Define the following terms.

Data

- Fact that can be recorded or stored.
- E.g. Person Name, Age, Gender and Weight etc.

Information

- When data is processed, organized, structured or presented in a given context so as to make it useful, it is called information.

Database

- A Database is a collection of inter-related (logically-related) data.
- E.g. Books Database in Library, Student Database in University etc.

DBMS (Database Management System)

- A database management system is a collection of inter-related data and set of programs to manipulate those data.
- DBMS = Database + Set of programs
- E.g. MS SQL Server, Oracle, My SQL, SQLite, MongoDB etc.

Metadata

- Metadata is data about data.
- Data such as table name, column name, data type, authorized user and user access privileges for any table is called metadata for that table.

Data dictionary

- Data dictionary is an information repository which contains metadata.
- It is usually a part of the system catalog.

Data warehouse

- Data warehouse is an information repository which stored data.
- It is design to facilitate reporting and analysis.

Field

- A field is a character or group of characters that have a specific meaning.
- It is also called a data item. It is represented in the database by a value.
- For Example customer id, name, society and city are all fields for customer Data.

Record

- A record is a collection of logically related fields.
- For examples, collection of fields (id, name, address & city) forms a record for customer.

Explain disadvantages of file system (file processing systems) compare to Database management system. *OR* Explain disadvantages of conventional file-based system compared to Database management system.

Data Redundancy

- It is possible that the same information may be duplicated in different files. This leads to data redundancy.

- Data redundancy results in memory wastage.
- For example, consider that some customers have both kinds of accounts - saving and current. In this case, data about customers such as name, address, e-mail and contact number will be duplicated in both files, saving accounts file and current account file.
- In other words, same information will be stored in two different locations (files). And, it wastes memory.

Data Inconsistency

- Due to data redundancy, it is possible that data may not be in consistent state.
- For example, consider that an address of some customer changes. And, that customer has both kinds of accounts. Now, it is possible that this changed address is updated in only one file, leaving address in other file as it is. As a result of this, same customer will have two different addresses in two different files, making data inconsistent.

Difficulty in Accessing Data

- Accessing data is not convenient and efficient in file processing system.
- For example, suppose, there is a program to find information about all customers. But, what if there is a need to find out all customers from some particular city. In this case, there are two choices here: One, find out all customers using available program, and then extract the needed customers manually. Second, develop new program to get required information. Both options are not satisfactory.
- For each and every different kind of data access, separate programs are required. This is neither convenient nor efficient.

Limited Data Sharing

- Data are scattered in various files.
- Different files may have different formats. And these files may be stored in different folders (directories) may be of different computers of different departments.
- So, due to this data isolation, it is difficult to share data among different applications.

Integrity Problems

- Data integrity means that the data contained in the database is both correct and consistent. For this purpose, the data stored in database must satisfy certain types of constraints (rules).
- For example, a balance for any account must not be less than zero. Such constraints are enforced in the system by adding appropriate code in application programs. But, when new constraints are added, such as balance should not be less than Rs. 5000, application programs need to be changed. But, it is not an easy task to change programs whenever required.

Atomicity Problems

- Any operation on database must be atomic. This means, operation completes either 100% or 0%.
- For example, a fund transfer from one account to another must happen in its entirely. But, computer systems are vulnerable to failure, such as system crash, virus attack. If a system failure occurs during the execution of fund transfer operation, it may possible that

amount to be transferred, say, Rs. 500, is debited from one account, but is not credited to another account.

- This leaves database in consistent state. But, it is difficult to ensure atomicity in a file processing system.

Concurrent Access Anomalies

- Multiple users are allowed to access data simultaneously (concurrently). This is for the sake of better performance and faster response.
- Consider an operation to debit (withdrawal) an account. The program reads the old balance, calculates the new balance, and writes new balance back to database. Suppose an account has a balance of Rs. 5000. Now, a concurrent withdrawal of Rs. 1000 and Rs. 2000 may leave the balance Rs. 4000 or Rs. 3000 depending upon their completion time rather than the correct value of Rs. 2000.
- Here, concurrent data access should be allowed under some supervision.
- But, due to lack of co-ordination among different application programs, this is not possible in file processing systems.

Security Problems

- Database should be accessible to users in a limited way.
- Each user should be allowed to access data concerning his application only.
- For example, a customer can check balance only for his/her own account. He/She should not have access for information about other accounts.
- But, in file processing system, application programs are added in an ad hoc manner by different programmers. So, it is difficult to enforce such kind of security constraints.

Explain advantages (benefits) of DBMS over file management system. *OR* Explain purpose of database system.

Minimal Data Redundancy (Duplication)

- Due to centralized database, it is possible to avoid unnecessary duplication of information.
- This leads to reduce data redundancy.
- It prevents memory wastage and reduces extra processing time to get required data.

Shared Data

- All authorized user and application program can share database easily.

Data Consistency

- Data inconsistency occurs due to data redundancy.
- With reduced data redundancy such type of data inconsistency can be eliminated.
- This results in improved data consistency.

Data Access

- DBMS utilizes a variety of techniques to retrieve data.
- Required data can be retrieved by providing appropriate query to the DBMS.
- Thus, data can be accessed in convenient and efficient manner.

Data Integrity

- Data in database must be correct and consistent.
- So, data stored in database must satisfy certain types of constraints (rules).
- DBMS provides different ways to implement such type of constraints (rules).
- This improves data integrity in a database.

Data Security

- Database should be accessible to user in a limited way.
- DBMS provides way to control the access to data for different user according to their requirement.
- It prevents unauthorized access to data.
- Thus, security can be improved.

Concurrent Access

- Multiple users are allowed to access data simultaneously.
- Concurrent access to centralized data can be allowed under some supervision.
- This results in better performance of system and faster response.

Guaranteed Atomicity

- Any operation on database must be atomic. This means, operation must be executed either 100% or 0%.
- This type of atomicity is guaranteed in DBMS.

List and explain the applications of DBMS.

Airlines and railways

- Airlines and railways use online databases for reservation, and for displaying the schedule information.

Banking

- Banks use databases for customer inquiry, accounts, loans, and other transactions.

Education

- Schools and colleges use databases for course registration, result, and other information.

Telecommunications

- Telecommunication departments use databases to store information about the communication network, telephone numbers, record of calls, for generating monthly bills, etc.

Credit card transactions

- Databases are used for keeping track of purchases on credit cards in order to generate monthly statements.

E-commerce

- Integration of heterogeneous information sources (for example, catalogs) for business activity such as online shopping, booking of holiday package, consulting a doctor, etc.

Health care information systems and electronic patient record

- Databases are used for maintaining the patient health care details in hospitals.

Digital libraries and digital publishing

- Databases are used for management and delivery of large bodies of textual and multimedia data.

Finance

- Databases are used for storing information such as sales, purchases of stocks and bonds or data useful for online trading.

Sales

- Databases are used to store product, customer and transaction details.

Human resources

- Organizations use databases for storing information about their employees, salaries, benefits, taxes, and for generating salary checks.

Describe functions (responsibility, roles, and duties) of DBA to handle DBMS.

DBA

- The full name of DBA is Database Administrator.
- Database Administrator is a person in the organization who controls the design and the use of database.

Functions or Responsibilities of DBA are as under:

Schema Definition

- DBA defines the logical schema of the database.
- A schema refers to the overall logical structure of the database.
- According to this schema, database will be designed to store required data for an organization.

Storage Structure and Access Method Definition

- DBA decides how the data is to be represented in the database.
- Based on this, storage structure of the database and access methods of data is defined.

Defining Security and Integrity Constraints

- DBA decides various security and integrity constraints.
- DDL (Data Definition Language) provides facilities to specifying such constraints.

Granting of Authorization for Data Access

- The DBA determines which user needs access to which part of the database.
- According to this, various types of authorizations (permissions) are granted to different users.
- This is required to prevent unauthorized access of a database.

Liaison with Users

- DBA is responsible to provide necessary data to user.
- User should be able to write the external schema, using DDL (Data Definition Language).

Assisting Application Programmers

- DBA provides assistance to application programmers to develop application programs.

Monitoring Performance

- The DBA monitors performance of the system.
- The DBA ensures that better performance is maintained by making change in physical or logical schema if required.

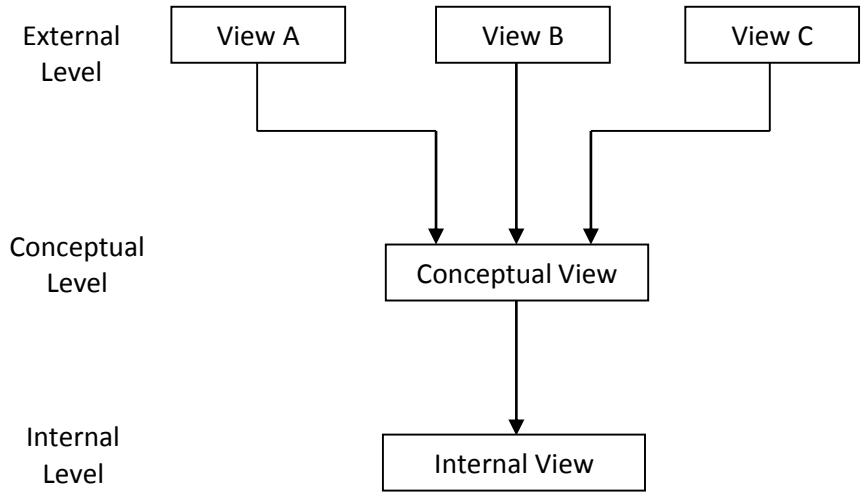
Backup and Recovery

- Database should not be lost or damaged.
- The task of DBA is to backing up the database on some storage devices such as DVD, CD or Magnetic Tape or remote servers.
- In case of failures, such as flood or virus attack, Database is recovered from this backup.

Explain three levels ANSI SPARC Database System. *OR* Explain three level Data abstraction.

The ANSI SPARC architecture divided into three levels:

1) External level
2) Conceptual level
3) Internal level



Three levels ANSI SPARC Database System

Internal Level

- This is the lowest level of the data abstraction.
- It describes **how** the data are actually stored on storage devices.
- It is also known as a **physical level**.
- The internal view is described by internal schema.
- Internal schema consists of definition of stored record, method of representing the data field and access method used.

Conceptual Level

- This is the next higher level of the data abstraction.

- It describes **what** data are stored in the database and what relationships exist among those data.
- It is also known as a **logical level**.
- Conceptual view is defined by conceptual schema. It describes all records and relationship.

External Level

- This is the highest level of data abstraction.
- It is also known as **view level**.
- It describes only part of the entire database that a particular end user requires.
- External view is describes by external schema.
- External schema consists of definition of logical records, relationship in the external view and method of deriving the objects from the conceptual view.
- This object includes entities, attributes and relationship.

Explain Mapping. *OR* Explain external and internal mapping. *OR* What is mapping? Describe type of mapping.

Mapping

- The process of transforming requests and results between the three levels is called mapping.

Types of Mapping

- Conceptual/Internal Mapping
- External/Conceptual Mapping

Conceptual/Internal Mapping

- It relates conceptual schema with internal schema.
- It defines correspondence between the conceptual schema and the database stored in physical devices.
- It specifies how conceptual records and fields are presented at the internal level.
- If the structure of stored database is changed, then conceptual/internal mapping must be changed accordingly and conceptual schema can remain invariant.
- There could be one mapping between conceptual and internal levels.

External/Conceptual Mapping

- It relates each external schema with conceptual schema.
- It defines correspondence between a particular external view and conceptual schema.
- If the structure of conceptual schema is changed, then external/conceptual mapping must be changed accordingly and external schema can remain invariant.
- There could be several mappings between external and conceptual levels.

Explain Data Independence.

Data Independence

- Data independency is the ability to modify a schema definition in one level without affecting a schema definition in the next higher level.

Types of data independence

- Physical data independence
- Logical data independence

Physical data independence

- Physical data independence allows changing in physical storage devices or organization of file without change in the conceptual view or external view.
- Modifications at the internal level are occasionally necessary to improve performance.
- Physical data independence separates conceptual level from the internal level.
- It is easy to achieve physical data independence.

Logical data independence

- Logical data independence is the ability to modify the conceptual schema without requiring any change in application programs.
- Conceptual schema can be changed without affecting the existing external schema.
- Modifications at the logical level are necessary whenever the logical structure of the database is altered.
- Logical data independence separates external level from the conceptual view.
- It is difficult to achieve logical data independence.

Explain Data Abstraction in DBMS.

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

Explain different database users.

There are four different database users.

Application programmers

- These users are computer professionals who write application programs using some tools. E.g. Software developers

Sophisticated users

- These users interact with system without writing program. They form their request in a database query language. E.g. Analyst.

Specialized users

- These users write specialized database applications that do not fit into the traditional data processing framework. E.g. Database Administrator.

Naive users

- These users are unsophisticated users who have very less knowledge of database system.

- These users interact with the system by using one of the application programs that have been written previously.
- Examples, e.g. Clerk in bank

Differentiate the DA and DBA.

DA (Data Administrator)	*DBA (Database Administrator)*
The data administrator is a person in the organization who controls the data of the database.	The database administrator is a person in the organization who controls the design and the use of the database.
DA determines what data to be stored in database based on requirements of the organization.	DBA provides necessary technical support for implementing a database.
DA is involved more in the requirements gathering, analysis, and design phases.	DBA is involved more in the design, development, testing and operational phases.
DA is a manager or some senior level person in an organization who understands organizational requirements with respect to data.	DBA is a technical person having knowledge of database technology.
DA does not need to be a technical person, but any kind of knowledge about database technology can be more beneficiary.	DBA does not need to be a business person, but any kind of knowledge about a functionality of an organization can be more beneficiary.
DA is a business focused person, but, he/she should understand more about the database technology.	DBA is a technically focused person, but, he/she should understand more about the business to administer the databases effectively.

Explain Database System Architecture.

Components of a DBMS

These functional units of a database system can be divided into two parts:

1. Query Processor Units (Components)
2. Storage Manager Units

Query Processor Units:

Query processor unit deal with execution of DDL (Data Definition Language) and DML (Data Manipulation Language) statements.

- **DDL Interpreter —** Interprets DDL statements into a set of tables containing metadata.
- **DML Compiler —** Translates DML statements into low level instructions that the query evaluation engine understands.
- **Embedded DML Pre-compiler —** Converts DML statements embedded in an application program into normal procedure calls in the host language.
- **Query Evaluation Engine —** Executes low level instructions generated by DML compiler.

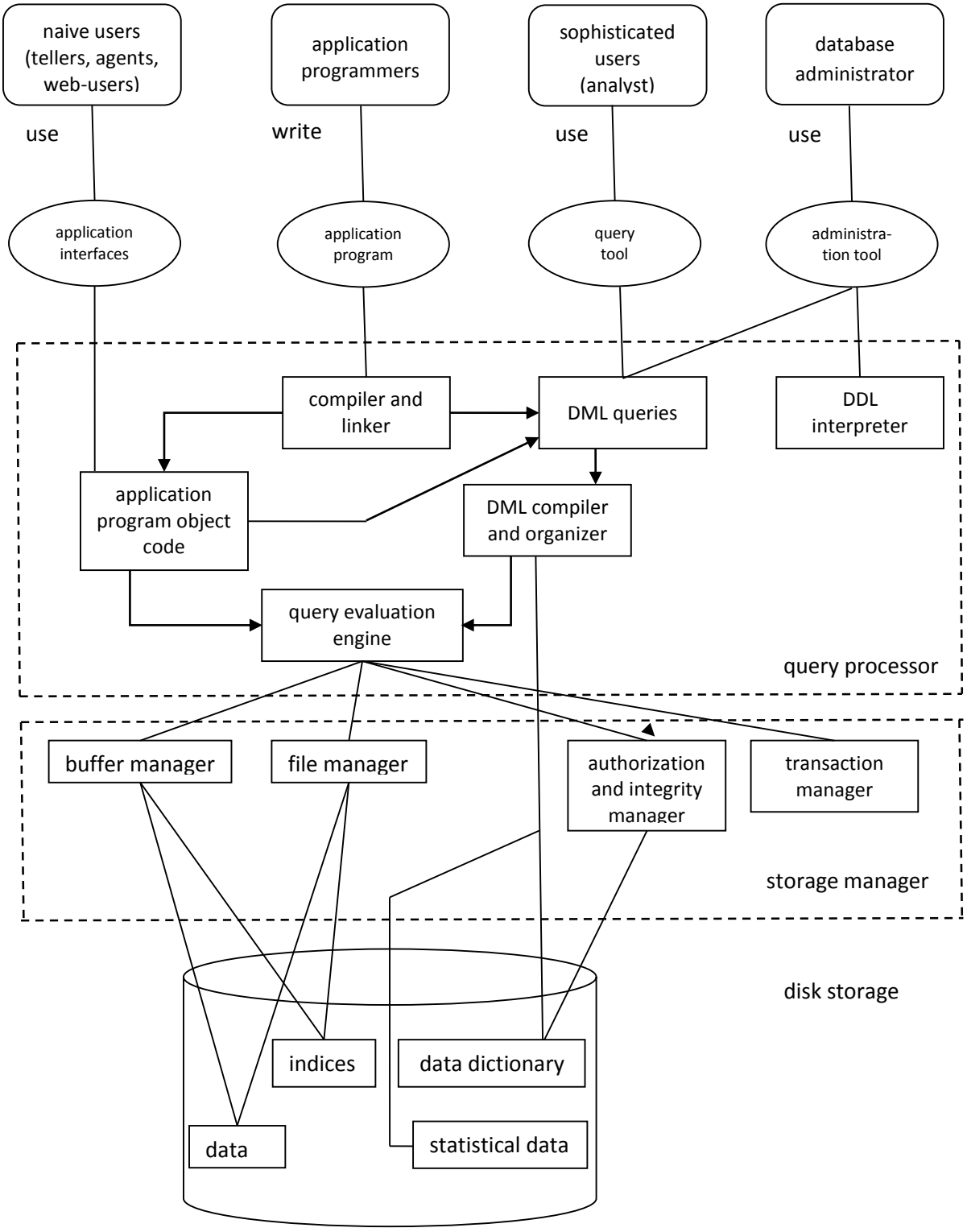
Storage Manager Units:

Storage manager units provide interface between the low level data stored in database and the application programs & queries submitted to the system.

- **Authorization Manager** — Checks the authority of users to access data.
- **Integrity Manager** — Checks for the satisfaction of the integrity constraints.
- **Transaction Manager** — Preserves atomicity and controls concurrency.
- **File Manager** — Manages allocation of space on disk storage.
- **Buffer Manager** — Fetches data from disk storage to memory for being used.

In addition to these functional units, several data structures are required to implement physical storage system. These are described below:

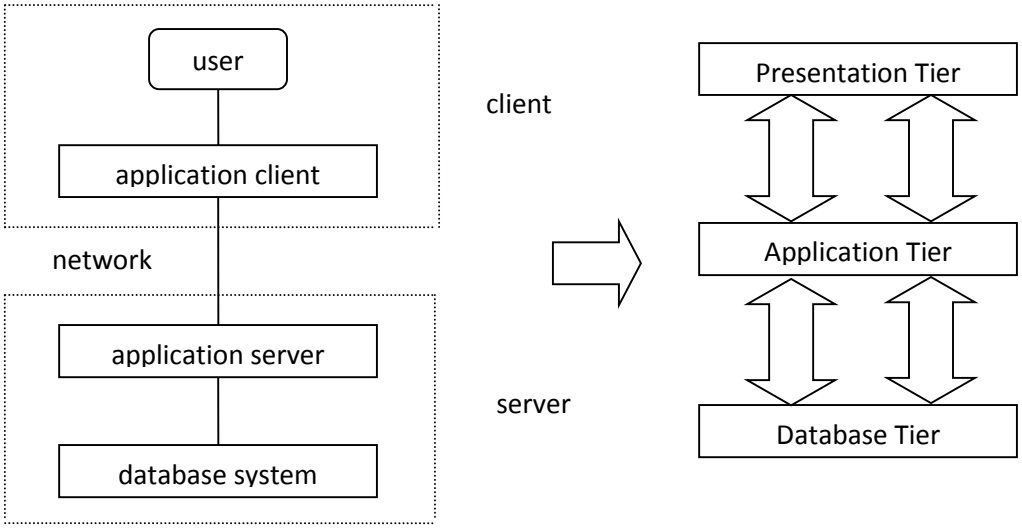
- **Data Files** — To store user data.
- **Data Dictionary and System Catalog** — To store metadata. It is used heavily, almost for each and every data manipulation operation. So, it should be accessed efficiently.
- **Indices** — To provide faster access to data items.
- **Statistical Data** — To store statistical information about the data in the database. This information is used by the query processor to select efficient ways to execute a query.



Database System Architecture

Explain database system 3 tier architecture with clear diagram in detail.

- Most widely used architecture is 3-tier architecture.
- 3-tier architecture separates it tier from each other on basis of users.



2 tier v/s 3 tier architecture

Database (Data) Tier

- At this tier, only database resides.
- Database along with its query processing languages sits in layer-3 of 3-tier architecture.
- It also contains all relations and their constraints.

Application (Middle) Tier

- At this tier, the application server and program, which access database, resides.
- For a user this application tier works as abstracted view of database.
- Users are unaware of any existence of database beyond application.
- For database-tier, application tier is the user of it.
- Database tier is not aware of any other user beyond application tier.
- This tier works as mediator between the two.

User (Presentation) Tier

- An end user sits on this tier.
- From a user's aspect, this tier is everything.
- He/she doesn't know about any existence or form of database beyond this layer.
- At this layer multiple views of database can be provided by the application.
- All views which are generated by an application, resides in application tier.

Define the following terms.

Entity

- An entity is a thing or object or person in the real world that is distinguishable from all other object.
- E.g. book, student, employee, college etc…

Entity sets

- An entity set is a set of entities of same type that share the same properties or attributes.
- E.g. the set of all students in a college can be defined as entity set student.

Relationship

- Relationship is an association (connection) between several entities.
- Relationship between 2 entities is called binary relationship.
- E.g. book is issued by student where book and student are entities and issue is relation.

Relationship set

- Relationship set is a set of relationships of the same type.

Attributes

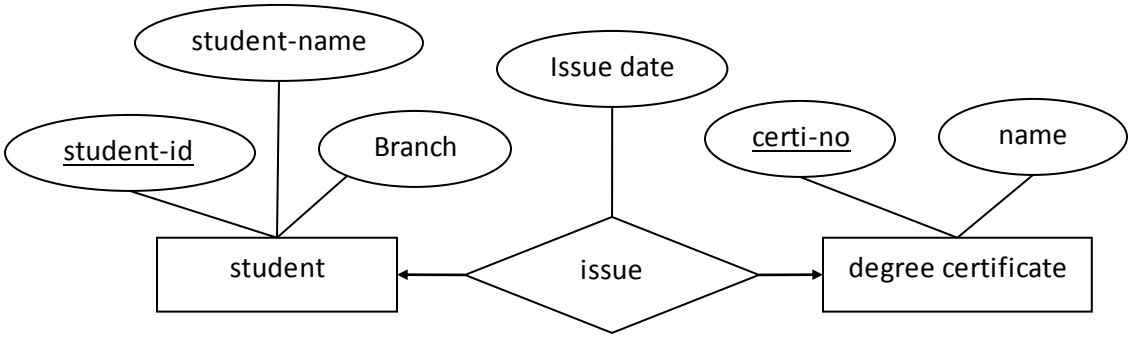
- Attributes are properties hold by each member of an entity set.
- E.g. entity is student and attributes of student are enrollmentno, name, address, cpi etc

Types of attributes

- **Simple attribute:** It cannot be divided into subparts. E.g. cpi, rollno
- **Composite attribute**: It can be divided into subparts. E.g. name (first-name, middle-name, last-name), address.
- **Single valued attribute:** It has single data value. E.g. enrollmentno, birthdate
- **Multi valued attribute**: It has multiple data value. E.g. phoneno (may have multiple phones).
- **Stored attribute:** It's value is stored manually in database. E.g. birthdate
- **Derived attribute**: It's value is derived or calculated from other attributes. E.g. age (can be calculated using current date and birthdate).

Descriptive attributes

- A relationship may also have attributes like an entity. These attributes are called descriptive attributes.
- E.g. Student gets degree certificate on 14th March 2011. Student and Degree are entities, gets certificate is relation and certificate date is an attribute of relationship.

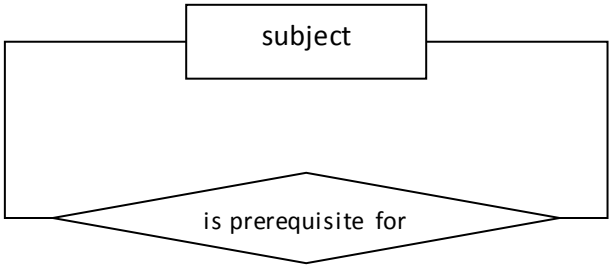


Recursive relationship set

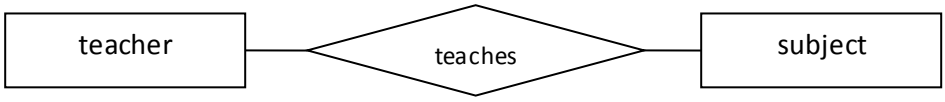
- The same entity set participates in a relationship set more than once then it is called recursive relationship set.
- E.g. an employee entity participated in relationship under with department entity as an employee as well manager also.

Degree of relationship

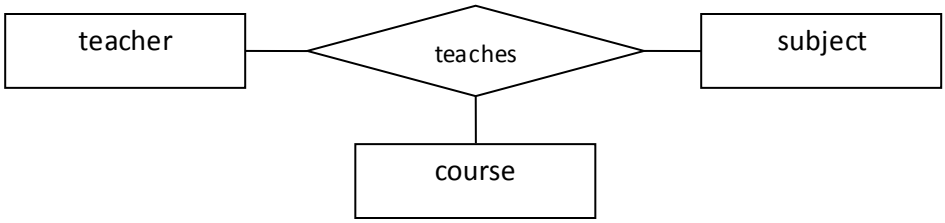
- The degree of a relationship is the number of entity types that participate in the relationship.
- The three most common relationships in ER models are Unary, Binary and Ternary.
- A unary relationship is when both participant entities in the relationship are the same entity.
- Example: Subjects may be prerequisites for other subjects.



- A binary relationship is the relationship between two different entities.
- The University might need to record which teachers taught which subjects.



- A ternary relationship is the relationship between three different entities.
- The University might need to record which teachers taught which subjects in which courses.

