CSPC404 - DATABASE MANAGEMENT SYSTEMS

Dr. L.R. Sudha
Associate Professor/CSE

Introduction

- **DBMS** = database + management system
  = collection of interrelated data + Set of programs to manage database

- **Defn:** A database-management system is a collection of interrelated data and a set of programs to manage those data.

  Management means - Definition of database
  Construction of database
  Manipulation of database

- **Goal of a DBMS**

  To provide a convenient and efficient way to store and retrieve database information

- **Examples of DBMS** : Oracle, MySQL, SQL Server, DB2 …

Database Applications

- Banking: customer information, accounts, loans, all transactions
- Airlines, railways, bus: reservations, schedules
- Universities: student registration, faculty information, grades
- Inventories: customers, products, orders, stock
- Manufacturing: production, inventory, orders
- Human resources: employee records, salaries, tax deductions
- Social-media: records of users, connections between users (such as friend/follows information), posts made by users, rating/like information about posts, etc.

Course Objectives

- To understand the fundamentals of DBMS and E-R Diagrams.
- To impart the concepts of the Relational model and SQL.
- To disseminate the knowledge on various Normal Forms.
- To inculcate the fundamentals of transaction management and Query processing.
- To familiarize on the current trends in data base technologies

Course Outcomes

1. Understand the fundamental concepts of Database Management Systems and Entity Relationship Model and develop ER Models.

2. Build SQL Queries to perform data creation and data manipulation operations on databases.

3. Understand the concepts of functional dependencies, normalization and apply such knowledge to the normalization of a database.

4. Identify the issues related to Query processing and Transaction management in database management systems.

5. Analyze the trends in data storage, query processing and concurrency control of modern database technologies.

Mapping of Course Outcomes with Programme Outcomes												
	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1	2	-	2	-	-	-	-	-	-	-	-	-
CO2	2	-	2	-	-	-	-	-	-	-	-	-
CO3	-	-	1	-	-	-	-	-	-	-	-	-
CO4	-	1	-	-	1	-	-	-	-	-	-	-
CO5	2	-	-	-	-	-	-	-	-	-	-	-

Importance of DBMS subject

- One of the subjects in GATE exam.

- If you Qualify GATE
  - ➢ Post Graduation from most reputed engineering colleges in India such as IIT/NIT
  - ➢ Eligible for Rs.12400/- stipend every month for the next two years during your post-graduation

  - ➢ Numerous PSU such as BHEL, GAIL, NLC, NTPC, BPCL, etc., are using the GATE score for choosing candidates for their organizations.

UNIT – I : Introduction

- File System vs. DBMS
- Views of data
- Data Models
- Database Languages
- Database Management System Services
- Overall System Architecture
- Data Dictionary
- Entity – Relationship (E-R)
- Enhanced Entity Relationship Model.

1. File System vs. DBMS

- Earlier days database is managed by file system.
- Instead of using files why DBMS is used?

Disadvantages in File Processing

1. Data redundancy and inconsistency.
2. Difficult in accessing data.
3. Data isolation.
4. Data integrity.
5. Atomicity problem
6. Concurrent access is not possible.
7. Security Problems.

```c
int main()
{
   char name[50];
   int marks, i, num;

   printf("Enter number of students: ");
   scanf("%d", &num);

   FILE *fptr;
   fptr = (fopen("C:\\student.txt", "w"));
   if(fptr == NULL)
   {
       printf("Error!");
       exit(1);
   }

   for(i = 0; i < num; ++i)
   {
      printf("For student%d\nEnter name: ", i+1);
      scanf("%s", name);

      printf("Enter marks: ");
      scanf("%d", &marks);

      fprintf(fptr,"\nName: %s \nMarks=%d \n", name, marks);
   }

   fclose(fptr);
   return 0;
}
```

- SQL – Structured Query Language
- CREATE TABLE statement is used to create a new table in the database.
- To create a table, you have to
  - name of table
  - define its columns
  - Define datatype of each column.
-

```
CREATE TABLE table_name
(
  column1 datatype [ NULL | NOT NULL ],
  column2 datatype [ NULL | NOT NULL ],
  ...
  column_n datatype [ NULL | NOT NULL ]
);
```

```
CREATE TABLE customers
( customer_id number(10) NOT NULL,
  customer_name varchar2(50) NOT NULL,
  city varchar2(50)
);
```

1.Data redundancy and inconsistency

- Redundancy - Same data in many places, multiple copies of same data

Example

  - ➢ Consider a database in University with sections Academics, Hostel, Accounts, Result
  - ➢ Name, Roll No, Phone number of a student may appear in all files

- This redundancy leads to higher storage and access cost. In addition, it may lead to data inconsistency
    - ➢ various copies of the same data may not match

Example

  - ➢ a changed phone number may be reflected in Hostel file but not elsewhere in the system.

2.Difficulty in accessing data

- Conventional file-processing environments do not allow needed data to be retrieved in a convenient and efficient manner.
- It needs application programs to access data.

Example

- Suppose that one of the university clerks needs to find out the names of all students who live within a particular postal-code area.
- Ask a programmer to write the necessary application program.
- Days later, the same clerk needs a list of students who have taken at least 60 credit hours.
- Once again the clerk will ask a programmer to write the necessary application program.
- So as time goes by, system acquires more application programs.

3. Data isolation

- Because data are scattered in various files, and files may be in different formats

4. Integrity problems

- The data values stored in the database must satisfy certain types of consistency constraints.

Example

In university system,

Register No of students must be unique

Address – not null

In Banking System,

Account balance of a customer never fall below minimum balance.

5. Atomicity problems (ALL or Nothing)

- A computer system, like any other device, is subject to failure.
- In many applications, it is crucial that, if a failure occurs, the data be restored to the consistent state that existed prior to the failure.

Example

- Consider a banking system with a program to transfer Rs.5000 from account *A* to account *B*.
- If a system failure occurs during the execution of the program, it is possible that the Rs.5000 was debited from the balance of account *A* but was not credited to the balance of account *B*
- This results in an inconsistent database state.
- So, the funds transfer must be *atomic*—it must happen in its entirety or not at all.
- It is difficult to ensure atomicity in a conventional file-processing system.

6. Concurrent-access anomalies – multiple access at the same time

- Suppose a registration program maintains a count of students registered for a course in order to enforce limits on the number of students registered.

- When a student registers, the program reads the current count for the courses, verifies that the count is not already at the limit, adds one to the count.

- Suppose two students register concurrently, with the count at 39. The two program executions may both read the value 39, and both would then write back 40, leading to an incorrect increase of only 1, instead of 2.

- Furthermore, suppose the course registration limit was 40; in the above case both students would be registered, leading to a violation of the limit of 40 students.

7. Security problems

- Data should be secured from unauthorized access

- Not every user of the database system should be able to access all the data.

- For example, in a university,
  - Accounts section staff should not be able to access academic records.
  - a student should not be able to see the payroll details.

- Enforcing such security constraints is difficult.

	FILE SYSTEM	DBMS
1.	Any management with the file system, user has to write the application programs	Not required to write the application programs for managing the database.
2.	File system gives the details of the data representation and Storage of data.	2. DBMS provides an abstract view of data that hides the details
3.	In File system storing and retrieving of data cannot be done efficiently.	3. DBMS is efficient to store and retrieve the data.
4.	Redundant data can be present	No redundant data
5.	Less data consistency	More data consistency
6.	Concurrent access to the data in the file system has many problems.	4. DBMS takes care of Concurrent access using someform of locking.
7.	File system doesn't provide crash recovery mechanism.	5. DBMS has crash recovery mechanism

2. VIEWS OF DATA

- To study the terminology and basic concepts that are used in DBMS.

  - **Data Models**
  - **Data Abstraction**
  - **Instances and Schemas**

- Main purpose of a database system is to provide users with an *abstract* view of the data.
- That is, the system hides certain details of how the data are stored and maintained.

Data Models

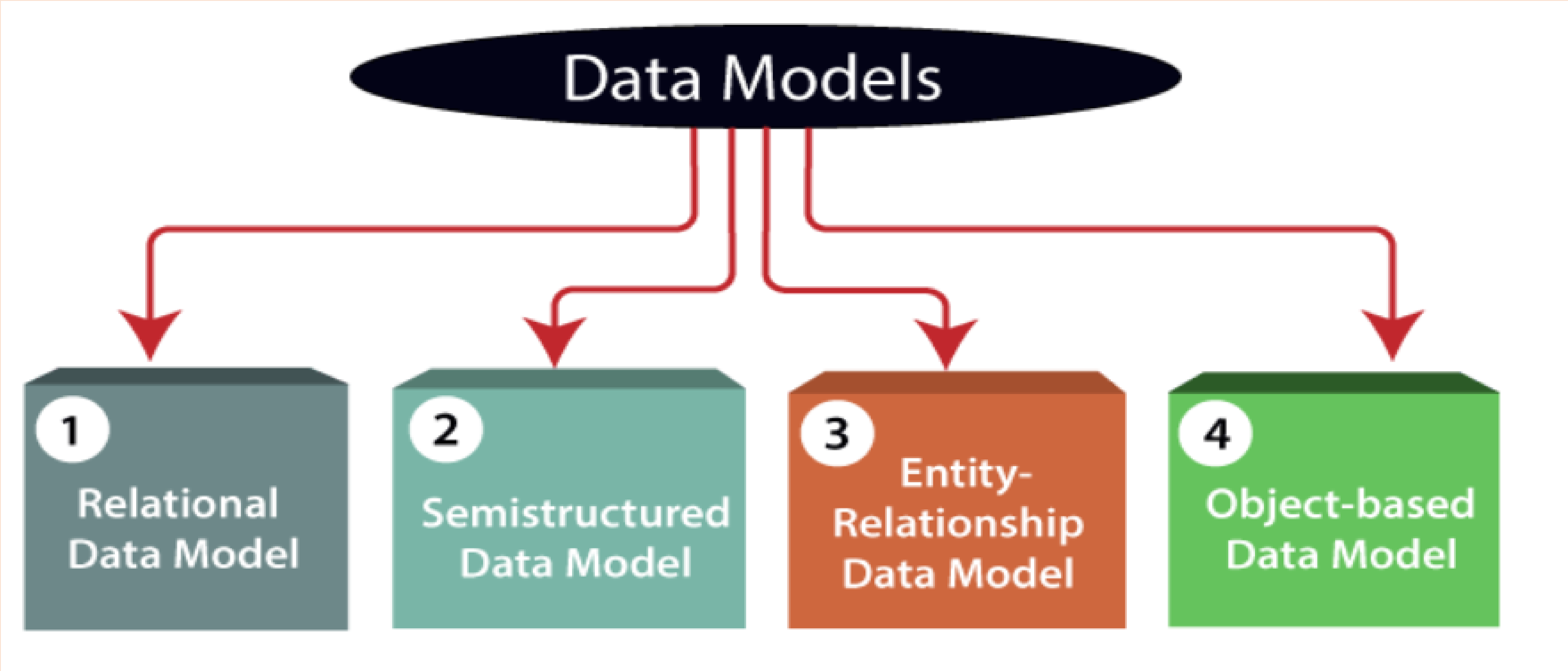
- Collection of conceptual tools used to describe the structure of a database.

- By structure of a database we mean data, relationships and constraints that apply to the data.

- Some data models also include
  - Set of **basic operations** such as insert, delete, modify, retrieve.
  - **Dynamic aspect** or **behavior** of a database application.

- **behavior -** Specify a set of valid user-defined operations that are allowed on the database objects - (fundamental to object-oriented data models)

• 4 Categories



1. Relational Model

- The relational model uses a collection of tables to represent both data and the relationships among those data.
- Each table has multiple columns and rows.
- Each column has a unique name called as fields, or attributes.
- Each row is called as record/tuple
- Row is in fixed format
  - records of a particular type

    (or)
  - fixed number of fields, or attributes.
- Tables are also known as relations.
- The relational data model is the most widely used data model.

Emp_id	Emp_name	Job_name	Salary	Mobile_no	Dep_id
AfterA001	John	Engineer	100000	9111037890	2
AfterA002	Adam	Analyst	50000	9587569214	3
AfterA003	Kande	Manager	890000	7895212355	2

2. Entity-Relationship Model

- The entity-relationship (E-R) data model describes data as entity, attribute and *relationships* .

- Entity is a real-world thing or object.
- It can be an object

  with a physical existence - a person, car, house, employee …

  or

  with a conceptual existence - an account, a job, course…

- Attributes - describes the property of an entity
- Each attribute will have a value

Example: name, street_address, city --- customer database

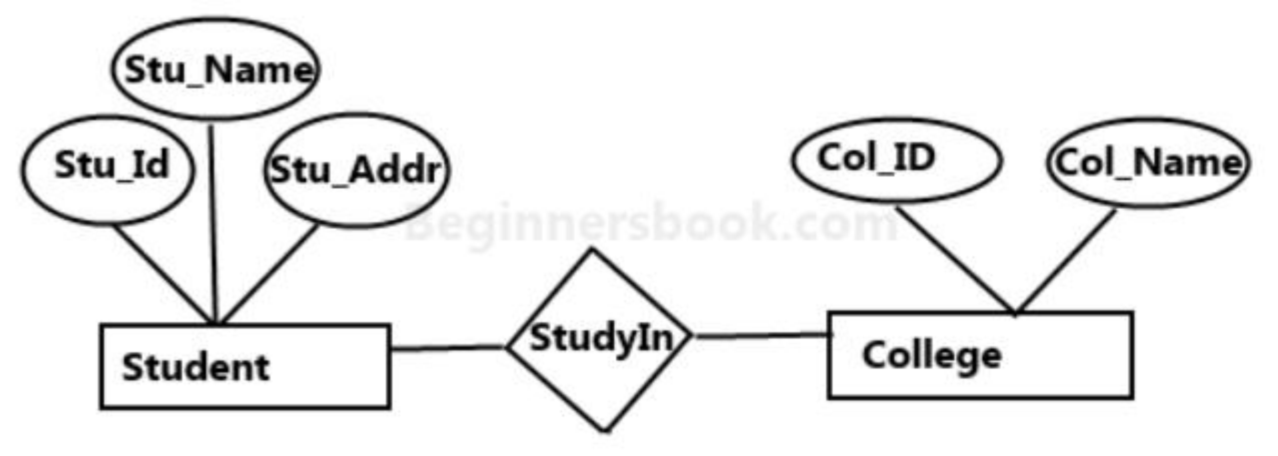
Acc-no, balance --- account database

name,designation,age,bpay ----- employee database

- Relationship - Relationship tells how two entities are related.

Association between entities

Example: student study in a college.



3. Semi-structured Data Model

- This type of data model is different from the other three data models
- This model gives flexibility in storing the data.
- The data is not constrained by a fixed schema

  - In this model entities may or may not have the same attributes or properties
  - Some entities may have missing attributes while others may have an extra attribute.
  - Size and type of the same attributes in a group may differ

- The Extensible Markup Language, XML, is used for representing the semistructured data.

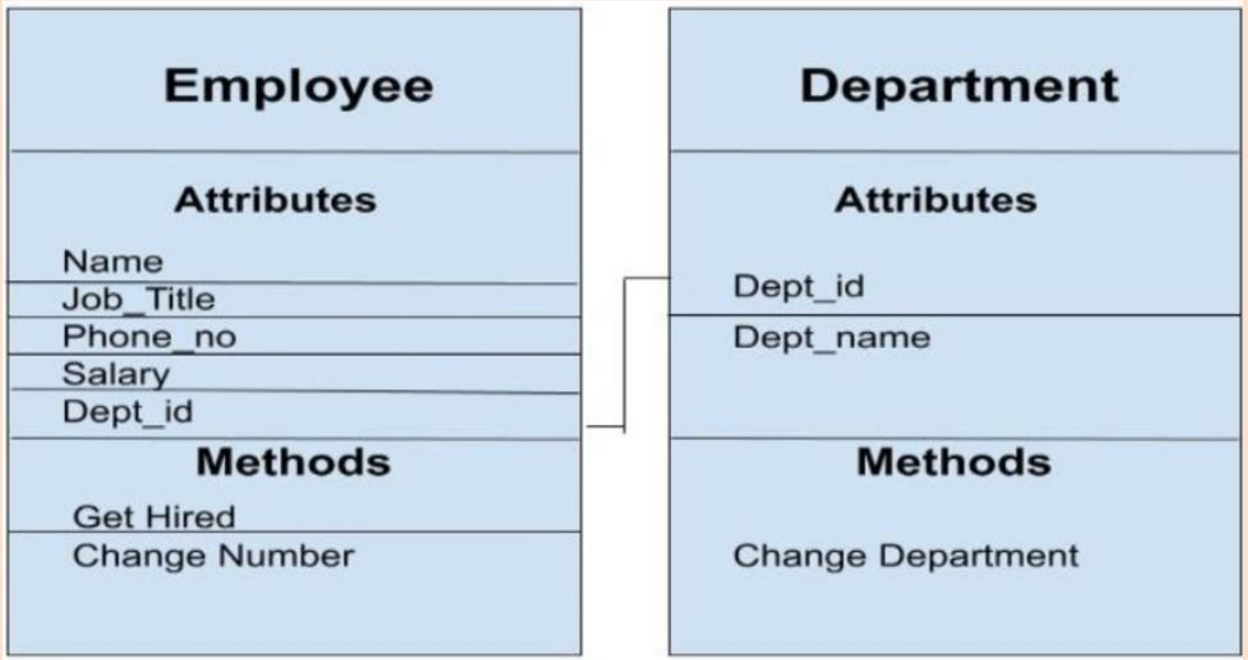
```xml
<course>
    <course_id> CS-101 </course_id>
    <title> Intro. to Computer Science </title>
    <dept_name> Comp. Sci. </dept_name>
    <credits> 4 </credits>
</course>
```

4. Object-Based Data Model

- After the development of object oriented programming, object based data model was developed.
- Concept of objects is integrated into relational databases.
- So this model can be seen as an extension of relational model.
- Allow procedures/methods to be stored in the database system and executed by the database system.
- This can be seen as extending the relational model with encapsulation concept.
- In this model, two are more objects are connected through links.

Example

- We have two objects Employee and Department.
- The attributes like Name, Job_title of the employee and the methods which will be performed by that object are stored as a single object.
- The two objects are connected through a common attribute i.e the Dept_id



Data Abstraction

- **Data abstraction** generally refers to
  - the suppression/hiding of details of data organization and storage
  - and highlighting of essential features.

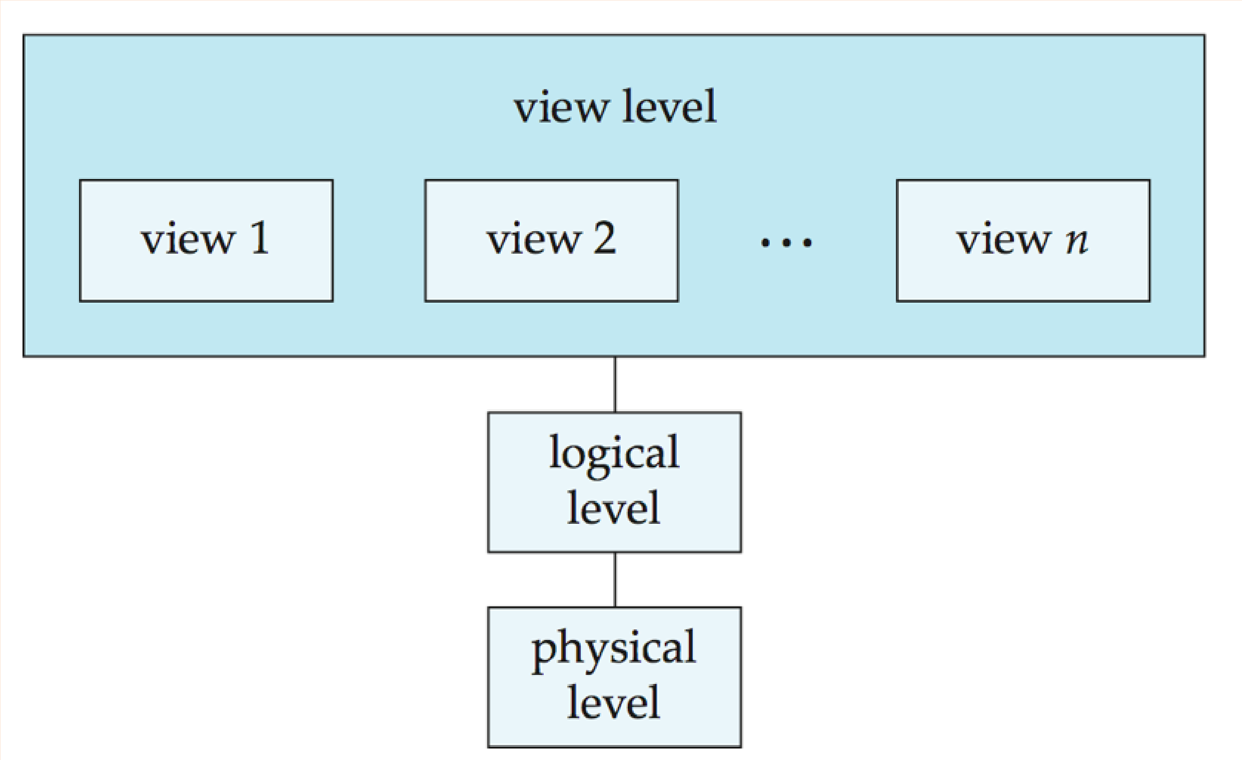
- One of the main characteristics of the database approach.

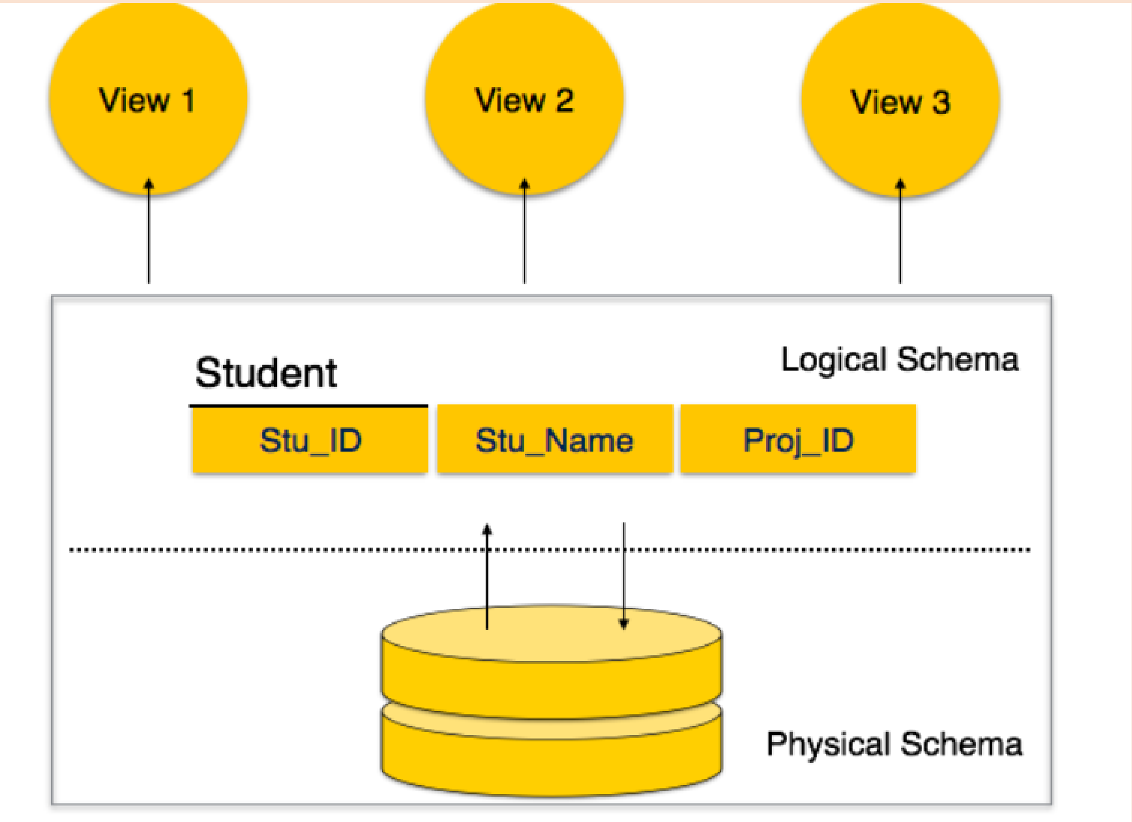
- Use - different users can perceive data at their preferred level of details.

