Database Management System

Notes

Ashirbad swain 8/21/20 SilentCoder

Database Management System Notes

Database

- The database is a collection of inter-related data which is used to retrieve, insert and delete the data efficiently or easily.
- It is also used to organize the data in the form of a table, schema, views and reports etc.
- Example: The college database organizes the data about the admin, staff, students and faculty etc.

Database Management System

- Database Management System is a software which is used to manage the database.
- Example: MySQL, Oracle etc.
- DBMS provides an interface to perform various operations like database creation, storing data in it, updating data, creating a table in the database etc.
- It provides protection & security to the database. In the case of multiple users, it also maintains data consistency.

Advantages of DBMS

- Controls Database Redundancy

It can control data redundancy because it stores all the data in one single database file and that recorded data is placed in the database.

- Data Sharing

In DBMS, the authorized users of an organization can share the data among multiple users.

- Easily Maintenance

It can be easily maintainable due to the centralized nature of the database system.

- Reduce Time

It reduces development time and maintenance need.

- Backup

It provides backup and recovery subsystems which create automatic backup of data from hardware and software failures and restores the data if required.

- Multiple User Interface

It provides different types of user interfaces like graphical user interfaces, application program interfaces.

Disadvantages of DBMS

- Cost of Hardware & Software

It requires a high speed of data processor and large memory size to run DBMS software.

- Size

It occupies a large space of disks and large memory to run them efficiently.

- Complexity

Database system creates additional complexity and requirements.

- **Higher impact of Failure**

Failures is highly impacted the database because in most of the organization, all the data stored in a single database and if the database is damaged due to electric failure or database corruption then the data may be lost forever.

What is the 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,

1. Storage of data
2. Retrieval of data

- **Storage of data**

According to the principle of database system, the data is stored in such a way that it acquires lot less space as the redundant or duplicate data has been removed before storage.

Example

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 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 (duplicate 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.

- **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.

Applications of DBMS

- **Telecom**

There is a database to keep track of the information regarding calls made, network usage, customer details etc. Without the database, 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 center, each one needs a database to keep track of the records of ins and outs.

- **Banking System**

For storing customer information, tracking day to day credit and debit transactions, generating bank statements etc. all this work has been done with the help of DBMS.

- **Sales**

To store customer information, production information and invoice details.

- **Airlines**

To travel through airlines, we make early reservations, this reservation information along with flight schedule is stored in database.

➲ 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, fee details etc.

➲ Online Shopping

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 recent activity.

DBMS Architecture

The architecture of DBMS depends on the computer system on which it runs.

Example: In a client-server DBMS architecture, the database systems at server machine can run several requests made by client machine.

Types of DBMS Architecture

There are 3 types of DBMS architecture,

1. Single Tier Architecture
2. Two Tier Architecture
3. Three Tier Architecture

Single Tier Architecture

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

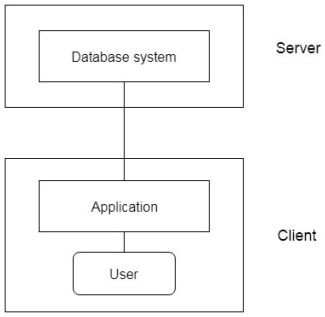
Example: Let's say you want to fetch the records of employee from the database and the database is available on your computer system, so the request to fetch employee details will be done by your computer and the records will be fetched from the database by your computer as well. This type of system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

2-Tier Architecture

The 2-Tier architecture is same as basic client-server.

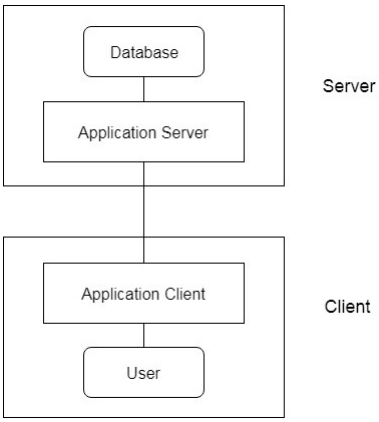
In two-tier architecture, the database system is present at the server machine and the DBMS application is present at the client machine, these two machines are connected with each other through a network connection.

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



3-Tier Architecture

In three-tier architecture, another layer is present between the client 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.



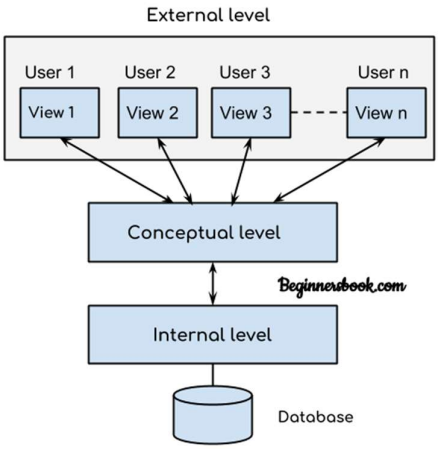
DBMS – Three Level Architecture

The three-level architecture is also called as ANSI/SPARC architecture or three-schema architecture.

This framework is used to describe the structure of a specific database system.

This architecture has 3 levels,

1. **External Level**
2. **Conceptual Level**
3. **Internal Level**



External Level

External level is the 'top level' of the Three Level DBMS Architecture.

It is also called as view level. The reason this level is called "view" is because several users can view their desired data from this level which is internally fetched from database with the help of conceptual and internal level mapping.

The user doesn't need to know the database schema details such as data structure, table definition etc. User only concerned about data which is what returned back to the view level after it has been fetched from database.

Conceptual Level

It is also called as logical level. The whole design of the database such as relationship among data, schema of data etc. are described in this level.

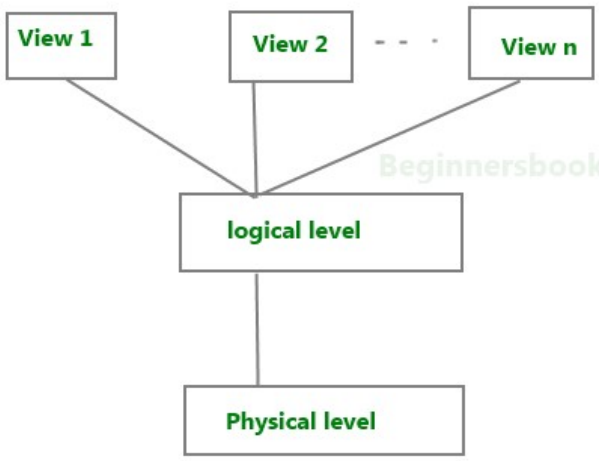
Database constraints and security are also implemented in this level of architecture. This level is maintained by DBA.

Internal Level

This level is also known as physical level. This level describes how the data is actually stored in the storage devices. This level is also responsible for allocating space to the data. This is the lowest level of the architecture.

Data Abstraction in DBMS

To ease the user interaction with database, the developers hide internal irrelevant details from the users. This process of hiding irrelevant details from user is called data abstraction.



Three Levels of data abstraction

We have 3 levels of abstraction,

➲ Physical Level

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

➲ Logical Level

This is the middle level of data abstraction. It describes what data is stored in database.

➲ View Level

This the highest level of data abstraction. This level describes the user interaction with database system.

Schema

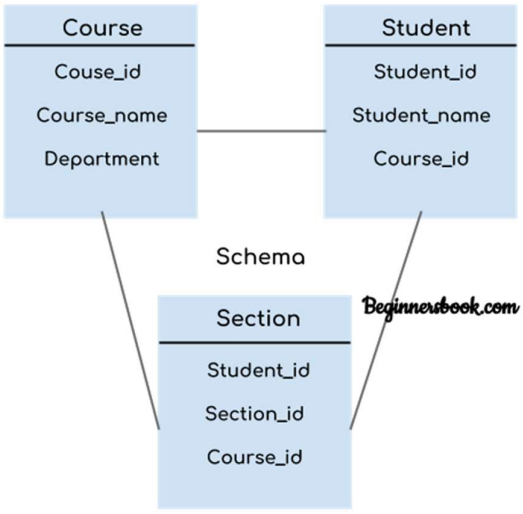
Design of a database is called schema.

Schema is of 3 types,

1. Physical Schema
2. Logical Schema
3. View Schema

Example

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.



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.

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

Instance

The data stored in database at a particular moment of time is called instance of database.

Database schema defines the variable declarations in tables that belongs to a particular database, the value of these variables at a moment of time is called the instance of that database.

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. Let's say we are going to add another 100 records in this table by tomorrow so the instance of the database tomorrow will have 200 records in the table. In short, at a particular moment the data stored in database is called the instance, that changes over time when we add or delete data from the database.

Data Models in DBMS

Data Model is logical structure of database.

It describes the design of database to reflect entities, attributes, relationship among data, constrains etc.

Types of Data Models

There are several types of data model such as,

Object based logical Models

Describe data at the conceptual and view level,

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 level. These models specify logical structure of database with records, fields and attributes.

1. Relational Model
2. Hierarchical Model
3. Network Model – network mode is same as hierarchical model except that it has graph like structure rather than a tree-based structure.

Physical Data Models

These models describe data at the lowest level of abstraction.

ER Model (Entity Relationship Model)

An Entity-Relationship 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 ER model are: Entity Set & Relationship Set.

ER Diagram

An ER diagram shows the relationship among entity sets.

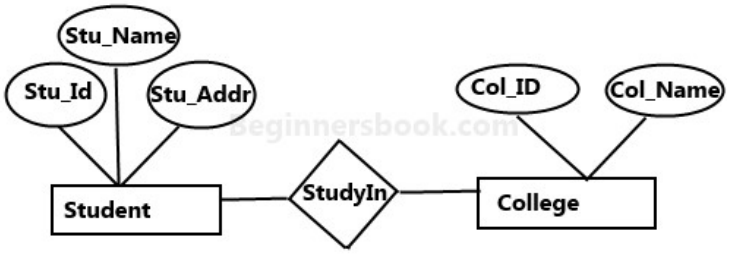
An entity set is a group of similar entities and these entities can have attributes.

ER Diagram shows the complete logical structure of a database.

Example

We have 2 entities Student & College and their relationship. The relationship between Student & 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.