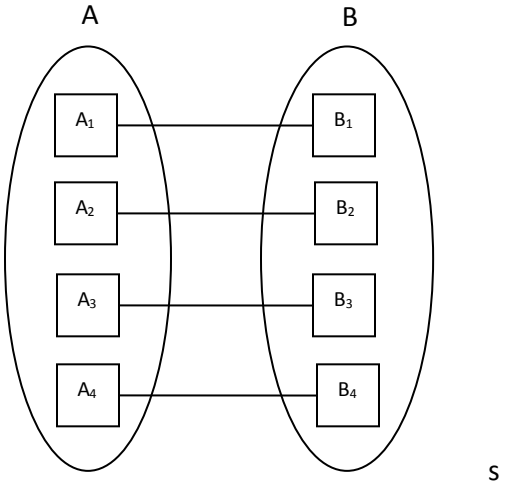


Explain various mapping cardinality (cardinality constraint).

Mapping cardinality (cardinality constraints)

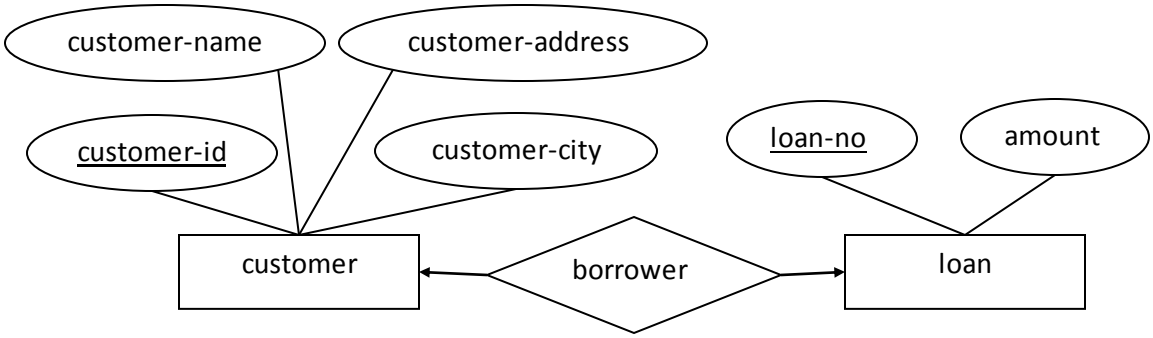
- It represents the number of entities of another entity set which are connected to an entity using a relationship set.
- It is most useful in describing binary relationship sets.
- For a binary relationship set the mapping cardinality must be one of the following types:
  - ✓ One to one
  - ✓ One to many
  - ✓ Many to one
  - ✓ Many to many

One-to-one relationship



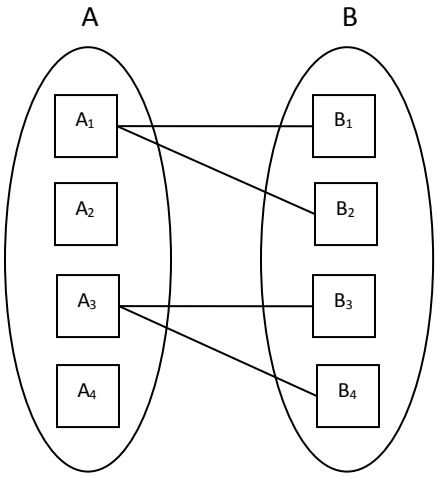
s

- An entity in A is associated with at most (only) one entity in B and an entity in B is associated with at most (only) one entity in A.

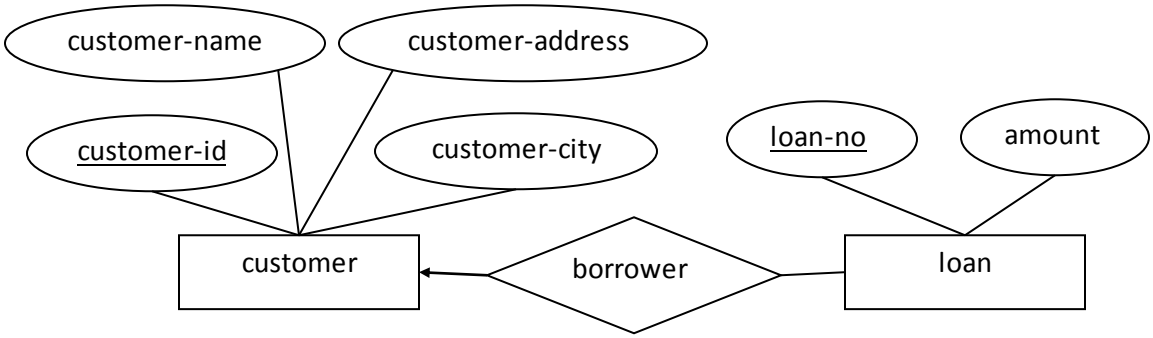


- A customer is connected with only one loan using the relationship borrower and a loan is connected with only one customer using borrower.

One-to-many relationship

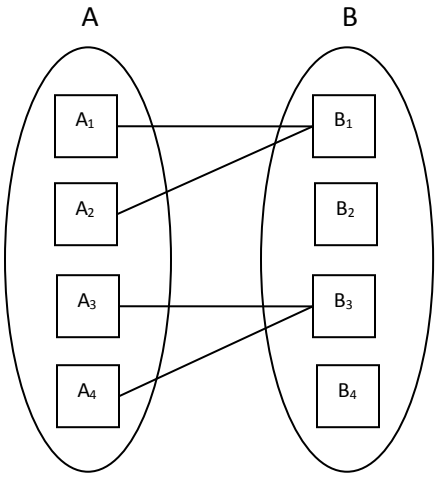


- An entity in A is associated with any number (zero or more) of entities in B and an entity in B is associated with at most (only) one entity in A.

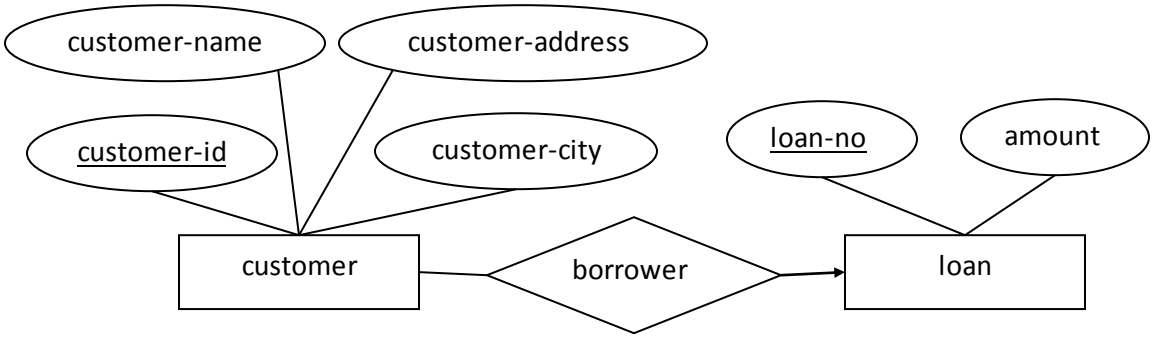


- In the one-to-many relationship a loan is connected with only one customer using borrower and a customer is connected with more than one loans using borrower.

Many-to-one relationship

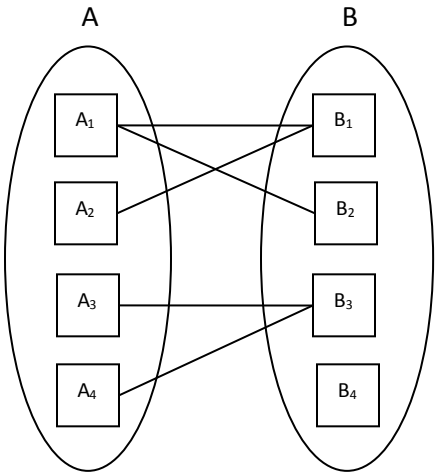


- An entity in A is associated with at most (only) one entity in B and an entity in B is associated with any number (zero or more) of entities in A.

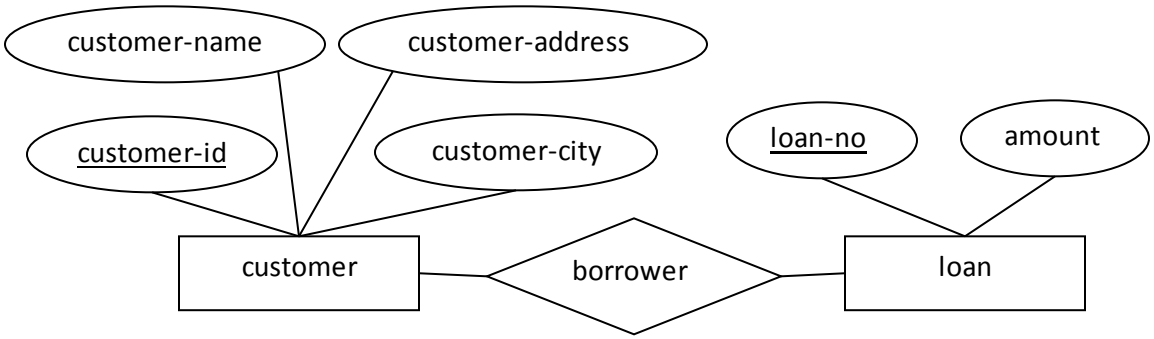


- In a many-to-one relationship a loan is connected with more than one customer using borrower and a customer is connected with only one loan using borrower.

Many-to-many relationship



- An entity in A is associated with any number (zero or more) of entities in B and an entity in B is associated with any number (zero or more) of entities in A.



- A customer is connected with more than one loan using borrower and a loan is connected with more than one customer using borrower.

Explain various participation constraints.

Participation constraints

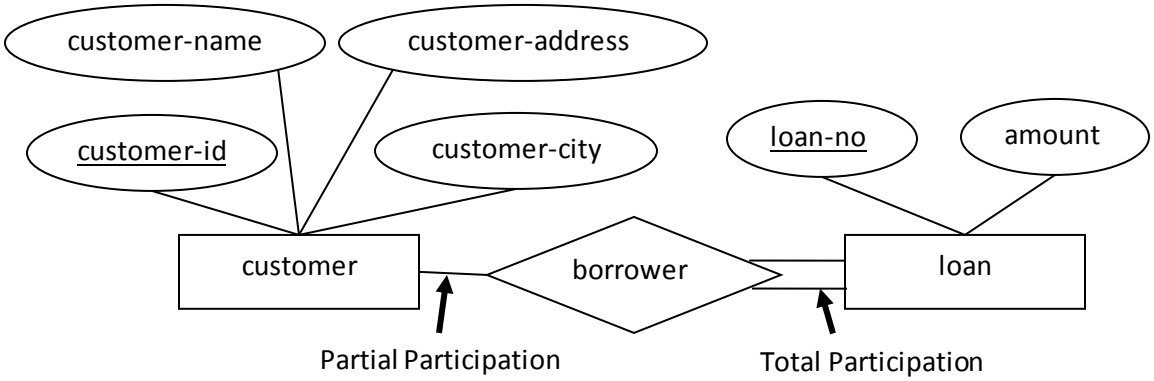
- It specifies the participation of an entity set in a relationship set.
- There are two types participation constraints
  - ✓ Total participation
  - ✓ Partial participation

Total participation

- In total participation, every entity in the entity set participates in at least one relationship in the relationship set.
- It specifies that every entity in super class must be member of some of its sub class.
- E.g. participation of loan in borrower is total because every loan must be connected to a customer using borrower.
- It is indicated by double line.

Partial participation

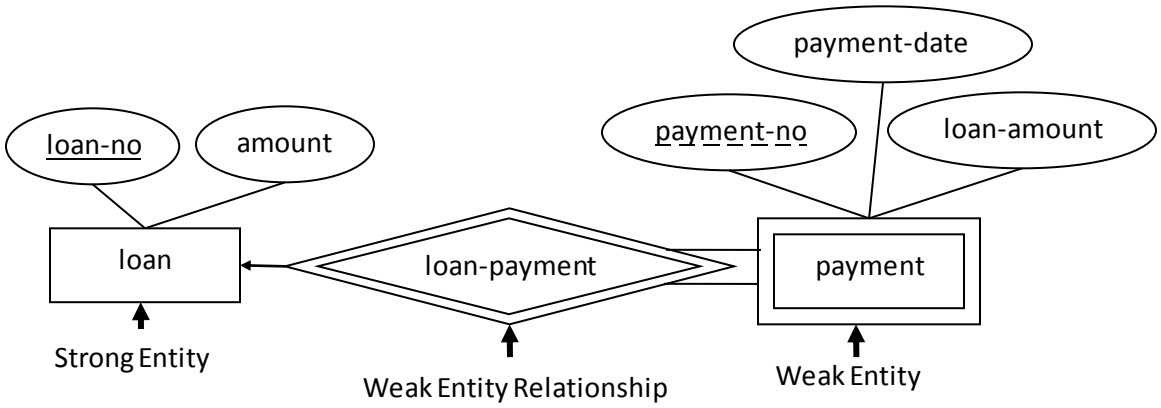
- In partial participation, some entities may not participate in any relationship in the relationship set.
- It specifies that an entity may not belong to any sub class.
- E.g. participation of customer in borrower is partial because every customer is not connected to loan using borrower.
- It is indicated by single line.



Explain weak entity set with the help of example.

Weak entity set

- An entity set that does not have a primary key is called weak entity set.
- The existence of a weak entity set depends on the existence of a strong entity set.
- Weak entity set is indicated by double rectangle.
- Weak entity relationship set is indicated by double diamond.
- The discriminator (partial key) of a weak entity set is the set of attributes that distinguishes between all the entities of a weak entity set.
- The primary key of a weak entity set is created by combining the primary key of the strong entity set on which the weak entity set is existence dependent and the weak entity set's discriminator.
- We underline the discriminator attribute of a weak entity set with a dashed line.
- E.g. in below fig. there are two entities loan and payment in which loan is strong entity set and payment is weak entity set.
- Payment entity has payment-no which is discriminator.
- Loan entity has loan-no as primary key.
- So primary key for payment is (loan-no, payment-no).



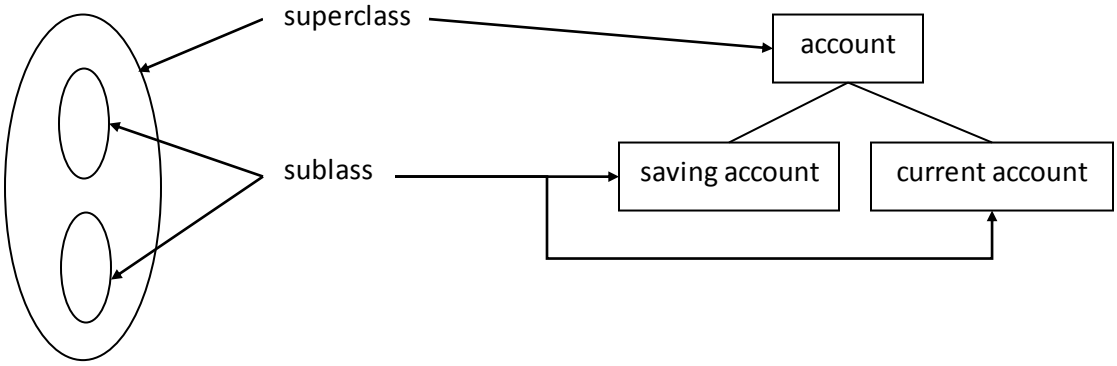
Explain the Superclass and Subclass in E-R diagram with the help of example.

Superclass

- A superclass is an entity from which another entity can be derived.
- A superclass is a generic entity set which has a relationship with one or more subclasses.
- For example, an entity set account has two subsets saving_account and current_account. So an account is superclass.
- Each member of subclass is also a member of superclass. So any saving account or a current account is a member of entity set account.

Subclass

- A subclass is an entity that is derived from another entity.
- A class is a subset of entities in an entity set which has attributes distinct from those in other subset.
- For example, entities of the entity set account are grouped in to two classes saving_account and current_account. So saving_account and current_account are subclasses.



Explain the difference between Specialization and Generalization in E-R diagram.

Specialization	*Generalization*
It will work in Top-down approach.	It will work in Bottom-up approach
The process of creating sub-groupings within an entity set is called specialization.	The process of creating groupings from various entity sets is called generalization.
Specialization is a process of taking a sub set of higher level entity set to form a lower-level entity set.	Generalization is a process of taking the union of two or more lower-level entity sets to produce a higher-level entity set.
Specialization starts from a single entity set; it creates different low-level entity set using some different features.	Generalization starts from the number of entity sets and creates high-level entity set using some common features.

Explain Specialization and Generalization in E-R diagram with example.

- For all practical purposes, generalization and specialization is inversion of each other.

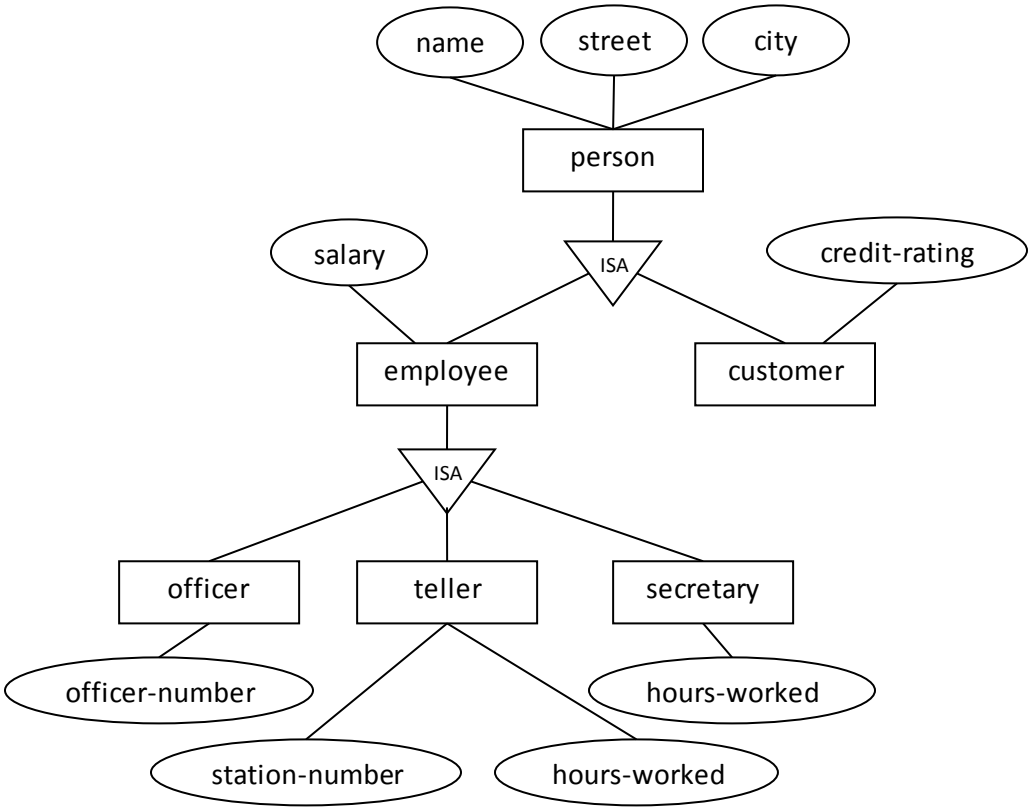
Specialization

- A top-down design process that creates subclasses based on some different characteristics of the entities in the superclass.
- An entity set may include sub groupings of entities that are distinct in some way from other entities in the set.
- For example, a subset of entities within an entity set may have attributes that are not shared by all the entities in the entity set.
- Consider an entity set person, with attributes name, street and city.
- A person may be further classified as one of the following:
  - ✓ customer
  - ✓ Employee
- For example, customer entities may be described further by the attribute customer-id, credit-rating and employee entities may be described further by the attributes employee-id and salary.
- The process of designating sub groupings within an entity set is called specialization. The specialization of person allows us to distinguish among persons according to whether they are employees or customers.
- Now again, employees may be further classified as one of the following:
  - ✓ officer
  - ✓ teller
  - ✓ secretary
- Each of these employee types is described by a set of attributes that includes all the attributes of entity set employee plus additional attributes.
- For example, officer entities may be described further by the attribute office-number, teller entities by the attributes station-number and hours-per-week, and secretary entities by the attribute hours-per-week.
- In terms of an E-R diagram, specialization is depicted by a triangle component labeled ISA.
- The label ISA stands for “is a” and represents, for example, that a customer “is a” person.
- The ISA relationship may also be referred to as a superclass subclass relationship.

Generalization

- A bottom-up design process that combines number of entity sets that have same features into a higher-level entity set.
- The design process proceed in a bottom-up manner, in which multiple entity sets are synthesized into a higher level entity set on the basis of common features.

- The database designer may have to first identify a customer entity set with the attributes name, street, city, and customer-id, and an employee entity set with the attributes name, street, city, employee-id, and salary.
- But customer entity set and the employee entity set have some attributes common. This commonality can be expressed by generalization, which is a containment relationship that exists between a higher level entity set and one or more lower level entity sets.
- In our example, person is the higher level entity set and customer and employee are lower level entity sets.
- Higher level entity set is called superclass and lower level entity set is called subclass. So the person entity set is the superclass of two subclasses customer and employee.
- Differences in the two approaches may be characterized by their starting point and overall goal.



Explain types of constraints on specialization and Generalization.

- There are mainly two types of constraints apply to a specialization/generalization:

Disjoint Constraint

- Describes relationship between members of the superclass and subclass and indicates whether member of a superclass can be a member of one, or more than one subclass.

- It may be disjoint or non-disjoint (overlapping).

1. ***Disjoint Constraint***
   - It specifies that the subclasses of the specialization must be disjointed (an entity can be a member of only one of the subclasses of the specialization).
   - Specified by 'd' in EER diagram or by writing disjoint.
2. ***Non-disjoint (Overlapping)***
   - It specifies that is the same entity may be a member of more than one subclass of the specialization.
   - Specified by 'o' in EER diagram or by writing overlapping.

Participation Constraint

- Determines whether every member in super class must participate as a member of a subclass or not.
- It may be total (mandatory) or partial (optional).

1. ***Total (Mandatory)***
   - Total specifies that every entity in the superclass must be a member of some subclass in the specialization.
   - Specified by a double line in EER diagram.
2. ***Partial (Optional)***
   - Partial specifies that every entity in the super class not belong to any of the subclass of specialization.
   - Specified by a single line in EER diagram.
   - Based on these two different kinds of constraints, a specialization or generalization can be one of four types
     - ✓ Total, Disjoint
     - ✓ Total, Overlapping
     - ✓ Partial, Disjoint
     - ✓ Partial, Overlapping.

Explain aggregation in E-R diagram.

- The E-R model cannot express relationships among relationships.
- When would we need such a thing at that time aggregation is used.
- Consider a database with information about employees who work on a particular project and use a number of machines doing that work.

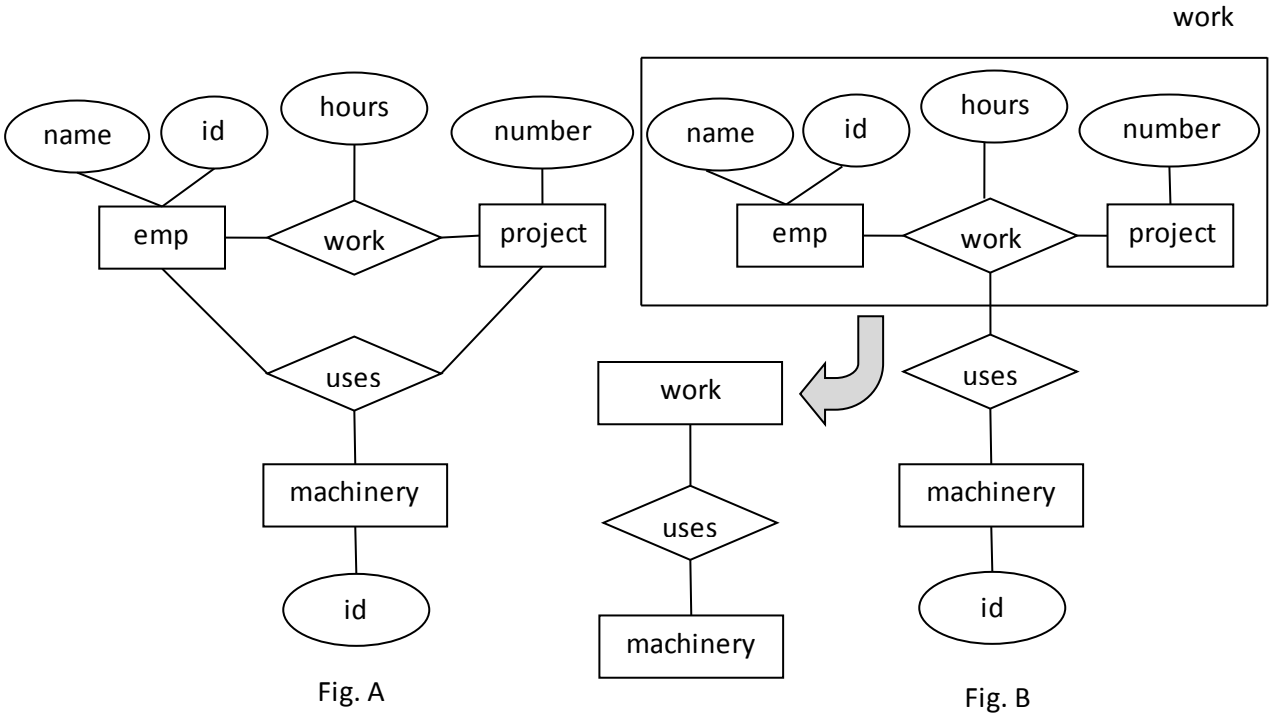


Fig. A

Fig. B

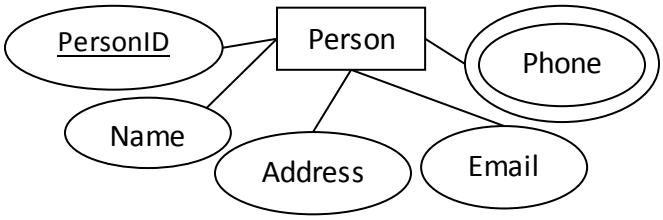
- Relationship sets *work* and *uses* could be combined into a single set. We can combine them by using aggregation.
- Aggregation is an abstraction through which relationships are treated as higher-level entities.
- For our example, we treat the relationship set work and the entity sets employee and project as a higher-level entity set called work.
- Transforming an E-R diagram with aggregation into tabular form is easy. We create a table for each entity and relationship set as before.
- The table for relationship set uses contains a column for each attribute in the primary key of machinery and work.

Explain the steps to reduce the ER diagram to ER database schema.

Step 1: Entities and Simple Attributes:

- An entity type within ER diagram is turned into a table. You may preferably keep the same name for the entity or give it a sensible name but avoid DBMS reserved words as well as avoid the use of special characters.
- Each attribute turns into a column (attribute) in the table. The key attribute of the entity is the primary key of the table which is usually underlined. It can be composite if required but can never be null.

