Module 1 - Introduction To Database Concepts

1. File Systems v/s Database

Points	File Processing System	DBMS
Data redundancy & inconsistency	The problem of duplication and inconsistency in data exists in FPS.	There is no redundancy and inconsistency in data due to centralization of the database.
Ease of data access	Accessing data in file system isn't as easy as DBMS.	Accessing data is easier in DBMS as compared to a file system.
Data independence	There is no data independence in file system.	Data independence exists in DBMS.
Atomicity	Atomicity is not present in file system.	DBMS provides atomicity of transactions.
Concurrency control	It doesn't have concurrency control.	DBMS has concurrency control.
Recovery	File system doesn't provide the facility of recovery, in case of data loss.	DBMS provides the facility of data backup and recovery.
Security	Data security is less in file processing system.	DBMS offers high data security.
Cost	File system is relatively cheaper as compared to DBMS.	DBMS is costlier as compared to a file system.

2. Database Administrator. Define roles and responsibility for the DBA

1. A Database Administrator, Database Analyst or Database Developer is the person responsible for managing the information within an organization.
2. As most companies continue to experience inevitable growth of their databases, these positions are probably the most solid within the IT industry.
3. The DBA has many different responsibilities, but the overall goal of the DBA is to keep the server up at all times and to provide users with access to the required information when they need it.
4. The DBA makes sure that the database is protected and that any chance of data loss is minimized.
5. A DBA can be a programmer who, by default or by volunteering, took over the responsibility of maintaining a SQL Server during project development and enjoyed the job so much that he switched.

Roles & Responsibilities:

1. Schema Definition: The DBA definition the logical Schema of the database.A Schema refers to the overall logical structure of the database. According to this schema, database will be developed to store required data for an organization.

2. Storage Structure and Access Method Definition: The DBA decides how the data is to be represented in the stored database.

3. Assisting Application Programmers: The DBA provides assistance to application programmers to develop application programs.

4. Physical Organization Modification: The DBA modifies the physical organization of the database to reflext the changing needs of the organization or to improve performance.

5. Approving Data Access: The DBA determines which user needs access to which part of the database. According to this,various types of authorizations are granted to different users.

6. Monitoring Performance: The DBA monitors performance of the system.The DBA ensures that better performance is maintained by making changes in physical or logical schema if required.

7. Backup and Recovery: Database should not be lost or damaged. The DBA ensures this periodically backing up the database on magnetic tapes or remote servers. In case of failure, such as virus attack database is recovered from this backup.

3. Write short notes on :

Data Independence

1. Data independence is a form of database management that keeps data separated from all programs that make use of it.
2. As a cornerstone for the idea of a DBMS or database management system, this independence ensures that the data cannot be redefined or reorganized by any of the programs that make use of it.
3. In this manner, the data remains accessible, but it is also stable and cannot be corrupted by the applications.
4. Within the environment of a centralized DBMS, database management relies on the process of data independence.
5. While data transparency still exists as far as the ability of different applications to access and use the data for completing tasks, no program or application can read and then begin to make changes to the data itself.
6. The consistency of the information makes the overall process of maintaining and managing a single database or multiple databases within a single environment much easier.
7. One of the functions of data independence is to restrict access to the storage structure of the data by the user applications.
8. By effectively hiding the specifics of the structure from the applications, the potential for any one application to alter the nature of the data becomes impossible.
9. As a result, the essential data retains its integrity and remains consistent no matter how many databases or database applications access it.

Types of Data Independence: (Extra, not mandatory to study)

Physical Data Independence

- This approach has to do with altering the organization or storage procedures related to the data, rather than modifying the data itself.
- Accomplishing this shift in file organization or the indexing strategy used for the data does not require any modification to the external structure of the applications, meaning that users of the applications are not likely to notice any difference at all in the function of their programs.

Logical Data Independence

- Logical data independence makes it possible to change the structure of the data independently of modifying the applications or programs that make use of the data.
- There is no need to rewrite current applications as part of the process of adding to or removing data from then system.

Data Abstraction

Data Abstraction refers to the process of hiding irrelevant details from the user.

Example

1. If we want to access any mail from our Gmail then we don't know where that data is physically stored i.e is the data present in India or USA or what data model has been used to store that data. We are not concerned about these things.
2. We are only concerned with our email.
3. So, information like these i.e. location of data and data models are irrelevant to us and we in used front end system.

Levels of Data Abstraction

There are mainly three levels of data abstraction and we divide it into three levels in order to achieve Data Independence.

- View Level
- Conceptual Level
- Physical Level

Level 1 : Physical Level

1. It defines how data are stored.
2. It tells the actual location of the data that is being stored by the user.
3. The Database Administrators(DBA) decide that which data should be kept at which particular disk drive, how the data has to be arranged, where it has to be stored etc.
4. They decide if the data has to be centralized or distributed.

Level 2 : Conceptual Level

1. This level defines what data are stored in database & What relationship exists among these data.
2. Logical levels decide structure of entire database.
3. DBA use the logical level for abstraction purpose.

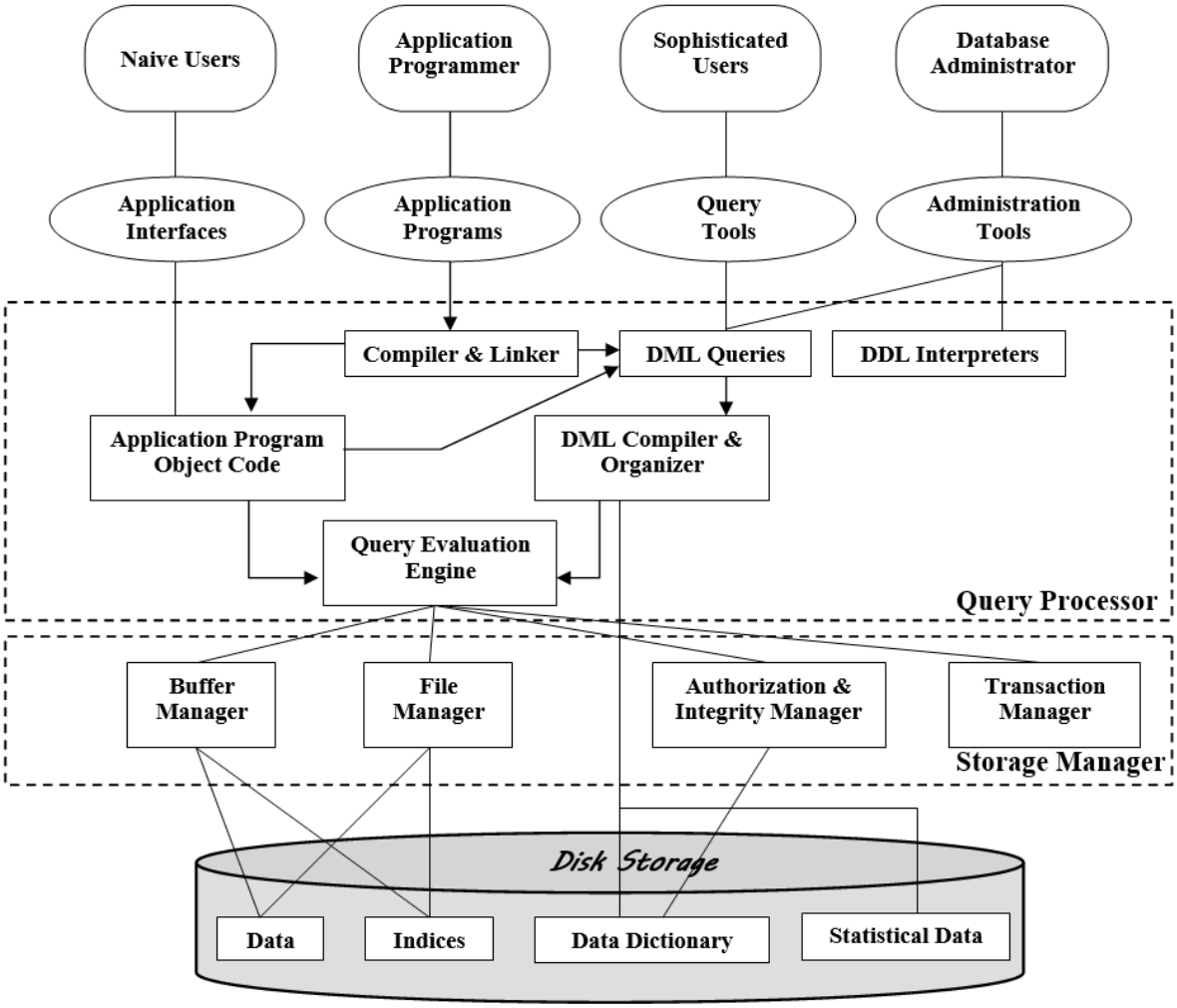
Level 3 : View Level

1. This level tells the application about how the data should be shown to the user.
2. Different views of same database can be created for user to interact with database for user friendly approach.

4. DBMS System Architecture

The architecture of a database system is greatly influenced by the underlying computer system on which the database is running:

i. Centralized.
ii. Client-server.
iii. Parallel (multi-processor).
iv. Distributed



<u>Database Users:</u>

Users are differentiated by the way they expect to interact with the system:

<u>1. Application programmers:</u>

- Application programmers are computer professionals who write application programs. Application programmers can choose from many tools to develop user interfaces.
- Rapid application development (RAD) tools are tools that enable an application programmer to construct forms and reports without writing a program.

2. Sophisticated users:

- Sophisticated users interact with the system without writing programs. Instead, they form their requests in a database query language.
- They submit each such query to a query processor, whose function is to break down DML statements into instructions that the storage manager understands.
-

3. Specialized users :

- Specialized users are sophisticated users who write specialized database applications that do not fit into the traditional data-processing framework.
- Among these applications are computer-aided design systems, knowledge base and expert systems, systems that store data with complex data types (for example, graphics data and audio data), and environment-modeling systems.

4. Naïve users :

- Naive users are unsophisticated users who interact with the system by invoking one of the application programs that have been written previously.
- For example, a bank teller who needs to transfer $50 from account A to account B invokes a program called transfer. This program asks the teller for the amount of money to be transferred, the account from which the money is to be transferred, and the account to which the money is to be transferred.

Database Administrator:

- Coordinates all the activities of the database system. The database administrator has a good understanding of the enterprise's information resources and needs.
- Database administrator's duties include:
    - Schema definition
    - Storage structure and access method definition.
    - Schema and physical organization modification
    - Granting user authority to access the database
    - Monitoring performance and responding to changes in requirements.

Query Processor:

There are various parts of query processor

DDL interpreter

This will interprets DDL statements and fetch the definitions in the data dictionary.

DML compiler

- This will translates DML statements in a query language into low level instructions that the query evaluation engine understands.
- A query can usually be translated into any of a number of alternative evaluation plans for same query result DML compiler will select best plan for query optimization.

Query evaluation engine
This engine will execute low-level instructions generated by the DML compiler on DBMS.

Storage Manager/Storage Management:

- A storage manager is a program module which acts like interface between the data stored in a database and the application programs and queries submitted to the system.
- Thus, the storage manager is responsible for storing, retrieving and updating data in the database.
- The storage manager components include:
    - Authorization and integrity manager
    - Transaction manager
    - File manager
    - Buffer manager

- Data files: Stored in the database itself.
- Data dictionary: Stores metadata about the structure of the database.
- Indices: Provide fast access to data items.

