DATABASE MANAGEMENT SYSTEM

Data Vs Information

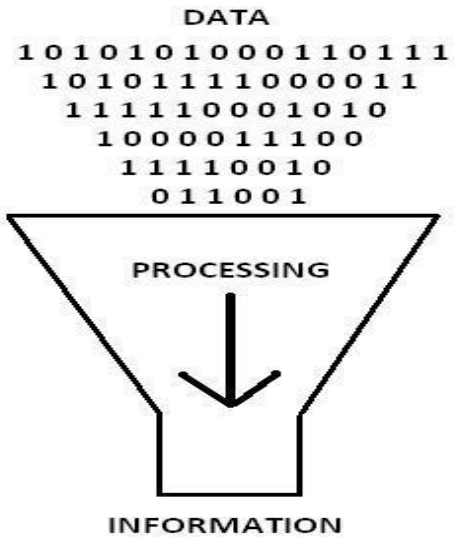
The term “**Data**” means any representation of facts, figures, symbols, concepts or instructions which is communicated interpreted or processed by human or electronic machine. Data is represented with the help of characters like alphabets (A-Z, a-z), digits (0-9) or special characters (+,-, /,*, <,>, = etc.).

Information is processed or structured data which has some meaningful values for the system or human. Information is the derived data on which decisions and actions are based. For the decision to be meaningful, the processed data must meet the following characteristics:

Timely - Information should be available when required.

Accuracy - Information should be accurate.

Completeness - Information should be complete.



Examples of Data and Information

The area and population of different cities of a country is data. If this data is organized and analysed to find out the population densities of cities, then that is information.

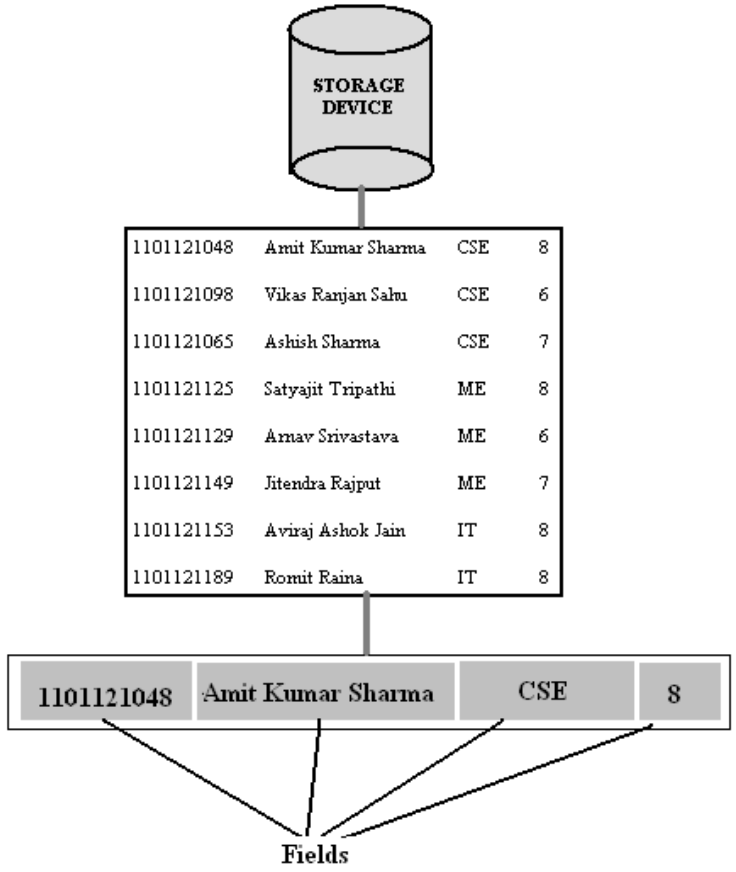
What is Data

- The history of temperature readings all over the world for the past 50 years is data. If this data is organized and analysed to find that global temperature is rising, then that is information.

Data	Information
Data is raw, unprocessed facts and figures.	Information is processed, organized, structured or presented data.
Data is used as input to process for the computer system.	Information is the output of processed data.
Data does not depend on information.	Information depends on data.
Data does not have any meaning and useful until it is organized.	Information can be meaningful and useful when it is organized by data.
Data is always correct – it is a piece of truth, a thing that has happened.	Information is processed or derived from data that can be wrong.
Example of data: Each student's examination marks is one part of data.	Example of information: The average marks of a class or of the entire college is information that can be derived from the given data.

What is Database?

- A database is a collection of information that is organized so that it can be easily accessed, managed and updated.
- A database can be as simple as an alphabetical arrangement of names in an address book or as complex as a database that provides information in a combination of formats.
- Traditional databases are organized by *fields*, *records*, and *files*. A field is a basic unit of data storage and contains information related to an attribute of the entity described by the database; a record is one complete set of fields; and a file is a collection of records. Generally, a database is stored as a file or a set of files on magnetic disk or tape, optical disk, or some other secondary storage device.
- For example, a telephone book is similar to a file. It contains a list of records, each of which consists of three fields: name, address, and telephone number.



What is Database?

- A database is an organized collection of structured information, or data, typically stored electronically in a computer system. A database is usually controlled by a database management system (DBMS). Together, the data and the DBMS, along with the applications that are associated with them, are referred to as a database system, often shortened to just database.

- A database is a data structure that stores organized information. Most databases contain multiple tables, which may each include several different fields. For example, a company database may include tables for products, employees, and financial records. Each of these tables would have different fields that are relevant to the information stored in the table.

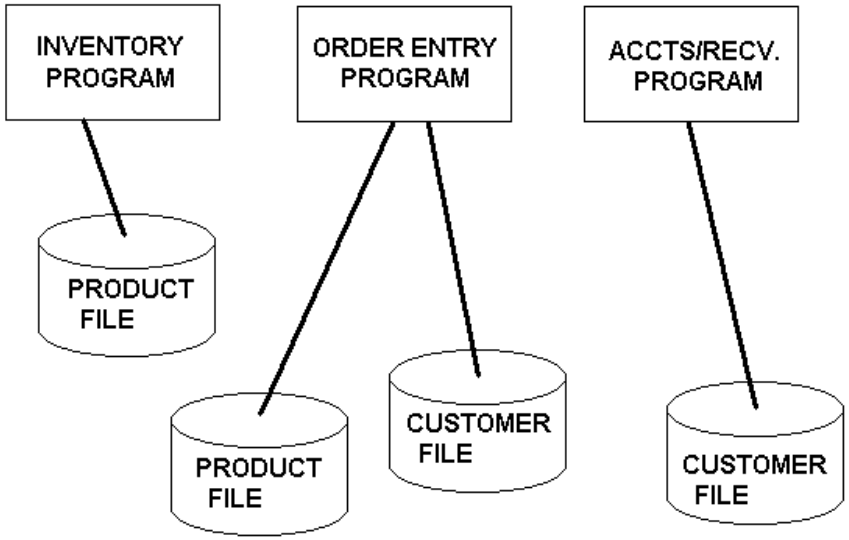
Traditional File System

Traditional file processing system or simple file processing system refers to the first computer-based approach of handling the commercial or business applications. That is why it is also called a replacement of the manual file system.

Before the use computers, the data in the offices or business was maintained in the files manually. Obviously, it was very laborious, time-consuming, inefficient task especially in large organizations. Computers were initially designed for engineering & scientific applications. Since they helped in efficient management of data in files, file processing environment simply transformed manual file work to computers. So, processing becomes fast and efficient.

The concept of conventional file processing system is shown in figure, which consists of three applications namely Inventory, Order Entry, and Accounts/Received programs. These programs process the data stored in different files such as Customer File and Product File.

As file processing systems were used, their problems were also realized and some of them are very severe.



Traditional File System

Most explicit and major disadvantages of file system when compared to database management systems are as follows:

- Data Redundancy
- Data Inconsistency
- Difficulty in Accessing Data
- Data Isolation
- Integrity Problems
- Security and access control
- Concurrency Problems

Traditional File System Vs DBMS

- A database management system coordinates both the physical and the logical access to the data, whereas a file-processing system coordinates only the physical access.

- A database management system reduces the amount of data duplication by ensuring that a physical piece of data is available to all programs authorized to have access to it, whereas data written by one program in a file-processing system may not be readable by another program.

- A database management system is designed to allow flexible access to data (i.e., queries), whereas a file-processing system is designed to allow predetermined access to data (i.e., compiled programs).

- A database management system is usually designed to allow one or more programs to access different data files at the same time. In a file-processing system, a file can be accessed by two programs concurrently only if both programs have read-only access to the file.

Traditional File System Vs DBMS

- Redundancy is control in DBMS, but not in file system.
- Unauthorized access is restricted in DBMS but not in file system.
- DBMS provide backup and recovery. When data is lost in file system then it not recover.
- DBMS provide multiple user interfaces. Data is isolated in file system.

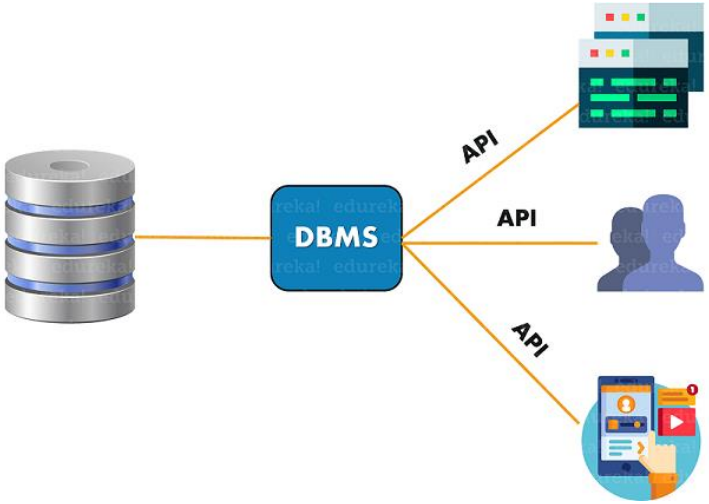
What is DBMS?

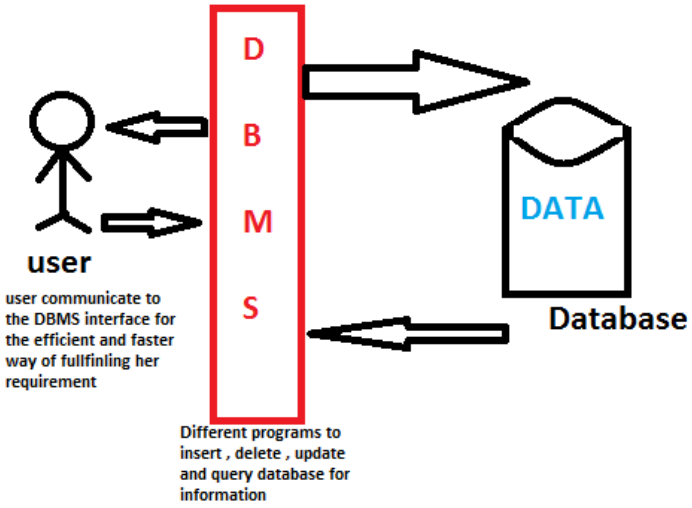
- A Database Management System (DBMS) is software designed to store, retrieve, define, and manage data in a database.

- Database Management System (DBMS) is a software for storing and retrieving users' data while considering appropriate security measures. It consists of a group of programs which manipulate the database.

- A database management system (DBMS) is system software for creating and managing databases. A DBMS makes it possible for end users to create, read, update and delete data in a database. The most prevalent type of data management platform, the DBMS essentially serves as an interface between databases and end users or application programs, ensuring that data is consistently organized and remains easily accessible.

What is DBMS?


DBMS
API
API
API


D
B
M
S
DATA
Database
user
user communicate to the DBMS interface for the efficient and faster way of fullfinling her requirement
Different programs to insert , delete , update and query database for information

Functions of DBMS?

DBMS performs several important functions that guarantee the integrity and consistency of the data in the database. The most important functions of Database Management System are

(i) **Data Storage Management:** It provides a mechanism for management of permanent storage of the data. The internal schema defines how the data should be stored by the storage management mechanism and the storage manager interfaces with the operating system to access the physical storage.

A modern DBMS system provides storage not only for the data, but also for related data entry forms or screen definitions, report definitions, data validation rules, procedural code, structures to handle video and picture formats, and so on.

Data storage management is also important for database performance tuning. Performance tuning relates to the activities that make the database perform more efficiently in terms of storage and access speed. So, the data storage management is another important function of Database Management System.

Functions of DBMS?

(ii) **Data Manipulation Management:** A DBMS furnishes users with the ability to retrieve, update and delete existing data in the database.

(iii) **Data Definition Services:** The DBMS accepts the data definitions such as external schema, the conceptual schema, the internal schema, and all the associated mappings in source form.

(iv) **Data Dictionary/System Catalog Management:** The DBMS provides a data dictionary or system catalog function in which descriptions of data items are stored and which is accessible to users. The data dictionary is full of **"Metadata",** Database about a database.
A data dictionary defines the structure of the database itself and is used in control and maintenance of large databases.
A data dictionary contains the following items-

- The definitions of all schema objects in the database (tables, views, indexes, clusters, synonyms, sequences, procedures, functions, packages, triggers, and so on).
- Space allotted and used by schema objects.
- Types of relationships between data elements.
- Access rights and frequency of access.
- All the users information.

Functions of DBMS?

(v) Database Communication Interfaces: The end-user's requests for database access are transmitted to DBMS in the form of communication messages.

(vi) Authorization / Security Management: The DBMS protects the database against unauthorized access, either intentional or accidental. It furnishes mechanism to ensure that only authorized users can access the database.

Database security refers to the use of the DBMS features and other related measures to prevent data from unauthorized access, loss, corruption, or mishandling.

DBMS has some features to provide security to database. Those features are

- Data encryption,
- Authentication,
- Authorization and
- Data views.

Functions of DBMS?

Encryption is a process to covert data into unreadable format. Unauthorized person cannot read and understand this encrypted data. Only authorized use will be able to read it.

Authentication is the process by which the system validates a user's logon information. A user's name and password are compared to an authorized list, and if the system detects a match, access is granted to the extent specified in the permission list for that user.

Authorization is any process by which someone is allowed to access the database by given user id and password, if password is not successfully entered, the user will be denied for accessing database.

In DBMS, sometimes it is required to provide partial information of database to some users and restrict them to access other part of database.

Data Views may be defined by DBA to allow user to access the database tables partially and hide other part of the tables or fields for them.

Functions of DBMS?

(vii) Backup and Recovery Management: The DBMS provides mechanisms for backing up data periodically and recovering from different types of failures. This prevents the loss of data

The DBMS provides mechanisms for backing up data periodically and recovering from different types of failures. This ensures data safety, data integrity and prevents the loss of data.

A *Data Backup* is a copy of data. This copy can include important parts of the database, such as the control file and data files. A backup is a safeguard against unexpected data loss and application errors. If you lose the original data due to any reason, then you can reconstruct it by using a backup.

The reasons for data loss can be divided into five main groups:

- Program errors
- Administrator (human) errors
- Computer failures (system crash)
- Disk failures
- Catastrophes (fire, earthquake) or theft

Functions of DBMS?

If database is damaged due to any reason, the DBMS restore a physical backup of a data file or control file to the correct state of the database and this process is called *Data Recovery*. It is very important to backup data periodically for proper recovery.

(viii) **Concurrency Control Service:** Since DBMSs support sharing of data among multiple users, they must provide a mechanism for managing concurrent access to the database. When more than one transactions are running simultaneously there are chances of a conflict to occur which can leave database to an inconsistent state. To handle these conflicts we need concurrency control in DBMS, which allows transactions to run simultaneously but handles them in such a way so that the integrity of data remains intact.

Functions of DBMS?

Problems of concurrency control

Several problems can occur when concurrent transactions are executed in an uncontrolled manner. Following are the three problems in concurrency control.

- Lost updates
- Dirty read
- Unrepeatable read

Concurrency Control Protocols

Different concurrency control protocols offer different benefits between the amount of concurrency they allow and the amount of overhead that they impose.

- Lock-Based Protocols
- Two Phase
- Timestamp-Based Protocols
- Validation-Based Protocols

Functions of DBMS?

(ix) **Transaction Management:** A transaction is a series of database operations, carried out by a single user or application program, which accesses or changes the contents of the database. Therefore, a DBMS must provide a mechanism to ensure either that all the updates corresponding to a given transaction are made or that none of them is made.

A transaction is a sequence of database operations, carried out by a single user or application program, which accesses or updates the contents of the database. DBMS provide a mechanism to ensure either that all the updates corresponding to a given transaction are made or that none of them is made.

In order to fully maintain data integrity and ensure good transactional behaviour, DBMS supports the ACID properties:

1) ***Atomicity:*** If any part of a transaction fails, the database state is left unchanged.
2) ***Consistency:*** Any transaction will leave the database in a consistent state.
3) ***Isolation:*** During a transaction, modified data cannot be accessed by other operations.
4) ***Durability:*** The DBMS can always recover the results of a committed transaction.

Functions of DBMS?

(x) Database Access and Application Programming Interfaces: All DBMS provide interface to enable applications to use DBMS services. They provide data access via Structured Query Language (SQL). The DBMS query language contains two components: (a) a Data Definition Language (DDL) and (b) a Data Manipulation Language (DML).

(xi) Data Integrity

- Data integrity refers to the overall completeness, accuracy and consistency of data and is an important feature of a database system.
- Data integrity means that the data contained in the database is accurate and reliable. Data integrity is a set of rules and standard procedures and enforced during the database design phase.
- Data integrity can be maintained through the use of various error checking methods and validation procedures.

DBMS supports basic types of database integrity constraints are:

1) *Entity integrity*, not allowing multiple rows to have the same identity within a table.
2) *Domain integrity*, restricting data to predefined data types, e.g.: dates.
3) *Referential integrity*, requiring the existence of a related row in another table, e.g. a Student for a given RegNo.

Advantages & Disadvantages of DBMS

Advantages

1) Reduction of Redundancies
2) Shared Data
3) Data Integrity
4) Security
5) Conflict Resolution or Concurrency Control
6) Transaction Management
7) Data Independence
8) Backup and Recovery

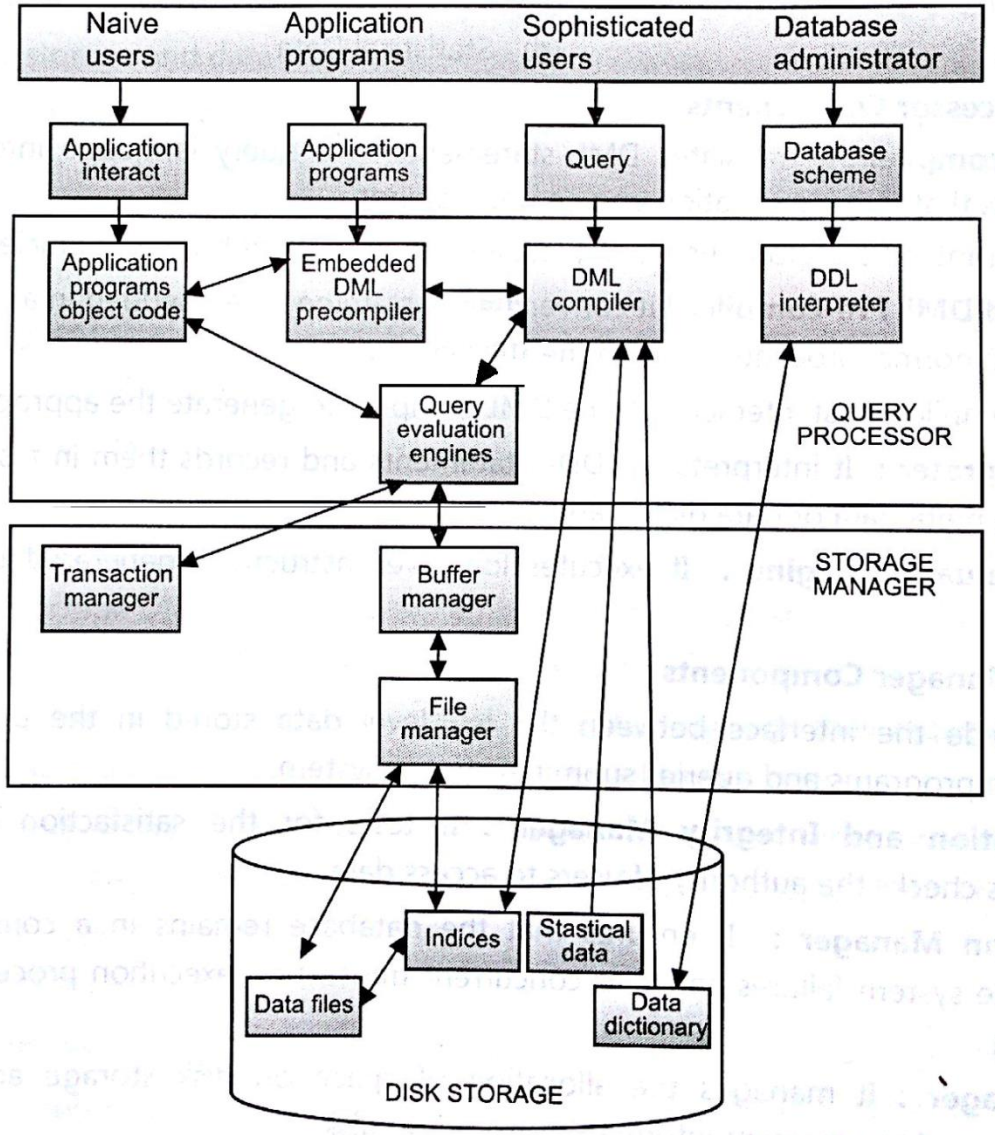
Disadvantages

1) Costly
2) Degradation of System Performance due to processing overhead
3) Backup and Recovery operations are complex

Structure of DBMS

The **database system** is divided into three components:

- Query Processor,
- Storage Manager, and
- Disk Storage



System structure

Structure of DBMS

1. Data Definition Language Interpreter

It execute the low-level statements and records them in a set of tables that having metadata.

2. Data Manipulation Language Compiler

It converts DML statements in source form of query language into necessary object form (i.e. low-level instructions) that query evaluation engine understands.

3.Embedded DML Precompiler

It converts DML statements embedded in application program to normal procedure calls in host language. To generate appropriate code, it must interacts with DML Compiler.

4. Query Evaluation Engine

It executes the low-level instruction generated by DML compiler.

5. Transaction Manager

It ensure that the database remains in consistent state despite the system failure, and concurrent transaction executions proceed without conflicting.

Structure of DBMS

6. Buffer Manager

It is responsible for data fetching from disk storage into main memory and deciding what data to cache in memory.

7. File Manager

It manages allocation of space on disk storage and data structure used to represent information that stored on disk.

Some of the Data Structure are needed as the part of physical system for implementation:

i. **Indices:** These provide fast access to the data items that hold particular values.
ii. **Statistical Data:** This store statistical information about data in database. To execute a query, this information is used by query processor.
iii. **Data Files:** These are actual files that store the data in database, i.e. these are database files.
iv. **Data Dictionary:** This stores metadata about each and every entity of the database along with the security and entity constraints.

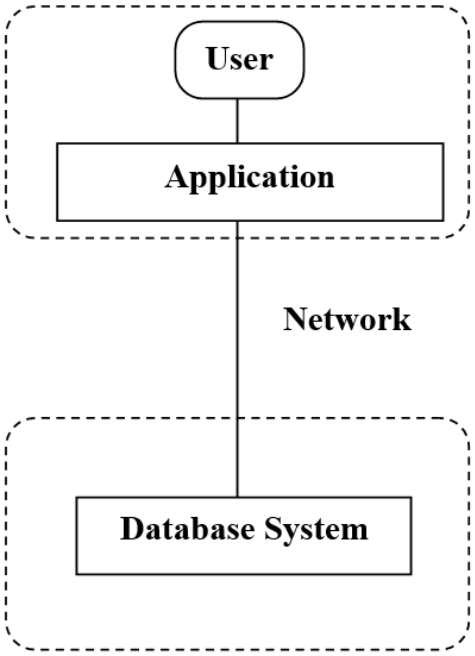
Structure of DBMS

Information Stored in Data Dictionary (DBMS)	Statistical Data Stored in DBMS
1. Database Schema Information Table and column details (names, data types, sizes). Indexes, views, and foreign key relationships. **2. Constraints and Rules** Primary keys, foreign keys, unique constraints. NOT NULL, CHECK constraints, referential integrity. **3. Storage and Performance Statistics** Number of rows in tables, table size. Index usage, fragmentation details. **4. User and Security Information** Usernames, roles, and permissions. Access privileges (READ, WRITE, EXECUTE). Audit logs (who accessed/modified data). **5. Procedural and Programmatic Information** Stored procedures, functions, triggers. Views, materialized views, default values. **6. Query Optimization and Execution Plans** Execution statistics of queries. Query optimization hints, indexing suggestions. **7. Backup and Recovery Information** Last backup time, transaction logs. Recovery and rollback details.	**1. Table and Index Statistics** Number of rows, pages, and distinct values. Index selectivity, clustering factor. **2. Column-Level Statistics** Min, max, mean, median, standard deviation. Null values count, histogram distribution. **3. Query Execution Statistics** Query execution time, disk I/O operations. Cache hit/miss ratio, join complexity. **4. Transaction and Concurrency Statistics** Number of transactions, commit/rollback rates. Lock contention, deadlock occurrences. **5. System Performance Statistics** CPU, memory, and disk usage. Network traffic in distributed databases. **6. User and Access Statistics** Number of active users, login attempts. Access logs and security audit trails. **7. Backup and Recovery Statistics** Backup frequency, recovery time, transaction logs.

Database Applications Architectures

Database Applications are partitioned into 2-tier and 3-tier parts as shown in figure.

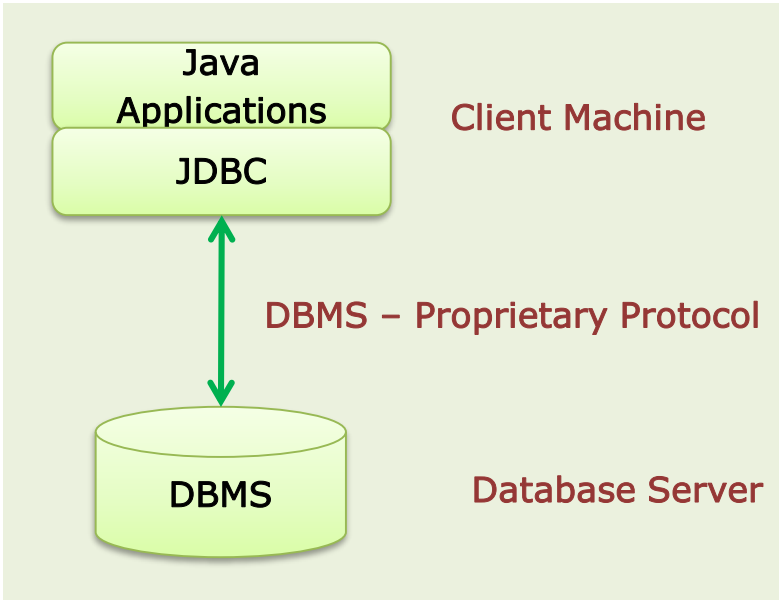
- **Two-tier Architecture:** In the two-tier architecture, the application is partitioned into a component that resides at the client machine and calls the database system functionality at the server machine using query language statements. Application program interface standards like ODBC and JDBC are used for interaction between the client and the server. Two-tiered application examples include desktop applications, games, and music players.



client

Server

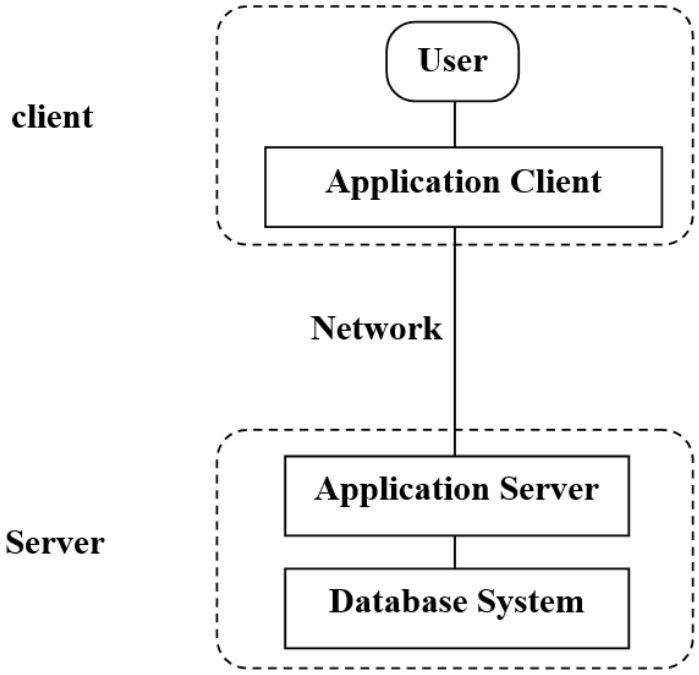
(a) Two-tier Architecture



2-tier DBMS Application Architecture

Database Applications Architectures

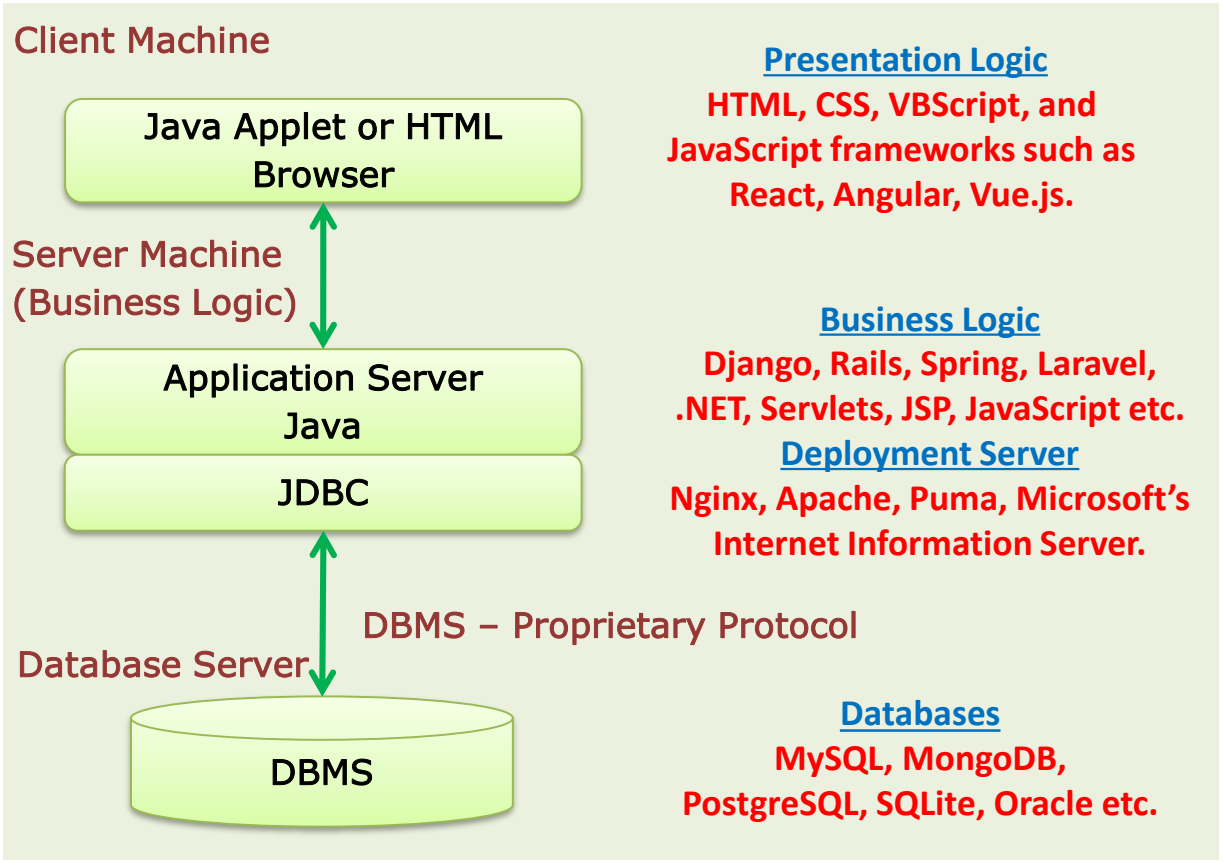
- **Three-tier Architecture:** In three-tier architecture, the client machine contains a form interface and does not contain any direct database calls. It communicates with an application server which in turn communicates with the database system. Thus, the required functionalities are not distributed among the multiple clients. This architecture is appropriate for large applications that run on the World Wide Web. A good example is modern web applications.



(a) Three-tier Architecture

Database Applications Architectures

The **Business Logic** of the application, which says what actions to carry out under what conditions, is embedded in the application server, instead on being distributed across multiple clients. 3-tier architecture is more appropriate for large organizations, and for applications that run on the World Wide Web.

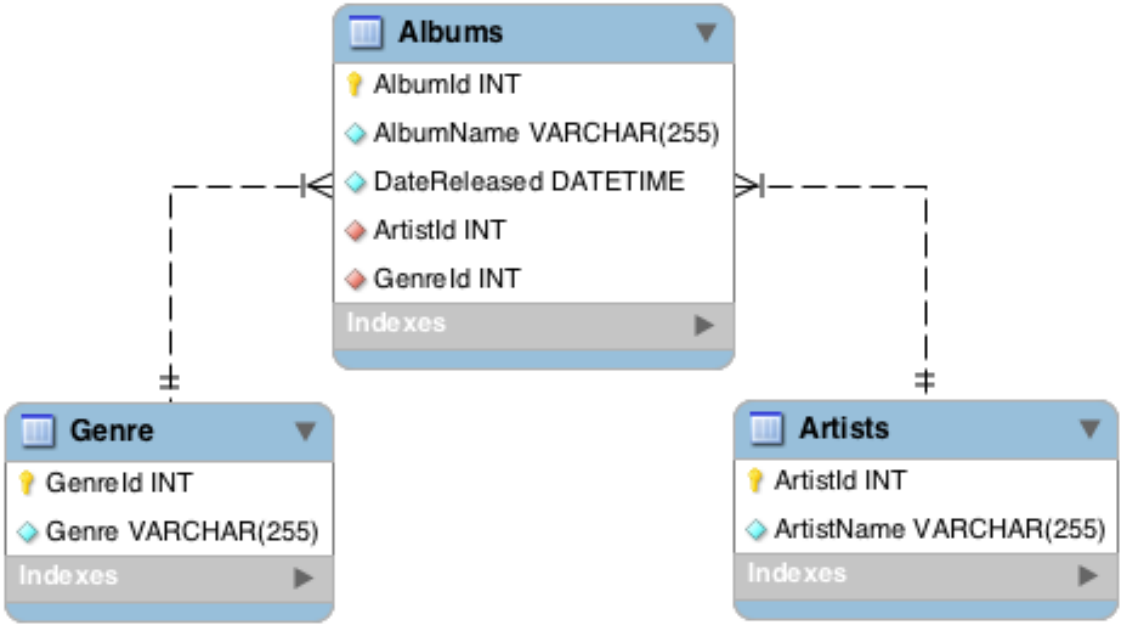


3-tier DBMS Application Architecture

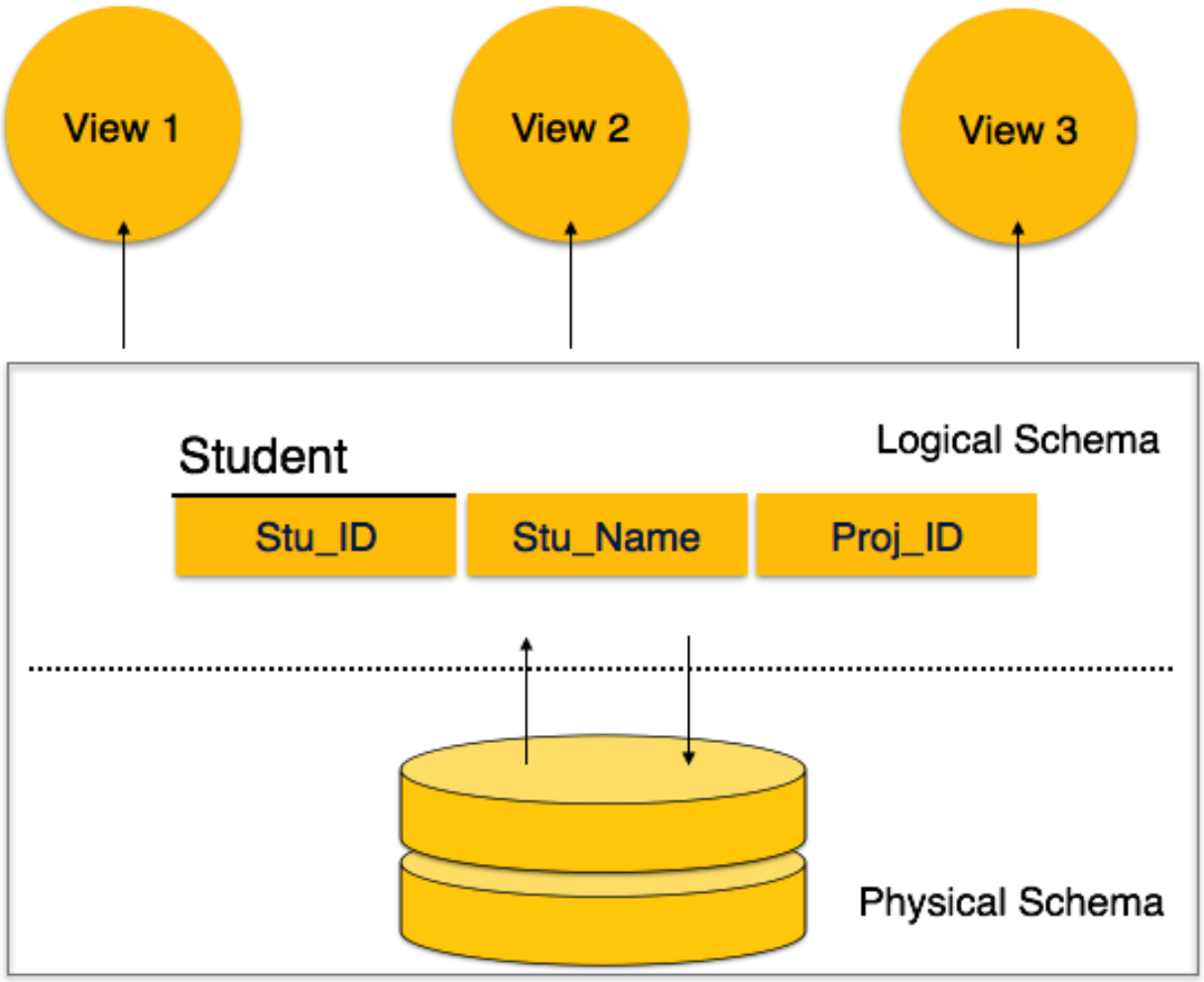
Schema, Subschema and Instance

Schema

- Schema is a the logical description of the database.
- The overall design of the database is called Database Schema.
- A schema contains schema objects, which could be tables, columns, data types, views, stored procedures, relationships, primary keys, foreign keys, etc.
- A database schema can be represented in a visual diagram, which shows the database objects and their relationship with each other.



Schema, Subschema and Instance



Schema, Subschema and Instance

Schema and Database are the same things or different?

Part of the reason for the confusion is that database systems tend to approach schemas in their own way.

- **MySQL Documentation**: A schema is synonymous with a database. Therefore, a schema and a database are the **same thing**.

- **Oracle Database Documentation:** Certain objects can be stored inside a database but not inside a schema. Therefore, a schema and a database are **two different things**.

- **SQL Server Technical Article:** A schema is a separate entity inside the database. So, they are **two different things**.

Schema, Subschema and Instance

1. **Internal Schema or Physical Schema**

   It describes that how data are actually stored in the blocks of storage devices such as hard disk.

2. **Logical Schema or Conceptual Schema**

   It describes the structure of the database to the database designer. Programmers and database administrators work at this level, at this level data can be described as certain types of data records gets stored in data structures

3. **External Schema or Subschema**

   It is a subset of main schema having the same properties as schema. It describes the different parts, sets, records and data names of the database to the end user. It allows users to view only the parts of the main database.

Schema, Subschema and Instance

Subschema

- It is a subset of the schema having the same properties that a schema has.
- It allows the user to view only that part of the database that is of interest to him.

How does a subschema work?

- A subschema lets users see only the parts of the database that are relevant to them.
- It can restrict access to the database.
- It can identify which records, areas, and elements are accessible.

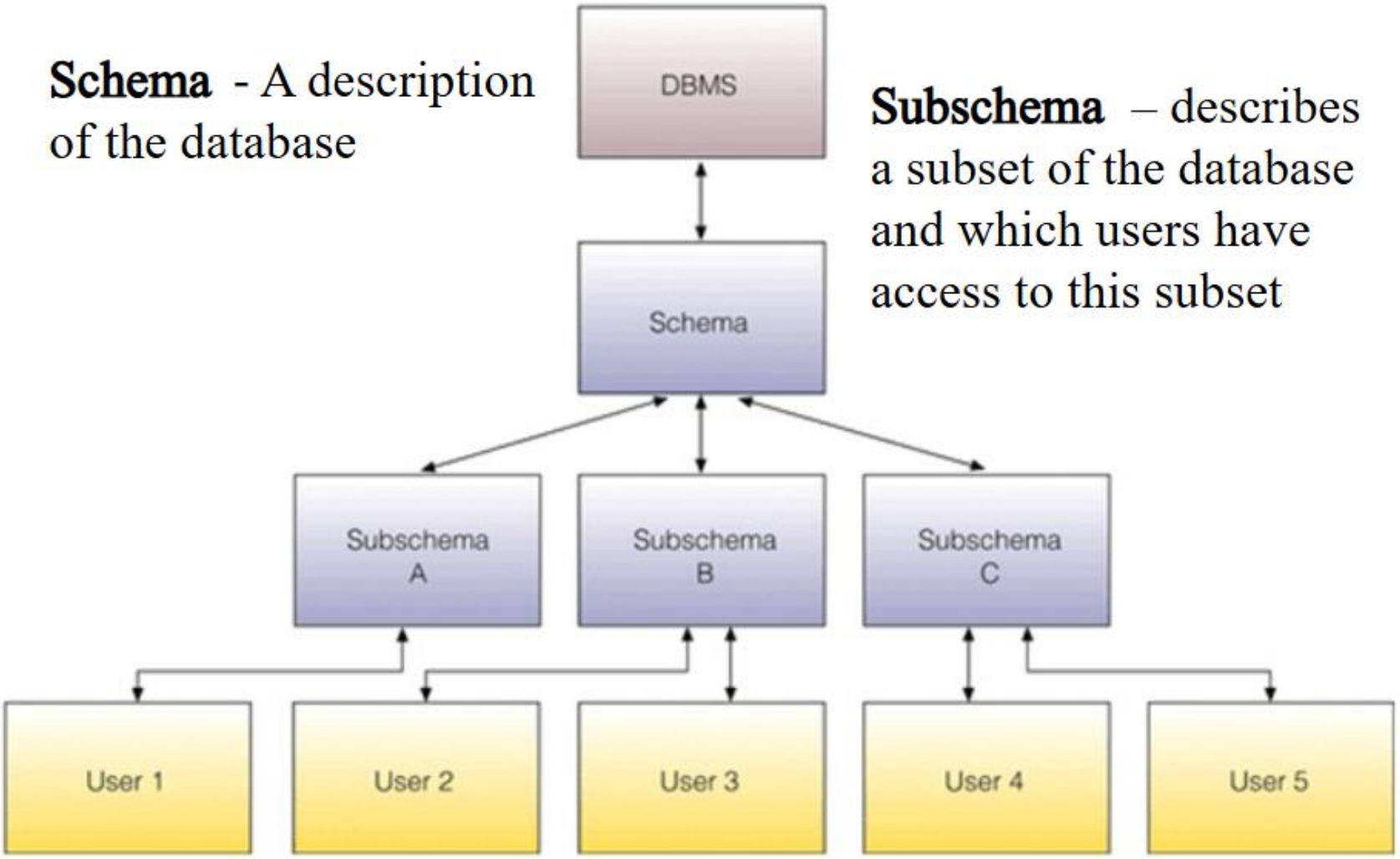
Why use subschemas?

- Subschemas help promote security by limiting access to data
- They help manage permissions
- They help control access to data at different levels
- They allow different application programs to have different views of the data

Schema, Subschema and Instance

Schema - A description of the database

Subschema – describes a subset of the database and which users have access to this subset



Schema, Subschema and Instance

Database Instance

The data stored in database at a particular moment of time is called instance of database. Database schema defines the variable declarations in tables that belong to a particular database; the value of these variables at a moment of time is called the instance of that database.

A database schema is variable declarations in a program. Variable has particular value at a given instant. Then, the value of variable at particular instant is called database instance.

Database schema (names of columns + the types associated with them)

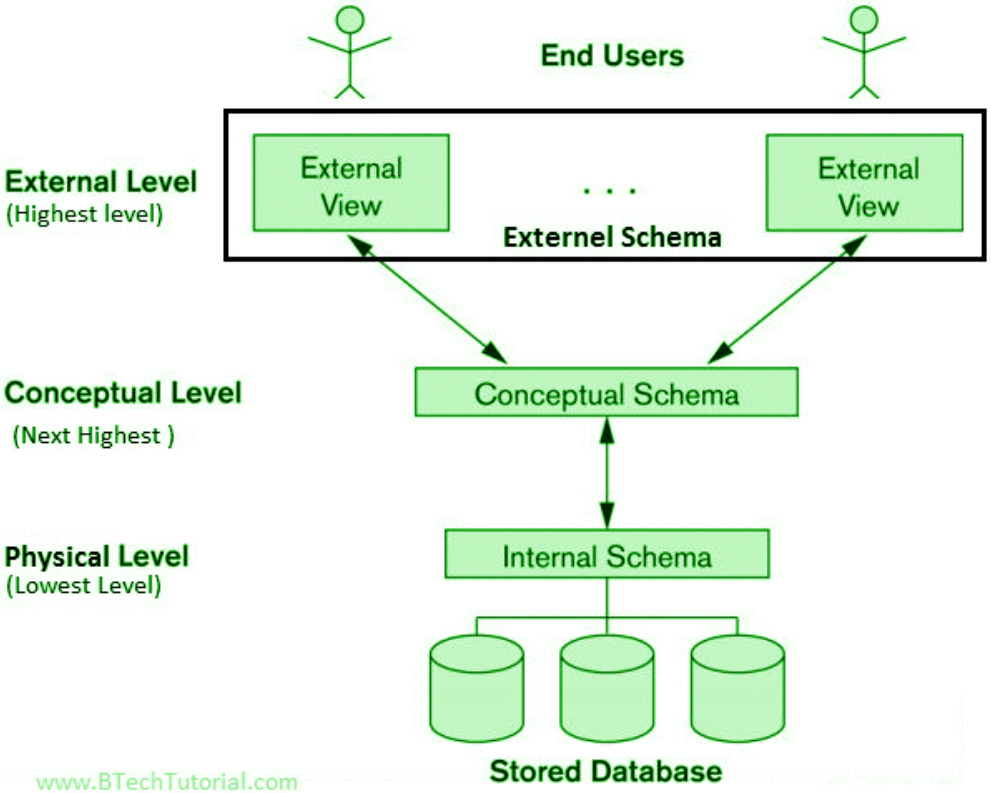
Name	DOB	Address	Job	Scale
String	*Date*	*String*	*String*	*Int*

A database instance

Name	DOB	Address	Job	Scale
A. Johnson	2/04/1960	London	Programmer	12
B. Holiday	3/10/1947	Leeds	Analyst	14
C. Clark	12/08/1971	York	Programmer	10

3-Level schema Architecture

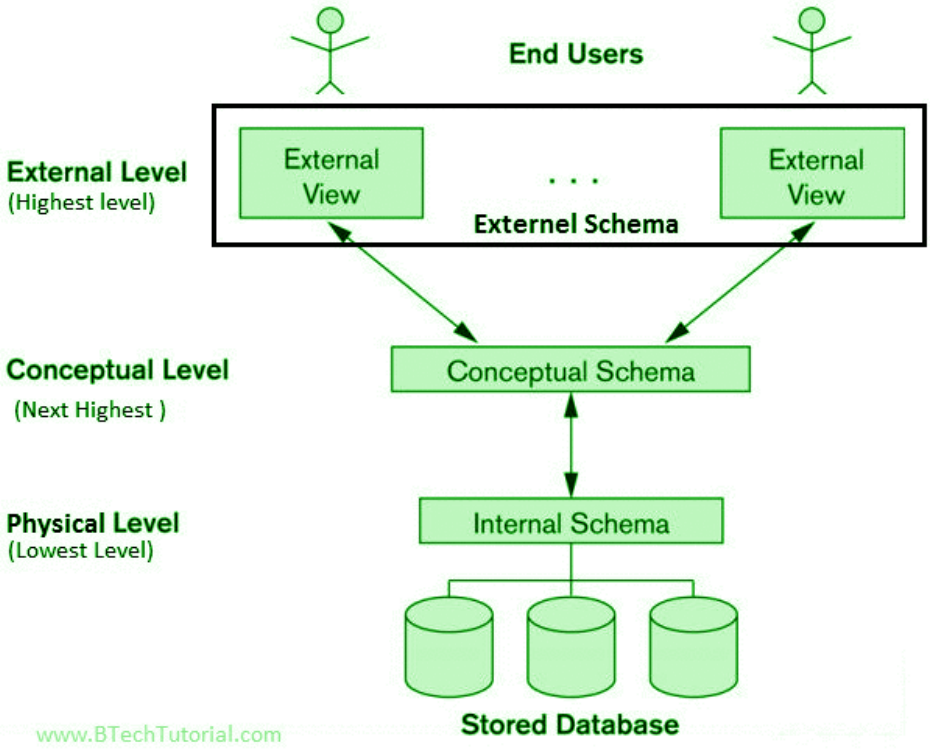
- The three schema architecture is also called ANSI/SPARC architecture or three-level architecture.
- This framework is used to describe the structure of a specific database system.
- The three schema architecture is also used to separate the user applications and physical database. The three schema architecture contains three-levels. It breaks the database down into three different categories.



3-Level schema Architecture

Figure shows the DBMS architecture.

Mapping is used to transform the request and response between various database levels of architecture. Mapping is not good for small DBMS because it takes more time. In External / Conceptual mapping, it is necessary to transform the request from external level to conceptual schema. In Conceptual / Internal mapping, DBMS transform the request from the conceptual to internal level.



3-Level schema Architecture

Internal level:

- This is the lowest level of data abstraction.
- It describes How the data are actually stored on storage devices.
- It is also known as physical level.
- It provides internal view of physical storage of data.
- It deals with complex low level data structures, file structures and access methods in detail.
- It also deals with Data Compression and Encryption techniques, if used.

Conceptual level:

- This is the next higher level than internal level of data abstraction.
- It describes What data are stored in the database and What relationships exist among those data.
- It is also known as Logical level.
- It hides low level complexities of physical storage.
- Database administrator and designers work at this level to determine What data to keep in database.
- Application developers also work on this level.

3-Level schema Architecture

External Level:

- This is the highest level of data abstraction.
- It describes only part of the entire database that a end user concern.
- It is also known as an view level.
- End users need to access only part of the database rather than entire database.
- Different user need different views of database. And so, there can be many view level abstractions of the same database.

Advantages of 3-level schema Architecture:

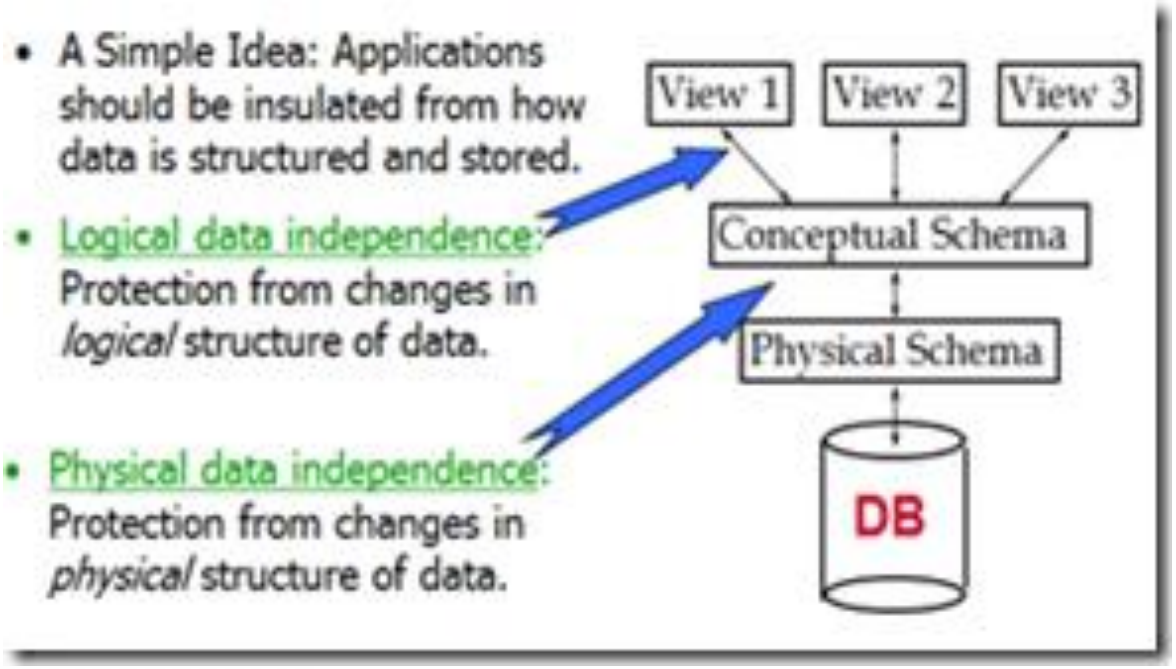
- The main objective of it is to provide data abstraction.
- Same data can be accessed by different users with different customized views.
- The user is not concerned about the physical data storage details.
- Physical storage structure can be changed without requiring changes in internal structure of the database as well as users view.
- Conceptual structure of the database can be changed without affecting end users.

Data Independence

A Database stores data about data i.e. Metadata. Metadata itself follows a layered architecture, so that when we change data at one layer, it does not affect the data at another level. This data is independent but mapped to each other.

Data independence can be explained using the three-schema architecture.

Data independence refers characteristic of being able to modify the schema at one level of the database system without altering the schema at the next higher level.



Data Independence

Logical Data Independence

- Logical data is data about database, that is, it stores information about how data is managed inside.
- Logical data independence is used to separate the external level from the conceptual view.
- If we do any changes in the conceptual view of the data, then the user view of the data would not be affected.
- Logical data independence occurs at the user interface level.
- For example, a table (relation) stored in the database and all its constraints, applied on that relation.
- Logical data independence is a kind of mechanism, which liberalizes itself from actual data stored on the disk. If we do some changes on table format, it should not change the data residing on the disk.

Data Independence

Physical Data Independence

- All the schemas are logical, and the actual data is stored in bit format on the disk. Physical data independence is the power to change the physical data without impacting the schema or logical data.

- If we do any changes in the storage size of the database system server, then the Conceptual structure of the database will not be affected.

- Physical data independence is used to separate conceptual levels from the internal levels.

- Physical data independence occurs at the logical interface level.

- For example, in case we want to change or upgrade the storage system itself − suppose we want to replace hard-disks with SSD − it should not have any impact on the logical data or schemas.

Data Abstraction

Database systems are made-up of complex data structures. To ease the user interaction with database, the developers hide internal irrelevant details from users. ***This process of hiding irrelevant details from user is called data abstraction..***

We have three levels of abstraction:

- **Physical level**: This is the lowest level of data abstraction. It describes how data is actually stored in database. You can get the complex data structure details at this level.

- **Logical level**: This is the middle level of 3-level data abstraction architecture. It describes what data is stored in database.

- **View level**: Highest level of data abstraction. This level describes the user interaction with database system.

Data Abstraction

Example: Let's say we are storing customer information in a customer table.

At **physical level** these records can be described as blocks of storage (bytes, gigabytes, terabytes etc.) in memory. These details are often hidden from the programmers.

At the **logical level** these records can be described as fields and attributes along with their data types, their relationship among each other can be logically implemented. The programmers generally work at this level because they are aware of such things about database systems.

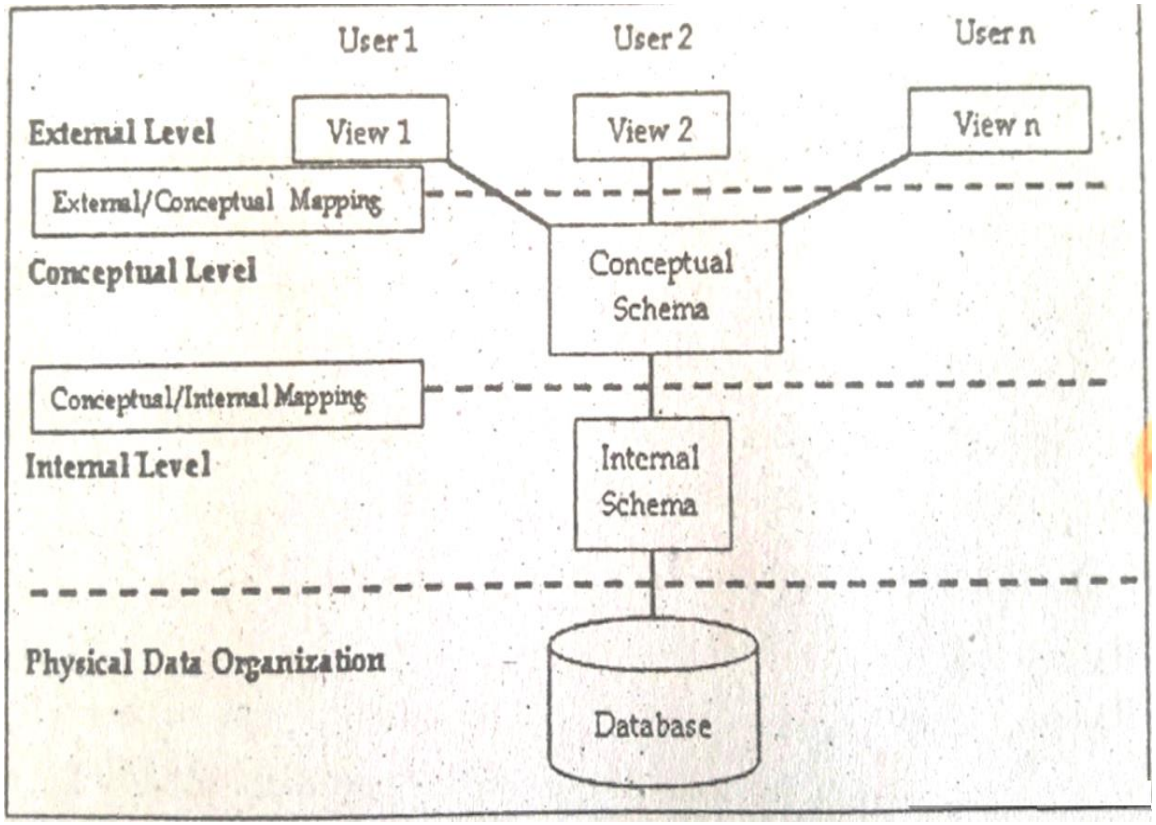
At **view level**, user just interact with system with the help of GUI and enter the details at the screen, they are not aware of how the data is stored and what data is stored; such details are hidden from them.