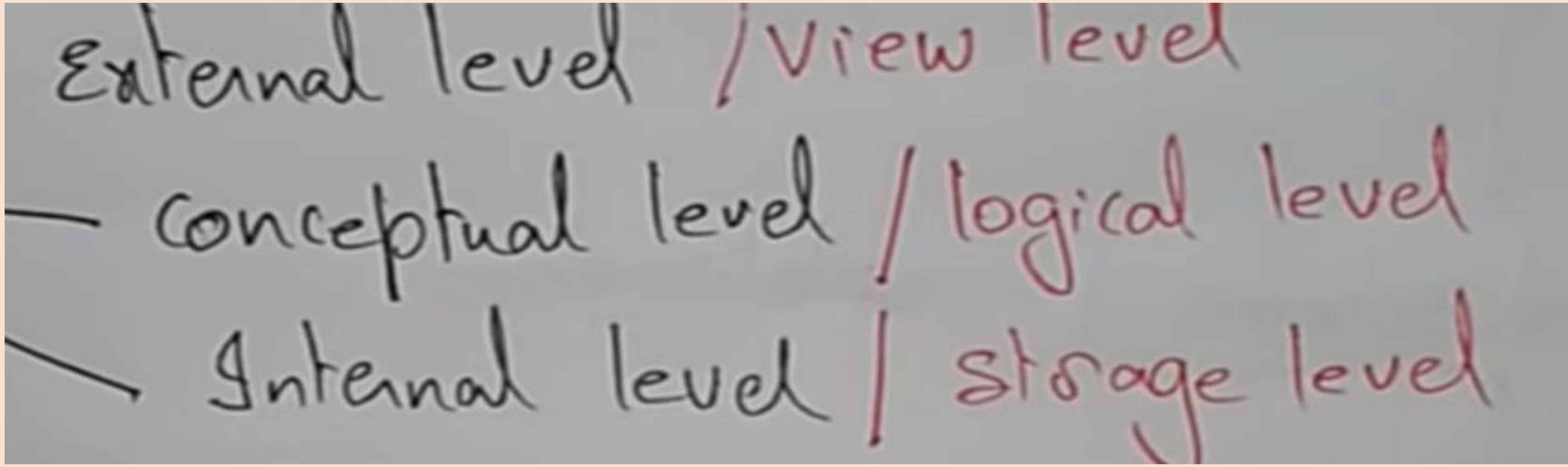
- Database system developers use complex data structures to represent data in the database.

- Since many database-system users are not computer trained, developers hide the complexity from users through several levels of data abstraction

- 3 levels of data abstraction







- Physical level - The lowest level of abstraction describes *how* the data are actually stored.
- Logical level - The next-higher level of abstraction describes *what* data are stored in the database, and what relationships exist among those data.

```
type instructor = record
        ID : string;
        name : string;
        dept_name : string;
        salary : integer;
    end;
```

- View level - The highest level of abstraction describes only part of the entire database

Schemas, Instances

- Database schema describes the overall design of a database.
- It is not expected to change frequently.
- Most data models have certain conventions for displaying schemas as diagrams.
- These diagrams are called as schema diagram.

Example

schema diagram of relational model.

STUDENT

Name	Student_number	Class	Major

COURSE

Course_name	Course_number	Credit_hours	Department

- The collection of information stored in the database at a particular moment is called an instance of the database.

- The concept of database schemas and instances can be understood by comparing with a program written in a programming language.

- A database schema corresponds to the variable declarations in a program.

- An *instance* of a database schema corresponds to the values of the variables in a program at a point of time.

- Each variable has a particular value at a given instant.

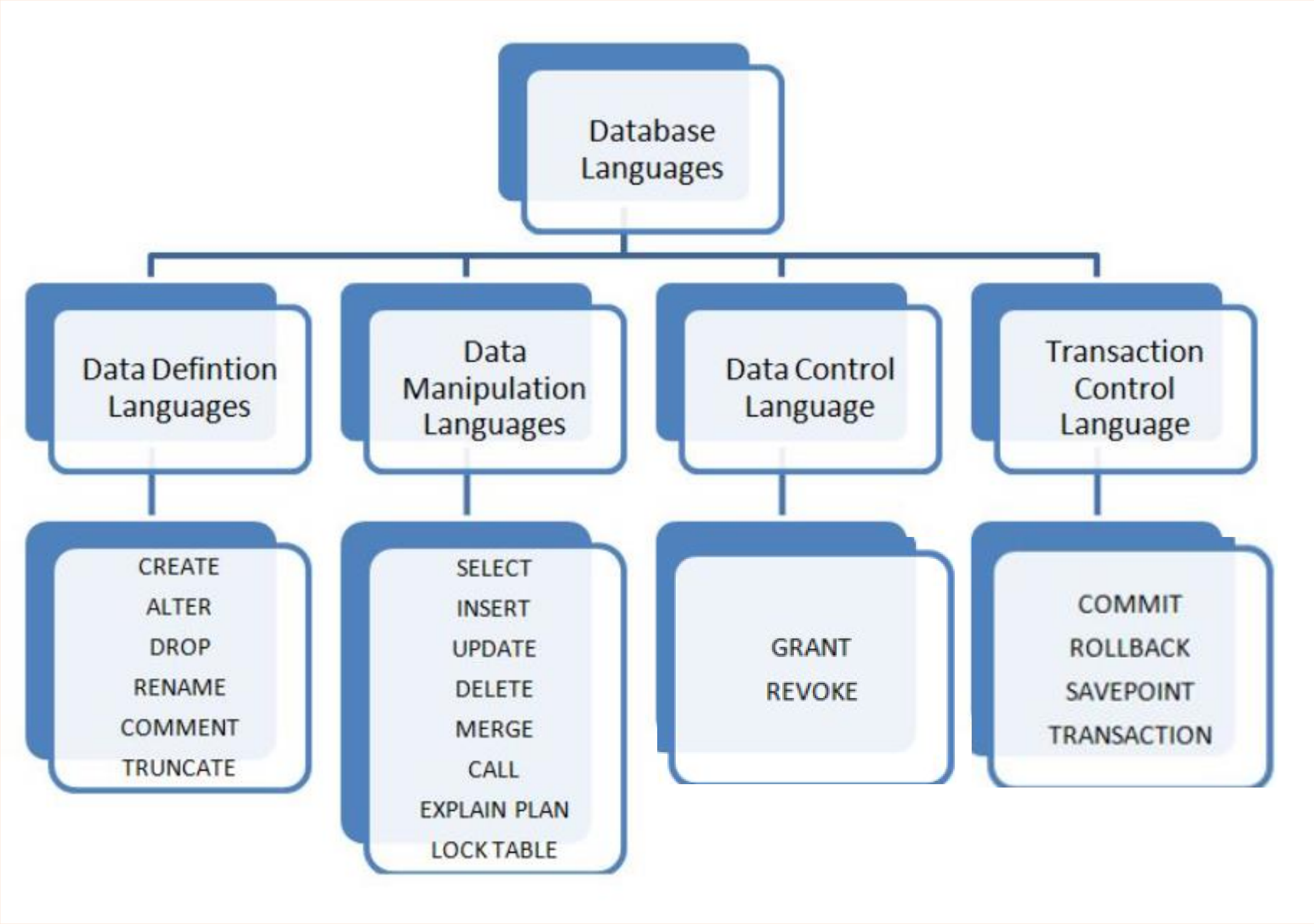
Types of Schema

1. physical schema describes the database design at the physical level,
2. logical schema describes the database design at the logical level.
3. subschemas describes different views of the database

- Of these, the logical schema is most important since programmers construct applications by using the logical schema.

- The physical schema is hidden beneath the logical schema and can be changed easily without affecting application programs.

3. Database Languages



- DDL is used to specify the database schema
- DML is used to access or manipulate data in the database
- DCL is used to control access privilege in Databases.
- TCL is used to manage the changes made by DML statements

Data Definition Language (DDL)

- DDL is used for specifying the database schema. – structure/skeleton of the database
- It is used for creating tables, indexes, constraints etc. in database.
- Processing of DDL statements, just like those of any other programming language, generates some output.
- The output of the DDL is placed in a special file called as data dictionary, which contains metadata—that is, data about data.

Example : number of schemas and tables, their names, constraints, columns in each table, etc.

- The data dictionary can be accessed and updated only by the database system itself (not a regular user).

- The database system consults the data dictionary before reading or modifying actual data.

Data Definition Languages (DDL) Commands:

- **Create:** To create a new table or a new database.
- **Alter:** To alter or change the structure of the database table.
- **Drop:** To delete a table, index, or views from the database.
- **Truncate:** To delete the records or data from the table, but its structure remains as it is.
- **Rename:** To rename an object from the database.
- **Comment:** To add comments in a table.

Consistency/Integrity Constraints

- Integrity constraints are a set of rules. It is used to maintain the quality of information.
- The data values stored in the database must satisfy certain consistency constraints.
- The DDL provides facilities to specify such constraints.
- The database system checks these constraints every time the database is updated.

Example :

account balance of a department must never be negative.

Types:

1. Domain Constraints
2. Entity Integrity Constraints
3. Referential Integrity Constraints
4. Authorization

1. Domain Constraints

- A domain of possible values must be associated with every attribute by restricting the type, the format, or the range of values.
- Valid set of values for an attribute.
- Domain constraints are the most elementary form of integrity constraint.
- They are tested easily by the system whenever a new data item is entered into the database.

ID	NAME	SEMENSTER	AGE
1000	Tom	1st	17
1001	Johnson	2nd	24
1002	Leonardo	5th	21
1003	Kate	3rd	19
1004	Morgan	8th	A

Not allowed. Because AGE is an integer attribute

2. Entity Integrity Constraints - Unique, primary key

- Ensures uniqueness of each record or row in the data table.
- No duplicate rows should be in a table.
- There must be a value in the primary key field. It should not be Null
- This is because the primary key value is used to identify individual rows in a table.
- If there were null values for primary keys, it would mean that we could not identify those rows.

EMPLOYEE

EMP_ID	EMP_NAME	SALARY
123	Jack	30000
142	Harry	60000
164	John	20000
	Jackson	27000

Not allowed as primary key can't contain a NULL value

3. Referential Integrity

- A referential integrity constraint is specified between two tables.
- It is to ensure valid relationship between two tables.
- Referential integrity is combination of a primary key and a foreign key.
- The main concept of REFERENTIAL INTEGRITY is that it does not allow to add any record in a table that contains the foreign key unless the reference table containing a corresponding primary key value.

- In the Referential integrity constraints, if a foreign key in Table 1 refers to the Primary Key of Table 2, then every value of the Foreign Key in Table 1 must be available in Table 2.

Roll_No	Name
1	Ishant
2	Aditya
3	Gautam

Link Broken (as 4 is not present in the parent table still 4 is present in the child table hence voilating the Referential Integrity rule)

Roll_No	Subject	Marks
3	Maths	90
4	Eng	85
3	Science	87

(Table 1)

EMP_NAME	NAME	AGE	D_No
1	Jack	20	11
2	Harry	40	24
3	John	27	18
4	Devil	38	13

Foreign key

Not allowed as D_No 18 is not defined as a Primary key of table 2 and In table 1, D_No is a foreign key defined

Relationships

(Table 2)

Primary Key

D_No	D_Location
11	Mumbai
24	Delhi
13	Noida

4. Authorization

- To differentiate users based on the type of access they are permitted in the database.

Types of authorization:

1. read authorization - which allows reading, but not modification, of data;
2. insert authorization - which allows insertion of new data, but not modification of existing data;
3. Update authorization - which allows modification, but not deletion, of data;
4. delete authorization - which allows deletion of data.

- We may assign the user all, none, or a combination of these types of authorization.

Data Manipulation Language (DML)

- DML is used to access or manipulate the data in the database.

- (ie) is used to retrieve the data from the database, insert new data into the database, update and delete the existing data from the database.

Data Manipulation Language is mainly of two types:

- **Procedural DML:** This type of DML describes what data is to be accessed and how to get that data.

- **Declarative DML or Non-procedural DML:** This type of DML only describes what data is to be accessed without specifying how to get it.

Data Manipulation Language (DML) Commands:

- **Select:** To retrieve or access data from the database table.
- **Insert:** To insert the records into the table.
- **Update:** To change/update the existing data in a table.
- **Delete:** To delete records from the table.
- **MERGE** - To perform UPSERT operation (insert or update)
- **CALL** - To call a PL/SQL or Java subprogram
- **EXPLAIN PLAN** - To explain the access path to data
- **LOCK TABLE** - To control concurrency

Data Control Language(DCL)

- DCL is used to control privilege in Databases.
- It is mainly used for revoking and granting user access on a database.

Data Control Language (DCL) Commands:

- **Grant:** To grant access privileges to users to the database.
- **Revoke:** To take back permissions from the user .

Transaction Control Languages(TCL)

- Transaction Control language is a language to manage the changes made by DML statements

Transaction Control Language (TCL) Commands:

- **Commit:** This command is used to save the changes made by DML commands in database .
- **Rollback:** This command is used to restore changes made to the database which was last committed.
- **SAVEPOINT** - It identifies a point in a transaction to which you can later roll back
- **SET TRANSACTION -** to initiate a database transaction.

- Original

Roll No	Name	Grade
01	AA	S
02	BB	A
03	CC	C
04	DD	B

Modify

Roll No	Name	Grade
01	AA	S
02	BB	A
03	CC	C
04	DD	S

Rollback

Roll No	Name	Grade
01	AA	S
02	BB	A
03	CC	C
04	DD	B

Commit

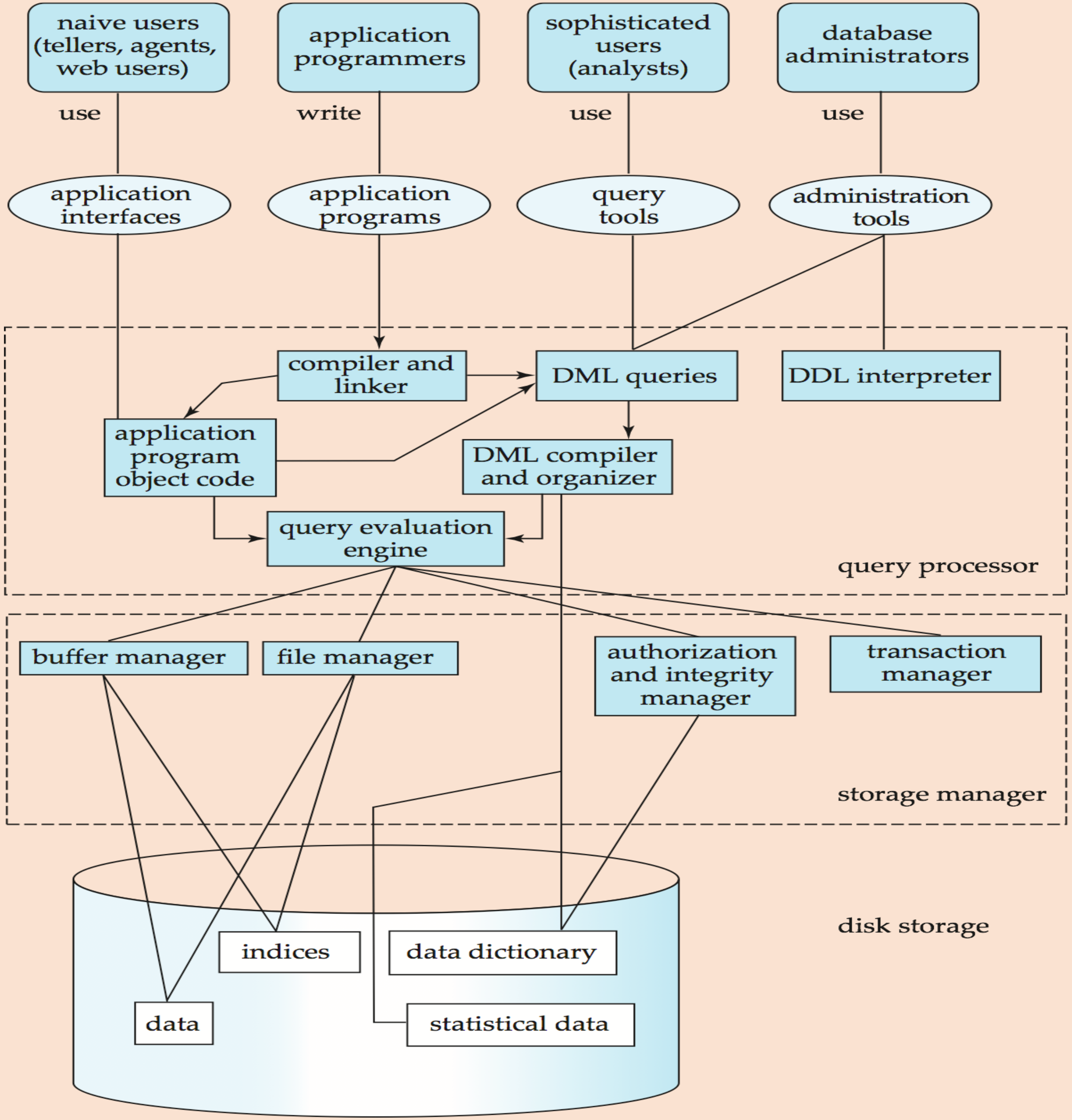
Roll No	Name	Grade
01	AA	S
02	BB	A
03	CC	C
04	DD	S

Roll No	Name	Grade
01	AA	S
02	BB	A
03	CC	C
04	DD	B

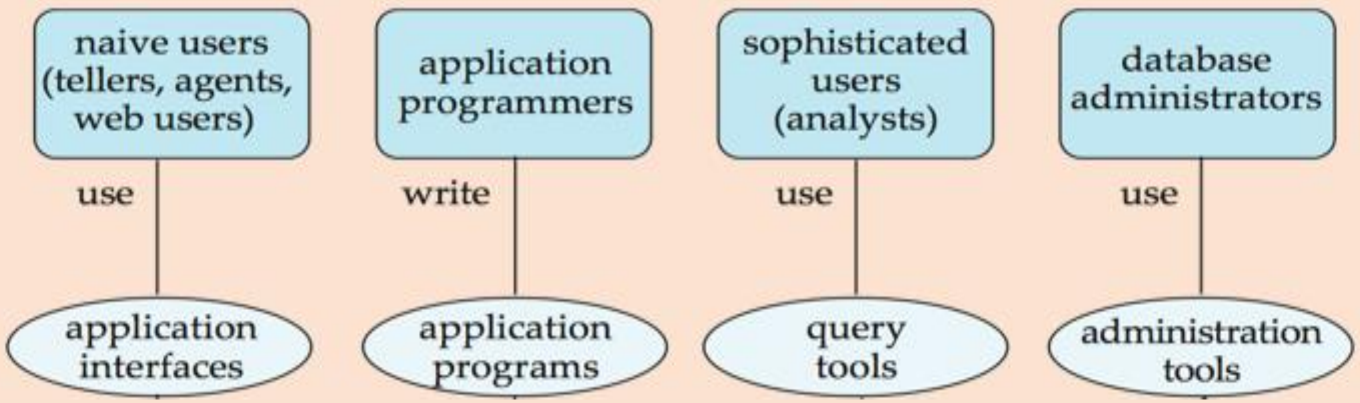
Roll No	Name	Grade
01	AA	S
02	BB	A
03	CC	C
04	DD	**S**

4. Architecture of a database system

- Components
  A. Database Users and User Interfaces
  B. Query processor
  C. Storage Manager


naive users (tellers, agents, web users)
application programmers
sophisticated users (analysts)
database administrators
use
write
use
use
application interfaces
application programs
query tools
administration tools
compiler and linker
DML queries
DDL interpreter
application program object code
DML compiler and organizer
query evaluation engine
query processor
buffer manager
file manager
authorization and integrity manager
transaction manager
storage manager
disk storage
indices
data dictionary
data
statistical data

A. Database Users and User Interfaces



- There are four different types of database-system users. They are differentiated by the way they interact with the system.

1. Naive users

- Unsophisticated users who interact with the system by using predefined user interfaces, such as web or mobile applications.
- Example for user interface is form interface where the user can fill in appropriate fields of the form.
- Naive users may also view/read *reports* generated from the database.

2. Application programmers

- Computer professionals who write application programs.
- Application programmers can choose from many tools to develop user interfaces.

3. Sophisticated users

- Interact with the system without writing programs
- Instead, they form their requests either using

  a database query language

  (or)

  tools such as data analysis software
- Analysts who submit queries to explore data in the database fall in this category.

4. Database Administrator (DBA)

- Has central control of both the data and the programs that access those data
- Functions of DBA include:

i. Schema definition

   by executing a set of statements in DDL.

ii. Storage structure and access-method definition

   - Schema and physical-organization modification
   - Granting of authorization for data access
   - Routine maintenance

- Schema and physical-organization modification.
  - To reflect the changing needs of the organization
  - To alter the physical organization to improve performance.