- It is highly recommended that every table should start with its primary key attribute conventionally named as TablenameID.
- Consider the following simple ER diagram:



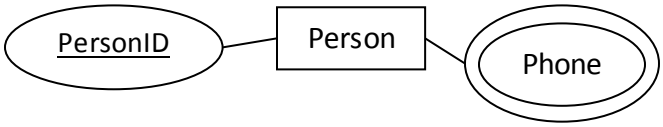
- The initial relational schema is expressed in the following format writing the table names with the attributes list inside a parentheses as shown below
  - ✓ Persons(personid, name, address, email)

Person			
personid	name	address	Email

- Persons and Phones are Tables and personid, name, address and email are Columns (Attributes).
- personid is the primary key for the table : Person

Step 2: Multi-Valued Attributes

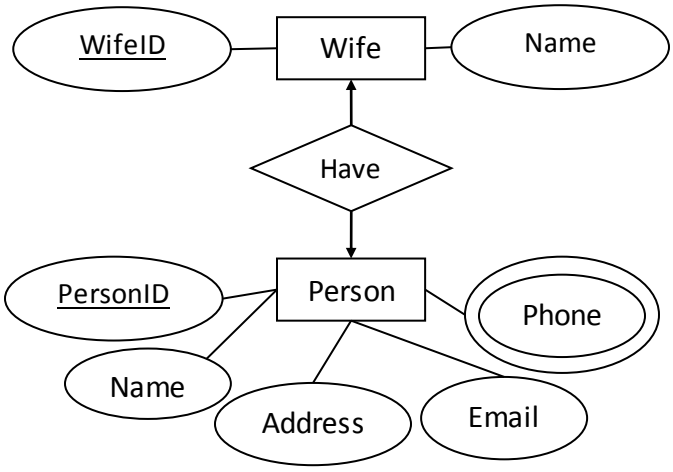
- A multi-valued attribute is usually represented with a double-line oval.



- If you have a multi-valued attribute, take that multi-valued attribute and turn it into a new entity or table of its own.
- Then make a 1:N relationship between the new entity and the existing one.
- In simple words.
  - ➢ 1. Create a table for that multi-valued attribute.
  - ➢ 2. Add the primary (id) column of the parent entity as a foreign key within the new table as shown below:
  - ✓ First table is Persons (personid, name, address, email)
  - ✓ Second table is Phones (phoneid , personid, phone)
- personid within the table Phones is a foreign key referring to the personid of Persons

Phone		
phoneid	personid	phone

Step 3: 1:1 Relationship



- Let us consider the case where the Person has one wife. You can place the primary key of the wife table wifeid in the table Persons which we call in this case Foreign key as shown below.
  - ✓ Persons(personid, name, address, email , wifeid)
  - ✓ Wife (wifeid , name)
- Or vice versa to put the personid as a foreign key within the wife table as shown below:
  - ✓ Persons(personid, name, address, email)
  - ✓ Wife (wifeid , name , personid)
- For cases when the Person is not married i.e. has no wifeID, the attribute can set to NULL

Persons				
personid	name	address	email	wifeid

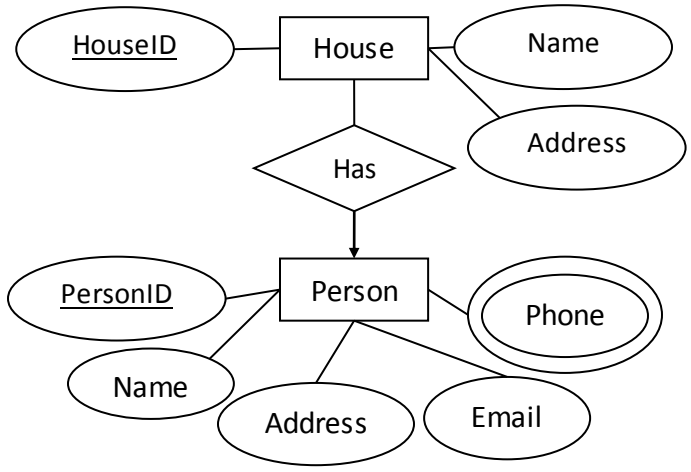
Wife	
wifeid	name

OR

Persons			
personid	name	address	email

Wife		
wifeid	name	personid

Step 4: 1:N Relationships

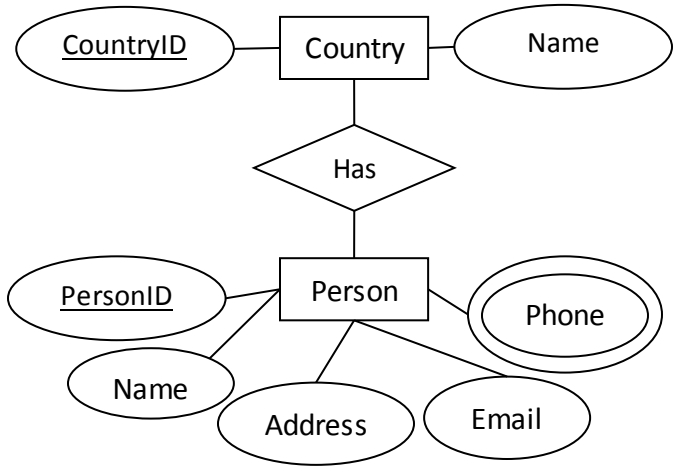


- For instance, the Person can have a House from zero to many, but a House can have only one Person.
- In such relationship place the primary key attribute of table having 1 mapping in to the table having many cardinality as a foreign key.
- To represent such relationship the personid as the Parent table must be placed within the Child table as a foreign key.
- It should convert to :
  - ✓ Persons(personid, name, address, email)
  - ✓ House (houseid, name , address, personid)

Persons			
personid	name	address	email

House			
houseid	name	address	personid

Step 5: N:N Relationships



- For instance, The Person can live or work in many countries. Also, a country can have many people. To express this relationship within a relational schema we use a separate table as shown below:
- It should convert into :
  - ✓ Persons(personid, name, address, email)
  - ✓ Countries (countryid, name)
  - ✓ HasRelat (hasrelatid, personid , countryid)

Persons			
personid	name	address	email

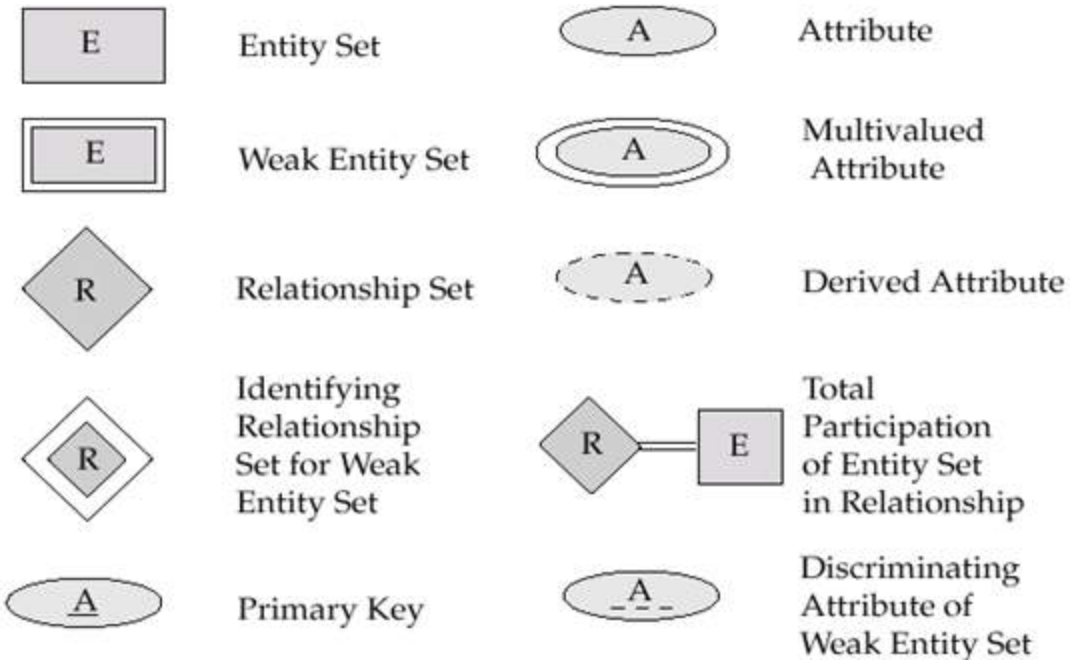
Countries	
countryid	name

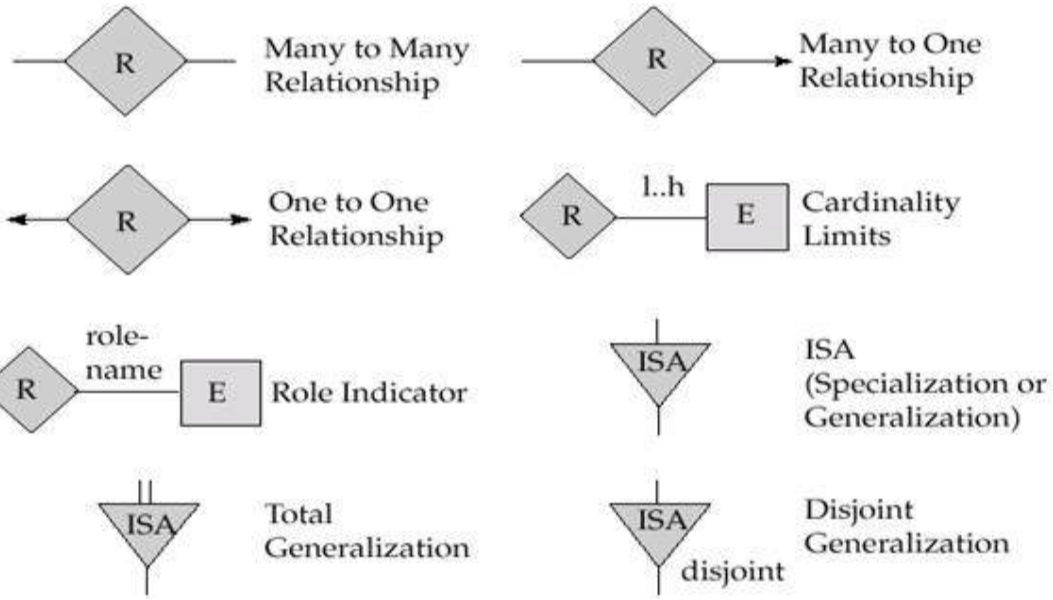
HasRelat		
hasrelatid	personid	countryid

What is E-R model (Entity-Relationship) model (diagram) also draw various symbols using in E-R diagram.

E-R model

- Entity-relationship (ER) model/diagram is a graphical representation of entities and their relationships to each other with their attributes.



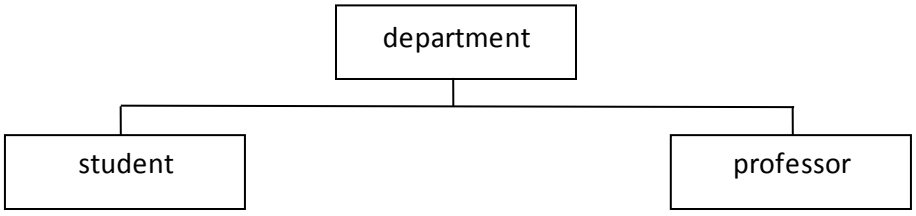


What is a Database Models?

- A database model is a type of data model that defines the logical structure of a database.
- It determine how data can be stored, accessed and updated in a database management system
- The most popular example of a database model is the relational model, which uses a table-based format.
- Type of Database Models are:
    1. Hierarchical Model
    2. Network Model
    3. Entity-relationship Model
    4. Relational Model
    5. Object-oriented database model

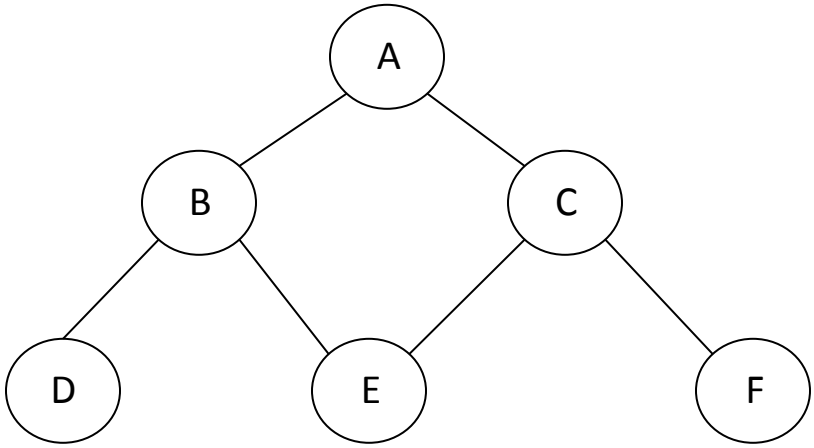
1. Hierarchical Model

- The hierarchical model organizes data into a tree-like structure, where each record has a single parent or root.

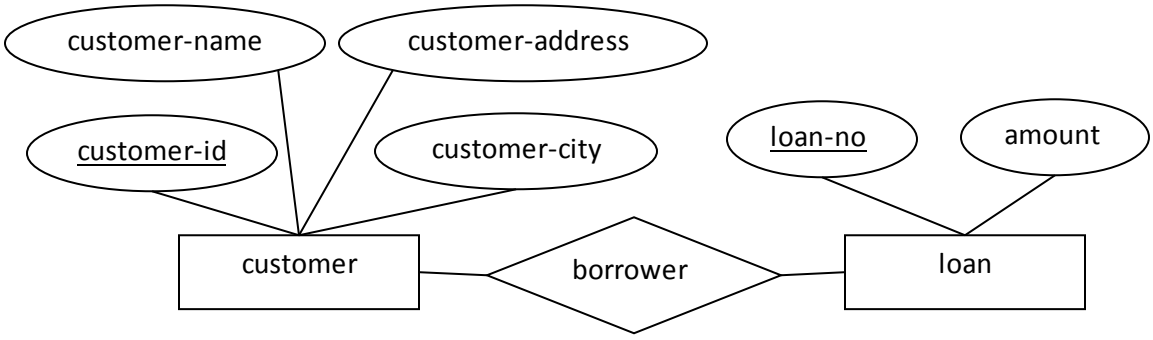


- The hierarchy starts from the Root data, and expands like a tree, adding child nodes to the parent nodes.
- In hierarchical model, data is organized into tree-like structure with one-to-many relationship between two different types of data, for example, one department can have many professors and many students.

2. *Network Model*
   - This is an extension of the hierarchical model, allowing many-to-many relationships in a tree-like structure that allows multiple parents.



3. *Entity-relationship Model*
   - In this database model, relationships are created by dividing object of interest into entity and its characteristics into attributes.



4. *Relational Model*
   - In this model, data is organized in two-dimensional tables and the relationship is maintained by storing a common attribute.

Rno	StudentName	Age
1	Raj	21
2	Meet	22

SubjectID	SubjectName	Teacher
1	DBMS	Patel
2	DS	Shah

Rno	SubjectID	Marks
1	1	98
1	2	95
2	1	95
2	2	90

5. *Object-oriented database model*
   - This data model is another method of representing real world objects.
   - It considers each object in the world as objects and isolates it from each other.
   - It groups its related functionalities together and allows inheriting its functionality to other related sub-groups.

What is Integrity Constraints?

- Integrity constraints are a set of rules. It is used to maintain the quality of information.
- Integrity constraints ensure that the data insertion, updating, and other processes have to be performed in such a way that data integrity is not affected.
- Thus, integrity constraint is used to guard against accidental damage to the database.
- Various Integrity Constraints are:
  1. Check
     - ✓ This constraint defines a business rule on a column. All the rows in that column must satisfy this rule.
     - ✓ Limits the data values of variables to a specific set, range, or list of values.
     - ✓ The constraint can be applied for a single column or a group of columns.
     - ✓ E.g. value of SPI should be between 0 to 10.
  2. Not null
     - ✓ This constraint ensures all rows in the table contain a definite value for the column which is specified as not null. Which means a null value is not allowed.
     - ✓ E.g. name column should have some value.

3. Unique
   - ✓ This constraint ensures that a column or a group of columns in each row have a distinct (unique) value.
   - ✓ A column(s) can have a null value but the values cannot be duplicated.
   - ✓ E.g enrollmentno column should have unique value.
4. Primary key
   - ✓ This constraint defines a column or combination of columns which uniquely identifies each row in the table.
   - ✓ Primary key = Unique key + Not null
   - ✓ E.g enrollmentno column should have unique value as well as can't be null.
5. Foreign key
   - ✓ A referential integrity constraint (foreign key) is specified between two tables.
   - ✓ In the referential integrity constraints, if a foreign key column in table 1 refers to the primary key column of table 2, then every value of the foreign key column in table 1 must be null or be available in primary key column of table 2.

Explain keys.

Super key

- A super key is a set of one or more attributes (columns) that allow us to identify each tuple (records) uniquely in a relation (table).
- For example, the enrollment_no, roll_no, semester with department_name of a student is sufficient to distinguish one student tuple from another. So {enrollment_no} and {roll_no, semester, department_name} both are super key.

Candidate key

- Candidate key is a super key for which no proper subset is a super key.
- For example, combination of roll_no, semester and department_name is sufficient to distinguish one student tuple from another. But either roll_no or semester or department_name alone or combination of any two columns is not sufficient to distinguish one student tuple from another. So {roll_no, semester, department_name} is candidate key.
- Every candidate key is super key but every super key may not candidate key.

Primary key

- A Primary key is a candidate key that is chosen by database designer to identify tuples uniquely in a relation.

Alternate key

- An Alternate key is a candidate key that is not chosen by database designer to identify tuples uniquely in a relation.

Foreign key

- A foreign key is a set of one or more attributes whose values are derived from the primary key attribute of another relation.

What is relational algebra? Explain relational algebraic operation.

- Relational algebra is a language for expressing relational database queries.
- Relation algebra is a procedural query language.
- Relational algebraic operations are as follows:

Selection:-

- **Operation:** Selects tuples from a relation that satisfy a given condition.
  It is used to select particular tuples from a relation.
  It selects particular tuples but all attribute from a relation.
- **Symbol:** σ (Sigma)
- **Notation:** $\sigma_{(condition)}$ <Relation>
- **Operators:** The following operators can be used in a condition.
  =, !=, <, >, <=,>=, Λ(AND), V(OR)

- Consider following table

Student			
Rno	**Name**	**Dept**	**CPI**
101	Ramesh	CE	8
108	Mahesh	EC	6
109	Amit	CE	7
125	Chetan	CI	8
138	Mukesh	ME	7
128	Reeta	EC	6
133	Anita	CE	9

- **Example**: Find out all the students of CE department.

$\sigma_{Dept="CE"}$ (Student)

- **Output**: The above query returns all tuples which contain CE as department name.
  Output of above query is as follows

Student			
Rno	**Name**	**Dept**	**CPI**
101	Ramesh	CE	8
109	Amit	CE	7
133	Anita	CE	9

Projection:-

- **Operation:** Selects specified attributes of a relation.
  It selects particular attributes but all unique tuples from a relation.
- **Symbol:** ∏ (Pi)
- **Notation:** $\prod_{(attribute\ set)}$ <Relation>
- Consider following table

Student			
Rno	**Name**	**Dept**	**CPI**
101	Ramesh	CE	8
108	Mahesh	EC	6
109	Amit	CE	7
125	Chetan	CI	8
138	Mukesh	ME	7
128	Reeta	EC	6
133	Anita	CE	9

- **Example**: List out all students with their roll no, name and department name.

$\prod_{Rno,\ Name,\ Dept}$ (Student)

- **Output**: The above query returns all tuples with three attributes roll no, name and department name.

Output of above query is as follows

Student		
Rno	**Name**	**Dept**
101	Ramesh	CE
109	Amit	CE
125	Chetan	CI
138	Mukesh	ME
133	Anita	CE

- **Example**: List out students of CE department with their roll no, name and department.

$$\prod_{Rno,\ Name,\ Dept} (\sigma_{Dept="CE"} (Student))$$

- **Output**: The above query returns tuples which contain CE as department with three attributes roll no, name and department name.

Output of above query is as follows

Student		
Rno	**Name**	**Dept**
101	Ramesh	CE
109	Amit	CE
133	Anita	CE

Division:-

- **Operation:** The division is a binary relation that is written as R1 ÷ R2.
- **Condition to perform operation:** Attributes of R2 is proper subset of attributes of R1.
- **The output of the division operator will have attributes =**

  All attributes of R1 – All attributes of R2
- **The output of the division operator will have tuples =**

  Tuples in R1, which are associated with the all tuples of R2
- **Symbol:** ÷
- **Notation:** R1 ÷ R2
- Consider following table

Project
Task
Database1
Database2

Work	
Student	**Task**
Shah	Database1
Shah	Database2
Shah	Compiler1
Vyas	Database1
Vyas	Compiler1
Patel	Database1
Patel	Database2

- **Example**: Find out all students having both tasks Database1 as well as Database2.

$$\prod_{(student,\ Task)}(Work) \div \prod_{(Task)}(Project)$$

- **Output**: It gives name of all students whose task is both Database1 as well as Database2.

Output of above query is as follows

Student
Shah
Patel

Cartesian product:-

- **Operation:** Combines information of two relations.

  It will multiply each tuples of first relation to each tuples of second relation.

  It is also known as Cross product operation and similar to mathematical Cartesian product operation.
- **Symbol:** X (Cross)
- **Notation:** Relation1 X Relation2
- **Resultant Relation :**
  - ✓ If relation1 and relation2 have n1 and n2 attributes respectively, then resultant relation will have n1 + n2 attributes from both the input relations.
  - ✓ If both relations have some attribute having same name, it can be distinguished by combing relation-name.attribute-name.
  - ✓ If relation1 and relation2 have n1 and n2 tuples respectively, then resultant relation will have n1*n2 tuples, combining each possible pair of tuples from both the input relations.

R

A	1
B	2
D	3

S

A	1
D	2
E	3

R × S

A	1	A	1
A	1	D	2
A	1	E	3
B	2	A	1
B	2	D	2
B	2	E	3
D	3	A	1
D	3	D	2
D	3	E	3

- Consider following table

Emp		
Empid	**Empname**	**Deptname**
S01	Manisha	Finance
S02	Anisha	Sales
S03	Nisha	Finance

Dept	
Deptname	**Manager**
Finance	Arun
Sales	Rohit
Production	Kishan

- **Example**:

Emp × Dept				
Empid	**Empname**	**Emp.Deptname**	**Dept.Deptname**	**Manager**
S01	Manisha	Finance	Finance	Arun
S01	Manisha	Finance	Sales	Rohit
S01	Manisha	Finance	Production	Kishan
S02	Anisha	Sales	Finance	Arun
S02	Anisha	Sales	Sales	Rohit
S02	Anisha	Sales	Production	Kishan
S03	Nisha	Finance	Finance	Arun
S03	Nisha	Finance	Sales	Rohit
S03	Nisha	Finance	Production	Kishan

Join:-

Natural Join Operation (⋈)

- **Operation:** Natural join will retrieve information from multiple relations. It works in three steps.
    1. It performs Cartesian product
    2. Then it finds consistent tuples and inconsistent tuples are deleted
    3. Then it deletes duplicate attributes
- **Symbol:** ⋈
- **Notation:** Relation1 ⋈ Relation2
- Consider following table

To perform a natural join there must be **one common attribute (column)** between two relations.

Emp		
Empid	**Empname**	**Deptname**
S01	Manisha	Finance
S02	Anisha	Sales
S03	Nisha	Finance

Dept	
Deptame	**Manager**
Finance	Arun
Sales	Rohit
Production	Kishan

- **Example**:

Emp ⋈ Dept			
Empid	**Empname**	**Deptname**	**Manager**
S01	Manisha	Finance	Arun
S02	Anisha	Sales	Rohit
S03	Nisha	Finance	Arun

$\prod_{Empname,\ Manager}$ (Emp ⋈ Dept)	
Empname	**Manager**
Manisha	Arun
Anisha	Rohit
Nisha	Arun

The Outer Join Operation

- In natural join some records are missing if we want that missing records than we have to use outer join.
- The outer join operation can be divided into three different forms:
  1. Left outer join (⟕)
  2. Right outer join (⟖)
  3. Full outer join (⟗)
- Consider following tables

College		
Name	**Id**	**Department**
Manisha	S01	Computer
Anisha	S02	Computer
Nisha	S03	I.T.

Hostel		
Name	**Hostel_name**	**Room_no**
Anisha	Kaveri hostel	K01
Nisha	Godavari hostel	G07
Isha	Kaveri hostel	K02

Left outer join (⟕)

- The left outer join returns all the tuples of the left relation even through there is no matching tuple in the right relation.
- For such kind of tuples having no matching, the attributes of right relation will be padded with null in resultant relation.
- Example : College ⟕ Hostel

College ⟕ Hostel				
Name	**Id**	**Department**	**Hostel_name**	**Room_no**
Manisha	S01	Computer	Null	Null
Anisha	S02	Computer	Kaveri hostel	K01
Nisha	S03	I.T.	Godavari hostel	G07

Right outer join (⟖)

- The right outer join returns all the tuples of the right relation even though there is no matching tuple in the left relation.
- For such kind of tuples having no matching, the attributes of left relation will be padded with null in resultant relation.
- Example : College ⟖ Hostel

College ⟖ Hostel				
Name	**Id**	**Department**	**Hostel_name**	**Room_no**
Anisha	S02	Computer	Kaveri hostel	K01
Nisha	S03	I.T.	Godavari hostel	G07
Isha	Null	Null	Kaveri Hostel	K02

Full outer join (⟗)

- The full outer join returns all the tuples of both of the relations. It also pads null values whenever required.
- Example : College ⟗ Hostel

College ⟗ Hostel				
Name	**Id**	**Department**	**Hostel_name**	**Room_no**
Manisha	S01	Computer	Null	Null
Anisha	S02	Computer	Kaveri hostel	K01
Nisha	S03	I.T.	Godavari hostel	G07
Isha	Null	Null	Kaveri Hostel	K02

Set Operators

- Set operators combine the results of two or more queries into a single result.
- **Condition to perform set operation:**
  - ✓ Both relations (queries) must be union compatible :
  - ✓ Relations R and S are union compatible, if
    - ❖ Both queries should have same (equal) number of columns, and
    - ❖ Corresponding attributes should have the same data type.
- **Types of set operators:**
  1. Union
  2. Intersect (Intersection)
  3. Minus (Set Difference)

Union

- **Operation:** Selects tuples those are in either or both of the relations.
- **Symbol :** U (Union)
- **Notation :** Relation1 U Relation2
- **Example :**

R

A	1
B	2
D	3
F	4
E	5

S

A	1
C	2
D	3
E	4

R U S

A	1
B	2
C	2
D	3
F	4
E	5
E	4

- Consider following tables

Emp	
Id	**Name**
1	Manisha
2	Anisha
3	Nisha

Cst	
Id	**Name**
1	Manisha
2	Anisha
4	Isha

- **Example**:

$\prod_{Name}$ (Emp) U $\prod_{Name}$ (Cst)
Name
Manisha
Anisha
Nisha
Isha

Intersection

- **Operation:** Selects tuples those are common in both relations.
- **Symbol :** ∩ (Intersection)
- **Notation :** Relation1 ∩ Relation2
- **Example**

R

A	1
B	2
D	3
F	4
E	5

S

A	1
C	2
D	3
E	4

R ∩ S

A	1
D	3

- Consider following tables

Emp	
Id	**Name**
1	Manisha
2	Anisha
3	Nisha

Cst	
Id	**Name**
1	Manisha
2	Anisha
4	Isha

- **Example**:

$\prod_{Name}$ (Emp) ∩ $\prod_{Name}$ (Cst)
Name
Manisha
Anisha

Difference:-

- **Operation:** Selects tuples those are in first (left) relation but not in second (right) relation.
- **Symbol : —** (Minus)
- **Notation :** Relation1 **—** Relation2
- **Example :**

R

A	1
B	2
D	3
F	4
E	5

S

A	1
C	2
D	3
E	4

R — S

B	2
F	4
E	5

- Consider following tables

Emp	
Id	**Name**
1	Manisha
2	Anisha
3	Nisha

Cst	
Id	**Name**
1	Manisha
2	Anisha
4	Isha

- **Example**:

$\prod_{Name}$ (Emp) - $\prod_{Name}$ (Cst)
Name
Nisha

$\prod_{Name}$ (Cst) - $\prod_{Name}$ (Emp)
Name
Isha

Rename:-

- **Operation:** It is used to rename a relation or attributes.
- **Symbol:** ρ (Rho)
- **Notation:** $\rho_A(B)$ Rename relation B to A.