Module 2 - Entity-Relationship Data Model

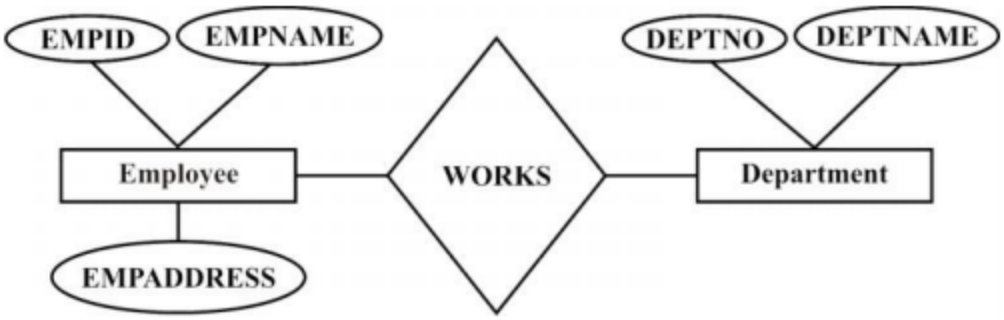
1. ER/EER Design

Entity Relationship:

1. In software engineering, an entity relational model is an abstract and conceptual representation of data entity-relationship modeling is a database modeling method, used to produce a type of conceptual schema or semantic data model of a system, often a relational database, and its requirements in a top-down fashion.
2. Diagrams created by this process are called entity-relationship diagrams, ER diagrams, or ERDs.
3. In 1976, the Entity-relationship model developed by Chen, ER Model is a high-level Conceptual model which used Conceptual design of database where relational models are used to logical design of database

Diagram

1. A database can be modeled as A collection of entities Relationship among the entities
2. An entity is a real-world object that exists and is distinguishable from other entities
3. Example Person, company, event, plant
4. All the entities in the data model have attributes as known as properties of an entities
5. Example: people have names and addresses An Entity set is a set of entities of all same type that share the same properties. Example: a set of all persons , companies, trees, holidays



- Rectangles represent entity sets.
- Diamonds represent relationship sets.
- Lines link attributes to entity sets and entity sets to relationship sets.
- Underline indicates primary key attributes
- Ellipses represent an attributes
- Double Lines represent total participation of an entity in a relationship set
- Double rectangle represent a weak entity sets

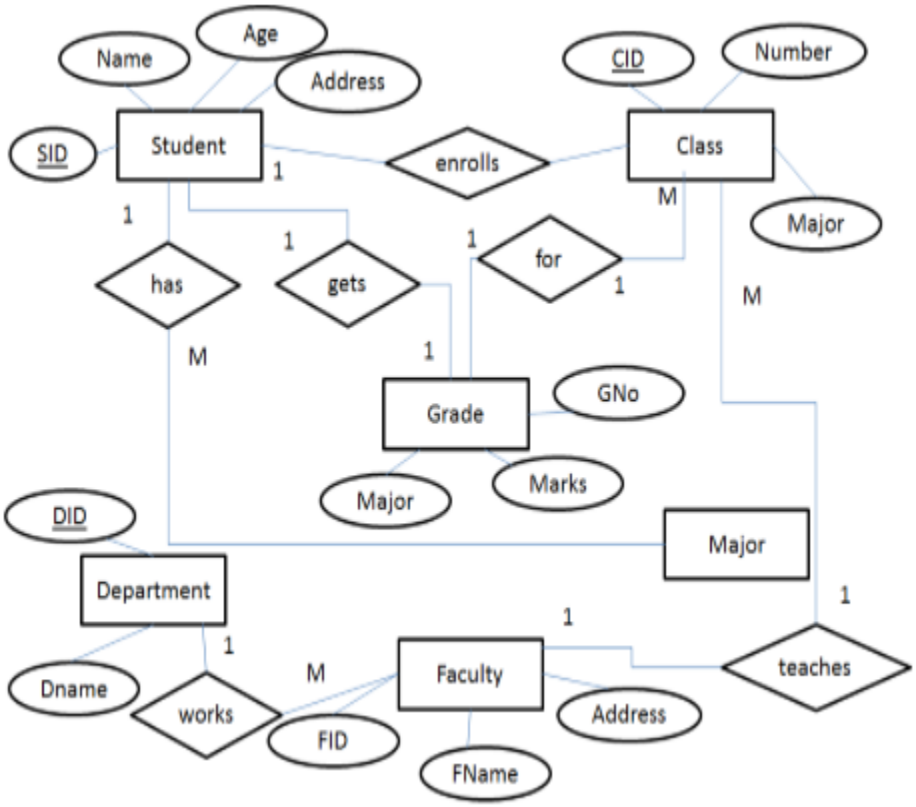
(After this you can add weak/strong entity sets and attributes from Q3 and Q4 in ER)

Draw an ER diagram for university database

ER diagram for university database consisting of four entities

1. Student
2. Department
3. Class
4. Faculty and convert into tables

Diagram:

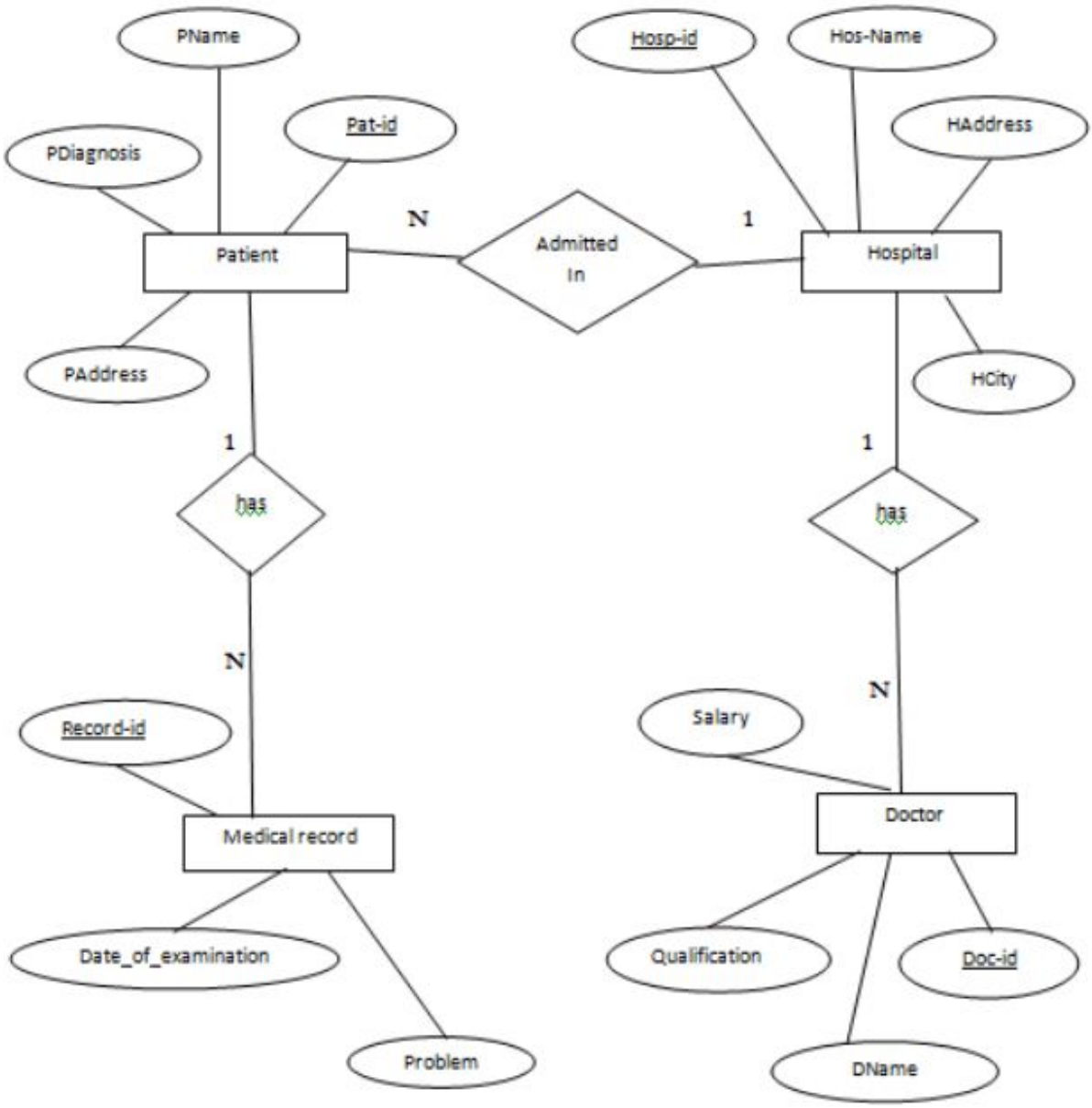


Explanation:

- A student has a unique id and can enroll for multiple classes and has at most one major
- Faculty must belong to department and faculty can take multiple classes
- Every student will get a grade for the class for he/she was enrolled

Draw E-R diagram for Hospital management System, convert them into tables and explain the mapping of attributes

Step 1: E-R Diagram



Step 2: Converting the E-R Diagram into Tables
Converting entity to table and attribute to columns

Hospital

Hosp-id	Primary Key
HCity	
HAddress	
Hos-Name	
Pat-id	Foreign key references to Pat-id of Patient table
Doc-id	Foreign key references to Doc-id of Doctor table

Patient

Pat-id	Primary Key
PName	
PAddress	
PDiagnosis	
Record-id	Foreign key references to Record-id of Medical Record table
Hosp-id	Foreign key references to Hosp-id of Hospital table

Medical Record

Record-id	Primary Key
Problem	
Date_of_examination	
Pat-id	Foreign key references to Pat-id of Patient table

Doctor

Doc-id	Primary Key
DName	
Qualification	
Salary	
Hosp-id	Foreign key references to Hosp-id of Hospital table

Step 3: Mapping of Attributes

Simple Attributes

1. Simple Attributes which can not be divided into subparts.
2. Example: Salary of Doctor

Composite Attributes

1. Composite Attributes which can be divided into subparts.
2. Example: Patient Name, Doctor Name

Step 4: Mapping of Relationships

Foreign Key approach

Hosp_patient

Pat-id	Hospital table makes foreign key references to Pat-id of Patient table
Hosp-id	Patient table makes foreign key references to Hosp-id of Hospital table

Hosp_Doctor

Hosp-id	Doctor table makes foreign key references to Hosp-id of Hospital table
Doc-id	Hospital table makes foreign key references to Doc-id of Doctor table

PatiPPatient_MedicalRecord

Pat-id	Medical Record table makes foreign key references to Pat-id of Patient table
Record-id	Patient table makes foreign key references to Record-id of Medical Record table

Step 5: Identifying the relationships

a. Hospital has a set of patients.
Therefore the relations is 1……..N.

b. Hospital has a set of doctors.
Therefore the relations is 1……..N.

c. Doctor are associated with each patient.
Therefore the relations is N……..1.

d. Each patient has record of various test and examination conducted.
Therefore the relations is 1……..N.

Extended Entity-Relationship:

EER is a high-level data model that incorporates the extensions to the original ER model. Enhanced ERD are high level models that represent the requirements and complexities of complex database.

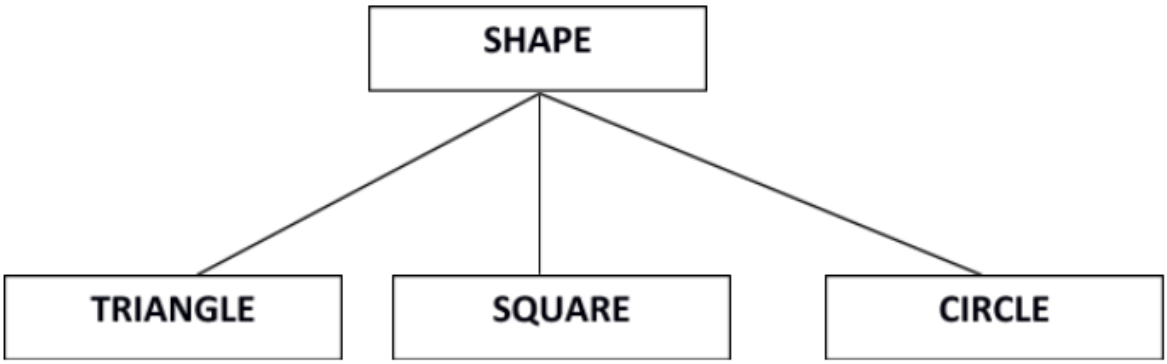
In addition to ER model concepts EE-R includes −

1. Subclasses and Super classes.
2. Specialization and Generalization.
3. Category or union type.
4. Aggregation.

These concepts are used to create EE-R diagrams.

Subclasses and Super class

- Super class is an entity that can be divided into further subtype.
- For example − consider Shape super class.



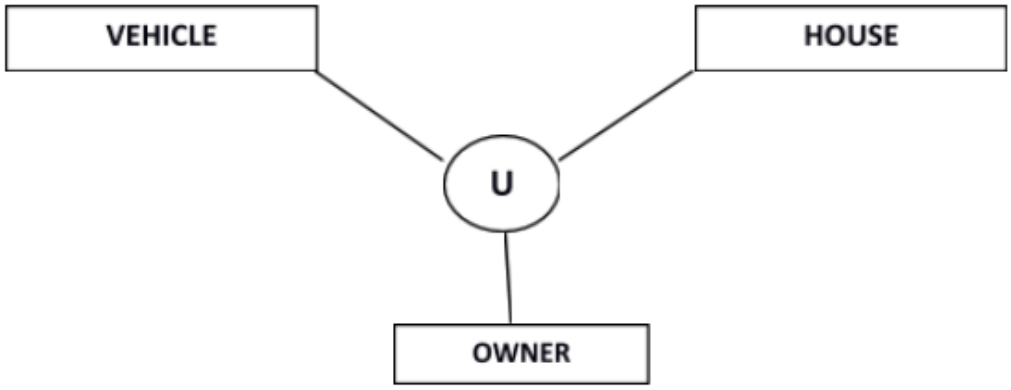
- Super class shape has sub groups: Triangle, Square and Circle.
- Sub classes are the group of entities with some unique attributes.Sub class inherits the properties and attributes from super class.

Specialization and Generalization

Refer Q2

Union

- Relationship of one super or sub class with more than one super class.