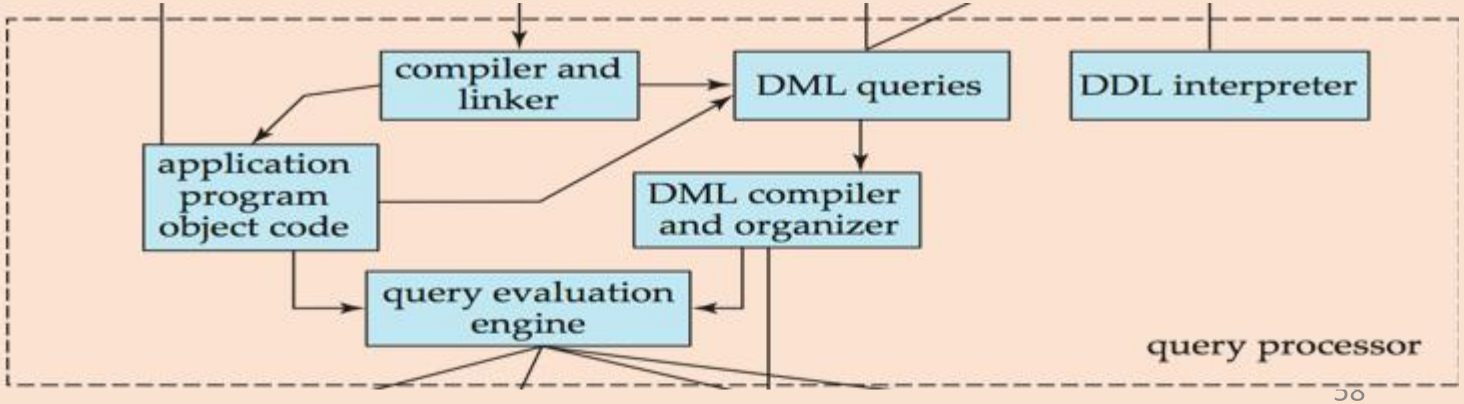
- Granting of authorization for data access.
  - To regulate which parts of the database various users can access.
  - The authorization information is kept in a special system structure that the database system consults whenever a user tries to access the data in the system.

- Routine maintenance
  - Backing up the database periodically to prevent loss of data.
  - Ensuring that enough free disk space is available, and upgrading disk space as required.
  - Monitoring jobs running on the database and ensuring that performance is not degraded by very expensive tasks submitted by some users.

B. The Query Processor

3 Components

- DDL interpreter, interprets DDL statements and records the definitions in the data dictionary.
- DML compiler, translates DML statements into low-level instructions that the query-evaluation engine understands.
- A query can be translated into any number of evaluation plans that all give the same result.
- The DML compiler performs query optimization; that is, it picks the lowest cost evaluation plan .
- Query evaluation engine, executes low-level instructions generated by the DML compiler.



C. Storage Manager

- Provides the interface between the low-level data stored in the database and the application programs and queries submitted to the system.

- The storage manager components include:
  - ➢ Authorization and integrity manager, tests for the satisfaction of integrity constraints and authority of users to access data.
  - ➢ Transaction manager ensures that
    - ✓ the consistency of the database
    - ✓ concurrent transactions proceed without conflicts.
  - ➢ File manager, manages
    - ✓ the allocation of space on disk storage
    - ✓ the data structures used to represent information stored on disk.
  - ➢ Buffer manager, responsible for
    - ✓ fetching data from disk storage into main memory
    - ✓ deciding what data to keep in cache memory.

The buffer manager is a critical part of the database system, since it enables the database to handle data sizes that are much larger than the size of main memory.

- The storage manager implements several data structures
  - Data files to store the database
  - Data dictionary, to store metadata about the structure of the database, in particular the schema of the database.
  - Indices, to provide fast access to data items. (Like the index in textbook)
  - Statistical and descriptive data about the relations and attributes,
    - ✓ number of tuples/records in each relation
    - ✓ number of distinct values for each attribute

UNIT II

UNIT – II Relational Approach

- Relational Model   Relational Data Structure
- Relational Data Integrity – Domain Constraints – Entity Integrity – Referential Integrity
- Keys
- Relational Algebra : Fundamental operations - Additional Operations
- Relational Calculus : Tuple Relational Calculus – Domain Relational Calculus
- SQL – Basic Structure – Set operations – Aggregate Functions – Null values – Nested Sub queries
- Derived Relations – Views – Modification of the database
- Joined Relations
- Data Definition Language
- Triggers.

Example of a *Instructor* Relation

attributes (or columns)

ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

tuples (or rows)

Attribute

- The set of allowed values for each attribute is called the **domain** of the attribute
- Attribute values are (normally) required to be **atomic**; that is, indivisible
- The special value ***null*** is a member of every domain. Indicated that the value is “unknown”
- The null value causes complications in the definition of many operations

- Order of tuples is irrelevant (tuples may be stored in an arbitrary order)

ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

Database schema & Instance

- Database schema --is the logical structure of the database.
- Database instance --is a snapshot of the data in the database at a given instant of time.

Example

- schema:

  instructor(*ID, name, dept_name, salary*)
- Instance:

ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

Relational Algebra

- A procedural language consisting of a set of operations that take one or two relations as input and produce a new relation as their result.
- Six basic / fundamental operators
  - ➢select: σ - sigma
  - ➢project: Π - pi
  - ➢Cartesian product: x - cross product
  - ➢Set Operations
    - ✓union: ∪
    - ✓Set intersection ∩
    - ✓set difference: –
  - ➢Join ⋈
  - ➢rename: ρ
- Unary Relational Operations - select, project, rename

Select Operation

- The **select** operation selects tuples that satisfy a given predicate.
- Notation: $\sigma_p(r)$

  p is called the selection predicate, ***r*** is relation name

Example:

1. Select tuples of the *instructor* relation where the instructor is in the "Physics" department.

$$\sigma_{dept\_name=\text{“}Physics\text{”}}(instructor)$$

ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
33456	Gold	Physics	87000

2. Find all instructors with salary greater than $90,000

$$\sigma_{salary>90000}(instructor)$$

- We allow comparisons using =, ≠, >, ≥. <. ≤ in the selection predicate.
- We can combine several predicates into a larger predicate by using the connectives:

  ∧(**and**), ∨(**or**), ¬(**not**)

Example:

- Find the instructors in Physics with a salary greater $90,000

$$\sigma_{dept\_name = \text{“Physics”} \wedge salary > 90000}(instructor)$$

Project Operation

- A unary operation that returns its argument relation, with certain attributes left out.
- Show desired attributes given in the list and eliminate the other attributes

Notation:

$$\Pi_{A_1, A_2, A_3 \ldots A_k}(r)$$

where *A1, A2* are attribute names and *r* is a relation name.

- The result is defined as the relation of *k* columns obtained by erasing the columns that are not given in the list
- Duplicate rows removed from result, since relations are sets.

- Example: eliminate the *dept_name* attribute of *instructor*
- Query*:*

$$\Pi_{ID,\ name,\ salary}\ (instructor)$$

- Result:

ID	*name*	*salary*
10101	Srinivasan	65000
12121	Wu	90000
15151	Mozart	40000
22222	Einstein	95000
32343	El Said	60000
33456	Gold	87000
45565	Katz	75000
58583	Califieri	62000
76543	Singh	80000
76766	Crick	72000
83821	Brandt	92000
98345	Kim	80000

Composition of Relational Operations

- Relational-algebra expressions

can be formed by compining relational-algebra operations

Example

- Find the names of all instructors in the Physics department.

$$\Pi_{name}\ (\sigma_{dept\_name = \text{“Physics”}}\ (instructor))$$

- Instead of giving the name of a relation as the argument of the projection operation, we give an expression that evaluates to a relation

- Just like composing arithmetic operations (such as +, −, ∗, and ÷) into arithmetic expressions.

Cartesian-Product Operation

- Allows to combine/merge information from any two relations.
- Notation $r1 \times r2$

  where r1 and r2 are relation names

Example

- The Cartesian product of the relations *instructor* and *teaches* is written as:

  instructor X *teaches*

- Concatenates each tuple from the *instructor* relation with each tuple from the teaches relation
- If same attribute name appear in the schemas of both *r1*and *r2,* the name of the relation will be attached with the attribute.
  - *instructor.ID*
  - *teaches.ID*

Instructor.ID	*name*	*dept_name*	*salary*	*teaches.ID*	*course_id*	*sec_id*	*semester*	*year*
10101	Srinivasan	Comp. Sci.	65000	10101	CS-101	1	Fall	2017
10101	Srinivasan	Comp. Sci.	65000	10101	CS-315	1	Spring	2018
10101	Srinivasan	Comp. Sci.	65000	10101	CS-347	1	Fall	2017
10101	Srinivasan	Comp. Sci.	65000	12121	FIN-201	1	Spring	2018
10101	Srinivasan	Comp. Sci.	65000	15151	MU-199	1	Spring	2018
10101	Srinivasan	Comp. Sci.	65000	22222	PHY-101	1	Fall	2017
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
12121	Wu	Finance	90000	10101	CS-101	1	Fall	2017
12121	Wu	Finance	90000	10101	CS-315	1	Spring	2018
12121	Wu	Finance	90000	10101	CS-347	1	Fall	2017
12121	Wu	Finance	90000	12121	FIN-201	1	Spring	2018
12121	Wu	Finance	90000	15151	MU-199	1	Spring	2018
12121	Wu	Finance	90000	22222	PHY-101	1	Fall	2017
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
15151	Mozart	Music	40000	10101	CS-101	1	Fall	2017
15151	Mozart	Music	40000	10101	CS-315	1	Spring	2018
15151	Mozart	Music	40000	10101	CS-347	1	Fall	2017
15151	Mozart	Music	40000	12121	FIN-201	1	Spring	2018
15151	Mozart	Music	40000	15151	MU-199	1	Spring	2018
15151	Mozart	Music	40000	22222	PHY-101	1	Fall	2017
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
22222	Einstein	Physics	95000	10101	CS-101	1	Fall	2017
22222	Einstein	Physics	95000	10101	CS-315	1	Spring	2018
22222	Einstein	Physics	95000	10101	CS-347	1	Fall	2017

Join Operation

- To combine a select operation and a Cartesian-Product operation into a single operation

$$r \bowtie_{\theta} s = \sigma_{\theta}(r \times s)$$

- Notation $r \bowtie_{\theta} s$
- θ is a predicate on attributes in the schema $R \cup S$.

Example:

- Associate every instructor with every course that was taught, regardless of whether that instructor taught that course.

$$instructor \bowtie_{instructor.ID = teaches.ID} teaches$$

This is equivalent to

$$\sigma_{instructor.id = teaches.id}(instructor \times teaches)$$

Set Operations – Union, Intersection, Set Difference

Union

- Notation : $r \cup s$
- For $r \cup s$ to be valid

1. r, s must have the *same* **arity**(same number of attributes)

2. The attribute domains must be **compatible.** The types of the i^{th} attributes of both input relations must be the same, for each i.

Example:

To find all courses taught in the Fall 2017 semester, or in the Spring 2018 semester, or in both

$$\Pi_{course\_id}\,(\sigma_{semester = \text{“Fall”} \land year=2017}\,(section)) \cup$$
$$\Pi_{course\_id}\,(\sigma_{semester = \text{“Spring”} \land year=2018}\,(section))$$

course_id
CS-101
CS-315
CS-319
CS-347
FIN-201
HIS-351
MU-199
PHY-101

Intersection

- The set-intersection operation allows us to find tuples that are in both the input relations.
- Notation: $r \cap s$

Example

- Find the set of all courses taught in both the Fall 2017 and the Spring 2018 semesters.

$$\Pi_{course\_id}\,(\sigma_{semester=\text{"Fall"} \wedge year=2017}\,(section)) \cap$$
$$\Pi_{course\_id}\,(\sigma_{semester=\text{"Spring"} \wedge year=2018}\,(section))$$

course_id
CS-101

Set Difference

- The set-difference operation allows us to find tuples that are in one relation but are not in another.
- Notation *r – s*

Example:

Find all courses taught in the Fall 2017 semester, but not in the Spring 2018 semester

$$\Pi_{course\_id}\,(\sigma_{semester = \text{“Fall”} \wedge year=2017}\,(section)) \;-$$
$$\Pi_{course\_id}\,(\sigma_{semester = \text{“Spring”} \wedge year=2018}\,(section))$$

course_id
CS-347
PHY-101

Assignment Operation

- It is a convenient way to express complex queries.
- It is used to assign part of a relational-algebra expression to temporary relation variables.
- It is denoted by $\leftarrow$
- It works like assignment in a programming language.

Example: Find all instructor in the "Physics" and Music department.

$$Physics \leftarrow \sigma_{dept\_name = \text{"Physics"}}(instructor)$$
$$Music \leftarrow \sigma_{dept\_name = \text{"Music"}}(instructor)$$
$$Physics \cup Music$$

- With the assignment operation, a query can be written as a sequential program consisting of a series of assignments followed by an expression whose value is displayed as the result of the query.

Rename Operation

- To give a name to the results of relational-algebra expressions that we can use to refer to them.
- The rename operator, ρ, is provided for that purpose
- The expression: $\rho_x(E)$

  returns the result of expression *E* under the name *x*
- Another form of the rename operation:

$$\rho_{x(A_1,A_2,\ldots,A_n)}(E)$$

returns the result of expression *E* under the name *x*, and with the attributes renamed to $A_1, A_2, \ldots, A_n$.

- This form of the rename operation can be used to give names to attributes in the results of relational algebra

- Consider the employee database

employee (*person_name*, *street*, *city*)
works (*person_name*, *company_name*, *salary*)
company (*company_name*, *city*)

- Give an expression in the relational algebra to express each of the following queries:

a. Find the name of each employee who lives in city “Miami”.

b. Find the name of each employee whose salary is greater than $100000.

c. Find the name of each employee who lives in “Miami” and whose salary is greater than $100000.

a. $\Pi_{name}\,(\sigma_{city\,=\,\text{“Miami”}}\,(employee))$

b. $\Pi_{name}\,(\sigma_{salary\,>\,100000}\,(employee))$

c. $\Pi_{name}\,(\sigma_{city\,=\,\text{“Miami”}\,\wedge\,salary>100000}\,(employee))$

- Find the names and cities of residence of all employees who work for "First Bank Corporation".
- Find the names, street address, and cities of residence of all employees who work for "First Bank Corporation" and earn more than $10,000.

$$\Pi_{person\_name,\, city}\,(employee \bowtie (\sigma_{company\_name\, =\, \text{"First Bank Corporation"}}\,(works)))$$

$$\Pi_{person\_name,\, street,\, city}\,(\sigma_{(company\_name\, =\, \text{"First Bank Corporation"}\, \wedge\, salary\, >\, 10000)}\,(works \bowtie employee))$$

- a. Find the ID and name of each employee who works for “BigBank”.
- b. Find the ID, name, and city of residence of each employee who works for “BigBank”.
- c. Find the ID, name, street address, and city of residence of each employee who works for “BigBank” and earns more than $10000.
- d. Find the ID and name of each employee in this database who lives in the same city as the company for which she or he works.

branch (branch-name, branch-city, assets)

customer (customer-name, customer-street, customer-only)

account (account-number, branch-name, balance)

loan (loan-number, branch-name, amount)

depositor (customer-name, account-number)

borrower (customer-name, loan-number)

branch(*branch_name*, *branch_city*, *assets*)
customer (*ID*, *customer_name*, *customer_street*, *customer_city*)
loan (*loan_number*, *branch_name*, *amount*)
borrower (*ID*, *loan_number*)
account (*account_number*, *branch_name*, *balance*)
depositor (*ID*, *account_number*)

- a.Find all loan numbers with a loan value greater than $10,000.
- b. Find the names of all depositors who have an account with a value greater than $6,000.
- c. Find the names of all depositors who have an account with a value greater than $6,000 at the “Uptown” branch.

$$\Pi_{loan\_number} (\sigma_{amount > 10000}(loan))$$

$$\Pi_{customer\_name} (\sigma_{balance > 6000} (depositor \bowtie account))$$

$$\Pi_{customer\_name} (\sigma_{balance > 6000 \wedge branch\_name = \text{“Uptown”}} (depositor \bowtie account))$$

branch(branch_name, branch_city, assets)
customer (ID, customer_name, customer_street, customer_city)
loan (loan_number, branch_name, amount)
borrower (ID, loan_number)
account (account_number, branch_name, balance)
depositor (ID, account_number)

1. Find the names of all branches located in "Chicago".
2. Find the names of all borrowers who have a loan in branch "Downtown".
3. Find each loan number with a loan amount greater than $10000.
4. Find the ID of each depositor who has an account with a balance greater than $6000.
5. Find the ID of each depositor who has an account with a balance greater than $6000 at the "Uptown" branch.

employee (*person-name*, *street*, *city*)
works (*person-name*, *company-name*, *salary*)
company (*company-name*, *city*)
manages (*person-name*, *manager-name*)

a. Find the names of all employees who work for First Bank Corporation.
b. Find the names and cities of residence of all employees who work for First Bank Corporation.
c. Find the names, street address, and cities of residence of all employees who work for First Bank Corporation and earn more than $10,000 per annum.
d. Find the names of all employees in this database who live in the same city as the company for which they work.
e. Find the names of all employees who live in the same city and on the same street as do their managers.
f. Find the names of all employees in this database who do not work for First Bank Corporation.
g. Find the names of all employees who earn more than every employee of Small Bank Corporation.
h. Assume the companies may be located in several cities. Find all companies located in every city in which Small Bank Corporation is located.

a. $\Pi_{person\text{-}name}\ (\sigma_{company\text{-}name\ =\ \text{"First Bank Corporation"}}\ (works))$

b. $\Pi_{person\text{-}name,\ city}\ (employee\ \bowtie$
$(\sigma_{company\text{-}name\ =\ \text{"First Bank Corporation"}}\ (works)))$

c. $\Pi_{person\text{-}name,\ street,\ city}$
$(\sigma_{(company\text{-}name\ =\ \text{"First Bank Corporation"}\ \wedge\ salary\ >\ 10000)}$
$works \bowtie\ employee)$

d. $\Pi_{person\text{-}name}\ (employee\ \bowtie\ works\ \bowtie\ company)$

e. $\Pi_{person\text{-}name}\ ((employee\ \bowtie\ manages)$
$\bowtie_{(manager\text{-}name = employee2.person\text{-}name\ \wedge\ employee.street = employee2.street}$
$_{\wedge\ employee.city = employee2.city)}\ (\rho_{employee2}\ (employee)))$

f. The following solutions assume that all people work for exactly one company. If one allows people to appear in the database (e.g. in *employee*) but not appear in *works*, the problem is more complicated. We give solutions for this more realistic case later.

$\Pi_{person\text{-}name}\ (\sigma_{company\text{-}name\ \neq\ \text{"First Bank Corporation"}}(works))$

If people may not work for any company:

$\Pi_{person\text{-}name}(employee)\ -\ \Pi_{person\text{-}name}$
$(\sigma_{(company\text{-}name\ =\ \text{"First Bank Corporation"})}(works))$

g. $\Pi_{person\text{-}name}\ (works)\ -\ (\Pi_{works.person\text{-}name}\ (works$
$\bowtie_{(works.salary\ \leq works2.salary\ \wedge\ works2.company\text{-}name = \text{"Small Bank Corporation"})}$
$\rho_{works2}(works)))$

h. Note: Small Bank Corporation will be included in each answer.

$\Pi_{company\text{-}name}\ (company\ \div$
$(\Pi_{city}\ (\sigma_{company\text{-}name = \text{"Small Bank Corporation"}}\ (company))))$

Normalization

- **Normalization** is a process of organizing the data in multiple related tables to avoid
    - ➢ data redundancy
    - ➢ insertion anomaly
    - ➢ update anomaly
    - ➢ & deletion anomaly

STUDENTS TABLE

rollno	name	branch	hod	office_tel
1	Akon	CSE	Mr. X	53337
2	Bkon	CSE	Mr. X	53337
3	Ckon	CSE	Mr. X	53337
4	Dkon	CSE	Mr. X	53337

STUDENTS TABLE

rollno	name	branch	hod	office_tel
1	Akon	CSE	~~Mr. X~~ Mr. Y	53337
2	Bkon	CSE	~~Mr. X~~ Mr. Y	53337
3	Ckon	CSE	Mr. X	53337
4	Dkon	CSE	~~Mr. X~~ Mr. Y	53337

Types of Normal Forms

Normal Forms	Description
1NF	A relation is in 1NF if it contains atomic attribute values.
2NF	A relation will be in 2NF if i)it is in 1NF ii) no partial functional dependency (ie all non-key attributes are fully functional dependent on the primary key.
3NF	A relation will be in 3NF if i) it is in 2NF ii) no transitive dependency exists.
3.5NF (BCNF)	A relation will be in 3.5NF if i)it is in 3 NF ii) Determinant of functional Dependency is super key.
4NF	A relation will be in 4NF if i) it is in Boyce Codd normal form ii) no multi-valued dependency. iii) atleast 3 attributes must be in the table
5NF	A relation is in 5NF if i) it is in 4NF ii) no join dependency and joining should be lossless.

Different ways to perform Normalization

- using Functional Dependency
  - 1NF
  - 2NF
  - 3NF
  - 3.5NF
- using Multivalued Dependency
  - 4NF
- using Join Dependency
  - 5NF

Normalization using Functional Dependencies

- First Normal Form (1NF)
  - A relation will be in 1NF if it contains **atomic value** (indivisible).
  - It should not have any **composite** attributes & **multivalued** attribute

- Example : **Customer**

cid	name	address	contact no
C01	aaa	01, Chennai	1234567988
C02	bbb	55, Chidambaram	{1233333322, 3331111111, 5555544457}
C03	ccc	32, Chennai	

Solutions

1. Insert new attributes for each sub-attribute of composite attributes.

cid	name	doorno	city	contact no
C01	aaa	01	Chennai	1234567988
C02	bbb	55	Chidambaram	{1233333322, 3331111111, 5555544457}
C03	ccc	32	Chennai	

2. Determine maximum allowable values for a multi-valued attribute & Insert new attributes

cid	name	doorno	city	contact_no1	contact_no2
C01	aaa	01	Chennai	1234567988	
C02	bbb	55	Chidambaram	1233333322	5555544457
C03	ccc	32	Chennai		

3. Insert new records for the multivalued attributes

cid	name	doorno	city	contact no
C01	aaa	01	Chennai	1234567988
C02	bbb	55	Chidambaram	1233333322
C02	bbb	55	Chidambaram	3331111111
C02	bbb	55	Chidambaram	5555544457
C03	ccc	32	Chennai	

4. Remove the multi-valued attribute that violates 1NF and place it in a separate relation along with the primary key of given original relation

cid	name	doorno	City
C01	aaa	01	Chennai
C02	bbb	55	Chidambaram
C03	ccc	32	Chennai

cid	contactno
C01	1234567988
C02	1233333322
C02	3331111111
C02	5555544457

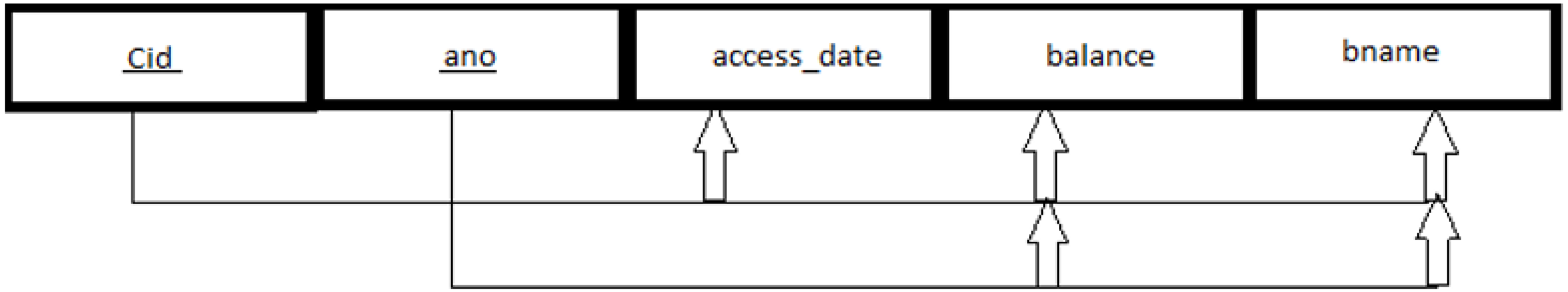
2NF

- A relation is in 2NF,
  - if it is in 1NF, and
  - every non_ prime attribute of relation is fully functionally dependent on primary key.