  $\rho_{A(X1,X2....Xn)}(B)$ Rename relation B to A and its attributes to X1, X2, …., Xn.

- Consider following table

Student			
Rno	**Name**	**Dept**	**CPI**
101	Ramesh	CE	8
108	Mahesh	EC	6
109	Amit	CE	7
125	Chetan	CI	8
138	Mukesh	ME	7
128	Reeta	EC	6
133	Anita	CE	9

- **Example**: Find out highest CPI from student table.

$$\prod_{CPI}(Student) - \prod_{A.CPI}(\sigma_{A.CPI<B.CPI}(\rho_A(Student) \times \rho_B(Student)))$$

- **Output**: The above query returns highest CPI.

  Output of above query is as follows

CPI
9

Aggregate Function:-

- **Operation:** It takes a more than one value as input and returns a single value as output (result).
- **Symbol:** G
- **Notation:** $G_{function\ (attribute)}$ (relation)
- **Aggregate functions:** Sum, Count, Max, Min, Avg.
- Consider following table

Student			
Rno	**Name**	**Dept**	**CPI**
101	Ramesh	CE	8
108	Mahesh	EC	6
109	Amit	CE	7
125	Chetan	CI	8
138	Mukesh	ME	7
128	Reeta	EC	6
133	Anita	CE	9

- **Example**: Find out sum of all students CPI.

  $G_{sum\ (CPI)}$ (Student)

  Output: The above query returns sum of CPI.

  Output of above query is as follows

sum
51

- **Example**: Find out max and min CPI.

  G max (CPI), min (CPI) (Student)

  Output: The above query returns sum of CPI.

  Output of above query is as follows

max	min
9	6

What is the difference between Open source and Commercial DBMS.

Open source DBMS	*Commercial DBMS*
DBMS, which is available in the market at free of cost.	DBMS, which is available in the market at a certain price.
The code of open source DBMS product can be viewed, shared or modified by the community.	The code of commercial DBMS product cannot be view, share or modify by the community.
There are chances of malfunctioning with code as source code is open.	The security is high and code is not accessible to unauthorized person.
Examples: MySQL, MongoDB, SQLite etc	Examples: Microsoft SQL Server, IBM Db2 etc

Consider following schema and represent given statements in relation algebra form.

* Branch(branch_name,branch_city)

* Account(branch_name, acc_no, balance)

* Depositor(customer_name, acc_no)

(i) Find out list of customer who have account at 'abc' branch.

$\prod_{customer_name} (\sigma_{branch_name="abc"} (Depositor \bowtie Account))$

(ii) Find out all customer who have an account in 'Ahmedabad' city and balance is greater than 10,000.

$\prod_{customer_name} (\sigma_{Branch.branch_city="Ahmedabad"} \wedge \sigma_{Account.balance>10000} (Branch \bowtie Account \bowtie Depositor))$

(iii) find out list of all branch name with their maximum balance.

$\prod_{branch_name}, G_{max\ (balance)} (Account)$

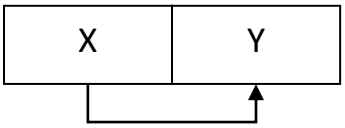
Define functional dependency. *OR* Explain different types of FD with the help of example.

Functional Dependency

- Let R be a relation schema having n attributes A1, A2, A3,…, An.
- Let attributes X and Y are two subsets of attributes of relation R.
- If the values of the X component of a tuple uniquely (or functionally) determine the values of the Y component, then, there is a functional dependency from X to Y.
- This is denoted by X → Y.
- It is referred as: Y is functionally dependent on the X, or X functionally determines Y.
- The abbreviation for functional dependency is FD or fd.
- The set of attributes X is called the left hand side of the FD, and Y is called the right hand side.
- The left hand side of the FD is also referred as determinant whereas the right hand side of the FD is referred as dependent.

Diagrammatic representation

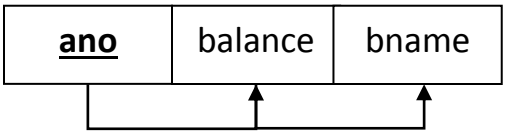
R:



Example

- Consider the relation Account(ano, balance, bname).
- In this relation ano can determines balance and bname. So, there is a functional dependency from ano to balance and bname.
- This can be denoted by ano → {balance, bname}.

Account:



Types of Functional Dependencies

Full Dependency

- In a relation, the attribute B is fully functional dependent on A if B is functionally dependent on A, but not on any proper subset of A.
- Eg. {Roll_No, Department_Name, Semester} →SPI

Partial Dependency

- In a relation, the attribute B is partial functional dependent on A if B is functionally dependent on A as well as on any proper subset of A.
- If there is some attribute that can be removed from A and the still dependency holds.

- Eg. {Enrollment_No, Department_Name} → SPI

Transitive Dependency

- In a relation, if attribute(s) A→B and B→C, then C is transitively depends on A via B (provided that A is not functionally dependent on B or C).
- Eg. Staff_No → Branch_No and Branch_No → Branch_Address

Trivial FD:

- X→Y is trivial FDif Y is a subset of X
- Eg.{Roll_No, Department_Name} → Roll_No

Nontrivial FD

- X→Y is nontrivial FDif Y is not a subset of X
- Eg.. {Roll_No, Department_Name} → Student_Name

List and explain Armstrong's axioms (inference rules).

- Let A, B, and C is subsets of the attributes of the relation R.

Reflexivity

- If B is a subset of A then A → B

Augmentation

- If A → B then AC → BC

Transitivity

- If A → B and B → C then A → C

Pseudo Transitivity

If A → B and BC → D then AC → D

Self-determination

- A → A

Decomposition

- If A → BC then A → B and A → C

Union

- If A → B and A → C then A → BC

Composition

- If A → B and C → D then A,C → BD

What is closure of a set of FDs? How to find closure of a set of FDs.

Closure of set of functional dependency (FDs)

- Given a set F set of functional dependencies, there are certain other functional dependencies that are logically implied by F.
- E.g.: F = {A → B and B → C}, then we can infer that A → C
- The set of functional dependencies (FDs) that is logically implied by F is called the closure of F.
- It is denoted by F^+.

Example-1

Suppose a relation R is given with attributes A, B, C, G, H and I.
Also, a set of functional dependencies F is given with following FDs.
F = {A → B, A → C, CG → H, CG → I, B → H}
Find Closure of F.

We have	Using rule	Derived FD
A → B and B → H	Transitivity rule	A → H
CG → H and CG → I	Union rule	CG → HI
A → C and CG → I	Pseudo-transitivity rule	AG → I
A → C and CG → H	Pseudo-transitivity rule	AG → H

- F^+ = { A → H, CG → HI, AG → I, AG → H }

Example-2

Compute the closure of the following set F of functional dependencies for relational schema R = (A, B, C, D, E, F):
F = (A → B, A → C, CD → E, CD → F, B → E).

We have	Using rule	Derived FD
A → B and A → C	Union rule	A → BC
CD → E and CD → F	Union rule	CD → EF
A → B and B → E	Transitivity rule	A → E
A → C and CD → E	Pseudo- Transitivity rule	AD → E
A → C and CD → F	Pseudo- Transitivity rule	AD → F

- F^+ = { A → BC, CD → EF, A → E, AD → E, AD → F }

Example-3

Compute the closure of the following set F of functional dependencies for relational schema R = (A, B, C, D, E):
F = (AB → C, D → AC, D → E).

We have	Using rule	Derived FD
D → AC	Decomposition rule	D → A and D → C
D → AC and D → E	Union rule	D → ACE

F^+ = { D → A, D → C, D → ACE }

What is closure of a set of attributes? Explain how (Write an algorithm) to find closure of a set of attributes.

Closure of set of attributes

- Given a set of attributes α, the closure of α under F is the set of attributes that are functionally determined by α under F.
- It is denoted by α^+.

Algorithm

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

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

Example

Consider the following relation schema Depositer_Account (cid, ano, acess_date, balance, bname).
For this relation, a set of functional dependencies F can be given as
F = { {cid, ano} → access_date and **ano → {balance, bname} }**
Find out the closure of {cid, ano} and ano.

Solution

- Find out { cid, ano }$^+$
  - Step-1 :{ cid, ano }$^+$ = { cid, ano }
  - Step-2 :{ cid, ano }$^+$ = { cid, ano, acess_date } # {cid, ano} $\subseteq X^+$
  - { cid, ano }$^+$ = { cid, ano, acess_date, balance, bname } # ano $\subseteq X^+$
  - Step-3 :{ cid, ano }$^+$ = { cid, ano, acess_date, balance, bname }

  So { cid, ano }$^+$ = { cid, ano, acess_date, balance, bname }
- Find out ano$^+$
  - Step-1 :ano$^+$ = ano
  - Step-2 :ano$^+$ = { ano, balance, bname } # ano $\subseteq X^+$
  - Step-3 :ano$^+$ = { ano, balance, bname }

  So ano$^+$ = { ano, balance, bname }

Given relation R with attributes A,B, C,D,E,F and set of FDs as A → BC, E → CF, B → E and CD → EF. Find out closure {A, B}$^+$ of the set of attributes.

Steps to find the closure {A, B}$^+$

Step-1: result = AB

Step-2: First loop

result = ABC	# for A → BC, A ⊆ result so result=result ∪ BC
result = ABC	# for E→ CF, E ∉ result so result=result
result = ABCE	# for B→ E, B ⊆ result so result=result ∪ E
result = ABCE	# for CD→ EF, CD ∉ result so result=result

result before step2 is AB and after step 2 is ABCE which is different so repeat same as step 2.

Step-3: Second loop

result = ABCE	# for A → BC, A ⊆ result so result=result ∪ BC
result = ABCEF	# for E→ CF, E ⊆ result so result=result ∪ CF
result = ABCEF	# for B→ E, B ⊆ result so result=result ∪ E
result = ABCEF	# for CD→ EF, CD ∉ result so result=result

result before step3 is ABCE and after step 3 is ABCEF which is different so repeat same as step 3.

Step-4: Third loop

result = ABCEF	# for A → BC, A ⊆ result so result=result ∪ BC
result = ABCEF	# for E→ CF, E ⊆ result so result=result ∪ CF
result = ABCEF	# for B→ E, B ⊆ result so result=result ∪ E
result = ABCEF	# for CD→EF, CD ∉result so result=result

result before step4 is ABCEF and after step 3 is ABCEF which is same so stop.

So Closure of {A, B}$^+$ is {A, B, C, E, F}.

What is canonical cover? Consider following set F of functional dependencies on schema R(A,B,C) and compute canonical cover for F.

A -> BC, B -> C, A -> B and AB -> C

Canonical cover

- A canonical cover of F is a minimal set of functional dependencies equivalent to F, having no redundant dependencies or redundant parts of dependencies.
- It is denoted by F_c.

A canonical cover for F is a set of dependencies Fc such that:

- A canonical cover for F is a set of dependencies Fc such that
  1) F logically implies all dependencies in Fc, and
  2) Fc logically implies all dependencies in F, and
  3) No functional dependency in Fc contains an extraneous attribute, and
  4) Each left side of functional dependency in Fc is unique.

Algorithm to find Canonical cover:

- To compute a canonical cover for F:
  repeat
    Step 1: Use the union rule to replace any dependencies in F
      $\alpha 1 \rightarrow \beta 1$ and $\alpha 1 \rightarrow \beta 2$ with $\alpha 1 \rightarrow \beta 1\ \beta 2$

    Step 2 : Find a functional dependency $\alpha \rightarrow \beta$ with an
      extraneous attribute either in α or in β
      If an extraneous attribute is found, delete it from $\alpha \rightarrow \beta$
  until F does not change
- Note: Union rule may become applicable after some extraneous attributes have been deleted, so it has to be re-applied

Steps to find Canonical Cover:

- Given : R = (A, B, C) and F = {A → BC, B → C, A → B, AB → C}
  Step 1: Combine A → BC and A → B into A → BC
    Set is now {A → BC, B → C, AB → C}
  Step 2: A is extraneous in AB → C
    Check if the result of deleting A from AB → C is implied by the other dependencies
    Yes: in fact, B → C is already present.
    Set is now {A → BC, B → C}
  Step 3: C is extraneous in A → BC
    Check if A → C is logically implied by A → B and the other dependencies
    Yes: using transitivity on A → B and B → C.
    Set is now {A → B and B → C}
    Can use attribute closure of A in more complex cases
- The canonical cover is: A → B, B → C

What is decomposition? Explain different types of decomposition.

Decomposition

- Decomposition is the process of breaking down given relation into two or more relations.
- Here, relation R is replaced by two or more relations in such a way that -
  1. Each new relation contains a subset of the attributes of R, and
  2. Together, they all include all tuples and attributes of R.
- Relational database design process starts with a universal relation schema R = {A1, A2, A3,..., An), which includes all the attributes of the database. The universal relation states that every attribute name is unique.

- Using functional dependencies, this universal relation schema is decomposed into a set of relation schemas D = {R1, R2, R3,…,Rm}.
- Now, D becomes the relational database schema and D is referred as decomposition of R.
- Generally, decomposition is used to eliminate the pitfalls of the poor database design during normalization process.
- For example, consider the relation Account_Branch given in figure:

Account_Branch			
Ano	**Balance**	**Bname**	**Baddress**
A01	5000	Vvn	Mota bazaar, VVNagar
A02	6000	Ksad	Chhota bazaar, Karamsad
A03	7000	Anand	Nana bazaar, Anand
A04	8000	Ksad	Chhota bazaar, Karamsad
A05	6000	Vvn	Mota bazaar, VVNagar

- This relation can be divided with two different relations
  1. Account (Ano, Balance, Bname)
  2. Branch (Bname, Baddress)
- These two relations are shown in below figure

Account		
Ano	**Balance**	**Bname**
A01	5000	Vvn
A02	6000	Ksad
A03	7000	Anand
A04	8000	Ksad
A05	6000	Vvn

Branch	
Bname	**Baddress**
Vvn	Mota bazaar, VVNagar
Ksad	Chhota bazaar, Karamsad
Anand	Nana Bazar, Anand

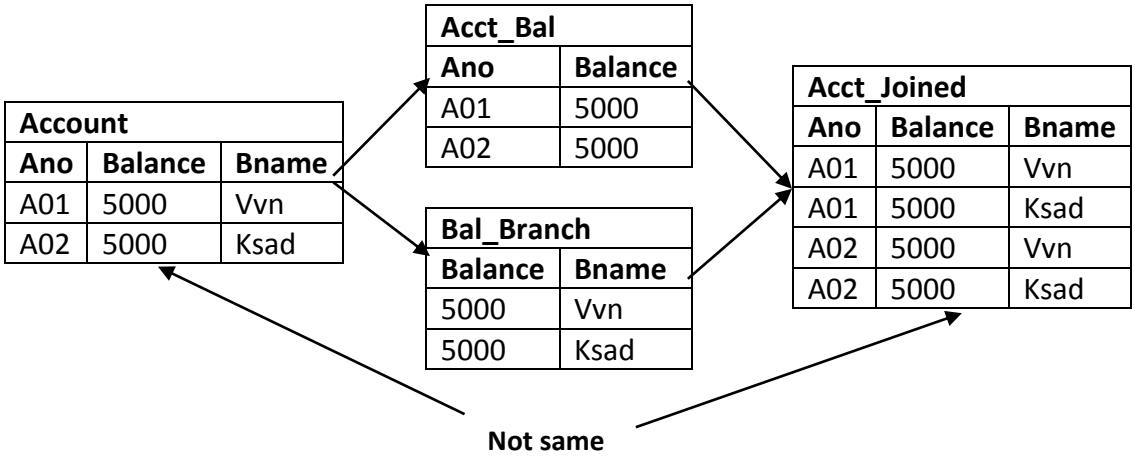
- A decomposition of relation can be either lossy decomposition or lossless decomposition.
- There are two types of decomposition
  1. lossy decomposition
  2. lossless decomposition (non-loss decomposition)

Lossy Decomposition

- The decomposition of relation R into R1 and R2 is lossy when the join of R1 and R2 does not yield the same relation as in R.
- This is also referred as lossy-join decomposition.
- The disadvantage of such kind of decomposition is that some information is lost during retrieval of original relation. And so, such kind of decomposition is referred as lossy decomposition.
- From practical point of view, decomposition should not be lossy decomposition.

Example

- A figure shows a relation Account. This relation is decomposed into two relations Acc_Bal and Bal_Branch.
- Now, when these two relations are joined on the common attributeBalance, the resultant relation will look like Acct_Joined. This Acct_Joined relation contains rows in addition to those in original relation Account.
- Here, it is not possible to specify that in which branch account A01 or A02 belongs.
- So, information has been lost by this decomposition and then join operation.



- In other words, decomposition is lossy if decompose into R1 and R2 and again combine (join) R1 and R2 we cannot get original table as R1, over X, where R is an original relation, R1 and R2 are decomposed relations, and X is a common attribute between these two relations.

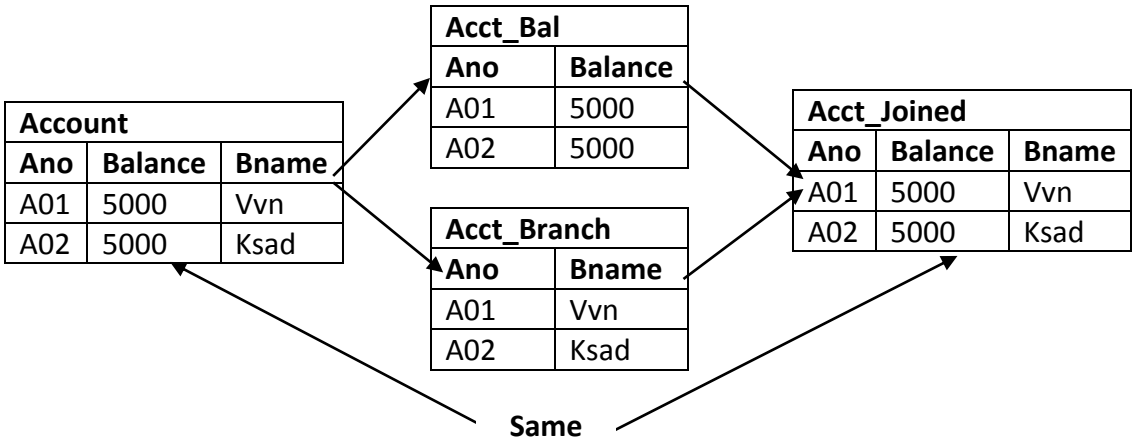
Lossless (Non-loss) Decomposition

- The decomposition of relation R into R1 and R2 is lossless when the join of R1 and R2 produces the same relation as in R.
- This is also referred as a non-additive (non-loss) decomposition.
- All decompositions must be lossless.

Example

- Again, the same relation Account is decomposed into two relations Acct_Bal and Acct_Branch.
- Now, when these two relations are joined on the common column Ano, the resultant relation will look like Acc_Joined relation. This relation is exactly same as that of original relation Account.
- In other words, all the information of original relation is preserved here.
- In lossless decomposition, no any fake tuples are generated when a natural join is applied to the relations in the decomposition.

- In other words, decomposition is lossy if R = join of R1 and R2, over X, where R is an original relation, R1 an R2 are decomposed relations, and x is a common attribute between these two relations.



What is an anomaly in database design? How it can be solved.

Anomaly in database design:

- Anomalies are problems that can occur in poorly planned, un-normalized database where all the data are stored in one table.
- There are three types of anomalies that can arise in the database because of redundancy are
  - ✓ Insert anomalies
  - ✓ Delete anomalies
  - ✓ Update / Modification anomalies
- Consider a relation emp_dept (E#, Ename, Address, D#, Dname, Dmgr#) with E# as a primary key.

Insert anomaly:

- Let us assume that a new department has been started by the organization but initially there is no employee appointed for that department, then the tuple for this department cannot be inserted in to this table as the E# will have NULL value, which is not allowed because E# is primary key.
- This kind of problem in the relation where some tuple cannot be inserted is known as insert anomaly.

Delete anomaly:

- Now consider there is only one employee in some department and that employee leaves the organization, then the tuple of that employee has to be deleted from the table, but in addition to that information about the department also will be deleted.
- This kind of problem in the relation where deletion of some tuples can lead to loss of some other data not intended to be removed is known as delete anomaly.

Update / Modification anomaly:

- Suppose the manager of a department has changed, this requires that the Dmgr# in all the tuples corresponding to that department must be changed to reflect the new status. If we fail to update all the tuples of given department, then two different records of employee working in the same department might show different Dmgr# lead to inconsistency in the database.
- This kind of problem is known as update or modification anomaly.

How anomalies in database design can be solved:

- Such type of anomalies in database design can be solved by using normalization.

What is normalization? What is the need of it? *OR* What is normalization? Why normalization process is needed?

Normalization

- Database normalization is the process of removing redundant data from your tables to improve storage efficiency, data integrity, and scalability.
- In the relational model, methods exist for quantifying how efficient a database is. These classifications are called normal forms (or NF), and there are algorithms for converting a given database between them.
- Normalization generally involves splitting existing tables into multiple ones, which must be re-joined or linked each time a query is issued.

Need of Normalization

- Eliminates redundant data
- Reduces chances of data errors
- Reduces disk space
- Improve data integrity, scalability and data consistency.

Explain different types of normal forms with example. *OR* Explain 1NF, 2NF, 3NF, BCNF, 4NF and 5NF with example.

1NF

- A relation R is in first normal form (1NF) if and only if all underlying domains contain atomic values only. **OR**
- A relation R is in first normal form (1NF) if and only if it does not contain any composite or multi valued attributes or their combinations.

Example

Cid	**Name**	**Address**		**TypeofAccountHold**
		Society	**City**	
C01	Riya	SaralSoc, Aand		Saving, Current, Salary
C02	Jiya	Birla Gruh, Rajkot		Saving, Current

- Above relation has four attributes Cid, Name, Address, Contact_no. Here address is composite attribute which is further divided in to sub attributes as Society and City. Another attribute TypeofAccountHold is multi valued attribute which can store more than one values. So above relation is not in 1NF.

Problem

- Suppose we want to find all customers for some particular city then it is difficult to retrieve. Reason is city name is combined with society name and stored whole as address.

Solution

- Divide composite attributes into number of sub- attribute and insert value in proper sub attribute. *AND*
- Split the table into two tables in such a way that
  - first table contains all attributes except multi-valued attribute and
  - other table contains multi-valued attribute and
  - insert primary key of first table in second table as a foreign key.
- So above table can be created as follows.

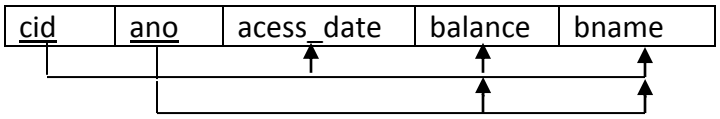
Cid	**Name**	**Society**	**City**
C01	Riya	SaralSoc	Aand
C02	Jiya	Birla Gruh	Rajkot

PhID	**Cid**	**Contact_no**
P01	C01	9879898798
P02	C01	9898052340
P03	C02	9825098254

2NF

- A relation R is in second normal form (2NF) if and only if it is in 1NF and every non-key attribute is fully dependent on the primary key. **OR**
- A relation R is in second normal form (2NF) if and only if it is in 1NF and no any non-key attribute is partially dependent on the primary key.

Example



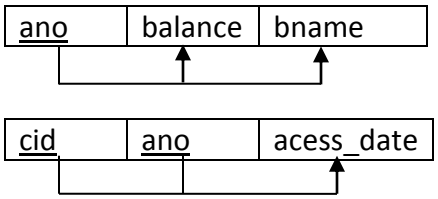
- Above relation has five attributes cid, ano, acess_date, balance, bname and two FDS
  FD1 {cid,ano}→{acess_date,balance,bname} and
  FD2 ano→{balance,bname}
- We have cid and ano as primary key. As per FD2 balace and bname are only depend on ano not cid. In above table balance and bname are not fully dependent on primary key but these attributes are partial dependent on primary key. So above relation is not in 2NF.

Problem

- For example in case of joint account multiple customers have common accounts. If some account says 'A02' is jointly by two customers says 'C02' and 'C04' then data values for attributes balance and bname will be duplicated in two different tuples of customers 'C02' and 'C04'.

Solution

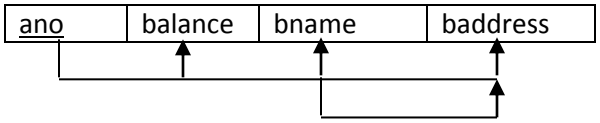
- Decompose relation in such a way that resultant relation does not have any partial FD.
- For this purpose remove partial dependent attribute that violets 2NF from relation. Place them in separate new relation along with the prime attribute on which they are full dependent.
- The primary key of new relation will be the attribute on which it if fully dependent.
- Keep other attribute same as in that table with same primary key.
- So above table can be decomposed as per following.



3NF

- A relation R is in third normal form (3NF) if and only if it is in 2NF and every non-key attribute is non-transitively dependent on the primary key.
- An attribute C is transitively dependent on attribute A if there exist an attribute B such that: A → B and B → C.