Mapping

Mapping is a process of converting one level to another level. In this process, the data in one level is related to the data at another level.

There are two level of Mapping:

- From the Conceptual Level to Internal Level
- From the External Level to Conceptual Level



Mapping

Conceptual to Internal Mapping

- It defines the correspondence between the conceptual view and stored data.
- It specifies that how conceptual records and fields are represented at internal level.
- If the structure of stored data is changed, then conceptual-internal mapping must be changed.
- It is the responsibility of DBA to manage such changes.

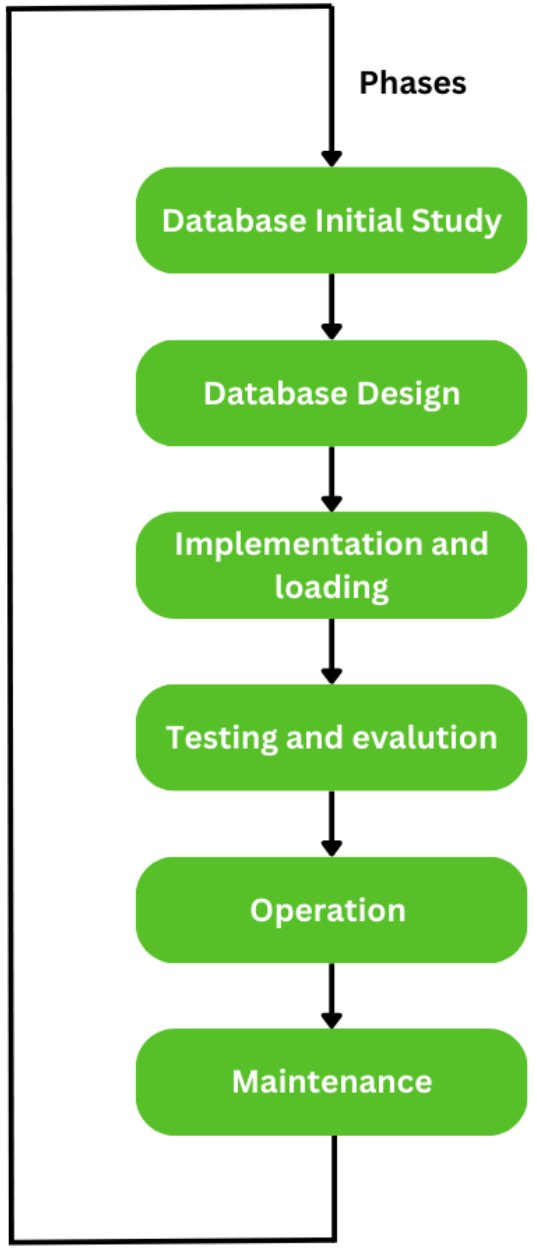
External to Conceptual Mapping

- It defines the correspondence between a particular external views and conceptual view.
- The difference between these two levels is similar to the difference between Conceptual Level and Internal Level.

Database Development Life Cycle

The Database Life Cycle (DBLC) outlines the stages involved in designing, implementing, and maintaining a database system. It encompasses the entire lifespan of a database, from initial planning to ongoing maintenance. The modern DBLC consists of six stages, which ensure a database meets business needs efficiently and remains effective over time.

1. **Database Initial Study**
2. **Database Design**
3. **Implementation and Loading**
4. **Testing and Evaluation**
5. **Operation**
6. **Maintenance and Evolution**



Database Development Life Cycle

The Database Life Cycle (DBLC) outlines the stages involved in designing, implementing, and maintaining a database system. It encompasses the entire lifespan of a database, from initial planning to ongoing maintenance. The modern DBLC consists of six stages, which ensure a database meets business needs efficiently and remains effective over time.

1. Database Initial Study

- **Analyze Company Situation:** Study the organization's structure, mission, and operations to understand system requirements.
- **Define Problems and Constraints:** Identify current system issues (like inefficiency or data duplication) and constraints such as budget or hardware limits.
- **Define Objectives:** Set clear goals for the database, including required queries, reports, and system integration needs.
- **Define Scope and Boundaries:** Decide the project scope and identify external limits such as existing hardware or software.

Database Development Life Cycle

2. Database Design

The database design stage converts requirements into a structured database blueprint. It consists of logical design and physical design.

Logical Design:

Creates a conceptual structure of the database using ER diagrams, defining tables, attributes, primary keys, and relationships. The design is normalized to reduce redundancy and improve data efficiency.

- **Conceptual Model:** Visual representation of entities and relationships
- **ER Diagram:** Shows data organization and relationships
- **Normalization:** Removes redundancy and improves data access

Physical Design:

Focuses on performance optimization by selecting storage structures, indexes, and access methods suitable for the hardware and software environment.

- **Performance Optimization:** Uses indexing and partitioning
- **Hardware Considerations:** Matches database design with available resources

Database Development Life Cycle

3. Implementation and Loading

In the Implementation and Loading stage, the database design is translated into a functional system. This involves creating database tables using SQL and loading initial data.

- **Create Tables:** Use SQL commands (e.g., CREATE TABLE) to build the database structure based on the logical and physical designs.
- **Load Data:** Populate the database with initial data, often imported from existing systems or files.
- **Configure Access:** Set up user permissions and security measures to protect the database.

4. Testing and Evaluation

- **Functional Testing:** Verify that the database supports required operations, such as queries, reports, and transactions.
- **Performance Testing:** Assess the database's speed and scalability under various loads.
- **Error Handling:** Test how the system handles invalid data or unexpected scenarios.

Database Development Life Cycle

5. Operation

During the Operation stage, the database is fully deployed and used in the organization's daily activities.

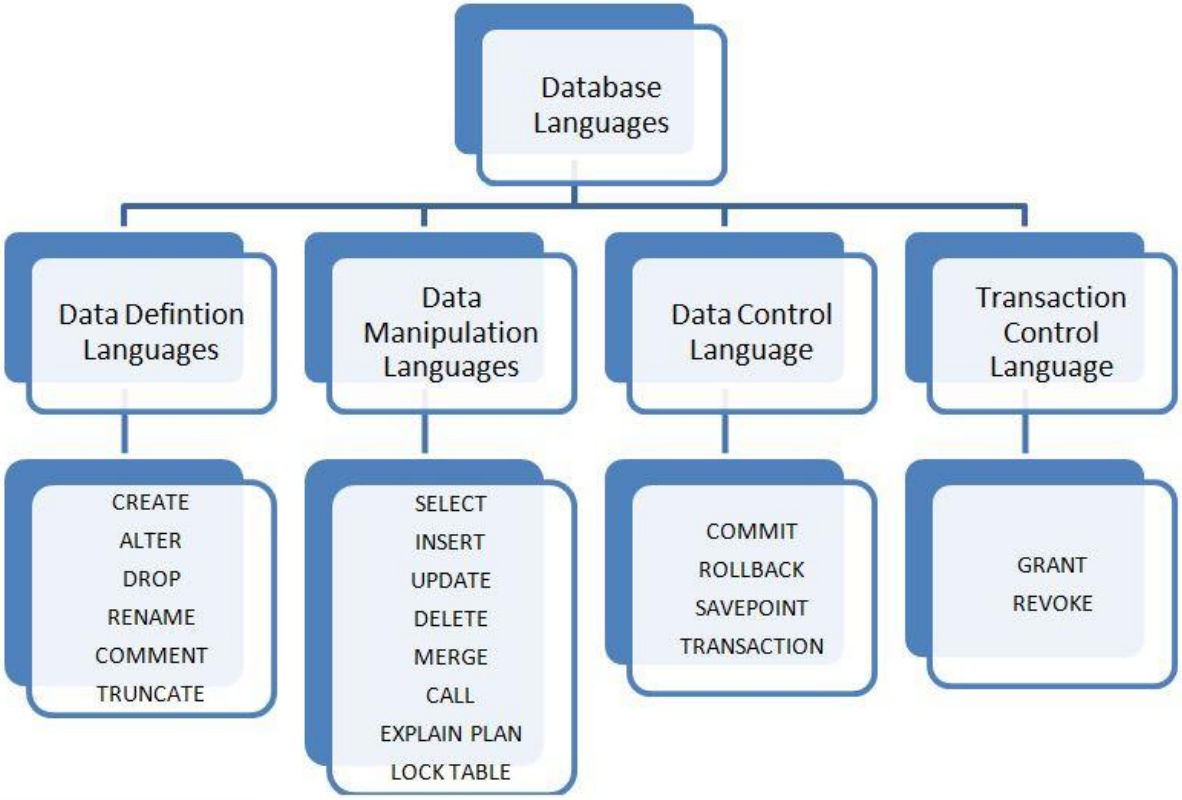
- **User Training:** Train end users and administrators to interact with the database effectively.
- **Data Management:** Support ongoing data entry, updates, and retrieval processes.
- **System Integration:** Ensure the database integrates with other organizational systems as planned.

6. Maintenance and Evolution

- **Monitoring:** Track performance and identify issues like slow queries or data inconsistencies.
- **Updates:** Modify the database schema or data to accommodate new requirements, such as adding new fields or tables.
- **Backups and Recovery:** Implement regular backups and recovery plans to protect data integrity.

Database Languages

A DBMS must provide appropriate languages and interfaces for each category of users to express database queries and updates. Database languages are used to read, update and store data in a database. There are large numbers of database languages like Oracle, MySQL, MS Access, dBase, FoxPro etc. SQL (Structured Query Language) is commonly used in Oracle and MS Access can be categorized as data definition language (DDL), data control language (DCL) and data manipulation language (DML).



Database Languages

Data Definition Language (DDL)

DDL is used for specifying the database schema. It is used for creating tables, schema, indexes, constraints etc. in database. Lets see the operations that we can perform on database using DDL:

- To create the database instance – **CREATE**
- To alter the structure of database – **ALTER**
- To drop database instances – **DROP**
- To delete tables in a database instance – **TRUNCATE**
- To rename database instances – **RENAME**
- To drop objects from database such as tables – **DROP**
- To Comment – **Comment**

All of these commands either defines or update the database schema that's why they come under Data Definition language.

Database Languages

Data Manipulation Language (DML)

DML is used for accessing and manipulating data in a database. The following operations on database comes under DML:

- To read records from table(s) – **SELECT**
- To insert record(s) into the table(s) – **INSERT**
- Update the data in table(s) – **UPDATE**
- Delete all the records from the table – **DELETE**

Data Control language (DCL)

DCL is used for granting and revoking user access on a database –

- To grant access to user – **GRANT**
- To revoke access from user – **REVOKE**

Database Languages

Transaction Control Language (TCL)

The changes in the database that we made using DML commands are either performed or roll backed using TCL.

- To persist the changes made by DML commands in database – **COMMIT**
- To rollback the changes made to the database – **ROLLBACK**

In practical data definition language, data manipulation language and data control languages are not separate language, rather they are the parts of a single database language such as SQL.

Different types of Database Users

Any person who uses the database and takes the benefits from the Database is considered as Database User. They can be programmer, scientist, engineers, businessman or can be any employee.

Database Administrator

- It is a person or team, who is responsible for overall management of the Database Management System design.
- It's a leader of the database; it is also like the superuser of the system.
- It is responsible for the administration of all 3 levels of database.
- He has all the privileges on the database; he can also assign or remove the privileges from the other database users.
- Each database requires at least one database administrator (DBA). A Database system can be large and can have many users. Therefore, database administration is sometimes not a one-person job, but a job for a group of DBAs who share responsibility.

Different types of Database Users

- A database administrator's responsibilities can include the following tasks:
  - ✓ Installing and upgrading the Oracle Database server and application tools
  - ✓ Allocating system storage and planning future storage requirements for the database system
  - ✓ Creating primary database storage structures (tablespaces) after application developers have designed an application
  - ✓ Creating primary objects (tables, views, indexes) once application developers have designed an application
  - ✓ Modifying the database structure, as necessary, from information given by application developers
  - ✓ Enrolling users and maintaining system security
  - ✓ Ensuring compliance with Oracle license agreements
  - ✓ Controlling and monitoring user access to the database
  - ✓ Monitoring and optimizing the performance of the database
  - ✓ Planning for backup and recovery of database information
  - ✓ Maintaining archived data on tape
  - ✓ Backing up and restoring the database
  - ✓ Contacting Oracle for technical support

Different types of Database Users

Application Programmers

- They are the developers who interact with the database by means of DML queries.
- These DML queries are written in the application programs like C, C++, JAVA, Pascal etc.
- These queries are converted into object code to communicate with the database.
- For example, writing a C program to generate the report of employees who are working in particular department will involve a query to fetch the data from database. It will include a embedded SQL query in the C Program.

Sophisticated Users

- They interact with the system without writing programs.
- They form requests by writing queries in a database query language. These are submitted to a **query processor** that breaks a DML statement down into instructions for the database manager module.
- They directly interact with the database by means of query language like SQL.
- These users will be scientists, engineers, analysts who thoroughly study SQL and DBMS to apply the concepts in their requirement. In short, we can say this category includes designers and developers of DBMS and SQL.

Different types of Database Users

Specialized Users

- These are also sophisticated users, but they write special database application programs.
- They are the developers who develop the complex programs to the requirement. These may be CADD systems, knowledge-based and expert systems, complex data systems (audio/video), etc.

Native Users

- These are the users who use the existing application to interact with the database.
- For example, online library system, ticket booking systems, ATMs etc which has existing application and users use them to interact with the database to fulfill their requests.

Different types of Database Users

System Analyst

- He/she analyses the requirements of the end users, especially naïve users and parametric end users. They are responsible for the design, structure and properties of the database.
- The main concerns of the system analyst is on feasibility, economic aspects and technical aspects.
- Analysts are one among the sophisticated users. They use the tools to perform their task such as:
  1. **Online analytical processing (OLAP)** - It helps the analysts to view them the summaries of the data in different ways.
  2. **Data Mining Tools** – It helps the analysts find a certain kind of pattern in the given data.

File Organizations

The File Organization is the physical organization of the records of a file for the convenience of storage and retrieval of data records. System designer basically choose to organize, access and process the records of the various files in different ways, depending upon the type of the application and needs of the uses are:

1. Ease of Retrieval
2. Convenience of Updates
3. Economy of Storage
4. Reliability
5. Security
6. Integrity

There are some commonly used file organizations are-

A. Serial File
B. Sequential File
C. Index Sequential File
D. Direct File
E. Multilist File
F. Tree based File

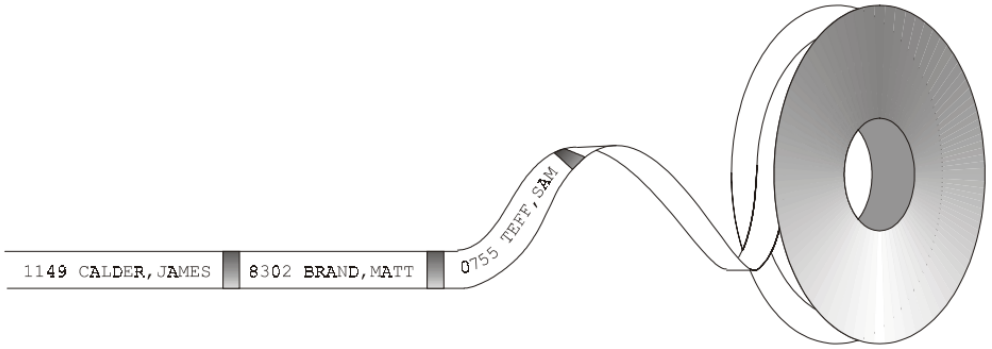
File Organizations

Serial File

- Records in a file are stored and accessed one after another.
- The records are not stored in any way on the storage medium this type of organization is mainly used on **magnetic tapes**.
- The records on a serial file are not in any particular sequence, and so this type of organisation would not be used for a master file as there would be no way to find a particular record except by reading through the whole file, starting at the beginning, until the right record was located. Serial files are used as temporary files to store transaction data.

Advantages

- It is simple
- It is cheap



Disadvantages

- It is cumbersome to access because you have to access all proceeding records before retrieving the one being searched.
- Wastage of space on medium in form of inter-record gap.
- It cannot support modern high speed requirements for quick record access.

File Organizations

Sequential File

- Storing and sorting in contiguous block within files on tape or disk is called as **sequential access file organization**.
- In sequential access file organization, all records are stored in a sequential order. The records are arranged in the ascending or descending order of a key field.
- Sequential file search starts from the beginning of the file and the records can be added at the end of the file.
- In sequential file, it is not possible to add a record in the middle of the file without rewriting the file.
- Searching of a record requires, on average, access to half the records in the file.

Advantages

- It is simple to program and easy to design.
- Sequential file is best use if storage space.

Disadvantages

- Sequential file is time consuming process.
- It has high data redundancy.
- Random searching is not possible.

A-217	Brighton	750	
A-101	Downtown	500	
A-110	Downtown	600	
A-215	Mianus	700	
A-102	Perryridge	400	
A-201	Perryridge	900	
A-218	Perryridge	700	
A-222	Redwood	700	
A-305	Round Hill	350	

File Organizations

Direct File or Random Access File

- Direct access file is also known as random access or relative file organization.
- In direct access file, all records are stored in direct access storage device (DASD), such as hard disk. The records are randomly placed throughout the file.
- The records does not need to be in sequence because they are updated directly and rewritten back in the same location.
- This file organization is useful for immediate access to large amount of information. It is used in accessing large databases.
- It is also called as hashing.

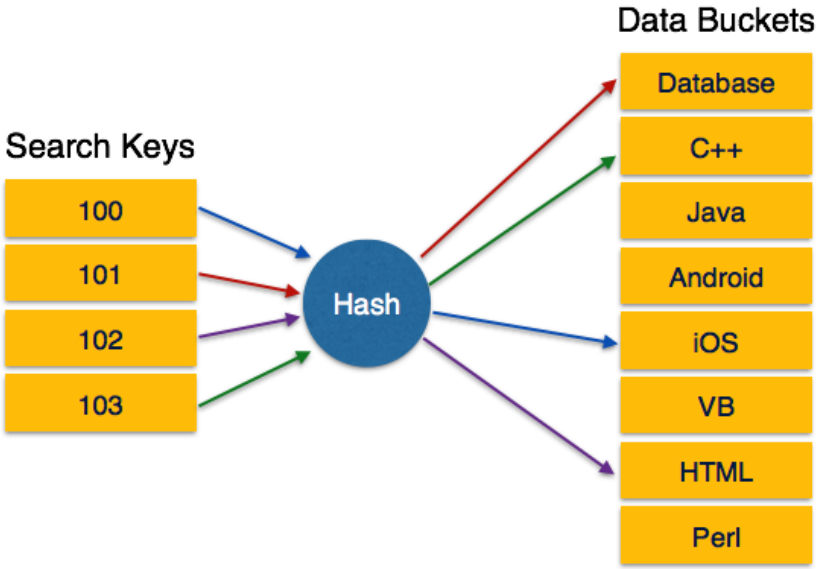
Advantages

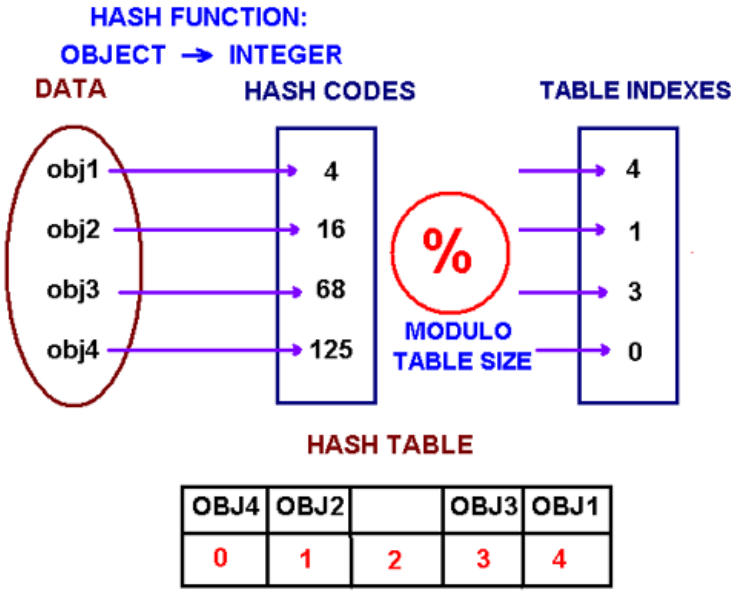
- In direct access file, sorting of the records are not required.
- It accesses the desired records immediately.
- It updates several files quickly.
- It has better control over record allocation.

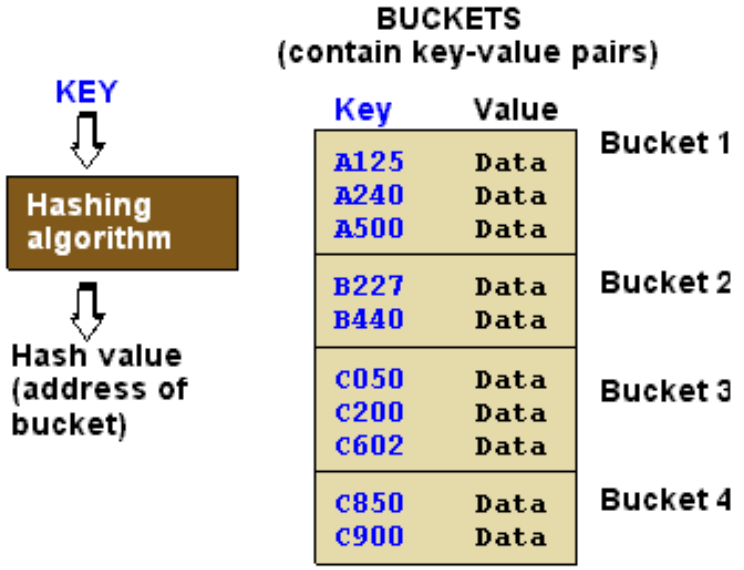
Disadvantages

- Direct access file does not provide back up facility.
- It is expensive.
- It has less storage space as compared to sequential file.

File Organizations







File Organizations

Indexed Sequential File

An indexed sequential file consists of records that can be accessed sequentially. Direct access is also possible. It consists of two parts −

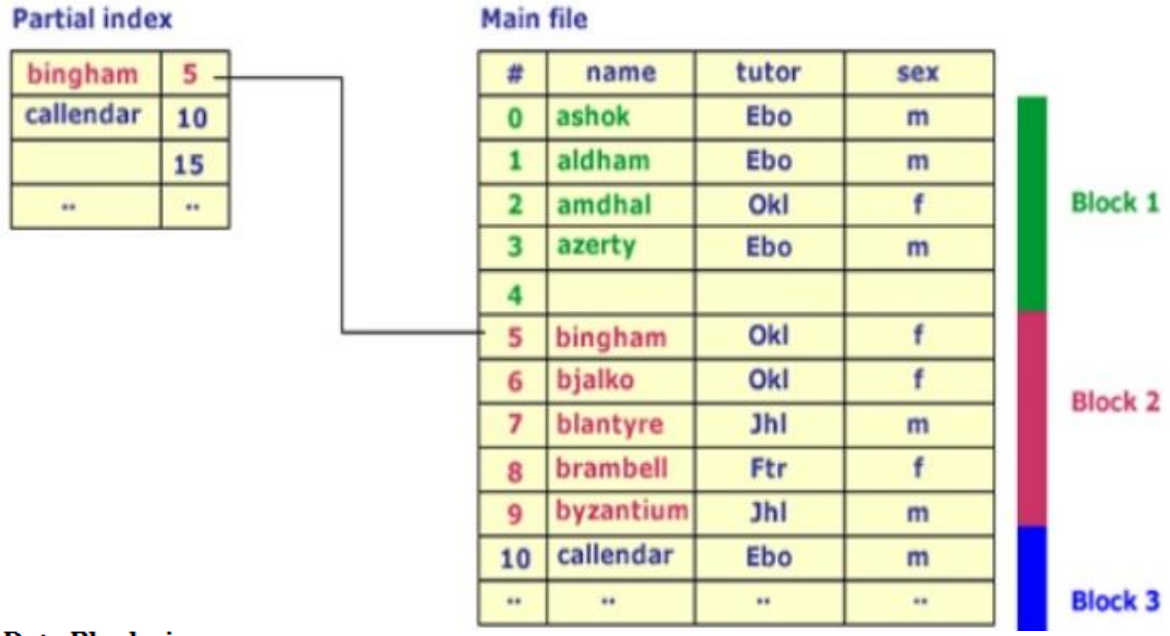
- **Data File** contains records in sequential scheme.
- **Index File** contains the primary key and its address in the data file.

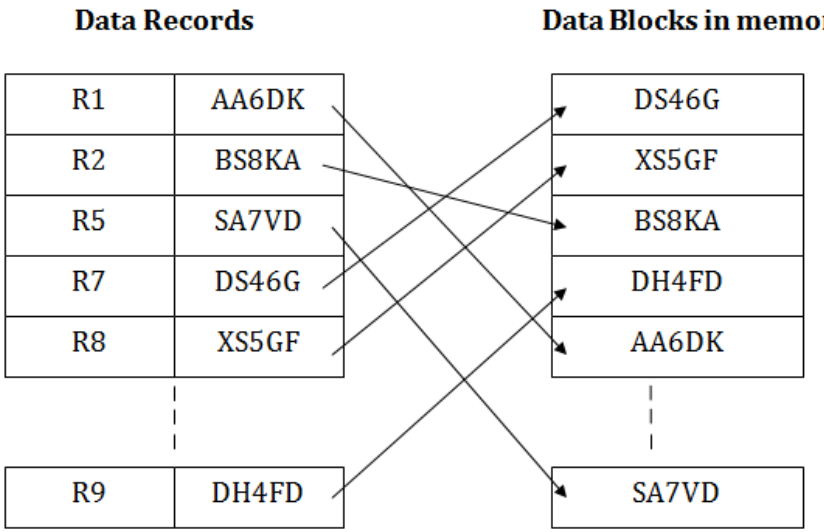
Following are the key attributes of sequential file organization −

- Records can be read in sequential order just like in sequential file organization.
- Records can be accessed randomly if the primary key is known. Index file is used to get the address of a record and then the record is fetched from the data file.
- Sorted index is maintained in this file system which relates the key value to the position of the record in the file.
- Alternate index can also be created to fetch the records.

File Organizations

Indexed Sequential File





File Organizations

Advantages

- Sequential file and random file access is possible.
- It accesses the records very fast if the index table is properly organized.
- It provides quick access for sequential and direct processing.
- It reduces the degree of the sequential search.

Disadvantages

- Indexed sequential access file requires unique keys and periodic reorganization.
- It requires more storage space.
- It is expensive because it requires special software.
- It is less efficient in the use of storage space as compared to other file organizations.

ER Model & Relational Model

ER Model (Entity Relationship Model)

- ER model stands for an Entity-Relationship model. It is a high-level data model. This model is used to define the data elements and relationship for a specified system.
- It develops a conceptual design for the database. It also develops a very simple and easy to design view of data.
- In ER modeling, the database structure is portrayed as a diagram called an entity-relationship diagram.
- ER Diagram graphically expresses the logical structure of database (schema), and it uses-
  - **Rectangle**: Represents Entity sets.
  - **Ellipses**: Attributes
  - **Diamonds**: Relationship Set
  - **Lines**: They link attributes to Entity Sets and Entity sets to Relationship Set
  - **Double Ellipses:** Multivalued Attributes
  - **Dashed Ellipses**: Derived Attributes
  - **Double Rectangles**: Weak Entity Sets
  - **Double Lines**: Total participation of an entity in a relationship set

ER Model (Entity Relationship Model)

- ER model stands for an Entity-Relationship model.
- It is a high-level data model. This model is used to define the data elements and relationship for a specified system.
- It develops a conceptual design for the database.
- It also develops a very simple and easy to design view of data.
- In ER modeling, the database structure is portrayed as a diagram called an entity-relationship diagram.

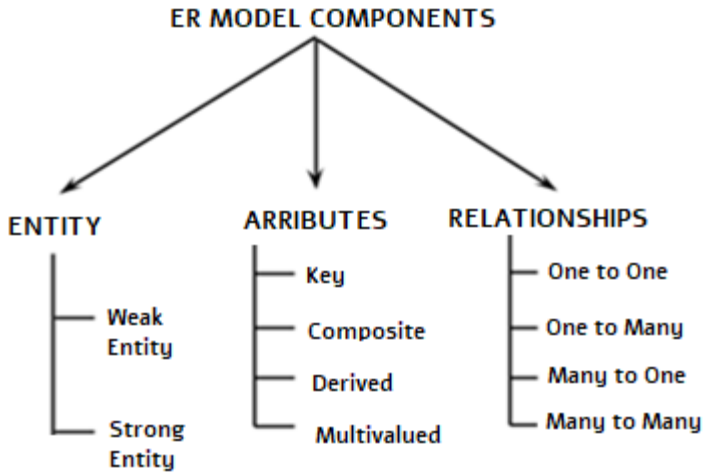
For example:

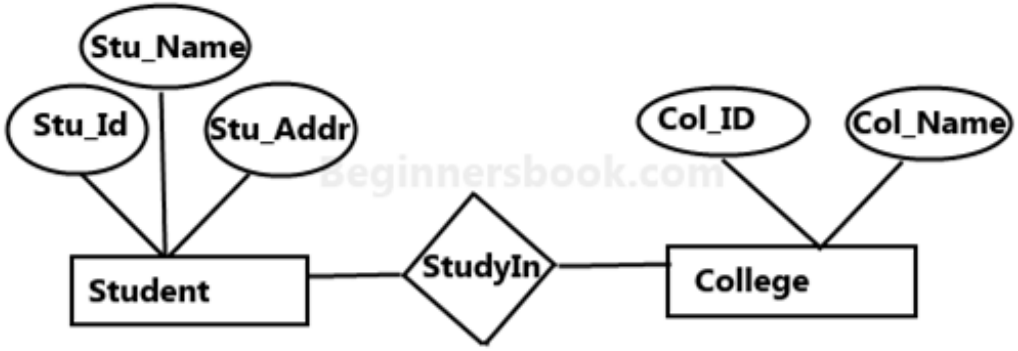
Suppose we design a school database. In this database, the student will be an entity with attributes like address, name, id, age, etc. The address can be another entity with attributes like city, street name, pin code, etc and there will be a relationship between them.

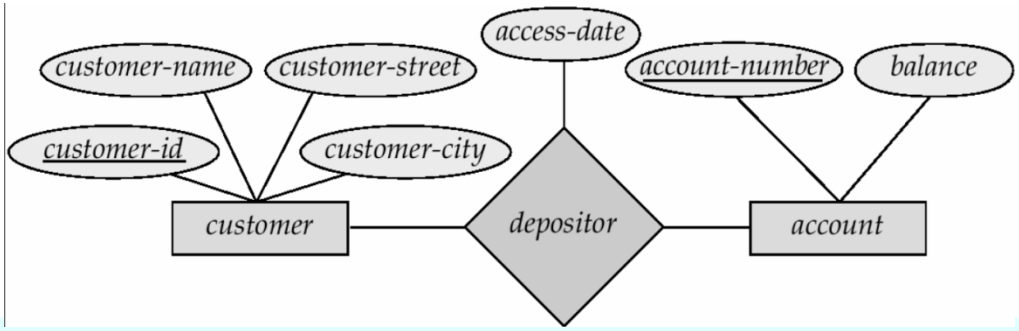
ER Model (Entity Relationship Model)

A simple ER Diagram:

- In the following diagram we have two entities Student and College and their relationship.
- The relationship between Student and College is many to one as a college can have many students however a student cannot study in multiple colleges at the same time.
- Student entity has attributes such as Stu_Id, Stu_Name & Stu_Addr and College entity has attributes such as Col_ID & Col_Name.



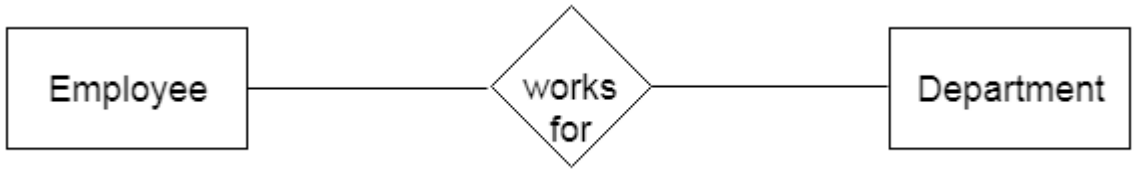




ER Model (Entity Relationship Model)

Entity:

- An entity may be any object, class, person or place. In the ER diagram, an entity can be represented as rectangles.
- An Entity may be an object with a physical existence – a particular person, car, house, or employee – or it may be an object with a conceptual existence – a company, a job, or a university course.
- An Entity is an object of Entity Type and set of all entities is called as entity set. e.g.; E1 is an entity having Entity Type Student and set of all students is called Entity Set.
- Consider an organization as an example manager, product, employee, department etc. can be taken as an entity.



ER Model (Entity Relationship Model)

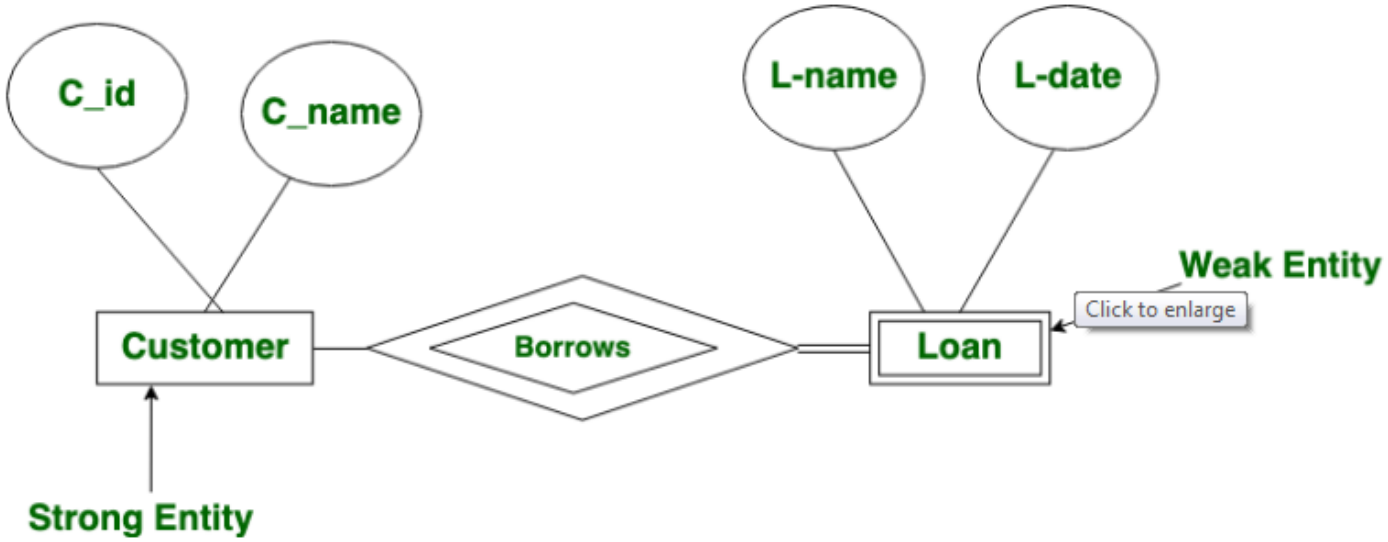
Strong Entity

A strong entity is not dependent of any other entity in the schema. A strong entity will always have a primary key.
Strong entities are represented by a single rectangle.
The relationship of two strong entities is represented by a single diamond.
Various strong entities, when combined together, create a strong entity set.

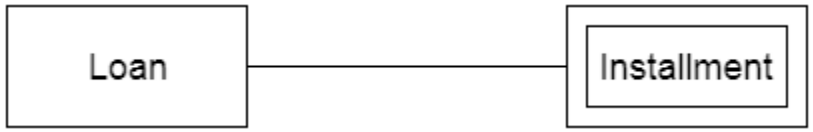
Weak Entity

A weak entity is dependent on a strong entity to ensure the its existence.
Unlike a strong entity, a weak entity does not have any primary key.
It instead has a partial discriminator key. A weak entity is represented by a double rectangle. The relation between one strong and one weak entity is represented by a double diamond.

ER Model (Entity Relationship Model)



A bank loan installment cannot be uniquely identified without knowing the loan to which the installment belongs, so loan installment is a weak entity.



ER Model (Entity Relationship Model)

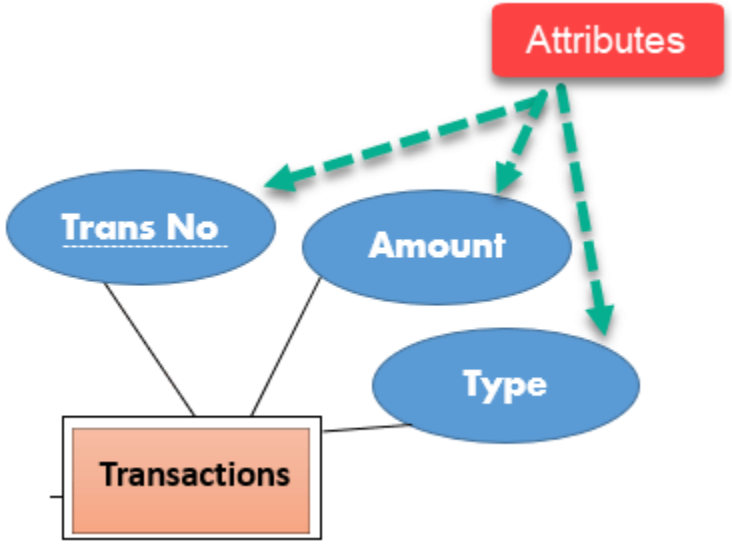
Attribute

Attributes are the **properties which define the entity type**. For example, RollNo, Name, DOB, Age, Address, MobileNo are the attributes which defines entity type Student. In ER diagram, attribute is represented by an oval.

There exists a domain or range of values that can be assigned to attributes. For example, a student's name cannot be a numeric value. It has to be alphabetic. A student's age cannot be negative, etc.

Types of Attributes

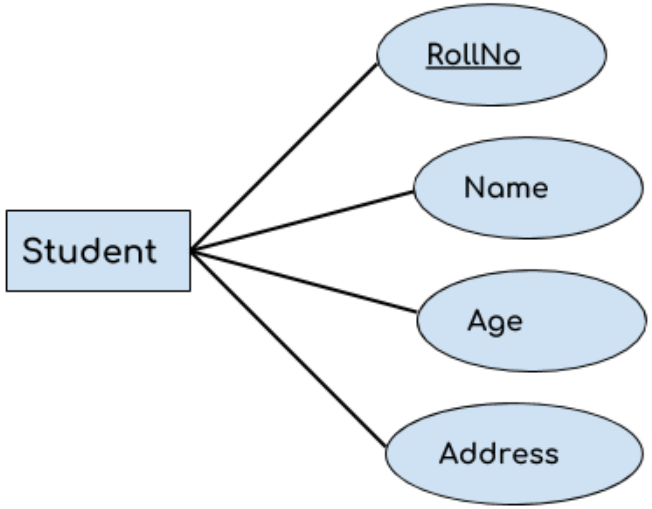
1. Simple attribute
2. Composite attribute
3. Derived attribute
4. Single-value attribute
5. Multi-value attribute



ER Model (Entity Relationship Model)

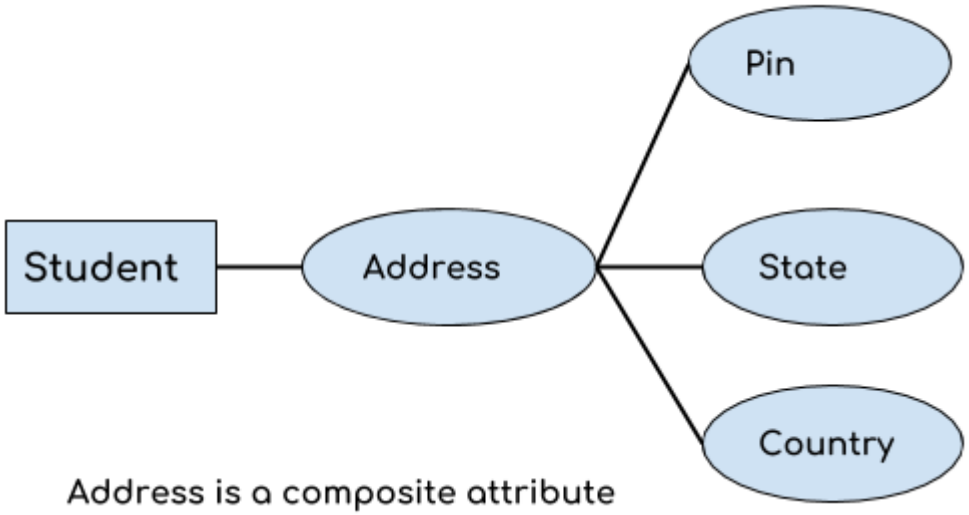
- **Key Attribute:**
  The attribute which **uniquely identifies each entity** in the entity set is called key attribute. For example, RollNo will be unique for each student. In ER diagram, key attribute is represented by an oval with underlying lines.

- **Simple attribute:** Simple attributes are atomic values, which cannot be divided further. For example, In the diagram, RollNo & Age are the Simple Attributes.



ER Model (Entity Relationship Model)

- **Composite Attribute:**
  An attribute **composed of many other attribute** is called as composite attribute. For example, Address attribute of student Entity type consists of Street, City, State, and Country.
  In ER diagram, composite attribute is represented by an oval comprising of ovals. For example, a student's complete name may have first_name and last_name.



Address is a composite attribute

ER Model (Entity Relationship Model)

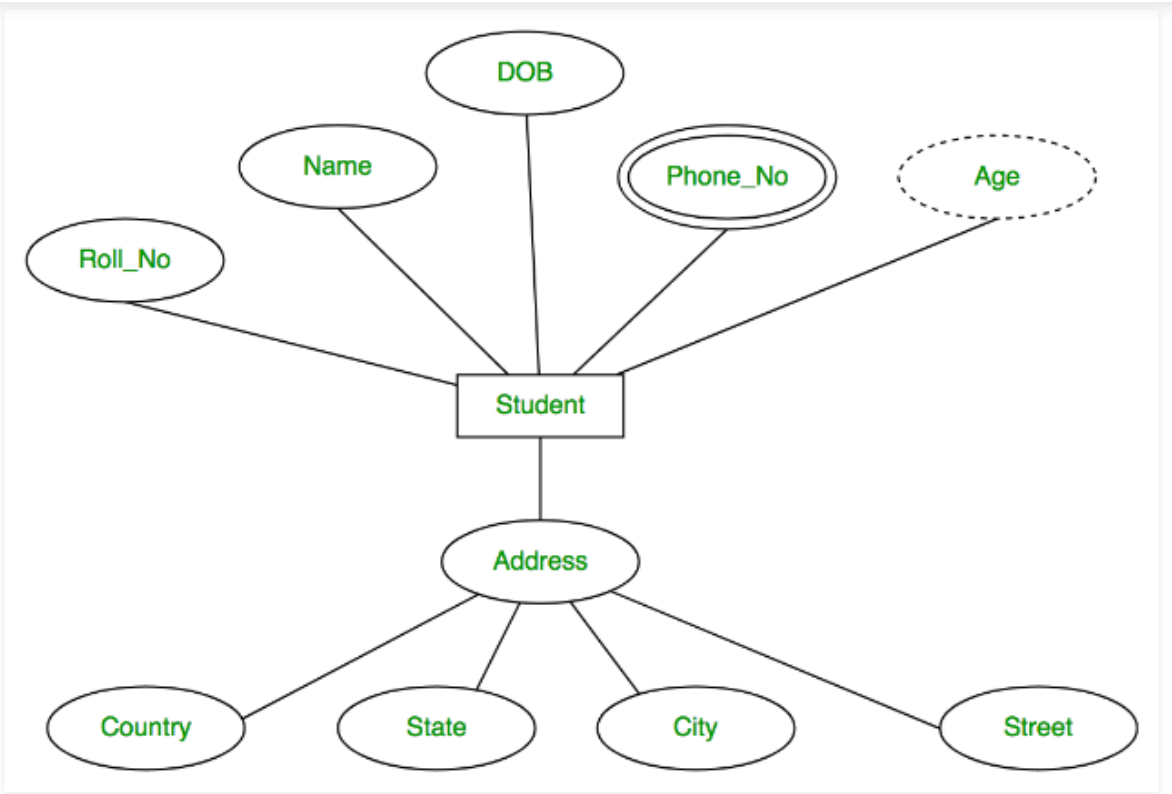
- **Derived attribute:** Derived attributes are the attributes that do not exist in the physical database, but their values are derived from other attributes present in the database.
  For example, Age (can be derived from DOB). In ER diagram, derived attribute is represented by dashed oval.

- **Single-value attribute:** Single-value attributes contain single value.
  For example – Roll Number

- **Multi-value attribute:** Multi-value attributes may contain more than one values. For example, a person can have more than one phone number, email_address, etc.

ER Model (Entity Relationship Model)

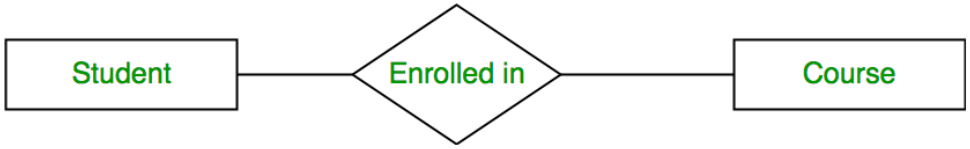
The complete entity type **Student** with its attributes can be represented as:



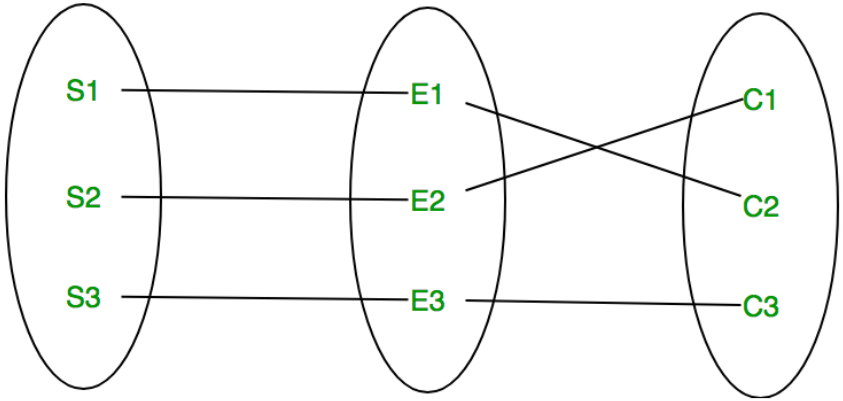
ER Model (Entity Relationship Model)

Relationship Type and Relationship Set

A relationship type represents the **association between entity types**. For example, 'Enrolled in' is a relationship type that exists between entity type Student and Course. In ER diagram, relationship type is represented by a diamond and connecting the entities with lines.



A set of relationships of same type is known as relationship set. The following relationship set depicts S1 is enrolled in C2, S2 is enrolled in C1 and S3 is enrolled in C3.



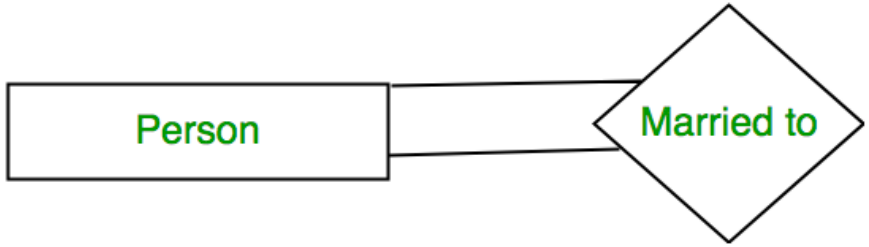
ER Model (Entity Relationship Model)

Degree of a relationship set

The number of different entity sets **participating in a relationship** set is called as degree of a relationship set.

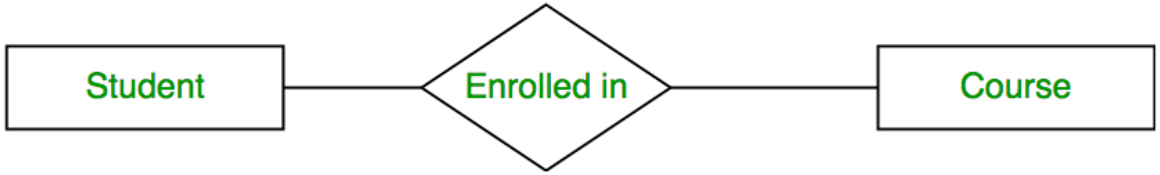
Unary Relationship

When there is **only ONE entity set participating in a relation**, the relationship is called as unary relationship. For example, one person is married to only (



Binary Relationship

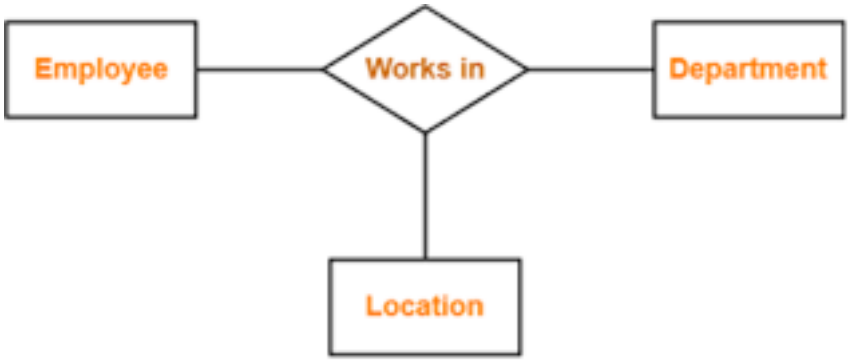
When there are **TWO entities set participating in a relation**, the relationship is called as binary relationship. For example, Student is enrolled in Course.



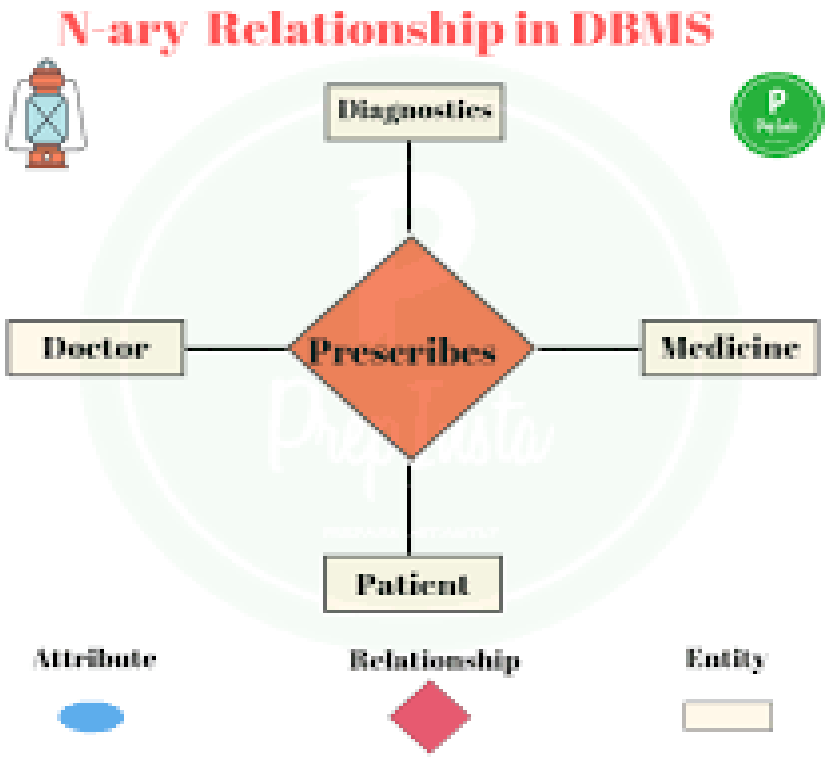
ER Model (Entity Relationship Model)

n-ary Relationship –

When there are n entities set participating in a relation, the relationship is called as n-ary relationship.



Ternary Relationship Set

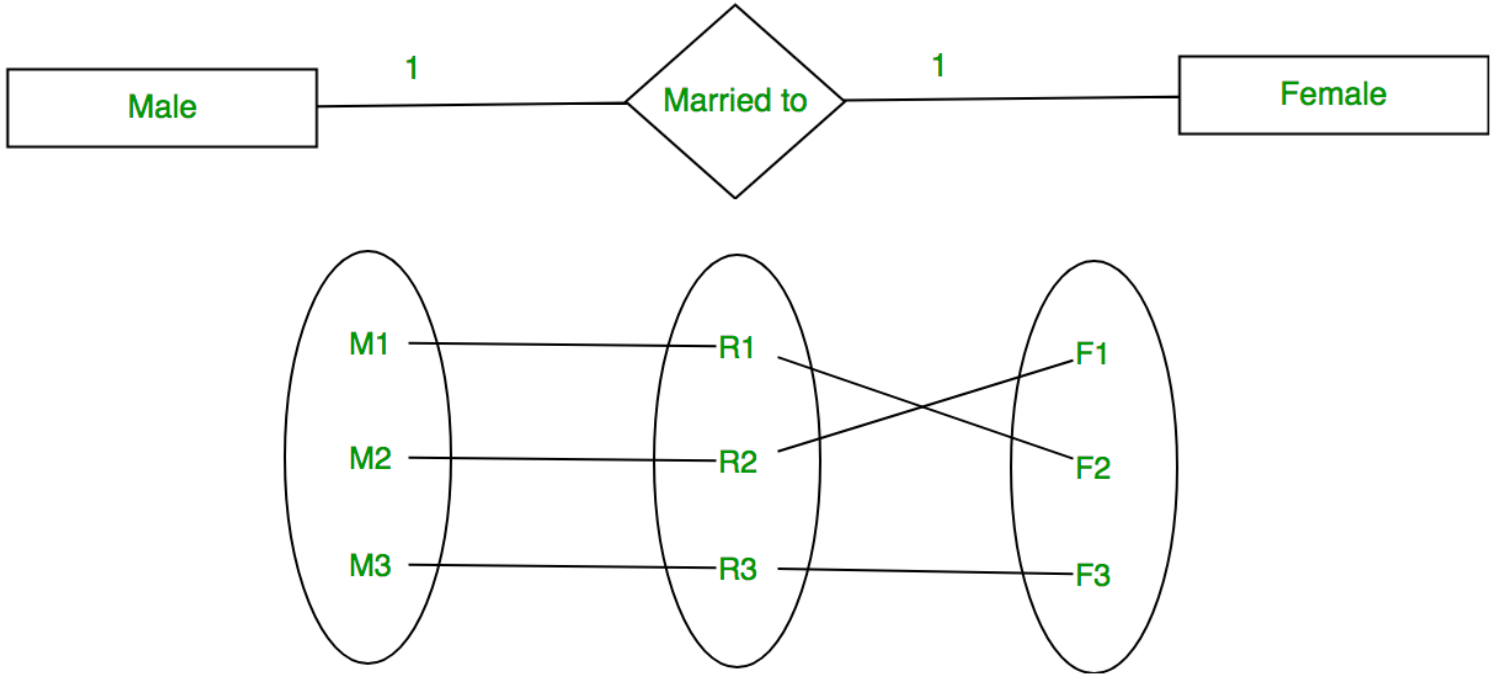


ER Model (Entity Relationship Model)

Cardinality

The **number of times an entity of an entity set participates in a relationship** set is known as cardinality. Cardinality can be of different types:

One to one – When each entity in each entity set can take part **only once in the relationship**, the cardinality is one to one. Let us assume that a male can marry to one female and a female can marry to one male. So the relationship will be one to one.

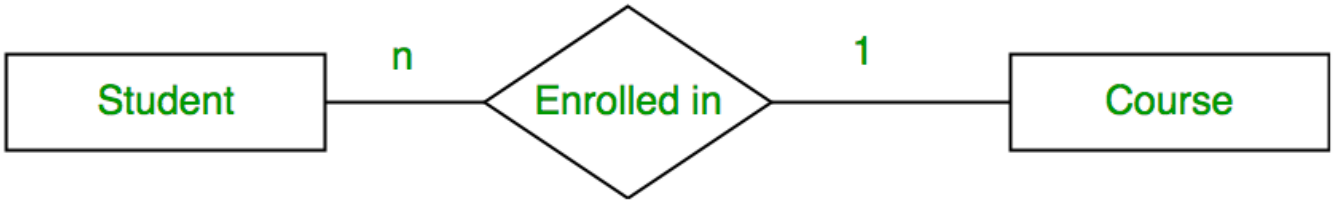


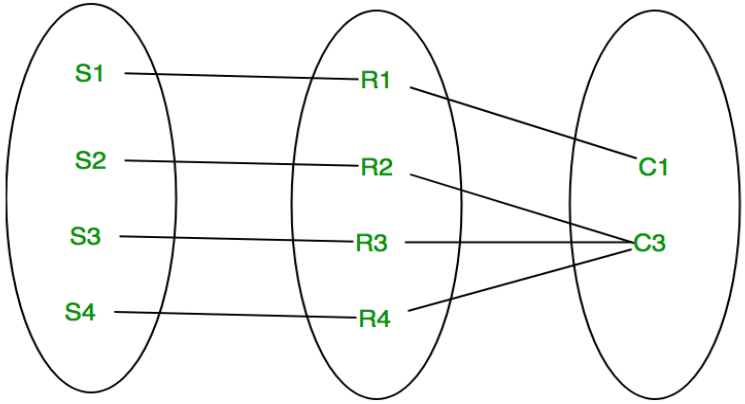
ER Model (Entity Relationship Model)

Many to one – When entities in one entity set **can take part only once in the relationship set and entities in other entity set can take part more than once in the relationship set,** cardinality is many to one.

Let us assume that a student can take only one course but one course can be taken by many students. So the cardinality will be n to 1. It means that for one course there can be n students but for one student, there will be only one course.

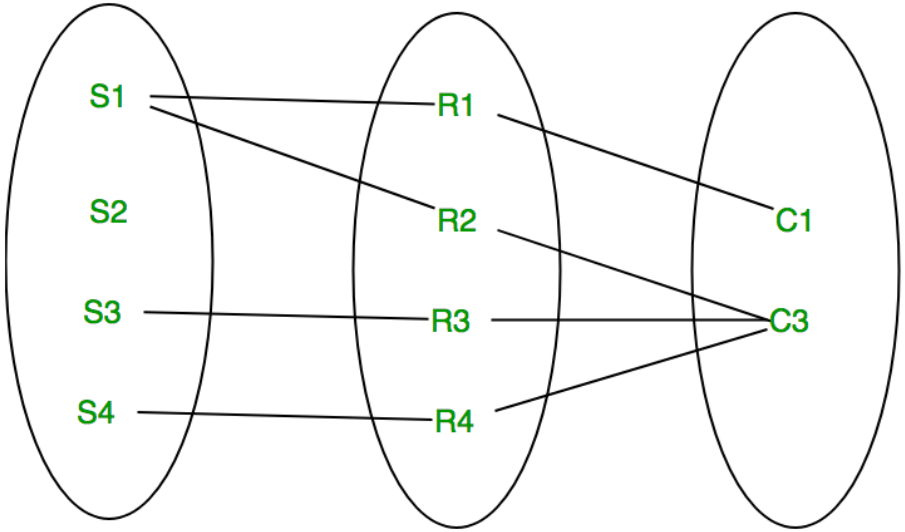
In this case, each student is taking only 1 course but 1 course has been taken by many students.