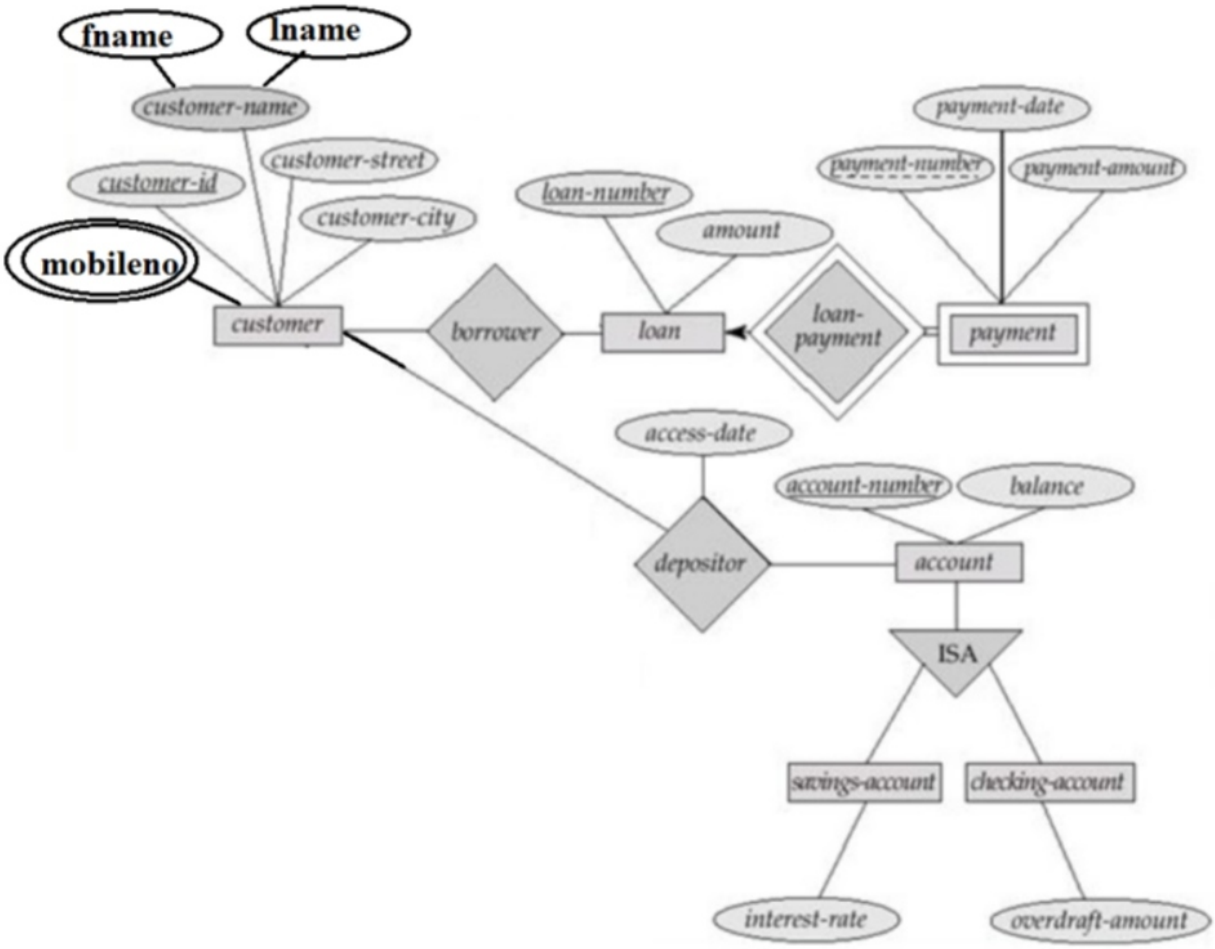


- Owner is the subset of two super class: Vehicle and House.

Aggregation

Refer Q2

Convert the following EER diagram to tables



Conversion of strong entities

- In the given EER diagram CUSTOMER, LOAN, and ACCOUNT are strong entities and form individual tables.

Make all single-valued attributes become a column for the table

- CUSTOMER Table - customer-id, customer-street, and customer-city are single-valued attributes.
- LOAN Table - loan-number and amount are single-valued attributes
- ACCOUNT Table - account-number and balance are single-valued attributes.

The main key attribute of the entity is represented by the primary key

- In the given EER diagram, customer-id, loan-number, and account-number are the primary key attribute for the entities CUSTOMER, LOAN, and ACCOUNT respectively.

The multivalued attribute is represented by a separate table

- In the CUSTOMER table, mobileno is a multivalued attribute.
- Multiple values can not be represented in a single column format of the CUSTOMER table.

- Therefore, create a separate table for the multivalued attribute named as Cust_Mobileno with column names customer_id and monbileno.

Composite attribute represented by components

- In the given CUSTOMER table, customer-name is a composite attribute.
- It contains fname and lname.
- In the CUSTOMER table, these attributes can merge as individual columns.

Conversion of weak entities

- In the given EER diagram PAYMENT is the weak entity type with attributes payment-number, payment-date, and payment-amount.
- Create a separate table with the same name for weak entities and include all their attributes.
- Weak entity PAYMENT holds, payment-number as a weak primary key, and identifying relationship is loan-payment.
- Weak entity table Include the primary key of a strong entity as a foreign key.
- Therefore, loan-number act as a foreign key reference to the loan-number of the LOAN table.

Conversion of Relationship Set

- In the given EER diagram depositor is the relationship set with attribute access-date.
- Create a separate table with the same name for the relationship set and include their attribute.
- The relationship set depositor shows the relationship between the two entities CUSTOMER and ACCOUNT.
- Therefore, the relationship set table depositor also Includes the primary keys of CUSTOMER and ACCOUNT as foreign keys along with attribute access-date.

Representation of ISA Hierarchy or Specialization or Generalization as a table

- In the given EER diagram ACCOUNT table act as a higher entity.
- Whereas, saving-account and checking-account act as lower entities.
- This ISA hierarchy work as an "either" clause which means the account either belongs to the saving-account type or either belongs to the checking-account type.
- This is a specialized approach in the EER diagram.
- To represent it as in table format create tables for both higher and lower level entities and add their local attributes in their respective tables.
- But, must include the primary key of a higher-level entity into lower-level attributes.

Using these rules, we can convert the given EER diagram to tables and columns and assign the mapping relations between the tables. Table structure for the given EER diagram is as below:

Customer Table:

Customer-id	Primary Key
Fname	
Lname	
Customer-street	
Customer-city	

Cust_Mobileno Table for Multivalued attribute of mobileno of CUSTOMER Table

Customer-id	Foreign key references to customer-id of CUSTOMER table.
Mobileno	

Account Table:

Account-number	Primary Key
Balance	

Saving-account table

Account-number	Primary key reference of higher-level entity ACCOUNT table
Interest-rate	

Checking-account table

Account-number	Primary key reference of higher-level entity ACCOUNT table
Overdraft-amount	

Depositor Table (Relationship Set Table)

Customer-id	Foreign key references to customer-id of CUSTOMER table.
Account-number	Foreign key references to account-number of ACCOUNT table.
Access-date	

Loan Table:

Loan-number	Primary Key
Amount	

PAYMENT Table (Weak Entity Table)

Loan-number	Foreign key references to loan-number of LOAN table.
Payment-number	Weak Primary key
Payment-date	
Payment-amount	

Relationships Mapping:

CUSTOMER Table

Loan-number	CUSTOMER table makes foreign key references to loan-number of LOAN table.
Account-number	CUSTOMER table makes foreign key references to account-number of ACCOUNT table.

LOAN Table

Customer-id	LOAN table makes foreign key references to customer-id of CUSTOMER table.
Account-number	LOAN table makes foreign key references to account-number of ACCOUNT table.

ACCOUNT Table

Loan-number	ACCOUNT table makes foreign key references to loan-number of LOAN table.
Customer-id	ACCOUNT table makes foreign key references to customer-id of CUSTOMER table.

Many-To-One Relationship

A loan is associated with several (including 0) customers via the borrower, a customer is associated with at most one loan via the borrower.

One-To-One Relationship

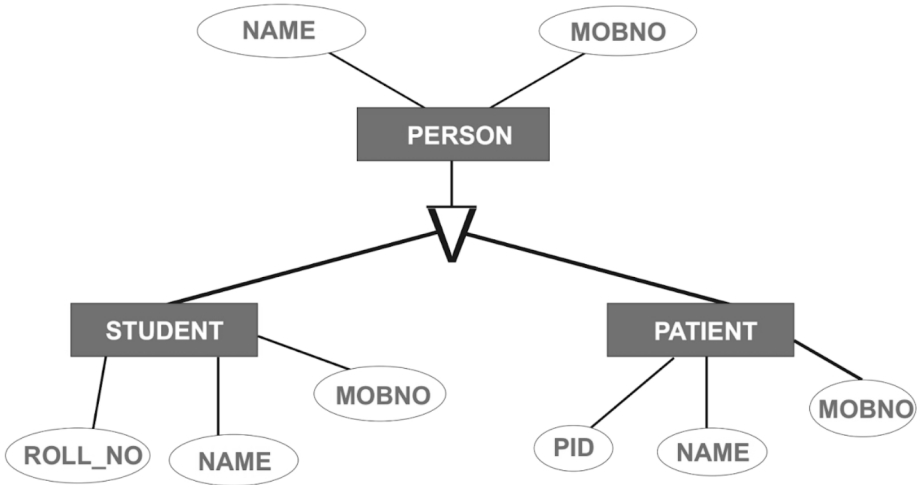
A customer can deposit money to one account, the account is associated with one customer.

2. Generalization, Specialization & Aggregation

Generalization:

1. It is a bottom-up approach in which two lower level entities combine to form higher entity.
2. In generalization, the higher level entity can also combine with other lower level entity to make further higher level entity.
3. Generalization proceeds from the recognition that a number of entity sets share some common features.
4. On the basis of the commonalities, generalization synthesizes these entity sets into a single, higher-level entity set.
5. Generalization is used to emphasize the similarities among lower-level entity sets and to hide the differences in the schema.

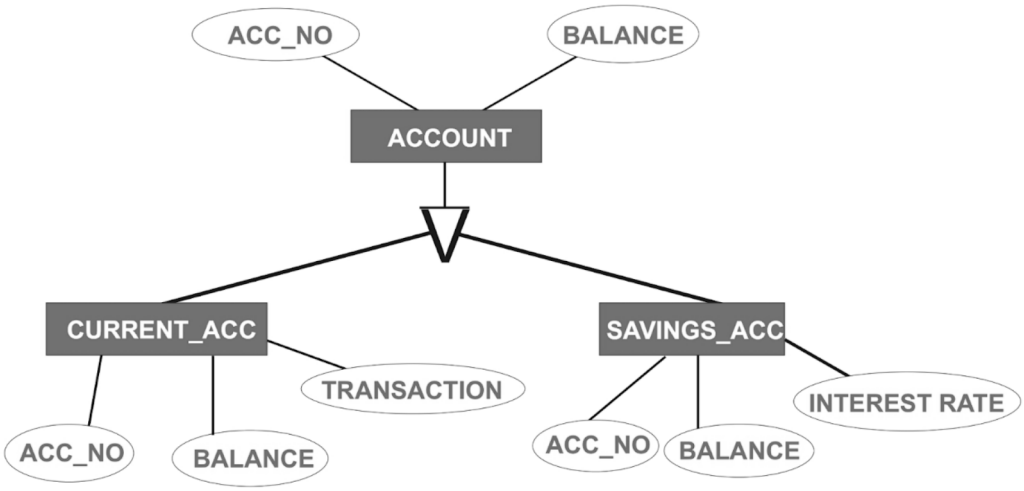
Example:



Specialization:

1. It is opposite to Generalization. It is a top-down approach in which one higher level entity can be broken down into lower level entity.
2. Example: The specialization of account allows us to distinguish among accounts according to whether they are savings or current account.
3. Specialization can be repeatedly applied to refine a design schema.