Sample E-R Diagram

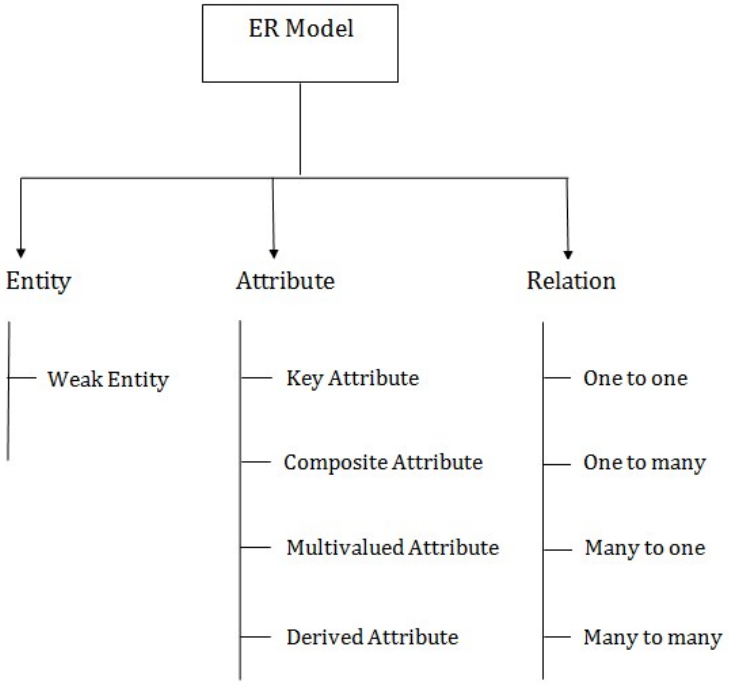
Here are the geometric shapes and their meaning in, an ER Diagram.

- **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 ER Diagram

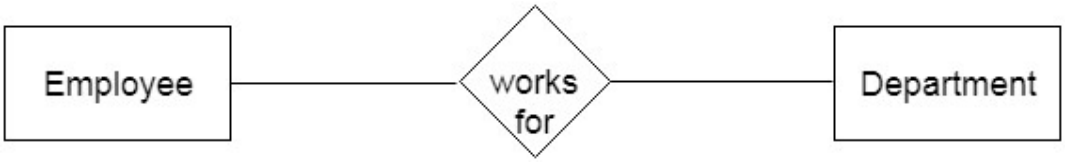
An ER Diagram has 3 main components,

1. **Entity**
2. **Attribute**
3. **Relationship**



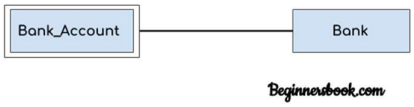
Entity

- **An entity is an object or component of data.**
- **An entity is represented as rectangle in an ER Diagram.**
- **Consider an organization as an example – manager, product, employee, department etc. can be taken as an entity.**



Weak Entity

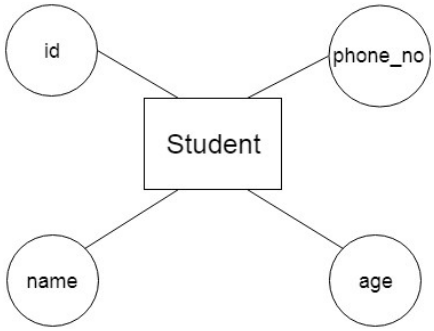
An entity that depends on another entity is called a weak entity. The weak entity doesn't contain any key attribute of its own.



Attribute

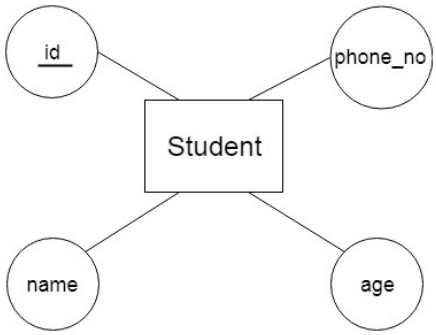
The attribute is used to describe the property of an entity. Eclipse is used to represent an attribute.

Example: id, age, contact number, name etc. can be attributes of a student.



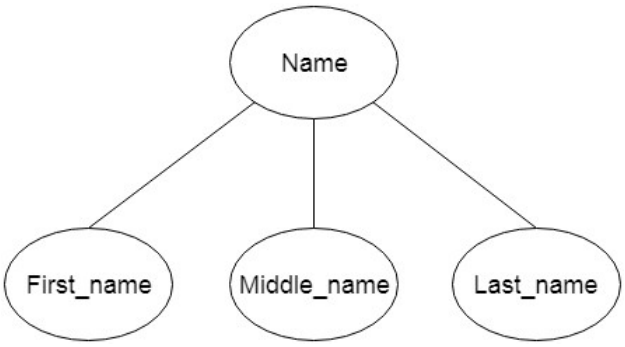
Key Attribute

The key attribute is used to represent the main characteristics of an entity. It represents a primary key. The key attribute is represented by an ellipse with the text underlined.



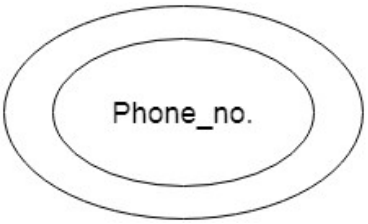
Composite Attribute

An attribute that composed of many other attributes is known as a composite attribute. The composite attribute is represented by an ellipse, and those ellipses are connected with an ellipse.



Multivalued Attribute

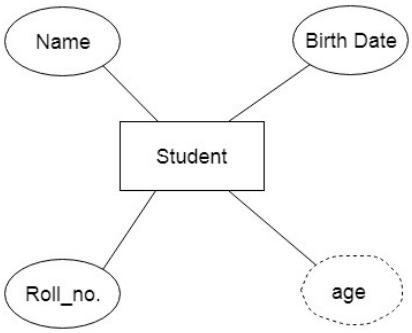
An attribute can have more than one value. These attributes are known as multivalued attribute. The double oval is used to represent multivalued attribute.



Derived Attribute

An attribute that can be derived from another attribute is known as derived attribute. It can be represented by a dashed ellipse.

Example: a person's age changes over time and can be derived from another attribute like Date of birth.



Relationship

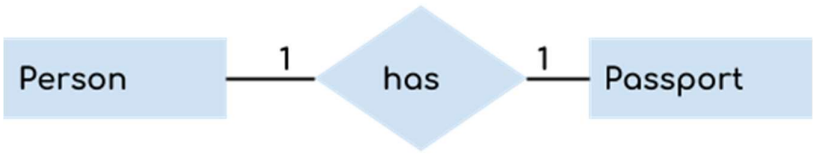
A relationship is used to describe the relation between entities. Diamond or rhombus is used to represent the relationship.

Types of relationship are,

One-to-One Relationship

When only one instance of an entity is associated with the relationship, then it is known as one to one relationship.

Example: A person has only one passport and a passport is given to one person.

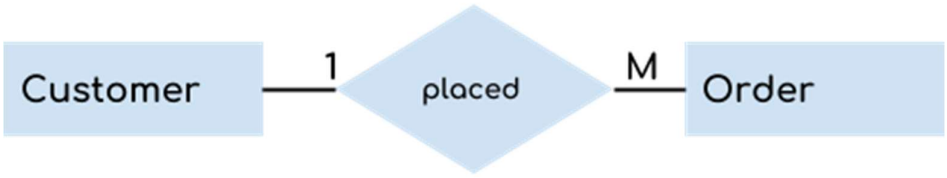


Beginnersbook.com

One-to-Many Relationship

When a single instance of an entity is associated with more than one instances of another entity then it is called one to many relationships.

Example: a customer can pace many orders but an order cannot be placed by many customers.

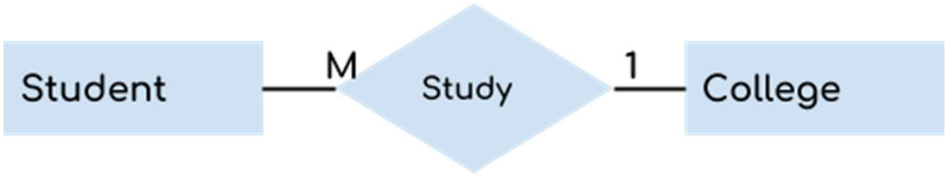


Beginnersbook.com

Many-to-One Relationship

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.

Example: many students can study in a single college but a student cannot study in many colleges at the same time.



Beginnersbook.com

Many-to-Many Relationship

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.

Example: a student can be assigned to many projects and a project can be assigned to many students.

Beginnersbook.com

DBMS Generalization

Generalization is a process in which the common attributes of more than one entity form a new entity. The newly formed entity is called generalized entity.

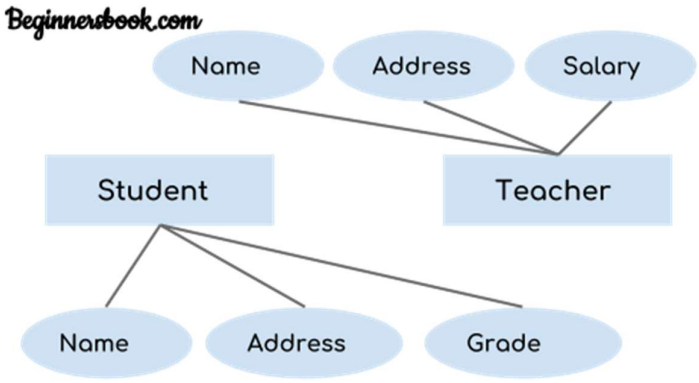
Example:

Let's say we have two entities Student & Teacher,

Attributes of Entity Student are: Name, Address & Grade

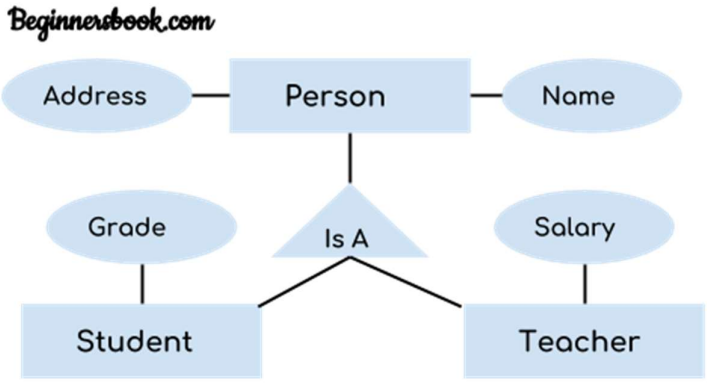
Attributes of Entity Teacher are: Name, Address & Salary

These 2 entities have two common attributes: Name & Address, we can make a generalized entity with these common attributes.



Before Generalization

We have created a new generalized entity Person and this entity has the common attributes of both the entities. As you can see the ER Diagram that after the generalization process the entities Student and Teacher only has the specialized attributes Grade & Salary respectively and their common attributes (name & address) are now associated with a new entity Person which is in the relationship with both the entities (Student & Teacher).



Generalization

Generalization uses bottom-up approach where two or more lower level entities combine together to form a higher-level new entity.