- A relation can violate 2NF only when it has more than one attribute in combination as a primary key.

- If relation has only single attribute as a primary key, then the relation will definitely be in 2NF.

- **Example:** Depositor_Account



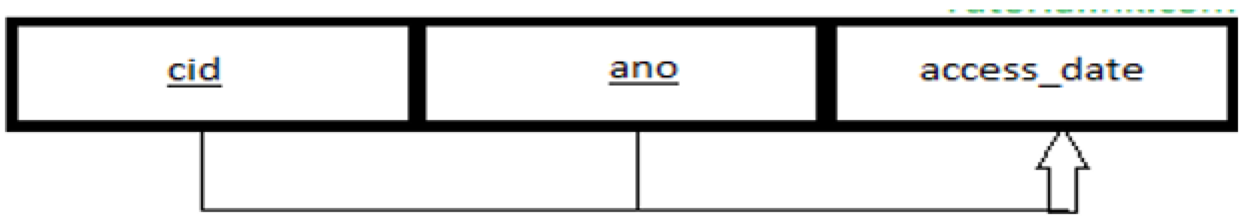
- In this relation schema,
    - access_date, balance and bname are non - prime attributes. access_date is fully dependent on primary key (cid and ano).
    - balance and bname are not fully dependent on primary key.
    - They depend on ano only.
    - So, this relation is not in Second normal form.

- This relation contains following functional dependencies.
    - FD1 : {cid, ano} -> {access_date, balance, bname}
    - FD2 : ano -> {balance, bname}

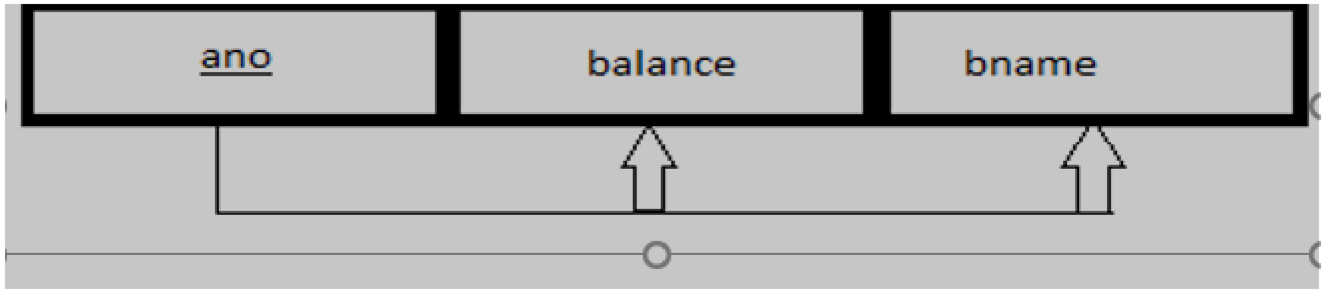
Solution

- ➢Decompose the relation in such a way that, resultant relations do not have any partial functional dependency.
- ➢For this purpose,
  - ✓remove the partially dependent non-prime attributes that violates 2NF in relation.
  - ✓Place them in a new relation along with the prime attribute on which they fully depend.

1.account

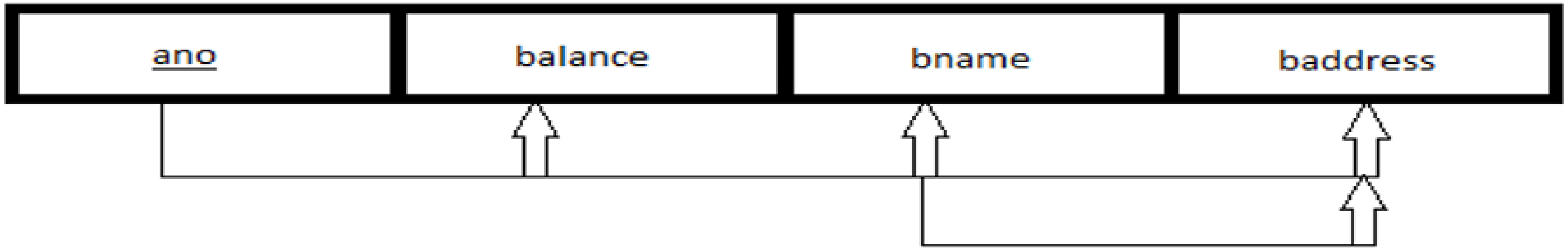


2.balance



3NF

- A relation schema R is in 3NF,
  - ➢ if it is in Second normal form
  - ➢no non-prime attribute of relation is transitively dependent on primary key
- Third normal form ensures that all the non-prime attributes of a relation directly depend on the primary key.



- This relation contains following functional dependencies.

  FD1 : ano -> {balance, bname, baddress}

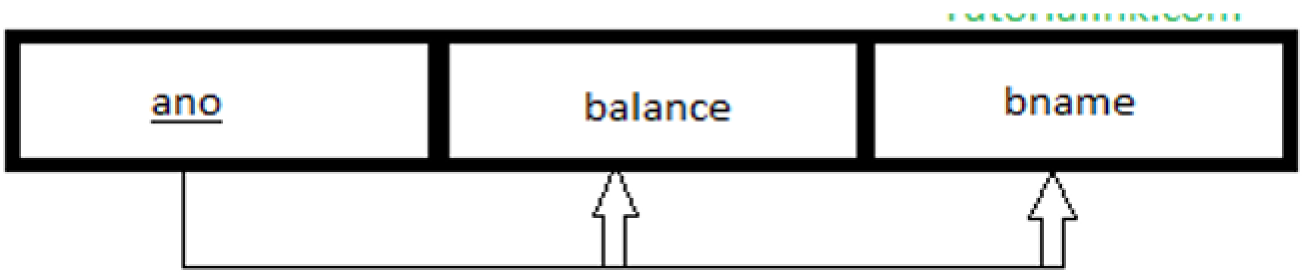
  FD2 : bname -> baddress

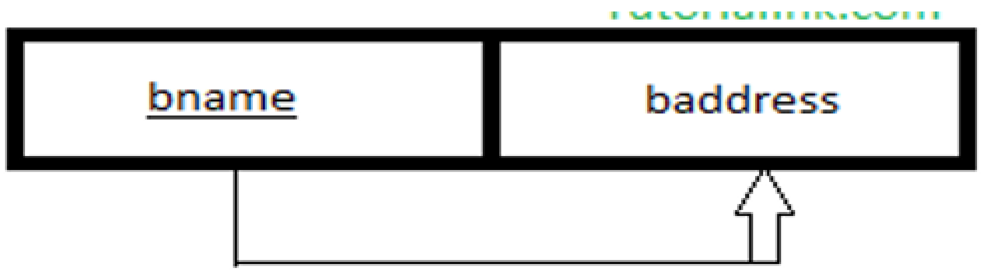
- In this relation , following transitive FD exists in baddress which is a nonprime attribute
    - ano->bname, bname->baddress ⇔ ano->baddress

Solution:

- Decompose the relation in such a way that, resultant relations do not have any non-prime attribute transitively dependent on primary key.
- For this purpose,
    - ➢ Remove the transitively dependant non-prime attributes that violates 3NF from relation.
    - ➢ Place them in a new relation along with the non-prime attribute due to which transitive dependency occurred.
    - ➢ The primary key of new relation will be the non-prime attribute.

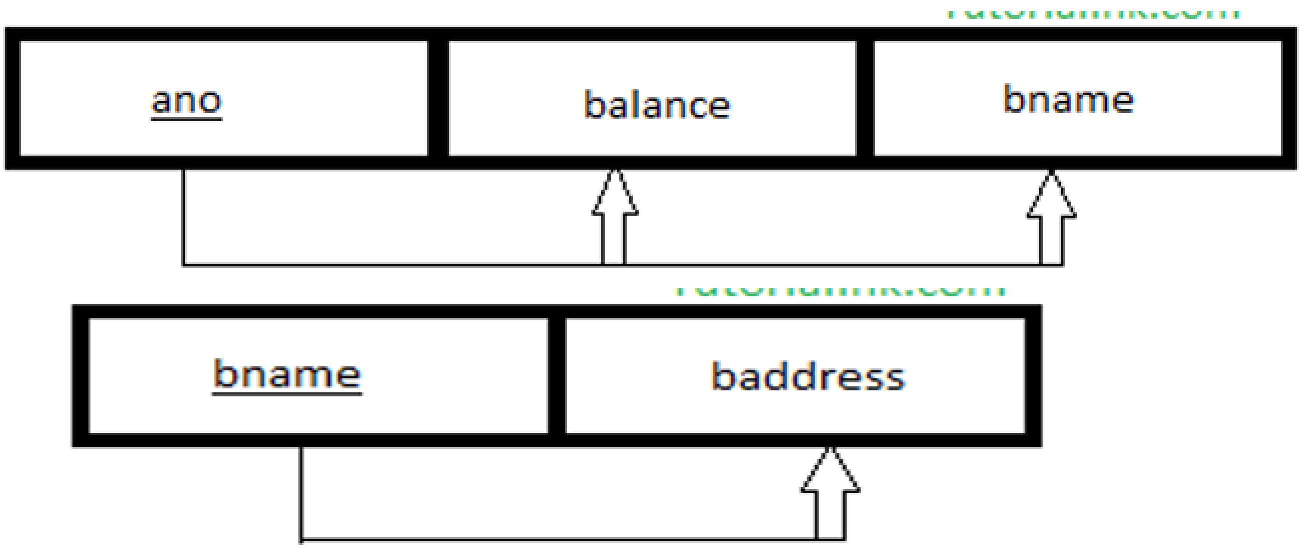
- In our example, baddress is transitively dependent on ano due to non-prime attribute bname.
- So, remove baddress and place it in separate relation called branch along with the non-prime attribute bname.
- For relation branch, bname will be a primary key.





Boyce Codd normal form (BCNF) / 3.5 NF

- It is stricter than 3NF.
- A table is in BCNF if
  - It is in 3 NF
  - In every functional dependency, determinant is super key of the table.



Multivalued Dependencies ->>

Let R be a relation schema and let $\alpha \subseteq R$ and $\beta \subseteq R$. The **multivalued dependency**

$$\alpha \rightarrow\rightarrow \beta$$

holds on R if in any legal relation $r(R)$, for all pairs for tuples t_1 and t_2 in r such that $t_1[\alpha] = t_2[\alpha]$, there exist tuples t_3 and t_4 in r such that:

$$
\begin{aligned}
t_1[\alpha] = t_2[\alpha] &= t_3[\alpha] = t_4[\alpha] \\
t_3[\beta] &= t_1[\beta] \\
t_3[R - \beta] &= t_2[R - \beta] \\
t_4[\beta] &= t_2[\beta] \\
t_4[R - \beta] &= t_1[R - \beta]
\end{aligned}
$$

STU_ID	COURSE	HOBBY
21	Computer	Dancing
21	Math	Singing
21	Computer	Singing
21	Math	Dancing
34	Chemistry	Dancing
74	Biology	Cricket
59	Physics	Hockey

Tabular representation of $\alpha \rightarrow\rightarrow \beta$

	α	β	$R - \alpha - \beta$
t_1	$a_1 \ldots a_i$	$a_{i+1} \ldots a_j$	$a_{j+1} \ldots a_n$
t_2	$a_1 \ldots a_i$	$b_{i+1} \ldots b_j$	$b_{j+1} \ldots b_n$
t_3	$a_1 \ldots a_i$	$a_{i+1} \ldots a_j$	$b_{j+1} \ldots b_n$
t_4	$a_1 \ldots a_i$	$b_{i+1} \ldots b_j$	$a_{j+1} \ldots a_n$

Fourth normal form (4NF)

STU_ID	COURSE	HOBBY
21	Computer	Dancing
21	Math	Singing
34	Chemistry	Dancing
74	Biology	Cricket
59	Physics	Hockey

- A relation will be in 4NF
  - if it is in **Boyce Codd normal form (BCNF)** and
  - has no **multi-valued dependency**.
- Table should have at least **3 attributes(columns)**
- Two attributes in a table are independent of one another, but both depend on a third attribute.
- For a dependency , if for a single value of A, multiple values of B and multiple values of C exists, then the relation is said to be in multi-valued dependency.

- **Example**
- The given STUDENT relation is in 3NF, but the COURSE and HOBBY are two independent entity. Hence, there is no relationship between COURSE and HOBBY.
- In the relation, a student with STU_ID, **21** contains
  - two courses, **Computer** and **Math**

    STU_ID ->> COURSE
  - two hobbies, **Dancing** and **Singing**

    STU_ID ->> HOBBY
- So there is a Multi-valued dependency on STU_ID, which leads to unnecessary repetition of data.
- To make the above table into 4NF, we can decompose it into two tables:

STUDENT_COURSE

STU_ID	COURSE
21	Computer
21	Math
34	Chemistry
74	Biology
59	Physics

STUDENT_HOBBY

STU_ID	HOBBY
21	Dancing
21	Singing
34	Dancing
74	Cricket
59	Hockey

Join Dependency

- Generalization concept of multivalued dependency.
- Let R is a schema of relation r.
- R1, R2,… be the decomposition of R and the relations are r1, r2… respectively
- The relation r is said to satisfy join dependency

    if and only if the join of r1, r2… is equal to relation r

  ie lossless decomposition

Agent	Company	Product
Aman	C1	Pen drive
Aman	C1	MIC
Aman	C2	Speaker
Mohan	C1	Speaker

R_1

Agent	Company
Aman	C1
Aman	C2
Mohan	C1

R_2

Agent	Product
Aman	Pendrive
Aman	MIC
Aman	Speaker
~~Aman~~ Mohan	Speaker

R_3

Comp.	Prod.
C1	P.D
C1	MIC
C1	Speaker
C2	Speaker

Normalization using Join Dependency

- A relation is in 5NF if
  - it is in 4NF
  - no join dependency or no lossless decomposition.
- Also called as **project-join normal form (PJNF)**

Ques) Is the table in 5NF?

PName	Skill	Job
Aman	DBA	J1
Mohan	Tester	J2
Rohan	Programmer	J3
Sohan	Analyst	J1

Domain-key normal form or DKNF

- It is a normal form in which database contains only two constraints which are:
  - ➢domain constraints,
  - ➢key constraints.
- The function of domain constraint is to specify the permissible values for a given attribute
- The main function of a key constraint is to specify the attributes which uniquely identify a row in a given table.
- Relationships which are impossible to express in foreign keys will violate the Domain Key Normal Form.
- Also called as 6NF

1. Which of the following is **TRUE**?
   (A) Every relation in 3NF is also in BCNF
   (B) A relation R is in 3NF if every non-prime attribute of R is fully functionally dependent on every key of R
   (C) Every relation in BCNF is also in 3NF
   (D) No relation can be in both BCNF and 3NF

2. Which of the following is NOT a superkey in a relational schema with attributes V, W, X, Y, Z and primary key V Y ?
   (A) V X Y Z
   (B) V W X Z
   (C) V W X Y
   (D) V W X Y Z

3. Let R (A, B, C, D, E, P, G) be a relational schema in which the following functional dependencies are known to hold: AB → CD, DE → P, C → E, P → C and B → G. The relational schema R is

   (A) in BCNF
   (B) in 3NF, but not in BCNF
   (C) in 2NF, but not in 3NF
   (D) not in 2NF

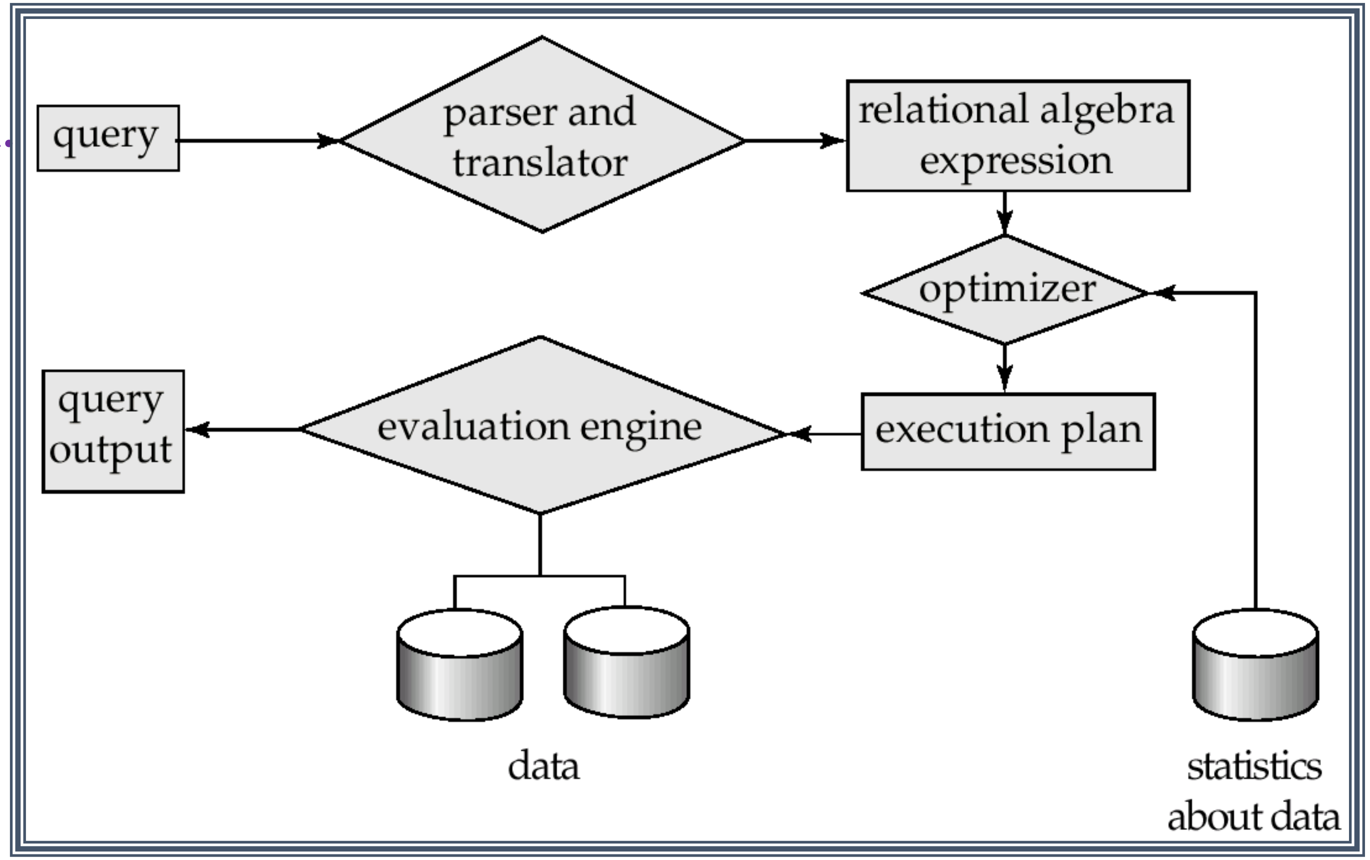
UNIT IV

Unit IV Topics

- Query Processing Overview
- Estimation of Query Processing Cost - Join strategies
- Transaction Processing – Concepts and States
- implementation of Atomicity and Durability
- Concurrent Executions
- Serializability
- implementation of Isolation :
  - ➢Testing for Serializability
  - ➢Concurrency control
    - ✓Lock Based Protocol
    - ✓Timestamp Based Protocols.

1.Query Processing

- **Query processing** refers to the range of activities involved in extracting data from a database.
- The basic steps are:
  1. Parsing and translation.
  2. Optimization.
  3. Evaluation.

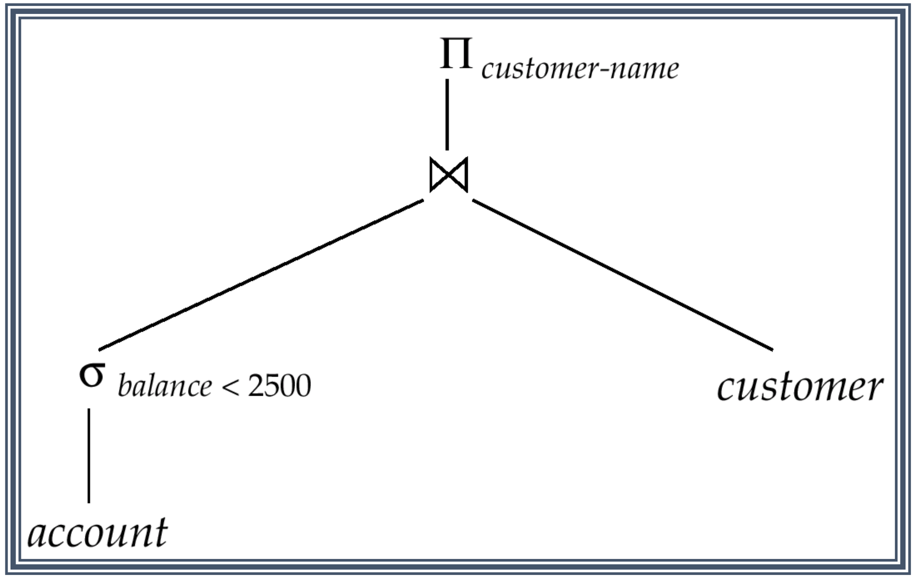


1. Parsing and translation

- This is the first action the system must take in query processing.
- This is to translate a given query into its internal form ie a relational-algebra expression .
- This translation process is similar to the work performed by the parser of a compiler.
- In generating the internal form of the query,
  - parser checks the syntax of the user's query,
  - verifies that the relation names appearing in the query are names of the relations in the database.
- The system then
  - i) constructs a parse-tree representation of the query
  - ii) translates into a relational-algebra expression.

Example:

- **select balance from account where balance <**2500;
- This query can be translated into either of the following relational-algebra expressions:
  - ✓ $\sigma_{balance<2500}(\Pi_{balance}(account))$
  - ✓ $\Pi_{balance}(\sigma_{balance<2500}(account))$



2. Optimization

➢Finding/choosing an evaluation plan with lowest estimation

3 steps

1. Generating logically equivalent expressions
   - Use **equivalence rules** to transform an expression into an equivalent one.

Example

$$\sigma_{\theta_1 \wedge \theta_2}(E) = \sigma_{\theta_1}(\sigma_{\theta_2}(E))$$

$$\sigma_{\theta_1}(\sigma_{\theta_2}(E)) = \sigma_{\theta_2}(\sigma_{\theta_1}(E))$$

$$\sigma_\theta(E_1 \times E_2) = E_1 \bowtie_\theta E_2$$

$$\sigma_{\theta 1}(E_1 \bowtie_{\theta 2} E_2) = E_1 \bowtie_{\theta 1 \wedge \theta 2} E_2$$

- Query: Find the names of all customers with an account at a Brooklyn branch whose account balance is over $1000.

