts to describe the ***structure*** of a database, the ***operations*** for manipulating these structures, and certain ***constraints*** that the database should obey.
 - Data Models provides the necessary means to achieve data abstraction.

Categories of Data Models

- **Conceptual (high-level, semantic) data models:**
 - Provide concepts that are close to the way many users perceive data.
 - (Also called **entity-based** or **object-based** data models.)
- **Implementation (representational) data models:**
 - Provide concepts that fall in between high and low level, used by many commercial DBMS implementations (e.g. relational data models used in many commercial systems).
- **Physical (low-level, internal) data models:**
 - Provide concepts that describe details of how data is stored in the computer. These are usually specified in an ad-hoc manner through DBMS design and administration manuals

Categories of Data Models

Conceptual (high-level)



Implementation (Representational)

USN	Name	Sem	Dep
1BM14CS001	Aditya	3	CSE
1BM14IS002	Bharath	3	ISE

Dep	HOD
CSE	Dr. H S Guruprasad
ISE	Dr. Gowrishankar

Physical (low-level)

```
000000: 1d5c b639 0000 0000 0000 0000 0000 .1.9.....
0000010: 0000 0005 38af 6135 0008 0000 0000 0000 ....8.a5.....
0000020: 0000 0000 002f 0000 002f 0000 0000 0000 ...../.../...
0000030: 0006 0000 0040 0000 0000 0000 0004 0000 .....8.
0000040: 0000 ffff ffff 0000 ffff ffff 0000 0000 .....
0000050: 0001 0000 0000 009e 0000 0000 009e 0000 .....
0000060: 0000 ffff ffff 0000 ffff ffff 0000 0000 .....
0000070: 0000 0000 0003 0000 0000 ffff ffff 0000 .....
0000080: ffff ffff 0000 0000 0001 0000 0002 0026 .....6.
0000090: 0000 0002 0026 0000 0000 0000 ffff .....6.
00000a0: ffff 0000 ffff ffff 0000 0000 0002 saff .....
00000b0: ffff ffff ffff ffff ffff ffff 0000 .....
00000c0: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000d0: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000e0: 0000 0000 0000 0000 0000 0000 0000 0000 .....
```

Schemas, Instance and Database State

- Database Schema:
 - The ***description*** of a database.
 - Includes descriptions of the database structure, data types, and the constraints on the database.
 - Schema Diagram:
 - An ***illustrative*** display of (most aspects of) a database schema
- Database State:
 - The actual data stored in a database at a ***particular moment in time***. This includes the collection of all the data in the database.
 - Also called **database instance** (or occurrence or snapshot).
 - The term *instance* is also applied to individual database components, e.g. *record instance*, *table instance*, *entity instance*

Schemas, Instance and Database State

□ Schema Diagram

Student

USN	Name	Sem	Dep
1BM14CS001	Aditya	3	CSE

Department

Dep	HOD
CSE	Chandan

Database State at time "X"

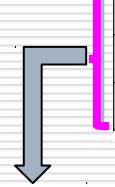
USN	Name	Sem	Dep
1BM14CS001	Aditya	3	CSE
1BM14IS002	Bharath	3	ISE



Database Instance

Database State at time "Y"

USN	Name	Sem	Dep
1BM14CS001	Aditya	3	CSE
1BM14IS002	Bharath	3	ISE
1BM14CS002	Chandan	3	CSE

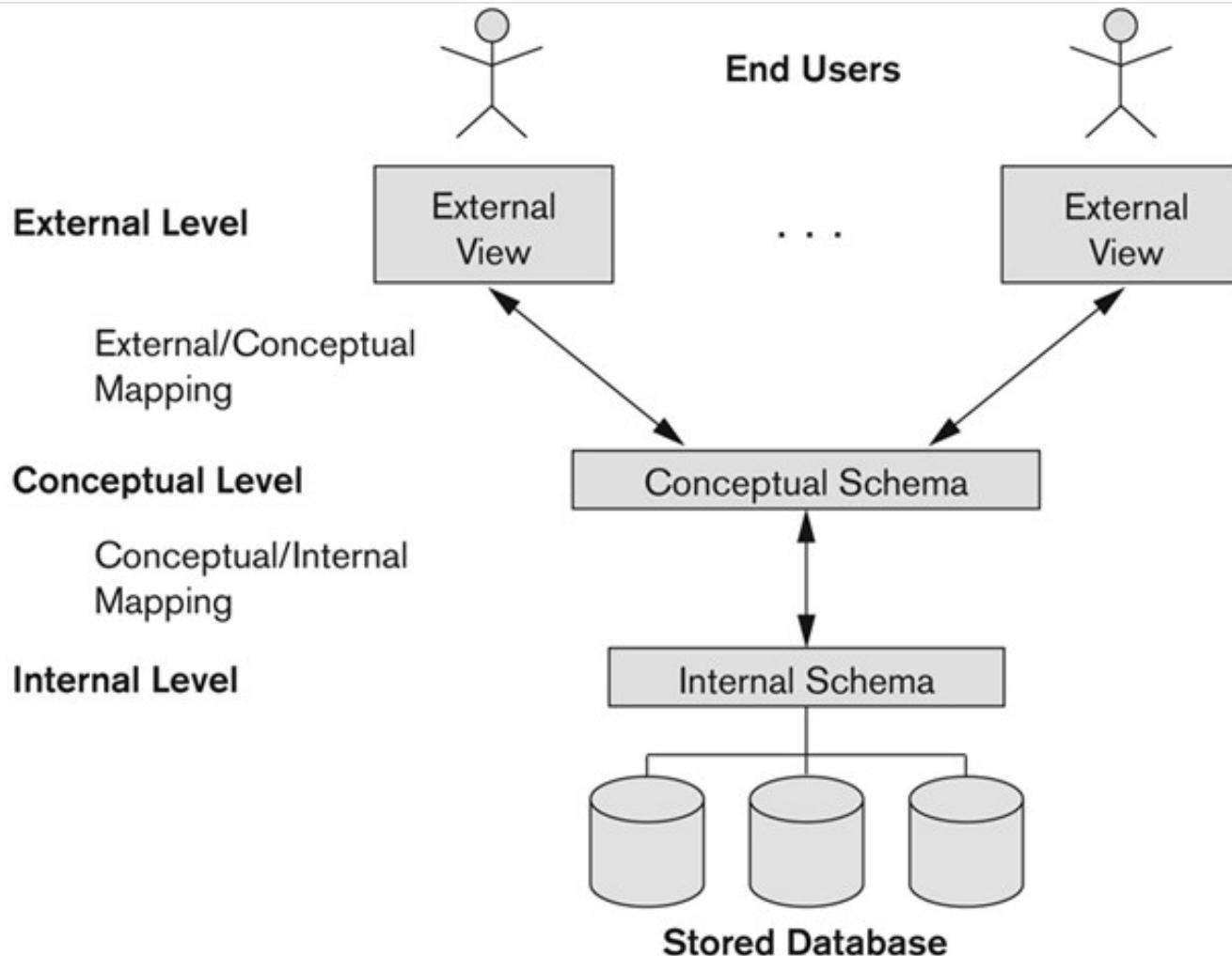


Database Instance

Three-Schema Architecture

- Defines DBMS schemas at **three** levels:
 - **Internal schema** at the internal level to describe physical storage structures and access paths (e.g indexes).
 - Typically uses a **physical** data model.
 - **Conceptual schema** at the conceptual level to describe the structure and constraints for the whole database for a community of users.
 - Uses a **conceptual** or an **implementation** data model.
 - **External schemas** at the external level to describe the various user views.
 - Usually uses the same data model as the conceptual schema.

The three-schema architecture



Data Independence

- **Logical Data Independence:**
 - The capacity to change the **conceptual schema** without having to change the **external schemas** and their associated application programs.
- **Physical Data Independence:**
 - The capacity to change the **internal schema** without having to change the conceptual schema.

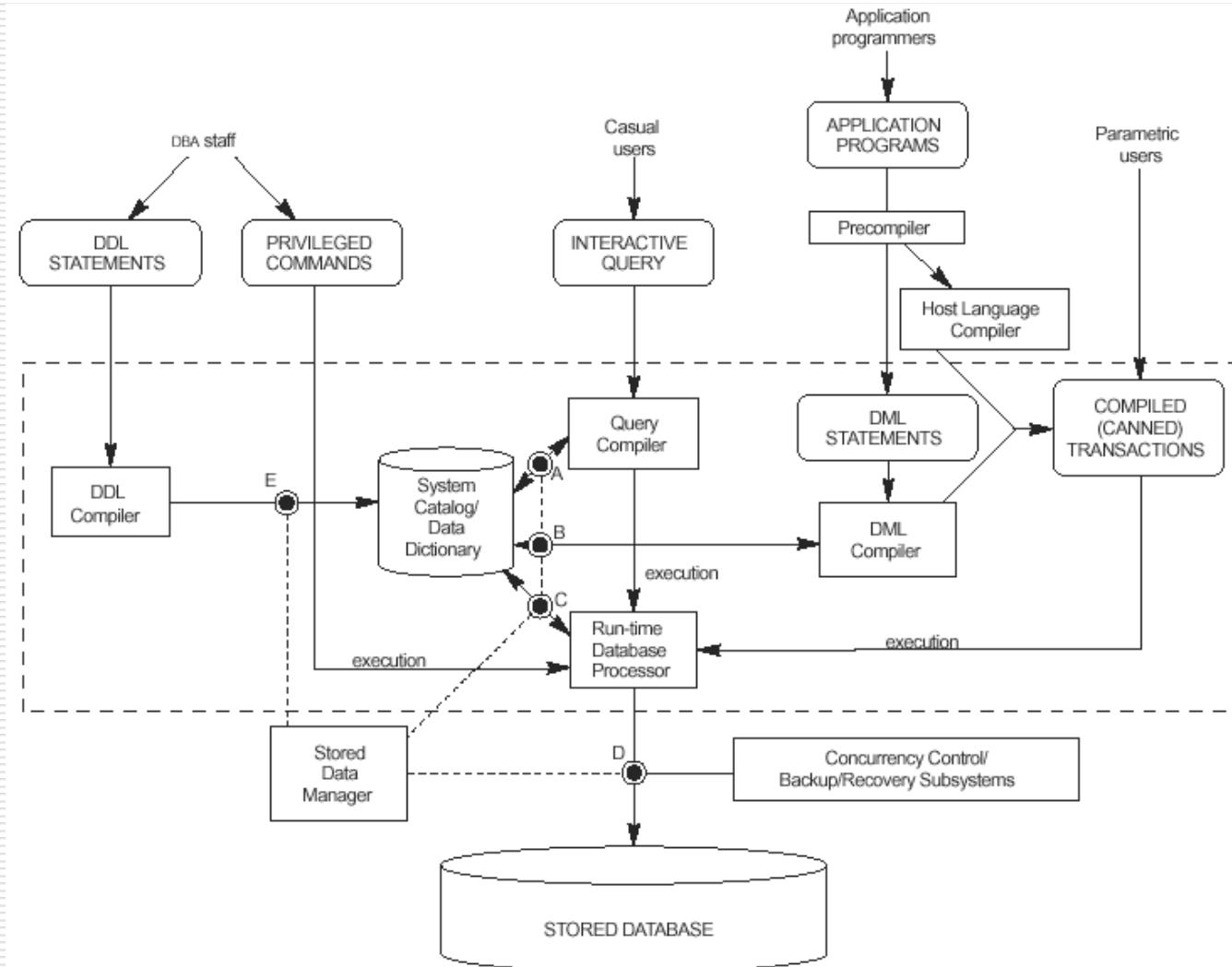
Review on Your Own

- Database language and interfaces
- The Database System Environment

Data Definition language and Interfaces

- DBMS Languages
 - Data Definition Language (DDL)
 - Storage Definition Language (SDL)
 - View Definition language (VDL)
 - Data manipulation Language (DML)
- Data Interfaces
 - Menu-based
 - Form-Based
 - Graphical User Interface
 - Natural language Interface
 - Speech Input and Output
 - Interfaces for Parametric users
 - Interfaces for DBA

The Database System Environment



Activity - Questionnaire

1. Mention three different levels in Three-Schema Architecture

Thanks for Listening

Summary

Student database

Student_Details

USN	Name	Sem	Sub
1BM14CS001	Aditya	3	WP
1BM14CS002	Bharath	3	DS

Student_ExamFee_Details

USN	Amount
1BM14CS001	1000
1BM14CS002	1000

Student_Grade_Details

USN	Grade
1BM14CS001	S
1BM14CS002	B



Accounts Section

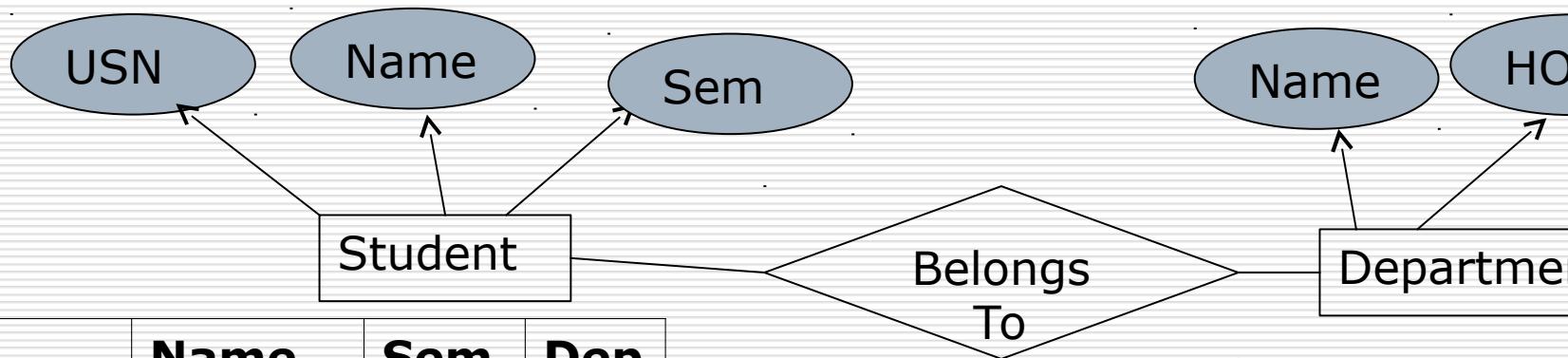
1BM14CS001 Aditya 3 WP 1000
1BM14CS002 Bahart 3 DS 1000



Examination Section

1BM14CS001 Aditya 3 WP S
1BM14CS002 Bahart 3 DS B

Categories of Data Models



USN	Name	Sem	Dep
1BM14CS001	Aditya	3	CSE
1BM14IS002	Bharath	3	ISE

Dep	HOD
CSE	Dr. H S Guruprasad
ISE	Dr. Gowrishankar

```
000000: 1d6c b639 0000 0000 0000 0000 0000 0000 .1.9.....
0000010: 0000 0005 38af 6135 0008 0000 0000 0000 ....8.a5...
0000020: 0000 0000 002f 0000 002e 0000 0000 0000 ...../.../
0000030: 0006 0000 0040 0000 0000 0000 0004 0000 .....8...
0000040: 0000 ffff ffff 0000 ffff ffff 0000 0000
0000050: 0001 0000 0000 009e 0000 0000 009e 0000 ...../.....
0000060: 0000 ffff ffff 0000 ffff ffff 0000 0000
0000070: 0000 0000 0003 0000 0000 ffff ffff 0000 ...../.....
0000080: ffff ffff 0000 0000 0001 0000 0002 0026 .....6...
0000090: 0000 0002 0026 0000 0000 0000 ffff .....6...
00000a0: ffff 0000 ffff ffff 0000 0000 0002 eaff
00000b0: ffff ffff ffff ffff ffff ffff 0000 ...../.....
00000c0: 0000 0000 0000 0000 0000 0000 0000 0000
00000d0: 0000 0000 0000 0000 0000 0000 0000 0000 ...../.....
00000e0: 0000 0000 0000 0000 0000 0000 0000 0000 ...../.....
```

USN	Name	Sem	Dep
1BM14CS001	Aditya	3	CSE
1BM14IS002	Bharath	3	ISE
1BM14CS002	Chandan	3	CSE

Session 4



K L University

u/s 3 of UGC Act. 1956
Koneru Lakshmaiah Education Foundation

Enhanced Entity Relationship (EER) Model



The Entity (Review)

- Entity Relationship (ER) Model – represents an object
 - Physical – person, car
 - Conceptual – school, company
- ER model is based on the perception of the real world as a collection of objects with attributes
- Attributes – describe the entity
 - Single, Multi-value
 - Composite, Simple
 - Derived, Stored

What is an EER Model?

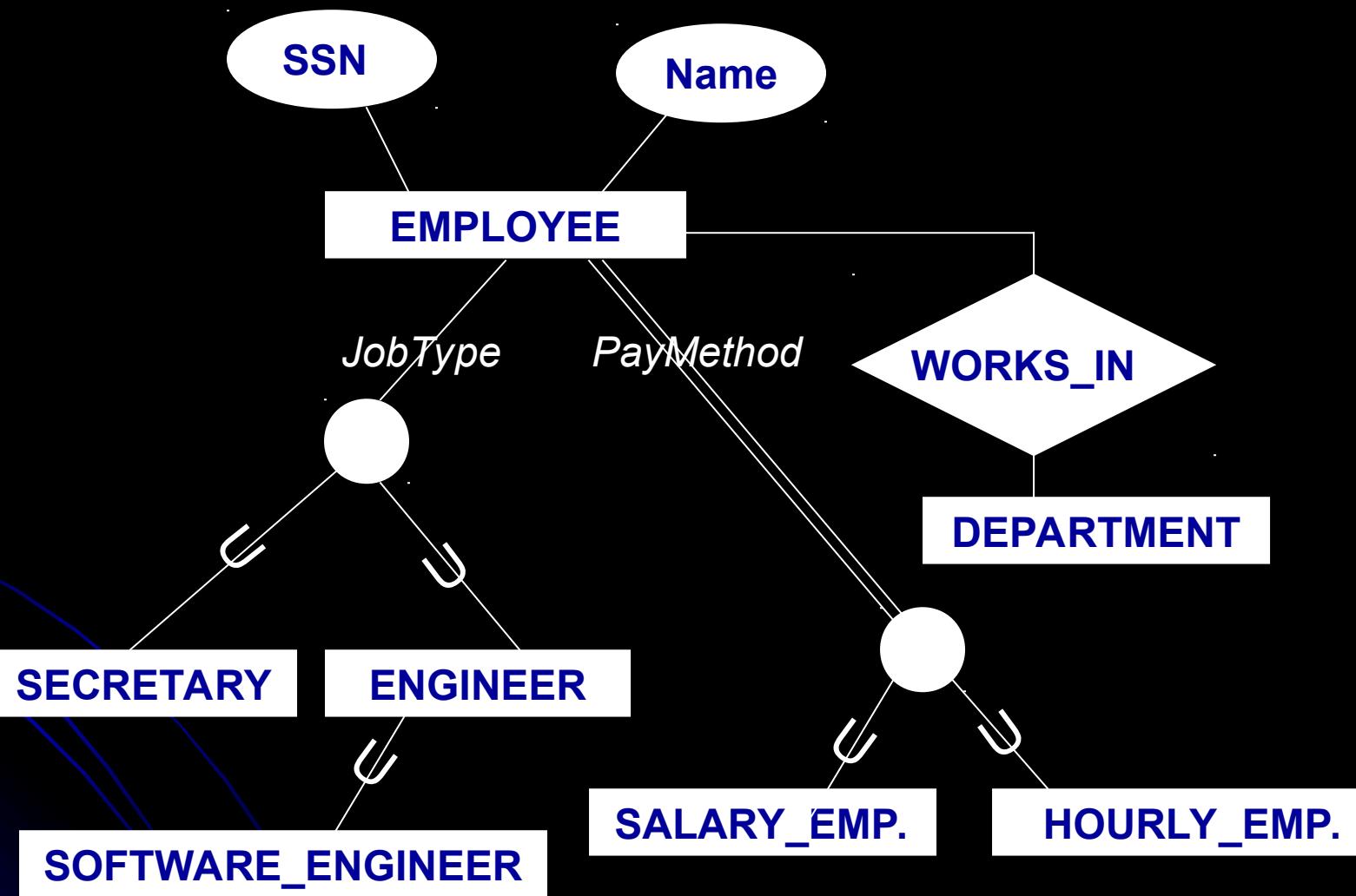
- Enhanced Entity Relationship (EER) – Data Modeling
- EER shows complex relationships between objects in a database (multimedia, geographical).
- Concepts of subclasses and superclasses, specializations and generalizations.
- Put concepts in diagram to form EER model

Specialization

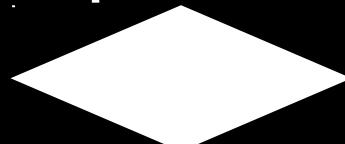
Subgrouping into subclasses (top-down approach)

- Example: EMPLOYEE -> SECRETARY
MANAGER, etc.
- Inheritance – Inherit attributes and relationships from superclass (Name, Birthdate, etc.)
- Subclasses may have unique attributes
 - SECRETARY has TypingSpeed attribute, MANAGER has BusinessUnitManaged, etc.

Specialization (cont.)



Model Shapes

- When you have more than one subclass based on the same defining attribute (*JobType*), use 
- To show class/subclass relationships, use 
 - Used for relationships between entity types
- To show relationship between two different entity types, use 

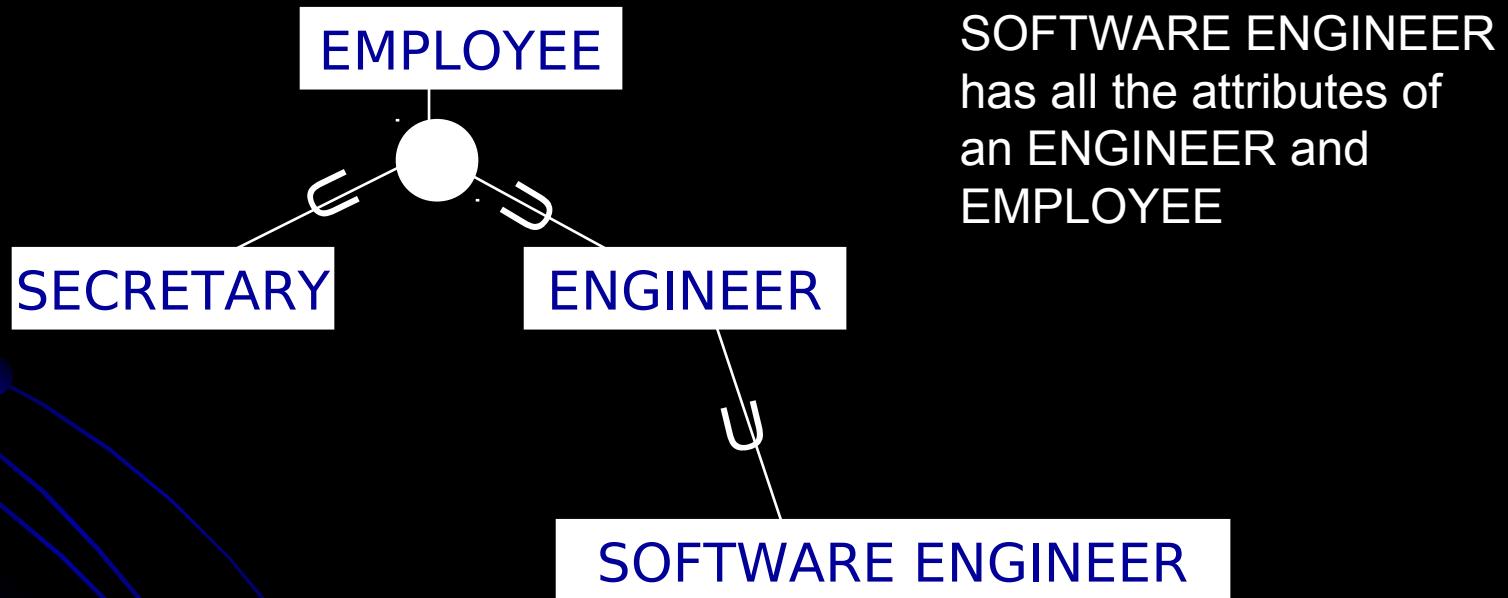
Generalization

Reverse processes of defining subclasses
(bottom-up approach)

- Bring together common attributes in entities
- Example: CAR (with attributes color, price, max speed) and TRUCK (with attributes color, price, tonnage) can be generalized into VEHICLE (with attributes color and price).

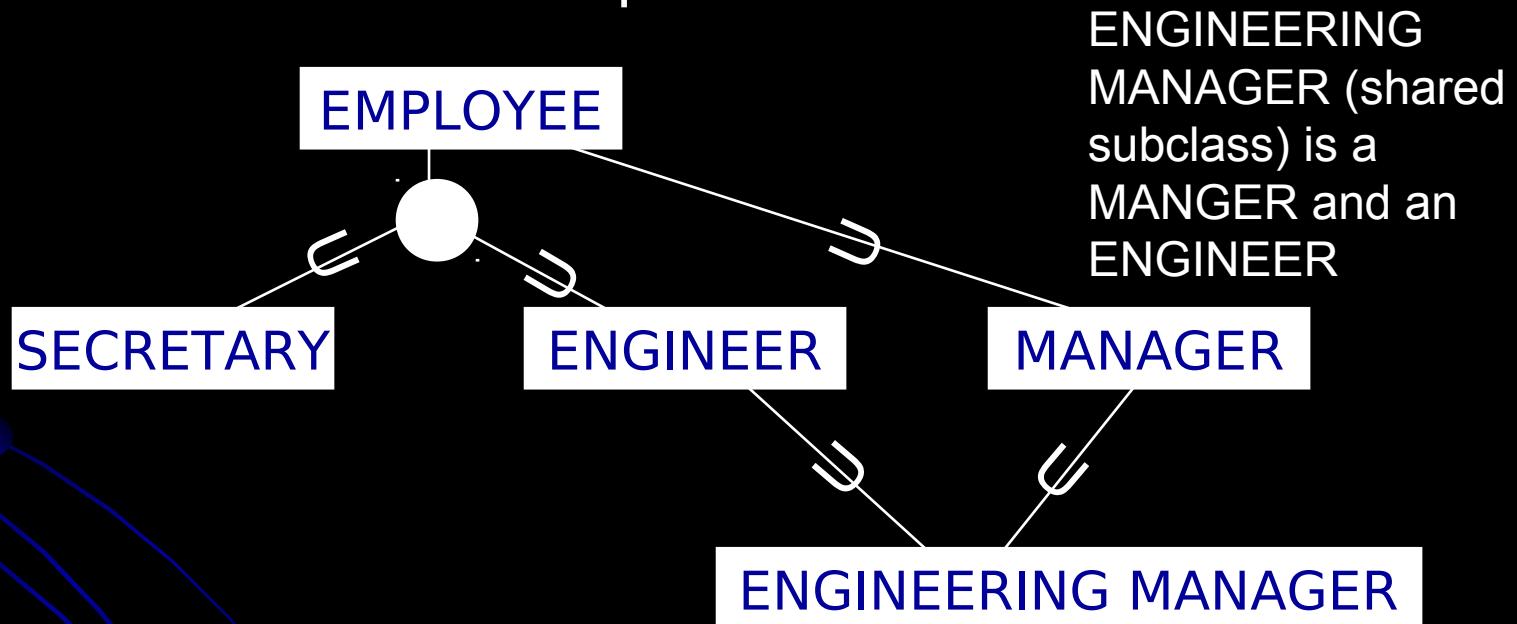
Hierarchies and Lattices

- Hierarchy – subclass participates in one class/subclass relationship



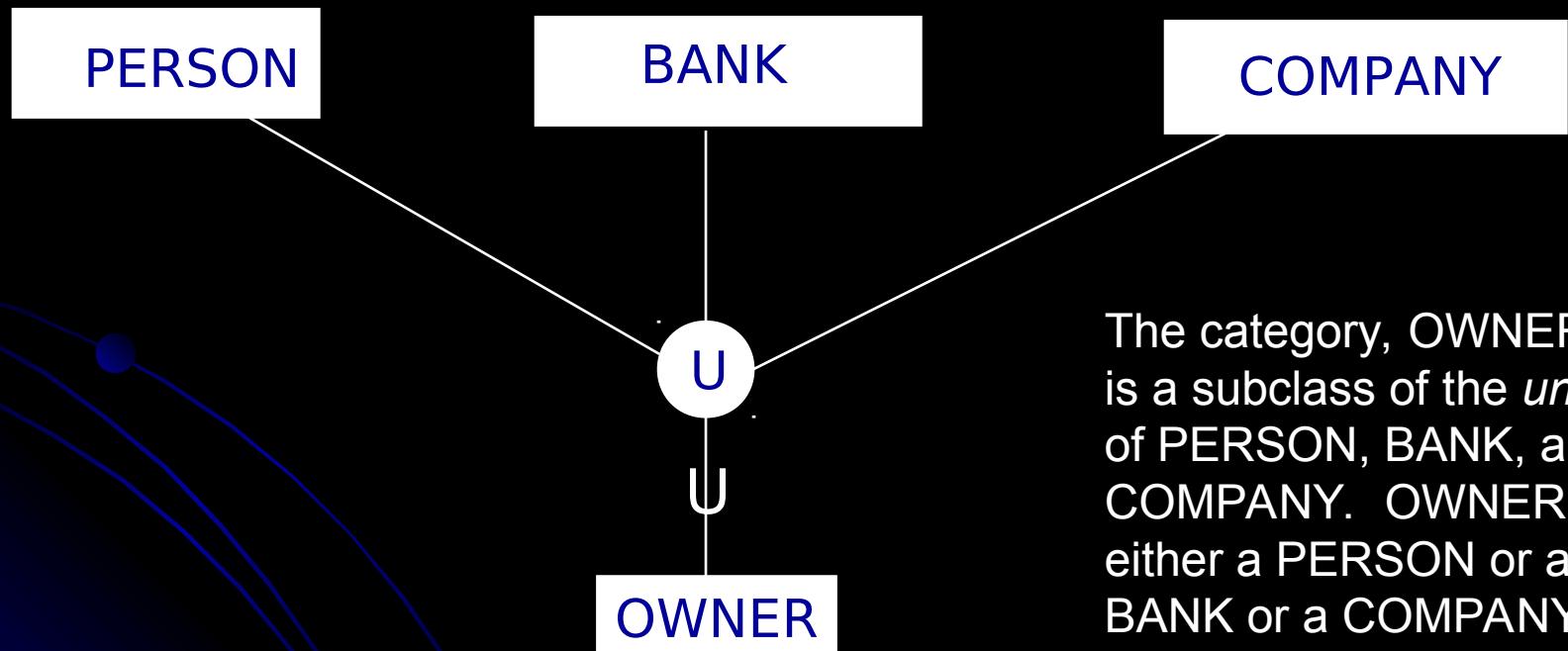
Hierarchies and Lattices

- Lattice – subclass participates in more than one class/subclass relationship



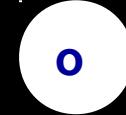
Categories

- Models a class/subclass with more than one superclass of *distinct* entity types. Attribute inheritance is selective.



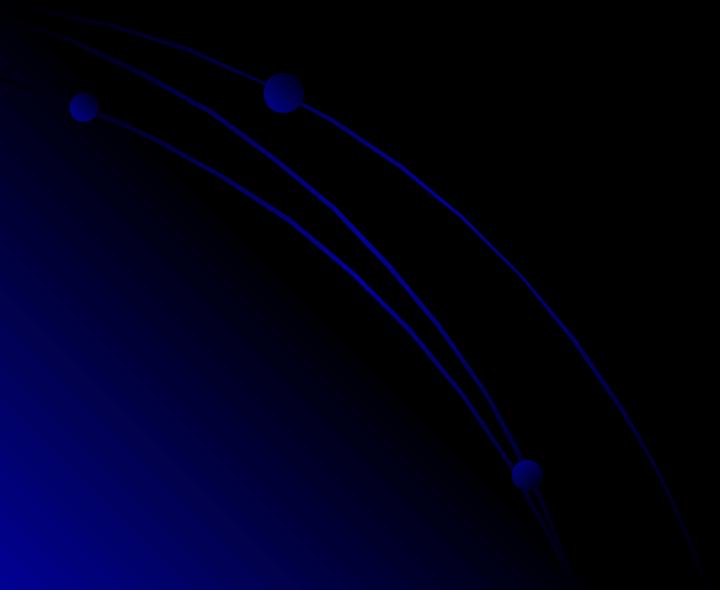
The category, OWNER, is a subclass of the *union* of PERSON, BANK, and COMPANY. OWNER is either a PERSON or a BANK or a COMPANY

Constraints

- Disjoint – an entity can be a member of at most one subclass of a specialization 
- Overlap – an entity may belong to more than one subclass of a specialization 
- Total specialization – each entity of a superclass belongs to some subclass of a specialization 
- Partial specialization – each entity of a superclass does not have to belong to some subclass of a specialization 

Putting It All Together

Example (Figure 4.7 in textbook)



Specialization / Generalization Lattice

Example (UNIVERSITY)

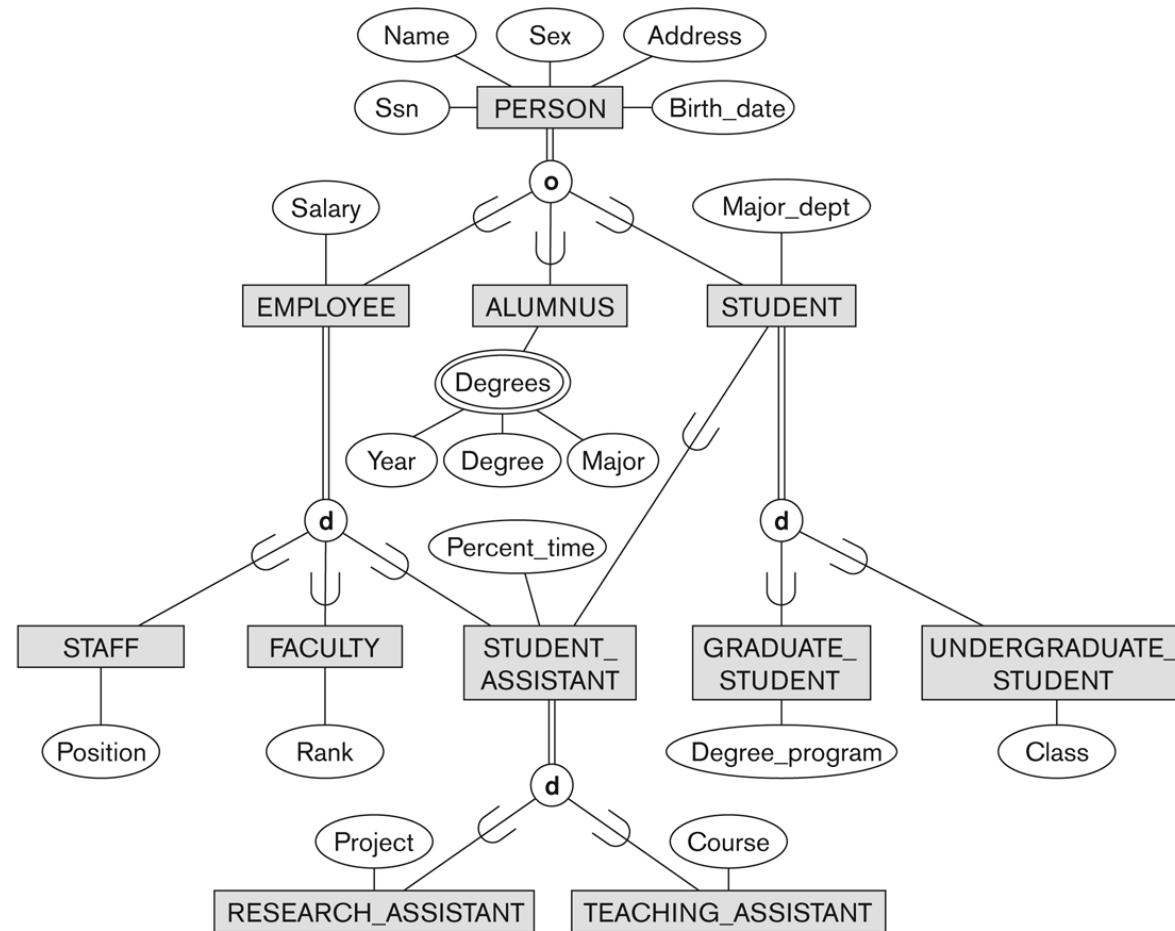


Figure 4.7

A specialization lattice with multiple inheritance for a UNIVERSITY database.

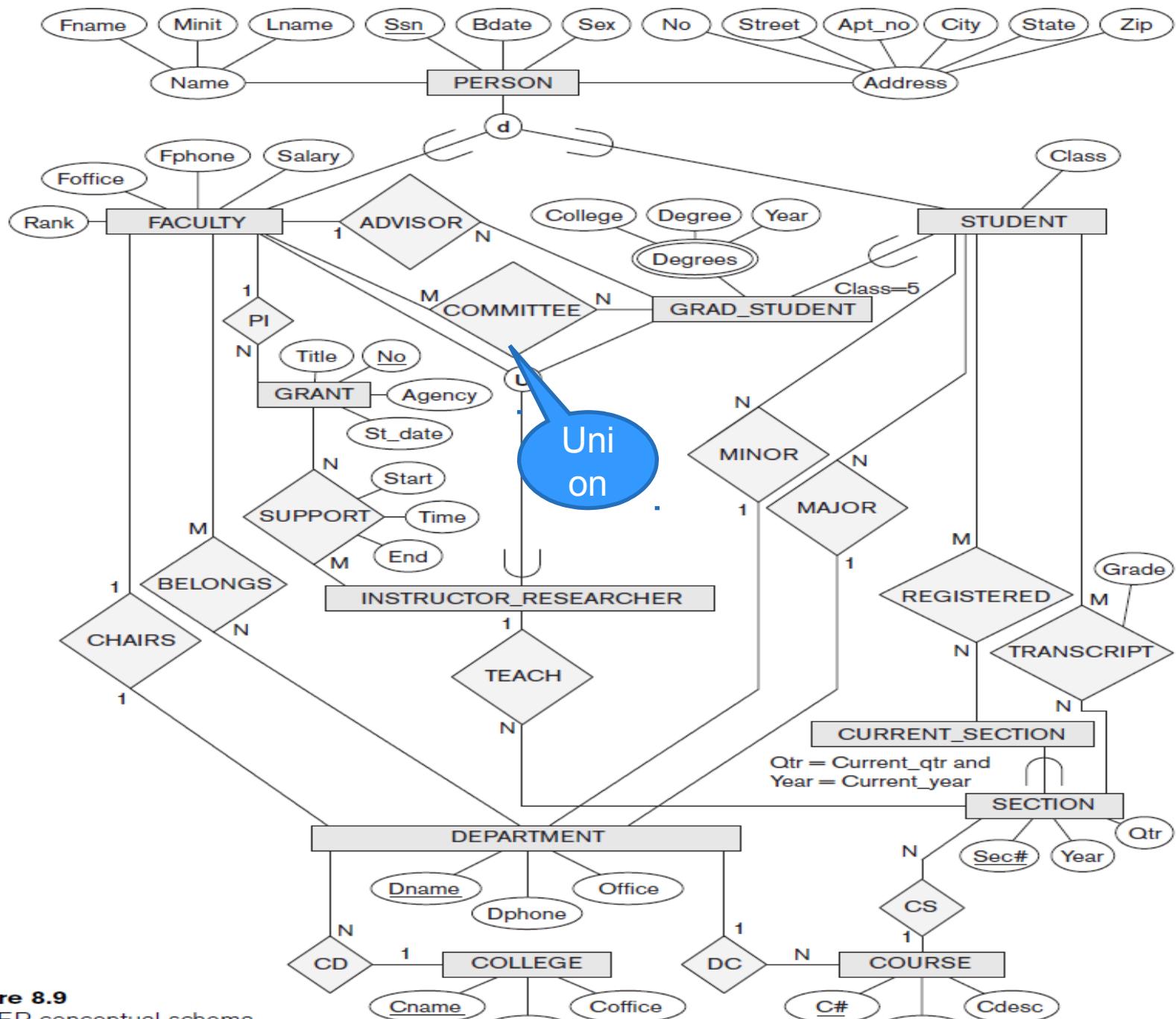


Figure 8.9
An EER conceptual schema

Normalization

What is Normalization?

- Database designed based on the E-R model may have some amount of
 - Inconsistency
 - Uncertainty
 - Redundancy

To eliminate these draw backs some **refinement** has to be done on the database.

- **Refinement** process is called **Normalization**
- Defined as a step-by-step process of decomposing a complex relation into a simple and stable data structure.
- The formal process that can be followed to achieve a good database design
- Also used to check that an existing design is of good quality
- The different stages of normalization are known as “normal forms”
- To accomplish normalization we need to understand the concept of Functional Dependencies.

Need for Normalization

Student_Course_Result Table

Student_Details			Course_Details				Result_Details		
101	Davis	11/4/1986	M4	Applied Mathematics	Basic Mathematics	7	11/11/2004	82	A
102	Daniel	11/6/1987	M4	Applied Mathematics	Basic Mathematics	7	11/11/2004	62	C
101	Davis	11/4/1986	H6	American History		4	11/22/2004	79	B
103	Sandra	10/2/1988	C3	Bio Chemistry	Basic Chemistry	11	11/16/2004	65	B
104	Evelyn	2/22/1986	B3	Botany		8	11/26/2004	77	B
102	Daniel	11/6/1987	P3	Nuclear Physics	Basic Physics	13	11/12/2004	68	B
105	Susan	8/31/1985	P3	Nuclear Physics	Basic Physics	13	11/12/2004	89	A
103	Sandra	10/2/1988	B4	Zoology		5	11/27/2004	54	D
105	Susan	8/31/1985	H6	American History		4	11/22/2004	87	A
104	Evelyn	2/22/1986	M4	Applied Mathematics	Basic Mathematics	7	11/11/2004	65	B

- Insert Anomaly
- Delete Anomaly
- Update Anomaly
- Data Duplication

Functional dependency

- In a given relation R, X and Y are attributes. Attribute Y is **functionally dependent** on attribute X if each value of X determines **EXACTLY ONE** value of Y, which is represented as $X \rightarrow Y$ (X can be composite in nature).
- We say here “x determines y” or “y is functionally dependent on x”
 $X \rightarrow Y$ does not imply $Y \rightarrow X$
- If the value of an attribute “Marks” is known then the value of an attribute “Grade” is determined since $\text{Marks} \rightarrow \text{Grade}$
- Types of functional dependencies:
 - Full Functional dependency
 - Partial Functional dependency
 - Transitive dependency

Functional Dependencies

Consider the following Relation

REPORT (STUDENT#,COURSE#, CourseName, IName, Room#, Marks, Grade)

- **STUDENT#** - Student Number
- **COURSE#** - Course Number
- **CourseName** - Course Name
- **IName** - Name of the Instructor who delivered the course
- **Room#** - Room number which is assigned to respective Instructor
- **Marks** - Scored in Course COURSE# by Student STUDENT#
- **Grade** - obtained by Student STUDENT# in Course COURSE#

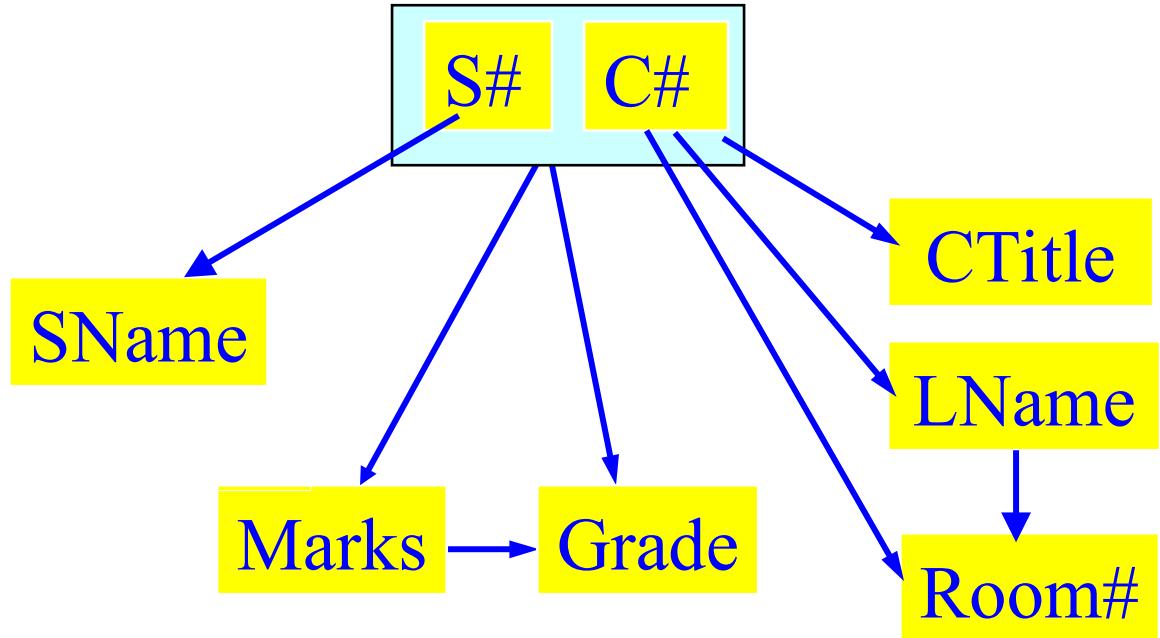
Functional Dependencies- From the previous example

- **STUDENT# COURSE# → Marks**
- **COURSE# → CourseName,**
- **COURSE# → IName (Assuming one course is taught by one and only one Instructor)**
- **IName → Room# (Assuming each Instructor has his/her own and non-shared room)**
- **Marks → Grade**

Dependency diagram

Report(S#,C#,SName,CTitle,LName,Room#,Marks,Grade)

- $S\# \rightarrow SName$
- $C\# \rightarrow CTitle$,
- $C\# \rightarrow LName$
- $LName \rightarrow Room\#$
- $C\# \rightarrow Room\#$
- $S\# C\# \rightarrow Marks$
- $Marks \rightarrow Grade$
- $S\# C\# \rightarrow Grade$



Assumptions:

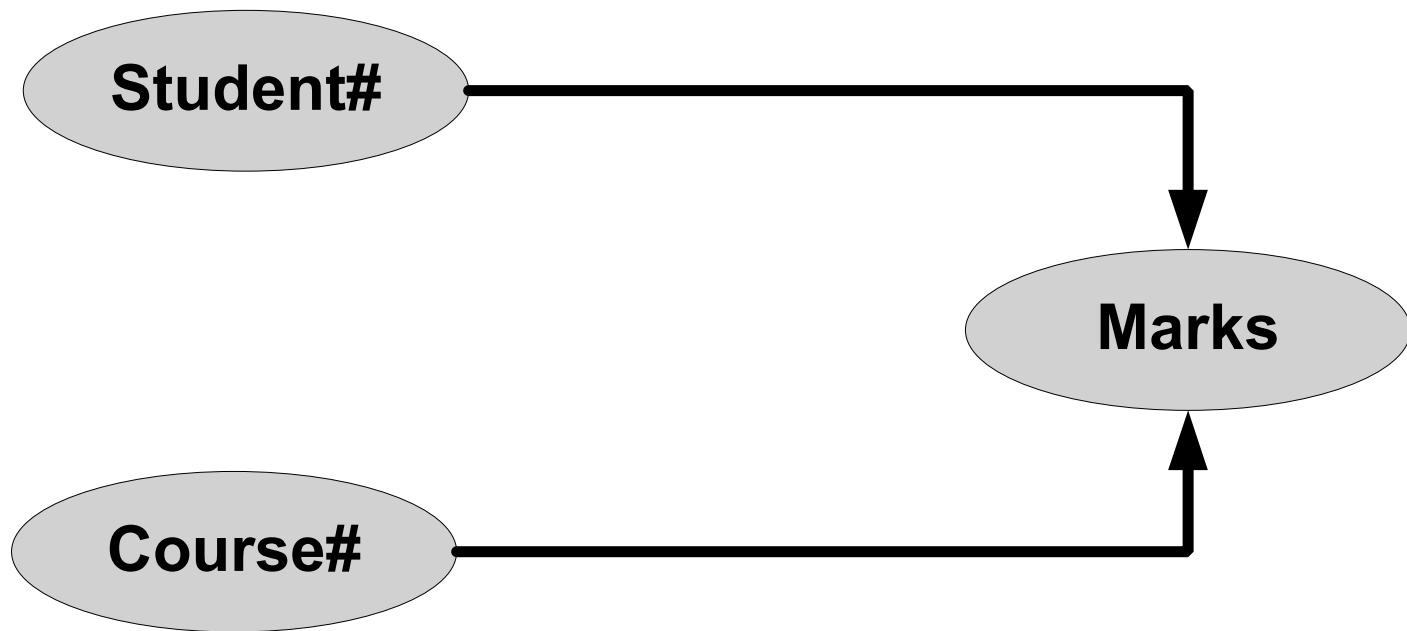
- Each course has only one lecturer and each lecturer has a room.
- Grade is determined from Marks.

Full dependencies

X and Y are attributes.

X Functionally determines Y

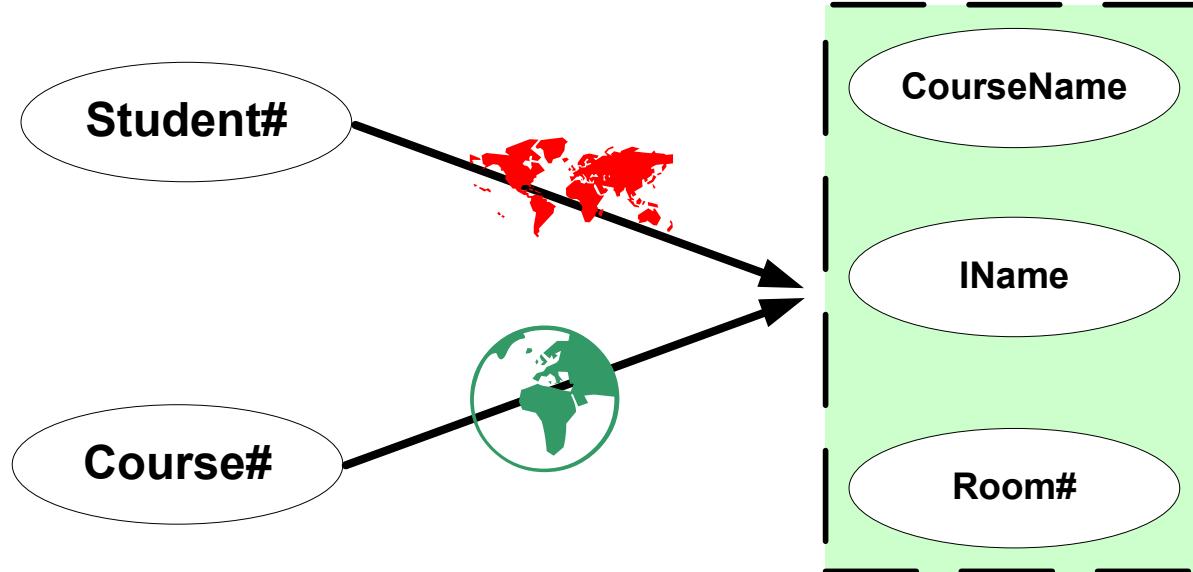
Note: Subset of X should not functionally determine Y



Partial dependencies

X and Y are attributes.