Example



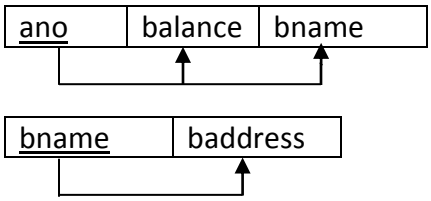
- Above relation has four attributes ano, balance, bname, baddress and two FDS
  FD1 ano→{balance, bname, baddress} and
  FD2 bname→baddress
- So from FD1 and FD2 and using transitivity rule we get ano→baddress.
- So there is transitively dependency from ano to baddress using bname in which baddress is non-prime attribute.
- So there is a non-prime attribute baddress which is transitively dependent on primary key ano.
- So above relation is not in 3NF.

Problem

- Transitively dependency results in data redundancy.
- In this relation branch address will be stored repeatedly from each account of same branch which occupy more space.

Solution

- Decompose relation in such a way that resultant relation does not have any non-prime attribute that are transitively dependent on primary key.
- For this purpose remove transitively dependent attribute that violets 3NF from relation. Place them in separate new relation along with the non-prime attribute due to which transitive dependency occurred. The primary key of new relation will be this non-prime attribute.
- Keep other attributes same as in that table with same primary key.
- So above table can be decomposed as per following.

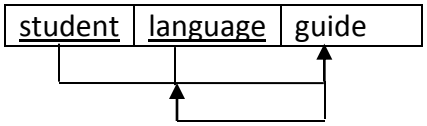


BCNF

- A relation R is in BCNF if and only if it is in 3NF and no any prime attribute is transitively dependent on the primary key. **OR**
- A relation R is in BCNF if and only if it is in 3NF and for every functional dependency X → Y, X should be the super key of the table **OR**
- A relation R is in BCNF if and only if it is in 3NF and for every functional dependency X → Y, X should be the primary key of the table.
- An attribute C is transitively dependent on attribute A if there exist an attribute B such that A→B and B→C.

Example

Student_Project		
Student	**Language**	**Guide**
Mita	JAVA	Patel
Nita	VB	Shah
Sita	JAVA	Jadeja
Gita	VB	Dave
Rita	VB	Shah
Nita	JAVA	Patel
Mita	VB	Dave
Rita	JAVA	Jadeja



- Above relation has five attributes cid, ano, acess_date, balance, bname and two FDS
  FD1 {student,language}→guide and
  FD2 guide→language
- So from FD1 and FD2 and using transitivity rule we get student→language
- So there is transitively dependency from student to language in which language is prime attribute.
- So there is on prime attribute language which is transitively dependent on primary key student.
- So above relation is not in BCNF.

Problem

- Transitively dependency results in data redundancy.
- In this relation one student have more than one project with different guide then records will be stored repeatedly from each student and language and guides combination which occupies more space.

Solution

- Decompose relation in such a way that resultant relation does not have any prime attribute transitively dependent on primary key.
- For this purpose remove transitively dependent prime attribute that violets BCNF from relation. Place them in separate new relation along with the non-prime attribute due to which transitive dependency occurred. The primary key of new relation will be this non-prime attribute.
- So above table can be decomposed as per following.

Student	**Guide**
Mita	Patel
Nita	Shah
Sita	Jadeja
Gita	Dave
Rita	Shah
Nita	Patel
Mita	Dave
Rita	Jadeja

Guide	**Language**
Patel	JAVA
Shah	VB
Jadeja	JAVA
Dave	VB

4NF

- A table is in the 4NF if it is in BCNF and has no non multivalued dependencies.

Example

- The multi-valued dependency X ↠ Y holds in a relation R if for a dependency X → Y, if for a single value of X, multiple (more than one) values of Y exists.
- Suppose a student can have more than one subject and more than one activity.

Student_Info		
Student_Id	**Subject**	**Activity**
100	Music	Swimming
100	Accounting	Swimming
100	Music	Tennis
100	Accounting	Tennis
150	Math	Jogging

- Note that all three attributes make up the Primary Key.
- Note that Student_Id can be associated with many subject as well as many activities (multi-valued dependency).
- Suppose student 100 signs up for skiing. Then we would insert (100, Music, Skiing). This row implies that student 100 skies as Music subject but not as an accounting subject, so in order to keep the data consistent we must add one more row (100, Accounting, Skiing). This is an insertion anomaly.
- Suppose we have a relation R(A) with a multivalued dependency X ↠ Y. The MVD can be removed by decomposing R into R1(R - Y) and R2(X U Y).
- Here are the tables Normalized

Student_Id	**Subject**
100	Music
100	Accounting
150	Math

StudentId	**Activity**
100	Swimming
100	Tennis
150	Jogging

5NF

- A table is in the 5NF if it is in 4NF and if for all Join dependency (JD) of (R_1, R_2, R_3, ...,R_m) in R, every R_i is a superkey for R. **OR**
- A table is in the 5NF if it is in 4NF and if it cannot have a lossless decomposition in to any number of smaller tables (relations).
- It is also known as Project-join normal form (PJ/NF).

Example

- We have a table that contains students subectwise result as per follows:

ResultID	RollNo	StudentName	SubjectName	Result
1	101	Raj	DBMS	Pass
2	101	Raj	DS	Pass
3	101	Raj	DE	Pass
4	102	Meet	DBMS	Pass
5	102	Meet	DS	Fail
6	102	Meet	DE	Pass
7	103	Suresh	DBMS	Fail
8	103	Suresh	DS	Pass
9	103	Suresh	DE	Fail

- Above table is not in 5NF because we can decompose into sub tables.
- If we decompose above table into multiple table as per follows:

ResultID	RollNo	SubjectID	Result
1	101	1	Pass
2	101	2	Pass
3	101	3	Pass
4	102	1	Pass
5	102	2	Fail
6	102	3	Pass
7	103	1	Fail
8	103	2	Pass
9	103	3	Fail

RollNo	StudentName
101	Raj
102	Meet
103	Suresh

SubjectID	SubjectName
1	DBMS
2	DS
3	DE

- We cannot decomposition any of above three tables into the sub tables so above three tables are in 5NF.

Normalize (decompose) following relation into lower to higher normal form. (From 1NF to 4 NF) ***OR***

PLANT	MANAGER	MACHINE	SUPPLIER_NAME	SUPPLIER_CITY
Plant-A	Ravi	Lathe Boiler	Jay industry Abb aplliance	Ahmedabad Surat
Plant-B	Meena	Cutter Boiler CNC	Raj machinery Daksh industry Jay industry	Vadodara Rajkot Ahmedabad

Explain with suitable example, the process of normalization converting from 1NF to 3NF.

1 Normal Form (1NF)

PLANT _ID	PLANT _NAME	MANAGER _ID	MANAGER _NAME	MACHINE _ID	MACHINE _NAME	SUPPLIER _ID	SUPPLIER _NAME	SUPPLIER _CITY
P1	Plant-A	E1	Ravi	M1	Lathe	S1	Jay industry	Ahmedabad
P1	Plant-A	E1	Ravi	M2	Boiler	S2	Abb aplliance	Surat
P2	Plant-B	E2	Meena	M3	Cutter	S3	Raj machinery	Vadodara
P2	Plant-B	E2	Meena	M2	Boiler	S4	Daksh industry	Rajkot
P2	Plant-B	E2	Meena	M4	CNC	S1	Jay industry	Ahmedabad

2 Normal Form (2NF)

Table-1

PLANT_ID	PLANT_NAME	MANAGER_ID	MANAGER_NAME
P1	Plant-A	E1	Ravi
P2	Plant-B	E2	Meena

Table-2

PLANT_ID	MACHINE_ID	MACHINE_NAME
P1	M1	Lathe
P1	M2	Boiler
P2	M3	Cutter
P2	M2	Boiler
P2	M4	CNC

Table-3

MANAGER_ID	SUPPLIER_ID	SUPPLIER_NAME	SUPPLIER_CITY
E1	S1	Jay industry	Ahmedabad
E1	S2	Abb aplliance	Surat
E2	S3	Raj machinery	Vadodara
E2	S4	Daksh industry	Rajkot
E2	S1	Jay industry	Ahmedabad

3 Normal Form or BCNF (3NF or BCNF)

Table-1

PLANT_ID	PLANT_NAME
P1	Plant-A
P2	Plant-B

Table-2

MANAGER_ID	MANAGER_NAME
E1	Ravi
E2	Meena

Table-3

PLANT_ID	MANAGER_ID
P1	E1
P2	E2

Table-4

MACHINE_ID	MANAGER_NAME
M1	Lathe
M2	Boiler
M3	Cutter
M4	CNC

Table-5

PLANT_ID	MACHINE_ID
P1	M1
P1	M2
P2	M3
P2	M2
P2	M4

Table-6

SUPPLIER_ID	SUPPLIER_NAME	SUPPLIER_CITY
S1	Jay industry	Ahmedabad
S2	Abb aplliance	Surat
S3	Raj machinery	Vadodara
S4	Daksh industry	Rajkot

Table-7

MANAGER_ID	SUPPLIER_ID
E1	S1
E1	S2
E2	S3
E2	S4
E2	S1

4 Normal Form (4NF)

Table-1

PLANT_ID	PLANT_NAME
P1	Plant-A
P2	Plant-B

Table-2

MANAGER_ID	MANAGER_NAME
E1	Ravi
E2	Meena

Table-3

MACHINE_ID	MACHINE_NAME
M1	Lathe
M2	Boiler
M3	Cutter
M4	CNC

Table-4

PLANT_MACHINE_ID	PLANT_ID	MACHINE_ID
PM1	P1	M1
PM2	P1	M2
PM3	P2	M3
PM4	P2	M2
PM5	P2	M4

Table-5

PLANT_MACHINE_ID	MANAGER_ID
PM1	E1
PM2	E1
PM3	E2
PM4	E2
PM5	E2

Table-6

SUPPLIER_ID	SUPPLIER_NAME	SUPPLIER_CITY
S1	Jay industry	Ahmedabad
S2	Abb aplliance	Surat
S3	Raj machinery	Vadodara
S4	Daksh industry	Rajkot

Table-7

MANAGER_ID	SUPPLIER_ID
E1	S1
E1	S2
E2	S3
E2	S4
E2	S1

How to find key?

- Conditions to find key
    1. If an attribute will not occurs on any side of any FD, then it is in every key.
    2. If an attribute occurs on the left-hand side of an FD, but never occurs on the right-hand side, then it is in every key.
    3. If an attribute occurs on the right-hand side of an FD, but never occurs on the left-hand side, then it is never in a key.
    4. If an attribute occurs on both the sides of an FD, then one cannot say anything about the attribute.

Example to find key

Let a relation R with attributes ABCD with FDs C → A, B → C. Find keys for relation R.

1. Attribute not occur on any side of FDs (D)
2. Attribute occurs on only left-hand side of an FDs (B)
3. Attribute occurs on only right-hand side of an FDs (A)
4. Attribute occurs on both the sides of an FDs (C)

- The core is BD. B determines C which determines A, so BD is a key. Therefore it is the only key.

Consider a relation R with five attribute A, B, C, D and E having following dependencies:

A → B, BC → E and ED → A

a) List all keys for R.

b) In which normal form table is, justify your answer. *OR*

Consider table R(A,B,C,D,E) with FDs as A → B, BC → E and ED → A. The table is in which normal form? Justify your answer.

Keys for R are:

{ACD} {BCD} {CDE}

- Above relation R is in 3NF because there is no non-prime attributes. That is, every column (attribute) is a part of Super Key, so the right hand side of every FD must be a part of super key. But it's not in BCNF (Or 4NF or 5NF).

In the BCNF decomposition algorithm, suppose you use a functional dependency α → β to decompose a relation schema r (α , β , γ) into r1 (α , β) and r2 (α , γ).

1. **What primary and foreign-key constraint do you expect to hold on the decomposed relations?**

Ans. α should be a primary key for r1, and α should be the foreign key from r2, referencing r1.

2. **Give an example of an inconsistency that can arise due to an erroneous update, if the foreign-key constraint were not enforced on the decomposed relations above.**

Ans. If the foreign key constraint is not enforced, then a deletion of a tuple from r1 would not have a corresponding deletion from the referencing tuples in r2. Instead of deleting a tuple from r2, this would amount to simply setting the value of α to null in some tuples

3. **When a relation is decomposed into 3NF, what primary and foreign key dependencies would you expect will hold on the decomposed schema?**

Ans. For every schema ri(αβ) added to the schema because of a rule α → β, α should be made the primary key. Also, a candidate key γ for the original relation is located in some newly created relation rk, and is a primary key for that relation.

Foreign key constraints are created as follows: for each relation ri created above, if the primary key attributes of ri also occur in any other relation rj , then a foreign key constraint is created from those attributes in rj , referencing (the primary key of) ri .

A college maintains details of its lecturers' subject area skills. These details comprise: Lecturer Number, Lecturer Name, Lecturer Grade, Department Code, Department Name, Subject

Code, Subject Name and Subject Level. Assume that each lecturer may teach many subjects but may not belong to more than one department. Subject Code, Subject Name and Subject Level are repeating fields. Normalize this data to Third Normal Form.

UNF

- Lecturer Number, Lecturer Name, Lecturer Grade, Department Code, Department Name, Subject Code, Subject Name, Subject Level

1NF

- Lecturer Number, Lecturer Name, Lecturer Grade, Department Code, Department Name
- Lecturer Number, Subject Code, Subject Name, Subject Level

2NF

- Lecturer Number, Lecturer Name, Lecturer Grade, Department Code, Department Name
- Lecturer Number, Subject Code
- Subject Code, Subject Name, Subject Level

3NF

- Lecturer Number, Lecturer Name, Lecturer Grade, Department Code
- Department Code, Department Name
- Lecturer Number, Subject Code
- Subject Code, Subject Name, Subject Level

A software contract and consultancy firm maintains details of all the various projects in which its employees are currently involved. These details comprise: Employee Number, Employee Name, Date of Birth, Department Code, Department Name, Project Code, Project Description and Project Supervisor
Assume the following:

1. **Each employee number is unique.**
2. **Each department has a single department code.**
3. **Each project has a single code and supervisor.**
4. **Each employee may work on one or more projects.**
5. **Employee names need not necessarily be unique.**
6. **Project Code, Project Description and Project Supervisor are repeating fields.**

Normalise this data to Third Normal Form.

UNF

- Employee Number, Employee Name, Date of Birth, Department Code, Department Name, Project Code, Project Description, Project Supervisor

1NF

- Employee Number, Employee Name, Date of Birth, Department Code, Department Name
- Employee Number, Project Code, Project Description, Project Supervisor

2NF

- Employee Number, Employee Name, Date of Birth, Department Code, Department Name
- Employee Number, Project Code,
- Project Code, Project Description, Project Supervisor

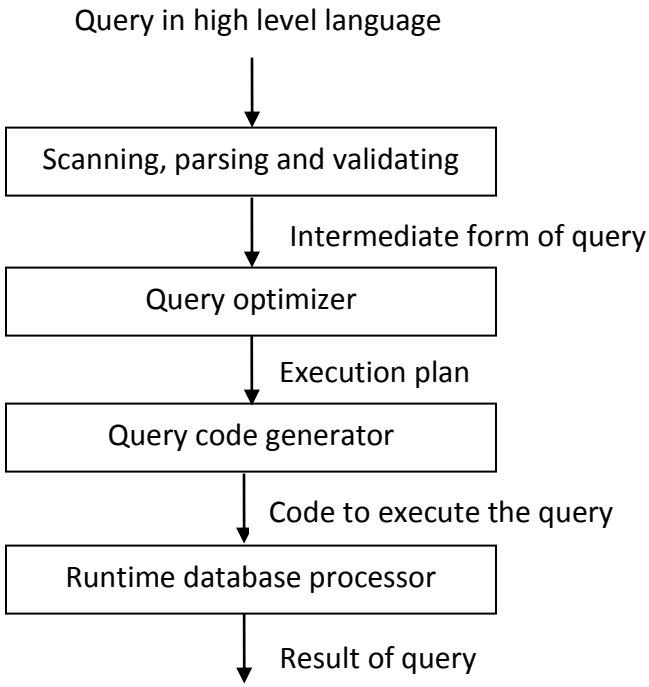
3NF

- Employee Number, Employee Name, Date of Birth, Department Code
- Department Code, Department Name
- Employee Number, Project Code
- Project Code, Project Description, Project Supervisor

Explain query processing.

Query processing

- It is a process of transforming a high level query (such as SQL) in to low level language.
- Query processing refers to the range of activities involved in extracting data from a database.



- A query expressed in a high level query language such as SQL must be
  - ✓ Scanned
  - ✓ Parsed
  - ✓ Validated
- The scanner identifies the language tokens such as SQL keywords, attribute names and relation names in the text of the query.
- Parser checks the query syntax to determine whether it is formulated according to the syntax rules of the query language.
- The query must also be validated by checking that all attributes and relation names are valid and semantically meaningful names in the schema of the particular database being queried.
- A query typically has many possible execution strategies and the process of choosing a suitable one for processing a query is known as query optimization.
- The query optimizer module has the task of producing an execution plan and code generator generates the code to execute that plan.
- The runtime database processor has the task of running the query code whether in compiled or interpreted mode, to produce the query result.

- If a runtime error results, an error message is generated by the runtime database processor.
- Query code generator will generate code for query.
- Runtime database processor will select optimal plan and execute query and gives result.

Explain different search algorithm for selection operation. *OR* Explain linear search and binary search algorithm for selection operation.

- There are two scan algorithms to implement the selection operation:
    1. Linear search
    2. Binary search

Linear search

- In a linear search, the systems scans each file block and tests all records to see whether they satisfy the selection condition.
- For a selection on a key attribute, the system can terminate the scan if the requires record is found, without looking at the other records of the relation.
- The cost of linear search in terms of number of I/O operations is $\mathbf{b_r}$ where $\mathbf{b_r}$ is the number of blocks in file.
- Selection on key attribute has an average cost of $\mathbf{b_r/2}$.
- It may be a slower algorithm than any another algorithm.
- This algorithm can be applied to any file regardless of the ordering of file or the availability of indices or the nature of selection operation.

Binary search

- If the file is ordered on attribute and the selection condition is an equality comparison on the attribute, we can use a binary search to locate the records that satisfy the condition.
- The number of blocks that need to be examined to find a block containing the required record is $\mathbf{\log(b_r)}$.
- If the selection is on non-key attribute more than one block may contain required records and the cost of reading the extra blocks has to be added to the cost estimate.

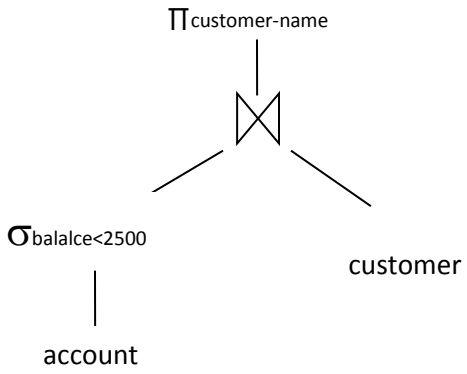
Explain various steps involved in query evaluation. *OR* Explain query evaluation process. *OR* Explain evaluation expression process in query optimization.

- There are two methods for the evaluation of expression
    1. Materialization
    2. Pipelining

Materialization

- In this method we start from bottom of the tree and each expression is evaluated one by one in bottom to top order. The result of each expression is materialized (stored) in temporary relation (table) for later use.

$\Pi_{customer\text{-}name}$ ($\sigma_{balalce<2500}$ (account) $\bowtie$ customer)



- In our example, there is only one such operation, selection operation on account.
- The inputs to the lowest level operation are relations in the database.
- We execute these operations and we store the results in temporary relations.
- We can use these temporary relations to execute the operation at the next level up in the tree, where the inputs now are either temporary relations or relations stored in the database.
- In our example the inputs to join are the customer relation and the temporary relation created by the selection on account.
- The join can now be evaluated, creating another temporary relation.
- By repeating the process, we will finally evaluate the operation at the root of the tree, giving the final result of the expression.
- In our example, we get the final result by executing the projection operation at the root of the tree, using as input the temporary relation created by the join. Evaluation just described is called materialized evaluation, since the results of each intermediate operation are created and then are used for evaluation of the next level operations.
- The cost of a materialized evaluation is not simply the sum of the costs of the operations involved. To compute the cost of evaluating an expression is to add the cost of all the operation as well as the cost of writing intermediate results to disk.
- The disadvantage of this method is that it will create temporary relation (table) and that relation is stored on disk which consumes space on disk.
- It evaluates one operation at a time, starting at the lowest level.

Pipelining

- We can reduce the number of temporary files that are produced by combining several relations operations into pipeline operations, in which the results of one operation are passed along to the next operation in the pipeline. Combining operations into a pipeline eliminates the cost reading and writing temporary relations.
- In this method several expression are evaluated simultaneously in pipeline by using the result of one operation passed to next without storing it in a temporary relation.

$\Pi_{customer\text{-}name}$ ($\sigma_{balalce<2500}$ (account) $\bowtie$ customer)

- First it will compute records having balance less than 2500 and then pass this result directly to project without storing that result in any temporary relation (table). And then by using this result it will compute the projections on customer-name.
- It is much cheaper than materialization because in this method no need to store a temporary relation to disk.
- Pipelining is not used in sort, hash joins.

Explain the method of query optimization. *OR* Explain query optimization process.

Query optimization

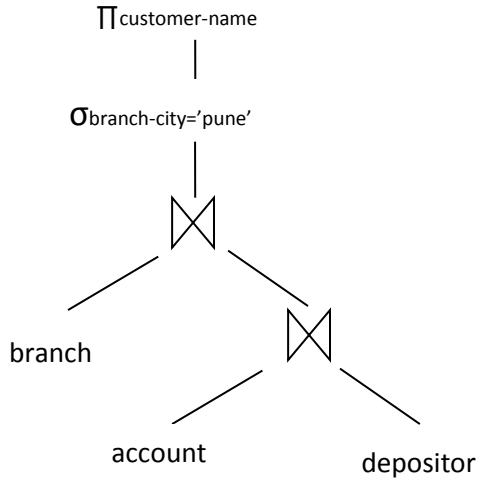
- It is a process of selecting the most efficient query evaluation plan from the available possible plans for processing a given query.
- There are two phases of query optimization
    1. Optimization which occur at the relational algebra level. In this phase the system will find an expression that is equivalent to the given expression but more efficient to execute.
    2. Selecting a detailed strategy for processing the query such as choosing algorithm and specific indices etc.
- For example consider the relational algebra expression for the query
- "Find the name of all customers who have account at any branch located in Pune"

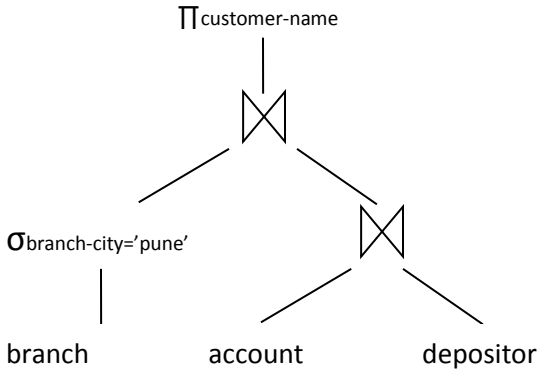
$\Pi_{customer\text{-}name}$ ($\sigma_{branch\text{-}city=\text{"pune"}}$ (branch $\bowtie$ (account $\bowtie$ depositor)))

- The above query may be written as below

$\Pi_{customer\text{-}name}$ ($\sigma_{branch\text{-}city=\text{"pune"}}$ (branch) $\bowtie$ (account $\bowtie$ depositor))

- In the second algebra expression the size of intermediate result is smaller than first because it will only contain the records of pune branch city. Final result of both the expression is same.





- To choose from the different query evaluation plan, the optimizer has to estimate the cost of each evaluation plan.
- Optimizer use statically information about the relation such as relation size and index depth to make a good estimate of the cost of a plan.

Explain transformation of relational expression to equivalent relational expression.

- Two relational algebra expressions are said to be equivalent (same) if on every legal database operation, the two expressions gives the same set of tuples (records). Sequence of records may be different but no of records must be same.

Equivalence rules

- This rule says that expressions of two forms are same.
- We can replace an expression of first form by an expression of the second form.
- The optimizer uses equivalence rule to transform expression into other logically same expression.
- We use

  θ1, θ2, θ3 and so on to denote condition
  L1, L2, L3 and so on to denote list of attributes (columns)
  E1, E2, E3 and so on to denote relational algebra expression.

Rules 1

- Combined selection operation can be divided into sequence of individual selections. This transformation is called cascade of σ.

  $\sigma_{\theta_1 \wedge \theta_2}(E) = \sigma_{\theta 1}(\sigma_{\theta 2}(E))$

Rules 2

- Selection operations are commutative.

  $\sigma_{\theta 1}(\sigma_{\theta 2}(E)) = \sigma_{\theta 2}(\sigma_{\theta 1}(E))$

Rules 3

- If more than one projection operation is used in expression then only the outer projection operation is required. So skip all the other inner projection operation.

  $\Pi_{L1}(\Pi_{L2}(\ldots(\Pi_{Ln}(E))\ldots)) = \Pi_{L1}(E)$

Rules 4

- Selection operation can be joined with Cartesian product and theta join.

  $\sigma_{\theta}(E1 \bowtie E2) = E1 \bowtie_{\theta} E2$

  $\sigma_{\theta 1}(E1 \bowtie_{\theta 2} E2) = E1 \bowtie_{\theta 1 \wedge \theta 2} E2$

Rules 5

- Theta operations are commutative.

  $E1 \bowtie_{\theta 2} E2 = E2 \bowtie_{\theta 2} E1$

Rules 6

- Natural join operations are associative.

  $(E1 \bowtie E2) \bowtie E3 = E1 \bowtie (E2 \bowtie E3)$

- Theta join operations are associative.

$(E1 \bowtie \theta 1 E2) \bowtie \theta 2 \Lambda \theta 3 E3 = E1 \bowtie \theta 1 \Lambda \theta 3 (E2 \bowtie \theta 2 E3)$

Rules 7

- The selection operation distribute over theta join operation under the following condition
  - ✓ It distribute when all the attributes in the selection condition θ0 involves only the attributes of the one of the expression (says E1) being joined.

    $\sigma\theta 0 (E1 \bowtie E2) = (\sigma\theta 0 (E1)) \bowtie \theta E2$
  - ✓ It distributes when the selection condition θ1involves only the attributes of E1 and θ2 involves only the attributes of E2.
  - ✓ $\sigma\theta 1 \Lambda \theta 2 (E1 \bowtie \theta E2) = (\sigma\theta 1(E1) \bowtie \theta (\sigma\theta 2 (E2)))$

Explain the purpose of sorting with example with reference to query optimization.

Purpose of Sorting

- Several of the relational operations such as joins can be implemented efficiently if the input relations are first sorted.
- We can sort a relation by building an index on the sort key and then using that index to read the relation in sorted order.
- Such a process orders the relation only logically rather than physically.
- Hence reading of tuples in the sorted order may lead to disk access for each record.
- So it is desirable to order the records physically.
- Sorting could be applied to both relations that fit entirely in main memory and for relations bigger than main memory.
- Sorting of relation that fit into main memory, standard sorting techniques such as quick-sort can be used.
- Sorting of relations that do not fit in main memory is called external sorting.
- Most commonly used algorithm for this type of sorting is external sort merge algorithm.
- It consists mainly two steps:
  - ✓ Step 1 (Create sorted runs)
    - Let *M* denotes memory size (in pages).
    - Create sorted runs. Let *i* be 0 initially.