DBMS Specialization

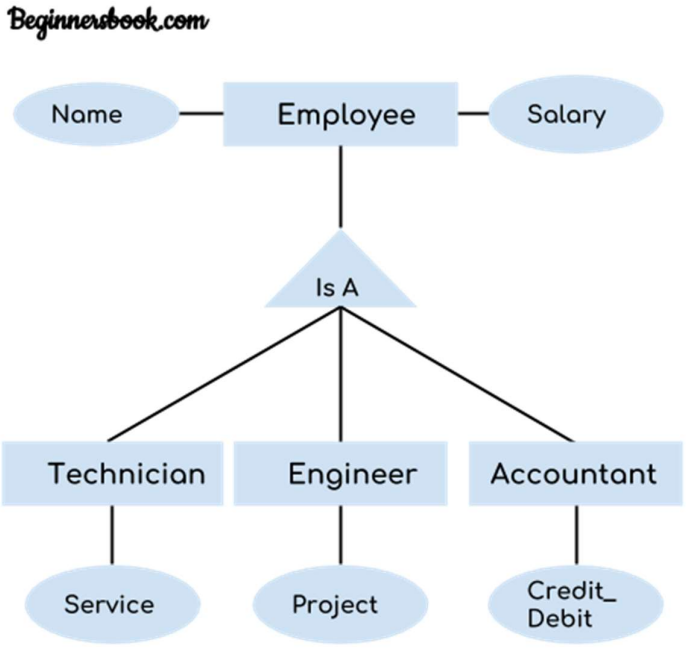
Specialization is a process in which an entity is divided into sub-entities.

It is the reverse process of generalization. In generalization two entities combine together to form a new higher-level entity.

Specialization is a top-down process.

Example

Consider an entity employee which can be further classified as sub-entities Technician, Engineer & Accountant because these sub-entities have some distinguish attributes like salary, name etc.



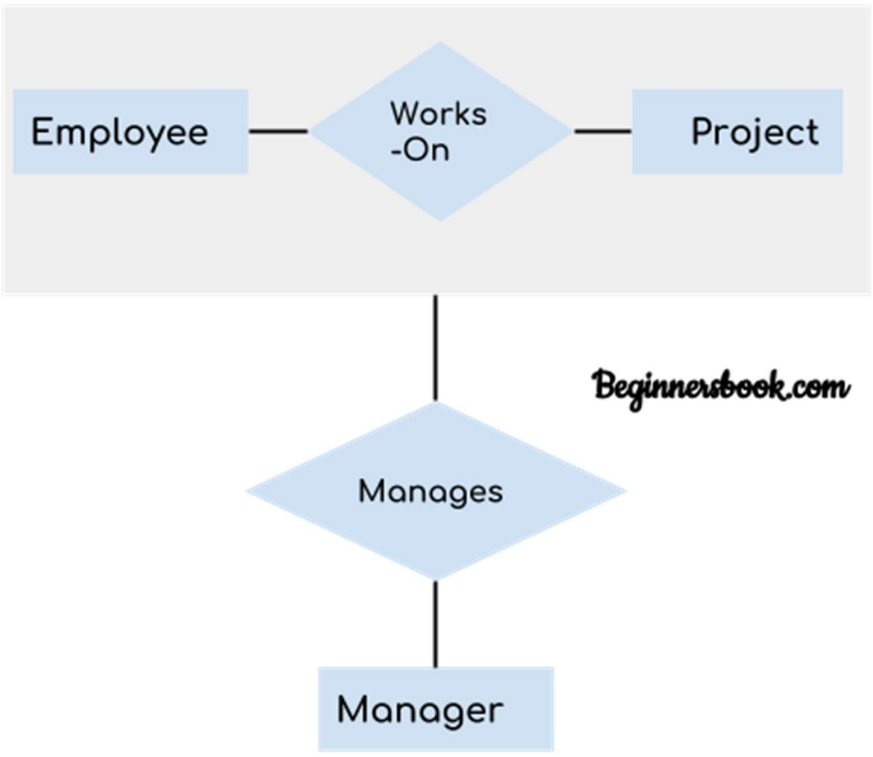
Specialization

Here, we have a higher-level entity “Employee” which we have divided in sub-entities “Technicians”, “Engineer” & “Accountant”. All of these are just an employee of a company; however, their role is completely different and they have few different attributes. Just for example, technicians handle service requests, Engineer works on a project and accountant handles the credit a& debit details. All of these 3 employee types have few attributes common such as name & salary which we had left associated with the parent entity “Employee”.

DBMS Aggregation

Aggregation is a process in which a single entity alone is not able to make sense in a relationship so the relationship of two entities acts as one entity.

Example



In real world, we know that a manager not only manages the employee working under them but he has to manage the project as well. In such scenario if entity “Manager” makes a “manages” relationship with either “Employee” or “Project” entity alone then it will not make any sense because he has to manage both. In these cases, the relationship of two entities acts as one entity. In our example, the relationship “Works-On” between “Employee” & “Project” acts as one entity that has a relationship “Manages” with the entity “Manager”.

Relational Model in DBMS

In relational model, the data and relationships are represented by collection of inter-related tables.

Each table is a group of column and rows, where column represents attribute of an entity and rows represent records.

Example:

Student table with 3 columns and four records,

Table: Student

Stu_Id	Stu_Name	Stu_age
111	Ashish	23
123	Saurav	22
169	Lester	24
234	Lou	26

Table: Course

Stu_Id	Course_Id	Course_Name
111	C01	Science
111	C02	DBMS
169	C22	Java
169	C39	Computer Networks

Here, Stu_Id, Stu_Name & Stu_Age are attributes of table Student and Stu_Id, Course_Id & Course_Name are attributes of table Course. The rows with values are the records (or tuples).

Hierarchical Model

In Hierarchical model, data is organized into a tree like structure with each record having one parent record and many children.

The main drawback of this model is that, it can have only one to many relationships between nodes.

Hierarchical models are rarely used now.

Constraints in DBMS

The main purpose of constraints is to maintain the data integrity (accuracy and consistency of data) during update/delete/insert into a table.

Types of Constraints

- NOT NULL
- UNIQUE
- DEFAULT
- CHECK
- Key Constraints – Primary key, foreign key

- Domain Constraints
- Mapping Constraints

NOT NULL

NOT NULL constraint makes sure that a column does not hold NULL value. When we don't provide any value for a particular column while inserting a record into a table, it takes NULL value by default.

Example

```
CREATE TABLE STUDENT (
ROLL_NO INT NOT NULL,
STU_NAME VARCHAR (35) NOT NULL,
STU_AGE INT NOT NULL,
STU_ADDRESS VARCHAR (235),
PRIMARY KEY (ROLL_NO)
);
```

UNIQUE

UNIQUE constraint enforces a column or set of columns to have unique values. If a column has unique constraint, it means that particular column cannot have duplicate values in a table.

Example

```
CREATE TABLE STUDENT (
ROLL_NO INT NOT NULL,
STU_NAME VARCHAR (35) NOT NULL UNIQUE,
STU_AGE INT NOT_NULL,
STU_ADDRESS VARCHAR (35) UNIQUE,
PRIMARY KY (ROLL_NO)
);
```

DEFAULT

The DEFAULT constraint provides a default value to a column when there is no value provided while inserting a record into a table.

```
CREATE TABLE STUDENT (
ROLL_NO INT NOT NULL,
STU_NAME VARCHAR (35) NOT NULL,
STU_AGE INT NOT NULL,
EXAM_FEE INT DEFAULT 10000,
STU_ADDRESS VARCHAR (35),
PRIMARY KEY (ROLL_NO)
);
```

CHECK

This constraint is used for specifying range of values for a particular column of a table. When this constraint is being set on a column, it ensures that the specified column must have the value falling in the specified range.

In the below example, we have set the CHECK constraint on ROLL_NO of STUDENT table. Now, the ROLL_NO field must have the value greater than 1000.

```
CREATE TABLE STUDENT (
ROLL_NO NOT NULL CHECK (ROLL_NO > 1000),
STU_NAME VARCHAR (35) NOT NULL,
STU_AGE INT NOT NULL,
EXAM_FEE INT DFFAULT 10000,
STU_ADDRESS VARCHAR (35),
```

```
PRIMARY KEY (ROLL_NO)
);
```

Domain Constraints

Each table has certain set of columns and each column allows a same type of data, based on its data type. The column does not accept values of any other data type.

Domain constraints are user defined data types and we can define them like,

Domain Constraint = Data type + Constraints (NOT NULL/ UNIQUE/ PRIMARY KEY/ FOREIGN KEY/ CHECK/ DEFAULT)

Mapping Constraints

Mapping constraints can be explained in terms of mapping cardinality,

- **One to One**
- **One to Many**
- **Many to One**
- **Many to Many**

Cardinality in DBMS

In Context of Data Models,

In terms of data models, cardinality refers to the relationship between two tables. Relationship can be of 4 types,

- **One to One – A single row of first table associates with single row of second table.**

For example, a relationship between person and passport table is one to one because a person can have only one passport and a passport can be assigned to only one person.

- **One to Many – A single row of first table associates with more than one rows of second table.**

For example, relationship between customer and order table is one to many because a customer can place many orders but a order can be placed by a single customer alone.

- **Many to One – Many rows of first table associate with a single row of second table.**

For example, relationship between student and university is many to one because a university can have many students but a student can only study only in single university at a time.

- **Many to Many – Many rows of first table associate with many rows of second table.**

For example, relationship between student and course table is many to many because a student can take many courses at a time and a course can be assigned to many students.

In Context of Query Optimization

In terms of query, the cardinality refers to the uniqueness of a column in a table. The column with all unique values would be having the high cardinality and the column with all duplicate values would be having the low cardinality. These cardinality scores help in query optimization.

RDBMS Concepts

- RDBMS stands for relational database management system.
- A relational model can be represented as a table of rows and columns.
- A relational database has following major components,

1. Table
2. Record or Tuple
3. Field or Column name or Attribute
4. Domain
5. Instance
6. Schema
7. Keys

Table

A table is a collection of data represented in rows and columns. Each table has a name in database.

For example, the following table "STUDENT" stores the information of students in database.

Table: STUDENT

Student_Id	Student_Name	Student_Addr	Student_Age
101	Ashirbad	Dhenkanal	22
102	Abinash	Delhi	26
103	Rahul	Bangalore	24
104	Shubham	Chennai	23

Record or Tuple

Each row of a table is known as a record. It is also known as tuple.

For example, the following row is a record that we have taken from the above table.

102	Abinash	Delhi	26

Filed or Column name or Attribute

The above table "STUDENT" has four fields or attribute: Student_Id, Student_Name, Student_Addr & Student_Age.

Domain

A domain is a set of permitted values for an attribute in table.

We specify domain of attribute while creating a table.

Example:

A domain of month-of-year can accept January, February… December as values, a domain of dates can accept all possible valid dates etc.

An attribute cannot accept values that are outside of their domains. For example, in the above table "Student", the Student_Id has integer domain so that field cannot accept values that are not integers.

Query Language

A language which is used to store and retrieve data from database is known as query language.