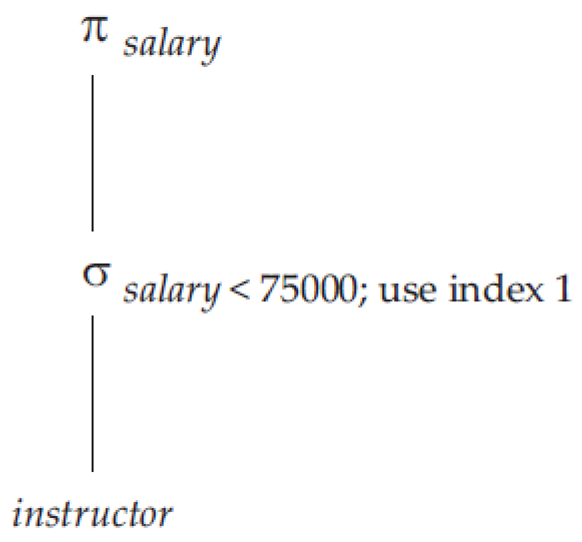
$$\Pi_{customer\text{-}name}(\sigma_{branch\text{-}city = \text{“Brooklyn”} \wedge balance > 1000}(branch \bowtie (account \bowtie depositor)))$$

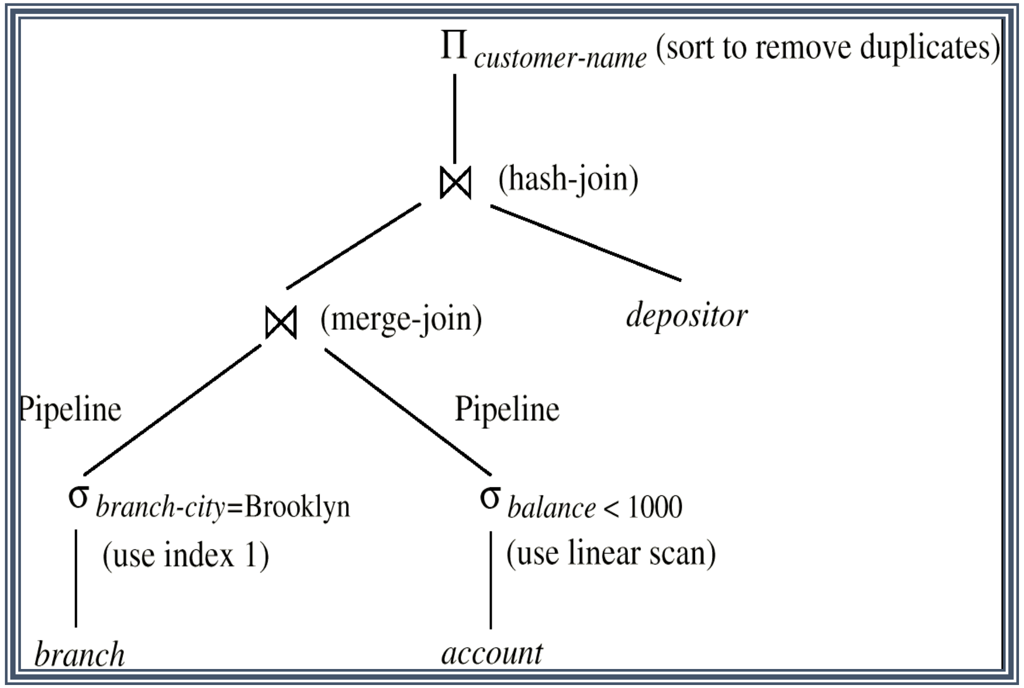
$$\Pi_{customer\text{-}name}((\sigma_{branch\text{-}city = \text{“Brooklyn”} \wedge balance > 1000}(branch \bowtie (account)) \bowtie depositor)$$

$$\sigma_{branch\text{-}city = \text{“Brooklyn”}}(branch) \bowtie \sigma_{balance > 1000}(account)$$

2. Annotating resultant expressions to get alternative query plans

- state the algorithm to be used for a specific operation, (linear scan/binary scan)
- state indices to use.





- A relational algebra operation annotated with instructions on how to evaluate it is called an **evaluation primitive**.
- A sequence of primitive operations that can be used to evaluate a query is a **query-execution plan** or **query-evaluation plan**

3. Choosing the cheapest plan based on **estimated cost**

Cost is estimated using statistical information from the database catalog

- number of tuples in each relation,
- size of tuples,
- number of blocks containing tuples
- number of distinct values that appear in *r* for attribute *A*
- average number of records that satisfy equality on *A*

Evaluation

Query-execution engine

- This takes a query-evaluation plan,
- It executes that plan and returns the output of the query.

2. Estimation of Query Processing Cost

- Cost is generally measured as total elapsed time for answering query
  - Many factors contribute to time cost
    - *disk accesses, CPU*, or even network *communication*
- Typically disk access is the predominant cost, and is also relatively easy to estimate.
- Measured by taking into account
  - Number of seeks * average-seek-cost
  - Number of blocks read * average-block-read-cost
  - Number of blocks written * average-block-write-cost
- Cost to write a block is greater than cost to read a block
  - data is read back after being written to ensure that the write was successful

- Costs depends on the size of the buffer in main memory
  - Having more memory reduces need for disk access

- For simplicity
  - use *number of block transfers from disk* as the cost measure
  - ignore the difference in cost between sequential and random I/O
  - ignore CPU costs

- Real systems take CPU cost into account, differentiate between sequential and random I/O, and take buffer size into account

- We do not include cost to writing output to disk in our cost formulae

3.Transaction Processing

- The term *transaction* refers to a collection of operations that form a single logical unit of work.

Example

Transfer of money from one account to another is a transaction consisting of two updates, one to each account.

1. **read**(A)
2. $A := A - 50$
3. **write**(A)
4. **read**(B)
5. $B := B + 50$
6. **write**(B)

Definition:

- A **transaction** is a **unit** of program execution that accesses and possibly updates various data items.

Transaction Concept

- Usually, a transaction is initiated by a user program written
  - in a high-level data-manipulation language (typically SQL),
  - or programming language (for example, C++, or Java), with embedded database accesses in JDBC or ODBC.
- A transaction is delimited by statements
  - **begin transaction** and **end transaction**.
- All operations to execute must be
  - between the **begin transaction** and **end transaction**.
- This collection of steps must appear to the user as a single, indivisible unit.

Properties of the transactions - ACID Properties

- **Atomicity**. Either all operations of the transaction reflect in database or none .
- **Consistency**. Refers to correctness. Transaction must preserves the consistency of the database.
  - To preserve consistency the execution of transaction should take place in isolation
- **Isolation**. During the execution of concurrent multiple transactions, the system must guarantee that, every pair of transactions is unaware of other transactions executing concurrently in the system.
  - Consider Ti and Tj are the 2 transactions
  - It should appears to Ti that
    - ✓ either Tj finished execution before Ti started
    - ✓ or Tj started execution after Ti finished.

- **Durability**. After a transaction completes successfully, the changes it has made to the database should be permanent, even if there are system failures.

- Example

  Transaction to transfer $50 from account *A* to account *B*:

1. **read**(A)
2. $A := A - 50$
3. **write**(A)
4. **read**(B)
5. $B := B + 50$
6. **write**(B)

- Atomicity requirement — if the transaction fails after step 3 and before step 6, the system should ensure that its updates are not reflected in the database.

- Consistency requirement – the sum of A and B is unchanged by the execution of the transaction.

- Isolation requirement — if between steps 3 and 6, another transaction is allowed to access the partially updated database, it will see an inconsistent database

- Durability requirement — once the user has been notified that the transaction has completed (i.e., the transfer of the 50 has taken place), the updates to the database by the transaction must persist despite failures.

Transaction State

Transaction must be in one of the following states:

- **Active :** If a transaction is in execution then it is said to be in active state.

  It doesn't matter which step is in execution,

  until the transaction is executing, it remains in active state.

- **Partially committed : After** all the statement has been executed then it is said to be in partially committed state.

  All the read and write operations performed on the main memory (local memory) instead of the actual database.

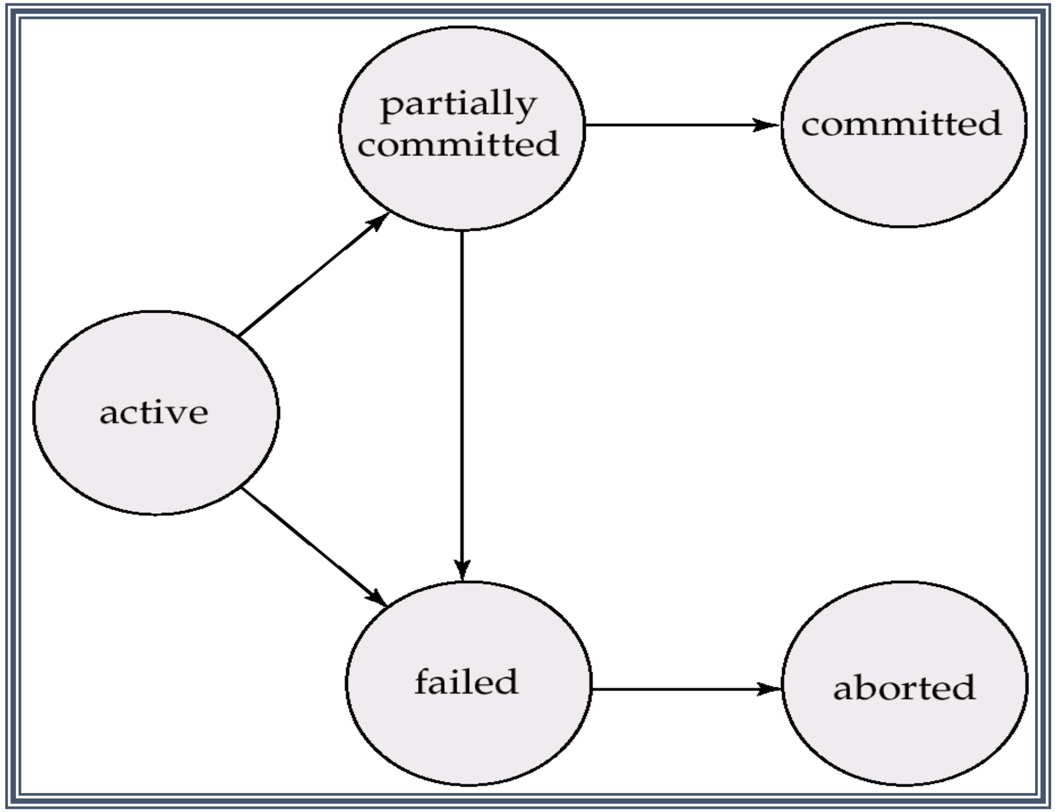
- **Failed:** If a failure occurs during transaction either a hardware failure or a software failure then the transaction goes into failed state from the active state.

- **Aborted**: After the transaction has been rolled back

  Database will be restored to its state prior to the start of the transaction.

- **Committed**: After successful completion of all statements.

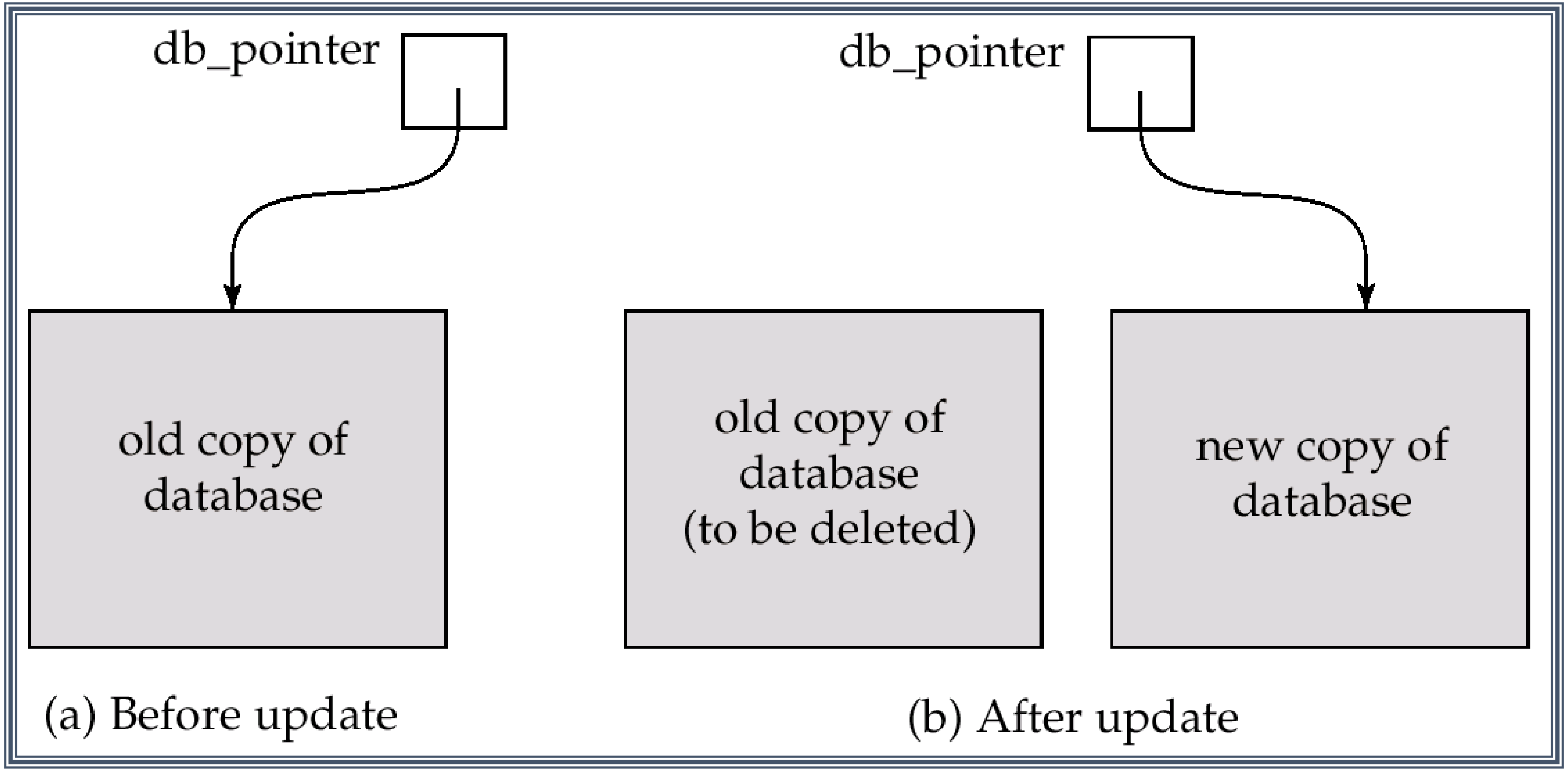
  All the changes made in the local memory during **partially committed** state are permanently stored in the database.



4. Implementation of Atomicity and Durability

- A transaction may not always complete its execution successfully. Such a transaction is termed **aborted**.
- If we are to ensure the atomicity property, an aborted transaction must have no effect on the state of the database.
- Thus, any changes that the aborted transaction made to the database must be undone.
- Once the changes caused by an aborted transaction have been undone, we say that the transaction has been **rolled back**.
- It is the responsibility of the recovery management component to manage transaction aborts.

Shadow-database scheme



The *shadow-database* scheme:

- Assume that only one transaction is active at a time.

- A pointer called db_pointer always points to the current consistent copy of the database.

- All updates are made on a *shadow copy* of the database, and **db_pointer** is made to point to the updated shadow copy only after the transaction reaches partial commit and all updated pages have been flushed to disk.

- In case transaction fails, old consistent copy pointed to by **db_pointer** can be used, and the shadow copy can be deleted.

- Useful for text editors, but extremely inefficient for large databases: executing a single transaction requires copying the *entire* database

- Maintaining a **log**
- Each database modification made by a transaction is first recorded in the log.
- We record the identifier of the transaction performing the modification, the identifier of the data item being modified, and both the old value (prior to modification) and the new value (after modification) of the data item.
- Only then is the database itself modified.
- Maintaining a log provides the possibility of redoing a modification to ensure atomicity and durability as well as the possibility of undoing a modification to ensure atomicity in case of a failure during transaction execution.
- A transaction that completes its execution successfully is said to be **committed**.
- A committed transaction that has performed updates transforms the database into a new consistent state, which must persist even if there is a system failure.

- Once a transaction has committed, we cannot undo its effects by aborting it.
- The only way to undo the effects of a committed transaction is to execute a **compensating transaction**.
- For instance, if a transaction added 2000 to an account, the compensating transaction would subtract 2000 from the account.
- However, it is not always possible to create such a compensating transaction. Therefore, the responsibility of writing and executing a compensating transaction is left to the user, and is not handled by the database system.

5. Concurrent Executions

- Allowing multiple transactions to update data concurrently

Advantages

- **Increased processor and disk utilization:**
  - ✓ one transaction can be using the CPU
  - ✓ while another is reading from or writing to the disk
  - ✓ This gives better transaction throughput

- **Reduced average response time for transactions:**
  - ✓ There may be a mix of transactions running on a system, some short and some long.
  - ✓ If transactions run serially, a short transaction may have to wait for a preceding long transaction to complete. This can lead to unpredictable delays in running a transaction.
  - ✓ But in concurrent executions short transactions need not wait behind long ones.

- When several transactions run concurrently,
  - ✓ the isolation property may be violated,
  - ✓ database inconsistent
- The database system must control the interaction among the concurrent transactions to prevent them from destroying the consistency of the database.
- It does so through a variety of mechanisms called **concurrency-control schemes**.

Schedules

S1: $R_1(A)$, $W_1(A)$, $R_2(A)$, $W_2(A)$, $R_1(B)$, $W_1(B)$, $R_2(B)$, $W_2(B)$

- This represents the chronological order in which instructions are executed in the system.
- A schedule for a set of transactions
  - ✓ must consist of all instructions of those transactions,
  - ✓ and must preserve the order in which the instructions appear in each individual transaction.
- Two types of schedules

1. Serial schedule:

- ➢Instructions belonging to one single transaction appear together in that schedule.
- ➢A serial schedule is always consistent.
- ➢If a schedule S has T1 and T2, possible serial schedules are
  - ✓T1 followed by T2 (T1->T2) or
  - ✓T2 followed by T1 ((T2->T1).
- ➢A serial schedule has low throughput and less resource utilization

2. Concurrent schedule

- Instructions of one transaction are interleaved with Instructions of other transactions of a schedule

- If two transactions are running concurrently, the operating system may execute one transaction for a little while, then perform a context switch, execute the second transaction for some time, and then switch back to the first transaction for some time, and so on.

- With multiple transactions, the CPU time is shared among all the transactions.

- Several execution sequences are possible, since the various instructions from both transactions may now be interleaved.

- In general, it is not possible to predict exactly how many instructions of a transaction will be executed before the CPU switches to another transaction.

- Concurrency can lead to inconsistency in the database.

A = 1000 B = 2000

Let T_1 transfer 50 from *A* to *B*, and T_2 transfer 10% of the balance from *A* to *B*.

Serial Schedule

T_1	T_2
read(A)	
$A := A - 50$	
write (A)	
read(B)	
$B := B + 50$	
write(B)	
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
	read(B)
	$B := B + temp$
	write(B)

Concurrent Schedule

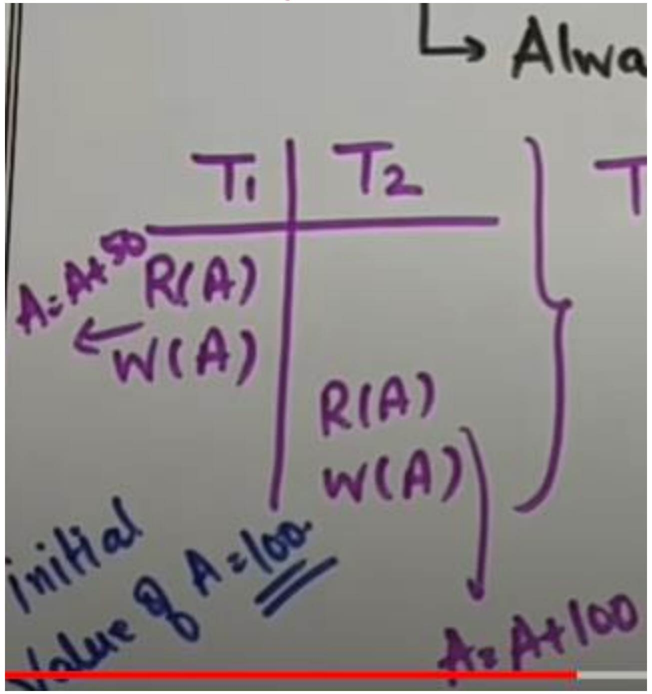
T_1	T_2
read(A)	
$A := A - 50$	
write(A)	
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
read(B)	
$B := B + 50$	
write(B)	
	read(B)
	$B := B + temp$
	write(B)

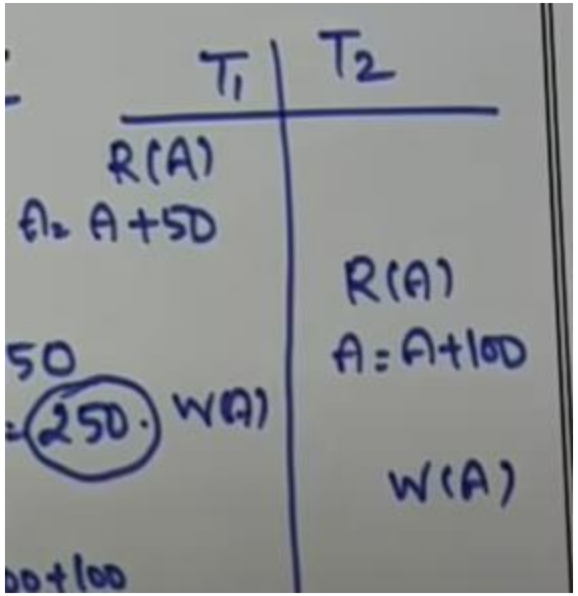
In both Schedules, the sum A + B is preserved

- Not all concurrent executions result in a correct state.
- After the execution of this schedule, we arrive at a state where the final values of accounts A and B are not same (950 and 2100), respectively.
- This final state is an *inconsistent state*.

T_1	T_2
read(A)	
$A := A - 50$	
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
	read(B)
write(A)	
read(B)	
$B := B + 50$	
write(B)	
	$B := B + temp$
	write(B)

- If control of concurrent execution is left entirely to the operating system, many possible schedules, including ones that leave the database in an inconsistent state, such as the one just described, are possible.
- It is the job of the database system to ensure that any schedule that is executed will leave the database in a consistent state.
- The **concurrency-control** component of the database system carries out this task





6. Serializability

- Checking correctness of schedules
- When multiple transactions are running concurrently then there is a possibility that the database may be left in an inconsistent state.
- Serializability is a concept that helps us to check which schedules are serializable. A serializable schedule is the one that always leaves the database in consistent state.
- A (concurrent) schedule is serializable if it is equivalent to a serial schedule.
- Types of serializability
    1. conflict serializability
    2. view serializability

Conflict Serializability

- A schedule is **conflict serializable** if it is **conflict equivalent** to a serial schedule

- If a schedule S can be transformed into a schedule S' by a series of swaps of non-conflicting instructions, we say that S and S' are **conflict equivalent**

- **Conflicting operations:** Two operations are said to be conflicting if following all conditions satisfy:

  They belong to different transactions

  They operate on the same data item

  At least one of them is a write operation

- If the conflicting operations are in the same order then it is conflict equivalent

Example: –

- **Conflicting** operations pair ($R_1(A)$, $W_2(A)$)

because they belong to two different transactions on same data item A and one of them is write operation.