      Repeatedly do the following till the end of the relation:

      (a) Read *M* blocks of relation into memory

      (b) Sort the in-memory blocks

      (c) Write sorted data to run R_i; increment *i.*

      Let the final value of *i* be *N*
  - ✓ Step 2 (Merge the runs)

- **Merge the runs (N-way merge)**. We assume (for now) that $N < M$.
  1) Use N blocks of memory to buffer input runs, and 1 block to buffer output. Read the first block of each run into its buffer page
  2) **repeat**
     I. Select the first record (in sort order) among all buffer pages
     II. Write the record to the output buffer. If the output buffer is full write it to disk.
     III. Delete the record from its input buffer page.
     If the buffer page becomes empty then read the next block (if any) of the run into the buffer.
  3) until all input buffer pages are empty:

- If $N \geq M$, several merge passes are required.
  - ✓ In each pass, contiguous groups of M - 1 runs are merged.
  - ✓ A pass reduces the number of runs by a factor of M -1, and creates runs longer by the same factor.
    - E.g. If M=11, and there are 90 runs, one pass reduces the number of runs to 9, each 10 times the size of the initial runs
  - ✓ Repeated passes are performed till all runs have been merged into one.
- The output of the merge stage is the sorted relation.
- The output file is buffered to reduce the number of disk operations.
- If the relation is much larger than memory there may be M or more runs generated in the first stage and it is not possible to allocate a page frame for each run during the merge stage.
- In this case merge operation proceed in multiple passes.
- Since there is enough memory for M-1 input buffer pages each merge can take M-1 runs as input.

Explain the measures of query cost, selection operation and join. *OR* Explain the measures of finding out the cost of a query in query processing.

Measures of query cost

- The cost of query evaluation can be measured in terms of a number of different resources including disk access, CPU time to execute a query and in a distributed or parallel database system the cost of communication.

- The response time for a query evaluation plan i.e the time required to execute the plan (assuming no other activity is going on) on the computer would account for all these activities.
- In large database system, however disk accesses are usually the most important cost, since disk access are slow compared to in memory operation.
- Moreover, CPU speeds have been improving much faster than have a disk speed.
- Therefore it is likely that the time spent in disk activity will continue to dominate the total time to execute a query.
- Estimating the CPU time is relatively hard, compared to estimating disk access cost.
- Therefore disk access cost a reasonable measure of the cost of a query evaluation plan.
- Disk access is the predominant cost (in terms of time) relatively easy to estimate; therefore number of block transfers from/to disk is typically used as measures.
- We use the number of block transfers from disk as a measure of actual cost.
- To simplify our computation, we assume that all transfer of blocks have same cost.
- To get more precise numbers we need to distinguish between sequential I/O where blocks read are contiguous on disk and random I/O where blocks are non-contiguous and an extra seek cost must be paid for each disk I/O operations.
- We also need to distinguish between read and write of blocks since it takes more time to write a block on disk than to read a block from disk.

Selection Operation

- There are two scan algorithms to implement the selection operation:
    1. Linear search
    2. Binary search

Linear search

- In a linear search, the system scans each file block and tests all records to see whether they satisfy the selection condition.
- For a selection on a key attribute, the system can terminate the scan if the requires record is found, without looking at the other records of the relation.
- The cost of linear search in terms of number of I/O operations is $\mathbf{b_r}$ where $\mathbf{b_r}$ is the number of blocks in file.
- Selection on key attribute has an average cost of $\mathbf{b_r/2}$.
- It may be a slower algorithm than any another algorithm.
- This algorithm can be applied to any file regardless of the ordering of file or the availability of indices or the nature of selection operation.

Binary search

- If the file is ordered on attribute and the selection condition is an equality comparison on the attribute, we can use a binary search to locate the records that satisfy the condition.

- The number of blocks that need to be examined to find a block containing the required record is **$\log(b_r)$.**
- If the selection is on non-key attribute more than one block may contain required records and the cost of reading the extra blocks has to be added to the cost estimate.

Join

- Like selection, the join operation can be implemented in a variety of ways.
- In terms of disk access, the join operation can be very expensive, so implementing and utilizing efficient join algorithms is critical in minimizing a query's execution time.
- For example consider

  depositor ⋈ customer
- We assume following information about the two above relations
  I. Number of records of customer $n_{customer}$ = 10,000
  II. Number of blocks of customer $b_{customer}$ = 400
  III. Number of records of depositor $n_{depositor}$ = 5,000
  IV. Number of blocks of depositor $b_{depositor}$ = 100
- There are four types of algorithms for join operations.
  1. Nested loop join
  2. Indexed nested loop join
  3. Merge join
  4. Hash join

What is database Index?

- Indexes are special lookup tables that the database search engine can use to speed up data retrieval.
- A database index is a data structure that improves the speed of data retrieval operations on a database table.
- An index in a database is very similar to an index in the back of a book.
- Indexes are used to retrieve data from the database very fast.
- The users cannot see the indexes, they are just used to speed up searches/queries.
- Updating a table with indexes takes more time than updating a table without (because the indexes also need an update).
- Syntax:

```
CREATE INDEX index_name
ON table_name (column1, column2, ...);
```

- Example:

```
CREATE INDEX idx_studentname
ON student (studentname);
```

- Indexing is a way to optimize the performance of a database by minimizing the number of disk accesses required when a query is processed.
- It is a data structure technique which is used to quickly locate and access the data in a database.

Explain the structure of Index in database.

- Indexes are created using a few database columns.

Search-key	Pointer

- The first column is the Search key that contains a copy of the primary key or candidate key of the table. These values are stored in sorted order so that the corresponding data can be accessed quickly.
- The second column is the Data Reference or Pointer which contains a set of pointers holding the address of the disk block where that particular key value can be found.

Explain different attributes of Indexing.

The indexing has various attributes:

- Access Types: This refers to the type of access such as value based search, range access, etc.
- Access Time: It refers to the time needed to find particular data element or set of elements.
- Insertion Time: It refers to the time taken to find the appropriate space and insert a new data.

- Deletion Time: Time taken to find an item and delete it as well as update the index structure.
- Space Overhead: It refers to the additional space required by the index.

Explain different Indexing Methods (Types).

Different indexing methods are:

- Primary Indexing
  - Dense Indexing
  - Parse Indexing
- Secondary Indexing
- Clustering Indexing

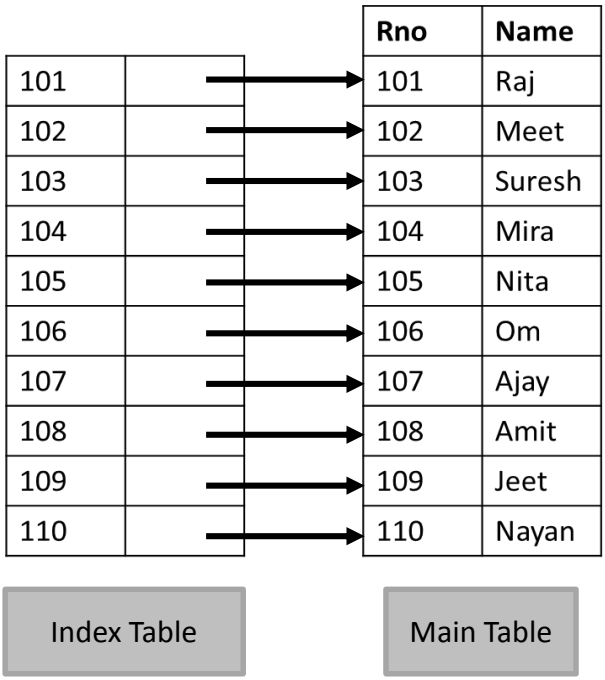
Primary Indexing

- If the index is created on the primary key of the table, then it is known as primary index.
- These primary keys are unique to each record.
- As primary keys are stored in sorted order, the performance of the searching operation is quite efficient.
- Student (RollNo, Name, Address, City, MobileNo) [RollNo is primary key]

```
CREATE INDEX idx_StudentRno
ON Student (RollNo);
```

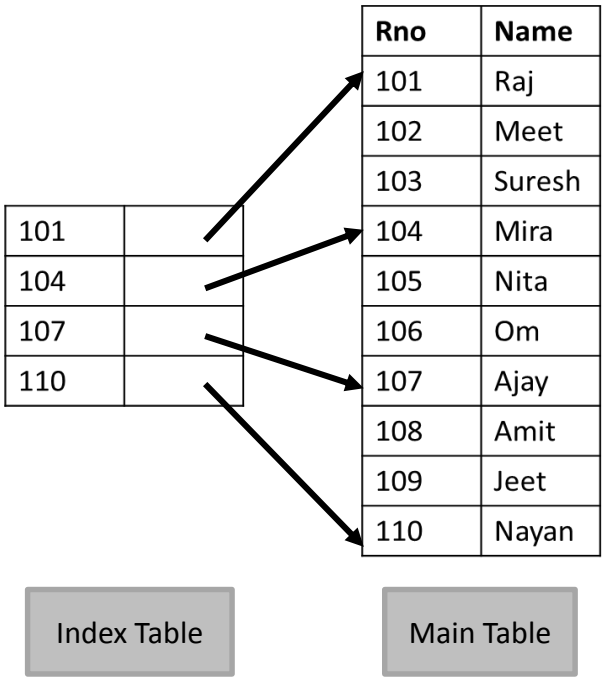
- The primary index can be classified into two types:
  - Dense index
  - Sparse index

Dense Index



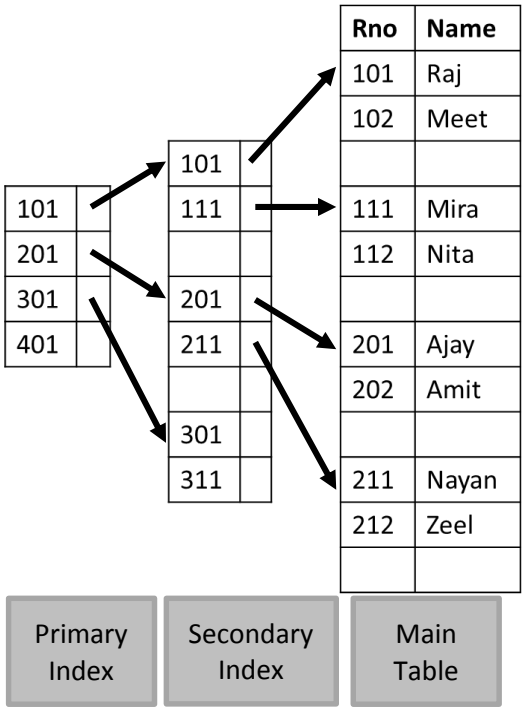
- In dense index, there is an index record for every search key value in the database.
- This makes searching faster but requires more space to store index records.
- In this, the number of records in the index table is same as the number of records in the main table.
- Index records contain search key value and a pointer to the actual record on the disk.

Sparse Index



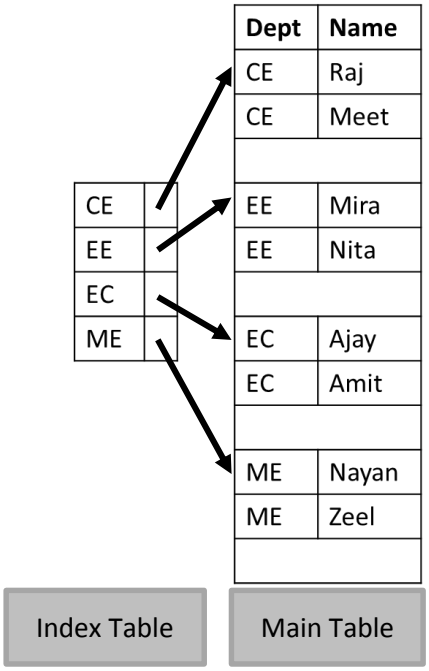
- In sparse index, index records are not created for every search key.
- The index record appears only for a few items in the data file.
- It requires less space, less maintenance overhead for insertion, and deletions but is slower compared to the dense index for locating records.
- To search a record in sparse index we search for a value that is less than or equal to value in index for which we are looking.
- After getting the first record, linear search is performed to retrieve the desired record.
- In the sparse indexing, as the size of the main table grows, the size of index table also grows.

Secondary Index



- In secondary indexing, to reduce the size of mapping, another level of indexing is introduced.
- In this method, the huge range for the columns is selected initially so that the mapping size of the first level becomes small.
- Then each range is further divided into smaller ranges.
- The mapping of the first level is stored in the primary memory, so that address fetch is faster.
- The mapping of the second level and actual data are stored in the secondary memory (hard disk).
- If you want to find the record of roll 112, then it will search the highest entry which is smaller than or equal to 112 in the first level index. It will get 101 at this level.
- Then in the second index level, again it does max (112) <= 112 and gets 111. Now using the address 111, it goes to the data block and starts searching each record till it gets 112.
- This is how a search is performed in this method.
- Inserting, updating or deleting is also done in the same manner.

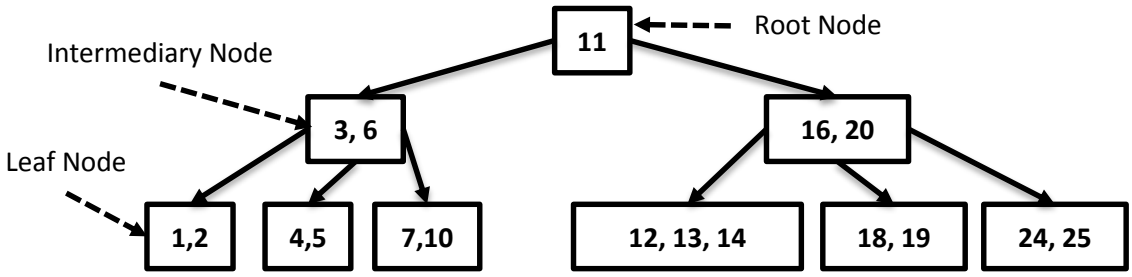
Clustering Index



- Sometimes the index is created on non-primary key columns which may not be unique for each record.
- In this case, to identify the record faster, we will group two or more columns to get the unique value and create index out of them. This method is called a clustering index.
- The records which have similar characteristics are grouped, and indexes are created for these group.

Explain B-tree.

- B-tree is a data structure that store data in its node in sorted order.
- We can represent sample B-tree as follows.



- B-tree stores data in such a way that each node contains keys in ascending order.
- Each of these keys has two references to another two child nodes.

- The left side child node keys are less than the current keys and the right side child node keys are greater than the current keys.

Searching a record in B-tree

- Suppose we want to search 18 in the above B tree structure.
- First, we will fetch for the intermediary node which will direct to the leaf node that can contain a record for 18.
- So, in the intermediary node, we will find a branch between 16 and 20 nodes.
- Then at the end, we will be redirected to the fifth leaf node. Here DBMS will perform a sequential search to find 18.

Explain hashing with its types.

- For a huge database, it can be almost next to impossible to search all the index values through all its level and then reach the destination data block to retrieve the desired data.
- Hashing is a technique to directly search the location of desired data on the disk without using index structure.
- Data is stored in the form of data blocks whose address is generated by applying a hash function in the memory location where these records are stored known as a data block or data bucket.
- Hashing uses hash functions with search keys as parameters to generate the address of a data record.
- Data bucket: Data buckets are the memory locations where the records are stored.
- Hash Function: Hash function is a mapping function that maps all the set of search keys to actual record address. Generally, hash function uses primary key to generate the hash index – address of the data block.

Types of hashing methods

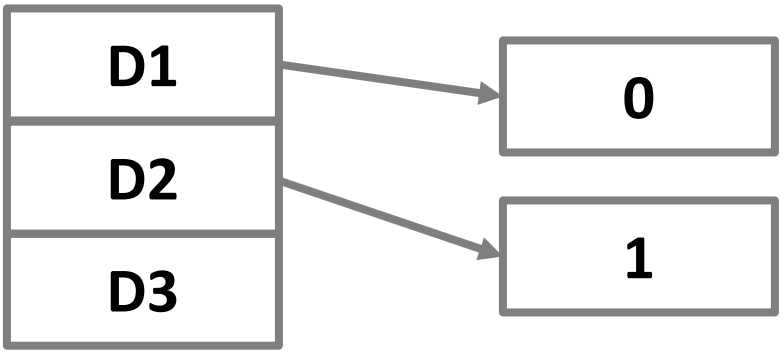
1. Static hashing
2. Dynamic hashing

Static hashing

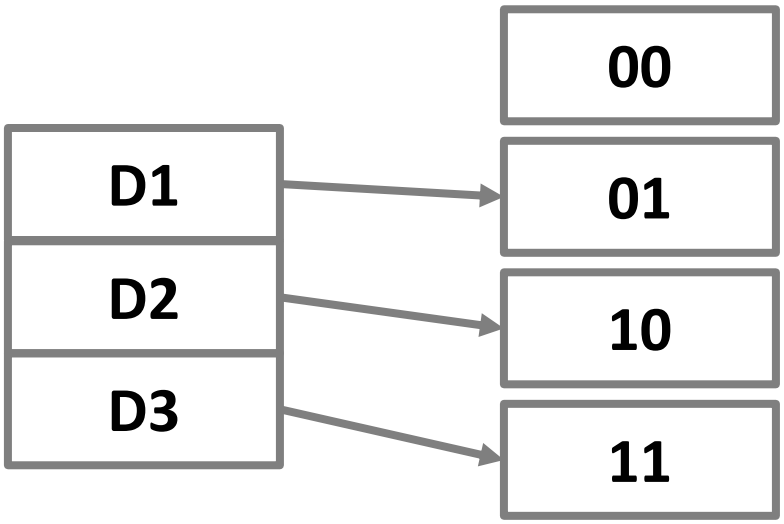
- In the static hashing, the resultant data bucket address will always remain the same.
- Therefore, if you generate an address for say Student_ID = 10 using hashing function mod(3), the resultant bucket address will always be 1. So, you will not see any change in the bucket address.
- Therefore, in this static hashing method, the number of data buckets in memory always remains constant.

Dynamic hashing

- The drawback of static hashing is that that it does not expand or shrink dynamically as the size of the database grows or shrinks.
- In dynamic hashing, data buckets grows or shrinks (added or removed dynamically) as the records increases or decreases.
- Dynamic hashing is also known as extended hashing.
- In dynamic hashing, the hash function is made to produce a large number of values.
- For Example, there are three data records D1, D2 and D3 .
- The hash function generates three addresses 1001, 0101 and 1010 respectively.
- This method of storing considers only part of this address – especially only first one bit to store the data.
- So it tries to load three of them at address 0 and 1.



- But the problem is that no bucket address is remaining for D3.
- The bucket has to grow dynamically to accommodate D3.
- So it changes the address have 2 bits rather than 1 bit, and then it updates the existing data to have 2 bit address.
- Then it tries to accommodate D3.


D1
D2
D3
00
01
10
11

What is transaction? List and explain ACID property of transaction with example.

Transaction

- A transaction is a part of program execution that accesses and updates various data items.
- A transaction can be defined as a group of tasks in which a single task is the minimum processing unit of work, which cannot be divided further.
- A transaction is a logical unit of work that contains one or more SQL statements.
- A transaction is an atomic unit (transaction either complete 0% or 100%).
- A database transaction must be atomic, meaning that it must be either entirely completed or aborted.

ACID property

Atomicity

- Either all operations of the transaction are properly reflected in the database or none are.
- Means either all the operations of a transaction are executed or not a single operation is executed.
- For example consider below transaction to transfer Rs. 50 from account A to account B:
    1. **read**(*A*)
    2. *A* := *A* – 50
    3. **write**(*A*)
    4. **read**(*B*)
    5. *B* := *B* + 50
    6. **write**(*B)*
- In above transaction if Rs. 50 is deducted from account A then it must be added to account B.

Consistency

- Execution of a transaction in isolation preserves the consistency of the database.
- Means our database must remain in consistent state after execution of any transaction.
- In above example total of A and B must remain same before and after the execution of transaction.

Isolation

- Although multiple transactions may execute concurrently, each transaction must be unaware of other concurrently executing transactions.
- Intermediate transaction results must be hidden from other concurrently executed transactions.
- In above example once your transaction start from step one its result should not be access by any other transaction until last step (step 6) is completed.

Durability

- After a transaction completes successfully, the changes it has made to the database persist, even if there are system failures.
- Once your transaction completed up to step 6 its result must be stored permanently. It should not be removed if system fails.

Explain different states in transaction processing in database.

OR

Explain State Transition Diagram (Transaction State Diagram).

- Because failure of transaction may occur, transaction is broken up into states to handle various situations.
- Following are the different states in transaction processing in database
  - ✓ Active
  - ✓ Partial committed
  - ✓ Failed
  - ✓ Aborted
  - ✓ Committed

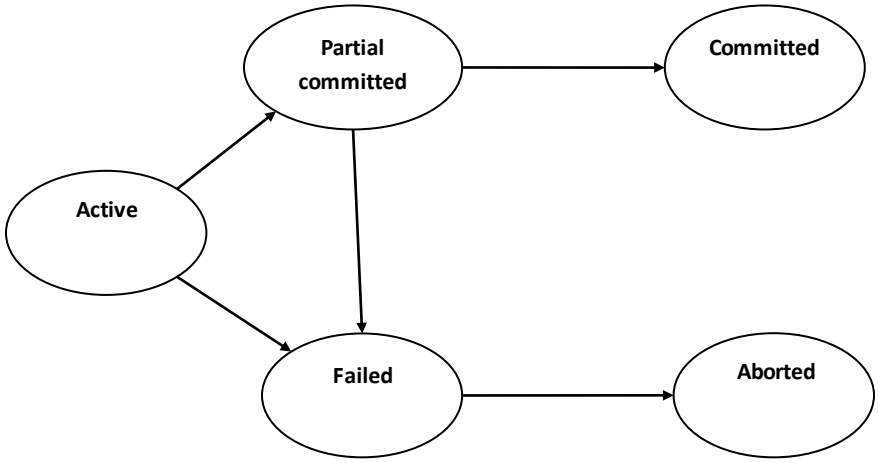


Fig. State Transition Diagram

Active

- This is the initial state. The transaction stays in this state while it is executing.

Partially Committed

- This is the state after the final statement of the transaction is executed.
- At this point failure is still possible since changes may have been only done in main memory, a hardware failure could still occur.
- The DBMS needs to write out enough information to disk so that, in case of a failure, the system could re-create the updates performed by the transaction once the system is brought back up.
- After it has written out all the necessary information, it is committed.

Failed

- After the discovery that normal execution can no longer proceed.
- Once a transaction cannot be completed, any changes that it made must be undone rolling it back.

Aborted

- The state after the transaction has been rolled back and the database has been restored to its state prior to the start of the transaction.

Committed

- The transaction enters in this state after successful completion of the transaction.
- We cannot abort or rollback a committed transaction.

Explain following terms.

Schedule

- A schedule is the chronological (sequential) order in which instructions are executed in a system.
- A schedule for a set of transaction must consist of all the instruction of those transactions and must preserve the order in which the instructions appear in each individual transaction.
- Example of schedule (Schedule 1)

T1	**T2**
read(A) A:=A-50 write(A) read(B) B:= B+ 50 write(B)	
	read(A) temp: A * 0.1 A: A-temp write (A) read(B) B:=B +temp write(B)

Serial schedule

- Schedule that does not interleave the actions of different transactions.
- In schedule 1 the all the instructions of T1 are grouped and run together. Then all the instructions of T2 are grouped and run together.
- Means schedule 2 will not start until all the instructions of schedule 1 are complete. This type of schedules is called serial schedule.

Interleaved schedule

- Schedule that interleave the actions of different transactions.
- Means schedule 2 will start before all instructions of schedule 1 are completed. This type of schedules is called interleaved schedule.

T1	**T2**
read(A)	
A:=A-50	
write(A)	
	read(A)
	temp: A * 0.1
	A: A-temp
	write (A)
read(B)	
B:= B+ 50	
write(B)	
	read(B)
	B:=B +temp
	write(B)

Equivalent schedules

- Two schedules are equivalent schedule if the effect of executing the first schedule is identical (same) to the effect of executing the second schedule.
- We can also say that two schedule are equivalent schedule if the output of executing the first schedule is identical (same) to the output of executing the second schedule.

Serializable schedule

- A schedule that is equivalent (in its outcome) to a serial schedule has the serializability property.
- Example of serializable schedule

Schedule 1		
T1	**T2**	**T3**
read(X)		
write(X)		
	read(Y)	
	write(Y)	
		read(Z)
		write(Z)

Schedule 2		
T1	**T2**	**T3**
read(X)		
	read(Y)	
		read(Z)
write(X)		
	write(Y)	
		write(Z)

- In above example there are two schedules as schedule 1 and schedule 2.
- In schedule 1 and schedule 2 the order in which the instructions of transaction are executed is not the same but whatever the result we get is same. So this is known as serializability of transaction.

Explain serializability of transaction. ***OR***
Explain both the forms of serializability with example. Also explain relation between two forms. ***OR***
Explain conflict serializability and view serializability with example.

Conflict serializability

- Instructions I_i and I_j of transactions T_i and T_j respectively, conflict if and only if there exists some item Q accessed by both I_i and I_j, and at least one of these instructions wrote Q.
    1. If I_i and I_j access different data item then I_i and I_j don't conflict.
    2. I_i = read(Q), I_j = read(Q). I_i and I_j don't conflict.
    3. I_i = read(Q), I_j = write(Q). I_i and I_j conflict.
    4. I_i = write(Q), I_j = read(Q). I_i and I_j conflict.
    5. I_i = write(Q), I_j = write(Q). I_i and I_j conflict.
- Intuitively, a conflict between I_i and I_j forces a (logical) temporal order between them.
- If a schedule *S* can be transformed into a schedule *S´* by a series of swaps of non-conflicting instructions, we say that *S* and *S´* are conflict equivalent.
- We say that a schedule *S* is conflict serializable if it is conflict equivalent to a serial schedule.

Example

- Schedule S can be transformed into Schedule S' by swapping of non-conflicting series of instructions. Therefore Schedule S is conflict serializable.

Schedule S	
T1	**T2**
read(A)	
write(A)	
	read(A)
	write(A)
read(B)	
write(B)	
	read(B)
	write(B)

Schedule S'	
T1	**T2**
read(A)	
write(A)	
read(B)	
write(B)	
	read(A)
	write(A)
	read(B)
	write(B)

- Instruction I_i of transaction T1 and I_j of transaction T2 conflict if both of these instruction access same data A and one of these two instructions performs write operation on that data (A).
- In above example the write(A) instruction of transaction T1 conflict with read(A) instruction of transaction T2 because both the instructions access same data A. But write(A) instruction of transaction T2 is not conflict with read(B) instruction of transaction T1 because both the instructions access different data. Transaction T2 performs write operation in A and transaction T1 is reading B.

