

Unit 1
Introduction

Introduction to DBMS

- DBMS stands for **Database Management System**.
- DBMS = Database + Management System.
- Database is a collection of data and Management System is a set of programs to store and retrieve those data.
- DBMS is a collection of inter-related data and set of programs to store & access those data in an easy and effective manner.

DBMS:-

- DBMS is a software that is used to manage the data. Some of the popular DBMS softwares are: MySQL, IBM Db2, Oracle,
- DBMS provides an interface to the user so that the operations on database can be performed using the interface.
- DBMS secure the data, that is the main advantage of DBMS over file system.
- DBMS also secures the data from unauthorised access as well as corrupt data insertions. It allows multiple users to access data simultaneously while maintaining the **data consistency and data integrity**.

DBMS allows following operations to the authorized users of the database:

Data Definition: Creation of table, table schema creation, removal of table definition etc. comes under data definition. It is basically a layout of the table and their relation with the other tables in the database. This allows to properly structure the data in such a way so that the data that is related or dependent on other data in real world can be represented the same way in database.

Data Modification: DBMS allows users to insert, update and delete the data from the tables. These tables contains rows and columns, where row represents a record of data while column represents attributes of the records. You can also bulk update the several records in DBMS with a single click.

Data Retrieval: DBMS allows users to fetch data from the database. **Searching and retrieval of data is fast in DBMS.** The size of the database doesn't impact this operation, on the other hand in file system, the size of the data can hugely impact the search operation efficiency.

User administration: DBMS also allows user management such as organizing users in different groups with different access levels. Granting users access to certain tables in database, revoking access from certain users etc. This allows the **admin of the database to efficiently manage the access to the database** and prevent unauthorised access to the databases.

Need of DBMS

Database systems are basically developed for large amount of data. When dealing with huge amount of data, there are two things that require optimization: **Storage of data** and **retrieval of data**.

Storage: According to the principles of database systems, the data is stored in such a way that it acquires lot less space as the redundant data (duplicate data) has been removed before storage. Let's take a layman example to understand this: In a banking system, suppose a customer is having two accounts, one is saving account and another is salary account. Let's say bank stores saving account data at one place (these places are called tables we will learn them later) and salary account data at another place, in that case if the customer information such as customer name, address etc. are stored at both places then this is just a wastage of storage (redundancy/ duplication of data), to organize the data in a better way the information should be stored at one place and both the accounts should be linked to that information somehow. The same thing we achieve in DBMS.

Fast Retrieval of data: Along with storing the data in an optimized and systematic manner, it is also important that we retrieve the data quickly when needed. Database systems ensure that the data is retrieved as quickly as possible.

Purpose of Database Systems

The **main purpose of database systems is to manage the data**. Consider a university that keeps the data of students, teachers, courses, books etc. To manage this data we need to store this data somewhere where we can add new data, delete unused data, update outdated data, retrieve data, to perform these operations on data we need a Database management system that allows us to store the data in such a way so that all these operations can be performed on the data efficiently.

DBMS applications

Applications where we use Database Management Systems are:

- **Telecom:** There is a database to keep track of the information regarding calls made, network usage, customer details etc. Without the database systems it is hard to maintain that huge amount of data that keeps updating every millisecond.
- **Industry:** Where it is a manufacturing unit, warehouse or distribution centre, each one needs a database to keep the records of ins and outs. For example distribution centre should keep a track of the product units that supplied into the centre as well as the products that got delivered out from the distribution centre on each day; this is where DBMS comes into picture.

- **Banking System:** For storing customer info, tracking day to day credit and debit transactions, generating bank statements etc. All this work has been done with the help of Database management systems. Also, banking system needs security of data as the data is sensitive, this is efficiently taken care by the DBMS systems.
- **Sales:** To store customer information, production information and invoice details. Using DBMS, you can track, manage and generate historical data to analyse the sales data.
- **Airlines:** To travel through airlines, we make early reservations, this reservation information along with flight schedule is stored in database. This is where the real-time update of data is necessary **as a flight seat reserved for one passenger should not be allocated to another passenger**, this is easily handled by the DBMS systems as the data updates are in real time and fast.
- **Education sector:** Database systems are frequently used in schools and colleges to store and retrieve the data regarding student details, staff details, course details, exam details, payroll data, attendance details, fees details etc. There is a large amount of inter-related data that needs to be stored and retrieved in an efficient manner.
- **Online shopping:** You must be aware of the online shopping websites such as Amazon, Flipkart etc. These sites store the product information, your addresses and preferences, credit details and provide you the relevant list of products based on your query. All this involves a Database management system. **Along with managing the vast catalogue of items, there is a need to secure the user private information such as bank & card details.** All this is taken care of by database management systems.

Advantages and Disadvantages of DBMS:

DBMS vs file System

Drawbacks of File system

- **Data redundancy:**
 - Data redundancy refers to the duplication of data,
 - Need more storage
 - Data redundancy often leads to higher storage costs
 - poor access time.
- **Data inconsistency:**
 - Data redundancy leads to data inconsistency, let's take the same example that we have taken above, a student is enrolled for two courses and we have student address stored twice, now let's say student requests to change his address, if the address is changed at one place and not on all the records then this can lead to data inconsistency.
- **Data Isolation:**

- Because data are scattered in various files, and files may be in different formats, writing new application programs to retrieve the appropriate data is difficult.
- **Dependency on application programs:**
 - Changing files would lead to change in application programs.
- **Atomicity issues:**
 - Atomicity of a transaction refers to “All or nothing”, which means either all the operations in a transaction executes or none.
 - It is **difficult to achieve atomicity in file processing systems**.
- **Data Security:**
 - Data should be secured from unauthorised access,
 - for example a student in a college should not be able to see the payroll details of the teachers, such kind of security constraints are difficult to apply in file processing systems.

Advantage of DBMS over file system

There are several advantages of Database management system over file system. Few of them are as follows:

- **No redundant data:**
 - Redundancy removed by data normalization. No data duplication saves storage and improves access time.
- **Data Consistency and Integrity:**
 - As we discussed earlier the root cause of data inconsistency is data redundancy, since data normalization takes care of the data redundancy, data inconsistency also been taken care of as part of it
- **Data Security:**
 - It is easier to apply access constraints in database systems so that only authorized user is able to access the data.
 - Each user has a different set of access thus data is secured from the issues such as identity theft, data leaks and misuse of data.
- **Privacy:**
 - Limited access means privacy of data. DBMS can grant and revoke access to the database on user level that ensures who is accessing which data. It also helps user to manage the constraints on database, this ensures which type of data can be entered into the table.
- **Easy access to data –**
 - Database systems manages data in such a way so that the data is easily accessible with fast response times. Even if the database size is huge, the DBMS can still provide faster access and updation of data.
- **Easy recovery:**
 - Since database systems keeps the backup of data, it is easier to do a full recovery of data in case of a failure. This is very useful especially for almost all the organizations, as the data maintained over time should not be lost during a system crash or failure.
- **Flexible:**

- Database systems are more flexible than file processing systems. DBMS systems are scalable,
- The database size can be increased and decreased based on the amount of storage required.
- It also allows addition of additional tables as well as removal of existing tables without disturbing the consistency of data.

Disadvantages of DBMS

- DBMS implementation cost is high compared to the file system
- Complexity: Database systems are complex to understand
- Performance: Database systems are generic, making them suitable for various applications. However this feature affect their performance for some applications

View of Data

View of data in DBMS

- Abstraction is one of the main features of database systems.
- Hiding irrelevant details from user and providing abstract view of data to users, helps in easy and efficient **user-database** interaction.
- The top level of that architecture is “view level”.
- The view level provides the “**view of data**” to the users and hides the irrelevant details such as data relationship, database schema, constraints, security etc from the user.

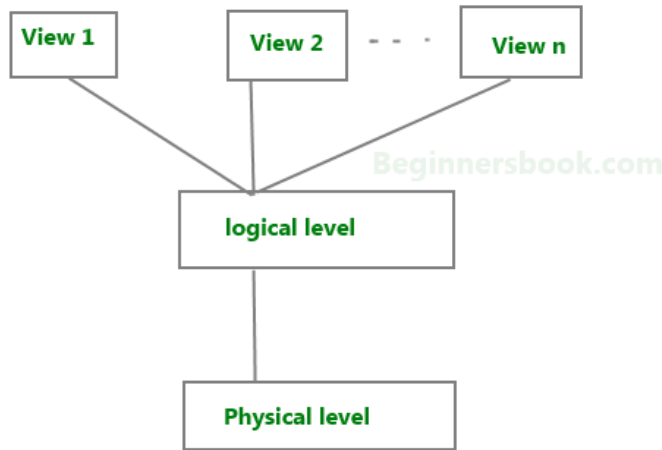
To fully understand the view of data, you must have a basic knowledge of data abstraction and instance & schema.

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.

1. Instance and schema:

- ✓ Design of a database is called the schema.
- ✓ Schema is of three types: Physical schema, logical schema and view schema.
- ✓ 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.

Three levels of abstraction



Three Levels of data 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.

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.

Instance and schema in DBMS

DBMS Schema

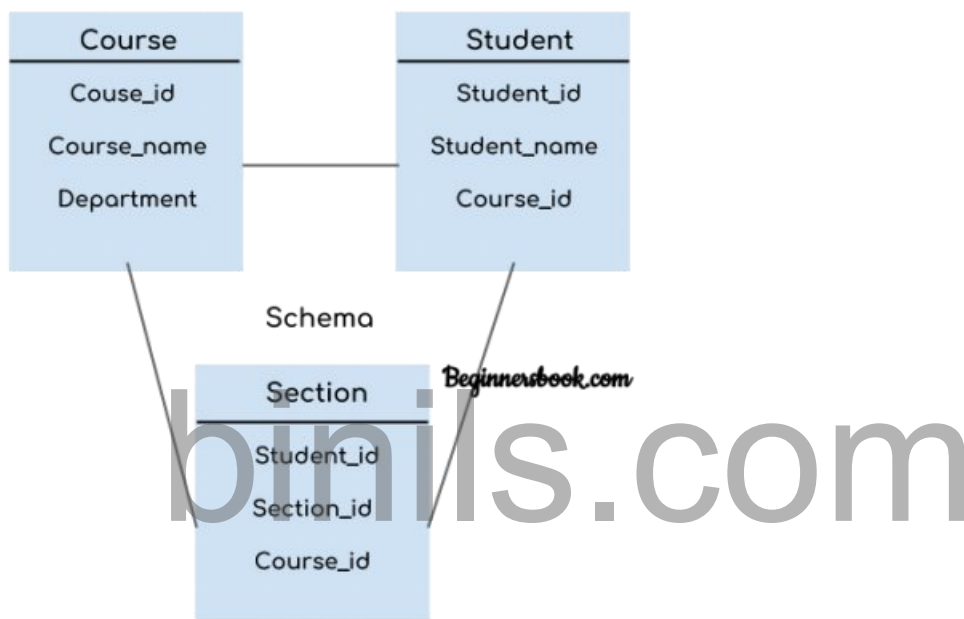
Definition of schema: Design of a database is called the schema. For example: An **employee** table in database exists with the following attributes:

EMP_NAME EMP_ID EMP_ADDRESS EMP_CONTACT

This is the schema of the **employee** table. Schema defines the attributes of tables in the database. **Schema is of three types: Physical schema, logical schema and view schema.**

- Schema represents the **logical view** of the database. It helps you understand what data needs to go where.
- Schema can be represented by a diagram as shown below.
- Schema **helps the database users to understand the relationship between data**. This helps in efficiently performing operations on database such as insert, update, delete, search etc.

In the following diagram, we have a schema that shows the relationship between **three tables: Course, Student and Section**. The diagram only shows the design of the database, it doesn't show the data present in those tables. Schema is only a structural view(design) of a database as shown in the diagram below.



The design of a database at physical level is called **physical schema**, how the data stored in blocks of storage is described at this level.

Design of database at logical level is called **logical schema**, 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, however the internal details such as implementation of data structure is hidden at this level (available at physical level).

Design of database at view level is called **view schema**. This generally describes end user interaction with database systems.

DBMS Instance

Definition of instance: The data stored in database at a particular moment of time is called instance of database. Database schema defines the attributes in tables that belong to a particular database. The value of these attributes at a moment of time is called the instance of that database.

For example, we have seen the schema of table “employee” above. Let’s see the table with the data now. At this moment the table contains two rows (records).

This is the the current instance of the table “employee” because this is the data that is stored in this table at this particular moment of time.

EMP_NAME	EMP_ID	EMP_ADDRESS	EMP_CONTACT

Chaitanya101	Noida95	*****	
Ajeet102	Delhi99	*****	

Let’s take another example: Let’s say we have a single table student in the database, today the table has 100 records, so today the instance of the database has 100 records. We are going to add another 100 records in this table by tomorrow so the instance of database tomorrow will have 200 records in table. In short, at a particular moment the data stored in database is called the instance, this changes over time as and when we add, delete or update data in the database.

DBMS languages

Database languages are used to read, update and store data in a database. There are several such languages that can be used for this purpose; one of them is SQL (Structured Query Language).

Types of DBMS languages:

- ✓ DDL Data Definition Language
- ✓ DCL Data Control Language
- ✓ DML Data Manipulation Language
- ✓ TCL Transaction Control Language

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.

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

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.

Transaction Control Language(TCL)

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

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

DBMS Architecture

The architecture of DBMS depends on the computer system on which it runs. For example, in a client-server DBMS architecture, the database systems at server machine can run several requests made by client machine. We will understand this communication with the help of diagrams.

Types of DBMS Architecture

There are three types of DBMS architecture:

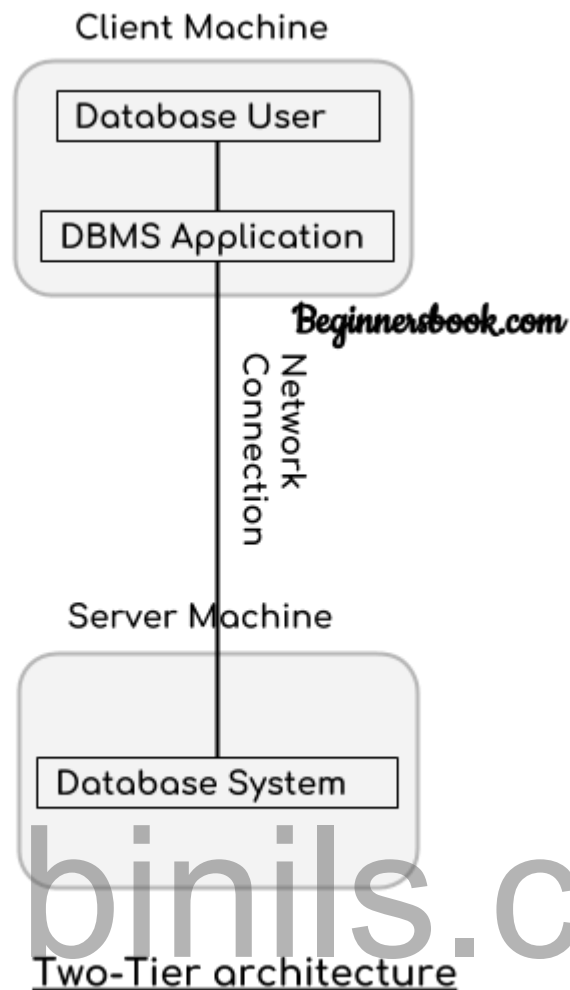
1. Single tier architecture
2. Two tier architecture
3. Three tier architecture

1. Single tier architecture

In this type of architecture, the database is readily available on the client machine, any request made by client doesn't require a network connection to perform the action on the database.

For example, let's say you want to fetch the records of employee from the database and the database is available on your computer system, so the request to fetch employee details will be done by your computer and the records will be fetched from the database by your computer as well. This type of system is generally referred as local database system.

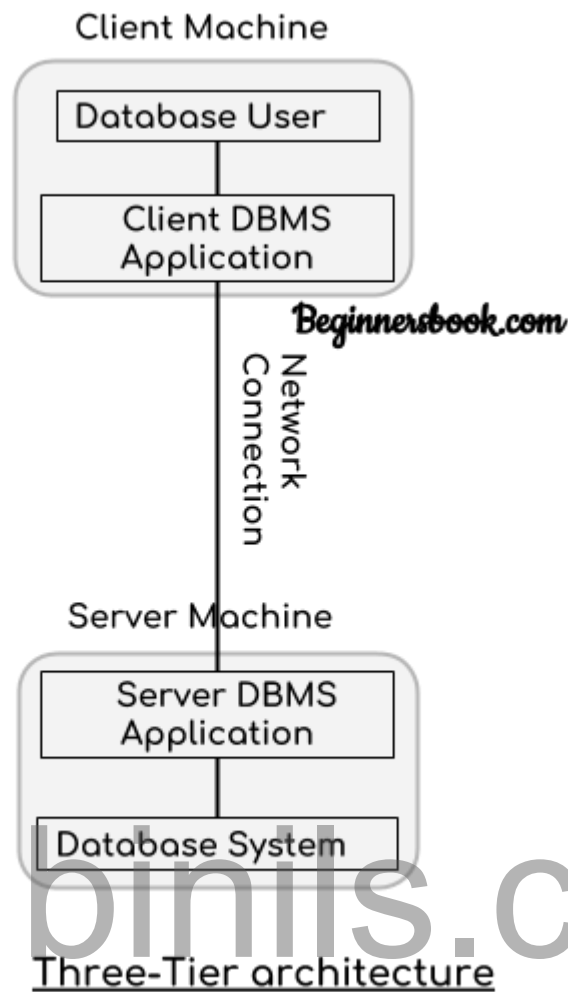
2. Two tier architecture



In two-tier architecture, the Database system is present at the server machine and the DBMS application is present at the client machine, these two machines are connected with each other through a reliable network as shown in the above diagram.

Whenever client machine makes a request to access the database present at server using a query language like sql, the server perform the request on the database and returns the result back to the client. The application connection interface such as JDBC, ODBC are used for the interaction between server and client.

3. Three tier architecture



In three-tier architecture, another layer is present between the client machine and server machine. In this architecture, the client application doesn't communicate directly with the database systems present at the server machine, rather the client application communicates with server application and the server application internally communicates with the database system present at the server.

Data models in DBMS

Types of Data Models

There are several types of data models in DBMS. We will cover them in detail in separate articles (Links to those separate tutorials are already provided below). In this guide, we will just see a basic overview of types of models.