ER Model (Entity Relationship Model)

Many to many – When entities in all entity sets can **take part more than once in the relationship** cardinality is many to many. Let us assume that a student can take more than one course and one course can be taken by many students. So the relationship will be many to many.
In this example, student S1 is enrolled in C1 and C3 and Course C3 is enrolled by S1, S3 and S4. So it is many to many relationships.

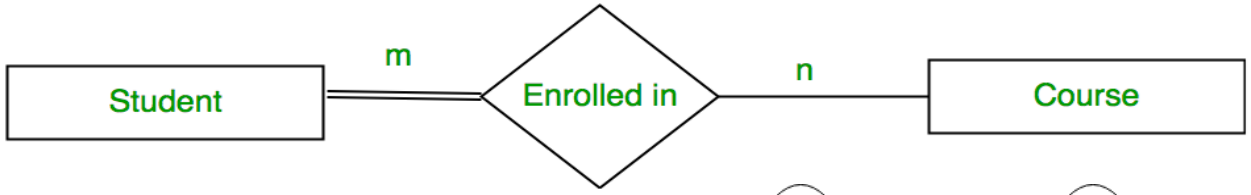


ER Model (Entity Relationship Model)

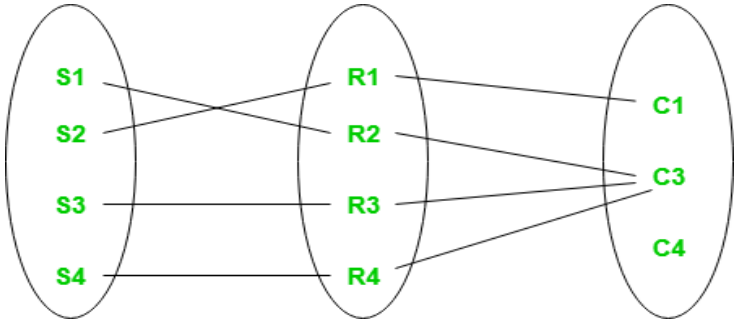
Participation Constraint

Participation Constraint is applied on the entity participating in the relationship set.

1. **Total Participation –** Each entity in the entity set **must participate** in the relationship. If each student must enroll in a course, the participation of student will be total. Total participation is shown by double line in ER diagram.
2. **Partial Participation –** The entity in the entity set **may or may NOT participate** in the relationship. If some courses are not enrolled by any of the student, the participation of course will be partial. The diagram depicts the 'Enrolled in' relationship set with Student Entity set having total participation and Course Entity set having partial participation.



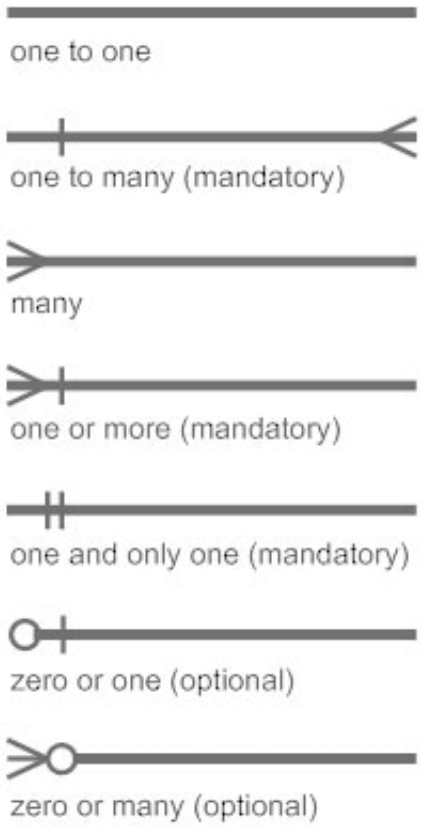
Every student in Student Entity set is participating in relationship but there exists a course C4 which is not taking part in the relationship.



ER Model (Entity Relationship Model)

Notation of ER diagram

Database can be represented using the notations. In ER diagram, many notations are used to express the cardinality. These notations are as follows:



ER Model (Entity Relationship Model)

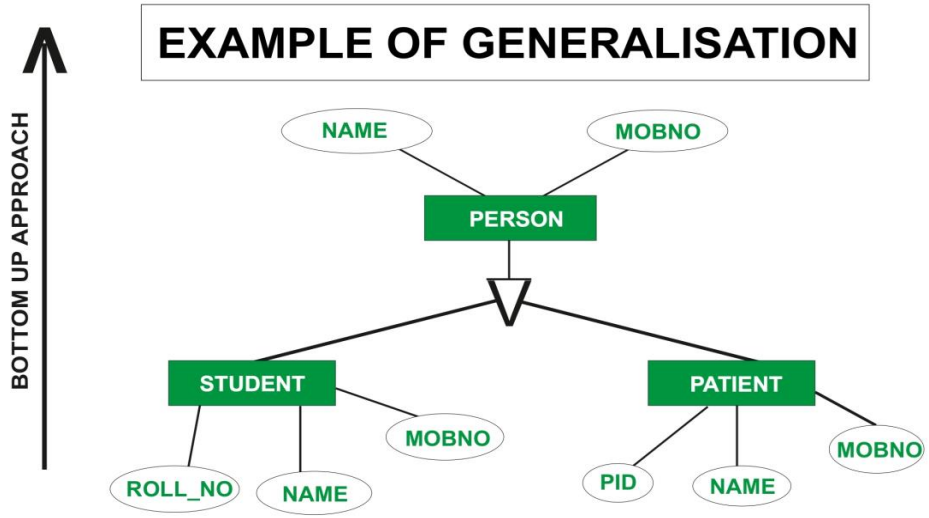
Generalization

Generalization is like a bottom-up approach in which two or more entities of lower level combine to form a higher level entity if they have some attributes in common.

In generalization, an entity of a higher level can also combine with the entities of the lower level to form a further higher level entity.

Generalization is more like subclass and super class system, but the only difference is the approach. Generalization uses the bottom-up approach.

In generalization, entities are combined to form a more generalized entity, i.e., subclasses are combined to make a super class.



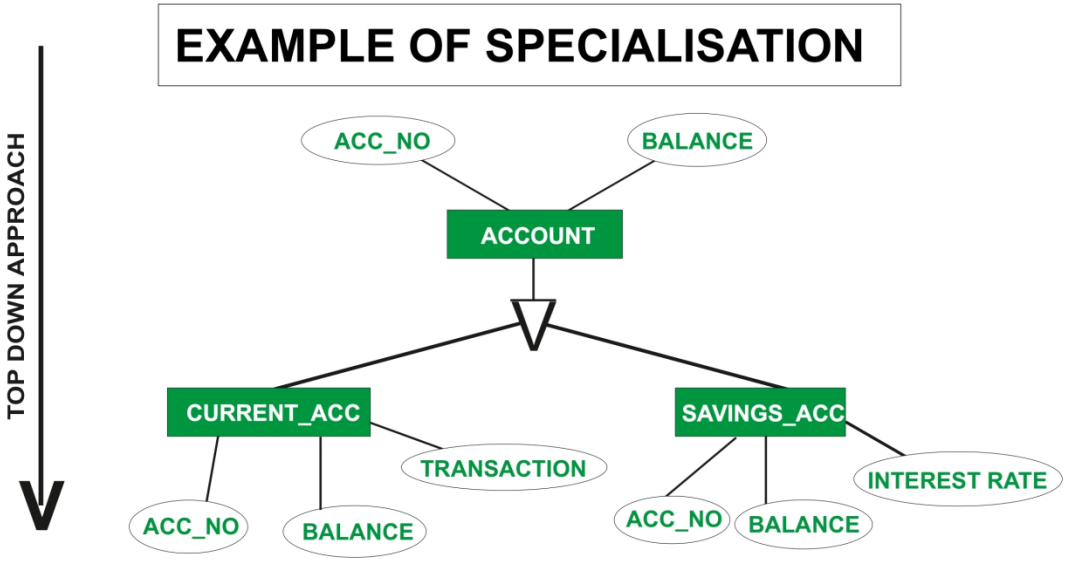
ER Model (Entity Relationship Model)

Specialization

Specialization is a top-down approach, and it is opposite to Generalization. In specialization, one higher level entity can be broken down into two lower level entities.

Specialization is used to identify the subset of an entity set that shares some distinguishing characteristics.

Normally, the super class is defined first, the subclass and its related attributes are defined next, and relationship set are then added.

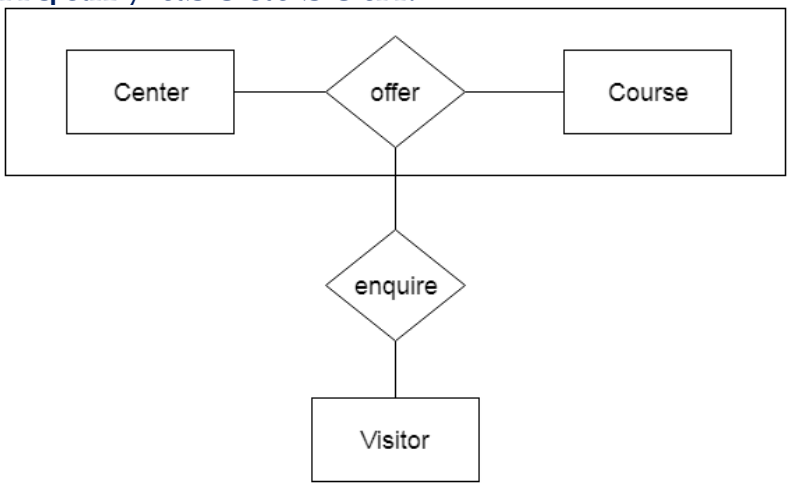


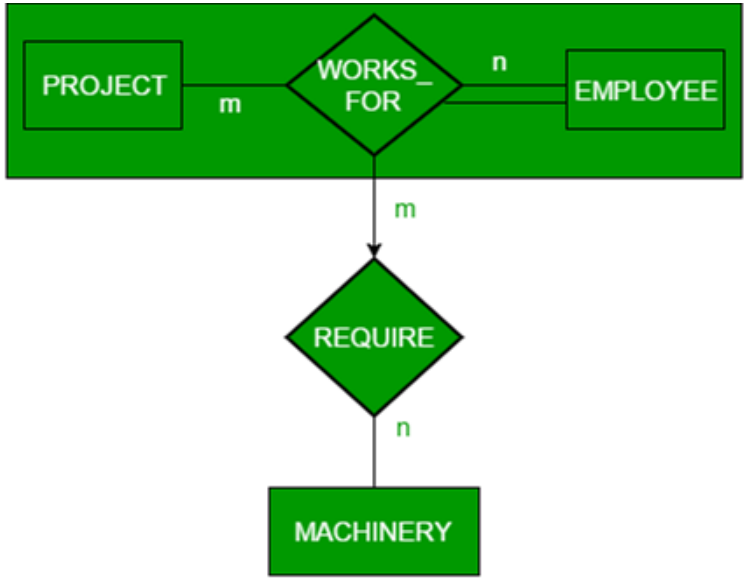
ER Model (Entity Relationship Model)

Aggregation

In aggregation, the relation between two entities is treated as a single entity. In aggregation, relationship with its corresponding entities is aggregated into a higher level entity.

For example: Center entity offers the Course entity act as a single entity in the relationship which is in a relationship with another entity visitor. In the real world, if a visitor visits a coaching center then he will never enquiry about the Course only or just about the Center instead he will ask the enquiry about both.

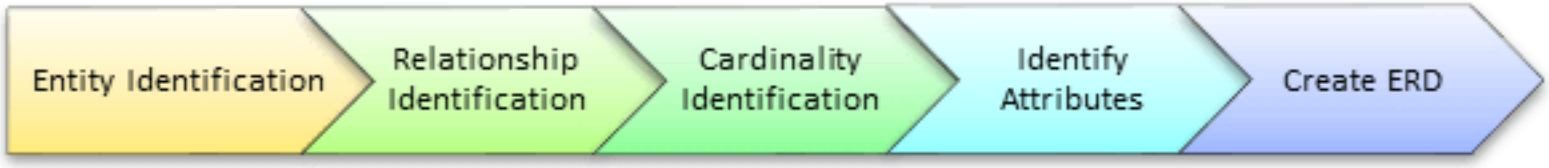




Aggregation

Steps to Create an ERD

Following are the steps to create an ERD.

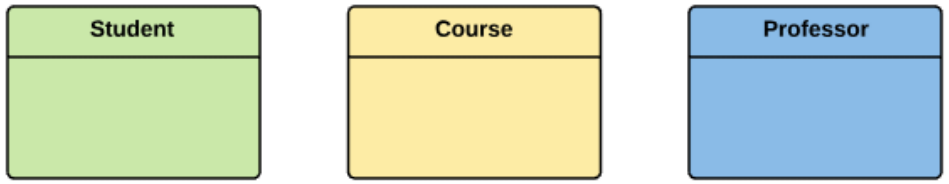


Let's study them with an example:
In a university, a Student enrolls in Courses. A student must be assigned to at least one or more Courses. Each course is taught by a single Professor. To maintain instruction quality, a Professor can deliver only one course

Step 1) Entity Identification

We have three entities

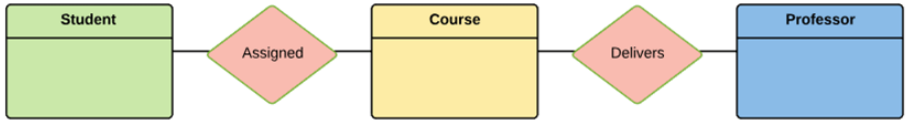
1. Student
2. Course
3. Professor



Step 2) Relationship Identification

We have the following two relationship

1. The student is **assigned** a cours
2. Professor **delivers** a course

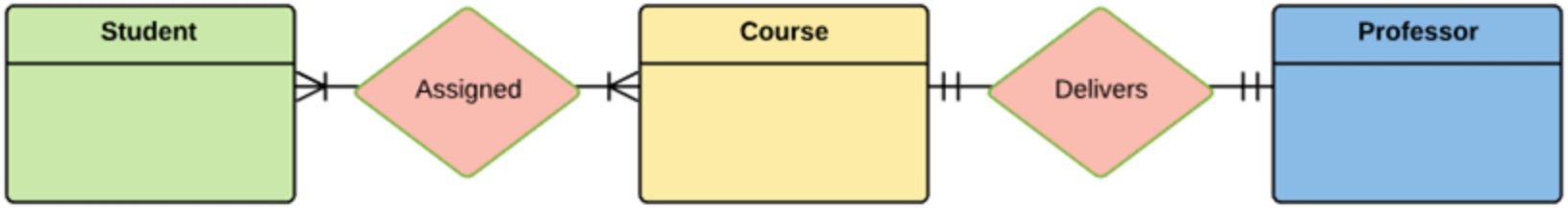


Steps to Create an ERD

Step 3) Cardinality Identification

For them problem statement we know that,

1. A student can be assigned **multiple** courses
2. A Professor can deliver only **one** course



Step 4) Identify Attributes

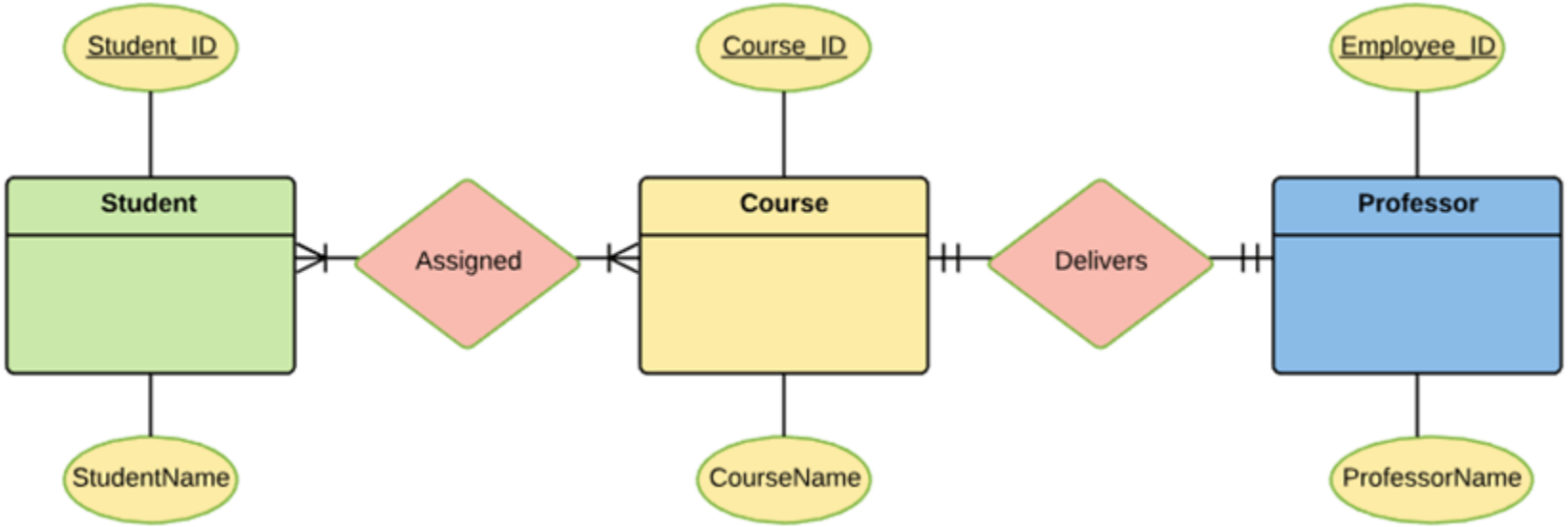
You need to study the files, forms, reports, data currently maintained by the organization to identify attributes. You can also conduct interviews with various stakeholders to identify entities. Initially, it's important to identify the attributes without mapping them to a particular entity.

Once, you have a list of Attributes, you need to map them to the identified entities. Ensure an attribute is to be paired with exactly one entity. If you think an attribute should belong to more than one entity, use a modifier to make it unique.

Once the mapping is done, identify the primary Keys. If a unique key is not readily available, create one.

Steps to Create an ERD

Entity	Primary Key	Attribute
Student	Student_ID	StudentName
Professor	Employee_ID	ProfessorName
Course	Course_ID	CourseName

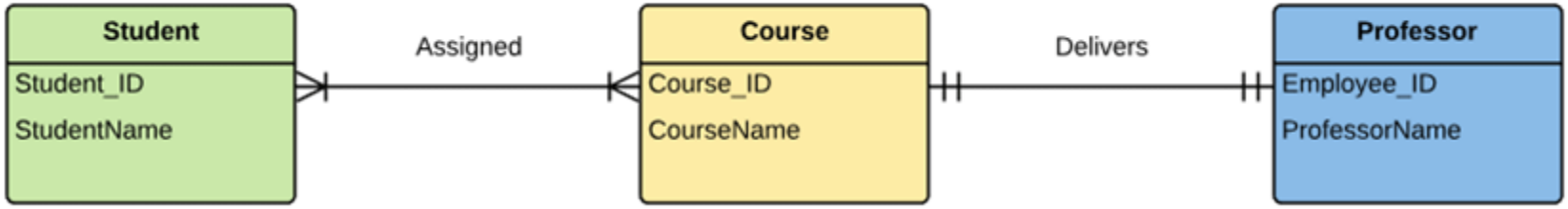


For Course Entity, attributes could be Duration, Credits, Assignments, etc. For the sake of ease we have considered just one attribute.

Steps to Create an ERD

Step 5) Create the ERD

A more modern representation of ERD Diagram

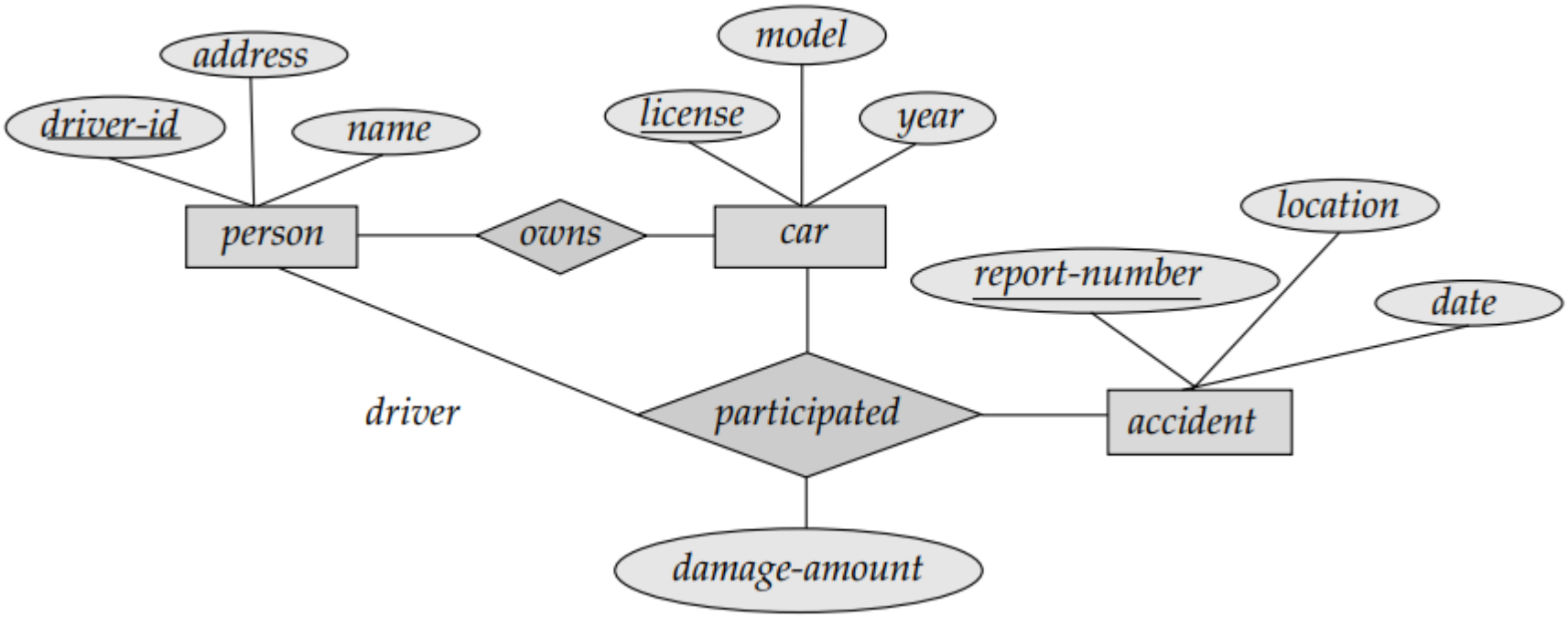


Best Practices for Developing Effective ER Diagrams

1. Eliminate any redundant entities or relationships
2. You need to make sure that all your entities and relationships are properly labeled
3. There may be various valid approaches to an ER diagram. You need to make sure that the ER diagram supports all the data you need to store
4. Never connect relationships to each other

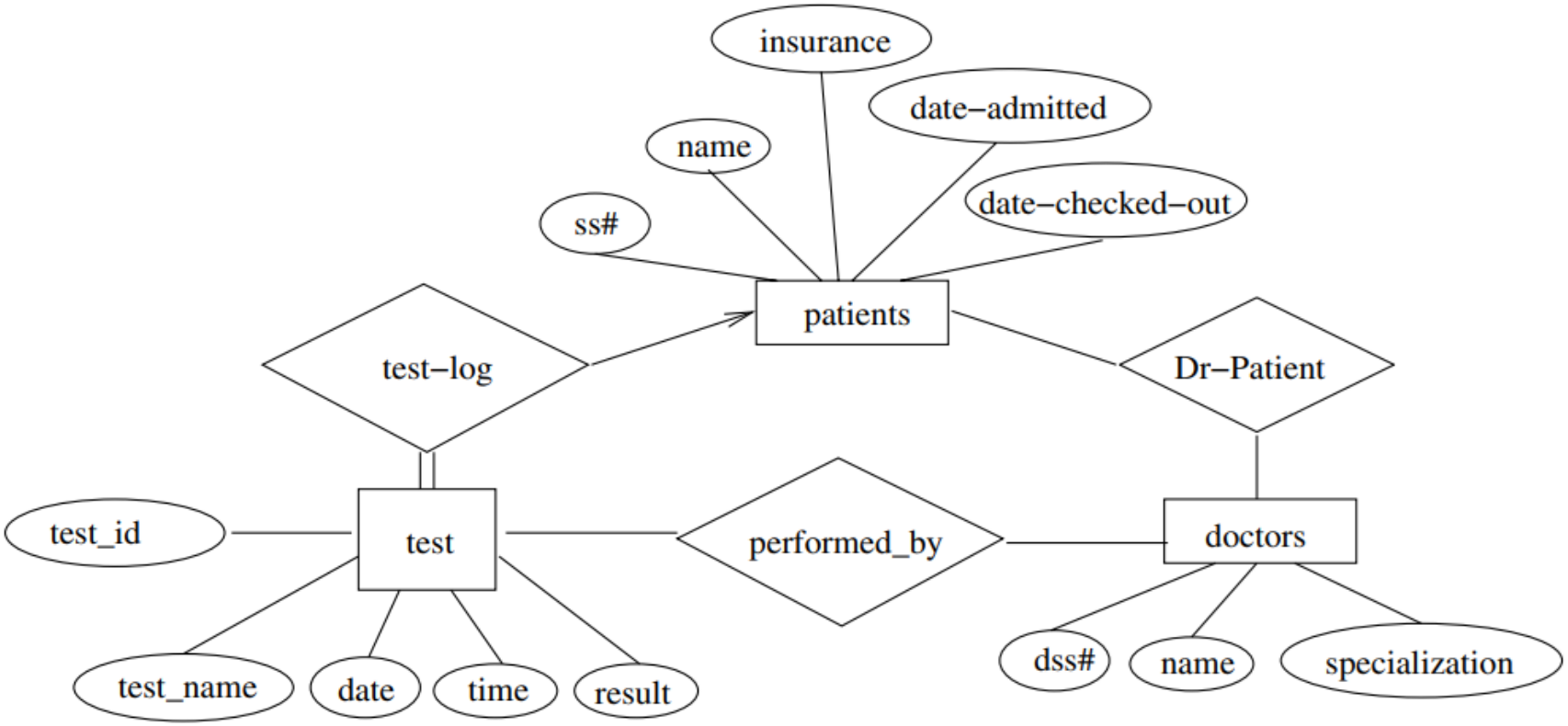
Car-Insurance Company

Construct an E-R diagram for a car-insurance company whose customers own one or more cars each. Each car has associated with it zero to any number of recorded accidents.



Hospital Record Management System

Construct an E-R diagram for a hospital with a set of patients and a set of medical doctors. Associate with each patient a log of the various tests and examinations conducted.



University Registrar Office

A university registrar's office maintains data about the following entities:

(a) courses, including number, title, credits, syllabus, and prerequisites;

(b) course offerings, including course number, year, semester, section number, instructor(s), timings, and classroom;

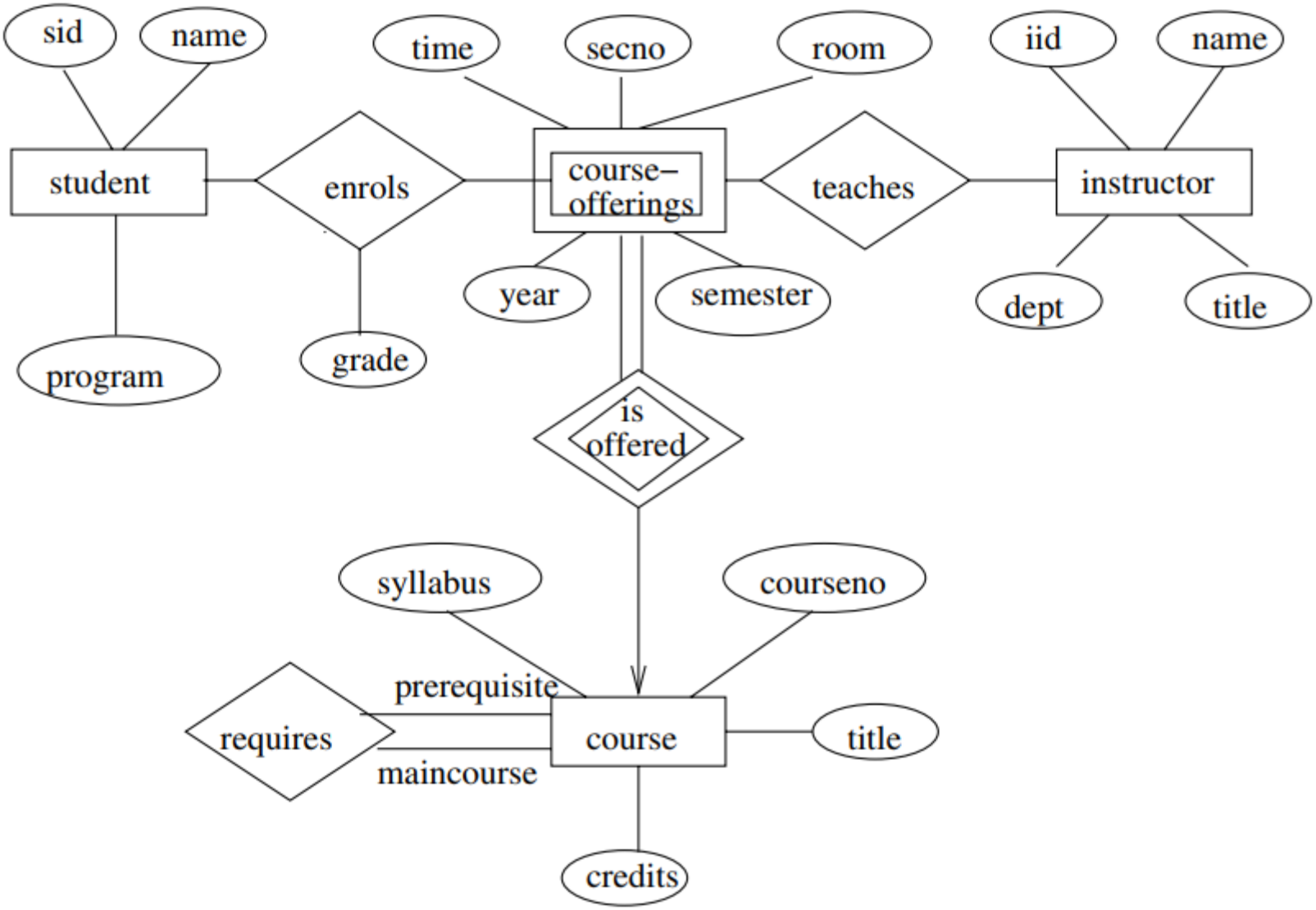
(c) students, including student-id, name, and program; and

(d) instructors, including identification number, name, department, and title. Further, the enrollment of students in courses and grades awarded to students in each course they are enrolled for must be appropriately modeled.

Construct an E-R diagram for the registrar's office. Document all assumptions that

you make about the mapping constraints.

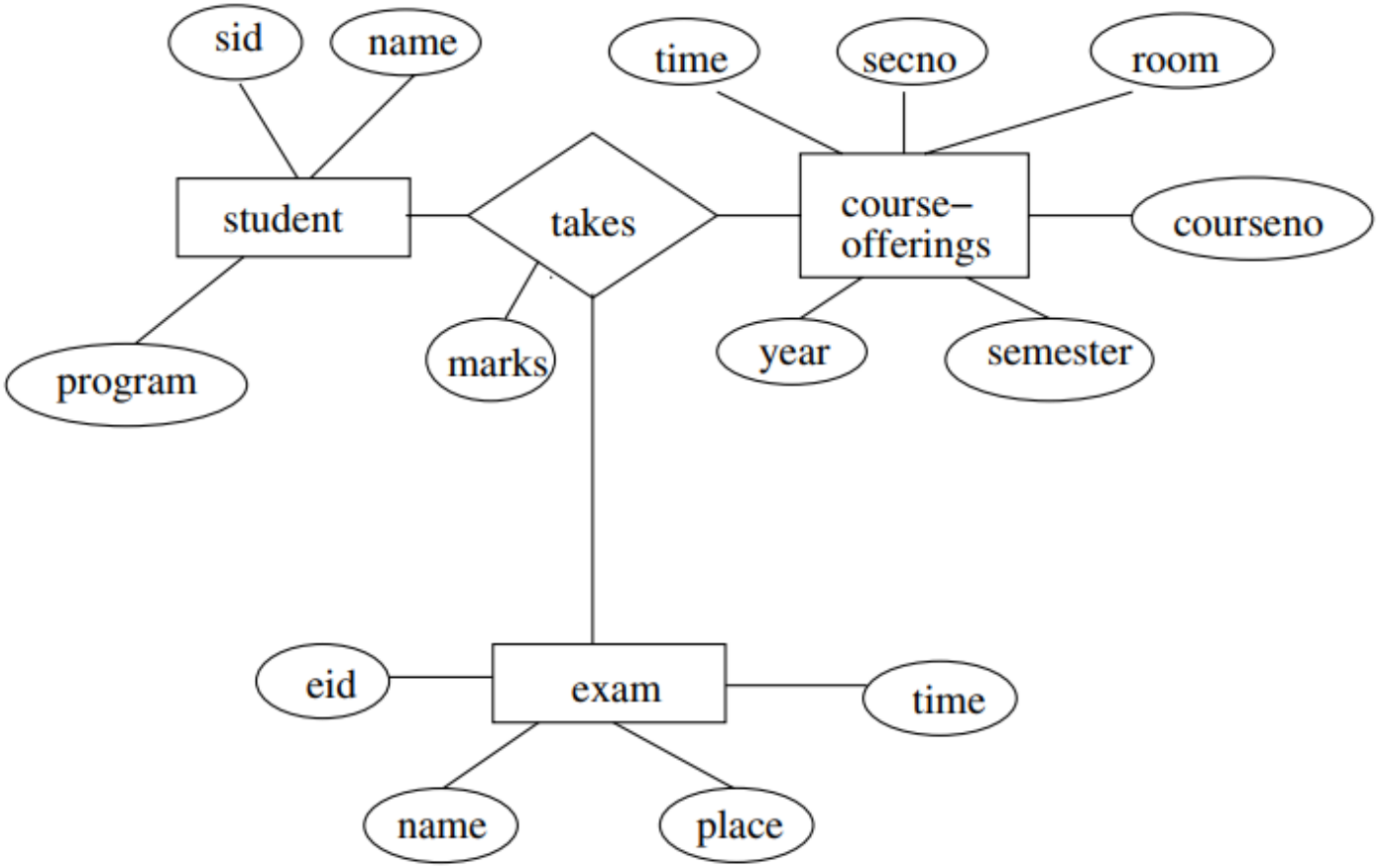
University Registrar Office



Model Examination System

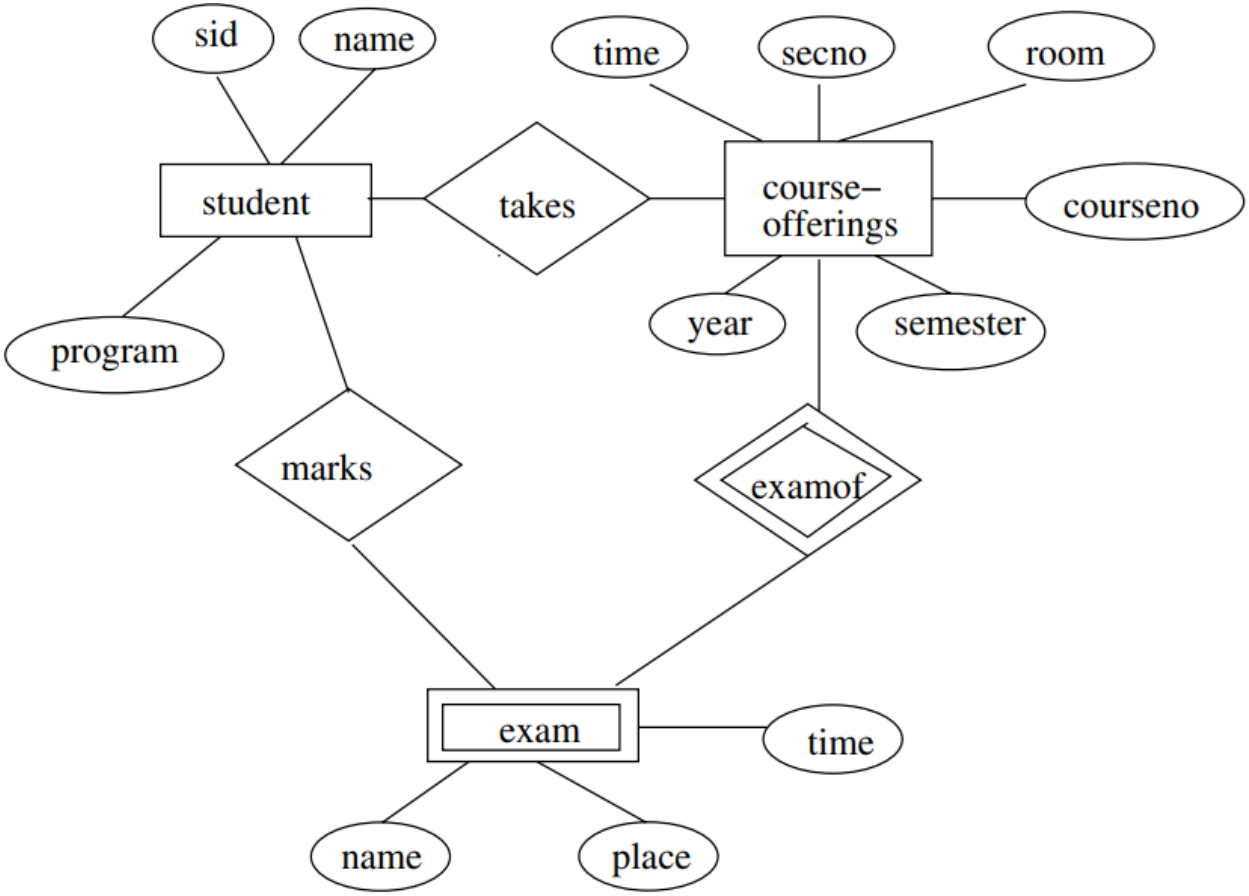
Consider a database used to record the marks that students get in different exams of different course offerings.

1. Construct an E-R diagram that models exams as entities, and uses a ternary relationship, for the above database.



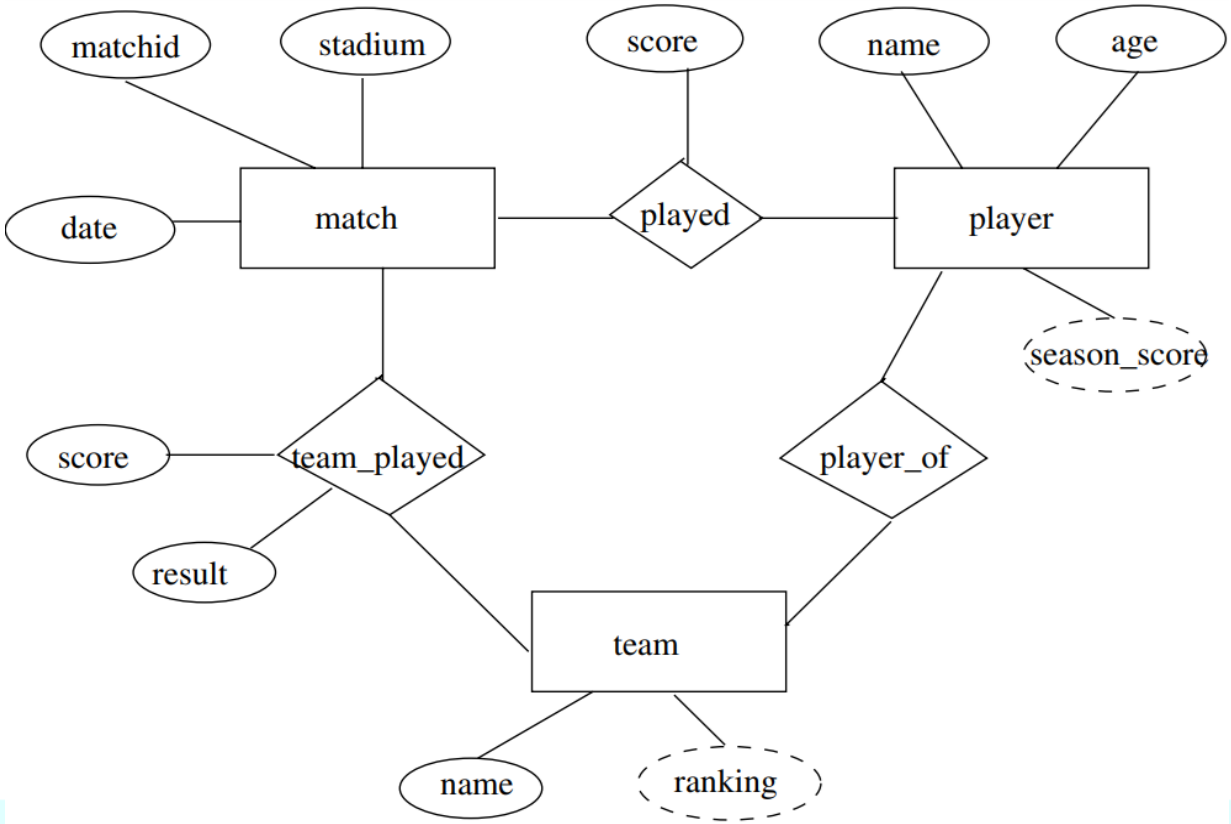
Model Examination System

2. Construct an alternative E-R diagram that uses only a binary relationship between students and course-offerings. Make sure that only one relationship exists between a particular student and course-offering pair, yet you can represent the marks that a student gets in different exams of a course offering



Model Examination System

3. Design an E-R diagram for keeping track of the exploits of your favorite sports team. You should store the matches played, the scores in each match, the players in each match and individual player statistics for each match. Summary statistics should be modeled as derived attributes. Extend the E-R diagram of the previous question to track the same information for all teams in a league.



Music Database

The Music Database

The music database stores details of a personal music library, and could be used to manage your MP3, CD, or vinyl collection. Because this database is for a personal collection, it's relatively simple and stores only the relationships between artists, albums, and tracks. It ignores the requirements of many music genres, making it most useful for storing popular music and less useful for storing jazz or classical music.

We first draw up a clear list of requirements for our database:

- The collection consists of albums.
- An album is made by exactly one artist.
- An artist makes one or more albums.
- An album contains one or more tracks
- **Artists, albums, and tracks each have a name.**
- Each track is on exactly one album.
- Each track has a time length, measured in seconds.
- When a track is played, the date and time the playback began (to the nearest second) should be recorded; this is used for reporting when a track was last played, as well as the number of times music by an artist, from an album, or a track has been played.

Music Database

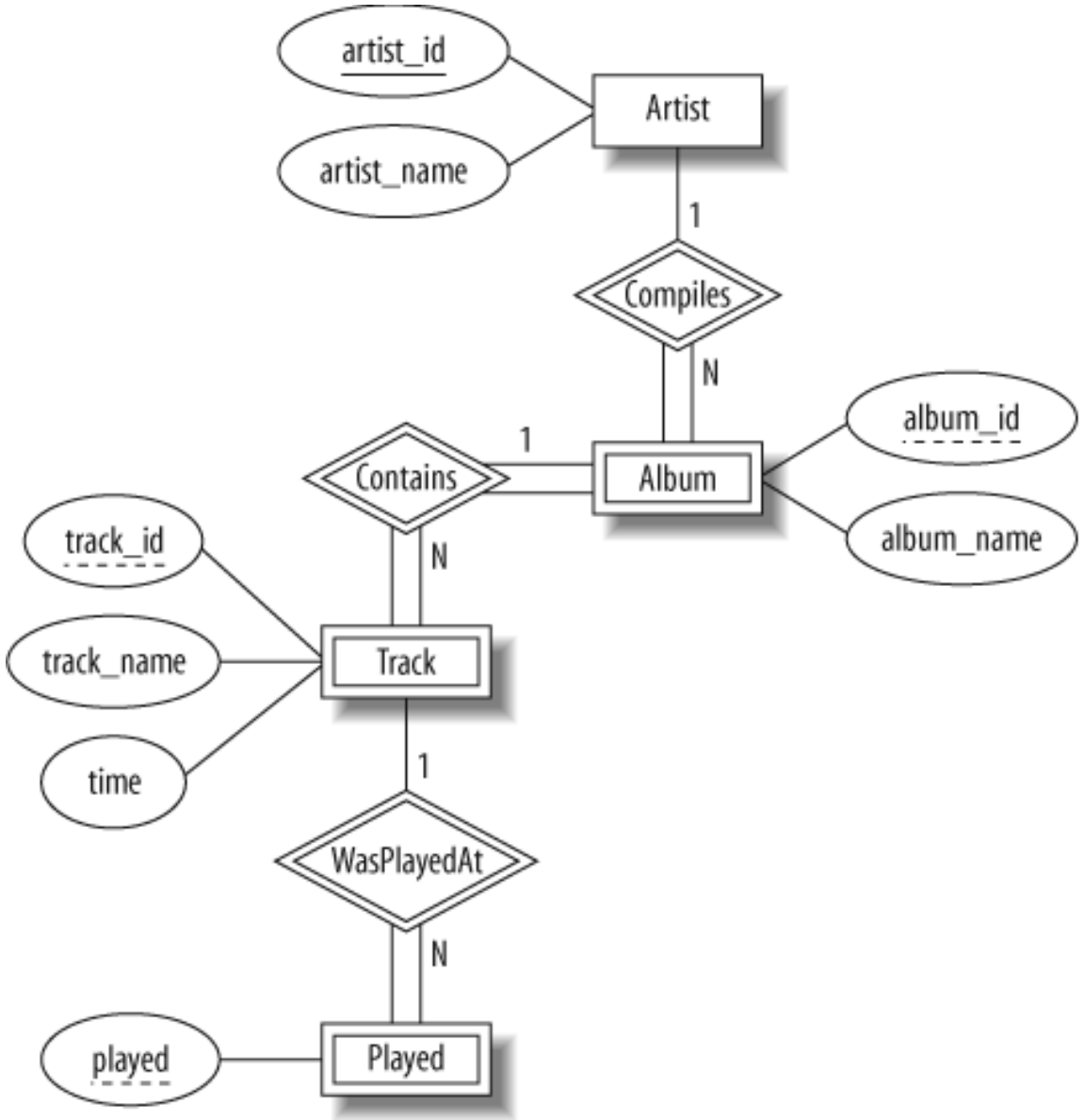
There's no requirement to capture composers, group members or sidemen, recording date or location, the source media, or any other details of artists, albums, or tracks.
The ER diagram derived from our requirements is shown in Figure.

You'll notice that it consists of only one-to-many relationships:

- one artist can make many albums,
- one album can contain many tracks, and
- one track can be played many times.

Conversely, each play is associated with one track, a track is on one album, and an album is by one artist. The attributes are straightforward: artists, albums, and tracks have names, as well as identifiers to uniquely identify each entity. The track entity has a time attribute to store the duration, and the played entity has a timestamp to store when the track was played.

Music Database



University Database

The University Database

The university database stores details about university students, courses, the semester a student took a particular course (and his mark and grade if he completed it), and what degree program each student is enrolled in. The database is a long way from one that'd be suitable for a large tertiary institution, but it does illustrate relationships that are interesting to query, and it's easy to relate to when you're learning SQL. We explain the requirements next and discuss their shortcomings at the end of this section.

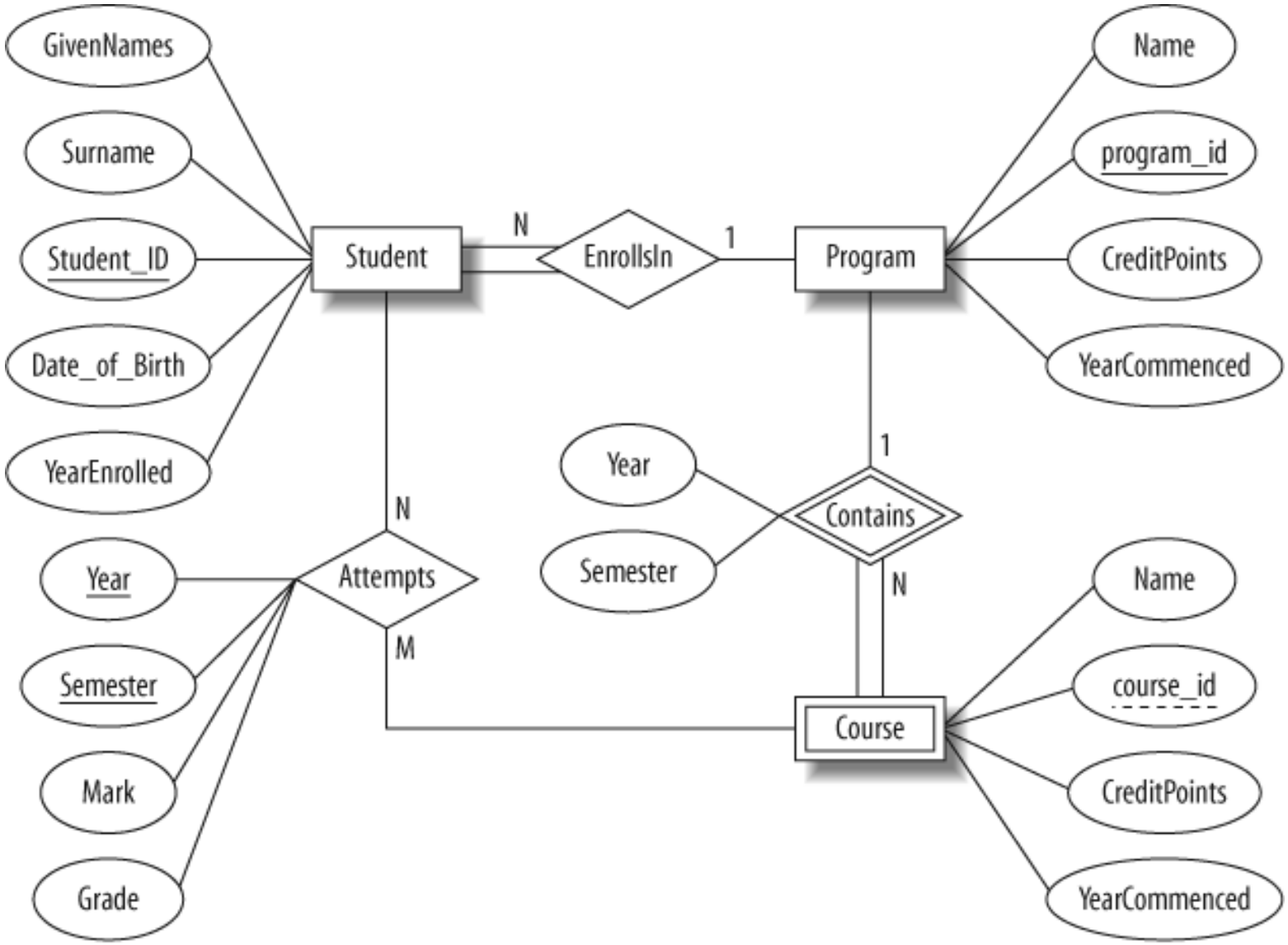
Consider the following requirements list:

- The university offers one or more programs.
- A program is made up of one or more courses.
- A student must enroll in a program.
- A student takes the courses that are part of her program.
- A program has a name, a program identifier, the total credit points required to graduate, and the year it commenced.
- A course has a name, a course identifier, a credit point value, and the year it commenced.

University Database

- A course has a name, a course identifier, a credit point value, and the year it commenced.
- Students have one or more given names, a surname, a student identifier, a date of birth, and the year they first enrolled. We can treat all given names as a single object—for example, “John Paul.”
- When a student takes a course, the year and semester he attempted it are recorded. When he finishes the course, a grade (such as A or B) and a mark (such as 60 percent) are recorded.
- Each course in a program is sequenced into a year (for example, year 1) and a semester (for example, semester 1).
- The ER diagram derived from our requirements is shown in Figure. Although it is compact, the diagram uses some advanced features, including relationships that have attributes and two many-to-many relationships.

University Database



University Database

In our design:

- Student is a strong entity, with an identifier, student_id, created to be the primary key used to distinguish between students (remember, we could have several students with the same name).
- Program is a strong entity, with the identifier program_id as the primary key used to distinguish between programs.
- Each student must be enrolled in a program, so the Student entity participates totally in the many-to-one Enrolls In relationship with Program. A program can exist without having any enrolled students, so it participates partially in this relationship.
- A Course has meaning only in the context of a Program, so it's a weak entity, with course_id as a weak key. This means that a Course is uniquely identified using its course_id and the program_id of its owning program.
- As a weak entity, Course participates totally in the many-to-one identifying relationship with its owning Program. This relationship has Year and Semester attributes that identify its sequence position.
- Student and Course are related through the many-to-many Attempts relationships; a course can exist without a student, and a student can be enrolled without attempting any courses, so the participation is not total.
- When a student attempts a course, there are attributes to capture the Year and Semester, and the Mark and Grade.

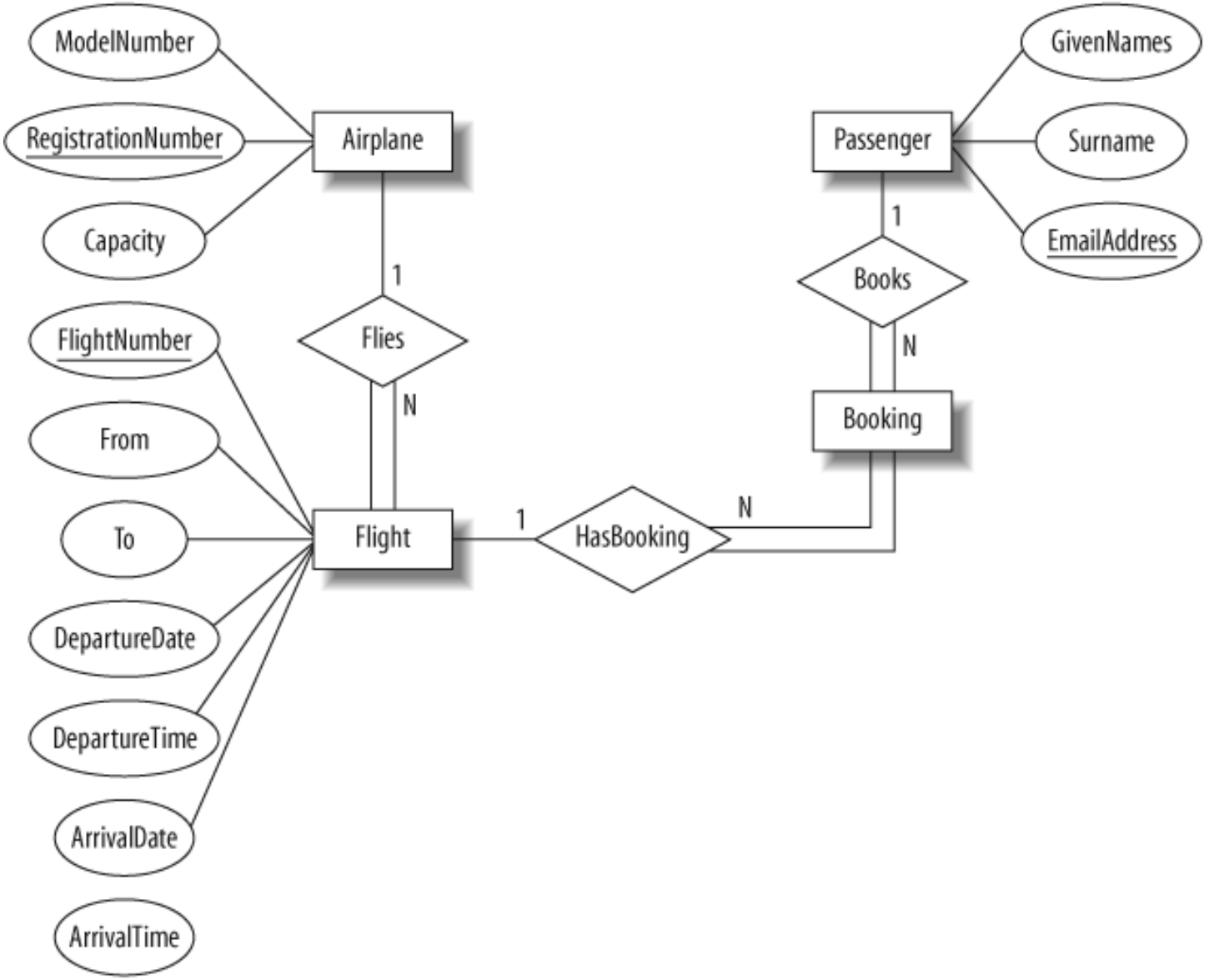
Flight Database

The Flight Database

- The flight database stores details about an airline's fleet, flights, and seat bookings. Again, it's a hugely simplified version of what a real airline would use, but the principles are the same.
- Consider the following requirements list:
- The airline has one or more airplanes.
- An airplane has a model number, a unique registration number, and the capacity to take one or more passengers.
- An airplane flight has a unique flight number, a departure airport, a destination airport, a departure date and time, and an arrival date and time.
- Each flight is carried out by a single airplane.
- A passenger has given names, a surname, and a unique email address.
- A passenger can book a seat on a flight.

The ER diagram derived from our requirements is shown in Figure:

Flight Database



Flight Database

- An Airplane is uniquely identified by its RegistrationNumber, so we use this as the primary key.
- A Flight is uniquely identified by its FlightNumber, so we use the flight number as the primary key. The departure and destination airports are captured in the From and To attributes, and we have separate attributes for the departure and arrival date and time.
- Because no two passengers will share an email address, we can use the EmailAddress as the primary key for the Passenger entity.
- An airplane can be involved in any number of flights, while each flight uses exactly one airplane, so the Flies relationship between the Airplane and Flight relationships has cardinality 1:N; because a flight cannot exist without an airplane, the Flight entity participates totally in this relationship.
- A passenger can book any number of flights, while a flight can be booked by any number of passengers.
- We could specify an M:N Books relationship between the Passenger and Flight relationship, but considering the issue more carefully shows that there is a hidden entity here: the booking itself.
- We capture this by creating the intermediate entity Booking and 1:N relationships between it and the Passenger and Flight entities. Identifying such entities allows us to get a better picture of the requirements.

Relational Data Model

Relational Data Model uses a collection of tables to represent both data and the relationship among those data.