For example – SQL

There are 2 types of query language,

1. Procedural Query Language
2. Non-Procedural Query Language

Procedural Query Language

In procedural Query Language, user instructs the system to perform a series of operations to produce the desired results.

Here users tell what data to be retrieved from database and how to retrieve it.

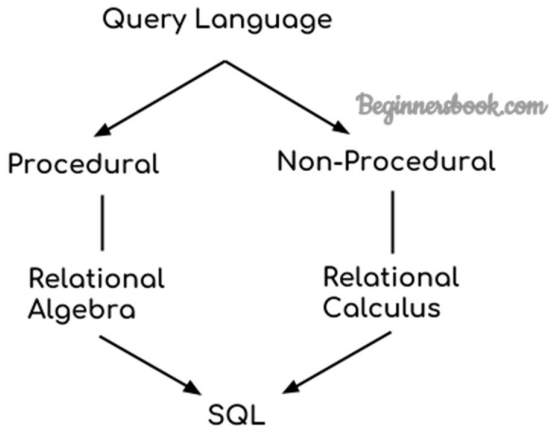
Example

Suppose you are asking your sister to make a cup of tea, if you are just telling her to make a tea and not telling the process then it is a non-procedural language, however if you are telling the step by step process like switch on the stove, boil the water, add milk etc. then it is a procedural language.

Non-Procedural Language

In non-procedural query language, user instructs the system to produce the desired result without telling the step-by-step process.

Here users tell what data to be retrieved from database but doesn't tell how to retrieve it.



Note:

- **Relational algebra and calculus are the theoretical concepts used on relational model.**
- **RDBMS is a practical implementation of relational model.**
- **SQL is a practical implementation of relational algebra and calculus.**
- **The purpose of a query language is to retrieve data from database or perform various operations such as insert, update, delete on the data.**

DBMS Relational Algebra

Relational algebra is a procedural query language that works on relational table.

The relational algebra is a procedural query language, it means that it tells what data to be retrieved and how to be retrieved.

Types of operations in relational algebra

The operations divided into 2 categories,

1. **Basic/Fundamental Operations**
    1. **Select (σ)**
    2. **Project (∏)**
    3. **Union (∪)**
    4. **Set difference (-)**
    5. **Cartesian Product (X)**
    6. **Rename (ρ)**

2. **Derived Operations**
    1. **Natural Join (⋈)**
    2. **Left, Right, Full, Outer Join (⟕, ⟖, ⟗)**
    3. **Intersection (∩)**
    4. **Division (÷)**

Select Operator (σ)

Select operator is denoted by sigma (σ) and it is used to find the tuples or rows in a relation (table) which satisfy the given condition.

It is like the where clause in SQL means it is used for the same purpose.

Syntax

```
σ Condition/Predicate (Relation/Table name)
```

Example:

Table: CUSTOMER

Customer_Id	Customer_Name	Customer_City
C10100	Steve	Agra
C10111	Raghu	Agra
C10115	Chaitanya	Noida
C10117	Ajeet	Delhi
C10118	Carl	Delhi

Query

```
σ Customer_City = "Agra" (CUSTOMER)
```

Output

Customer_Id	Customer_Name	Customer_City
C10100	Steve	Agra
C10111	Raghu	Agra

Project Operator (∏)

Project operator is denoted by capital pi (∏) symbol and it is used to select desired columns or attributes from a table.

Project operator in relational algebra is similar to the Select statement in SQL.

Syntax

```
∏ column_name1, column_name2, ...., column_nameN(table_name)
```

Example

We have a table CUSTOMER with 3 columns, we want to fetch only 2 columns of the table, which we can do with the help of project operator.

Table: CUSTOMER

Customer_Id	Customer_Name	Customer_City
C10100	Steve	Agra
C10111	Raghu	Agra
C10115	Chaitanya	Noida
C10117	Ajeet	Delhi
C10118	Carl	Delhi

Query

```
∏ Customer_Name, Customer_City (CUSTOMER)
```

Output

Customer_Name	Customer_City
Steve	Agra
Raghu	Agra
Chaitanya	Noida
Ajeet	Delhi
Carl	Delhi

Union Operator (∪)

Union operator is denoted by (∪) symbol and it is used to select all the rows (tuples) from two tables.

Example: let's say we have 2 relations R1 and R2 both have same columns and we want to select all the tuples from these relations then we can apply the union operator on these relations.

The rows that are present in both the tables will only appear once in the union set. In short you can say that there are no duplicates present after the union operation.

Syntax

```
table_name1 ∪ table_name2
```

Example: Table 1: COURSE

```
Course_Id   Student_Name   Student_Id
---------  ------------   ----------
C101        Aditya         S901
C104        Aditya         S901
C106        Steve          S911
C109        Paul           S921
C115        Lucy           S931
```

Table 2: STUDENT

```
Student_Id     Student_Name   Student_Age
------------   ----------     -----------
S901           Aditya         19
S911           Steve          18
S921           Paul           19
S931           Lucy           17
S941           Carl           16
S951           Rick           18
```

Query

```
∏ Student_Name (COURSE) ∪ ∏ Student_Name (STUDENT)
```

Output

```
Student_Name
------------
Aditya
Carl
Paul
Lucy
Rick
Steve
```

Intersection Operator (∩)

Intersection operator is denoted by (∩) symbol and it is used to select common rows (tuples) from 2 tables.

Let's say we have 2 relations R1 & R2 both have same columns and we want to select all those tuples that are present in both the relations, then in that case we can apply intersection operation.

Only those rows that are present in both the tables will appear in the result set.

Syntax

```
table_name1 ∩ table_name2
```

Example:

Table COURSE & STUDENT

Query

```
∏ Student_Name (COURSE) ∩ ∏ Student_Name (STUDENT)
```

Query

```
Student_Name
------------
Aditya
Steve
Paul
Lucy
```

Set Difference (-)

Set Difference is denoted by (-) symbol.

Let's say we have 2 relations R1 & R2 and we want to select all those tuples (rows) that are in R1 but not present in R2, this can be done using Set Difference R1 – R2.

```
Syntax: table_name1 - table_name2
```

Query

```
∏ Student_Name (STUDENT) - ∏ Student_Name (COURSE)
```

Output

```
Student_Name
------------
Carl
Rick
```

Cartesian Product (X)

Cartesian Product is denoted by (X) symbol.

Let's say we have 2 relations R1 & R2 then the cartesian product of these 2 relations (R1 X R2) would be combine each tuple of first relation R1 with each tuple of second relation R2.

Syntax

```
R1 X R2
```

Example

Table 1: R

```
Col_A    Col_B
-----    ------
AA        100
BB        200
CC        300
```

Table 2: S

```
Col_X     Col_Y
-----     -----
XX         99
YY         11
ZZ         101
```

Query

```
R X S
```

Output

```
Col_A    Col_B     Col_X     Col_Y
-----    ------    ------    ------
AA        100       XX        99
AA        100       YY        11
AA        100       ZZ        101
BB        200       XX        99
BB        200       YY        11
BB        200       ZZ        101
CC        300       XX        99
CC        300       YY        11
CC        300       ZZ        101
```

The number of rows in the output will always be the cross product of number of rows in each table. In our example table 1 has 3 rows and table 2 has 3 rows so the output is 3x3 = 9 rows.

Rename (ρ)

Rename (ρ) operation can be used to rename a relation or an attribute of a relation.

Syntax

```
ρ (new_relation_name, old_relation_name)
```

Example

Let's say we have a table CUSTOMER; we are fetching customer names and we are renaming the resulted relation to CUST_NAMES.

```
Customer_Id     Customer_Name      Customer_City
-----------     -------------      -------------
C10100           Steve              Agra
C10111           Raghu              Agra
C10115           Chaitanya          Noida
C10117           Ajeet              Delhi
C10118           Carl               Delhi
```

Query

```
P (CUST_NAMES, ∏(Customer_Name) (CUSTOMER))
```

Output

```
CUST_NAMES
----------
Steve
Raghu
Chaitanya
Ajeet
Carl
```

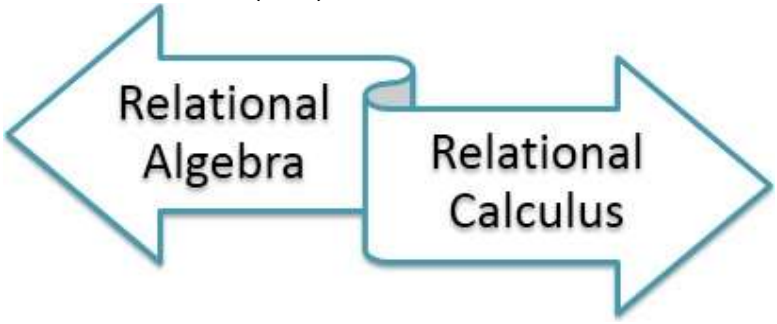
DBMS Relational Calculus

Relational Calculus is a non-procedural query language that tells the system what data to be retrieved but doesn't tell how to retrieve it.

Types of Relational Calculus

There are 2 types of Relational Calculus,

1. **Tuple Relational Calculus (TRC)**
2. **Domain Relational Calculus (DRC)**



Tuple Relational Calculus

Tuples Relational calculus is used for selecting those tuples that satisfy the given condition.

Example

Table: STUDENT

```
First_Name      Last_Name      Age
----------      ---------      ----
Ajeet           Singh          30
Chaitanya       Singh          31
Rajeev          Bhatia         27
Carl            Pratap         28
```

Query

Query to display the last name of those students where age id greater than 30.

```
{t. Last_Name | Student(t) AND t.age > 30}
```

Query to display all the details of student where Last name is "Singh".

```
{t | Student(t) AND t. Last_Name = 'Singh'}
```

In the above query you can see 2 parts separated by | symbol. The second part is where we define the condition and in h first part, we specify the fields which we want to display for the selected tuples.

Output

```
Last_Name
---------
Singh
```

```
First_Name      Last_Name      Age
----------      ---------      ----
```

```
Ajeet          Singh          30
Chaitanya      Singh          31
```

Domain Relational Calculus

In domain relational calculus the records are filtered based on the domains.

Example:

Table: STUDENT

```
First_Name     Last_Name      Age
----------     ---------      ----
Ajeet          Singh          30
Chaitanya      Singh          31
Rajeev         Bhatia         27
Carl           Pratap         28
```

Query to find the first name and age of students where student age is greater than 27.

```
{< First_Name, Age > | ∈ Student ∧ Age > 27}
```

The symbol used for logical operations are: ∧ for AND, v for OR.

Output

```
First_Name     Age
----------     ----
Ajeet          30
Chaitanya      31
Carl           28
```

Keys in DBMS

It is used for identifying unique rows from table. It also establishes relationship among tables.

Example: in Student table, ID is used as a key because it is unique for each student.