Attribute Y is partially dependent on the attribute X only if it is dependent on a sub-set of attribute X.



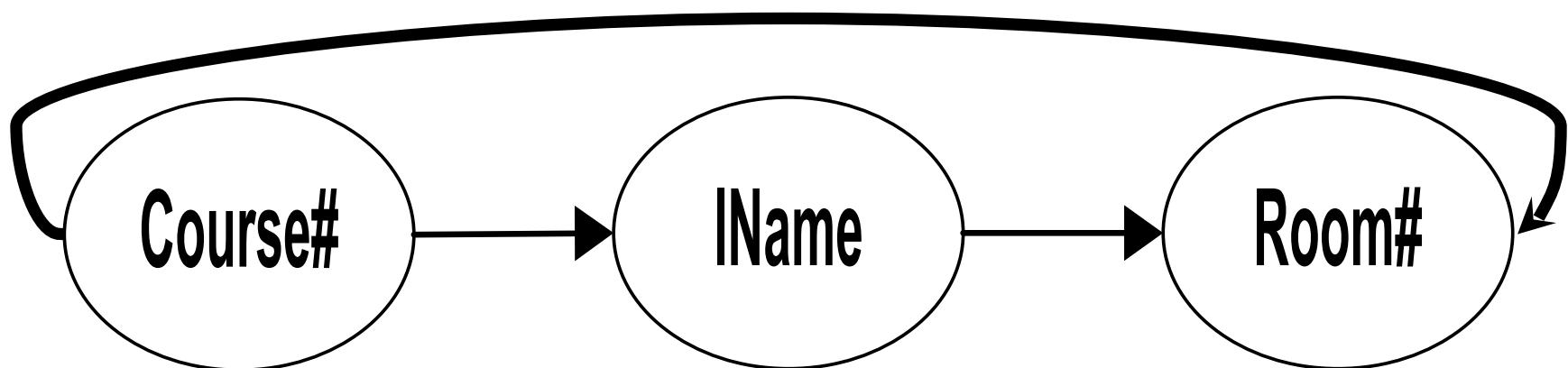
Transitive dependencies

X Y and Z are three attributes.

$X \rightarrow Y$

$Y \rightarrow Z$

$\Rightarrow X \rightarrow Z$



First normal form: 1NF

- **A relation schema is in 1NF :**
 - if and only if all the attributes of the relation R are atomic in nature.
 - **Atomic:** the smallest level to which data may be broken down and remain meaningful

Student_Course_Result Table

Student_Details			Course_Details				Results		
101	Davis	11/4/1986	M4	Applied Mathematics	Basic Mathematics	7	11/11/2004	82	A
102	Daniel	11/6/1987	M4	Applied Mathematics	Basic Mathematics	7	11/11/2004	62	C
101	Davis	11/4/1986	H6	American History		4	11/22/2004	79	B
103	Sandra	10/2/1988	C3	Bio Chemistry	Basic Chemistry	11	11/16/2004	65	B
104	Evelyn	2/22/1986	B3	Botany		8	11/26/2004	77	B
102	Daniel	11/6/1987	P3	Nuclear Physics	Basic Physics	13	11/12/2004	68	B
105	Susan	8/31/1985	P3	Nuclear Physics	Basic Physics	13	11/12/2004	89	A
103	Sandra	10/2/1988	B4	Zoology		5	11/27/2004	54	D
105	Susan	8/31/1985	H6	American History		4	11/22/2004	87	A
104	Evelyn	2/22/1986	M4	Applied Mathematics	Basic Mathematics	7	11/11/2004	65	B

Table in 1NF

Student_Course_Result Table

Student#	Student Name	Dateof Birth	Cour se #	CourseName	Pre Requisite	Dura tion InDa ys	DateOf Exam	Marks	Grade
101	Davis	04-Nov-1986	M4	Applied Mathematics	Basic Mathematics	7	11-Nov-2004	82	A
102	Daniel	06-Nov-1986	M4	Applied Mathematics	Basic Mathematics	7	11-Nov-2004	62	C
101	Davis	04-Nov-1986	H6	American History		4	22-Nov-2004	79	B
103	Sandra	02-Oct-1988	C3	Bio Chemistry	Basic Chemistry	11	16-Nov-2004	65	B
104	Evelyn	22-Feb-1986	B3	Botany		8	26-Nov-2004	77	B
102	Daniel	06-Nov-1986	P3	Nuclear Physics	Basic Physics	13	12-Nov-2004	68	B
105	Susan	31-Aug-1985	P3	Nuclear Physics	Basic Physics	13	12-Nov-2004	89	A
103	Sandra	02-Oct-1988	B4	Zoology		5	27-Nov-2004	54	D
105	Susan	31-Aug-1985	H6	American History		4	22-Nov-2004	87	A
104	Evelyn	22-Feb-1986	M4	Applied Mathematics	Basic Mathematics	7	11-Nov-2004	65	B

Second normal form: 2NF

- *A Relation is said to be in Second Normal Form if and only if :*
 - *It is in the First normal form, and*
 - *No partial dependency exists between non-key attributes and key attributes.*
- An attribute of a relation R that belongs to any key of R is said to be a prime attribute and that which doesn't is a **non-prime attribute**

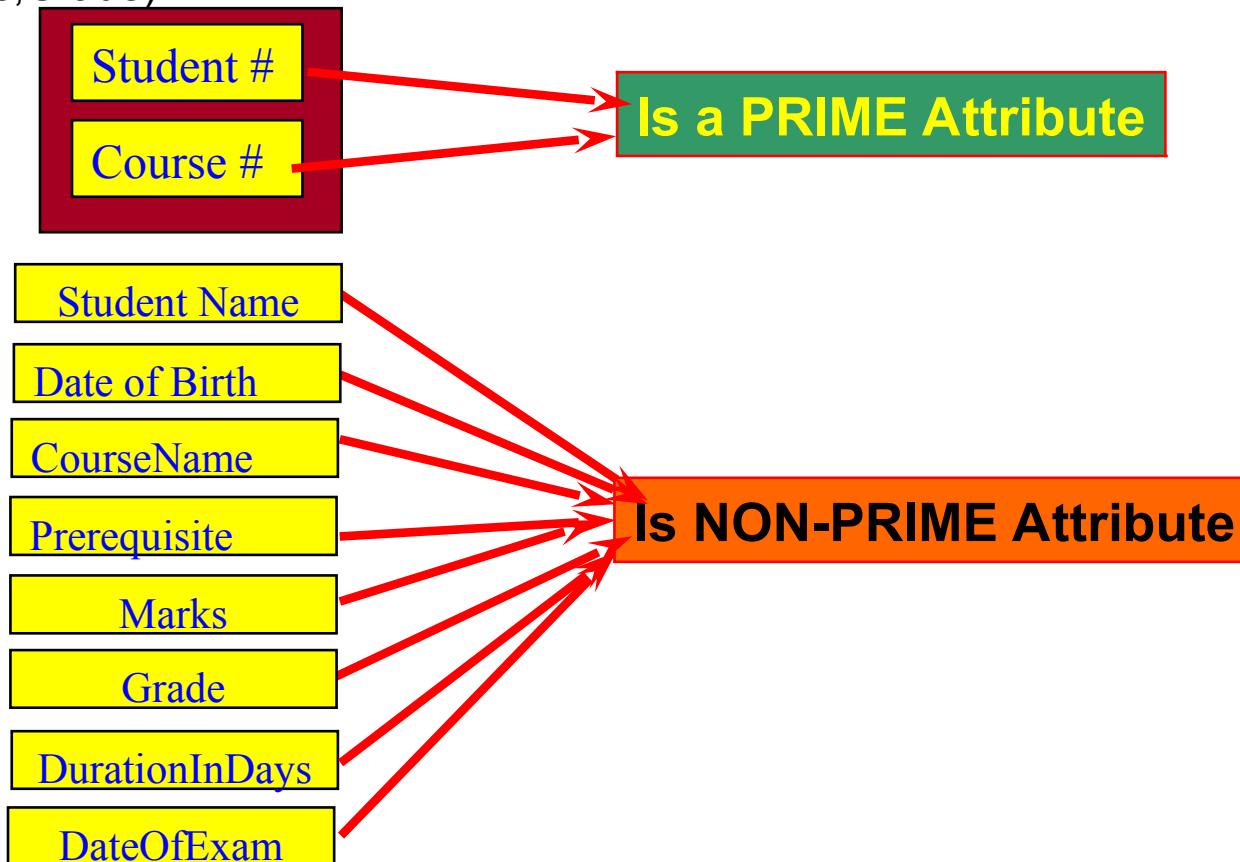
To make a table 2NF compliant, we have to remove all the partial dependencies

Note : - All partial dependencies are eliminated

Prime Vs Non-Prime Attributes

- An attribute of a relation R that belongs to any key of R is said to be a **prime attribute** and that which doesn't is a **non-prime attribute**

Report(S#,C#,StudentName,DateOfBirth,CourseName,PreRequisite,DurationInDays,Dat eOfExam,Marks,Grade)



Second Normal Form

- STUDENT# is key attribute for Student,
- COURSE# is key attribute for Course
- STUDENT# COURSE# together form the composite key attributes for Results relationship.
- Other attributes like StudentName (Student Name), DateofBirth, CourseName, PreRequisite, DurationInDays, DateofExam, Marks and Grade are non-key attributes.

To make this table 2NF compliant, we have to remove all the partial dependencies.

Student #, Course# -> Marks, Grade

Student# -> StudentName, DOB,

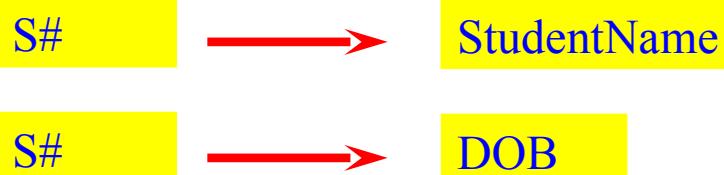
Course# -> CourseName, Prerequisite, DurationInDays

Course# -> Date of Exam

Second Normal Form



Fully Functionally
dependent on composite
Candidate key



Partial Dependency



Partial Dependency



Partial Dependency

Second Normal Form - Tables in 2 NF

STUDENT TABLE

Student#	StudentName	DateofBirth
101	Davis	04-Nov-1986
102	Daniel	06-Nov-1987
103	Sandra	02-Oct-1988
104	Evelyn	22-Feb-1986
105	Susan	31-Aug-1985
106	Mike	04-Feb-1987
107	Juliet	09-Nov-1986
108	Tom	07-Oct-1986
109	Catherine	06-Jun-1984

COURSE TABLE

	Course#	Course Name	Pre Requisite	Duration InDays
M1	Basic Mathematics			11
M4	Applied Mathematics	M1		7
H6	American History			4
C1	Basic Chemistry			5
C3	Bio Chemistry	C1		11
B3	Botany			8
P1	Basic Physics			8
P3	Nuclear Physics	P1		13
B4	Zoology			5

Second Normal form – Tables in 2 NF

Student#	Course#	Marks	Grade
101	M4	82	A
102	M4	62	C
101	H6	79	B
103	C3	65	B
104	B3	77	B
102	P3	68	B
105	P3	89	A
103	B4	54	D
105	H6	87	A
104	M4	65	B

Second Normal form – Tables in 2 NF

Exam_Date Table

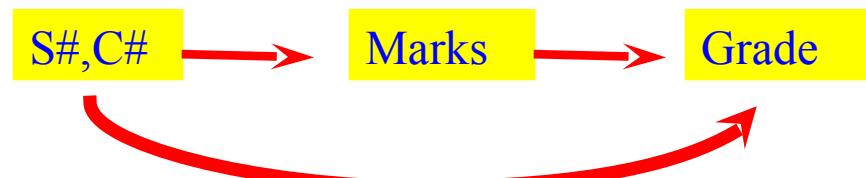
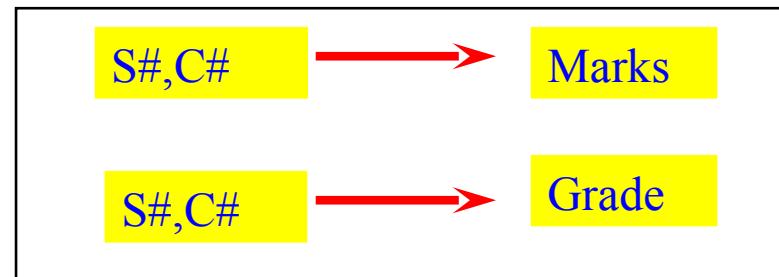
Course#	DateOfExam
M4	11-Nov-04
H6	22-Nov-04
C3	16-Nov-04
B3	26-Nov-04
P3	12-Nov-04
B4	27-Nov-04

Third normal form: 3 NF

A relation R is said to be in the Third Normal Form (3NF) if and only if

- It is in 2NF and
- No transitive dependency exists between non-key attributes and key attributes.

- STUDENT# and COURSE# are the key attributes.
- All other attributes, except grade are non-partially, non-transitively dependent on key attributes.
- Student#, Course# -> Marks
- Marks -> Grade



Note : - All transitive dependencies are eliminated

3NF Tables

Student#	Course#	Marks
101	M4	82
102	M4	62
101	H6	79
103	C3	65
104	B3	77
102	P3	68
105	P3	89
103	B4	54
105	H6	87
104	M4	65

Third Normal Form – Tables in 3 NF

MARKSGRADE TABLE

UpperBound	LowerBound	Grade
100	95	A+
94	85	A
84	70	B
69	65	B-
64	55	C
54	45	D
44	0	E

Boyce-Codd Normal form - BCNF

A relation is said to be in Boyce Codd Normal Form (BCNF)

- if and only if all the determinants are candidate keys.

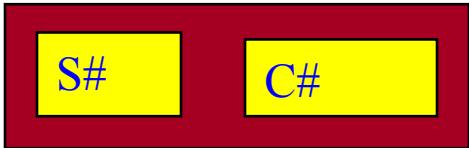
BCNF relation is a strong 3NF, but not every 3NF relation is BCNF.

Consider this Result Table

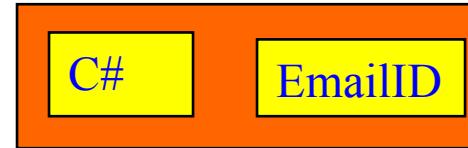
Student#	EmailID	Course#	Marks
101	<u>Davis@myuni.edu</u>	M4	82
102	<u>Daniel@myuni.edu</u>	M4	62
101	<u>Davis@myuni.edu</u>	H6	79
103	<u>Sandra@myuni.edu</u>	C3	65
104	<u>Evelyn@myuni.edu</u>	B3	77
102	<u>Daniel@myuni.edu</u>	P3	68
105	<u>Susan@myuni.edu</u>	P3	89
103	<u>Sandra@myuni.edu</u>	B4	54
105	<u>Susan@myuni.edu</u>	H6	87
104	<u>Evelyn@myuni.edu</u>	M4	65

BCNF

Candidate Keys for the relation are



and



Since **Course #** is overlapping, it is referred as Overlapping Candidate Key.



Valid Functional Dependencies are



STUDENT TABLE

Student#	EmailID
101	<u>Davis@myuni.edu</u>
102	<u>Daniel@myuni.edu</u>
103	<u>Sandra@myuni.edu</u>
104	<u>Evelyn@myuni.edu</u>
105	<u>Susan@myuni.edu</u>

BCNF Tables

Student#	Course#	Marks
101	M4	82
102	M4	62
101	H6	79
103	C3	65
104	B3	77
102	P3	68
105	P3	89
103	B4	54
105	H6	87
104	M4	65

Merits of Normalization

- Normalization is based on a mathematical foundation.
- Removes the redundancy to a greater extent. After 3NF, data redundancy is minimized to the extent of foreign keys.
- Removes the anomalies present in INSERTs, UPDATEs and DELETEs.

Demerits of Normalization

- Data retrieval or SELECT operation performance will be severely affected.
- Normalization might not always represent real world scenarios.

Summary of Normal Forms

Input	Operation	Output
Un-normalized Table	Create separate rows or columns for every combination of multivalued columns	Table in 1 NF
Table in 1 NF	Eliminate Partial dependencies	Tables in 2NF
Tables in 2 NF	Eliminate Transitive dependencies	Tables in 3 NF
Tables in 3 NF	Eliminate Overlapping candidate key columns	Tables in BCNF

Points to Remember:

Normal Form	Test	Remedy (Normalization)
1NF	Relation should have atomic attributes. The domain of an attribute must include only atomic (simple, indivisible) values.	Form new relations for each non-atomic attribute
2NF	For relations where primary key contains multiple attributes (composite primary key), non-key attribute should not be functionally dependent on a part of the primary key.	Decompose and form a new relation for each partial key with its dependent attribute(s). Retain the relation with the original primary key and any attributes that are fully functionally dependent on it.
3NF	Relation should not have a non-key attribute functionally determined by another non-key attribute (or by a set of non-key attributes). In other words there should be no transitive dependency of a non-key attribute on the primary key.	Decompose and form a relation that includes the non-key attribute(s) that functionally determine(s) other non-key attribute(s).

Summary

- Normalization is a refinement process. It helps in removing anomalies present in INSERTs/UPDATEs/DELETEs
- Normalization is also called “**Bottom-up approach**”, because this technique requires very minute details like every participating attribute and how it is dependant on the key attributes, is crucial. If you add new attributes after normalization, it may change the normal form itself.
- There are four normal forms that were defined being commonly used.
- 1NF makes sure that all the attributes are atomic in nature.
- 2NF removes the partial dependency.

Summary – contd.

- 3NF removes the transitive dependency.
- BCNF removes dependency among key attributes.
- Too much of normalization adversely affects SELECT or RETRIEVAL operations.
- It is always better to normalize to 3NF for INSERT, UPDATE and DELETE intensive (On-line transaction) systems.
- It is always better to restrict to 2NF for SELECT intensive (Reporting) systems.
- While normalizing, use common sense and don't use the normal forms as absolute measures.



Thank You!

SQL select statement

student Table

department Table

<u>d_id</u>	dep_name
10	CSE
20	ISE

usn	name	dep_num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS005	Avinash	20	90

Question

1. List out the USN's of the students who belong to CSE department ?

```
SQL> select usn from student,department where student.dep_num=department.d_id  
and department.dep_name='CSE';
```

USN

1BM14CS001
1BM14CS002
1BM14CS003
1BM14CS004

SQL select statement

student Table

department Table

<u>d_id</u>	dep_name
10	CSE
20	ISE

usn	name	dep_num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS005	Avinash	20	90

```
SQL> select usn from student,department where dep_num=d_id and dep_name='CSE' ;
```

USN

1BM14CS001
1BM14CS002
1BM14CS003
1BM14CS004

SQL select statement: Aliasing

student Table

department Table

<u>d_id</u>	dep_name
10	CSE
20	ISE



<u>usn</u>	name	dep_num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS005	Avinash	20	90

```
SQL> select usn from student s,department d where s.dep_num=d.d_id and d.dep_name='CSE';
```


USN

1BM14CS001
1BM14CS002
1BM14CS003
1BM14CS004

SQL select statement: Distinct

Student Table

<u>usn</u>	name	dep_nu m	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Avinash	20	90

```
SQL> select name from student;
```



Avinash
Balaji
Chandan
Dinesh
Avinash

SQL select statement: Distinct

Student Table

<u>usn</u>	name	dep_nu m	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Avinash	20	90

```
SQL> select distinct name from student;
```



Avinash
Balaji
Chandan
Dinesh

SQL select statement: Order By

Student Table

<u>usn</u>	name	dep_nu m	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Avinash	20	90

```
SQL> select usn,name,marks from student order by marks asc;
```



USN	NAME	MARKS
1BM14CS003	Chandan	45
1BM14CS004	Dinesh	60
1BM14CS002	Balaji	80
1BM14IS001	Avinash	90
1BM14CS001	Avinash	100

SQL select statement: Order By

Student Table

<u>usn</u>	name	dep_nu m	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Avinash	20	90

```
SQL> select usn,name,marks from student order by marks desc;
```



USN	NAME	MARKS
1BM14CS001	Avinash	100
1BM14IS001	Avinash	90
1BM14CS002	Balaji	80
1BM14CS004	Dinesh	60
1BM14CS003	Chandan	45

SQL select statement: Order By

Student Table

<u>usn</u>	name	dep_nu m	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Avinash	20	90

Question

List usn, name of the students who belong to department number 10 ordered by ascending order of their marks ?

SQL select statement: Order By

Student Table

<u>usn</u>	name	dep_nu m	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Avinash	20	90

Question

List usn, name of the students who belong to department number 10 ordered by ascending order of their marks ?

```
SQL> select usn,name,marks from student where dep_num=10 order by marks asc;
```

↓

USN	NAME	MARKS
1BM14CS003	Chandan	45
1BM14CS004	Dinesh	60
1BM14CS002	Balaji	80
1BM14CS001	Avinash	100

Activity: To do

Employee table

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-Dec-80	800		20
7499	ALLEN	SALESMAN	7698	20-Feb-81	1600	300	30
7521	WARD	SALESMAN	7698	22-Feb-81	1250	500	30
7566	JONES	MANAGER	7839	02-Apr-81	2975		20
7654	MARTIN	SALESMAN	7698	28-Sep-81	1250	1400	30
7698	BLAKE	MANAGER	7839	01-May-81	2850		30
7782	CLARK	MANAGER	7839	09-Jun-81	2450		10
7788	SCOTT	ANALYST	7566	09-Dec-82	3000		20
7839	KING	PRESIDENT		17-Nov-81	5000		10
7844	TURNER	SALESMAN	7698	08-Sep-81	1500	0	30
7876	ADAMS	CLERK	7788	12-Jan-83	1100		20
7900	JAMES	CLERK	7698	03-Dec-81	950		30
7902	FORD	ANALYST	7566	03-Dec-81	3000		20
7934	MILLER	CLERK	7782	23-Jan-82	1300		10

Write SQL queries for the following

1. Display all the information of the Employee table ?
2. Display unique Jobs from Employee table?
3. List names of the employees in the ascending order of their Salaries ?

Topics Covered in Today's class

Unit 1: Basic queries in SQL

w.r.t SELECT statement
SQL operators: LIKE, IN, BETWEEN

Structured Query Language (SQL)

SQL can be divided into two parts:

1. The Data Manipulation Language (DML) and
2. The Data Definition Language (DDL)

1. DDL statements in SQL are:

- CREATE DATABASE - creates a new database
- ALTER DATABASE - modifies a database
- CREATE TABLE - creates a new table
- ALTER TABLE - modifies a table
- DROP TABLE - deletes a table

2. DML part of SQL:

- SELECT - extracts data from a database
- UPDATE - updates data in a database
- DELETE - deletes data from a database
- INSERT INTO - inserts new data into a database

The SQL SELECT Statement

- The SELECT statement is used to select data from a database.
- The result is a table.

Student Table

<u>usn</u>	name	dep_num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Avinash	20	90

```
SQL> select * from student where name='Avinash';
```



USN	NAME	DEP_NUM	MARKS
1BM14CS001	Avinash	10	100
1BM14IS001	Avinash	20	90

- Note: **SQL is not case sensitive**. SELECT is the same as select.

w.r.t to SELECT SQL statement

1. SQL Alias: We can give a table or a column another name by using an alias.

- **SQL Alias Syntax for Tables**

```
SELECT column_name(s)  
FROM table_name AS alias_name;
```

- **SQL Alias Syntax for Columns**

```
SELECT column_name AS alias_name  
FROM table_name;
```

2. The **DISTINCT** keyword can be used to return only distinct (different) values.

- **SQL SELECT DISTINCT Syntax**

```
SELECT DISTINCT column_name(s)  
FROM table_name
```

w.r.t to SELECT sql statement

3. The **ORDER BY** keyword is used to sort the result-set by a specified column.

- **SQL ORDER BY Syntax**

```
SELECT column_name(s)
FROM table_name
ORDER BY column_name(s) ASC | DESC;
```

4. The **WHERE** clause is used to filter records. The WHERE clause is used to extract only those records that fulfill a specified criterion.

- **SQL WHERE Syntax**

```
SELECT column_name(s)
FROM table_name
WHERE column_name operator value
```

- With the WHERE clause

Operator	Description
=	Equal
!=	Not Equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
BETWEEN	Between an inclusive range
LIKE	Search for a pattern
IN	If you know the exact value you want to return for at least one of the columns

Substring Pattern Matching: SQL LIKE Operator

- The LIKE operator is used in a WHERE clause **to search for a specified pattern in a column.**
- SQL LIKE Syntax

```
SELECT column_name(s)  
      FROM table_name  
 WHERE column_name LIKE pattern
```

SQL LIKE Operator

- The **LIKE** operator is used in a WHERE clause **to search for a specified pattern** in a column.
- SQL LIKE Syntax

```
SELECT column_name(s)  
      FROM table_name  
 WHERE column_name LIKE pattern
```

Example: Write SQL statement to list the names starting with letter “A” from following student table.

Student Table

<u>usn</u>	name	dep_num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Avinash	20	90

SQL LIKE Operator

Write SQL statement to list the names starting with letter “A” from the fol

Student Table

<u>usn</u>	name	dep_num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Arvind	20	90

```
SQL> select name from student where name like 'A%' ;
```



NAME

Avinash
Arvind

SQL LIKE Operator

Write SQL statement to list the names starting with letter “A” from the fol

Student Table

<u>usn</u>	name	dep_num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Arvind	20	90

```
SQL> select name from student where name like 'A%';
```

NAME

Avinash
Arvind

Note:

Wildcard character **%**, A substitute for **zero or more characters**

SQL LIKE Operator

Student Table

<u>usn</u>	name	dep_num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Arvind	20	90

Question

Write SQL statement to list name of the students whose names **end** with letter '**h**'?

Note:

Wildcard character **%**, A substitute for **zero or more characters**

SQL LIKE Operator

Student Table

<u>usn</u>	name	dep_num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Arvind	20	90

Question

Write SQL statement to list name of the students whose names end with letter 'h'?

```
SQL> select name from student where name like '%h';
```

NAME

Avinash
Dinesh

SQL LIKE Operator

Student Table

<u>usn</u>	name	dep_num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Arvind	20	90

Question

Write SQL statement to list name of the students whose names are having the substring '**in**'?

Note:

Wildcard character **%**, A substitute for **zero or more characters**

SQL LIKE Operator

Student Table

<u>usn</u>	name	dep_num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Arvind	20	90

Question

Write SQL statement to list name of the students whose names are having the substring 'in'?

```
SQL> select name from student where name like '%in%';
```

NAME

Avinash
Dinesh
Arvind

SQL LIKE Operator

Student Table

<u>usn</u>	name	dep_num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Arvind	20	90

Question

Write SQL statement to list name of the students whose names start with letter 'A' or 'B'

SQL LIKE Operator

Student Table

<u>usn</u>	name	dep_num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Arvind	20	90

Question

Write SQL statement to list name of the students whose names start with letter 'A' or 'B'

```
SQL> select name from student where name like 'A%' or name like 'B%';
```

NAME

Avinash
Balaji
Arvind

SQL LIKE Operator

Student Table

<u>usn</u>	name	dep_num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Arvind	20	90

Question

Write SQL statement to list name of the students whose names start with letter 'A' or 'D' but end with letter 'h'

SQL LIKE Operator

Student Table

<u>usn</u>	name	dep_num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Arvind	20	90

Question

Write SQL statement to list name of the students whose names start with letter 'A' or 'D' but end with letter 'h'

```
SQL> select name from student where name like 'A%h' or name like 'D%h';
```

↓

NAME

Avinash
Dinesh

SQL LIKE Operator

Student Table

<u>usn</u>	name	dep_num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Arvind	20	90

Question

Write SQL statement to list name of the students whose third letter in the name is 'a' .

SQL LIKE Operator

Student Table

usn	name	dep_num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Arvind	20	90

Question

Write SQL statement to list name of the students whose third letter in the name is 'a' .

```
SQL> select name from student where name like '__a%';
```

NAME

Chandan

Note:

An **underscore** (_) in the pattern matches exactly **one character**

A **percent sign** (%) in the pattern can match **zero or more characters**

underscore (_) and **percent sign** (%) are referred as wildcard characters

SQL Wildcard Characters

- In SQL, wildcard characters are used with the SQL LIKE operator.
- SQL wildcards are used to search for data within a table.

With SQL, the wildcards are:

Wildcard	Description
%	A substitute for zero or more characters
-	A substitute for a single character
[charlist]	Sets and ranges of characters to match
[^charlist] or [!charlist]	Matches only a character NOT specified within the brackets

SQL IN Operator

- The IN operator allows you to specify multiple values in a WHERE clause.

SQL IN Syntax

```
SELECT column_name(s)
      FROM table_name
 WHERE column_name IN (value1,value2,...)
```

SQL IN Operator

- The IN operator allows you to specify multiple values in a WHERE clause.

SQL IN Syntax

```
SELECT column_name(s)
      FROM table_name
 WHERE column_name IN (value1,value2,...)
```

Student Table

<u>usn</u>	name	dep_num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Arvind	20	90

Example: List USN's of the students with name equal to "Avinash" or "Dinesh" from the table above.

SQL IN Operator

- The IN operator allows you to specify multiple values in a WHERE clause.

Student Table

<u>usn</u>	name	dep_num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Arvind	20	90

Example: List USN's of the students with name equal to "Avinash" or "Dinesh" from the table above.

```
SQL> select usn from student where name in ('Avinash', 'Dinesh');
```



```
USN
-----
1BM14CS001
1BM14CS004
```

SQL BETWEEN operator

- The BETWEEN operator selects a range of data between two values. The values can be numbers, text, or dates.

SQL BETWEEN Syntax

```
SELECT column_name(s)
```

```
FROM table_name
```

```
WHERE col1 <= value1 AND col1 >= value2
```

usn	name	dep_num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Arvind	20	90

Example: List USN's and Names of students whose marks is in between 40 and 80

SQL BETWEEN operator

- The BETWEEN operator selects a range of data between two values. The values can be numbers.

Student Table

<u>usn</u>	name	dep_num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Arvind	20	90

Example: List USN's and Names of students whose marks is in between 40 and 80

```
SQL> select usn,name from student where marks between 40 and 80;
```



USN	NAME
1BM14CS002	Balaji
1BM14CS003	Chandan
1BM14CS004	Dinesh

yr	subject	winner
1960	Chemistry	Willard F. Libby
1960	Literature	Saint-John Perse
1960	Medicine	Sir Frank Macfarlane Burnet
1960	Medicine	Peter Medawar
1960	Physics	Donald A. Glaser
1960	Peace	Albert Lutuli
...		

Activity To Do

Consider table of Nobel prize winners:

nobel(yr, subject, winner)

Answer the following

1. Pick the code which shows the name of winner's names beginning with C and ending in n

```
SELECT name FROM nobel
WHERE winner LIKE '%C%' AND winner LIKE '%n%'
```

```
SELECT name FROM nobel
WHERE winner LIKE '%C' AND winner LIKE 'n%'
```

```
SELECT name FROM nobel
WHERE winner LIKE 'C%' AND winner LIKE '%n'
```

```
SELECT winner FROM nobel
WHERE winner LIKE '%C' AND winner LIKE 'n%'
```

```
SELECT winner FROM nobel
WHERE winner LIKE 'C%' AND winner LIKE '%n'
```

yr	subject	winner
1960	Chemistry	Willard F. Libby
1960	Literature	Saint-John Perse
1960	Medicine	Sir Frank Macfarlane Burnet
1960	Medicine	Peter Medawar
1960	Physics	Donald A. Glaser
1960	Peace	Albert Lutuli
...		

Activity To Do

Consider table of Nobel prize winners:
nobel(yr, subject, winner)

Answer the following

1. Pick the code which shows the name of winner's names beginning with C and ending in n

```
SELECT name FROM nobel
WHERE winner LIKE '%C%' AND winner LIKE '%n%'
```

```
SELECT name FROM nobel
WHERE winner LIKE '%C' AND winner LIKE 'n%'
```

```
SELECT name FROM nobel
WHERE winner LIKE 'C%' AND winner LIKE '%n'
```

```
SELECT winner FROM nobel
WHERE winner LIKE '%C' AND winner LIKE 'n%'
```

```
SELECT winner FROM nobel
WHERE winner LIKE 'C%' AND winner LIKE '%n'
```

yr	subject	winner
1960	Chemistry	Willard F. Libby
1960	Literature	Saint-John Perse
1960	Medicine	Sir Frank Macfarlane Burnet
1960	Medicine	Peter Medawar
1960	Physics	Donald A. Glaser
1960	Peace	Albert Lutuli
...		

Activity To Do

Consider table of Nobel prize winners:

Answer the following

2. Select the code that shows how many Chemistry awards were given between 1950 and 1960

```
SELECT COUNT(subject) FROM nobel
WHERE subject = 'Chemistry'
AND BETWEEN 1950 AND 1960
```

```
SELECT COUNT(subject) FROM nobel
WHERE subject = 'Chemistry'
AND yr BETWEEN (1950, 1960)
```

```
SELECT COUNT(subject) FROM nobel
WHERE subject = 'Chemistry'
AND yr BETWEEN 1950 AND 1960
```

```
SELECT subject FROM nobel
WHERE subject = 'Chemistry'
AND yr BETWEEN 1950 AND 1960
```

```
SELECT subject FROM nobel
WHERE subject = 'Chemistry'
AND yr BETWEEN (1950, 1960)
```

Activity To Do

Consider table of Nobel prize winners:

nobel(yr, subject, winner)

Answer the following

2. Select the code that shows how many Chemistry awards were given between 1950 and 1960

```
SELECT COUNT(subject) FROM nobel  
WHERE subject = 'Chemistry'  
AND BETWEEN 1950 AND 1960
```

```
SELECT COUNT(subject) FROM nobel  
WHERE subject = 'Chemistry'  
AND yr BETWEEN (1950, 1960)
```

```
SELECT COUNT(subject) FROM nobel  
WHERE subject = 'Chemistry'  
AND yr BETWEEN 1950 AND 1960
```

```
SELECT subject FROM nobel  
WHERE subject = 'Chemistry'  
AND yr BETWEEN 1950 AND 1960
```

```
SELECT subject FROM nobel  
WHERE subject = 'Chemistry'  
AND yr BETWEEN (1950, 1960)
```

nobel		
yr	subject	winner
1960	Chemistry	Willard F. Libby
1960	Literature	Saint-John Perse
1960	Medicine	Sir Frank Macfarlane Burnet
1960	Medicine	Peter Medawar
1960	Physics	Donald A. Glaser
1960	Peace	Albert Lutuli
...		

Aggregate Functions in SQL

- Why we need aggregate functions ??
- Example say we want find maximum marks scored by the student

Student Table

<u>usn</u>	name	dep_num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Arvind	20	90

- We use aggregate function to group multiple rows together to form a single value output.

Topics Covered in Todays Class

Unit 1: Basic queries in SQL

Aggregate functions

Group BY Clause

Having Clause

Arithmetic operators in Queries

- Arithmetic operators for addition (+), subtraction (-), multiplication(*), and division (/) can be applied to numeric values or attributes with numeric domain.

Student Table

<u>usn</u>	name	dep_num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	arvind	20	90

- Example:

```
SQL> select usn, name, 10+marks as total from student;
```



USN	NAME	TOTAL
1BM14CS001	Avinash	110
1BM14CS002	Balaji	90
1BM14CS003	Chandan	55
1BM14CS004	Dinesh	70
1BM14IS001	arvind	100

Aggregate Functions in SQL

- SQL aggregate functions return a single value, calculated from values in a column.

Useful aggregate functions:

- AVG() - Returns the average value
- COUNT() - Returns the number of rows
- MAX() - Returns the largest value
- MIN() - Returns the smallest value
- SUM() - Returns the sum

Aggregate Functions in SQL

The AVG() Function

- The AVG() function returns the average value of a numeric column.
- **SQL AVG() Syntax**

```
SELECT AVG(column_name) FROM table_name  
Student Table
```

<u>usn</u>	name	dep_num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Arvind	20	90

Find average marks of all the students in the class.

```
SQL> select avg(marks) from student;
```

↓
AVG(MARKS)

75

Aggregate Functions in SQL

- **SQL COUNT() Function**

The COUNT() function returns the number of rows that matches a specified criteria.

- **SQL COUNT(column_name) Syntax**

The COUNT(column_name) function returns the number of values (NULL values will not be counted) of the specified column:

```
SELECT COUNT(column_name) FROM table_name
```

- **SQL COUNT(*) Syntax**

The COUNT(*) function returns the number of records in a table:

```
SELECT COUNT(*) FROM table_name
```

- **SQL COUNT(DISTINCT column_name) Syntax**

The COUNT(DISTINCT column_name) function returns the number of distinct values of the specified column:

```
SELECT COUNT(DISTINCT column_name) FROM table_name
```

Aggregate Functions in SQL

- **SQL COUNT(*) Syntax**

The COUNT(*) function returns the number of records in a table.

Student Table

<u>usn</u>	name	dep_num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Arvind	20	90

Find SQL> select count(*) from student;

ent table



COUNT(*)

5

Aggregate Functions in SQL

- **SQL COUNT(*) Syntax**

The COUNT(*) function returns the number of records in a table.

Student Table

<u>usn</u>	name	dep_num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Arvind	20	90

Find total number of students who belong to department number

10

```
SQL> select count(*) from student where dep_num=10;
```

↓
COUNT(*)

4

Aggregate Functions in SQL

- **SQL COUNT(column_name) Syntax**

The COUNT(column_name) function returns the number of values

(NULL

student1 Table				
<u>usn</u>	name	dep_num	marks	emailid
1BM14CS001	Avinash	10	100	avinash@gmail.com
1BM14CS002	Balaji	10	80	
1BM14CS003	Chandan	10	45	chandan@gmail.com
1BM14CS004	Dinesh	10	60	
1BM14IS001	Arvind	20	90	arvind@gmail.com

Find total

```
SQL> select count(emailid) from student1;
```



```
COUNT(EMAILID)
-----
3
```

Aggregate Functions in SQL

Student Table

usn	name	dep_num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Arvind	20	90

What will be the output of following SQL queries ?

```
SQL> select count(distinct dep_num) from student;
```

```
SQL> select count(dep_num) from student;
```

```
SQL> select min(*) from student;
```

```
SQL> select min(marks) from student;
```

Aggregate Functions in SQL

Student Table

<u>usn</u>	name	dep_num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Arvind	20	90

```
SQL> select count(distinct dep_num) from student;
COUNT(DISTINCTDEP_NUM)
-----
2

SQL> select count(dep_num) from student;
COUNT(DEP_NUM)
-----
5

SQL> select min(*) from student;
select min(*) from student
*
ERROR at line 1:
ORA-00936: missing expression

SQL> select min(marks) from student;
MIN(MARKS)
-----
45
```

Problem to solve

Consider three tables

SAILORS(Sal_ID, SalName, Rating, Age)

RESERVES(Sal_ID , Boat-ID, Rdate)

BOATS(Boat-ID, BoatName, Color)

Write Relational Algebra express for the following

- Find all the names of Sailors who have reserved boat with ID 2

```
SQL> select * from RESERVES;
```

SAL_ID	BOAT_ID	RDATE
101	1	12-1-2016
102	2	18-2-2016
103	2	25-2-2016

```
temp1 <-  $\sigma_{\text{Boat\_ID}=2}$ (RESERVES)
```

```
temp2 <-  $\pi_{\text{SAL\_ID}}(\text{temp1})$ 
```

temp2

SAL_ID
102
103

```
SELECT SAIL_ID  
FROM RESERVES  
WHERE Boat_ID = 2;
```

Problem to solve

i. Find all the names of Sailors who have reserved boat with **ID 2**

```
SQL> select * from RESERVES;
```

SAL_ID	BOAT_ID	RDATE
101	1	12-1-2016
102	2	18-2-2016
103	2	25-2-2016

```
SQL> select * from SAILORS;
```

SAL_ID	SALNAME	RATING	AGE
101	Avinash	200	19
102	Balaji	150	18
103	Dinesh	150	18

```
temp1 <-  $\sigma_{\text{Boat\_ID}=2}$  (RESERVES)
```

```
temp2 <-  $\pi_{\text{SAL\_ID}}$ (temp1)
```

```
temp3 <- SAILORS  $\bowtie$  SAILORS.Sal_ID=temp2.Sal_ID (temp2)
```

temp3

SAL_ID	SALNAME	RATING	AGE
102	Balaji	150	18
103	Dinesh	150	18

```
SELECT *
FROM SAILORS
NATURAL JOIN
(SELECT SAL_ID
FROM RESERVES
WHERE Boat_ID = 2);
```

Problem to solve

i. Find all the names of Sailors who have reserved boat with **ID 2**

```
SQL> select * from RESERVES;
```

SAL_ID	BOAT_ID	RDATE
101	1	12-1-2016
102	2	18-2-2016
103	2	25-2-2016

```
SQL> select * from SAILORS;
```

SAL_ID	SALNAME	RATING	AGE
101	Avinash	200	19
102	Balaji	150	18
103	Dinesh	150	18

```
temp1 <-  $\sigma_{\text{Boat\_ID}=2}$  (RESERVES)
```

```
temp2 <-  $\pi_{\text{SAL\_ID}}$ (temp1)
```

```
temp3 <- SAILORS  $\bowtie$  SAILORS.Sal_ID=temp2.Sal_ID (temp2)
```

```
result <-  $\pi_{\text{SAL\_Name}}$ (temp3)
```

result

SALNAME

Balaji

Dinesh

SELECT SalName
FROM SAILORS
NATURAL JOIN
(SELECT SAL_ID
FROM RESERVES
WHERE Boat_ID = 2);

Problem to solve

i. Find all the names of Sailors who have reserved boat with **ID 2**

```
SQL> select * from RESERVES;
```

SAL_ID	BOAT_ID	RDATE
101	1	12-1-2016
102	2	18-2-2016
103	2	25-2-2016

```
SQL> select * from SAILORS;
```

SAL_ID	SALNAME	RATING	AGE
101	Avinash	200	19
102	Balaji	150	18
103	Dinesh	150	18

```
temp1 <-  $\sigma_{\text{Boat\_ID}=2}$  (RESERVES)
```

```
temp2 <-  $\pi_{\text{SAL\_ID}}$ (temp1)
```

```
temp3 <- SAILORS  $\bowtie$  SAILORS.Sal_ID=temp2.Sal_ID (temp2)
```

```
result <-  $\pi_{\text{SAL\_Name}}$ (temp3)
```

```
SQL> select SalName from SAILORS, RESERVES where BOAT_ID=2 and SAILORS.Sal_ID=RESERVES.Sal_ID;
```

SALNAME

Balaji
Dinesh

SAILORS(Sal_ID, SalName, Rating, Age), RESERVES(Sal_ID , Boat-ID, Rdate),
BOATS(Boat-ID, BoatName, Color)

Write Relational Algebra express for the following

ii. Find names of sailors who have reserved RED boat

SQL> select * from BOATS;

BOAT_ID	BOATNAME	COLOR
1	Kaveri	Blue
2	Ganga	Red

SQL> select * from RESERVES;

SAL_ID	BOAT_ID	RDATE
101	1	12-1-2016
102	2	18-2-2016
103	2	25-2-2016

SQL> select * from SAILORS;

SAL_ID	SALNAME	RATING	AGE
101	Avinash	200	19
102	Balaji	150	18
103	Dinesh	150	18

OUTPUT

Balaji
Dinesh

SAILORS(Sal_ID, SalName, Rating, Age), RESERVES(Sal_ID , Boat-ID, Rdate),
BOATS(Boat-ID, BoatName, Color)

Write Relational Algebra express for the following

ii. Find the colors of the boat reserved by **Avinash**

SQL> select * from SAILORS;

SAL_ID	SALNAME	RATING	AGE
101	Avinash	200	19
102	Balaji	150	18
103	Dinesh	150	18

SQL> select * from RESERVES;

SAL_ID	BOAT_ID	RDATE
101	1	12-1-2016
102	2	18-2-2016
103	2	25-2-2016

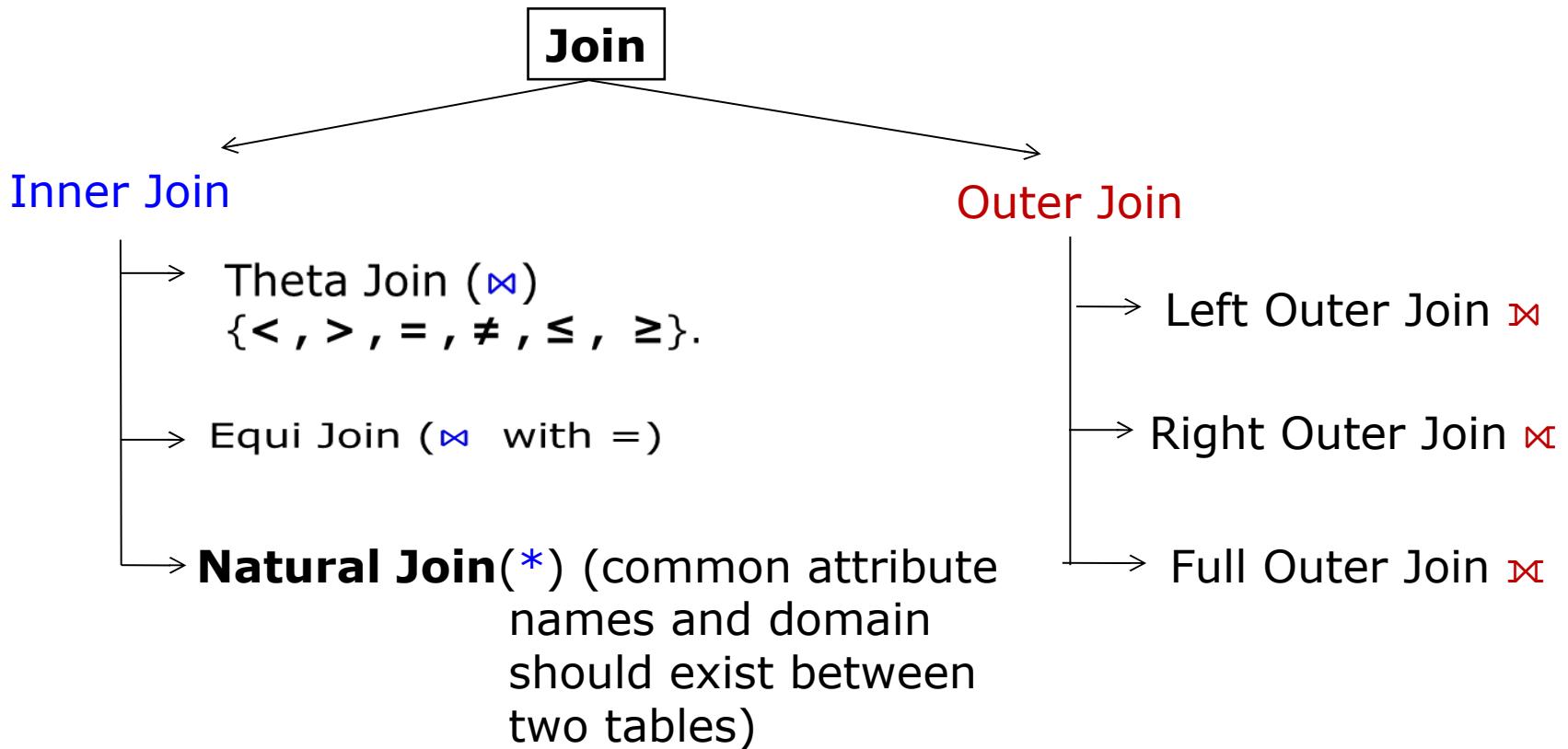
SQL> select * from BOATS;

BOAT_ID	BOATNAME	COLOR
1	Kaveri	Blue
2	Ganga	Red

OUTPUT

Red

Join Operations



Left Outer Join ✎

result <- table1 ✎ table2

table1

ID	M
1	a
2	b
4	c

table2

ID	N
2	p
3	q
5	r

result

ID	M	ID	N
2	b	2	P
1	a		
4	c		

```
SQL> select *
  from table1 left outer join table2
  on table1.id=table2.id;
```

Right Outer Join ✖

result <- table1 ✖ table2

table1

ID	M
1	a
2	b
4	c

table2

ID	N
2	p
3	q
5	r

result

ID	M	ID	N
2	b	2	P
		3	q
		5	r

```
SQL> select *
  from table1 right outer join table2
  on table1.id=table2.id;
```

Full Outer Join ✖

result <- **table1** ✖ **table2**

table1

ID	M
1	a
2	b
4	c

table2

ID	N
2	p
3	q
5	r

result

ID	M	ID	N
2	b	2	p
		3	q
		5	r
1	a		
4	c		

```
SQL> select *
      from table1 full join table2
      on table1.id = table2.id;
```

Relational Algebra Set Operations - semantics

Consider two relations R and S.

- **UNION** of R and S
the union of two relations is a relation that includes all the tuples that are either in R or in S or in both R and S. Duplicate tuples are eliminated.
- **INTERSECTION** of R and S
the intersection of R and S is a relation that includes all tuples that are both in R and S.
- **DIFFERENCE** of R and S
the difference of R and S is the relation that contains all the tuples that are in R but that are not in S.

For set operations to function correctly the relations R and S must be union compatible. Two relations are union compatible if

- they have the same number of attributes
- the domain of each attribute in column order is the same in both R and S.