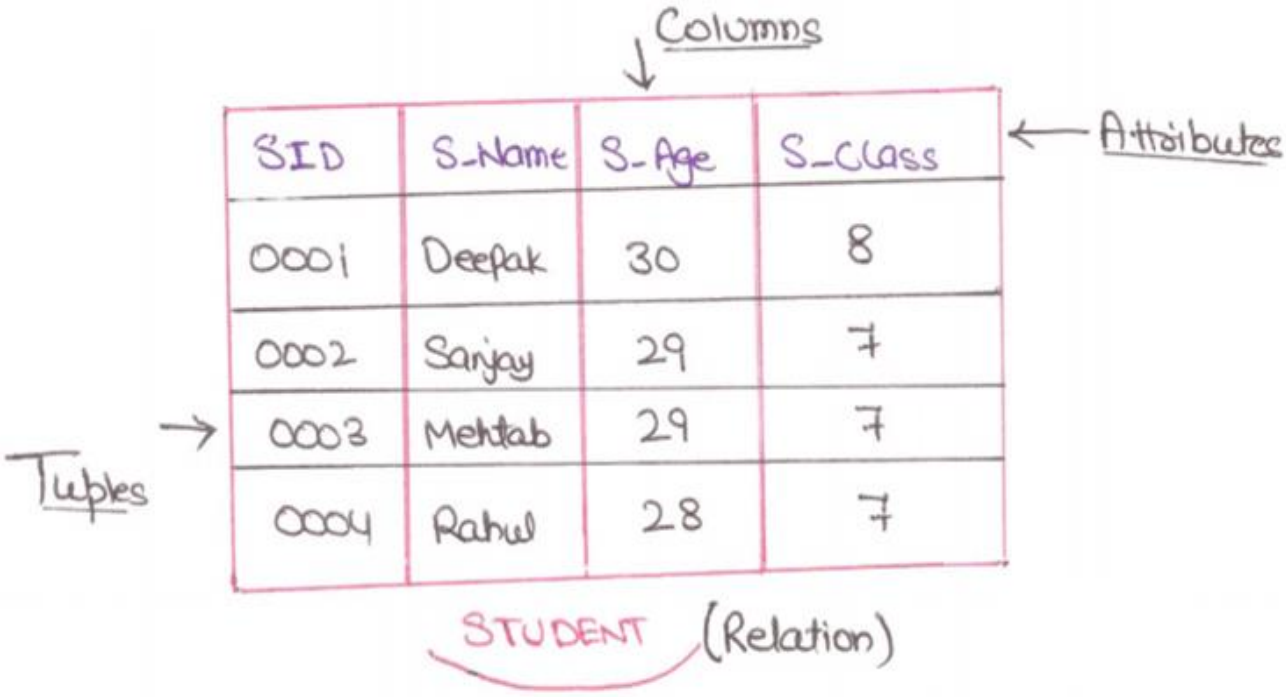
1) Each table has multiple columns and each column as **unique name.**
2) The Data is arranged in a Relation which is visually represented in a **two dimensional table.**
3) The data is inserted into the table in the form of Tuples (Rows).A Tuple is formed by one or more than one attributes. **Tuple in this example is a row (complete row).**
4) Attributes are used as basic building block in the formation of various expression that are used to drive a meaningful information.
5) There can be number of **tuples** in the table (relation), but all the tuples contains fixed and some attributes with varying values.
6) A **Relation** is represented by a **table.**

Relational Data Model

Tuple is represented by row.
An **attribute** is represented by a **column of the table.** Attribute name is the name of the column

Example S_ID, S_Name, S_Age.

Attribute values contains the values for the column in the row.



Codd's Rules

Codd's Rules, formulated by **Dr. Edgar F. Codd**, define the criteria that a database management system (DBMS) must satisfy to be considered a **relational database management system (RDBMS)**. These **12 rules (actually 13, as Rule 0 is included)** ensure data integrity, consistency, and efficient handling of relational data.

Codd's 12 Rules for RDBMS

Rule 0: Foundation Rule

- The system must manage databases entirely through its **relational capabilities**.

Rule 1: Information Rule

- All information in the database must be **stored in tables** as values in rows and columns.

Rule 2: Guaranteed Access Rule

- Every data item should be uniquely accessible using a **combination of table name, primary key, and column name**.

Codd's Rules

Rule 3: Systematic Treatment of NULL Values

- NULL values should be **distinctly stored** and **treated differently** from other values (not confused with zero or empty string).

Rule 4: Dynamic Online Catalog Based on the Relational Model

- The database metadata (schema, tables, etc.) must be stored **in relational tables** and accessible using SQL queries.

Rule 5: Comprehensive Data Sublanguage Rule

- The system must support at least one **relational language** (like SQL) for data definition, manipulation, and transaction control.

Rule 6: View Updating Rule

- The database must allow updates to **views (virtual tables)** as if they were real tables, whenever possible.

Rule 7: High-Level Insert, Update, and Delete

- The system must support **set-based operations**, allowing manipulation of multiple rows in a single operation.

Codd's Rules

Rule 8: Physical Data Independence

- Changes in **physical storage** should not affect how data is accessed by users or applications.

Rule 9: Logical Data Independence

- Changes in **table structure (schema modifications)** should not require application modifications.

Rule 10: Integrity Independence

- Integrity constraints (like primary keys, foreign keys, and domain constraints) should be **stored in the database** and not in application code.

Rule 11: Distribution Independence

- The system should allow **distributed databases** without requiring changes to applications.

Rule 12: Nonsubversion Rule

- If the system provides a **low-level access method**, it must not bypass the relational security and integrity constraints.

Relational Data Model

Constraint - Set of rules and limitations:

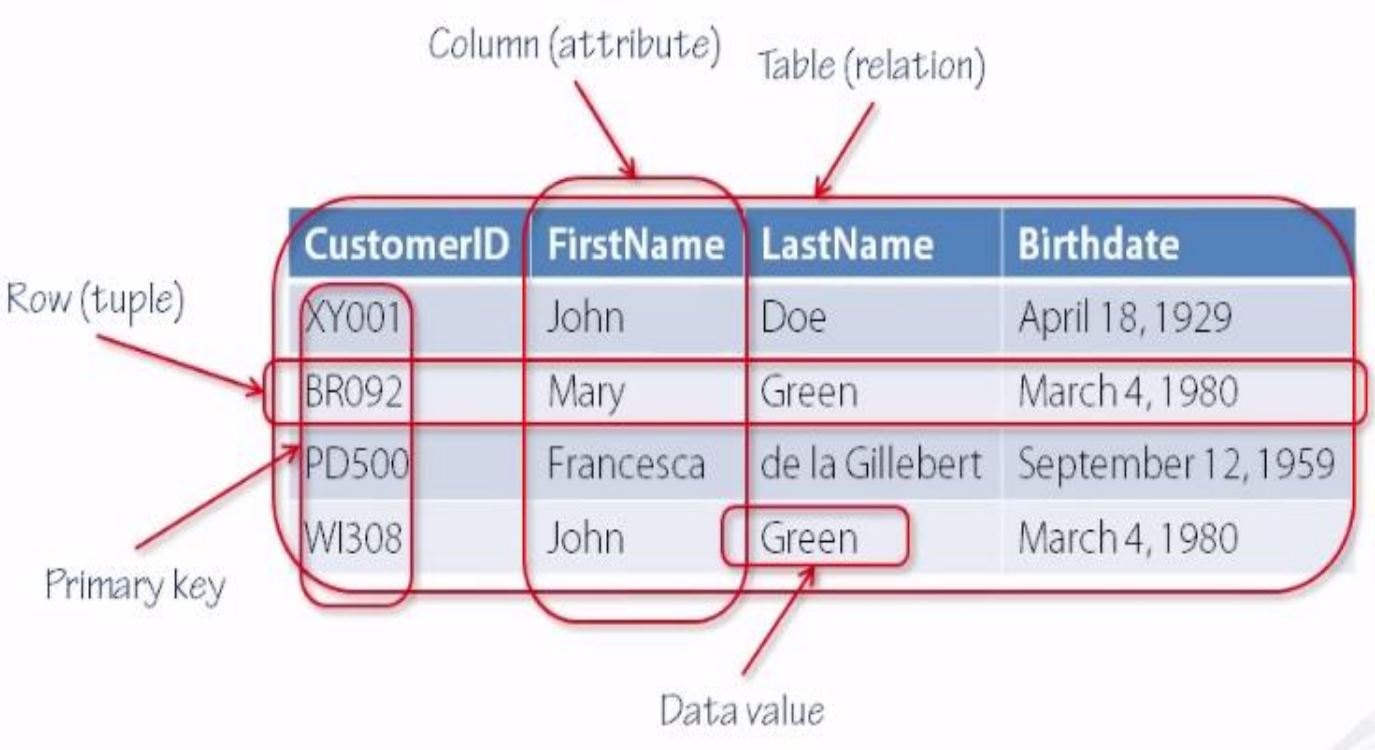
Constraints are applied to tables and forms the logical schema.

- To select a particular row/tuple from table/relation we use attribute/columns name with the help of unique value field of an attributes.
- This field which are unique from other fields are used as indexes which helps in searching fast.
- All the relational algebra operations, like Select, Intersection, Product, union, join, Division, Merge can also be performed on the relation data model.
- Operations on RDM (Relational Data Model) are facilitated with the help of different conditional expression, various key attributes, and predefined constraints etc.
- Data Integrity is maintained by process like Normalization.
- Description of data in terms of this model is called a schema.

Schema for relation specifies its name , name of each field.

Student(S_id: Integer, S_Name: String, S_Login : String etc)

Relational Data Model



CustomerID	FirstName	LastName	Birthdate
XY001	John	Doe	April 18, 1929
BR092	Mary	Green	March 4, 1980
PD500	Francesca	de la Gillebert	September 12, 1959
WI308	John	Green	March 4, 1980

Relational Data Model

Relational Model Concepts

- **Attribute:** Each column in a Table. Attributes are the properties which define a relation. e.g., Student, Rollno, NAME etc.
- **Tables** – In the Relational model the, relations are saved in the table format. It is stored along with its entities. A table has two properties rows and columns. Rows represent records and columns represent attributes.
- **Tuple** – It is nothing but a single row of a table, which contains a single record.
- **Relation Schema:** A relation schema represents the name of the relation with its attributes.
- **Degree:** The total number of attributes which in the relation is called the degree of the relation.
- **Cardinality:** Total number of rows present in the Table.
- **Column:** The column represents the set of values for a specific attribute.
- **Relation instance** – Relation instance is a finite set of tuples in the RDBMS system. Relation instances never have duplicate tuples.
- **Relation key** - Every row has one, two or multiple attributes, which is called relation key.
- **Attribute domain** – Every attribute has some pre-defined value and scope which is known as attribute domain.

Relational Data Model

Example: STUDENT Relation

- In the given table, NAME, ROLL_NO, PHONE_NO, ADDRESS, and AGE are the attributes.
- The instance of schema STUDENT has 5 tuples.
- t3 = <Laxman, 33289, 8583287182, Gurugram, 20>.

NAME	ROLL_NO	PHONE_NO	ADDRESS	AGE
Ram	14795	7305758992	Noida	24
Shyam	12839	9026288936	Delhi	35
Laxman	33289	8583287182	Gurugram	20
Mahesh	27857	7086819134	Ghaziabad	27
Ganesh	17282	9028 913988	Delhi	40

Relational Data Model

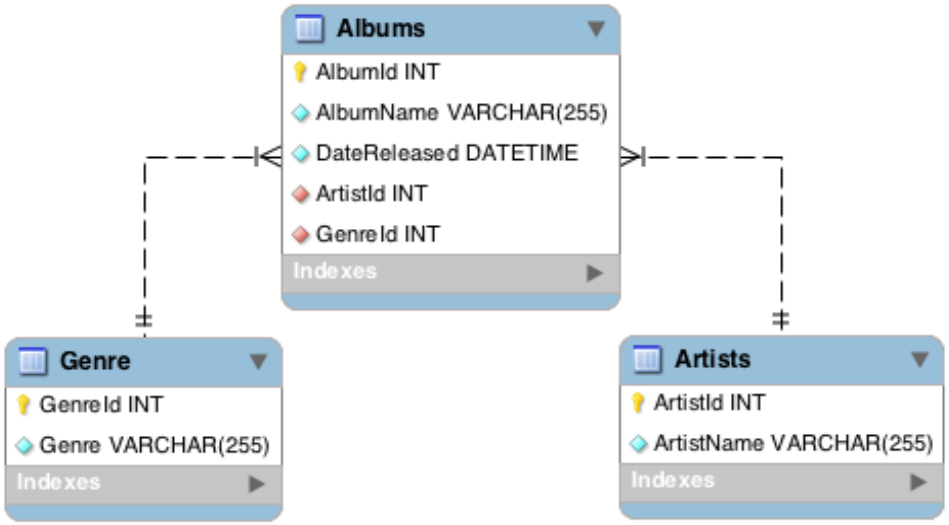
Properties of Relations

- Name of the relation is distinct from all other relations.
- Each relation cell contains exactly one atomic (single) value
- Each attribute contains a distinct name
- Attribute domain has no significance
- tuple has no duplicate value
- Order of tuple can have a different sequence

Schema, Subschema and Instance

Schema

- Schema is a the logical description of the database.
- The overall design of the database is called Database Schema.
- In database terms, a *schema* (pronounced “skee-muh” or “skee-mah”) is the organization and structure of a database. Both *schemas* and *schemata* can be used as plural forms.
- A schema contains schema objects, which could be tables, columns, data types, views, stored procedures, relationships, primary keys, foreign keys, etc.
- A database schema can be represented in a visual diagram, which shows the database objects and their relationship with each other.



Schema, Subschema and Instance

Schema and Database are the same things or different?

Part of the reason for the confusion is that database systems tend to approach schemas in their own way.

- **MySQL Documentation**
  A schema is synonymous with a database. Therefore, a schema and a database are the **same thing**.

- **Oracle Database Documentation**
  Certain objects can be stored inside a database but not inside a schema. Therefore, a schema and a database are **two different things**.

- **SQL Server Technical Article**
  A schema is a separate entity inside the database. So, they are **two different things**.

Schema, Subschema and Instance

Schema

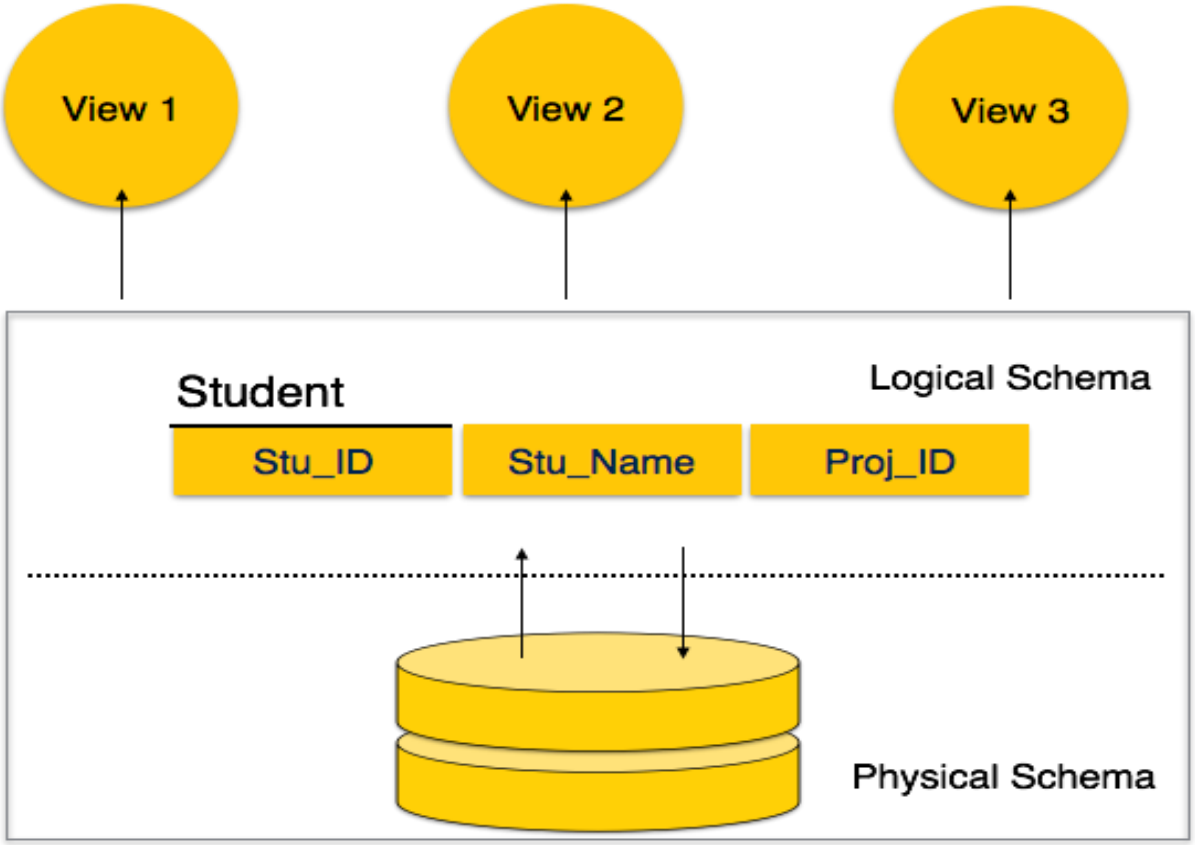
Schema is a the logical description of the database.
The overall design of the database is called Database Schema.
Database system has several schemas and partitioned according to the levels of abstractions. There are three level of schemas in database system.

1. **Internal Schema or Physical Schema**
   It describes that how data are actually stored in the blocks of storage devices such as hard disk.

2. **Logical Schema or Conceptual Schema**
   It describes the structure of the database to the database designer. Programmers and database administrators work at this level, at this level data can be described as certain types of data records gets stored in data structures

3. **External Schema or Subschema**
   It is a subset of main schema having the same properties as schema. It describes the different parts, sets, records and data names of the database to the end users. It allows users to view only the parts of the main database.

Schema, Subschema and Instance



Schema, Subschema and Instance

Database Instance

The data stored in database at a particular moment of time is called instance of database. Database schema defines the variable declarations in tables that belong to a particular database; the value of these variables at a moment of time is called the instance of that database.

A database schema is variable declarations in a program. Variable has particular value at a given instant. Then, the value of variable at particular instant is called database instance.

Database schema (names of columns + the types associated with them)

Name	DOB	Address	Job	Scale
String	*Date*	*String*	*String*	*Int*

A database instance

Name	DOB	Address	Job	Scale
A. Johnson	2/04/1960	London	Programmer	12
B. Holiday	3/10/1947	Leeds	Analyst	14
C. Clark	12/08/1971	York	Programmer	10

Relational Integrity Constraints

Relational Integrity constraints in DBMS are referred to conditions which must be present for a valid relation. These Relational constraints in DBMS are derived from the rules in the mini-world that the database represents.

Integrity = (Correctness + Consistency)

There are many types of Integrity Constraints in DBMS. Constraints on the Relational database management system is mostly divided into three main categories are:

1. Domain Constraints
2. Key Constraints
3. Referential Integrity Constraints

Relational Integrity Constraints

Domain Constraints

Domain constraints can be violated if an attribute value is not appearing in the corresponding domain or it is not of the appropriate data type.

Domain constraints specify that within each tuple, and the value of each attribute must be unique. This is specified as data types which include standard data types integers, real numbers, characters, Booleans, variable length strings, etc.

Example:

Create DOMAIN CustomerName CHECK (value not NULL)

The example shown demonstrates creating a domain constraint such that CustomerName is not NULL

If a constrains AGE>0 is applied on STUDENT relation, inserting negative value of AGE will result in failure.

Relational Integrity Constraints

Constraint Type	Code Snippet	Explanation
NOT NULL	CREATE TABLE Employees (name VARCHAR(50) NOT NULL);	Ensures that the name column cannot have NULL values, enforcing that every employee must have a name.
CHECK	CREATE TABLE Employees (age INT CHECK (age >= 18 AND age <= 65));	Ensures that the age column values must be between 18 and 65, enforcing that employees fall within this age range.
DEFAULT	CREATE TABLE Employees (hire_date DATE DEFAULT CURRENT_DATE);	Ensures that the hire_date column will default to the current date if no value is provided during insertion.
UNIQUE	CREATE TABLE Employees (email VARCHAR(100) UNIQUE);	Ensures that the email column values must be unique across all rows, preventing duplicate

Relational Integrity Constraints

```
CREATE TABLE Employees
(    id INT PRIMARY KEY,
     name VARCHAR(50) NOT NULL,
     age INT CHECK (age >= 18 AND age <= 65),
     email VARCHAR(100) UNIQUE,
     hire_date DATE DEFAULT CURRENT_DATE
);
```

This ensures domain integrity in our SQL Server by validating that all the values of the table are the ones we would expect:

- Every employee has a name.
- Employees are of a working age.
- Each employee has their own unique email.
- All hire dates are valid, and have a default just in case.

Relational Integrity Constraints

Key Constraints

There must be at least one minimal subset of attributes in the relation, which can identify a tuple uniquely. This minimal subset of attributes is called **key** for that relation. If there are more than one such minimal subsets, these are called ***candidate keys***.

Every relation in the database should have at least one set of attributes which defines a tuple uniquely. Those set of attributes is called key. e.g.; ROLL_NO in STUDENT is a key. No two students can have same roll number.

Key constraints force that –

- In a relation with a key attribute, no two tuples can have identical values for key attributes.
- A key attribute can not have NULL values.
- Key constraints are also referred to as Entity Constraints.

Relational Integrity Constraints

Key plays an important role in relational database; it is used for identifying unique rows from table. It also establishes relationship among tables.

Types of keys in DBMS

1. **Primary Key** – A primary is a column or set of columns in a table that uniquely identifies tuples (rows) in that table.
2. **Super Key** – A super key is a set of one of more columns (attributes) to uniquely identify rows in a table.
3. **Candidate Key** – A super key with no redundant attribute is known as candidate key
4. **Alternate Key** – Out of all candidate keys, only one gets selected as primary key, remaining keys are known as alternate or secondary keys.
5. **Composite Key** – A key that consists of more than one attribute to uniquely identify rows (also known as records & tuples) in a table is called composite key.
6. **Foreign Key** – Foreign keys are the columns of a table that points to the primary key of another table. They act as a cross-reference between tables.

Relational Integrity Constraints

Primary Key

A **primary key** is a minimal set of attributes (columns) in a table that uniquely identifies tuples (rows) in that table.

Stu_Id	Stu_Name	Stu_Age
101	Steve	23
102	John	24
103	Robert	28
104	Steve	29
105	Carl	29

Primary Key Example in DBMS

Lets take an example to understand the concept of primary key. In the following table, there are three attributes: Stu_ID, Stu_Name & Stu_Age. Out of these three attributes, one attribute or a set of more than one attributes can be a primary key.

1. Attribute Stu_Name alone cannot be a primary key as more than one students can have same name.
2. Attribute Stu_Age alone cannot be a primary key as more than one students can have same age.
3. Attribute Stu_Id alone is a primary key as each student has a unique id that can identify the student record in the table.

Note: In some cases an attribute alone cannot uniquely identify a record in a table, in that case we try to find a set of attributes that can uniquely identify a row in table. We will see the example of it after this example.

Relational Integrity Constraints

Definition of Super Key in DBMS: A super key is a set of one or more attributes (columns), which can uniquely identify a row in a table. Often **DBMS beginners** get confused between super key and **candidate key**, so we will also discuss candidate key and its relation with super key in this article.

Candidate Key Vs Super Key

Candidate keys are selected from the set of super keys, the only thing we take care while selecting candidate key is: It should not have any redundant attribute. That's the reason they are also termed as minimal super key.

Table: Employee

Super Keys: The above table has following super keys. All of the following sets of super key are able to uniquely identify a row of the employee table.

{Emp_SSN}
{Emp_Number}
{Emp_SSN, Emp_Number}
{Emp_SSN, Emp_Name}
{Emp_SSN, Emp_Number, Emp_Name}
{Emp_Number, Emp_Name}

Emp_SSN	Emp_Number	Emp_Name
123456789	226	Steve
999999321	227	Ajeet
888997212	228	Chaitanya
777778888	229	Robert

Relational Integrity Constraints

Candidate Keys: As I mentioned in the beginning, a candidate key is a minimal super key with no redundant attributes. The following two set of super keys are chosen from the above sets as there are no redundant attributes in these sets.

{Emp_SSN}

{Emp_Number}

Only these two sets are candidate keys as all other sets are having redundant attributes that are not necessary for unique identification.

Primary key

A Primary key is selected from a set of candidate keys. This is done by database admin or database designer.

We can say that either {Emp_SSN} or {Emp_Number} can be chosen as a primary key for the table Employee.

Alternate Key

As we have seen in the **candidate key** guide that a table can have multiple candidate keys. Among these candidate keys, only one key gets selected as **primary key**, the remaining keys are known as **alternative or secondary keys**.

Since we have selected Emp_SSN as primary key, the remaining key Emp_Number would be called alternative or secondary key.

Relational Integrity Constraints

Definition of Composite key: A key that has more than one attributes is known as composite key. It is also known as compound key.

Note: Any key such as **super key**, **primary key**, **candidate key** etc. can be called composite key if it has more than one attributes.

Composite key Example

All of the following sets of super key are able to uniquely identify a row of the employee table.

{Emp_SSN}
{Emp_Number}
{Emp_SSN, Emp_Number}
{Emp_SSN, Emp_Name}
{Emp_SSN, Emp_Number, Emp_Name}
{Emp_Number, Emp_Name}

Out of the above given keys, the following are the composite keys.

{Emp_SSN, Emp_Number}
{Emp_SSN, Emp_Name}
{Emp_SSN, Emp_Number, Emp_Name}
{Emp_Number, Emp_Name}

As all the keys are made up of more than one attributes.

Referential Integrity Constraints

Definition: Foreign keys are the columns of a table that points to the **primary key** of another table. They act as a cross-reference between tables.

Referential integrity Constraints

Referential integrity constraints work on the concept of Foreign Keys. A foreign key is a key attribute of a relation that can be referred in other relation.

Referential integrity constraint states that if a relation refers to a key attribute of a different or same relation, then that key element must exist.

Referential Integrity Constraints

When one attribute of a relation can only take values from other attribute of same relation or any other relation, it is called referential integrity. Let us suppose we have 2 relations

Student

ROLL_NO	NAME	ADDRESS	PHONE	AGE	BRANCH_CODE
1	RAM	DELHI	9455123451	18	CSE
2	RAMESH	GURGAON	9652431543	18	CSE
3	SUJIT	ROHTAK	9156253131	20	ECE
4	SURESH	DELHI	9136263161	18	IT

Branch

BRANCH_CODE	BRANCH_NAME
CSE	COMPUTER SCIENCE & ENGINEERING
IT	INFORMATION TECHNOLOGY
ECE	ELECTRONICS AND COMMUNICATION ENGINEERING
CE	CIVIL ENGINEERING

BRANCH_CODE of STUDENT can only take the values which are present in BRANCH_CODE of BRANCH which is called referential integrity constraint. The relation which is referencing to other relation is called REFERENCING RELATION (STUDENT in this case) and the relation to which other relations refer is called REFERENCED RELATION (BRANCH in this case).

Referential Integrity Constraints

```
Cerate table branch
( branch_code varchar(8) PRIMARY KEY,
  branch_name varchar(50));

Create table student
( rollno number(10) PRIMARY KEY,
  name varchar(50) NOT NULL,
  address varchar(50) NOT NULL,
  contact number(10) NOT NULL,
  age number(2),
  branch_code carchar(8) REFERENCES branch(branch_code));
```

OR

```
 Create table student
( rollno number(10) PRIMARY KEY,
  name varchar(50) NOT NULL,
  address varchar(50) NOT NULL,
  contact number(10) NOT NULL,
  age number(2),
  branch_code carchar(8)
  CONSTRAINT FK_Dept
  FOREIGN KEY (Dept_ID)
  REFERENCES Department(Dept_ID));
```

Referential Integrity Constraints

An anomaly is an irregularity, or something which deviates from the expected or normal state. When designing databases, we identify three types of anomalies: Insert, Update and Delete.

Insertion Anomaly in Referencing Relation:

We can't insert a row in REFERENCING RELATION if referencing attribute's value is not present in referenced attribute value. e.g.; Insertion of a student with BRANCH_CODE 'ME' in STUDENT relation will result in error because 'ME' is not present in BRANCH_CODE of BRANCH.

Deletion/Updation Anomaly in Referenced Relation:

We can't delete or update a row from REFERENCED RELATION if value of REFRENCED ATTRIBUTE is used in value of REFERENCING ATTRIBUTE. e. g.; if we try to delete tuple from BRANCH having BRANCH_CODE 'CS', it will result in error because 'CS' is referenced by BRANCH_CODE of STUDENT, but if we try to delete the row from BRANCH with BRANCH_CODE CV, it will be deleted as the value is not been used by referencing relation. It can be handled by following method:

Referential Integrity Constraints

ON DELETE CASCADE: It will delete the tuples from REFERENCING RELATION if value used by REFERENCING ATTRIBUTE is deleted from REFERENCED RELATION. e.g.;, if we delete a row from BRANCH with BRANCH_CODE 'CS', the rows in STUDENT relation with BRANCH_CODE CS (ROLL_NO 1 and 2 in this case) will be deleted.

```
Create table student
( rollno number(10) PRIMARY KEY,
  name varchar(50) NOT NULL,
  address varchar(50) NOT NULL,
  contact number(10) NOT NULL,
  age number(2),
  branch_code carchar(8)
  CONSTRAINT FK_Dept FOREIGN KEY (Dept_ID)
  REFERENCES Department(Dept_ID) ON DELETE CASCADE/
  ON DELETE SET NULL);
```

ON UPDATE CASCADE: It will update the REFERENCING ATTRIBUTE in REFERENCING RELATION if attribute value used by REFERENCING ATTRIBUTE is updated in REFERENCED RELATION. e.g.;, if we update a row from BRANCH with BRANCH_CODE 'CS' to 'CSE', the rows in STUDENT relation with BRANCH_CODE CS (ROLL_NO 1 and 2 in this case) will be updated with BRANCH_CODE 'CSE'.

Referential Integrity Constraints

Oracle does not support ON UPDATE CASCADE because primary keys should remain stable, and updates can be handled using triggers if required.

ON UPDATE CASCADE: It will update the REFERENCING ATTRIBUTE in REFERENCING RELATION if attribute value used by REFERENCING ATTRIBUTE is updated in REFERENCED RELATION. e.g.;, if we update a row from BRANCH with BRANCH_CODE 'CS' to 'CSE', the rows in STUDENT relation with BRANCH_CODE CS (ROLL_NO 1 and 2 in this case) will be updated with BRANCH_CODE 'CSE'.

Converting ER Diagram into Tables

- ER diagram is converted into the tables in relational model.
- This is because relational models can be easily implemented by RDBMS like MySQL , Oracle etc.

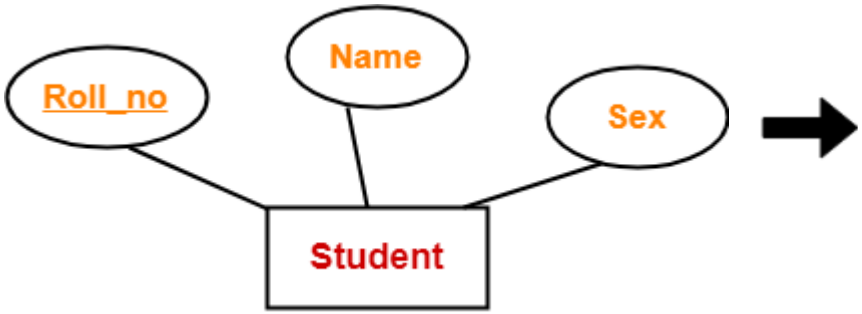
Following rules are used for converting an ER diagram into the tables-

Rule-01: For Strong Entity Set With Only Simple Attributes-

A strong entity set with only simple attributes will require only one table in relational model.

- Attributes of the table will be the attribute of the entity set.
- The primary key of the table will be the key attribute of the entity set.

Example:



Roll_no	Name	Sex

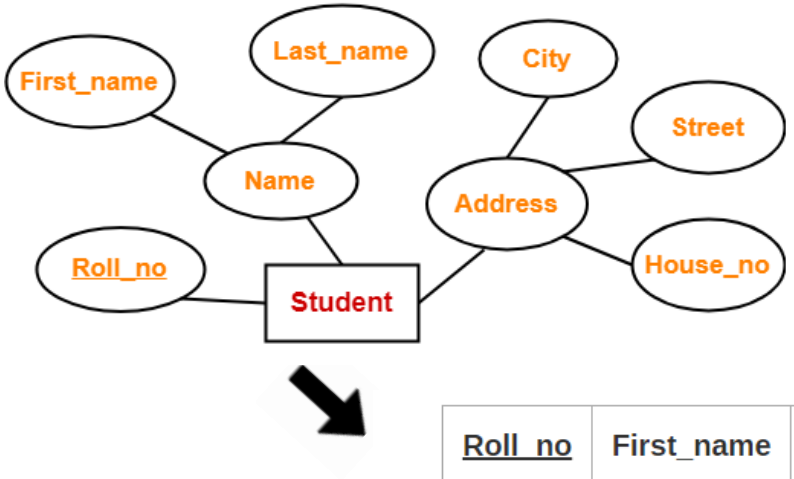
Schema : Student (Roll_no , Name , Sex)

Converting ER Diagram into Tables

Rule-02: For Strong Entity Set With Composite Attributes-

- A strong entity set with any number of composite attributes will require only one table in relational model.
- While conversion, simple attributes of the composite attributes are taken into account and not the composite attribute itself.

Example-



Roll_no	First_name	Last_name	House_no	Street	City

Schema : Student (Roll_no , First_name , Last_name , House_no , Street , City)

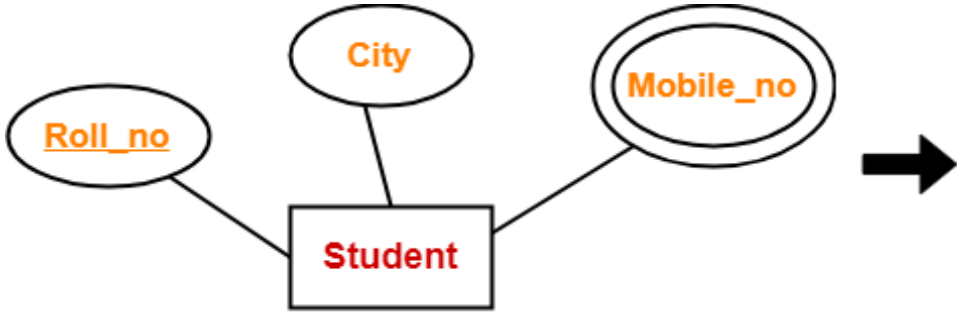
Converting ER Diagram into Tables

Rule-03: For Strong Entity Set With Multi Valued Attributes-

A strong entity set with any number of multi valued attributes will require two tables in relational model.

- One table will contain all the simple attributes with the primary key.
- Other table will contain the primary key and all the multi valued attributes.

Example-



Roll_no	City

Roll_no	Mobile_no

Converting ER Diagram into Tables

Rule-04: Translating Relationship Set into a Table-

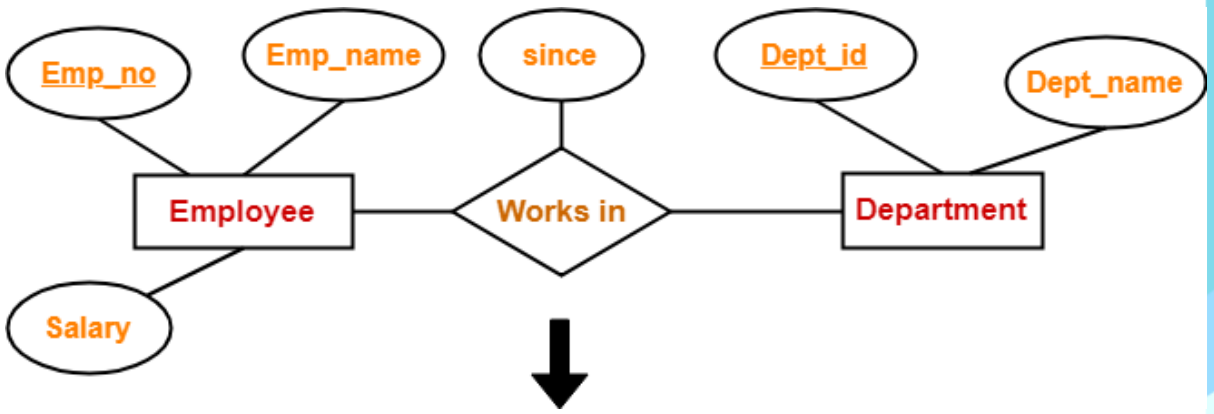
A relationship set will require one table in the relational model.

Attributes of the table are-

- Primary key attributes of the participating entity sets
- Its own descriptive attributes if any.

Set of non-descriptive attributes will be the primary key.

Example:



NOTE-

If we consider the overall ER diagram, three tables will be required in relational model.

1. **One table for the entity set "Employee"**
2. **One table for the entity set "Department"**
3. **One table for the relationship set "Works in"**

Emp_no	Dept_id	since

Schema : Works in (Emp_no , Dept_id , since)

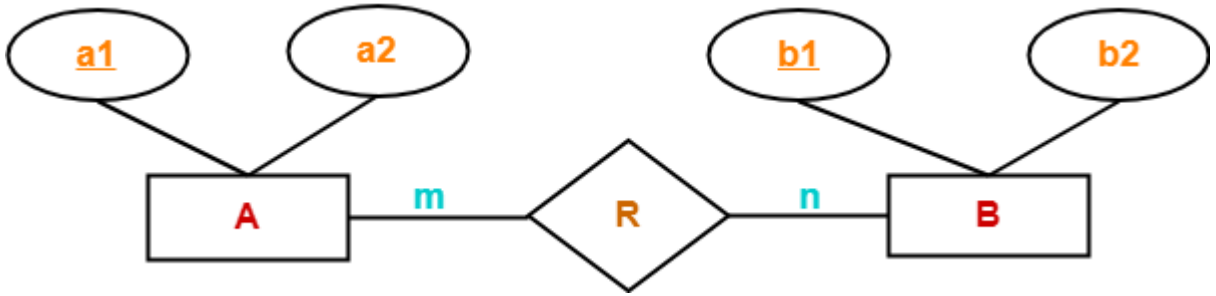
Converting ER Diagram into Tables

Rule-05: For Binary Relationships With Cardinality Ratios-

The following four cases are possible-

Case-01: Binary relationship with cardinality ratio m:n
Case-02: Binary relationship with cardinality ratio 1:n
Case-03: Binary relationship with cardinality ratio m:1
Case-04: Binary relationship with cardinality ratio 1:1

Case-01: For Binary Relationship With Cardinality Ratio m:n

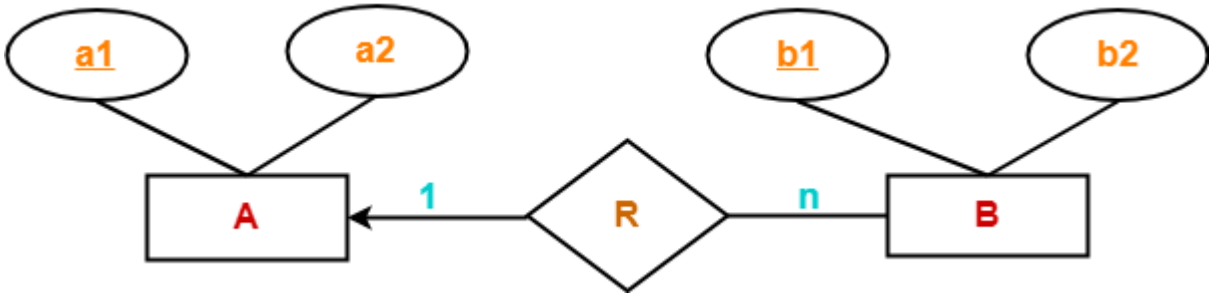


Here, three tables will be required-

1. A (a1 , a2)
2. R (a1 , b1)
3. B (b1 , b2)

Converting ER Diagram into Tables

Case-02: For Binary Relationship With Cardinality Ratio 1:n

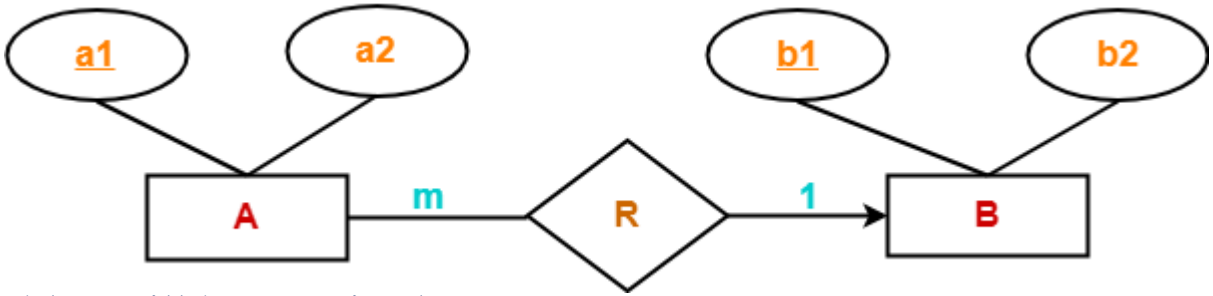


Here, two tables will be required-

1. A (a1 , a2)
2. BR (a1 , b1 , b2)

NOTE- Here, combined table will be drawn for the entity set B and relationship set R.

Case-03: For Binary Relationship With Cardinality Ratio m:1



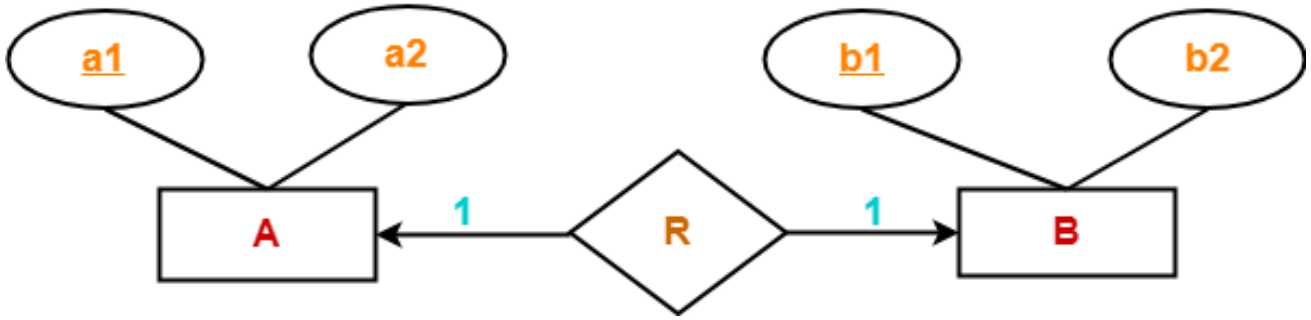
Here, two tables will be required-

1. AR (a1 , a2 , b1)
2. B (b1 , b2)

NOTE- Here, combined table will be drawn for the entity set A and relationship set R.

Converting ER Diagram into Tables

Case-04: For Binary Relationship With Cardinality Ratio 1:1



Here, two tables will be required. Either combine 'R' with 'A' or 'B'

Way-01:

1. AR (a1 , a2 , b1)
2. B (b1 , b2)

Way-02:

1. A (a1 , a2)
2. BR (a1 , b1 , b2)

Converting ER Diagram into Tables

Thumb Rules to Remember

While determining the minimum number of tables required for binary relationships with given cardinality ratios, following thumb rules must be kept in mind-

- For binary relationship with cardinality ration m : n , separate and individual tables will be drawn for each entity set and relationship.

- For binary relationship with cardinality ratio either m : 1 or 1 : n , always remember “many side will consume the relationship” i.e. a combined table will be drawn for many side entity set and relationship set.

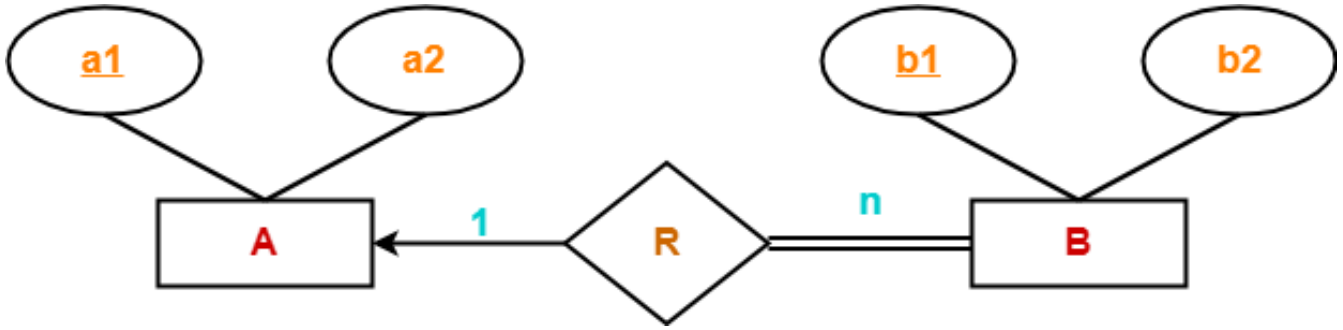
- For binary relationship with cardinality ratio 1 : 1 , two tables will be required. You can combine the relationship set with any one of the entity sets.

Converting ER Diagram into Tables

Rule-06: For Binary Relationship With Both Cardinality Constraints and Participation Constraints-

- Cardinality constraints will be implemented as discussed in Rule-05.
- Because of the total participation constraint, foreign key acquires **NOT NULL** constraint i.e. now foreign key can not be null.

Case-01: For Binary Relationship With Cardinality Constraint and Total Participation Constraint From One Side-



Because cardinality ratio = 1 : n , so we will combine the entity set B and relationship set R.

Then, two tables will be required-

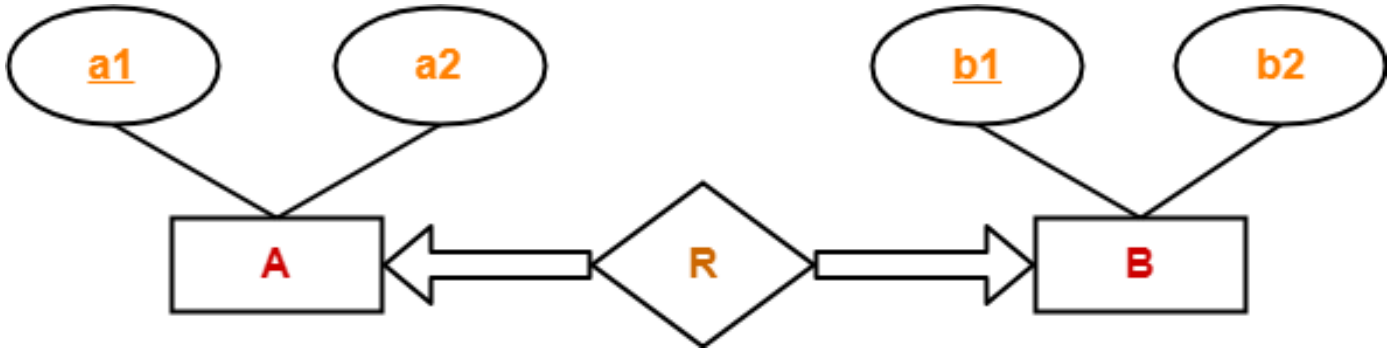
1. A (a1 , a2)
2. BR (a1 , b1 , b2)

Because of total participation, foreign key a1 has acquired NOT NULL constraint, so it can't be null now.

Converting ER Diagram into Tables

Case-02: For Binary Relationship With Cardinality Constraint and Total Participation Constraint From Both Sides-

If there is a key constraint from both the sides of an entity set with total participation, then that binary relationship is represented using only single table.



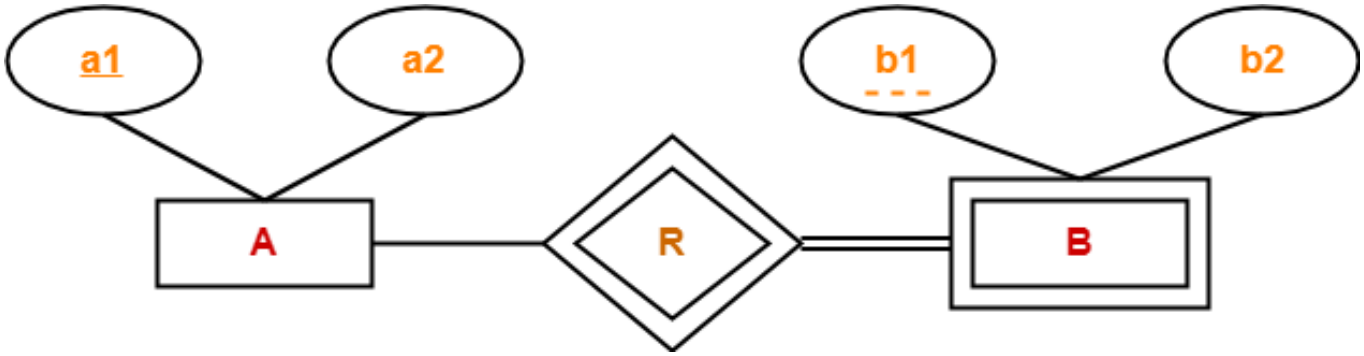
Here, Only one table is required.

1. ARB (a1 , a2 , b1 , b2)

Converting ER Diagram into Tables

Rule-07: For Binary Relationship With Weak Entity Set-

Weak entity set always appears in association with identifying relationship with total participation constraint.



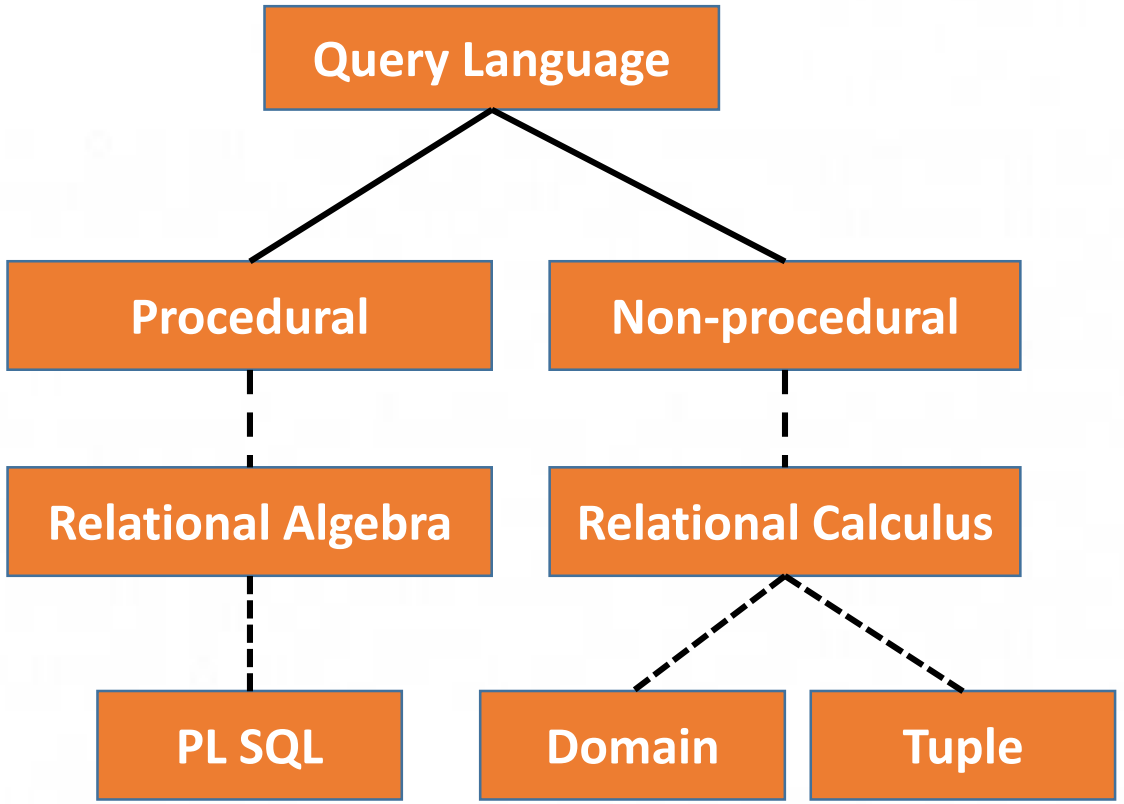
Here, two tables will be required-

1. A (a1 , a2)
2. BR (a1 , b1 , b2)

DATA QUERY LANGUAGES

- Query languages, often known as DQLs or Data Query Languages, are computer languages that are used to make various queries in information systems and databases.
- A query language is a language in which user requests information from the database.

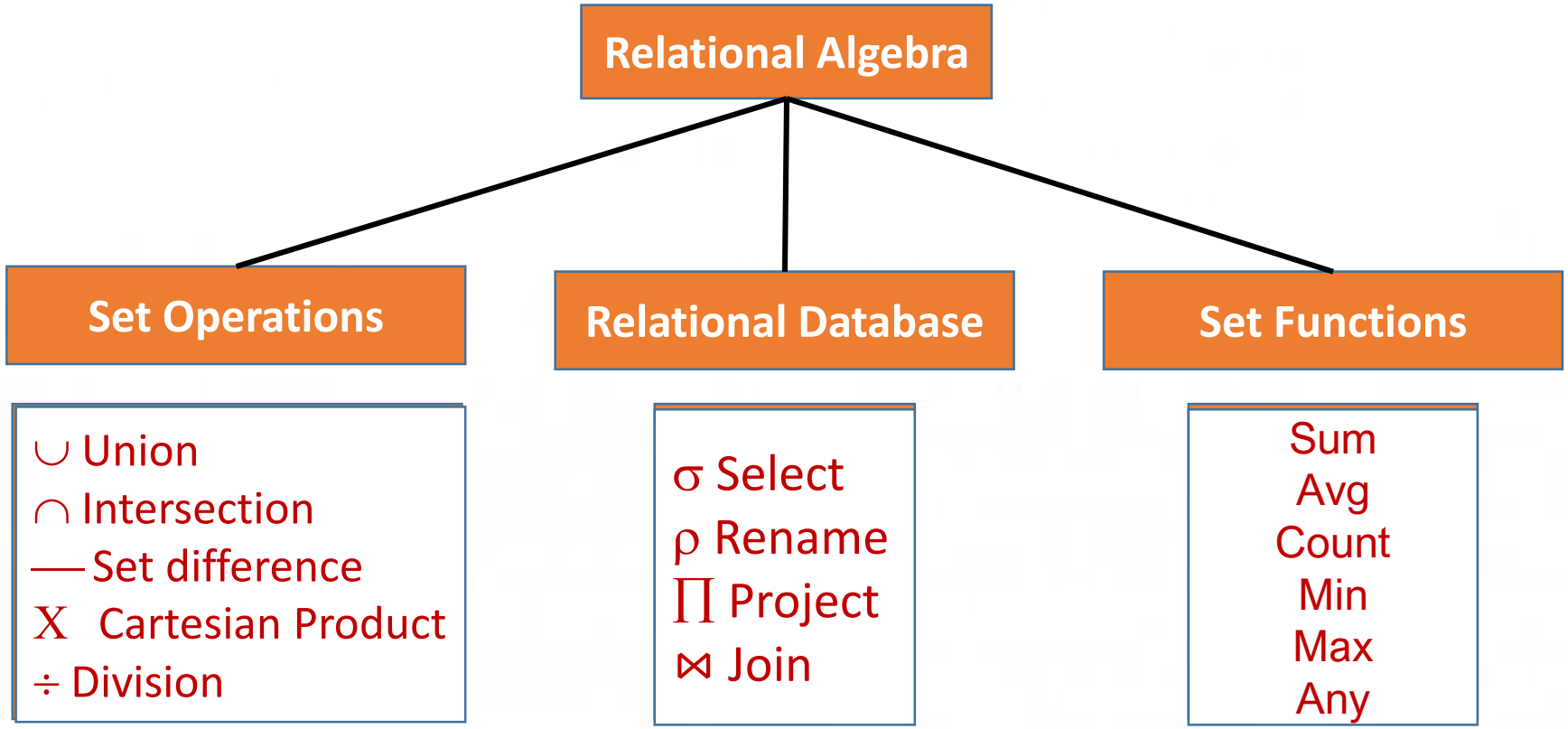
- **Procedural Query Language:** User instructs the system to perform a sequence of operations on the database to compute the desired result.
  For Example: Relational algebra
  Structure Query language (SQL) is based on relational algebra.
- **Non-procedural Query Language:** Information is retrieved from the database without specifying the sequence of operation to be performed. Users only specify what information is to be retrieved.
  For Example: Relational Calculus
  Query by Example (QBE) is based on Relational calculus



RELATIONAL ALGEBRA

- Relational Algebra came in 1970 and was given by Edgar F. Codd (Father of DBMS). It is also known as Procedural Query Language(PQL) as in PQL, a programmer/user has to mention two things, **"What to Do"** and **"How to Do"**.
- **Relational algebra:** It is a collection of operations to manipulate relations.
- Relational Algebra is a procedural query language. It consists of a set of operations that take one or two relations a input and produce a new relation as their result.
- It specifies the operations to be performed on existing relations to derive the result relations.
- Relational Algebra are usually divided into two groups.
  - Mathematical Set Operations e.g. Union, Intersection, Set Difference, Cartesian Product.
  - Relational Database Operations e.g. Select, Project, Rename, Join, Assignment.