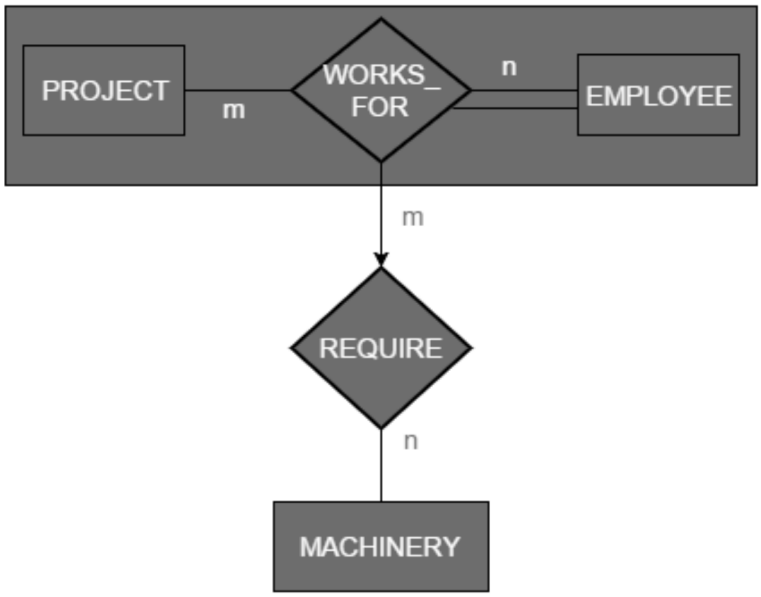
Example:



Aggregation:

1. One limitation of the E-R model is that it cannot express relationships among relationships. To illustrate the need for such a construct, quaternary relationships are used which lead to redundancy in data storage.
2. The best way to mode such situations is to use aggregation.
3. Aggregation is an abstraction through which relationships are treated as higher-level entities.
4. Below is the example of aggregation relation between works-for (which is binary relation between project and employee) and machinery.

Example:

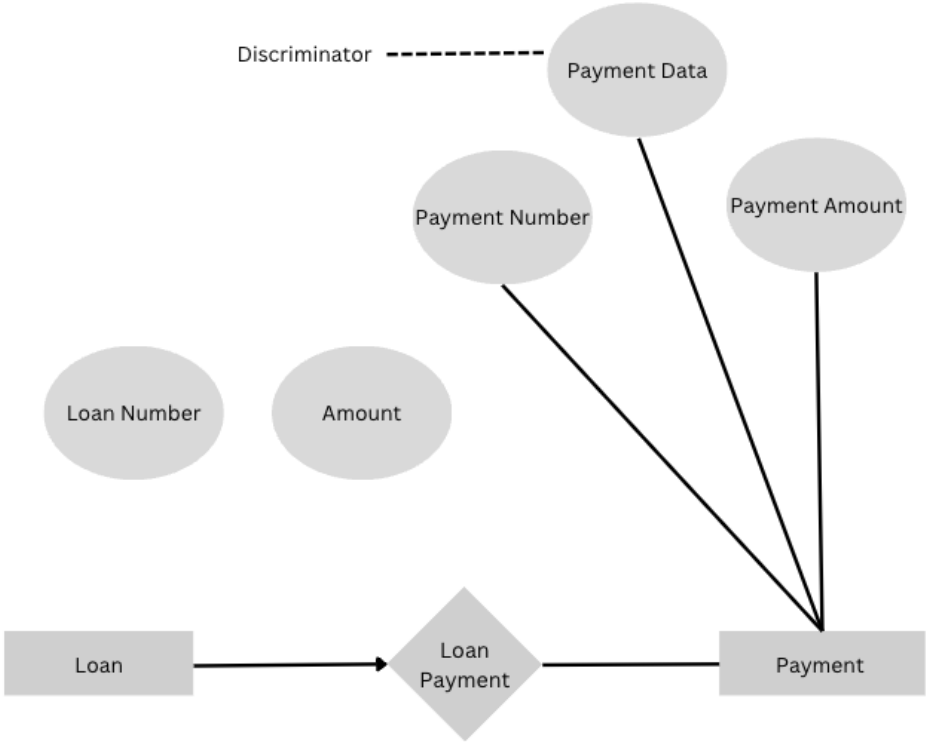


3. Weak & Strong Entity

Strong Entity Set	Weak Entity Set
It has its own Primary Key.	It does not save sufficient attributes to form a primary key on its own.
It is represented by a rectangle.	It is represented by a double rectangle.
It contains a primary key represent by an underline.	It contains a partial key or discriminator represent by a dashed underline.
The number of strong entity set is called dominant entity set.	The number of weak entity set is called subordinate entity set.
The primary key is one of its attributes which unique identifies its member	The primary key of weak entity set is a combination of partial key and primary key of the strong entity set
The relationship between two strong entity set is represent by a diamond symbol	The relationship between one strong and a weak entity set is represent by a double diamond sign. It is known as identifying relationship.
The line connecting strong entity set with the relationship is single.	The line connecting Weak entity set with the relationship is double.
Total participation in the relationship may or may not exist.	Total participation in the identifying relationship always exist.

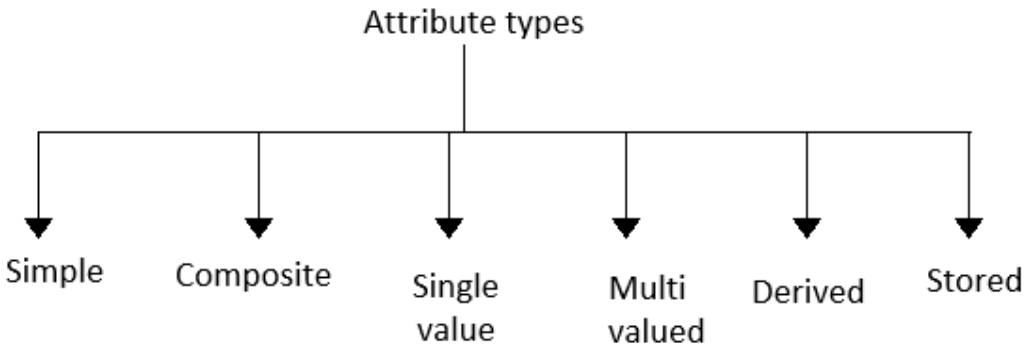
<u>Example:</u>

1. In figure below,, loan-payment is the identifying relationship for payment entity.
2. A weak entity set is represented by doubly outlined box and corresponding identifying relation by a doubly outlined diamond as shown in figure.
3. Here double lines indicate total participation of weak entity in strong entity set it means that every payment must be related via loan-payment to some account.
4. The arrow from loan-payment to loan indicates that each payment is for a single loan.
5. The discriminator of a weak entity set is underlined with dashed lines rather than solid line.



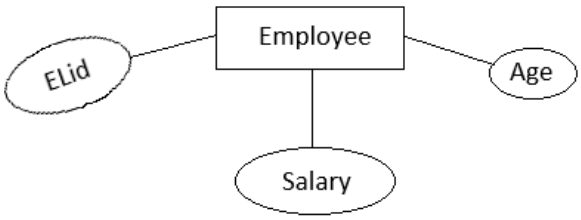
4. Types Of Attributes

<u>Attribute</u> - It can be defined as set of all properties possessed by an entity or entity set.



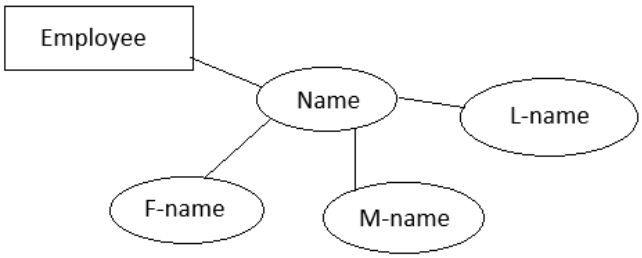
<u>Simple Attribute</u>

- It is composed of single component with individual existence. simple attribute can not be further sub divided.
- example, for entity employee age, salary can be simple attribute.



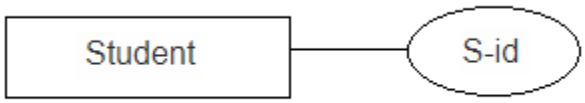
Composite attribute

- An attribute composed of multiples components with independent existence is called as composite attribute.



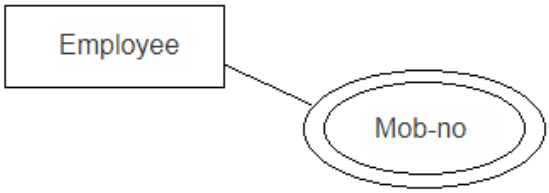
Single value and multi-valued

- An attribute having atomic value for a entity is called as single valued attribute.



every student have one id only.

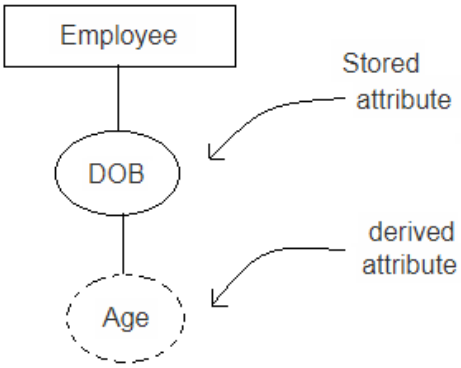
- An attribute having many value, for an entity is called multi-valued attribute.



An employee may have multiple mobile numbers.

Stored and derived attribute

- A simple attribute stored in database is called as stored attribute.
- A value of attribute which can be derived from related stored attribute is called derived attribute.



Null attribute

It takes null values for entities that does not have value for it.

Module 3 - Relational Model & Relational Algebra

1. Steps for ER to relational mapping

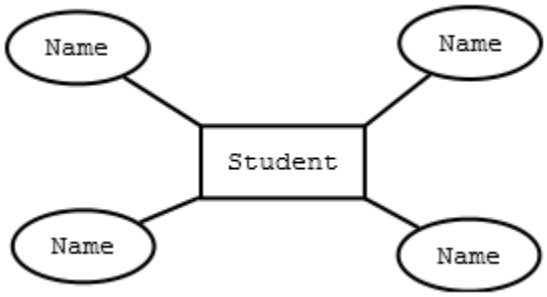
- ER Model when conceptualized into diagrams gives a good overview of entity-relationship, which is easier to understand.
- ER diagrams can be mapped to Relational schema using step by step procedure.
- Though all the ER constraints cannot be imported into Relational model but an approximate schema can be generated.

<u>ER Diagrams mainly comprised of:</u>

1. Entity and its attributes
2. Relationship which is association among entities

<u>Mapping Entity</u>

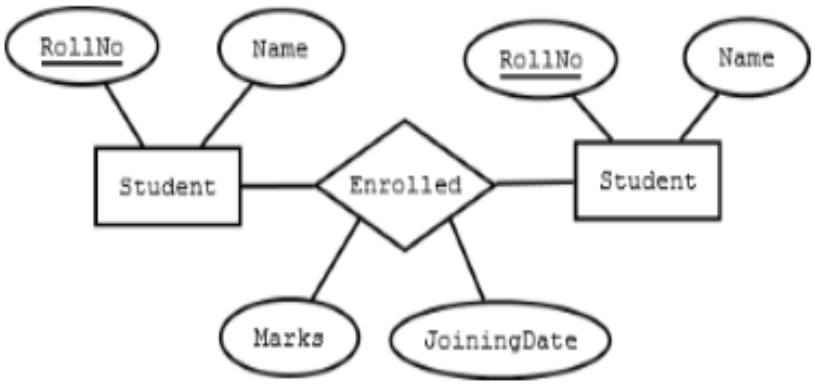
An entity is a real world object with some attributes.



- Create table for each entity
- Entity's attributes should become fields of tables with their respective data types.
- Declare primary key

<u>Mapping relationship</u>

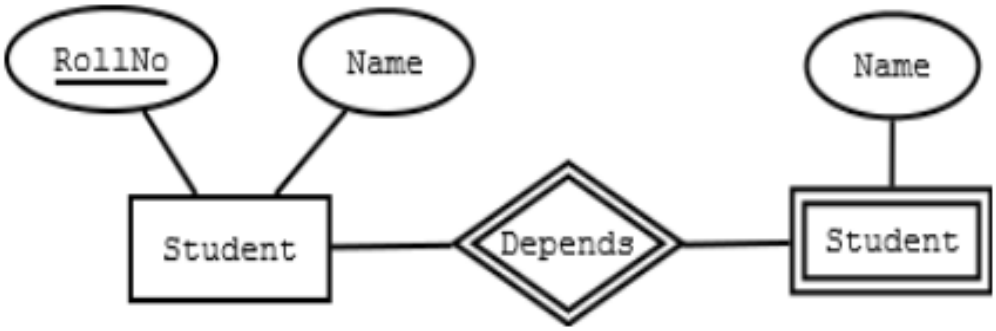
A relationship is association among entities.



- Create table for a relationship
- Add the primary keys of all participating Entities as fields of table with their respective data types.
- If relationship has any attribute, add each attribute as field of table.
- Declare a primary key composing all the primary keys of participating entities.
- Declare all foreign key constraints.

Mapping Weak Entity Sets

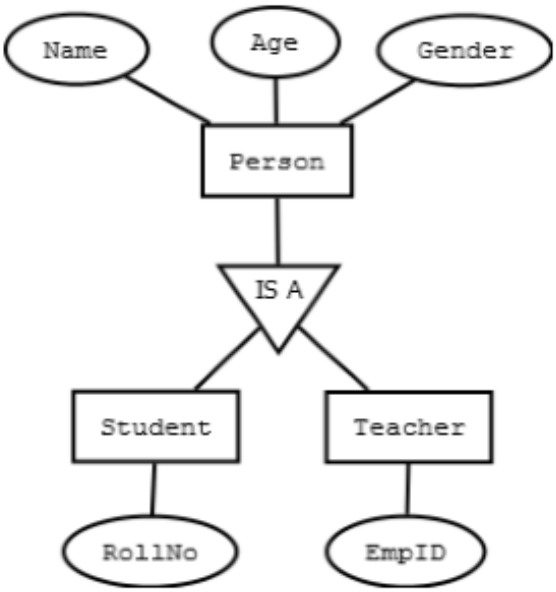
A weak entity sets is one which does not have any primary key associated with it.