Object based logical Models – Describe data at the conceptual and view levels.

1. E-R Model
2. Object oriented Model

Record based logical Models – Like Object based model, they also describe data at the conceptual and view levels. These models specify logical structure of database with records, fields and attributes.

1. Relational Model
2. Hierarchical Model
3. Network Model – Network Model is same as hierarchical model except that it has graph-like structure rather than a tree-based structure. Unlike hierarchical model, this model allows each record to have more than one parent record.

Physical Data Models – These models describe data at the lowest level of abstraction.

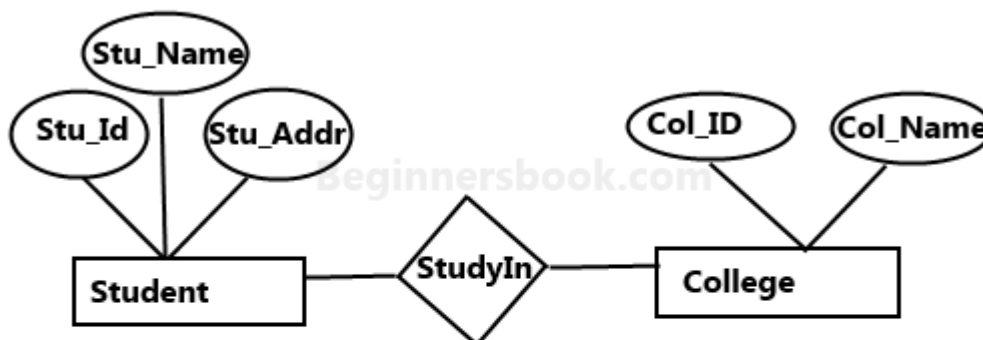
Entity Relationship Diagram – ER Diagram in DBMS

An **Entity–relationship model (ER model)** describes the structure of a database with the help of a diagram, which is known as **Entity Relationship Diagram (ER Diagram)**. An ER model is a design or blueprint of a database that can later be implemented as a database. The main components of E-R model are: entity set and relationship set.

Entity Relationship Diagram (ER Diagram)

An ER diagram shows the relationship among entity sets. An entity set is a group of similar entities and these entities can have attributes. In terms of DBMS, an entity is a table or attribute of a table in database, so by showing relationship among tables and their attributes, ER diagram shows the complete logical structure of a database. Lets have a look at a simple ER diagram to understand this concept.

A simple ER Diagram:



Sample E-R 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.

Here are the geometric shapes and their meaning in an E-R Diagram. We will discuss these terms in detail in the next section(Components of a ER Diagram) of this guide so don't worry too much about these terms now, just go through them once.

Rectangle:Represents Entity sets.

Ellipses: Attributes

Diamonds: Relationship Set

Lines: They link attributes to Entity Sets and Entity sets to Relationship Set

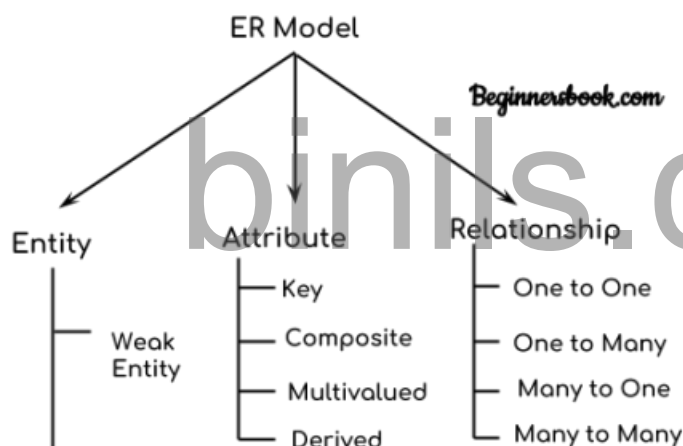
Double Ellipses: Multivalued Attributes

Dashed Ellipses: Derived Attributes

Double Rectangles: Weak Entity Sets

Double Lines: Total participation of an entity in a relationship set

Components of a ER Diagram



Components of ER Diagram

As shown in the above diagram, an ER diagram has three main components:

1. Entity
2. Attribute
3. Relationship

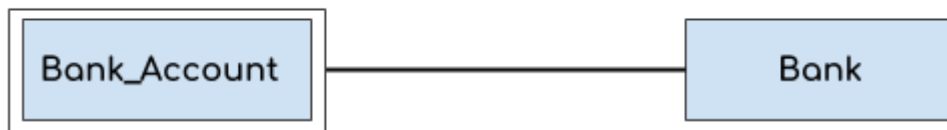
1. Entity

An entity is an object or component of data. An entity is represented as rectangle in an ER diagram.

For example: In the following ER diagram we have two entities Student and College and these two entities have many to one relationship as many students study in a single college. We will read more about relationships later, for now focus on entities.



Beginnerbook.com



Beginnerbook.com

Weak Entity:

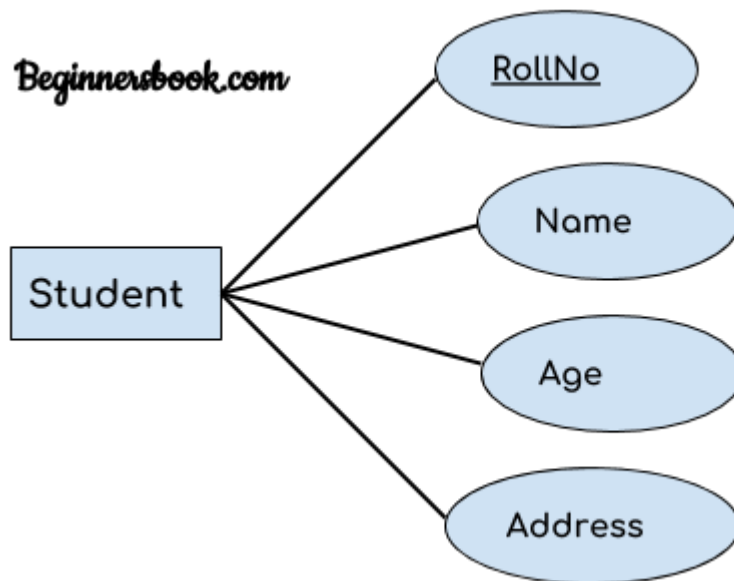
An entity that cannot be uniquely identified by its own attributes and relies on the relationship with other entity is called weak entity. The weak entity is represented by a double rectangle. For example – a bank account cannot be uniquely identified without knowing the bank to which the account belongs, so bank account is a weak entity.

2. Attribute

An attribute describes the property of an entity. An attribute is represented as Oval in an ER diagram. There are four types of attributes:

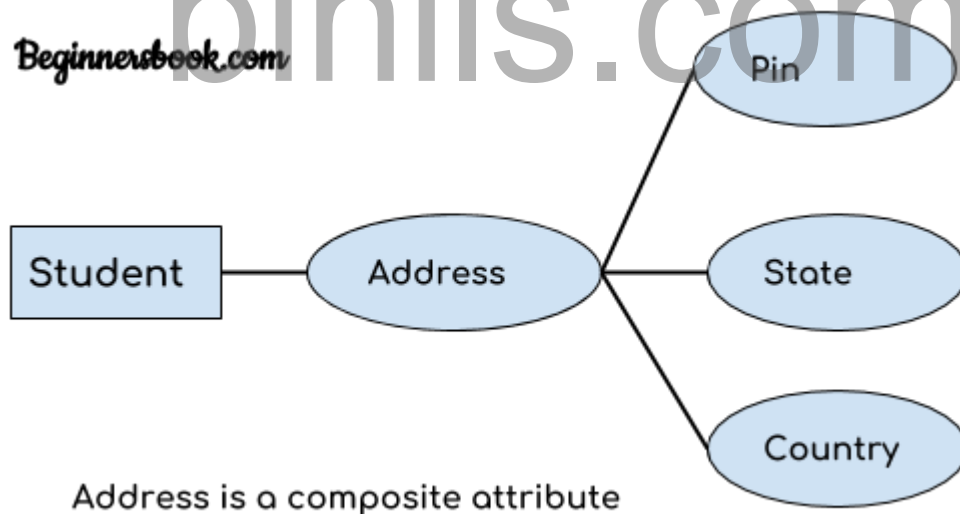
1. Key attribute
2. Composite attribute
3. Multivalued attribute
4. Derived attribute

1. Key attribute:



A key attribute can uniquely identify an entity from an entity set. For example, student roll number can uniquely identify a student from a set of students. Key attribute is represented by oval same as other attributes however the **text of key attribute is underlined**.

2. Composite attribute:



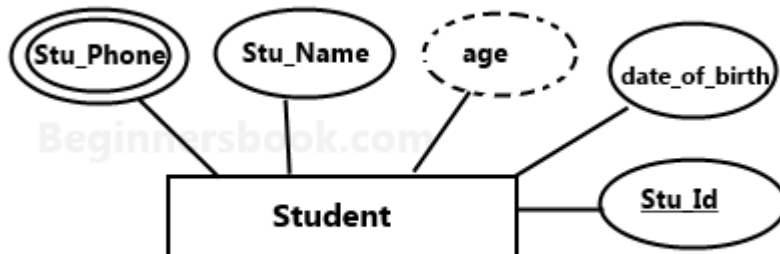
An attribute that is a combination of other attributes is known as composite attribute. For example, In student entity, the student address is a composite attribute as an address is composed of other attributes such as pin code, state, country.

3. Multivalued attribute:

An attribute that can hold multiple values is known as multivalued attribute. It is represented with **double ovals** in an ER Diagram. For example – A person can have more than one phone numbers so the phone number attribute is multivalued.

4. Derived attribute:

A derived attribute is one whose value is dynamic and derived from another attribute. It is represented by **dashed oval** in an ER Diagram. For example – Person age is a derived attribute as it changes over time and can be derived from another attribute (Date of birth).



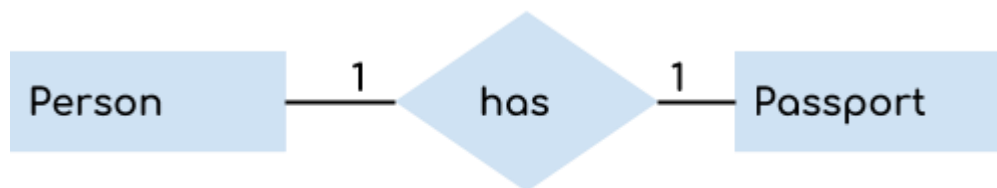
E-R diagram with multivalued and derived attributes:

3. Relationship

A relationship is represented by diamond shape in ER diagram, it shows the relationship among entities. There are four types of relationships:

1. One to One
2. One to Many
3. Many to One
4. Many to Many

1. One to One Relationship



Beginnerbook.com

When a single instance of an entity is associated with a single instance of another entity then it is called one to one relationship. For example, a person has only one passport and a passport is given to one person.

2. One to Many Relationship



Beginnerbook.com

When a single instance of an entity is associated with more than one instances of another entity then it is called one to many relationship. For example – a customer can place many orders but a order cannot be placed by many customers.

3. Many to One Relationship



Beginnerbook.com

When more than one instances of an entity is associated with a single instance of another entity then it is called many to one relationship. For example – many students can study in a single college but a student cannot study in many colleges at the same time.

4. Many to Many Relationship



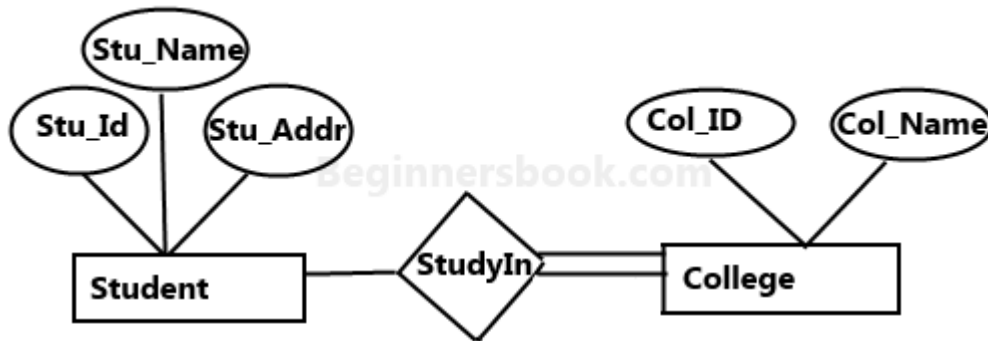
Beginnerbook.com

When more than one instances of an entity is associated with more than one instances of another entity then it is called many to many relationship. For example, a can be assigned to many projects and a project can be assigned to many students.

Total Participation of an Entity set

Total participation of an entity set represents that each entity in entity set must have at least one relationship in a relationship set. It is also called **mandatory participation**. For example: In the following diagram each college must have at-least

one associated Student. Total participation is represented using a **double line** between the entity set and relationship set.



E-R Diagram with total participation of College entity set in StudyIn relationship Set - This indicates that each college must have atleast one associated Student.

Partial participation of an Entity Set

Partial participation of an entity set represents that each entity in the entity set may or may not participate in the relationship instance in that relationship set. It is also called as **optional participation**

Partial participation is represented using a single line between the entity set and relationship set.

Example: Consider an example of an IT company. There are many employees working for the company. Let's take the example of relationship between **employee** and role **software engineer**. Every software engineer is an employee but not every employee is software engineer as there are employees for other roles as well, such as housekeeping, managers, CEO etc. so we can say that participation of employee entity set to the software engineer relationship is partial.

DBMS – ER Design Issues

1. Choosing Entity Setvs Attributes

Here we will discuss how choosing an entity set vs an attribute can change the whole ER design semantics. To understand this lets take an example, let's say we have an entity set Student with attributes such as student-name and student-id. Now we can say that the student-id itself can be an entity with the attributes like student-class and student-section.

Now if we compare the two cases we discussed above, in the first case we can say that the student can have only one student id, however in the second case when we chose student id as an entity it implied that a student can have more than one student id.

2. Choosing Entity Set vs. Relationship Sets

It is hard to decide that an object can be best represented by an entity set or relationship set. To comprehend and decide the perfect choice between these two (entity vs relationship), the user needs to understand whether the entity would need a

new relationship if a requirement arise in future, if this is the case then it is better to choose entity set rather than relationship set.

Let's take an example to understand it better: A person takes a loan from a bank, here we have two entities person and bank and their relationship is loan. This is fine until there is a need to disburse a joint loan, in such case a new relationship needs to be created to define the relationship between the two individuals who have taken joint loan. In this scenario, it is better to choose loan as an entity set rather than a relationship set.

3. Choosing Binary vs n-ary Relationship Sets

In most cases, the relationships described in an **ER diagrams** are binary. The **n-ary** relationships are those where entity sets are more than two, if the entity sets are only two, their relationship can be termed as binary relationship.

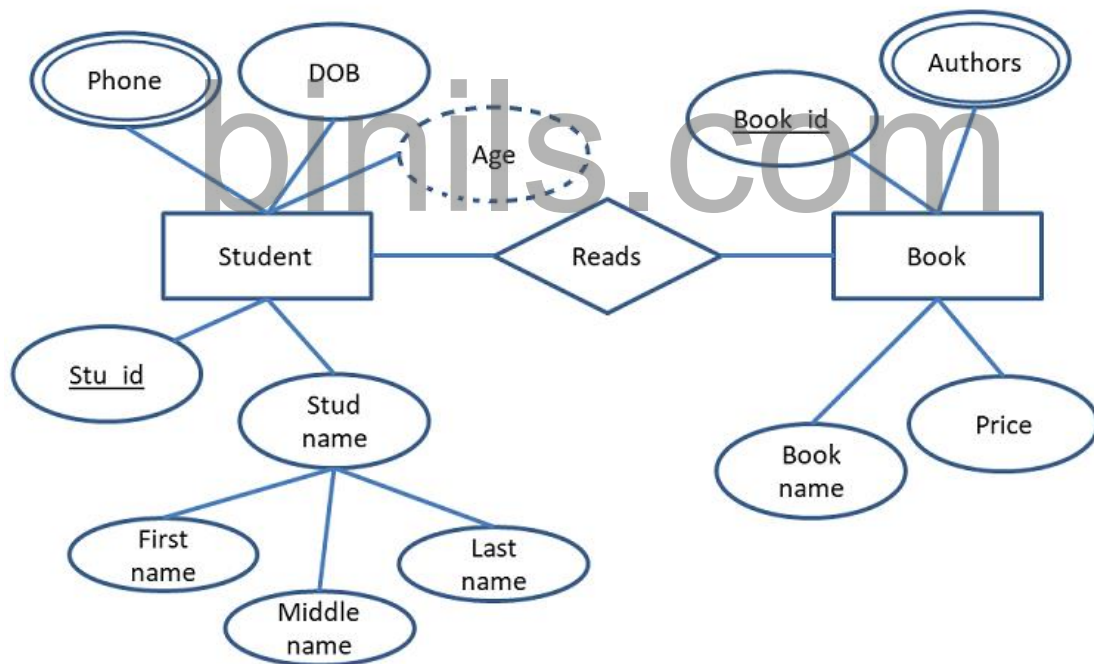
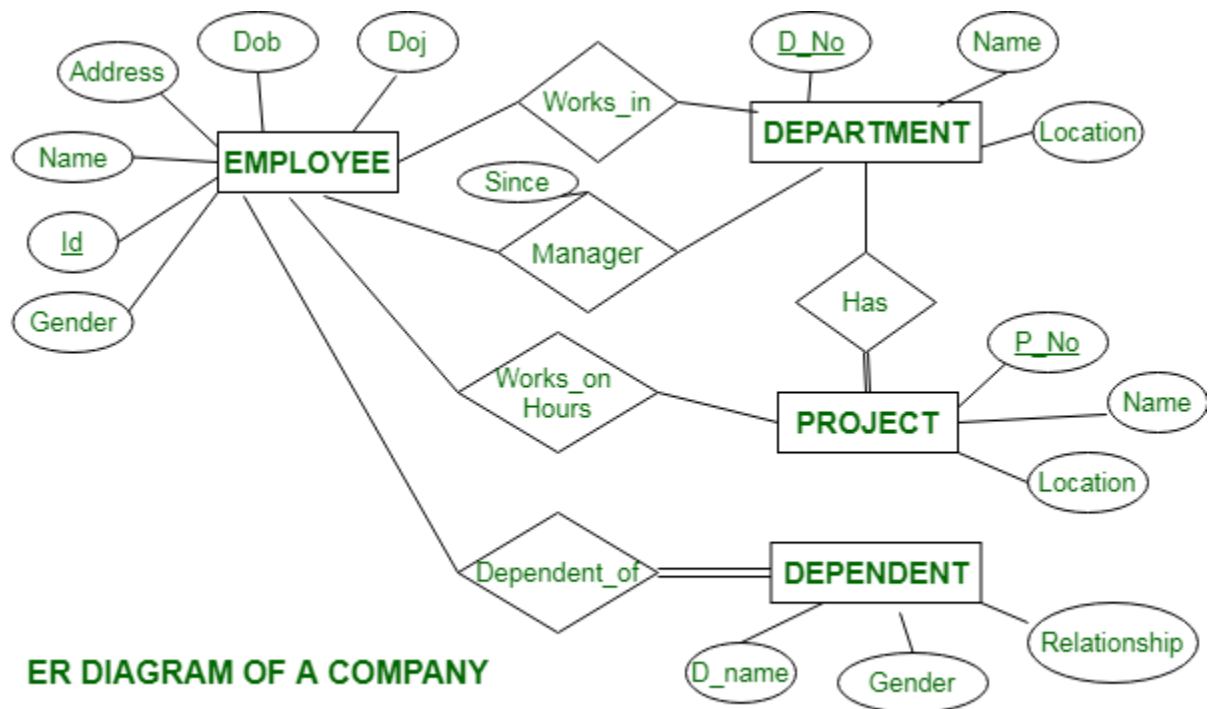
The n-ary relationships can make ER design complex, however the good news is that we can convert and represent any n-ary relationship using multiple binary relationships.