- $W_2(A)$) and ($W_1(A)$, $R_2(A)$) pairs are also **conflicting**.

- ($R_1(A)$, $W_2(B)$) pair is **non-conflicting** because they operate on different data item.

- (($W_1(A)$, $W_2(B)$) pair is **non-conflicting.**

- Consider the following schedule:

S1: $R_1(A)$, $W_1(A)$, $R_2(A)$, $W_2(A)$, $R_1(B)$, $W_1(B)$, $R_2(B)$, $W_2(B)$

Two transactions of schedule S1

T1: $R_1(A)$, $W_1(A)$, $R_1(B)$, $W_1(B)$

T2: $R_2(A)$, $W_2(A)$, $R_2(B)$, $W_2(B)$

- R1(A),W2(A)
- W1(A),R2(A)
- W1(A), W2(A)
- R1(B), W2(B)
- W1(B),W2(B)

- Two serial schedules
- T1T2 : $R_1(A)$, $W_1(A)$, $R_1(B)$, $W_1(B)$ $R_2(A)$, $W_2(A)$, $R_2(B)$, $W_2(B)$
- T2T1 $R_2(A)$, $W_2(A)$, $R_2(B)$, $W_2(B)$ $R_1(A)$, $W_1(A)$, $R_1(B)$, $W_1(B)$

- **Conflict Operations**
  - R1(A),W2(A)
  - W1(A),R2(A)
  - W1(A), W2(A)
  - R1(B), W2(B)
  - W1(B),W2(B)

Check the order

- Order is maintained in a serial schedule
- So conflict serializable

S: R1(x), R2(x) ,W1(x) ,R1(y), W2(x), W1(y)

Transactions

T1 : R1(x), W1(x) ,R1(y), W1(y)

T2: R2(x) ,W2(x)

Serial Schedules

T1 T2: R1(x), W1(x) ,R1(y), W1(y), R2(x) ,W2(x)

T2 T1: R2(x) ,W2(x),R1(x), W1(x) ,R1(y), W1(y)

Conflict operations

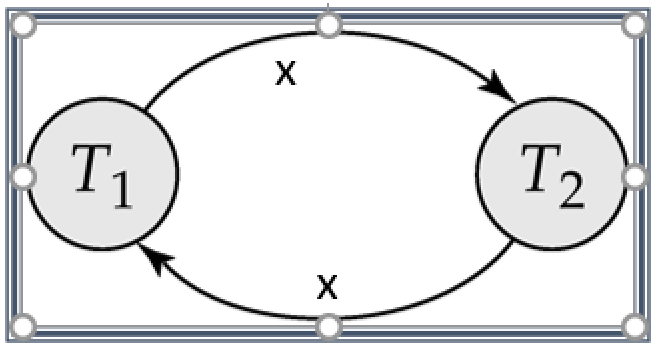
1. R1(x), W2(x)
2. R2(x) W1(x)
3. W1(x), W2(x)

Check the order

Not maintaining the order. So not conflict serializable

Precedence Graph

- Simple and efficient method for determining conflict serializability of a schedule.
- It is a directed graph, constructed from schedule .
- This graph consists of a pair $G = (V, E)$ where
  - ➢ V is a set of vertices (Nodes)
  - ➢ E is a set of edges.



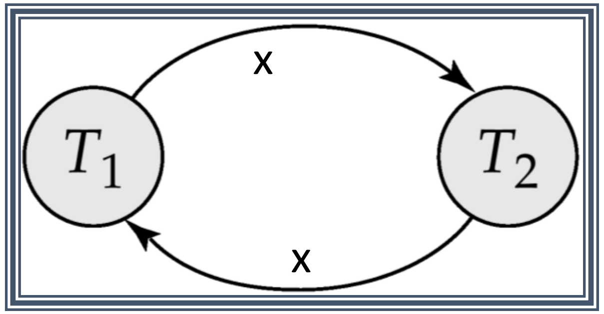
- No. of Vertices = No. of transactions in the schedule.
- Edges - for all conflict operations draw a directed edge. Label it with the dataitem
- If R1(A), W2(A) are the 2 operations direction of the edge is from V1 to V2
- If W2(A), R1(A) are the 2 operations direction of the edge is from V2 to V1

- If the precedence graph for schedule *S* has a cycle, then the schedule *S* is not conflict serializable.
- If the graph contains no cycles, then the schedule *S* is conflict serializable.
- Eg.

S: R1(x), R2(x) ,W1(x) ,R1(y), W2(x), W1(y)

- Conflict operations

1. R1(x), W2(x)
2. R2(x) W1(x)
3. W1(x), W2(x)



View Serializability

- A schedule S is **view serializable** it is view equivalent to a serial schedule.
- Let S and S' be two schedules with the same set of transactions. S and S' are **view equivalent** if the following three conditions are met:

1. For each data item Q, if transaction T_i reads the initial value of Q in schedule S, then transaction T_i must, in schedule S', also read the initial value of Q.
2. For each data item Q if transaction T_i executes **read**(Q) in schedule S, and that value was produced by transaction T_j (if any), then transaction T_i must in schedule S' also read the value of Q that was produced by transaction T_j.
3. For each data item Q, the transaction performs the final **write**(Q) operation in schedule S, must perform the final **write**(Q) operation in schedule S'.

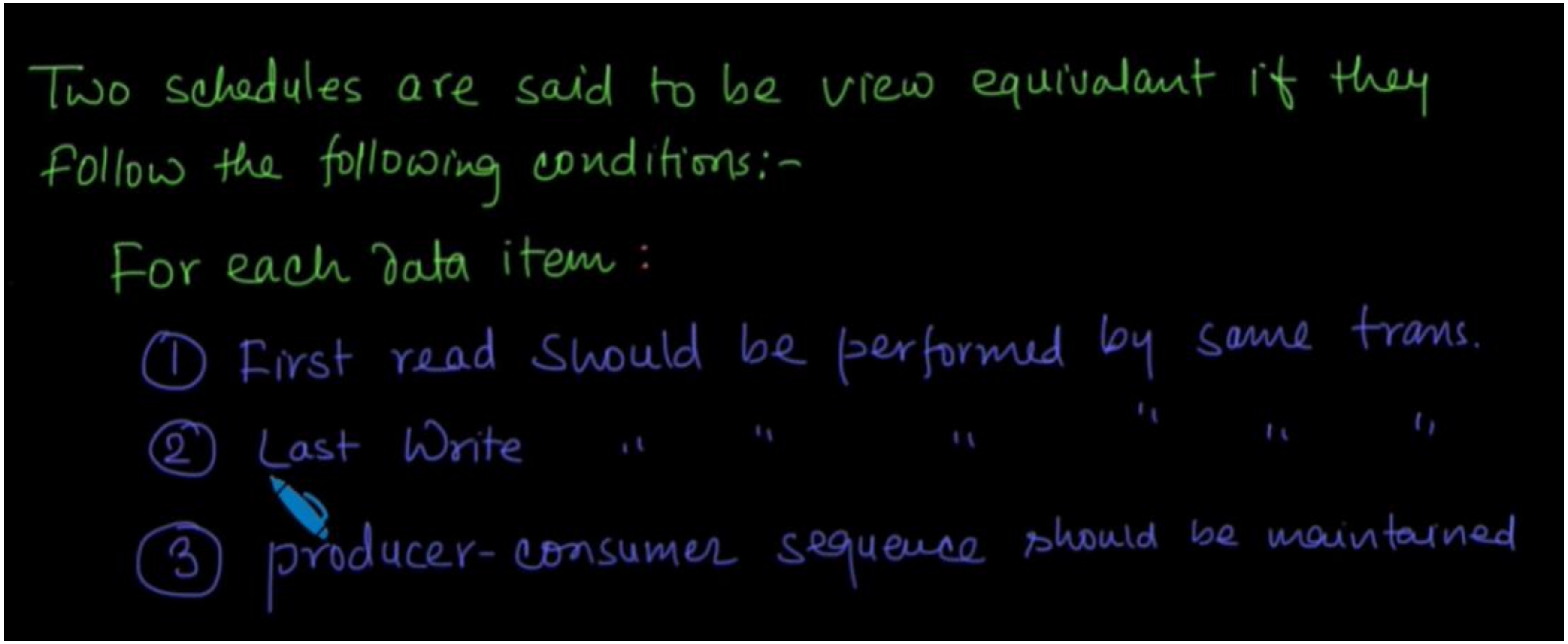
Let

- T1   T2
- A   B
- S   S’
- R   W

- R -   S : T1   A   -----   S’:   T1   A   (first)
- W - S: T1 A   -----   S’: T1   A   (last)
- Producer Consumer

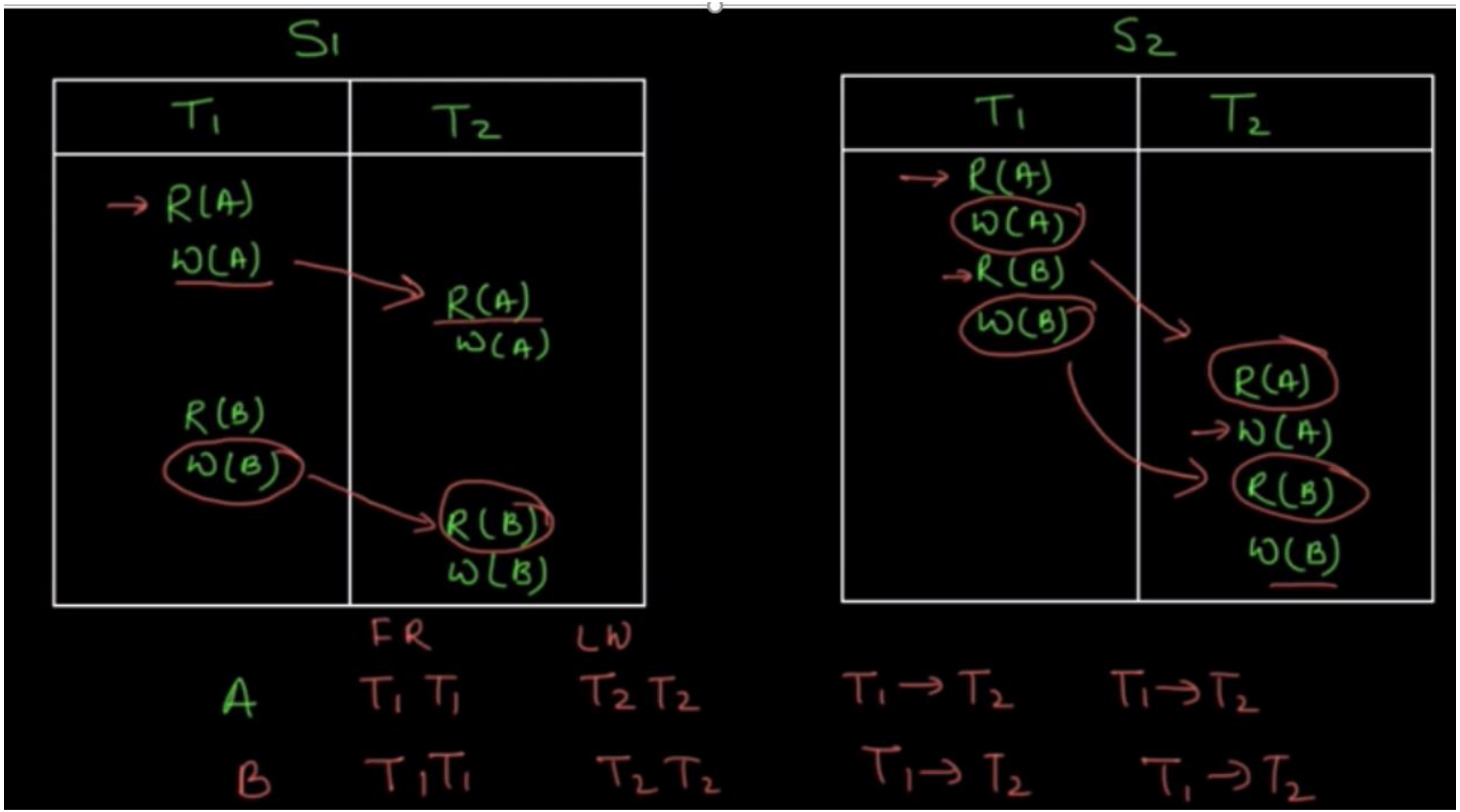
  S: T1   A   W   T2   A   R   ------ S’ : T1   A   W   T2   A   R

- ie



- S1: R1(A), W2(B)
- S2: R2(A), W1(B)
- S1 : W1(B) R2(B)

Suppose B is a dataitem produced by transaction T2 and used by transaction T1 in S2 - not view serializable



S1 S2 are view serializable
Every conflict serializable schedule is also view serializable.
But a schedule which is view-serializable but *not* conflict serializable.

- Like conflict serializability we can also draw precedence graph
- For each dataitem

  edge from first read transaction to first write transaction

  edges to last write transaction.

Prove whether the following schedule is view serializable

```
S' : read1(A), write2(A), read3(A), write1(A), write3(A)
```

- Draw the precedence graph and determine conflict serializability

Example Schedule (Schedule A)

T_1	T_2	T_3	T_4	T_5
	read(X)			
read(Y)				
read(Z)				
				read(V)
				read(W)
				read(W)
	read(Y)			
	write(Y)			
		write(Z)		
read(U)				
			read(Y)	
			write(Y)	
			read(Z)	
			write(Z)	
read(U)				
write(U)				

7. Concurrency Control
i) Lock-Based Protocols

- A lock is a mechanism to control concurrent access to a data item

- Data items can be locked in two modes :
  1. *exclusive (X) mode*.

     Data item can be both read and write.

     X-lock is requested using **lock-X** instruction.
  2. *shared (S) mode*.

     Data item can only be read.

     S-lock is requested using **lock-S** instruction.

- Lock requests are made to concurrency-control manager. Transaction can proceed only after request is granted.

Lock-compatibility matrix

	S	X
S	true	false
X	false	false

- A transaction may be granted a lock on an item if the requested lock is compatible with locks already held on the item by other transactions

- Any number of transactions can hold shared locks on an item.

- But if any transaction holds an exclusive lock on the item no other transaction may hold any lock on the item.

- To access a data item, transaction must first lock that item.

- If the data item is already locked by another transaction in an incompatible mode, the concurrency control manager will not grant the lock.

- The requesting transaction must wait till all incompatible locks held by other transactions have been released.

- The lock is then granted.

Example

- Let A and B be two accounts that are accessed by transactions T1and T2.
- Transaction T1 transfers 50 from account B to account A.
- Transaction T2 displays the total amount of money in accounts A and B—that is, the sum A + B

T1
read B
B-50
Update B
____
Read A
A+50
Update A

T_1: 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
Read A
Read B
Display A+B

T2: lock-S(A)

read(A)
unlock(A)
lock-S(B)

read(B)
unlock(B)
display($A + B$)

- Suppose that the values of accounts A and B are 100 and 200, respectively.
- If these two transactions are executed serially, in the order T1, T2 then transaction T2 will display the value 300.

- If, the transactions are executed concurrently

- T2 displays 250, which is incorrect.

- The reason for this mistake is that
  T1 unlocked data item B too early,
  as a result of which T2
  saw an inconsistent state.

T_1	T_2
lock-X(B)	
read(B) $B := B - 50$ write(B) unlock(B)	
	lock-S(A)
	read(A) unlock(A) lock-S(B)
	read(B) unlock(B) display($A + B$)
lock-X(A)	
read(A) $A := A - 50$ write(A) unlock(A)	

- But it is not possible with T3 and T4.

T_3: lock-X(B);
read(B);
$B := B - 50$;
write(B);
lock-X(A);
read(A);
$A := A + 50$;
write(A);
unlock(B);
unlock(A).

T_4: lock-S(A);
read(A);
lock-S(B);
read(B);
display($A + B$);
unlock(A);
unlock(B).

- Locking can lead to an undesirable situation - deadlock
- Consider the schedule

T_3	T_4
lock-X(B)	
read(B)	
$B := B - 50$	
write(B)	
	lock-S(A)
	read(A)
	lock-S(B)
lock-X(A)	

- Since T3 is holding an exclusive mode lock on B and T4 is requesting a shared-mode lock on B, T4 is waiting for T3 to unlock B.
- Similarly, since T4 is holding a shared-mode lock on A and T3 is requesting an exclusive-mode lock on A, T3 is waiting for T4 to unlock A.
- Thus, we have arrived at a state where neither of these transactions can ever proceed with its normal execution.
- This situation is called deadlock. When deadlock occurs, the system must roll back one of the two transactions.

Two-Phase Locking Protocol (2PL)

- This protocol ensures serializability. There are 2 phases
  - 1. Growing phase. A transaction acquire locks, but not release lock.
  - 2. Shrinking phase. A transaction release locks, but not acquire any new locks.
- Initially, a transaction is in the growing phase. The transaction acquires locks as needed.
- Once the transaction releases a lock, it enters the shrinking phase, and it can issue no more lock requests.
- The point at which the growing phase ends is called as locking point.

Consider the schedule

Transaction T_1:

- Growing Phase is from steps 1-3.
- Shrinking Phase is from steps 5-7.
- Lock Point at 3

Transaction T_2:

- Growing Phase is from steps 2-6.
- Shrinking Phase is from steps 8-9.
- Lock Point at 6

	T_1	T_2
1	lock-S(A)	
2		lock-S(A)
3	lock-X(B)	
4	…….	……
5	Unlock(A)	
6		Lock-X(C)
7	Unlock(B)	
8		Unlock(A)
9		Unlock(C)
10	…….	……

- Transactions can be ordered according to their lock points.
- This ordering is a serializability ordering for the transactions.
- By this, the two-phase locking protocol ensures conflict serializability but limit the amount of concurrency.
- But two-phase locking does not ensure
  - freedom from deadlock, and
  - cascading rollback.

T_5	T_6
lock-X(A) read(A) lock-S(B) read(B) write(A) unlock(A)	
	lock-X(A) read(A) write(A) unlock(A)

Types of 2PL

1. Strict 2PL

- This requires that all Exclusive(X) Locks held by the transaction be released after the transaction Commits.
- This ensures recoverable and Cascadeless Rollbacks .
- But deadlocks are possible.

2. Rigorous 2PL

- All Exclusive(X) and Shared(S) Locks held by the transaction be released until after the Transaction Commits.
- Rigorous is more restrictive.

3. Conservative 2PL

- Lock all the data items to access before the transaction begins execution.
- If any of the predeclared items needed cannot be locked, the transaction should not lock any of the items, instead it waits until all the items are available for locking.
- Conservative 2-PL is Deadlock free.

ii) Time stamp based protocol

- Timestamp is a
  - ✓unique identifier
  - ✓ created by the DBMS
  - ✓to identify a transaction.

- They are usually assigned in the order in which they are submitted to the system.

- If a transaction Ti has been assigned timestamp TS(Ti), and a new transaction Tj enters the system, then TS(Ti) < TS(Tj).

- There are two simple methods for implementing this scheme:
  1. System clock
  2. logical counter

- Based on the timestamp, system will produce a schedule to ensure serializability

- To implement this scheme, two timestamp values will be assigned for each data item Q :

  - W-timestamp(Q)

    denotes the largest timestamp of any transaction that executed write(Q) successfully.

  - R-timestamp(Q)

    denotes the largest timestamp of any transaction that executed read(Q) successfully.

Suppose that transaction Ti issues read(Q)

(Consider Conflict operations of read)

- If (TS(Ti) < W-timestamp(Q))

  abort Ti and roll back.

  else

  {

  read(Q)

  R-timestamp(Q) = TS(Ti)

  }

- Suppose that transaction Ti issues wriite(Q)

```
If (TS(Ti) < W-timestamp(Q) or R-timestamp(Q))
    abort Ti and  roll back.
    else
     {
       write(Q)
       W-timestamp(Q) = TS(Ti)
     }
```

- If a transaction is rolled back, the system assigns it a new timestamp and restarts it.

- The timestamp-ordering protocol ensures conflict serializability. This is because conflicting operations are processed in timestamp order.

- The protocol ensures freedom from deadlock, since no transaction ever waits.

- However, there is a possibility of starvation of long transactions if a sequence of conflicting short transactions causes repeated restarting of the long transaction.

Thomas' Write Rule

- This is a modified version of the timestamp-ordering protocol in which obsolete write operations can be ignored.
- The protocol rules for read operations remain unchanged.
- The protocol rules for write operations, however, are slightly different from the timestamp-ordering protocol.

- Suppose that transaction Ti issues write(Q).
    1. If TS(Ti) < R-timestamp(Q), then the value of Q that Ti is producing was previously needed, and it had been assumed that the value would never be produced. Hence, the system rejects the write operation and rolls Ti back.

    2. If TS(Ti) < W-timestamp(Q), then Ti is attempting to write an obsolete value of Q. Hence, this write operation can be ignored.

    3. Otherwise, the system executes the write operation and sets W-timestamp(Q) to TS(Ti).

Suppose that transaction Ti issues write(Q)

```
If (TS(Ti) < R-timestamp(Q))
        abort Ti and  roll back.
If (TS(Ti) < W-timestamp(Q) )
        ignore write(Q)
else
        {
           write(Q)
          W-timestamp(Q) = TS(Ti)
        }
```

- By deleting obsolete write operations from the transactions , Thomas' write rule ensures view serializability.

- This modification of transactions makes it possible to generate serializable schedules that would not be possible under the other protocols .