- Create table for weak entity set
- Add all its attributes to table as field
- Add the primary key of identifying entity set
- Declare all foreign key constraints

Mapping hierarchical entities

ER specialization or generalization comes in the form of hierarchical entity sets.

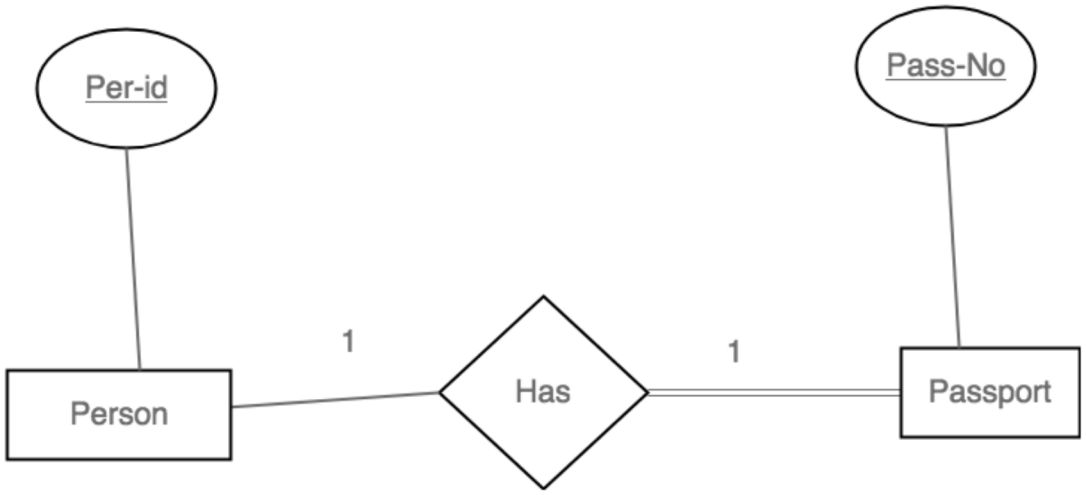


- Create tables for all higher level entities
- Create tables for lower level entities
- Add primary keys of higher level entities in the table of lower level entitiesIn lower level tables, add all other attributes of lower entities.
- Declare primary key of higher level table the primary key for lower level table

Examples:

Case 1: Binary Relationship with 1:1 cardinality with total participation of an entity

A person has 0 or 1 passport number and Passport is always owned by 1 person. So it is 1:1 cardinality with full participation constraint from Passport.



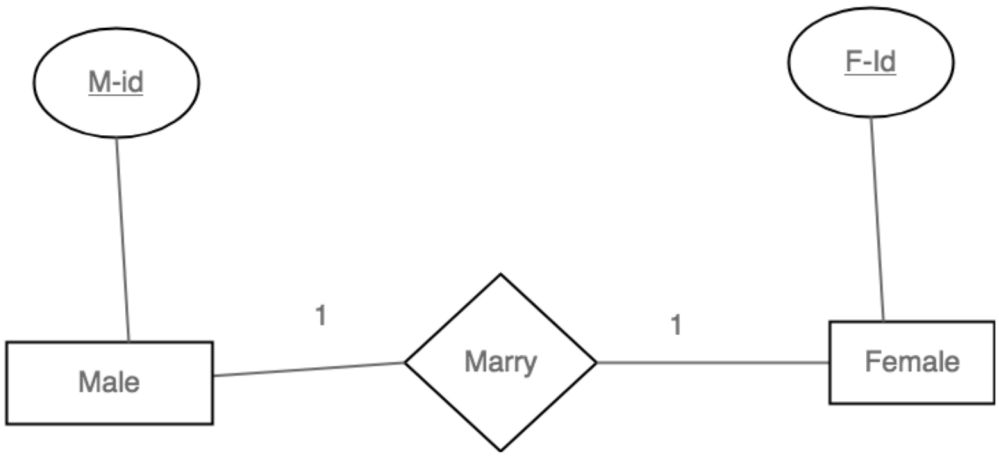
First Convert each entity and relationship to tables. Person table corresponds to Person Entity with key as Per-Id. Similarly Passport table corresponds to Passport Entity with key as Pass-No. Has Table represents relationship between Person and Passport (Which person has which passport). So it will take attribute Per-Id from Person and Pass-No from Passport.

Person		**Has**		**Passport**	
Per-Id	**Other Person Attribute**	**Per-Id**	**Pass-No**	**Pass-No**	**Other PassportAttribute**
PR1	–	PR1	PS1	PS1	–
PR2	–	PR2	PS2	PS2	–
PR3	–				

As we can see from Table 1, each Per-Id and Pass-No has only one entry in Has Table. So we can merge all three tables into 1 with attributes shown in Table 2. Each Per-Id will be unique and not null. So it will be the key. Pass-No can't be key because for some person, it can be NULL.

Per-Id	Other Person Attribute	Pass-No	Other PassportAttribute

Case 2: Binary Relationship with 1:1 cardinality and partial participation of both entities



A male marries 0 or 1 female and vice versa as well. So it is 1:1 cardinality with partial participation constraint from both. First Convert each entity and relationship to tables. Male table corresponds to Male Entity with key as M-Id. Similarly Female table corresponds to Female Entity with key as F-Id. Marry Table represents relationship between Male and Female (Which Male marries which female). So it will take attribute M-Id from Male and F-Id from Female.

Male		**Marry**		**Female**	
M-Id	Other Male Attribute	M-Id	F-Id	F-Id	Other FemaleAttribute
M1	–	M1	F2	F1	–
M2	–	M2	F1	F2	–
M3	–			F3	–

As we can see from Table 3, some males and some females do not marry. If we merge 3 tables into 1, for some M-Id, F-Id will be NULL. So there is no attribute which is always not NULL. So we can't merge all three tables into 1. We can convert into 2 tables. In table 4, M-Id who are married will have F-Id associated. For others, it will be NULL. Table 5 will have information of all females. Primary Keys have been underlined.

M-Id	Other Male Attribute	F-Id

F-Id	Other FemaleAttribute

Note: Binary relationship with 1:1 cardinality will have 2 table if partial participation of both entities in the relationship. If atleast 1 entity has total participation, number of tables required will be 1.

2. Relational Algebra operators

Select:

• The select operator is used to select the rows from a table which satisfy particular selection condition given in selection operation.
• Select operator selects a set of tuples that satisfy a selection condition.
• Output of query is exactly same as input schema of table.
• This is unary relational operator having only one input table.

Syntax:

```
σ<attribute_name><comparison_operator><constant_value>(<input_table_name>)
```

Where,
Attribute_name: Name of the column in table
Comparison_operator: =,<,<=,>,>=,<>

Example:

Eid	EName	Age
1	Nidhi	34
2	Ajit	24
3	Amit	28

Select all the employees having age below 25 years

```
σage<25(Employee)
```

Eid	EName	Age
2	Ajit	24

Project:

• Project operator is used foe selecting some of many columns in table to display in result set.
• Few rows and columns can be selected as per requirement.
• This unary relational operator having only one input table.

Syntax:

```
π<column_list>(<input_table_name>)
```

Example:
Find Employees salary from Employee table and Salary

```
πEid,Salary,Age(Employee)
```

Eid	Salary	Age
1	100000	34
2	12000	24
3	24000	28

<u>Natural Join:</u>

• A Natural join returns all rows by matching values in comman columns having same name and data type of columns should be presentin both sides

<u>Example:</u>

Find all the Employees and their department name

Employee Table

Eid	EName	Did
1	Amit	10
2	Nidhi	30
3	Ajit	50

Department Table

Did	DName
10	IPF
30	HR
70	TIS

```
πeid,dname(Employee ⋈ Department)
```

Eid	DName
1	IPF
2	HR

Set Intersection:

• Set Intersection operator finds out all rows that are common in both result of Query 1 and in the result of Query 2.

Syntax:

```
(Query Expression 1) ∩ (Query Expression 2)
```

Example:

R

A	1
B	2
C	3
D	4
E	5

S

A	1
B	2
C	3
D	4

R ∩S

A	1
B	2
C	3
D	4

<u>Examples:</u>
All Employees in IT Department under Mumbai University

IT_Employee

Eid	EName	Age
11	Suhas	24
12	Jayendra	24
13	Sachin	25
14	Mahesh	23

All Employess in ABC College

Eid	EName	Age
11	Suhas	24
12	Jayendra	24
23	Geeta	25
24	Amruta	23
35	Sangeeta	21

Intersection of the above is employees in IT Department of ABC College

Eid	EName	Age
11	Suhas	24
12	Jayendra	24

Module 4 - Structured Query Language (SQL)

1. SQL Problem Statements

Consider the following employee database

Employee (emp_name, street, city, date_of_joining)
Works(emp_name, company_name, salary)
Company (company_name, city)
Manager (emp_name, manager_name)

Modify the database so that “Deepa” lives in “Pune”

```
UPDATE Employee

SET city='Pune'

Where emp_name='Deepa'
```

Give all employees of “XYZ corporation” a 10% rise in salary

```
UPDATE Works

SET salary=salary*1.10

WHERE company_name='XYZ corporation'
```

List all employee who lives in the same city as their company city

```
SELECT emp_nameFROM Employee e, Company c, Works w

WHERE e.empname = w.empname

&&w.company_name = c.company_name

&&e.city = c.city
```

Display all employees who joined in the month of “March”

```
SELECT emp_nameFROM Employee

WHERE date_of_joining ='MARCH'
```

Find all employees who earn more than average salary of all the employees of their company

```
SELECT emp_name, salaryFROM Employee.e, Works.wONe.emp_name=w.emp_name

WHERE w.salary>avg(w.salary)
```

Consider the following relation for book club

Members (Member_Id,Name,Designation,Age)
Books (Book_Id,BookAuthor,BookTitle,BookPublisher,BookPrice)
Reserves (Member_Id,Book_Id,Date)

Write the sql queries for the following statements:

List the names of members who are professors older than 50 years

```
SELECT Name FROM Members

WHERE Designation='Professor'

AND Age>50
```

List the titles of book reserved by professors

```
SELECT BookTitle

FROM Members M, Books B, Reserves R

WHERE M.Member_Id = R.Member_Id &&

M.Designation = 'Professor'&&

B.Book_Id = R.Book_Id
```

Find the Id's of members who have reserved books which cost more than Rs.200

```
SELECT Member_Id

FROM Books B, Reserves R

WHERE B.Book_Id = R.Book_Id &&

BookPrice > 200
```

Find the authors and titles of book reserved on '20-9-2012'

```
SELECT BookAuthor, BookTitle

FROM Books B, Reserves R

WHERE B.Book_Id = R.Book_Id &&

R.Date = '20-9-2012'
```

2. Keys in SQL: Primary, Foreign, Super, Candidate Key

A key allows us to identify a set of attributes that suffice to distinguish entities from each other. Keys also help uniquely identify relationships, and thus distinguish relationships from each other because no two entities in an entity set are allowed to have exactly the same value for all attributes.

Super key

1. A super key is a set of one or more attributes that allow us to identify an entity uniquely in the entity set.
2. For example, the employee-id attribute of the entity set Employee is sufficient enough to distinguish one employee from another. Thus, employee-id is a super key.
3. Similarly, the combination of employee-name and employee-id is a super key for the entity set employee.
4. However, the employee-name attribute of employee is not a super key, because several people in an organization might have the same name.

Candidate key

1. Candidate key is nothing but minimal super keys for which no proper subset is a super key.
2. Several distinct sets of attributes could serve as a candidate key.
3. Let's assume that a combination of employee-name and employee-address is sufficient to distinguish among members of the employee entity set.

4. Then, both {employee-id} and {employee-name, employee-address} are candidate keys.
5. Although the attributes employee-id and employee-name together can distinguish employee entities, their combination {employee-id, employee-address} does not form a candidate key, since the attribute employee-id alone is a candidate key.

Primary key

1. The term primary key is used to denote a candidate key that is chosen by database designer as principal means of identifying entities within an entity set.
2. Primary key entity in the set cannot have the same value for two or more tuples i.e. unique and it cannot be null.
3. Example: Aadhar number of an Indian person or Social security number of US resident would qualify for a good primary key.
4. However, since not all countries do not have a unique number associated with every individual, international enterprise must generate their own unique identifiers. An alternative is to use some unique combination of other attributes as a key.
5. The primary key should be chosen such that its attributes are never, or very rarely, changed. For instance, the address field of a person should not be part of the primary key, since it is likely to change. Social-security numbers, on the other hand, are guaranteed to never change.