This may sound confusing so lets take an example to understand how we can convert an n-ary relationship to multiple binary relationships. Now lets say we have to describe a relationship between four family members: father, mother, son and daughter. This can easily be represented in forms of multiple binary relationships, father-mother relationship as "spouse", son and daughter relationship as "siblings" and father and mother relationship with their child as "child".

4. Placing Relationship Attributes

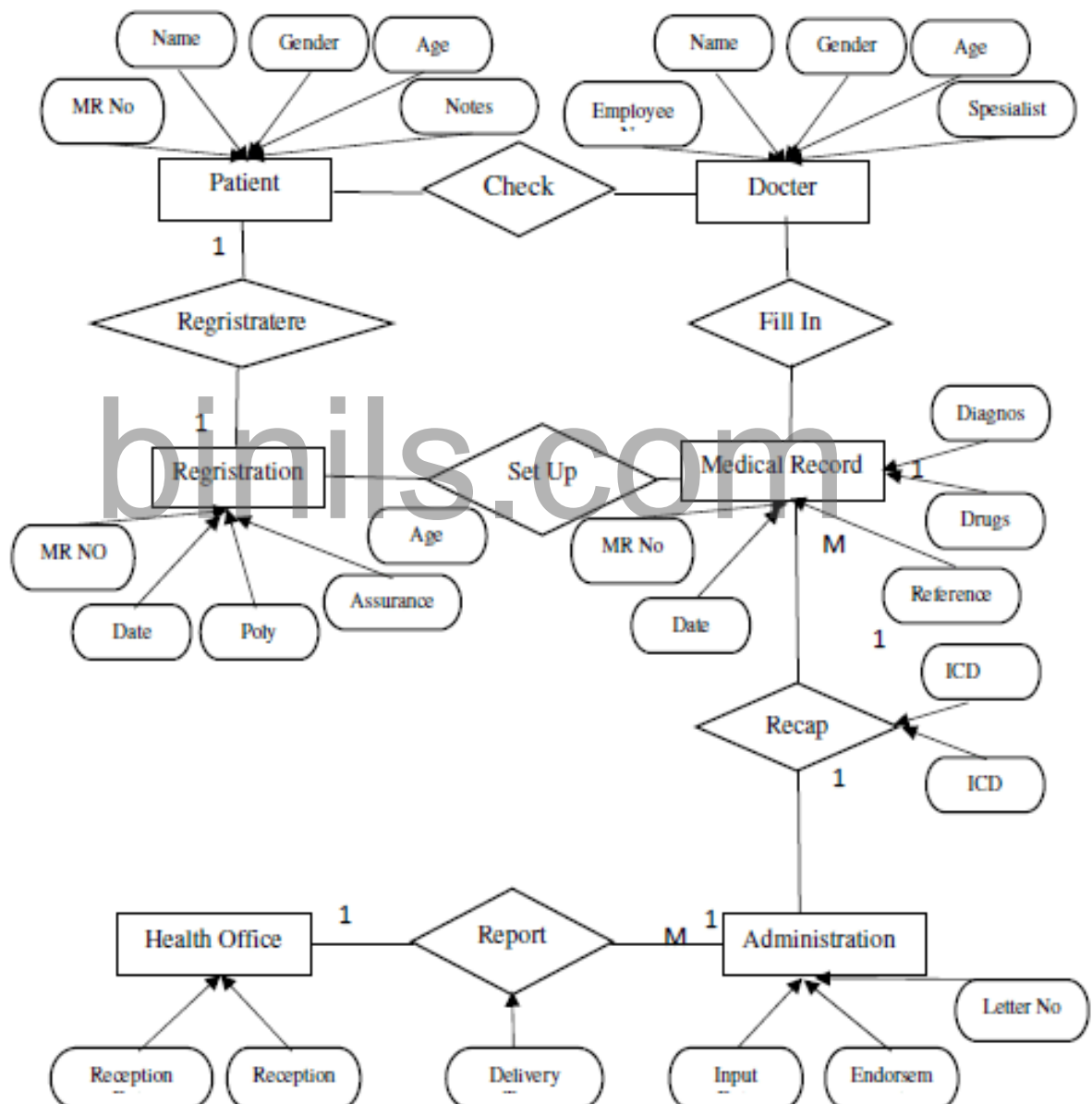
The cardinality ratio in DBMS can help us determine in which scenarios we need to place relationship attributes. It is recommended to represent the attributes of **one to one** or **one to many** relationship sets with any participating entity sets rather than a relationship set.

For example, if an entity cannot be determined as a separate entity rather it is represented by the combination of participating entity sets. In such case it is better to associate these entities to many-to-many relationship sets.



ER Diagram for Library

Hospital Management System



Unit II

RELATIONAL MODEL

Relational Model

Relational Model (RM) represents the database as a collection of relations. A relation is nothing but a table of values. Every row in the table represents a collection of related data values. These rows in the table denote a real-world entity or relationship.

The table name and column names are helpful to interpret the meaning of values in each row. The data are represented as a set of relations. In the relational model, data are stored as tables. However, the physical storage of the data is independent of the way the data are logically organized.

Some popular Relational Database management systems are:

- DB2 and Informix Dynamic Server – IBM
- Oracle and RDB – Oracle
- SQL Server and Access – Microsoft

Relational Model Concepts in DBMS

1. **Attribute:** Each column in a Table. Attributes are the properties which define a relation. e.g., Student_Rollno, NAME, etc.
2. **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.
3. **Tuple** – It is nothing but a single row of a table, which contains a single record.
4. **Relation Schema:** A relation schema represents the name of the relation with its attributes.
5. **Degree:** The total number of attributes which in the relation is called the degree of the relation.
6. **Cardinality:** Total number of rows present in the Table.
7. **Column:** The column represents the set of values for a specific attribute.
8. **Relation instance** – Relation instance is a finite set of tuples in the RDBMS system. Relation instances never have duplicate tuples.
9. **Relation key** – Every row has one, two or multiple attributes, which is called relation key.
10. **Attribute domain** – Every attribute has some pre-defined value and scope which is known as attribute domain

Table also called Relation

Primary Key

Domain
Ex: NOT NULL

© guru99.com

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Inactive

Tuple OR Row
Total # of rows is **Cardinality**

Column OR Attributes
Total # of column is **Degree**

Structure of Relational Database

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.

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

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

Key Constraints

An attribute that can uniquely identify a tuple in a relation is called the key of the table. The value of the attribute for different tuples in the relation has to be unique.

Example:

In the given table, CustomerID is a key attribute of Customer Table. It is most likely to have a single key for one customer, CustomerID =1 is only for the CustomerName =” Google”.

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Inactive

Referential Integrity Constraints

Referential Integrity constraints in DBMS are based on the concept of Foreign Keys. A foreign key is an important attribute of a relation which should be referred to in other relationships. Referential integrity constraint state happens where relation refers to a key attribute of a different or same relation. However, that key element must exist in the table.

Example:

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Inactive

Customer

InvoiceNo	CustomerID	Amount
1	1	\$100
2	1	\$200
3	2	\$150

Billing

In the above example, we have 2 relations, Customer and Billing.

Tuple for CustomerID =1 is referenced twice in the relation Billing. So we know CustomerName=Google has billing amount \$300

Best Practices for creating a Relational Model

- Data need to be represented as a collection of relations

- Each relation should be depicted clearly in the table
- Rows should contain data about instances of an entity
- Columns must contain data about attributes of the entity
- Cells of the table should hold a single value
- Each column should be given a unique name
- No two rows can be identical
- The values of an attribute should be from the same domain

Advantages of Relational Database Model

- **Simplicity:** A Relational data model in DBMS is simpler than the hierarchical and network model.
- **Structural Independence:** The relational database is only concerned with data and not with a structure. This can improve the performance of the model.
- **Easy to use:** The Relational model in DBMS is easy as tables consisting of rows and columns are quite natural and simple to understand
- **Query capability:** It makes possible for a high-level query language like [SQL](#) to avoid complex database navigation.
- **Data independence:** The Structure of Relational database can be changed without having to change any application.
- **Scalable:** Regarding a number of records, or rows, and the number of fields, a database should be enlarged to enhance its usability.

Disadvantages of Relational Model

- Few relational databases have limits on field lengths which can't be exceeded.
- Relational databases can sometimes become complex as the amount of data grows, and the relations between pieces of data become more complicated.
- Complex relational database systems may lead to isolated databases where the information cannot be shared from one system to another.

Relational database systems are expected to be equipped with a query language that can assist its users to query the database instances. There are two kinds of query languages – relational algebra and relational calculus.

Relational Algebra

Relational algebra is a procedural query language, which takes instances of relations as input and yields instances of relations as output. It uses operators to perform queries. An operator can be either **unary** or **binary**. They accept relations as their input and yield relations as their output. Relational algebra is performed recursively on a relation and intermediate results are also considered relations.

The fundamental operations of relational algebra are as follows –

- Select
- Project
- Union

- Set different
- Cartesian product
- Rename

We will discuss all these operations in the following sections.

Select Operation (σ)

It selects tuples that satisfy the given predicate from a relation.

Notation – $\sigma_p(r)$

Where σ stands for selection predicate and r stands for relation. p is propositional logic formula which may use connectors like **and**, **or**, and **not**. These terms may use relational operators like $=$, \neq , \geq , $<$, $>$, \leq .

For example –

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

Output – Selects tuples from books where subject is 'database'.

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

Output – Selects tuples from books where subject is 'database' and 'price' is 450.

$\sigma_{\text{subject} = \text{"database"} \text{ and } \text{price} = \text{"450"} \text{ or } \text{year} > \text{"2010"}}(\text{Books})$

Output – Selects tuples from books where subject is 'database' and 'price' is 450 or those books published after 2010.

Project Operation (Π)

It projects column(s) that satisfy a given predicate.

Notation – $\Pi_{A_1, A_2, A_n}(r)$

Where A_1, A_2, A_n are attribute names of relation r .

Duplicate rows are automatically eliminated, as relation is a set.

For example –

$\Pi_{\text{subject, author}}(\text{Books})$

Selects and projects columns named as subject and author from the relation Books.

Union Operation (\cup)

It performs binary union between two given relations and is defined as –

$r \cup s = \{ t \mid t \in r \text{ or } t \in s \}$

Notation – $r \cup s$

Where r and s are either database relations or relation result set (temporary relation).

For a union operation to be valid, the following conditions must hold –

- **r**, and **s** must have the same number of attributes.
- Attribute domains must be compatible.
- Duplicate tuples are automatically eliminated.

$\Pi_{\text{author}}(\text{Books}) \cup \Pi_{\text{author}}(\text{Articles})$

Output – Projects the names of the authors who have either written a book or an article or both.

Set Difference ($-$)

The result of set difference operation is tuples, which are present in one relation but are not in the second relation.

Notation – $\mathbf{r} - \mathbf{s}$

Finds all the tuples that are present in **r** but not in **s**.

$\Pi_{\text{author}}(\text{Books}) - \Pi_{\text{author}}(\text{Articles})$

Output – Provides the name of authors who have written books but not articles.

Cartesian Product (\times)

Combines information of two different relations into one.

Notation – $\mathbf{r} \times \mathbf{s}$

Where **r** and **s** are relations and their output will be defined as –

$\mathbf{r} \times \mathbf{s} = \{ \langle \mathbf{q}, \mathbf{t} \rangle \mid \mathbf{q} \in \mathbf{r} \text{ and } \mathbf{t} \in \mathbf{s} \}$

$\sigma_{\text{author} = \text{'tutorialspoint'}}(\text{Books} \times \text{Articles})$

Output – Yields a relation, which shows all the books and articles written by tutorialspoint.

Rename Operation (ρ)

The results of relational algebra are also relations but without any name. The rename operation allows us to rename the output relation. 'rename' operation is denoted with small Greek letter **rho** ρ .

Notation – $\rho_x(\mathbf{E})$

Where the result of expression **E** is saved with name of **x**.

Additional operations are –

- Set intersection
- Assignment
- Natural join

Relational Calculus

In contrast to Relational Algebra, Relational Calculus is a non-procedural query language, that is, it tells what to do but never explains how to do it.

Relational calculus exists in two forms –

Tuple Relational Calculus (TRC)

Filtering variable ranges over tuples

Notation – $\{T \mid \text{Condition}\}$

Returns all tuples T that satisfies a condition.

For example –

$\{ T.\text{name} \mid \text{Author}(T) \text{ AND } T.\text{article} = \text{'database'} \}$

Output – Returns tuples with 'name' from Author who has written article on 'database'.

TRC can be quantified. We can use Existential (\exists) and Universal Quantifiers (\forall).

For example –

$\{ R \mid \exists T \in \text{Authors}(T.\text{article} = \text{'database'} \text{ AND } R.\text{name} = T.\text{name}) \}$

Output – The above query will yield the same result as the previous one.

Domain Relational Calculus (DRC)

In DRC, the filtering variable uses the domain of attributes instead of entire tuple values (as done in TRC, mentioned above).

Notation –

$\{ a_1, a_2, a_3, \dots, a_n \mid P(a_1, a_2, a_3, \dots, a_n) \}$

Where a_1, a_2 are attributes and P stands for formulae built by inner attributes.

For example –

$\{ \langle \text{article}, \text{page}, \text{subject} \rangle \mid \in \text{TutorialsPoint} \wedge \text{subject} = \text{'database'} \}$

Output – Yields Article, Page, and Subject from the relation TutorialsPoint, where subject is database.

Just like TRC, DRC can also be written using existential and universal quantifiers. DRC also involves relational operators.

The expression power of Tuple Relation Calculus and Domain Relation Calculus is equivalent to Relational Algebra.

SQL NULL Values

In SQL there may be some records in a table that do not have values or data for every field. This could be possible because at a time of data entry information is not available. So SQL supports a special value known as NULL which is used to represent the values of attributes that may be unknown or not apply to a tuple. SQL places a NULL value in the field in the absence of a user-defined value. For example, the Apartment_number attribute of an address applies only to address that are in apartment buildings and not to other types of residences.

Importance of NULL value:

- It is important to understand that a NULL value is different from a zero value.
- A NULL value is used to represent a missing value, but that it usually has one of three different interpretations:

- The value unknown (value exists but is not known)
- Value not available (exists but is purposely withheld)
- Attribute not applicable (undefined for this tuple)
- It is often not possible to determine which of the meanings is intended. Hence, SQL does not distinguish between the different meanings of NULL.

Principles of NULL values:

- Setting a NULL value is appropriate when the actual value is unknown, or when a value would not be meaningful.
- A NULL value is not equivalent to a value of ZERO if the data type is a number and is not equivalent to spaces if the data type is character.
- A NULL value can be inserted into columns of any data type.
- A NULL value will evaluate NULL in any expression.
- Suppose if any column has a NULL value, then UNIQUE, FOREIGN key, CHECK constraints will ignore by SQL.

In general, each NULL value is considered to be different from every other NULL in the database. When a NULL is involved in a comparison operation, the result is considered to be UNKNOWN. Hence, SQL uses a three-valued logic with values **True**, **False**, and **Unknown**. It is, therefore, necessary to define the results of three-valued logical expressions when the logical connectives AND, OR, and NOT are used.

AND	TRUE	FALSE	UNKNOWN
TRUE	TRUE	FALSE	UNKNOWN
FALSE	FALSE	FALSE	FALSE
UNKNOWN	UNKNOWN	FALSE	UNKNOWN

OR	TRUE	FALSE	UNKNOWN
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	UNKNOWN
UNKNOWN	TRUE	UNKNOWN	UNKNOWN

NOT	TRUE
TRUE	FALSE
FALSE	TRUE
UNKNOWN	UNKNOWN

How to test for NULL Values?

SQL allows queries that check whether an attribute value is NULL. Rather than using = or to compare an attribute value to NULL, SQL uses **IS** and **IS NOT**. This is because SQL considers each NULL value as being distinct from every other NULL value, so equality comparison is not appropriate.

Now, consider the following Employee Table,

Fname	Lname	SSN	Salary	Super_ssn
John	Smith	123456789	30000	33344555
Franklin	Wong	333445555	40000	888665555
Joyce	English	453453453	80000	333445555
Ramesh	Narayan	666884444	38000	333445555
James	Borg	888665555	55000	NULL
Jennifer	Wallace	987654321	43000	88866555
Ahmad	Jabbar	987987987	25000	987654321
Alicia	Zeala	999887777	25000	987654321

Suppose if we find the Fname, Lname of the Employee having no Super_ssn then the query will be:

Query

SELECT Fname, Lname FROM Employee WHERE Super_ssn IS NULL;

Output:

Fname	Lname
James	Borg

Now if we find the Count of the number of Employees having Super_ssn.

Query:

SELECT COUNT(*) AS Count FROM Employee WHERE Super_ssn IS NOT NULL;

Output:

Count
7

Modification of Relational Database

Four basic update operations performed on relational database model are

Insert, update, delete and select.