UNIT V

TOPICS

- Distributed Databases
- Homogeneous and Heterogeneous Databases
- Distributed Data Storage
- Distributed Transactions
- Commit Protocols
- Concurrency Control in Distributed Databases
- Availability
- Distributed Query Processing
- Heterogeneous Distributed Databases
- Cloud-Based Databases
- Directory Systems

1. Distributed Database System

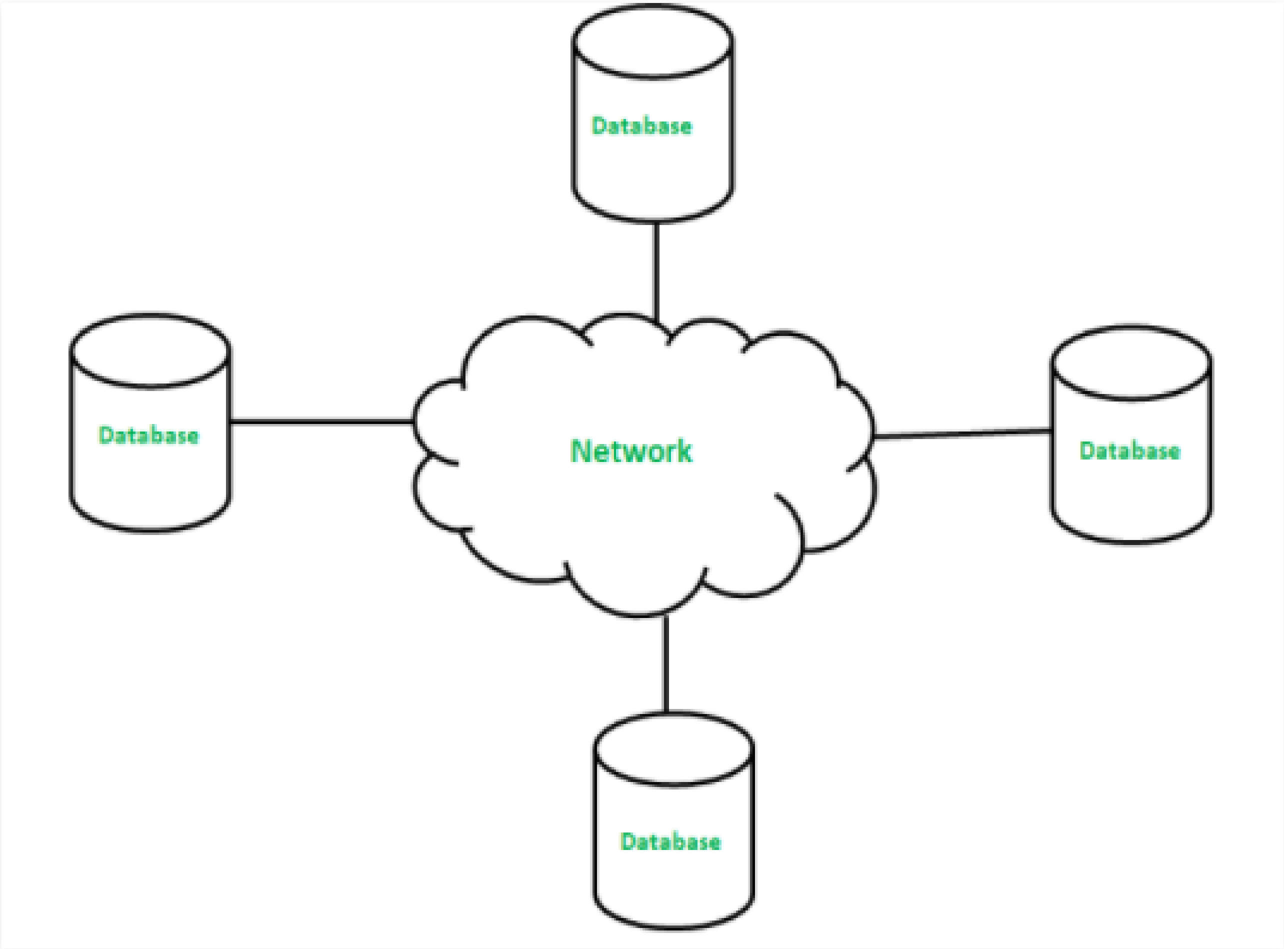
- A Distributed database is defined as a
  logically related collection of data
  that is physically distributed
  over a computer network
  on different sites.

- A distributed database system consists of loosely coupled sites that share no physical component

- Database systems that run on each site are independent of each other

- Transactions may access data at one or more sites

- This distribution of data is the cause of many difficulties in transaction processing and query processing.


Database
Database
Network
Database
Database

2. Homogeneous & Heterogeneous Distributed Databases

Homogeneous distributed database

- All sites have identical software
- Sites are aware of each other and agree to cooperate in processing user requests.
- Each site surrenders part of its autonomy in terms of right to change schemas or software
- Appears to user as a single system

Heterogeneous distributed database

- Different sites may use different schemas and software
  - ➢ Difference in schema is a major problem for query processing
  - ➢ Difference in software is a major problem for transaction processing

- Sites may not be aware of each other and may provide only limited facilities for cooperation in transaction processing

Types of Distributed Databases:

Homogeneous	Heterogeneous
i) Share a common Global Schema.	ii) diff. sites can have diff. schema
ii) Run identical DBMS S/W.	ii) Run diff. DBMS S/W.
iii) Each site provides part of its autonomy in terms of right to change schema or S/W.	iii) Each Site maintains its own right to change the schema or S/W.
iv) Same S/W - No Problem in transaction processing.	iv) Diff. S/W - Major Problem in transaction Processing.
v) Same schema - No Problem in Query Processing.	v) Diff. Schema - Problem in Query Processing.

3. Distributed Data Storage

- There are 3 approaches for storing a relation r in the distributed database:

1. Replication.

   - The system maintains several identical replicas (copies) of the relation, and stores each replica at a different site.

   - The alternative to replication is to store only one copy of relation r.

2. Fragmentation.

   The system partitions the relation into several fragments, and stores each fragment at a different site.

3. Hybrid approach : Fragmentation and replication can be combined

   - A relation can be partitioned into several fragments and there may be several replicas of each fragment.

Advantages of replication

- Availability.
  - If one of the sites containing relation r fails, then the relation r can be found in another site.
  - Thus, the system can continue to process queries involving r, despite the failure of one site.
- Increased parallelism.
  - In the case where the majority of accesses to the relation r result in only the reading of the relation, then several sites can process queries involving r in parallel.
  - The more replicas of r , the greater the chance that the needed data will be found in the site where the transaction is executing.
  - Hence, data replication minimizes movement of data between sites.

Disadvantages

- Increased overhead on update.
  - The system must ensure that all replicas of a relation r are consistent; otherwise, erroneous computations may result.
  - Thus, whenever r is updated, the update must be propagated to all sites containing replicas.
  - The result is increased overhead.
  - In general, replication
    - ✓ enhances the performance of read operations and increases the availability of data .
    - ✓ However, update operations incur greater overhead.
  - Controlling concurrent updates by several transactions to replicated data is more complex than in centralized systems

Fragmentation

- In this approach, the relations are divided into smaller parts and each of the fragments is stored in different sites where they're required.
- It must be made sure that the fragments are such that they can be used to reconstruct the original relation (i.e, there isn't any loss of data).
- Fragmentation is advantageous as it doesn't create copies of data, consistency is not a problem.
- Fragmentation of relations can be done in two ways:

➢ **Horizontal fragmentation** – Splitting by rows

  ➢ The relation is fragmented into groups of tuples so that each tuple is assigned to at least one fragment.

➢ **Vertical fragmentation** – Splitting by columns

  ➢ The schema of the relation is divided into smaller schemas. Each fragment must contain a common candidate key so as to ensure lossless join.

branch-name	*account-number*	*balance*
Hillside	A-305	500
Hillside	A-226	336
Hillside	A-155	62

$account_1 = \sigma_{branch\text{-}name=\text{“Hillside”}}(account)$

branch-name	*account-number*	*balance*
Valleyview	A-177	205
Valleyview	A-402	10000
Valleyview	A-408	1123
Valleyview	A-639	750

$account_2 = \sigma_{branch\text{-}name=\text{“Valleyview”}}(account)$

branch-name	*customer-name*	*tuple-id*
Hillside	Lowman	1
Hillside	Camp	2
Valleyview	Camp	3
Valleyview	Kahn	4
Hillside	Kahn	5
Valleyview	Kahn	6
Valleyview	Green	7

$deposit_1 = \Pi_{branch\text{-}name,\ customer\text{-}name,\ tuple\text{-}id}(employee\text{-}info)$

account number	*balance*	*tuple-id*
A-305	500	1
A-226	336	2
A-177	205	3
A-402	10000	4
A-155	62	5
A-408	1123	6
A-639	750	7

$deposit_2 = \Pi_{account\text{-}number,\ balance,\ tuple\text{-}id}(employee\text{-}info)$

Transparency: In distributed system, the user should be able to access the database exactly as if the system were local. Hiding details such as data storage, how data can be accessed is called as Data transparency.

i) Location transparency
ii) Fragmentation transparency
iii) Replication transparency
iv) Naming transparency

4. Distributed Transactions

- Transaction may access data at several sites.

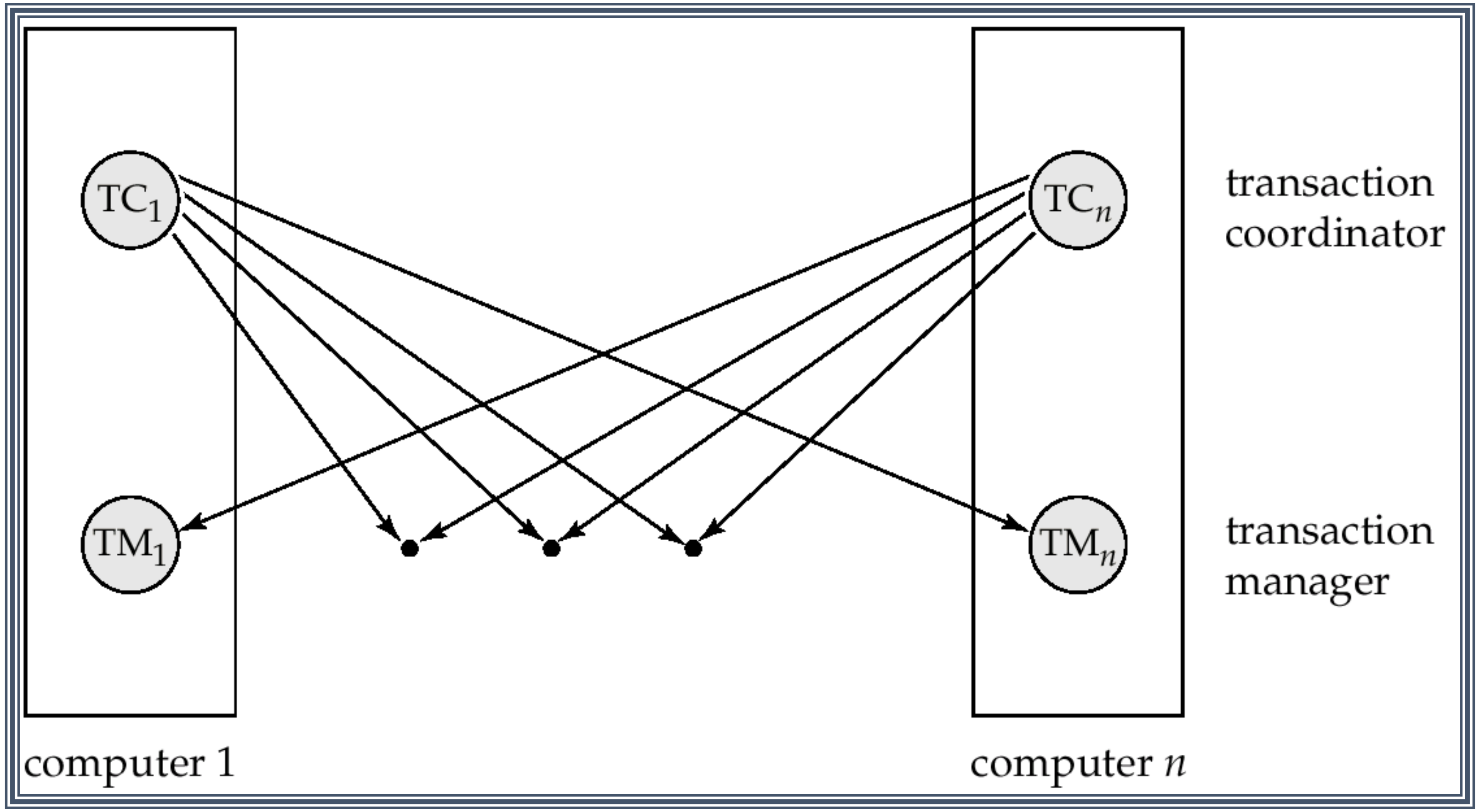
- Each site has
  - transaction manager
  - transaction coordinator

- Transaction manager is responsible for

  Managing the execution of transactions that access data in that site

- Transaction coordinator is responsible for

  Coordinating the execution of transactions that originates at the site


TC$_1$
TC$_n$
transaction coordinator
TM$_1$
TM$_n$
transaction manager
computer 1
computer n

Failures unique to distributed systems:

- Failure of a site
- Loss of massages
  - Handled by network transmission control protocols such as TCP-IP
- Failure of a communication link
  - Handled by network protocols, by routing messages via alternative links
- **Network partition**
  - A network is said to be **partitioned** when it has been split into two or more subsystems that lack any connection between them

5. Commit Protocols

- Commit protocols are used to ensure atomicity across sites
  - a transaction which executes at multiple sites must either be committed at all the sites, or aborted at all the sites.

- It is not acceptable to have a transaction committed at one site and aborted at another

- The *two-phase commit* (2 *PC*) protocol is widely used

- The *three-phase commit* (3 *PC*) protocol is
  - more complicated and more expensive,
  - but avoids some drawbacks of two-phase commit protocol.

Recovery System must ensure ATOMICITY.
(ALL or NONE)

Commit Protocol:

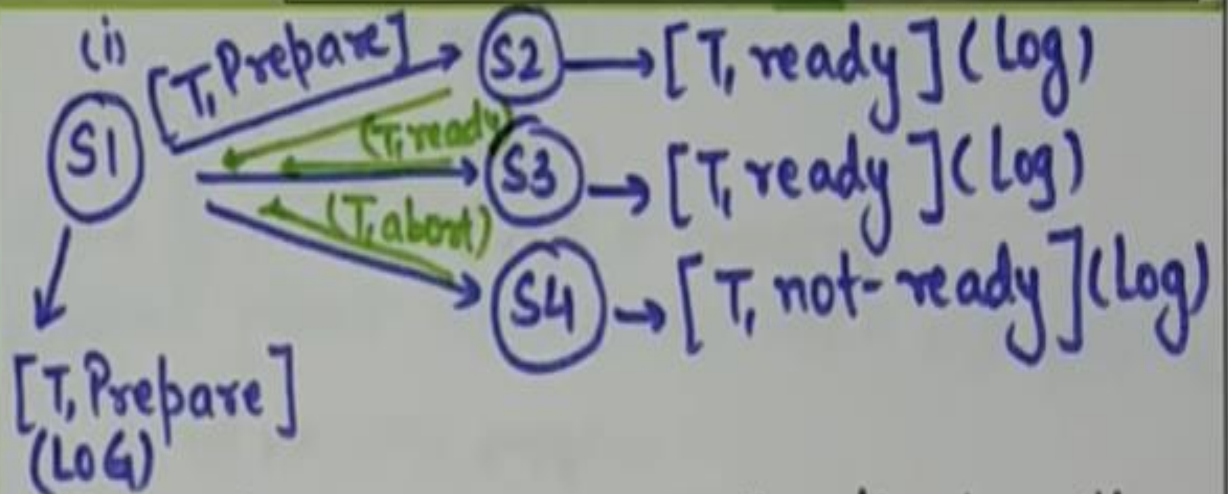
(i) **Two-Phase Commit:** Two Phases of Commit

Protocol → Voting Phase
Protocol → Decision Phase

Transaction T is initiated at Site S1 (C.S), S2, S3 and S4] P.S → S2, S3, S4

VOTING PHASE: In this, participating sites vote on whether they are ready to commit the transaction or not.

(i)



DECISION PHASE: In this, the Coordinator Site decides whether the transaction can be commited or has to be aborted.

↳ i) [Ready, T] message from all P.S
↳ Commit

S1 → S2 →, S3 →, [T, Commit]

ii) At least one [not-ready, T], Abort the transaction T.

S1 →, →, [T, Abort]

Failure of Participating Site:

- Site fails before sending "Ready T" (Transaction is Aborted)
- Site fails after sending "Ready T" (Proceed in Normal Way)

Recovery Process when participating site Restarts after failure: LOG file is checked

- i) LOG → [T, Commit] Redo T / [T, Abort] Undo T → ack (C.S)

ii) LOG → [T, ready] → Site Contacts the C.S to determine whether to Commit or Abort T.

iii) NO Such Record in LOG → Undo T and Abort the Transaction

Failure of Coordinator's Site:

All the Participating Sites communicate with each other to determine Status of 'T'.

- (i) Any Site → [T, Commit] Commited / [T, Abort] Aborted

(ii) No [T, Ready] in log Abort T

iii) No case holds → P.S waits until C.S is recovered.

P.S are in blocking stage.

Recovery Process when Coordinator Site Restarts after failure: LOG File is checked.

- i) LOG → [T, Commit] RedoT / [T, Abort] UNDOT → All P.S.

ii) LOG → [T, Prepare] → Abort T → All P.S.

2PC

Phase 1 – Voting Phase

- Coordinator asks all participants to *prepare* to commit transaction *T*
  - C adds the records <**prepare** *T*> to the log
  - sends **prepare** *T* messages to all sites at which *T* executed

- Upon receiving message, transaction manager at site determines if it can commit the transaction
  - if not,
    - add a record <**not ready** *T*> to the log
    - and send **abort** *T* message to C_i
  - if the transaction can be committed, then:
    - add the record <**ready** *T*> to the log
    - send **ready** *T* message to C_i

Phase II – Decision Phase

- *T* can be committed if *C* received a **ready** *T* message from all the participating sites: otherwise *T* must be aborted.

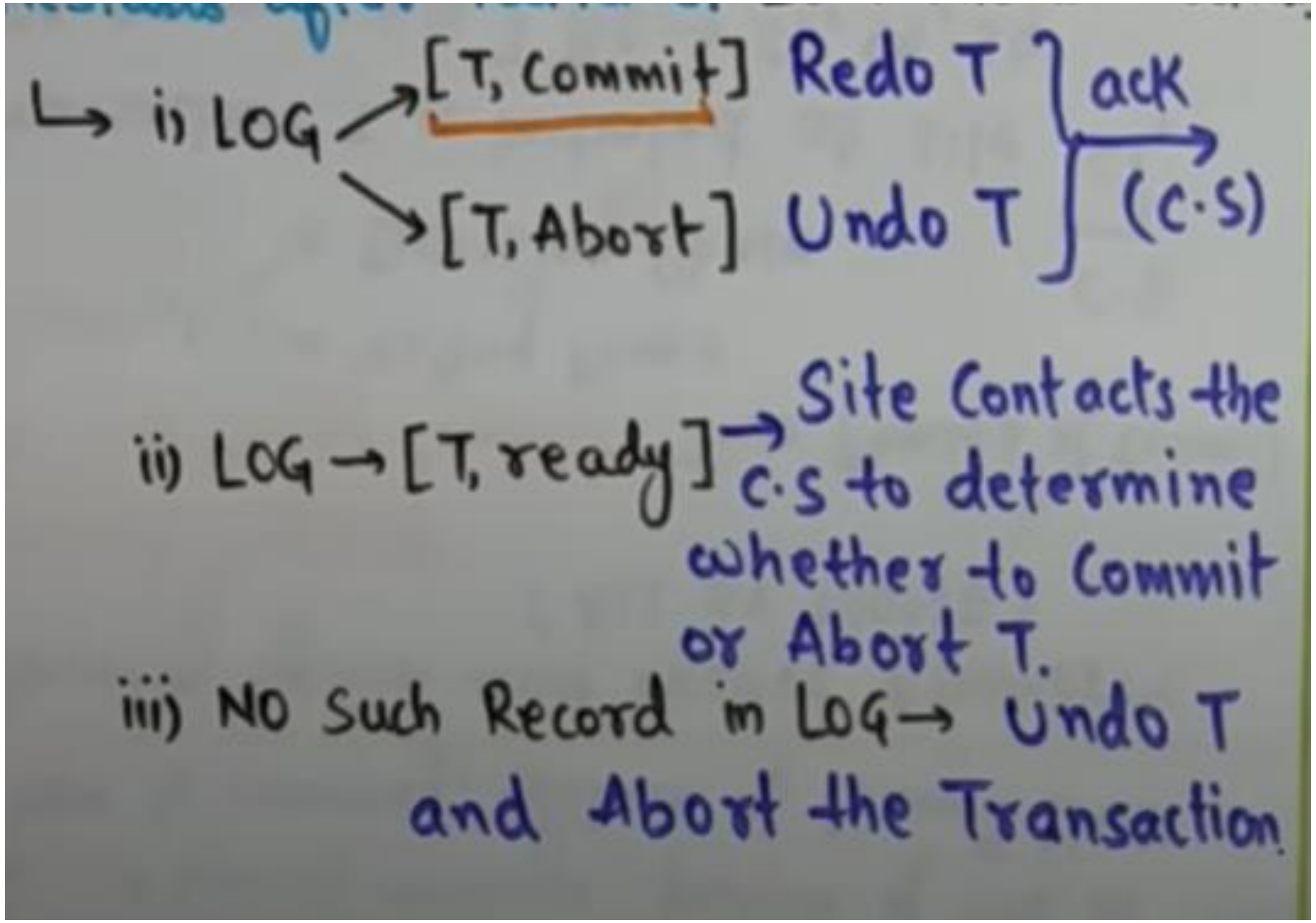
- Coordinator adds a decision record, <**commit** *T*> or <a**bort** *T*>, to the log .

- Coordinator sends a message to each participant informing it of the decision commit or abort

- Participants take appropriate action locally.

Handling of Failures - Site Failure

When site S_i recovers, it examines its log to determine the fate of transactions active at the time of the failure.



Handling of Failures- Coordinator Failure

- If coordinator fails while the commit protocol for *T* is executing then participating sites must decide on *T*'s fate:
  1. If an active site contains a <**commit** *T*> record in its log, then *T* must be committed.
  2. If an active site contains an <**abort** *T*> record in its log, then *T* must be aborted.
  3. If some active participating site does not contain a <**ready** *T*> record in its log, then the failed coordinator C_i cannot have decided to commit *T*. Can therefore **abort *T*.**
  4. If none of the above cases holds, then all active sites must have a <**ready** *T*> record in their logs, but no additional control records (such as <**abort** *T*> of <**commit** *T*>). In this case active sites must wait for C_i to recover, to find decision.

- **Blocking problem** : active sites may have to wait for failed coordinator to recover.

Participating sites communicate with each other to determine status of 'T'.

↳ (i) Any site → [T, Commit] Commited
→ [T, Abort] Aborted

(ii) No [T, Ready] in log Abort T

P.S are in blocking stage. iii) No case holds → P.S waits until C.S is recovered.

Recovery Process when Coordinator Site Restarts after failure: LOG File is checked.

↳ i) LOG → [T, Commit] RedoT
→ [T, Abort] UNDOT
→ All P.S.

ii) LOG → [T, Prepare] → Abort T → All P.S.

3 Phase Commit Protocol

- Phase 1: Identical to 2PC Phase 1.
  - Every site is ready to commit if instructed to do so
- Phase 2 of 2PC is split into 2 phases, Phase 2 and Phase 3 of 3PC
  - In phase 2
    - ➢ coordinator makes a decision as in 2PC (called the pre-commit decision)
    - ➢ and records it in multiple (at least K) sites
- In phase 3,
  - ➢ coordinator sends commit/abort message to all participating sites
- Under 3PC, knowledge of pre-commit decision can be used to commit despite coordinator failure
  - ➢ Avoids blocking problem as long as < K sites fail
- Drawbacks:
  - ➢ higher overheads

6. Concurrency Control

- Updates to be done on all replicas of a data item.
- If any site containing a replica of a data item has failed, updates to the data item cannot be processed.