Types of keys,

- **Primary Key**
- **Super Key**
- **Candidate key**
- **Alternate Key**
- **Composite Key**
- **Foreign Key**

Primary Key

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

Example-1

In the table STUDENT, there are 3 attributes: Stu_ID, Stu_Name, Stu_Age. Out of these 3 attributes, one attribute or a set of attributes can be a primary key.

Attribute Stu_Name alone cannot be a primary key as more than one student can have same name.

Attribute Stu_Age alone cannot be a primary key as more than one student can have same.

Attribute Stu_ID alone is a primary key as each student has a unique id that can identify the student record in the table.

In some cases, an attribute alone cannot uniquely identify a record in a table, in that case we try to find a set of attributes that can uniquely identify a row in table.

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

Example – 2

Consider the table ORDER, which keeps the daily record of the purchases made by the customer. This table has 3 attributes: Customer_ID, Product_ID & Order_Quantity.

Customer_ID alone cannot be a primary key as a single customer can place more than one order with the same Customer_ID. As in the following table that Customer_ID 1011 has placed 2 orders with Product_ID 9023 and 9111.

Product_ID alone cannot be a primary key as more than one customer can place an order for the same Product_ID. In the following table, Customer_ID 1011 & 1112 placed an order for the same product.

Order_Quantity alone cannot be a primary key as more than one customer can place the order for same quantity.

Since none of the attributes alone were able to become a primary key, {Customer_ID, Product_ID} together can identify the rows uniquely in the table so this set is primary key for his table.

Customer_ID	Product_ID	Order_Quantity
1011	9023	10
1122	9023	15
1099	9031	20

1177	9031	18
1011	9111	50

While choosing a set of attributes for a primary key, we always choose the minimal set that has minimum number of attributes.

How to define primary key?

Let's say we want to create Customer_ID & Product_ID set as primary key for the ORDER table,

```
Create table ORDER
(
    Customer_ID int not null,
    Product_ID int not null,
    Order_Quantity int not null,
    Primary key (Customer_ID, Product_ID)
)
```

Suppose we didn't define the primary key while creating the table, then we can define it like,

```
ALTER TABLE ORDER
ADD CONSTRAINT PK_Order PRIMARY KEY (Customer_ID, Product_ID);
```

OR

When we have only one attribute as primary key, like in STUDENT table. We can define the key like,

```
Create table STUDENT
(
    Stu_Id int primary key,
    Stu_Name varchar (255) not null,
    Stu_Age int not null
)
```

Super Key

A super key is a set of one or more attributes (columns), which can uniquely identify a row in a table.

How Candidate Key is different from super key?

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

Example

Table: Employee

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

The above table has following super keys. All of the following sets of super keys are uniquely identifying a row of the employee table.

- **{Emp_SSN}**
- **{Emp_Number}**
- **{Emp_SSN, Emp_Number}**
- **{Emp_SSN, Emp_Name}**
- **{Emp_SSN, Emp_Number, Emp_Name}**
- **{Emp_Number, Emp_Name}**

Candidate Key

A super key with no redundant attribute is known as candidate key.

Candidate keys are selected from the set of super keys.

Example:

Table: Employee

This table has 3 attributes: Emp_Id, Emp_Number & Emp_Name. Here Emp_Id & Emp_Number will be having unique values and Emp_Name can have duplicate values as more than one employee can have same name.

Emp_Id	Emp_Number	Emp_Name
E01	2264	Steve
E22	2278	Ajeet
E23	2288	Chaitanya
E45	2290	Robert

The super keys are,

1. {Emp_Id}
2. {Emp_Number}
3. {Emp_Id, Emp_Number}
4. {Emp_Id, Emp_Name}
5. {Emp_Id, Emp_Number, Emp_Name}
6. {Emp_Number, Emp_Name}

Let's select the candidate keys from the above set of super keys,

1. {Emp_Id} – No redundant attributes
2. {Emp_Number} – No redundant attributes
3. {Emp_Id, Emp_Number} – Redundant attribute. Either of those attributes can be a minimal super key as both of these columns have unique values.
4. {Emp_Id, Emp_Name} – Redundant attribute Emp_Name.
5. {Emp_Id, Emp_Number, Emp_Name} – Redundant attributes. Emp_Id or Emp_Number alone are sufficient enough to uniquely identify a row of Employee table.
6. {Emp_Number, Emp_Name} – Redundant attribute Emp_Name.

So, the candidate keys are,

{Emp_Id}
{Emp_Number}

A primary key is selected from the set of candidate keys. That means we can either have Emp_Id or Emp_Number as primary key. The decision is made by DBA.

Foreign Key

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

Example

In the following example the Stu_Id column in Course_enrollment table is a foreign key as it points to the primary key of the Student table.

Course_enrollment table:

Course_Id	Stu_Id
C01	101
C02	102
C03	101
C05	102
C06	103
C07	102

Student table:

Stu_Id	Stu_Name	Stu_Age
101	Chaitanya	22
102	Arya	26
103	Bran	25
104	Jon	21

Practically, the foreign key has nothing to do with the primary key tag of another table. If it points to a unique column of another table then, it would be a foreign key.

So, a correct definition of foreign key would be: Foreign keys are the columns of a table that points to the candidate key of another table.

Composite Key

A key that has more than one attribute is known as composite key. It is also known as compound key.

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

Example:

Let's consider a table Sales. This table has 4 columns (attributes) – cust_Id, order_Id, product_code & product_count.

Table: Sales

```
cust_Id    order_Id    product_code    product_count
--------   --------    ------------    -------------
C01        O001          P007           23
C02        O123          P007           19
C02        O123          P230           82
C01        O001          P890           42
```

None of these columns alone can play a role of key in this table.

Column cust_Id alone cannot become a key as a same customer can place multiple orders, thus the same customer can have multiple entries.

Column order_Id alone cannot be a primary key as a same order can contain the order of multiple products, thus same order_Id can be present multiple times.

Column product_code cannot be a primary key as more than one customer can place order for the same product.

Column product_count alone cannot be a primary key because two orders can be placed for the same product count.

Based on this, it is safe to assume that the key should be having more than one attributes: Key in above table: {cust_Id, product_code}

This is a composite key as it is made up of more than one attributes.

Alternate Key

As we know, A table can have multiple candidate key. Among these candidate keys, only one key gets selected as primary key, the remaining keys are known as alternate or secondary keys.

Example:

The table "Employee" has 3 attributes: Emp_Id, Emp_Number & Emp_Name.

```
Emp_Id   Emp_Number   Emp_Name
------   ----------   --------
```

```
E01       2264          Steve
E22       2278          Ajeet
E23       2288          Chaitanya
E45       2290          Robert
```

There are 2 candidate keys in the above table,

{Emp_Id}

{Emp_Number}

DBA can choose any of the above key as primary key. Let's say Emp_Id is chosen as primary key.

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

Normalization

Normalization is a process of organizing the data in database to avoid data redundancy, insertion anomaly, update anomaly & deletion anomaly.

Anomalies in DBMS

There are 3 types of anomalies that occurs when the database is not normalized. There are – Insertion, Update and deletion anomaly.

Example: suppose a manufacturing company stores the employee details in a table named "EMPLOYEE" that has 4 attributes: emp_id for storing employee's id, emp_name for storing employee's name, emp_address for storing employee's address and emp_dept for sorting the department details in which the employee works.

The table is not normalized.

Emp_id	Emp_name	Emp_address	Emp_dept
101	Rick	Delhi	D001
101	Rick	Delhi	D002
123	Maggie	Agra	D890
166	Glenn	Chennai	D900
166	Glenn	Chennai	D004

Update anomaly

In the above table we have two rows for employee Rick as he belongs to two departments of the company. If we want to update the address of Rick then we have to update the same in two rows or the data will become inconsistent. If somehow, the correct address gets updated in one department but not in other then as per the database, Rick would be having two different addresses, which is not correct and would lead to inconsistent data.

Insert anomaly

Suppose a new employee joins the company, who is under training and currently not assigned to any department then we would not be able to insert the data into the table if emp_dept field doesn't allow nulls.

Delete anomaly

Suppose, if at a point of time the company closes the department D890 then deleting the rows that are having emp_dept as D890 would also delete the information of employee Maggie since she is assigned only to this department.

So, to overcome these anomanlies we need to normalize the data.

Functional Dependency

The functional dependency is a relationship that exists between 2 attributes. It typically exists between the primary key and non-key attribute within a table.

```
X -----> Y
```

The left side of FD is known as a **determinant**, the right side of the production is known as a **dependent**.

Example:

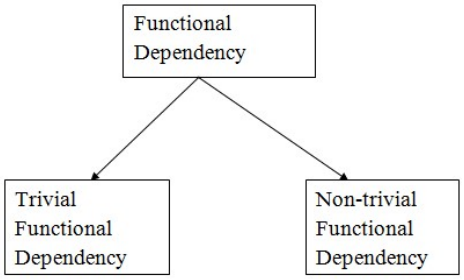
Assume we have an employee table with attributes: Emp_Id, Emp_Name, Emp_Address.

Here Emp_Id attribute can uniquely identify the Emp_Name attribute of employee table because if we know the Emp_Id, we can tell the employee name associated with it.

We can say that Emp_Name is functionally dependent on Emp_Id.

```
Emp_Id -→ Emp_Name
```

Types of FD



Trivial Functional Dependency

A -→ B has trivial functional dependency if B is a subset of A.

Example:

Consider a table with 2 columns Employee_Id and Employee_Name

{Employee_Id, Employee_Name} -→ Employee_Id is a trivial functional dependency as Employee_Id is a subset of {Employee_Id, Employee_Name}

Non-Trivial Functional Dependency

A --→ B has a non-trivial functional dependency if B is not a subset of A.

When A intersection b is NULL, then A --→ B is called as complete non-trivial.

Example: ID -→ Name, Name --→ DOB

Types of Normalization

Here are the most commonly used normal forms,

1. First Normal Form (1 NF)
2. Second Normal Form (2 NF)
3. Third Normal Form (3 NF)
4. Boyce & Codd Normal Form (BCNF)

First Normal Form (1 NF)

A per the rule of 1 NF, an attribute of a table cannot hold multiple values. It should hold only atomic values.

Example:

Suppose a company wants to store the names and contact details of its employee.

Emp_id	Emp_name	Emp_address	Emp_mobile
101	Ashirbad	New Delhi	9777028624
102	Abinash	Kanpur	9178026754 9536521255
103	Alok	Noida	9583313378
104	Pabitra	Bangalore	9777037645 7456325626

Two employees (Abinash & Pabitra) are having 2 mobile numbers so the company stored them in the same field as you can see in the table.

This table is not in 1NF as the Emp_mobile attribute contains multiple values.