- Insert is used to insert data into the relation
- Delete is used to delete tuples from the table.
- Modify allows you to change the values of some attributes in existing tuples.
- Select allows you to choose a specific range of data.

Whenever one of these operations are applied, integrity constraints specified on the relational database schema must never be violated.

Insert Operation

The insert operation gives values of the attribute for a new tuple which should be inserted into a relation.

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Inactive



CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Inactive
4	Alibaba	Active

Update Operation

You can see that in the below-given relation table CustomerName= 'Apple' is updated from Inactive to Active.

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Inactive
4	Alibaba	Active




CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Active
4	Alibaba	Active

Delete Operation

To specify deletion, a condition on the attributes of the relation selects the tuple to be deleted.

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Active
4	Alibaba	Active



CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
4	Alibaba	Active

In the above-given example, CustomerName= "Apple" is deleted from the table.

The Delete operation could violate referential integrity if the tuple which is deleted is referenced by foreign keys from other tuples in the same [database](#).

Select Operation

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
4	Alibaba	Active



CustomerID	CustomerName	Status
2	Amazon	Active

In the above-given example, CustomerName="Amazon" is selected

Structured Query Language (SQL)

Structured Query Language is a standard Database language which is used to create, maintain and retrieve the relational database. Following are some interesting facts about SQL.

- SQL is case insensitive. But it is a recommended practice to use keywords (like SELECT, UPDATE, CREATE, etc) in capital letters and use user defined things (like table name, column name, etc) in small letters.
- We can write comments in SQL using “--” (double hyphen) at the beginning of any line.
- SQL is the programming language for relational databases (explained below) like MySQL, Oracle, Sybase, SQL Server, Postgre, etc. Other non-relational databases (also called NoSQL) databases like MongoDB, DynamoDB, etc do not use SQL
- Although there is an ISO standard for SQL, most of the implementations slightly vary in syntax. So we may encounter queries that work in SQL Server but do not work in MySQL.

What is Relational Database?

Relational database means the data is stored as well as retrieved in the form of relations (tables). Table 1 shows the relational database with only one relation called **STUDENT** which stores **ROLL_NO**, **NAME**, **ADDRESS**, **PHONE** and **AGE** of students.

STUDENT

ROLL_NO	NAME	ADDRESS	PHONE	AGE
1	RAM	DELHI	9455123451	18
2	RAMESH	GURGAON	9652431543	18
3	SUJIT	ROHTAK	9156253131	20
4	SURESH	DELHI	9156768971	18

TABLE 1

These are some important terminologies that are used in terms of relation.

Attribute: Attributes are the properties that define a relation. e.g.; **ROLL_NO**, **NAME** etc.

Tuple: Each row in the relation is known as tuple. The above relation contains 4 tuples, one of which is shown as:

1	RAM	DELHI	9455123451	18
---	-----	-------	------------	----

Degree: The number of attributes in the relation is known as degree of the relation. The **STUDENT** relation defined above has degree 5.

Cardinality: The number of tuples in a relation is known as cardinality. The **STUDENT** relation defined above has cardinality 4.

Column: Column represents the set of values for a particular attribute. The column **ROLL_NO** is extracted from relation STUDENT.

ROLL_NO

1

2

3

4

The queries to deal with relational database can be categories as:

Data Definition Language: It is used to define the structure of the database. e.g; CREATE TABLE, ADD COLUMN, DROP COLUMN and so on.

Data Manipulation Language: It is used to manipulate data in the relations. e.g.; INSERT, DELETE, UPDATE and so on.

Data Query Language: It is used to extract the data from the relations. e.g.; SELECT
So first we will consider the Data Query Language. A generic query to retrieve from a relational database is:

1. **SELECT** [**DISTINCT**] Attribute_List **FROM** R1,R2....RM
2. [**WHERE** condition]
3. [**GROUP BY** (Attributes)[**HAVING** condition]]
4. [**ORDER BY**(Attributes)[**DESC**]];

Part of the query represented by statement 1 is compulsory if you want to retrieve from a relational database. The statements written inside [] are optional. We will look at the possible query combination on relation shown in Table 1.

Case 1: If we want to retrieve attributes **ROLL_NO** and **NAME** of all students, the query will be:

SELECT ROLL_NO, NAME **FROM** STUDENT;

ROLL_NO	NAME
1	RAM
2	RAMESH
3	SUJIT

4

SURESH

Case 2: If we want to retrieve **ROLL_NO** and **NAME** of the students whose **ROLL_NO** is greater than 2, the query will be:

SELECT ROLL_NO, NAME FROM STUDENT WHERE ROLL_NO>2;

ROLL_NO	NAME
3	SUJIT
4	SURESH

CASE 3: If we want to retrieve all attributes of students, we can write * in place of writing all attributes as:

SELECT * FROM STUDENT WHERE ROLL_NO>2;

ROLL_NO	NAME	ADDRESS	PHONE	AGE
3	SUJIT	ROHTAK	9156253131	20
4	SURESH	DELHI	9156768971	18

CASE 4: If we want to represent the relation in ascending order by **AGE**, we can use **ORDER BY** clause as:

SELECT * FROM STUDENT ORDER BY AGE;

ROLL_NO	NAME	ADDRESS	PHONE	AGE
1	RAM	DELHI	9455123451	18
2	RAMESH	GURGAON	9652431543	18
4	SURESH	DELHI	9156768971	18
3	SUJIT	ROHTAK	9156253131	20

Note: **ORDER BY AGE** is equivalent to **ORDER BY AGE ASC**. If we want to retrieve the results in descending order of **AGE**, we can use **ORDER BY AGE DESC**.

CASE 5: If we want to retrieve distinct values of an attribute or group of attribute, DISTINCT is used as in:

SELECT DISTINCT ADDRESS FROM STUDENT;

ADDRESS

DELHI

GURGAON

ROHTAK

If DISTINCT is not used, DELHI will be repeated twice in result set. Before understanding GROUP BY and HAVING, we need to understand aggregations functions in SQL.

AGGRATION FUNCTIONS: Aggregation functions are used to perform mathematical operations on data values of a relation. Some of the common aggregation functions used in SQL are:

- **COUNT:** Count function is used to count the number of rows in a relation. e.g;
SELECT COUNT (PHONE) FROM STUDENT;

COUNT(PHONE)

4

- **SUM:** SUM function is used to add the values of an attribute in a relation. e.g;
SELECT SUM (AGE) FROM STUDENT;

SUM(AGE)

74

In the same way, MIN, MAX and AVG can be used. As we have seen above, all aggregation functions return only 1 row.

AVERAGE: It gives the average values of the tupples. It is also defined as sum divided by count values.

Syntax:AVG(attributename)

OR

Syntax:SUM(attributename)/COUNT(attributename)

The above mentioned syntax also retrieves the average value of tupples.

MAXIMUM:It extracts the maximum value among the set of tupples.

Syntax:MAX(attributename)

MINIMUM:It extracts the minimum value amongst the set of all the tupples.

Syntax:MIN(attributename)

GROUP BY: Group by is used to group the tuples of a relation based on an attribute or group of attribute. It is always combined with aggregation function which is computed on group. e.g.;

**SELECT ADDRESS, SUM(AGE) FROM STUDENT
GROUP BY (ADDRESS);**

In this query, SUM(AGE) will be computed but not for entire table but for each address. i.e.; sum of AGE for address DELHI(18+18=36) and similarly for other address as well. The output is:

ADDRESS	SUM(AGE)
DELHI	36
GURGAON	18
ROHTAK	20

If we try to execute the query given below, it will result in error because although we have computed SUM(AGE) for each address, there are more than 1 ROLL_NO for each address we have grouped. So it can't be displayed in result set. We need to use aggregate functions on columns after SELECT statement to make sense of the resulting set whenever we are using GROUP BY.

**SELECT ROLL_NO, ADDRESS, SUM(AGE) FROM STUDENT
GROUP BY (ADDRESS);**

Advanced SQL

Accessing SQL From a Programming Language

- API (application-program interface) for a program to interact with a database server
- Application makes calls to
 - Connect with the database server
 - Send SQL commands to the database server
 - Fetch tuples of result one-by-one into program variables
- Various tools:
 - ODBC (Open Database Connectivity) works with C, C++, C#, and Visual Basic. Other APIs such as ADO.NET sit on top of ODBC
 - JDBC (Java Database Connectivity) works with Java
 - Embedded SQL

Integrity Constraints

- The Set of rules which is used to maintain the quality of information are known as integrity constraints.
- Integrity constraints make sure about data intersection, update and so on.
- Integrity constraints can be understood as a guard against unintentional damage to the database.

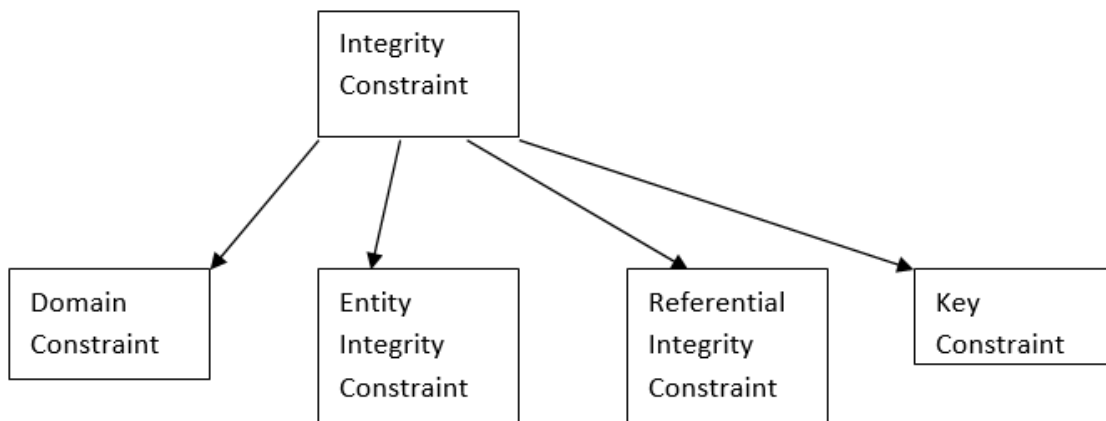
For any stored data if we want to preserve the consistency and correctness, a relational DBMS typically imposes one or more data integrity constraints. These constraints restrict the data values which can be inserted into the database or created by a database update.

Data Integrity Constraints

There are different types of data integrity constraints that are commonly found in relational databases, including the following –

- **Required data** – Some columns in a database contain a valid data value in each row; they are not allowed to contain NULL values. In the sample database, every order has an associated customer who placed the order. The DBMS can be asked to prevent NULL values in this column.
- **Validity checking** – Every column in a database has a domain, a set of data values which are legal for that column. The DBMS allowed preventing other data values in these columns.
- **Entity integrity** – The primary key of a table contains a unique value in each row that is different from the values in all other rows. Duplicate values are illegal because they are not allowing the database to differentiate one entity from another. The DBMS can be asked to enforce this unique values constraint.
- **Referential integrity** – A foreign key in a relational database links each row in the child table containing the foreign key to the row of the parent table containing the matching primary key value. The DBMS can be asked to enforce this foreign key/primary key constraint.
- **Other data relationships** – The real-world situation which is modeled by a database often has additional constraints which govern the legal data values that may appear in the database. The DBMS is allowed to check modifications to the tables to make sure that their values are constrained in this way.
- **Business rules** – Updates to a database that are constrained by business rules governing the real-world transactions which are represented by the updates.
- **Consistency** – Many real-world transactions that cause multiple updates to a database. The DBMS is allowed to enforce this type of consistency rule or to support applications that implement such rules.

Different types of Integrity Constraints



Domain Constraint

- The Definition of an applicable set of values is known as domain constraint.
- Strings, character, time, integer, currency, date etc. Are examples of the data type of domain constraints.

Example

ID	NAME	SEMESTER	AGE
100	Jai	1st	27
101	BKadam	4th	34
102	Rajeev	3rd	31
103	Asmita	6th	29
104	Mahesh	2nd	Twenty two

‘Twenty two’ is not allowed for 104 id because the attribute AGE is an integer

Entity Integer Constraint

- Entity Integrity Constraints states that the primary value key cannot be null because the primary value key is used to find out individual rows in relation and if the value of the primary key is null then it is not easy to identify those rows.
- There can be a null value in the table apart from the primary key field.

Example

Emp_ID	Emp_Name	Salary
11	Manish	30000

12	Vikram	20000
13	Sudhir	10000
	Rajeev	40000

Null is not allowed in Emp_ID as it is a Primary key and cannot have a NULL value.

Referential Integrity Constraint

1. Referential Integrity Constraint is specific between two tables.
2. A foreign key in the 1st table refers to the primary key of the 2nd table, in this case each value of the foreign key in the 1st table has to be null or present in the 2nd table.

Key Constraints

- The Entity within its entity set is identified uniquely by the key which is the entity set.
- There can be a number of keys in an entity set but only one will be the primary key out of all keys. In a relational table a primary key can have a unique as well as a null value.

Example

ID	NAME	SEMESTER	AGE
100	Naren	4	27
101	Lalit	6	28
102	Shivanshu	3	22
103	Navdeep	5	29
102	Karthik	7	25

All row ID must be unique hence 102 is not allowed.

Database authorization

Authorization is the process where the database manager gets information about the authenticated user. Part of that information is determining which database operations the user can perform and which data objects a user can access.

A privilege is a type of permission for an authorization name, or a permission to perform an action or a task. The privilege allows a user to create or access database resources. Privileges are stored in the database catalogs. Authorized users can pass on privileges on their own objects to other users by using the GRANT statement. Privileges can be granted to individual users, to groups, or to PUBLIC. PUBLIC is a special group that consists of all users, including future users. Users that are members of a group will indirectly take advantage of the privileges granted to the group, where groups are supported.

A role is a database object that groups one or more privileges. Roles can be assigned to users or groups or other roles by using the GRANT statement. Users that are members of roles have the privileges that are defined for the role with which to access data.

The forms of authorization, such as administrative authority, privileges, and Row and column access (RCAC) access, are discussed in Authorization of Big SQL objects. In addition, ownership of objects brings with it a degree of authorization on the objects created.

- Administrative authority includes system-level authorization and database-level authorization:

System-level authorization

SYSADM (system administrator) authority

The SYSADM (system administrator) authority provides control over all the resources created and maintained by the database manager. The system administrator possesses all the authorities of SYSCTRL, SYSMANT, and SYSMON authority. The user who has SYSADM authority is responsible both for controlling the database manager, and for ensuring the safety and integrity of the data.

SYSCTRL authority

The SYSCTRL authority provides control over operations that affect system resources. For example, a user with SYSCTRL authority can create, update, start, stop, or drop a database. This user can also start or stop an instance, but cannot access table data. Users with SYSCTRL authority also have SYSMON authority.

SYSMANT authority

The SYSMANT authority provides the authority required to perform maintenance operations on all databases that are associated with an instance. A user with SYSMANT authority can update the database configuration, backup a database or table space, restore an existing database, and monitor a database. Like SYSCTRL, SYSMANT does not provide access to table data. Users with SYSMANT authority also have SYSMON authority.

SYSMON (system monitor) authority

The SYSMON (system monitor) authority provides the authority required to use the database system monitor.

Database-level authorization

DBADM (database administrator)

The DBADM authority level provides administrative authority over a single database. This database administrator possesses the privileges required to create objects and issue database commands. The DBADM authority can be granted only by a user with SECADM authority. The DBADM authority cannot be granted to PUBLIC.

SECADM (security administrator)

The SECADM authority level provides administrative authority for security over a single database. The security administrator authority possesses the ability to manage database security objects (database roles, audit policies, trusted contexts, security label components, and security labels) and grant and revoke all database privileges and authorities. A user with SECADM authority can transfer the ownership of objects that they do not own. They can also use the AUDIT statement to associate an audit policy with a particular database or database object at the server. The SECADM authority has no inherent privilege to access data stored in tables. It can only be granted by a user with SECADM authority. The SECADM authority cannot be granted to PUBLIC.

SQLADM (SQL administrator)

The SQLADM authority level provides administrative authority to monitor and tune SQL statements within a single database. It can be granted by a user with ACCESSCTRL or SECADM authority.

WLMADM (workload management administrator)

The WLMADM authority provides administrative authority to manage workload management objects, such as service classes, work action sets, work class sets, and workloads. It can be granted by a user with ACCESSCTRL or SECADM authority. EXPLAIN (explain authority) The EXPLAIN authority level provides administrative authority to explain query plans without gaining access to data. It can only be granted by a user with ACCESSCTRL or SECADM authority.

EXPLAIN (explain authority)

The EXPLAIN authority level provides administrative authority to explain query plans without gaining access to data. It can only be granted by a user with ACCESSCTRL or SECADM authority.

ACCESSCTRL (access control authority)

ACCESSCTRL authority can only be granted by a user with SECADM authority. The ACCESSCTRL authority cannot be granted to PUBLIC. The ACCESSCTRL authority level provides administrative authority to issue the following GRANT (and REVOKE) statements:

- GRANT (Database Authorities)
- GRANT (Global Variable Privileges)
- GRANT (Index Privileges)
- GRANT (Module Privileges)
- GRANT (Package Privileges)
- GRANT (Routine Privileges)
- GRANT (Schema Privileges)
- GRANT (Sequence Privileges)
- GRANT (Server Privileges)
- GRANT (Table, View, or Nickname Privileges)
- GRANT (Table Space Privileges)
- GRANT (Workload Privileges)
- GRANT (XSR Object Privileges)

For more information about granting and revoking privileges, see Granting and revoking access.

DATA ACCESS (data access authority)

DATAACCESS authority can be granted only by a user who holds SECADM authority. It cannot be granted to PUBLIC. The DATAACCESS authority level provides the following privileges and authorities:

- LOAD authority
- SELECT, INSERT, UPDATE, DELETE privilege on tables, views, nicknames, and materialized query tables
- EXECUTE privilege on packages
- EXECUTE privilege on modules
- EXECUTE privilege on routines, except on the audit routines.
- USAGE privilege on all sequences

Database authorities (non-administrative)

To perform activities such as creating a table or a routine, or for loading data into a table, specific database authorities are required. For example, the LOAD database authority is required for use of the load utility to load data into tables (a user must also have INSERT privilege on the table).

- Privileges

CONTROL privilege

If you possess the CONTROL privilege on an object, you can access that database object, and grant and revoke privileges to or from other users on that object. The CONTROL privilege only applies to tables, views, nicknames, indexes, and packages..