- So in above example in schedule S two instructions read(A) and write(A) of transaction T2 and two instructions read(B) and write(B) of transaction T1 are interchanged and we get schedule S'.
- Therefore Schedule S is conflict serializable.

Schedule S''	
T3	**T4**
read(Q)	
	write(Q)
write(Q)	

- We are unable to swap instructions in the above schedule S'' to obtain either the serial schedule < T_3, T_4 >, or the serial schedule < T_4, T_3 >.
- So above schedule S'' is not conflict serializable.

View serializability

- Let *S* and *S´* be two schedules with the same set of transactions. *S* and *S´* are view equivalent if the following three conditions are met, for each data item *Q,*
  1. If in schedule S, transaction T_i reads the initial value of Q, then in schedule S' also transaction T_i must read the initial value of Q.
  2. If in schedule S transaction T_i executes read(Q), and that value was produced by transaction T_j (if any), then in schedule S' also transaction T_i must read the value of Q that was produced by the same write(Q) operation of transaction T_j .
  3. The transaction T_i (if any) that performs the final write(Q) operation in schedule S then in schedule S' also the final write(Q) operation must be performed by T_i.
- A schedule *S* is view serializable if it is view equivalent to a serial schedule.
- Every conflict serializable schedule is also view serializable but every view serializable is not conflict serializable.
- Below is a schedule which is view serializable but not conflict serializable.

Schedule S		
T3	**T4**	**T6**
read(Q)		
	write(Q)	
write(Q)		
		write(Q)

- Above schedule is view serializable but not conflict serializable because all the transactions can use same data item (Q) and all the operations are conflict with each other due to one operation is write on data item (Q) and that's why we cannot interchange any non conflict operation of any transaction.

Explain two phase commit protocol. *OR* Explain working of two phase commit protocol.

Two phase commit protocol

- The two phase commit protocol provides an automatic recovery mechanism in case a system or media failure occurs during execution of the transaction.
- The two phase commit protocol ensures that all participants perform the same action (either to commit or to roll back a transaction).
- The two phase commit strategy is designed to ensure that either all the databases are updated or none of them, so that the databases remain synchronized.
- In two phase commit protocol there is one node which is act as a coordinator and all other participating node are known as cohorts or participant.
- Coordinator – the component that coordinates with all the participants.
- Cohorts (Participants) – each individual node except coordinator are participant.
- As the name suggests, the two phase commit protocol involves two phases.
    1. The first phase is Commit Request phase OR phase 1
    2. The second phase is Commit phase OR phase 2

Commit Request Phase (Obtaining Decision)

- To commit the transaction, the coordinator sends a request asking for **"**ready for commit**"** to each cohort.
- The coordinator waits until it has received a reply from all cohorts to **"**vote**"** on the request.
- Each participant votes by sending a message back to the coordinator as follows:
    - ✓ It votes YES if it is prepared to commit
    - ✓ It may vote NO for any reason if it cannot prepare the transaction due to a local failure.
    - ✓ It may delay in voting because cohort was busy with other work.

Commit Phase (Performing Decision)

- If the coordinator receives YES response from all cohorts, it decides to commit. The transaction is now officially committed. Otherwise, it either receives a NO response or gives up waiting for some cohort, so it decides to abort.
- The coordinator sends its decision to all participants (i.e. COMMIT or ABORT).
- Participants acknowledge receipt of commit or about by replying DONE.

What is system recovery?

- Database recovery is the process of restoring a database to the correct state in the event of a failure.
- Database recovery is a service that is provided by the DBMS to ensure that the database is reliable and remain in consistent state in case of a failure.
- Restoring a physical backup means reconstructing it and making it available to the database server.

- To recover a restored backup, data is updated using redo command after the backup was taken.
- Database server such as SQL server or ORACLE server performs cash recovery and instance recovery automatically after an instance failure.
- In case of media failure, a database administrator (DBA) must initiate a recovery operation.
- Recovering a backup involves two distinct operations: rolling the backup forward to a more recent time by applying redo data and rolling back all changes made in uncommitted transactions to their original state.
- In general, recovery refers to the various operations involved in restoring, rolling forward and rolling back a backup.
- Backup and recovery refers to the various strategies and operations involved in protecting the database against data loss and reconstructing the database.

Explain Log based recovery method.

Log based recovery

- The most widely used structure for recording database modification is the log.
- The log is a sequence of log records, recording all the update activities in the database.
- In short Transaction log is a journal or simply a data file, which contains history of all transaction performed and maintained on stable storage.
- Since the log contains a complete record of all database activity, the volume of data stored in the log may become unreasonable large.
- For log records to be useful for recovery from system and disk failures, the log must reside on stable storage.
- Log contains
    1. Start of transaction
    2. Transaction-id
    3. Record-id
    4. Type of operation (insert, update, delete)
    5. Old value, new value
    6. End of transaction that is committed or aborted.
- All such files are maintained by DBMS itself. Normally these are sequential files.
- Recovery has two factors Rollback (Undo) and Roll forward (Redo).
- When transaction T_i starts, it registers itself by writing a $<T_i$ start>log record
- Before T_i executes write(X), a log record $<T_i, X, V_1, V_2>$ is written, where V_1 is the value of X before the write, and V_2 is the value to be written to X.
    - ✓ Log record notes that T_i has performed a write on data item X_j
    - ✓ X_j had value V_1 before the write, and will have value V_2 after the write.
- When T_i finishes it last statement, the log record $<T_i$ commit> is written.
- Two approaches are used in log based recovery
    1. Deferred database modification
    2. Immediate database modification

Log based Recovery Techniques

- Once a failure occurs, DBMS retrieves the database using the back-up of database and transaction log. Various log based recovery techniques used by DBMS are as per below:
    1. **Deferred Database Modification**
    2. **Immediate Database Modification**
- Both of the techniques use transaction logs. These techniques are explained in following sub-sections.

Explain Deferred Database Modification log based recovery method.

Concept

- Updates (changes) to the database are deferred (or postponed) until the transaction commits.
- During the execution of transaction, updates are recorded only in the transaction log and in buffers. After the transaction commits, these updates are recorded in the database.

When failure occurs

- If transaction has not committed, then it has not affected the database. And so, no need to do any undoing operations. Just restart the transaction.
- If transaction has committed, then, still, it may not have modified the database. And so, redo the updates of the transaction.

Transaction Log

- In this technique, transaction log is used in following ways:
- Transaction T starts by writing <T start> to the log.
- Any update is recorded as <T, X, V>, where V indicates new value for data item X. Here, no need to preserve old value of the changed data item. Also, V is not written to the X in database, but it is deferred.
- Transaction T commits by writing <T commit> to the log. Once this is entered in log, actual updates are recorded to the database.
- If a transaction T aborts, the transaction log record is ignored, and no any updates are recorded to the database.

Example

- Consider the following two transactions, T_0 and T_1 given in figure, where T_0 executes before T_1. Also consider that initial values for A, B and C are 500, 600 and 700 respectively.

Transaction – T_0	Transaction – T_1
Read (A) A =A -100 Write (A) Read (B) B =B+ 100 Write (B)	Read (C) C=C-200 Write (C)

- The following figure shows the transaction log for above two transactions at three different instances of time.

Time Instance (a)	Time Instance (b)	Time Instance (c)
<T_0 start> < T_0, A, 400> < T_0, B, 700>	< T_0 start> < T_0, A, 400> < T_0, B, 700> < T_0 commit> <T_1 start> <T_1, C, 500>	< T_0 start> < T_0, A, 400> < T_0, B, 700> < T_0 commit> < T_1 start> < T_1, C, 500> < T_1 commit>

- If failure occurs in case of
  1. No any REDO actions are required.
  2. As Transaction T_0 has already committed, it must be redone.
  3. As Transactions T_0 and T_1 have already committed, they must be redone.

Explain Immediate Database Modification log based recovery method.

Concept

- Updates (changes) to the database are applied immediately as they occur without waiting to reach to the commit point.
- Also, these updates are recorded in the transaction log.
- It is possible here that updates of the uncommitted transaction are also written to the database. And, other transactions can access these updated values.

When failure occurs

- If transaction has not committed, then it may have modified the database. And so, undo the updates of the transaction.
- If transaction has committed, then still it may not have modified the database. And so, redo the updates of the transaction.

Transaction Log

- In this technique, transaction log is used in following ways:
- Transaction T starts by writing <T start> to the log.
- Any update is recorded as <T, X, V_{old}, V_{new} > where V_{old} indicates the original value of data item X and V_{new} indicates new value for X. Here, as undo operation is required, it requires preserving old value of the changed data item.
- Transaction T commits by writing <T commit> to the log.
- If a transaction T aborts, the transaction log record is consulted, and required undo operations are performed.

Example

- Again, consider the two transactions, T_0 and T_1, given in figure, where T_0 executes before T_1.
- Also consider that initial values for A, B and C are 500, 600 and 700 respectively.

- The following figure shows the transaction log for above two transactions at three different instances of time. Note that, here, transaction log contains original values also along with new updated values for data items.
- If failure occurs in case of -
- Undo the transaction T0 as it has not committed, and restore A and B to 500 and 600 respectively.
- Undo the transaction T1, restore C to 700; and, Redo the Transaction T0 set A and B to 400 and 700 respectively.
- Redo the Transaction T0 and Transaction T0; and, set A and B to 400 and 700 respectively, while set C to 500.

Time Instance (a)	Time Instance (b)	Time Instance (c)
<T_0 start> < T_0, A, 500, 400> < T_0, B, 600, 700>	< T_0 start> < T_0, A, 500, 400> < T_0, B, 600, 700> < T_0 commit> <T_1 start> <T_1, C, 700, 500>	< T_0 start> < T_0, A, 500, 400> < T_0, B, 600, 700> < T_0 commit> < T_1 start> < T_1, C, 700, 500> < T_1 commit>

Explain system recovery procedure with Checkpoint record concept.

Problems with Deferred & Immediate Updates

- Searching the entire log is time-consuming.
- It is possible to redo transactions that have already been stored their updates to the database.

Checkpoint

- A point of synchronization between database and transaction log file.
- Specifies that any operations executed before this point are done correctly and stored safely.
- At this point, all the buffers are force-fully written to the secondary storage.
- Checkpoints are scheduled at predetermined time intervals
- Used to limit -
    1. The size of transaction log file
    2. Amount of searching, and
    3. Subsequent processing that is required to carry out on the transaction log file.

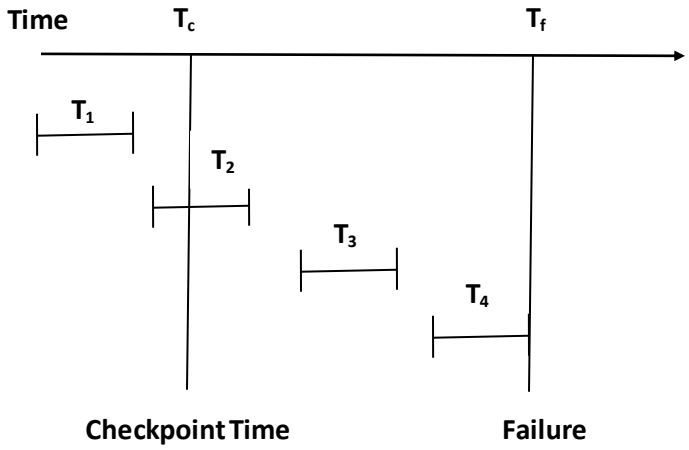
When failure occurs

- Find out the nearest checkpoint.
- If transaction has already committed before this checkpoint, ignore it.
- If transaction is active at this point or after this point and has committed before failure, redo that transaction.
- If transaction is active at this point or after this point and has not committed, undo that

transaction.

Example

- Consider the transactions given in following figure. Here, T_c indicates checkpoint, while T_f indicates failure time.
- Here, at failure time -
  1. Ignore the transaction T_1 as it has already been committed before checkpoint.
  2. Redo transaction T_2 and T_3 as they are active at/after checkpoint, but have committed before failure.
  3. Undo transaction T_4 as it is active after checkpoint and has not committed.



Explain Shadow Paging Technique.

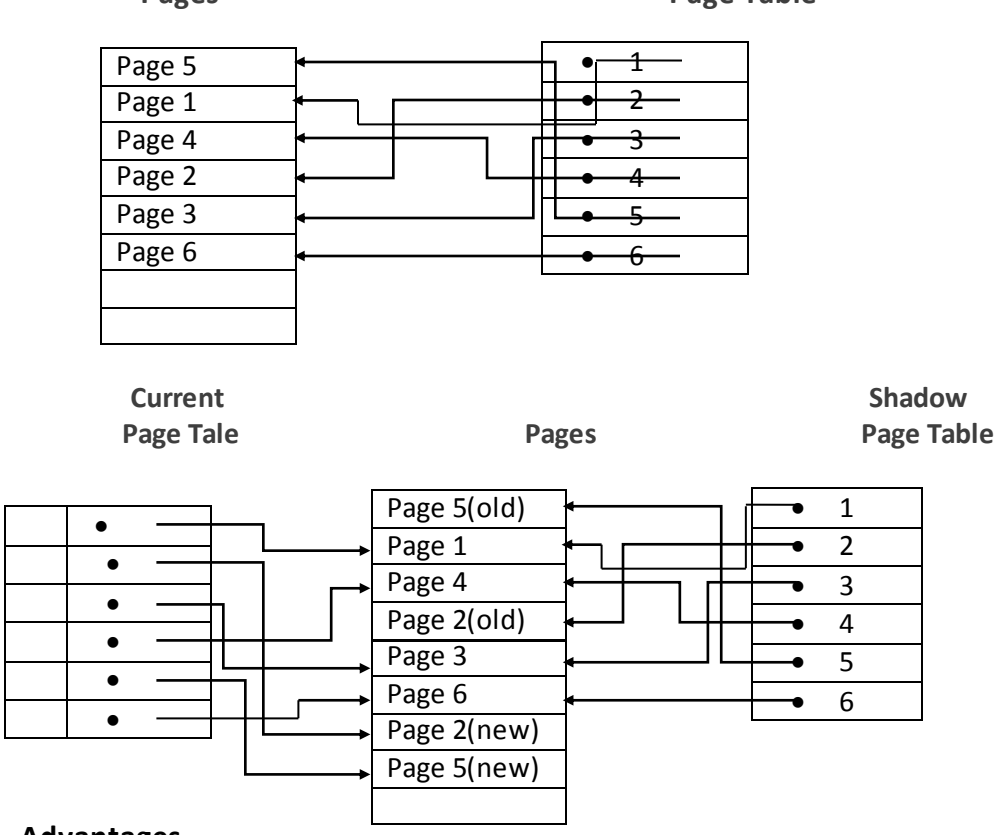
Concept

- Shadow paging is an alternative to transaction-log based recovery techniques.
- Here, database is considered as made up of fixed size disk blocks, called pages. These pages are mapped to physical storage using a table, called page table.
- The page table is indexed by a page number of the database. The information about physical pages, in which database pages are stored, is kept in this page table.
- This technique is similar to paging technique used by Operating Systems to allocate memory, particularly to manage virtual memory.
- The following figure depicts the concept of shadow paging.

Execution of Transaction

- During the execution of the transaction, two page tables are maintained.
  1. **Current Page Table**: Used to access data items during transaction execution.
  2. **Shadow Page Table**: Original page table, and will not get modified during transaction execution.
- Whenever any page is about to be written for the first time
  1. A copy of this page is made onto an free page,
  2. The current page table is made to point to the copy,
  3. The update is made on this copy.

- At the start of the transaction, both tables are same and point· to same pages.
- The shadow page table is never changed, and is used to restore the database in case of any failure occurs. However, current page table entries may change during transaction execution, as it is used to record all updates made to the database.
- When the transaction completes, the current page table becomes shadow page table. At this time, it is considered that the transaction has committed.
- The following figure explains working of this technique.
- As shown in this figure, two pages - page 2 & 5 - are affected by a transaction and copied to new physical pages. The current page table points to these pages.
- The shadow page table continues to point to old pages which are not changed by the transaction. So, this table and pages are used for undoing the transaction.



Advantages

- No overhead of maintaining transaction log.
- Recovery is quite faster, as there is no any redo or undo operations required.

Disadvantages

- Copying the entire page table is very expensive.
- Data are scattered or fragmented.
- After each transaction, free pages need to be collected by garbage collector. Difficult to extend this technique to allow concurrent transactions.

What is concurrency? What are the methods to control concurrency?

Concurrency

- Concurrency is the ability of a database to allow multiple (more than one) users to access data at the same time.

Methods to control concurrency (Mechanisms)

- **Optimistic** - Delay the checking of whether a transaction meets the isolation and other integrity rules (e.g., serializability and recoverability) until its end, without blocking any of its (read, write) operations and then abort a transaction to prevent the violation, if the desired rules are to be violated upon its commit. An aborted transaction is immediately restarted and re-executed, which incurs an obvious overhead. If not too many transactions are aborted, then being optimistic is usually a good strategy.
- **Pessimistic** - Block an operation of a transaction, if it may cause violation of the rules, until the possibility of violation disappears. Blocking operations is typically involved with performance reduction.
- **Semi-optimistic** - Block operations in some situations, if they may cause violation of some rules, and do not block in other situations while delaying rules checking (if needed) to transaction's end, as done with optimistic.

Methods to control concurrency (Methods)

- **Locking (Two-phase locking - 2PL)** - Controlling access to data by locks assigned to the data. Access of a transaction to a data item (database object) locked by another transaction may be blocked (depending on lock type and access operation type) until lock release.
- **Serialization graph checking (also called Serializability, or Conflict, or Precedence graph checking)** - Checking for cycles in the schedule's graph and breaking them by aborts.
- **Timestamp ordering (TO)** - Assigning timestamps to transactions, and controlling or checking access to data by timestamp order.
- **Commitment ordering (Commit ordering or CO)** - Controlling or checking transactions' chronological order of commit events to be compatible with their respective precedence order.

What are the three problems due to concurrency? How the problems can be avoided.

Three problems due to concurrency

1. **The lost update problem**: This problem indicate that if two transactions T1 and T2 both read the same data and update it then effect of first update will be overwritten by the second update.

T1	Time	T2
---	T0	---
Read X	T1	---
---	T2	Read X
Update X	T3	---
---	T4	Update X
---	T5	---

How to avoid: In above example a transaction T2 must not update the data item (X) until the transaction T1 can commit data item (X).

2. **The dirty read problem**: The dirty read arises when one transaction update some item and then fails due to some reason. This updated item is retrieved by another transaction before it is changed back to the original value.

T1	Time	T2
---	T0	---
---	T1	Update X
Read X	T2	---
---	T3	Rollback
---	T5	---

How to avoid: In above example a transaction T1 must not read the data item (X) until the transaction T2 can commit data item (X).

3. **The incorrect retrieval problem**: The inconsistent retrieval problem arises when one transaction retrieves data to use in some operation but before it can use this data another transaction updates that data and commits. Through this change will be hidden from first transaction and it will continue to use previous retrieved data. This problem is also known as inconsistent analysis problem.

Balance (A=200 B=250 C=150)		
T1	**Time**	**T2**
---	T0	---
Read (A) Sum ← 200	T1	---
Read (B) Sum ← Sum + 250 = 450	T2	---
---	T3	Read (C)
---	T4	Update (C) 150 → 150 – 50 = 100
---	T5	Read (A)
---	T6	Update (A) 200 → 200 + 50 = 250
---	T7	COMMIT
Read (C) Sum ←Sum + 100 = 550	T8	---

How to avoid: In above example a transaction T2 must not read or update data item (X) until the transaction T1 can commit data item (X).

What is concurrency control? Why Concurrency control is needed?

Concurrency control

- The technique is used to protect data when multiple users are accessing (using) same data concurrently (at same time) is called concurrency control.

Concurrency control needed

- If transactions are executed serially, i.e., sequentially with no overlap in time, no transaction concurrency exists. However, if concurrent transactions with interleaving operations are allowed in an uncontrolled manner, some unexpected, undesirable result may occur. Here are some typical examples:
  1. **The lost update problem**: This problem indicates that if two transactions T1 and T2 both read the same data and update it then effect of first update will be overwritten by the second update.
  2. **The dirty read problem**: The dirty read arises when one transaction updates some item and then fails due to some reason. This updated item is retrieved by another transaction before it is changed back to the original value.
  3. **The incorrect retrieval problem**: The inconsistent retrieval problem arises when one transaction retrieves data to use in some operation But before it can use this data another transaction update that data and commits. Through this change will be hidden from first transaction and it will continue to use previous retrieved data. This problem is also known as inconsistent analysis problem.
- Most high-performance transactional systems need to run transactions concurrently to meet their performance requirements. Thus, without concurrency control such systems can neither provide correct results nor maintain their databases consistent.

Define lock. Define locking. Explain lock based protocol.

Lock

- A lock is a variable associated with data item to control concurrent access to that data item.
- Lock requests are made to concurrency-control manager.
- Transaction can proceed only after request is granted.

Locking

- One major problem in databases is concurrency.
- Concurrency problems arise when multiple users try to update or insert data into a database table at the same time. Such concurrent updates can cause data to become corrupt or inconsistent.
- Locking is a strategy that is used to prevent such concurrent updates to data.

Lock based protocol

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

- Data items can be locked in two modes :
    1. Exclusive (X) mode. Data item can be both read as well as written. X-lock is requested using lock-X instruction.
    2. Shared (S) mode. Data item can only be read. S-lock is requested using lock-S instruction.
- Lock requests are made to concurrency-control manager.
- Transaction can proceed only after request is granted.
- Lock-compatibility matrix

	S	**X**
S	TRUE	FALSE
X	FALSE	FALSE

- A transaction may be granted a lock on an item if the requested lock is compatible with locks already held on the item by other transactions
- Any number of transactions can hold shared locks on an item, but if any transaction holds an exclusive on the item no other transaction may hold any lock on the item.
- If a lock cannot be granted, the requesting transaction is made to wait till all incompatible locks held by other transactions have been released. The lock is then granted.

T1	**T2**	**Concurrency Control Manager**
Lock-X(B)		
		Grant-X(B,T1)
Read(B)		
B=B-50		
Write(B)		
Unlock(B)		
	Lock-S(A)	
		Grant-S(A,T2)
	Read (A)	
	Unlock(A)	
	Lock-S(B)	
		Grant-S(B,T2)
	Read (B)	
	Unlock(B)	
	Display(A+B)	
Lock-X(A)		
		Grant-X(A,T1)
Read(A)		
A=A+50		
Write(A)		
Unlock(A)		

- This locking protocol divides transaction execution phase into three parts.
  1. In the first part, when transaction starts executing, transaction seeks grant for locks it needs as it executes.
  2. Second part is where the transaction acquires all locks and no other lock is required. Transaction keeps executing its operation.
  3. As soon as the transaction releases its first lock, the third phase starts. In this phase a transaction cannot demand for any lock but only releases the acquired locks.

Explain two phase locking protocol. What are its advantages and disadvantages? *OR* Explain two phase locking. Explain its advantage and disadvantage.

Two-Phase Locking Protocol

- The use of locks has helped us to create neat and clean concurrent schedule.
- The Two Phase Locking Protocol defines the rules of how to acquire the locks on a data item and how to release the locks.
- Two phase locking (2PL) is a concurrency control method that guarantees serializability.
- The protocol utilizes locks, applied by a transaction on data, which may block (stop) other transactions from accessing the same data during the transaction's life.
- The Two Phase Locking Protocol assumes that a transaction can only be in one of two phases.

Phase 1 - Growing Phase

- ✓ In this phase the transaction can only acquire locks, but cannot release any lock.
- ✓ The transaction enters the growing phase as soon as it acquires the first lock it wants.
- ✓ From now on it has no option but to keep acquiring all the locks it would need.
- ✓ It cannot release any lock at this phase even if it has finished working with a locked data item.
- ✓ Ultimately the transaction reaches a point where all the lock it may need has been acquired. This point is called **Lock Point**.

Phase 2 - Shrinking Phase

- ✓ After Lock Point has been reached, the transaction enters the shrinking phase.

- ✓ In this phase the transaction can only release locks, but cannot acquire any new lock.
- ✓ The transaction enters the shrinking phase as soon as it releases the first lock after crossing the Lock Point.
- ✓ From now on it has no option but to keep releasing all the acquired locks.

- Initially the transaction is in growing phase, that is the transaction acquires locks as needed.
- Once the transaction releases lock, it enters the shrinking phase and no more lock request may be issued.
- Upgrading of lock is not possible in shrinking phase, but it is possible in growing phase.
- The two phase locking protocol ensures serializability.
- There are two different versions of the Two Phase Locking Protocol.
    1) Strict Two Phase Locking Protocol
    2) Rigorous Two Phase Locking Protocol.

Explain strict two phase locking. Explain its advantage and disadvantage.

Strict Two Phase Locking Protocol

- In this protocol, a transaction may release all the shared locks after the Lock Point has been reached, but it cannot release any of the exclusive locks until the transaction commits or aborts.
- This ensures that any data is written by an uncommitted transaction are locked in exclusive mode until the transaction commits and preventing other transaction from reading that data.
- This protocol solves dirty read problem.

Rigorous Two Phase Locking Protocol

In Rigorous Two Phase Locking Protocol, a transaction is not allowed to release any lock (either shared or exclusive) until it commits. This means that until the transaction commits, other transaction might acquire a shared lock on a data item on which the uncommitted transaction has a shared lock; but cannot acquire any lock on a data item on which the uncommitted transaction has an exclusive lock.

Advantages

- Recovery is very easy.

Disadvantages

- Concurrency is reduced

Explain time stamp based protocol.

Time stamp based protocol

- This protocol uses either system time or logical counter to be used as a time-stamp.
- Every transaction has a time-stamp associated with it and the ordering is determined by the age of the transaction.

- A transaction created at 0002 clock time would be older than all other transaction, which come after it.
- For example, any transaction 'y' entering the system at 0004 is two seconds younger and priority may be given to the older one.
- In addition, every data item is given the latest read and write-timestamp. This lets the system know, when last read was and write operation made on the data item.

Time stamp ordering protocol

- The timestamp-ordering protocol ensures serializability among transaction in their conflicting read and writes operations.
- This is the responsibility of the protocol system that the conflicting pair of tasks should be executed according to the timestamp values of the transactions.
  - ✓ Time-stamp of Transaction Ti is denoted as TS(Ti).
  - ✓ Read time-stamp of data-item X is denoted by R-timestamp(X).
  - ✓ Write time-stamp of data-item X is denoted by W-timestamp(X).
- Timestamp ordering protocol works as follows:
  - ✓ If a transaction Ti issues read(X) operation:
    - ➢ If TS(Ti) < W-timestamp(X)
      - ❖ Operation rejected.
    - ➢ If TS(Ti) >= W-timestamp(X)
      - ❖ Operation executed.
    - ➢ All data-item Timestamps updated.
  - ✓ If a transaction Ti issues write(X) operation:
    - ➢ If TS(Ti) < R-timestamp(X)
      - ❖ Operation rejected.
    - ➢ If TS(Ti) < W-timestamp(X)
      - ❖ Operation rejected and Ti rolled back.
    - ➢ Otherwise, operation executed.