Foreign key

1. The concept of foreign key is used to establish referential integrity between two relations and maintain consistency among tuples in the two relations.
2. The referential integrity constraint simply states that a tuple in one relation that refers to another relation must refer to an existing tuple in that relation. By default, foreign key references the primary key attributes of the referenced table.
3. Foreign keys can be specified as parts of the SQL CREATE table statement by using the FOREIGN key clause. SQL also supports a version of the references clause where a list of attributes of the referenced relation can be specified explicitly. The specified list of attributes must be declared as a candidate key of the referenced relation.

Following tables illustrates various keys as follows:
Super key: {Emp_id, Emp_name}
Candidate key: Emp_id OR Emp_AadharNo
Primary key: Emp_id (Note: Emp_AadharNo qualifies for primary key too but choice is of database designer)
Foreign key: Emp_id in table Salary referenced from table Employee

Emp_id	Emp_name	Emp_AadharNo
EMP001	Jack	INDA203487
EMP002	Jill	INDA204234
EMP003	Hill	INDA983921

Emp_id	Emp_salary
EMP001	11000
EMP002	9000
EMP003	25000

3. SQL Joins

Natural Join

- A natural join returns all rows by matching values in comman columns having same name and data types of columns and that column should be present in both tables.
- Natural join eliminates duplicate columns present in JOIN table. Therefore comman column will be printed only once in resultant table.

Syntax:

```
SELECT Column_List

FROM Table1

NATURAL JOIN

Table 2
```

Employee Table

Eid	EName	Did
1	Mahesh	100
2	Suhas	200
3	Jayendra	300

Department Table

Did	Dname
100	HR
200	TIS

```
SELECT E.Eid[Eid],

D.Did[Did]
D.Dname[Dname]

FROM Employee E

NATURAL JOIN

Department D
```

Eid	Did	DName
1	100	HR
2	200	TIS
3	300	HR

Cartesian Product/CROSS JOIN

- CROSS join occur due to WHERE condition is missing in query or some invalid operations in where clause leads to undesired results or CROSS Join.

Syntax:

```
SELECT Column_List

FROM Table1

CROSS JOIN

Table 2
```

Employee Table

Eid	EName	Did
1	Mahesh	100
2	Suhas	200
3	Jayendra	300

Department Table

Did	DName
100	HR
200	TIS

```
SELECT e.Eid[Eid],

e.Ename[Ename]

e.Did[Did]

d.Did[Did]

d.Dname[Dname]

FROM Employee e

CROSS JOIN

Department d
```

Eid	EName	Did	Did	Dname
1	Mahesh	100	100	HR
1	Mahesh	100	200	TIS
2	Suhas	200	100	HR
2	Suhas	200	200	TIS
3	Jayendra	100	100	HR
3	Jayendra	100	200	TIS

Self Join

- Any table can be joined by itself as long as aeachtable reference is given different name using table alias.

Example:

Employee Table (E)

Eid	Ename	Mid
1	Mahesh	2
2	Suhas	3
3	Jayendra	3

Manager Table(M)

Eid	Ename	Mid
1	Mahesh	2
2	Suhas	3
3	Jayendra	3

```
SELECT E.Ename[Employee],
M.Ename[Manager]
FROM Employee E
CROSS
EmployeeM
ON E.Mid=M.Eid
```

Employee	Manager
Mahesh	Suhas
Suhas	Jayendra
Jayendra	Jayendra

Inner Join

- Inner Join joins two table when there is atleast one match between two tables.

Syntax:

```
SELECT Column_List

FROM Table1

INNER JOIN

Table 2

ON (JOIN_Condition)
```

Employee Table

Eid	EName	Did
1	Mahesh	100
2	Suhas	200
3	Jayendra	300

Department Table

Did	DName
50	DEV
100	HR
200	TIS

```
SELECT E.Eid[Eid],

D.Did[Did]

D.Dname[Dname]

FROM Employee E

INNER JOIN

Department D

ON E.Did=D.DeptId
```

Eid	Did	Dname
1	100	HR
2	200	TIS
3	300	HR

Outer Join

- The join that can be used to see rows of tables that do not meet the join condition along with rows that satisfies join condition is called as Outer Join.

Left Outer Join:

- A left outer join returns all the rows for which the join condition is true and returns all other rows from the dominant table and displays the corresponding values from the subordinate table as NULL.

Employee Table

Eid	EName	Did
1	Mahesh	100
2	Suhas	200
3	Jayendra	500

Department Table

Did	DName
50	DEV
100	HR
200	TIS

```
SELECT E.Eid[Eid],
D.Did[Did]
D.Dname[Dname]
```

```
FROM Employee E
LEFT OUTER JOIN
Department D
ON E.Did=D.DeptId
```

Eid	Did	Dname
1	100	HR
2	200	TIS
3	500	NULL

<u>Right Outer Join:</u>

- A right outer join returns all the rows for which the join condition is true and returns all other rows from the dominant table and displays the corresponding values from the subordinate table as NULL.

Employee Table

Eid	EName	Did
1	Mahesh	100
2	Suhas	200
3	Jayendra	500

Department Table

Did	DName
50	DEV
100	HR
200	TIS

```
SELECT E.Eid[Eid],

D.Did[Did]

D.Dname[Dname]

FROM Employee E

RIGHT OUTER JOIN

Department D

ON E.Did=D.DeptId
```

Eid	Did	Dname
1	100	HR
2	200	TIS
NULL	50	DEV

<u>Full Outer Join:</u>

- A full outer join returns all the rows for which the join condition is true and returns
    - All other rows from the right table and displays the corresponding values from the left table as NULL.
    - All other rows from the left table and displays the corresponding values from the right table as NULL.

Employee Table

Eid	EName	Did
1	Mahesh	100
2	Suhas	200
3	Jayendra	500

Department Table

Did	DName
50	DEV
100	HR
200	TIS

```
SELECT E.Eid[Eid],

E.Did[Did],

D.Did[Did],

D.Dname[Dname]

FROM Employee E

FULL OUTER JOIN

Department D

ON E.Did=D.DeptId
```

Eid	Did	Dname
1	100	HR
2	200	TIS
3	500	NULL
NULL	50	DEV

4. Triggers in SQL

1. A trigger is a block of code that is executed automatically from a database statement.
2. Triggers is generally executed for DML statements such as INSERT, UPDATE or DELETE.
3. It resides in a database code and is fired automatically when the database code requires to perform the INSERT ,UPDATE or DELETE statement.
4. It is also stored in the database as stored procedures. It does not involve any COMMIT transaction rather it is a part of a transaction.

Syntax:

```
CREATE OR REPLACE TRIGGER <trigger-name>
[BEFORE/AFTER]
[INSERT/UPDATE/DELETE]
OF<column-name>
ON<table-name>
[REFERENCING OLD AS O NEW AS N]
[FOR EACH ROW]
WHEN <trigger-condition>
DECLARE
BEGIN
<sql-statement>
END;
```

CREATE OR REPLACE TRIGGER <trigger-name>:
It is used to create a new trigger or replace an existing trigger.

[BEFORE/AFTER]:
It is used to mention the execution time of the trigger. It specifies whether the trigger should fire after or before the DML statement.

[INSERT/UPDATE/DELETE]:
Triggers can be fired against any of these DML statements.

OF<column-name>:
It is used to indicate the column name on which the trigger will be fired but it used for only update statement.

ON<table-name>
It is used to indicate on which table the trigger will be fired

[REFERENCING OLD AS O NEW AS N]:
This is option and used to define the aliases.

[FOR EACH ROW]:
It is used to indicate if the trigger is a row-level trigger or a statement-level trigger.

WHEN <trigger-condition>:
It is used to indicate the condition for trigger execution.

DECLARE:
IT is used for declaring the variable.

BEGIN……END:
In this actual query for trigger is written.

Example:1

```
Create Trigger ABC

Before Insert On Students
```

This trigger is activated when an insert statement is issued, but before the new record is inserted

Example:2

```
Create Trigger XYZ

After Update On Students
```

This trigger is activated when an update statement is issued and after the update is executed
Let us see the one more example to understand the trigger

Example:3

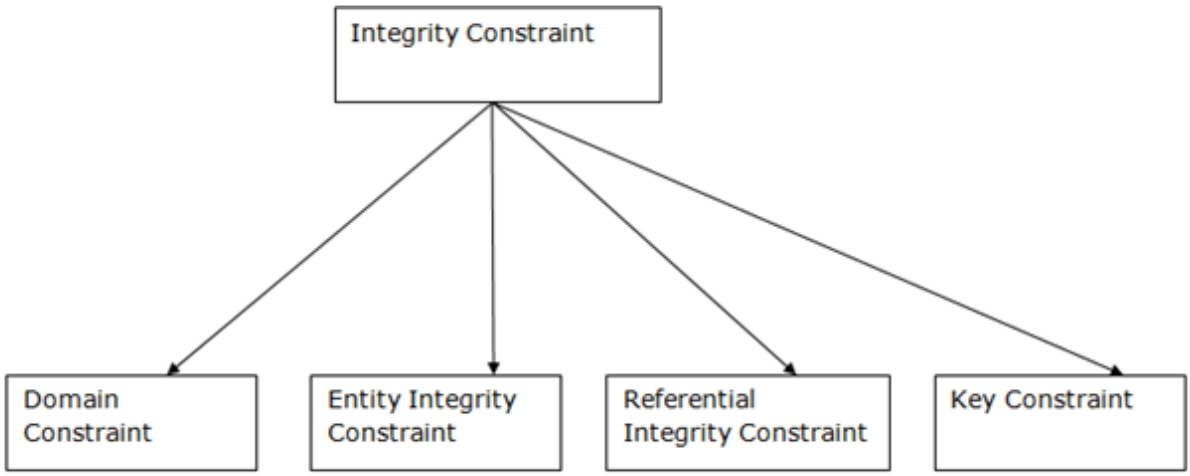
```
CREATE TRIGGER Amount-CHECK BEFORE UPDATE ON account

FOR EACH ROW

BEGIN

IF (NEW.amount < 0) THEN

SET NEW.amount = 0;

ELSEIF (NEW.amount > 100) THEN
```

```
SET NEW.amount = 100;

END IF;

END;
```

In this the trigger will first check the amount if it is less than zero it will set amount as zero but if it is greater than hundred then it will set amount as hundred

5. Integrity Constraints

1. Integrity constraints are a set of rules. It is used to maintain the quality of information.
2. Integrity constraints ensure that the data insertion, updating, and other processes have to be performed in such a way that data integrity is not affected.
3. Thus, integrity constraint is used to guard against accidental damage to the database.



1. Domain constraints

- Domain constraints can be defined as the definition of a valid set of values for an attribute.
- The data type of domain includes string, character, integer, time, date, currency, etc. The value of the attribute must be available in the corresponding domain.

Example:

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

Not allowed. Because AGE is an integer attribute

2. Entity integrity constraints

- The entity integrity constraint states that primary key value can't be null.
- This is because the primary key value is used to identify individual rows in relation and if the primary key has a null value, then we can't identify those rows.
- A table can contain a null value other than the primary key field.

Example:

EMPLOYEE

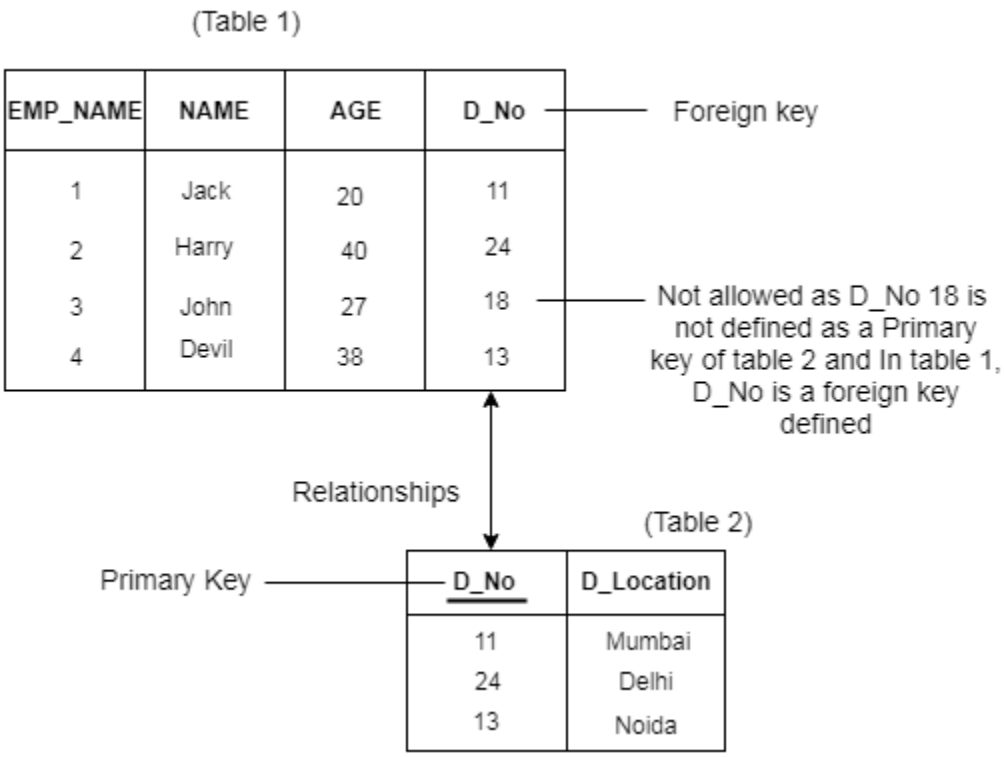
EMP_ID	EMP_NAME	SALARY
123	Jack	30000
142	Harry	60000
164	John	20000
	Jackson	27000

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

3. Referential Integrity Constraints

- A referential integrity constraint is specified between two tables.
- In the Referential integrity constraints, if a foreign key in Table 1 refers to the Primary Key of Table 2, then every value of the Foreign Key in Table 1 must be null or be available in Table 2.