If a different user requires the CONTROL privilege to that object, a user with SECADM or ACCESSCTRL authority can grant the CONTROL privilege to that object. The CONTROL privilege cannot be revoked from the object owner, however, the object owner can be changed by using the TRANSFER OWNERSHIP statement.

Individual privileges

Individual privileges can be granted to allow a user to carry out specific tasks on specific objects. Users with the administrative authorities ACCESSCTRL or SECADM, or with the CONTROL privilege, can grant and revoke privileges to and from users.

Revoking privileges

The REVOKE statement is used to revoke previously granted privileges. The revoking of a privilege from an authorization name revokes the privilege granted by all authorization names.

Authorization ID privileges: SETSESSION USER

Authorization ID privileges involve actions on authorization IDs. There is currently only one such privilege: the SETSESSIONUSER privilege.

Schema privileges

Schema privileges are in the object privilege category.

Table and view privileges

Table and view privileges involve actions on tables or views in a database.

Package privileges

A package is a database object that contains the information needed by the database manager to access data in the most efficient way for a particular application program. Package privileges enable a user to create and manipulate packages.

Sequence privileges

The creator of a sequence automatically receives the USAGE and ALTER privileges on the sequence. The USAGE privilege is needed to use NEXT VALUE and PREVIOUS VALUE expressions for the sequence.

Routine privileges

Execute privileges involve actions on all types of routines such as functions, procedures, and methods within a database. Once having EXECUTE privilege, a user can then invoke that routine, create a function that is sourced from that routine (applies to functions only), and reference the routine in any DDL statement such as CREATE VIEW or CREATE TRIGGER.

Usage privilege on workloads

To enable use of a workload, a user who holds ACCESSCTRL, SECADM, or WLMADM authority can grant USAGE privilege on that workload to a user, a group, or a role using the GRANT USAGE ON WORKLOAD statement.

Introduction to embedded SQL

Embedded SQL applications connect to databases and execute embedded SQL statements. The embedded SQL statements are contained in a package that must be bound to the target database server.

You can develop embedded SQL applications for the Db2® database in the following host programming languages: C, C++, and COBOL.

Building embedded SQL applications involves two prerequisite steps before application compilation and linking.

- Preparing the source files containing embedded SQL statements using the Db2 precompiler.

The PREP (PRECOMPILE) command is used to invoke the Db2 precompiler, which reads your source code, parses and converts the embedded SQL statements to Db2 run-time services API calls, and finally writes the output to a new modified source file. The precompiler produces access plans for the SQL statements, which are stored together as a package within the database.

- Binding the statements in the application to the target database.

Binding is done by default during precompilation (the PREP command). If binding is to be deferred (for example, running the BIND command later), then the BINDFILE option needs to be specified at PREP time in order for a bind file to be generated.

Once you have precompiled and bound your embedded SQL application, it is ready to be compiled and linked using the host language-specific development tools.

To aid in the development of embedded SQL applications, you can refer to the embedded SQL template in C. Examples of working embedded SQL sample applications can also be found in the %DB2PATH%\SQLLIB\samples directory.

Note: %DB2PATH% refers to the Db2 installation directory

Static and dynamic SQL

SQL statements can be executed in one of two ways: statically or dynamically.

Statically executed SQL statements

For statically executed SQL statements, the syntax is fully known at precompile time. The structure of an SQL statement must be completely specified for a statement to be considered static. For example, the names for the columns and tables referenced in a statement must be fully known at precompile time. The only information that can be specified at run time are values for any host variables referenced by the statement. However, host variable information, such as data types, must still be precompiled. You precompile, bind, and compile statically executed SQL statements before you run your application. Static SQL is best used on databases whose statistics do not change a great deal.

Dynamically executed SQL statements

Dynamically executed SQL statements are built and executed by an application at run-time. An interactive application that prompts the end user for key parts of an SQL statement, such as the names of the tables and columns to be searched, is a good example of a situation suited for dynamic SQL.

- **Embedding SQL statements in a host language**
Structured Query Language (SQL) is a standardized language that you can use to manipulate database objects and the data that they contain. Despite differences between host languages, embedded SQL applications are made up of three main elements that are required to setup and issue an SQL statement.

- **Supported development software for embedded SQL applications**
Before you begin writing embedded SQL applications, you must determine if your development software is supported. The operating system that you are developing for determines which compilers, interpreters, and development software you must use.
- **Setting up the embedded SQL development environment**
Before you can start building embedded SQL applications, install the supported compiler for the host language you will be using to develop your applications and set up the embedded SQL environment.
- **Designing embedded SQL applications**
When designing embedded SQL applications you must use static or dynamic executed SQL statements.
- **Programming embedded SQL applications**
Programming embedded SQL applications involves the same steps required to assemble an application in your host programming language.
- **Building embedded SQL applications**
After you have created the source code for your embedded SQL application, you must follow additional steps to build the application. You should consider building 64-bit executable files when developing new embedded SQL database applications. Along with compiling and linking your program, you must precompile and bind it.
- **Deploying and running embedded SQL applications**
Embedded SQL applications are portable and can be placed in remote database components. You can compile the application in one location and run the package on a different component.
- **Compatibility features for migration**
The Db2 database manager provides features that facilitate the migration of embedded SQL C applications from other database systems.

Dynamic SQL

Dynamic SQL enables you to write programs that reference SQL statements whose full text is not known until runtime. Before discussing dynamic SQL in detail, a clear definition of static SQL may provide a good starting point for understanding dynamic SQL. Static SQL statements do not change from execution to execution. The full text of static SQL statements are known at compilation, which provides the following benefits:

- Successful compilation verifies that the SQL statements reference valid database objects.
- Successful compilation verifies that the necessary privileges are in place to access the database objects.
- Performance of static SQL is generally better than dynamic SQL.

Because of these advantages, you should use dynamic SQL only if you cannot use static SQL to accomplish your goals, or if using static SQL is cumbersome compared to dynamic SQL. However, static SQL has limitations that can be overcome with dynamic SQL. You may not always know the full text of the SQL statements that must be executed in a PL/SQL procedure. Your program may accept user input that defines the SQL statements to execute, or your program may need to complete some processing work to determine the correct course of action. In such cases, you should use dynamic SQL.

For example, consider a reporting application that performs standard queries on tables in a data warehouse environment where the exact table name is unknown until runtime. To accommodate the large amount of data in the data warehouse efficiently, you create a new table every quarter to store the invoice information for the quarter. These tables all have exactly the same definition and are named according to the starting month and year of the quarter, for example INV_01_1997, INV_04_1997, INV_07_1997, INV_10_1997, INV_01_1998, etc. In such a case, you can use dynamic SQL in your reporting application to specify the table name at runtime.

With static SQL, all of the data definition information, such as table definitions, referenced by the SQL statements in your program must be known at compilation. If the data definition changes, you must change and recompile the program. Dynamic SQL programs can handle changes in data definition information, because the SQL statements can change "on the fly" at runtime. Therefore, dynamic SQL is much more flexible than static SQL. Dynamic SQL enables you to write application code that is reusable because the code defines a process that is independent of the specific SQL statements used.

In addition, dynamic SQL lets you execute SQL statements that are not supported in static SQL programs, such as data definition language (DDL) statements. Support for these statements allows you to accomplish more with your PL/SQL programs.

Tuple Relational Calculus

Tuple Relational Calculus is a **non-procedural query language** unlike relational algebra. Tuple Calculus provides only the description of the query but it does not provide the methods to solve it. Thus, it explains what to do but not how to do. In Tuple Calculus, a query is expressed as $\{t \mid P(t)\}$

where t = resulting tuples,
 $P(t)$ = known as Predicate and these are the conditions that are used to fetch t

Thus, it generates set of all tuples t , such that Predicate $P(t)$ is true for t .

$P(t)$ may have various conditions logically combined with OR (\vee), AND (\wedge), NOT (\neg). It also uses quantifiers:
 $\exists t \in r (Q(t))$ = "there exists" a tuple in t in relation r such that predicate $Q(t)$ is true.
 $\forall t \in r (Q(t))$ = $Q(t)$ is true "for all" tuples in relation r .

Example:

Table-1: Customer

Customer name	Street	City
Saurabh	A7	Patiala
Mehak	B6	Jalandhar
Sumiti	D9	Ludhiana

Customer name	Street	City
Ria	A5	Patiala

Table-2: Branch

Branch name	Branch city
ABC	Patiala
DEF	Ludhiana
GHI	Jalandhar

Table-3: Account

Account number	Branch name	Balance
1111	ABC	50000
1112	DEF	10000
1113	GHI	9000
1114	ABC	7000

Table-4: Loan

Loan number	Branch name	Amount
L33	ABC	10000
L35	DEF	15000
L49	GHI	9000
L98	DEF	65000

Table-5: Borrower

Customer name	Loan number
---------------	-------------

Customer name	Loan number
Saurabh	L33
Mehak	L49
Ria	L98

Table-6: Depositor

Customer name	Account number
Saurabh	1111
Mehak	1113
Sumiti	1114

Queries-1: Find the loan number, branch, amount of loans of greater than or equal to 10000 amount.

$\{t \mid t \in \text{loan} \wedge t[\text{amount}] \geq 10000\}$

Resulting relation:

Loan number	Branch name	Amount
L33	ABC	10000
L35	DEF	15000
L98	DEF	65000

In the above query, $t[\text{amount}]$ is known as tuple variable.

Queries-2: Find the loan number for each loan of an amount greater or equal to 10000.

$\{t \mid \exists s \in \text{loan}(t[\text{loan number}] = s[\text{loan number}]$

$\wedge s[\text{amount}] \geq 10000)\}$

Resulting relation:

Loan number
L33
L35

Loan number

L98

Queries-3: Find the names of all customers who have a loan and an account at the bank.

$\{t \mid \exists s \in \text{borrower}(t[\text{customer-name}] = s[\text{customer-name}])$

$\wedge \exists u \in \text{depositor}(t[\text{customer-name}] = u[\text{customer-name}])\}$

Resulting relation:

Customer name

Saurabh

Mehak

Queries-4: Find the names of all customers having a loan at the “ABC” branch.

$\{t \mid \exists s \in \text{borrower}(t[\text{customer-name}] = s[\text{customer-name}]$

$\wedge \exists u \in \text{loan}(u[\text{branch-name}] = \text{“ABC”} \wedge u[\text{loan-number}] = s[\text{loan-number}])\}$

Resulting relation:

Customer name

Saurabh

Domain Relational Calculus

Domain Relational Calculus is a non-procedural query language equivalent in power to Tuple Relational Calculus. Domain Relational Calculus provides only the description of the query but it does not provide the methods to solve it. In Domain Relational Calculus, a query is expressed as,

$\{ \langle x_1, x_2, x_3, \dots, x_n \rangle \mid P(x_1, x_2, x_3, \dots, x_n) \}$

where, $\langle x_1, x_2, x_3, \dots, x_n \rangle$ represents resulting domains variables and $P(x_1, x_2, x_3, \dots, x_n)$ represents the condition or formula equivalent to the Predicate calculus.

Predicate Calculus Formula:

1. Set of all comparison operators
2. Set of connectives like and, or, not
3. Set of quantifiers

Example:

Table-1: Customer

Customer name	Street	City
Debomit	Kadamtala	Alipurduar

Customer name	Street	City
Sayantana	Udayapur	Balurghat
Soumya	Nutanchati	Bankura
Ritu	Juhu	Mumbai

Table-2: Loan

Loan number	Branch name	Amount
L01	Main	200
L03	Main	150
L10	Sub	90
L08	Main	60

Table-3: Borrower

Customer name	Loan number
Ritu	L01
Debomita	L08
Soumya	L03

Query-1: Find the loan number, branch, amount of loans of greater than or equal to 100 amount.

$\{ \langle l, b, a \rangle \mid \langle l, b, a \rangle \in \text{loan} \wedge (a \geq 100) \}$

Resulting relation:

Loan number	Branch name	Amount
L01	Main	200
L03	Main	150

Query-2: Find the loan number for each loan of an amount greater or equal to 150.

$\{ \langle l \rangle \mid \exists b, a (\langle l, b, a \rangle \in \text{loan} \wedge (a \geq 150)) \}$

Resulting relation:

Loan number

L01

L03

Query-3: Find the names of all customers having a loan at the “Main” branch and find the loan amount .

$\{ \langle c, a \rangle \mid \exists l (\langle c, l \rangle \in \text{borrower} \wedge \exists b (\langle l, b, a \rangle \in \text{loan} \wedge (b = \text{“Main”}))) \}$

Resulting relation:

Customer Name	Amount
---------------	--------

Ritu	200
------	-----

Debomit	60
---------	----

Soumya	150
--------	-----

Query By Example (QBE)

Normal queries we fire on the database they should be correct and in a well-defined structure which means they should follow a proper syntax if the syntax or query is wrong definitely we will get an error and due to that our application or calculation definitely going to stop. So to overcome this problem QBE was introduced. QBE stands for **Query By Example** and it was developed in 1970 by Moshe Zloof at IBM.

It is a graphical query language where we get a user interface and then we fill some required fields to get our proper result.

In [SQL](#) we will get an error if the query is not correct but in the case of QBE if the query is wrong either we get a wrong answer or the query will not be going to execute but we will never get any error.

Note-:

In QBE we don't write complete queries like SQL or other database languages it comes with some blank so we need to just fill that blanks and we will get our required result.

Example

Consider the example where a table ‘SAC’ is present in the database with Name, Phone_Number, and Branch fields. And we want to get the name of the SAC-Representative name who belongs to the MCA Branch. If we write this query in SQL we have to write it like

SELECT NAME

FROM SAC

WHERE BRANCH = 'MCA'

And definitely, we will get our correct result. But in the case of QBE, it may be done as like there is a field present and we just need to fill it with “MCA” and then click on the SEARCH button we will get our required result.

Points about QBE:

- Supported by most of the database programs.
- It is a Graphical Query Language.
- Created in parallel to SQL development.

SQL Trigger

Trigger: A trigger is a stored procedure in database which automatically invokes whenever a special event in the database occurs. For example, a trigger can be invoked when a row is inserted into a specified table or when certain table columns are being updated.

Syntax:

create trigger [trigger_name]

[before | after]

{insert | update | delete}

on [table_name]

[for each row]

[trigger_body]

Explanation of syntax:

1. create trigger [trigger_name]: Creates or replaces an existing trigger with the trigger_name.
2. [before | after]: This specifies when the trigger will be executed.
3. {insert | update | delete}: This specifies the DML operation.
4. on [table_name]: This specifies the name of the table associated with the trigger.
5. [for each row]: This specifies a row-level trigger, i.e., the trigger will be executed for each row being affected.
6. [trigger_body]: This provides the operation to be performed as trigger is fired

BEFORE and AFTER of Trigger:

BEFORE triggers run the trigger action before the triggering statement is run.
AFTER triggers run the trigger action after the triggering statement is run.

Example:

Given Student Report Database, in which student marks assessment is recorded. In such schema, create a trigger so that the total and average of specified marks is automatically inserted whenever a record is insert.

Here, as trigger will invoke before record is inserted so, BEFORE Tag can be used.

Suppose the database Schema –

mysql> desc Student;

Field	Type	Null	Key	Default	Extra
tid	int(4)	NO	PRI	NULL	auto_increment
name	varchar(30)	YES		NULL	
subj1	int(2)	YES		NULL	
subj2	int(2)	YES		NULL	
subj3	int(2)	YES		NULL	
total	int(3)	YES		NULL	
per	int(3)	YES		NULL	

7 rows in set (0.00 sec)

SQL Trigger to problem statement.

create trigger stud_marks

before INSERT

on

Student

for each row

set Student.total = Student.subj1 + Student.subj2 + Student.subj3, Student.per = Student.total * 60 / 100;

Above SQL statement will create a trigger in the student database in which whenever subjects marks are entered, before inserting this data into the database, trigger will compute those two values and insert with the entered values. i.e.,

mysql> insert into Student values(0, "ABCDE", 20, 20, 20, 0, 0);

Query OK, 1 row affected (0.09 sec)

mysql> select * from Student;

tid	name	subj1	subj2	subj3	total	per
100	ABCDE	20	20	20	60	36

1 row in set (0.00 sec)

Unit 3
Database Design

Functional dependencies in DBMS

A functional dependency is a constraint that specifies the relationship between two sets of attributes where one set can accurately determine the value of other sets. It is denoted as $X \rightarrow Y$, where X is a set of attributes that is capable of determining the value of Y . The attribute set on the left side of the arrow, X is called **Determinant**, while on the right side, Y is called the **Dependent**.

Functional dependencies are used to mathematically express relations among database entities and are very important to understand advanced concepts in Relational Database System.

Example:

Roll_No	Name	Dept_Name	Dept_Building
42	abc	CO	A4
43	pqr	IT	A3
44	xyz	CO	A4
45	xyz	IT	A3
46	mno	EC	B2
47	jkl	ME	B2

From the above table we can conclude some valid functional dependencies:

- $\text{roll_no} \rightarrow \{ \text{name}, \text{dept_name}, \text{dept_building} \}$, \rightarrow Here, roll_no can determine values of fields name, dept_name and dept_building, hence a valid Functional dependency
- $\text{roll_no} \rightarrow \text{dept_name}$, Since, roll_no can determine whole set of {name, dept_name, dept_building}, it can determine its subset dept_name also.
- $\text{dept_name} \rightarrow \text{dept_building}$, Dept_name can identify the dept_building accurately, since departments with different dept_name will also have a different dept_building
- More valid functional dependencies: $\text{roll_no} \rightarrow \text{name}$, $\{ \text{roll_no}, \text{name} \} \twoheadrightarrow \{ \text{dept_name}, \text{dept_building} \}$, etc.

Here are some invalid functional dependencies:

- $\text{name} \rightarrow \text{dept_name}$ Students with the same name can have different dept_name, hence this is not a valid functional dependency.
- $\text{dept_building} \rightarrow \text{dept_name}$ There can be multiple departments in the same building, For example, in the above table departments ME and EC are in the same building B2, hence $\text{dept_building} \rightarrow \text{dept_name}$ is an invalid functional dependency.
- More invalid functional dependencies: $\text{name} \rightarrow \text{roll_no}$, $\{\text{name}, \text{dept_name}\} \rightarrow \text{roll_no}$, $\text{dept_building} \rightarrow \text{roll_no}$, etc.