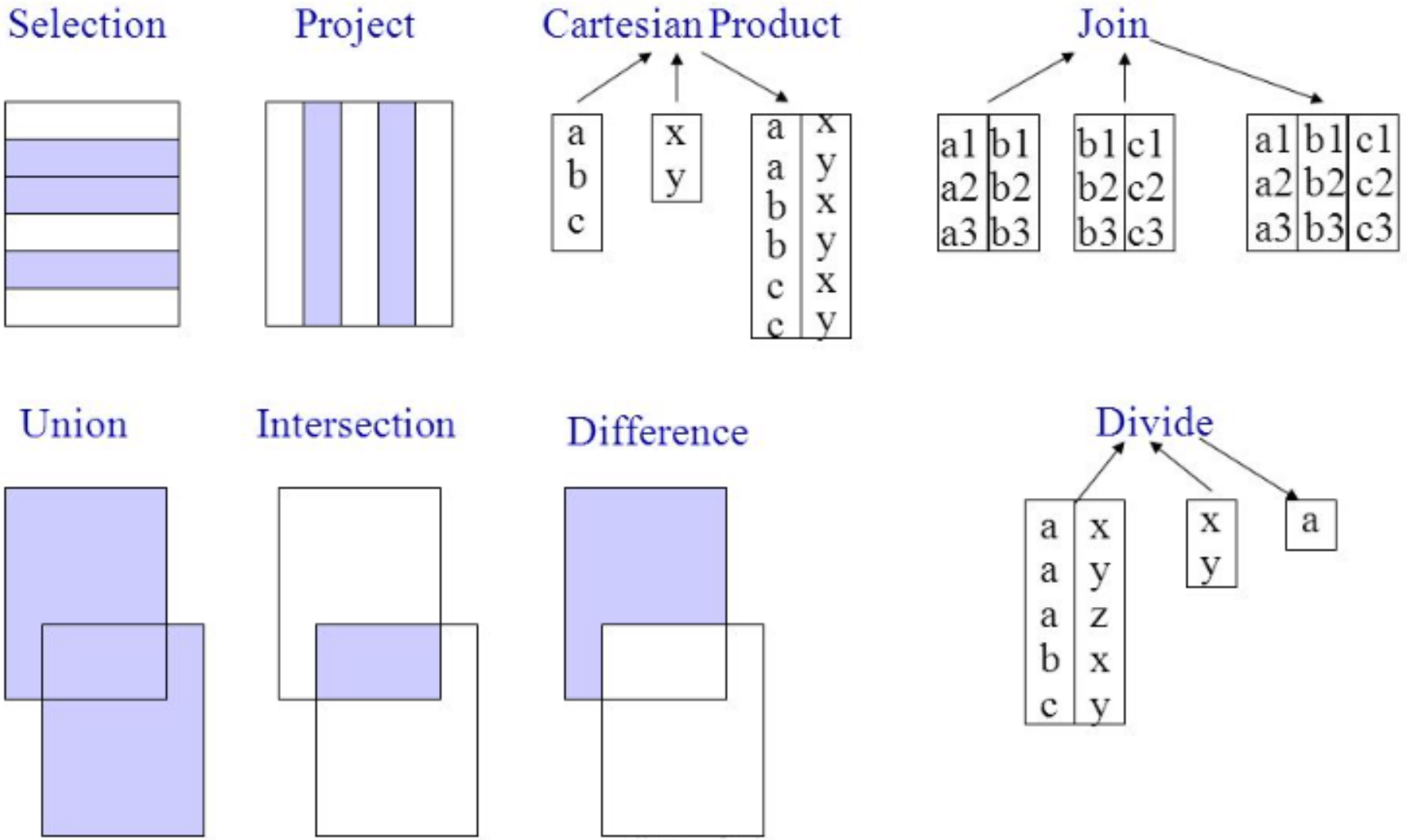
RELATIONAL ALGEBRA



RELATIONAL ALGEBRA

- **Select:** *It returns a relation containing all tuples from specified relation that satisfy a condition.*
- **Project:** *It returns a relation containing all tuples that remain in a specified relation after specified attributes have been removed.*
- **Product:** *It returns a new relation that is an outcome of concatenation (that is chaining) of each tuple of one relation with each tuple of another relation.*
- **Join:** *It returns a relation containing all possible tuples that are a combination of two tuples, one from each of two specified relations such as the two tuples contributing to a given combination have a common value for the common attributes of the two relations.*
- **Union:** *It returns a relation containing all tuples that appear in either or both of two specified relations.*
- **Intersect**: *It returns a relation containing all tuples that appear in both of two specified relations.*
- **Difference**: *It returns a relation containing all tuples that appear in the first not in second of the two specified relations.*
- **Divide:** *The division operator is used when we have to evaluate queries which contain the keyword 'all'. It permits to find values in an attribute of R that have all values of S in the attribute of the same name.*

RELATIONAL ALGEBRA



RELATIONAL ALGEBRA

- **Select Operator (σ):** *It returns a relation containing all tuples from specified relation that satisfy a condition.* It is denoted by sigma (σ).
- Syntax: $\sigma_p(R)$

  σ is used for selection prediction

  R is used for relation

  p is used as a propositional logic formula which may use connectors like: AND (∧), OR (∨), NOT (!). These relational can use as relational operators like =, ≠, ≥, <, >, ≤.
- **Examples-**
  - Select tuples from a relation "Books" where subject is "database"

    $\sigma_{\text{subject = "database"}}$ (Books)

    Select * from Books where subject='database';

  - Select tuples from a relation "Books" where subject is "database" and price is "450"

    $\sigma_{\text{subject = "database"} \wedge \text{price = "450"}}$ (Books)

    Select * from Books where subject='database' and price=450;

RELATIONAL ALGEBRA

- Select tuples from a relation “Books” where subject is “database” and price is “450” or have a publication year after 2010

$$\sigma_{\text{subject = “database” } \wedge \text{ price = “450” } \vee \text{ year >”2010″}} (\text{Books})$$

Select * from Books where subject=‘database’ and price=450 or year=2010;

Points to be remembered for Select operator

- We may use logical operators like ∧ , ∨ , ! and relational operators like = , ≠ , > , < , <= , >= with the selection condition.
- Selection operator only selects the required tuples according to the selection condition.
- Selection operator always selects the entire tuple. It can not select a section or part of a tuple.
- Selection operator is commutative in nature i.e.

  $\sigma_{A \wedge B} (R) = \sigma_{B \wedge A} (R)$
- Degree of the relation from a selection operation is same as degree of the input relation.
- The number of rows returned by a selection operation is obviously less than or equal to the number of rows in the original table.

  Thus,

  Minimum Cardinality = 0, Maximum Cardinality = |R|

RELATIONAL ALGEBRA

- Project Operator (π) is a unary operator in relational algebra that performs a projection operation.
- It displays the columns of a relation or table based on the specified attributes.

Syntax: $\pi_{<attribute\ list>}(R)$

- Example-

Consider the following Student relation

ID	Name	Subject	Age
100	Ashish	Maths	19
200	Rahul	Science	20
300	Naina	Physics	20
400	Sameer	Chemistry	21

$\pi_{Name, Age}(Student)$

Select name, age from student

Name	Age
Ashish	19
Rahul	20
Naina	20
Sameer	21

$\pi_{ID , Name}(Student)$

Select ID, Name from Student

ID	Name
100	Ashish
200	Rahul
300	Naina
400	Sameer

RELATIONAL ALGEBRA

Points to be remembered for Project Operator

- The degree of output relation (number of columns present) is equal to the number of attributes mentioned in the attribute list.
- Projection operator automatically removes all the duplicates while projecting the output relation. So, cardinality of the original relation and output relation may or may not be same. If there are no duplicates in the original relation, then the cardinality will remain same otherwise it will surely reduce.
- If attribute list is a super key on relation R, then we will always get the same number of tuples in the output relation. This is because then there will be no duplicates to filter.
- Projection operator does not obey commutative property i.e.

$$\pi_{<list2>}(\pi_{<list1>}(R)) \neq \pi_{<list1>}(\pi_{<list2>}(R))$$

- Selection Operator performs horizontal partitioning of the relation. Projection operator performs vertical partitioning of the relation.
- There is only one difference between Project and Select operation of SQL. Projection operator does not allow duplicates while SELECT operation allows duplicates. To avoid duplicates in SQL, we use “distinct” keyword and write SELECT distinct. Thus, projection operator of relational algebra is equivalent to SELECT operation of SQL.

RELATIONAL ALGEBRA

- **Product:** The Cartesian product is used to combine each row in one table with each row in the other table. It is also known as a cross product. It is denoted by X.

  Syntax: R X S

- Example-

  Consider the following relations

Employee

DEPT_NO	DEPT_NAME
A	Marketing
B	Sales
C	Legal

Department

EMP_ID	EMP_NAME	EMP_DEPT
1	Smith	A
2	Harry	C
3	John	B

Employee X Department

Select Emp_name, Emp_id,dept_name From Employee, department;

Select Emp_name, Emp_id,dept_name from Employee Cross Join Department;

EMP_ID	EMP_NAME	EMP_DEPT	DEPT_NO	DEPT_NAME
1	Smith	A	A	Marketing
1	Smith	A	B	Sales
1	Smith	A	C	Legal
2	Harry	C	A	Marketing
2	Harry	C	B	Sales
2	Harry	C	C	Legal
3	John	B	A	Marketing
3	John	B	B	Sales
3	John	B	C	Legal

RELATIONAL ALGEBRA

- **Union Operator (∪):** *It returns a relation containing all tuples that appear in either or both of two specified relations.*

  Let R and S be two relations.

  Then-

  - R ∪ S is the set of all tuples belonging to either R or S or both.
  - In R ∪ S, duplicates are automatically removed.
  - Union operation is both commutative and associative.

- Example-

  Consider the following two relations R and S

Relation R

ID	Name	Subject
100	Ankit	English
200	Pooja	Maths
300	Komal	Science

Relation S

ID	Name	Subject
100	Ankit	English
400	Kajol	French

Relation R ∪ S

ID	Name	Subject
100	Ankit	English
200	Pooja	Maths
300	Komal	Science
400	Kajol	French

Select * from R Union Select * from S

RELATIONAL ALGEBRA

- **Intersection Operator (∩):** *It returns a relation containing all tuples that appear in both of two specified relations.*

  Let R and S be two relations.

  Then-

  - R ∩ S is the set of all tuples belonging to both R and S.
  - In R ∩ S, duplicates are automatically removed.
  - Intersection operation is both commutative and associative.

- Example-

Consider the following two relations R and S

Relation R

ID	Name	Subject
100	Ankit	English
200	Pooja	Maths
300	Komal	Science

Relation S

ID	Name	Subject
100	Ankit	English
400	Kajol	French

Relation R ∩ S

ID	Name	Subject
100	Ankit	English

Select * from R Intersect Select * from S

RELATIONAL ALGEBRA

- **Difference Operator (-):** *It returns a relation containing all tuples that appear in the first not in second of the two specified relations.*

  Let R and S be two relations.

  Then-

  - R – S is the set of all tuples belonging to R and not to S.
  - In R – S, duplicates are automatically removed.
  - Difference operation is associative but not commutative.

- Example-

Consider the following two relations R and S

Relation R

ID	Name	Subject
100	Ankit	English
200	Pooja	Maths
300	Komal	Science

Relation S

ID	Name	Subject
100	Ankit	English
400	Kajol	French

Relation R - S

ID	Name	Subject
200	Pooja	Maths
300	Komal	Science

Select * from R Minus Select * from S

RELATIONAL ALGEBRA

- Division Operation is represented by "division"(÷ or /) operator and is used in queries that involve keywords **"every"**, **"all"**, etc.

  Syntax : R(X,Y)/S(Y)

  Here,
  - R is the first relation from which data is retrieved.
  - S is the second relation that will help to retrieve the data.
  - X and Y are the attributes/columns present in relation. We can have multiple attributes in relation, but keep in mind that attributes of S must be a proper subset of attributes of R.
  - For each corresponding value of Y, the above notation will return us the value of X from tuple<X,Y> which exists **everywhere**.
- It's a bit difficult to understand this in a theoretical way, but you will understand this with an example.
- Let's have two relations, ENROLLED and COURSE. ENROLLED consist of two attributes STUDENT_ID and COURSE_ID. It denotes the map of students who are enrolled in given courses.
- COURSE contains the list of courses available.
- See, here attributes/columns of COURSE relation are a proper subset of attributes/columns of ENROLLED relation. Hence Division operation can be used here.

RELATIONAL ALGEBRA

Query 1: STUDENT_ID of students who are enrolled in **every** course.

ENROLLED(STUDENT_ID, COURSE_ID) ÷ COURSE(COURSE_ID)

ENROLLED

STUDENT_ID	COURSE_ID
Student_1	DBMS
Student_2	DBMS
Student_1	OS

÷

COURSE

COURSE_ID
DBMS
OS

ENROLLED ÷ COURSE

STUDENT_ID
Student_1

Query 2: Retrieve the name of subject that is taught in all courses.

SUBJECT(NAME, COURSE) ÷ COURSE(COURSE)

SUBJECT

NAME	COURSE
Systems	BCS
Database	BCS
Database	MCS
Algebra	MCS

÷

COURSE

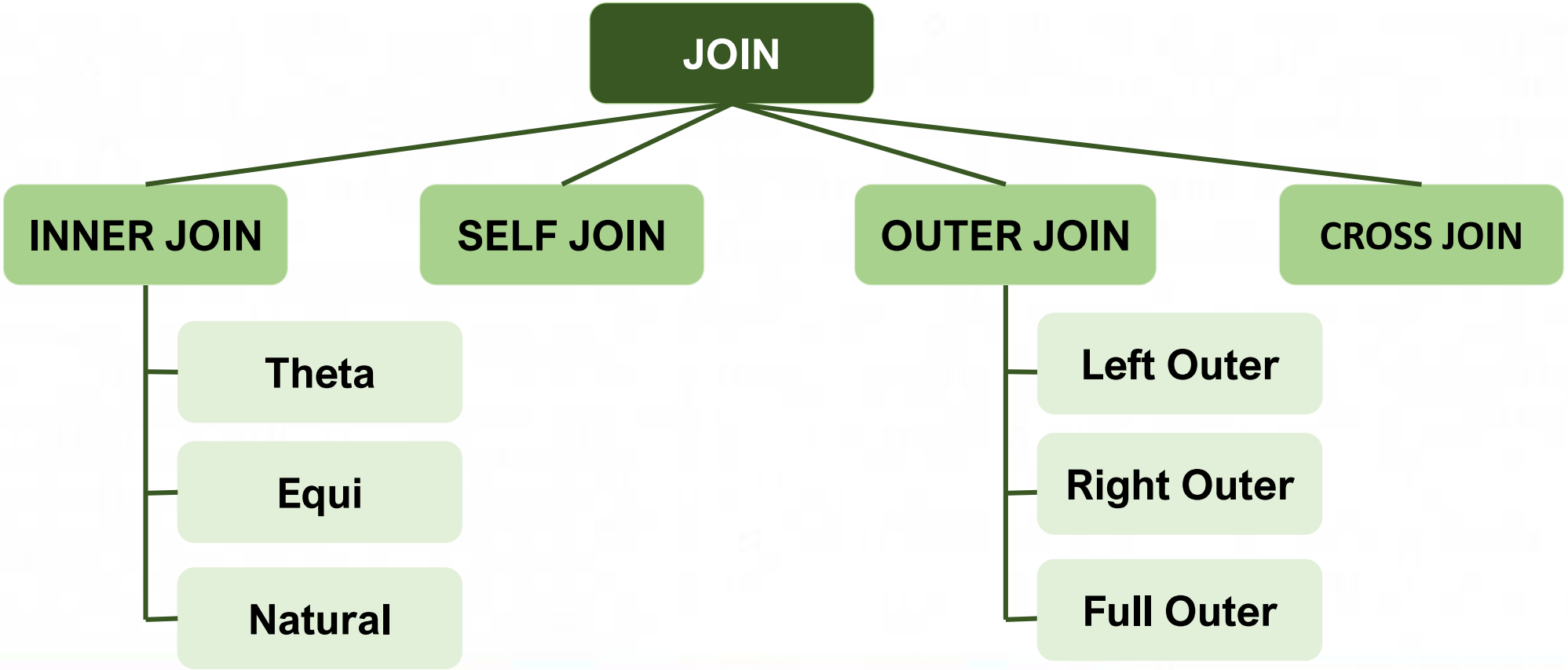
COURSE
BCS
MCS

SUBJECT ÷ COURSE

NAME
Database

RELATIONAL ALGEBRA

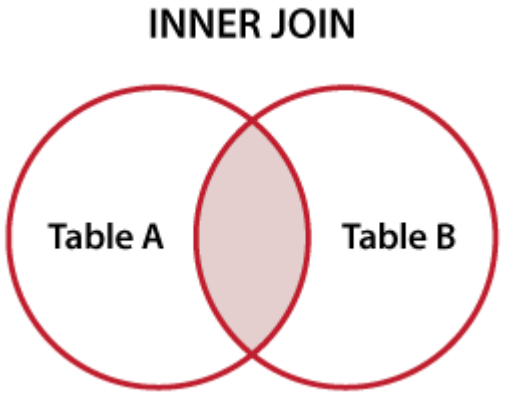
- **Join Operation:** *It returns a relation containing all possible tuples that are a combination of two tuples, one from each of two specified relations such as the two tuples contributing to a given combination have a common value for the common attributes of the two relations.*
- Join Operation in DBMS are binary operations that allow us to combine two or more relations.
- They are further classified into two types: Inner Join, and Outer Join.



RELATIONAL ALGEBRA

- **Inner Join:** When we perform Inner Join, only those tuples returned that satisfy the certain condition. It is also classified into three types: Theta Join, Equi Join and Natural Join.
- **Theta Join (θ):** Theta Join combines two relations using a condition. This condition is represented by the symbol "theta"(θ). Here conditions can be inequality conditions such as >,<,>=,<=, etc. Notation : $R \bowtie_\theta S$, Where R is the first relation, S is the second relation, and θ is the condition.

Let there be a database of all the class 12th boys students in a school. Let's understand Theta Join with the Boys and Interest tables used above :



Boys

ID	Name	Percentage %
1	Rohan	56
2	Rohit	85
3	Amit	75
4	Ravi	79
5	Saiz	65
6	Tejan	84
7	Rishabh	75

Interest

ID	Name	Gender	Sport
3	Amit	M	Cricket
23	Aman	M	Chess
5	Saiz	M	Cricket
10	Shreya	F	Badminton
6	Tejan	M	Chess
15	Sakshi	F	Chess
2	Rohit	M	Cricket

RELATIONAL ALGEBRA

Theta Join -

Boys $\bowtie_{\text{(Boys.ID = Interest.ID and Interest.Sport = Chess and Boys.Percentage > 70)}}$ Interest

So the condition here is
Boys.ID = Interest.ID and Interest.Sport = Chess and Boys.Percentage > 70

so while performing join, we will have to check this condition every time two rows are joined.

Boys

ID	Name	Percentage %
1	Rohan	56
2	Rohit	85
3	Amit	75
4	Ravi	79
5	Saiz	65
6	Tejan	84
7	Rishabh	75

Interest

ID	Name	Gender	Sport
3	Amit	M	Cricket
23	Aman	M	Chess
5	Saiz	M	Cricket
10	Shreya	F	Badminton
6	Tejan	M	Chess
15	Sakshi	F	Chess
2	Rohit	M	Cricket

Boys ⋈θ Interest

ID	Name	Percentage	Gender	Sport
2	Rohit	85	M	Cricket
3	Amit	75	M	Cricket
6	Tejan	84	M	Chess

Select * from Boys Join Interest
On Boys.ID=Interest.ID and
Interest.Sport='Chess' and
Boys.Percentage>70;

RELATIONAL ALGEBRA

Equi join is same as Theta Join, but the only condition is it only uses equivalence condition while performing join between two tables.

A ⋈(... = ...) B, where (... = ...) is the equivalence condition on any of the attributes of the joining table.

In the above example, what if we are told to find out all the students of class 12th who have interest in chess only?

We can perform Equi join as :

Equi join: Boys ⋈(Boys.ID = Interset.ID and Interest.Sport = Chess) Interest

Result after performing Equi join:

Boys

ID	Name	Percentage %
1	Rohan	56
2	Rohit	85
3	Amit	75
4	Ravi	79
5	Saiz	65
6	Tejan	84
7	Rishabh	75

Interest

ID	Name	Gender	Sport
3	Amit	M	Cricket
23	Aman	M	Chess
5	Saiz	M	Cricket
10	Shreya	F	Badminton
6	Tejan	M	Chess
15	Sakshi	F	Chess
2	Rohit	M	Cricket

Boys ⋈(... = ...) Interest

ID	Name	Percentage	Gender	Sport
6	Tejan	84	M	Chess

Select * from Boys Join Interest
On Boys.ID=Interest.ID;

RELATIONAL ALGEBRA

Natural Join is also considered a type of inner join but it does not use any comparison operator for join condition. *It joins the table only when the two tables have at least one common attribute with same name and domain*.

In the result of the Natural Join the common attribute only appears once.

It will be more clear with help of an example :

What if we are told to find all the Students of class 12th and their sports interest we can apply Natural Join as : Boys ⋈ Interest

So when we perform Natural Join on table Boys and table Interest they both have a common attribute ID and have the same domain. So, the Result of Natural Join will be:

Boys

ID	Name	Percentage %
1	Rohan	56
2	Rohit	85
3	Amit	75
4	Ravi	79
5	Saiz	65
6	Tejan	84
7	Rishabh	75

Interest

ID	Name	Gender	Sport
3	Amit	M	Cricket
23	Aman	M	Chess
5	Saiz	M	Cricket
10	Shreya	F	Badminton
6	Tejan	M	Chess
15	Sakshi	F	Chess
2	Rohit	M	Cricket

Boys ⋈ Interest

ID	Name	Percentage	Gender	Sport
2	Rohit	85	M	Cricket
3	Amit	75	M	Chess
5	Saiz	65	M	Cricket
6	Tejan	84	M	Chess

Select * from Boys Natural Join Interest ;

RELATIONAL ALGEBRA

Outer Join

Outer Join in Relational algebra returns all the attributes of both the table depending on the condition. If some attribute value is not present for any one of the tables it returns NULL in the respective row of the table attribute.

It is further classified as:

- Left Outer Join
- Right Outer Join
- Full Outer Join

Let's see how these Joins are performed.

Left Outer Join

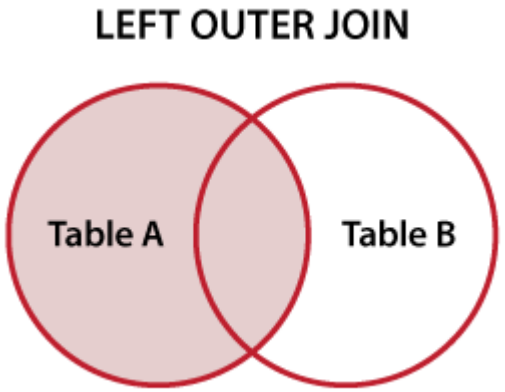
It returns all the rows of the left table even if there is no matching row for it in the right table performing Left Outer Join.

$$A \ ⟕ \ B$$

Let's perform Left Outer Join on table Boys and Interest and find out all the boys of class 12th and their sports interest.

RELATIONAL ALGEBRA

If we perform Left Outer Join on table Boys and table Interest such that Boys.ID = Interest.ID . Then Result of the Join will be:



Boys ⟕ Interest

ID	Name	Percentage %
1	Rohan	56
2	Rohit	85
3	Amit	75
4	Ravi	79
5	Saiz	65
6	Tejan	84
7	Rishabh	75

ID	Name	Gender	Sport
3	Amit	M	Cricket
23	Aman	M	Chess
5	Saiz	M	Cricket
10	Shreya	F	Badminton
6	Tejan	M	Chess
15	Sakshi	F	Chess
2	Rohit	M	Cricket

Boys.ID	Boys.Name	Boys.Percentage	Interest.ID	Interest.Name	Interest.Gender	Interest.Sport
1	Rohan	56	NULL	NULL	NULL	NULL
2	Rohit	85	2	Rohit	M	Cricket
3	Amit	75	3	Amit	M	Cricket
4	Ravi	79	NULL	NULL	NULL	NULL
5	Saiz	65	5	Saiz	M	Cricket
6	Tejan	84	6	Tejan	M	Chess
7	Rishabh	75	NULL	NULL	NULL	NULL

Select * from Boys Left Outer Join Interest On Boys.ID=Interest.ID;

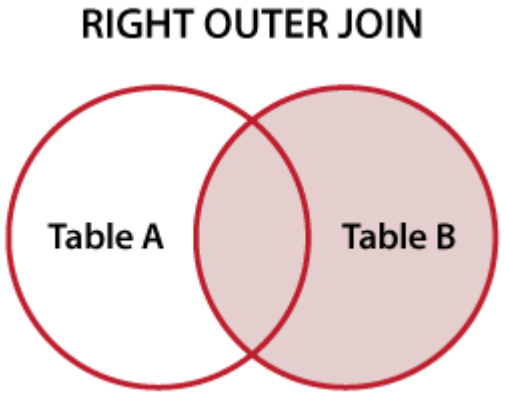
RELATIONAL ALGEBRA

Right Outer Join

It returns all the rows of the second table even if there is no matching row for it in the first table performing Right Outer Join.

A ⟖ B

Let's perform Right Outer Join on table Boys and Interest and find out all the boys of class 12th and their sports interest. If we perform Right Outer Join on table Boys and table Interest such that Boys.ID = Interest.ID . Then Result of the join will be:



RELATIONAL ALGEBRA

If we perform Right Outer Join on table Boys and table Interest such that Boys.ID = Interest.ID . Then Result of the join will be:

Clearly, we can observe that all the rows of the right table, i.e., table Interest is present in the result.

ID	Name	Percentage %
1	Rohan	56
2	Rohit	85
3	Amit	75
4	Ravi	79
5	Saiz	65
6	Tejan	84
7	Rishabh	75

ID	Name	Gender	Sport
3	Amit	M	Cricket
23	Aman	M	Chess
5	Saiz	M	Cricket
10	Shreya	F	Badminton
6	Tejan	M	Chess
15	Sakshi	F	Chess
2	Rohit	M	Cricket

Boys ⋈ Interest

Boys.ID	Boys.Name	Boys.Percentage	Interest.ID	Interest.Name	Interest.Gender	Interest.Sport
3	Amit	75	3	Amit	M	Cricket
NULL	NULL	NULL	23	Aman	M	Chess
5	Saiz	65	5	Saiz	M	Cricket
NULL	NULL	NULL	10	Shreya	F	Badminton
6	Tejan	84	6	Tejan	M	Chess
NULL	NULL	NULL	15	Sakshi	F	Chess
2	Rohit	85	2	Rohit	M	Cricket

Select * from Boys Right Outer Join Interest On Boys.ID=Interest.ID;

RELATIONAL ALGEBRA

Full Outer Join

It returns all the rows of the first and second Table.

A ⟗ B

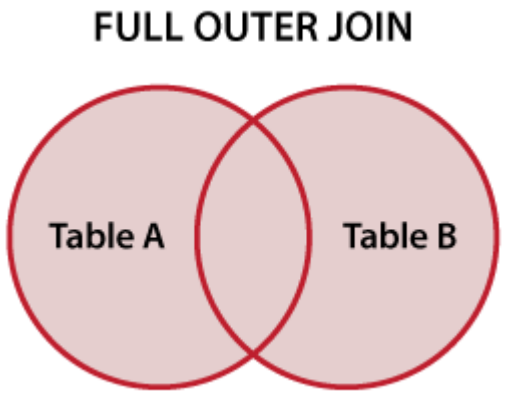
Clearly, we can observe that all the rows of the right table and left Table, i.e., Table B and A are present in the result.

ID	Name	Percentage %
1	Rohan	56
2	Rohit	85
3	Amit	75
4	Ravi	79
5	Saiz	65
6	Tejan	84
7	Rishabh	75

ID	Name	Gender	Sport
3	Amit	M	Cricket
23	Aman	M	Chess
5	Saiz	M	Cricket
10	Shreya	F	Badminton
6	Tejan	M	Chess
15	Sakshi	F	Chess
2	Rohit	M	Cricket

Boys ⟗ Interest

Boys.ID	Boys.Name	Boys.Percentage	Interest.ID	Interest.Name	Interest.Gender	Interest.Sport
1	Rohan	56	NULL	NULL	NULL	NULL
2	Rohit	85	2	Rohit	M	Cricket
3	Amit	75	3	Amit	M	Cricket
4	Ravi	79	NULL	NULL	NULL	NULL
5	Saiz	65	5	Saiz	M	Cricket
6	Tejan	84	6	Tejan	M	Chess
7	Rishabh	75	NULL	NULL	NULL	NULL
NULL	NULL	NULL	23	Aman	M	Chess
NULL	NULL	NULL	10	Shreya	F	Badminton
NULL	NULL	NULL	15	Sakshi	F	Chess



Select * from Boys Full Outer Join Interest On Boys.ID=Interest.ID;

RELATIONAL CALCULUS

- Relational Calculus is a **non-procedural query language** used in database management systems (DBMS) to specify queries.

- Unlike relational algebra, which focuses on operations, **relational calculus specifies what to retrieve rather than how to retrieve it**.

- It is based on **predicate logic** and consists of formulas that define the required result set without specifying a step-by-step execution method.

- **Types of Relational Calculus**
  Relational Calculus is mainly divided into two types:
  - **Tuple Relational Calculus (TRC)** – Works with **tuples (rows)** in a relation.
  - **Domain Relational Calculus (DRC)** – Works with **domains (column values)** instead of tuples.

RELATIONAL CALCULUS

Tuple Relational Calculus (TRC)

- In TRC, queries are expressed using **tuple variables**.
- The result is a set of tuples that satisfy a given condition.
- The general form of TRC query: {t | P(t)}

  where:

  t is a **tuple variable** (representing a row in the table).

  P(t) is a **predicate (condition)** that must be true for t to be included in the result.

Example

Find the employees who work in the "HR" department:

$$\{t \mid t \in Employees \text{ AND } t.dept =' HR'\}$$

This retrieves all tuples t from the **Employees** table where the department is "HR".

Operators in TRC

- **Logical Operators:** AND (∧), OR (∨), NOT (¬)
- **Comparison Operators:** =, ≠, >, <, ≥, ≤
- **Existential (∃) and Universal (∀) Quantifiers**

RELATIONAL CALCULUS

Operators in TRC

- **Logical Operators:** AND (∧), OR (∨), NOT (¬)
- **Comparison Operators:** =, ≠, >, <, ≥, ≤
- **Existential (∃) and Universal (∀) Quantifiers**

Example with Existential Quantifier (∃)

Find employees who work in at least one department:

$$\{t \mid t \in Employees \text{ AND } \exists d(d \in Departments \text{ AND } t.dept\_id = d.dept\_id)\}$$

Here, **∃ d** ensures that an employee is included only if a matching department exists.

RELATIONAL CALCULUS

Domain Relational Calculus (DRC)

- In DRC, queries are expressed in terms of **column values (domains)** instead of tuples.
- The result is a set of values rather than complete tuples.
- The general form of DRC query:

  {(x1,x2,...,xn) | P(x1,x2,...,xn)}

  where:

  x1,x2,...,xn represent attribute values (domains).

  P(x1,x2,...,xn) is a condition that must be true for the values to be included in the result.

Example

Find the names of employees who work in the "HR" department:

$$\{e\_name \mid \exists d(emp\_id, e\_name, d\_id) \in Employees \text{ AND } (d\_id, dept\_name) \in Departments \text{ AND } dept\_name =' HR'\}$$

This retrieves only the **employee names** instead of entire tuples.

RELATIONAL CALCULUS

Operators in TRC

- **Logical Operators:** AND (∧), OR (∨), NOT (¬)
- **Comparison Operators:** =, ≠, >, <, ≥, ≤
- **Existential (∃) and Universal (∀) Quantifiers**

- **Example with Universal (∀) Quantifier**

  Find employees who work in every department:

$$\{e\_name \mid \forall d(d \in Departments \rightarrow \exists emp(emp \in Employees \text{ AND } emp.dept\_id = d.dept\_id))\}$$

  Here, **∀ d** ensures that an employee works in every department.

RELATIONAL CALCULUS

Relational Algebra and Relational Calculus

Feature	Relational Algebra	Relational Calculus
Type	Procedural (specifies how to retrieve data)	Declarative (specifies what to retrieve)
Operators Used	Uses operators like SELECT (σ), PROJECT (π), JOIN (⋈), UNION (∪), etc.	Uses predicate logic, quantifiers (∃, ∀)
Flexibility	Less flexible, requires sequence of operations	More flexible, does not specify sequence
Implementation	Easier to implement in DBMS	More theoretical, used for query formulation

Relational Calculus

- **Declarative Query Language:** Focuses on describing the desired result rather than the retrieval process.
- **Based on Predicate Logic:** Uses conditions and quantifiers to filter data.
- **Supports Expressive Queries:** Can handle complex queries using existential and universal quantifiers.
- **Forms the Basis for SQL:** SQL is influenced by relational calculus, particularly in its use of predicates and logical expressions.

RELATIONAL CALCULUS

Exercises on Tuple Relational Calculus (TRC) and Domain Relational Calculus (DRC)

Given Database Schema

Employee(Emp_id, Emp_Name, Dept_id, Salary)

Department(Dept_id, Dept_Name)

Employee Table

emp_id	emp_name	dept_id	salary
1	Alice	10	50000
2	Bob	20	60000
3	Charlie	NULL	55000
4	David	10	65000
5	Emma	30	70000

Department Table

dept_id	dept_name
10	HR
20	Finance
30	IT
40	Marketing

1. Retrieve the names of all employees who earn more than 55,000.

$$\{t.emp\_name \mid t \in Employee \text{ AND } t.salary > 55000\}$$

$$\{e\_name \mid \exists emp\_id, dept\_id, salary\ (emp\_id, e\_name, dept\_id, salary) \in Employee \text{ AND } salary > 55000\}$$

RELATIONAL CALCULUS

2. Find employees who work in the HR department.

$$\{t.emp\_name \mid t \in Employee \text{ AND } \exists d(d \in Department \text{ AND } d.dept\_id = t.dept\_id \text{ AND } d.dept\_name =' HR')\}$$

$$\{e\_name \mid \exists emp\_id, dept\_id, salary, dept\_name\ (emp\_id, e\_name, dept\_id, salary) \in Employee \text{ AND } (dept\_id, dept\_name) \in Department \text{ AND } dept\_name =' HR'\}$$

3. Find the employees who do not belong to any department (i.e., dept_id is NULL).

$$\{t.emp\_name \mid t \in Employee \text{ AND } t.dept\_id = \text{NULL}\}$$

$$\{e\_name \mid \exists emp\_id, salary\ (emp\_id, e\_name, NULL, salary) \in Employee\}$$

4. Find all departments that have at least one employee.

$$\{d.dept\_name \mid d \in Department \text{ AND } \exists t(t \in Employee \text{ AND } t.dept\_id = d.dept\_id)\}$$

$$\{dept\_name \mid \exists dept\_id\ (dept\_id, dept\_name) \in Department \text{ AND } \exists emp\_id, e\_name, salary\ (emp\_id, e\_name, dept\_id, salary) \in Employee\}$$

5. Retrieve employees who work in every department.

$$\{t.emp\_name \mid t \in Employee \text{ AND } \forall d(d \in Department \rightarrow \exists e(e \in Employee \text{ AND } e.emp\_id = t.emp\_id \text{ AND } e.dept\_id = d.dept\_id))\}$$

$$\{e\_name \mid \forall dept\_id, dept\_name((dept\_id, dept\_name) \in Department \rightarrow \exists emp\_id, salary((emp\_id, e\_name, dept\_id, salary) \in Employee))\}$$

RELATIONAL CALCULUS

6. Retrieve the department names that have no employees.

$$\{d.dept\_name \mid d \in Department \text{ AND } \neg\exists t(t \in Employee \text{ AND } t.dept\_id = d.dept\_id)\}$$

$$\{dept\_name \mid \exists dept\_id(dept\_id, dept\_name) \in Department \text{ AND } \neg\exists emp\_id, e\_name, salary((emp\_id, e\_name, dept\_id, salary) \in Employee)\}$$

7. Find employees who work in Finance and earn more than ₹70,000.

$$\{t.Name \mid t \in Employee \text{ AND } t.Salary > 70000 \text{ AND } \exists d(d \in Department \text{ AND } d.Dept\_ID = t.Dept\_ID \text{ AND } d.Dept\_Name =' Finance')\}$$

$$\{Name \mid \exists Emp\_ID, Age, Gender, Salary\ (Emp\_ID, Name, Age, Gender, Salary, Dept\_ID) \in Employee \text{ AND } Salary > 70000 \text{ AND}$$

$$(Dept\_ID, Dept\_Name, Location) \in Department \text{ AND } Dept\_Name =' Finance'\}$$

8. Find the highest-paid employee in the company.

$$\{t.Name \mid t \in Employee \text{ AND } \neg\exists e(e \in Employee \text{ AND } e.Salary > t.Salary)\}$$

$$\{Name \mid \exists Emp\_ID, Age, Gender, Dept\_ID\ (Emp\_ID, Name, Age, Gender, Salary, Dept\_ID) \in Employee \text{ AND}$$

$$\neg\exists Emp\_ID1, Name1, Age1, Salary1\ (Emp\_ID1, Name1, Age1, Gender1, Salary1, Dept\_ID1) \in Employee \text{ AND } Salary1 > Salary\}$$

MODULE – 3

RELATIONAL DATABASE DESIGN

Data Dependency/Functional Dependency

A functional dependency is an association between two attributes of the same relational database table. One of the attributes is called the determinant and the other attribute is called the determined. For each value of the determinant there is associated one and only one value of the determined.

If A is the determinant and B is the determined then we say that *A functionally determines B* and graphically represent this as A -> B. The symbols A & B can also be expressed as *B is functionally determined by* A.

Since for each value of A there is associated one and only one value of B.

The following table illustrates A ⇨ B:

A	B
1	1
2	4
3	9
4	16
2	4
7	9

Data Dependency/Functional Dependency

A	B
1	1
2	4
3	9
4	16
2	4
3	11
7	9

In this table ***A does not functionally determine B.***

Since for A = 3 there is associated more than one value of B.

Functional dependency can also be defined as follows:
An attribute in a relational model is said to be functionally dependent on another attribute in the table if it can take only one value for a given value of the attribute upon which it is functionally dependent.

Data Dependency/Functional Dependency

A functional dependency denoted (FD) is denoted by X $\rightarrow$ Y, between two sets of attributes X and Y that are subset of relation R specifies a constraint on the tuples that can form a relation state of r of R. The constraint is that, for any two tuple t1 and t2 in r that have t1[X] = t2[X], we must have t1[Y] = t2[Y].

It means that the values of the Y of a tuple in r depend upon or determined by, the value of X component. Alternatively, the values of X component of a tuple uniquely (or Functionally) determine the values of the Y component.

In other words, there is a functional dependency from X to Y or that Y is functionally dependent on X.

Thus, X functionally determines Y in a relation schema R if and only if, whenever two tuples of r(R) agree on their X-value, they must necessarily agree on their Y value.

Data Dependency/Functional Dependency

For example, in the relation schema PROJECT of the employees, the following functional dependencies should hold-

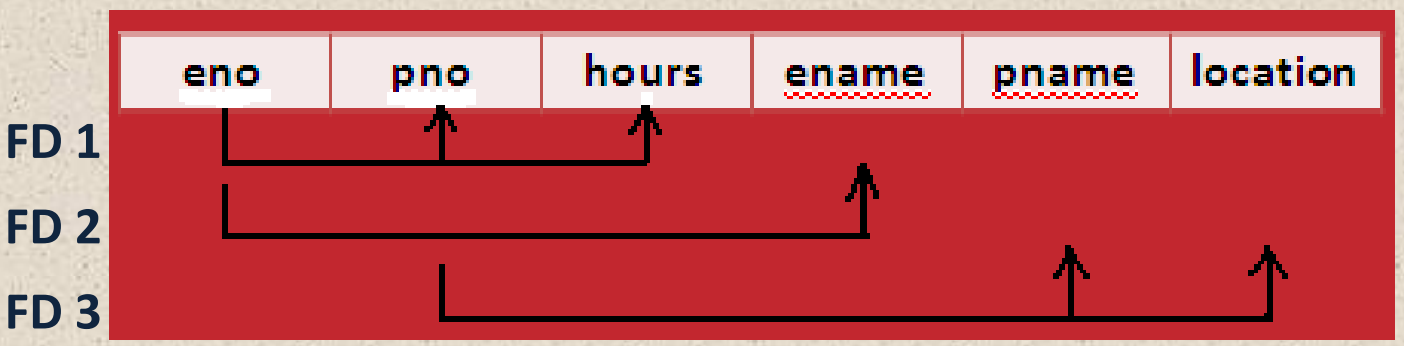
1. **eno → ename**
2. **pno → pname, location**
3. **Eno, pno → hours**

These functional dependencies specifies that

1. The value of an employee's number (eno) uniquely determines the employee name (ename).
2. The value of project number (pno) uniquely determines that project name (pname) and project location (location).
3. A combination of eno and pno values uniquely determines the number of hours the employee works on the project per week.

Alternatively, we can say that ename is functionally determined by eno and so on.

PROJECT SCHEMA



Data Dependency/Functional Dependency

Types of Functional Dependencies

There are many types of functional dependencies that depending on different criteria-

1. Full Functional Dependency
2. Partial Dependency
3. Transitive Dependency
4. Trivial and Non-trivial Dependency

- **Full Functional Dependency:**

  For a relation schema R and a FD, A $\rightarrow$ B, B is fully functionally dependent on A if there is no C, where C is proper subset of A, such that C $\rightarrow$ B.

  The Dependency A $\rightarrow$ B is left reduced, that is, there is no extraneous attributes in left side in the dependency.

- **Partial Dependency:**

  A functional dependency that holds in a relation is partial when removing one of the determining attributes gives a functional dependency that holds in the relation.

  E.g. if {A,B} → {C} but also {A} → {C} then {C} is partially functionally dependent on {A,B}.

Data Dependency/Functional Dependency

- **Partial Dependency:**

  Partial Dependency is a form of Functional dependency that holds on a set of attributes. It is about the complete dependency of a right hand side attribute on one of the left hand side attributes. In a functional dependency XY →Z, if Z (RHS attribute) can be uniquely identified by one of the LHS attributes, then the functional dependency is partial dependency.

  Example:

  Let us assume a relation R with attributes A, B, C, and D. Also, assume that the set of functional dependencies F that hold on R as follows;

  F = {A → B, D → C}.

  From set of attributes F, we can derive the primary key. For R, the key can be (A,D), a composite primary key. That means, AD → BC, AD can uniquely identify B and C. But, for this case A and D is not required to identify B or C uniquely. To identify B, attribute A is enough. Likewise, to identify C, attribute D is enough. The functional dependencies AD → B or AD → C are called as Partial functional dependencies.

Data Dependency/Functional Dependency

- **Transitive Dependency:** The functional dependency follows the mathematical property of *transitivity*, which states that if A=B and B=C, then A=C. Because ItemNo determines CategoryID, which in turn determines CategoryName and CategoryManager, the relation contains a transitive dependency.

ItemNo → Title, Price, CategoryID
CategoryId → CategoryName, CategoryManager

Transitive dependencies occur when there is an indirect relationship that causes a functional dependency.

Examples: For example, ”A -> C” is a transitive dependency when it is true only because both “A -> B” and “B -> C” are true.

Data Dependency/Functional Dependency

- **Trivial and Non-trivial Dependency:**

  Trivial: If an FD X → Y holds where Y subset of X, then it is called a trivial FD. Trivial FDs are always hold.

  Symbolically: A → B is trivial functional dependency if B is a subset of A.

  The following dependencies are also trivial:

  A → A

  AB → A

  AB → B

  B → B

  For example:

  Consider a table with two columns Student_id and Student_Name.

  {Student_Id, Student_Name} → Student_Id

  is a trivial functional dependency as Student_Id is a subset of {Student_Id, Student_Name}. That makes sense because if we know the values of Student_Id and Student_Name then the value of Student_Id can be uniquely determined. Also,

  Student_Id→Student_Id

  Student_Name→Student_Name

  are trivial dependencies too.

Data Dependency/Functional Dependency

- **Trivial and Non-trivial Dependency:**

  Non-trivial: If an FD X → Y holds where Y is not subset of X, then it is called non-trivial FD.

  For example:

  An employee table with three attributes:

  emp_id, emp_name, emp_address.

  The following functional dependencies are non-trivial:

  emp_id→ emp_name (emp_name is not a subset of emp_id)

  emp_id → emp_address (emp_address is not a subset of emp_id)

  On the other hand, the following dependencies are trivial:

  emp_id, emp_name → emp_name

  emp_name is a subset of {emp_id, emp_name}

  Completely non-trivial: If an FD X → Y holds where X intersect Y = Φ, is said to be completely non-trivial FD.

Data Dependency/Functional Dependency

Conclusion of Functional Dependency

Functional Dependencies:

- Data dependencies are constraints imposed on data in database.
- They are part of the scheme definition.
- FDs allow us to formally define keys.
- A conjecture (It has to be proven) is that a set of functional dependencies and one join dependency are enough to express the dependency structure of a relational database scheme.

Motivation:

Functional dependencies help in accomplishing the following two goals:

(a) controlling redundancy and
(b) enhancing data reliability.

Data Dependency/Functional Dependency

Problematic Issue:

Representing the set of all FDs for a relation R.

Solution:

- Find a basic set of FDs.
- Use axioms for inferring.
- Represent the set of all FDs as the set of FDs that can be inferred from the basic set of FDs.

Axioms:

An inference axiom is a rule that states if a relation satisfies certain FDs then it must satisfy certain other FDs.

Data Dependency/Functional Dependency

Armstrong's axioms are a set of axioms (or, more precisely, inference rules) used to infer all the functional dependencies on a relational database. They were developed by William W. Armstrong.

FD manipulations:
Soundness -- no incorrect FD's are generated
Completeness -- all FD's can be generated

Let R(U) be a relation scheme over the set of attributes U. We will use the letters X, Y, Z & W to represent any subset of and, for short, the union of two sets of attributes and by instead of the usual X U Y.

F1. Reflexivity/Self Determination X → X
F2. Augmentation If ($Z \subseteq W$; X → Y) then XW → YZ, If A → B then AC → BC
F3. Additivity/Union If { (X → Y) (X → Z)} then X → YZ
F4. Projectivity/Decomposition If (X → YZ) then X → Y, X → Z
F5. Transitivity If (X → Y) and (Y → Z) then (X → Z)
F6. Pseudotransitivity If (X → Y) and (YZ → W) then XZ → W

Data Dependency/Functional Dependency

Axiom Name	Axiom	Example
Reflexivity	if a is set of attributes, b ⊆ a, then a →b	SSN,Name → SSN
Augmentation	if a→ b holds and c is a set of attributes, then ca→cb	SSN → Name then SSN,Phone → Name, Phone
Transitivity	if a →b holds and b→c holds, then a→ c holds	SSN →Zip and Zip→City then SSN →City
Union or Additivity *	if a → b and a → c holds then a→ bc holds	SSN→Name and SSN→Zip then SSN→Name,Zip
Decomposition or Projectivity*	if a → bc holds then a → b and a → c holds	SSN→Name,Zip then SSN→Name and SSN→Zip
Pseudotransitivity*	if a → b and cb → d hold then ac → d holds	Address → Project and Date→Amount then Address,Date → Amount
(NOTE)	ab→ c does NOT imply a → b and b → c	

Data Dependency/Functional Dependency

Prove or disprove the following inference rules for functional dependencies-

i. $\{W \rightarrow Y, X \rightarrow Z\} \Rightarrow \{WX \rightarrow Y\}$

ii. $\{X \rightarrow Y\}$ and $Y \supseteq Z \Rightarrow \{X \rightarrow Z\}$

iii. $\{X \rightarrow Y, Y \rightarrow Z\} \Rightarrow \{X \rightarrow YZ\}$

iv. $\{X \rightarrow Z, Y \rightarrow Z\} \Rightarrow \{X \rightarrow Y\}$

v. $\{XY \rightarrow Z, Z \rightarrow W\} \Rightarrow \{X \rightarrow W\}$

i. $\{W \rightarrow Y, X \rightarrow Z\} \Rightarrow \{WX \rightarrow Y\}$

Given

$W \rightarrow Y$ --------- (a)

$X \rightarrow Z$ --------- (b)

by augmenting in (a) by X

$WX \rightarrow XY$ ------- (c)

by decomposing in (c)

$WX \rightarrow Y$ Hence, it is proved.

Data Dependency/Functional Dependency

Prove or disprove the following inference rules for functional dependencies-

ii. $\{X \rightarrow Y\}$ and $Y \supseteq Z \Rightarrow \{X \rightarrow Z\}$

Given

$X \rightarrow Y$ --------- (a)

$Y \supseteq Z$ and that two tuples t1 and t2 exist in some relation instance r of R such that t1[Y] = t2[Y]. Then t1[Z] = t2[Z] because $Y \supseteq Z$; hence $Y \rightarrow Z$ must hold.

by transitivity $X \rightarrow Y$ and $Y \rightarrow Z$ then $X \rightarrow Z$. Hence, it is proved.

iii. $\{X \rightarrow Y, Y \rightarrow Z\} \Rightarrow \{X \rightarrow YZ\}$

Given

$X \rightarrow Y$ --------- (a)

$Y \rightarrow Z$ --------- (b)

by augmentation rule on (b) by Y

$Y \rightarrow YZ$ --------- (c)

and by transitivity rule on (a) and (c)

$X \rightarrow Y$ and $Y \rightarrow YZ \Rightarrow \{X \rightarrow YZ\}$

Hence, it is proved.

Data Dependency/Functional Dependency

Prove or disprove the following inference rules for functional dependencies-

iv. $\{X \rightarrow Z, Y \rightarrow Z\} \Rightarrow \{X \rightarrow Y\}$

Given $X \rightarrow Z$ i.e. $X \supseteq Z$ and that any two tuples t1 and t2 of relation R such that t1[X] = t2[X] then t1[Z] = t2[Z].

Also given $Y \rightarrow Z$ i.e. $Y \supseteq Z$ and that any two tuples t1 and t2 of relation R such that t1[Y] = t2[Y] then t1[Z] = t2[Z].

This implies that $X \supseteq Y$; i.e. $X \rightarrow Y$. Hence, it is proved.

iv. $\{XY \rightarrow Z, Z \rightarrow W\} \Rightarrow \{X \rightarrow W\}$

Given

$XY \rightarrow Z$ --------- (a)

$Z \rightarrow W$ --------- (b)

by transitivity rule on (a) and (c)

$XY \rightarrow Z$ and $Z \rightarrow W \Rightarrow \{XY \rightarrow W\}$ -------- (c)

by decomposition rule on (c)

$X \rightarrow W, Y \rightarrow W$

Hence, it is proved.

Data Dependency/Functional Dependency

Closure of Functional Dependencies

Suppose F is a Set of functional dependencies for a given relation R. Then,

Closure of F:

It is a set of all functional dependencies that include F as well as all dependencies that are inferred from F. It is denoted by

$$F^+$$

$X \rightarrow Y$ is inferred from F specified in R if $X \rightarrow Y$ holds in every legal relation state r of R.

By applying the following 6 inference rules, we can inferred FDs of F.

1. Reflexivity $X \rightarrow X$
2. Augmentation If ($Z \subseteq W$; $X \rightarrow Y$) then $XW \rightarrow YZ$
3. Additivity If { ($X \rightarrow Y$) ($X \rightarrow Z$)} then $X \rightarrow YZ$
4. Projectivity If ($X \rightarrow YZ$) then $X \rightarrow Y$
5. Transitivity If ($X \rightarrow Y$) and ($Y \rightarrow Z$) then ($X \rightarrow Z$)
6. Pseudotransitivity If ($X \rightarrow Y$) and ($YZ \rightarrow W$) then $XZ \rightarrow W$

Data Dependency/Functional Dependency

Example for Closure of F:

Suppose we are given a relation scheme $R=(A,B,C,G,H,I)$, and the set of functional dependencies:

$$A \rightarrow B$$
$$A \rightarrow C$$
$$CG \rightarrow H$$
$$CG \rightarrow I$$
$$B \rightarrow H$$

Applying the rules to the scheme and set F mentioned above, we can derive the following:

1. $A \rightarrow H$, as we saw by the transitivity rule.
2. $CG \rightarrow HI$ by the union rule.
3. $AG \rightarrow I$ by several steps:
   i. Note that $A \rightarrow C$ holds.
   ii. Then $AG \rightarrow CG$, by the augmentation rule.
   iii. Now by transitivity, $AG \rightarrow I$.

(You might notice that this is actually pseudotransivity if done in one step.)

Data Dependency/Functional Dependency

Example for Closure of F:

Assume that there are 4 attributes A, B, C, D, and that F = {A → B, B → C}. Then, F + includes all the following:

FDs: A → A, A → B, A → C, B → B, B → C, C → C, D → D, AB → A, AB → B, AB → C, AC → A, AC → B, AC → C, AD → A, AD → B, AD → C, AD → D, BC → B, BC → C, BD → B, BD → C, BD → D, CD → C, CD → D, ABC → A, ABC → B, ABC → C, ABD → A, ABD → B, ABD → C, ABD → D, BCD → B, BCD → C, BCD → D, ABCD → A, ABCD → B, ABCD → C, ABCD → D.

Data Dependency/Functional Dependency

Example for Closure of F:

Assume that there are 4 attributes A, B, C, D, and that F = {A → B, B → C}.

To compute F+,

we first get:

A+ = AB+ = AC+ = ABC+ = {A, B, C}

B+ = BC+ = {B, C}

C+ = {C}

D+ = {D}

AD+ = {A, B, D}

BC+ = {B, C}

BD+ = BCD+ = {B, C, D}

ABD+ = ABCD+ = {A, B, C, D}

ACD+ = {A, B, C, D}

It is easy to generate the FDs in F + from the closures of the above attribute sets.

Data Dependency/Functional Dependency

Closure of Attribute

Define the closure of α under F (denoted by α+) as the set of attributes that are functionally determined by α under F:

$\alpha \rightarrow \beta$ is in F+ $\Leftrightarrow \beta \subseteq \alpha+$

- Algorithm to compute α+, the closure of α under F

```
result := α;
while (changes to result) do
        for each β → γ in F do
                begin
                if β ⊆ result then result := result ∪ γ;
                end
```

Example:

R = (A, B, C, G, H, I) and F = {A → B A → C CG → H CG → I B → H}

- (AG+)
  1. result = AG
  2. result = ABCG (A → C and A ⊆ AGB)
  3. result = ABCGH (CG → H and CG ⊆ AGBC)
  4. result = ABCGHI (CG → I and CG ⊆ AGBCH)

Data Dependency/Functional Dependency

Finding Candidate Key

Let F be a set of FDs, and R a relation.

A candidate key is a set X of attributes in R such that

- X + includes all the attributes in R.
- There is no proper subset Y of X such that Y + includes all the attributes in R.

Note: A proper subset Y is a subset of X such that Y != X (i.e., X has at least one element not in Y).

Example.

Consider a table R(A, B, C, D), and that F = {A → B, B → C}.

A is not a candidate key, because A + = {A, B, C} which does not include D.
ABD is not a candidate key even though ABD+ = {A, B, C, D}.
This is because AD+ = {A, B, C, D}, namely, there is a proper subset AD of ABD such that AD+ includes all the attributes. AD is a candidate key.

Data Dependency/Functional Dependency

Finding Candidate Key

Example.

Consider a table R(A, B, C, D, E, F), and that F = {A → C, C → D, D → B, E → F}.
Find all the possible candidate key.

Given A → C, A determines C
C → D, C determines D
D → B, D determines B
E → F E determines F

Now, the easiest way is to find which attributes are not determined. In this example, A and E are not determined. Then, find out the closure of (AE)+.

(AE)+ = AE
= ACE
= ACDE
= ACDBE
= ACDBEF

Closure of AE has all the attributes. Thus, AE is a candidate key. In this way, we can find out more candidate keys for this problem.

Normalization

While designing a database out of an entity–relationship model, the main problem existing in that “raw” database is redundancy. Redundancy is storing the same data item in more one place. A redundancy creates several problems like the following:

- Extra storage space: storing the same data in many places takes large amount of disk space.
- Entering same data more than once during data insertion.
- Deleting data from more than one place during deletion.
- Modifying data in more than one place.
- Anomalies may occur in the database if insertion, deletion, modification etc are no done properly. It creates inconsistency and unreliability in the database.

To solve this problem, the “raw” database needs to be normalized. This is a step by step process of removing different kinds of redundancy and anomaly at each step. At each step a specific rule is followed to remove specific kind of impurity in order to give the database a slim and clean look.

Normalization

Normalization of Database

Database Normalization is a technique of organizing the data in the database. Normalization is a systematic approach of decomposing tables to eliminate data redundancy and undesirable characteristics like Insertion, Update and Deletion Anomalies. It is a multi-step process that puts data into tabular form by removing duplicated data from the relation tables.

Normalization is the process of organizing data in a database to reduce redundancy and improve data integrity. It involves decomposing large tables into smaller ones and establishing relationships between them to ensure efficient data management.

Normalization

Why Normalization of Database?

1. *Eliminates Data Redundancy*
   - Reduces duplicate data, saving storage space.
   - Ensures consistency, as data is stored only in one place.

2. *Improves Data Integrity and Consistency*
   - Avoids anomalies (insertion, update, and deletion anomalies).
   - Ensures that any change in data reflects across related tables correctly.

3. *Enhances Data Retrieval Efficiency*
   - Structured tables improve query performance.
   - Indexing becomes more efficient.

4. *Reduces Anomalies in Database Operations*
   - Insertion Anomaly: Prevents unnecessary storage of null values.
   - Update Anomaly: Ensures that updates occur at a single place.
   - Deletion Anomaly: Avoids unintentional data loss due to deletion.

5. *Maintains Data Dependencies*
   - Ensures that attributes are logically related.
   - Helps in enforcing referential integrity.

Normalization

Un-Normalized Form (UNF)

If a table contains non-atomic values at each row, it is said to be in UNF. An **atomic value** is something that can not be further decomposed. A **non-atomic value**, as the name suggests, can be further decomposed and simplified. Consider the following table:

Emp-Id	Emp-Name	Month	Sales	Bank-Id	Bank-Name
E01	AA	Jan	1000	B01	SBI
		Feb	1200		
		Mar	850		
E02	BB	Jan	2200	B02	UTI
		Feb	2500		
E03	CC	Jan	1700	B01	SBI
		Feb	1800		
		Mar	1850		
		Apr	1725		

In the sample table above, there are multiple occurrences of rows under each key Emp-Id. Although considered to be the primary key, Emp-Id cannot give us the unique identification facility for any single row. Further, each primary key points to a variable length record (3 for E01, 2 for E02 and 4 for E03).

Normalization

Problems without Normalization

If a database design is not perfect, it may contain anomalies, which are like a bad dream for any database administrator. Managing a database with anomalies is next to impossible.

Update anomalies − If data items are scattered and are not linked to each other properly, then it could lead to strange situations. For example, when we try to update one data item having its copies scattered over several places, a few instances get updated properly while a few others are left with old values. Such instances leave the database in an inconsistent state.

Deletion anomalies − We tried to delete a record, but parts of it was left undeleted because of unawareness, the data is also saved somewhere else.

Insert anomalies – We tried to insert data in a record that does not exist at all.

Normalization is a method to remove all these anomalies and bring the database to a consistent state.

Normalization

Normalization Forms

Normalization is performed in stages known as **Normal Forms (NF)**:

- **First Normal Form (1NF)** – Eliminates duplicate columns, ensures atomicity.
- **Second Normal Form (2NF)** – Removes partial dependencies.
- **Third Normal Form (3NF)** – Eliminates transitive dependencies.
- **Boyce-Codd Normal Form (BCNF)** – Strengthens 3NF to handle special cases.
- **Fourth (4NF) and Fifth Normal Form (5NF)** – Deals with multi-valued dependencies and complex relationships.

First Normal Form (1NF)

A relation is in first normal form if it meets the definition of a relation:

- Each attribute (column) value must be a single value only.
- All values for a given attribute (column) must be of the same type.
- Each attribute (column) name must be unique.
- The order of attributes (columns) is insignificant
- No two tuples (rows) in a relation can be identical.
- The order of the tuples (rows) is insignificant.

If you have a *key* defined for the relation, then you can meet the *unique row* requirement.

Normalization

First Normal Form (1NF)

A relation is said to be in 1NF if it contains no non-atomic values and each row can provide a unique combination of values. The above table in UNF can be processed to create the following table in 1NF.

Emp-Id	Emp-Name	Month	Sales	Bank-Id	Bank-Name
E01	AA	Jan	1000	B01	SBI
E01	AA	Feb	1200	B01	SBI
E01	AA	Mar	850	B01	SBI
E02	BB	Jan	2200	B02	UTI
E02	BB	Feb	2500	B02	UTI
E03	CC	Jan	1700	B01	SBI
E03	CC	Feb	1800	B01	SBI
E03	CC	Mar	1850	B01	SBI
E03	CC	Apr	1725	B01	SBI

As you can see now, each row contains unique combination of values. Unlike in UNF, this relation contains only atomic values, i.e. the rows can not be further decomposed, so the relation is now in 1NF.