Set Operation – Union \cup

R	
A	1
B	2
D	3
F	4
E	5

S	
A	1
C	2
D	3
E	4

result

result $\leftarrow R \cup S$

result	
A	1
B	2
C	2
D	3
E	5
F	4
E	4

Set Operation - Intersection \cap

R	
A	1
B	2
D	3
F	4
E	5

S	
A	1
C	2
D	3
E	4

result

result <- R \cap S

A	1
D	3

Set Operation – Difference –

R	
A	1
B	2
D	3
F	4
E	5

S	
A	1
C	2
D	3
E	4

result <- R - S

B	2
F	4
E	5

result <- S - R

C	2
E	4

Problem to Solve: Writing Relational Algebra Expression

Consider the following three tables

- STUDENT(StudNum, StudName)
- PROJECT(ProjNum, ProjArea)
- ASSIGED_TO(StudNum,ProjNum)

- i. Obtain student number and student name of all students who are working on both the projects having project number 75 and 81
- ii. Obtain student number and student name of all those students who do not work on project number 68
- iii. Obtain the student number and student name of all those students who are working on project with name “Database”
- iv. Obtain student number and student name of all students other than the student with number 554 who works on atleast one project.

Problem to Solve: Writing Relational Algebra Expression

Consider the following three tables

- STUDENT(StudNum, StudName), PROJECT(ProjNum, ProjArea),
ASSIGNED_TO(StudNum,ProjNum)

i. Obtain student number and student name of all students who are working on both the projects having project number 75 and 81

STUDENT

StudNum	StudName
554	Avinash
555	Balaji
556	Chandan
557	Dinesh
558	Harish

PROJECT

ProjNum	ProjArea
56	Java
68	Database
75	Database
81	Database

ASSIGNED_TO

StudNum	Proj Num
554	56
555	68
556	75
556	81
557	75

Problem to Solve: Writing Relational Algebra Expression

- i. Obtain student number and student name of all students who are working on **both** the projects having project number 68 and 75.

STUDENT

StudNum	StudName
554	Avinash
555	Balaji
556	Chandan
557	Dinesh
558	Harish

ASSIGNED_TO

Stud Num	Proj Num
554	56
555	68
556	75
556	81
557	75

result

556 Chandan

Problem to Solve: Writing Relational Algebra Expression

- i. Obtain student number and student name of all students who are working on **both** the projects having

STUDENT	
StudNum	StudName
554	Avinash
555	Balaji
556	Chandan
557	Dinesh
558	Harish

ASSIGNED_TO	
Stud Num	Proj Num
554	56
555	68
556	75
556	81
557	75

result	

556	Chandan

```
temp1 <-  $\sigma_{ProjNum=75}(ASSIGNED\_TO)$ 
temp2 <-  $\pi_{StudNum}(temp1)$ 
temp3 <-  $\sigma_{ProjNum=81}(ASSIGNED\_TO)$ 
temp4 <-  $\pi_{StudNum}(temp3)$ 

temp5 <- temp2  $\bowtie$  temp2
result <- STUDENT  $\bowtie$  STUDENT.StudNum=temp5.StudNum (temp5)
```

Problem to Solve: Writing Relational Algebra Expression

- ii. Obtain student number and student name of all those students who do not work on project number 68

STUDENT

StudNum	StudName
554	Avinash
555	Balaji
556	Chandan
557	Dinesh
558	Harish

ASSIGNED_TO

StudNu m	Proj Num
554	56
555	68
556	75
556	81
557	75

result

554 Avinash
556 Chandan
557 Dinesh
558 Harish

Problem to Solve: Writing Relational Algebra Expression

i. Obtain student number and student name of all those students who do not work on project

num

StudNum	StudName
554	Avinash
555	Balaji
556	Chandan
557	Dinesh
558	Harish

ASSIGNED_TO

Stud Num	Proj Num
554	56
555	68
556	75
556	81
557	75

result	
<hr/>	
554	Avinash
556	Chandan
557	Dinesh
558	Harish

```
temp1 <-  $\sigma_{ProjNum=68}(\text{ASSIGNED\_TO})$ 
```

```
temp2 <-  $\pi_{\text{StudNum}}(\text{temp1})$ 
```

```
temp3 <-  $\pi_{\text{StudNum}}(\text{STUDENT})$ 
```

```
temp4 <- temp3 - temp2
```

```
result <- STUDENT  $\bowtie$  STUDENT.StudNum=temp4.StudNum (temp4)
```

Homework Problem : Writing Relational Algebra Expression

- iii Obtain the student number and student name of all those students who are working on
pr

STUDENT	
StudNum	StudName
554	Avinash
555	Balaji
556	Chandan
557	Dinesh
558	Harish

PROJECT	
ProjNum	ProjArea
56	Java
68	Database
75	Database
81	Database

ASSIGNED_TO	
Stud Num	Proj Num
554	56
555	68
556	75
556	81
557	75

result

555 Balaji
556 Chandan
557 Dinesh

Homework Problem: Writing Relational Algebra Expression

- iv. Obtain student number and student name of all students other than the student with number 554 who works on atleast one project

STUDENT	
StudNum	StudName
554	Avinash
555	Balaji
556	Chandan
557	Dinesh
558	Harish

PROJECT	
ProjNum	ProjArea
56	Java
68	Database
75	Database
81	Database

ASSIGNED_TO	
Stud Num	Proj Num
554	56
555	68
556	75
556	81
557	75

result

555	Balaji
556	Chandan
557	Dinesh

Relational Algebra : Division Operation \div

The division operator is used for queries which involve the ‘all’ qualifier such as

- “Which persons have a bank account at ALL the banks in the country?”
- “Which students are registered on ALL the courses given by Sonthos?”
- “Which students are registered on ALL the courses that are taught in period 1?”
- Find sailors who have reserved ALL boats

$R \div S$ is used when we wish to express queries with “ALL”

Relational Algebra : Division Operation \div

The division operator takes as input two relations, called the dividend relation (a on scheme A) and the divisor relation (b on scheme B) such that all the attributes in B also appear in A and B is not empty. The output of the division operation is a relation on scheme A with all the attributes common with B .

A		B	A \div B
sno	pno	pno	sno
s1	p1	p1	s1
s1	p2	p2	
s1	p3	p3	
s1	p4	p4	
s2	p1		
s2	p2		
s3	p2		
s4	p2		
s4	p4		

Relational Algebra : Division Operation \div

The division operator takes as input two relations, called the dividend relation (a on scheme A) and the divisor relation (b on scheme B) such that all the attributes in B also appear in A and B is not empty. The output of the division operation is a relation on scheme A with all the attributes common with B .

A		B		A \div B	
sno	pno	pno		sno	
s1	p1				
s1	p2	p2			
s1	p3				
s1	p4	p4			
s2	p1				
s2	p2				
s3	p2				
s4	p2				
s4	p4				

Relational Algebra : Division Operation \div

A

sno	pno
s1	p1
s1	p2
s1	p3
s1	p4
s2	p1
s2	p2
s3	p2
s4	p2
s4	p4

B

pno
p1
p2
p3
p4

$A \div B$

sno
s1

Relational Algebra : Division Operation \div

Completed

Student	Task
Feroz	Database1
Feroz	Database2
Feroz	Compiler1
Eshwar	Database1
Sara	Database1
Sara	Database2
Eshwar	Compiler1

DBProject

Task
Database1
Database2

What is the Output of the following relational Algebra Expression

Completed \div DBProject

Relational Algebra : Division Operation \div

Completed

Student	Task
Feroz	Database1
Feroz	Database2
Feroz	Compiler1
Eshwar	Database1
Sara	Database1
Sara	Database2
Eshwar	Compiler1

DBProject

Task
Database1
Database2

Completed \div DBProject

Student
Feroz
Sara

Relational Algebra : Division Operation \div

Completed

Student	Task
F	D1
F	D2
F	C1
E	D1
E	C1

DBProject

Task
D1
D2

Completed \div DBProject

Student
F

Is the following two relational algebra expressions logically equivalent ?

$T \leftarrow \text{Completed} \div \text{DBProject}$

$T_1 \leftarrow \pi_{\text{Student}}(\text{Completed})$
 $T_2 \leftarrow T_1 \times \text{DBProject}$
 $T_3 \leftarrow T_2 - \text{Completed}$
 $T_4 \leftarrow \pi_{\text{Student}}(T_3)$
 $T \leftarrow T_1 - T_4$

Write relational Algebra Expression

Find all bank customers who have account in all Branches of Bommasandra

Account

cid	branch_id	acct_no	balance
1	52	8103	43101.45
3	53	4826	752.80
1	53	7898	48206.10
2	59	2135	468923.06
1	59	1290	456.50
2	54	0073	1006.28

Branch

branch_id	branch_city
51	Belgaum
52	Bijapur
53	Bommasandra
54	Hubli
55	Bijapur
59	Bommasandra

Customer

cid	c_name
1	Harish
2	Triveni
3	Eshwar

RESULT

c_name
Harish

Find all bank customers who have account in all Branches in Bommasandra

Account			
cid	branch_id	acct_no	balance
1	52	8103	43101.45
3	53	4826	752.80
1	53	7898	48206.10
2	59	2135	468923.06
1	59	1290	456.50
2	54	0073	1006.28

Branch	
branch_id	branch_city
51	Belgaum
52	Bijapur
53	Bommasandra
54	Hubli
55	Bijapur
59	Bommasandra

Customer	
cid	c_name
1	Harish
2	Triveni
3	Eshwar

BommB $\leftarrow \pi_{\text{branch_id}}(\sigma_{\text{branch_city}=\text{'Bommasandra'}}(\text{Branch}))$ --find all branches located in Bommasandra

BommB

branch_id
53
59

Find all bank customers who have account in all Branches in Bommasandra

Account			
cid	branch_id	acct_no	balance
1	52	8103	43101.45
3	53	4826	752.80
1	53	7898	48206.10
2	59	2135	468923.06
1	59	1290	456.50
2	54	0073	1006.28

Branch	
branch_id	branch_city
51	Belgaum
52	Bijapur
53	Bommasandra
54	Hubli
55	Bijapur
59	Bommasandra

Customer	
cid	c_name
1	Harish
2	Triveni
3	Eshwar

$\text{BommB} \leftarrow \pi_{\text{branch_id}}(\sigma_{\text{branch_city}='\text{Bommasandra}'}(\text{Branch}))$ --find all branches located
 in Bommasandra
 $\text{CB} \leftarrow \pi_{\text{c_name}, \text{branch_id}}(\text{Customer} * \text{Account})$ -- find all customers' branches

BommB	
branch_id	
53	
59	

CB	
c_name	branch_id
Harish	52
Eshwar	53
Harish	53
Triveni	59
Harish	59
Triveni	54

Find all bank customers who have account in all Branches in Bommasandra

Account			
cid	branch_id	acct_no	balance
1	52	8103	43101.45
3	53	4826	752.80
1	53	7898	48206.10
2	59	2135	468923.06
1	59	1290	456.50
2	54	0073	1006.28

Branch	
branch_id	branch_city
51	Belgaum
52	Bijapur
53	Bommasandra
54	Hubli
55	Bijapur
59	Bommasandra

Customer	
cid	c_name
1	Harish
2	Triveni
3	Eshwar

BommB $\leftarrow \pi_{\text{branch_id}}(\sigma_{\text{branch_city}=\text{'Bommasandra'}}(\text{Branch}))$ --find all branches located in Bommasandra

CB $\leftarrow \pi_{\text{c_name}, \text{branch_id}}(\text{Customer} * \text{Account})$ -- find all customers' branches

RESULT $\leftarrow \text{CB} \div \text{BinB}$ -- divide to get those customers with an account in every Bommasandra branch

BommB	
branch_id	
53	
59	

CB	
c_name	branch_id
Harish	52
Eshwar	53
Harish	53
Triveni	59
Harish	59
Triveni	54

RESULT	
c_name	
Harish	

Table 6.1 Operations of Relational Algebra

OPERATION	PURPOSE	NOTATION
SELECT	Selects all tuples that satisfy the selection condition from a relation R .	$\sigma_{\text{selection condition}}(R)$
PROJECT	Produces a new relation with only some of the attributes of R , and removes duplicate tuples.	$\pi_{\text{attribute list}}(R)$
THETA JOIN	Produces all combinations of tuples from R_1 and R_2 that satisfy the join condition.	$R_1 \bowtie_{\text{join condition}} R_2$
EQUIJOIN	Produces all the combinations of tuples from R_1 and R_2 that satisfy a join condition with only equality comparisons.	$R_1 \bowtie_{\text{join condition}} R_2$, OR $R_1 \bowtie_{(<\text{join attributes 1}>), (<\text{join attributes 2}>)} R_2$
NATURAL JOIN	Same as EQUIJOIN except that the join attributes of R_2 are not included in the resulting relation; if the join attributes have the same names, they do not have to be specified at all.	$R_1 *_{\text{join condition}} R_2$, OR $R_1 *_{(<\text{join attributes 1}>), (<\text{join attributes 2}>)} R_2$ OR $R_1 * R_2$
UNION	Produces a relation that includes all the tuples in R_1 or R_2 or both R_1 and R_2 ; R_1 and R_2 must be union compatible.	$R_1 \cup R_2$
INTERSECTION	Produces a relation that includes all the tuples in both R_1 and R_2 ; R_1 and R_2 must be union compatible.	$R_1 \cap R_2$
DIFFERENCE	Produces a relation that includes all the tuples in R_1 that are not in R_2 ; R_1 and R_2 must be union compatible.	$R_1 - R_2$
CARTESIAN PRODUCT	Produces a relation that has the attributes of R_1 and R_2 and includes as tuples all possible combinations of tuples from R_1 and R_2 .	$R_1 \times R_2$
DIVISION	Produces a relation $R(X)$ that includes all tuples $t[X]$ in $R_1(Z)$ that appear in R_1 in combination with every tuple from $R_2(Y)$, where $Z = X \cup Y$.	$R_1(Z) \div R_2(Y)$

Query Tree Notation

- Query Tree
 - An internal data structure to represent a query
 - Standard technique for estimating the work involved in executing the query, the generation of intermediate results, and the optimization of execution
 - Nodes stand for operations like selection, projection, join, renaming, division,
 - Leaf nodes represent base relations
 - A **tree** gives a **good visual feel of the complexity of the query** and the operations involved
 - Algebraic Query Optimization consists of rewriting the query or modifying the query tree into an equivalent tree.

Query Tree Notation: Example

- Write query: For every project located in 'Surat', list the project number, the controlling department number, and the department manager's last name, address, and birth date.

PROJECT

Pname	Pnumber	Plocation	Dnum

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Salary	Super_ssn	Dno

Query Tree Notation: Example

- Example: For every project located in 'Surat', list the project number, the controlling department number, and the department manager's last name, address, and birth date.

PROJECT

Pname	<u>Pnumber</u>	Plocation	Dnum

DEPARTMENT

Dname	<u>Dnumber</u>	Mgr_ssn	Mgr_start_date

EMPLOYEE

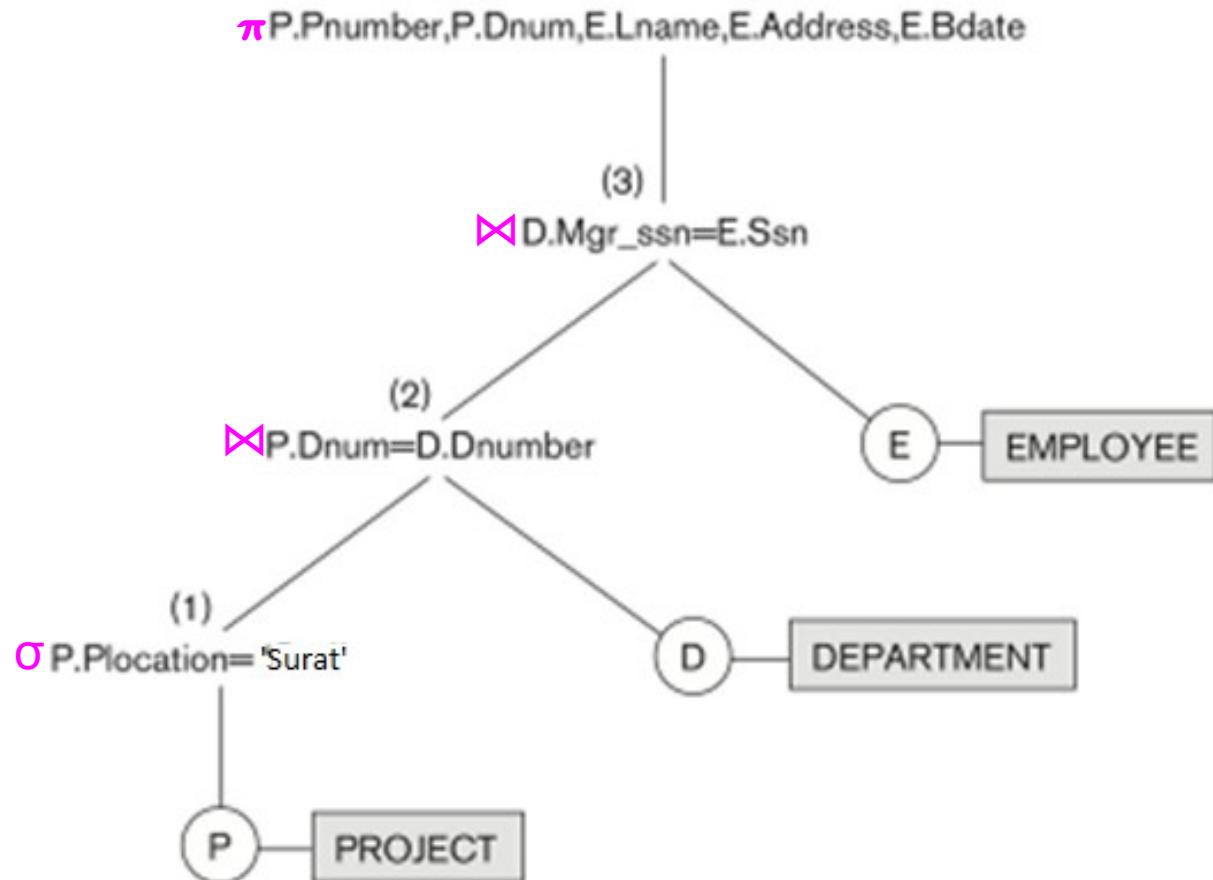
Fname	Minit	Lname	<u>Ssn</u>	Bdate	Address	Salary	Super_ssn	Dno

$$\pi_{Pnumber, Dnum, Lname, Address, Bdate} ((\sigma_{Plocation='Surat'} (Project) \bowtie_{Dnum=Dnumber} Department) \bowtie_{Mgr_ssn=Ssn} Employee)$$

Query Tree Notation: Example

$\pi_{Pnumber, Dnum, Lname, Address, Bdate}$

$(\sigma_{Plocation='Surat'}(Project) \bowtie_{Dnum=Dnumber} Department) \bowtie_{Mgr_ssn=Ssn} Employee$



Generalized Projection π

- Extends the projection operation by allowing arithmetic functions to be used in the projection list.

$$\pi_{F_1, F_2, \dots, F_n}(E)$$

- E is any relational-algebra expression
- Each of F_1, F_2, \dots, F_n are arithmetic expressions involving constants and attributes in the schema of E .

Generalized Projection π

- Given relation

credit-info(CustomerName, Limit, CreditBalance)

Customer-name	Limit	CreditBalance
Avinash	2000	500
Balaji	700	100
Chandan	1500	1000

Find how much money each person can spend:

Generalized Projection π

- Given relation

credit-info(CustomerName, Limit, CreditBalance)

Customer-name	Limit	CreditBalance
Avinash	2000	500
Balaji	700	100
Chandan	1500	1000

Find how much money each person can spend:

Customer-name	Limit - CreditBalance
Avinash	1500
Balaji	600
Chandan	500

Generalized Projection π

- Another Example

Consider a relation

EMPLOYEE(EMP-ID, Salary, Deduction, Years-of-Service)

A **report** may be required to show:

- Net_salary = Salary - Deduction
- Bonus = 2000 * Years-of-Service
- Tax = Salary * 25%

Then a generalized projection combined with renaming may be:

report $\rho_{(Net_salary, Bonus, Tax)}$

$(\pi_{EMP-ID, (Salary - Deduction), (2000 * Years-of-Service), (Salary * 0.25)} (EMPLOYEE))$

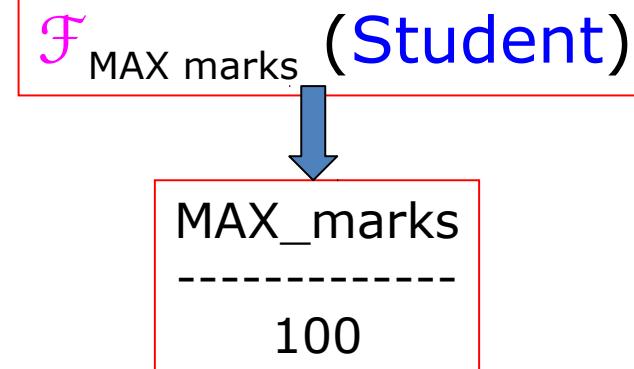
Aggregate Function \mathcal{F} (script F)

- Aggregate functions return a single value, calculated from values in a column.

Useful aggregate functions:

- AVG() - Returns the average value
- COUNT() - Returns the number of rows
- MAX() - Returns the largest value
- MIN() - Returns the smallest value
- SUM() - Returns the sum

Student Table			
<u>usn</u>	name	dep_num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	arvind	20	90



Aggregate Function \mathcal{F} (script F)

Student Table

<u>usn</u>	<u>name</u>	<u>dep_num</u>	<u>marks</u>
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	arvind	20	90

\mathcal{F} COUNT usn, AVERAGE marks (**Student**)



COUNT_usn AVERAGE_marks

5

75

Using Grouping with Aggregation

<grouping attribute> \exists <aggregate function list>

- <grouping attribute> is a list of attributes of the relation specified in R and <aggregate function list> is a list of (<function> <attribute>)

Student Table			
<u>usn</u>	name	dep_num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	arvind	20	90

dep_num	COUNT_usn	AVERAGE_marks	(Student)
10	4	71.25	
20	1	90	

Problem to Solve: Writing Relational Algebra Expression

Consider the following three tables

- **SALESPERSON**(SalesPersonID,Name)
 - **TRIP**(SalesPersonID, From, To, Trip-ID)
 - **EXPENSE**(TripID, Amount)
- i. Print the total trip expenses incurred by sales person with ID 504
 - ii. Give the trip details for the trip that exceeded Rs. 10,000/-
 - iii. Print the sales person ID and Name of the sales men who took trips to Delhi

Problem to Solve: Writing Relational Algebra Expression

Consider the following three tables

- **SALESPERSON**(SalesPersonID,Name)
- **TRIP**(SalesPersonID, From, To, Trip-ID)
- **EXPENSE**(TripID, Amount)

SALESPERSON

SalesPersonID	Name
504	Avinash
505	Balaji
506	Chandan

EXPENSE

Trip-ID	Amount
10	10000
11	8000
12	15000

TRIP

SalesPersonID	From	To	Trip-ID
504	Chennai	Delhi	10
504	Bangalore	Bombay	11
505	Bangalore	Srinagar	12

Problem to Solve: Writing Relational Algebra Expression

EXPENSE

Trip-ID	Amount
10	10000
11	8000
12	15000

TRIP

SalesPersonID	From	To	Trip-ID
504	Chennai	Delhi	10
504	Bangalore	Bombay	11
505	Bangalore	Srinagar	12

result

18000

Problem to Solve: Writing Relational Algebra Expression

EXPENSE		TRIP			
Trip-ID	Amount	SalesPersonID	From	To	Trip-ID
10	10000	504	Chennai	Delhi	10
11	8000	504	Bangalore	Bombay	11
12	15000	505	Bangalore	Srinagar	12

result

18000

```
temp1 <-  $\sigma_{SalesPersonID=504}(TRIP)$ 
temp2 <-  $\pi_{Trip-ID}(temp1)$ 
temp3 <- EXPENSE  $\bowtie$  EXPENSE.Trip-ID=temp2.Trip-ID (temp2)
result <-  $\Sigma_{Amount}$  (temp3)
```

Problem to Solve: Writing Relational Algebra Expression

Given the two relations EXPENSE and TRIP, find the salesperson ID of the salesperson who travelled to Delhi.

EXPENSE

Trip-ID	Amount
10	10000
11	8000
12	15000

TRIP

SalesPersonID	From	To	Trip-ID
504	Chennai	Delhi	10
504	Bangalore	Bombay	11
505	Bangalore	Srinagar	12

result

504 Chennai Delhi 10 10000
505 Bangalore Srinagar 12 15000

Problem to Solve: Writing Relational Algebra Expression

Given the two relations EXPENSE and TRIP, find the details of the trips whose total amount is greater than 10000.

EXPENSE

Trip-ID	Amount
10	10000
11	8000
12	15000

TRIP

SalesPersonID	From	To	Trip-ID
504	Chennai	Delhi	10
504	Bangalore	Bombay	11
505	Bangalore	Srinagar	12

result

504 Chennai Delhi 10 10000
505 Bangalore Srinagar 12 15000

temp1 <- $\sigma_{\text{Amount} > 10000}(\text{EXPENSE})$

result <- $\pi_{(\text{TRIP.SalesPersonID}, \text{TRIP.From}, \text{TRIP.To}, \text{TRIP.Trip-ID}, \text{temp2.Amount})} (\text{TRIP} \bowtie$

$\text{TRIP.Trip-ID} = \text{temp1.Trip-ID}$ (temp1))

Problem to Solve: Writing Relational Algebra Expression

iii. Print the sales person ID and Name of the sales men who took trains to Delhi

SALESPERSON

SalesPersonID	Name
504	Avinash
505	Balaji
506	Chandan

TRIP

SalesPersonID	From	To	Trip-ID
504	Chennai	Delhi	10
504	Bangalore	Bombay	11
505	Bangalore	Srinagar	12

result

504 Avinash

Problem to Solve: Writing Relational Algebra Expression

iii. Print the sales person ID and Name of the sales men who took trains to Delhi

SALESPERSON

SalesPersonID	Name
504	Avinash
505	Balaji
506	Chandan

TRIP

SalesPersonID	From	To	Trip-ID
504	Chennai	Delhi	10
504	Bangalore	Bombay	11
505	Bangalore	Srinagar	12

result

504 Avinash

```
temp1 <-  $\sigma_{To='Delhi'}(TRIP)$ 
temp2 <-  $\pi_{SalesPerson-ID}(temp1)$ 
temp3 <- SALESPERSON  $\bowtie$  SALESPERSON.SalesPersonID=temp2.SalesPerson-ID (temp2)
result <-  $\pi_{SalesPersonID, Name}(temp3)$ 
```

Problem to Solve: Writing Relational Algebra Expression

Consider the following relational schema describing a movie database

- Schedule(Theater, Title, Time)
- Movies(Title, Director, Actor)
- Produced(Producer, Title)
- See(Spectator, Title)
- Liked(Spectator, Title)
- A movie is directed by only one director but can be produced by several Producers. A spectator may like a movie without having seen it. Write the following two queries in both Relational Algebra and SQL.

Note: In relational algebra you must use the expression form and are not allowed to use linear sequence or expression trees. You are also NOT allowed renaming of relations. You may use renaming of attributes. You may use numerical comparisons (e.g. $R:A > 5$) in both SQL and Relational Algebra.

- List the people who liked movies that they have not seen
- List the producers who produced a movie that does not appear in a theater.

Problem to Solve: Writing Relational Algebra Expression

See(Spectator, Title)

Liked(Spectator, Title)

- List $\Pi_{\text{spectator}}(\text{Liked} - \text{See})$ have not seen

```
SELECT Spectator
FROM (SELECT Spectator, Title
      FROM Liked
      EXCEPT
      SELECT Spectator, Title
      FROM See
    )
```

Problem to Solve: Writing Relational Algebra Expression

Schedule(Theater, Title, Time)

Movies(Title, Director, Actor)

Produced(Producer, Title)

- List the producers who produced a movie that does not appear in a theater.

$$\Pi_{Producer}(Produced \bowtie (\Pi_{title}(Movies) - \Pi_{title}(Schedule)))$$

```
SELECT Producer
FROM Produced
WHERE title IN (SELECT title
                 FROM MOVIES
                 EXCEPT
                 SELECT title
                 FROM Schedule
               )
```

Relational Algebra Operations

- Unary Operations - operate on one relation. These include select, project and rename operators.
- Binary Operations - operate on pairs of relations. These include union, set difference, intersection, division, cartesian product, join, equality join, natural join, Left Outer join, Right outer join and full outer join.

Thank You for Your Time and Attention !

Students Should read through the

- Relational algebra example queries given in the ELMARSI and NAVATHE text book in chapter 6 of section 6.5
- Relational Model constraints and Relational database schema, Update Transactions and Dealing with constraint violation from ELMARSI and NAVATHE text book in chapter 5 of sections 5.2 and 5.3

Course – DBMS

Course Instructor

Dr. K. SUBRAHMANYAM

Department of CSE, KL UNIVERSITY

Instruction to Students

Apart from the contents presented in the slides
students should read through the text book also.

Unit 4

Transaction Processing, Concurrency Control and Recovery

Why you should Learn Transaction Processing in Database ?

What is Transaction ?

Transactions : Introduction

- Example: Bank database application
- Consider Ram and Shyam has an account in SBI bank at Basvangudi branch.



ACCOUNTS

Account Number	Name	Balance
501	Ram	150
502	Shyam	100

Transactions : Introduction

- Example: Bank database application
- Consider Ram and Shyam has an account in SBH bank at KLU branch.

ACCOUNTS

Account Number	Name	Balance
501	Ram	150
502	Shyam	100



Transaction:

Transfer Rs. 100 from Ram account to Shyam account.

Transactions : Introduction

- Example: Bank database application
- Consider Ram and Shyam has an account in SBH bank at KLU branch.



ACCOUNTS

Account Number	Name	Balance
501	Ram	150
502	Shyam	100

Transaction:

Transfer Rs. 100 Ram account to Shyam account.

Question:

What are the SQL statement to be executed to perform the above transaction ?

Transactions : Introduction

- Example: Bank database application
- Consider Ram and Shyam has an account in SBH bank at KLU branch.

ACCOUNTS		
Account Number	Name	Balance
501	Ram	150
502	Shyam	100



Transaction: Transfer Rs. 100 Ram account to Shyam account.

Subtract Rs. 100/- from Ram account

```
update ACCOUNTS
set Balance = Balance-100
where AccountNumber=501;
```

Transactions : Introduction

- Example: Bank database application
- Consider Ram and Shyam has an account in SBH bank at KLU branch.

ACCOUNTS		
Account Number	Name	Balance
501	Ram	150
502	Shyam	100



Transaction: Transfer Rs. 100 Ram account to Shyam account.

Subtract Rs. 100/- from Ram account

```
update ACCOUNTS  
set Balance = Balance-100  
where AccountNumber=501;
```

Add Rs. 100/- to Shyam account

```
update ACCOUNTS  
set Balance = Balance+100  
where AccountNumber=502;
```

Transactions : Introduction

- Example: Bank database application
- Consider Ram and Shyam has an account in SBH bank at KLU branch.

ACCOUNTS		
Account Number	Name	Balance
501	Ram	150
502	Shyam	100



Transaction: Transfer Rs. 100 Ram account to Shyam account.

Subtract Rs. 100/- from Ram account

```
update ACCOUNTS  
set Balance = Balance - 100  
where AccountNumber=501;
```

Add Rs. 100/- to Shyam account

```
update ACCOUNTS  
set Balance = Balance + 100  
where AccountNumber=502;
```

ACCOUNTS		
Account Number	Name	Balance
501	Ram	50
502	Shyam	200

Transactions : Introduction

- Example: Bank database application
- Consider Ram and Shyam has an account in SBH bank at KLU branch.

ACCOUNTS		
Account Number	Name	Balance
501	Ram	150
502	Shyam	100



Transaction: Transfer Rs. 100 Ram account to Shyam account.

Subtract Rs. 100/- from Ram account

Update **ACCOUNTS**

Set Balance =Balance-100

Where AccountNumber=**501**;

FIRST

Add Rs. 100/- to Shyam account

Update **ACCOUNTS**

Set Balance =Balance+100

Where AccountNumber=**502**;

SECOND

Question:

What will be the status of **ACCOUNTS** table say if **FIRST** Update SQL statement has been executed but **SECOND** Update SQL statement **has not been executed** because of electricity failure on the computer system.

Transactions : Introduction

- Example: Bank database application
- Consider Ram and Shyam has an account in SBH bank at KLU branch.

ACCOUNTS		
Account Number	Name	Balance
501	Ram	150
502	Shyam	100



Transaction: Transfer Rs. 100 Ram account to Shyam account.

Subtract Rs. 100/- from Ram account

Update **ACCOUNTS**

Set Balance =Balance-100

Where AccountNumber=501;

FIRST

Add Rs. 100/- to Shyam account

Update **ACCOUNTS**

Set Balance =Balance+100

Where AccountNumber=502;

SECOND

Question:

What will be the status of **ACCOUNTS** table say if **SECOND** Update SQL statement has been executed but **FIRST** Update SQL statement **has not been executed** because of electricity failure on the computer system.

Transactions : Introduction

- Example: Bank database application
- Consider Ram and Shyam has an account in SBH bank at KLU branch.

ACCOUNTS		
Account Number	Name	Balance
501	Ram	150
502	Shyam	100



Transaction: Transfer Rs. 100 Ram account to Shyam account.

Subtract Rs. 100/- from Ram account

Update **ACCOUNTS**

Set Balance =Balance-100

Where AccountNumber=**501**;

Add Rs. 100/- to Shyam account

Update **ACCOUNTS**

Set Balance =Balance+100

Where AccountNumber=**502**;

FIRST

SECOND

SOLUTION:

Both **FIRST** and **SECOND** Update SQL statements should be executed successfully for transaction to complete

In situations when any one of the UPDATE SQL statements i.e., **FIRST or SECOND** has been executed then transaction should be **aborted** i.e., in ACCOUNTS table Balance column should not be changed

Demonstration

Placing Transaction in between START TRANSACTION
and COMMIT.

Demonstration

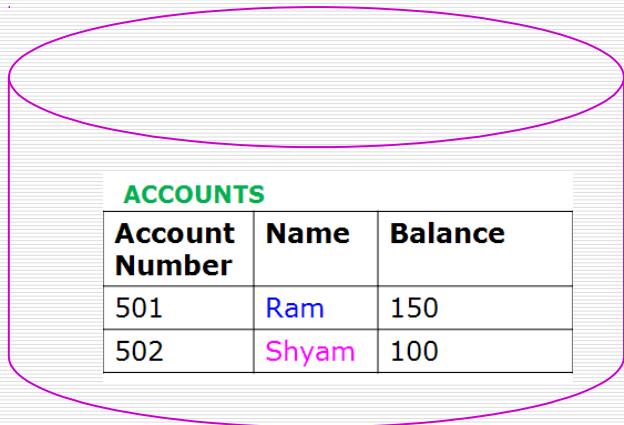
```
DROP TABLE IF EXISTS accounts;
CREATE TABLE accounts ( account_id INT PRIMARY KEY, owner
VARCHAR(30), balance FLOAT );
INSERT INTO accounts VALUES (501, 'Ram', 150.0);
INSERT INTO accounts VALUES (502, 'Shyam', 100.0);

select * from accounts;
```

ACCOUNTS

Account Number	Name	Balance
501	Ram	150
502	Shyam	100

Demonstration



User 1

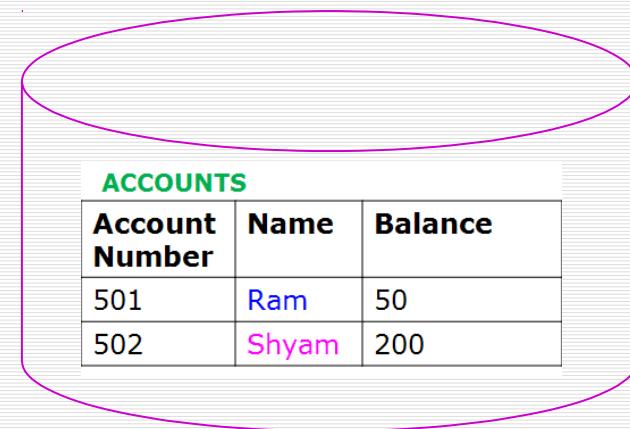
```
use bank;  
START TRANSACTION;  
UPDATE accounts SET balance = balance - 100 WHERE Name = 'Ram';  
UPDATE accounts SET balance = balance + 100 WHERE Name = 'Shyam';  
COMMIT;  
select * from accounts;
```

Demonstration

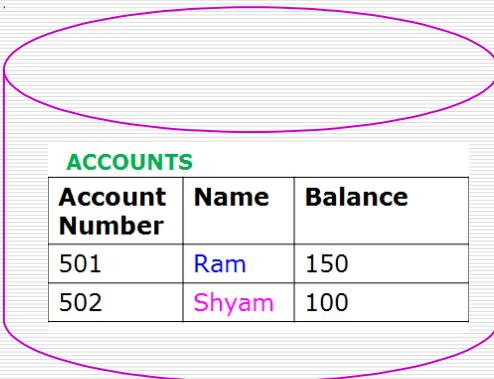


User 1

```
use bank;  
START TRANSACTION;  
UPDATE accounts SET balance = balance - 100 WHERE Name = 'Ram';  
UPDATE accounts SET balance = balance + 100 WHERE Name = 'Shyam';  
COMMIT;  
select * from accounts;
```



Demonstration



Bank database



User 1

```
use bank;  
START TRANSACTION;  
UPDATE accounts SET balance = balance - 100 WHERE owner = 'Ram';  
UPDATE accounts SET balance = balance + 100 WHERE owner =  
'Shyam';  
COMMIT;
```



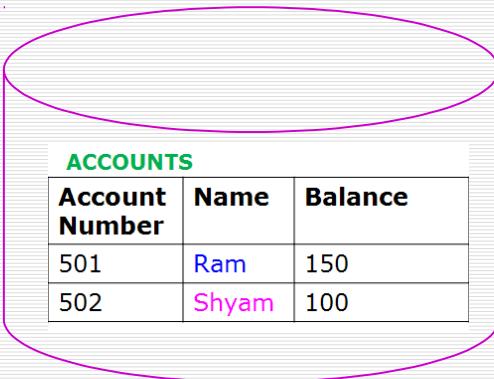
User 2

```
Use bank;  
select * from accounts;
```

Question:

What will be the balance of Ram and Shyam when User 2 executes Select statement on accounts table

Demonstration



Bank database



User 1

```
use bank;  
START TRANSACTION;  
UPDATE accounts SET balance = balance - 100 WHERE owner = 'Ram';  
UPDATE accounts SET balance = balance + 100 WHERE owner = 'Shyam';  
COMMIT;
```

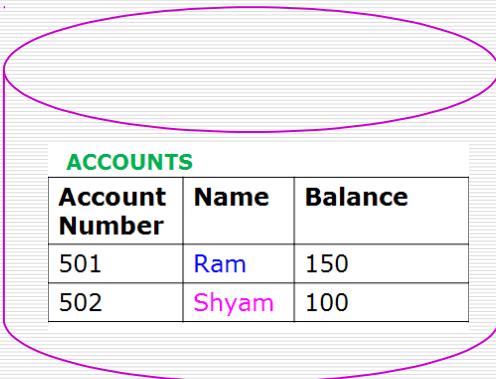


User 2

```
Use bank;  
select * from accounts;
```

ACCOUNTS		
Account Number	Name	Balance
501	Ram	50
502	Shyam	200

Demonstration



Bank database



User 1

```
use bank;  
START TRANSACTION;  
UPDATE accounts SET balance = balance - 100 WHERE owner = 'Ram';  
UPDATE accounts SET balance = balance + 100 WHERE owner = 'Shyam';  
COMMIT;
```



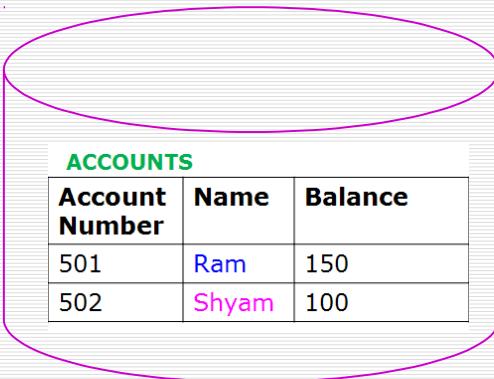
User 2

```
Use bank;  
select * from accounts;
```

Question:

What will be the balance of Ram and Shyam when User2 executes Select statement on accounts table after User1 has executed the set statements mentioned above for him

Demonstration



Bank database



User1

```
use bank;  
START TRANSACTION;  
UPDATE accounts SET balance = balance - 100 WHERE owner = 'Ram';  
UPDATE accounts SET balance = balance + 100 WHERE owner = 'Shyam';  
COMMIT;
```



User2

```
Use bank;  
select * from accounts;
```

ACCOUNTS		
Account Number	Name	Balance
501	Ram	150
502	Shyam	100

Changes done by User1
Will not be seen by User2
because User1 has not completed
the transaction i.e
COMMIT statement
has not been executed by User1

Transactions

Example:

- **Transaction:** Transfer Rs. 100 from Ram account to Shyam account.

UPDATE accounts SET balance = balance - 100 WHERE owner = 'Ram';
UPDATE accounts SET balance = balance + 100 WHERE owner = 'Shyam';

Above two Update statements must run, or neither must run. You cannot have the money being transferred out of one person's account, and then 'disappearing' if for some reason the second query fails. Both these queries form one *transaction*.

A transaction is simply a number of individual queries that are grouped together in between **START TRANSACTION** and **COMMIT** statement.

START TRANSACTION

UPDATE accounts SET balance = balance - 100 WHERE owner = 'Ram';
UPDATE accounts SET balance = balance + 100 WHERE owner = 'Shyam';

COMMIT

Transactions : Basic Definition

- A **transaction ("TXN")** is a sequence of one or more ***operations*** (reads or writes) which reflects ***a single real-world transition***.
- In the real world, a TXN either happened completely or not at all
- **Examples:**
 - Transfer money between accounts
 - Purchase a group of products
 - Register for a class (either waitlist or allocated)

START TRANSACTION

 UPDATE Product SET Price = Price - 1.99 WHERE pname = 'Gizmo'
 COMMIT

Transactions in SQL

- In “ad-hoc” SQL:
 - Default: each statement = one transaction

- In a program, multiple statements can be grouped together as a transaction:

```
START TRANSACTION
```

```
    UPDATE Bank SET amount = amount - 100 WHERE Name = 'Ram'
```

```
    UPDATE Bank SET amount = amount + 100 WHERE Name =  
'Shyam'
```

```
COMMIT
```

Model of Transaction for 15CS4DCDBM

Note: For 15CS4DCDBM, we assume that the DBMS *only sees reads and writes to data*

- User may do much more
- In real systems, databases do have more info...

Model of Transaction: Read-Write Operations

- `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.

Model of Transaction: Read-Write Operations

READ AND WRITE OPERATIONS:

- Basic unit of data transfer from the disk to the computer main memory is one block. In general, a data item (what is read or written) will be the field of some record in the database, although it may be a larger unit such as a record or even a whole block.
- `read_item(X)` command includes the following steps:
 - Find the address of the disk block that contains item X.
 - Copy that disk block into a buffer in main memory (if that disk block is not already in some main memory buffer).
 - Copy item X from the buffer to the program variable named X.

Model of Transaction: Read Operations

Example of transaction Read Operation: **read(balance)**

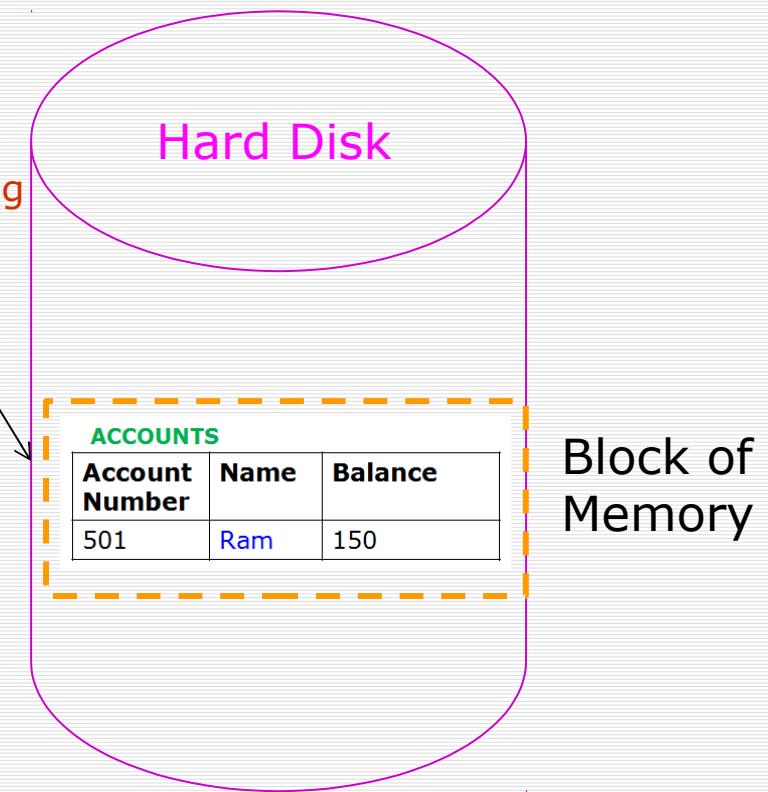
```
select balance from accounts;
```

Model of Transaction: Read Operations

Example of transaction Read Operation: **read(balance)**

select balance from **accounts**;

Step 1
Find address
Of the disk
Block containing
Balance

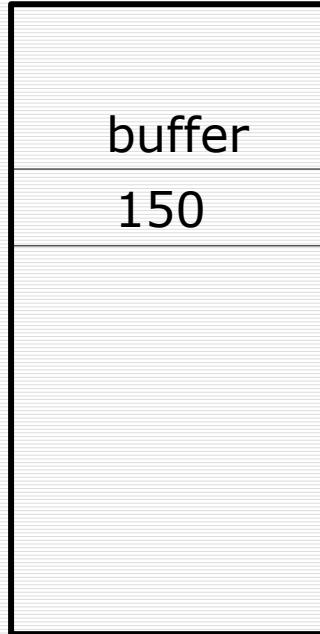


Model of Transaction: Read Operations

Example of transaction Read Operation: **read(balance)**

select balance from **accounts**;

Main memory



Step 1
Find address
Of the disk
Block containing
Balance

Step 2
Copy
Disk block
into buffer of
Main memory

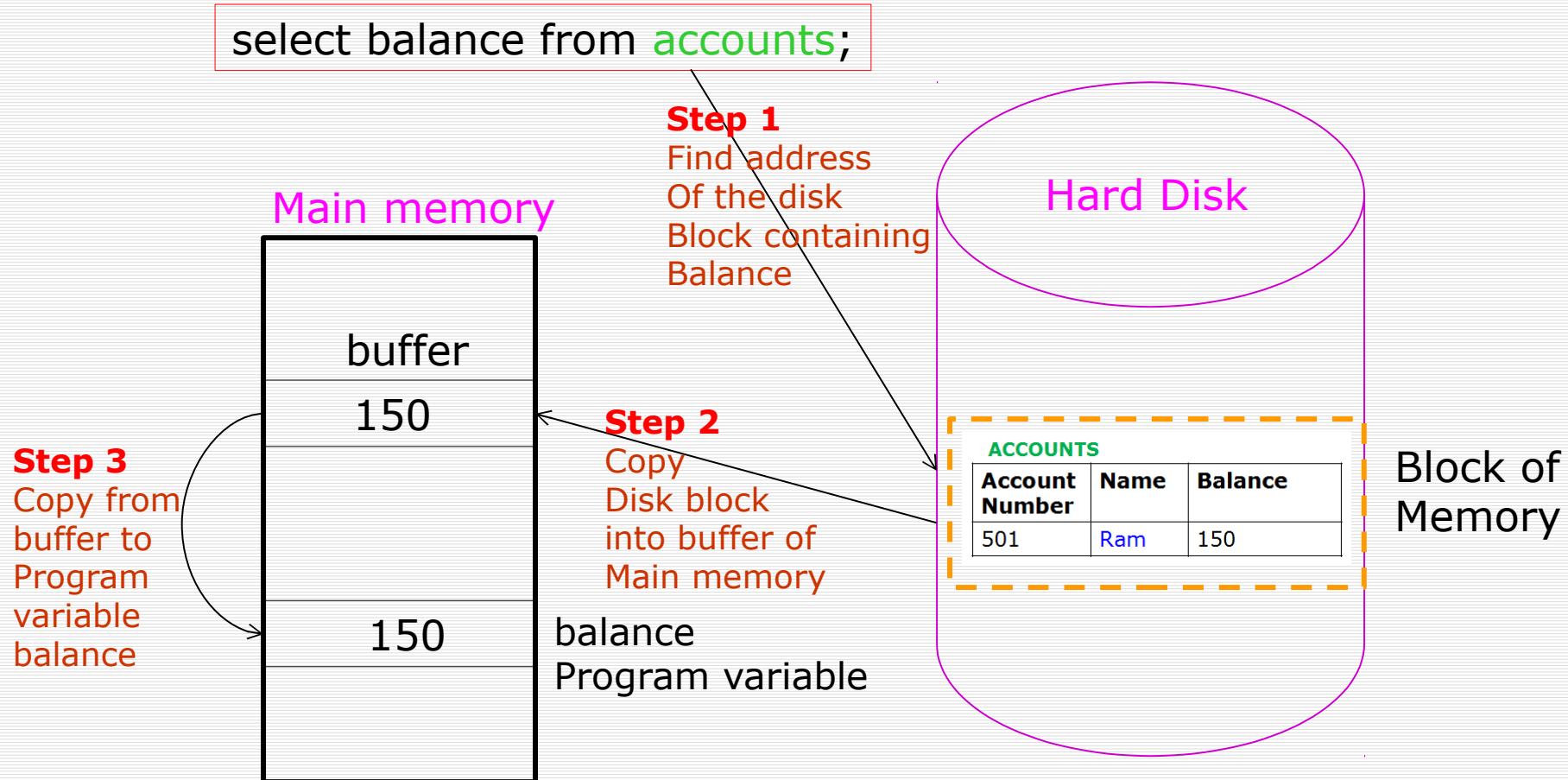
Hard Disk

ACCOUNTS		
Account Number	Name	Balance
501	Ram	150

Block of
Memory

Model of Transaction: Read Operations

Example of transaction Read Operation: **read(balance)**



Model of Transaction: Read-Write Operations

READ AND WRITE OPERATIONS (contd.):

- **write_item(X)** command includes the following steps:
 - Find the address of the disk block that contains item X.
 - Copy that disk block into a buffer in main memory (if that disk block is not already in some main memory buffer).
 - Copy item X from the program variable named X into its correct location in the buffer.
 - Store the updated block from the buffer back to disk (either immediately or at some later point in time).