Example:



4. Key constraints

- Keys are the entity set that is used to identify an entity within its entity set uniquely.
- An entity set can have multiple keys, but out of which one key will be the primary key. A primary key can contain a unique and null value in the relational table.

Example:

ID	NAME	SEMENSTER	AGE
1000	Tom	1st	17
1001	Johnson	2nd	24
1002	Leonardo	5th	21
1003	Kate	3rd	19
1002	Morgan	8th	22

Not allowed. Because all row must be unique

Module 5 - Relational Database Design

1. Explain normalization. Also explain 1NF, 2NF, 3NF, BCNF

- Database normalization is a technique of organizing the data in the database.
- Normalization of data can be considered a process of analysing the given relation schemas based on their Functional Dependencies and primary keys to achieve the following properties:
    - Minimizing redundancy
    - Minimizing the insertion, deletion, and update anomalies
    - Ensuring data is stored in correct table
- It can be considered as a filtering process to make the design have successively better quality
- It is a multi-step process that puts data into tabular form by removing duplicated data from the relation tables.
- Without normalization it becomes difficult to handle and update database without facing data loss.

The various forms of normalization are described below:

First Normal Form (1NF):

1. First normal form (1NF) states that the domain of an attribute must include only atomic values and that the value of any attribute in a tuple must be a single value from the domain of that attribute.
2. A relation is said to be in 1NF if it contains no non-atomic values and each row can provide a unique combination of values.
3. 1NF disallows having a set of values, a tuple of values, or a combination of both as an attribute value for a single tuple.
4. The only attribute values permitted by 1NF are single atomic values.

Example:
Un-Normalized Table

Student	Age	Subject
Rooney	15	Java, C++
Kane	16	HTML, PHP

Normalized Table: Any Row must not have a column in which more than one value is saved, instead data is separated in multiple rows as shown below.

Student	Age	Subject
Rooney	15	JAVA
Rooney	15	C++
Kane	16	HTML
Kane	16	PHP

<u>Second Normal Form (2NF):</u>

1. A relation is said to be in 2NF, if it is already in 1NF and each and every attribute fully depends on the primary key of the relation.
2. There must not be any partial dependency of any column on the primary key.
3. Second normal form (2NF) is based on the concept of full functional dependency. A functional dependency X -> Y is a full functional dependency if removal of any attribute A from X means that the dependency does not hold any more.
4. A functional dependency X->Y is a partial dependency if some attribute A belongs to X can be removed from X and the dependency still holds.

<u>Example:</u>

Student_Project Table

Stud_ID	Proj_ID	Stud_Name	Proj_Name
100	001	Rooney	Cloud
200	002	Kane	Servers

Stud_Name depends on Stud_ID and Proj_Name depends on Proj_ID
The above table can be normalized to 2NF as shown below.

Student Table in 2NF

Stud_ID	Proj_ID	Stud_Name
100	001	Rooney
200	001	Kane

Project Table in 2NF

Proj_ID	Proj_Name
001	001
002	Servers

<u>Third Normal Form (3NF):</u>

- A relation is said to be in 3NF, if it is already in 2NF and there exists no transitive dependency in that relation.
- If a table contains transitive dependency, then it is not in 3NF, and the table must be split to bring it into 3NF.

<u>What is a transitive dependency?</u>
A -> B [B depends on A] & B -> C [C depends on B]
Then A -> C[C depends on A] can be derived.

<u>Example:</u>
Below table not in 3NF

Stud_ID	Stud_Name	City	Zip
100	Rooney	Manchester	4001
200	Kane	Stoke	4002

Stud_ID is the only prime key attribute. City can be identified by Stu_ID as well as Zip. Neither Zip is a superkey nor City is a prime attribute.
Stud_ID -> Zip -> City, so there exists transitive dependency. Hence 3NF table is below

Student_Detail

Stud_ID	Stud_Name	Zip
100	Rooney	4001
200	Kane	4002

Zip_Code

Zip	City
4001	Manchester
4002	Stoke

Boyce-Codd Normal Form (BCNF):

- BCNF is an extension of Third Normal Form in strict way.
- A relationship is said to be in BCNF if it is already in 3NF and for any non-trivial functional dependency, X -> A, then X must be a super-key.
- A 3NF table which does not have multiple overlapping candidate keys is said to be in BCNF.

Example:
In the 3NF example,Stud_ID is super-key in Student_Detail relation and Zip is super-key in ZipCodes relation.

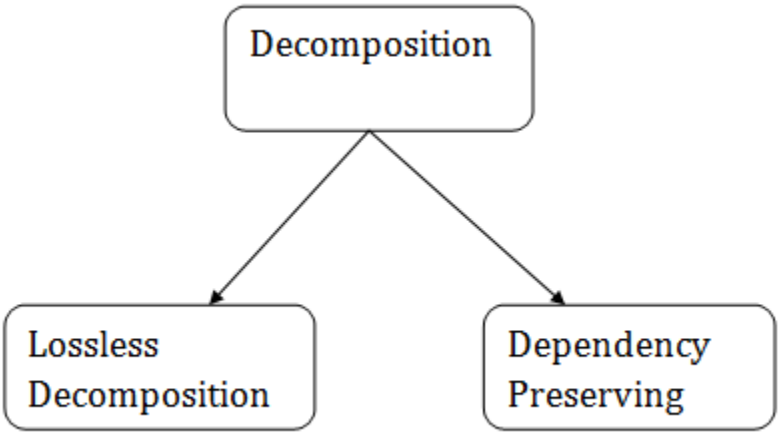
So Stud_ID ->Stud_Name, Zip and

Zip ->City

Confirms, that both relations are in BCNF.

2. Lossless Decomposition

- When a relation in the relational model is not in appropriate normal form then the decomposition of a relation is required.
- In a database, it breaks the table into multiple tables.
- If the relation has no proper decomposition, then it may lead to problems like loss of information.
- Decomposition is used to eliminate some of the problems of bad design like anomalies, inconsistencies, and redundancy.



Lossless Decomposition

1. If the information is not lost from the relation that is decomposed, then the decomposition will be lossless.
2. The lossless decomposition guarantees that the join of relations will result in the same relation as it was decomposed.

3. The relation is said to be lossless decomposition if natural joins of all the decomposition give the original relation.

Example:

EMPLOYEE_DEPARTMENT table:

EMP_ID	EMP_NAME	EMP_AGE	EMP_CITY	DEPT_ID	DEPT_NAME
22	Denim	28	Mumbai	827	Sales
33	Alina	25	Delhi	438	Marketing
46	Stephan	30	Bangalore	869	Finance
52	Katherine	36	Mumbai	575	Production
60	Jack	40	Noida	678	Testing

The above relation is decomposed into two relations EMPLOYEE and DEPARTMENT

EMPLOYEE table:

EMP_ID	EMP_NAME	EMP_AGE	EMP_CITY
22	Denim	28	Mumbai
33	Alina	25	Delhi
46	Stephan	30	Bangalore
52	Katherine	36	Mumbai
60	Jack	40	Noida

DEPARTMENT table

DEPT_ID	EMP_ID	DEPT_NAME
827	22	Sales
438	33	Marketing
869	46	Finance
575	52	Production
678	60	Testing

Now, when these two relations are joined on the common column "EMP_ID", then the resultant relation will look like:

Employee ⋈ Department

EMP_ID	EMP_NAME	EMP_AGE	EMP_CITY	DEPT_ID	DEPT_NAME
22	Denim	28	Mumbai	827	Sales
33	Alina	25	Delhi	438	Marketing
46	Stephan	30	Bangalore	869	Finance
52	Katherine	36	Mumbai	575	Production
60	Jack	40	Noida	678	Testing

Hence, the decomposition is Lossless join decomposition.

Dependency Preserving

1. It is an important constraint of the database.
2. In the dependency preservation, at least one decomposed table must satisfy every dependency.
3. If a relation R is decomposed into relation R1 and R2, then the dependencies of R either must be a part of R1 or R2 or must be derivable from the combination of functional dependencies of R1 and R2.
4. For example, suppose there is a relation R (A, B, C, D) with functional dependency set (A->BC). The relational R is decomposed into R1(ABC) and R2(AD) which is dependency preserving because FD A->BC is a part of relation R1(ABC).

Module 6 - Transactions Management and Concurrency and Recovery

1. Deadlock (prevention, handling, recovery)

- The computer system uses may types of resource which are then used by various processes to carry out their individual functions.
- But problem is that the amount of resources available is limited and many process needs to use it.
- A set of process is said to be in a deadlocked state when every process in the set is waiting for an event that can be caused only by another process in the set.
- The event can be resource acquisition, resource release etc. The resource can be physical (printers, memory space) or logical (semaphores, files)

The necessary and sufficient conditions for deadlock to occur are:

Mutual Exclusion

1. A resource at a time can only be used by one process.
2. If another process is requesting for the same resource, then it must be delayed until that resource is released.

Hold and Wait

1. A process is holding a resource and waiting to acquire additional resources that are currently being held by other processes.

No Pre-emption

1. Resources cannot be pre-empted
2. Resource can be released only by the process currently holding it based on its voluntary decision after completing the task

Circular wait

1. A set of processes { P0,P1,….,Pn-1,Pn } such that the process P0 is waiting for resource held by P1,P1 is waiting for P2 ,and Pn is waiting for P0 to release its resources.
2. Every process holds a resource needed by the next process.
3. All the four above mentioned conditions should occur for a deadlock to occur.

The three major approaches for handling deadlocks are as follows:

Deadlock Prevention:

1. That the system never enters a deadlock state.
2. It provides a set of methods to make sure that at least one of the four necessary conditions for a deadlock is never satisfied.
3. Mutual Exclusion this condition is needed to be checked for non-sharable resources (e.g. Printer)

4. Hold and Wait - It requires a process to request a resource and get allocated before execution or allow process to request resources when the process has none.
5. No preemption - If a process that is holding some resources requests another resource that cannot be immediately allocated to it, then all resources currently being held are released
6. Circular Wait- we can impose a total ordering of all resource types, and ask that each process requests resources in an increasing order of enumeration.
7. Disadvantage is that it can lead to low device utilization and reduced system throughput.

Deadlock Avoidance:

1. It is the simplest and most friendly method.
2. It requires that each process declare the maximum number of resources of each type that it will need.
3. The deadlock-avoidance algorithm dynamically checks the resource-allocation state to ensure the system can never be in a circular-wait condition.
4. When a process requests a resource, the system must make sure that the allocation would leave the system in a safe state.
5. It the system is in a safe state, then there would be no deadlock. However, if it is in an unsafe state, that there is a possibility (not certainty) of a deadlock.
6. The avoidance approach requires that knowledge of all processes, all the resources available, the resources allocated presently and the future requests by the processes.
7. For a single instance of a resource type we use the resource allocation graph.
8. For multiple instance of a resource type, we use the banker's algorithm.
9. A major drawback of this method is that it is difficult to know at the beginning itself of the maximum resource required.

Deadlock Detection and Recovery:

1. Here we allow the system to enter into a deadlock state and then try to recover the system back from it
2. This method has two parts:
    i. Detection: an algorithm to check the system state whether in deadlock
    ii. Recovery: initiated when deadlock detected, to recover from that deadlock.
3. This method doesn't require the maximum need MAX or the Need matrix. It works using the Available, Allocation and Request data structures
4. For single instance of a resource type we use the wait-for graphs and search for cycles in the graph.
5. For multiple instances of each resource type, we use the Deadlock-detection algorithm.
6. To recover we have two options
    i. Abort one or more processes
    ii. Pre-empt one or more resources from one or more deadlocked process

2. ACID Properties

Atomicity

1. Atomicity refers to the ability of the database to guarantee that either all of the tasks of a transaction are performed or none of them are. Database modifications must follow an all or nothing rule.
2. Each transaction is said to be atomic if when one part of the transaction fails, the entire transaction fails.
3. There must be no state in database where the 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

1. The consistency property ensures that the database remains in a consistent state before the start of the transaction and after the transaction is over.
2. There must not be any possibility that some data is incorrectly affected by the execution of transaction.
3. If the database was in a consistent state before the execution of the transaction, it must remain in consistent state after the execution of the transaction.