Normalization

Un-Normal Form Table

Company	Symbol	Headquarters	Date	Close Price
Microsoft	MSFT	Redmond, WA	09/07/2013	23.96
			09/08/2013	23.93
			09/09/2013	24.01
Oracle	ORCL	Redwood Shores, CA	09/07/2013	24.27
			09/08/2013	24.14
			09/09/2013	24.33

Table in First Normal Form

Company	Symbol	Headquarters	Date	Close Price
Microsoft	MSFT	Redmond, WA	09/07/2013	23.96
Microsoft	MSFT	Redmond, WA	09/08/2013	23.93
Microsoft	MSFT	Redmond, WA	09/09/2013	24.01
Oracle	ORCL	Redwood Shores, CA	09/07/2013	24.27
Oracle	ORCL	Redwood Shores, CA	09/08/2013	24.14
Oracle	ORCL	Redwood Shores, CA	09/09/2013	24.33

n can be removed into another related relation, it would come to 2NF.

Normalization

Second Normal Form (2NF)

A relation is said to be in 2NF f if it is already in 1NF and each and every attribute fully depends on the primary key of the relation. Speaking inversely, if a table has some attributes which is not dependant on the primary key of that table, then it is not in 2NF.

Let us explain. Emp-Id is the primary key of the above relation. Emp-Name, Month, Sales and Bank-Name all depend upon Emp-Id. But the attribute Bank-Name depends on Bank-Id, which is not the primary key of the table. So the table is in 1NF, but not in 2NF. If this position can be removed into another related relation, it would come to 2NF.

Emp-Id	Emp-Name	Month	Sales	Bank-Id
E01	AA	JAN	1000	B01
E01	AA	FEB	1200	B01
E01	AA	MAR	850	B01
E02	BB	JAN	2200	B02
E02	BB	FEB	2500	B02
E03	CC	JAN	1700	B01
E03	CC	FEB	1800	B01
E03	CC	MAR	1850	B01
E03	CC	APR	1726	B01

Bank-Id	Bank-Name
B01	SBI
B02	UTI

After removing the portion into another relation we store lesser amount of data in two relations without any loss information. There is also a significant reduction in redundancy.

Normalization

The following example relation *is not* in 2NF:
STOCKS (Company, Symbol, Headquarters, Date, Close_Price)

To start the normalization process, list the functional dependencies (FD):

FD1: Symbol, Date → Company, Headquarters, Close Price
FD2: Symbol → Company, Headquarters

Consider that Symbol, Date → Close Price. So we might use Symbol, Date as our key.
However we also see that: Symbol → Headquarters

- This violates the rule for 2NF in that a *part of our key*. key determines a non-key attribute.
- Another name for this is a *Partial key dependency*. Symbol is only a "part" of the key and it determines a non-key attribute.
- Also, consider the insertion and deletion anomalies.

One Solution:
Split this up into two new relations:
COMPANY (Company, Symbol, Headquarters)
STOCK_PRICES (Symbol, Date, Close_Price)

Normalization

- At this point we have two new relations in our relational model. The original "STOCKS" relation we started with is removed form the model.
- Sample data and functional dependencies for the two new relations:
- COMPANY Relation:

Company	Symbol	Headquarters	Date	Close Price
Microsoft	MSFT	Redmond, WA	09/07/2013	23.96
Microsoft	MSFT	Redmond, WA	09/08/2013	23.93
Microsoft	MSFT	Redmond, WA	09/09/2013	24.01
Oracle	ORCL	Redwood Shores, CA	09/07/2013	24.27
Oracle	ORCL	Redwood Shores, CA	09/08/2013	24.14
Oracle	ORCL	Redwood Shores, CA	09/09/2013	24.33

FD1: Symbol, Date → Company, Headquarters, Close Price

Symbol	Date	Close Price
MSFT	09/07/2013	23.96
MSFT	09/08/2013	23.93
MSFT	09/09/2013	24.01
ORCL	09/07/2013	24.27
ORCL	09/08/2013	24.14
ORCL	09/09/2013	24.33

FD2: Symbol → Company, Headquarters

Company	Symbol	Headquarters
Microsoft	MSFT	Redmond, WA
Oracle	ORCL	Redwood Shores, CA

n can be removed into another related relation, it would come to 2NF.

Normalization

Third Normal Form (3NF)

A relation is in third normal form (3NF) if it is in second normal form and it contains no *transitive dependencies*.

Consider relation R containing attributes A, B and C. R(A, B, C)

If A → B and B → C then A → C

Transitive Dependency: Three attributes with the above dependencies.

Example: At CUNY:

Course_Code → Course_Number, Section

Course_Number, Section → Classroom, Professor

Consider one of the new relations we created in the STOCKS example for 2nd normal form:

The functional dependencies we can see are:

FD1: Symbol → Company

FD2: Company → Headquarters

so therefore: Symbol → Headquarters

This is a transitive dependency.

What happens if we remove Oracle?

We loose information about 2 different facts.

Company	**Symbol**	**Headquarters**
Microsoft	MSFT	Redmond, WA
Oracle	ORCL	Redwood Shores, CA

Normalization

The solution again is to split this relation up into two new relations:

STOCK_SYMBOLS(Company, Symbol)

COMPANY_HEADQUARTERS(Company, Headquarters)

This gives us the following sample data and FD for the new relations

FD1: Symbol → Company

Company	**Symbol**
Microsoft	MSFT
Oracle	ORCL

FD2: Company → Headquarters

Company	**Headquarters**
Microsoft	Redmond, WA
Oracle	Redwood Shores, CA

Normalization

Boyce-Codd Normal Form (BCNF)

- A relation is in BCNF if every determinant is a candidate key.
- Recall that not all determinants are keys.
- Those determinants that are keys we initially call *candidate keys*.
- Eventually, we select a single candidate key to be *the key* for the relation.
- Consider the following example:
  - Funds consist of one or more Investment Types.
  - Funds are managed by one or more Managers
  - Investment Types can have one more Managers
  - Managers only manage one type of investment.

Relation: FUNDS (FundID, InvestmentType, Manager)

FD1: FundID, InvestmentType → Manager

FD2: FundID, Manager → InvestmentType

FD3: Manager → InvestmentType

FundID	InvestmentType	Manager
99	Common Stock	Smith
99	Municipal Bonds	Jones
33	Common Stock	Green
22	Growth Stocks	Brown
11	Common Stock	Smith

Normalization

- In this case, the combination FundID and InvestmentType form a *candidate key* because we can use FundID,InvestmentType to uniquely identify a tuple in the relation.
- Similarly, the combination FundID and Manager also form a *candidate key* because we can use FundID, Manager to uniquely identify a tuple.
- Manager by itself is not a candidate key because we cannot use Manager alone to uniquely identify a tuple in the relation.
- Is this relation FUNDS(FundID, InvestmentType, Manager) in 1NF, 2NF or 3NF ? Given we pick FundID, InvestmentType as the *Primary Key:*
  - ✓ 1NF for sure.
  - ✓ 2NF because all of the non-key attributes (Manager) is dependant on all of the key.
  - ✓ 3NF because there are no transitive dependencies.

- However consider what happens if we delete the tuple with FundID 22. We loose the fact that Brown manages the InvestmentType “Growth Stocks.”

Normalization

- Therefore, while FUNDS relation is in 1NF, 2NF and 3NF, it is in BCNF because not all determinants (Manager in FD3) are candidate keys.
- The following are steps to normalize a relation into BCNF:
  - ✓ List all of the determinants.
  - ✓ See if each determinant can act as a key (candidate keys).
  - ✓ For any determinant that is *not* a candidate key, create a new relation from the functional dependency. Retain the determinant in the original relation.
- For our example:FUNDS (FundID, InvestmentType, Manager)
- The determinants are:FundID, InvestmentType FundID, Manager Manager
- Which determinants can act as keys?

  FundID, InvestmentType *YES*

  FundID, Manager *YES*

  Manager *NO*
- Create a new relation from the functional dependency:

  MANAGERS(Manager, InvestmentType),

  FUND_MANAGERS(FundID, Manager)

In this last step, we have retained the determinant "Manager" in the original relation MANAGERS.Each of the new relations should be checked to ensure they meet the definitions of 1NF, 2NF, 3NF and BCNF

Normalization

- For our example:FUNDS (FundID, InvestmentType, Manager)
- The determinants are:FundID, InvestmentType FundID, Manager Manager
- Which determinants can act as keys?

  FundID, InvestmentType *YES*

  FundID, Manager *YES*

  Manager *NO*
- Create a new relation from the functional dependency:

  MANAGERS(Manager, InvestmentType),

  FUND_MANAGERS(FundID, Manager)

In this last step, we have retained the determinant "Manager" in the original relation MANAGERS. Each of the new relations should be checked to ensure they meet the definitions of 1NF, 2NF, 3NF and BCNF

FundID	Manager
99	Smith
99	Jones
33	Green
22	Brown
11	Smith

InvestmentType	Manager
Common Stock	Smith
Municipal Bonds	Jones
Common Stock	Green
Growth Stocks	Brown
Common Stock	Smith

Normalization

Fourth Normal Form (4NF)

A relation is in fourth normal form if it is in BCNF and it contains no *multivalued dependencies*.

Multivalued Dependency: A type of functional dependency where the determinant can determine more than one value.

More formally, there are 3 criteria:

- There must be at least 3 attributes in the relation. call them A, B, and C, for example.
- Given A, one can determine multiple values of B.
- Given A, one can determine multiple values of C.
- B and C are independent of one another.

Normalization

Example (Before 4NF - Table with Multi-Valued Dependency)

Consider a **STUDENT_SKILLS_PROJECTS** table where a student can have multiple skills and work on multiple projects:

Student_ID	Skill	Project
101	Python	AI Model
101	Java	AI Model
101	Python	Web App
101	Java	Web App
102	C++	IoT System
102	C	IoT System

Problem: Multi-Valued Dependency (MVD)

- A **student's skills** are independent of the **projects** they work on.
- **(Student_ID →→ Skill)** and **(Student_ID →→ Project)** are two independent multi-valued dependencies.
- This causes **redundancy** because every combination is stored multiple times.

Normalization

Example (After Applying 4NF - Removing Multi-Valued Dependency)
We split the table into two:

Table 1: STUDENT_SKILLS Table (Stores student skills)

Student_ID	Skill
101	Python
101	Java
102	C++
102	C

Table 2: STUDENT_PROJECTS Table (Stores student projects)

Student_ID	Project
101	AI Model
101	Web App
102	IoT System

- **Eliminates multi-valued dependencies** by separating independent data.
- **Reduces redundancy and anomalies** in data storage.
- **Ensures that attributes depend only on the primary key.**

Normalization

A few characteristics:

- No regular functional dependencies
- All three attributes taken together form the key.
- Later two attributes are independent of one another.
- Insertion anomaly: Cannot add a stock fund without adding a bond fund (NULL Value). Must always maintain the combinations to preserve the meaning.

Stock Fund and Bond Fund form a multivalued dependency on Portfolio ID.

PortfolioID →→ Stock Fund

PortfolioID →→ Bond Fund

Portfolio ID	Stock Fund	Bond Fund
999	Janus Fund	Municipal Bonds
999	Janus Fund	Dreyfus Short-Intermediate Municipal Bond Fund
999	Scudder Global Fund	Municipal Bonds
999	Scudder Global Fund	Dreyfus Short-Intermediate Municipal Bond Fund
888	Kaufmann Fund	T. Rowe Price Emerging Markets Bond Fund

Normalization

Resolution: Split into two tables with the common key:

Portfolio ID	Stock Fund
999	Janus Fund
999	Scudder Global Fund
888	Kaufmann Fund

Portfolio ID	Bond Fund
999	Municipal Bonds
999	Dreyfus Short-Intermediate Municipal Bond Fund
888	T. Rowe Price Emerging Markets Bond Fund

Normalization

Fifth Normal Form (5NF)

- Also called “Projection Join” Normal form.

A table is in **Fifth Normal Form (5NF)** if:

- It is already in **Fourth Normal Form (4NF)**.
- It does **not have join dependencies** that lead to **lossless decomposition** (i.e., a table should not be broken into smaller tables unless it ensures correct data reconstruction when joined back).

❖ **5NF focuses on eliminating redundancy caused by complex relationships between multiple entities.**

Normalization

Example (Before 5NF - Table with Join Dependency)

Consider a **STUDENT_COURSE_PROFESSOR** table where:

- A student can enroll in multiple courses.
- A professor can teach multiple courses.
- A student can learn from multiple professors.

Student_ID	Course_ID	Professor_ID
101	C01	P01
101	C02	P02
102	C01	P01
102	C03	P03

Issue (Join Dependency):

- This table is **not in 5NF** because there is a **many-to-many relationship** between Students, Courses, and Professors.

- If a professor no longer teaches a course, deleting a row might remove information about the student's enrollment, leading to **data loss**.

Normalization

Example (After Applying 5NF - Removing Join Dependency)

We decompose the table into three smaller tables:

STUDENT_COURSE Table

Course_ID	Professor_ID
C01	P01
C02	P02
C03	P03

COURSE_PROFESSOR Table

Student_ID	Course_ID
101	C01
101	C02
102	C01
102	C03

STUDENT_PROFESSOR Table

Student_ID	Professor_ID
101	P01
101	P02
102	P01
102	P03

- **Eliminates redundancy by breaking down complex many-to-many relationships.**
- **Ensures lossless join decomposition** (We can reconstruct the original table by joining these tables).
- **Avoids unnecessary duplication of data.**

Normalization

What is Lossless Decomposition?

Lossless decomposition ensures that when a table is split into smaller tables, no data is lost, and we can reconstruct the original table by joining the decomposed tables.

A decomposition is lossless if the natural join of the decomposed tables results in the original table without any extra or missing tuples.

To verify that this decomposition is lossless, we join the decomposed tables:

Performing Natural Joins

- Join STUDENT_COURSE and COURSE_PROFESSOR on Course_ID:
  - This gives us a table with (Student_ID, Course_ID, Professor_ID), but it might have extra rows.
- Join the result with STUDENT_PROFESSOR on (Student_ID, Professor_ID).

If we get back the original STUDENT_COURSE_PROFESSOR table exactly as it was, the decomposition is lossless.

Since our decomposition maintains all relationships and allows perfect reconstruction of the original table, this decomposition is lossless.

Minimal/Canonical Cover (Fc) of FDs

Minimal Cover/Canonical Cover (also called **Minimal Basis**) of a set of **functional dependencies (FDs)** is an **equivalent** set of FDs that is:

- **Minimal** – It has the smallest number of FDs.
- **Canonical** – The right-hand side of each FD contains **only one attribute**.
- **Irredundant** – No FD is unnecessary (i.e., no FD can be derived from others).

Steps to Find Minimal Cover

To find the minimal cover for a set of functional dependencies, follow these steps:

- **Make RHS Atomic** – Ensure that each FD has a **single attribute** on the right-hand side.
- **Remove Extraneous Attributes** – Eliminate unnecessary attributes from the left-hand side.
- **Remove Redundant FDs** – Delete redundant functional dependencies.

Minimal/Canonical Cover (Fc) of FDs

Example 1

Consider a relation R(A, B, C, D) with the following functional dependencies:

$F=\{A \rightarrow BC, B \rightarrow C, A \rightarrow B, AB \rightarrow C\}$

Step 1: Make RHS Atomic

Each FD should have only one attribute on the right-hand side.

A → BC can be split into:

A → B

A → C

Now, our set becomes:

$F=\{A \rightarrow B, A \rightarrow C, B \rightarrow C, A \rightarrow B, AB \rightarrow C\}$

(Remove duplicate A → B)$F=\{A \rightarrow B, A \rightarrow C, B \rightarrow C, AB \rightarrow C\}$

Minimal/Canonical Cover (Fc) of FDs

Step 2: Remove Extraneous Attributes

A left-side attribute is extraneous if removing it does not change the closure of the set.

Check if **AB → C** can be simplified:

Compute A+ (closure of A):

A → B (so, **A+ = {A, B}**)

B → C (so, **A+ = {A, B, C}**)

Since A+ already contains C, AB → C is redundant.

Remove AB → C, leaving: $F=\{A \rightarrow B, A \rightarrow C, B \rightarrow C\}$

Step 3: Remove Redundant FDs

An FD **X → Y** is redundant if **Y** can be derived from other FDs.

Check if **A → C** is redundant:

Compute **A+** using **A → B** and **B → C**:

A → B (so, **A+ = {A, B}**)

B → C (so, **A+ = {A, B, C}**)

Since **C** is already in **A+**, **A → C** is redundant.

Remove **A → C**, leaving:

F_{min}={A→B,B→C}

Minimal/Canonical Cover (Fc) of FDs

Example 2

Consider a relation R(X, Y, Z, W, V) with the following functional dependencies:

F={XZ→YW, Y→W, X→Y, XW→V}

Find the **Minimal Cover F_{min}** by following these steps:

1. Make RHS atomic (Each FD should have only one attribute on the right-hand side).
2. Remove extraneous attributes from LHS.
3. Remove redundant FDs to get the minimal cover.

Step 1: Make RHS Atomic

Each FD should have only one attribute on the right-hand side.

XZ → YW can be split into:

XZ → Y

XZ → W

Now, the new set of **F** is:

F={XZ→Y, XZ→W, Y→W, X→Y, XW→V}

Now, all FDs have atomic RHS.

Minimal/Canonical Cover (Fc) of FDs

Step 2: Remove Extraneous Attributes from LHS

Now, check if any attribute in LHS is unnecessary.

Check if Z is extraneous in XZ → Y:

Compute $\mathbf{X^+}$ (closure of X) using remaining FDs:

X → Y (Given)

Y → W (Given)

$\mathbf{X^+ = \{X, Y, W\}}$

Since X alone can determine Y, **XZ → Y** can be reduced to **X → Y**.

Updated FDs after removal:

F={X→Y, XZ→W, Y→W, XW→V}

Check if Z is extraneous in XZ → W:

Compute $\mathbf{X^+}$:

X → Y

Y → W

$\mathbf{X^+ = \{X, Y, W\}}$

Since X alone can determine W, **XZ → W** is redundant and can be removed.

Updated FDs after removal:

F={X→Y,Y→W,XW→V}

Now, no extraneous attributes in LHS.

Minimal/Canonical Cover (Fc) of FDs

Step 3: Remove Redundant FDs

Check if X → Y is redundant:

We need **Y** to determine **W**, so **X → Y** is required.

Not redundant.

Check if Y → W is redundant:

If we remove **Y → W**, then we can't derive W from X.

Not redundant.

Check if XW → V is redundant:

If we remove **XW → V**, then V cannot be derived.

Not redundant.

No redundant FDs left.

Final Minimal Cover F_{min}

F_{min}={X→Y, Y→W, XW→V}

Dangling Tuple

A dangling tuple in DBMS refers to a tuple (row) in a database that does not have a valid reference in another related table due to referential integrity violations. This typically happens in foreign key constraints, where a foreign key in one table refers to a primary key in another table, but the referenced primary key does not exist.

Causes of Dangling Tuples:

- **Deletion of a Referenced Tuple** – If a referenced row (primary key) in the parent table is deleted, but the foreign key in the child table still refers to it.
- **Insertion of an Invalid Foreign Key** – If a tuple with a foreign key is inserted into a table, but the referenced primary key does not exist in the parent table.
- **Update Anomalies** – If the value of the referenced primary key is updated, making existing foreign key references invalid.

Dangling Tuple

Example:

Parent Table (Department)

Dept_ID (Primary Key)	Dept_Name
101	CSE
102	ECE

Child Table (Student)

Student_ID	Name	Dept_ID (Foreign Key)
1	Alex	101
2	Bob	103

In the above example, the Student table has a **dangling tuple** because **Dept_ID = 103 does not exist** in the Department table.

Dangling Tuple

Causes of Dangling Tuples

1. **Deletion of the referenced row** in the parent table (DELETE FROM Departments WHERE Dept_ID = 101;).

2. **Insertion of an invalid foreign key** in the child table (INSERT INTO Students VALUES (3, 'Charlie', 104); where 104 does not exist).

3. **Updating the primary key** in the parent table, making foreign key references invalid.

Prevention Methods:

1. **ON DELETE CASCADE** – Automatically deletes dependent rows when the referenced row is deleted.
2. **ON DELETE SET NULL** – Sets foreign key values to NULL if the referenced row is deleted.
3. **Foreign Key Constraints** – Ensure referential integrity by preventing invalid insertions.
4. **Triggers** – Custom database triggers can be used to handle integrity checks.

Canonical Cover (Fc) of FDs

Consider two sets of FDs

F = { A→B, AB →C, D→AC, D→E}

G= { A→BC, D→AB}

Which one is true?

a) F covers G

b) G covers F

c) F & G are equal

d) None

Let us consider first set of FDs F = { A→B, AB →C, D→AC, D→E}

1. Here given D→AC apply decompose rule D→A, D→C
2. After this FDs are F = { A→B, AB →C, D→A, D→C, D→E}
   Only one FD is a partial FD AB →C where B is extraneous. Because, A+ = AB & B+ = B.

Canonical Cover (Fc) of FDs

3. Now find out redundant FDs

F = { A→B, A →C, D→A, D→C, D→E}

redundant FD can be find out by finding closure of each attribute set of FD, without any help of that FD.

for example:

A→B For this FD calculate A+ and try to derive B from A without help of A→B is it possible? No, then it is non-redundant.

A+ = AC

A→C For this FD calculate A+ and try to derive C from A without help of A→C is it possible? No, then it is non-redundant.

A+ = AB

D→A For this FD calculate D+ and try to derive A from D without help of D→A is it possible? No, then it is non-redundant.

D+ = DCE

D→C For this FD calculate D+ and try to derive C from D without help of D→C is it possible? **Yes, then it is redundant.**

D+ = DAEBC

D→E For this FD calculate D+ and try to derive E from D without help of D→E is it possible? No, then it is non-redundant.

D+ = DACB

Canonical Cover (Fc) of FDs

Hence Fc for given FD set F = { A→B, A →C, D→A, D→C, D→E} is

Fc = { A→B, A →C, D→A, D→E}

Similarly, we'll consider G = { A→BC, D→AB}

1. Here given A→BC, D→AB apply decompose rule A→B, A →C, D→A, D→B
2. After this FDs are G = {A→B, A →C, D→A, D→B}
   Here, No FD has extraneous attribute. Follow third step
3. Now find out redundant FDs
   G = {A→B, A →C, D→A, D→B}
   A→B For this FD calculate A+ and try to derive B from A without help of A→B is it possible? No, then it is non-redundant.
   A+ = AC
   A→C For this FD calculate A+ and try to derive C from A without help of A→C is it possible? No, then it is non-redundant.
   A+ = AB
   D→A For this FD calculate D+ and try to derive A from D without help of D→A is it possible? No, then it is non-redundant.
   D+ = DB
   D→ B For this FD calculate D+ and try to derive C from D without help of D→B is it possible? **Yes, then it is redundant.**
   D+ = DABC

Canonical Cover (Fc) of FDs

Hence Gc for given FD set G = { A→B, A →C, D→A, D→B} is

Gc = { A→B, A →C, D→A}

Now, Compare Fc and Gc

Fc = { A→B, A →C, D→A, D→E}

Gc = { A→B, A →C, D→A}

a) F covers G
b) G covers F
c) F & G are equal
d) None

Answer is d) None

Relational Database Design

The **algorithm for relational database design** involves a series of steps to ensure that the database is well-structured, follows normalization principles, and maintains **data integrity**. The goal is to minimize redundancy, prevent anomalies, and optimize query performance.

Algorithm for Relational Database Design

Step 1: Requirement Analysis

- Identify the data that needs to be stored.
- Determine the relationships among different data entities.
- Understand business rules and constraints.

Step 2: Construct an Entity-Relationship (ER) Model

- Identify **entities** and **attributes**.
- Define **primary keys** for each entity.
- Establish **relationships** (one-to-one, one-to-many, many-to-many).
- Draw an **ER diagram** to visualize data organization.

Step 3: Convert ER Model to Relational Schema

- Convert **entities** into tables.
- Convert **relationships** into foreign key constraints.
- Handle **many-to-many** relationships using bridge tables.

Relational Database Design

Step 4: Define Functional Dependencies

- Identify **functional dependencies** (FDs) among attributes.
- Determine how attributes depend on primary keys.

Step 5: Normalize the Database

- **First Normal Form (1NF)** – Remove duplicate columns and ensure atomicity.
- **Second Normal Form (2NF)** – Ensure no partial dependency on primary keys.
- **Third Normal Form (3NF)** – Ensure no transitive dependencies.
- **Boyce-Codd Normal Form (BCNF)** – Strengthen 3NF if necessary.
- Further normalization (**4NF, 5NF**) if required.

Step 6: Define Integrity Constraints

- **Primary Key Constraint** – Ensure unique identification.
- **Foreign Key Constraint** – Enforce referential integrity.
- **Unique, Not Null, and Check Constraints** – Enforce business rules.

Relational Database Design

Step 7: Indexing and Query Optimization

- Create **indexes** on frequently searched columns.
- Use **denormalization** if necessary for performance tuning.
- Optimize SQL queries for faster execution.

Step 8: Implement the Database

- Use SQL commands (CREATE TABLE, ALTER TABLE, etc.) to implement the schema.
- Load initial data into tables.

Step 9: Test and Validate

- Insert sample data and test constraints.
- Check for anomalies in insertion, deletion, and updates.
- Verify data integrity and consistency.

DATABASE TRANSACTION PROCESSING, CONCURRENCY CONTROL & RECOVERY SYSTEM

Transaction Processing

- **A transaction is a logical unit of database processing that includes one or more database access operations such as insertion, deletion, modification and retrieval.**
- **Database operations that form a transaction can either be embedded within an application program or they can be specified interactively through a high level query language such as SQL, where it is delimited by statements of the form *begin transaction* and *end transaction*. All database operations between these two points are considered as one transaction.**

e.g.

begin transaction

…

…

…

end transaction

One Transaction

Transaction Processing Contd…

- A Single application program may contain more than one transaction it it contains several boundaries.
- Transactions access data using two operations-
  - Read (X) – It transfers the data items X from the database to a local buffer belonging to the transaction that executed the read operation.
  - Write(X) – It transfers the data item X from the local buffer of the transaction that executed the write back of the database.
- Example of two very simple transactions.

T1	T2

read (X);
X := X-N;
write(X);
read (Y);
Y := Y+N;
write(Y);
	read (X);
	X := X+M;
	write(X);

Transaction T1	Transaction T2

Transaction Properties

- **The database system maintains the main four properties of the transactions to ensure integrity of the data.**

  i. **Atomicity – A Transaction is an atomic unit of processing. It is either performed in its entirely or not performed at all.**

  ii. **Consistency – Transaction preserve database consistency. That is, a transaction transform a consistent state of the database into another consistent state, without necessarily preserving consistency at all intermediate points.**

  iii. **Isolation – A Transaction should appear as though it is being executed in isolation from other transactions. That is, the execution of a transaction should not be interfered with by any other transactions executing concurrently.**

  iv. **Durability or Permanency – The changes applied to the database by a committed transaction must persist in the database. These changes must not lost because of any failure.**

These properties are known as the ACID properties.

Transaction Properties

For Example: Consider a banking system consisting of several accounts and a set of Transactions that access and update those accounts to understand and need of ACID properties.

Let Ti be a transaction that transfer amount Rs. 500 from account A to account B. This transaction can be defined as –

Ti:

1. **read (A);**
2. **A := A-500;**
3. **write (A);**
4. **read (B);**
5. **B := B+500;**
6. **write (B);**

Now, each of the ACID requirement is considered as follows-

Transaction Properties

1. ***Consistency:*** **The consistency requirement here is that the sum of A & B be unchanged by the execution of the transaction. Without the consistency requirement, money can be created or destroyed by the transaction.**

   Ensuring consistency for an individual transaction is the responsibility of the application programmer. This task may be facilitated by automatic testing of integrity constraints.

2. ***Atomicity:*** **Suppose that, just before the execution of transaction Ti the values of A and B are 1000 and 2000. Now, suppose that the execution of transaction Ti, a failure happened after the write(A) operation but before the write (B) operation due to power failures, hardware failures and software failures. In this case the values of A and B reflected in the database are 950 and 2000. These system destroyed 50 as a result of this failure. In this case, the sum A+B is no longer preserved.**

   Thus, the system reaches to an inconsistent state. We must ensure that such inconsistency are not visible in database system. However, this state is eventually replaced by the consistent state, where the value of account A is 950 and the value of account B is 2050. That is the reason for the atomicity requirement- if the atomicity property is present, all actions of transaction are reflected in the database, or none are. (Transaction-Management Task)

Transaction Properties

3. ***Durability:*** **Once the user has been notified that the transaction has completed (i.e. the transfer of 50 has taken place), the update to the database by the transaction must persist despite failures.**

   Thus the durability property guarantee that once a transaction completes successfully, all the updates that carried out on the database persist, even if there is a system failure after the transaction completes execution.

   (Recovery-Management Task)

4. ***Isolation:*** **If between states 3 & 6, another transaction is allowed to access the partially updated database, it will see an inconsistent database (the sum A + B will be loss then it should be).**

   The problem concurrently executing transactions are avoided by executing transactions serially i. e. one after the other. However, executing multiple transactions concurrently has significant benefits. (Concurrency-Control Task)

Transaction States

A Transaction must be in one of the following states-

(i) **Active:** It's initial state; the transaction stays in this state while it is executing.

(ii) **Partially Committed:** After the final statement has been executed.

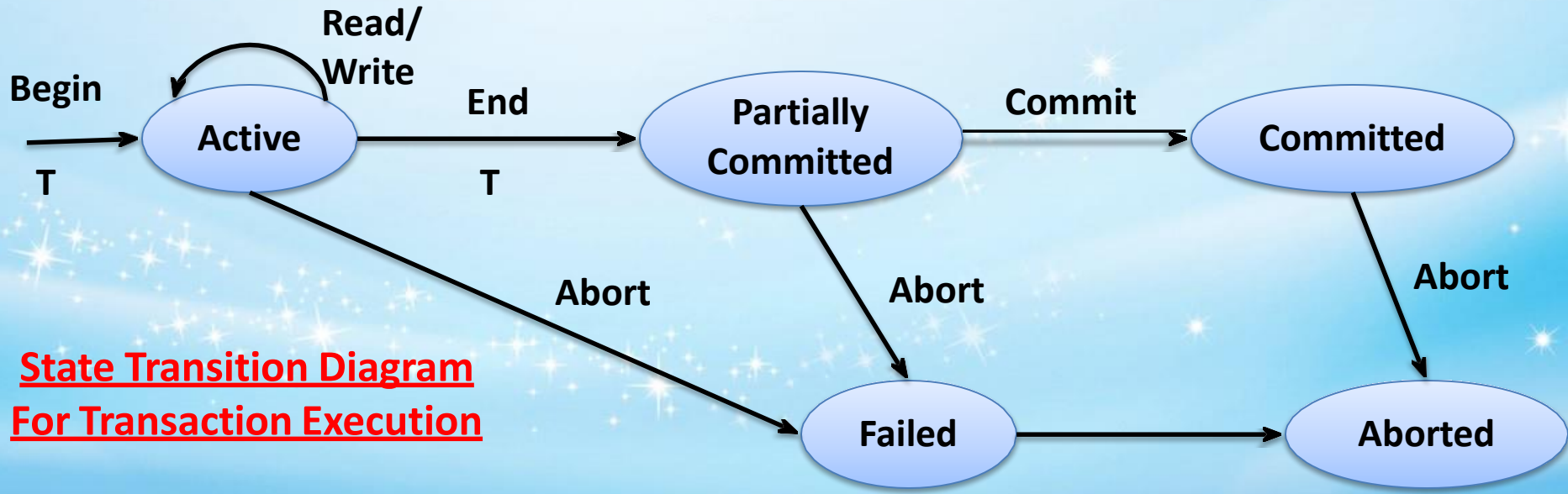
(iii) **Failed:** After the discovery that normal execution can no longer proceed.

(iv) **Aborted:** After the transaction has been called back and the database restored to its state prior to the start of the transaction. Two options are available after the transaction has been aborted:

a) Restart the transaction, only if there is no internal logical error.

b) Kill the transaction.

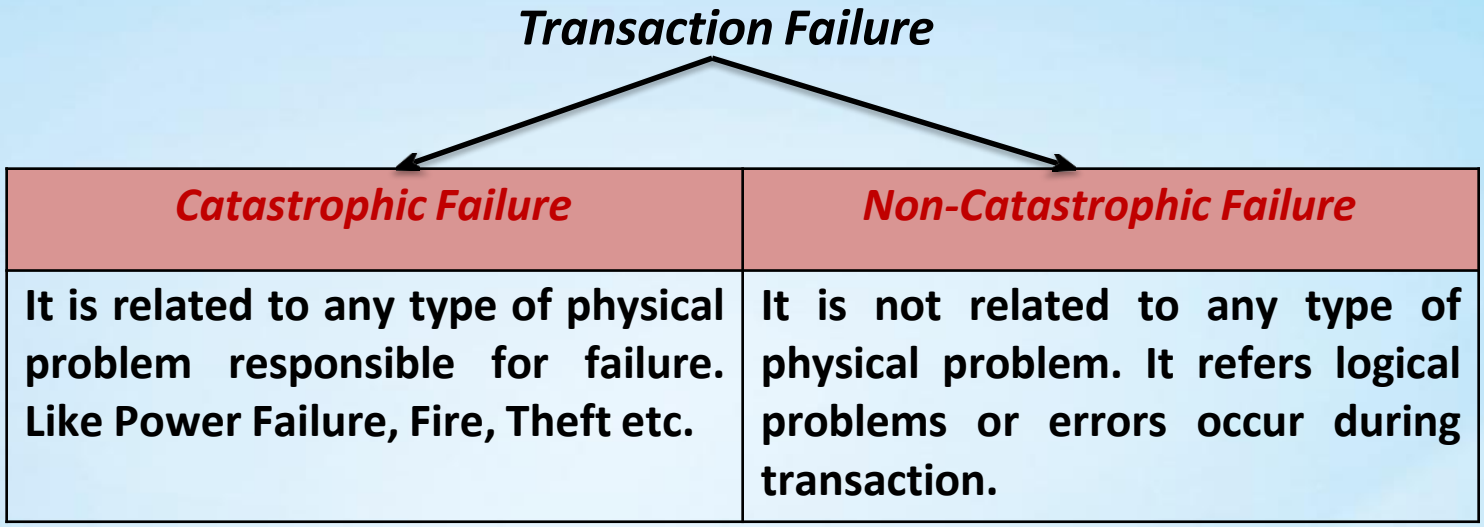
(v) **Committed:** After successful completion.



State Transition Diagram
For Transaction Execution

Transaction Failure

A transaction may be failed due to many different reason during its execution time. A transaction can be aborted by several reasons from its any state of execution. There are mainly two different types of failure.



There are many problems for transaction failure:

1. **System Crash:** *It refers to the hardware, software or network error occurs in the system during execution. Hardware crashes are usually media failure like Hard sick, RAM, Processor etc.*
2. **System Error:** *System errors are the errors which are occurred during execution of transaction. These errors are division by zero, integer overflow, buffer overflow etc. These errors occurs due to logical programming errors.*

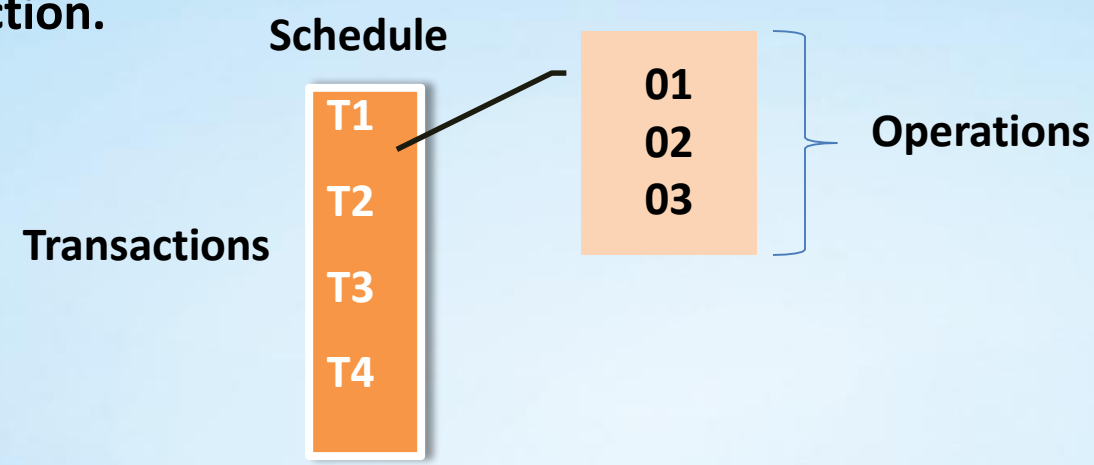
Transaction Failure…

3. ***Exception or errors in transaction:*** *Some errors are occurred during execution of transaction which can not be handled by transactions known as Exception. These exceptions can be programmed in the transaction to overcome the failure. The exceptions are:*
   - *a) Data not found for transaction*
   - *b) Insufficient Account Balance*
   - *c) Illegal Operation Performed*

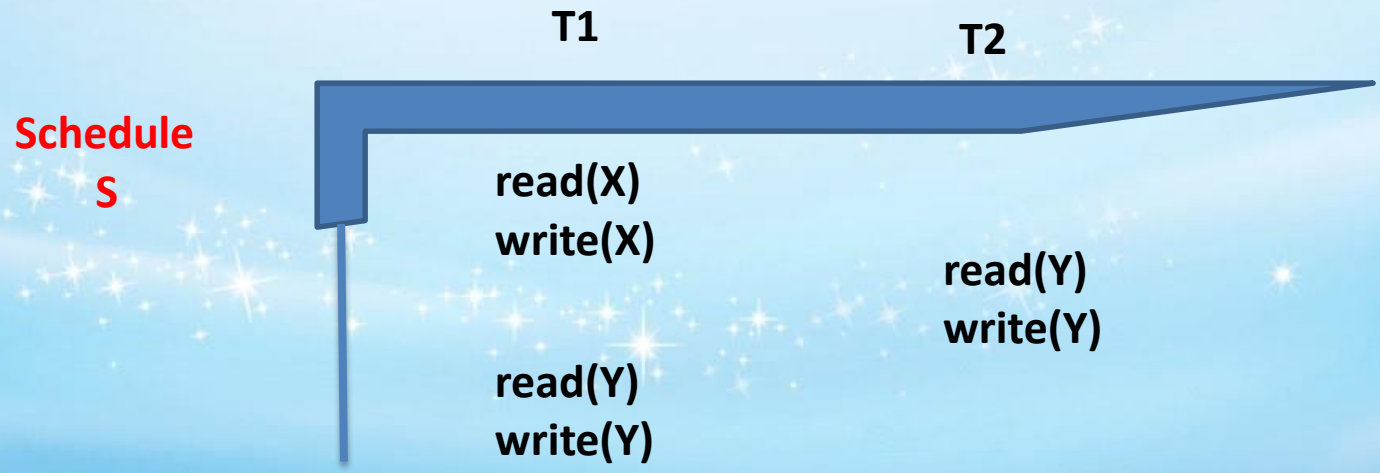
4. ***Catastrophic Failure or Physical Problems:*** *Physical Problems are endless list of problems like power failure, fire, theft, natural calamity, overwriting disk etc. Such problems do not happen frequently; if they occur, recovery is a major tack.*

Transaction Schedule

Schedule- A Schedule is an ordering sequence of the operations of the transactions. A schedule should follow the constraint that the sequence of operations for each transactions should be same as they appear in the transaction.



In a schedule, sequence of operations of the transaction should not be changed if two transactions are interleaved in the process of execution.



Complete Schedule

A Schedule is n transactions T1, T2, T3 …… Tn is said to be a complete schedule if the following conditions exists-

i) **The operations in S are exactly those operations in T1, T2, T3……Tn, including a commit or abort operationas the last operation of each transaction in the schedule.**

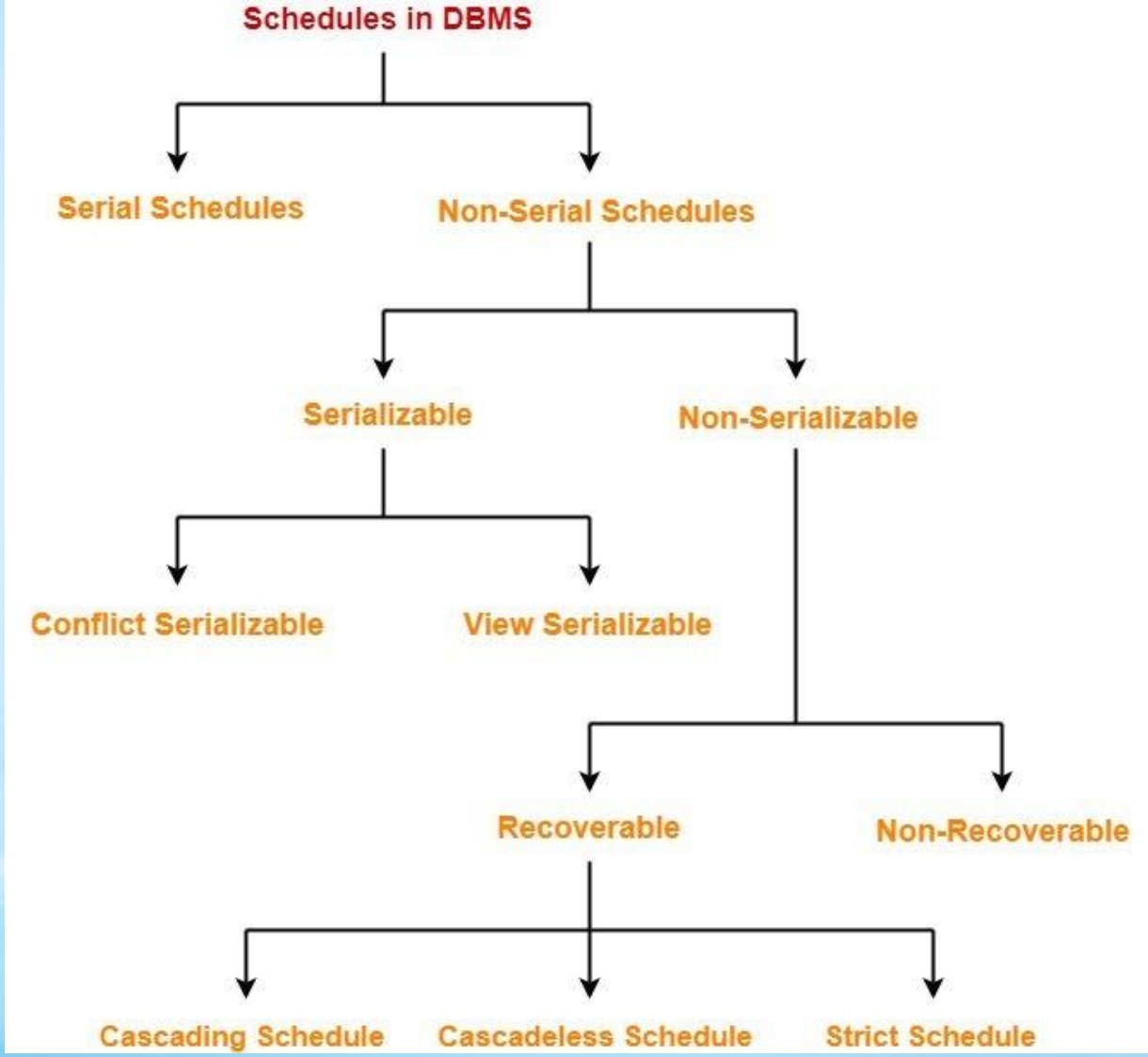
ii) **For any pair of operations from the same transaction Ti, their order of appearance in S is the same as their order of appearance in Ti.**

iii) **For any two conflicting operations, one of the two must occur before the other in the schedule.**

$$D = \begin{bmatrix} T1 & T2 & T3 \\ R(X) & & \\ W(X) & & \\ Com. & & \\ & R(Y) & \\ & W(Y) & \\ & Com. & \\ & & R(Z) \\ & & W(Z) \\ & & Com. \end{bmatrix}$$

Complete Schedule

The order in which the operations of multiple transactions appear for execution is called as a schedule.



Serial Schedule

A Schedule is called Serial, if the operations of each transaction are executed consecutively without any interleaved operations from other transaction.

T1	T2

read (X);
X := X-N;
write(X);
read (Y);
Y := Y+N;
write(Y);	

	read (X); X := X+M; write(X);

Serial Schedule

A Schedule S is Serial, if for every transaction T participating in the schedule, all the operations of T are executed consecutively. Otherwise the schedule is called non-serial. Hence in a serial schedule, only one transaction at a time is active and no interleaving occurs.

Serial Schedule

Serial schedules are always-

- **Consistent**
- **Recoverable**
- **Cascadeless**
- **Strict**

Transaction T1	Transaction T2
R (A)	
W (A)	
R (B)	
W (B)	
Commit	
	R (A)
	W (B)
	Commit

In this schedule,

1. There are two transactions T1 and T2 executing serially one after the other.
2. Transaction T1 executes first.
3. After T1 completes its execution, transaction T2 executes.

So, this schedule is an example of a Serial Schedule.

Non-serial Schedule

A Schedule is called Non-serial, if each sequence interleaves operations from two transactions.

T1	T2
read (X); X := X-N; write(X);	
	read (X); X := X+M; write(X);

read(Y);
Y := Y+N;
write(Y);	

Non-Serial Schedule

Time

Interleaving of transactions T1 and T2

Non-serial schedules are NOT always-

- **Consistent**
- **Recoverable**
- **Cascadeless**
- **Strict**

Non-serial Schedule

Transaction T1	Transaction T2
R (A)	
W (B)	
	R (A)
R (B)	
W (B)	
Commit	
	R (B)
	Commit

In this schedule,

- There are two transactions T1 and T2 executing concurrently.
- The operations of T1 and T2 are interleaved.

So, this schedule is an example of a **Non-Serial Schedule**.

In this schedule,

- There are two transactions T1 and T2 executing concurrently.
- The operations of T1 and T2 are interleaved.

So, this schedule is an example of a **Non-Serial Schedule.**

Transaction T1	Transaction T2
	R (A)
R (A)	
W (B)	
	R (B)
	Commit
R (B)	
W (B)	
Commit	

Finding Number of Schedules

Finding Number Of Schedules

Consider there are n number of transactions T1, T2, T3 …. , Tn with N1, N2, N3 …. , Nn number of operations respectively.

Total Number of Schedules

Total number of possible schedules (serial + non-serial) is given by-

$$\frac{(N1 + N2 + N3 + ... + Nn)!}{N1! \times N2! \times N3! \times .... \times Nn!}$$

Total Number of Serial Schedules

Total number of serial schedules =

Number of different ways of arranging n transactions = n!

Total Number of Non-Serial Schedules

Total number of non-serial schedules =

Total number of schedules – Total number of serial schedules

Finding Number of Schedules

Problem

Consider there are three transactions with 2, 3, 4 operations respectively, find-
How many total number of schedules are possible?
How many total number of serial schedules are possible?
How many total number of non-serial schedules are possible?

Total Number of Schedules

Using the above formula, we have

$$\text{Total number of schedules} = \frac{(2+3+4)!}{2! \times 3! \times 4!}$$

$$= 1260$$

Total Number of Serial Schedules

Total number of serial schedules
= Number of different ways of arranging 3 transactions
= 3!
= 6

Total Number of Non-Serial Schedules

Total number of non-serial schedules
= Total number of schedules – Total number of serial schedules
= 1260 – 6
= 1254

Serializability

The concept of Serializability of schedules is used to identify which schedules are correct when the transaction executions have interleaving of their operations in the schedules.

A Schedule S of n transactions is serializable if it is equivalent to some serial schedule of the same n transactions. Each serial schedule consists of a sequence of instructions from the various transactions, where the instructions belonging to one single transaction appear together in that schedule.

A non-serial schedule S is serializable is equivalent to say that it is correct schedule, because it is same as a serial schedule.

Serializability

Serializability

- Some non-serial schedules may lead to inconsistency of the database.
- Serializability is a concept that helps to identify which non-serial schedules are correct and will maintain the consistency of the database.

Serializable Schedules

If a given non-serial schedule of ‘n’ transactions is equivalent to some serial schedule of ‘n’ transactions, then it is called as a **serializable schedule**.

Serializable schedules behave exactly same as serial schedules.

Thus, serializable schedules are always-

- Consistent
- Recoverable
- Cascadeless
- Strict

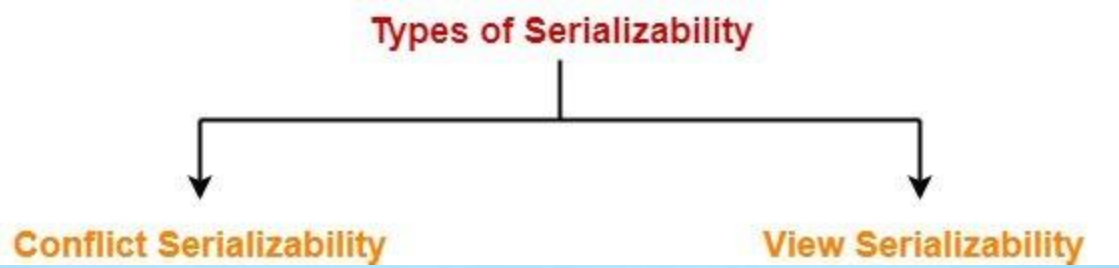
Serial Schedules Vs Serializable Schedules

Serial Schedules	Serializable Schedules
No concurrency is allowed. Thus, all the transactions necessarily execute serially one after the other.	Concurrency is allowed. Thus, multiple transactions can execute concurrently.
Serial schedules lead to less resource utilization and CPU throughput.	Serializable schedules improve both resource utilization and CPU throughput.
Serial Schedules are less efficient as compared to serializable schedules.	Serializable Schedules are always better than serial schedules.

Types of Serializability

Serializability is mainly of two types-

1. Conflict Serializability
2. View Serializability



Conflict Serializability

Conflict Serializability-

If a given non-serial schedule can be converted into a serial schedule by swapping its non-conflicting operations, then it is called as a **conflict serializable schedule**.

Conflicting Operations-

Two operations are called as **conflicting operations** if all the following conditions hold true for them-

1. Both the operations belong to different transactions
2. Both the operations are on the same data item
3. At least one of the two operations is a write operation

Example-

Consider the following schedule-

In this schedule,

- W1 (A) and R2 (A) are called as conflicting operations.

This is because all the above conditions hold true for them.

Transaction T1	Transaction T2
R1 (A)	
W1 (A)	
	R2 (A)
R1 (B)	

Conflict Serializability

Follow the following steps to check whether a given non-serial schedule is conflict serializable or not-

Step-01:

Find and list all the conflicting operations.

Step-02:

Start creating a precedence graph by drawing one node for each transaction.

Step-03:

Draw an edge for each conflict pair such that if X_i (V) and Y_j (V) forms a conflict pair then draw an edge from T_i to T_j.

This ensures that T_i gets executed before T_j.

Step-04:

Check if there is any cycle formed in the graph.

If there is no cycle found, then the schedule is conflict serializable otherwise not.

Conflict Serializability

Problem-01:

Check whether the given schedule S is conflict serializable or not-

S : $R_1(A)$, $R_2(A)$, $R_1(B)$, $R_2(B)$, $R_3(B)$, $W_1(A)$, $W_2(B)$

T1	T2	T3
R1 (A)		
	R2 (A)	
R1 (B)		
	R2 (B)	
		R3 (B)
W1(A)		
	W2(B)	

Solution

Step-01:

List all the conflicting operations and determine the dependency between the transactions-

$R_2(A)$, $W_1(A)$ $(T_2 \rightarrow T_1)$

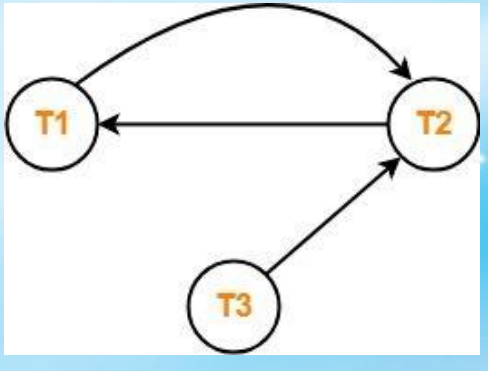
$R_1(B)$, $W_2(B)$ $(T_1 \rightarrow T_2)$

$R_3(B)$, $W_2(B)$ $(T_3 \rightarrow T_2)$

Step-02:

Draw the precedence graph-

- Clearly, there exists a cycle in the precedence graph.
- Therefore, the given schedule S is not conflict serializable.



Conflict Serializability

Problem-02:

Check whether the given schedule S is conflict serializable and recoverable or not-

T1	T2	T3	T4
	R(X)		
		W(X)	
		Commit	
W(X)			
Commit			
	W(Y)		
	R(Z)		
	Commit		
			R(X)
			R(Y)
			Commit

Checking Whether S is Conflict Serializable Or Not-

Step-01:

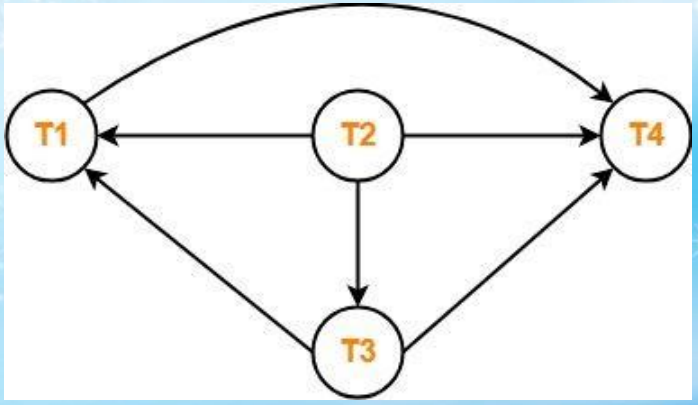
List all the conflicting operations and determine the depe

$R_2(X)$, $W_3(X)$ $(T_2 \rightarrow T_3)$
$R_2(X)$, $W_1(X)$ $(T_2 \rightarrow T_1)$
$W_3(X)$, $W_1(X)$ $(T_3 \rightarrow T_1)$
$W_3(X)$, $R_4(X)$ $(T_3 \rightarrow T_4)$
$W_1(X)$, $R_4(X)$ $(T_1 \rightarrow T_4)$
$W_2(Y)$, $R_4(Y)$ $(T_2 \rightarrow T_4)$

Step-02:

Draw the precedence graph-

- Clearly, there exists no cycle in the precedence graph.
- Therefore, the given schedule S is conflict serializable.



Conflict Serializability

Problem-03:

Check whether the given schedule S is conflict serializable or not. If yes, then determine all the possible serialized schedules-

T1	T2	T3	T4
			R(A)
	R(A)		
		R(A)	
W(B)			
	W(A)		
		R(B)	
	W(B)		

Checking Whether S is Conflict Serializable Or Not

Step-01:

List all the conflicting operations and determine the dependency between the transactions-

$R_4(A)$, $W_2(A)$ $(T_4 \rightarrow T_2)$

$R_3(A)$, $W_2(A)$ $(T_3 \rightarrow T_2)$

$W_1(B)$, $R_3(B)$ $(T_1 \rightarrow T_3)$

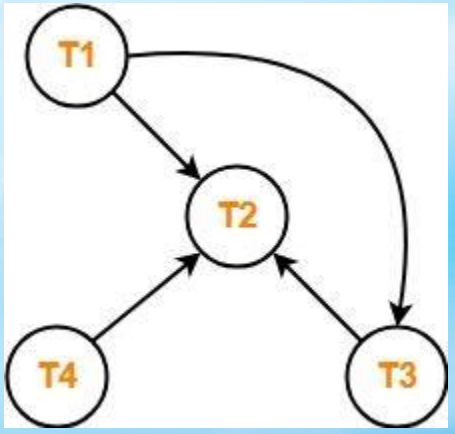
$W_1(B)$, $W_2(B)$ $(T_1 \rightarrow T_2)$

$R_3(B)$, $W_2(B)$ $(T_3 \rightarrow T_2)$

Step-02:

Draw the precedence graph-

- Clearly, there exists no cycle in the precedence graph.
- Therefore, the given schedule S is conflict serializable.



View Serializability

If a given schedule is found to be view equivalent to some serial schedule, then it is called as a view serializable schedule.

View Equivalent Schedules

Consider two schedules S1 and S2 each consisting of two transactions T1 and T2.
Schedules S1 and S2 are called view equivalent if the following three conditions hold true for them:

Condition-01

For each data item X, if transaction T_i reads X from the database initially in schedule S1, then in schedule S2 also, T_i must perform the initial read of X from the database.

Thumb Rule: "Initial readers must be same for all the data items".

Condition-02

If transaction T_i reads a data item that has been updated by the transaction T_j in schedule S1, then in schedule S2 also, transaction T_i must read the same data item that has been updated by the transaction T_j.

Thumb Rule: "Write-read sequence must be same.".

Condition-03

For each data item X, if X has been updated at last by transaction T_i in schedule S1, then in schedule S2 also, X must be updated at last by transaction T_i.

Thumb Rule: "Final writers must be same for all the data items".

View Serializability

Checking Whether a Schedule is View Serializable Or Not

Method-01

Check whether the given schedule is conflict serializable or not.

1. If the given schedule is conflict serializable, then it is surely view serializable. Stop and report your answer.
2. If the given schedule is not conflict serializable, then it may or may not be view serializable. Go and check using other methods.

Rules

- All conflict serializable schedules are view serializable.
- All view serializable schedules may or may not be conflict serializable.

Method-02:

Check if there exists any blind write operation.

(Writing without reading is called as a blind write).

1. If there does not exist any blind write, then the schedule is surely not view serializable. Stop and report your answer.
2. If there exists any blind write, then the schedule may or may not be view serializable. Go and check using other methods.

Rules

- No blind write means not a view serializable schedule.

View Serializability

Method-03:

In this method, try finding a view equivalent serial schedule.

1. By using the above three conditions, write all the dependencies.
2. Then, draw a graph using those dependencies.
3. If there exists no cycle in the graph, then the schedule is view serializable otherwise not.

View Serializability

Problem-01

Check whether the given schedule S is view serializable or not-

T1	T2	T3	T4
R (A)			
	R (A)		
		R (A)	
			R (A)
W (B)			
	W (B)		
		W (B)	
			W (B)

Solution-

We know, if a schedule is conflict serializable, then it is surely view serializable.

So, let us check whether the given schedule is conflict serializable or not.

Checking Whether S is Conflict Serializable Or Not

Step-01:

List all the conflicting operations and determine the dependency between the transactions-

$W_1(B)$, $W_2(B)$ $(T_1 \rightarrow T_2)$

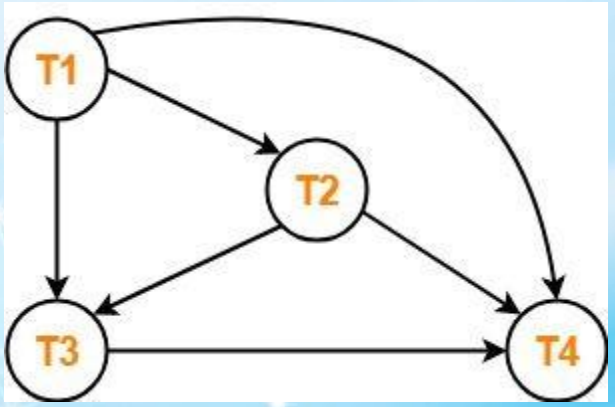
$W_1(B)$, $W_3(B)$ $(T_1 \rightarrow T_3)$

$W_1(B)$, $W_4(B)$ $(T_1 \rightarrow T_4)$

$W_2(B)$, $W_3(B)$ $(T_2 \rightarrow T_3)$

$W_2(B)$, $W_4(B)$ $(T_2 \rightarrow T_4)$

$W_3(B)$, $W_4(B)$ $(T_3 \rightarrow T_4)$



Step-02:

Draw the precedence graph-

- Clearly, there exists no cycle in the precedence graph.
- Therefore, the given schedule S is conflict serializable.