Model of Transaction: Read-Write Operations

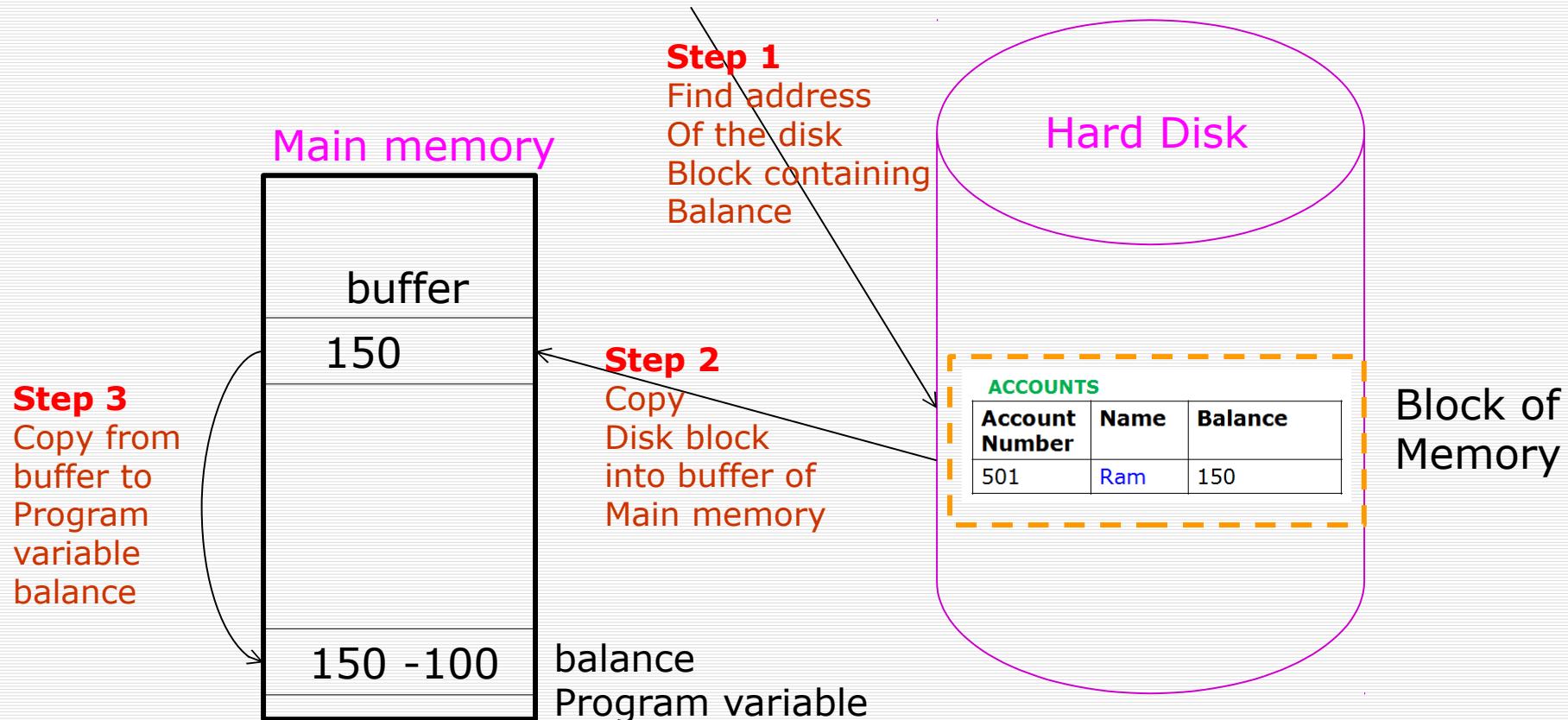
Example of transaction write Operation: **write(balance)**

Update **accounts** set balance=balance-100 where Name='Ram';

Model of Transaction: Read-Write Operations

Example of transaction write Operation: **write(balance)**

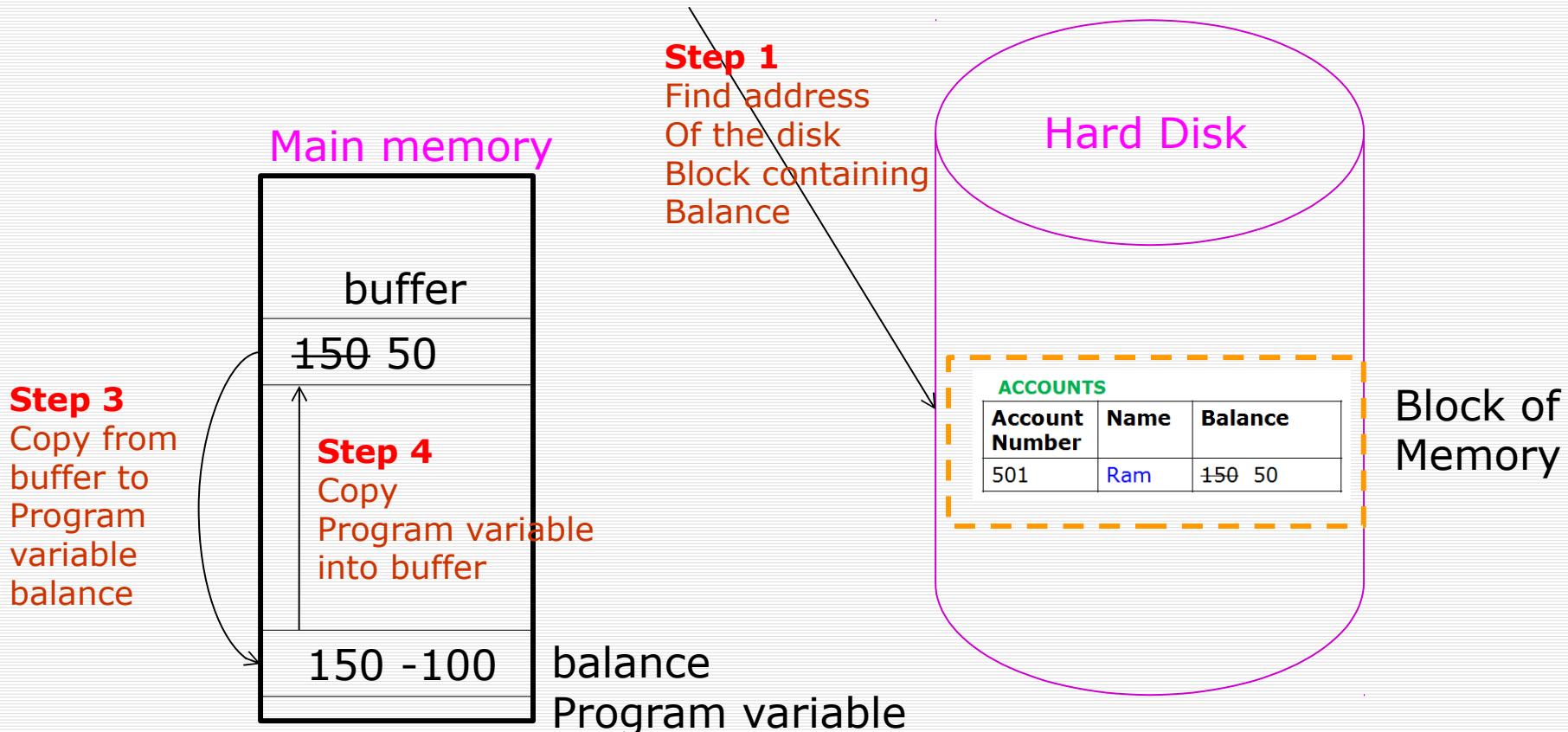
Update **accounts** set balance=balance-100 where Name='Ram';



Model of Transaction: Read-Write Operations

Example of transaction write Operation: **write(balance)**

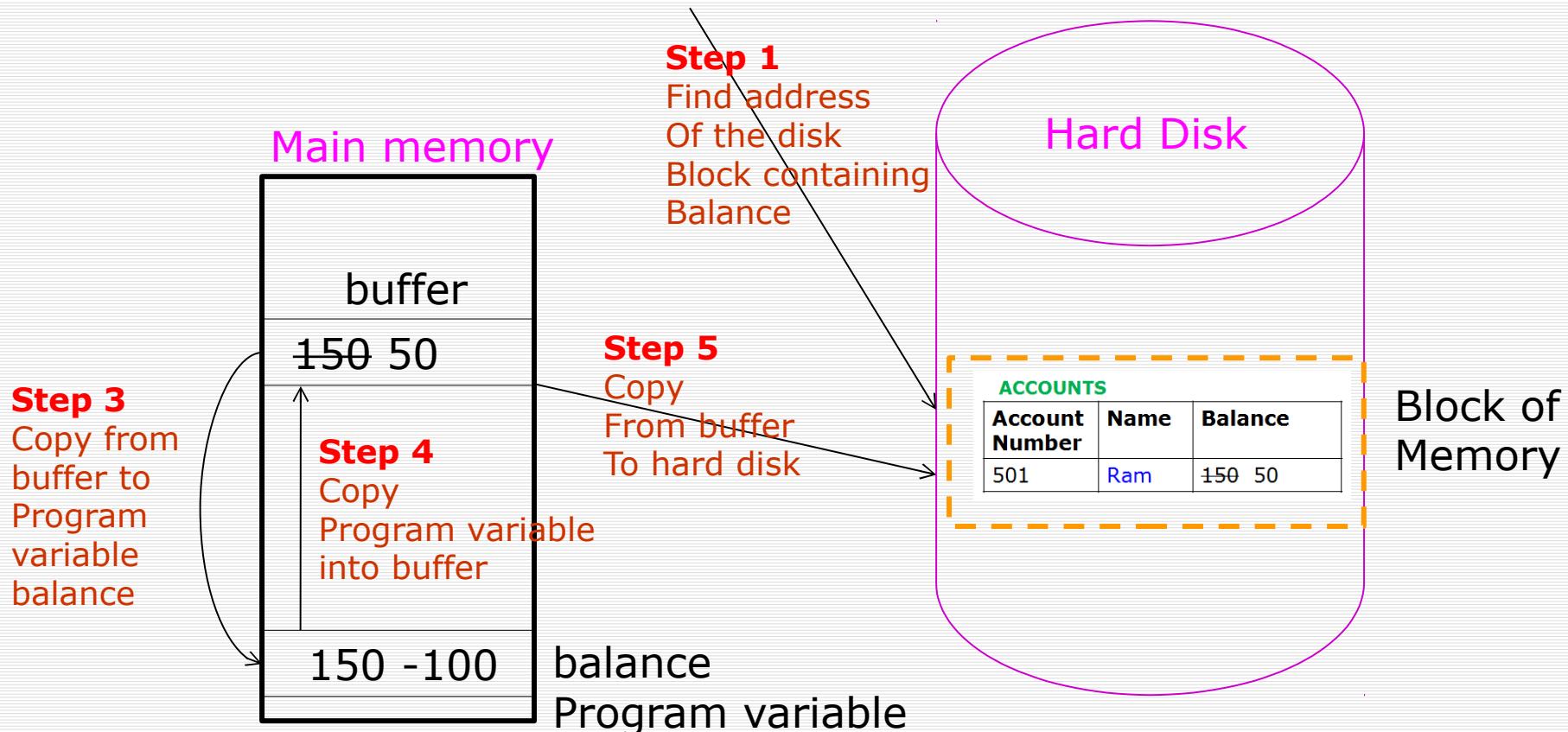
Update **accounts** set balance=balance-100 where Name='Ram';



Model of Transaction: Read-Write Operations

Example of transaction write Operation: **write(balance)**

Update **accounts** set balance=balance-100 where Name='Ram';



Problems with Concurrent Execution

□ What is Concurrent Execution ?

When Multiple users trying to access same database record in an uncontrolled manner.

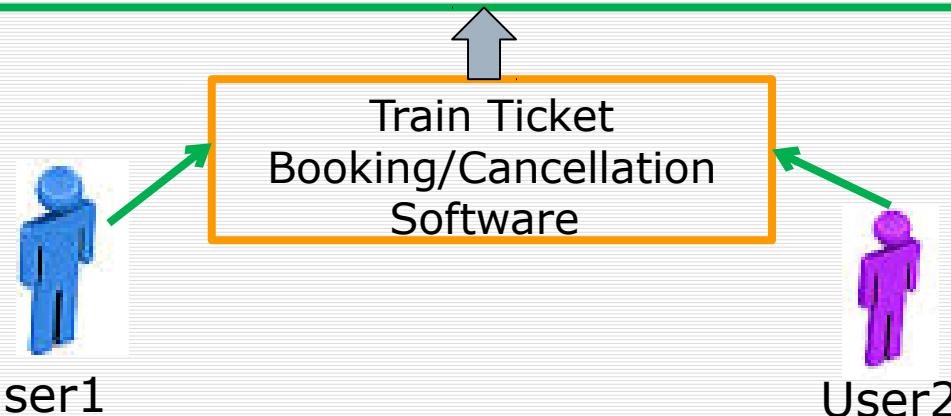
□ Problems with Concurrent execution

1. Lost Update Problem
2. Temporary Update (or Dirty Read) Problem
3. Incorrect Summary Problem
4. Unrepeatable Read

To Understand problems with concurrent executions

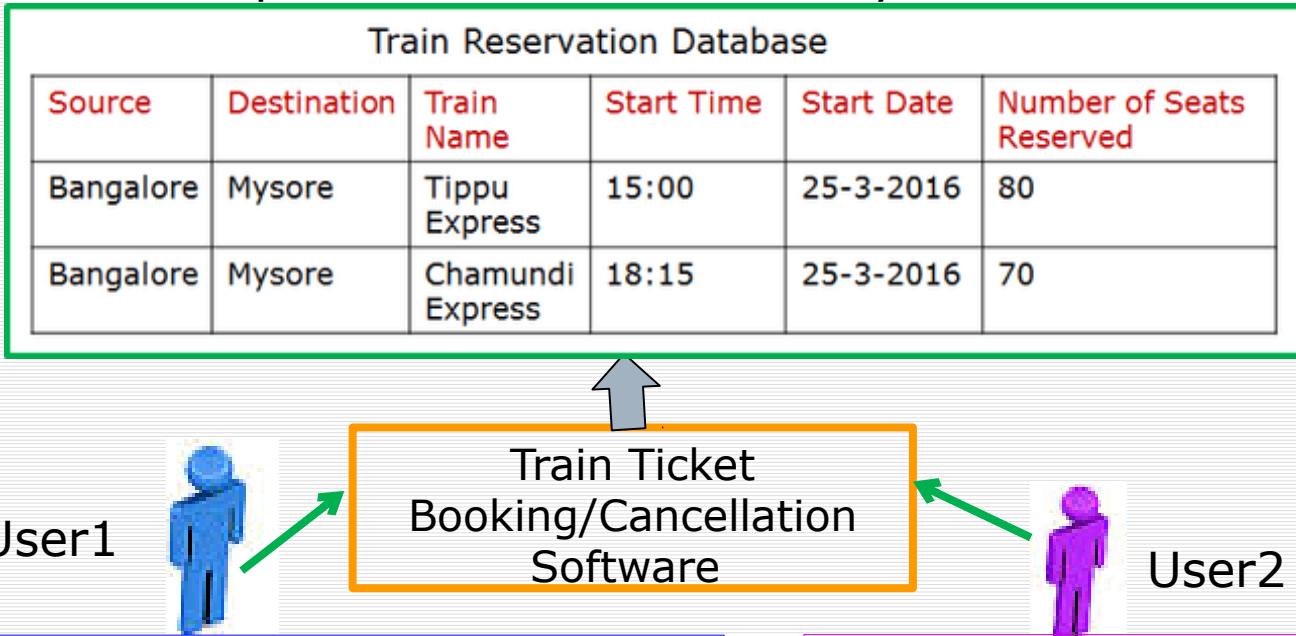
Consider an example of Train Reservation System

Train Reservation Database					
Source	Destination	Train Name	Start Time	Start Date	Number of Seats Reserved
Bangalore	Mysore	Tippu Express	15:00	25-3-2016	80
Bangalore	Mysore	Chamundi Express	18:15	25-3-2016	70



To Understand problems with concurrent executions

Consider an example of Train Reservation System



User1

1. Wants to **cancel** 5 seats on **Tippu** express
2. Wants to **reserve** 5 seats on **Chamundi** express

User2

1. Wants to **reserve** 4 seats on **Tippu** express

To Understand problems with concurrent executions

Consider an example of Train Reservation System

Train Reservation Database					
Source	Destination	Train Name	Start Time	Start Date	Number of Seats Reserved
Bangalore	Mysore	Tippu Express	15:00	25-3-2016	80
Bangalore	Mysore	Chamundi Express	18:15	25-3-2016	70

reservation table



User1

- 1.Wants to **cancel** 5 seats on **Tippu** express
- 2.Wants to **reserve** 5 seats on **Chamundi** express

Update reservation set **seats=seats-5**

Where TrainName='TippuExpress';

Update reservation set **seats=seats+5**

Where TrainName='ChamundiExpress';

User2

- 1.Wants to **reserve** 4 seats on **Tippu** Express

Update reservation set **seats=seats+4**

Where TrainName='TippuExpress';

To Understand problems with concurrent executions

Train Reservation Database					
Source	Destination	Train Name	Start Time	Start Date	Number of Seats Reserved
Bangalore	Mysore	Tippu Express	15:00	25-3-2016	80
Bangalore	Mysore	Chamundi Express	18:15	25-3-2016	70



User1

1. Wants to **cancel** 5 seats on **Tippu** express
2. Wants to **reserve** 5 seats on **Chamundi** express

Read(TippuSeats)

TippuSeats=TippuSeats-5

Write(TippuSeats)

Read(ChamundiSeats)

ChamundiSeats=ChamundiSeats+5

Write(ChamundiSeats)

User2

1. Wants to **reserve** 4 seats on **Tippu** Express

Read(TippuSeats)

TippuSeats=TippuSeats+4

Write(TippuSeats)

To Understand problems with concurrent executions

Train Reservation Database					
Source	Destination	Train Name	Start Time	Start Date	Number of Seats Reserved
Bangalore	Mysore	Tippu Express	15:00	25-3-2016	80
Bangalore	Mysore	Chamundi Express	18:15	25-3-2016	70



Transaction1

```
Read(TippuSeats)
TippuSeats=TippuSeats-5
Write(TippuSeats)
Read(ChamundiSeats)
ChamundiSeats=ChamundiSeats+5
Write(ChamundiSeats)
```

Transaction2

```
Read(TippuSeats)
TippuSeats=TippuSeats+4
Write(TippuSeats)
```

Question

What will be the total number of seats on Tippu express and Chamundi express if first User1 executes Transaction1 and then User2 Executes Transaction2

To Understand problems with concurrent executions

Train Reservation Database					
Source	Destination	Train Name	Start Time	Start Date	Number of Seats Reserved
Bangalore	Mysore	Tippu Express	15:00	25-3-2016	80 79
Bangalore	Mysore	Chamundi Express	18:15	25-3-2016	70 75



Transaction1

```
Read(TippuSeats)
TippuSeats=TippuSeats-5
Write(TippuSeats)
Read(ChamundiSeats)
ChamundiSeats=ChamundiSeats+5
Write(ChamundiSeats)
```

Transaction2

```
Read(TippuSeats)
TippuSeats=TippuSeats+4
Write(TippuSeats)
```

Answer

Tippu express seats: 79
Chamundi express seats: 75

Non-interleaved vs Interleaved transactions

Non-interleaved transaction (or Serial transaction): In this case first completely transaction T1 gets executed and then transaction T2 gets executed

Transaction1

```
Read(TippuSeats)
TippuSeats=TippuSeats-5
Write(TippuSeats)
Read(ChamundiSeats)
ChamundiSeats=ChamundiSeats+5
Write(ChamundiSeats)
```

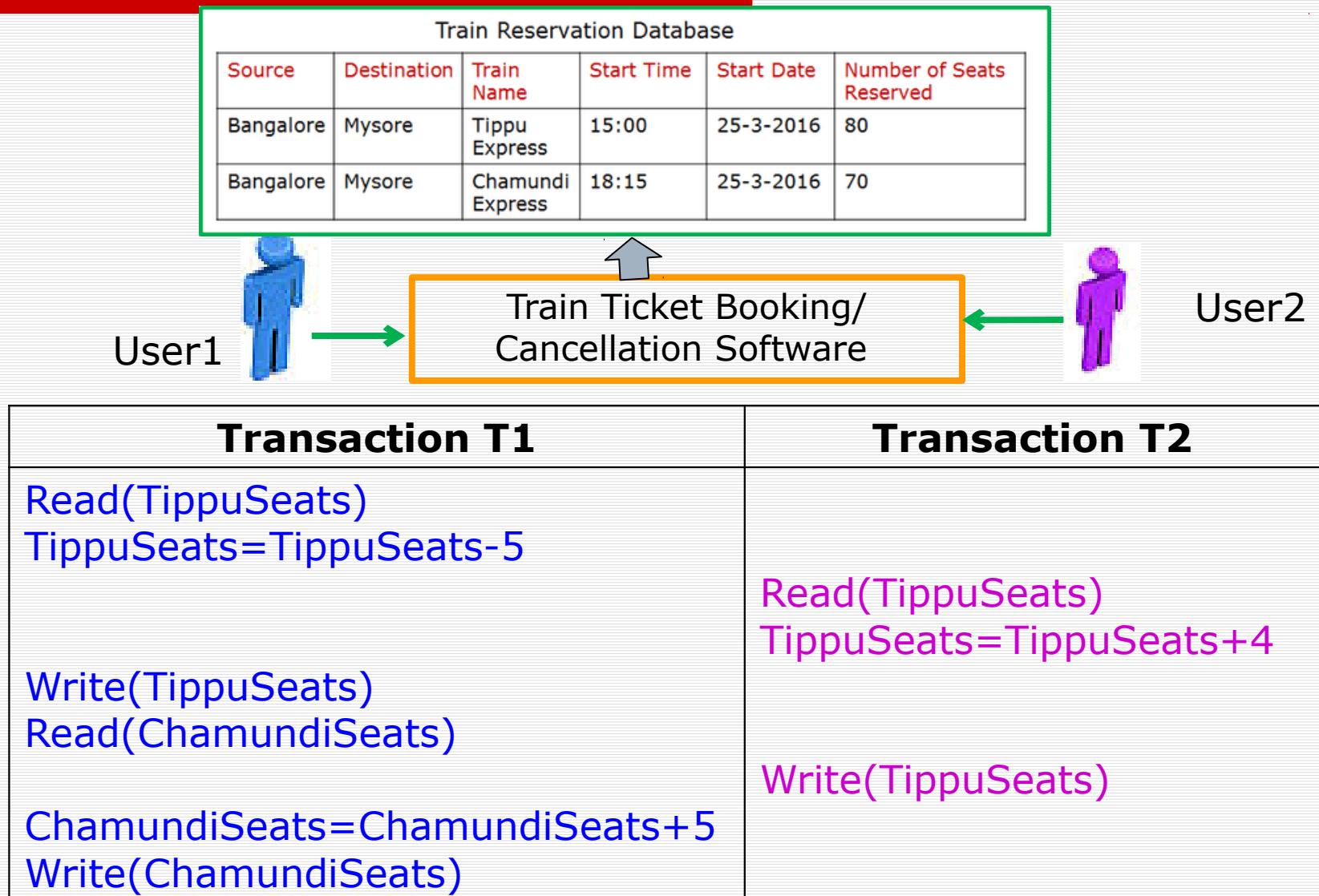
Transaction2

```
Read(TippuSeats)
TippuSeats=TippuSeats+4
Write(TippuSeats)
```

Transaction T1	Transaction T2
<pre>Read(TippuSeats) TippuSeats=TippuSeats-5 Write(TippuSeats) Read(ChamundiSeats) ChamundiSeats=ChamundiSeats+5 Write(ChamundiSeats)</pre>	<pre>Read(TippuSeats) TippuSeats=TippuSeats+4 Write(TippuSeats)</pre>

Interleaved transaction (or Non-Serital or Concurrent):
First, Part of Transaction T1 gets executed,
second Transaction T2 gets executed and third
remaining part of transaction T1 gets executed

Lost Update Problem: Example

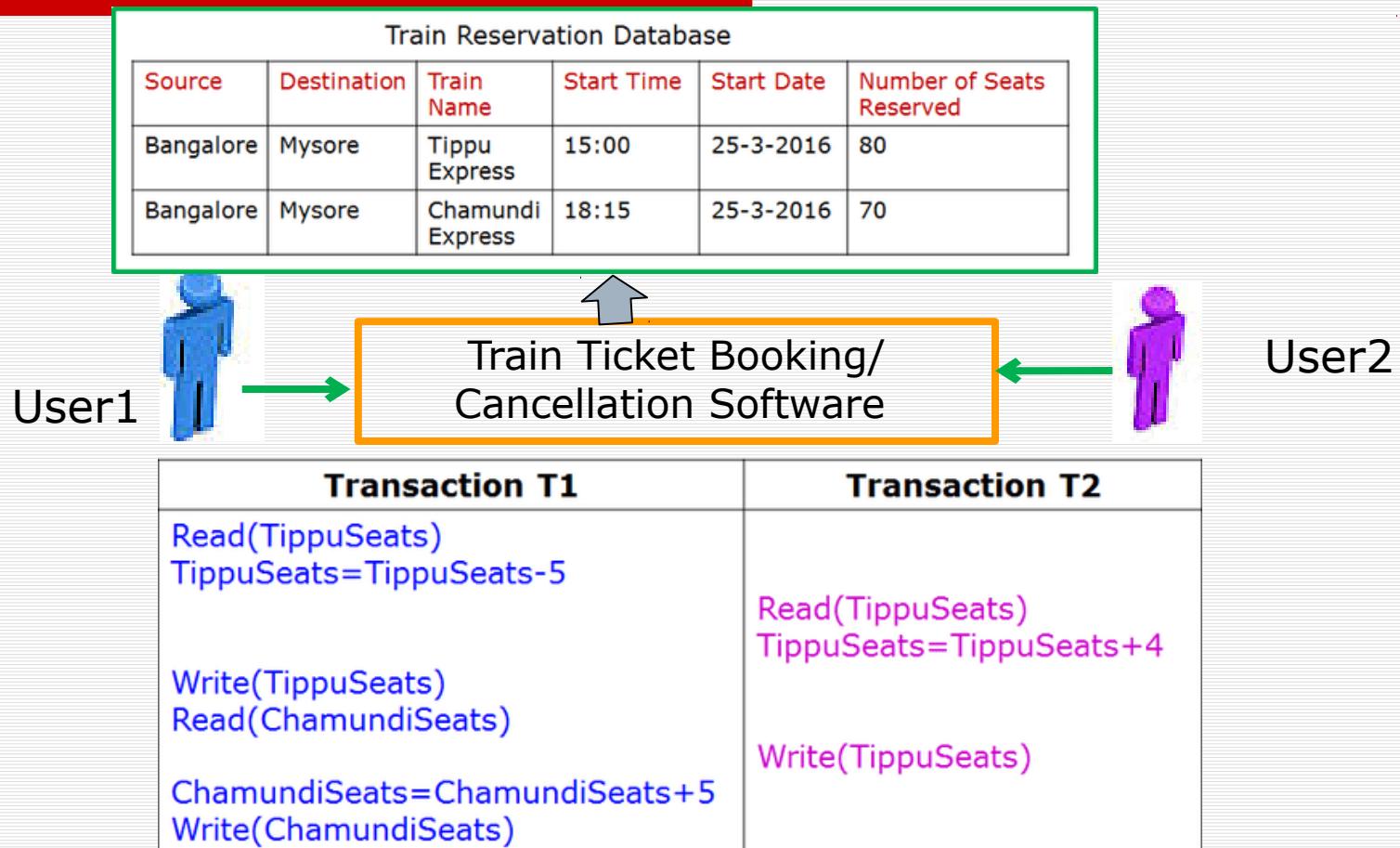


Problems with Concurrent Execution

1. 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.

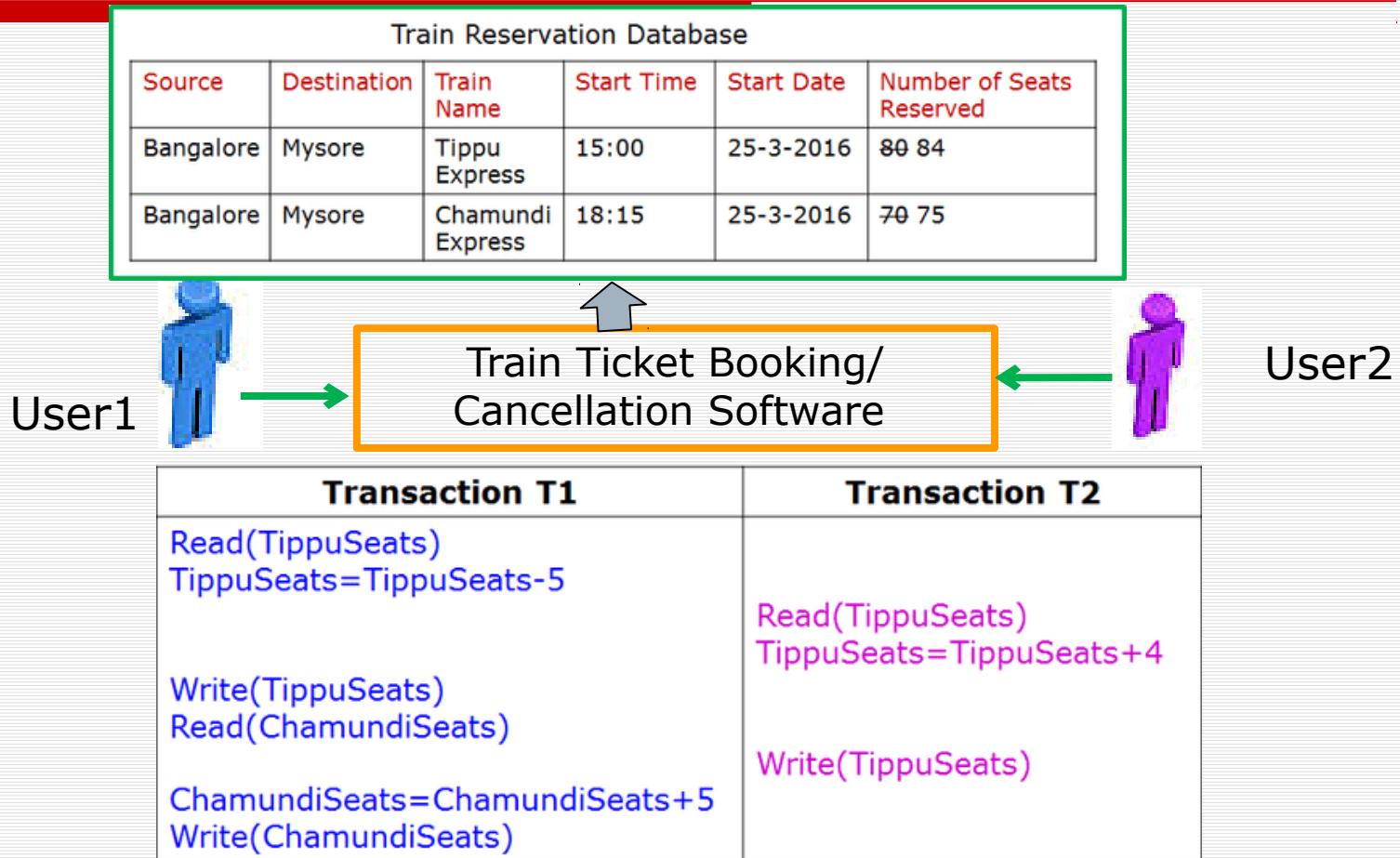
Lost Update Problem: Example



Question

What will be the total number of seats on Tippu express and Chamundi express after execution of the above set Transaction statements

Lost Update Problem: We are Loosing update



Answer

Tippu express seats: 84 <- **INCORRECT**

Update made by one Transaction
is overridden by another Transaction

Chamundi express seats: 75

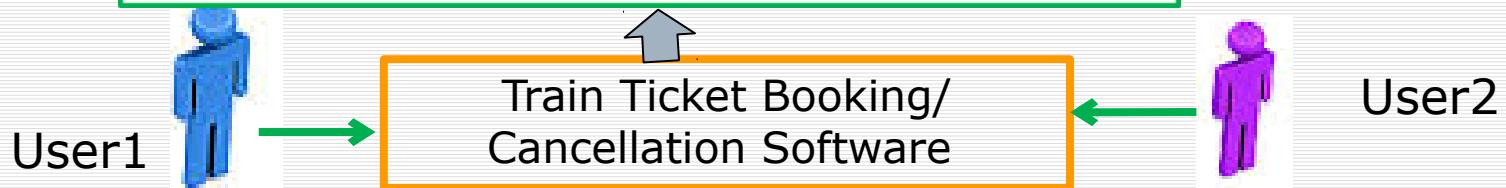
Problems with Concurrent Execution

2. The Dirty Read (or Temporary Update) 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.

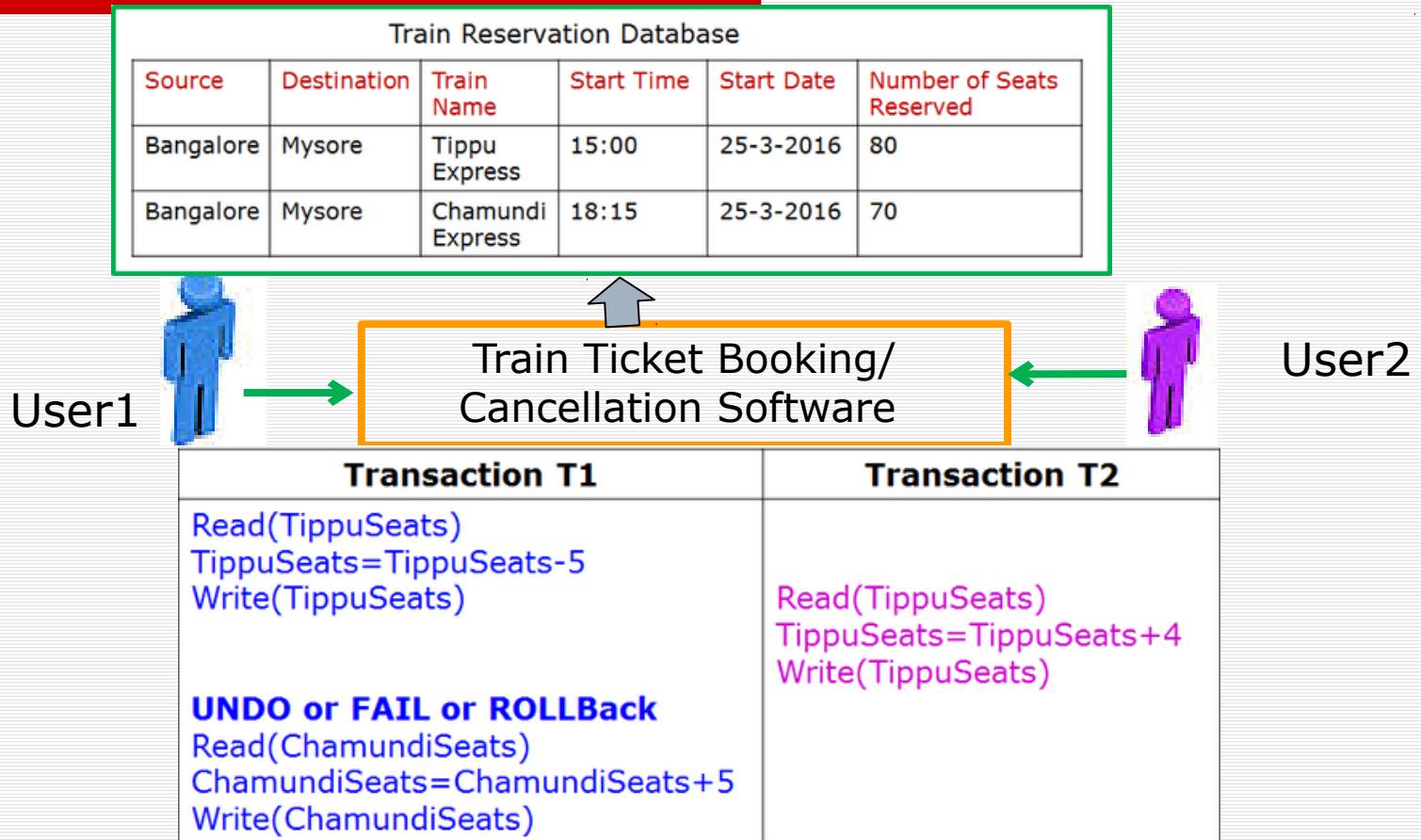
Dirty Read (or Temporary Update) Problem

Train Reservation Database					
Source	Destination	Train Name	Start Time	Start Date	Number of Seats Reserved
Bangalore	Mysore	Tippu Express	15:00	25-3-2016	80
Bangalore	Mysore	Chamundi Express	18:15	25-3-2016	70



Transaction T1	Transaction T2
<pre>Read(TippuSeats) TippuSeats=TippuSeats-5 Write(TippuSeats)</pre>	<pre>Read(TippuSeats) TippuSeats=TippuSeats+4 Write(TippuSeats)</pre>
UNDO or FAIL or ROLLBack <pre>Read(ChamundiSeats) ChamundiSeats=ChamundiSeats+5 Write(ChamundiSeats)</pre>	

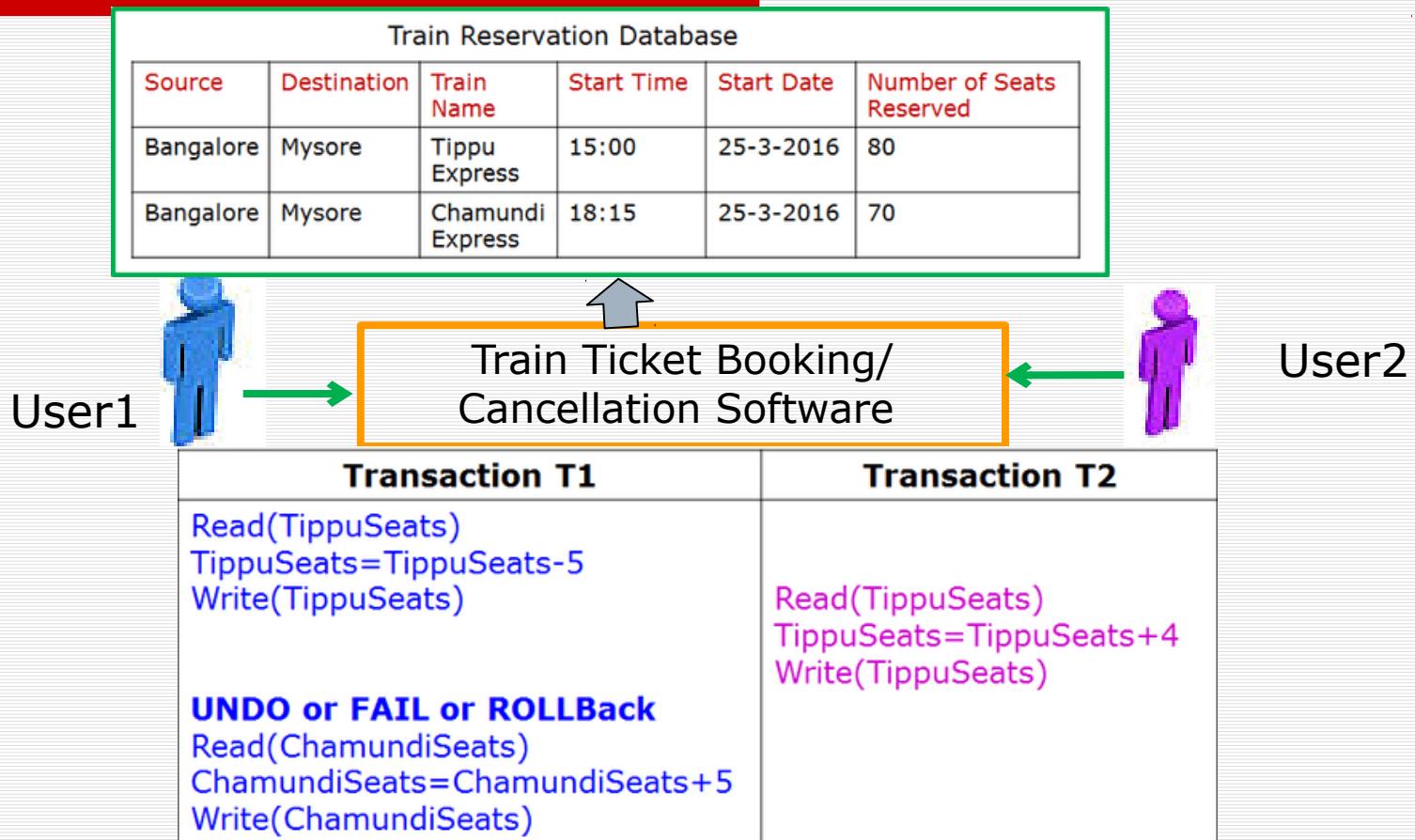
Dirty Read (or Temporary Update) Problem



Question

What will be the total number of seats on Tippu express and Chamundi express after execution of the above set Transaction statements

Dirty Read (or Temporary Update) Problem



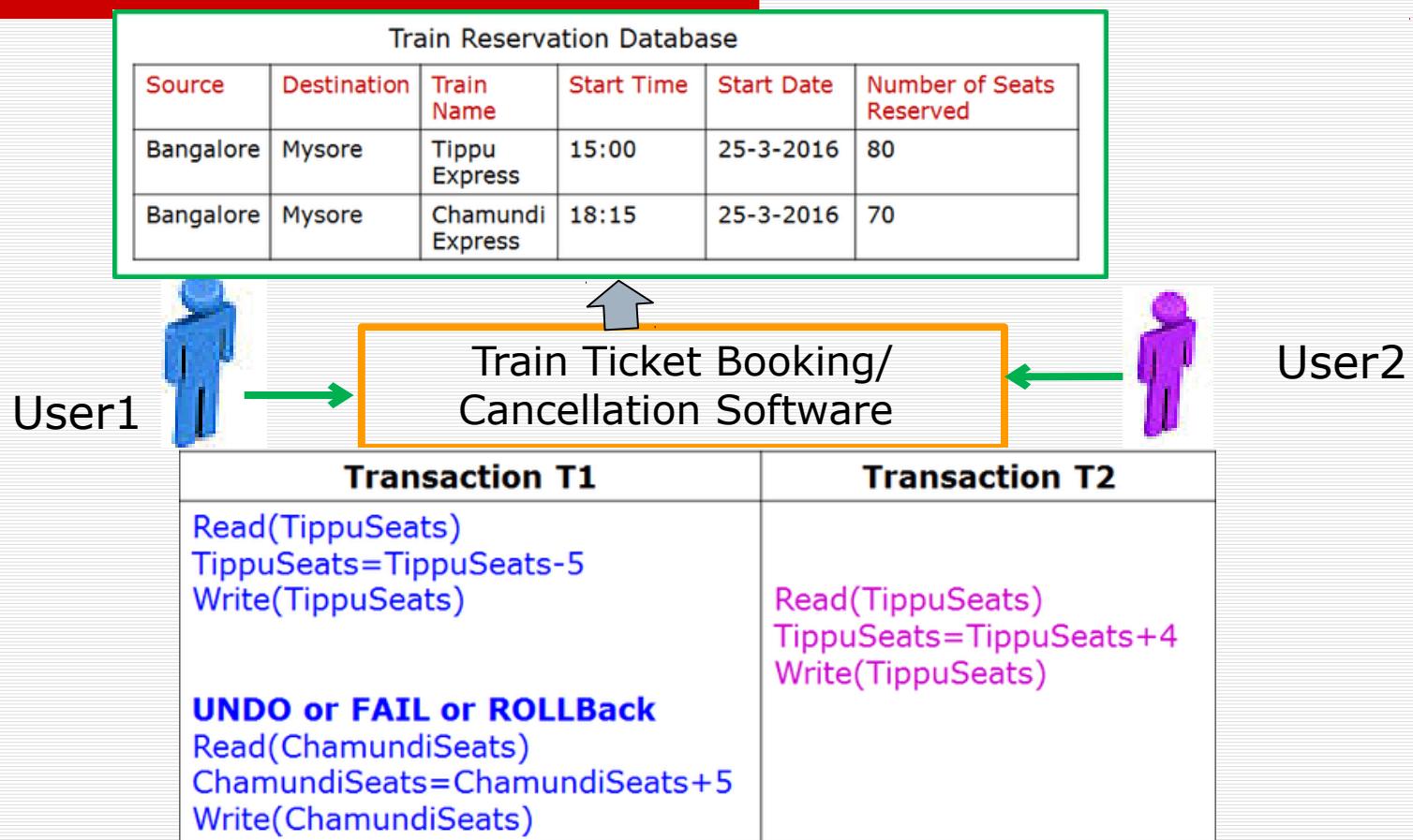
Answer

Tippu express seats: 80 <- **INCORRECT**

"Dirty read" / Reading uncommitted data
Occurring with / because of a **write conflict**

Chamundi express seats: 75

Example



Answer

Tippu express seats: 80 <- **INCORRECT**

"Dirty read" / Reading uncommitted data
Occurring with / because of a **write conflict**

Chamundi express seats: 75

Problems with Concurrent Execution

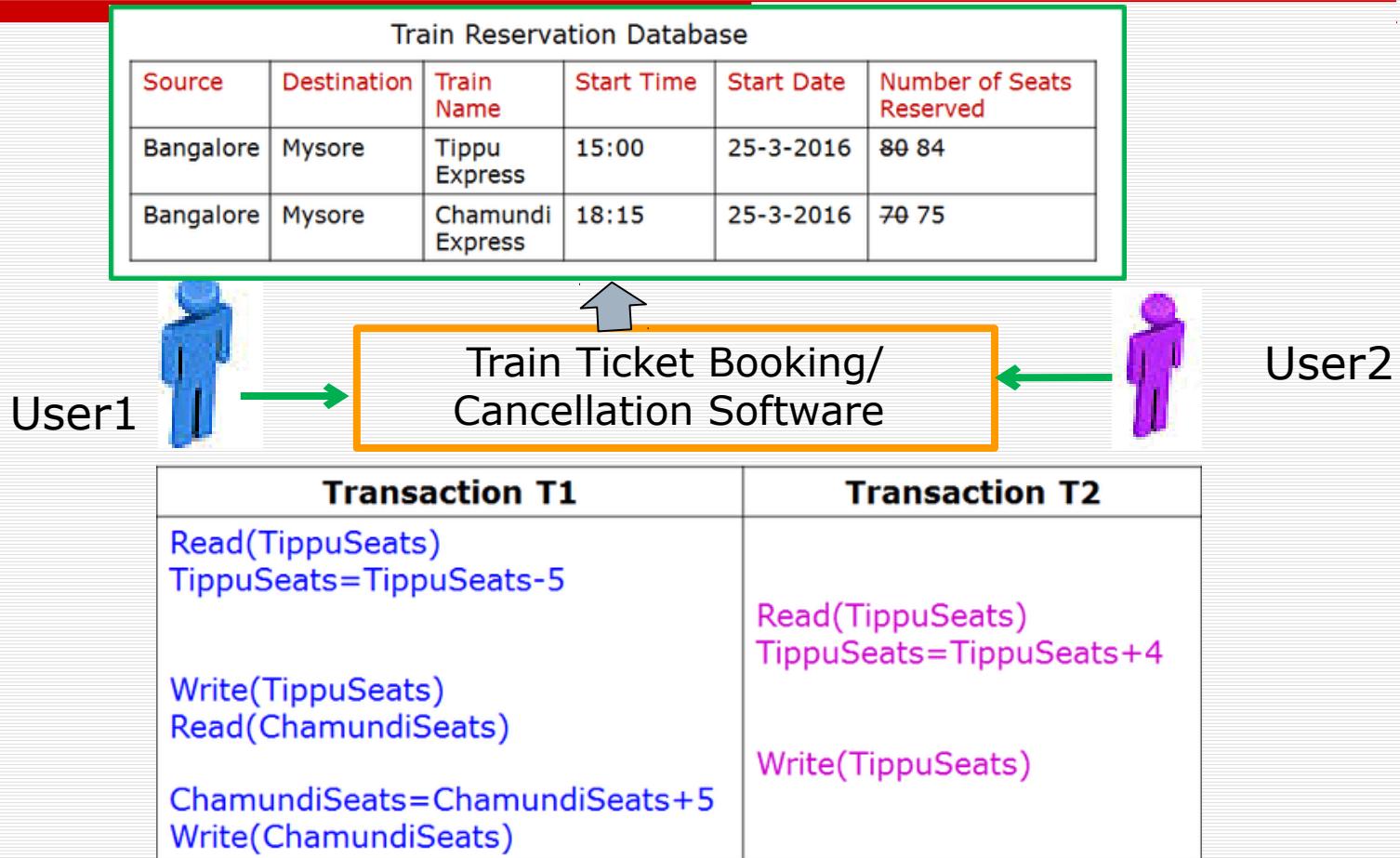
What is Concurrent Execution ?

When Multiple users trying to access same database record in an uncontrolled manner.