Isolation

1. Isolation refers to the requirement that other operations cannot access or see the data in an intermediate state during a transaction.
2. 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.
3. This constraint is required to maintain the performance as well as the consistency between transactions in a database.
4. Thus, each transaction is unaware of another transactions executing concurrently in the system.

Durability

1. Durability refers to the guarantee that once the user has been notified of success, the transaction will persist, and not be undone.
2. All updates made on the database will persist even if the system fails and restarts. If a transaction writes or updates some data in database and commits that data will always be there in the database.
3. If the transaction commits but data is not written on the disk and the system fails, that data will be updated once.This means it will survive system failure, and that the database system has checked the integrity constraints and won't need to abort the transaction.

3. Lock-Based & Timestamp-Based Concurrency Control

- Concurrency controlling techniques ensure that multiple transactions are executed simultaneously while maintaining the ACID properties of the transactions.
- Simultaneous execution of transactions overashared database can create several data integrity and consistency problems.
    1. Lost Update
    2. Dirty Read Problems
    3. İncorrect Summary Problems
    4. Unrepeatable Read Problem
- The various approaches for concurrency control,
    1. Lock based concurrency control
    2. Timestamp concurrency control

Lock-based Protocols

1. Database systems equipped with lock-based protocols usea mechanism by which any transaction cannot read or write data until it acquires an appropriate lock on it.
2. Locks are of two kinds ,
    a. Binary Locks - A lock onadata item can be in two states; it is either locked or unlocked.
    b. Shared/exclusive - If a lock is acquired onadata item to performawrite operation, it is an exclusive lock. Read locks are shared because no data value is being changed.

Types of lock protocols

Single phase:

1. Lock-based protocols allow transactions to obtainalock on every object beforeaoperation is performed.
2. Transactions may unlock the data item after completing the operation.

Two phase:

1. This locking protocol divides the execution phase ofatransaction into two phase.
2. In the first part, the transaction acquires all the locks (Growing phase).
3. Second phase starts, the transaction cannot demand any new locks; it only releases the each acquired locks lock after use (Shrinking phase).

Strict Two-Phase Locking:

1. The first phase of Strict-2PL is the same as 2PL.
2. After acquiring all the locks in the first phase, the transaction continues to execute normally.
3. But in contrast to 2PL, Strict-2PL does not releasealock after using it.
4. Strict-2PL holds all the exclusive locks until the commit point and releases all the locks at a time.

Rigorous 2PL:

1. The first phase of Rigorous 2PL is the same as 2PL
2. After acquiring all the locks in the first phase, the transaction continues to execute normally
3. In Rigorous 2PL all locks are released only after commit(both shared and exclusive.

Locking rules for nested transactions

1. Every lock that is acquired byasuccessful sub transaction is inherited by its parent when it completes.
2. Parent transactions are not allowed to run concurrently with their child transactions.
3. Sub transactions at the same level are allowed to run concurrently.
4. For a sub transaction to acquiresread lock on an object, no other active transaction can haveawrite lock on that object.
5. For a sub transaction to acquireawrite lock on an object, no other active transaction can havearead or write lock on that object.
6. When a sub transaction commits, its locks are inherited by its parent
7. When a sub transaction aborts, its locks are discarded.

Timestamp-based Protocols

- The basic idea is to order the transaction based on their timestamps. This protocol ensures that any conflicting read and write operation are executed in time stamp order. This protocol works as follow:
- Suppose that transaction Ti issues read (q)

```
If Ts(Ti) < W - time stamp (Q)
```

- Then Ti needs to read a value of Q was already overwritten, hence the read operation is rejected and Ti rolled back. Here co-timestamp (Q) denotes the largest timestamp of any transaction that executed write (Q) successfully.

```
If Ts (Ti) > w - timestamp (2)
```

- Then Ti needs to read the value of Q i.e. read operation is executed and (Q) is set to the maximum of R and Ts (Ti)
- suppose that transaction Ti issues write Q of Ts(Ti) < R – timestamp (q) then the value of Q that Ti is producing was needed previously, and the spectrum assumed that the value will never be produced.
- Hence, the system rejects the write operation and rolls Ti back. If Ts(Ti) < W – write stamp (Q) then Ti is attempting to write an absolute value of Q, hence, the system rejects this write operation and rolls Ti back.
- Otherwise, the system executed the write operation and sits, w- timestamp (Q) to Ts (Ti)

Example:

Suppose there are 2 transactions T1 & T2, T2 displays the content 01 account A and B there as T2 transfer Rs 100 from account A to account B and display sum of both the two transactions T1 & T2 can be illustrated as follow:

```
Ti : read (A)

Read (B)

Display (A + B)

T2 : read (B)

B = B - 100

Write (B)

Read (A)

A = A + 100

Write (A)

Display (A + B)
```

Here, the assumption is that a transaction is assigned to a time stamp immediately before its first instruction. Thus a schedule Ts(t1) < TS(+2)

T1	T2
Read (B)	Read (B) B = B – 100 Write (B)
Read (A)	Read (A)
Display (A + B)	A = A + 100 Write (A) Display (A + B)

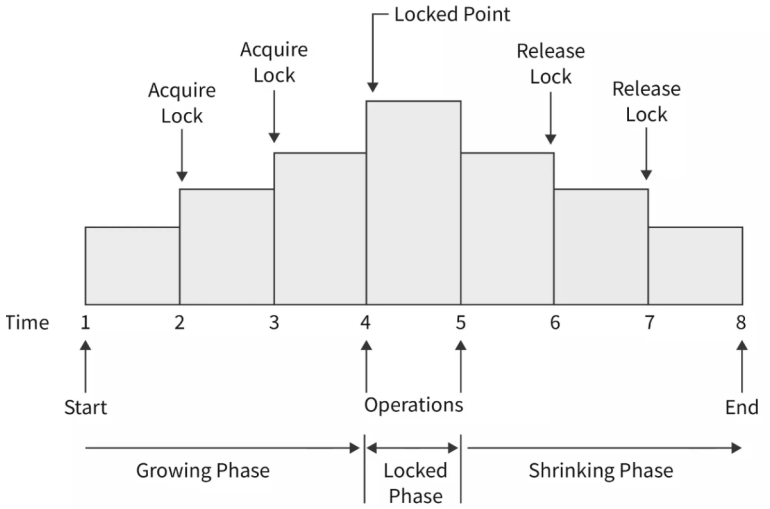
The time stamp adding protocol ensures conflicted serialization. This is because conflicting operations are processed in timestamp ordering. This protocol ensures freedom from dead lock since no transaction ever waits.

4. Two-Phase (2PL) Locking Protocol

The two phases of Locking are :

Growing Phase: In the growing phase, the transaction only obtains the lock. The transaction can not release the lock in the growing phase. Only when the data changes are committed does the transaction start the Shrinking phase.

Shrinking Phase: Neither are locks obtained nor are they released in this phase. When all the data changes are stored, only then the locks are released.



In the above figure, we can see that in the growing phase, all the locks are obtained till the point when all the locks needed by the transactions are acquired. This pint is called Lock-Point. After the lock point, the transaction enters the Shrinking phase.

Two-Phase Locking Types

Two-phase Locking is further classified into two types :

Strict two-phase locking protocol :

1. The transaction can release the shared lock after the lock point.
2. The transaction can not release any exclusive lock until the transaction commits.
3. In strict two-phase locking protocol, if one transaction rollback then the other transaction should also have to roll back.
4. The transactions are dependent on each other. This is called Cascading schedule.

Rigorous two-phase locking protocol :

1. The transaction cannot release either of the locks, i.e., neither shared lock nor exclusive lock.
2. Serializability is guaranteed in a Rigorous two-phase locking protocol.
3. Deadlock is not guaranteed in rigorous two-phase locking protocol.

Cascading Roll Back in 2PL

Cascading rollback means if one transaction rollback then other transactions dependent on it should also roll back.

S no.	T1	T2	T3
1	Lock-X(R1)		
2	Read(R1)		
3	Write(R1)		
4	Lock-S(R2) LP		
5	Read(R2)		
6	Unlock-R1,Unlock-R2		
7		Lock-X(R1) LP	
8		Read(R1)	
9		Write(R1)	
10		Unlock-R1	
11			Lock-S(R1) LP
12			Read-R1

- Because of Write(R1) in T1 and after that, Read(R1) in T2 and Read(R1) in T3, there is a dirty read which means the data written by transaction T1 is being read by transaction T2 and T3.
- So if T1 Rollback, T2 and T3 also have to rollback, causing cascading Rollback. So cascading Rollback is possible in a two-phase locking protocol.
- In the above transaction, LP denotes Lock Point.
- Before the lock point in all three transactions, T1, T2, and T3, there is a growing phase in transactions, respectively, and after LP, the shrinking phase starts.

5. Log Based Recovery

1. The most widely used structure for recording database modification is the log.
2. The log is a sequence of log records, recording all the update activities in the database
3. In short Transaction log is a journal, which contains history of all transaction performed
4. Log contains Start of transaction, trans-id, record-id, type of operation (insert, update, delete), old value, new value, End of transaction that is commit or aborted.

<u>There are several types of log records.</u>

1. Transaction identifier
2. Data-item identifier
3. Old value
4. New value
5. Whenever a transaction performs a write, it is essential that the log record for that write be created before the database is modified.
6. Once a log record exists, we can output the modification that has already been output to the database, by using the old value field in the log records.

<u>Log based recovery works as follows:</u>

1. The log file is kept on stable storage media
2. When a transaction enters the system and starts execution, it writes a log about it
   <Tn, Start>
3. When the transaction modifies an item X, it write logs as follows:
   <Tn, X, V1, V2>
4. It reads Tn has changed the value of X, from V1 to V2.
5. When transaction finishes, it logs:
   <Tn,commit>

<u>Database can be modified using two approaches:</u>

Deferred database modification:
All logs are written on to the stable storage and database is updated when transaction commits.

Immediate database modification:
Each log follows an actual database modification. That is, database is modified immediately after every operation.

6. Conflict / View Serializability

A schedule is serializable if it is equivalent to a serial schedule.

Conflict serializability:

1. Instructions Ii and Ij, of transactions Ti and Tj respectively, conflict if and only if there exists some item P accessed by both Ii and Ij, and atleast one of these instructions wrote P.
2. Consider the below operations-
    a. i. Ii = read(P), Ij = read(P). Ii and Ij don't conflict.
    b. ii. Ii = read(P), Ij = write(P). They conflict.
    c. iii. Ii = write(P), Ij = read(P). They conflict.
    d. iv. Ii = write(P), Ij = write(P). They conflict.
3. A conflict between Ii and Ij forces a temporal order between them.
4. If Ii and Ij are consecutive in a schedule and they do not conflict, their results would remain the same even if they had been interchanged in the schedule.
5. If a schedule Scan be transformed in to a schedule S by a series of swaps of non-conflicting instructions, then S and S` are conflict equivalent.
6. In other words a schedule S is conflict serializable if it is conflict equivalent to a serial schedule.
7. Example of a schedule that is not conflict serializable:

T3	T4
Read(P)	
	Write(P)
Write(P)	

8. The instructions cannot be swapped in the above schedule to obtain either the serial schedule < T 3 , T 4 >, or the serial schedule < T 4 ,T 3 >.
9. A serial schedule T2 follows T1, by a series of swaps of non-conflicting instructions making the below Schedule conflict serializable.

T1	T2
Read(X)	
Write(X)	
	Read(X)
	Write(X)
Read(Y)	
Write(Y)	
	Read(Y)
	Write(Y)

View serializability:

1. S and S` are view equivalent if the following three conditions are met:
    a. i. For each data item P, if transaction Ti reads the initial value of P in schedule S, then transaction Ti must, in schedule S`, also read the initial value of P.
    b. ii. For each data item P, if transaction Ti executes read (P)in schedule S, and that value was produced by transaction Tj, then transaction Ti must in schedule S` also read the value of P that was produced by transaction Tj.
    c. iii. For each data item P, the transaction that performs the final write(P) operation in schedule S must perform the final write(P) operation in schedule S`.
2. View equivalence is also based purely on reads and writes alone.
3. A schedule S is view serializable if it is ie w equivalent to a serial schedule.
4. Every conflict serializable schedule is also view serializable.
5. Every view serializable schedule which is not conflict serializable has blind writes.

T3	T4	T6
Read(P)		
	Write(P)	
Write(P)		
		Write(P)