Armstrong's axioms/properties of functional dependencies:

1. **Reflexivity:** If Y is a subset of X, then $X \rightarrow Y$ holds by reflexivity rule
For example, $\{\text{roll_no}, \text{name}\} \rightarrow \text{name}$ is valid.
2. **Augmentation:** If $X \rightarrow Y$ is a valid dependency, then $XZ \rightarrow YZ$ is also valid by the augmentation rule.
For example, If $\{\text{roll_no}, \text{name}\} \rightarrow \text{dept_building}$ is valid, hence $\{\text{roll_no}, \text{name}, \text{dept_name}\} \rightarrow \{\text{dept_building}, \text{dept_name}\}$ is also valid. \rightarrow
3. **Transitivity:** If $X \rightarrow Y$ and $Y \rightarrow Z$ are both valid dependencies, then $X \rightarrow Z$ is also valid by the Transitivity rule.
For example, $\text{roll_no} \rightarrow \text{dept_name}$ & $\text{dept_name} \rightarrow \text{dept_building}$, then $\text{roll_no} \rightarrow \text{dept_building}$ is also valid.

Types of Functional dependencies in DBMS:

1. Trivial functional dependency
2. Non-Trivial functional dependency
3. Multivalued functional dependency
4. Transitive functional dependency

1. Trivial Functional Dependency

In **Trivial Functional Dependency**, a dependent is always a subset of the determinant.

i.e. If $X \rightarrow Y$ and **Y is the subset of X**, then it is called trivial functional dependency

For example,

roll_no	name	age
42	abc	17
43	pqr	18
44	xyz	18

Here, $\{\text{roll_no}, \text{name}\} \rightarrow \text{name}$ is a trivial functional dependency, since the dependent **name** is a subset of determinant set $\{\text{roll_no}, \text{name}\}$
Similarly, $\text{roll_no} \rightarrow \text{roll_no}$ is also an example of trivial functional dependency.

2. Non-trivial Functional Dependency

In **Non-trivial functional dependency**, the dependent is strictly not a subset of the determinant.

i.e. If $X \rightarrow Y$ and **Y is not a subset of X**, then it is called Non-trivial functional dependency.

For example,

roll_no	name	age
42	abc	17
43	pqr	18
44	xyz	18

Here, $\text{roll_no} \rightarrow \text{name}$ is a non-trivial functional dependency, since the dependent **name** is **not a subset of** determinant **roll_no**
Similarly, $\{\text{roll_no}, \text{name}\} \rightarrow \text{age}$ is also a non-trivial functional dependency, since **age** is **not a subset of** $\{\text{roll_no}, \text{name}\}$

3. Multivalued Functional Dependency

In **Multivalued functional dependency**, entities of the dependent set are **not dependent on each other**.

i.e. If $a \rightarrow \{b, c\}$ and there exists **no functional dependency** between **b and c**, then it is called a **multivalued functional dependency**.

For example,

roll_no	name	age
42	abc	17
43	pqr	18
44	xyz	18

roll_no	name	age
45	abc	19

Here, $\text{roll_no} \rightarrow \{\text{name}, \text{age}\}$ is a multivalued functional dependency, since the dependents **name** & **age** are **not dependent** on each other (i.e. $\text{name} \rightarrow \text{age}$ or $\text{age} \rightarrow \text{name}$ doesn't exist !)

4. Transitive Functional Dependency

In transitive functional dependency, dependent is indirectly dependent on determinant. i.e. If $\text{a} \rightarrow \text{b}$ & $\text{b} \rightarrow \text{c}$, then according to axiom of transitivity, $\text{a} \rightarrow \text{c}$. This is a **transitive functional dependency**

For example,

enrol_no	name	dept	building_no
42	abc	CO	4
43	pqr	EC	2
44	xyz	IT	1
45	abc	EC	2

Here, $\text{enrol_no} \rightarrow \text{dept}$ and $\text{dept} \rightarrow \text{building_no}$,

Hence, according to the axiom of transitivity, $\text{enrol_no} \rightarrow \text{building_no}$ is a valid functional dependency. This is an indirect functional dependency, hence called Transitive functional dependency.

Non-loss Decomposition in DBMS

Lossless join decomposition is a decomposition of a relation R into relations R1, R2 such that if we perform a natural join of relation R1 and R2, it will return the original relation R. This is effective in removing redundancy from databases while preserving the original data...

In other words by lossless decomposition, it becomes feasible to reconstruct the relation R from decomposed tables R1 and R2 by using Joins.

In Lossless Decomposition, we select the common attribute and the criteria for selecting a common attribute is that the common attribute must be a candidate key or super key in either relation R1, R2, or both.

Decomposition of a relation R into R1 and R2 is a lossless-join decomposition if at least one of the following functional dependencies are in F+ (Closure of functional dependencies)

$$R1 \cap R2 \rightarrow R1$$

OR

$$R1 \cap R2 \rightarrow R2$$

Functional dependencies

Normal Forms in DBMS

Normalization is the process of minimizing **redundancy** from a relation or set of relations. Redundancy in relation may cause insertion, deletion, and update anomalies. So, it helps to minimize the redundancy in relations. **Normal forms** are used to eliminate or reduce redundancy in database tables.

1. First Normal Form –

If a relation contain composite or multi-valued attribute, it violates first normal form or a relation is in first normal form if it does not contain any composite or multi-valued attribute. A relation is in first normal form if every attribute in that relation is **singled valued attribute**.

- Example 1** – Relation STUDENT in table 1 is not in 1NF because of multi-valued attribute STUD_PHONE. Its decomposition into 1NF has been shown in table 2.

STUD_NO	STUD_NAME	STUD_PHONE	STUD_STATE	STUD_COUNTRY
1	RAM	9716271721, 9871717178	HARYANA	INDIA
2	RAM	9898297281	PUNJAB	INDIA
3	SURESH		PUNJAB	INDIA

Table 1

Conversion to first normal form

STUD_NO	STUD_NAME	STUD_PHONE	STUD_STATE	STUD_COUNTRY
1	RAM	9716271721	HARYANA	INDIA
1	RAM	9871717178	HARYANA	INDIA
2	RAM	9898297281	PUNJAB	INDIA
3	SURESH		PUNJAB	INDIA

Table 2

- Example 2** –

ID Name Courses

1	A	c1, c2
2	E	c3
3	M	C2, c3

In the above table Course is a multi-valued attribute so it is not in 1NF.

Below Table is in 1NF as there is no multi-valued attribute

ID Name Course

1	A	c1
1	A	c2
2	E	c3
3	M	c2
3	M	c3

2. Second Normal Form –

To be in second normal form, a relation must be in first normal form and relation must not contain any partial dependency. A relation is in 2NF if it has **No Partial Dependency**, i.e., no non-prime attribute (attributes which are not part of any candidate key) is dependent on any proper subset of any candidate key of the table.

Partial Dependency – If the proper subset of candidate key determines non-prime attribute, it is called partial dependency.

- **Example 1** – Consider table-3 as following below.

STUD_NO	COURSE_NO	COURSE_FEE
1	C1	1000
2	C2	1500
1	C4	2000
4	C3	1000
4	C1	1000
2	C5	2000

{Note that, there are many courses having the same course fee. }

Here,

COURSE_FEE cannot alone decide the value of COURSE_NO or STUD_NO;

COURSE_FEE together with STUD_NO cannot decide the value of COURSE_NO;

COURSE_FEE together with COURSE_NO cannot decide the value of STUD_NO;

Hence,

COURSE_FEE would be a non-prime attribute, as it does not belong to the one only candidate key {STUD_NO, COURSE_NO} ;

But, COURSE_NO → COURSE_FEE, i.e., COURSE_FEE is dependent on

COURSE_NO, which is a proper subset of the candidate key. Non-prime attribute

COURSE_FEE is dependent on a proper subset of the candidate key, which is a partial dependency and so this relation is not in 2NF.

To convert the above relation to 2NF,
 we need to split the table into two tables such as :

Table 1: STUD_NO, COURSE_NO

Table 2: COURSE_NO, COURSE_FEE

Table 1		Table 2	
STUD_NO	COURSE_NO	COURSE_NO	COURSE_FEE
1	C1	C1	1000
2	C2	C2	1500
1	C4	C3	1000
4	C3	C4	2000
4	C1	C5	2000
2	C5		

NOTE: 2NF tries to reduce the redundant data getting stored in memory. For instance, if there are 100 students taking C1 course, we don't need to store its Fee as 1000 for all the 100 records, instead, once we can store it in the second table as the course fee for C1 is 1000.

- **Example 2** – Consider following functional dependencies in relation R (A, B, C, D)
- $AB \rightarrow C$ [A and B together determine C]
- $BC \rightarrow D$ [B and C together determine D]

In the above relation, AB is the only candidate key and there is no partial dependency, i.e., any proper subset of AB doesn't determine any non-prime attribute.

3. Third Normal Form –

A relation is in third normal form, if there is **no transitive dependency** for non-prime attributes as well as it is in second normal form.

A relation is in 3NF if **at least one of the following condition holds** in every non-trivial function dependency $X \rightarrow Y$

1. X is a super key.
2. Y is a prime attribute (each element of Y is part of some candidate key).

STUD_NO	STUD_NAME	STUD_STATE	STUD_COUNTRY	STUD_AGE
1	RAM	HARYANA	INDIA	20
2	RAM	PUNJAB	INDIA	19
3	SURESH	PUNJAB	INDIA	21

Table 4

Transitive dependency – If $A \rightarrow B$ and $B \rightarrow C$ are two FDs then $A \rightarrow C$ is called transitive dependency.

- **Example 1** – In relation STUDENT given in Table 4,
 FD set: { $STUD_NO \rightarrow STUD_NAME$, $STUD_NO \rightarrow STUD_STATE$,
 $STUD_STATE \rightarrow STUD_COUNTRY$, $STUD_NO \rightarrow STUD_AGE$ }
 Candidate Key: { $STUD_NO$ }

For this relation in table 4, STUD_NO \rightarrow STUD_STATE and STUD_STATE \rightarrow STUD_COUNTRY are true. So STUD_COUNTRY is transitively dependent on STUD_NO. It violates the third normal form. To convert it in third normal form, we will decompose the relation STUDENT (STUD_NO, STUD_NAME, STUD_PHONE, STUD_STATE, STUD_COUNTRY, STUD_AGE) as:
STUDENT (STUD_NO, STUD_NAME, STUD_PHONE, STUD_STATE, STUD_AGE)
STATE_COUNTRY (STATE, COUNTRY)

- **Example 2** – Consider relation R(A, B, C, D, E)
A \rightarrow BC,
CD \rightarrow E,
B \rightarrow D,
E \rightarrow A
All possible candidate keys in above relation are {A, E, CD, BC} All attributes are on right sides of all functional dependencies are prime.

4. Boyce-Codd Normal Form (BCNF) –

A relation R is in BCNF if R is in Third Normal Form and for every FD, LHS is super key. A relation is in BCNF iff in every non-trivial functional dependency $X \rightarrow Y$, X is a super key.

- **Example 1** – Find the highest normal form of a relation R(A,B,C,D,E) with FD set as {BC \rightarrow D, AC \rightarrow BE, B \rightarrow E}
Step 1. As we can see, (AC)⁺ = {A,C,B,E,D} but none of its subset can determine all attribute of relation. So AC will be candidate key. A or C can't be derived from any other attribute of the relation, so there will be only 1 candidate key {AC}.
- Step 2. Prime attributes are those attributes that are part of candidate key {A, C} in this example and others will be non-prime {B, D, E} in this example.
- Step 3. The relation R is in 1st normal form as a relational DBMS does not allow multi-valued or composite attribute.
The relation is in 2nd normal form because BC \rightarrow D is in 2nd normal form (BC is not a proper subset of candidate key AC) and AC \rightarrow BE is in 2nd normal form (AC is candidate key) and B \rightarrow E is in 2nd normal form (B is not a proper subset of candidate key AC).
- The relation is not in 3rd normal form because in BC \rightarrow D (neither BC is a super key nor D is a prime attribute) and in B \rightarrow E (neither B is a super key nor E is a prime attribute) but to satisfy 3rd normal for, either LHS of an FD should be super key or RHS should be prime attribute.
- So the highest normal form of relation will be 2nd Normal form.
- **Example 2** –For example consider relation R(A, B, C)
A \rightarrow BC,
B \rightarrow C
A and B both are super keys so above relation is in BCNF.

Key Points –

- BCNF is free from redundancy.
- If a relation is in BCNF, then 3NF is also satisfied.
- If all attributes of relation are prime attribute, then the relation is always in 3NF.
- A relation in a Relational Database is always and at least in 1NF form.
- Every Binary Relation (a Relation with only 2 attributes) is always in BCNF.
- If a Relation has only singleton candidate keys(i.e. every candidate key consists of only 1 attribute), then the Relation is always in 2NF(because no Partial functional dependency possible).
- Sometimes going for BCNF form may not preserve functional dependency. In that case go for BCNF only if the lost FD(s) is not required, else normalize till 3NF only.
- There are many more Normal forms that exist after BCNF, like 4NF and more. But in real world database systems it's generally not required to go beyond BCNF.

Exercise 1: Find the highest normal form in R (A, B, C, D, E) under following functional dependencies.

ABC \rightarrow D

CD \rightarrow AE

Important Points for solving above type of question.

- 1) It is always a good idea to start checking from BCNF, then 3 NF, and so on.
- 2) If any functional dependency satisfied a normal form then there is no need to check for lower normal form. For example, ABC \rightarrow D is in BCNF (Note that ABC is a superkey), so no need to check this dependency for lower normal forms.

Candidate keys in the given relation are {ABC, BCD}

BCNF: ABC \rightarrow D is in BCNF. Let us check CD \rightarrow AE, CD is not a super key so this dependency is not in BCNF. So, R is not in BCNF.

3NF: ABC \rightarrow D we don't need to check for this dependency as it already satisfied BCNF. Let us consider CD \rightarrow AE. Since E is not a prime attribute, so the relation is not in 3NF.

2NF: In 2NF, we need to check for partial dependency. CD is a proper subset of a candidate key and it determines E, which is non-prime attribute. So, given relation is also not in 2 NF. So, the highest normal form is 1 NF.

Multivalued dependency (MVD)

Multivalued dependency (MVD) is having the presence of one or more rows in a table. It implies the presence of one or more other rows in that same table. A multivalued dependency prevents fourth normal form. A multivalued dependency involves at least three attributes of a table.

It is represented with a symbol " \twoheadrightarrow " in DBMS.

X \twoheadrightarrow Y relates one value of X to one value of Y.

$X \twoheadrightarrow Y$ (read as X multidetermines Y) relates one value of X to many values of Y.

A Nontrivial MVD occurs when $X \twoheadrightarrow Y$ and $X \twoheadrightarrow Z$ where Y and Z are not dependent are independent to each other. Non-trivial MVD produces redundancy.

We use multivalued conditions in two different ways –

- To test relations to decide if they are lawful under a given arrangement of practical and multivalued dependencies.
- To determine limitations on the arrangement of lawful relations. We will concern ourselves just with relations that fulfill a given arrangement of practical and multivalued dependencies.

MVD transitive rule

If $A \rightarrow B$ holds, and $B \twoheadrightarrow C$ holds, then $A \twoheadrightarrow C$ holds.

Example

Given FD set is as follows –

ISBN \twoheadrightarrow TITLE, PUBLISHER

ISBN, NO \twoheadrightarrow AUTHOR

PUBLISHER \twoheadrightarrow PU_URL

We need to prove the rule. Consider $A=ISBN, B=PUBLISHER, C=PU_URL$. To find the Transitive rule is implied, find the cover of A^+ and compute.

- Now start with $x=\{ISBN\}$
- The FD ISBN \twoheadrightarrow TITLE, PUBLISHER has LHS which is completely contained in current attribute set x.
- Extend x by FD RHS attribute set, giving $x=\{ISBN, TITLE, PUBLISHER\}$
- Now FD: PUBLISHER \twoheadrightarrow PU_URL is applicable
- Add RHS attribute set of FD to current attribute SET x, giving $x=\{ISBN, TITLE, PUBLISHER, PU_URL\}$

Here we can conclude that ISBN \twoheadrightarrow PU_URL

Multivalued Dependencies

The 4th Normal Form can cause the Multivalued Dependencies. If a relation is in Boyce codee Normal form, it has to remove the multivalued Dependencies.

Explanation – The multivalued dependencies is that, if there is a dependency or relation in a table, then one value has multiple dependencies occur.

Let us consider an example as given below. Consider the following table –

id	department	shift
1	coding	day
2	Hr	day

id	department	shift
3	Network	night

In the above table, id 2 has two departments Hr and Network. And shift timing day and night.

When we select the details with the id 2, then it will result the table as follows –

id	department	shift
2	Hr	day
2	Network	night
2	Hr	night
2	Network	day

This means there exist multivalued dependencies. In this, the relation between department and shift is nothing.

This can be rectified by removing the multivalued dependency as, making this data in to two tables as below –

Table 1

id	department
1	coding
2	Hr
2	network

Table 2

id	shift
1	day
2	day

id	shift
2	night

The 4th normal form is applied to remove the multivalued dependencies in the data table.

The fourth normal form thus defines the multivalued dependencies.

If two or more independent relation are kept in a single relation or we can say **multivalued dependency** occurs when the presence of one or more rows in a table implies the presence of one or more other rows in that same table. Put another way, two attributes (or columns) in a table are independent of one another, but both depend on a third attribute.

A **multivalued dependency** always requires at least three attributes because it consists of at least two attributes that are dependent on a third.

For a dependency $A \twoheadrightarrow B$, if for a single value of A, multiple value of B exists, then the table may have multi-valued dependency. The table should have at least 3 attributes and B and C should be independent for $A \twoheadrightarrow B$ multivalued dependency. For example,

Person	Mobile	Food_Likes
Mahesh	9893/9424	Burger / pizza
Ramesh	9191	Pizza

Person \twoheadrightarrow mobile,

Person \twoheadrightarrow food_likes

This is read as “person multidetermines mobile” and “person multidetermines food_likes.”

Note that a functional dependency is a special case of multivalued dependency. In a functional dependency $X \rightarrow Y$, every x determines exactly one y, never more than one.

Fourth normal form (4NF):

Fourth normal form (4NF) is a level of database normalization where there are no non-trivial multivalued dependencies other than a candidate key. It builds on the first three normal forms (1NF, 2NF and 3NF) and the Boyce-Codd Normal Form (BCNF). It states that, in addition to a database meeting the requirements of BCNF, it must not contain more than one multivalued dependency.