Thus, we conclude that the given schedule is also view serializable.

View Serializability

Problem-02:

Check whether the given schedule S is view serializable or not-

Solution-

We know, if a schedule is conflict serializable, then it is surely view serializable. So, let us check whether the given schedule is conflict serializable or not.

T1	T2	T3
R (A)		
	R (A)	
		W (A)
W (A)		

Checking Whether S is Conflict Serializable Or Not-

Step-01:

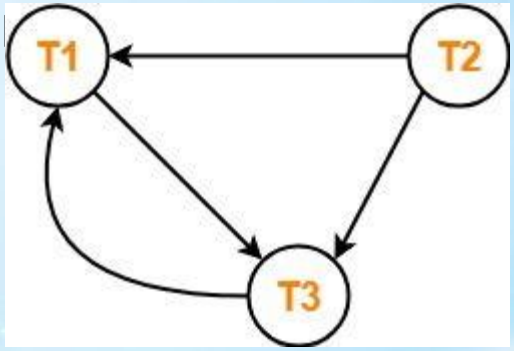
List all the conflicting operations and determine the dependency

$R_1(A)$, $W_3(A)$ $(T_1 \rightarrow T_3)$

$R_2(A)$, $W_3(A)$ $(T_2 \rightarrow T_3)$

$R_2(A)$, $W_1(A)$ $(T_2 \rightarrow T_1)$

$W_3(A)$, $W_1(A)$ $(T_3 \rightarrow T_1)$



Step-02:

Draw the precedence graph-

- Clearly, there exists a cycle in the precedence graph.
- Therefore, the given schedule S is not conflict serializable.

View Serializability

Now,

- Since, the given schedule S is not conflict serializable, so, it may or may not be view serializable. To check whether S is view serializable or not, let us use another method.

Let us check for blind writes.

Checking for Blind Writes

- There exists a blind write $W_3(A)$ in the given schedule S.
- Therefore, the given schedule S may or may not be view serializable.

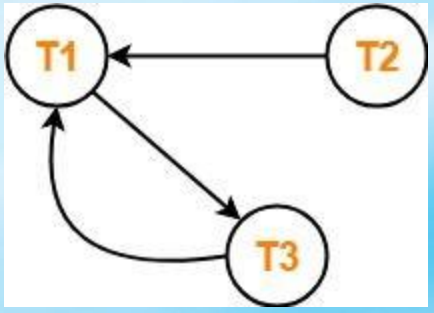
Now,

To check whether S is view serializable or not, let us use another method.

Let us derive the dependencies and then draw a dependency graph.

Drawing a Dependency Graph-

- T1 firstly reads A and T3 firstly updates A.
- So, T1 must execute before T3.
- Thus, we get the dependency **T1 → T3**.
- Final updation on A is made by the transaction T1.
- So, T1 must execute after all other transactions.
- Thus, we get the dependency **(T2, T3) → T1**.
- There exists no write-read sequence.



Now, let us draw a dependency graph using these dependencies-

- Clearly, there exists a cycle in the dependency graph.
- Thus, we conclude that the given schedule S is not view serializable.

View Serializability

Problem-03:

Check whether the given schedule S is view serializable or not

T1	T2
R (A)	
A = A + 10	
	R (A)
	A = A + 10
W (A)	
	W (A)
R (B)	
B = B + 20	
	R (B)
	B = B x 1.1
W (B)	
	W (B)

Solution-

We know, if a schedule is conflict serializable, then it is surely view serializable. So, let us check whether the given schedule is conflict serializable or not.

Checking Whether S is Conflict Serializable Or Not-

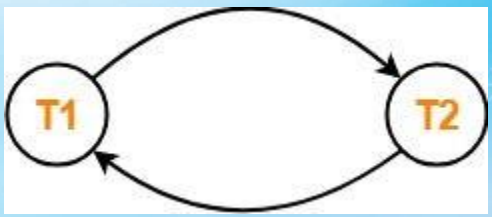
Step-01:

List all the conflicting operations and determine the dependency between the transactions-

$R_1(A)$, $W_2(A)$ $(T_1 \rightarrow T_2)$
$R_2(A)$, $W_1(A)$ $(T_2 \rightarrow T_1)$
$W_1(A)$, $W_2(A)$ $(T_1 \rightarrow T_2)$
$R_1(B)$, $W_2(B)$ $(T_1 \rightarrow T_2)$
$R_2(B)$, $W_1(B)$ $(T_2 \rightarrow T_1)$

Step-02:

Draw the precedence graph-



- Clearly, there exists a cycle in the precedence graph.
- Therefore, the given schedule S is not conflict serializable.

View Serializability

Now,

- Since, the given schedule S is not conflict serializable, so, it may or may not be view serializable. To check whether S is view serializable or not, let us use another method.

Let us check for blind writes.

Checking for Blind Writes-

- There exists no blind write in the given schedule S.
- Therefore, it is surely not view serializable.

Alternatively,

- You could directly declare that the given schedule S is not view serializable.
- This is because there exists no blind write in the schedule.
- You need not check for conflict Serializability.

Non-Serializable Schedule

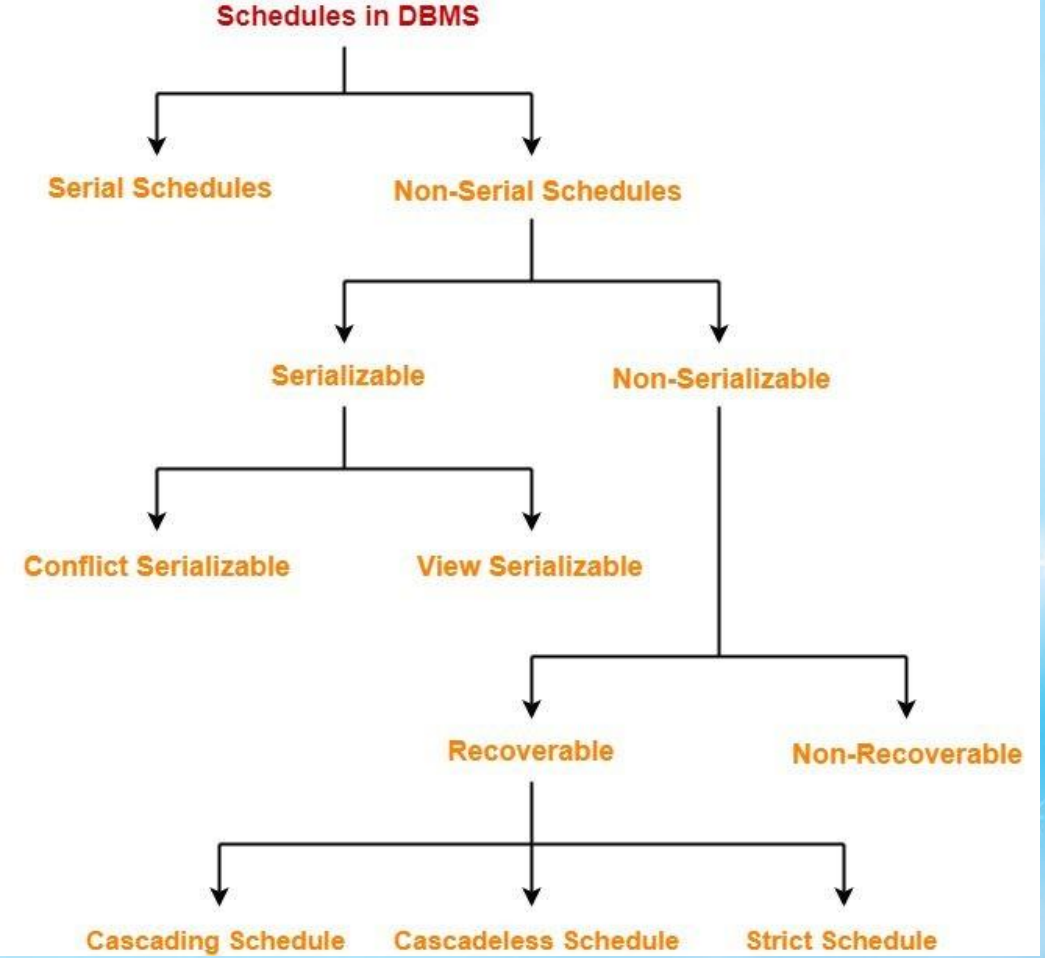
Non-Serializable Schedules-

- A non-serial schedule which is not serializable is called as a non-serializable schedule.
- A non-serializable schedule is not guaranteed to produce the same effect as produced by some serial schedule on any consistent database.

Characteristics-

Non-serializable schedules

- may or may not be consistent
- may or may not be recoverable



Non-Serializable Schedule

Irrecoverable Schedules (Non-recoverable)

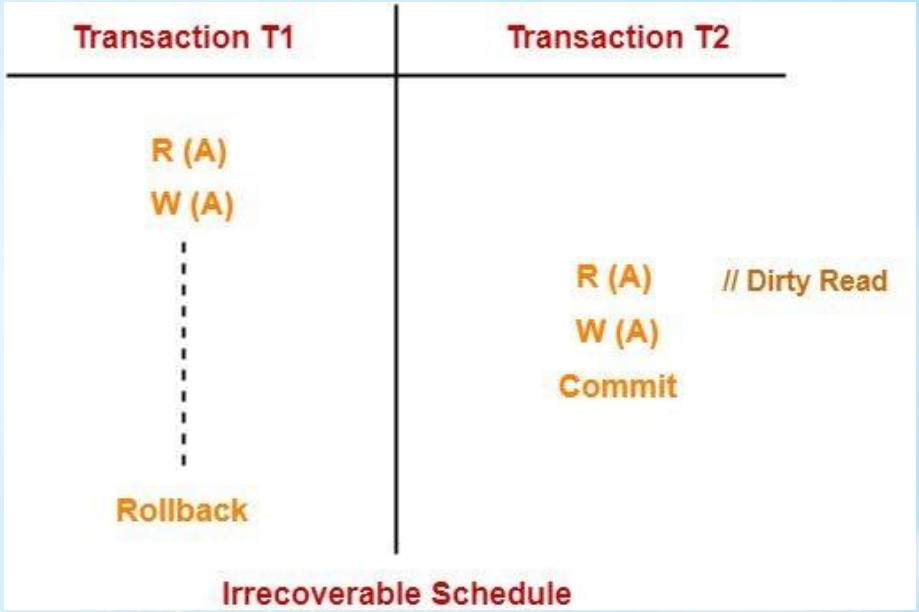
If in a schedule,

- A transaction performs a dirty read operation from an uncommitted transaction
- And commits before the transaction from which it has read the value

then such a schedule is known as an **Irrecoverable Schedule**.

Example-

Consider the given schedule-



Here,

- T2 performs a dirty read operation.
- T2 commits before T1.
- T1 fails later and roll backs.
- The value that T2 read now stands to be incorrect.
- T2 can not recover since it has already committed.

Non-Serializable Schedule

Recoverable Schedules

If in a schedule,

- A transaction performs a dirty read operation from an uncommitted transaction
- And its commit operation is delayed till the uncommitted transaction either commits or roll backs

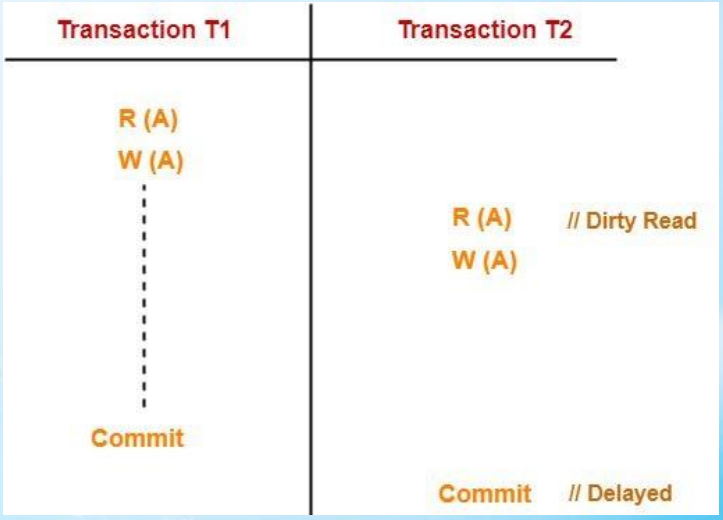
then such a schedule is known as a **Recoverable Schedule**.

Here,

- The commit operation of the transaction that performs the dirty read is delayed.
- This ensures that it still has a chance to recover if the uncommitted transaction fails later.

Example-

Consider the following schedule-



Here,

- T2 performs a dirty read operation.
- The commit operation of T2 is delayed till T1 commits or roll backs.
- T1 commits later.
- T2 is now allowed to commit.

Non-Serializable Schedule

Checking Whether a Schedule is Recoverable or Irrecoverable-

Method-01:

Check whether the given schedule is conflict serializable or not.

- If the given schedule is conflict serializable, then it is surely recoverable. Stop and report your answer.
- If the given schedule is not conflict serializable, then it may or may not be recoverable. Go and check using other methods.

Rules

- All conflict serializable schedules are recoverable.
- All recoverable schedules may or may not be conflict serializable.

Non-Serializable Schedule

Method-02:

Check if there exists any dirty read operation.

(Reading from an uncommitted transaction is called as a dirty read)

- If there does not exist any dirty read operation, then the schedule is surely recoverable. Stop and report your answer.
- If there exists any dirty read operation, then the schedule may or may not be recoverable.

If there exists a dirty read operation, then follow the following cases-

Case-01:

If the commit operation of the transaction performing the dirty read occurs before the commit or abort operation of the transaction which updated the value, then the schedule is irrecoverable.

Case-02:

If the commit operation of the transaction performing the dirty read is delayed till the commit or abort operation of the transaction which updated the value, then the schedule is recoverable.

Rule

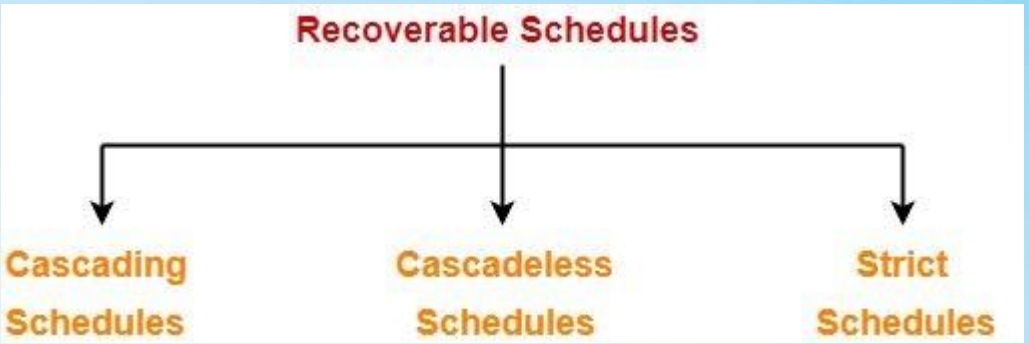
No dirty read means a recoverable schedule.

Non-Serializable Schedule

Types of Recoverable Schedules

A recoverable schedule may be any one of these kinds-

1. Cascading Schedule
2. Cascadeless Schedule
3. Strict Schedule



Cascading Schedule

- If in a schedule, failure of one transaction causes several other dependent transactions to rollback or abort, then such a schedule is called as a **Cascading Schedule** or **Cascading Rollback** or **Cascading Abort**.
- It simply leads to the wastage of CPU time.

Non-Serializable Schedule

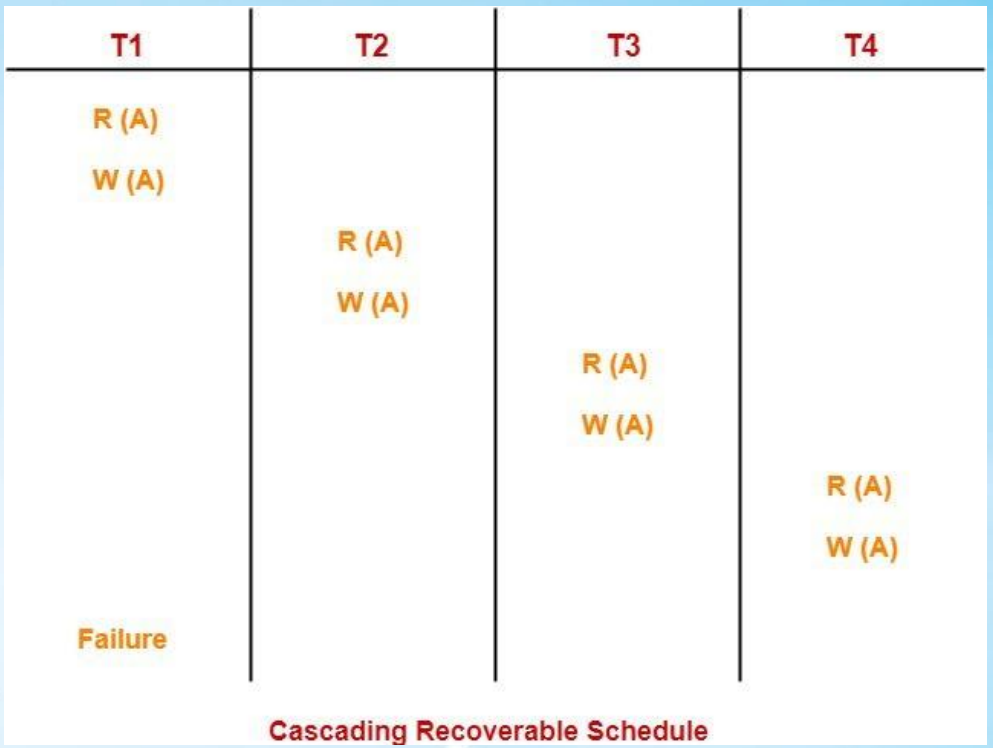
Example

Here,

Transaction T2 depends on transaction T1.

Transaction T3 depends on transaction T2.

Transaction T4 depends on transaction T3.



Cascading Recoverable Schedule

In this schedule,

- The failure of transaction T1 causes the transaction T2 to rollback.
- The rollback of transaction T2 causes the transaction T3 to rollback.
- The rollback of transaction T3 causes the transaction T4 to rollback.

Such a rollback is called as a **Cascading Rollback**.

NOTE

If the transactions T2, T3 and T4 would have committed before the failure of transaction T1, then the schedule would have been irrecoverable.

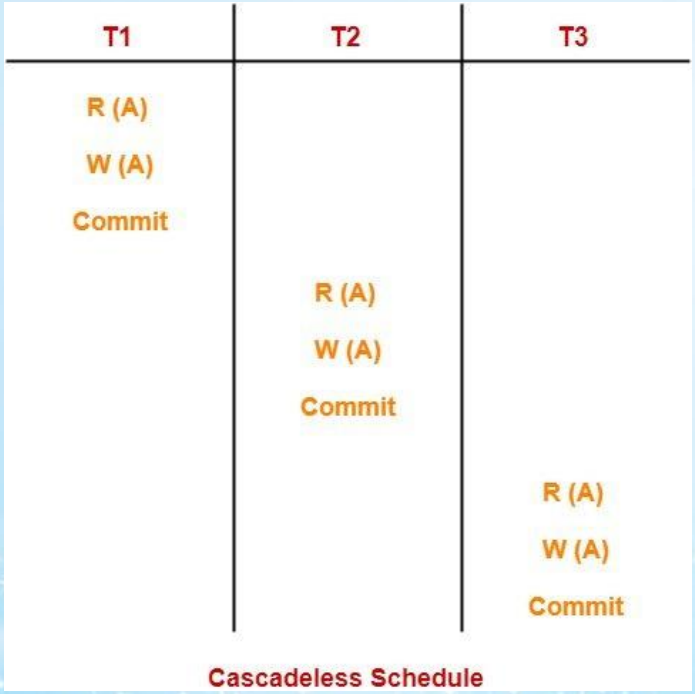
Non-Serializable Schedule

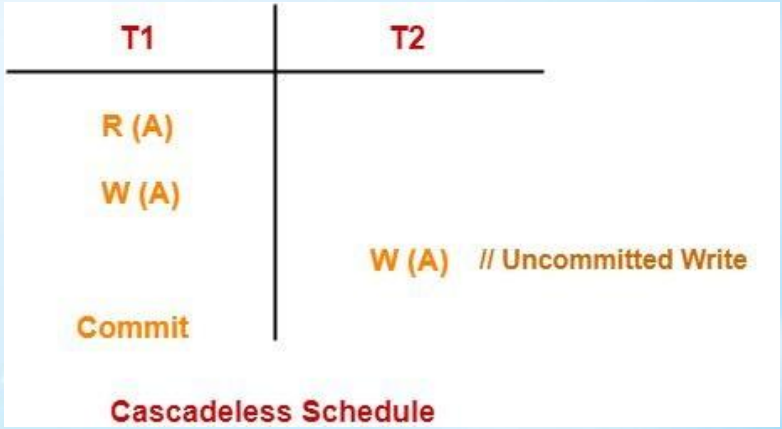
Cascadeless Schedule-

If in a schedule, a transaction is not allowed to read a data item until the last transaction that has written it is committed or aborted, then such a schedule is called as a **Cascadeless Schedule**.

In other words,

- Cascadeless schedule allows only committed read operations.
- Therefore, it avoids cascading roll back and thus saves CPU time.





NOTE

- Cascadeless schedule allows only committed read operations.
- However, it allows uncommitted write operations.

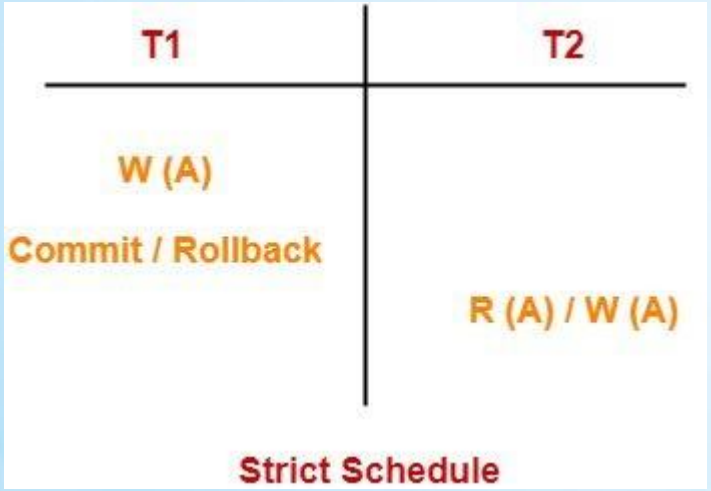
Non-Serializable Schedule

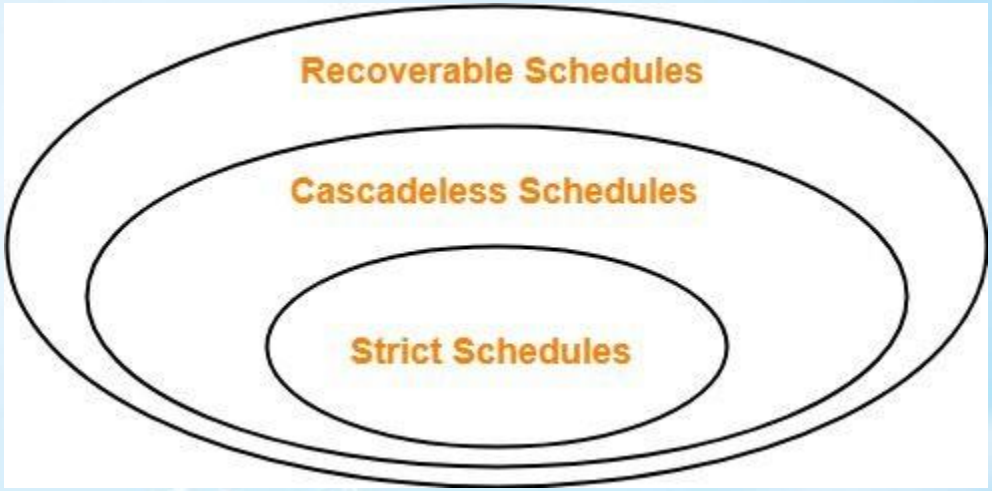
Strict Schedule

If in a schedule, a transaction is neither allowed to read nor write a data item until the last transaction that has written it is committed or aborted, then such a schedule is called as a **Strict Schedule**.

In other words,

- Strict schedule allows only committed read and write operations.
- Clearly, strict schedule implements more restrictions than Cascadeless Schedule.





Remember

- Strict schedules are more strict than Cascadeless schedules.
- All strict schedules are Cascadeless schedules.
- All Cascadeless schedules are not strict schedules.

Concurrency Control

Concurrency Problems in DBMS

When multiple transactions execute concurrently in an uncontrolled or unrestricted manner, then it might lead to several problems.

Such problems are called as **concurrency problems**.

The concurrency problems are

1. Dirty Read Problem
2. Unrepeatable Read Problem
3. Lost Update Problem
4. Phantom Read Problem

Concurrency Control

Dirty Read Problem

Reading the data written by an uncommitted transaction is called as dirty read.

This read is called as dirty read because-

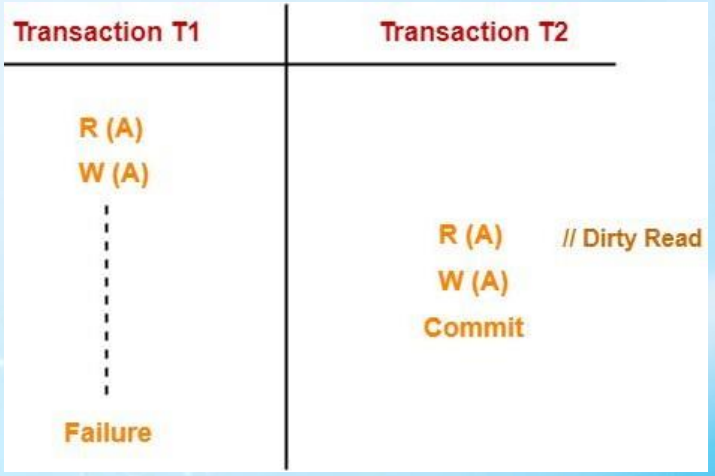
- There is always a chance that the uncommitted transaction might roll back later.
- Thus, uncommitted transaction might make other transactions read a value that does not even exist.
- This leads to inconsistency of the database.

NOTE-

- Dirty read does not lead to inconsistency always.
- It becomes problematic only when the uncommitted transaction fails and roll backs later due to some reason.

Here,

- T1 reads the value of A.
- T1 updates the value of A in the buffer.
- T2 reads the value of A from the buffer.
- T2 writes the updated the value of A.
- T2 commits.
- T1 fails in later stages and rolls back. In this example,



- T2 reads the dirty value of A written by the uncommitted transaction T1.
- T1 fails in later stages and roll backs.
- Thus, the value that T2 read now stands to be incorrect.
- Therefore, database becomes inconsistent.

Concurrency Control

Unrepeatable Read Problem

This problem occurs when a transaction gets to read unrepeated i.e. different values of the same variable in its different read operations even when it has not updated its value.

Example

Transaction T1	Transaction T2
R (X)	
	R (X)
W (X)	
	R (X) // Unrepeated Read

Here,

- T1 reads the value of X (= 10 say).
- T2 reads the value of X (= 10).
- T1 updates the value of X (from 10 to 15 say) in the buffer.
- T2 again reads the value of X (but = 15).

In this example,

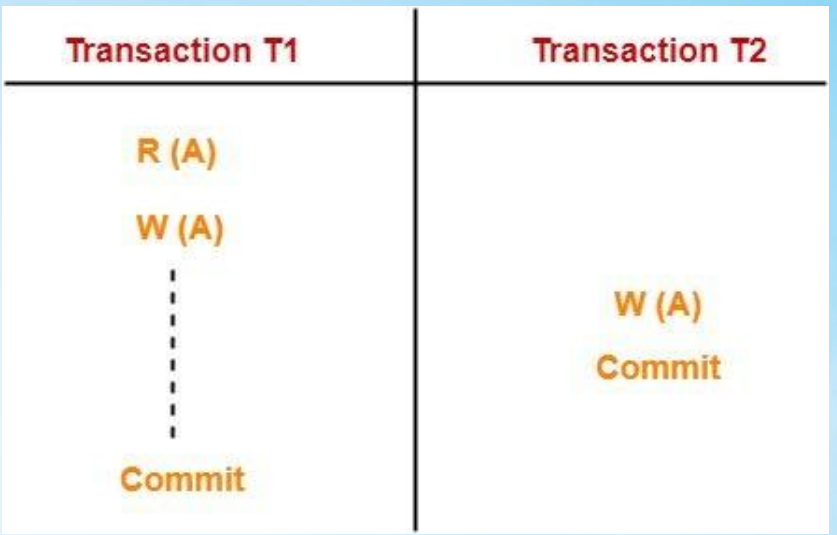
- T2 gets to read a different value of X in its second reading.
- T2 wonders how the value of X got changed because according to it, it is running in isolation.

Concurrency Control

Lost Update Problem

This problem occurs when multiple transactions execute concurrently and updates from one or more transactions get lost.

Example



Here,

- T1 reads the value of A (= 10 say).
- T1 updates the value to A (= 15 say) in the buffer.
- T2 does blind write A = 25 (write without read) in the buffer.
- T2 commits.
- When T1 commits, it writes A = 25 in the database.

In this example,

- T1 writes the over written value of X in the database.
- Thus, update from T1 gets lost.

NOTE

This problem occurs whenever there is a write-write conflict.

In write-write conflict, there are two writes one by each transaction on the same data item without any read in the middle.

Concurrency Control

Phantom Read Problem

This problem occurs when a transaction reads some variable from the buffer and when it reads the same variable later, it finds that the variable does not exist.

Example

Transaction T1	Transaction T2
R (X)	
	R (X)
Delete (X)	
	Read (X)

Here,

- T1 reads X.
- T2 reads X.
- T1 deletes X.
- T2 tries reading X but does not find it.

In this example,

- T2 finds that there does not exist any variable X when it tries reading X again.
- T2 wonders who deleted the variable X because according to it, it is running in isolation.

Avoiding Concurrency Problems

To ensure consistency of the database, it is very important to prevent the occurrence of above problems.

Concurrency Control Protocols help to prevent the occurrence of above problems and maintain the consistency of the database.

Concurrency Control

Avoiding Concurrency Problems

To ensure consistency of the database, it is very important to prevent the occurrence of above problems.

Concurrency Control Protocols help to prevent the occurrence of above problems and maintain the consistency of the database.

Concurrency control protocols ensure atomicity, isolation, and Serializability of concurrent transactions. The concurrency control protocol can be divided into three categories:

1. Lock based protocol
2. Time-stamp protocol
3. Validation based protocol

Concurrency Control

Lock-based Protocols

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

- A ***lock*** is a variable associated with a data item that describes the status of the data item with respect to the possible operations that can be applied to it. Manipulation the value of a lock is called ***locking***.

- Generally there is a one lock for each data item in the database.

- Locks are used as a means of synchronizing the access by concurrent transactions to the database items.

Locks are of two kinds:

1. **Binary Locks** − A lock on a data item can be in two states; it is either locked or unlocked.
2. **Shared/exclusive** − This type of locking mechanism differentiates the locks based on their uses. If a lock is acquired on a data item to perform a write operation, it is an exclusive lock. Allowing more than one transaction to write on the same data item would lead the database into an inconsistent state. Read locks are shared because no data value is being changed.

Concurrency Control

Shared/Exclusive Lock − This type of locking mechanism differentiates the locks based on their uses. any transaction cannot read or write data until it acquires an appropriate lock on it. There are two types of lock:

1. Shared lock:

- It is also known as a Read-only lock. In a shared lock, the data item can only read by the transaction.
- It can be shared between the transactions because when the transaction holds a lock, then it can't update the data on the data item.
- If a transaction Ti has obtained a shared-mode lock (denoted by S) on item Q, then Ti can read, but can not write Q.

2. Exclusive lock:

- In the exclusive lock, the data item can be both reads as well as written by the transaction.
- This lock is exclusive, and in this lock, multiple transactions do not modify the same data simultaneously.
- If a transaction Ti has obtained an exclusive-mode lock (denoted by X) on item Q, then Ti can both read and write Q.

We require that every transaction request a lock in an appropriate mode on data item Q, depending on the types of operations that it will perform on Q. The transaction makes the request to the concurrency-control manager. The transaction can proceed with the operation only after the concurrency-control manager grants the lock to the transaction.

Concurrency Control

A compatibility function on a set of lock modes can be defined as follows - Let A and B represent arbitrary lock modes. Suppose that a transaction Ti requests a lock of mode A on item Q on which transaction Tj (Ti ≠ Tj) currently holds a lock of mode B. If transaction Ti can be granted a lock on Q immediately, in spite of the presence of the mode B lock, then mode A is compatible with mode B, This function can be represented by a matrix-

The compatibility relation between the two modes is shown in the matrix of figure. An element comp (A, B) of the matrix has the value true if and only if mode A is compatible with mode B.

- A shared mode is compatible with shared mode, but not with exclusive mode.
- At any time, several shared-mode locks can be held simultaneously by different transactions on a particular data item
- A subsequent exclusive mode lock request has to wait until the currently held shared mode lock are released.

	S	X
S	True	False
X	False	False

Concurrency Control

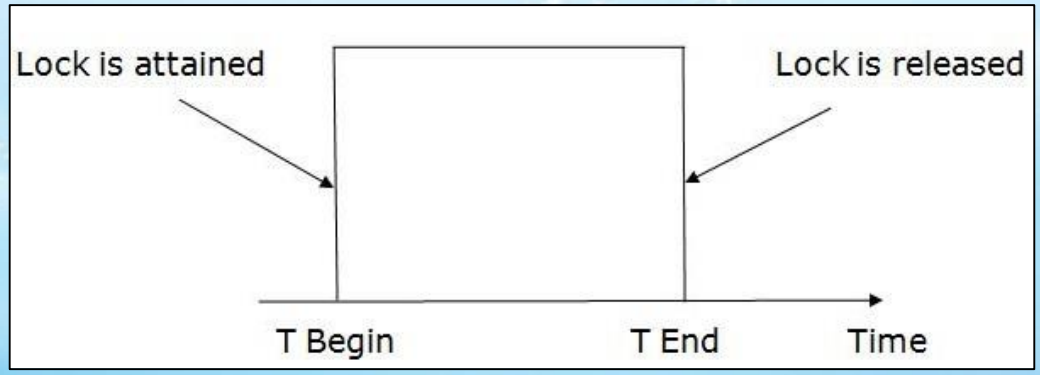
There are four types of lock protocols available:

1. Simplistic lock protocol

- It is the simplest way of locking the data while transaction.
- Simplistic lock-based protocols allow all the transactions to get the lock on the data before insert or delete or update on it.
- It will unlock the data item after completing the transaction.

2. Pre-claiming Lock Protocol

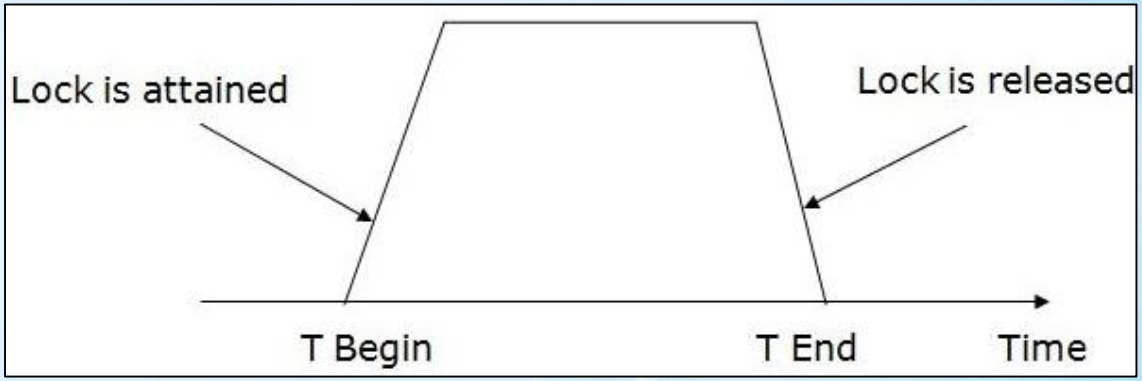
- Pre-claiming Lock Protocols evaluate the transaction to list all the data items on which they need locks.
- Before initiating an execution of the transaction, it requests DBMS for all the lock on all those data items.
- If all the locks are granted then this protocol allows the transaction to begin. When the transaction is completed then it releases all the lock.
- If all the locks are not granted then this protocol allows the transaction to rolls back and waits until all the locks are granted.



Concurrency Control

3. Two-phase locking (2PL)

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



There are two phases of 2PL:

2. **Growing phase:** In the growing phase, a new lock on the data item may be acquired by the transaction, but none can be released.
3. **Shrinking phase:** In the shrinking phase, existing lock held by the transaction may be released, but no new locks can be acquired.

In the below example, if lock conversion is allowed then the following phase can happen:

- Upgrading of lock (from S(a) to X (a)) is allowed in growing phase.
- Downgrading of lock (from X(a) to S(a)) must be done in shrinking phase.

Concurrency Control

The following way shows how unlocking and locking work with 2-PL.

Transaction T1:

- **Growing phase:** from step 1-3
- **Shrinking phase:** from step 5-7
- **Lock point:** at 3

Transaction T2:

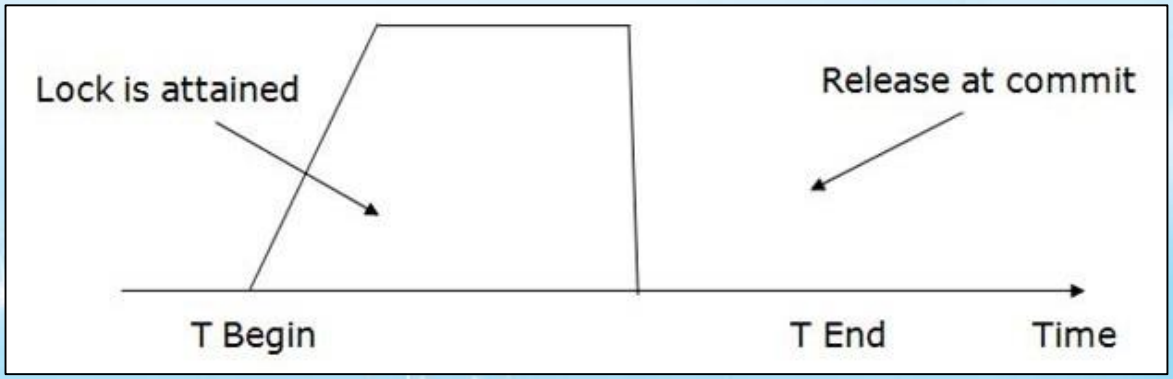
- **Growing phase:** from step 2-6
- **Shrinking phase:** from step 8-9
- **Lock point:** at 6

	T1	T2
0	LOCK-S(A)	
1		LOCK-S(A)
2	LOCK-X(B)	
3	——	——
4	UNLOCK(A)	
5		LOCK-X(C)
6	UNLOCK(B)	
7		UNLOCK(A)
8		UNLOCK(C)
9	——	——

Concurrency Control

4. **Strict Two-phase locking (Strict-2PL)**

- The first phase of Strict-2PL is similar to 2PL. In the first phase, after acquiring all the locks, the transaction continues to execute normally.
- The only difference between 2PL and strict 2PL is that Strict-2PL does not release a lock after using it.
- Strict-2PL waits until the whole transaction to commit, and then it releases all the locks at a time.
- Strict-2PL protocol does not have shrinking phase of lock release.
- It does not have cascading abort as 2PL does.



Concurrency Control

Timestamp Based Protocol

- Timestamp Based Protocol helps DBMS to identify the transactions.
- It is a unique identifier. Each transaction is issued a timestamp when it enters into the system.
- Timestamp protocol determines the Serializability order.
- It is most commonly used concurrency protocol.
- It uses either system time or logical counter as a timestamp.
- It starts working as soon as a transaction is created.

Timestamp Ordering Protocol

- The TO Protocol ensures Serializability among transactions in their conflicting read and write operations.

- The Timestamp Ordering Protocol is used to order the transactions based on their Timestamps. The order of transaction is nothing but the ascending order of the transaction creation.

- The priority of the older transaction is higher that's why it executes first. To determine the timestamp of the transaction, this protocol uses system time or logical counter.

- The lock-based protocol is used to manage the order between conflicting pairs among transactions at the execution time. But Timestamp based protocols start working as soon as a transaction is created.

Concurrency Control

- Let's assume there are two transactions T1 and T2. Suppose the transaction T1 has entered the system at 007 times and transaction T2 has entered the system at 009 times. T1 has the higher priority, so it executes first as it is entered the system first.

- The timestamp ordering protocol also maintains the timestamp of last 'read' and 'write' operation on a data.

- The transaction of timestamp (T) is denoted as TS(T).
- Data item (X) of read timestamp is denoted by R–TS(X).
- Data item (X) of write timestamp is denoted by W–TS(X).

There are three types of the Timestamp Ordering Protocol.

1. **Basic Timestamp Ordering Protocol**
2. **Validation Based Protocol**
3. **Thomas Write Rule**

Concurrency Control

1. **Basic Timestamp ordering protocol works as follows:**

Condition 1:

- Check the following condition whenever a transaction Ti issues a **Read (X)** operation:
  - If W_TS(X) >TS(Ti) then the operation is rejected.
  - If W_TS(X) <= TS(Ti) then the operation is executed.
- Timestamps of all the data items are updated.

Condition 2:

- Check the following condition whenever a transaction Ti issues a **Write(X)** operation:
  - If TS(Ti) < R_TS(X) then the operation is rejected.
  - If TS(Ti) < W_TS(X) then the operation is rejected and Ti is rolled back otherwise the operation is executed.

Advantages and Disadvantages of TO protocol:

- TO protocol ensures Serializability since the precedence graph is as follows:

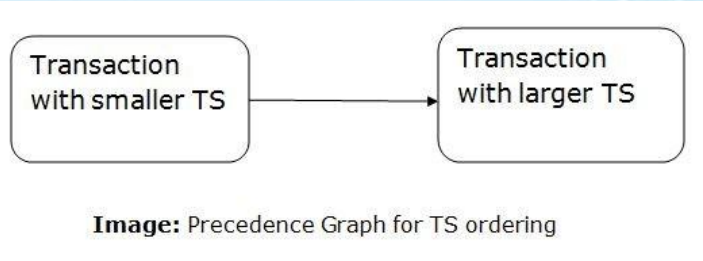


Image: Precedence Graph for TS ordering

- TS protocol ensures freedom from deadlock that means no transaction ever waits.
- But the schedule may not be recoverable and may not even be cascade-free.

Concurrency Control

Revisited

Rule 1

If **TS(Ti) < W-TS(X),** this violates timestamp order of **Ti** with regard to the writer of **X**.

→ Abort **Ti** and restart it with same TS.

Else:

→ Allow **Ti** to read **X**.

→ Update **R-TS(X)** to **max(R-TS(X), TS(Ti)**)

→ Have to make a local copy of X to ensure repeatable reads for Ti .

Rule 2

If **TS(Ti) < R-TS(X)** or **TS(Ti) < W-TS(X)**

→ Abort and restart **Ti** .

Else:

→ Allow **Ti** to write **X** and update **W-TS(X)**

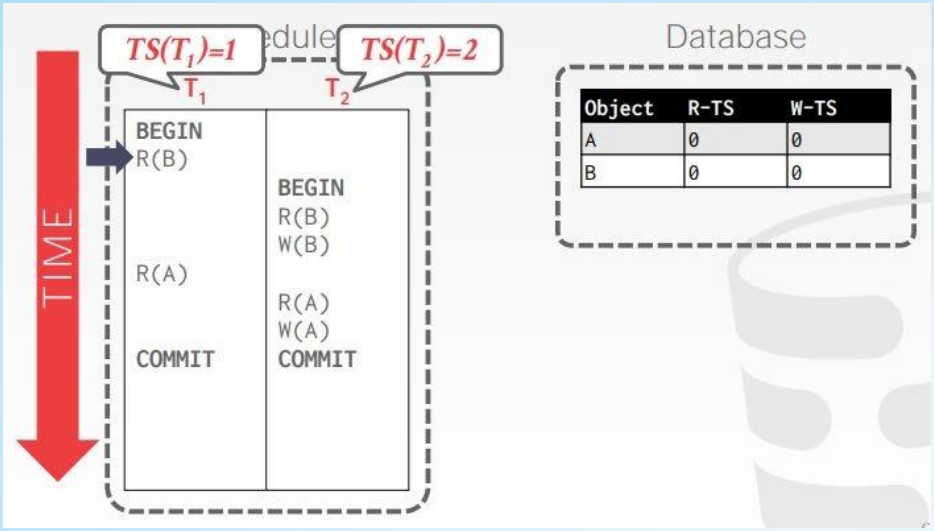
→ Also have to make a local copy of **X** to ensure repeatable reads for **Ti** .

Concurrency Control

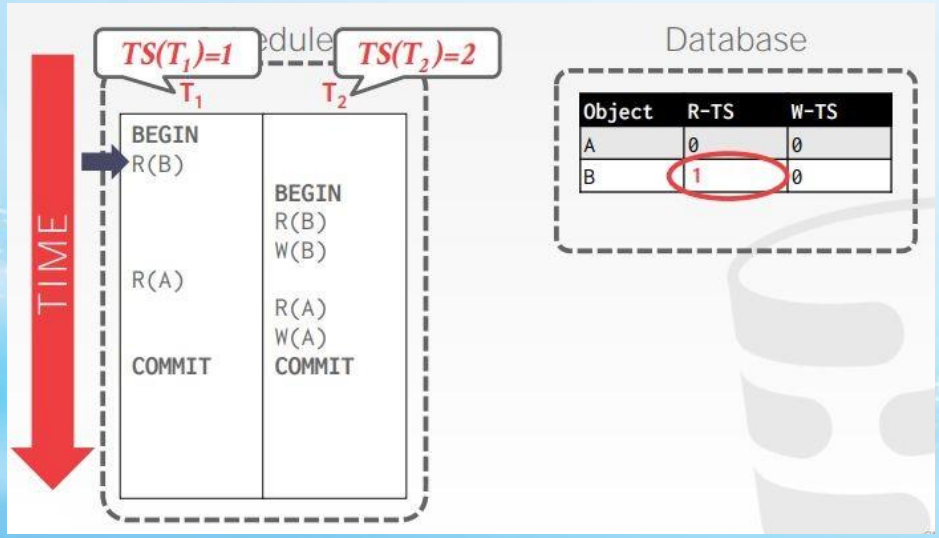
Basic Timestamp ordering protocol

Example:

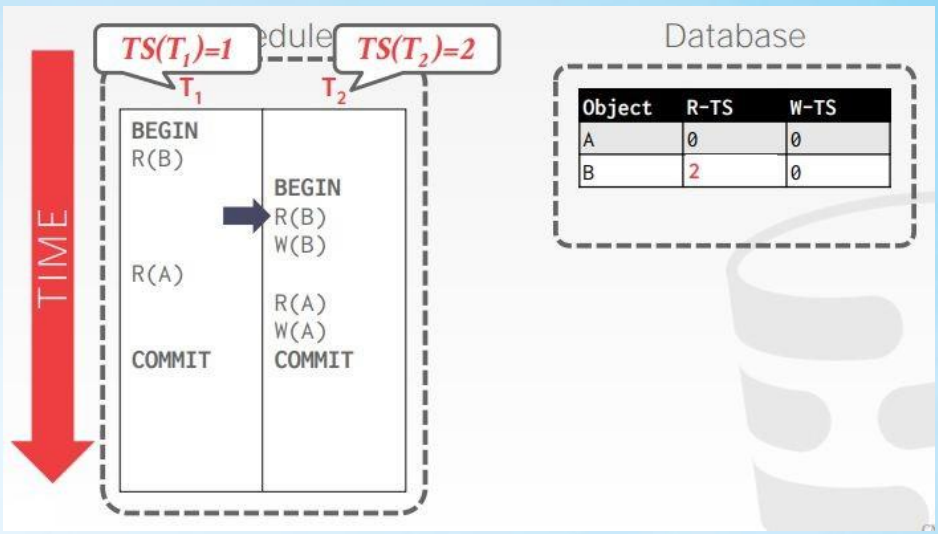
1



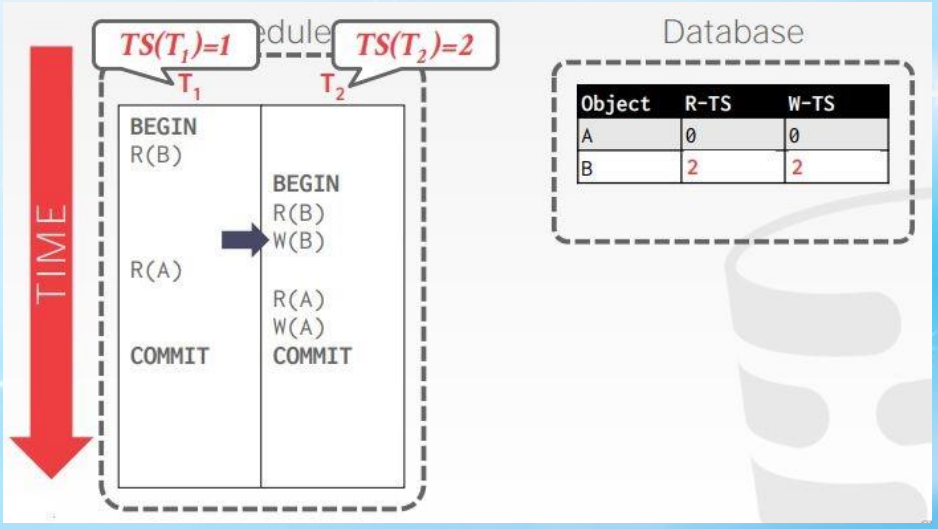
2



3



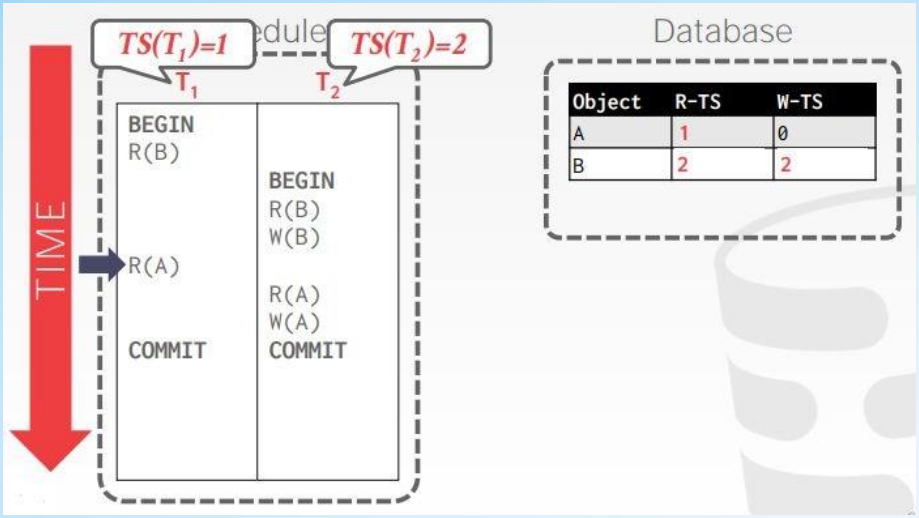
4



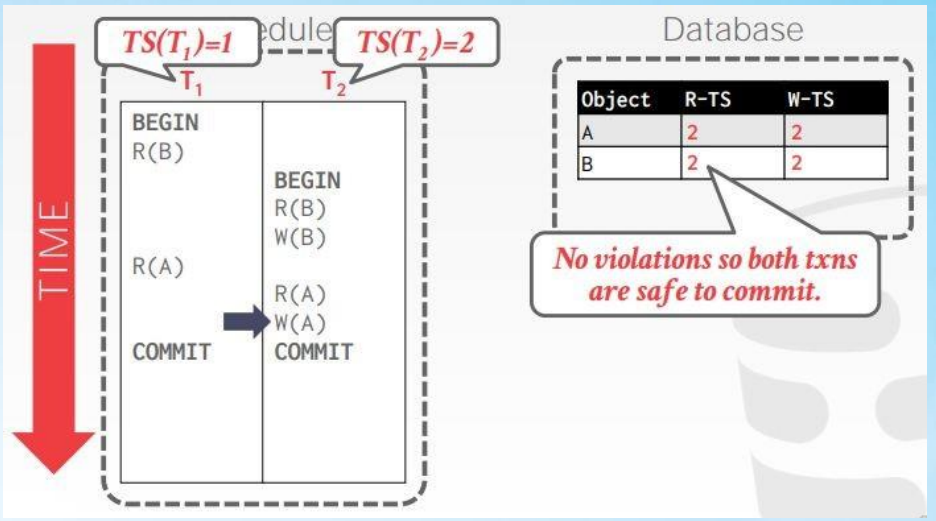
Basic Timestamp ordering protocol

Example:

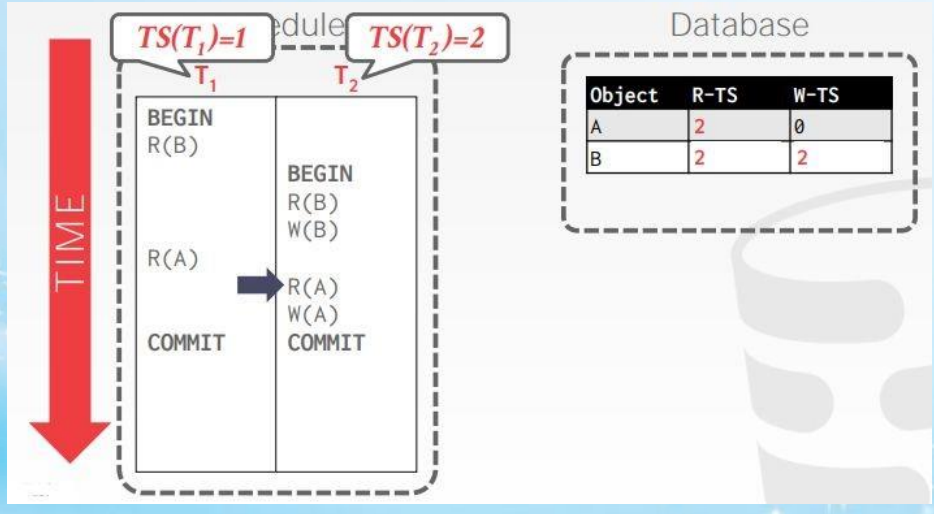
5



7



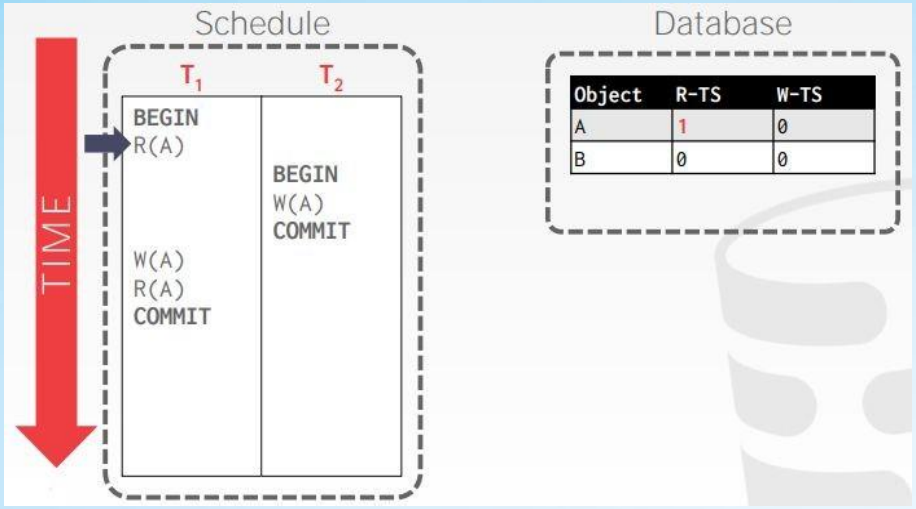
6



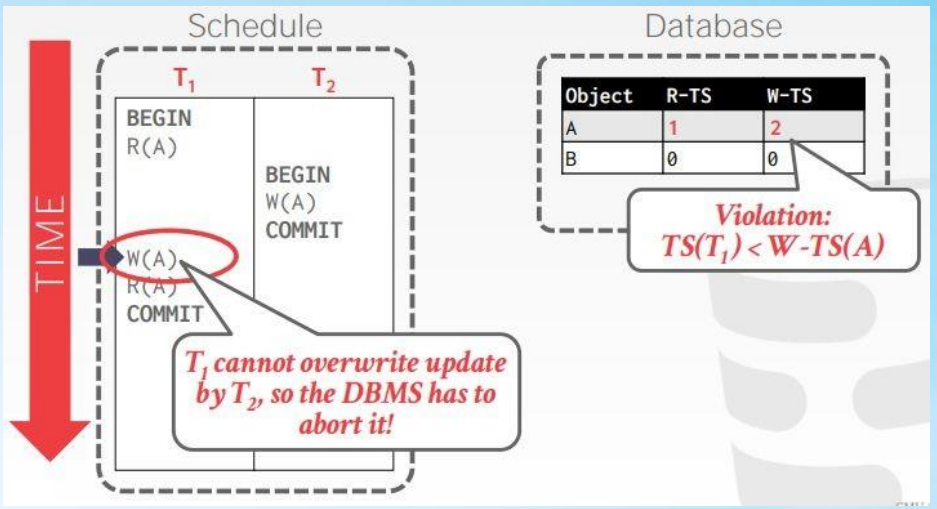
Basic Timestamp ordering protocol

Example:

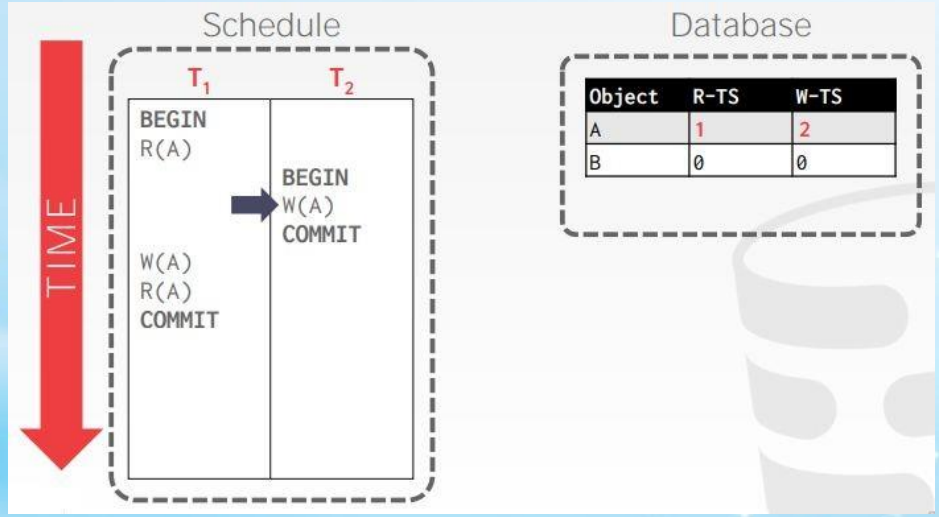
1



3



2



Concurrency Control

Thomas write Rule

Thomas Write Rule provides the guarantee of Serializability order for the protocol. It improves the Basic Timestamp Ordering Algorithm.