Problems with Concurrent execution

- 1. Lost Update Problem**
- 2. Temporary Update (or Dirty Read) Problem**
3. Incorrect Summary Problem
4. Unrepeatable Read

Lost Update Problem: We are Loosing update



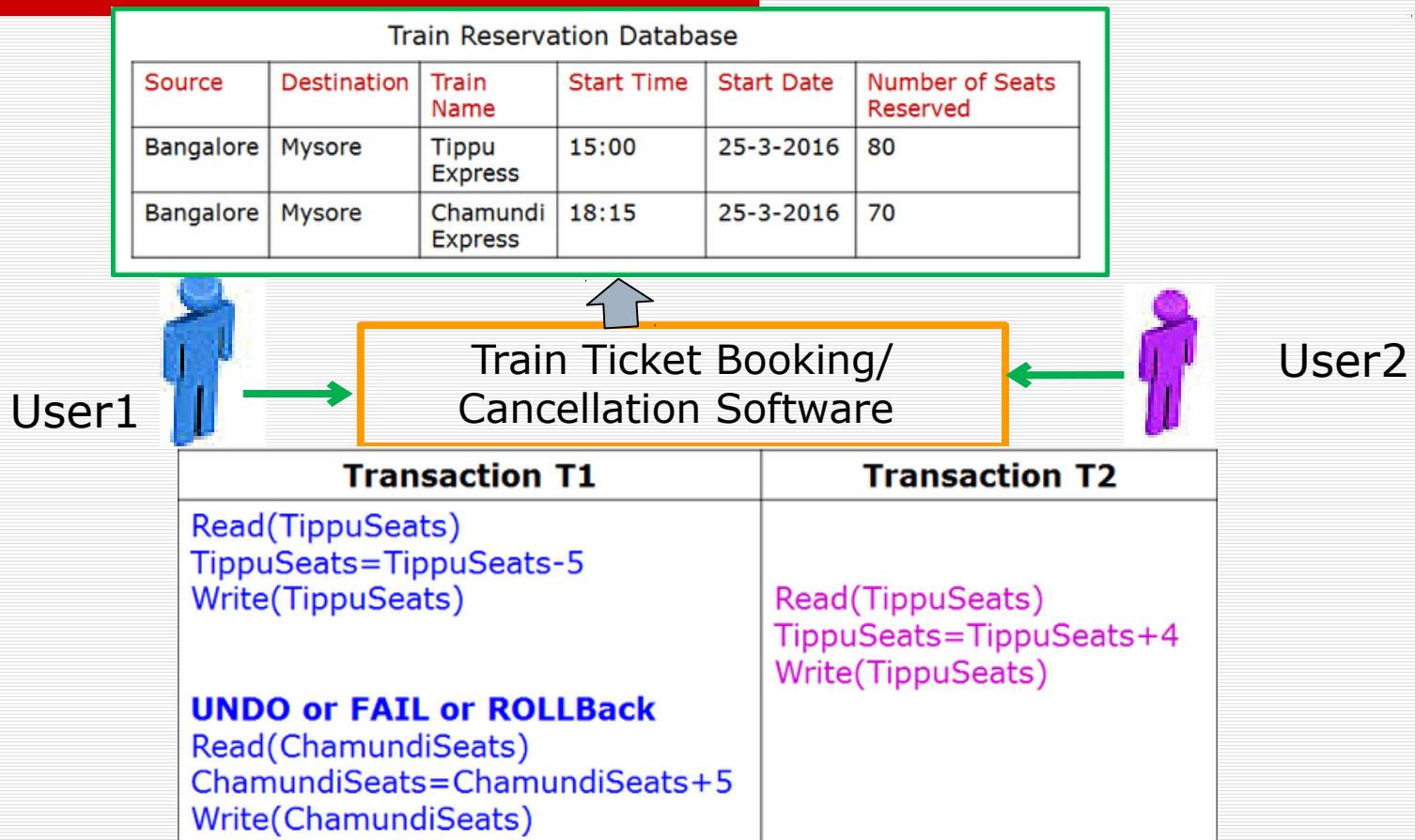
Answer

Tippu express seats: 84 <- **INCORRECT**

Update made by one Transaction
is overridden by another Transaction

Chamundi express seats: 75

Dirty Read (or Temporary Update) Problem



Answer

Tippu express seats: 80 <- **INCORRECT**

"Dirty read" / Reading uncommitted data
Occurring with / because of a **write conflict**

Chamundi express seats: 75

Problems with Concurrent Execution

□ What is Concurrent Execution ?

When Multiple users trying to access same database record in an uncontrolled manner.

□ Problems with Concurrent execution

1. Lost Update Problem
2. Temporary Update (or Dirty Read) Problem
- 3. Incorrect Summary Problem**
4. Unrepeatable Read

Example

Train Reservation Database					
Source	Destination	Train Name	Start Time	Start Date	Number of Seats Reserved
Bangalore	Mysore	Tippu Express	15:00	25-3-2016	80
Bangalore	Mysore	Chamundi Express	18:15	25-3-2016	70



User3

Transaction T1

```
Read(TippuSeats)
TippuSeats=TippuSeats-5
Write(TippuSeats)
Read(ChamundiSeats)
ChamundiSeats=ChamundiSeats+5
Write(ChamundiSeats)
```

Transaction T3

```
sum:=0
Read(TippuSeats)
sum=sum+TippuSeats
Read(ChamundiSeats)
sum=sum+ChamundiSeats
```

Example

Train Reservation Database					
Source	Destination	Train Name	Start Time	Start Date	Number of Seats Reserved
Bangalore	Mysore	Tippu Express	15:00	25-3-2016	80
Bangalore	Mysore	Chamundi Express	18:15	25-3-2016	70



Transaction T1

```
Read(TippuSeats)
TippuSeats=TippuSeats-5
Write(TippuSeats)
Read(ChamundiSeats)
ChamundiSeats=ChamundiSeats+5
Write(ChamundiSeats)
```



User3

Transaction T3

```
sum:=0
Read(TippuSeats)
sum=sum+TippuSeats
Read(ChamundiSeats)
sum=sum+ChamundiSeats
```

Question

Say if first Transaction T1 has been executed first and second Transaction T3 has been executed then,

What will be the total number of seats on Tippu express & Chamundi express; and **sum** value

Example

Train Reservation Database					
Source	Destination	Train Name	Start Time	Start Date	Number of Seats Reserved
Bangalore	Mysore	Tippu Express	15:00	25-3-2016	80
Bangalore	Mysore	Chamundi Express	18:15	25-3-2016	70



Transaction T1

```
Read(TippuSeats)
TippuSeats=TippuSeats-5
Write(TippuSeats)
Read(ChamundiSeats)
ChamundiSeats=ChamundiSeats+5
Write(ChamundiSeats)
```



User3

Transaction T3

```
sum:=0
Read(TippuSeats)
sum=sum+TippuSeats
Read(ChamundiSeats)
sum=sum+ChamundiSeats
```

Answer

Tippu express seats=75
Chamundi express seats=75
sum=150

Problems with Concurrent Execution

3. 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.

Incorrect Summary Problem

Train Reservation Database					
Source	Destination	Train Name	Start Time	Start Date	Number of Seats Reserved
Bangalore	Mysore	Tippu Express	15:00	25-3-2016	80
Bangalore	Mysore	Chamundi Express	18:15	25-3-2016	70

User1



User3



Transaction T1

```
Read(TippuSeats)  
TippuSeats=TippuSeats-5  
Write(TippuSeats)
```

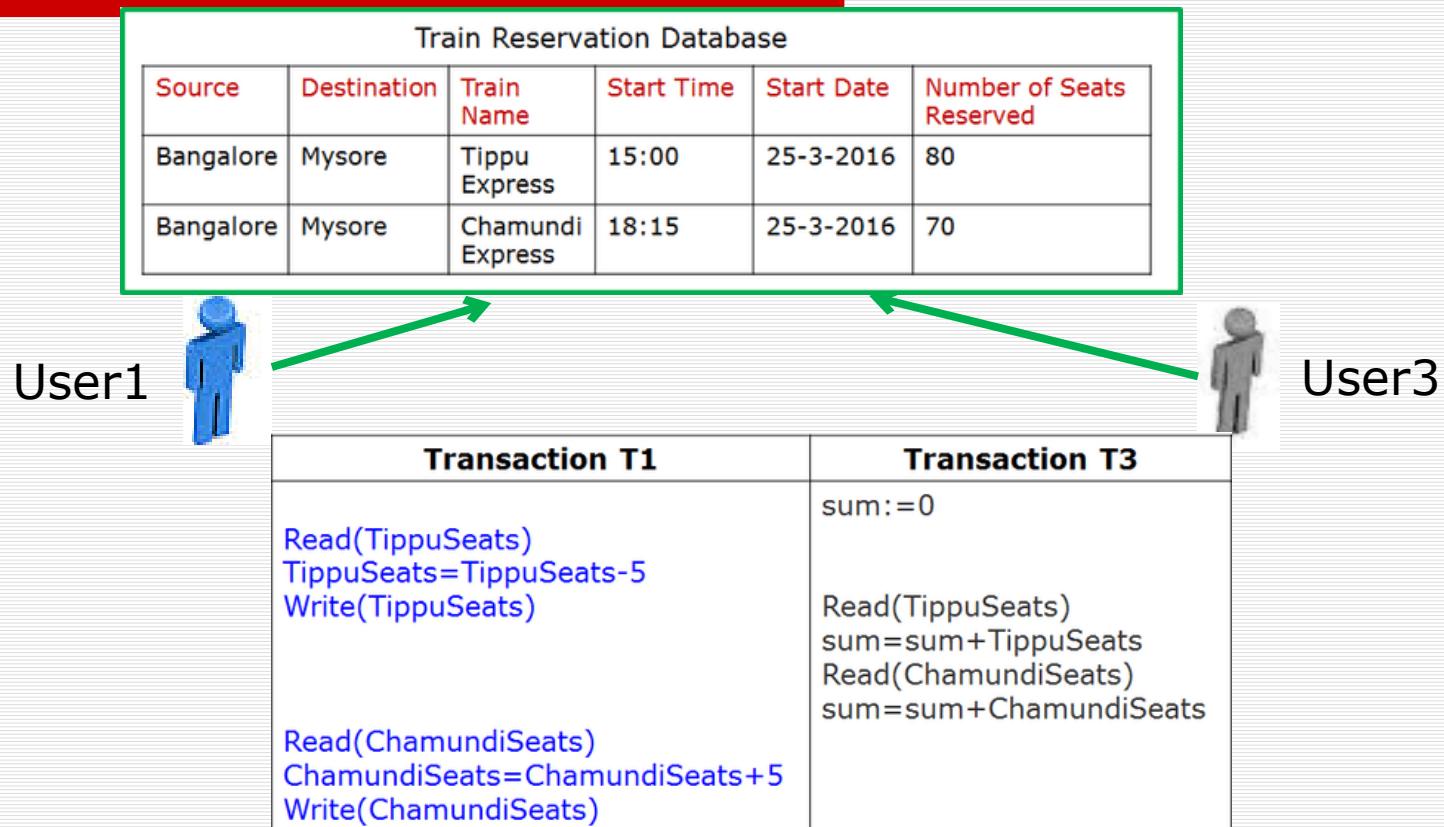
```
Read(ChamundiSeats)  
ChamundiSeats=ChamundiSeats+5  
Write(ChamundiSeats)
```

Transaction T3

```
sum:=0
```

```
Read(TippuSeats)  
sum=sum+TippuSeats  
Read(ChamundiSeats)  
sum=sum+ChamundiSeats
```

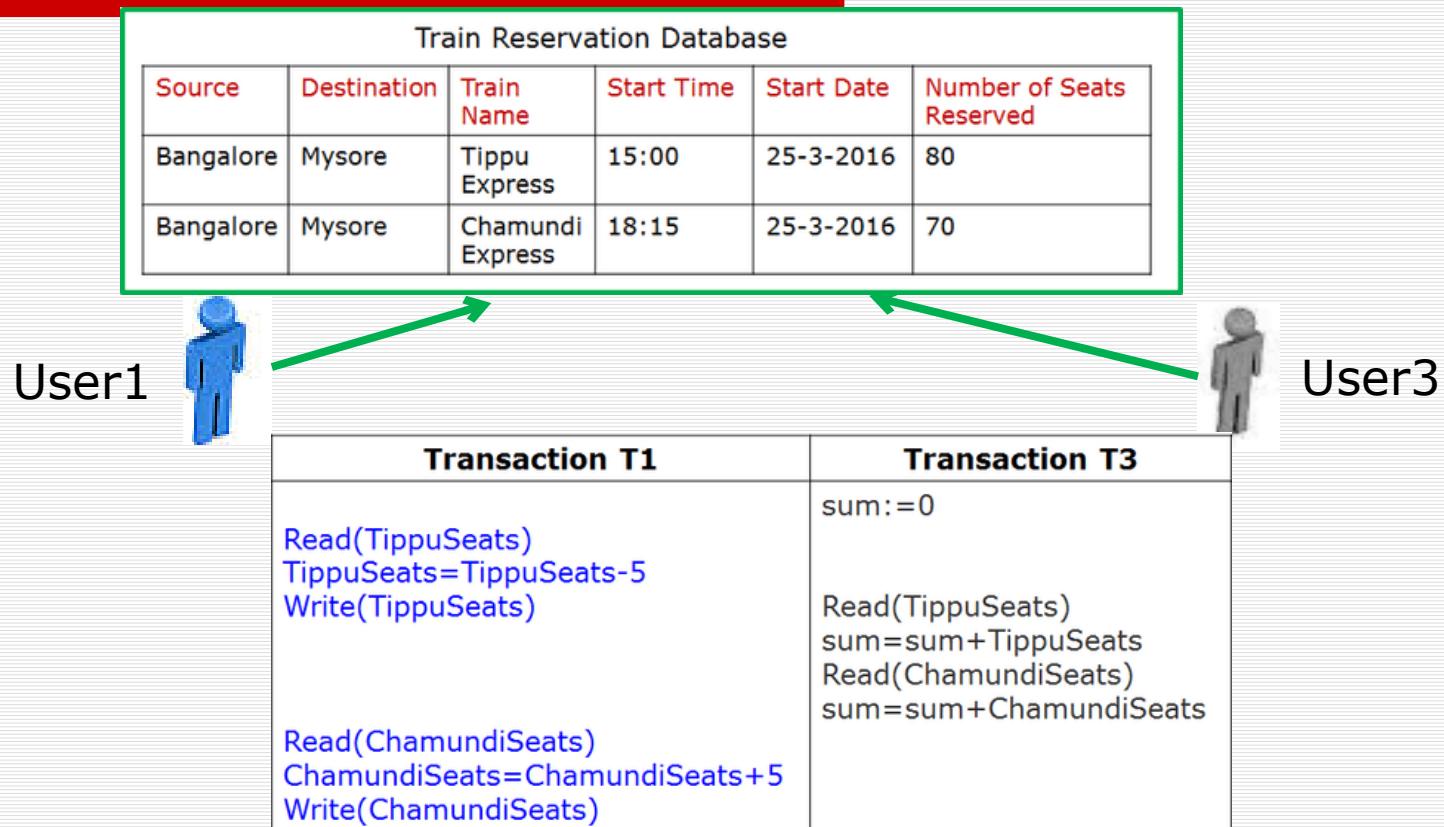
Incorrect Summary Problem



Question

What will be the total number of seats on Tippu express & Chamundi express; and sum value when above set of transactions statements are executed

Incorrect Summary Problem



Answer

Tippu express seats=75
Chamundi express seats=75
Sum=145 <- **INCORRECT**

Problems with Concurrent Execution

□ What is Concurrent Execution ?

When Multiple users trying to access same database record in an uncontrolled manner.

□ Problems with Concurrent execution

1. Lost Update Problem
2. Temporary Update (or Dirty Read) Problem
3. Incorrect Summary Problem
4. **Unrepeatable Read**

Unrepeatable Read

- Unrepeatable Read occurs, if transaction T1 reads an item twice and the item is changed by another transaction T2 between two reads hence T1 finds two different values on it's two reads.
- Example: If during train reservation, a user inquires about seat availability on several trains. When user decides on a particular train, the transaction reads the number of seats on that train a second time before completing the reservation.

Unrepeatable Read: Example

Train Reservation Database					
Source	Destination	Train Name	Start Time	Start Date	Number of Seats Reserved
Bangalore	Mysore	Tippu Express	15:00	25-3-2016	80
Bangalore	Mysore	Chamundi Express	18:15	25-3-2016	70

Transaction T1	Transaction T2
Read(TippuSeats)	Read(TippuSeats) TippuSeats=TippuSeats+4 Write(TippuSeats)
Read(TippuSeats)	

First time when Transaction T1 reads, TippuSeats value will be 80 but second time when the same Transaction T1 Reads, TippuSeats value will be 84. T1 is seeing two different values for same item TippuSeats

Why Concurrency Control is needed ?

To avoid following Problems

1. Lost Update Problem
2. Temporary Update (or Dirty Read) Problem
3. Incorrect Summary Problem
4. Unrepeatable Read

Next we will Understand Schedules

Schedule is a sequence of operations of various transactions

Transaction Schedules

Example to Understand Transaction Schedules

Train Reservation Database					
Source	Destination	Train Name	Start Time	Start Date	Number of Seats Reserved
Bangalore	Mysore	Tippu Express	15:00	25-3-2016	80
Bangalore	Mysore	Chamundi Express	18:15	25-3-2016	70

Consider TWO transactions **T1** and **T2**

- **T1:** Cancel FIVE seats on Tippu express

Read(TippuSeats)

TippuSeats=TippuSeats-5

Write(TippuSeats)

- **T2:** Reserve FOUR seats on Chamundi express

Read(ChamundiSeats)

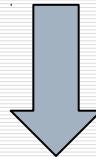
ChamundiSeats=ChamundiSeats+4

Write(ChamundiSeats)

Transaction Schedules

Schedule 1: T1, T2

```
T1:Read(TippuSeats)
T1:TippuSeats=TippuSeats-5
T1:Write(TippuSeats)
T2:Read(ChamundiSeats)
T2:ChamundiSeats=ChamundiSeats+4
T2:Write(ChamundiSeats)
```



Shorthand Notation

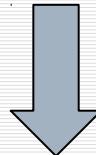
Schedule 1: T1, T2

```
R1, W1, R2, W2
```

Transaction Schedules

Schedule 1: T1, T2

```
T1:Read(TippuSeats)
T1:TippuSeats=TippuSeats-5
T1:Write(TippuSeats)
T2:Read(ChamundiSeats)
T2:ChamundiSeats=ChamundiSeats+4
T2:Write(ChamundiSeats)
```



Shorthand Notation

Schedule 1: T1, T2

```
R1, W1, R2, W2
```

Schedule 2: T2, T1

```
T2:Read(ChamundiSeats)
T2:ChamundiSeats=ChamundiSeats+4
T2:Write(ChamundiSeats)
T1:Read(TippuSeats)
T1:TippuSeats=TippuSeats-5
T1:Write(TippuSeats)
```



Shorthand Notation

Schedule 2: T2, T1

```
R2, W2, R1, W1
```

Different Possible Schedules for given set of transactions

Consider Two Transactions

T1: Cancel FIVE seats on Tippu express (R1, W1)

T2: Reserve four seats on Chamundi express (R2,W2)

Different possible schedules for the above two transactions are as follows:

Schedule 1	R1, W1, R2,W2
Schedule 2	R2,W2, R1, W1
Schedule 3	R1, R2, W1,W2
Schedule 4	R2, R1, W2, W1

Transaction Schedules

Consider Two Transactions

T1: Cancel FIVE seats on Tippu express (R1, W1)

T2: Reserve four seats on Chamundi express (R2,W2)

Different possible schedules for the above two transactions are as follows:

Schedule 1	R1, W1, R2,W2
Schedule 2	R2,W2, R1, W1
Schedule 3	R1, R2, W1,,W2
Schedule 4	R2, R1, W2, W1

Following are **not feasible schedules** because they do not preserve the order of operations of the individual Transactions

W1, R1, W2, R2

W2, R2, W1, R1

Transaction Schedules

Consider Two Transactions

T1: Cancel FIVE seats on Tippu express (R1, W1)

T2: Reserve four seats on Chamundi express (R2,W2)

Different possible schedules for the above two transactions are as follows:

Schedule 1	R1, W1, R2,W2
Schedule 2	R2,W2, R1, W1
Schedule 3	R1, R2, W1,,W2
Schedule 4	R2, R1, W2, W1

Following are **not feasible schedules** because they do not preserve the order of operations of the individual Transactions

W1, R1, W2, R2

W2, R2, W1, R1

W1, W2, R1, R2

W2, W1, R2, R1

Serial Schedule vs Interleaved Schedule

Consider Two Transactions

T1: Cancel FIVE seats on Tippu express

T2: Reserve four seats on Chamundi express

Different possible schedules for the above two transactions are as follows

Schedule 1	R1, W1, R2, W2	}	Schedule 1 & 2 are Serial Schedule
Schedule 2	R2, W2, R1, W1		
Schedule 3	R1, R2, W1, W2	}	Schedule 3 & 4 are Interleaved or Parallel Schedule
Schedule 4	R2, R1, W2, W1		

Note:

Serial Schedule: A schedule in which the different transactions are NOT interleaved (i.e., transactions are executed from start to finish one-by-one)

-The result of **Interleaved Schedule** should be equivalent Some serial schedule

Question

- What will be the total number of **serial schedules** that can occur if there are m Transactions i.e., T₁, T₂,...,T_m

Question

- What will be the total number of **serial schedules** that can occur if there are m Transactions i.e., T1, T2,...,Tm
- Answer: **m!** ($m * (m-1) * (m-2)....*1$)

Example:

- Two Transactions T1 & T2

Two (2!) Serial Schedules:

T1, T2

T2,T1

- Three Transactions T1 , T2 & T3

Six (3!) Serial Schedules :

T1,T2,T3

T1,T3,T2

T2,T1,T3

T2,T3,T1

T3,T1,T2

T3,T2,T1

Scheduling Definitions

- A **serial schedule** is one that does not interleave the actions of different transactions
- A and B are **equivalent schedules** if, ***for any database state***, the effect on DB of executing A **is identical to** the effect of executing B
- A **serializable schedule** is a schedule that is equivalent to ***some*** serial execution of the transactions.

The word “***some***” makes this definition powerful and tricky!

Problem to Solve

To check whether given schedule is **serializable or not**

Example- consider two TXNs:

T1: START TRANSACTION
UPDATE Accounts
SET Amt = Amt + 100
WHERE Name = 'A'

UPDATE Accounts
SET Amt = Amt - 100
WHERE Name = 'B'
COMMIT

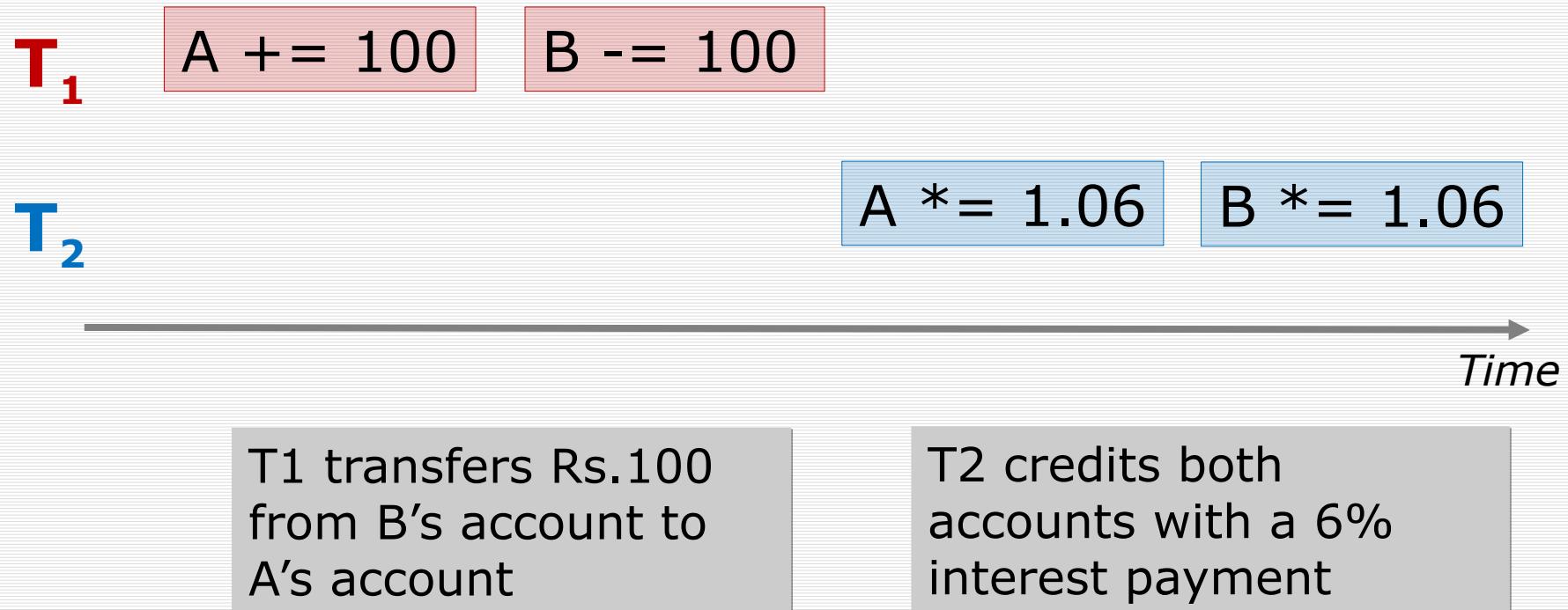
T2: START TRANSACTION
UPDATE Accounts
SET Amt = Amt * 1.06
COMMIT

T1 transfers Rs.100/-
from B's account to A's
account

T2 credits both accounts
with a 6% interest
payment

Example- consider two Transactions (T1 and T2):

We can look at the transactions in a timeline view- serial execution:



Example- consider two Transactions (T1 and T2):

The transactions could occur in either order... DBMS allows!

T₁

A += 100

B -= 100

T₂

A *= 1.06

B *= 1.06

Time

T2 credits both accounts
with a 6% interest
payment

T1 transfers Rs.100/-
from B's account to A's
account

Starting Balance

A	B
Rs.50	Rs.200

Serial schedule T_1, T_2 :

T_1

A += 100

B -= 100

T_2

A *= 1.06

B *= 1.06

Time

Serial schedule T_2, T_1 :

Result of Executing T_1, T_2

A	B
Rs.159	Rs.106

Starting Balance

A	B
Rs.50	Rs.200

T_1

A += 100

B -= 100

Result of Executing T_2, T_1

T_2

A *= 1.06

B *= 1.06

A	B
Rs.153	Rs.112

Problem to Solve

Check whether the following schedule i.e., ($A+=100$, $A^*=1.06$, $B-=100$, $B^*=1.06$) is Serializable ?

Starting Balance

A	B
Rs.50	Rs.200

Serial schedule results:

A	B
T_1, T_2	Rs.159
T_2, T_1	Rs.153

Transaction Schedule

T₁ A += 100

B -= 100

T₂ A *= 1.06

B *= 1.06

Note: To check whether given schedule is serializable or not, we should check whether the given interleaved schedule result is equivalent to result of some serial schedule

Serializable, Yes

Starting Balance

A	B
Rs.50	Rs.200

Serial schedules:

A	B
T_1, T_2	Rs.159 Rs.106
T_2, T_1	Rs.153 Rs.112

Transaction Schedule

T_1

A += 100

B -= 100

T_2

A *= 1.06

B *= 1.06

A B

Rs.159 Rs.106

Same as a serial schedule **for all possible values of A, B = serializable**

Problem to Solve

Check whether the following schedule i.e., ($A+=100$, $A^*=1.06$, $B-=100$, $B^*=1.06$) is Serializable ?

Starting Balance

A	B
Rs.50	Rs.200

Serial schedule results:

A	B
T_1, T_2	Rs.159
T_2, T_1	Rs.153

Transaction Schedule

T₁ A += 100

B -= 100

T₂

A *= 1.06 B *= 1.06

Note: To check whether given schedule is serializable or not, we should check whether the given interleaved schedule result is equivalent to result of some serial schedule

Serializable, No

Starting Balance

A	B
Rs.50	Rs.200

Serial schedules:

A	B
T_1, T_2	Rs.159 Rs.106
T_2, T_1	Rs.153 Rs.112

Transaction Schedule

T₁

A += 100

T₂

A *= 1.06

B *= 1.06

B -= 100

A	B
Rs.159	Rs.112

Not equivalent to
any serializable
schedule = **not
serializable**

Complete Schedule

- Complete Schedule: A schedule that **contains either a commit or an abort** action for EACH transaction

Complete Schedule		Complete Schedule		Complete (Serial) Schedule	
T1	T2	T1	T2	T1	T2
R(A)		R(A)		R(A)	
W(A)	R(B)	W(A)		W(A)	
Commit	Abort	Commit	R(B)	Commit	
			W(B)	R(B)	
			Abort	W(B)	
				Abort	

Note: consequently, a complete schedule will not contain any active transactions at the end of the schedule

Next we will Understand

Conflicting operations in Schedules

Conflicting Operations in Schedules

Two operations in schedule are said to CONFLICT if they satisfy **all three** of the following **conditions**

1. If two operations belong to different transactions
2. If two operations access same data item
3. Among two operations at least one operation is write

Conflicting Operations in Schedules

Two operations in schedule are said to **CONFLICT** if they satisfy all three of the following conditions

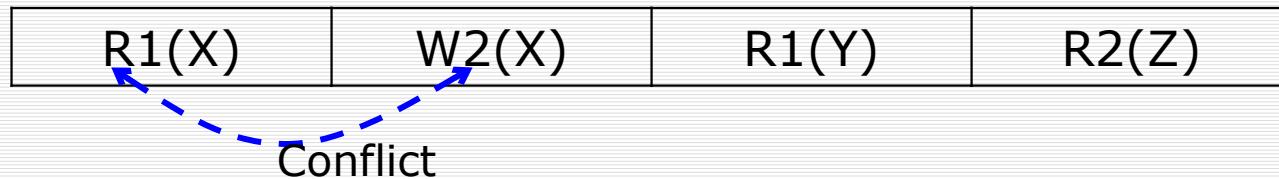
1. If two operations belong to **different transactions**
2. If two operations access same data item
3. Among two operations at least one operation is write

Example: Consider two transactions

T1 with operations R1(X) and R1(Y)

T2 with operations W2(X) and R2(Z)

Consider the schedule as: R1(X), W2(X), R1(Y), R2(Z)



Conflicting Operations in Schedules

Two operations in schedule are said to **CONFLICT** if they satisfy all three of the following conditions

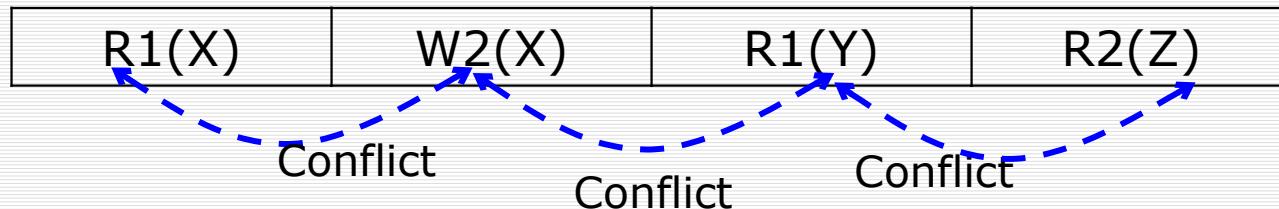
1. If two operations belong to **different transactions**
2. If two operations access same data item
3. Among two operations at least one operation is write

Example: Consider two transactions

T1 with operations R1(X) and R1(Y)

T2 with operations W2(X) and R2(Z)

Consider the schedule as: R1(X), W2(X), R1(Y), R2(Z)



Conflicting Operations in Schedules

Two operations in schedule are said to **CONFLICT** if they satisfy all three of the following conditions

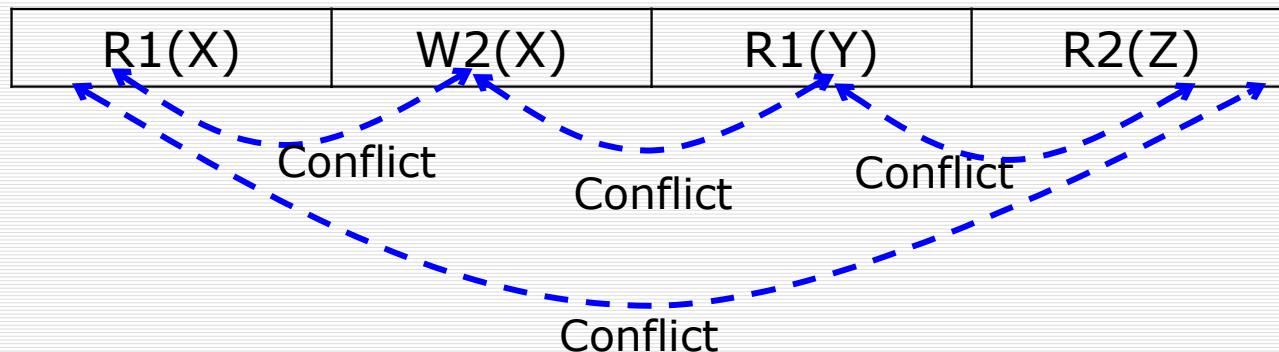
1. If two operations belong to **different transactions**
2. If two operations access same data item
3. Among two operations at least one operation is write

Example: Consider two transactions

T1 with operations R1(X) and R1(Y)

T2 with operations W2(X) and R2(Z)

Consider the schedule as: R1(X), W2(X), R1(Y), R2(Z)



Conflicting Operations in Schedules

Two operations in schedule are said to CONFLICT if they satisfy all three of the following conditions

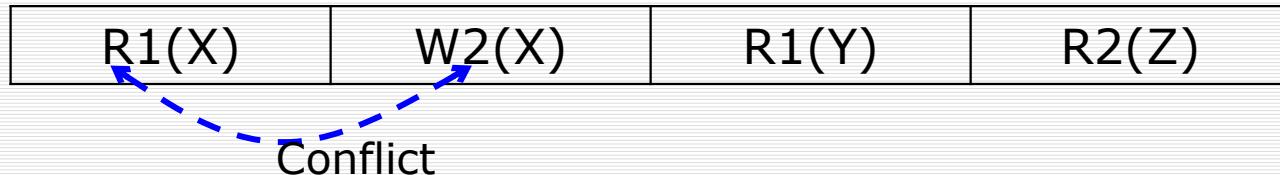
1. If two operations belong to different transactions
2. If two operations **access same data item**
3. Among two operations at least one operation is write

Example: Consider two transactions

T1 with operations R1(X) and R1(Y)

T2 with operations W2(X) and R2(Z)

Consider the schedule as: R1(X), W2(X), R1(Y), R2(Z)



Conflicting Operations in Schedules

Two operations in schedule are said to CONFLICT if they satisfy all three of the following conditions

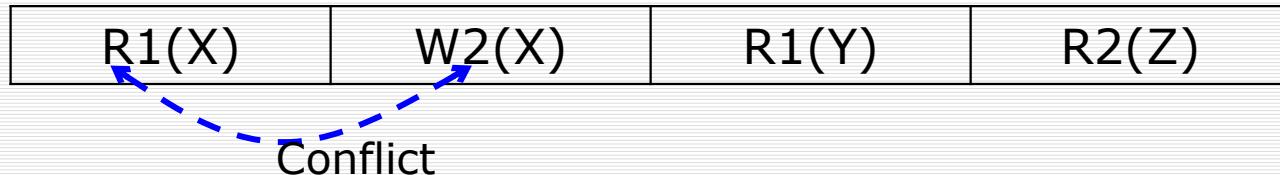
1. If two operations belong to different transactions
2. If two operations access same data item
3. Among two operations **at least one operation is write**

Example: Consider two transactions

T1 with operations R1(X) and R1(Y)

T2 with operations W2(X) and R2(Z)

Consider the schedule as: R1(X), W2(X), R1(Y), R2(Z)



Conflicting Operations in Schedules

Two operations in schedule are said to CONFLICT if they satisfy **all three of the following conditions**

1. If two operations belong to different transactions
2. If two operations access same data item
3. Among two operations at least one operation is write

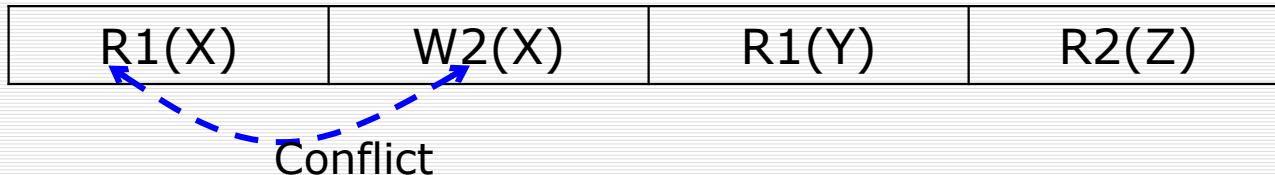
Example: Consider two transactions

T1 with operations R1(X) and R1(Y)

T2 with operations W2(X) and R2(Z)

Consider the schedule as: R1(X), W2(X), R1(Y), R2(Z)

Now only CONFLICTing operation in the given schedule which satisfy all three conditions is **(R1(X), W2(X))**



Conflicting Operations in Schedules

Two operations in schedule are said to CONFLICT if they satisfy all three of the following conditions

1. If two operations belong to different transactions
2. If two operations access same data item
3. Among two operations at least one operation is write

Question:

Check which of the following schedules are having Conflicting operations

Schedule 1	R1(X), W2(X), W3(X)	
Schedule 2	R1(X), R2(X), R3(X)	
Schedule 3	R1(X), W2(Y), R3(X)	

Conflicting Operations in Schedules

Two operations in schedule are said to CONFLICT if they satisfy all three of the following conditions

1. If two operations belong to different transactions
2. If two operations access same data item
3. Among two operations at least one operation is write

Question:

Check which of the following schedules are having Conflicting operations

Schedule 1	R1(X), W2(X), W3(X)	YES, because (R1(X), W2(X))
Schedule 2	R1(X), R2(X), R3(X)	NO
Schedule 3	R1(X), W2(Y), R3(X)	NO

Serializability

What is Serializability ? – “Correctness Measure” of some Schedule

- Why is it useful? It answers the question: “Will an interleaved schedule execute correctly”
- i.e., a Serializable schedule will execute as correctly as serial schedule ... but in an interleaved manner!

Example: Consider two transactions

T1 with operations

Read(X)

X=X-5

Write(X)

Read(Y)

Y=Y+5

Write(Y))

T2 with operations

Read(X)

X=X-4

Write(X)

Serializability

What is Serializability ? – “Correctness Measure” of some Schedule

- Why is it useful? It answers the question: “Will an interleaved schedule execute correctly”
- i.e., a Serializable schedule will execute as correctly as serial schedule ... but in an interleaved manner!

Example: Consider two transactions

T1 with operations (Read(X); X=X-5,Write(X); Read(Y); Y=Y+5; Write(Y))

T2 with operations (Read(X), X=X-4,Write(X));

For this two Transactions (T1 & T2) two possible serial schedules are:

Serial schedule (T1, T2)

Transaction T1	Transaction T2
Read(X) X=X-5 Write(X) Read(Y) Y=Y+5 Write(Y)	Read(X) X=X+4 Write(X)

Order of Execution Of operations



Serial schedule (T2, T1)

Transaction T1	Transaction T2
	Read(X) X=X+4 Write(X) Read(X) X=X-5 Write(X) Read(Y) Y=Y+5 Write(Y)

Order of Execution Of operations



Serializability

Order of Execution Of operations

Serial schedule (T1, T2)	
Transaction T1	Transaction T2
Read(X)	
X=X-5	
Write(X)	
Read(Y)	
Y=Y+5	
Write(Y)	
	Read(X)
	X=X+4
	Write(X)

Serial schedule (T2, T1)

Serial schedule (T2, T1)	
Transaction T1	Transaction T2
	Read(X)
	X=X+4
	Write(X)
Read(X)	
X=X-5	
Write(X)	
Read(Y)	
Y=Y+5	
Write(Y)	

Order of Execution Of operations

Non-Serial schedule which is **Serializable** because it is equivalent to serial schedule (T1,T2)

Order of Execution Of operations

Non-Serial Schedule	
Transaction T1	Transaction T2
Read(X)	
X=X-5	
Write(X)	
	Read(X)
	X=X+4
	Write(X)
Read(Y)	
Y=Y+5	
Write(Y)	

Serializability

Order of Execution Of operations

Serial schedule (T1, T2)	
Transaction T1	Transaction T2
Read(X)	
X=X-5	
Write(X)	
Read(Y)	
Y=Y+5	
Write(Y)	
	Read(X)
	X=X+4
	Write(X)

Serial schedule (T2, T1)

Transaction T1	Transaction T2
	Read(X)
	X=X+4
	Write(X)
Read(X)	
X=X-5	
Write(X)	
Read(Y)	
Y=Y+5	
Write(Y)	

Order of Execution Of operations

Non-Serial schedule which is **serializable** because it is equivalent to serial schedule (T1,T2)

Transaction T1	Transaction T2
Read(X)	
X=X-5	
Write(X)	
	Read(X)
	X=X+4
	Write(X)
Read(Y)	
Y=Y+5	
Write(Y)	

Non-Serial schedule, but it is **not serializable** because it is not equivalent to any serial schedule

Transaction T1	Transaction T2
Read(X)	
X=X-5	
	Read(X)
	X=X+4
Write(X)	
Read(Y)	
Y=Y+5	
Write(Y)	

Order of Execution Of operations

Characterizing Schedules based on Serializability

Based on Serializability

Characterize which schedules are correct when concurrent transactions are executing.

1. Conflict Serializable Schedule
2. View Serializable Schedule

Conflict Serializability Schedule

- A schedule **S** is said to be conflict serializable if it is **conflict equivalent** to some serial schedule **S'**.

What is Conflict Equivalent ?

- Two schedules are said to be conflict equivalent if the order of any two **conflicting operations** is the same in both schedules.

Conflict Serializability Schedule

- A schedule **S** is said to be conflict serializable if it is **conflict equivalent** to some serial schedule **S'**.

What is Conflict Equivalent ?

- Two schedules are said to be conflict equivalent if the order of any two **conflicting operations** is the same in both schedules.

What are Conflicting Operations ?

Transaction T1	Transaction T2
Write(X)	
	Read(X)

Transaction T1	Transaction T2
Read(X)	
	Write(X)

Transaction T1	Transaction T2
Write(X)	
	Write(X)

Check whether the given Two schedules are conflict equivalent ?

- Definition: Two schedules are said to be conflict equivalent if the order of any two **conflicting operations** is the same in both schedules.

Schedule S1

T1	T2
Read(A)	
Read(B)	
	Write(A)
	Write(B)

Schedule S2

T1	T2
Read(A)	
	Write(A)
Read(B)	
	Write(B)

Schedule S1: R1(A), R1(B), W2(A), W2(B)

Schedule S2: R1(A),W2(A), R1(B),W2(B)

Check whether the given Two schedules are conflict equivalent ?

Schedule S1

T1	T2
Read(A)	
Read(B)	
	Write(A)
	Write(B)

Schedule S2

T1	T2
Read(A)	
	Write(A)
Read(B)	
	Write(B)

Answer: Schedule S1 and S2 are conflict equivalent

Schedule 1	R1(A), R1(B), W2(A),W2(B)	Conflict Operations R1(A) and W2(A) R1(B) and W2(B)	Schedule 1 and 2 are conflict equivalent because the order of conflict operations are same
Schedule 2	R1(A),W2(A), R1(B),W2(B)	Conflict Operations R1(A) and W2(A) R1(B) and W2(B)	

Check whether the given Two schedules are conflict equivalent ?

- Definition: Two schedules are said to be conflict equivalent if the order of any two **conflicting operations** is the same in both schedules.

Schedule S1

T1	T2
	Write(A)
	Write(B)
Read(A)	
Read(B)	

Schedule S2

T1	T2
Read(A)	
	Write(A)
Read(B)	
	Write(B)

Schedule S1: W2(A), W2(B), R1(A), R1(B)

Schedule S2: R1(A),W2(A), R1(B),W2(B)

Check whether the given Two schedules are conflict equivalent ?

Schedule S1

T1	T2
	Write(A)
	Write(B)
Read(A)	
Read(B)	

Schedule S2

T1	T2
Read(A)	
	Write(A)
Read(B)	
	Write(B)

Answer: Schedule S1 and S2 **are not** conflict equivalent

Schedule 1	W2(A), W2(B), R1(A), R1(B)	Conflict Operations W2(A) and R1(A) W2(B) and R1(B)	Schedule 1 and 2 are not conflict equivalent because the order of conflict operations are different in the schedules
Schedule 2	R1(A), W2(A), R1(B), W2(B)	Conflict Operations R1(A) and W2(A) R1(B) and W2(B)	

Problem to Solve

Check whether the following two schedules are conflict equivalent ?

Schedule 1

T1	T2
R(A) W(A)	
	R(A) W(A)

Schedule 2

T1	T2
	R(A) W(A)
R(A) W(A)	

T1	T2
R(A)	
	R(A)
W(A)	
	W(A)

Schedule 1	R1(A), W1(A), R2(A), W2(A)	Conflict Operations R1(A) and W2(A) W1(A) and R2(A) W1(A) and W2(A)	Schedule 1 and 2 are not conflict equivalent because the order of conflict operations are not same
Schedule 2	R2(A), W2(A), R1(A), W1(A)	Conflict Operations W2(A) and R1(A) R2(A) and W1(A) W2(A) and W1(A)	

Note: Two schedules are said to be conflict equivalent if the order of any two conflicting operations is the same in both schedules.

Problem to Solve

Check whether the following two schedules are conflict equivalent ?

Schedule 1

T1	T2
R(A) W(A)	R(A) W(A)

Schedule 3

T1	T2
R(A) W(A)	R(A) W(A)

Schedule 1	R1(A), W1(A), R2(A), W2(A)	Conflict Operations
Schedule 3	R1(A), R1(A), W2(A), W1(A)	Conflict Operations

Note: Two schedules are said to be conflict equivalent if the order of any two conflicting operations is the same in both schedules.

Testing for conflict-serializability of a schedule

- Looks at only `read_Item (X)` and `write_Item (X)` operations
- Constructs a **precedence graph** (serialization graph) - a graph with directed edges
 - 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
- The schedule is serializable if and only if the precedence graph has no cycles

Definition of Conflict Serializability Schedule:

A schedule **S** is said to be conflict serializable if it is **conflict equivalent** to some serial schedule **S'**.

Algorithm Testing for conflict-serializability

Algorithm:

1. For each transaction T_i participating in schedule S , create a node labeled T_i in the precedence graph.
2. For each case in S where T_j executes a **read_item(X)** after T_i executes a **write_item(X)**, create an edge $(T_i \rightarrow T_j)$ in the precedence graph.
3. For each case in S where T_j executes a **write_item(X)** after T_i executes a **read_item(X)**, create an edge $(T_i \rightarrow T_j)$ in the precedence graph.
4. For each case in S where T_j executes a **write_item(X)** after T_i executes a **write_item(X)**, create an edge $(T_i \rightarrow T_j)$ in the precedence graph.
5. The schedule S is **serializable** if and only if the precedence graph has **no cycles**.