Properties – A relation R is in 4NF if and only if the following conditions are satisfied:

1. It should be in the Boyce-Codd Normal Form (BCNF).

2. the table should not have any Multi-valued Dependency.

A table with a multivalued dependency violates the normalization standard of Fourth Normal Form (4NF) because it creates unnecessary redundancies and can contribute to inconsistent data. To bring this up to 4NF, it is necessary to break this information into two tables.

Example – Consider the database table of a class which has two relations R1 contains student ID(SID) and student name (SNAME) and R2 contains course id(CID) and course name (CNAME).

Table – R1(SID, SNAME)

SID	SNAME
-----	-------

S1	A
----	---

S2	B
----	---

Table – R2(CID, CNAME)

CID	CNAME
-----	-------

C1	C
----	---

C2	D
----	---

When there cross product is done it resulted in multivalued dependencies:

Table – R1 X R2

SID	SNAME	CID	CNAME
-----	-------	-----	-------

S1	A	C1	C
----	---	----	---

S1	A	C2	D
----	---	----	---

S2	B	C1	C
----	---	----	---

S2	B	C2	D
----	---	----	---

Multivalued dependencies (MVD) are:

SID->->CID; SID->->CNAME; SNAME->->CNAME

Joint dependency

Join decomposition is a further generalization of Multivalued dependencies. If the join of R1 and R2 over C is equal to relation R then we can say that a join dependency (JD) exists, where R1 and R2 are the decomposition R1(A, B, C) and R2(C, D) of a given relations R (A, B, C, D). Alternatively, R1 and R2 are a lossless decomposition of R. A JD $\bowtie \{R_1, R_2, \dots, R_n\}$ is said to hold over a relation R if R1, R2,, Rn is a lossless-join decomposition. The $\pi(A, B, C, D), (C, D)$ will be a JD of R if the join of join's attribute is equal to the relation R. Here, $\pi(R_1, R_2, R_3)$ is used to indicate that relation R1, R2, R3 and so on are a JD of R.

Let R is a relation schema R1, R2, R3.....Rn be the decomposition of R. $r(R)$ is said to satisfy join dependency if and only if

$$\bowtie_{i=1}^n \pi_{R_i}(r) = r,$$

Example –

Table – R1

Company	Product
---------	---------

C1	pendrive
----	----------

C1	mic
----	-----

C2	speaker
----	---------

C2	speaker
----	---------

Company->->Product

Table – R2

Agent	Company
-------	---------

Aman	C1
------	----

Aman	C2
------	----

Agent	Company
-------	---------

Mohan	C1
-------	----

Agent->->Company

Table – R3

Agent	Product
-------	---------

Aman	pendrive
------	----------

Aman	mic
------	-----

Aman	speaker
------	---------

Mohan	speaker
-------	---------

Agent->->Product

Table – R1 ⋈ R2 ⋈ R3

Company	Product	Agent
C1	pendrive	Aman
C1	mic	Aman
C2	speaker	speaker
C1	speaker	Aman

Agent->->Product

Fifth Normal Form / Projected Normal Form (5NF):

A relation R is in 5NF if and only if every join dependency in R is implied by the candidate keys of R. A relation decomposed into two relations must have loss-less join Property, which ensures that no spurious or extra tuples are generated, when relations are reunited through a natural join.

Properties – A relation R is in 5NF if and only if it satisfies following conditions:

1. R should be already in 4NF.
2. It cannot be further non loss decomposed (join dependency)

Example – Consider the above schema, with a case as “if a company makes a product and an agent is an agent for that company, then he always sells that product for the company”. Under these circumstances, the ACP table is shown as:

Table – ACP

Agent	Company	Product
A1	PQR	Nut
A1	PQR	Bolt
A1	XYZ	Nut
A1	XYZ	Bolt
A2	PQR	Nut

The relation ACP is again decompose into 3 relations. Now, the natural Join of all the three relations will be shown as:

Table – R1

Agent	Company
A1	PQR
A1	XYZ
A2	PQR

Table – R2

Agent	Product
-------	---------

Agent	Product
-------	---------

A1	Nut
----	-----

A1	Bolt
----	------

A2	Nut
----	-----

Table – R3

Company	Product
---------	---------

PQR	Nut
-----	-----

PQR	Bolt
-----	------

XYZ	Nut
-----	-----

XYZ	Bolt
-----	------

Result of Natural Join of R1 and R3 over ‘Company’ and then Natural Join of R13 and R2 over ‘Agent’and ‘Product’ will be table **ACP**.

Hence, in this example, all the redundancies are eliminated, and the decomposition of ACP is a lossless join decomposition. Therefore, the relation is in 5NF as it does not violate the property of [lossless join](#).

Unit 4

TRANSACTIONS

The transaction is a set of logically related operation. It contains a group of tasks.

A transaction is an action or series of actions. It is performed by a single user to perform operations for accessing the contents of the database.

Example: Suppose an employee of bank transfers Rs 800 from X's account to Y's account. This small transaction contains several low-level tasks:

X's Account

Open_Account(X)

1. Old_Balance = X.balance
2. New_Balance = Old_Balance - 800
3. X.balance = New_Balance
4. Close_Account(X)

Y's Account

1. Open_Account(Y)
2. Old_Balance = Y.balance
3. New_Balance = Old_Balance + 800
4. Y.balance = New_Balance
5. Close_Account(Y)

Operations of Transaction:

Read(X): Read operation is used to read the value of X from the database and stores it in a buffer in main memory.

Write(X): Write operation is used to write the value back to the database from the buffer.

Let's take an example to debit transaction from an account which consists of following operations:

1. R(X);
2. X = X - 500;
3. W(X);

Let's assume the value of X before starting of the transaction is 4000.

- The first operation reads X's value from database and stores it in a buffer.
- The second operation will decrease the value of X by 500. So buffer will contain 3500.
- The third operation will write the buffer's value to the database. So X's final value will be 3500.

But it may be possible that because of the failure of hardware, software or power, etc. that transaction may fail before finished all the operations in the set.

For example: If in the above transaction, the debit transaction fails after executing operation 2 then X's value will remain 4000 in the database which is not acceptable by the bank.

To solve this problem, we have two important operations:

Commit: It is used to save the work done permanently.

Rollback: It is used to undo the work done.

TRANSACTION RECOVERY

UNDO and REDO: lists of transactions UNDO = all transactions running at the last checkpoint REDO = empty For each entry in the log, starting at the last checkpoint If a BEGIN TRANSACTION entry is found for T Add T to UNDO If a COMMIT entry is found for T Move T from UNDO to REDO

- **Types of Transaction Recovery**
- Recovery information is divided into two types:
 - Undo (or Rollback) Operations
 - Redo (or Cache Restore) Operations
- Ingres performs both online and offline recovery, as described in Recovery Modes (see page [Recovery Modes](#)).
- *Undo Operation*
- Undo or transaction backout recovery is performed by the DBMS Server. For example, when a transaction is aborted, transaction log file information is used to roll back all related updates. The DBMS Server writes the Compensation Log Records (CLRs) to record a history of the actions taken during undo operations.
- *Redo Operation*
- A Redo recovery operation is database-oriented. Redo recovery is performed after a server or an installation fails. Its main purpose is to recover the contents of the DMF cached data pages that are lost when a fast-commit server fails. Redo recovery is performed by the recovery process. Redo recovery precedes undo recovery.
- **Redo Operation in a Cluster Environment**
- In an Ingres cluster environment where all nodes are active, the local recovery server performs transaction redo/undo for a failed DBMS server on its node, just like in the non-cluster case. The difference in a cluster installation is that if the recovery process (RCP) dies on one node, either because of an Ingres failure, or a general failure of the hardware, an RCP on another node will take responsibility for cleaning up transactions for the failed nodes.

ACID PROPERTIES

A transaction is a very small unit of a program and it may contain several lowlevel tasks. A transaction in a database system must maintain Atomicity, Consistency, Isolation, and Durability – commonly known as ACID properties – in order to ensure accuracy, completeness, and data integrity.

- **Atomicity** – This property states that a transaction must be treated as an atomic unit, that is, either all of its operations are executed or none. There must be no state in a database where a transaction is left partially completed. States should be defined either

before the execution of the transaction or after the execution/abortion/failure of the transaction.

- **Consistency** – The database must remain in a consistent state after any transaction. No transaction should have any adverse effect on the data residing in the database. If the database was in a consistent state before the execution of a transaction, it must remain consistent after the execution of the transaction as well.
- **Durability** – The database should be durable enough to hold all its latest updates even if the system fails or restarts. If a transaction updates a chunk of data in a database and commits, then the database will hold the modified data.
- If a transaction commits but the system fails before the data could be written on to the disk, then that data will be updated once the system springs back into action.
- **Isolation** – In a database system where more than one transaction are being executed simultaneously and in parallel, the property of isolation states that all the transactions will be carried out and executed as if it is the only transaction in the system. No transaction will affect the existence of any other transaction.

SYSTEM RECOVERY

- Any transaction that was running at the time of failure needs to be undone and restarted
- Any transactions that committed since the last checkpoint need to be redone
- Transactions of type T1 need no recovery • Transactions of type T3 or T5 need to be undone and restarted
- Transactions of type T2 or T4 need to be redone.

Media Failures

System failures are not too severe • Only information since the last checkpoint is affected • This can be recovered from the transaction log • Media failures (disk crashes etc) are more serious • The data stored to disk is damaged • The transaction log itself may be damaged

Recovery from Media Failure

- Restore the database from the last backup
- Use the transaction log to redo any changes made since the last backup
- If the transaction log is damaged you can't do step 2
- Store the log on a separate physical device to the database
- The risk of losing both is then reduced.

MEDIA RECOVERY

If you restore the archived redo log files and data files, then you must perform media recovery before you can open the database. Any database transactions in the archived redo log files not reflected in the data files are applied to the data files, bringing them to a transaction-consistent state before the database is opened.

Media recovery requires a control file, data files (typically restored from backup), and online and archived redo log files containing changes since the time the data files were backed up. Media recovery is most often used to recover from media failure, such as the loss of a file or disk, or a user error, such as the deletion of the contents of a table.

Media recovery can be a complete recovery or a point-in-time recovery. Complete recovery can apply to individual datafiles, tablespaces, or the entire database. Point-in-time recovery applies to the whole database (and also sometimes to individual tablespaces, with automation help from Oracle Recover Manager (RMAN)).

In a complete recovery, you restore backup data files and apply all changes from the archived and online redo log files to the data files. The database is returned to its state at the time of failure and can be opened with no loss of data.

In a point-in-time recovery, you return a database to its contents at a user-selected time in the past. You restore a backup of data files created before the target time and a complete set of archived redo log files from backup creation through the target time. Recovery applies changes between the backup time and the target time to the data files. All changes after the target time are discarded.

RMAN enables you to perform both a complete and a point-in-time recovery of your database. However, this documentation focuses on complete recovery.

TWO-PHASE COMMIT (2PC)

A two-phase commit is a standardized protocol that ensures that a database commit is implementing in the situation where a commit operation must be broken into two separate parts.

In database management, saving data changes is known as a commit and undoing changes is known as a rollback. Both can be achieved easily using transaction logging when a single server is involved, but when the data is spread across geographically-diverse servers in distributed computing (i.e., each server being an independent entity with separate log records), the process can become more tricky.

Two Phase commit protocol is a type of distributed commit protocol. There are two different types of databases. In a local database system, every transaction needs to be committed. Therefore, the transaction manager has the role to commit the decision by conveying it to the reporting manager.

when it comes to a distributed system, the transaction manager should convey it from all the servers from various sites included in the distributed system to commit the decision. When each server completes the processing at each site. The transaction reaches a partially committed state. But it has to wait until all the transaction reaches that state. Once all the transactions from different servers reach the partially committed state, the transaction manager can commit the transaction. However, it is necessary that all the sites must commit the transaction.

Role of two phase commit protocol in Database Management

- Distributed Commit Protocols
- One-phase commit protocol
- Two Phase commit protocol
- Three Phase commit protocol
- Advantages of Two Phase commit protocol
- Disadvantages of Two Phase commit protocol

SAVE POINTS

A save point is a way of implementing sub transactions (also known as nested transactions) within a relational database management system by indicating a point within a

transaction that can be "rolled back to" without affecting any work done in the transaction before the savepoint was created.

SAVEPOINT command

SAVEPOINT command is used to temporarily save a transaction so that you can rollback to that point whenever required.

Following is savepoint command's syntax,

```
SAVEPOINT savepoint_name;
```

Copy

In short, using this command we can **name** the different states of our data in any table and then rollback to that state using the **ROLLBACK** command whenever required.

savepoint

Specify the name of the savepoint to be created.

Savepoint names must be distinct within a given transaction. If you create a second savepoint with the same identifier as an earlier savepoint, then the earlier savepoint is erased. After a savepoint has been created, you can either continue processing, commit your work, roll back the entire transaction, or roll back to the savepoint.

Example

Creating Savepoints: Example To update the salary for **Banda** and **Greene** in the sample table **hr.employees**, check that the total department salary does not exceed 314,000, then reenter the salary for **Greene**:

```
UPDATE employees  
  
  SET salary = 7000  
  
  WHERE last_name = 'Banda';  
  
SAVEPOINT banda_sal;  
  
UPDATE employees  
  
  SET salary = 12000  
  
  WHERE last_name = 'Greene';  
  
SAVEPOINT greene_sal;  
  
SELECT SUM(salary) FROM employees;  
  
ROLLBACK TO SAVEPOINT banda_sal;
```

```
UPDATE employees  
  
    SET salary = 11000  
  
    WHERE last_name = 'Greene';  
  
COMMIT;
```

Recovery Facilities

Checkpoint facility allows updates to the database for getting the latest patches to be made permanent and keep secure from vulnerability. Recovery manager allows the database system for restoring the database to a reliable and steady-state after any failure occurs.

SQL BASIC FACILITIES

In addition to the advanced facilities noted above, SQL is rich in the type of ease of use capabilities that are necessary to support relational databases from the simple to the complex. Table Facility First and foremost, SQL provides a table facility that enables a prompted, intuitive interface for the following functions: 9 Defining databases 9 Populating databases with rows 9 Manipulating databases.

Table Editor SQL also provides a table editor that makes it easy for you to perform the following functions against rows in table data that is structured in row and column format:. 9 Access 9 Insert 9 Update 9 Delete Query Facility: With the Query facility, SQL permits you to interactively define queries and have results displayed in a variety of report formats including the following: 9 Tabular 9 Matrix 9 Free format For those readers who have a System i5 background, you will notice that SQL brings with it its own naming scheme that is significantly different from corresponding native objects. See table 4-1 for specifics

CONCURRENCY

Database concurrency is the ability of a database to allow multiple users to affect multiple transactions. This is one of the main properties that separates a database from other forms of data storage, like spreadsheets.

The ability to offer concurrency is unique to databases. Spreadsheets or other flat file means of storage are often compared to databases, but they differ in this one important regard.

Spreadsheets cannot offer several users the ability to view and work on the different data in the same file, because once the first user opens the file it is locked to other users. Other users can read the file, but may not edit data.

NEED FOR CONCURRENCY

Database concurrency is the ability of a database to allow multiple users to affect multiple transactions. This is one of the main properties that separates a database from other forms of data storage, like spreadsheets.

The ability to offer concurrency is unique to databases. Spreadsheets or other flat file means of storage are often compared to databases, but they differ in this one important regard.

Spreadsheets cannot offer several users the ability to view and work on the different data in the same file, because once the first user opens the file it is locked to other users. Other users can read the file, but may not edit data.

LOCKING PROTOCOLS

Lock Based Protocol in DBMS

The database management system (DBMS) stores data that can connect with one another as well as can be altered at any point. There are instances where more than one user may attempt to access the same data item simultaneously, resulting in concurrency.

As a result, there is a requirement to handle concurrency in order to handle the concurrent processing of transactions across many databases in the picture. Lock based protocol in dbms are an example of such an approach.

Introduction to Lock Based Protocol

We can define a lock based protocol in dbms as a mechanism that is responsible to prevent a transaction from reading or writing data until the necessary lock is obtained. The concurrency problem can be solved by securing or locking a transaction to a specific user. The lock is a variable that specifies which activities are allowed on a certain data item.

Types of Locks in DBMS

In DBMS Lock based Protocols, there are two modes for locking and unlocking data items Shared Lock (lock-S) and Exclusive Lock (lock-X). Let's go through the two types of locks in detail:

Shared Lock

- Shared Locks, which are often denoted as lock-S(), are defined as locks that provide Read-Only access to the information associated with them. Whenever a shared lock is used on a database, it can be read by several users, but these users who are reading the information or the data items will not have the permission to edit it or make any changes to the data items.
- To put it another way, we can say that shared locks don't provide the access to write. Because numerous users can read the data items simultaneously, multiple shared locks can be installed on them at the same time, but the data item must not have any other locks connected with it.
- A shared lock, also known as a read lock, is solely used to read data objects. Read integrity is supported via shared locks.
- Shared locks can also be used to prevent records from being updated.
- S-lock is requested via the Lock-S instruction.

Exclusive Lock

- Exclusive Lock allows the data item to be read as well as written. This is a one-time use mode that can't be utilized on the exact data item twice. To obtain X-lock, the user

needs to make use of the lock-x instruction. After finishing the 'write' step, transactions can unlock the data item.

- By imposing an X lock on a transaction that needs to update a person's account balance, for example, you can allow it to proceed. As a result of the exclusive lock, the second transaction is unable to read or write.
- The other name for an exclusive lock is write lock.
- At any given time, the exclusive locks can only be owned by one transaction.

Example of exclusive locks: Consider the instance where the value of a data item X is equal to 50 and a transaction requires a deduction of 20 from the data item X. By putting a Y lock on this particular transaction, we can make it possible. As a result, the exclusive lock prevents any other transaction from reading or writing.

Types of Lock-Based Protocols

There are basically four lock based protocols in dbms namely Simplistic Lock Protocol, Pre-claiming Lock Protocol, Two-phase Locking Protocol, and Strict Two-Phase Locking Protocol. Let's go through each of these lock-based protocols in detail.

Simplistic Lock Protocol

The simplistic method is defined as the most fundamental method of securing data during a transaction. Simple lock-based protocols allow all transactions to lock the data before inserting, deleting, or updating it. After the transaction is completed, the data item will be unlocked.