Two approaches

- Lock based Approach
  - ➢Single lock Manager Approach
  - ➢Distributed lock Manager Approach
- Timestamped Approach

Single lock manager Approach

- System maintains a *single* lock manager that resides in a *single* chosen site, say S_i
- When a transaction needs to lock a data item, it sends a lock request to S_i and lock manager determines whether the lock can be granted immediately
  - If yes, lock manager sends a message to the site which initiated the request
  - If no, request is delayed until it can be granted
  - Sends the message at the time it can grant the request.
- The transaction can read the data item from *any* one of the sites at which a replica of the data item resides.
- Writes must be performed on all replicas of a data item

Advantages :

- Simple implementation
  - ➢ This requires two messages for handling lock requests and one message for handling unlock requests.
- Simple deadlock handling
  - ➢ Since all lock and unlock requests are made at one site, the deadlock-handling algorithms can be applied directly.

Disadvantages:

- Bottleneck:
  - ➢ lock manager site becomes a bottleneck since all requests must be processed there
- Vulnerability:
  - ➢ If the site Si fails, the concurrency controller is lost.
  - ➢ Either processing must stop, or a recovery scheme must be used so that a backup site can take over lock management from Si

Distributed Lock Manager Approach

- lock-manager function is distributed over several sites.
- Each site maintains a local lock manager whose function is to handle the lock and unlock requests for those data items that are stored in that site.

Data not replicated

- When a transaction wishes to lock a data item Q that is not replicated and resides at site Si, a message is sent to the lock manager at site Si requesting a lock.
- If data item Q is locked in an incompatible mode, then the request is delayed until it can be granted.
- Once it has determined that the lock request can be granted, the lock manager sends a message back to the initiator indicating that it has granted the lock request.

- Advantage:

  work is distributed and can be made robust to failures

- Disadvantage:

  deadlock detection is more complicated

- Several variants of this approach – To replicate data

  - Primary copy
  - Majority protocol
  - Biased protocol
  - Quorum consensus

Primary Copy

- Choose one replica of data item to be the **primary copy**.
  - Site containing the replica is called the **primary site** for that data item
  - Different data items can have different primary sites
- When a transaction needs to lock a data item *Q*, it requests a lock at the primary site of *Q*.
  - Implicitly gets lock on all replicas of the data item

Benefit

- Concurrency control for replicated data handled similarly to unreplicated data - simple implementation.

Drawback

- If the primary site of *Q* fails, *Q* is inaccessible even though other sites containing a replica may be accessible

Majority Protocol

- If Q is replicated at n sites, then a lock request message must be sent to more than half of the n sites in which Q is stored.
- The transaction does not operate on Q until it has obtained a lock on a majority of the replicas of Q.
- When writing the data item, transaction performs writes on *all* replicas.

Benefit

- Can be used even when some sites are unavailable

Drawback

- more complicated to implement
- Requires $2(n/2 + 1)$ messages for handling lock requests, and $(n/2 + 1)$ messages for handling unlock requests
- Complex deadlock handling

Biased protocol

- Requests for shared locks are handled differently than requests for exclusive locks.
- **Shared locks**. When a transaction needs to lock data item *Q*, it simply requests a lock on *Q* from the lock manager at one site containing a replica of *Q*.
- **Exclusive locks**. When transaction needs to lock data item *Q*, it requests a lock on *Q* from the lock manager at all sites containing a replica of *Q*.
- Advantage - imposes less overhead on **read** operations.
- Disadvantage - additional overhead on writes. Similar to biased protocol

Quorum Consensus Protocol

- A generalization of both majority and biased protocols
- Each site is assigned a weight.
  - Let S be the total of all site weights
- Choose two values read quorum Q_r and write quorum Q_w
  - Such that $Q_r + Q_w > S$ and $2 * Q_w > S$
- Quorums can be chosen separately for each item
- Each read must lock enough replicas that the sum of the site weights is $>= Q_r$
- Each write must lock enough replicas that the sum of the site weights is $>= Q_w$

- Ex :

- Sites $\boxed{S1}$ $\boxed{S2}$ $\boxed{S3}$ $\boxed{S4}$
- Weight 3 1 2 4
- Total weight S = 10
- Let Qr = 5 Qw = 6 $[Q_r + Q_w > S$ and $2 * Q_w > S]$

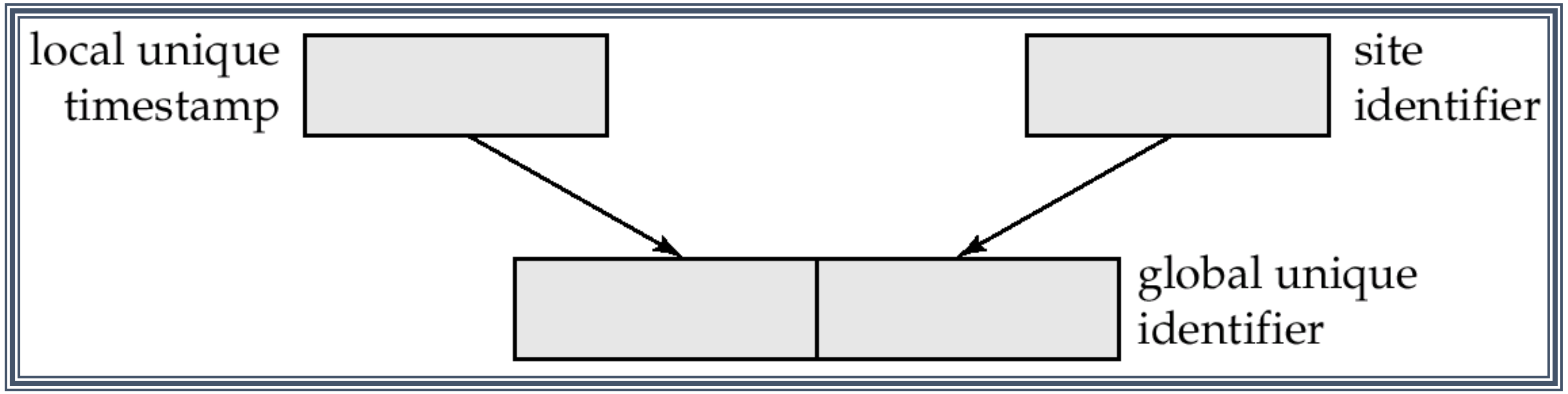
- Transaction can read dataitems if sites s1 and s3 is locked because the sum of weights of s1 and s3 is 5 which is >= Qr
- But transaction can't write dataitems if sites s1 and s3 is locked

Benefits

- It can permit the cost of either read or write locking to be selectively reduced by appropriately defining the read and write quorums.
- small read quorum - reads need to obtain fewer locks
- high write quorum - obtain more locks.
- If higher weights are defined to sites those are less likely to fail, fewer sites need to be accessed for acquiring locks.

Timestamping

- Each transaction must be given a unique timestamp
- Main problem: how to generate a timestamp in a distributed fashion
  - Each site generates a unique local timestamp using either a logical counter or the local clock.
  - Global unique timestamp is obtained by concatenating the unique local timestamp with the unique identifier.



7. Distributed Query Processing

- There are many methods/strategies for processing a query.
- Choose a good strategy for processing the query
  - The primary criteria is minimum time to compute the answer.
- For centralized systems the main factor to consider
  - is the number of disk accesses.
- In a distributed system, we must take into account several other factors
  - The cost of data transmission over the network.
  - The potential gain in performance

    several sites process parts of the query in parallel.

i) Query Transformation

- Consider a simple query
  - Find all the tuples in the account relation.
- Although the query is simple, processing it in distributed database is not easy, since the account relation may be
  - replicated
  - fragmented
  - or both
- If the account relation is replicated, choose the replica for which the transmission cost is lowest.
- If a replica is fragmented, the choice is not so easy to make,
  - since we need to compute several joins or unions to reconstruct the account relation.
- In this case, the number of strategies for the simple example may be large.

Query Transformation - Horizontal fragmentation

- Assume account relation is horizontally fragmented into $account_1$ and $account_2$

$$account_1 = \sigma_{\textit{branch-name} = \text{“chennai”}}(account)$$
$$account_2 = \sigma_{\textit{branch-name} = \text{“madurai”}}(account)$$

- User may write a query such as

$$\sigma_{\textit{branch-name} = \text{“chennai”}}(account)$$

- Since account is defined as: $account_1 \cup account_2$

$$\sigma_{\text{branch-name} = \text{“chennai”}}(account_1 \cup account_2)$$

which is optimized into

$$\sigma_{\textit{branch-name} = \text{“chennai”}}(account_1)$$
$$\cup$$
$$\sigma_{\text{branch-name} = \text{“chennai”}}(account_2)$$

- This contains two sub expressions
- First expression is transformed into

$$\sigma_{\textit{branch-name} = \text{“chennai”}} (\sigma_{\textit{branch-name} = \text{“chennai”}} (\textit{account}))$$

- Second expression is transformed into

$$\sigma_{\textit{branch-name} = \text{“chennai”}} (\sigma_{\textit{branch-name} = \text{“madurai”}} (\textit{account}))$$

ii)Simple Join Processing

- Must consider following factors:
  - amount of data being shipped/transferred
  - cost of transmitting a data block between sites
  - relative processing speed at each site

- Consider the following relational algebra expression in which the three relations are neither replicated nor fragmented

$$r1 \bowtie r2 \bowtie r3$$

r1 is stored at site S_1

r2 at S_2

r3 at S_3

(eg. account, depositor, branch)

- For a query issued at site S_I, the system needs to produce the result at site S_I

Different strategies

1. Ship copies of a relations r2 and r3 to site S1

   compute $r1 \bowtie r2 \bowtie r3$ at site S1

2. Ship a copy of the r1 relation to site S_2 and

   compute $temp_1 = r1 \bowtie r2$ *at* S_2

   Ship $temp_1$ from S_2 to S_3

   and compute $temp_2 = temp_1 \bowtie r3$ *at* S_3

   Ship the result $temp_2$ to S_1.

- Devise similar strategies, by exchanging the roles of S_1, S_2, S_3

- Example : Choose a strategy with less amount of data to be shipped and low Communication cost

Simple Join Processing: Join is one of the most expensive operations.

eg:- $\pi_{ISBN, Price, P\_id, Pname}$ (Book $\bowtie_{Book.P\_ID = PUBLISHER.P\_ID}$ PUBLISHER)

ISBN = 15, Price = 6, P_id = 4, Pname = 50

Book is at (Site S1); PUBLISHER is at (Site S2)

500 tuples in Book and each tuple is 100 byte long = 500x100 = 50,000 bytes

100 tuples in PUBLISHER and each tuple is 150 byte long = 100 x 150 = 15,000 bytes. → o/p at (site S3)

After join, 500 tuples and each tuple is 75 bytes

Strategies:

① Transfer replicas of both relation to site S3 & process the Query.

$S1(BOOK) \xrightarrow{50,000} S3\,(65,000)$

$S2(PUB) \xrightarrow{15000} S3$

② Transfer replica of (PUBLISHER) relⁿ to Site S1 for processing at S1 and then move result to S3.

$S2(PUB) \xrightarrow{15,000} S1(BOOK)\,(o/p) \xrightarrow{37,500} S3\,(52,500 \text{ bytes})$

③ Transfer replica of (BOOK) relⁿ to Site S2 for processing at S2 and then ship result to S3.

$S1(BOOK) \xrightarrow{50,000} S2(PUB)\,(o/p) \xrightarrow{37,500} S3\,(87,500)$

iii) Semijoin Strategy

- Semijoin is denoted by $\ltimes$
- Semijoin takes the natural join of 2 relations and projects the attributes of first relation only.
- Eg

FACULTY

FacID	Name	Dept	Desigantion
1001	Usman Khalil	CS	Assistant
1002	Abdullah	Fin	Professor
1003	Rana Khalil	Econ	Professor
1004	Rana Sher Jang	Math	Lecturer

COURSE

CourseID	Title	FID
CS-502	DBMS	1001
CS-511	OOP	
CS-430	FM	1003

- Result of faculty $\ltimes$ course

FacID	Name	Dept	Desigantion
1001	Usman Khalil	CS	Assistant
1003	Rana Khalil	Econ	Professor

- Reduce communication cost by reducing the size of relation that needs to be transmitted

eg:- $\pi_{ISBN, Price, P\_ID, Pname}(BOOK \bowtie PUBLISHER)$ (Book.P_ID = PUBLISHER.P_ID)

(15) (6) (4) (50)

O/p at site (S2)

Strategies:

Step ① Project the join attribute (P_ID) of PUBLISHER at Site S2 and move it to S1.

$S2 \rightarrow \pi_{P\_id} \xrightarrow{100 \times 4} S1$ [400 bytes]

Step ② Join this transferred attribute with Book relation and transfer required attributes from result to S2.

$S1 \xrightarrow{500 \times 25} S2$ [12,500 bytes]

iv) **Join Strategies that Exploit Parallelism**

- Conside $r_1 \bowtie r_2 \bowtie r_3 \bowtie r_4$ where relation r_i is stored at site S_i.
- The result must be at site S_1.
- r1 is shipped to S2, and r1 $\bowtie$ r2 computed at S2.
- At the same time, r3 is shipped to S4, and r3 $\bowtie$ r4 computed at S4.
- Site S2 can ship tuples of $(r_1 \bowtie r_2)$ to S1
- Similarly, S4 can ship tuples of $(r_3 \bowtie r_4)$ to S1.
- Once tuples of (r1 $\bowtie$ r2) and (r3 $\bowtie$ r4) arrive at S1, the computation of $(r_1 \bowtie r_2) \bowtie (r_3 \bowtie r_4)$ can begin.
- Thus, computation of the final join result at S1 can be done in parallel with the computation of $(r_1 \bowtie r_2)$ at S2, and with the computation of $(r_3 \bowtie r_4)$ at S4.

$$r_1 \bowtie r_2 \bowtie r_3 \bowtie r_4$$

- result must be at site S_1

$A \bowtie B \bowtie C \bowtie D$

$S1 \quad S2 \quad S3 \quad S4$

X

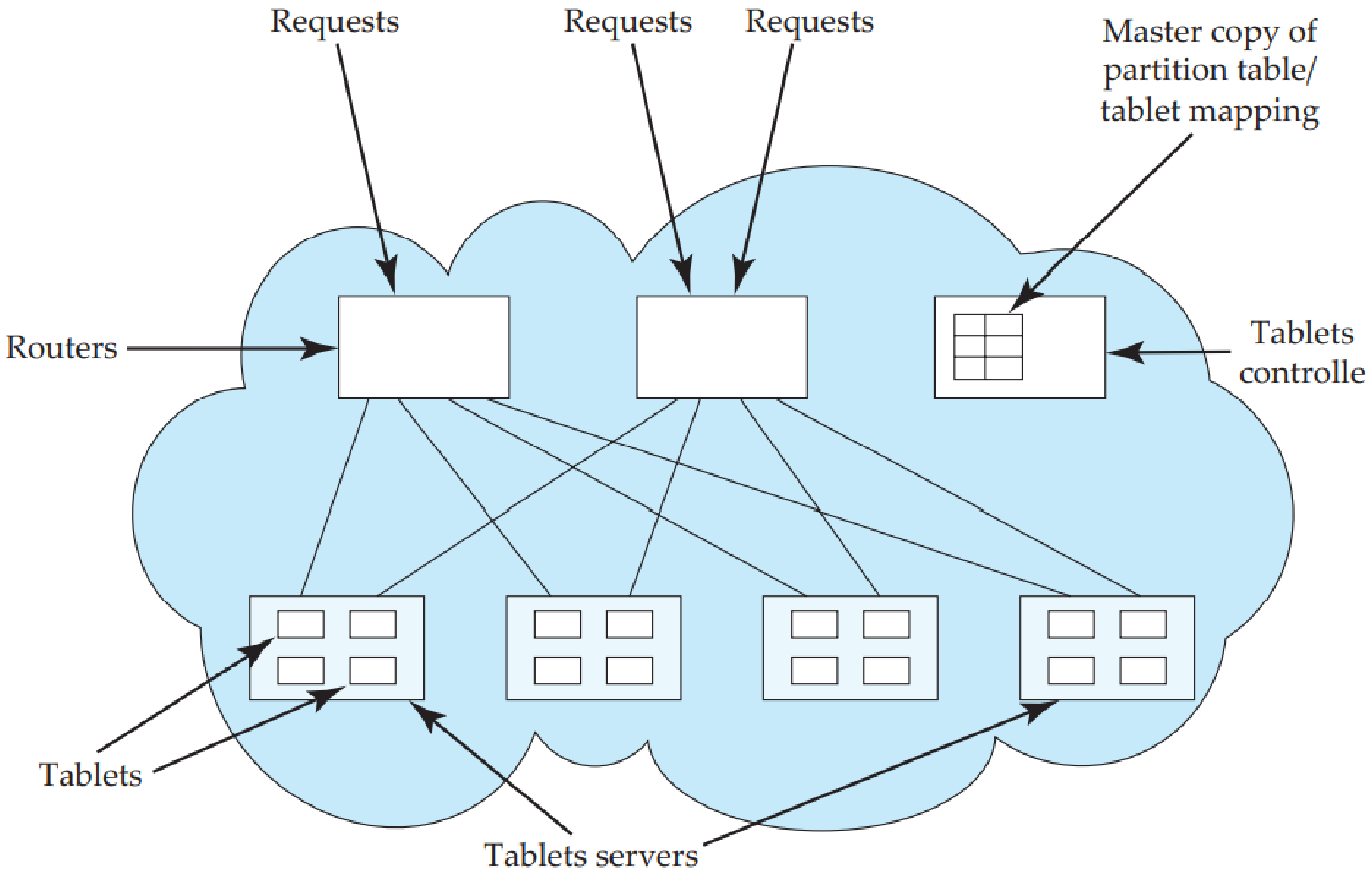
$A \longrightarrow S2[A \bowtie B] \longrightarrow S1[X \bowtie Y]$

$C \longrightarrow S4[C \bowtie D] \longrightarrow S1[X \bowtie Y]$

Y

8. Cloud-Based Databases

- Web applications that need to store and retrieve data for very large numbers of users (ranging from millions to hundreds of millions)
- It is the major driver of cloud-based databases.
- The needs of these applications differ from those of traditional database applications, since they value availability and scalability over consistency.
- Several cloud-based data-storage systems have been developed in recent years to serve the needs of such applications
  - ➢Bigtable from Google,
  - ➢Simple Storage Service (S3) from Amazon
  - ➢Cassandra from FaceBook,
  - ➢wSherpa/PNUTS from Yahoo,
  - ➢ Azure from Microsoft


Requests
Requests
Requests
Master copy of
partition table/
tablet mapping
Routers
Tablets
controlle
Tablets
Tablets servers

Data Storage Systems on the Cloud

- Applications on the Web have extremely high scalability requirements.
- Popular applications have hundreds of millions of users, and many applications have seen their load increase manyfold within a single year, or even within a few months.
- To handle the data management needs of such applications, data must be partitioned across thousands of processors.
- A number of systems for data storage on the cloud have been developed and deployed over the past few years to address data management requirements of such applications
- These include Bigtable from Google, Simple Storage Service (S3) from Amazon, which provides a Web interface to Dynamo, which is a keyvalue storage system, Cassandra, from FaceBook, wSherpa/PNUTS from Yahoo, the data storage component of the Azure environment from Microsoft, and several other systems.

i) Data Representation

- As an example of data management needs of Web applications, consider the profile of a user, which needs to be accessible to a number of different applications that are run by an organization.
- The profile contains a variety of attributes, and there are frequent additions to the attributes stored in the profile. Some attributes may contain complex data.
- A simple relational representation is often not sufficient for such complex data.
- Some cloud-based data-storage systems support XML for representing such complex data.
- Others support the JavaScript Object Notation (JSON) representation, which has found increasing acceptance for representing complex data.
- The XML and JSON representations provide flexibility in the set of attributes that a record contains, as well as the types of these attributes.
- Yet others, such as Bigtable, define their own data model for complex data including support for records with a very large number of optional columns.

- Further, many such Web applications either do not need extensive query language support, or at least, can manage without such support.
- The primary mode of data access is to store data with an associated key, and to retrieve data with that key.
- In the above user profile example, the key for user-profile data would be the user’s identifier.
- There are applications that conceptually require joins, but implement the joins by a form of view materialization.
- For example, in a social-networking application, each user should be shown new posts from all her friends.
- Unfortunately, finding the set of friends and then querying each one to find their posts may lead to a significant amount of delay when the data are distributed across a large number of machines.
- An alternative is as follows: whenever a user makes a post, a message is sent to all friends of that user, and the data associated with each of the friends is updated with a summary of the new post.
- When that user checks for updates, all required data are available in one place and can be retrieved quickly

- Thus, cloud data-storage systems are, at their core, based on two primitive functions, put(key, value), used to store values with an associated key, and get(key), which retrieves the stored value associated with the specified key.
- Some systems such as Bigtable additionally provide range queries on key values.
- In Bigtable, a record is not stored as a single value, but is instead split into component attributes that are stored separately.
- Thus, the key for an attribute value conceptually consists of (record-identifier, attribute-name). Each attribute value is just a string as far as Bigtable is concerned.
- To fetch all attributes of a record, a range query, or more precisely a prefix-match query consisting of just the record identifier, is used.
- The get() function returns the attribute names along with the values.
- For efficient retrieval of all attributes of a record, the storage system stores entries sorted by the key, so all attribute values of a particular record are clustered together.

ii)Partitioning and Retrieving Data

- Partitioning of data is, of course, the key to handling extremely large scale in data-storage systems.
- data-storage systems typically partition data into relatively small units (small on such systems may mean of the order of hundreds of megabytes). These partitions are often called tablets,
- The partitioning of data should be done on the search key, so that a request for a specific key value is directed to a single tablet; otherwise each request would require processing at multiple sites, increasing the load on the system greatly. Two approaches are used: either range partitioning is used directly on the key, or a hash function is applied on the key, and range partitioning is applied on the result of the hash function.
- The site to which a tablet is assigned acts as the master site for that tablet. All updates are routed through this site, and updates are then propagated to replicas of the tablet. Lookups are also sent to the same site, so that reads are consistent with writes.
- The partitioning of data into tablets is not fixed up front, but happens dynamically. As data are inserted, if a tablet grows too big, it is broken into smaller parts.

- It is important to know which site in the overall system is responsible for a particular tablet.
- This can be done by having a tablet controller site which tracks the partitioning function, to map a get() request to one or more tablets, and a mapping function from tablets to sites, to find which site were responsible for which tablet.
- Each request coming into the system must be routed to the correct site; if a single tablet controller site is responsible for this task, it would soon get overloaded.
- Instead, the mapping information can be replicated on a set of router sites, which route requests to the site with the appropriate tablet.

Transactions and Replication

- Data-storage systems on the cloud typically do not fully support ACID transactions.
- The cost of two-phase commit is too high, and two-phase commit can lead to blocking in the event of failures, which is not acceptable to typical Web applications.
- Sherpa/PNUTS also provides a test and-set function, which allows an update to a data item to be conditional on the current version of the data item being the same as a specified version number.
- If the current version number of the data item is more recent than the specified version number, the update is not performed.
- The test-and-set function can be used by applications to implement a limited form of validation-based concurrency control, with validation restricted to data items in a single tablet.
- A data-storage system on the cloud must be able to continue normal processing even with many sites down. Such systems replicate data (such as tablets) to multiple machines in a cluster, so that a copy of the data is likely to be available even if some machines of a cluster are down.