Problem To Solve

- Construct precedence graph for the following schedule
Schedule S1

T1	T2	T3
		Read(A)
	Read(A)	
		Write(A)
Read(A)		
Write(A)		

Solution

- Construct precedence graph for the following schedule

Schedule S1

T1	T2	T3
		Read(A)
	Read(A)	
		Write(A)
Read(A)		
Write(A)		

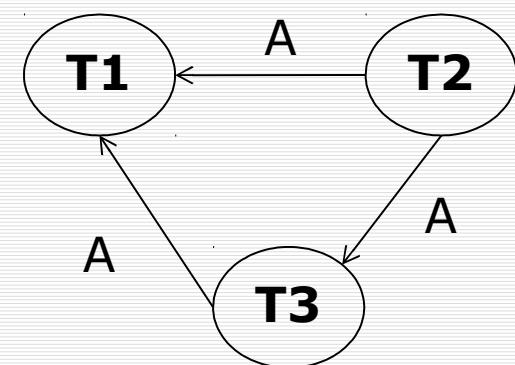
Given: Three Transactions

T1 with operations Read(A), Write(A)

T2 with operations Read(A)

T3 with operations Read(A), Write(A)

Schedule S1: R3(A), R2(A), W3(A), R1(A), W1(A)



Problem To Solve

- Construct precedence graph for the following schedule
Schedule S1

T1	T2	T3
Read(A)		
	Write(A)	
Write(A)		
		Write(A)

Problem To Solve

- Construct precedence graph for the following schedule
Schedule S1

T1	T2	T3
Read(A)		
	Write(A)	
Write(A)		
		Write(A)

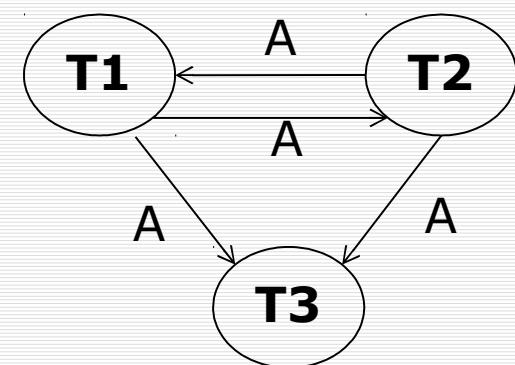
Given: Three Transactions

T1 with operations Read(A), Write(A)

T2 with operations Read(A)

T3 with operations Write(A)

Schedule S1: R1(A), W2(A), W1(A), W3(A)



Problem To Solve

- What is the Equivalent serial schedule for non-serial schedule
S1:R3(A), R2(A),W3(A), R1(A), W1(A)

Problem To Solve

- What is the Equivalent serial schedule for non-serial schedule S1:R3(A), R2(A), W3(A), R1(A), W1(A)

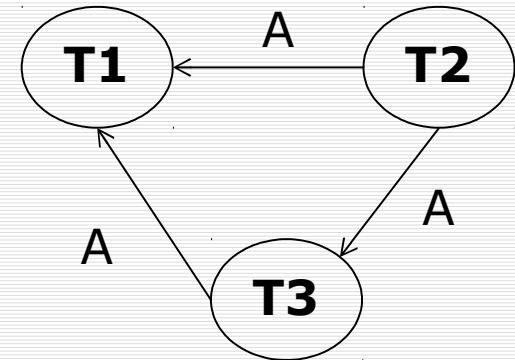
Given: Three Transactions

T1 with operations Read(A), Write(A)

T2 with operations Read(A)

T3 with operations Read(A), Write(A)

Schedule S1: R3(A), R2(A), W3(A), R1(A), W1(A)



Problem To Solve

- What is the Equivalent serial schedule for non-serial schedule S1:R3(A), R2(A), W3(A), R1(A), W1(A)

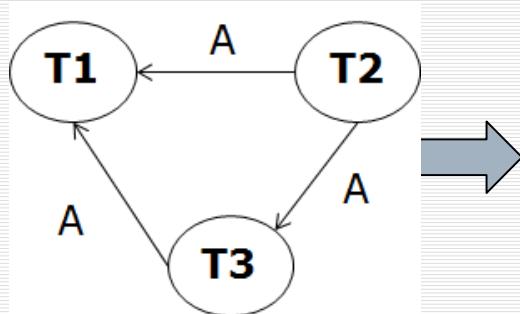
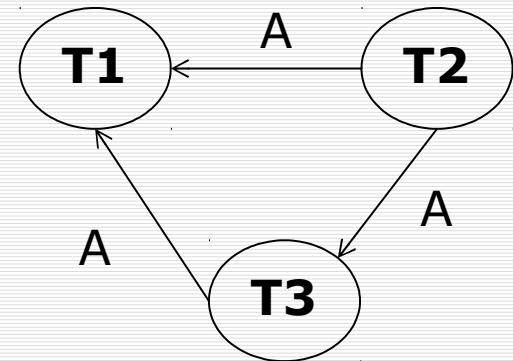
Given: Three Transactions

T1 with operations Read(A), Write(A)

T2 with operations Read(A)

T3 with operations Read(A), Write(A)

Schedule S1: R3(A), R2(A), W3(A), R1(A), W1(A)



Consider first T2
Because indegree is one

Problem To Solve

- What is the Equivalent serial schedule for non-serial schedule S1:R3(A), R2(A), W3(A), R1(A), W1(A)

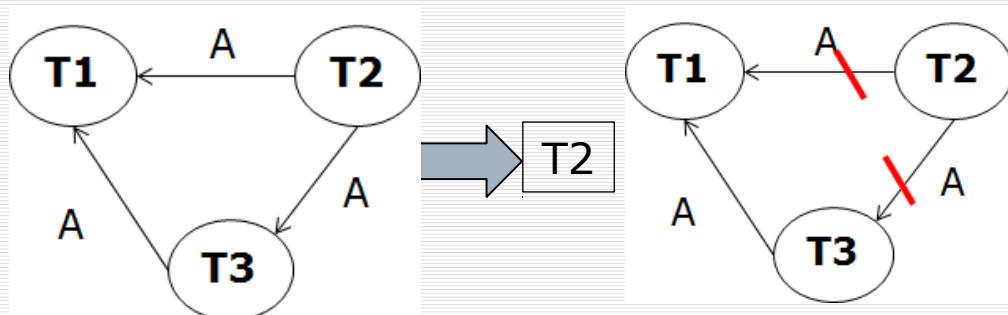
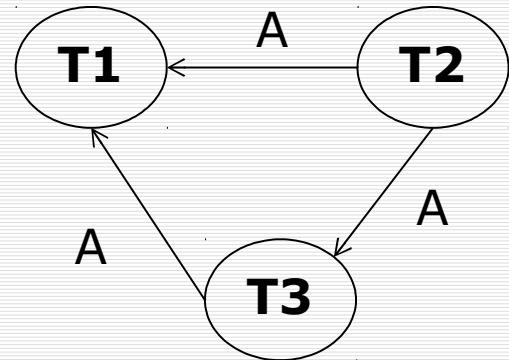
Given: Three Transactions

T1 with operations Read(A), Write(A)

T2 with operations Read(A)

T3 with operations Read(A), Write(A)

Schedule S1: R3(A), R2(A), W3(A), R1(A), W1(A)



Consider first T2
Because indegree is
one

Problem To Solve

- What is the Equivalent serial schedule for non-serial schedule S1:R3(A), R2(A), W3(A), R1(A), W1(A)

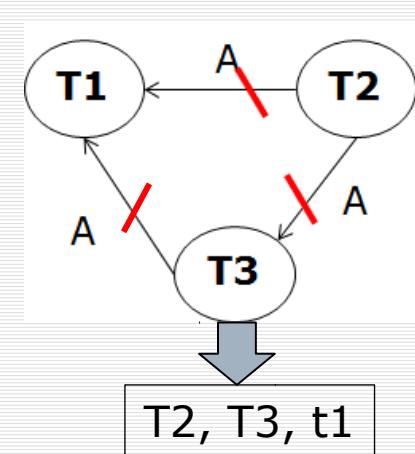
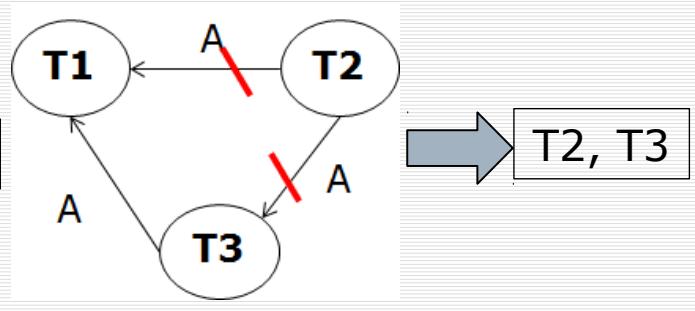
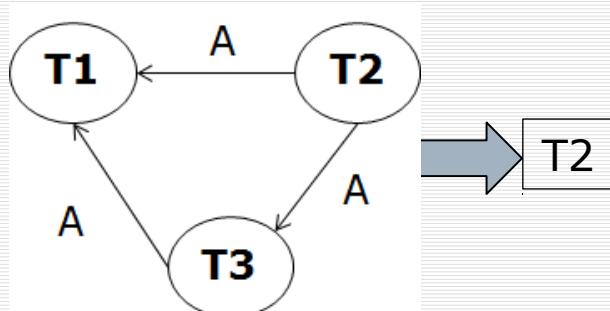
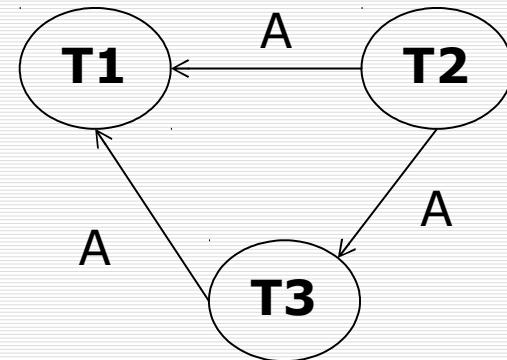
Given: Three Transactions

T1 with operations Read(A), Write(A)

T2 with operations Read(A)

T3 with operations Read(A), Write(A)

Schedule S1: R3(A), R2(A), W3(A), R1(A), W1(A)



Consider first T2
Because indegree is one

Solution

- What is the Equivalent serial schedule for non-serial schedule
S1:R3(A), R2(A),W3(A), R1(A), W1(A)

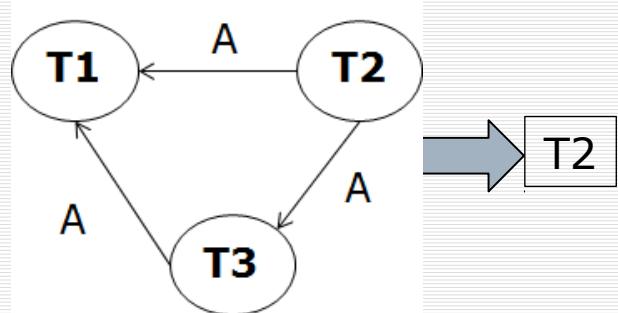
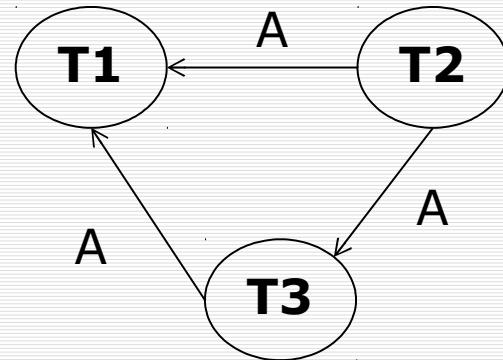
Given: Three Transactions

T1 with operations Read(A), Write(A)

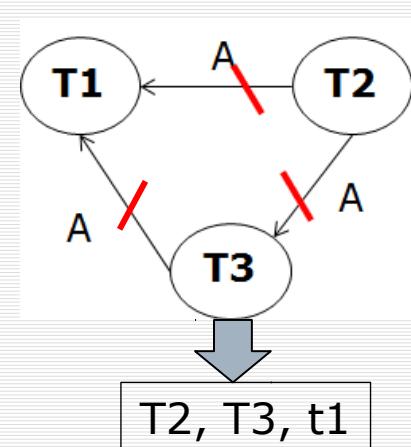
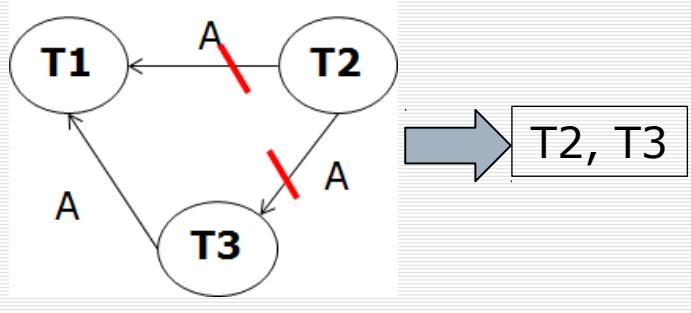
T2 with operations Read(A)

T3 with operations Read(A), Write(A)

Schedule S1: R3(A), R2(A),W3(A), R1(A), W1(A)



Consider first T2
Because indegree is
one



Equivalent Serial Schedule is: T2, T3, T1
R2(A), R3(A),W3(A), R1(A), W1(A)

Problem to Solve

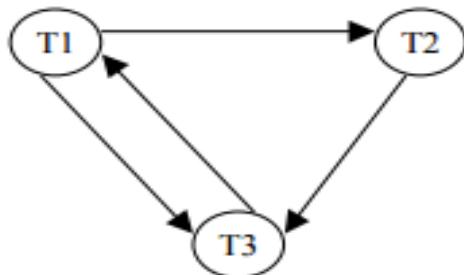
Which of the following schedules is (conflict) serializable? For each serializable schedule, determine the equivalent serial schedules.

- (a) r1 (X); r3 (X); w1(X); r2(X); w3(X)
- (b) r1 (X); r3 (X); w3(X); w1(X); r2(X)
- (c) r3 (X); r2 (X); w3(X); r1(X); w1(X)
- (d) r3 (X); r2 (X); r1(X); w3(X); w1(X)

Solution

- a). $r1(X); r3(X); w1(X); r2(X); w3(X);$

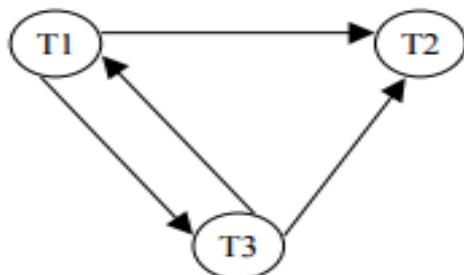
The serialization graph is:



Not serializable.

- b). $r1(X); r3(X); w3(X); w1(X); r2(X);$

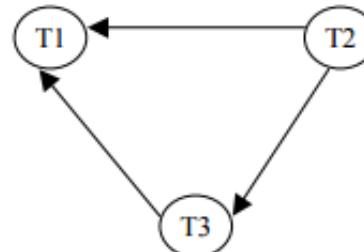
The serialization graph is:



Not serializable.

- c). $r3(X); r2(X); w3(X); r1(X); w1(X);$

The serialization graph is:

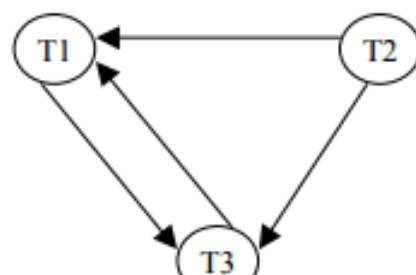


Serializable.

The equivalent serial schedule is: $r2(X); r3(X); w3(X); r1(X); w1(X);$

- d). $r3(X); r2(X); r1(X); w3(X); w1(X);$

The serialization graph is:



Not serializable.

Problem to Solve

Consider the three transactions T1, T2, and T3, and the schedules S1 and S2 given below. Draw the serializability (precedence) graphs for S1 and S2 and state whether each schedule is serializable or not. If a schedule is serializable, write down the equivalent serial schedule(s).

T1: r1(x); r1(z); w1(x)

T2: r2(z); r2(y); w2(z); w2(y)

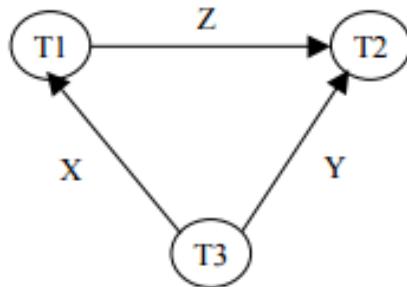
T3: r3(x); r3(y); w3(y)

S1: r1(x); r2(z); r1(z); r3(x); r3(y); w1(x); w3(y); r2(y); w2(z); w2(y)

S2: r1(x); r2(z); r3(x); r1(z); r2(y); r3(y); w1(x); w2(z); w3(y); w2(y)

Solution

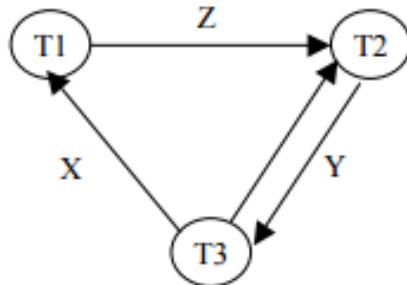
- 1). The serialization graph for S1 is:



S1 is serializable. The equivalent serial schedule is: (T3, T1, T2)

$r3(X); r3(Y); w3(Y); r1(X); r1(Z); w1(X); r2(Z); r2(Y); w2(Z); w2(Y);$

- 2). The serialization graph for S2 is:



S2 is not serializable.

What you have learned until now in Unit4:Transactions

Serial Schedule

Serializable Schedule

Conflict Serializable Schedule (CS)

- Algorithm for testing CS and Converting to Serial Schedule

Characterizing Schedules based on Serializability

Based on Serializability

1. Conflict Serializable Schedule
2. **View Serializable Schedule**

View Serializability Schedule

- A Schedule is View Serializable if it is **view equivalent** to some **serial schedule**
- What is View equivalence ?

In **View Equivalence**, respective transactions in the two schedules **read** and **write** the **same data** values

- Same WR order: **If** in S1: $w_j(A) \rightarrow r_i(A)$
then in S2: $w_j(A) \rightarrow r_i(A)$
- First Read: **If** in S1: $r_i(A)$ **reads initial** DB value,
then in S2: $r_i(A)$ also reads initial DB value
- Last Write: **If** in S1: $w_i(A)$ does **last write** on A,
then in S2: $w_i(A)$ also does last write on A

Note: i,j are identifiers
of Transactions

View Serializability Schedule

- A Schedule is View Serializable if it is **view equivalent** to some **serial schedule**
- What is View equivalence ?

In **View Equivalence**, respective transactions in the two schedules **read** and **write** the **same data** values

- Same WR order: **If** in S1: $w_j(A) \rightarrow r_i(A)$
then in S2: $w_j(A) \rightarrow r_i(A)$
- First Read: **If** in S1: $r_i(A)$ **reads initial** DB value,
then in S2: $r_i(A)$ also reads initial DB value
- Last Write: **If** in S1: $w_i(A)$ does **last write** on A,
then in S2: $w_i(A)$ also does last write on A

Note: i,j are identifiers
of Transactions

Note: The premise behind view equivalence:

- “The view”: the read operations are said to see *the same view* in both schedules.
- Rule: As long as each read operation of a transaction reads the result of *the same write operation* in both schedules, the write operations of each transaction must produce the same results.

View Equivalence Schedules

View Equivalence, respective transactions in the two schedules **read** and **write** the **same data values**

- Initial Reads: **If** in S1: $ri(A)$ **reads initial** DB value,
then in S2: $ri(A)$ also reads initial DB value
- Same WR order: **If** in S1: $wj(A) \rightarrow ri(A)$
then in S2: $wj(A) \rightarrow ri(A)$
- Final Writes: **If** in S1: $wi(A)$ does **last write** on A,
then in S2: $wi(A)$ also does last write on A

Schedule S1

T1	T2	T3
	R(X)	
		R(X)
		W(X)
R(X)		
W(X)		

Schedule S2

T1	T2	T3
		R(X)
		R(X)
		W(X)
R(X)		
W(X)		

Schedule S2 is view equivalent to serial schedule S1

View Equivalence Schedules

View Equivalence, respective transactions in the two schedules **read** and **write** the **same data** values

- Initial Reads: **If** in S1: $ri(A)$ **reads initial** DB value,
then in S2: $ri(A)$ also reads initial DB value
- Same WR order: **If** in S1: $wj(A) \rightarrow ri(A)$
then in S2: $wj(A) \rightarrow ri(A)$
- Final Writes: **If** in S1: $wi(A)$ does **last write** on A,
then in S2: $wi(A)$ also does last write on A

Schedule S2 is view equivalent to serial schedule S1

T1	T2	T3
	R(X)	
		R(X)
		W(X)
R(X)		
W(X)		

Schedule S1

T1	T2	T3
		R(X)
	R(X)	
		W(X)
R(X)		
W(X)		

Schedule S2

Same Initial
Reads in S1 and S2

View Equivalence Schedules

View Equivalence, respective transactions in the two schedules **read** and **write** the **same data** values

- Initial Reads: **If** in S1: $ri(A)$ **reads initial** DB value,
then in S2: $ri(A)$ also reads initial DB value
- Same WR order: **If** in S1: $wj(A) \rightarrow ri(A)$
then in S2: $wj(A) \rightarrow ri(A)$
- Final Writes: **If** in S1: $wi(A)$ does **last write** on A,
then in S2: $wi(A)$ also does last write on A

Schedule S2 is view equivalent to serial schedule S1

T1	T2	T3
	R(X)	
		R(X)
		W(X)
R(X)		
W(X)		

Schedule S1

T1	T2	T3
		R(X)
	R(X)	
		W(X)
R(X)		
W(X)		

Schedule S2

Same Write
Reads orders in
S1 and S2

View Equivalence Schedules

View Equivalence, respective transactions in the two schedules **read** and **write** the **same data** values

- Initial Reads: **If** in S1: $ri(A)$ **reads initial** DB value,
then in S2: $ri(A)$ also reads initial DB value
- Same WR order: **If** in S1: $wj(A) \rightarrow ri(A)$
then in S2: $wj(A) \rightarrow ri(A)$
- Final Writes: **If** in S1: $wi(A)$ does **last write** on A,
then in S2: $wi(A)$ also does last write on A

Schedule S2 is view equivalent to serial schedule S1

T1	T2	T3
	R(X)	
		R(X)
		W(X)
R(X)		
W(X)		

Schedule S1

T1	T2	T3
		R(X)
	R(X)	
		W(X)
R(X)		
W(X)		

Schedule S2

Same Final
Writes in
S1 and S2

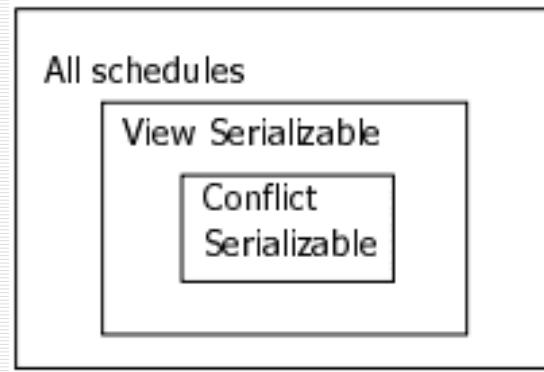
View Equivalence

View Serializability Summary:

- i. Same Transaction **Reads Data First.**
- ii. Same **WR Order** of actions.
- iii. Same Transaction **Writes Data Last.**

Summarizing Serializability Schedules

□ Venn diagram



Note:

Any Conflict serializable schedule is also View serializable schedule but not vice-versa

Note:

- There is an algorithm to test whether a schedule S is view serial schedule.
- However, the problem of testing view serializability has been shown to be NP-hard, meaning that finding an efficient polynomial time algorithm for this problem is highly unlikely.

Characterizing Schedules

Characterizing different schedules based on the following two properties:

A. Based on Serializability

- We shall ignore Commits and Aborts for this section
- Characterize which **schedules are correct** when concurrent transactions are executing.
 1. Conflict Serializable Schedule
 2. View Serializable Schedule

B. Based on Recoverability

- Commits and Aborts become important for this section
- Characterize which **schedules can be recovered** and how easily.
 3. Recoverable Schedule
 4. Cascadeless schedule
 5. Strict Schedules

Characterizing Schedules based on Recoverability

Based on Recoverability

Characterize which **schedules can be recovered** and how easily.

1. Recoverable Schedule
2. Cascadeless or Avoid Cascading Rollback schedule
3. Strict Schedules

Recoverable Schedule

Recoverability is a situation where we can recover database system to a consistent way after failure.

- Recoverable schedule: A **schedule S** is **recoverable** if no transaction **T** in **S** commits until all transactions **T'**, that have written an item that **T** reads, have committed.

Recoverable Schedule

Recoverability is a situation where we can recover database system to a consistent way after failure.

- Recoverable schedule: A **schedule S** is **recoverable** if no transaction **T** in **S** commits until all transactions **T'**, that have written an item that **T** reads, have committed.

Recoverable Schedule

T1	T2
R(X)	
X=X+10	
W(X)	
Commit	
	R(X)
	X=X-5
	W(X)
	Commit

Note: A committed transaction
should never be rolled back

Recoverable Schedule

Recoverability is a situation where we can recover database system to a consistent way after failure.

- Recoverable schedule: A **schedule S** is **recoverable** if no transaction **T** in **S** commits until all transactions **T'**, that have written an item that **T reads, have committed.**

Non-Recoverable Schedule

T1	T2
R(X)	
X=X+10	
W(X)	
	R(X)
	X=X-5
	W(X)
	Commit

Recoverable Schedule

T1	T2
R(X)	
X=X+10	
W(X)	
Commit	
	R(X)
	X=X-5
	W(X)
	Commit

Note: A committed transaction should never be rolled back

Recoverable Schedule

Recoverability is a situation where we can recover database system to a consistent way after failure.

- Recoverable schedule: A **schedule S** is **recoverable** if no transaction **T** in **S** commits until all transactions **T'**, that have written an item that **T** reads, have committed.

Question: Following is Recoverable Schedule ? Yes / No

T1	T2
R(X)	
W(X)	
	R(X)
W(X)	
	W(X)
	Commit
Abort	

Recoverable Schedule

Recoverability is a situation where we can recover database system to a consistent way after failure.

- Recoverable schedule: A **schedule S** is **recoverable** if no transaction **T** in **S** commits until all transactions **T'**, that have written an item that **T** reads, have committed.

Question: Following is Recoverable Schedule ? Yes / No

T1	T2
R(X)	
W(X)	Dirty Read R(X)
W(X)	
	W(X)
	Commit
Abort	

Answer: **NO**

Why NOT recoverable?

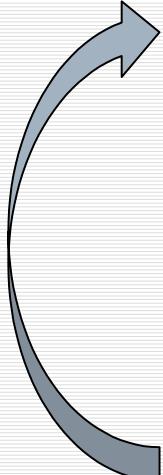
- Because **T2** made a **dirty read** and committed before **T1**

Recoverable Schedule

Recoverability is a situation where we can recover database system to a consistent way after failure.

- Recoverable schedule: A **schedule S** is **recoverable** if no transaction **T** in **S** commits until all transactions **T'**, that have written an item that **T** reads, have committed.

Question: Following is Recoverable Schedule ? Yes / No



T1	T2
R(X)	
W(X)	R(X)
W(X)	
	W(X)
	Commit
Abort	

Step a:
Rollback

Answer: NO

Why it not a recoverable schedule?

- Because **T2** made a **dirty read** and committed before **T1**

But why is the schedule Nonrecoverable ?

- Because when the **recovery manager** rolls back (step a) **T1** then A gets its initial value.
- But T2 has already utilized this wrong value and committed something to the DB
- The DB is consequently in an inconsistent state!

Problem to Solve on Recoverable Schedules

- Check whether the following schedule is recoverable schedule

2. R1(x), R2(x), R1(z), R3(x), R3(y), W1(x), W3(y), R2(y),
W2(z), W2(y), C1, C2, C3

Recoverable schedule: A schedule is recoverable if the following condition is satisfied:

- Tj should commit after Ti if Tj has read any data item written by Ti.

..... Wi(x) Rj(x) Ci Cj → This schedule is **recoverable**

..... Wi(x) Rj(x) Cj Ci → This schedule is **non-recoverable**

Problem to Solve on Recoverable Schedules

- Check whether the following schedule is recoverable schedule

2. R1(x), R2(x), R1(z), R3(x), R3(y), W1(x), W3(y), R2(y),
W2(z), W2(y), C1, C2, C3

Answer: Non-recoverable

Transaction T2 reads the data item R2(y) written by T3 w3(y)

Schedule is non-recoverable because transaction T2 commits C2 before T3 commits C3.

Recoverable schedule: A schedule is recoverable if the following condition is satisfied:

- Tj should commit after Ti if Tj has read any data item written by Ti.

..... Wi(x) Rj(x) Ci Cj -> This schedule is **recoverable**

..... Wi(x) Rj(x) Cj Ci -> This schedule is **non-recoverable**

Problem to Solve

Consider the following schedules:

S1: r1(X); w1(X); r1(Y); w1(Y); r2(X); w2(X); C2; C1;

S2: r1(X); w1(X); r2(X); r1(Y); w2(X); w1(Y); C1; C2;

Which of the following is true?

- (A) Both S1 and S2 are recoverable
- (B) S1 is recoverable, but S2 is not
- (C) S2 is recoverable, but S1 is not
- (D) Both schedules are non recoverable

Recoverable schedule: A schedule is recoverable if the following condition is satisfied:

- Tj should commit after Ti if Tj has read any data item written by Ti.

..... Wi(x) Rj(x) Ci Cj -> This schedule is **recoverable**

..... Wi(x) Rj(x) Cj Ci -> This schedule is **non-recoverable**

Problem to Solve

Consider the following schedules:

S1: r1(X); w1(X); r1(Y); w1(Y); r2(X); w2(X); C2; C1;

S2: r1(X); w1(X); r2(X); r1(Y); w2(X); w1(Y); C1; C2;

Which of the following is true?

- (A) Both S1 and S2 are recoverable
- (B) S1 is recoverable, but S2 is not
- (C) S2 is recoverable, but S1 is not
- (D) Both schedules are non recoverable

S1

T1	T2
R1(X)	
W1(X)	
R1(Y)	
W1(Y)	
	R2(X)
	W2(X)
	C2
C1	

S2

T1	T2
R1(X)	
W1(X)	
	R2(X)
R1(Y)	
	W2(X)
W1(Y)	
C1	
	C2

S1 Non-Recoverable

S2 Recoverable

Problem to Solve on Recoverable Schedules

- Check whether the following schedule is recoverable schedule

$S4: r1(X); r2(Z); r1(Z); r3(X); r3(Y); w1(X); w3(Y); r2(Y); w2(Z); w2(Y); c1; c2; c3;$

Recoverable schedule: A schedule is recoverable if the following condition is satisfied:

- T_j should commit after T_i if T_j has read any data item written by T_i .

..... $W_i(x)$ $R_j(x)$ C_i C_j -> This schedule is **recoverable**

..... $W_i(x)$ $R_j(x)$ C_j C_i -> This schedule is **non-recoverable**

Problem to Solve on Recoverable Schedules

- Check whether the following schedule is recoverable schedule

$S4: r1(X); r2(Z); r1(Z); r3(X); r3(Y); w1(X); w3(Y); r2(Y); w2(Z); w2(Y); c1; c2; c3;$

Answer: Non-recoverable

In $S4$, $T2$ reads item Y from $T3$ but $T2$ commits before $T3$ commits. So $S4$ is nonrecoverable.

Recoverable schedule: A schedule is recoverable if the following condition is satisfied:

- Tj should commit after Ti if Tj has read any data item written by Ti .

..... $Wi(x)$ $Rj(x)$ Ci Cj -> This schedule is **recoverable**

..... $Wi(x)$ $Rj(x)$ Cj Ci -> This schedule is **non-recoverable**

Characterizing Schedules based on Recoverability

Based on Recoverability

1. Recoverable Schedule
2. **Cascadeless or Avoid Cascading Rollback schedule**
3. Strict Schedules

Cascading Abort

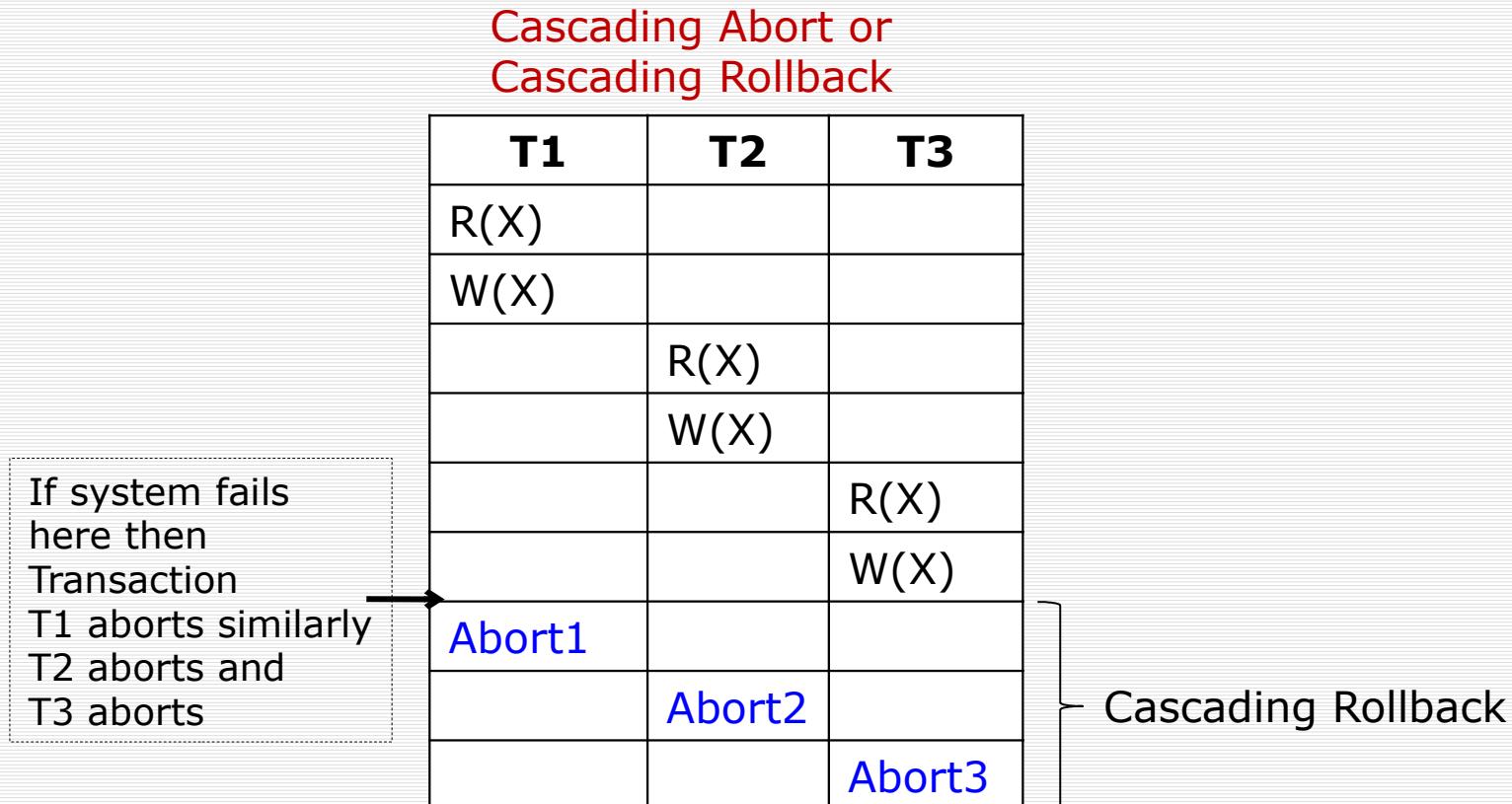
What is Cascading? When effect of one thing is migrated to other and followed by another.

Cascading Abort or
Cascading Rollback

T1	T2	T3
R(X)		
W(X)		
	R(X)	
	W(X)	
		R(X)
		W(X)

Cascading Abort

What is Cascading? When effect of one thing is migrated to other and followed by another.



Cascadeless Schedule

Cascadeless Schedule: Schedule that Avoids Cascading Rollbacks

- ❑ A schedule is said to be cascadeless schedule or avoid cascading rollback, if every transaction in the schedule reads only items that were written by committed transactions

Cascadeless Schedule

T1	T2	T3
R(X)		
W(X)		
Commit		
	R(X)	
	W(X)	
	Commit	
		R(X)
		W(X)
		Commit

Problem to Solve on Cascadeless Schedules

- Check whether the following schedule is cascadeless schedule

$S : r_1(x), r_3(y), r_3(x), w_1(x), c_1, w_2(y), r_2(x), w_3(y), c_2, c_3$

Cascadeless: A schedule is said to be cascadeless, if every transaction in the schedule reads only items that were written by committed (or aborted) transactions.

Cascadeless schedule: A schedule is cascadeless if the following condition is satisfied:

- Tj reads X only **after** Ti has written to X and committed (aborted or terminated).

.....Wi(x).....Ci(x)/Ai(x).....Rj(x) → This schedule is **Cascadeless**

Problem to Solve on Cascadeless Schedules

- Check whether the following schedule is cascadeless schedule

$S : r_1(x), r_3(y), r_3(x), w_1(x), c_1, w_2(y), r_2(x), w_3(y), c_2, c_3$

Answer: Yes, Cascadeless

T2 reads x after the last transaction that writes x i.e. T1 has committed.
Hence S is Cascadeless

Cascadeless: A schedule is said to be cascadeless, if every transaction in the schedule reads only items that were written by committed (or aborted) transactions.

Cascadeless schedule: A schedule is cascadeless if the following condition is satisfied:

- Tj reads X only **after** Ti has written to X and committed (aborted or terminated).

..... $W_i(x)$ $C_i(x)/A_i(x)$ $R_j(x)$ -> This schedule is **Cascadeless**

Characterizing Schedules based on Recoverability

Based on Recoverability

1. Recoverable Schedule
2. Cascadeless or Avoid Cascading Rollback schedule
3. **Strict Schedules**

Strict Schedule

- A restrictive type of schedule, called **strict schedule**, in which transaction neither read nor write an item X until the last transaction that wrote X has committed (terminated or aborted).

Formally, if it satisfies the following conditions:

- T_j **reads** a data item X **after** T_i has written to X and T_i is committed (aborted or terminated)
- T_j **writes** a data item X **after** T_i has written to X and T_i is committed (aborted or terminated)

Strict Schedule

T1	T2
R(X)	
W(X)	
Commit1	
	R(X)
	W(X)

Problem to Solve on Strict Schedules

- Check whether the following schedule is strict schedule

S3: $r1(X); r2(Z); r1(Z); r3(X); r3(Y); w1(X); c1; w3(Y); c3; r2(Y); w2(Z); w2(Y); c2;$

Strict schedule: A schedule is strict if it satisfies the following conditions:

1. Tj reads a data item X **after** Ti has written to X and Ti is committed (aborted or terminated)
2. Tj writes a data item X **after** Ti has written to X and Ti is committed (aborted or terminated)

..... $Wi(x)$ $Ci(x)$ $Rj(x)/Wj(x)$ -> This schedule is **Strict**
..... $Wi(x)$ $Rj(x)/Wj(x)$ $Ci(x)$ -> This schedule is **not strict**

Problem to Solve on Strict Schedules

- Check whether the following schedule is strict schedule

$S3: r1(X); r2(Z); r1(Z); r3(X); r3(Y); w1(X); c1; w3(Y); c3; r2(Y); w2(Z); w2(Y); c2;$

Answer: Strict

In S3, every transaction commits right after it writes some items. There is no write to or read from an item before the last transaction that wrote that item has committed. So S3 is **strict**.

Strict schedule: A schedule is strict if it satisfies the following conditions:

1. Tj reads a data item X **after** Ti has written to X and Ti is committed (aborted or terminated)
2. Tj writes a data item X **after** Ti has written to X and Ti is committed (aborted or terminated)

..... $Wi(x)$ $Ci(x)$ $Rj(x)/Wj(x)$ -> This schedule is **Strict**
..... $Wi(x)$ $Rj(x)/Wj(x)$ $Ci(x)$ -> This schedule is **not strict**

Problem to Solve on Strict Schedules

- Check whether the following schedule is strict schedule

$S5: r1(X); r2(Z); r3(X); rl(Z); r2(Y); r3(Y); wl(X); cl; w2(Z); w3(Y); w2(Y); c3; c2;$

Strict schedule: A schedule is strict if it satisfies the following conditions:

1. Tj reads a data item X **after** Ti has written to X and Ti is committed (aborted or terminated)
2. Tj writes a data item X **after** Ti has written to X and Ti is committed (aborted or terminated)

..... $Wi(x)$ $Ci(x)$ $Rj(x)/Wj(x)$ -> This schedule is **Strict**
..... $Wi(x)$ $Rj(x)/Wj(x)$ $Ci(x)$ -> This schedule is **not strict**

Problem to Solve on Strict Schedules

- Check whether the following schedule is strict schedule

$S5: r1(X); r2(Z); r3(X); r1(Z); r2(Y); r3(Y); w1(X); c1; w2(Z); w3(Y); w2(Y); c3; c2;$

Answer: Not, Strict

S5 is not strict because T2 writes Y before T3 commits.

Strict schedule: A schedule is strict if it satisfies the following conditions:

1. Tj reads a data item X **after** Ti has written to X and Ti is committed (aborted or terminated)
2. Tj writes a data item X **after** Ti has written to X and Ti is committed (aborted or terminated)

..... $Wi(x)$ $Ci(x)$ $Rj(x)/Wj(x)$ -> This schedule is **Strict**
..... $Wi(x)$ $Rj(x)/Wj(x)$ $Ci(x)$ -> This schedule is **not strict**

Problem to Solve on Strict Schedules

- Check whether the following schedule is strict schedule

1. R1(x), R2(x), R1(z), R3(x), R3(y), W1(x), C1, W3(y),
C3, R2(y), W2(z), W2(y), C2

Strict schedule: A schedule is strict if it satisfies the following conditions:

1. Tj reads a data item X **after** Ti has written to X and Ti is committed (aborted or terminated)
2. Tj writes a data item X **after** Ti has written to X and Ti is committed (aborted or terminated)

..... Wi(x) Ci(x) Rj(x)/Wj(x) -> This schedule is **Strict**
..... Wi(x) Rj(x)/Wj(x) Ci(x) -> This schedule is **not strict**

Problem to Solve on Strict Schedules

- Check whether the following schedule is strict schedule

1. R1(x), R2(x), R1(z), R3(x), R3(y), W1(x), C1, W3(y),
C3, R2(y), W2(z), W2(y), C2

Answer: Yes, Strict

Strict schedule: A schedule is strict if it satisfies the following conditions:

1. Tj reads a data item X **after** Ti has written to X and Ti is committed (aborted or terminated)
2. Tj writes a data item X **after** Ti has written to X and Ti is committed (aborted or terminated)

..... Wi(x) Ci(x) Rj(x)/Wj(x) -> This schedule is **Strict**
..... Wi(x) Rj(x)/Wj(x) Ci(x) -> This schedule is **not strict**

Summary of Schedules based on Recoverability

All Schedules

Recoverable Schedule

Cascadeless schedule

Strict Schedule

Note:

- Strict schedules are Cascadeless and Recoverable schedule
- Cascadeless schedules are Recoverable schedule

Transactions and Schedule

Transactions and Schedules

- Serial Schedule ... one after the other...
- Complete Schedule ... with Commit, Abort

Serializability

- "Correctness Measure" of some Schedule
- Why is it useful? It answers the question: "Will an interleaved schedule execute correctly
- i.e., a Serializable schedule will execute as correctly as serial schedule ... but in an interleaved manner!

Recoverability

- "Recoverability Measure" of some Schedule.
- Why is it useful? It answers the question: "Do we need to rollback a some (or all) transactions in an interleaved schedule after some Failure (e.g., ABORT)"
- i.e., in a Recoverable schedule no transaction needs to be rolled back once committed!

Properties of Transactions

ACID Properties

1. **A**tomicity
2. **C**onsistency preservation
3. **I**solation
4. **D**urability or permanency

Properties of Transactions

1. Atomicity

- Either all operations of a transaction occurs or none
- There should not be the case where half of the operations of transaction has been executed and other remaining half of the operations has not been executed.

Example of Transaction:

Increasing salary of an employee by 10%.

The three operations of this transactions are:

read(salary), $salary=salary+salary*0.10$,write(salary)

All this three operations should be executed for the transaction to be successful

Properties of Transactions

2. Consistency preservation (or Correctness)

A transaction should lead database from one consistent state to another consistent state.

Example: Say in a database table if both Date of Birth (**DOB**) and **Age** values are stored.

If any transaction changes DOB then appropriately the change in age value should reflected in database table. When DOB is changed but Age value has not been changed then table data will not be in consistent state.

Properties of Transactions

3. Isolation

Ensures that concurrent execution results in a system state that would be obtained if transaction would be executed serially.

Non-interleaved transaction (or Serial transaction): In this case first completely transaction T1 gets executed and then transaction T2 gets executed

Transaction1

```
Read(TippuSeats)
TippuSeats=TippuSeats-5
Write(TippuSeats)
Read(ChamundiSeats)
ChamundiSeats=ChamundiSeats+5
Write(ChamundiSeats)
```

Transaction2

```
Read(TippuSeats)
TippuSeats=TippuSeats+4
Write(TippuSeats)
```

Transaction T1	Transaction T2
<pre>Read(TippuSeats) TippuSeats=TippuSeats-5 Write(TippuSeats) Read(ChamundiSeats) ChamundiSeats=ChamundiSeats+5 Write(ChamundiSeats)</pre>	<pre>Read(TippuSeats) TippuSeats=TippuSeats+4 Write(TippuSeats)</pre>

Interleaved transaction (or Non-Serial or Concurrent):

First, Part of Transaction T1 gets executed, second Transaction T2 gets executed and third remaining part of transaction T1 gets executed

Properties of Transactions

4. Durability or permanency

Changes should be permanent. The changes must NOT be lost due to some database failure.

Summarizing Properties of Transactions

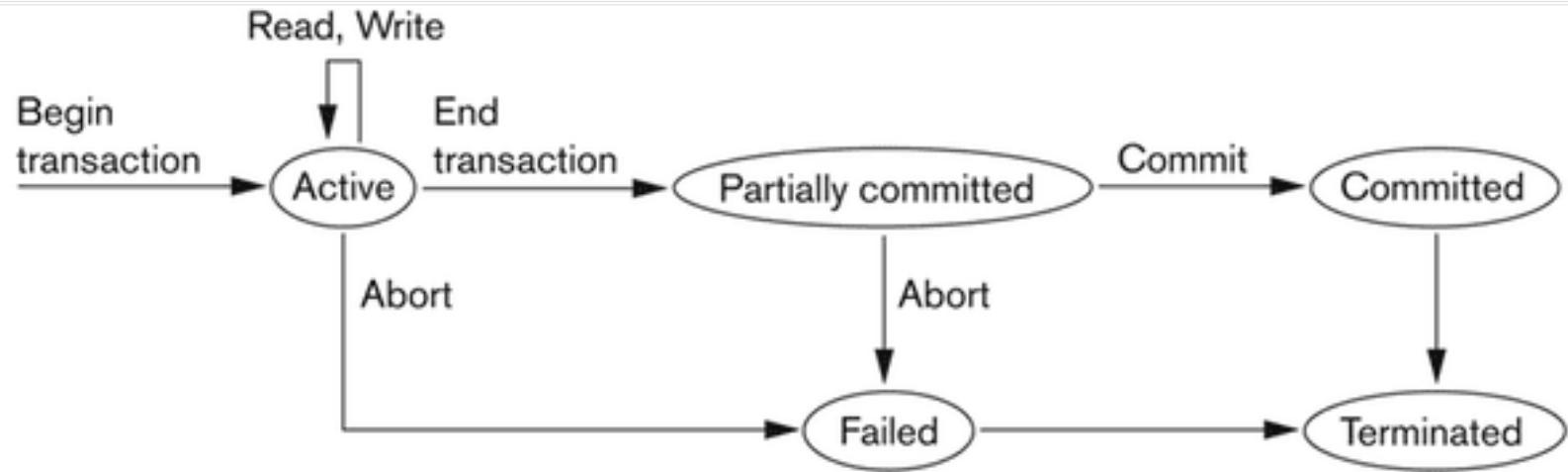
ACID Properties

1. **A**tomicity
2. **C**onsistency preservation
3. **I**solation
4. **D**urability or permanency

Transaction

- A transaction is an atomic unit of work that is either completed in its entirety or not done at all.
 - For **recovery purposes**, the system needs to keep track of when the transaction starts, terminates, and commits or aborts.
- **Transaction states:**
 - Active state
 - Partially committed state
 - Committed state
 - Failed state
 - Terminated State

State transition diagram illustrating the states for transaction execution



Active, the initial state; the transaction stays in this state while it is executing

Partially committed, after the final statement has been executed.

Failed, after the discovery that normal execution can no longer proceed.

Aborted, after the transaction has been rolled back and the database restored to its state prior to the start of the transaction. Two options after it has been aborted:

- restart the transaction – only if no internal logical error
- kill the transaction

Committed, after successful completion

Next what you are going learn is.....

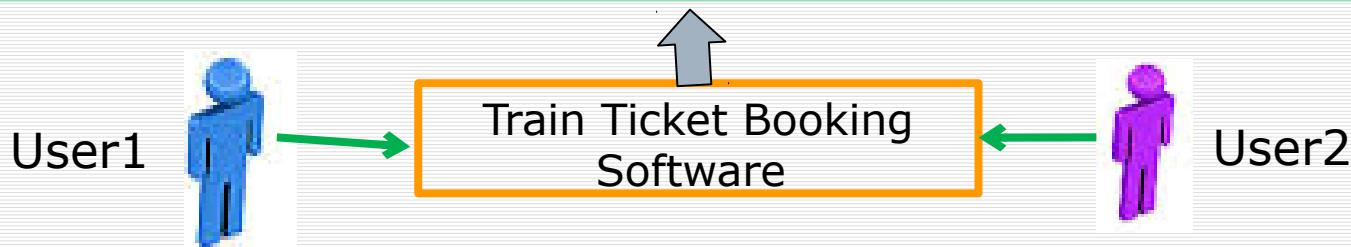
Concurrency Control Protocols

- To ensure when to give access to data item when transactions are getting executed concurrently or in interleaved way

Why we need Concurrency Control Protocols ?

Consider an example of Train Reservation System

Train Reservation Database					
Source	Destination	Train Name	Start Time	Start Date	Availability of Seats
Bangalore	Mysore	Tippu Express	15:00	25-3-2016	80
Bangalore	Mysore	Chamundi Express	18:15	25-3-2016	70



User1: Wants to **reserve** 5 seats on
Tippu express

Transaction T1

```
Read(TippuSeats)  
TippuSeats=TippuSeats-5  
Write(TippuSeats)
```

User2: Wants to **reserve** 4 seats on
Tippu express

Transaction T2

```
Read(TippuSeats)  
TippuSeats=TippuSeats-4  
Write(TippuSeats)
```

Why we need Concurrency Control Protocols ?

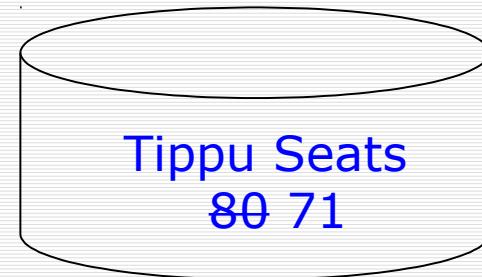
Case 1: When the **User1** Logins at **4:00pm** to booking system and **User 2** Logins at **4:03pm** to booking system

Time	T1	T2
4:00pm	Read(TippuSeats)	
4:01pm	TippuSeats=TippuSeats-5	
4:02pm	Write(TippuSeats)	
4:03pm		Read(TippuSeats)
4:04pm		TippuSeats=TippuSeats-4
4:05pm		Write(TippuSeats)

Database



Initial Value



Final Value

Why we need Concurrency Control Protocols ?

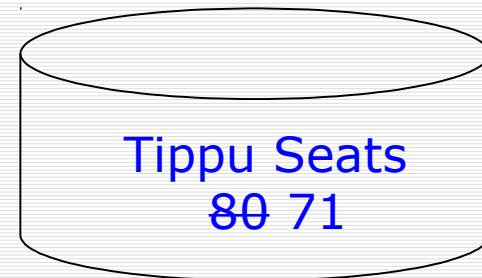
Case 2: When the **User1** Logins at **4:03pm** to booking system and **User 2** Logins at **4:00pm** to booking system

Time	T1	T2
4:00pm		Read(TippuSeats)
4:01pm		TippuSeats=TippuSeats-4
4:02pm		Write(TippuSeats)
4:03pm	Read(TippuSeats)	
4:04pm	TippuSeats=TippuSeats-5	
4:05pm	Write(TippuSeats)	

Database



Initial Value



Final Value

Why we need Concurrency Control Protocols ?

Case 3: When the **User1** Logins at **4:00pm** to booking system and **User 2** Logins at **4:01pm** to booking system

Time	T1	T2
4:00pm	Read(TippuSeats)	
4:01pm	TippuSeats=TippuSeats-5	Read(TippuSeats)
4:02pm	Write(TippuSeats)	TippuSeats=TippuSeats-4
4:03pm		Write(TippuSeats)
4:04pm		
4:05pm		

Database



Initial Value

Question:
What will be the final value Of TippuSeats when above Transactions are executed ?

Why we need Concurrency Control Protocols ?

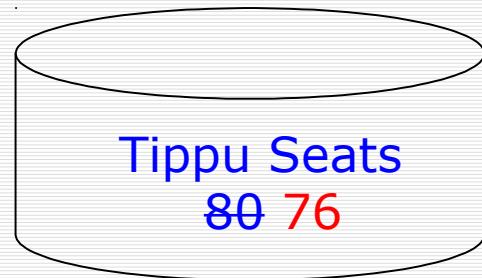
Case 3: When the User1 Logins at 4:00pm to booking system and User 2 Logins at 4:01pm to booking system

Time	T1	T2
4:00pm	Read(TippuSeats)	
4:01pm	TippuSeats=TippuSeats-5	Read(TippuSeats)
4:02pm	Write(TippuSeats)	TippuSeats=TippuSeats-4
4:03pm		Write(TippuSeats)
4:04pm		
4:05pm		

Database



Initial Value



Final Value

Database is in
Inconsistent state
WRONG value **76**
Updaed for Tippu Seats

Why we need Concurrency Control Protocols ?