So, to make the table to be in 1NF, we should have the data like,

Emp_id	Emp_name	Emp_address	Emp_mobile
101	Ashirbad	New Delhi	9777028624
102	Abinash	Kanpur	9178026754
102	Abinash	Kanpur	9536521255
103	Alok	Noida	9583313378
104	Pabitra	Bangalore	9777037645

104	Pabitra	Bangalore	7456325626

Second Normal Form (2NF)

A table is said to be in 2NF if both the following condition hold,

- Table is in 1NF
- No non-prime attribute is dependent on the proper subset of any candidate key.

An attribute that is not part of any candidate key is known as non-prime attribute.

Example:

Suppose a school wants to store the data of teachers and the subjects they teach. They create a table that looks like this: Since a teacher can teach more than one subjects, the table can have multiple rows for a same teacher.

teacher_id	subject	teacher_age
111	Maths	38
111	Physics	38
222	Biology	38
333	Physics	40
333	Chemistry	40

Candidate Keys: {teacher_id, subject}
Non-prime attribute: teacher_age

The table is in 1 NF because each attribute has atomic values. However, it is not in 2NF because non-prime attribute teacher_age is dependent on teacher_id alone which is a proper subset of candidate key. This violates the rule for 2NF as the rule says "**no** non-prime attribute is dependent on the proper subset of any candidate key of the table".

To make the table complies with 2NF we can break it in two tables like this:

Now the tables comply with 2NF.

teacher_details table:

teacher_id	teacher_age

111	38
222	38
333	40

teacher_subject table:

teacher_id	subject
111	Maths
111	Physics
222	Biology
333	Physics
333	Chemistry

Second Normal Form (2NF)

A table is said to be in 3NF if both the following conditions hold:

- Table must be in 2NF
- Transitive functional dependency of non-prime attribute on any super key should be removed.

In other words, 3NF can be explained as: A table is in 3NF if it is in 2NF and for each functional dependency X --→ Y at least one of the following conditions hold,

- X is super key
- Y is a prime attribute

An attribute that is a part of one of the candidate keys is known as prime attribute.

Transitive Functional Dependency

A functional dependency is said to be transitive if it I indirectly formed by two functional dependencies. Like,

X -→ Z is a transitive dependency if the following 3 functional dependencies hold true,

- X ---→ Y
- Y does not ----→ X
- Y --→ Z

A transitive dependency can only occur in a relation of 3 or more attribute. This dependency helps us normalizing the database in 3NF.

Example:

<MovieListing>

Movie_ID	Listing_ID	Listing_Type	DVD_Price ($)
M08	L09	Crime	180
M03	L05	Drama	250
M05	L09	Crime	180

The above table is not in 3NF because it has a transitive functional dependency

Movie_ID -> Listing_ID
Listing_ID -> Listing_Type

Therefore, the following has transitive functional dependency.

Movie_ID -> Listing_Type

The above states the relation <MovieListing> violates the 3rd Normal Form (3NF).

To remove the violation, you need to split the tables and remove the transitive functional dependency.

<Movie>

Movie_ID	Listing_ID	DVD_Price ($)
M08	**L09**	**180**
M03	**L05**	**250**
M05	**L09**	**180**

<Listing>

Listing_ID	Listing_Type
L09	Crime
L05	Drama
L09	Crime

Now the above relation is in 3NF.

Boyce Codd Normal Form (BCNF)

BCNF is the advance version of 3NF. It is stricter than 3NF.

A table is in BCNF if every functional dependency X -→ Y, X is the super key of the table.

For BCNF, the table should be in 3NF and for every FD, LHS is super key.

Example:

Let's assume there is a company where employees work in more than one department.

EMPLOYEE Table:

EMP_ID	EMP_COUNTRY	EMP_DEPT	DEPT_TYPE	EMP_DEPT_NO
264	India	Designing	D394	283
264	India	Testing	D394	300
364	UK	Stores	D283	232
364	UK	Developing	D283	549

In the above table functional dependencies are as follows,

EMP_ID → EMP_COUNTRY

EMP_DEPT → {DEPT_TYPE, EMP_DEPT_NO}

Candidate key: {EMP_ID, EMP_DEPT}

The table is not in BCNF because neither EMP_DEPT nor EMP_ID alone are keys.

To convert the given table into BCNF, we decompose it into 3 tables,

EMP_COUNTRY Table:

EMP_ID	EMP_COUNTRY
264	India
264	India
364	UK
364	UK

EMP_DEPT Table:

EMP_DEPT	DEPT_TYPE	EMP_DEPT_NO
Designing	D394	283
Testing	D394	300
Stores	D283	232
Developing	D283	549

EMP_DEPT_MAPPING Table:

EMP_ID	EMP_DEPT
264	Designing
264	Testing
364	Stores
364	Developing

Functional Dependencies:

EMP_ID → EMP_COUNTRY

EMP_DEPT → {DEPT_TYPE, EMP_DEPT_NO}

Candidate keys:

For the first table: EMP_ID

For the second table: EMP_DEPT

For the third table: {EMP_ID, EMP_DEPT}

Now, this is in BCNF because left side part of both the functional dependencies is a key.

Transaction Management

A transaction is a set of logically related operations.

Example: Suppose you are transferring money from your bank account to your friend's account, the set of operations would be like,

- Read your account balance
- Deduct the amount from your balance
- Write the remaining balance to your account
- Read your friend's account balance
- Add the amount to his account balance
- Write new updated balance to his account

This whole set of operations can be called a transaction.

In DBMS, we write the above 6 steps transaction like this,

Let's say your account is A and your friend's account is B, you are transferring 10000 from A to B, the steps of the transaction are,

In the below transaction R refers to the Read Operation and W refers to the Write Operation.

```
1. R(A);
2. A = A - 10000;
3. W(A);
4. R(B);
5. B = B + 10000;
6. W(B);
```

Transaction Failure in between in the operations,

The main problem that can happen during a transaction is that the transaction can fail before finishing all the operations in the set. This can happen due to power failure, system crash etc. This is a serious problem that can leave database in an inconsistent state. Assume that transaction fail after third operation (see the example above) then the amount would be deducted from your account but your friend will not receive it.

To solve this problem, we have the following 2 operations,

Commit: if all the operations in a transaction are completed successfully then commit those changes to the database permanently.

Rollback: if any of the operation fails then rollback all the changes done by previous operations.

Even though these operations can help us avoiding several issues that may arise during transaction but they are not sufficient when two transactions are running concurrently. To handle those problems, we need to understand database ACID Properties.

ACID Properties

The ACID properties are,

Atomicity

This property ensures that either all the operations of a transaction reflect in database or none.

Example:

Let's take an example of banking system to understand this,

Suppose Account **A** has a balance of 400$ & **B** has 700$. Account **A** is transferring 100$ to Account **B**. This is a transaction that has two operations,

1. Debiting 100$ from A's account
2. Crediting 100$ to B's balance

Let's say first operation passed successfully while second failed, in this case A's balance would be 300$ while B would be having 700$ instead of 800$. This is unacceptable in a banking system. Either the transaction should fail without executing any of the operation or it should process both the operations.

The Atomicity property ensures that.

Consistency

To preserve the consistency of database, the execution of transaction should take place in isolation means no other transaction should run concurrently when there is a transaction already running.

Example:

For example, account A is having a balance of 400$ and it is transferring 100$ to account B & C both. So, we have two transactions here. Let's say these transactions run concurrently and both the transactions read 400$ balance, in that case the final balance of A would be 300$ instead of 200$. This is wrong. If the transaction were to run in isolation, then the second transaction would have read the correct balance 300$ (before debiting 100$) once the first transaction went successful.

Isolation

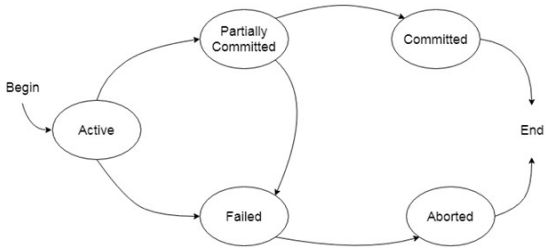
For every pair of transactions, one transaction should start execution only when the other finished execution.

Durability

Once a transaction completes successfully, the changes it has made into the database should be permanent even if there is a system failure. The recovery-management component of database systems ensures the durability of transaction.

Transaction States

A transaction in DBMS can be one of the following states,



Active State

If a transaction is in execution, then it is said to be in active state. It doesn't matter which step is in execution, until unless the transaction is executing, it remains in active state.

Failed State

If a transaction is executing and a failure occurs, either a hardware failure or a software failure then the transaction goes into failed state from the active state.

Partially Committed State

A transaction goes into "partially committed" state from the active state when there are read and write operations present in the transaction.

A transaction contains number of read and write operations. Once the whole transaction is successfully executed, the transaction goes into partially committed state where we have all the read and write operations performed on the main memory (local memory) instead of the actual database.

The reason why we have this state is because a transaction can fail during execution so if we are making the changes in the actual database instead of local memory, database may be left in an inconsistent state in case of any failure. **This state helps us to roll back the changes made to the database in case of a failure during execution.**

Committed State

If a transaction completes the execution successfully then all the changes made in the local memory during **partially committed** state are permanently stored in the database.

Aborted State

If a transaction fails during execution, then the transaction goes into a failed state. The changes made into the local memory (or buffer) are rolled back to the previous consistent state and the transaction goes into aborted state from the failed state.

DBMS Schedule

When multiple transactions are running concurrently then there needs to be a sequence in which the operations are performed because at a time only one operation can be performed on the database. This sequence of operations is known as **Schedule**.

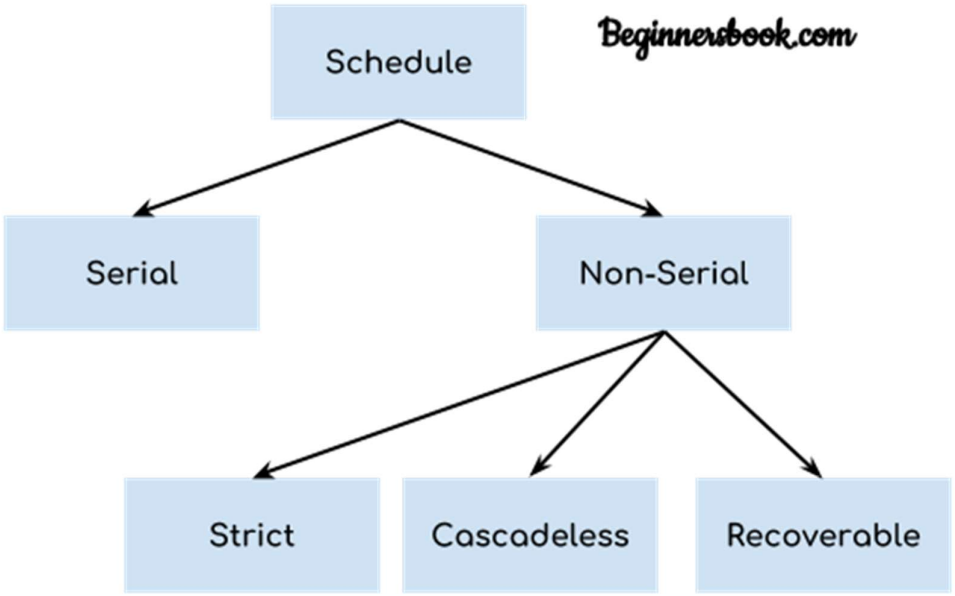
Example

The following sequence of operations is a schedule. Here we have two transactions T1 & T2 which are running concurrently.