The basic Thomas write rules are as follows:

- If TS(T) < R_TS(X) then transaction T is aborted and rolled back, and operation is rejected.
- If TS(T) < W_TS(X) then don't execute the W_item(X) operation of the transaction and continue processing.
- If neither condition 1 nor condition 2 occurs, then allowed to execute the WRITE operation by transaction Ti and set W_TS(X) to TS(T).

If we use the Thomas write rule then some serializable schedule can be permitted that does not conflict serializable as illustrate by the schedule in a given figure:

In Short

If **TS(Ti)** < **R-TS(X)**:

→ Abort and restart **Ti** .

If **TS(Ti)** < **W-TS(X)**:

→ Thomas Write Rule: Ignore the write and allow the transaction to continue.

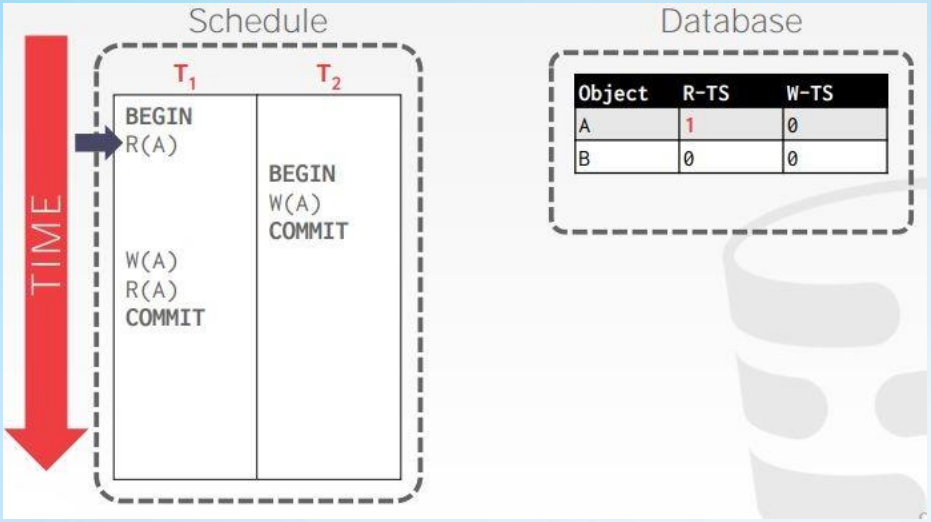
→ This violates timestamp order of **Ti** .

Else:

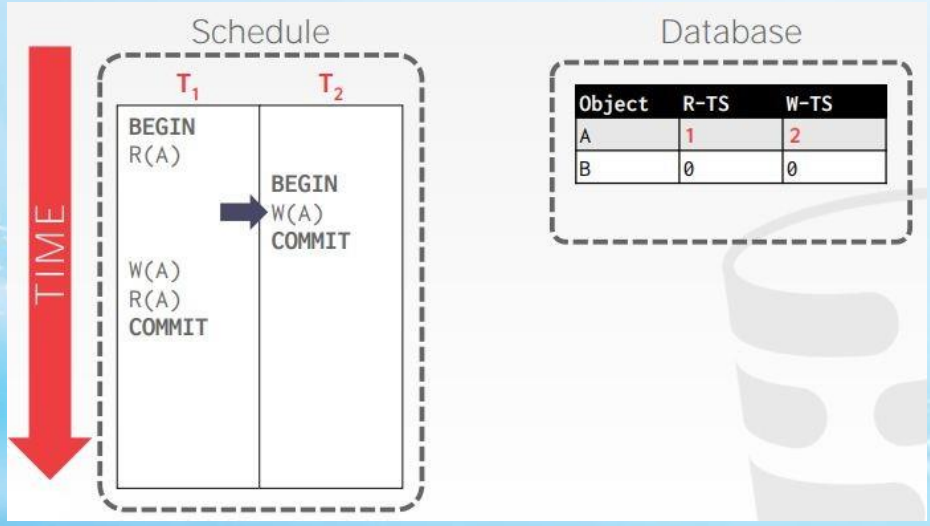
→ Allow **Ti** to write **X** and update **W-TS(X)**.

Concurrency Control

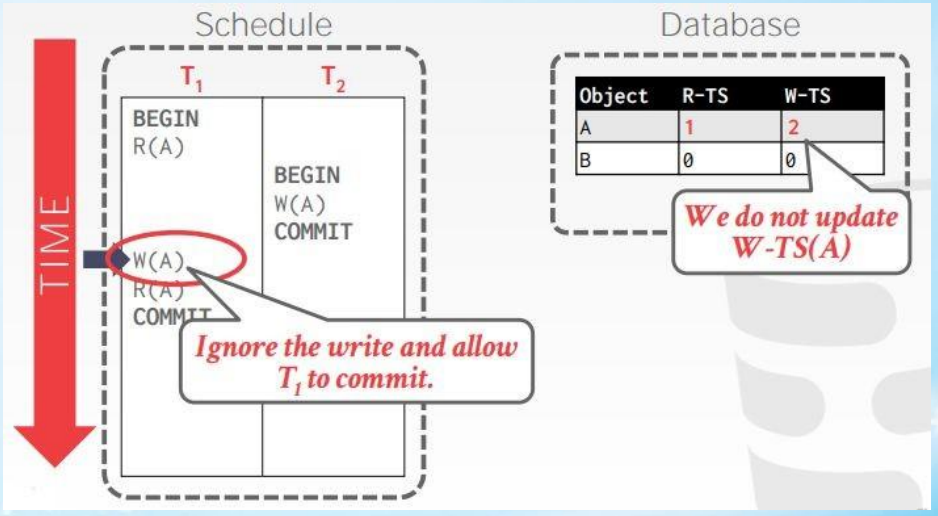
1



2



3



Concurrency Control

Validation Protocol

- A **concurrency control method** used to ensure **serializability** of transactions.
- Assumes that **conflicts between transactions are rare** — so it allows transactions to proceed without locking resources.
- Conflicts are only checked **at the end**, during a special **validation phase**, before the transaction commits.

Phases of Validation Protocol

There are **three phases**:

1. Read Phase

- The transaction reads data and makes changes to **local (temporary) variables**.
- No changes are made to the actual database yet.

2. Validation Phase

- Before committing, the system checks whether this transaction **conflicts with any other transactions** that committed during its lifetime.
- If **no conflict**, the transaction can proceed.
- If there is a **conflict**, the transaction is **aborted and restarted**.

3. Write Phase

- If validation is successful, all changes are written from temporary variables to the actual database.
- The transaction is **committed**.

Concurrency Control

Let's look at the timestamps of each phase of a transaction:

- **Start(Tn):** It represents the timestamp when the transaction Tn starts the execution.
- **Validation(Tn):** It represents the timestamp when the transaction Tn finishes the read phase and starts the validation phase.
- **Finish(Tn):** It represents the timestamp when the transaction Tn finishes all the write operations.

This protocol uses the Validation(Tn) as the timestamp of the transaction Tn because this is actual phase of the transaction where all the checks happen. So it is safe to say that TS(Tn) = Validation(Tn).

If there are two transactions T1 & T2 managed by validation based protocol and if Finish(T1) < Start(T2) then the validation will be successful as the serializability is maintained because T1 finished the execution well before the transaction T2 started the read phase.

Concurrency Control

Example:

We have two transactions: **T1** and **T2**

- **T1** starts at time = 1 and ends at time = 4
- **T2** starts at time = 2 and ends at time = 5

They both operate on a **data item X.**

Transaction Details:

T1:

- Reads X = 100
- Prepares to write X = 150 (but doesn't write yet)

T2:

- Reads X = 100
- Prepares to write X = 200 (but doesn't write yet)

So both transactions are working on **temporary copies** of X during their **Read Phase**.

Concurrency Control

Step-by-Step Execution Using Validation Protocol:

1. T1 Read Phase

- T1 starts first
- Reads X = 100
- Plans to write X = 150
- No issues — keeps changes in temporary space

2. T2 Read Phase

- T2 starts during T1's execution
- Reads X = 100
- Plans to write X = 200
- No issues yet — still in temporary space

3. T1 Validation Phase

- T1 finishes its read phase and begins validation
- No transactions finished before T1 → so no conflicts
- T1 passes validation

4. T1 Write Phase

- T1 writes X = 150 to the database
- Transaction T1 is **committed**

Concurrency Control

Step-by-Step Execution Using Validation Protocol:

5. T2 Validation Phase

- T2 now tries to validate
- It checks: Did any **committed transaction write to X** during T2's lifetime?
- YES! T1 wrote to X while T2 was still running
  → ✗ **Conflict Detected**

6. T2 is Aborted and Restarted

- Since T1 modified a data item that T2 also accessed, T2 **cannot be serialized after T1**
- So T2 is **aborted**, and the system will **restart it later**

Transaction	Phase	Action	Status
T1	Read	Reads X	✓
T1	Validate	No earlier commit → valid	✓
T1	Write	Writes X = 150	✓ Commit
T2	Read	Reads X	✓
T2	Validate	Conflict with committed T1	✗ Abort
T2	Write	Not reached	

Concurrency Control

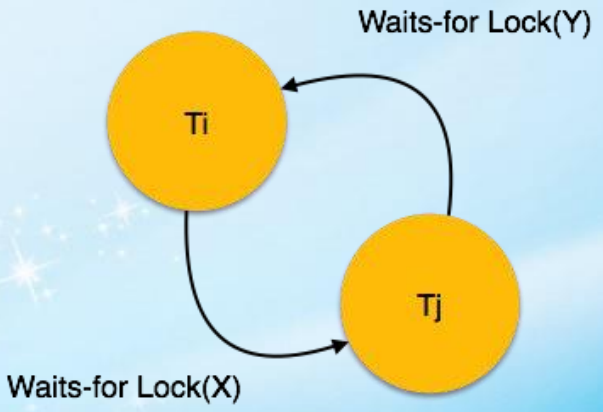
Deadlock

A system is in a deadlock stat if each transaction T in a set of two or more transactions is waiting for some item or resource that is locked by some other transaction T' in the set. Hence, each transaction in the set is on a waiting queue, waiting for one of the other transitions in the set to release the lock on an item.

Figure shows the two transaction T1 and T2 are deadlocked in a partial schedule. T1 is on the waiting queue for item X, which is locked by T2 and T2 is on waiting queue for Y which is locked by T1.

Time

T1	T2
Read_Lock(Y);	
Write_Lock(Y);	
	Read_Lock(X);
	Write_Lock(X);
Write_Lock(X);	
	Write_Lock(Y);



Concurrency Control

Deadlock Avoidance

When a database is stuck in a deadlock state, then it is better to avoid the database rather than aborting or restating the database. This is a waste of time and resource.

Deadlock avoidance mechanism is used to detect any deadlock situation in advance. A method like "wait for graph" is used for detecting the deadlock situation but this method is suitable only for the smaller database. For the larger database, deadlock prevention method can be used.

Deadlock Detection

In a database, when a transaction waits indefinitely to obtain a lock, then the DBMS should detect whether the transaction is involved in a deadlock or not.

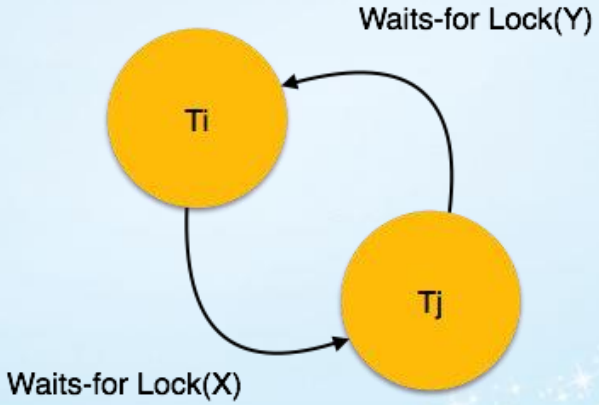
1. Wait for Graph
2. Timeout

Concurrency Control

1. Wait for Graph

- This is the suitable method for deadlock detection. In this method, a graph is created based on the transaction and their lock. If the created graph has a cycle or closed loop, then there is a deadlock.
- The wait for the graph is maintained by the system for every transaction which is waiting for some data held by the others. The system keeps checking the graph if there is any cycle in the graph.

The wait for a graph for the above scenario is shown below:



2. Timeout

- This method is practical because of its low overhead and simplicity.
- In this method, if a transaction waits for a period longer than a system-defined timeout period, the system assumes that the transaction may be deadlocked and aborts it regardless of whether a deadlock actually exists or not.

Concurrency Control

Deadlock Prevention

- To prevent any deadlock situation in the system, the DBMS aggressively inspects all the operations, where transactions are about to execute.
- The DBMS inspects the operations and analyzes if they can create a deadlock situation. If it finds that a deadlock situation might occur, then that transaction is never allowed to be executed.

There are deadlock prevention schemes that use timestamp ordering mechanism of transactions in order to predetermine a deadlock situation.

Wait-Die Scheme

In this scheme, if a transaction requests to lock a resource (data item), which is already held with a conflicting lock by another transaction, then one of the two possibilities may occur:

- If $TS(T_i) < TS(T_j)$ − that is T_i, which is requesting a conflicting lock (T_i is older than T_j)− then T_i is allowed to wait until the data-item is available.
- If $TS(T_i) > TS(T_j)$ − that is T_i is younger than T_j − then T_i is rolled back (dies). T_i is restarted later with a random delay but with the same timestamp.

This scheme allows the older transaction to wait but kills the younger one.

Concurrency Control

Wound-Wait Scheme

In this scheme, if a transaction requests to lock a resource (data item), which is already held with conflicting lock by some another transaction, one of the two possibilities may occur −

- If $TS(T_i) < TS(T_j)$, then T_i forces T_j to be rolled back − that is T_i wounds T_j. T_j is restarted later with a random delay but with the same timestamp.
- If $TS(T_i) > TS(T_j)$, then T_i is forced to wait until the resource is available.

This scheme, allows the younger transaction to wait; but when an older transaction requests an item held by a younger one, the older transaction forces the younger one to abort and release the item.

In both the cases, the transaction that enters the system at a later stage is aborted.

Recovery Management

Recovery and Atomicity

When a system crashes, it may have several transactions being executed and various files opened for them to modify the data items. Transactions are made of various operations, which are atomic in nature. But according to ACID properties of DBMS, atomicity of transactions as a whole must be maintained, that is, either all the operations are executed or none.

When a DBMS recovers from a crash, it should maintain the following:

- It should check the states of all the transactions, which were being executed.
- A transaction may be in the middle of some operation; the DBMS must ensure the atomicity of the transaction in this case.
- It should check whether the transaction can be completed now or it needs to be rolled back.
- No transactions would be allowed to leave the DBMS in an inconsistent state.

There are two types of techniques, which can help a DBMS in recovering as well as maintaining the atomicity of a transaction −

1. Maintaining the logs of each transaction, and writing them onto some stable storage before actually modifying the database.
2. Maintaining shadow paging, where the changes are done on a volatile memory, and later, the actual database is updated.

Recovery Management

Recovery from failure means that the database is restored to the most consistent state just before the tome of failure. To do this, the system must keep the information about the changes that were applied to data items by the various transactions. This information is kept in the System Log.

Recovery may me summarized by the following two strategies:

- **Recovery from Catastrophic Failure**
  If there is an extensive damage to a wide portion of the database due to catastrophic failure, such as a disk crash, the recovery method store a past copy of the database that was backed up to archival storage and reconstructs a more consistent state by reapplying or redoing the operations of committed transactions from the backed up log, up to the time of failure.

- **Recovery from non-catastrophic failures**
  When database is not physically damage but has become inconsistent due to non-catastrophic failures. The strategy is to reverse any changes caused the inconsistency by undoing some operations.

Recovery Management

There are two major techniques for recovery from non-catastrophic transaction failures:

1. Deferred updates
2. Immediate updates.

Deferred Update

- This technique does not physically update the database on disk until a transaction has reached its commit point.
- Before reaching commit, all transaction updates are recorded in the local transaction workspace.
- During commit, the updates are first recorded persistently in the log and then written to the database.
- If a transaction fails before reaching its commit point, it will not have changed the database in any way so UNDO is not needed.
- It may be necessary to REDO the effect of the operations that are recorded in the local transaction workspace, because their effect may not yet have been written in the database.
- Deferred update is also known as the **No-undo/redo algorithm.**
- It ensures transaction atomicity by recording all database modification in the log.
- All the write statements of the transactions are applied on the database only when the transaction is partially committed.
- A transaction is said to be partially committed once the final action of the transaction has been completed.

Recovery Management

Example:

The execution of transaction Ti proceeds as follows:

- Before Ti starts its execution, a record <Ti, Start> is written in the log.
- A write(X) operation by Ti results in the writing of a new record to the log as <Ti, Item, Value>.
- Finally, when Ti partially commits, a record <Ti, Commit> is written to the log.

For example, a banking system have accounts of A, B and C with initial balances 1000, 2000 and 700 respectively. We transfer 50 from account A to account B through transaction T_0. For this, we write

```
T0 : read(A);
     A := A-50;
     write(A);
     read(B);
     B :=B+50;
     write(B);
```

Let transaction T1 withdraws 100 from account C. Then transaction T1 can be defined as

```
T1: read(C);
    C := C-100;
    write(C);
```

Also, assume that these transactions executes serially in the order T_0 followed by T_1.

Recovery Management

Log record for these transactions will have values.

< T0 Start>
< T0, A, 950>
< T0, B, 2050)
< T0, Commit>
< T1 start>
< T1, C, 600>
< T1 Commit>

Using the log, the system can handle any failure that results in the loss of information on volatile storage.

Redo(Ti): sets the value of all data items updated by transaction Ti to the new values. The set of data items updated by Ti and their respective new values can be found in the log. The redo operation must be idempotent.

After a failure, the recovery subsystem consults the log to determine which transactions need to be redone. Transaction Ti is redone if and only if the log contains both <Ti Start> and <Ti Commit> statements.

Recovery Management

We again consider our banking example with transactions To and T1 executed one after the other in the order To followed T1.

A	B	C
< T0 Start> < T0, A, 950> < T0, B, 1050)	< T0 Start> < T0, A, 950> < T0, B, 1050) < T0, Commit> < T1 start> < T1, C, 600>	< T0 Start> < T0, A, 950> < T0, B, 1050) < T0, Commit> < T1 start> < T1, C, 600> < T1 Commit>

Case A:

If system fails just after the log record for the step write(B) of transaction T0 as shown in figure (A). Then, during recovery no redo operation will be done as we have only <T0 Start> in the log but not <T0 Commit>.

Case B:

If system crash occurs just after the log record write(C) as shown in figure (B). Then during recovery only T0 is done, as we have only <T0 Start> and <T0 Commit> in log disk. At the same time, we have <T1 Start> in the log but not <T1 Commit> so redo t1 will not be done.

Case C:

Similarly, if crash occurs just after the log record <T1 Commit> as shown in figure (C), the during recovery we will perform both redo(To) and redo(T1) as we have both <T0 Start> <T0 Commit> and <T1 Start> <T1 Commit> in the log disk.

Recovery Management

Immediate Update

- The database may be updated by some operations of a transaction before the transaction reaches its commit point.
- The operations are recorded forcibly in a log on disk before they are applied to the database, making recovery still possible.
- If a transaction fails to reach its commit point, the effect of its operation must be undone i.e. the transaction must be rolled back by undoing the effects of its operations on the database.
- It also requires to redo the effects of committed transaction.
- Immediate Update require both undo and redo. This technique is known as **undo/redo algorithm.**
- In this method for recovery, we use the following two operations:
  - ***Undo(Ti)* –** Restores the value of all data items updated by the transaction Ti to the old values.
  - ***Redo(Ti)* –** Sets the values of all data items updated by transaction Ti to the new values.
- We need to undo a transaction T only when log contains the record <T Start> but does not contains the <T Commit>.
- *We need to do redo transaction T only when log contains the record* <T Start> and <T Commit> both.

Recovery Management

We again consider our banking example with transactions To and T1 executed one after the other in the order To followed T1.

A	B	C
< T0 Start> < T0, A, 950> < T0, B, 1050)	< T0 Start> < T0, A, 950> < T0, B, 1050) < T0, Commit> < T1 start> < T1, C, 600>	< T0 Start> < T0, A, 950> < T0, B, 1050) < T0, Commit> < T1 start> < T1, C, 600> < T1 Commit>

Case A:

If system fails just after the log record for the step write(B) of transaction T0 as shown in figure (A). Then, during recovery we do undo(T0) operation as we have only <T0 Start> in the log but not <T0 Commit>.

Case B:

If system crash occurs just after the log record write(C) as shown in figure (B). Then during recovery, we do redo(T0) and undo(T1) as we have only <T0 Start> and <T0 Commit> in log disk. At the same time, we have <T1 Start> in the log but not <T1 Commit>. Undo(T1) should be done first than redo(T0) should be done.

Case C:

Similarly, if crash occurs just after the log record <T1 Commit> as shown in figure (C), the during recovery we will perform both redo(To) and redo(T1) as we have both <T0 Start> <T0 Commit> and <T1 Start> <T1 Commit> in the log disk.

Recovery Management

Recovery Process using Checkpoints

Recovery using the log records contains two major problems –

- The search process is time consuming & as one might conclude that recovery requires just the scanning of log as a whole for recent transactions.
- Most of the transactions that according to our algorithm, need to be redone have already written their updates into the database. Although redoing them will cause no harm, it will nevertheless cause recovery to take longer.

To reduce these types of overheads, we introduce checkpoints.

During execution, the DBMS maintains the log, but periodically performs checkpoints consisting of the following actions –

(i) Temporarily halting the initiation of new transaction until all the active ones arc committed or aborted.

(ii) Making a backup copy of the database.

(iii) Writing all records currently residing in primary memory to stable storage.

(iv) Appending to the end to the log a record indicating that the checkpoint has occurred, then writing it to the disk.

In this a failure has occurred, a recovery process examines the log to determine the most recent transaction Ti that started executing before the most recent checkpoint took place.

For this, we search log backward from the end of log until it finds the first checkpoint record; it continue backwards until it finds the next <Ti start> record.

Recovery Management

This record identifies a transaction Ti. For example, consider the set transactions {To, T1, ……., T100} executed in the order of the subscripts. Suppose the most recent checkpoint took place during the execution of transaction T80. Thus, only transaction T80, T81, T82, …… T100 to be considered during the recovery scheme.

Each of them need to be redone if it has committed otherwise, it needs be undone.

FILE ORGANIZATION

File Organization

A file organization is a way of arranging the records in a file when the file is stored on secondary storage (disk, tape etc.).

The different ways of arranging the records enable different operations to be carried out efficiently over the file.

A database management system supports several file organization techniques. The most important task of DBA is to choose a best - organization for each file, based on its use.

File Organization

The organization of records in a file is influenced by number of factors that must be taken into consideration while choosing a particular technique. These factors are:

a) fast retrieval, updation and transfer of records,

b) efficient use of disk space,

c) high throughput,

d) type of use,

e) efficient manipulation,

f) security from unauthorized access,

g) scalability,

h) reduction in cost,

i) protection from failure.

File & Record

- File is a collection of related sequence of records.
- A collection of field names and their corresponding data types constitutes a *record.*
- A *data type,* associated with each field, specifies the types of values a field can take. All records in a file are of the same record type.
- **Records and Record Types**

  Data is generally stored in the form of records. A record is a collection of fields or data items and data items is formed of one or more bytes. Each record has a unique identifier called record-id. The records in a file are one of the following two types :

  (i) Fixed Length records.

  (ii) Variable Length records.

Fixed Length Record

- Each record in the file has exactly of same size.
- The record slots are uniform and are arranged in continuous manner in a file.

Advantage

1. Insertion and deletion of records in the file are simple to implement since the space made available by a deleted record is same as needed to insert a new record.

Disadvantages

1. In fixed length records, since the length of record is fixed, it causes wastage of memory space. For example, if the length is set up to 50 characters and most of the are less than 25 characters, it causes wastage of precious memory space.
2. It is an inflexible approach. For example, if it is required to increase the length the record, then major changes in program and database are needed.

Variable Length Record

- Every record in the file need not be of the same size. Therefore, the records in the file have different sizes.

Advantage

1. It reduces manual mistakes as database automatically adjust the size of record
2. It saves lot of memory space in case of records of variable lengths.
3. It is a flexible approach since future enhancements are very easy to implement.

Disadvantages

1. It increases the overhead of DBMS because database have to keep record of the size of all records.

Types Of Files

The following three types of files are used in database systems :

1. Master file
2. Transaction file
3. Report file.

1. **Master file** : This file contains information of permanent nature about the **entities.** The master file act as a source of reference data for processing transactions. They accumulate the information based on the transaction data.

2. **Transaction file** : This file contains records that describe the activities carried out by the organization. This file is created as a result of processing transactions and preparing transaction documents. These are also used to update the master file permanently.

3. **Report file** : This file is created by extracting data from the different records to prepare a report *e.g.* A report file about the weekly sales of a particular item

File Organization Techniques

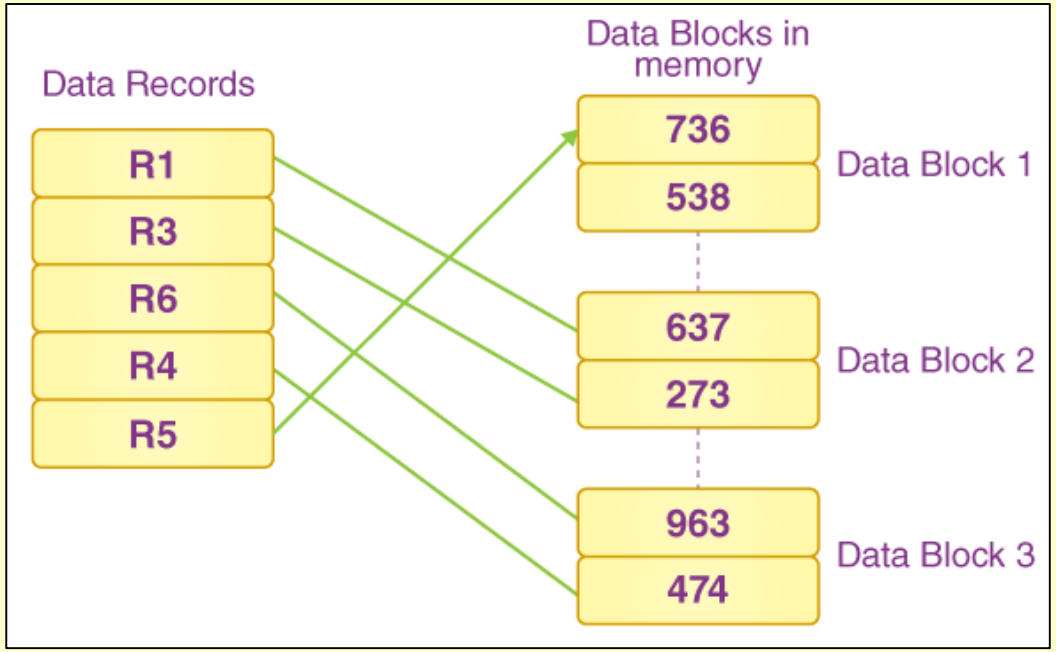
A file organization is a way of arranging the records in a file when the file is stored on secondary storage (disk, tape etc.). There are different types of file organizations that are used by applications. The operations to be performed and the selection of storage device are the major factors that influence the choice of a particular file organization. The different types of

file organizations are as follows :

1. Heap file organization
2. Sequential file organization
3. lndexed-Sequential. file organization
4. Hashing or Direct file organization

Heap File Organization

- A *heap file* is an unordered set of records.
- Heap File Organization is the most basic form of file organization, based on data chunks.
- In such an organization, records are stored in the file in the order in which they are inserted, and new records are always placed at the end of the file.
- In the file, every record has a unique id, and every page in a file is of the same size. It is the DBMS responsibility to store and manage the new records.



Heap File Organization

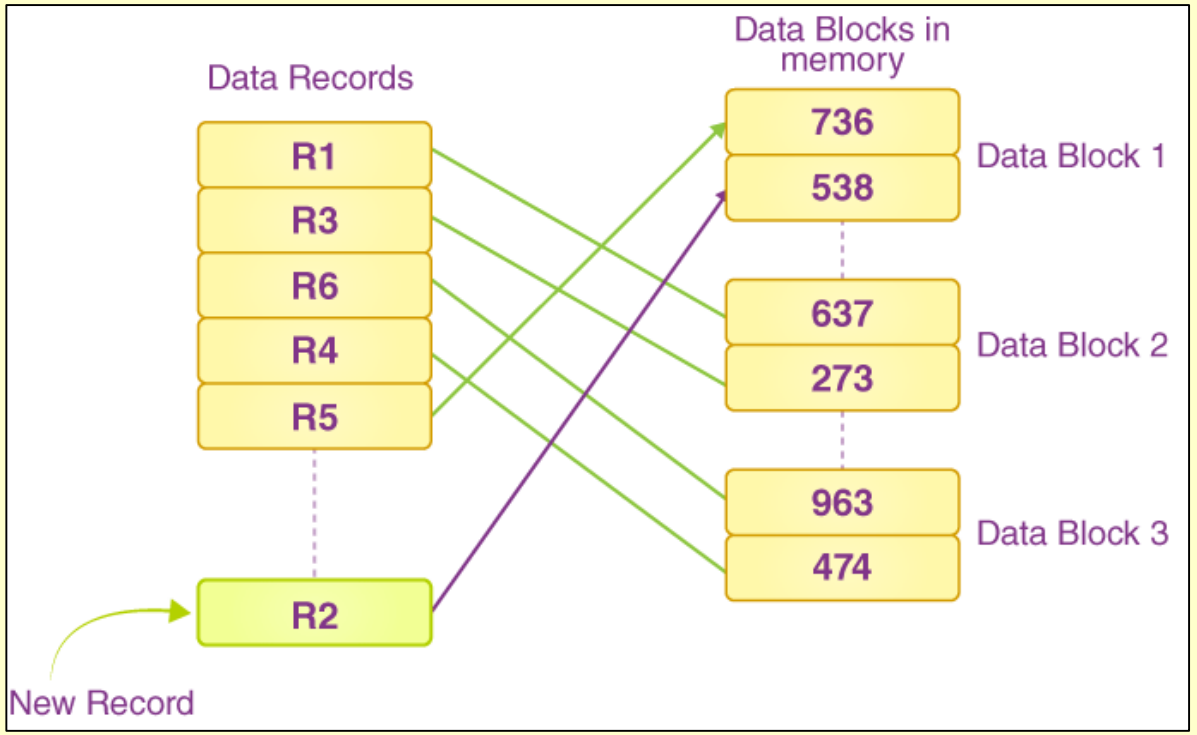
❑ Inserting a New Record

The insertion of a new record is very efficient. It is performed in the following steps:

- The last disk block of the file is copied into a buffer.
- The new record is added.
- The block in the buffer is then rewritten back to the disk.

Let's consider a scenario where we have five records in a heap, R1, R3, R6, R4, and R5, and we need to add a new record, R2.

If data block 3 is filled, the DBMS will place it in a selected database, such as data block 1.



Heap File Organization

- **Accessing a record**, in this method, we need to traverse from the beginning of the file till we get the requested record. Hence fetching the records in very huge tables, it is time consuming. This is because there is no sorting or ordering of the records. We need to check all the data.

- **Delete or update a record**, first we need to search for the record. Again, searching a record is similar to retrieving it-start from the beginning of the file till the record is fetched. If it is a small file, it can be fetched quickly. But larger the file, greater amount of time needs to be spent in fetching.

- **In addition, while deleting a record**, the record will be deleted from the data block. But it will not be freed and it cannot be re-used. Hence as the number of record increases, the memory size also increases and hence the efficiency. For the database to perform better, DBA has to free this unused memory periodically.

Heap File Organization

Advantages

1. Insertion of new record is fast & efficient.
2. The filling factor of this file organization is 100%.
3. Space is fully utilized & conserved.

Disadvantages

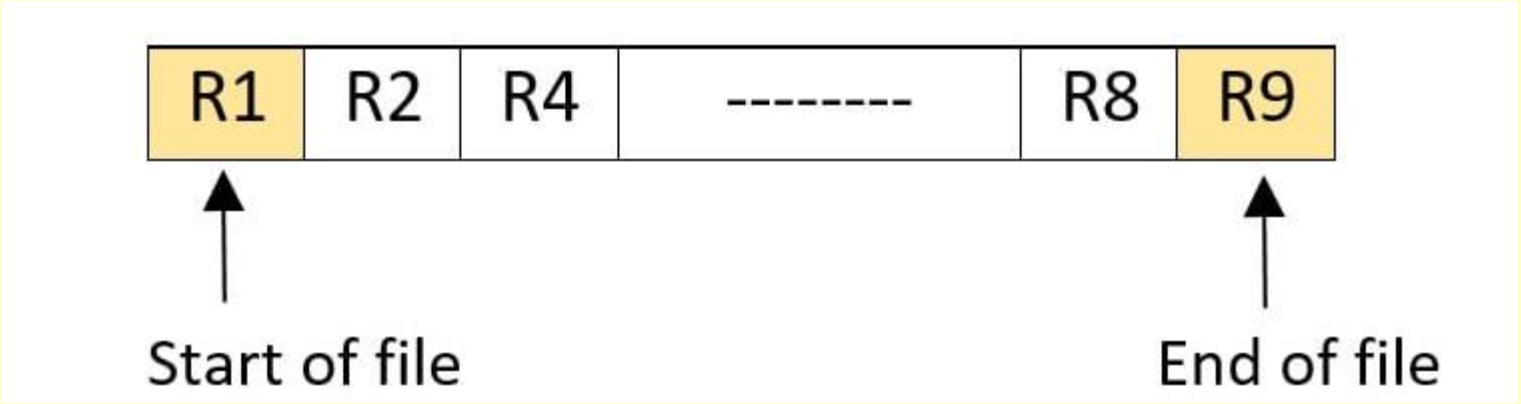
1. Searching & accessing of record is too slow.
2. Deletion of many records result in wastage of space.
3. Updation cost of data is comparatively high.

Sequential File Organization

- In sequential file organization, records are stored in a sequential order according to the "search key".
- A search Key is an attribute or a set of attributes which are used to serialize the records.
- It is not necessary that search key must be primary key.
- It is the simplest method of file organization.
- Sequential method is based on tape model. Devices who support sequential access are magnetic tapes, cassettes, card readers etc. Editors and compilers also use this approach to access files.

Sequential File Organization

- Structure of a sequential file is shown in Figure.
- The records are stored in sequential order one after another.
- To reach at the consecutive record from any record pointers are used. The Pointers are used for fast retrieval of records.
- Sequential file organization is a type of file organization used in database management systems (DBMS) where data is stored sequentially in a file or table.
- In this method, data is accessed sequentially in the order it is stored, starting from the beginning of the file and proceeding towards the end.



Sequential File Organization

- In a sequential file, each record is stored one after the other, without any index or key. This makes it easy to add new records to the end of the file, but searching for specific records can be slow and inefficient because the system has to search through the entire file to find the desired record.

- Sequential file organization is suitable for situations where data is accessed in a serial manner, such as batch processing or generating reports. It is also used in situations where data is not frequently updated, as adding or deleting a record can cause the entire file to be rewritten.

- However, this method is not efficient for situations that require frequent updates or random access to data. To improve the efficiency of accessing data in a sequential file, various techniques like buffering, caching, and memory-mapped files can be used.

Sequential File Organization

Advantages

1. It is easy to understand.
2. Efficient file system for small size files.
3. Construction and reconstruction are much easier in comparison to other files
4. Supports tape media, editors and compilers.
5. It contains sorted records.

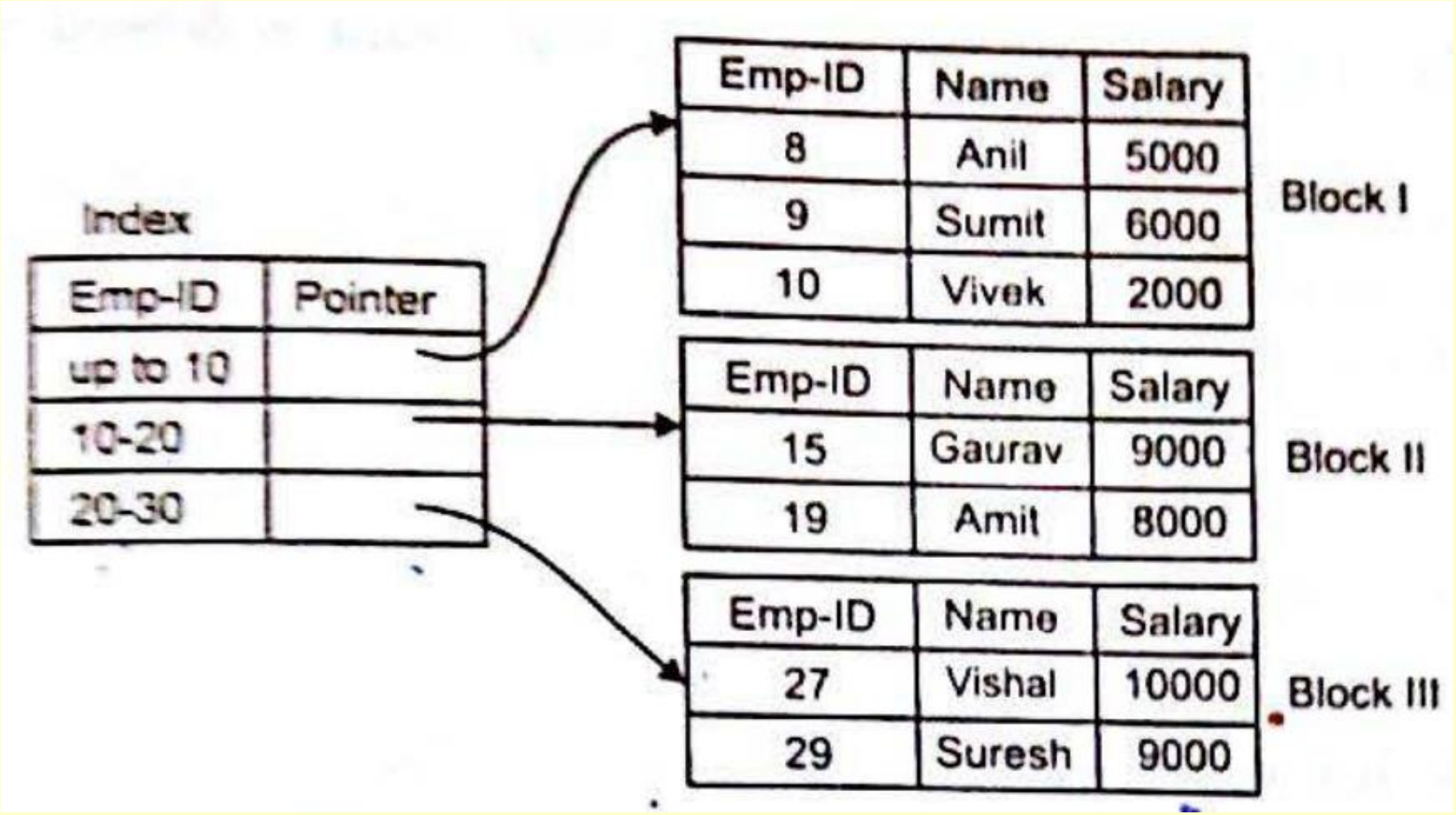
Disadvantages

1. Inefficient file system for medium and large size files.
2. Updations and maintenance are not easy.
3. Inefficient use of storage space because of fixed size blocks.
4. Linear search takes more time.
5. Before Updations all transactions are stored sequentially.

Indexed Sequential File Organization

- Index sequential file organization is used to overcome the disadvantages of sequential file organization.
- It also preserves the advantages of sequential access.
- This organization enables fast searching of records with the use of index.
- Some basic terms associated with indexed sequential file organization are as follows :
  - ***Block:*** Block is a unit of storage in which records are saved.
  - ***Index:*** Index is a table with a search key by which block of a record can be find.
  - ***Pointer:*** Pointer is a variable which points from index entry to starting address of block.

Indexed Sequential File Organization



Components Of Indexed Sequential File Organization

Data File: The records in the data file are stored in sorted order based on a key field.

Index File: The index file contains index entries, which are pointers to blocks (or records) in the data file. These index entries are also sorted based on the key.

Overflow Area: This is used to store new records that are inserted but cannot fit into the primary data area. The overflow area is also organized in a sorted manner.

Operations of ISAM

Searching: When searching for a record, the system first looks in the index to find the disk block that contains the record. Then, that specific block is fetched to retrieve the record. This usually takes two disk I/O operations.

Insertion: If a new record needs to be inserted, it is placed in its correct sorted position. If the block is full, the record is placed in the overflow area.

Deletion: Deleting a record involves marking it as deleted. The space is then reclaimed during a subsequent reorganization of the data file.

Updating: Updating a record involves either modifying it if the key is not changed, or deleting and re-inserting it if the key is modified.

Components Of Indexed Sequential File Organization

Example of ISAM

Let's consider a simplified library database where we have a file containing book information sorted by ISBN numbers.

Data File

ISBN	TITLE
123	Book A
234	Book B
345	Book C
456	Book D

Index File

ISBN	Block Address
123	1
234	2
345	3
456	4

Searching for ISBN 345

1. Search in the index for ISBN 345, and find that it is located at block 3.
2. Go to block 3 in the data file to retrieve the record for ISBN 345 ("Book C").

Components Of Indexed Sequential File Organization

Inserting a new book with ISBN 133

1. The new book would ideally go into block position 1 (between ISBN 123 and 234).
2. Since the block is full, the new record with ISBN 133 ("New Book") will go into the overflow area.
3. An index entry for ISBN 133 pointing to the overflow area is added.

Updated Overflow Area

ISBN	TITLE
133	New Book

Components Of Indexed Sequential File Organization

Advantages

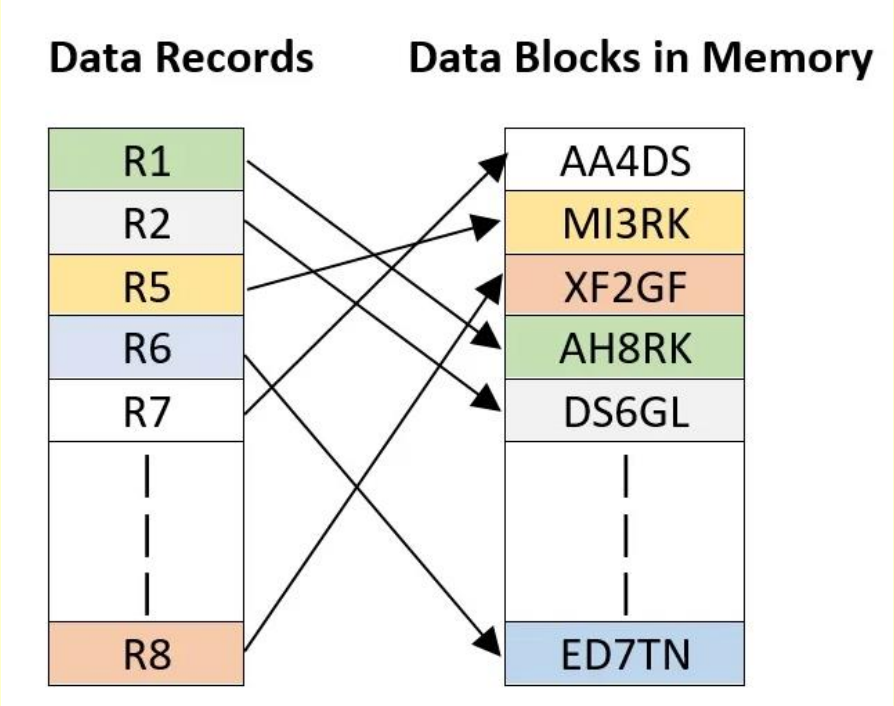
1. Efficient file system for medium and large size files.
2. Easy to update.
3. Easy to update than direct files.
4. Efficient use of storage.
5. Searching of records is fast.
6. Maintain advantages of sequential file.

Disadvantages

1. Inefficient file system for small size files.
2. It is an expensive method.
3. Typical structure than sequential file.
4. Indexes need additional storage space.
5. Performance degradation with respect to growth of files.

Hash File Organization

- Hash file organization is a type of file organization where data is stored in a file or table using a hash function.
- A hash function is a mathematical function that converts a key value into a hash code, which is used to map the key to the location in the file or table where the data is stored.



Hash File Organization

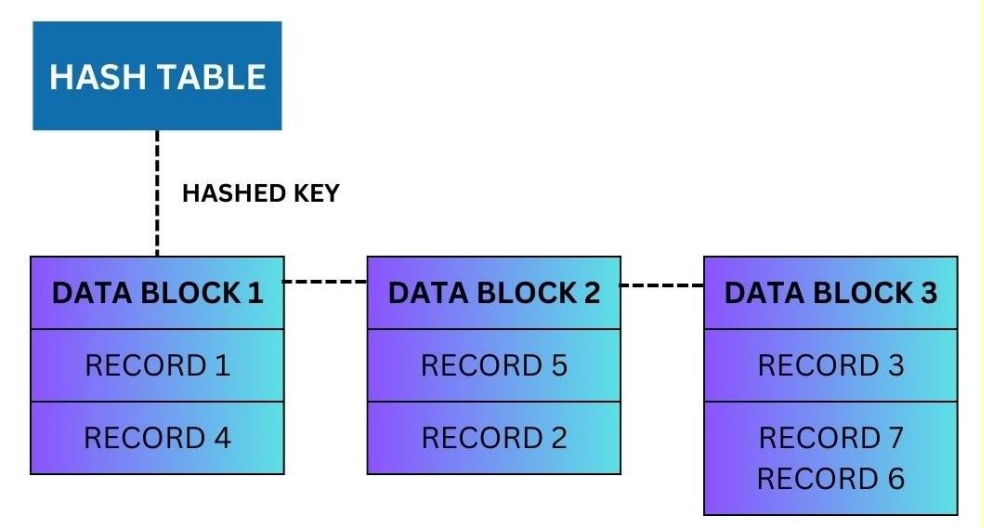
- Any type of mathematical function can be used as a hash function. It can be simple or complex.
- Hash function is applied to columns or attributes to get the block address. The records are stored randomly. So, it is also known as Direct or Random file organization.
- If the generated hash function is on the column which is considered as key, then the column can be called as hash key and if the generated hash function is on the column which is considered as non-key, then the column can be called as hash column.
- Hash file organization is suitable for situations where data needs to be accessed quickly and efficiently based on the value of its key, and where the data is not frequently updated. However, if the hash function is poorly designed or if there are collisions (where two different keys map to the same location), it can result in poor performance and decreased efficiency.

Hash File Organization

- In the diagram above, a hash table maps the hashed keys to a specific data block in memory. Data blocks are represented as rectangular boxes, with each box containing one or more data records. Each data record is shown as a separate row in the data block.

- When a new data record is inserted into the hash file, the hash function is used to hash its key, and the resulting hashed key is used to determine which data block it should be stored in. Then the data record is inserted, including the appropriate data block.

- In this example, Record 1 and Record 4 are stored in Data Block 1, Record 5 and Record 2 in Data Block 2, and Record 3, Record 7, and Record 6 are stored in Data Block 3, respectively.



Hash File Organization

- There are various hashing techniques. These are as follows :
  - Mid Square Method
  - Folding Method
  - Division
  - Division-Remainder Method
  - Radix Information Method
  - Polynomial Conversion Method
  - Truncation Method
  - Conversion using Digital Gates

Direct File Organization

- Direct File Organization is used to access the records of file randomly.
- In direct file organization, records can be stored anywhere in storage area, but can be accessed directly without any sequential searching.
- It overcomes the drawbacks of sequential, indexed sequential and B-tree organization.
- For efficient organization and direct access of individual record, some mapping and transformation process is required that converts key field of a record into its physical address location.
- Actually, direct file organization depends upon the hashing that provides the base of mapping procedure.
- To overcome the drawbacks of hashing algorithm, collision resolution techniques are needed. Devices that support direct access are CD’s, Floppy etc.
- Direct file organization is known as Random File Organization.

Direct File Organization

- **Searching (Reading or retrieving from direct file)**: To read a record from direct file, just enter the key field of the record. With the help of hashing algorithm that key field is mapped into physical location of that record.
- **Updation of records in direct file:**
  1. **Adding a new record:** To add a new record in direct file, specify its key field. With the help of mapping procedure and collision resolution technique get the free address location for that record.
  2. **Deleting record from direct file:** To delete a record, first search that record and after, searching, change its status code to deleted or vacant.
  3. **Modify any record:** To modify any record, first search that record, then make the necessary modifications. Then re-write the modified record to the same location.

Direct File Organization

Advantages

1. Records are not needed to sorted in order during addition.
2. It gives fastest retrieval of records.
3. It gives efficient use of memory.
4. Supports fast storage devices.
5. Searching time depends upon mapping procedure not logarithm of the number of search keys as in B tree.

Disadvantages

1. Wastages of space if hashing method is not chosen properly.
2. It does not support sequential storage devices.
3. Direct file system is complex and expensive.
4. Extra overhead due to collision resolution techniques.

Indexing

- Index is a collection of data entries which is used to locate a record in a file.
- Index table records consist of two parts, the first part consists of value of prime or non-prime attributes of file record known as indexing field. and, the second part consists of a pointer to the location where the record is physically stored in memory.
- In general, index table is like the index of a book, that consists of the name of topic and the page number.
- During searching of a file record, index is searched to locate the record memory address instead of searching a record in secondary memory.

Indexing

- On the basis of properties that affect the efficiency of searching, the indexes can be classified into two categories.
  1. Ordered indexing
  2. Hashed indexing

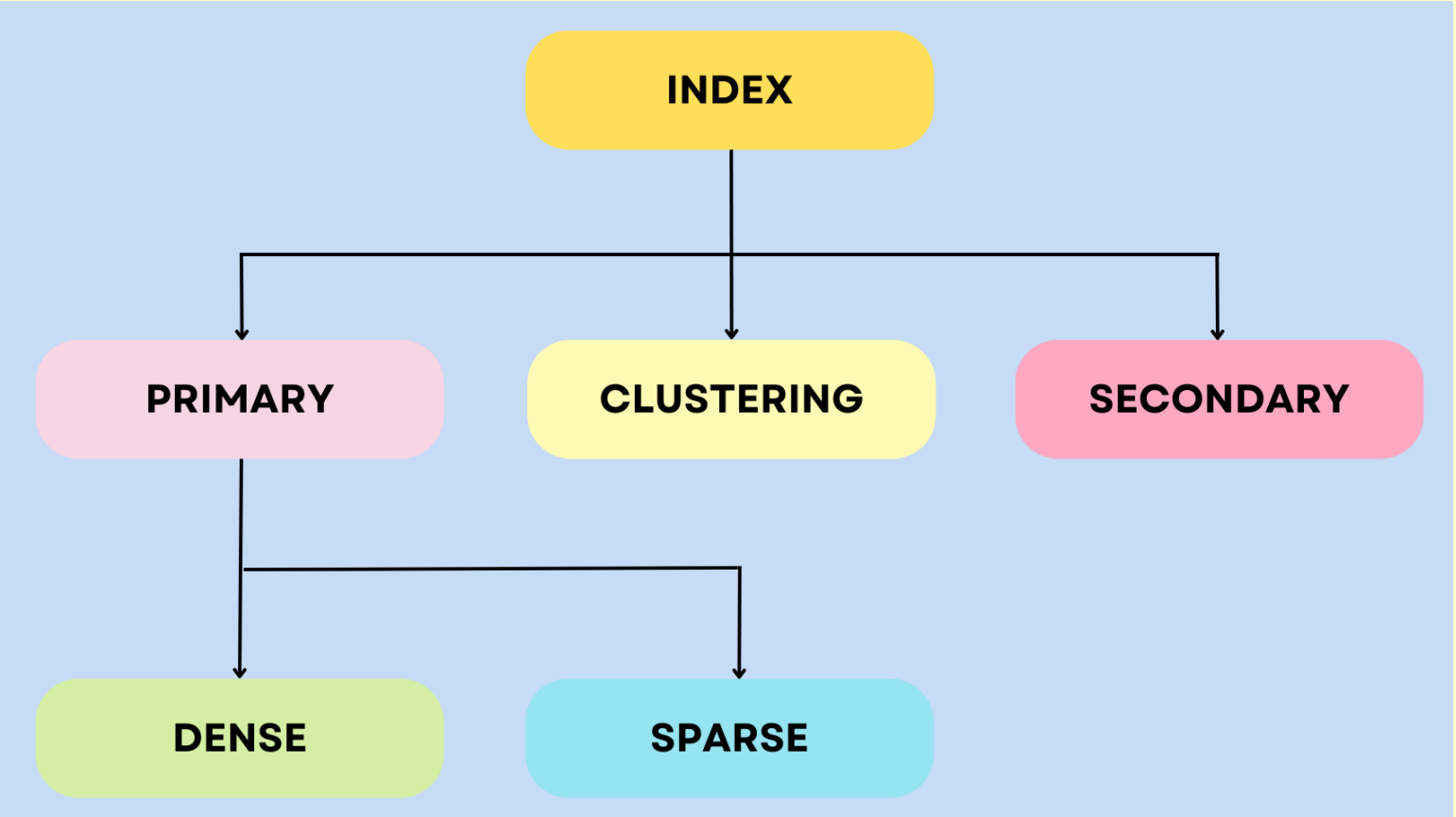
- Ordered Indexing In ordered indexing, records of file are stored in some sorted order in physical memory. The values in the index are ordered (sorted) so that binary search can be performed on the index.

- Hashing allow us to avoid accessing an index structure. A hashed index consists of two fields, the first field consists of search key attribute values and second field consists of pointer to the hash file structure. Hashing index is based on values of records being uniformly distributed using a hashed function.

Ordering Indexing

Ordered indexes can be divided into two categories.

1. Dense indexing
2. Sparse indexing.



Dense and Sparse Indexes

Dense index : In dense indexing there is a record in index table for each unique value of the search-key attribute of file and a pointer to the first data record with that value. The other records with the same value of search-key attribute are stored sequentially after the first record. The order of data entries in the index differs from the order of data records as shown in Figure.

Advantages

1. It is efficient technique for small and medium sized data files.
2. Searching is comparatively fast and efficient.

Disadvantages

1. Index table is large and require more memory space.
2. Insertion and deletion is comparatively complex.
3. In-efficient for large data files.

Dense and Sparse Indexes

Sparse index : On contrary, in sparse indexing there are only some records in index table for unique values of the search-key attribute of file and a pointer to the first data record with that value. To search a record in sparse index we search for a value that is less than or equal to value in index for which we are looking. After getting the first record, linear search is performed to retrieve the desired record. There is at most one sparse index since it is not possible to build a sparse index that is not clustered.

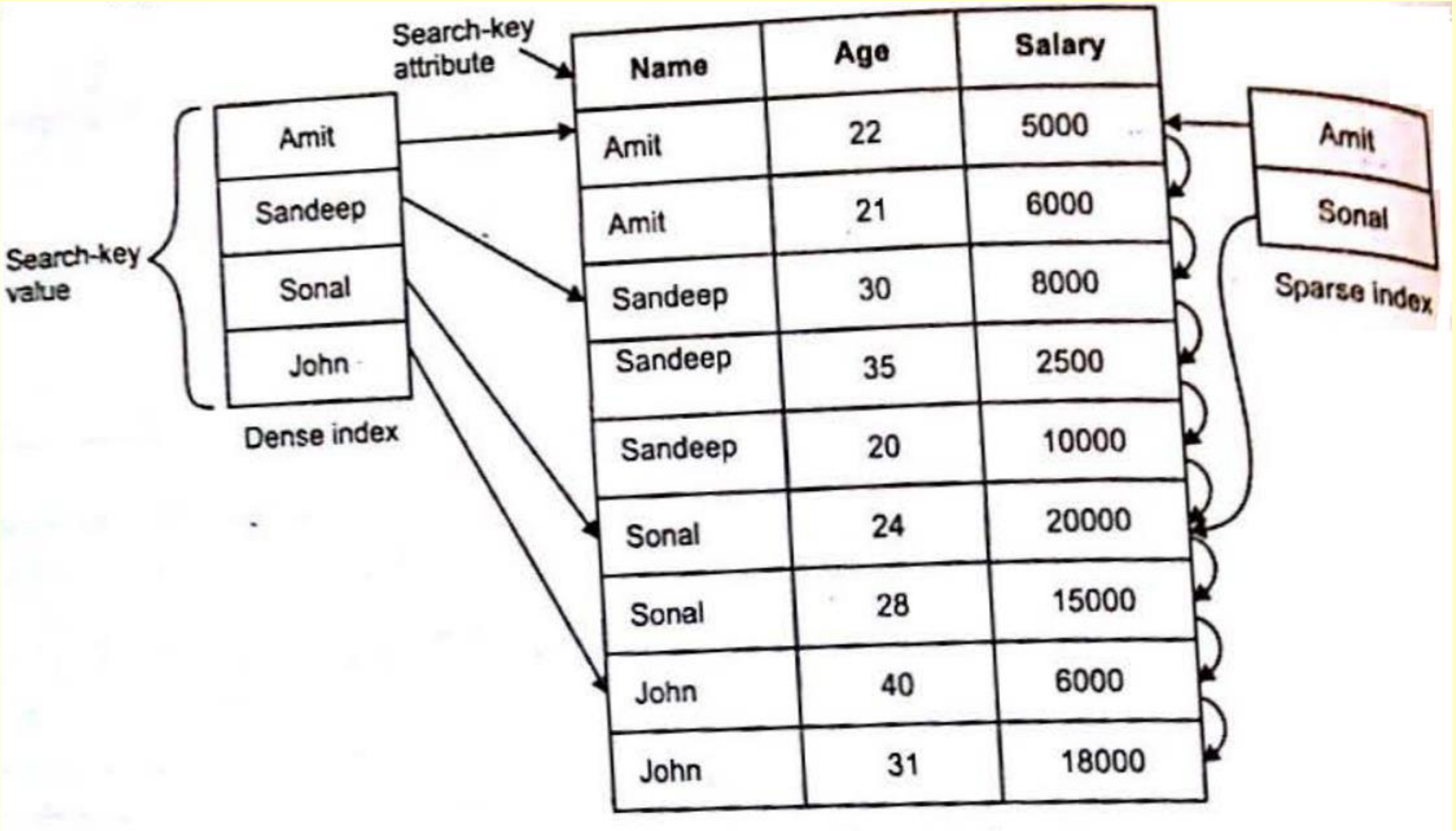
Advantages

1. Index table is small and hence save memory space (specially in large files).
2. Insertion and deletion is comparatively easy.

Disadvantages

1. Searching is comparatively slower, since index table is searched and then linear search is performed inside secondary memory.

Dense and Sparse Indexes



Clustered and Non-clustered Indexes

Clustered index : In clustering index file, records are stored physically in order on a non-prime key. attribute that does not have a unique value for each record. The non-prime key field is known as clustering field and index is known as clustering index. It is same as dense index. A file can have at most one clustered index as it can be clustered on at most one search key attribute. It may be sparse.

Non-Clustered index : An index that is not clustered is known as non-clustered index. Data file can have more than one non-clustered index.