Pre-Claiming Lock Protocol

Pre-claiming Lock Protocols are known to assess transactions to determine which data elements require locks. Prior to actually starting the transaction, it asks the Database management system for all of the locks on all of the data items. The pre-claiming protocol permits the transaction to commence if all of the locks are obtained. Whenever the transaction is finished, the lock is released. This protocol permits the transaction to roll back if all of the locks are not granted, and then waits until all of the locks are granted.

TWO-PHASE LOCKING PROTOCOL

If Locking as well as the Unlocking can be performed in 2 phases, a transaction is considered to follow the Two-Phase Locking protocol. The two phases are known as the growing and shrinking phase.

1. Growing Phase: In this phase, we can acquire new locks on data items but none of these locks can be released.
2. Shrinking Phase: In this phase, the existing locks can be released but no new locks can be obtained.

Two-phase locking helps to reduce the amount of concurrency in a schedule but just like the two sides of a coin two-phase locking has a few cons too. The protocol raises transaction processing costs and may have unintended consequences. The likelihood of establishing deadlocks is one bad result.

Strict Two-Phase Locking Protocol

In DBMS, Cascaded rollbacks are avoided with the concept of a Strict Two-Phase Locking Protocol. This protocol necessitates not only two-phase locking but also the

retention of all exclusive locks until the transaction commits or aborts. The two-phase is with deadlock.

It is responsible for assuring that if 1 transaction modifies data, there can be no other transaction that will be able to read it until the first transaction commits. The majority of database systems use a strict two-phase locking protocol.

Deadlock

When a transaction must wait an unlimited period for a lock, it is referred to as starvation. The following are the causes of starvation :

1. When the locked item waiting scheme is not correctly controlled.
2. When a resource leak occurs.
3. The same transaction is repeatedly chosen as a victim.

Let's know how starvation can be prevented. Random process selection for resource or processor allocation should be avoided since it encourages hunger. The resource allocation priority scheme should contain ideas like aging, in which a process' priority rises as it waits longer. This prevents starvation.

Deadlock- In a circular chain, a deadlock situation occurs when two or more processes are expecting each other to release a resource, or when more than 2 processes are waiting for the resource.

Two-Phase Locking –

A transaction is said to follow the Two-Phase Locking protocol if Locking and Unlocking can be done in two phases.

1. **Growing Phase:** New locks on data items may be acquired but none can be released.
 2. **Shrinking Phase:** Existing locks may be released but no new locks can be acquired.
- Note** – If lock conversion is allowed, then upgrading of lock(from S(a) to X(a)) is allowed in the Growing Phase, and downgrading of lock (from X(a) to S(a)) must be done in shrinking phase.

Let's see a transaction implementing 2-PL.

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

This is just a skeleton transaction that shows how unlocking and locking work with 2-PL.
Note for:

Transaction T₁:

- The growing Phase is from steps 1-3.
- The shrinking Phase is from steps 5-7.
- Lock Point at 3

Transaction T₂:

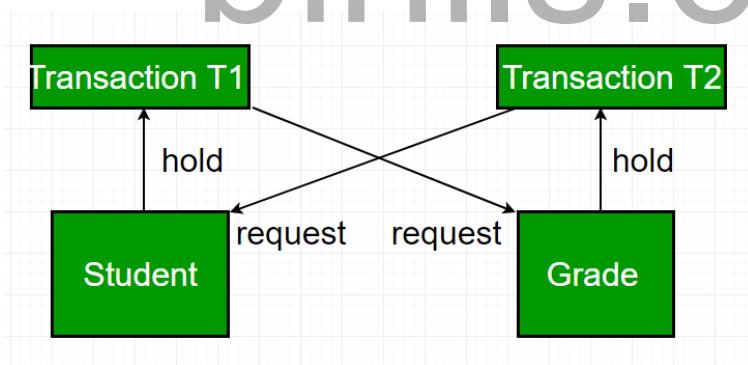
- The growing Phase is from steps 2-6.
- The shrinking Phase is from steps 8-9.
- Lock Point at 6

DEADLOCK

In a database, a deadlock is an unwanted situation in which two or more transactions are waiting indefinitely for one another to give up locks. Deadlock is said to be one of the most feared complications in DBMS as it brings the whole system to a Halt.

Example – let us understand the concept of Deadlock with an example : Suppose, Transaction T₁ holds a lock on some rows in the Students table and **needs to update** some rows in the Grades table. Simultaneously, Transaction T₂ holds locks on those very rows (Which T₁ needs to update) in the Grades table **but needs** to update the rows in the Student table **held by Transaction T₁**.

Now, the main problem arises. Transaction T₁ will wait for transaction T₂ to give up the lock, and similarly, transaction T₂ will wait for transaction T₁ to give up the lock. As a consequence, All activity comes to a halt and remains at a standstill forever unless the DBMS detects the deadlock and aborts one of the transactions.



Deadlock

Avoidance

—

When a database is stuck in a deadlock, It is always better to avoid the deadlock rather than restarting or aborting the database. The deadlock avoidance method is suitable for smaller databases whereas the deadlock prevention method is suitable for larger databases. One method of avoiding deadlock is using application-consistent logic. In the above-given example, Transactions that access Students and Grades should always access the tables in the same order. In this way, in the scenario described above, Transaction T₁ simply waits for transaction T₂ to release the lock on Grades before it begins. When transaction T₂ releases the lock, Transaction T₁ can proceed freely. Another method for avoiding deadlock is to apply both row-level locking mechanism and READ COMMITTED isolation level. However, It does not guarantee to remove deadlocks completely.

Deadlock

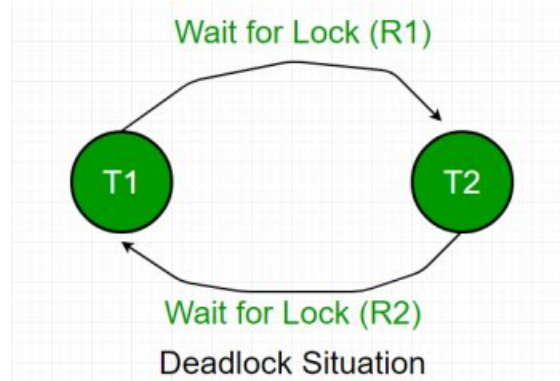
Detection

–

When a transaction waits indefinitely to obtain a lock, The database management system should detect whether the transaction is involved in a deadlock or not.

Wait-for-graph is one of the methods for detecting the deadlock situation. This method is suitable for smaller databases. In this method, a graph is drawn based on the transaction and their lock on the resource. If the graph created has a closed-loop or a cycle, then there is a deadlock.

For the above-mentioned scenario, the Wait-For graph is drawn below



Deadlock

prevention

–

For a large database, the deadlock prevention method is suitable. A deadlock can be prevented if the resources are allocated in such a way that deadlock never occurs. The DBMS analyzes the operations whether they can create a deadlock situation or not, If they do, that transaction is never allowed to be executed.

Deadlock prevention mechanism proposes two schemes :

- **Wait-Die**

Scheme

–

In this scheme, If a transaction requests a resource that is locked by another transaction, then the DBMS simply checks the timestamp of both transactions and allows the older transaction to wait until the resource is available for execution. Suppose, there are two transactions T1 and T2, and Let the timestamp of any transaction T be TS (T). Now, If there is a lock on T2 by some other transaction and T1 is requesting for resources held by T2, then DBMS performs the following actions:

Checks if $TS(T1) < TS(T2)$ – if T1 is the older transaction and T2 has held some resource, then it allows T1 to wait until resource is available for execution. That means if a younger transaction has locked some resource and an older transaction is waiting for it, then an older transaction is allowed to wait for it till it is available. If T1 is an older transaction and has held some resource with it and if T2 is waiting for it, then T2 is killed and restarted later with random delay but with the same timestamp. i.e. if the older transaction has held some resource and the younger transaction waits for the resource, then the younger transaction is killed and restarted with a very minute delay with the same timestamp. This scheme allows the older transaction to wait but kills the younger one.

- **Wound**

Wait

Scheme

–

In this scheme, if an older transaction requests for a resource held by a younger transaction, then an older transaction forces a younger transaction to kill the transaction and release the resource. The younger transaction is restarted with a minute delay but with the same timestamp. If the younger transaction is requesting a resource that is held

by an older one, then the younger transaction is asked to wait till the older one releases it.

SERIALIZABILITY

A schedule is serialized if it is equivalent to a serial schedule. A concurrent schedule must ensure it is the same as if executed serially means one after another. It refers to the sequence of actions such as read, write, abort, commit are performed in a serial manner.

Example

Let's take two transactions T1 and T2,

If both transactions are performed without interfering each other then it is called as serial schedule, it can be represented as follows –

T1	T2
READ1(A)	
WRITE1(A)	
READ1(B)	
C1	
	READ2(B)
	WRITE2(B)
	READ2(B)
	C2

Non serial schedule – When a transaction is overlapped between the transaction T1 and T2.

Example

Consider the following example –

T1	T2
READ1(A)	
WRITE1(A)	

T1	T2
	READ2(B)
	WRITE2(B)
READ1(B)	
WRITE1(B)	
READ1(B)	

Types of serializability

There are two types of serializability –

View serializability

A schedule is view-serializability if it is viewed equivalent to a serial schedule.

The rules it follows are as follows –

- T1 is reading the initial value of A, then T2 also reads the initial value of A.
- T1 is the reading value written by T2, then T2 also reads the value written by T1.
- T1 is writing the final value, and then T2 also has the write operation as the final value.

Conflict serializability

It orders any conflicting operations in the same way as some serial execution. A pair of operations is said to conflict if they operate on the same data item and one of them is a write operation.

That means

- Read_i(x) read_j(x) - non conflict read-read operation
- Read_i(x) write_j(x) - conflict read-write operation.
- Write_i(x) read_j(x) - conflict write-read operation.
- Write_i(x) write_j(x) - conflict write-write operation.

RECOVERY ISOLATION LEVELS

In case of transaction the term ACID has been used significantly to state some of important properties that a transaction must follow. We all know ACID stands for Atomicity, Consistency, Isolation and Durability and these properties collectively called as ACID Properties.

Properties of transaction

Database system ensures ACID property –

- **Atomicity** – Either all or none of the transaction operation is done.
- **Consistency** – A transaction transfer from one consistent (correct) state to another consistent state.
- **Isolation** – A transaction is isolated from other transactions. i.e. A transaction is not affected by another transaction. Although multiple transactions execute concurrently it must appear as if the transaction are running serially (one after the other).
- **Durability** – The results of transactions are permanent i.e. the result will never be lost with subsequent failure, durability refers to long lasting i.e. permanency.

Isolation

It determines the visibility of transactions of other systems. A lower level allows every user to access the same data. Therefore, it involves high risk of data privacy and security of the system. However, a higher isolation level reduces the type of concurrency over the data but requires more resources and is slower than lower isolation levels.

The isolation protocols help safeguards the data from unwanted transactions. They maintain the integrity of every data by defining how and when the changes made by one operation are visible to others.

Levels of isolation

There are four levels of isolations which are explained below –

- **Read Uncommitted** – It is the lowest level of isolation. At this level; the dirty reads are allowed, which means one can read the uncommitted changes made by another.
- **Read committed** – It allows no dirty reads, and clearly states that any uncommitted data is committed now it is read.
- **Repeatable Read** – This is the most restricted level of isolation. The transaction holds read locks on all the rows it references and write locks over all the rows it updates/inserts/deletes. So, there is no chance of non-repeatable reads.
- **Serializable** – The highest level of civilization. It determines that all concurrent transactions be executed serially.

Example

Consider an example of isolation.

What is the isolation level of transaction E?

session begins

SET GLOBAL TRANSACTION

ISOLATION LEVEL SERIALIZABLE;

session ends

session begins

SET SESSION TRANSACTION

ISOLATION LEVEL REPEATABLE READ;

transaction A

transaction B

SET TRANSACTION

ISOLATION LEVEL READ UNCOMMITTED;

transaction C

SET TRANSACTION

ISOLATION LEVEL READ COMMITTED;

transaction D

transaction E

session ends

Check which option –

A- Serializable

B- Repeatable read

C- Read uncommitted

Solution

Repeatable Read is the right answer.

Reason & Explanation

- **Step 1** – In the above program, the first session starts and ends without doing any transaction.
- **Step 2** – The second session begins at session-level with isolation level "Repeatable Read". Transaction A & B gets executed with these settings.
- **Step 3** – Once again a new transaction begins with isolation level "Read uncommitted". This setting is used only for "Transaction C" since "Set transaction" alone is mentioned. If the "SET transaction" is used without global or session keywords, then these particular settings will work only for a single transaction.
- **Step 4** – Once again "Set Transaction" with isolation level Read committed works only for Transaction D. (Refer step 3 for reason)
- **Step 5** – "Transaction E" gets continued at the "Repeatable Read" since the transaction started at step 2 has not ended still. Transaction isolation level set at Step 3 and Step 4 vanishes once a single transaction is executed. So, automatically "Transaction E" will refer to the prior transaction settings.

Concurrency Control in SQL Server

A “Transaction” in SQL Server

The standard definition of a transaction states that “every query that runs in a SQL Server is in a transaction,” that means any query you run on a SQL Server is considered as being in a transaction. It could either be a simple SELECT query or any UPDATE or ALTER query.

- If you run a query without mentioning the BEGIN TRAN keyword then it would be considered an **implicit** transition.
- If you run a query that starts with BEGIN TRAN and ends with COMMIT or ROLLBACK, then it would be considered an **explicit** transaction.

Transaction Properties

A database management system (DBMS) is considered a relational database management system (RDBMS) if it follows the transactional properties, ACID.

- A: Atomicity
- C: Consistency
- I: Isolation
- D: Durability

The SQL Server takes care of the Atomicity, Consistency, and Durability of the system, and the user has to care about the Isolation property of the transaction. The meaning of each of these properties is described below, as it applies to a transaction.

Atomicity

Transaction work should be atomic, which means all the work is one unit. If the user performs a transition, either the transaction should complete and perform all the asked operations, or it should fail and don't do anything. Atomicity deals with the transaction process and an RDBMS transaction does not leave the work incomplete.

Consistency

After the transaction is completed, the database should not be left in an inconsistent state, which means the data on which transaction is applied must be logically correct, according to the rules of the system.

Isolation

If two transactions are applied on a similar database, then both the transaction should be isolated from each other, and the user must see the result. It can also be defined as a transaction that should see the data only after or before the concurrent transaction process is completed, which means if a one transaction process is in between, the other transaction process should wait until the first transaction is completed.

For instance, if **A** performs a transaction process on data **d1**, and before the transaction process gets completed, **B** also performs another transaction process on the same data **d1**. Here, the isolation property will isolate the transaction process of **A** and **B**, and the transaction process of **B** will only start after the transaction process of **A** gets completed.

Durability

Even if the system fails, the transaction should be persistent, which means, if the system fails during a transaction process, the transaction should be dropped, too, without affecting the data.

SQL FACILITIES FOR CONCURRENCY

Concurrency is a situation that arises in a database due to the transaction process. Concurrency occurs when two or more than two users are trying to access the same data or information. DBMS concurrency is considered a problem because accessing data simultaneously by two different users can lead to inconsistent results or invalid behaviour.

Concurrency Problem Types

The concurrency problem mostly arises when both the users try to write the same data, or when one is writing and the other is reading. Apart from this logic, there are some common types of concurrency problems:

- Dirty Reads
- Lost Updates
- Non-repeatable Reads
- Phantom Reads

Dirty Read

This problem occurs when another process reads the changed, but uncommitted data. For instance, if one process has changed data but not committed it yet, another process is able to read the same data. This leads to the inconsistent state for the reader.

Lost Updates

This problem occurs when two processes try to manipulate the same data simultaneously. This problem can lead to data loss, or the second process might overwrite the first process's change.

Non-repeatable Reads

This problem occurs when one process is reading the data, and another process is writing the data. In non-repeatable reads, the first process reading the value might get two different values, as the changed data is read a second time because the second process changes the data.

Phantom Reads

If two same queries executed by two users show different output, then it would be a Phantom Read problem. For instance, If user **A** select a query to read some data, at the same time the user **B** insert some new data but the user **A** only get able to read the old data at the first attempt, but when user **A** re-query the same statement then he/she gets a different set of data.

Solve Concurrency Problems

SQL Server provides 5 different levels of transaction isolation to overcome these Concurrency problems. These 5 isolation levels work on two major concurrency models:

1. Pessimistic model - In the pessimistic model of managing concurrent data access, the readers can block writers, and the writers can block readers.
2. Optimistic model - In the optimistic model of managing concurrent data access, the readers cannot block writers, and the writers cannot block readers, but the writer can block another writer.

Note that readers are users are performing the SELECT operations. Writers are users are performing INSERT, ALTER, UPDATE, S.E.T. operations.

Isolation Level

When we connect to a SQL server database, the application can submit queries to the database with one of five different isolation levels. These levels are:

- Read Uncommitted
- Read Committed
- Repeatable Read
- Serializable
- Snapshot

Out of these five isolation levels, Read Uncommitted, Read Committed, Repeatable Read, and Serializable come under the pessimistic concurrency model. Snapshot comes under the optimistic concurrency model. These levels are ordered in terms of the separation of work by two different processes, from minimal separation to maximal.

Let's look at each of these isolation levels and how they affect concurrency of operations.

Read Uncommitted

This is the first level of isolation, and it comes under the pessimistic model of concurrency. In Read Uncommitted, one transaction is allowed to read the data that is about to be changed by the commit of another process. Read Uncommitted allows the dirty read problem.

Read Committed

This is the second level of isolation and also falls under the pessimistic model of concurrency. In the Read Committed isolation level, we are only allowed to read data that is committed, which means this level eliminates the dirty read problem. In this level, if you are reading data then the concurrent transactions that can delete or write data, some work is blocked until other work is complete.

Repeatable Read

The Repeatable Read isolation level is similar to the Read Committed level and eliminates the Non-Repeatable Read problem. In this level, the transaction has to wait till another transaction's update or read query is complete. But if there is an insert transaction, it does not wait for anyone. This can lead to the Phantom Read problem.

Serializable

This is the highest level of isolation in the pessimistic model. By implementing this level of isolation, we can prevent the Phantom Read problem. In this level of isolation, we can ask any transaction to wait until the current transaction completes.

Snapshot

Snapshot follows the optimistic model of concurrency, and this level of isolation takes a snapshot of the current data and uses it as a copy for the different transactions. Here each

transaction has its copy of data, so if a user tries to perform a transaction like an update or insert, it asks him to re-verify all the operation before the process gets started executing.

binils.com