This schedule determines the exact order of operations that are going to be performed on database. In this example, all the instructions of transaction T1 are executed before the instructions of transaction T2, however this is not always necessary.

```
T1      T2
----    ----
R(X)
W(X)
R(Y)
        R(Y)
        R(X)
        W(Y)
```

Types of Schedules



A serial schedule doesn't support concurrent execution of transactions while a non-serial schedule supports concurrency.

Serial Schedule

In **Serial schedule**, a transaction is executed completely before starting the execution of another transaction. In other words, you can say that in serial schedule, a transaction does not start execution until the currently running transaction finished execution. This type of execution of transaction is also known as **non-interleaved** execution.

Example:

Here R refers to the read operation and W refers to the write operation. In this example, the transaction T2 does not start execution until the transaction T1 is finished.

```
T1      T2
----    ----
R(A)
R(B)
W(A)
commit
        R(B)
        R(A)
        W(B)
        commit
```

Strict Schedule

In Strict schedule, if the write operation of a transaction precedes a conflicting operation (Read or Write operation) of another transaction then the commit or abort operation of such transaction should also precede the conflicting operation of other transaction.

Example:

Let's say we have two transactions Ta and Tb. The write operation of transaction Ta precedes the read or write operation of transaction Tb, so the commit or abort operation of transaction Ta should also precede the read or write of Tb.

Here the write operation W(X) of Ta precedes the conflicting operation (Read or Write operation) of Tb so the conflicting operation of Tb had to wait the commit operation of Ta.

```
Ta      Tb
-----   -----
R(X)
        R(X)
W(X)
commit
        W(X)
        R(X)
        commit
```

Cascade less Schedule

In Cascadeless Schedule, if a transaction is going to perform read operation on a value, it has to wait until the transaction who is performing write on that value commits.

Example:

For example, let's say we have two transactions Ta and Tb. Tb is going to read the value X after the W(X) of Ta then Tb has to wait for the commit operation of transaction Ta before it reads the X.

```
Ta       Tb
-----    -----
R(X)
W(X)
         W(X)
commit
         R(X)
         W(X)
         commit
```

Recoverable Schedule

In Recoverable schedule, if a transaction is reading a value which has been updated by some other transaction then this transaction can commit only after the commit of other transaction which is updating value.

Example:

Here Tb is performing read operation on X after the Ta has made changes in X using W(X) so Tb can only commit after the commit operation of Ta.

```
Ta        Tb
-----     -----
R(X)
W(X)
          R(X)
          W(X)
          R(X)
commit
          commit
```

DBMS Serializable

When multiple transactions are running concurrently then there is a possibility that the database may be left in an inconsistent state. Serializability is a concept that helps us to check which schedules are serializable.

A serializable schedule is the one that always leaves the database in consistent state.

A serial schedule is always a serializable schedule because in serial schedule, a transaction only starts when the other transaction finished execution. However, a non-serial schedule needs to be checked for Serializability.

A non-serial schedule of n number of transactions is said to be serializable schedule, if it is equivalent to the serial schedule of those n transactions. A serial schedule doesn't allow concurrency, only one transaction executes at a time and the other starts when the already running transaction finished.

Types of Serializability

There are 2 types of serializability,

1. Conflict Serializability
2. View Serializability

Conflict Serializability

Conflict Serializability is one of the types of Serializability, which can be used to check whether a non-serial schedule is conflict serializable or not.

A schedule is called conflict serializable if we can convert it into a serial schedule after swapping its non-conflicting operations.

Conflict Operations

Two operations are said to be in conflict, if they satisfy all the following three conditions:

1. Both the operations should belong to different transactions
2. Both the operations are working on same data item
3. At least one of the operations is a write operation

Examples:

1. Operation W(X) of transaction T1 and operation R(X) of transaction T2 are conflicting operations, because they satisfy all the three conditions mentioned above. They belong to different transactions, they are working on same data item X, one of the operations in write operation.
2. Similarly, Operations W(X) of T1 and W(X) of T2 are conflicting operations.

3. Operations W(X) of T1 and W(Y) of T2 are non-conflicting operations because both the write operations are not working on same data item so these operations don't satisfy the second condition.
4. Similarly, R(X) of T1 and R(X) of T2 are non-conflicting operations because none of them is write operation.
5. Similarly, W(X) of T1 and R(X) of T1 are non-conflicting operations because both the operations belong to same transaction T1.

Conflict Equivalent Schedules

Two schedules are said to be conflict Equivalent if one schedule can be converted into other schedule after swapping non-conflicting operations.

Conflict Serializable Check

Let's check whether a schedule is conflict serializable or not. If a schedule is conflict Equivalent to its serial schedule then it is called Conflict Serializable schedule.

Example:

Let's consider this schedule,

```
T1          T2
-----      ------
R(A)
R(B)
           R(A)
           R(B)
           W(B)
W(A)
```

To convert this schedule into a serial schedule we must have to swap the R(A) operation of transaction T2 with the W(A) operation of transaction T1. However, we cannot swap these two operations because they are conflicting operations, thus we can say that this given schedule is **not Conflict Serializable**.

View Serializability

View Serializability is a process to find out that a given schedule is view serializable or not.

To check whether a given schedule is view serializable, we need to check whether the given schedule is **View Equivalent** to its serial schedule.

View Equivalent

Two schedules T1 and T2 are said to be view equivalent, if they satisfy all the following conditions:

1. **Initial Read:** Initial read of each data item in transactions must match in both schedules. For example, if transaction T1 reads a data item X before transaction T2 in schedule S1 then in schedule S2, T1 should read X before T2.

Read vs Initial Read: You may be confused by the term initial read. Here initial read means the first read operation on a data item, for example, a data item X can be read multiple times in a schedule but the first read operation on X is called the initial read.

2. **Final Write:** Final write operations on each data item must match in both the schedules. For example, a data item X is last written by Transaction T1 in schedule S1 then in S2, the last write operation on X should be performed by the transaction T1.

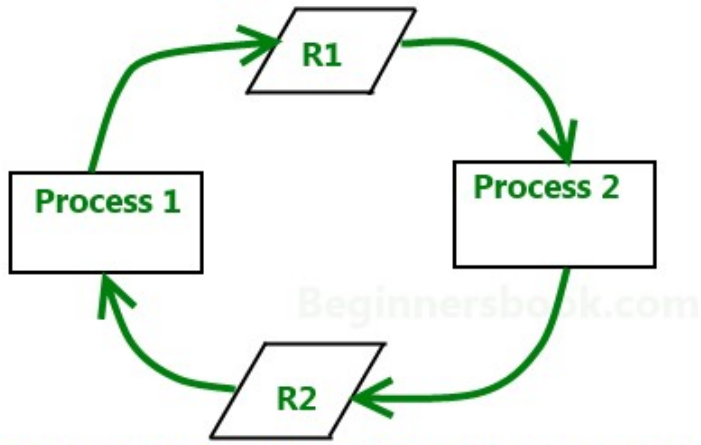
3. **Update Read:** If in schedule S1, the transaction T1 is reading a data item updated by T2 then in schedule S2, T1 should read the value after the write operation of T2 on same data item. For example, in schedule S1, T1 performs a read operation on X after the write operation on X by T2 then in S2, T1 should read the X after T2 performs write on X.

View Serializable

If a schedule is view equivalent to its serial schedule, then the given schedule is said to be View Serializable.

Deadlock

A **deadlock** is a condition wherein two or more tasks are waiting for each other in order to be finished but none of the task is willing to give up the resources that other task needs. In this situation no task ever gets finished and is in waiting state forever.



Coffman Condition

Coffman stated four conditions for a deadlock occurrence. A deadlock may occur if all the following conditions holds true.

- **Mutual exclusion condition**:

  There must be at least one resource that cannot be used by more than one process at a time.

- **Hold and wait condition**:

A process that is holding a resource can request for additional resources that are being held by other processes in the system.

- **No pre-emption condition**:

  A resource cannot be forcibly taken from a process. Only the process can release a resource that is being held by it.

- **Circular wait condition**:

  A condition where one process is waiting for a resource that is being held by second process and second process is waiting for third process ….so on and the last process is waiting for the first process. Thus, making a circular chain of waiting.

Deadlock Handling

Ignore the deadlock (Ostrich Algorithm)

You may be wondering how ignoring a deadlock can come under deadlock handling. But to let you know that the windows you are using on your PC, uses this approach of deadlock handling and that is reason sometimes it hangs up and you have to reboot it to get it working. Not only Windows but UNIX also uses this approach.

The question is why? Why instead of dealing with a deadlock they ignore it why this being called as Ostrich algorithm?

This is known as Ostrich algorithm because in this approach we ignore the deadlock and pretends that it would never occur, just like Ostrich behaviour "to stick one's head in the sand and pretend there is no problem."

Let's discuss why we ignore it: When it is believed that deadlocks are very rare and cost of deadlock handling is higher, in that case ignoring is better solution than handling it. For example: Let's take the operating system example – If the time requires handling the deadlock is higher than the time requires rebooting the windows then rebooting would be a preferred choice considering that deadlocks are very rare in windows.

Deadlock Detection

Resource scheduler is one that keeps the track of resources allocated to and requested by processes. Thus, if there is a deadlock it is known to the resource scheduler. This is how a deadlock is detected.

Once a deadlock is detected it is being corrected by following methods:

- **Terminating processes involved in deadlock**:

  Terminating all the processes involved in deadlock or terminating process one by one until deadlock is resolved can be the solutions but both of these approaches are not good. Terminating all processes cost high and partial work done by processes gets lost. Terminating one by one takes lot of time because each time a process is terminated, it needs to check whether the deadlock is resolved or not. Thus, the best approach is considering process age and priority while terminating them during a deadlock condition.

- **Resource Pre-emption**:

Another approach can be the pre-emption of resources and allocation of them to the other processes until the deadlock is resolved.

Deadlock Prevention

We know that if all 4 Coffman conditions hold true then a deadlock occurs so preventing one or more of them could prevent the deadlock.

Removing mutual exclusion:

All resources must be sharable that means at a time more than one processes can get a hold of the resources. That approach is practically impossible.