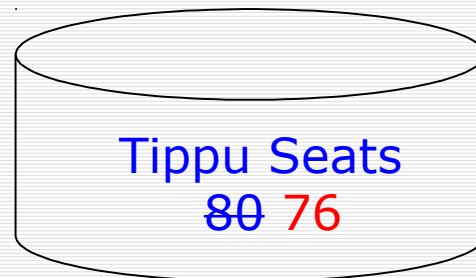
Case 3: When the User1 Logins at 4:00pm to booking system and User 2 Logins at 4:01pm to booking system

Time	T1	T2
4:00pm	Read(TippuSeats)	
4:01pm	TippuSeats=TippuSeats-5	Read(TippuSeats)
4:02pm	Write(TippuSeats)	TippuSeats=TippuSeats-4
4:03pm		Write(TippuSeats)
4:04pm		
4:05pm		

Database



Initial Value



Final Value

Database is in
Inconsistent state
WRONG value **76**
Updaed for Tippu Seats

**To avoid this type of
Problems Concurrency
Control Protocols
will be used**

Why we need Concurrency Control Protocols ?

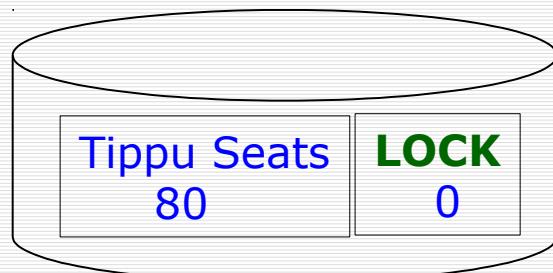
- When two or more users or transactions wants to access the **same data item** then concurrency control protocols should be followed to leave database to a consistent state after completion of executing transactions.
- One type Concurrency protocols are **LOCK based protocols**
- Under LOCK based protocol any transaction that needs to access the data item should first obtain the LOCK on the data item

Concurrency Control Protocol: LOCK based

For the following transactions schedule we will see by using LOCK based protocol, how to obtain correct value i.e., leaving the Database system in a consistent state after execution of transactions

Time	T1	T2
4:00pm	Read(TippuSeats)	
4:01pm	TippuSeats=TippuSeats-5	Read(TippuSeats)
4:02pm	Write(TippuSeats)	TippuSeats=TippuSeats-4
4:03pm		Write(TippuSeats)
4:04pm		
4:05pm		

Database

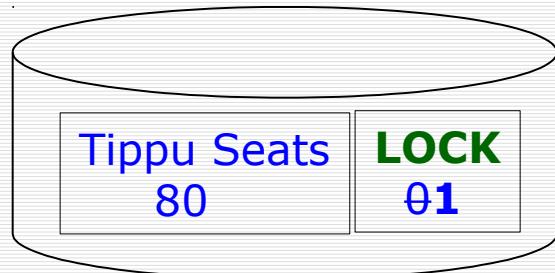


Initial Value

Concurrency Control Protocol: LOCK based

Time	T1	T2
4:00pm	LOCK (TippuSeats), Read(TippuSeats)	
4:01pm		
4:02pm		
4:03pm		
4:04pm		
4:05pm		

Database

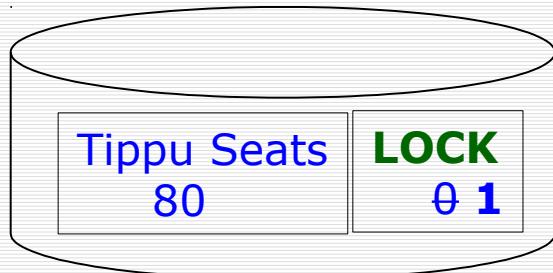


When Transaction T1 executes,
LOCK(TippuSeats), LOCK variable value of
TippuSeats will be changed from zero to one

Concurrency Control Protocol: LOCK based

Time	T1	T2
4:00pm	LOCK (TippuSeats), Read(TippuSeats)	
4:01pm	TippuSeats=TippuSeats-5	T2 will WAIT because it cannot obtain LOCK on data item TippuSeats
4:02pm		
4:03pm		
4:04pm		
4:05pm		

Database

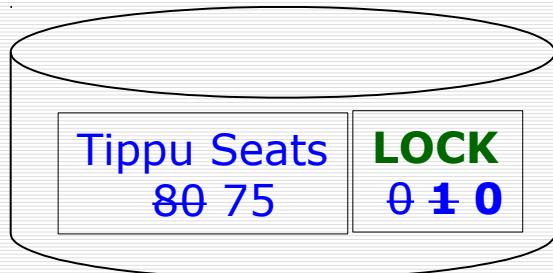


When transaction T2 comes at 4:01pm,
Then T2 will check for LOCK variable value
Of TippuSeats it is **ONE** so T2 **WAITS**

Concurrency Control Protocol: LOCK based

Time	T1	T2
4:00pm	LOCK (TippuSeats), Read(TippuSeats)	
4:01pm	TippuSeats=TippuSeats-5	T2 will WAIT because it cannot obtain LOCK on data item TippuSeats
4:02pm	Write(TippuSeats), UNLOCK (TippuSeats)	WAIT
4:03pm		
4:04pm		
4:05pm		

Database

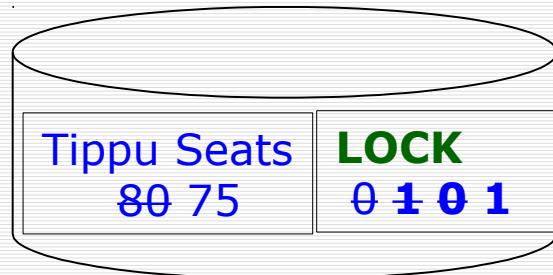


When transaction T1 executes
UNLOCK(TippuSeats), LOCK value of
TippuSeats will be changed from ONE
to ZERO

Concurrency Control Protocol: LOCK based

Time	T1	T2
4:00pm	LOCK (TippuSeats), Read(TippuSeats)	
4:01pm	TippuSeats=TippuSeats-5	T2 will WAIT because it cannot obtain LOCK on data item TippuSeats
4:02pm	Write(TippuSeats), UNLOCK (TippuSeats)	WAIT
4:03pm		LOCK (TippuSeats), Read(TippuSeats)
4:04pm		
4:05pm		

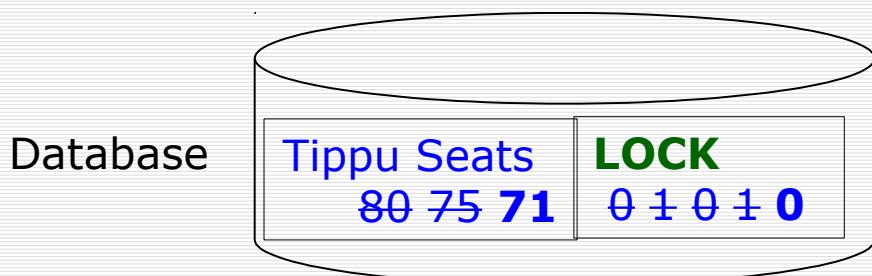
Database



When transaction T2 executes
LOCK(TippuSeats), LOCK value of
TippuSeats will be changed from ZERO
to ONE

Concurrency Control Protocol: LOCK based

Time	T1	T2
4:00pm	LOCK (TippuSeats), Read(TippuSeats)	
4:01pm	TippuSeats=TippuSeats-5	T2 will WAIT because it cannot obtain LOCK on data item TippuSeats
4:02pm	Write(TippuSeats), UNLOCK (TippuSeats)	WAIT
4:03pm		LOCK (TippuSeats), Read(TippuSeats)
4:04pm		TippuSeats=TippuSeats-4
4:05pm		Write(TippuSeats), UNLOCK (TippuSeats)



Now final value of TippuSeats is 71, which is CORRECT

Lock-based Protocols

- A transaction *must* get a *lock* before operating on the data. LOCKS are used to ensure concurrency.
- Two types of locks:
 - **Shared** (S) locks (also called **read** locks)
 - Obtained if transactions want to only read an item
 - **Exclusive** (X) locks (also called **write** locks)
 - Obtained for updating a data item

Lock-based Protocols

- A transaction *must* get a *lock* before operating on the data. LOCKS are used to ensure concurrency.
- Two types of locks:
 - **Shared** (S) locks (also called **read** locks)
 - Obtained if transactions want to only read an item
 - **Exclusive** (X) locks (also called **write** locks)
 - Obtained for updating a data item

		User 1, HOLDER of the LOCK on data item X	
		Shared or Read Lock	Write or Exclusive Lock
User2, REQUESTOR of LOCK for data item X	Shared or Read Lock		
	Write or Exclusive Lock		

Lock-based Protocols

- A transaction *must* get a *lock* before operating on the data
- Two types of locks:
 - **Shared** (S) locks (also called **read** locks)
 - Obtained if we want to only read an item
 - **Exclusive** (X) locks (also called **write** locks)
 - Obtained for updating a data item

		User 1, HOLDER of the LOCK on data item X	
		Shared or Read Lock	Write or Exclusive Lock
User2, REQUESTOR of LOCK for data item X	Shared or Read Lock	YES: USER 2 will granted with Read lock	
	Write or Exclusive Lock		

Lock-based Protocols

- A transaction *must* get a *lock* before operating on the data
- Two types of locks:
 - **Shared** (S) locks (also called **read** locks)
 - Obtained if transactions want to only read an item
 - **Exclusive** (X) locks (also called **write** locks)
 - Obtained for updating a data item

		User 1, HOLDER of the LOCK on data item X	
		Shared or Read Lock	Write or Exclusive Lock
User2, REQUESTOR of LOCK for data item X	Shared or Read Lock	YES: USER 2 will be granted with Read lock	NO: USER 2 will not be granted with Read lock
	Write or Exclusive Lock		

Lock-based Protocols

- A transaction *must* get a *lock* before operating on the data
- Two types of locks:
 - **Shared** (S) locks (also called **read** locks)
 - Obtained if transactions want to only read an item
 - **Exclusive** (X) locks (also called **write** locks)
 - Obtained for updating a data item

		User 1, HOLDER of the LOCK on data item X	
		Shared or Read Lock	Write or Exclusive Lock
User2, REQUESTOR of LOCK for data item X	Shared or Read Lock	YES: USER 2 will be granted with Read lock	NO: USER 2 will not be granted with Read lock
	Write or Exclusive Lock	??	??

Lock-based Protocols

- A transaction *must* get a *lock* before operating on the data
- Two types of locks:
 - **Shared** (S) locks (also called **read** locks)
 - Obtained if transactions want to only read an item
 - **Exclusive** (X) locks (also called **write** locks)
 - Obtained for updating a data item

		User 1, HOLDER of the LOCK on data item X	
		Shared or Read Lock	Write or Exclusive Lock
User2, REQUESTOR of LOCK for data item X	Shared or Read Lock	YES: USER 2 will be granted with Read lock	NO: USER 2 will not be granted with Read lock
	Write or Exclusive Lock	NO: USER 2 will not be granted with Write lock	NO: USER 2 will not be granted with Write lock

Lock instructions

- New LOCK instructions
 - Lock-S: shared (or read) lock request
 - Lock-X: exclusive (or write) lock request
 - Unlock: release previously held lock
- Example schedule:

T1	T2
read(B) B = B - 50 write(B) read(A) A = A + 50 write(A)	read(A) read(B) display(A+B)

Lock instructions

□ New LOCK instructions

- Lock-S: shared (or read) lock request
- Lock-X: exclusive (or write) lock request
- Unlock: release previously held lock

□ Example schedule:

T1	T2
<p>Lock-X(B) read(B) $B = B - 50$ write(B) Unlock(B) Lock-X(A) read(A) $A = A + 50$ write(A) Unlock(A)</p>	<p>Lock-S(A) read(A) Unlock(A) Lock-S(B) read(B) Unlock(B) display(A+B)</p>

Lock-based Protocols

- Lock requests are made to the *concurrency control manager*
 - It decides whether to *grant* a lock request
- T1 asks for a lock on data item A, and T2 currently has a lock on it ?
 - Depends

<u>T2 lock type</u>	<u>T1 lock type</u>	<u>Should allow ?</u>
Shared	Shared	YES
Shared	Exclusive	NO
Exclusive	-	NO

- If *compatible*, grant the lock, otherwise T1 waits in a *queue*.

Lock-based Protocols

- How do we actually use this to guarantee serializability/recoverability ?
 - Not enough just to take locks when you need to read/write something

T1

```
Lock-X(B)  
read(B)  
B ≈ B-50  
write(B)  
Unlock(B)
```

```
Lock-X(A)  
read(A)  
A ≈ A + 50  
write(A)  
Unlock(A)
```

T2

```
Lock-X(A), Lock-X(B)  
A = A-50  
B = B+50  
Unlock(A), Unlock(B)
```



Two-Phase Locking Protocol (2PL)

A transaction is said to follow two phase locking protocol if all locking operations (read-lock or shared lock, write-lock or Exclusive- lock) precede the first Unlock operation in the transaction. Such transaction can be divided into two phases

- Phase 1: Growing (or Lock Acquiring) phase
 - Transaction may obtain locks
 - But may not release them
- Phase 2: Shrinking (or Lock Releasing) phase
 - Transaction may only release locks

Two-Phase Locking Protocol (2PL)

- Phase 1: Growing (or Lock Acquiring) phase
 - Transaction may obtain locks
 - But may not release them
- Phase 2: Shrinking (or Lock Releasing) phase
 - Transaction may only release locks

T1: Not following 2PL

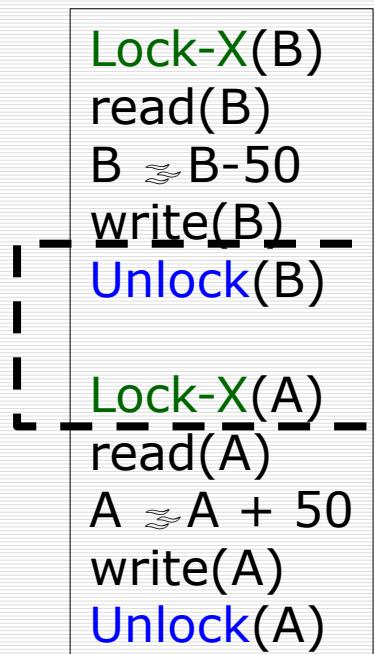
```
Lock-X(B)
read(B)
B  $\rightsquigarrow$  B-50
write(B)
Unlock(B)
```

```
Lock-X(A)
read(A)
A  $\rightsquigarrow$  A + 50
write(A)
Unlock(A)
```

Two-Phase Locking Protocol (2PL)

- Phase 1: Growing (or Lock Acquiring) phase
 - Transaction may obtain locks
 - But may not release them
- Phase 2: Shrinking (or Lock Releasing) phase
 - Transaction may only release locks

T1: Not following 2PL



Two-Phase Locking Protocol (2PL)

- Phase 1: Growing (or Lock Acquiring) phase
 - Transaction may obtain locks
 - But may not release them
- Phase 2: Shrinking (or Lock Releasing) phase
 - Transaction may only release locks

T1: Not following 2PL

```
Lock-X(B)
read(B)
B = B - 50
write(B)
Unlock(B)

Lock-X(A)
read(A)
A = A + 50
write(A)
Unlock(A)
```

Growing phase

T1: Follows 2PL

Shrinking phase

```
Lock-X(B)
read(B)
B = B - 50
write(B)
Lock-X(A)
read(A)
A = A - 50
write(A)
unlock(B)
unlock(A)
```

Variations of 2PL

1. Basic 2PL

- Two Phase: Growing and Shrinking Phase

2. Conservative (or static) 2PL

- Conservative 2PL requires a transaction to lock all the items it accesses before the transaction begins execution by pre declaring its write set and read set.
- If any of the pre declared items needed cannot be locked, the transaction does not lock any item; instead it **waits** until all the items are available for locking

3. Strict 2PL

- A transaction T **does not release any of its exclusive (write)** locks until after it **commits** or aborts.
- Hence no other transaction can read or write an item that is written by T unless T has committed.

4. Rigorous 2PL

- A transaction T **does not release any of its locks (exclusive or shared)** until after it **commits** or aborts and so it is easier to implement than strict 2PL.

Database Recovery Systems

Database Recovery Systems looks to achieve

- Transaction Atomicity
- Transaction Durability

Why recovery is Needed ?

To avoid following failures (or What causes a Transaction to fail)

1. **A computer failure (system crash):** A **hardware or software error** occurs in the computer system during transaction execution. If the hardware crashes, the contents of the computer's internal memory may be lost.
2. **A transaction or system error:** Some operation in the transaction may cause it to fail, such as **integer overflow or division by zero**. Transaction failure may also occur because of erroneous parameter values or because of a logical programming error. In addition, the user may interrupt the transaction during its execution.
3. Local errors or **exception conditions detected by the transaction:** Certain conditions necessitate cancellation of the transaction. For example, data for the transaction may not be found. A condition, such as **insufficient account balance in a banking database**, may cause a transaction, such as a fund withdrawal from that account, to be canceled. A programmed abort in the transaction causes it to fail.
4. **Concurrency control enforcement:** The concurrency control method may decide to abort the transaction, to be restarted later, because it **violates serializability** or because several transactions are in a **state of deadlock**.
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. This may happen during a read or a write operation of the transaction.
6. **Physical problems and catastrophes:** This refers to an endless list of problems that includes power or air-conditioning failure, fire, theft, sabotage, overwriting disks by mistake.

Failure Classification

Transaction failure :

- Logical errors: transaction cannot complete due to some internal error condition
- System errors: the database system must terminate an active transaction due to an error condition (e.g., deadlock)

System crash: a power failure or other hardware or software failure causes the system to crash.

- Fail-stop assumption: non-volatile storage contents are assumed to not be corrupted by system crash
 - Database systems have numerous integrity checks to prevent corruption of disk data

Disk failure: a head crash or similar disk failure destroys all or part of disk storage

- Destruction is assumed to be detectable: disk drives use checksums to detect failures

Recovery Algorithms

- Recovery algorithms are techniques to ensure database consistency and transaction atomicity and durability despite failures
- Recovery algorithms have two parts
 1. Actions taken during normal transaction processing to ensure enough information exists to recover from failures
 2. Actions taken after a failure to recover the database contents to a state that ensures atomicity, consistency and durability

Example: Database Transaction

- Transferring Rs.100/- from account A to account B

The diagram illustrates a database transaction. It features a large pink oval representing a transaction boundary. Inside the oval, the word "ACCOUNTS" is written in green capital letters. Below it is a table with a black border, containing two columns labeled "Account" and "Balance". The table has three rows. The first row contains "A" in the Account column and "150" in the Balance column. The second row contains "B" in the Account column and "100" in the Balance column.

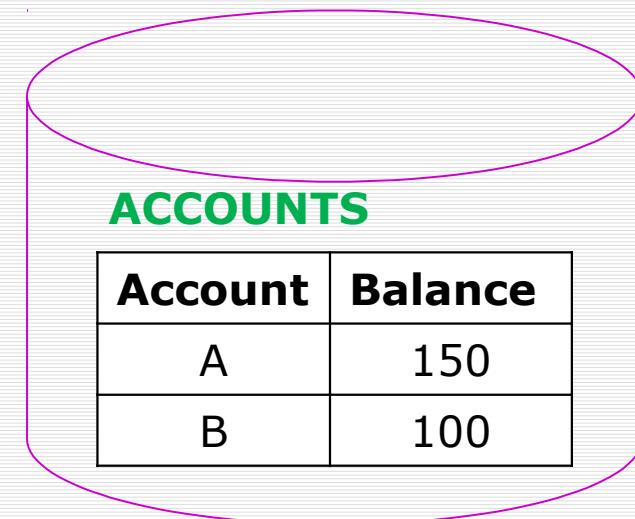
Account	Balance
A	150
B	100

Example: Database Transaction

- Transferring Rs.100/- from account A to account B

Transaction T1

Read(A)
A=A-100
Write(A)
Read(B)
B=B+100
Write(B)



Example: Database Transaction

- Transferring Rs.100/- from account A to account B

Transaction T1

Read(A)
A=A-100
Write(A)
Read(B)
B=B+100
Write(B)

The diagram shows a pink oval representing a transaction boundary. Inside the oval is a table titled "ACCOUNTS". The table has two columns: "Account" and "Balance". It contains two rows: one for account A with balance 150 50, and one for account B with balance 100 200.

ACCOUNTS	
Account	Balance
A	150 50
B	100 200

State of Database, After
completely executing
all operations of
Transaction T1

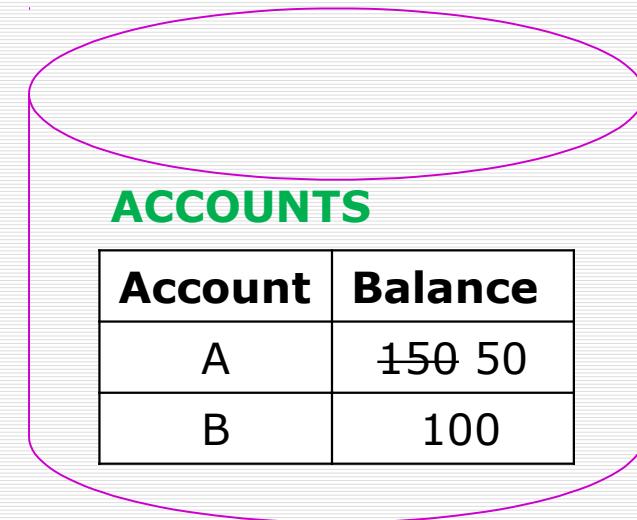
Example: Database Transaction

- Transferring Rs.100/- from account A to account B

Transaction T1

Read(A)
A=A-100
Write(A)
Read(B)
B=B+100
Write(B)

System Crash →

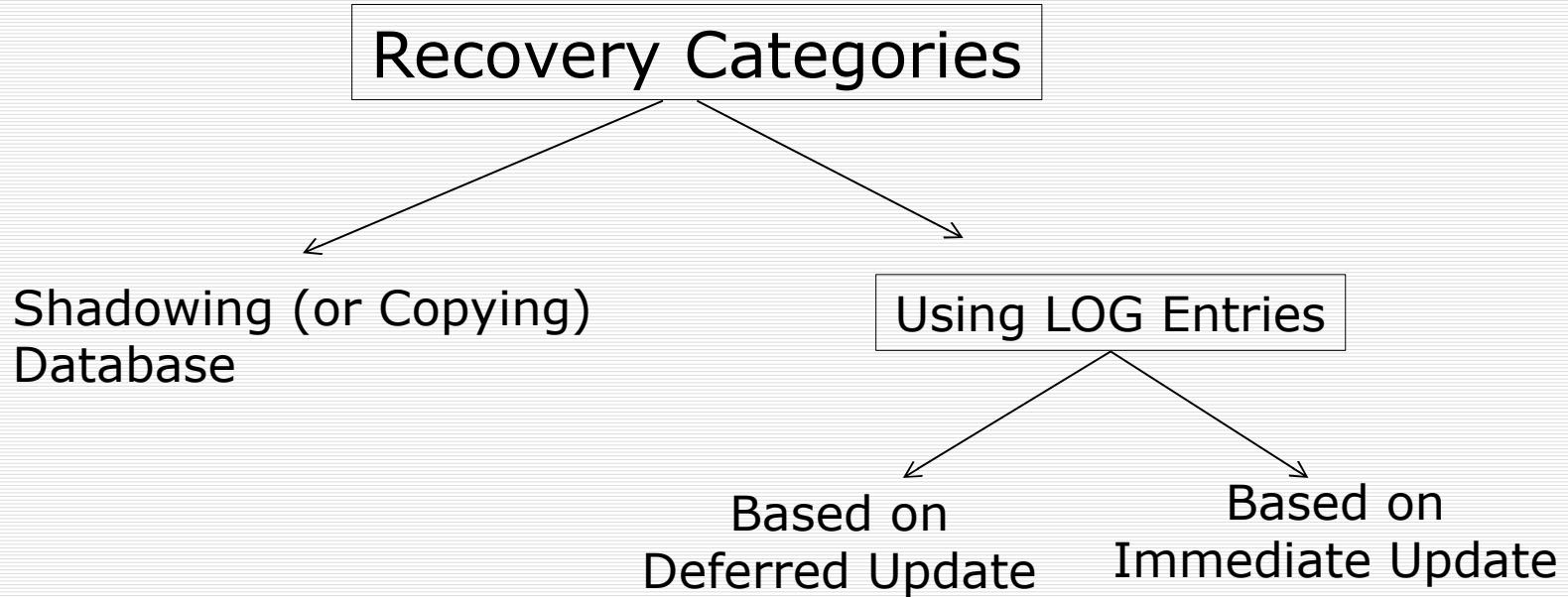


ACCOUNTS		
Account	Balance	
A	150	50
B	100	

State of Database, until
The system crash.

Now database is inconsistent

Database Recovery Mechanisms



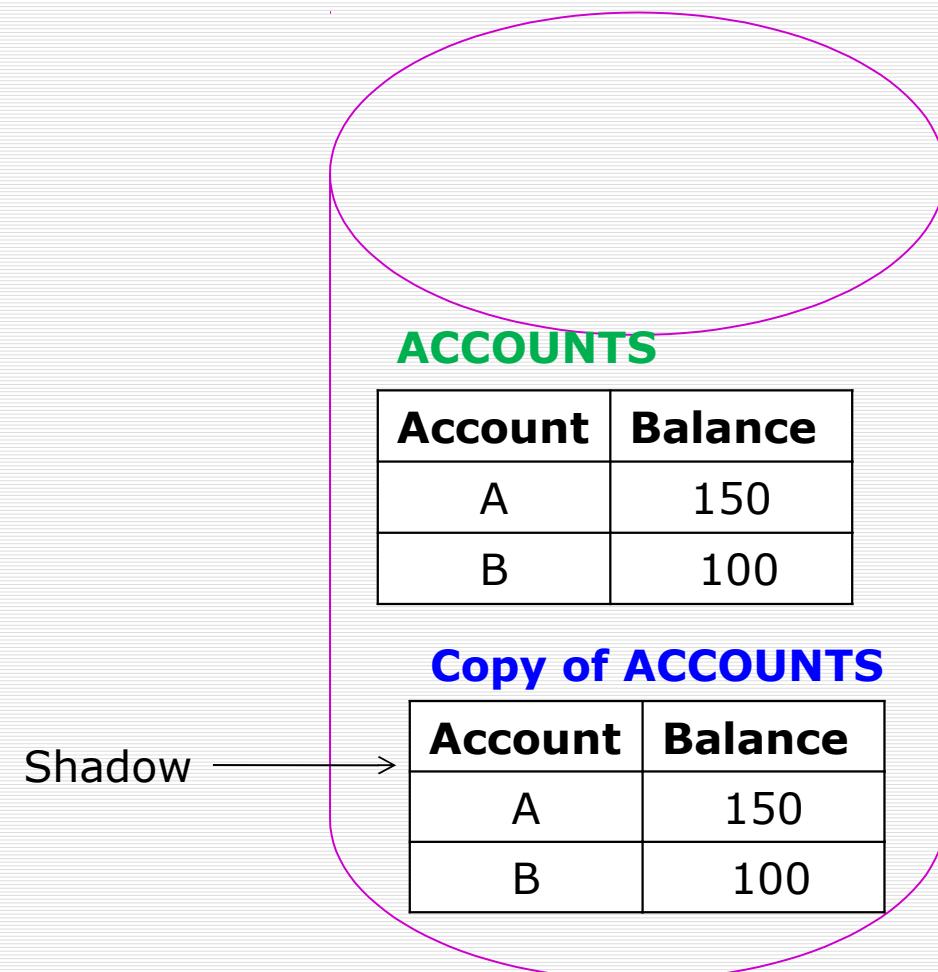
Recovery using Shadowing

- Transferring Rs.100/- from account A to account B

T1 Transaction T1

Begin →

Read(A)
A=A-100
Write(A)
Read(B)
B=B+100
Write(B)



Recovery using Shadowing

- Transferring Rs.100/- from account A to account B

T1 Transaction T1

Begin →

Read(A)
A=A-100
Write(A)
Read(B)
B=B+100
Write(B)

System Crash →

Question:

Now, when system crash
Occurs how the database
Can be restored back to original
Data values

ACCOUNTS

Account	Balance
A	150 50
B	100

Copy of ACCOUNTS

Account	Balance
A	150
B	100

Recovery using Shadowing

- Transferring Rs.100/- from account A to account B

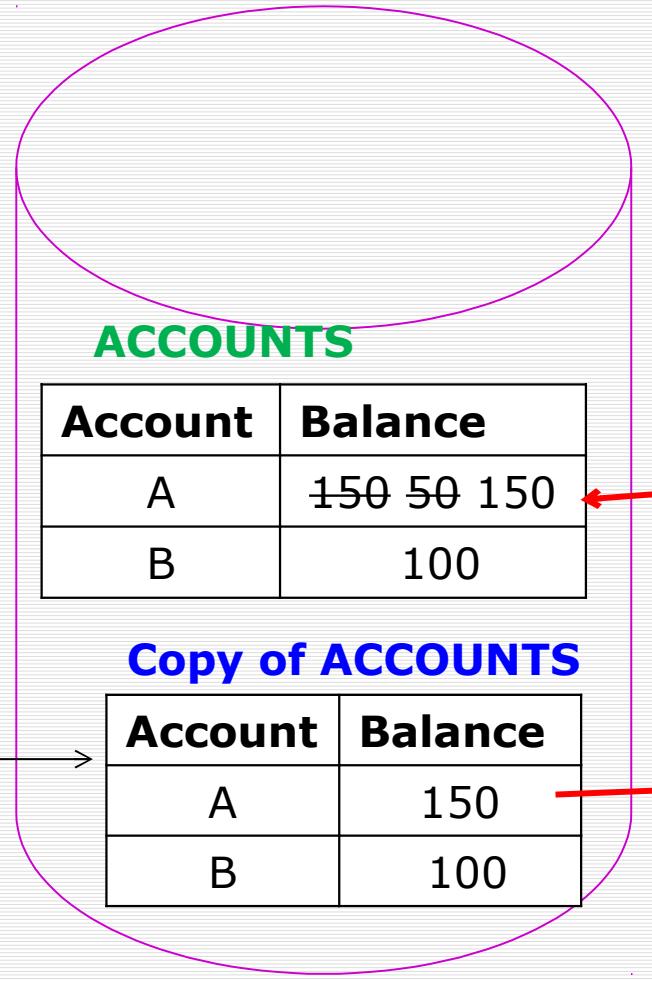
T1 Transaction T1

Begin →

Read(A)
A=A-100
Write(A)
Read(B)
B=B+100
Write(B)

System Crash →

Shadow
Of
Accounts
Table



When System
crash Occurs
copy from
Shadow

Recovery using Log Entries

- Transferring Rs.100/- from account A to account B

Transaction T1

Read(A)
A=A-100
Write(A)
Read(B)
B=B+100
Write(B)
Commit

ACCOUNTS

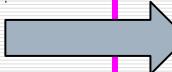
Account	Balance
A	150
B	100

LOG of T1

[Start, T1]

[Read, T1, A]

Log
Entries
Of T1



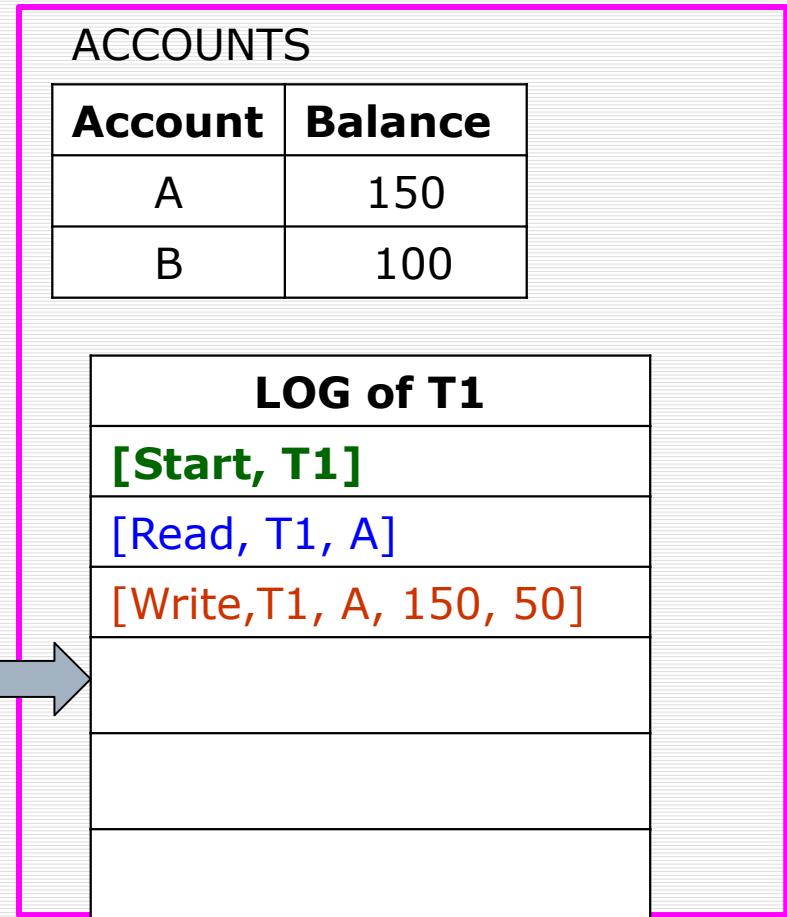
Recovery using Log Entries

- Transferring Rs.100/- from account A to account B

Transaction T1

Read(A)
A=A-100
Write(A)
Read(B)
B=B+100
Write(B)
Commit

Log
Entries
Of T1



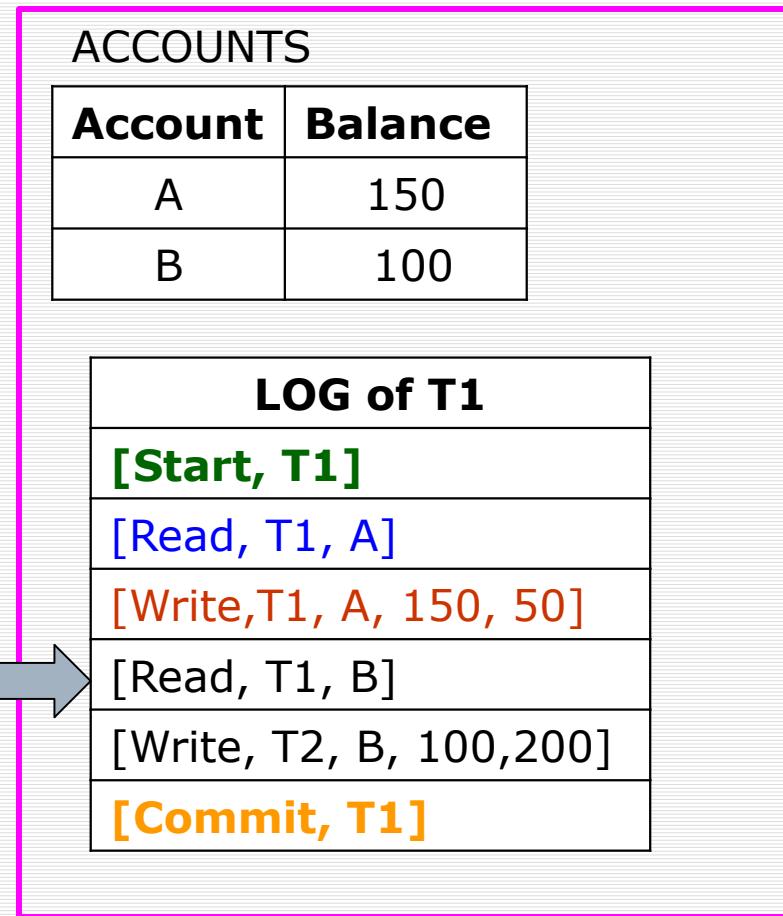
Recovery using Log Entries

- Transferring Rs.100/- from account A to account B

Transaction T1

Read(A)
A=A-100
Write(A)
Read(B)
B=B+100
Write(B)
Commit

Log
Entries
Of T1



Recovery using Log Entries

- Transferring Rs.100/- from account A to account B

Transaction T1

Read(A)
A=A-100
Write(A)
Read(B)
B=B+100
Write(B)
Commit

System Crash →

Question:

Now, when system crash
Occurs how the database
Can be restored back to original
Data values

ACCOUNTS

Account	Balance
A	150 50
B	100

LOG of T1

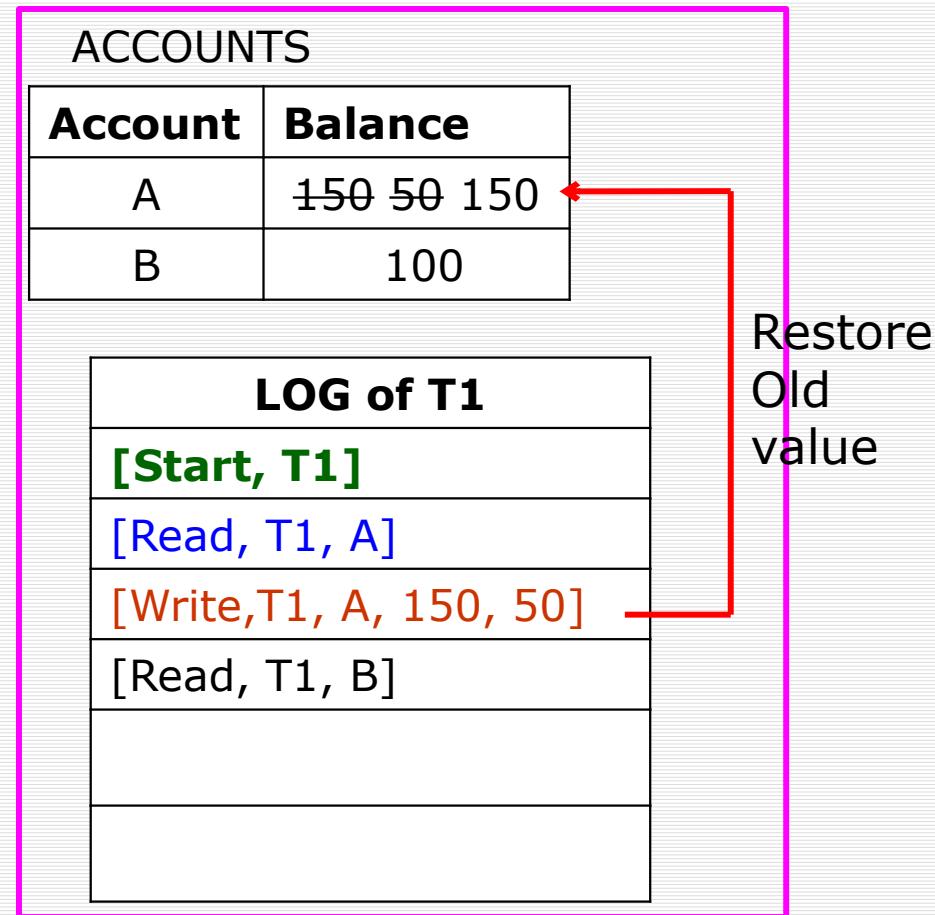
[Start, T1]
[Read, T1, A]
[Write, T1, A, 150, 50]
[Read, T1, B]

Recovery using Log Entries

- Transferring Rs.100/- from account A to account B

Transaction T1	
Read(A)	
A=A-100	
Write(A)	
Read(B)	
B=B+100	
Write(B)	
Commit	

System Crash →



Recovery using Log Entries

- Transferring Rs.100/- from account A to account B

Transaction T1

Read(A)
A=A-100
Write(A)
Read(B)
B=B+100
Write(B)
Commit

ACCOUNTS

Account	Balance
A	150
B	100

LOG of T1

[Start, T1]
[Read, T1, A]
[Write, T1, A, 150, 50]

Question:

Why should we maintain old value in LOG for write operation

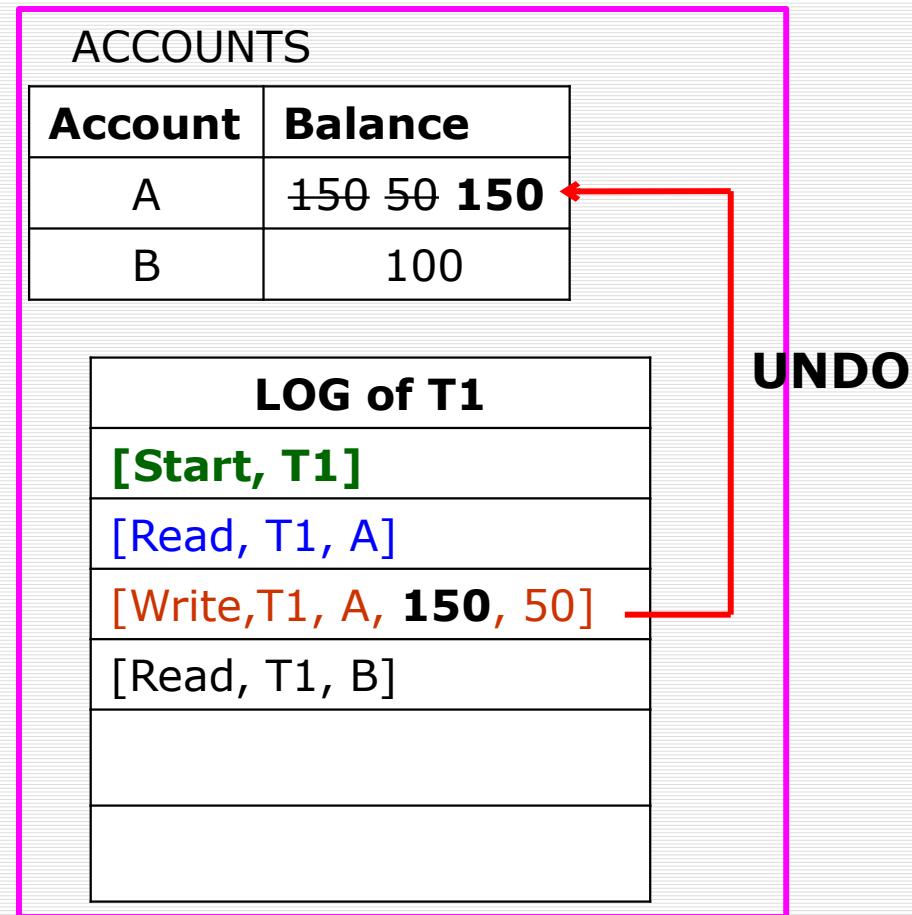
Recovery using Log Entries

- Transferring Rs.100/- from account A to account B

Transaction T1	
Read(A)	
A=A-100	
Write(A)	
Read(B)	
B=B+100	
Write(B)	
Commit	

System Crash →

Old value is required for **UNDO**,
If changes were made to database
Before transaction reaches commit



Recovery using Log Entries

Consider two transactions, Transferring Rs.100/- from account A to account B and Rs. 200/- from account C to D

T1
Read(A)
A=A-100
Write(A)
Read(B)
B=B+100
Write(B)
Commit
T2
Read(C)
C=C-200
Write(C)
Read(D)
D=D+200
Write(D)
Commit

Account	Balance
A	150
B	100
C	400
D	200

Recovery using Log Entries

Consider two transactions, Transferring Rs.100/- from account A to account B and Rs. 200/- from account C to D

T1
Read(A)
A=A-100
Write(A)
Read(B)
B=B+100
Write(B)
Commit
T2
Read(C)
C=C-200
Write(C)
Read(D)
D=D+200
Write(D)
Commit

Account	Balance
A	150 50
B	100 200
C	400 200
D	200 400

LOG of T1
[Start, T1]
[Read, T1, A]
[Write, T1, A, 150, 50]
[Read, T1, B]
[Write, T1, B, 100,200]
[Commit, T1]
[Start, T2]
[Read, T2, C]
[Write, T2, C, 400, 200]
[Read, T2, D]
[Write, T2, D, 200,400]
[Commit, T2]

Recovery using Log Entries

Consider two transactions, Transferring Rs.100/- from account A to account B and Rs. 200/- from account C to D

T1
Read(A)
A=A-100
Write(A)
Read(B)
B=B+100
Write(B)
Commit
T2
Read(C)
C=C-200
Write(C)
Read(D)
D=D+200
Write(D)
Commit

System Crash →

Question:
Why we should maintain
New value
in LOG for write operation

Account	Balance
A	150
B	100
C	400
D	200

LOG of T1
[Start, T1]
[Read, T1, A]
[Write, T1, A, 150, 50]
[Read, T1, B]
[Write, T1, B, 100, 200]
[Commit, T1]
[Start, T2]
[Read, T2, C]
[Write, T2, C, 400, 200]
[Read, T2, D]

Recovery using Log Entries

Consider two transactions, Transferring Rs.100/- from account A to account B and Rs. 200/- from account C to D

T1
Read(A)
A=A-100
Write(A)
Read(B)
B=B+100
Write(B)
Commit
T2
Read(C)
C=C-200
Write(C)
Read(D)
D=D+200
Write(D)
Commit

System Crash →

New value is required for **REDO** operation in Case where you have Reached commit point of Transaction but Changes were not Updated To database table

Account	Balance
A	150 50
B	100 200
C	200
D	400

LOG of T1	
[Start, T1]	
[Read, T1, A]	
[Write, T1, A, 150, 50]	
[Read, T1, B]	
[Write, T1, B, 100, 200]	
[Commit, T1]	
LOG of T2	
[Start, T2]	
[Read, T2, C]	
[Write, T2, C, 400, 200]	
[Read, T2, D]	

REDO

Recovery using Log Entries

1. Old value in LOG is required For UNDO operation

Old value is required for UNDO, If changes where made to database

Before transaction reaches commit

2. New value in LOG is required For REDO operation

New value is required for REDO operation in Case where we have Reached commit point of Transaction but Changes where not Updated To database table.

Recovery using LOG entries

Consider three transactions T_1 , T_2 , T_3 as follows

T_1	T_2	T_3
read_item(A)	read_item(B)	read_item(C)
read_item(D)	write_item(B)	write_item(B)
write_item(D)	read_item(D)	read_item(A)
	write_item(D)	write_item(A)

Database

A	30
B	15
C	40
D	20

Recovery using LOG entries

Consider three transactions T1, T2, T3 as follows

T_1
read_item(A)
read_item(D)
write_item(D)

T_2
read_item(B)
write_item(B)
read_item(D)
write_item(D)

T_3
read_item(C)
write_item(B)
read_item(A)
write_item(A)

Database

A	30
B	15
C	40
D	20

[start_transaction, T_3]
[read_item, T_3, C]
[write_item, $T_3, B, 15, 12$]
[start_transaction, T_2]
[read_item, T_2, B]
[write_item, $T_2, B, 12, 18$]
[start_transaction, T_1]
[read_item, T_1, A]
[read_item, T_1, D]
[write_item, $T_1, D, 20, 25$]
[read_item, T_2, D]
[write_item, $T_2, D, 25, 26$]
[read_item, T_3, A]

Recovery using LOG entries

Consider three transactions T1, T2, T3 as follows

Database

A	30
B	15 12 18
C	40
D	20 25 26

System
Crash 

A	B	C	D
[start_transaction, T_3]			
[read_item, T_3, C]			
[write_item, $T_3, B, 15, 12$]	12		
[start_transaction, T_2]			
[read_item, T_2, B]			
[write_item, $T_2, B, 12, 18$]	18		
[start_transaction, T_1]			
[read_item, T_1, A]			
[read_item, T_1, D]			
[write_item, $T_1, D, 20, 25$]			25
[read_item, T_2, D]			
[write_item, $T_2, D, 25, 26$]			26
[read_item, T_3, A]			

Recovery using LOG entries

Consider three transactions T1, T2, T3 as follows

Database

A	30
B	15 12 18
C	40
D	20 25 26

Question:

After System, how can we recovery
Database i.e
B value to 15,
D value to 20

System
Crash 

A	B	C	D
[start_transaction, T_3]			
[read_item, T_3, C]			
[write_item, $T_3, B, 15, 12$]	12		
[start_transaction, T_2]			
[read_item, T_2, B]			
[write_item, $T_2, B, 12, 18$]	18		
[start_transaction, T_1]			
[read_item, T_1, A]			
[read_item, T_1, D]			
[write_item, $T_1, D, 20, 25$]			25
[read_item, T_2, D]			
[write_item, $T_2, D, 25, 26$]			26
[read_item, T_3, A]			

Recovery using LOG entries

Consider three transactions T1, T2, T3 as follows

- * T3 transaction is rolled back because it did not reached its commit point
- ** T2 is rolled back because it reads the value of item B written by T3

Database

A	30
B	15 12 18 15
C	40
D	20 25 26

System
Crash 

A	B	C	D
30	15	40	20
[start_transaction, T_3]			
[read_item, T_3 , C]			
[write_item, T_3 , B, 15, 12]		12	
[start_transaction, T_2]			
[read_item, T_2 , B]			
[write_item, T_2 , B, 12, 18]		18	
[start_transaction, T_1]			
[read_item, T_1 , A]			
[read_item, T_1 , D]			
[write_item, T_1 , D, 20, 25]			25
[read_item, T_2 , D]			
[write_item, T_2 , D, 25, 26]			26
[read_item, T_3 , A]			

Recovery using LOG entries

Consider three transactions T1, T2, T3 as follows

Question:

Can you answer now, why we maintain
Data item name and transaction number
For read operation in LOG ?

Database

A	30
B	15 12 18
C	40
D	20 25 26

System
Crash →

A	B	C	D
30	15	40	20
[start_transaction, T_3]			
[read_item, T_3 ,C]			
[write_item, T_3 ,B,15,12]	12		
[start_transaction, T_2]			
[read_item, T_2 ,B]			
[write_item, T_2 ,B,12,18]	18		
[start_transaction, T_1]			
[read_item, T_1 ,A]			
[read_item, T_1 ,D]			
[write_item, T_1 ,D,20,25]	25		
[read_item, T_2 ,D]			
[write_item, T_2 ,D,25,26]	26		
[read_item, T_3 ,A]			

Recovery using LOG entries

Consider three transactions T1, T2, T3 as follows

T3 and T1 transaction is rolled back because it did not reached its commit point

T2 is rolled back because it reads the value of item

- B written by T3
- D written by T1

Database

A	30
B	15 12 18 15
C	40
D	20 25 26 20

System
Crash 

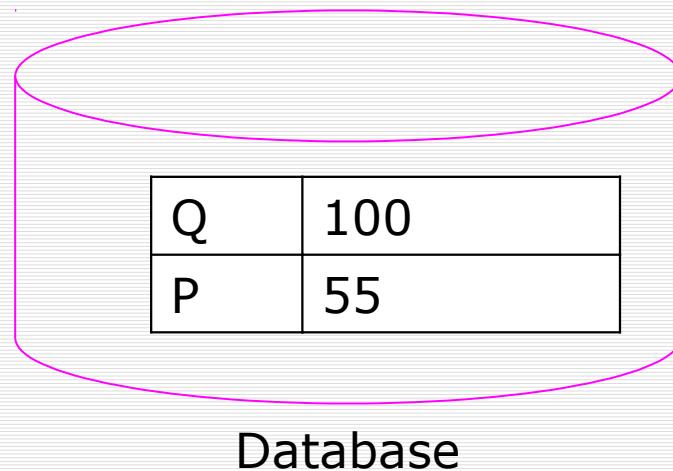
	A	B	C	D
*	30	15	40	20
*	[start_transaction, T_3]			
*	[read_item, T_3 ,C]			
*	[write_item, T_3 ,B,15,12]	12		
**	[start_transaction, T_2]			
**	[read_item, T_2 ,B]			
**	[write_item, T_2 ,B,12,18]	18		
**	[start_transaction, T_1]			
**	[read_item, T_1 ,A]			
**	[read_item, T_1 ,D]			
**	[write_item, T_1 ,D,20,25]	25		
**	[read_item, T_2 ,D]			
**	[write_item, T_2 ,D,25,26]			26
**	[read_item, T_3 ,A]			

Problem to Solve

- Consider the following example of log for two transactions.

1. (Start, T1);
2. (Write, T1, Q, 100,50);
3. (Commit, T1);
4. (Start, T2);
5. (Write, T2, P, 55, 10);
6. (Commit, T2);

Consider the case where the schedule crashes after Step 4 and before Step 5, then the question is which operation should we REDO if following is the scenario of database just before crash.

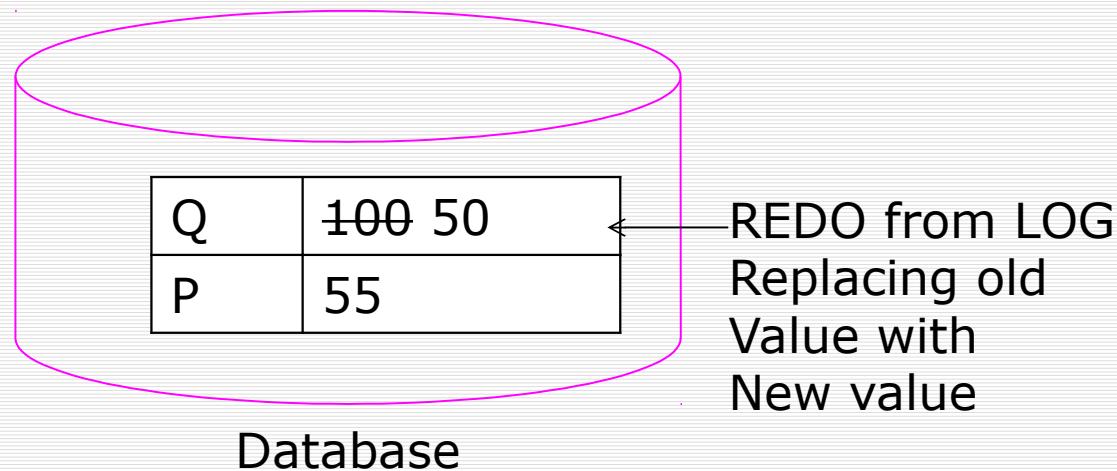


Answer

- Consider the following example of log for two transactions.

1. (Start, T1);
2. (Write, T1, Q, 100,50);
3. (Commit, T1);
4. (Start, T2);
5. (Write, T2, P, 55, 10);
6. (Commit, T2);

Consider the case where the schedule crashes after Step 4 and before Step 5, then the question is which operation should we REDO if following is the scenario of database just before crash.

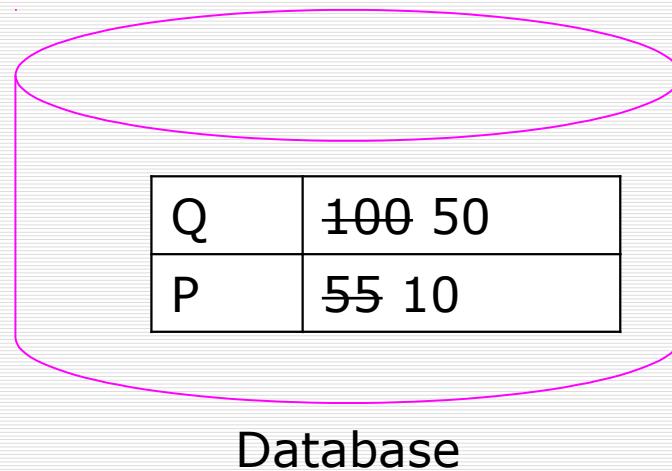


Problem to Solve

- Consider the following example of log for two transactions.

1. (Start, T1);
2. (Write, T1, Q, 100,50);
3. (Commit, T1);
4. (Start, T2);
5. (Write, T2, P, 55, 10);
6. (Commit, T2);

Consider the case where the schedule crashes after Step 5 and before Step 6, then the question is which operation should we UNDO if following is the scenario of database just before crash.

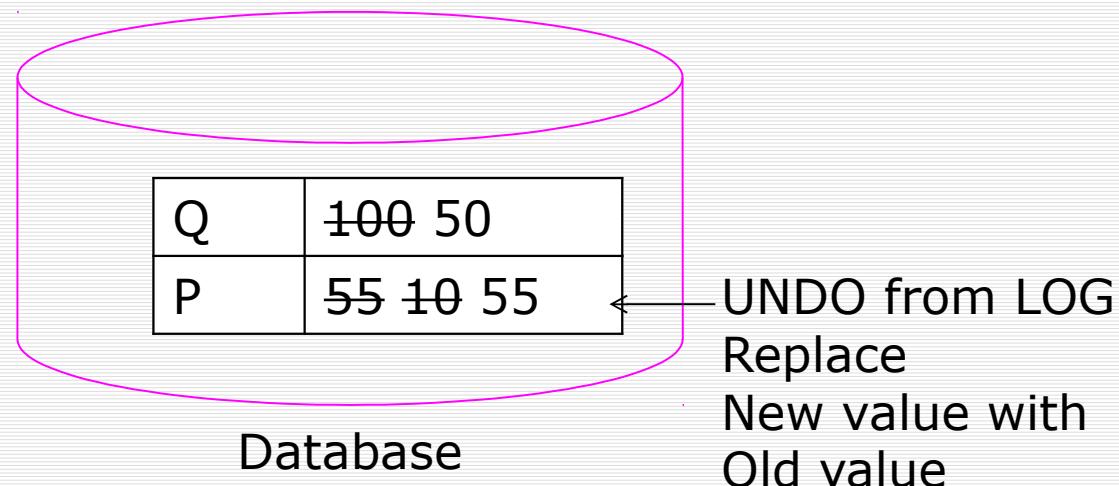


Answer

- Consider the following example of log for two transactions.

1. (Start, T1);
2. (Write, T1, Q, 100,50);
3. (Commit, T1);
4. (Start, T2);
5. (Write, T2, P, 55, 10);
6. (Commit, T2);

Consider the case where the schedule crashes after Step 5 and before Step 6, then the question is which operation should we UNDO if following is the scenario of database just before crash.



Approaches to Recovery

Two Types using LOG entries

1. Deferred Update
2. Immediate Update

Log-Based Recovery

- A log is kept on stable storage.
 - The log is a sequence of log records, and maintains a record of update activities on the database.
- When transaction T_i starts, it registers itself by writing a $\langle T_i \text{ start} \rangle$ log record
- Before T_i executes $\text{write}(X)$, a log record $\langle \text{write}, T_i, X, V_1, V_2 \rangle$ is written, where V_1 is the value of X before the write, and V_2 is the new value to be written to X .
 - Log record notes that T_i has performed a write on data item X that had value V_1 before the write, and will have value V_2 after the write.
- When T_i finishes its last statement, the log record $\langle \text{Commit } T_i \rangle$ is written.
- We assume for now that log records are written directly to stable storage (that is, they are not buffered)

Deferred Database Modification

- The deferred database modification scheme records all modifications to the log, but defers all the writes to after partial commit.
- Assume that transactions execute serially
- Transaction starts by writing $\langle \text{start } Ti \rangle$ record to log.
- A write(X) operation results in a log record $\langle \text{write}, Ti, X, V1, V2 \rangle$ being written, where V1 is old value of X, V2 is the new value for X
 - Note: old value V1 is not needed for this scheme
- The write is not performed on X at this time, but is deferred.
- When Ti partially commits, $\langle \text{Commit, } Ti \rangle$ is written to the log
- Finally, the log records are read and used to actually execute the previously deferred writes.
- During recovery after a crash, a transaction needs to be redone if and only if both $\langle \text{start } Ti \rangle$ and $\langle \text{commit } Ti \rangle$ are there in the log.
- Redoing a transaction Ti (redo Ti) sets the value of all data items updated by the transaction to the new values.
- Crashes can occur while: the transaction is executing the original updates or while recovery action is being taken
- Deferred modification is also referred as **No-Undo/Redo** method

Deferred Database Modification

Example of Deferred based recovery

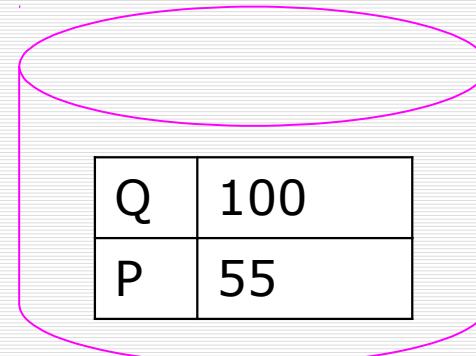
- Consider log entries for two transactions T1 and T2 as follows.

LOG

System
Crash →

(Start, T1)
(Write, T1, Q, 100,50)
(Commit, T1)
(Start, T2)
(Write, T2, P, 55, 10)

Database



Under deferred recovery method, after system crash the algorithm looks at LOG entries for all transactions T_i if the entry of both $\langle \text{start}, T_i \rangle$ and $\langle \text{commit}, T_i \rangle$ is present then for all write operations REDO will be carried out i.e., replacing with new value

Deferred Database Modification

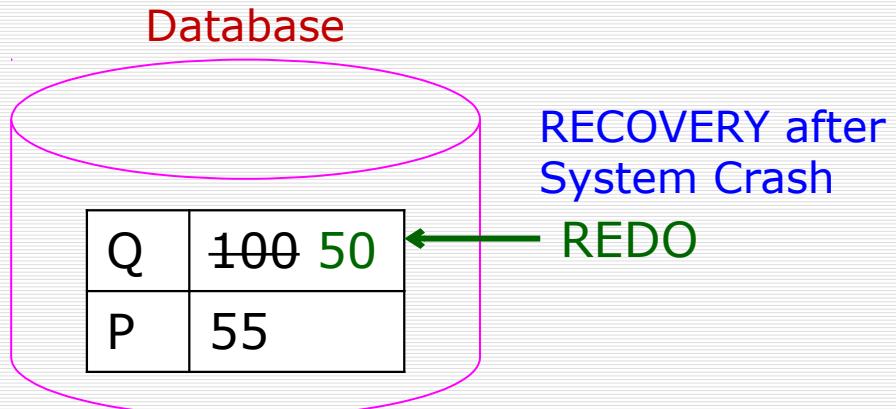
Example of Deferred based recovery

- Consider log entries for two transactions T1 and T2 as follows.

LOG

(Start, T1)
(Write, T1, Q, 100, 50)
(Commit, T1)
(Start, T2)
(Write, T2, P, 55, 10)

System
Crash →



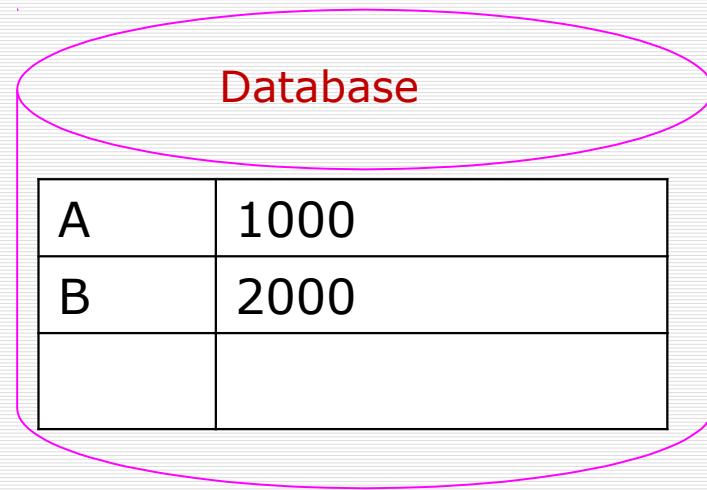
Under deferred recovery method, after system crash the algorithm looks at LOG entries for all transactions T_i if the entry of both $\langle \text{start}, T_i \rangle$ and $\langle \text{commit}, T_i \rangle$ is present then for all write operations REDO will be carried out i.e., replacing with new value

Deferred Database Modification

Question:

If following is the LOG on stable storage after systems crash,
then recovery algorithm should carry out any REDO operation ?

<Start, T1>
<write,T1,A,1000,950>
<write,T1,B,2000,2050>

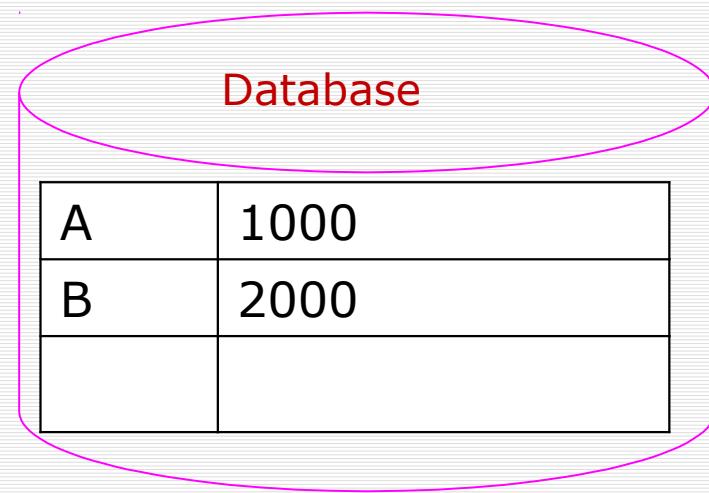


Deferred Database Modification

Question:

If following is the LOG on stable storage after systems crash,
then recovery algorithm should carry out any REDO operation ?

<Start, T1>
<write,T1,A,1000,950>
<write,T1,B,2000,2050>



Answer:

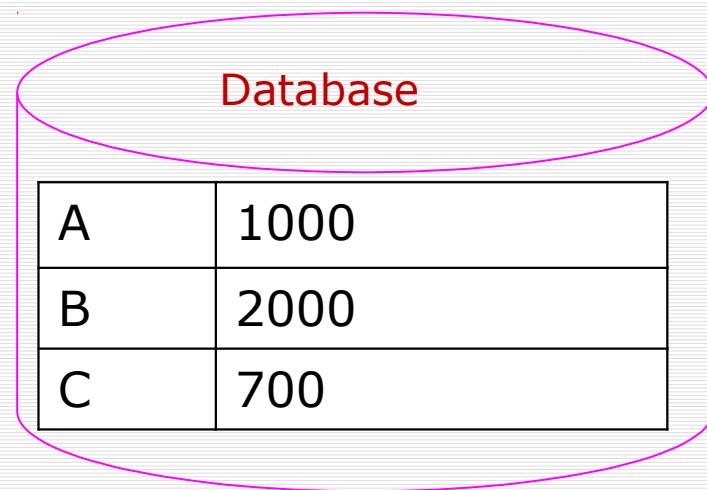
No REDO operation need to be taken because no commit
LOG entry found for any transaction

Deferred Database Modification

Question:

If following is the LOG on stable storage after systems crash,
then recovery algorithm should carry out any REDO operation ?

<Start, T1>
<write,T1,A,1000,950>
<write,T1,B,2000,2050>
<Commit, T1>
<Start, T2>
<write,T2,C,700,600>

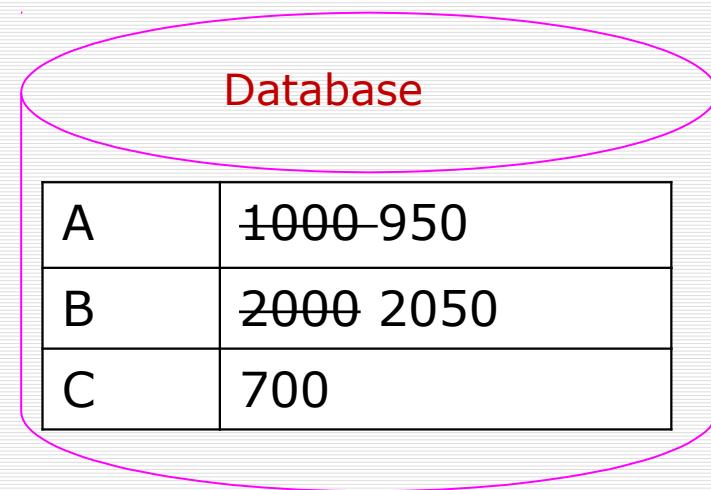


Deferred Database Modification

Question:

If following is the LOG on stable storage after systems crash,
then recovery algorithm should carry out any REDO operation ?

<Start, T1>
<write,T1,A,1000,950>
<write,T1,B,2000,2050>
<Commit, T1>
<Start, T2>
<write,T2,C,700,600>



Answer:

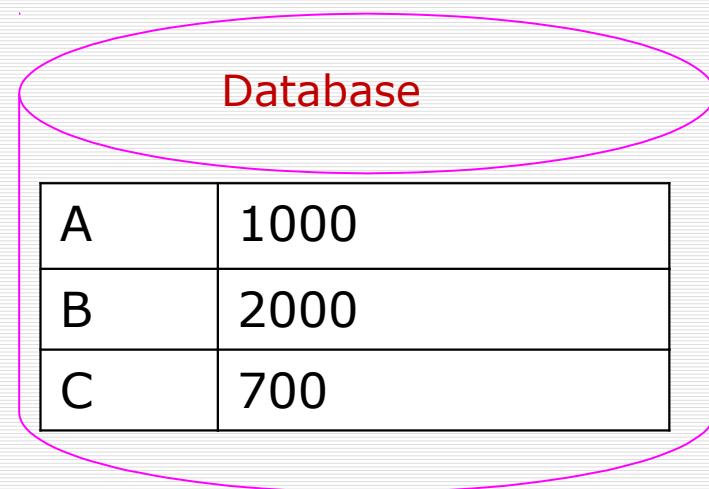
REDO operation for transaction T1 to data items A and B
should be performed because commit LOG entry found for
the transaction T1 <Commit, T1>

Deferred Database Modification

Question:

If following is the LOG on stable storage after systems crash,
then recovery algorithm should carry out any REDO operation ?

<Start, T1>
<write,T1,A,1000,950>
<write,T1,B,2000,2050>
<Commit, T1>
<Start, T2>
<write,T2,C,700,600>
<Commit, T2>

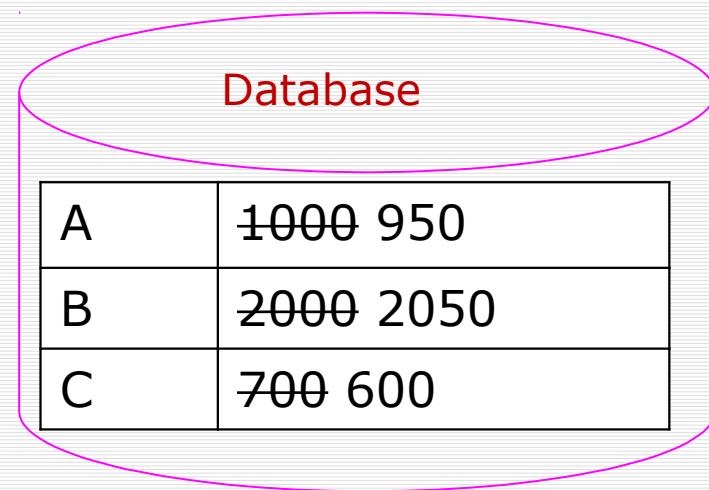


Deferred Database Modification

Question:

If following is the LOG on stable storage after systems crash,
then recovery algorithm should carry out any REDO operation ?

<Start, T1>
<write,T1,A,1000,950>
<write,T1,B,2000,2050>
<Commit, T1>
<Start, T2>
<write,T2,C,700,600>
<Commit, T2>



Answer:

REDO operation for Transaction T1 must be performed followed by the transaction T2 since <Commit, T1> and <Commit, T2> are present.

Immediate Database Modification

- The immediate database modification scheme allows database updates of an uncommitted transaction to be made as the writes are issued
 - since undoing may be needed, update logs must have both old value and new value
- Update log record must be written before database item is written
- We assume that the log record is output directly to stable storage
- Can be extended to postpone log record output, so as long as prior to execution of an output(B) operation for a data block B, all log records corresponding to items B must be flushed to stable storage
- Output of updated blocks can take place at any time before or after transaction commit
- Order in which blocks are output can be different from the order in which they are written.

Immediate Database Modification

- Recovery procedure has two operations instead of one:
 - **undo(T_i)** restores the value of all data items updated by T_i to their **old values**, going backwards from the last log record for T_i
 - **redo(T_i)** sets the value of all data items updated by T_i to the **new values**, going forward from the first log record for T_i
- Both operations must be **idempotent**
 - That is, even if the operation is executed multiple times the effect is the same as if it is executed once
 - Needed since operations may get re-executed during recovery
- When recovering after failure:
 - Transaction T_i needs to be **undone** if the log contains the <start, T_i> record , but **does not contain the <commit, T_i>** record .
 - Transaction T_i needs to be **redone** if the log contains **both the <start, T_i> record and <commit, T_i>** the record .
- Undo operations are performed first, then redo operations.

Immediate Database Modification

Two main categories of Immediate update algorithm

UNDO/No-REDO

- If the recovery technique ensures that all updates of a transaction are recorded in the database on disk before the transaction commits, there is never a need to redo any operations of committed transactions. Such an algorithm is called undo/no-redo.

UNDO/REDO

- On the other hand, if the transaction is allowed to commit before all its changes are written to the database, we have the undo/redo method, the most general recovery algorithm.

Immediate Database Modification

Example of Immediate database recovery

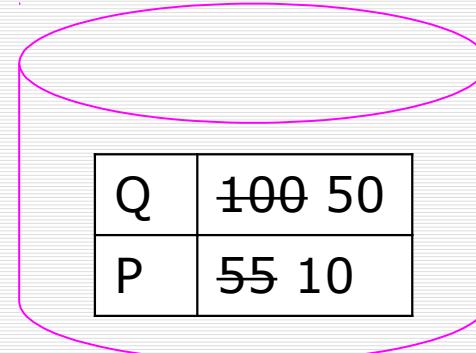
- Consider log entries for two transactions T1 and T2 as follows.

LOG

System
Crash →

(Start, T1)
(Write, T1, Q, 100,50)
(Commit, T1)
(Start, T2)
(Write, T2, P, 55, 10)

Database



Under immediate recovery method, after system crash the algorithm looks at LOG entries for all transactions Ti

- if the entry of both <start, Ti> and <commit, Ti> is present then for all write operations of Ti REDO will be carried out if updates have not been carried out on stable storage of database
- if the entry of only <start, Ti> is present but not <commit, Ti> then for all write operations of Ti UNDO will be carried out

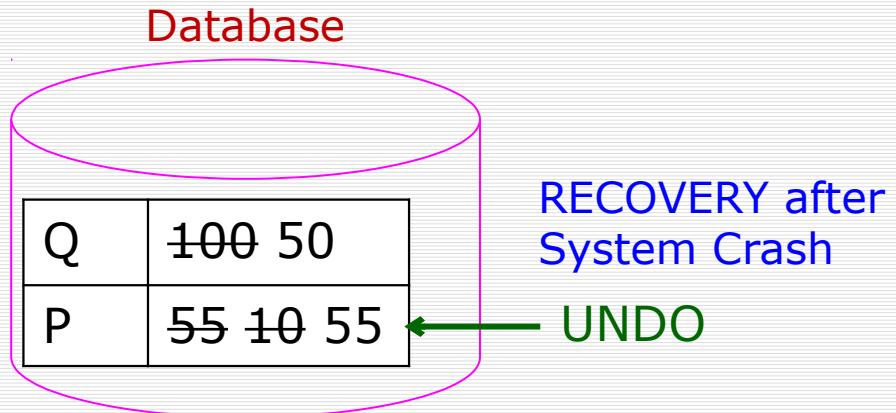
Deferred Database Modification

Example of Deferred based recovery

- Consider log entries for two transactions T1 and T2 as follows.

LOG

(Start, T1)
(Write, T1, Q, 100, 50)
(Commit, T1)
(Start, T2)
(Write, T2, P, 55, 10)
System Crash



Under immediate recovery method, after system crash the algorithm looks at LOG entries for all transactions Ti

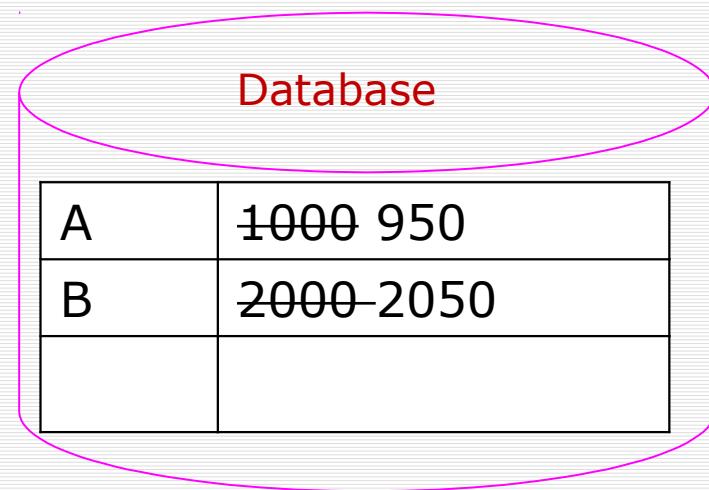
- if the entry of both $\langle \text{start, } T_i \rangle$ and $\langle \text{commit, } T_i \rangle$ is present then for all write operations of T_i REDO will be carried out if updates have not been carried out on stable storage of database
- if the entry of only $\langle \text{start, } T_i \rangle$ is present but not $\langle \text{commit, } T_i \rangle$ then for all write operations of T_i UNDO will be carried out

Immediate Database Modification

Question:

If following is the LOG and database on stable storage after systems crash, then recovery algorithm should carry out any UNDO/REDO operation ?

<Start, T1>
<write,T1,A,1000,950>
<write,T1,B,2000,2050>

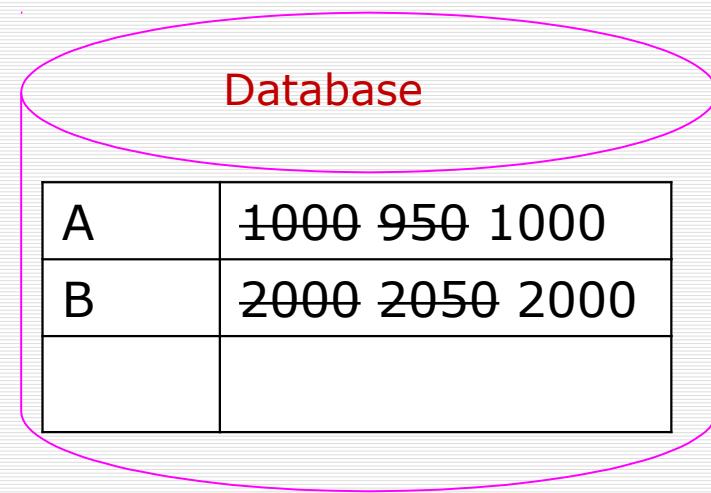


Immediate Database Modification

Question:

If following is the LOG and database on stable storage after systems crash, then recovery algorithm should carry out any UNDO/REDO operation ?

<Start, T1>
<write,T1,A,1000,950>
<write,T1,B,2000,2050>



Answer:

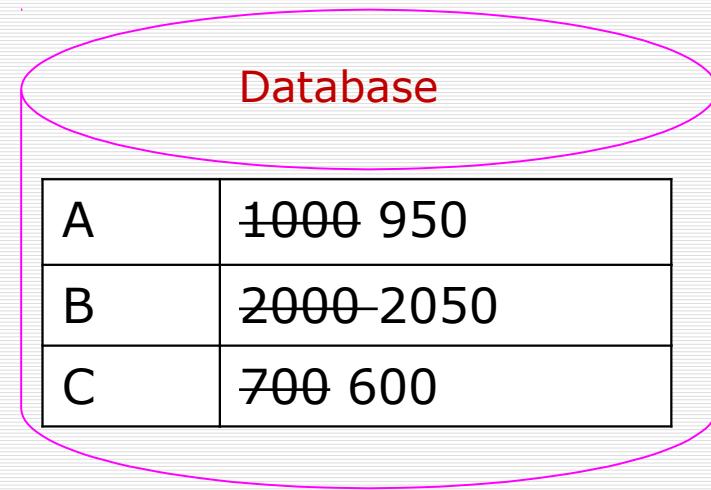
UNDO for T1 because <commit,T1> not found: A should be restored to 1000 and B should be restored to 2000

Immediate Database Modification

Question:

If following is the LOG and database on stable storage after systems crash, then recovery algorithm should carry out any UNDO/REDO operation ?

<Start, T1>
<write,T1,A,1000,950>
<write,T1,B,2000,2050>
<Commit, T1>
<Start, T2>
<write,T2,C,700, 600>

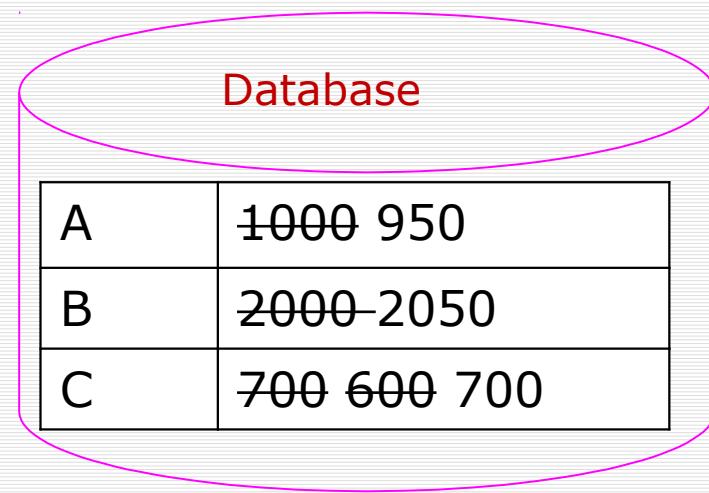


Immediate Database Modification

Question:

If following is the LOG and database on stable storage after systems crash, then recovery algorithm should carry out any UNDO/REDO operation ?

<Start, T1>
<write,T1,A,1000,950>
<write,T1,B,2000,2050>
<Commit, T1>
<Start, T2>
<write,T2,C,700, 600>



Answer:

Recovery actions

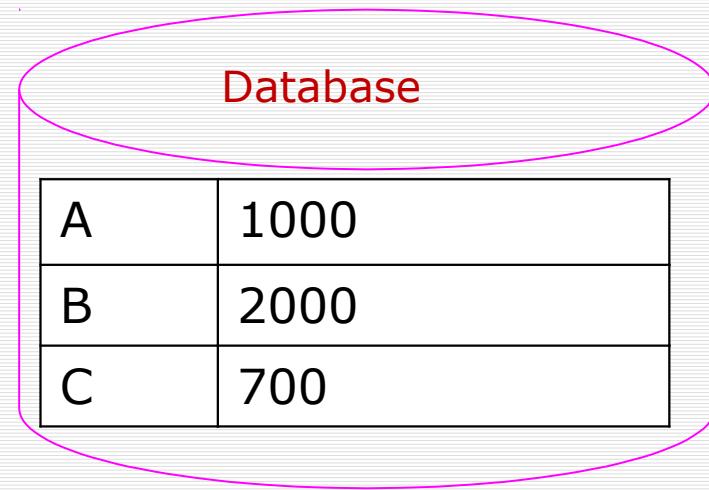
UNDO T2 because <commit, T2> not found: C is restored to 700

Immediate Database Modification

Question:

If following is the LOG and database on stable storage after systems crash, then recovery algorithm should carry out any UNDO/REDO operation ?

<Start, T1>
<write,T1,A,1000,950>
<write,T1,B,2000,2050>
<Commit, T1>
<Start, T2>
<write,T2,C,700, 600>
<Commit, T2>

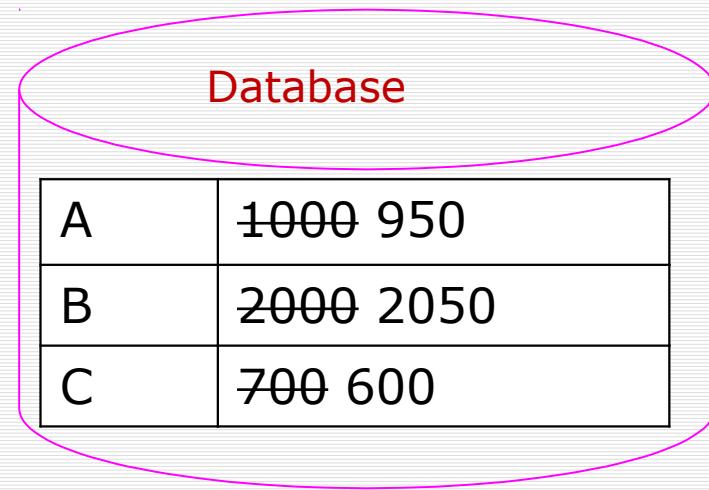


Immediate Database Modification

Question:

If following is the LOG and database on stable storage after systems crash, then recovery algorithm should carry out any UNDO/REDO operation ?

<Start, T1>
<write,T1,A,1000,950>
<write,T1,B,2000,2050>
<Commit, T1>
<Start, T2>
<write,T2,C,700, 600>
<Commit, T2>



Answer:

Recovery actions

REDO operation for Transaction T1 and T2 since <Commit, T1> and <Commit, T2> are present.

Checkpoints

Problems in recovery procedure as discussed earlier :

1. Searching the entire log is time-consuming
2. We might unnecessarily redo transactions which have already output their updates to the database.

Streamline recovery procedure by periodically performing **checkpointing**

1. Output all log records currently residing in main memory onto stable storage.
2. Output all modified buffer blocks to the disk.
3. Write a log record <**checkpoint**> onto stable storage.

Checkpoints

During recovery we need to consider only the most recent transaction T_i that started before the checkpoint, and transactions that started after T_i .

1. Scan backwards from end of log to find the most recent **<checkpoint>** record
2. Continue scanning backwards till a record **<start T_i >** is found.
3. Need only consider the part of log following above **start** record. Earlier part of log can be ignored during recovery, and can be erased whenever desired.
4. For all transactions (starting from T_i or later) with no **<commit T_i >**, execute **undo(T_i)**. (Done only in case of immediate modification.)
5. Scanning forward in the log, for all transactions starting from T_i or later with a **<commit T_i >**, execute **redo(T_i)**.

Recovery using Deferred Update with Concurrent Transactions

Consider the following LOG entries

[start_transaction, T_1]
[write_item, T_1 , D , 20]
[commit, T_1]
[checkpoint]
[start_transaction, T_4]
[write_item, T_4 , B , 15]
[write_item, T_4 , A , 20]
[commit, T_4]
[start_transaction, T_2]
[write_item, T_2 , B , 12]
[start_transaction, T_3]
[write_item, T_3 , A , 30]
[write_item, T_2 , D , 25]

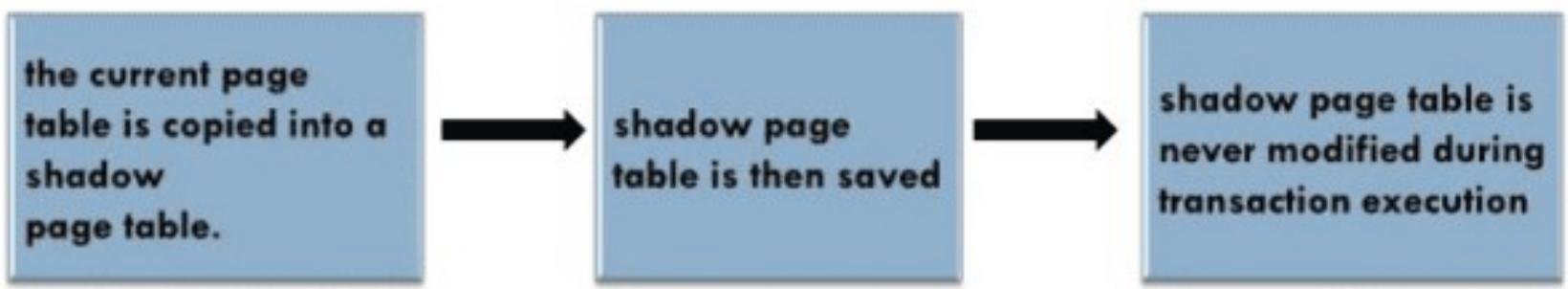
System crash

T_2 and T_3 are ignored because they did not reach their commit points
 T_4 is redone because its commit point is after the last system check point

Shadow Paging: Recovery Scheme that does not require log and useful for single-user environment

- In this technique, the database is considered to be made up of fixed-size disk blocks or pages for recovery purposes.
- Maintains two tables during the lifetime of a transaction-current page table and shadow page table.
- Store the shadow page table in nonvolatile storage, to recover the state of the database prior to transaction execution
- This is a technique for providing atomicity and durability.

When a transaction begins executing



Shadow Paging

Hard Disk

Database table

Page No.	Page/ Sub-table Data	Location
1	Data Item A 1000	Addr1
2	B 2000	Addr2
3	C 700	Addr3
4	D 50	Addr4

Shadow Paging

Hard Disk

Database table

Page No.	Page/ Sub-table Data	Location
1	Data Item A 1000	Addr1
2	B 2000	Addr2
3	C 700	Addr3
4	D 50	Addr4

Transaction operations

```
(Start,T1)
(Write,T1,A,1000,950)
(Write,T1,B,2000,2050)
(Commit,T1)
(Start, T2)
(Write,T2,C,700,600)
(Write,T2,D, 50,30)
```

Shadow Paging

Hard Disk

Main Memory

Current Page Directory

Page 1: Addr1
Page 2: Addr2
Page 3: Addr3
Page 4: Addr4

Database table

Page No.	Page/ Sub-table Data	Location
1	Data Item A 1000	Addr1
2	B 2000	Addr2
3	C 700	Addr3
4	D 50	Addr4

Transaction operations

(Start,T1)
(Write,T1,A,1000,950)
(Write,T1,B,2000,2050)
(Commit,T1)
(Start, T2)
(Write,T2,C,700,600)
(Write,T2,D,50,30)

When transaction starts
Current Page directory is
kept in main memory and
Shadow directory on disk

Shadow Page Directory

Page 1: Addr1
Page 2: Addr2
Page 3: Addr3
Page 4: Addr4

Shadow Paging

Hard Disk

Main Memory

Current Page Directory

Page 1: Addr1 **Addr5**
Page 2: Addr2
Page 3: Addr3
Page 4: Addr4

Transaction operations

(Start,T1)
(Write,T1,A,1000,950)
(Write,T1,B,2000,2050)
(Commit,T1)
(Start, T2)
(Write,T2,C,700,600)
(Write,T2,D,50,30)

After execution of
(write,T1,A,1000,950)
Then new page table
is created on disk and
Current page
Directory updated.
But shadow directory will
not be updated

Database table

Page No.	Page/ Sub-table Data	Location
1	Data Item A 1000	Addr1
2	B 2000	Addr2
3	C 700	Addr3
4	D 50	Addr4
5	A 950	Addr5

Shadow Page Directory

Page 1: Addr1
Page 2: Addr2
Page 3: Addr3
Page 4: Addr4

Shadow Paging

Hard Disk

Main Memory

Current Page Directory

Page 1: Addr1 Addr5
Page 2: Addr2 **Addr6**
Page 3: Addr3
Page 4: Addr4

Transaction operations

(Start,T1)
(Write,T1,A,1000,950)
(Write,T1,B,2000,2050)
(Commit,T1)
(Start, T2)
(Write,T2,C,700,600)
(Write,T2,D,50,30)

After execution of
(write,T2,2000,2050)
Then new page table
is created on disk and
Current page
Directory updated.
But shadow directory will
not be updated

Database table

Page No.	Page/ Sub-table Data	Location
1	Data Item A 1000	Addr1
2	B 2000	Addr2
3	C 700	Addr3
4	D 50	Addr4
5	A 950	Addr5
6	B 2050	Addr6

Shadow Page Directory

Page 1: Addr1
Page 2: Addr2
Page 3: Addr3
Page 4: Addr4

Shadow Paging

Hard Disk

Main Memory

Current Page Directory

Page 1: Addr1 Addr5
Page 2: Addr2 Addr6
Page 3: Addr3
Page 4: Addr4

Transaction operations

(Start,T1)
(Write,T1,A,1000,950)
(Write,T1,B,2000,2050)
(Commit,T1)
(Start, T2)
(Write,T2,C,700,600)
(Write,T2,D,50,30)

After execution of
(commit,T1)
Shadow directory will
be updated and old page
On the hard disk will be
discarded

Database table

Page No.	Page/ Sub-table Data	Location
1	Data Item A 1000	Addr1
2	B 2000	Addr2
3	C 700	Addr3
4	D 50	Addr4
5	A 950	Addr5
6	B 2050	Addr6

Shadow Page Directory

Page 1: Addr1 **Addr5**
Page 2: Addr2 **Addr6**
Page 3: Addr3
Page 4: Addr4

Shadow Paging

To recover from a failure

execution is available through the shadow page table



discard current page table



the shadow page table to become the current page table

Advantages

- No-redo/no-undo

Disadvantages

- Creating shadow directory may take a long time.
- Updated database pages change locations.
- Garbage collection is needed

“ARIES” recovery algorithm

Recovery algorithms are techniques to ensure database consistency ,transaction atomicity and durability without any failure.

- **Recovery algorithms have two parts**
 1. Actions taken during normal transaction processing to ensure enough information exists to recover from failures.
 2. Actions taken after a failure to recover the database contents to a state that ensures atomicity, consistency and durability.

“ARIES” recovery algorithm

- **ARIES (Algorithms for Recovery and Isolation Exploiting Semantics)**

- **The ARIES recovery algorithm consist of three steps**
 - Analysis
 - Redo
 - Undo

“ARIES” recovery algorithm

- **Analysis** - Identify the dirty pages(updated pages) in the buffer and set of active transactions at the time of failure.
- **Redo** - Re-apply updates from the log to the database. It will be done for the committed transactions.
- **Undo** - Scan the log backward and undo the actions of the active transactions in the reverse order.

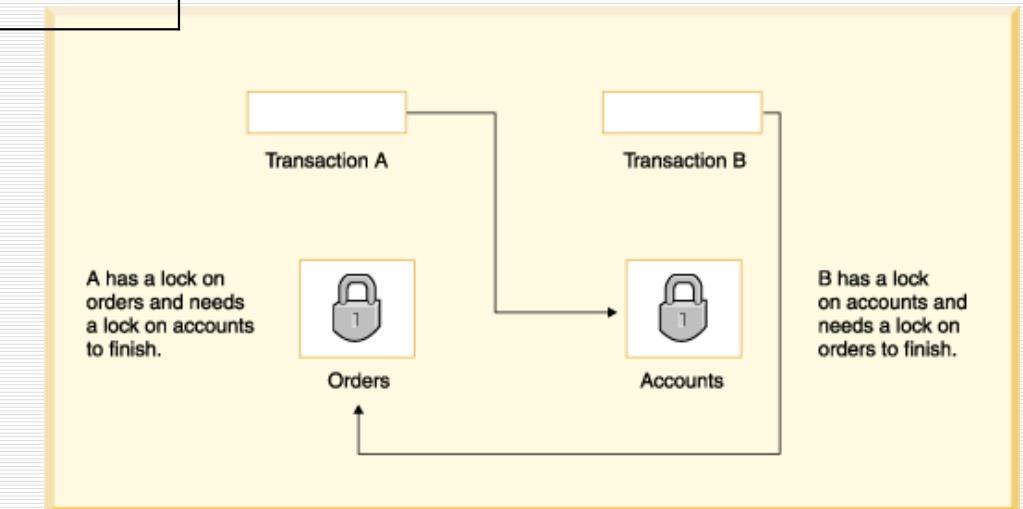
Next we will learn Deadlocks

Deadlocks occur when each transaction T in a set of two or more transactions is waiting for some item that is locked by some other transaction T' in the set.

Illustrating Deadlock Problem

Transaction A	Transaction B
Read_lock(orders)	
Read_item(orders)	
	Read_lock(accounts)
	Read_item(accounts)
Write_lock(accounts)) (wait for accounts)	
	Write_lock(orders) (wait for orders)

Deadlock between
Transaction A and B



Approaches for dealing with deadlocks

1. Deadlock Prevention Protocols

1.1. Based on Timestamp

1.1.1. Wait-Die

1.1.2. Wound-Wait

1.2 Without using timestamp

1.2.1. No-waiting

1.2.2. Cautious waiting

2. Deadlock Detection Protocols

2.1. Using wait-for graph

3. Timeouts

Deadlock Prevention Protocols

Based on Timestamp: 1. Wait-Die 2. Wound-Wait

Suppose T_i tries to lock an item X but is not able because X is locked by some other transaction T_j .

1. wait-die:

- If $TS(T_i) < TS(T_j)$, then (T_i older than T_j), T_i is allowed to wait
- Otherwise (T_i younger than T_j) abort & rollback T_i and restart it using the same timestamp

4:00pm, Older Transaction To	4:05pm, Younger transaction Ty
Then To waits	If Ty holds X
If To holds X	Then Ty is discarded (die)

2. wound-wait:

- If $TS(T_i) < TS(T_j)$, then (T_i older than T_j), abort & rollback T_j , and restart it using the same timestamp
- Otherwise (T_i younger than T_j), T_i is allowed to wait

4:00pm, Older Transaction To	4:05pm, Younger transaction Ty
If To holds X	Then Ty waits
Then To pre-empts (wounds) Ty	If Ty holds X

Deadlock Prevention protocol

Without using Timestamp

1. No-Waiting: If transaction cannot get lock it assumes deadlock. So it dies, waits and restarts later

2. Cautious Waiting

If transaction cannot get lock, it check the lock holder

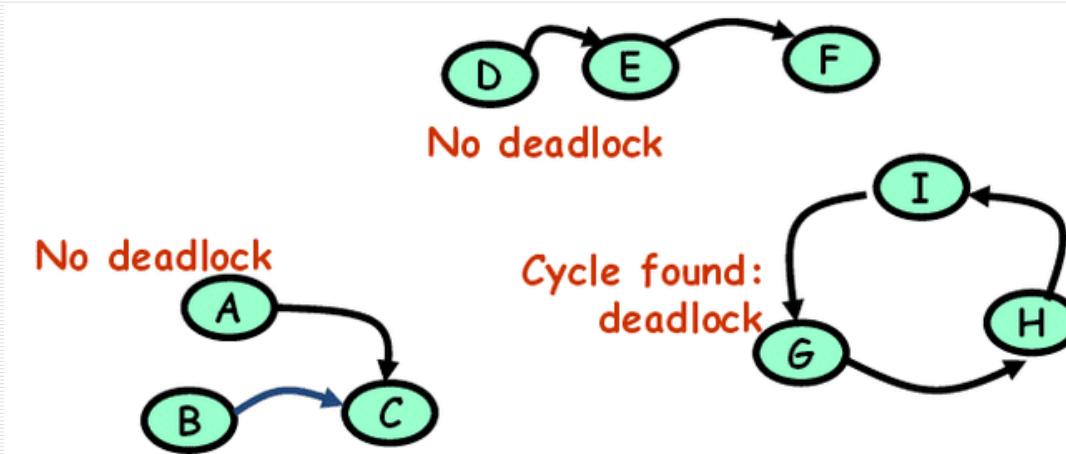
- If lock holder is already waiting then transaction dies and restarts later
- If Lock holder is active then transaction waits

Deadlock Prevention protocol

- **Timeouts:** If a transaction waits for a period longer than a **system-defined timeout period**, the system assumes that the transaction may deadlocked and aborts it – regardless of whether a deadlock actually exists or not

Deadlock Detection protocols

- In this approach, deadlocks are allowed to happen. The scheduler maintains a **wait-for-graph** for detecting cycle. **If a cycle exists**, then one transaction involved in the cycle is selected (victim) and rolled-back.
- A wait-for-graph is created using the lock table. As soon as a transaction is blocked, it is added to the graph. When a chain like: T_i waits for T_j waits for T_k waits for T_i or T_j occurs, then this creates a cycle. One of the transaction of the cycle is selected and rolled back.



Starvation

- Starvation occurs when a particular transaction consistently waits or restarted and never gets a chance to proceed further.
- In a deadlock resolution it is possible that the same transaction may consistently be selected as victim and rolled-back.
- This limitation is inherent in all priority based scheduling mechanisms.
- In Wound-Wait scheme a younger transaction may always be wounded (aborted) by a long running older transaction which may create starvation.
- The algorithm can use higher priorities for transactions that have been aborted multiple times to avoid starvation problem.

Approaches for dealing with deadlocks

1. Deadlock Prevention Protocols

1.1. Based on Timestamp

1.1.1. Wait-Die

1.1.2. Wound-Wait

1.2 Without using timestamp

1.2.1. No-waiting

1.2.2. Cautious waiting

2. Deadlock Detection Protocols

2.1. Using wait-for graph

3. Timeouts

Thanks for Listening