Removing hold and wait condition:

This can be removed if the process acquires all the resources that are needed before starting out. Another way to remove this to enforce a rule of requesting resource when there are none in held by the process.

Preemption of resources:

Preemption of resources from a process can result in rollback and thus this needs to be avoided in order to maintain the consistency and stability of the system.

Avoid circular wait condition:

This can be avoided if the resources are maintained in a hierarchy and process can hold the resources in increasing order of precedence. This avoid circular wait. Another way of doing this to force one resource per process rule – A process can request for a resource once it releases the resource currently being held by it. This avoids the circular wait.

Deadlock Avoidance

Deadlock can be avoided if resources are allocated in such a way that it avoids the deadlock occurrence. There are two algorithms for deadlock avoidance.

- Wait/Die
- Wound/Wait

Here is the table representation of resource allocation for each algorithm. Both of these algorithms take process age into consideration while determining the best possible way of resource allocation for deadlock avoidance.

	Wait/Die	Wound/Wait
Older process needs a resource held by younger process	Older process Waits	Younger process Dies

Younger process needs a resource held by older process	Younger process Dies	Younger process Waits

One of the famous deadlock avoidance algorithms is Banker's algorithm.

Concurrency Control in DBMS

When more than one transaction is running simultaneously there are chances of a conflict to occur which can leave database to an inconsistent state. To handle these conflicts, we need concurrency control in DBMS, which allows transactions to run simultaneously but handles them in such a way so that the integrity of data remains intact.

Example:

You and your brother have a joint bank account, from which you both can withdraw money. Now let's say you both go to different branches of the same bank at the same time and try to withdraw 5000 INR, your joint account has only 6000 balance. Now if we don't have concurrency control in place you both can get 5000 INR at the same time but once both the transactions finish the account balance would be -4000 which is not possible and leaves the database in inconsistent state.
We need something that controls the transactions in such a way that allows the transaction to run concurrently but maintaining the consistency of data to avoid such issues.

Solution of Conflicts

Locks

A lock is kind of a mechanism that ensures that the integrity of data is maintained. There are two types of a lock that can be placed while accessing the data so that the concurrent transaction cannot alter the data while we are processing it.

1. Shared Lock (S)
2. Exclusive Lock (X)

Shared Lock

Shared lock is placed when we are reading the data, multiple shared locks can be placed on the data but when a shared lock is placed no exclusive lock can be placed.

Example:

when two transactions are reading Steve's account balance, let them read by placing shared lock but at the same time if another transaction wants to update the Steve's account balance by placing Exclusive lock, do not allow it until reading is finished.

Exclusive Lock

Exclusive lock is placed when we want to read and write the data. This lock allows both the read and write operation, once this lock is placed on the data no other lock (shared or Exclusive) can be placed on the data until Exclusive lock is released.

Example: when a transaction wants to update the Steve's account balance, let it do by placing X lock on it but if a second transaction wants to read the data(S lock) don't allow it, if another transaction wants to write the data(X lock) don't allow that either.

Lock Compatibility Matrix

	S	X
S	True	False
X	False	False

How to read this matrix?

There are two rows, first row says that when S lock is placed, another S lock can be acquired so it is marked true but no Exclusive locks can be acquired so marked False.

In second row, When X lock is acquired neither S nor X lock can be acquired so both marked false.

DBMS Interview Questions

What is the difference between DBMS & RDBMS?

The key difference is that RDBMS (Relational database management system) applications store data in a tabular form, while DBMS applications store data as file.

A DBMS is a system software for creating and managing database. It provides the user and programmer with a systematic way to create, retrieve, update and manage data.

A relational database refers to a database that stores data in a structured format, using rows and columns. This make it easy to locate and access specific values within the database.

Example:

DBMS: File systems, XML

RDBMS: MySQL, SQL Server, Oracle

What is checkpoint in DBMS?

The checkpoint is a type of mechanism where all the previous logs are removed from the system and permanently stored in the storage disk.

When does checkpoint occurs,

A checkpoint is like a snapshot of the DBMS state. Using checkpoints, the DBMS can reduce the amount of work to be done during a restart in the event of subsequent crashes. Checkpoints are used for the recovery of the database after the system crash. Checkpoints are used in the log-based recovery system. When due to a system crash, we need to restart the system then at that point we use checkpoints. So that, we don't have to perform the transactions from the very starting.

What do you mean by transparent DBMS?

The transparent DBMS is a type of DBMS which keeps its physical structure hidden from users. Physical structure or physical storage structure implies to the memory manager of the DBMS, and it describes how the data stored on disk.

What are the unary operations in Relational Algebra?

PROJECTION and SELECTION are the unary operations in relational algebra. Unary operations are those operations which use single operands. Unary operations are SELECTION, PROJECTION, and RENAME.

As in SELECTION relational operators are used for example - =, <=, >=, etc.

How many types of database languages are?

There are 4 types of database language,

1. DDL – Data Definition Language
2. DML – Data Manipulation Language
3. DCL – Data Control Language
4. TCL – Transaction Control Language

what do you understand by data model?

The Data model is specified as a collection of conceptual tools for describing data, data relationships, data semantics and constraints. These models are used to describe the relationship between the entities and their attributes.

There is the number of data models:

- Hierarchical data model
- network model
- relational model
- Entity-Relationship model and so on.

What is a degree of Relation?

The degree of relation is a number of attributes of its relation schema. A degree of relation is also known as Cardinality it is defined as the number of occurrences of one entity which is connected to the number of occurrences of other entity. There are three degree of relation they are one-to-one (1:1), one-to-many(1:M), many-to-one(M:M).

Explain the functionality of DML Compiler?

The DML Compiler translates DML statements in a query language that the query evaluation engine can understand. DML Compiler is required because the DML is the family of syntax

element which is very similar to the other programming language which requires compilation. So, it is essential to compile the code in the language which query evaluation engine can understand and then work on those queries with proper output.

what do you mean by query optimization?

The term query optimization specifies an efficient execution plan for evaluating a query that has the least estimated cost. The concept of query optimization came into the frame when there were a number of methods, and algorithms existed for the same task then the question arose that which one is more efficient and the process of determining the efficient way is known as query optimization.

There are many benefits of query optimization:

- It reduces the time and space complexity.
- More queries can be performed as due to optimization every query comparatively takes less time.
- User satisfaction as it will provide output fast

What is an Entity?

The Entity is a set of attributes in a database. An entity can be a real-world object which physically exists in this world. All the entities have their attribute which in the real world considered as the characteristics of the object.

For example: In the employee database of a company, the employee, department, and the designation can be considered as the entities. These entities have some characteristics which will be the attributes of the corresponding entity.

What is Entity type?

An entity type is specified as a collection of entities, having the same attributes. Entity type typically corresponds to one or several related tables in the database. A characteristic or trait which defines or uniquely identifies the entity is called entity type.

For example, a student has student_id, department, and course as its characteristics.

What is Entity Set?

The entity set specifies the collection of all entities of a particular entity type in the database. An entity set is known as the set of all the entities which share the same properties.

For example, a set of people, a set of students, a set of companies, etc.

What is weak entity set?

An entity set that doesn't have sufficient attributes to form a primary key is referred to as a weak entity set. The member of a weak entity set is known as a subordinate entity. Weak entity set does not have a primary key, but we need a mean to differentiate among all those entries in the entity set that depend on one particular strong entity set.

What is an attribute?

An attribute refers to a database component. It is used to describe the property of an entity. An attribute can be defined as the characteristics of the entity. Entities can be uniquely identified using the attributes. Attributes represent the instances in the row of the database.

For example: If a student is an entity in the table, then age will be the attribute of that student.

What are the integrity rules in DBMS?

Data integrity is one significant aspect while maintaining the database. So, data integrity is enforced in the database system by imposing a series of rules. Those set of integrity is known as the integrity rules.

There are two integrity rules in DBMS:

Entity Integrity: It specifies that "Primary key cannot have a NULL value."

Referential Integrity: It specifies that "Foreign Key can be either a NULL value or should be the Primary Key value of other relation.

What do you mean by extension and intension?

Extension: The Extension is the number of tuples present in a table at any instance. It changes as the tuples are created, updated and destroyed. The actual data in the database change quite frequently. So, the data in the database at a particular moment in time is known as extension or database state or snapshot. It is time dependent.

Intension: Intension is also known as Data Schema and defined as the description of the database, which is specified during database design and is expected to remain unchanged. The Intension is a constant value that gives the name, structure of tables and the constraints laid on it.

What is system R? How many of its 2 major subsystems?

System R was designed and developed from 1974 to 1979 at IBM San Jose Research Centre. System R is the first implementation of SQL, which is the standard relational data query language, and it was also the first to demonstrate that RDBMS could provide better transaction processing performance. It is a prototype which is formed to show that it is possible to build a Relational System that can be used in a real-life environment to solve real-life problems.

Following are two major subsystems of System R:

- Research Storage
- System Relational Data System

What is Data Independence?

Data independence specifies that "the application is independent of the storage structure and access strategy of data." It makes you able to modify the schema definition at one level without altering the schema definition in the next higher level.

It makes you able to modify the schema definition in one level should not affect the schema definition in the next higher level.

There are two types of Data Independence:

Physical Data Independence: Physical data is the data stored in the database. It is in the bit-format. Modification in physical level should not affect the logical level.

For example: If we want to manipulate the data inside any table that should not change the format of the table.

Logical Data Independence: Logical data in the data about the database. It basically defines the structure. Such as tables stored in the database. Modification in logical level should not affect the view level.

For example: If we need to modify the format of any table, that modification should not affect the data inside it.

What is the difference between a DELETE command and TRUNCATE command?

DELETE command: DELETE command is used to delete rows from a table based on the condition that we provide in a WHERE clause.

- DELETE command delete only those rows which are specified with the WHERE clause.
- DELETE command can be rolled back.
- DELETE command maintain a log, that's why it is slow.
- DELETE use row lock while performing DELETE function.

TRUNCATE command: TRUNCATE command is used to remove all rows (complete data) from a table. It is similar to the DELETE command with no WHERE clause.

- The TRUNCATE command removes all the rows from the table.
- The TRUNCATE command cannot be rolled back.
- The TRUNCATE command doesn't maintain a log. That's why it is fast.
- TRUNCATE use table log while performing the TRUNCATE function.

How do you communicate with an RDBMS?

You have to use Structured Query Language (SQL) to communicate with the RDBMS. Using queries of SQL, we can give the input to the database and then after processing of the queries database will provide us the required output.

What is the difference between a shared lock and exclusive lock?

Shared lock: Shared lock is required for reading a data item. In the shared lock, many transactions may hold a lock on the same data item. When more than one transaction is allowed to read the data items then that is known as the shared lock.

Exclusive lock: When any transaction is about to perform the write operation, then the lock on the data item is an exclusive lock. Because, if we allow more than one transaction then that will lead to the inconsistency in the database.