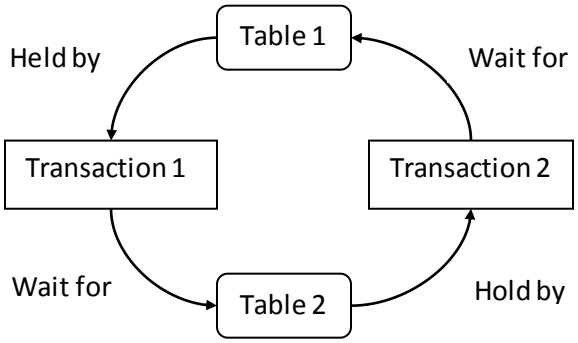
What is deadlock? Explain wait-for-graph. When it occurs? *OR*

Define deadlock. Explain wait-for-graph. Explain different conditions that lead to deadlock.

Deadlock

- A deadlock is a condition when two or more transactions are executing and each transaction is waiting for the other to finish but none of them are ever finished. So all the transactions will wait for infinite time and not a single transaction is completed.

Wait-for-graph



- In the above figure there are two transactions 1 and 2 and two table's as table1 and table 2.
- Transaction 1 hold table 1 and wait for table 2. Transaction 2 hold table 2 and wait for table 1.
- Now the table 1 is wanted by transaction 2 and that is hold by transaction 1 and same way table 2 is wanted by transaction 1 and that is hold by transaction 2. Until any one can't get this table they can't precede further so this is called wait for graph. Because both of these transaction have to wait for some resources.

When dead lock occurs

- A deadlock occurs when two separate processes struggle for resources are held by one another.
- Deadlocks can occur in any concurrent system where processes wait for each other and a cyclic chain can arise with each process waiting for the next one in the chain.
- Deadlock can occur in any system that satisfies the four conditions:
    1. **Mutual Exclusion Condition**: only one process at a time can use a resource or each resource assigned to 1 process or is available.
    2. **Hold and Wait Condition**: processes already holding resources may request new resources.
    3. **No Preemption Condition**: only a process holding a resource can release it voluntarily after that process has completed its task or previously granted resources cannot forcibly taken away from any process.
    4. **Circular Wait Condition**: two or more processes forms circular chain where each process requests a resource that the next process in the chain holds.

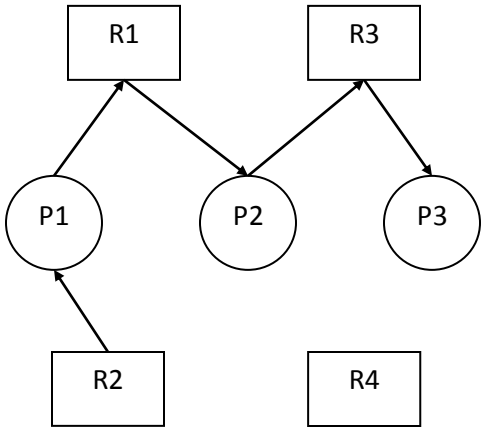
Explain deadlock detection and recovery.

Resource-Allocation Graph

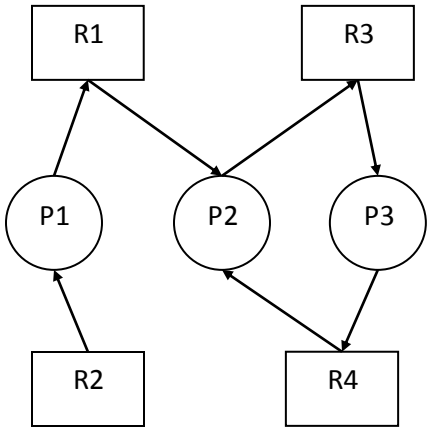
- A set of vertices V and a set of edges E.
    - ✓ V is partitioned into two types:
        1. P = {P1, P2, …, Pn}, the set consisting of all the processes in the system.

2. R = {R1, R2, …, Rm}, the set consisting of all resource types in the system.

- request edge – directed edge Pi → Rj
- assignment edge – directed edge Rj → Pi
- Process - ◯
- Resource – □
- Process Pi requests resource Rj ◯→□
- Process Pi is holding resource Rj ◯←□
- If graph contains no cycles then no deadlock.
- If graph contains cycles then deadlock.



No cycle (circular chain) created
So no deadlock



Cycle (circular chain) created
(P2, R3, P3, R4, P2) So deadlock

- When a deadlock is detected, the system must recover from the deadlock.
- The most common solution is to roll back one or more transactions to break the deadlock.
- Choosing which transaction to abort is known as Victim Selection.
- In the above wait-for graph transactions R3 and R4 are deadlocked.
- In order to remove deadlock one of the transaction out of these two (R3, R4) transactions must be roll backed.
- We should rollback those transactions that will incur the minimum cost.
- When a deadlock is detected, the choice of which transaction to abort can be made using following criteria:
- The transaction which have the fewest locks
- The transaction that has done the least work
- The transaction that is farthest from completion

Explain deadlock prevention methods. OR Explain methods to prevent deadlock.

Deadlock prevention

- A protocols ensure that the system will never enter into a deadlock state.
- Some prevention strategies :
  - ✓ Require that each transaction locks all its data items before it begins execution (predeclaration).
  - ✓ Impose partial ordering of all data items and require that a transaction can lock data items only in the order specified by the partial.
- Following schemes use transaction timestamps for the sake of deadlock prevention alone.

1. Wait-die scheme — non-preemptive
   - If an older transaction is requesting a resource which is held by younger transaction, then older transaction is allowed to wait for it till it is available.
   - If an younger transaction is requesting a resource which is held by older transaction, then younger transaction is killed.
2. Wound-wait scheme — preemptive
   - If an older transaction is requesting a resource which is held by younger transaction, then older transaction forces younger transaction to kill the transaction and release the resource.
   - If an younger transaction is requesting a resource which is held by older transaction, then younger transaction is allowed to wait till older transaction will releases it.
3. Timeout-Based Schemes :
   - A transaction waits for a lock only for a specified amount of time. After that, the wait times out and the transaction is rolled back. So deadlocks never occur.
   - Simple to implement; but difficult to determine good value of the timeout interval.

	Wait/Die	**Wound/Wait**
O needs a resource held by Y	O waits	Y dies
Y needs a resource held by O	Y dies	Y waits

What is data security? (Define data security). What are the objectives while designing secure database?

Data security

- Data security is the protection of the data from unauthorized users.
- Only the authorized users are allowed to access the data.
- Most of the users are allowed to access a part of database i.e., the data that is related to them or related to their department.
- Mostly, the DBA or head of department can access all the data in the database.
- Some users may be permitted only to retrieve data, whereas others are allowed to retrieve as well as to update data.
- The database access is controlled by the DBA.
- He/she creates the accounts of users and gives rights to access the database.
- Users or group of users are given usernames protected by passwords.
- The user enters his/her account number (or user name) and password to access the data from database.
- For example, if you have an account in the "yahoo.com", then you have to give your correct username and password to access your account of e-mail.
- Similarly, when you insert your ATM card into the Automated Teller Machine (ATM), the machine reads your ID number printed on the card and then asks you to enter your pin code (or password). In this way you can access your account.

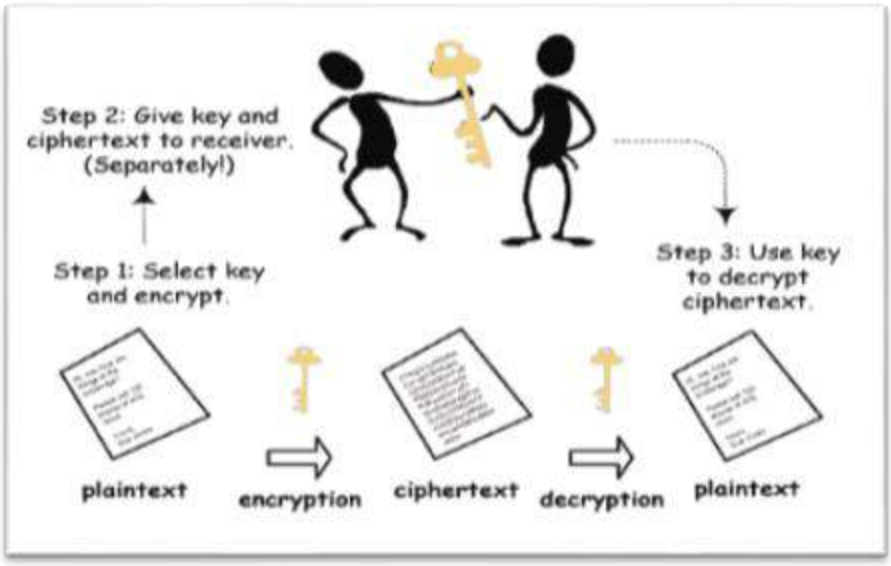
What is the difference between security and integrity?

Data Security	*Data Integrity*
Data security defines a prevention of data corruption through the use of controlled access mechanisms.	Data integrity defines a quality of data, which guarantees the data is complete and has a whole structure.
Data security deals with protection of data	Data integrity deals with the validity of data
Data security is making sure only the people who should have access to the data are the only ones who can access the data.	Data integrity is making sure the data is correct and not corrupt.
Data security refers to the making sure that data is accessed by its intended users, thus ensuring the privacy and protection of data.	Data integrity refers to the structure of the data and how it matches the schema of the database.
Authentication/authorization, encryptions and masking are some of the popular means of data security.	Backing up, designing suitable user interface and error detection/correction in data are some of the means to preserve integrity.

Explain (Describe) data encryption in detail (brief). *OR* Write short note on data encryption *OR*

Data encryption

- Encryption is a technique of encoding data, so that only authorized user can understand (read) it.
- The data encryption technique converts readable data into unreadable data by using some techniques so that unauthorized person cannot read it



Data encryption process

- In above figure sender having data that he/she wants to send his/her data is known as plaintext.
- In first step sender will encrypt data (plain text) using encryption algorithm and some key.
- After encryption the plaintext becomes ciphertext.
- This ciphertext is not able to read by any unauthorized person.
- This ciphertext is send to receiver.
- The sender will send that key separately to receiver.
- Once receiver receives this ciphertext, he/she will decrypt it by using that key send by sender and decryption algorithm.
- After applying decryption algorithm and key, receiver will get original data (plaintext) that is sended by sender.
- This technique is used to protect data when there is a chance of data theft.
- In such situation if encrypted data is theft then it cannot be used (read) directly without knowing the encryption technique and key.

- There are two different method of data encryption
  1. Symmetric key encryption / Private key encryption
     - In symmetric key schemes, the encryption and decryption keys are the same.
     - This same key is used by the sender to encrypt the data, and again by the receiver to decrypt the data.
     - Symmetric encryption is fast in execution.
  2. Asymmetric key encryption / Public key encryption
     - In asymmetric key schemes, the encryption and decryption keys are the different (Public Key and Private Key).
     - Messages are encrypted by sender with one key (Public Key) and can be decrypted by receiver only by the other key (Private Key).
     - Asymmetric Encryption is slow in execution due to the high computational burden.

Explain types of access control methods of data security *OR* Explain discretionary access control and mandatory access control of data security.

- There are two different methods of data access control:-
  1. Discretionary access control
  2. Mandatory access control

Discretionary access control

- In this method, user is given access rights (privileges) to access database object or data items such as table or view.
- This method is based on the concept of access rights and mechanisms for giving rights to user.
- In this method a single user can have different rights on different data items, as well as different user can have different rights on the same data item.
- SQL support discretionary access control through the GRANT and REVOKE commands.

GRANT

- This command gives rights to user for a data items.
  Syntax:-

  GRANT privilege ON object TO user [WITH GRANT OPTION]

REVOKE

- This command takes back rights from user for a data items.
  Syntax:-

  REVOKE privilege ON object FROM user {RESTRICT/CASCADE}
- Privileges are various operations that we will perform on table or view E.g. INSERT, UPDATE, DELETE, SELECT etc.

- Object is table or view or any data item.
- User is the name of user.
- The grant option is used when user wants to pass the rights to other user. Means if a user get a rights with the grant option then he/she can give this rights to another user.
- When user executes a REVOKE command with the cascade option then it will take back given rights from all the users who get those rights from this user.

Mandatory access control

- In this method individual user cannot get rights. But all the users as well as all the objects are classified into different categories. Each user is assigned a clearance level and each object is assigned a security level.
- A user can access object of particular security level only if he has proper clearance level.
- The DBMS determines whether a given user can read or write a given object based on some rules.
- This rule contains security level of object and clearance level of the user.
- This rule makes sure that sensitive data can never be passed to a user without necessary clearance.

Components

- The commonly used mandatory access control technique for multi-level security uses four components as given below
    1. **Subjects:-**Such as users, accounts, programs etc.
    2. **Objects:-**Such as relation (table), tuples (records), attribute (column), view etc.
    3. **Clearance level:-**Such as top secret (TS), secret (S), confidential (C), Unclassified (U). Each subject is classified into one of these four classes.
    4. **Security level: -**Such as top secret (TS), secret (S), confidential (C), Unclassified (U). Each object is classified into one of these four classes.
- In the above system TS>S>C>U, where TS>S means class TS data is more sensitive than class S data.

Rules:-

- A user can access data by following two rules
    1. Security property:-
        - Security property states that a subject at a given security level may not read an object at a higher security level.
    2. Star (*) security property:-
        - Star (*) property states that a subject at a given security level may not write to any object at a lower security level.

Role based access control (RBAC) rules

- Role based access control (RBAC) rules
- It restricts database access based on a person's role within an organization. The roles in RBAC refer to the levels of access that employees have to the network.
- Employees are only allowed to access the information necessary to effectively perform their job duties.

- Access can be based on several factors, such as authority, responsibility, and job competency.
- In addition, access to computer resources can be limited to specific tasks such as the ability to view, create, or modify a file.
- Lower-level employees usually do not have access to sensitive data if they do not need it to fulfil their responsibilities.
- Using RBAC will help in securing your company's sensitive data and important applications.

What is Intrusion Detection System?

- An Intrusion Detection System (IDS) is a system or software application that monitors network traffic or system for suspicious activity or policy violations and issues alerts when such activity is discovered.
- It is a software application that scans a network or a system for harmful activity or policy breaching.
- Any malicious venture or violation is normally reported either to an administrator or collected centrally using a security information and event management (SIEM) system.
- A SIEM system integrates outputs from multiple sources and uses alarm filtering techniques to differentiate malicious activity from false alarms.

Types of Intrusion Detection Systems (IDS)

1. Network Intrusion Detection Systems (NIDS)
2. Host Intrusion Detection Systems (HIDS)
3. Knowledge-based (Signature-based) IDS
4. Behavior-based (Anomaly-based) IDS

Features of an Intrusion Detection System:

- It monitors and analysis the user and system activities.
- It performs auditing of the system files and other configurations and the operating system.
- It assesses the integrity of system and data files
- It conducts analysis of patterns based on known attacks.
- It detects errors in system configuration.
- It detects and cautions if the system is in danger.

Advantages of Intrusion Detection Systems

- The network or computer is constantly monitored for any invasion or attack.
- The system can be modified and changed according to needs of specific client and can help outside as well as inner threats to the system and network.
- It effectively prevents any damage to the network.
- It provides user friendly interface which allows easy security management systems.
- Any alterations to files and directories on the system can be easily detected and reported.

What is SQL injection?

- SQL injection, also known as SQLI, is a common attack vector that uses malicious SQL code for backend database manipulation to access information that was not intended to be displayed.
- This information may include any number of items, including sensitive company data, user lists or private customer details.
- A successful attack may result in the unauthorized viewing of user lists, the deletion of entire tables, gaining administrative rights to a database, all of which are highly detrimental to a business.

What is authorization and authentication? *OR* What is difference between authorization and authentication?

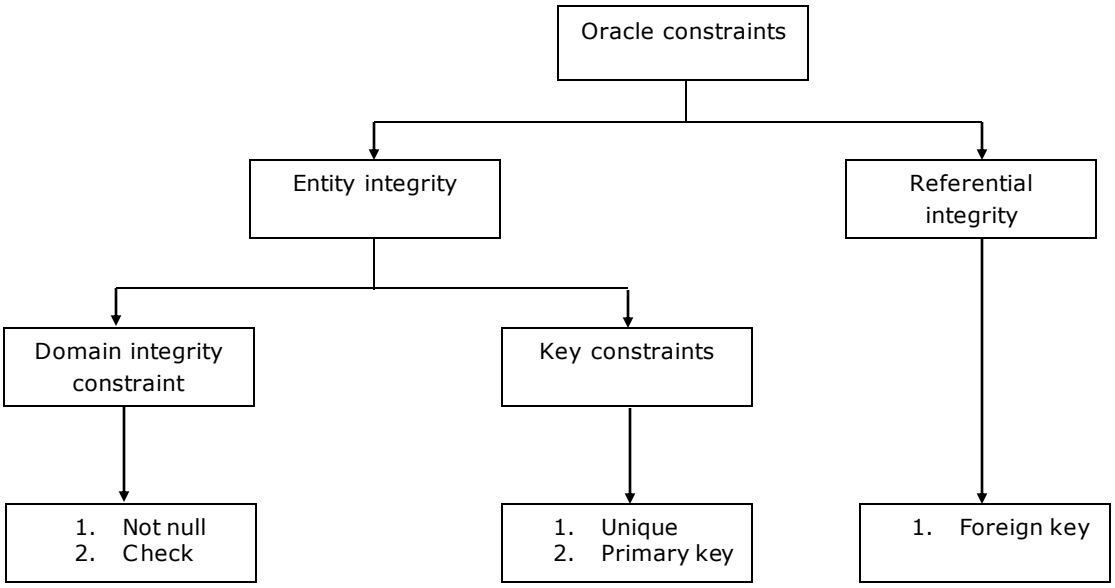
Authorization	*Authentication*
It is protecting the data to ensure privacy and access control of data. Authorization is giving access to authorized users.	Authentication is providing integrity control and security to the data.
Authorization is the process of verifies what you are authorized to do or not to do.	Authentication is the process of verifying who you are.
Accessing a file from hard disk is authorization because the permissions are given to you to access that file allow you access it that is authorization.	Logging on to a PC with a username and password is authentication.

What is audit trail (audit log)?

- An audit trail (audit log) is one record which will be generated against each and every transactions.
- Regarding the transaction, it will keep certain information.
- An audit trail (audit log) records
    1. Who (user or the application program and a transaction number)
    2. When (date and time)
    3. From where (location of the user and/or terminal)
    4. What (identification of the data affected, as well as a before-and-after image of that portion of the database that was affected by the update operation)

What is constraint? Explain types of constraints. *OR* What are integrity constraints? Explain various types of integrity constraints with suitable example.

- A constraint is a rule that restricts the values that may be present in the database.
- Constraints can be mainly classified in to two categories.
    1. Entity integrity constraints
    2. Referential integrity constraints



Not Null constraint

- A null value indicates 'not applicable', 'missing', or 'not known'.
- A null value is distinct from zero or blank space.
- A column, defined as a not null, cannot have a null value.
- Such a column becomes a mandatory (compulsory) column and cannot be left empty for any record.
- **Syntax** : ColumnName datatype (size) NOT NULL
- **Example** :

```
create table Account  (ano int,
                       Balance decimal(8,2) NOT NULL,
                       Branch varchar(10));
```

- Now, if we insert NULL value to Balance column then it will generate an error.

Check constraint

- The check constraint is used to implement business rule. So, it is also called business rule constraint.
- Example of business rule: A balance in any account should not be negative.
- Business rule define a domain for a particular column.

- The check constraint is bound to a particular column.
- Once check constraint is implemented, any insert or update operation on that table must follow this constraint.
- If any operation violates condition, it will be rejected.
- **Syntax** *:* ColumnName datatype (size) check(condition)
- **Example** :

```
create table Account  (ano int,
                       Balance decimal(8,2) CHECK (balance >= 0),
                       Branch varchar(10));
```

- Any business rule validations can be applied using this constraint.
- A condition must be some valid logical expression.
- A check constraint takes longer time to execute.
- On violation of this constraint, oracle display error message like – "check constraint violated".

Unique Constraint

- Sometime there may be requirement that column cannot contain duplicate values.
- A column, defined as a unique, cannot have duplicate values across all records.
- **Syntax** *:* ColumnName datatype (size) UNIQUE
- **Example** :

```
create table Account  (ano int UNIQUE,
                       Balance decimal(8,2),
                       Branch varchar(10));
```

- Though, a unique constraint does not allow duplicate values, NULL values can be duplicated in a column defined as a UNIQUE column.
- A table can have more than one column defined as a unique column.
- If multiple columns need to be defined as composite unique column, then only table level definition is applicable.
- Maximum 16 columns can be combined as a composite unique key in a table.

Primary key Constraint

- A primary key is a set of one or more columns used to identify each record uniquely in a column.
- A single column primary key is called a simple key, while a multi-column primary key is called composite primary key.
- Oracle provides a primary key constraint to define primary key for a table.
- A column, defined as a primary key, cannot have duplicate values across all records and cannot have a null value.
- **Syntax** *:* ColumnName datatype (size) primary key

- **Example** :

```
create table Account  (ano int NOT NULL PRIMARY KEY,
                       Balance decimal(8,2),
                       Branch varchar(10));
```

- A primary key constraint is combination of UNIQUE constraint and NOT NULL constraint.
- A table cannot have more than one primary key.
- If multiple columns need to be defined as primary key column, then only table level definition is applicable.
- Maximum 16 columns can be combined as a composite primary key in a table.

Foreign Key Constraint

- A foreign key constraint is also called referential integrity constraint, is specified between two tables.
- This constraint is used to ensure consistency among records of the two tables.
- The table, in which a foreign key is defined, is called a foreign table, detail table or child table.
- The table, of which primary key or unique key is defined, is called a primary table, master table or parent table.
- Restriction on child table :
  - ✓ Child table contains a foreign key. And, it is related to master table.
  - ✓ Insert or update operation involving value of foreign key is not allowed, if corresponding value does not exist in a master table.
- Restriction on master table :
  - ✓ Master table contains a primary key, which is referred by foreign key in child table.
  - ✓ Delete or update operation on records in master table are not allowed, if corresponding records are present in child table.
- **Syntax** *:* ColumnName datatype (size) REFERENCES TableName (ColumnName)
- **Example** :

```
create table Account  (ano int,
                       Balance decimal(8,2),
                       BranchID varchar(10) REFERENCES branchs(branchID));
```

- Master table must be exist before creating child table.

Explain DDL, DML, DCL and DQL. *OR* Describe component of SQL.

DDL (Data Definition Language)

- It is a set of SQL commands used to create, modify and delete database objects such as tables, views, indices, etc.
- It is normally used by DBA and database designers.

- It provides commands like:
  - ✓ CREATE: to create objects in a database.
  - ✓ ALTER: to alter the schema, or logical structure of the database.
  - ✓ DROP: to delete objects from the database.
  - ✓ TRUNCATE: to remove all records from the table.

CREATE: Create is used to create the database or its objects like table, view, index etc.

- **Create Table**

  The CREATE TABLE statement is used to create a new table in a database.

  Syntax:

```
CREATE TABLE table_name
(
    Column1 Datatype(Size) [ NULL | NOT NULL ],
    Column2 Datatype(Size) [ NULL | NOT NULL ],
  ...
);
```

  Example:

```
CREATE TABLE Students
(
    Roll_No int(3) NOT NULL,
    Name varchar(20),
    Subject varchar(20)
);
```

Explanation:

- The column should either be defined as NULL or NOT NULL. By default, a column can hold NULL values.
- The NOT NULL constraint enforces a column to NOT accept NULL values. This enforces a field to always contain a value, which means that you cannot insert a new record, or update a record without adding a value to this field.

ALTER: ALTER TABLE statement is used to add, modify, or drop columns in a table.

- **Add Column**

  The ALTER TABLE statement in SQL to add new columns in a table.

  Syntax:

```
ALTER TABLE table_name
ADD Column1 Datatype(Size), Column2 Datatype(Size), … ;
```

  Example:

```
ALTER TABLE Students
ADD Marks int;
```

- **Drop Column**

  The ALTER TABLE statement in SQL to drop a column in a table.

Syntax:

```
ALTER TABLE table_name
DROP COLUMN column_name;
```

Example:

```
ALTER TABLE Students
DROP COLUMN Subject;
```

- **Modify Column**

  The ALTER TABLE statement in SQL to change the data type/size of a column in a table.

  Syntax:

  ```
  ALTER TABLE table_name
  ALTER COLUMN column_name datatype(size);
  ```

  Example:

  ```
  ALTER TABLE Students
  ALTER COLUMN Roll_No float;
  ```

DROP: Drop is used to drop the database or its objects like table, view, index etc.

- **Drop Table**

  The DROP TABLE statement is used to drop an existing table in a database.

  Syntax:

  ```
  DROP TABLE table_name;
  ```

  Example:

  ```
  DROP TABLE Students;
  ```

TRUNCATE: Truncate is used to remove all records from the table.

Syntax:

```
TRUNCATE TABLE table_name;
```

Example:

```
TRUNCATE TABLE Students;
```

DML (Data Manipulation Language)

- It is a set of SQL commands used to insert, modify and delete data in a database.
- It is normally used by general users who are accessing database via pre-developed applications.
- It provides commands like:
  - ✓ INSERT: to insert data into a table.
  - ✓ UPDATE: to modify existing data in a table.
  - ✓ DELETE: to delete records from a table.

INSERT: The INSERT STATEMENT is used to insert data into a table.

Syntax:

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
                    OR
INSERT INTO table_name
VALUES (value1, value2, value3, ...);
```

Example:

```
INSERT INTO Students (Roll_No,Name,Subject)
VALUES (1,'anil','Maths');
```

UPDATE: The UPDATE STATEMENT is used to modify existing data in a table.

Syntax:

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

Example:

```
UPDATE Students
SET Name= 'Mahesh'
WHERE Roll_No=1;
```

DELETE: The DELETE STATEMENT is used to delete records from a table.

Syntax:

```
DELETE FROM table_name
WHERE condition;
```

Example:

```
DELETE FROM Students
WHERE Subject='Maths';
```

DQL (Data Query Language)

- It is a component of SQL that allows data retrieval from the database.
- It provides command like SELECT. This command is a heart of SQL, and allows data retrieval in different ways.

SELECT: The SELECT statement is used to select data from a database. The data returned is stored in a result table, called the result-set.

Syntax:

```
SELECT column1, column2, ...
FROM table_name
WHERE condition;
            OR
SELECT *
FROM table_name
WHERE condition;
```

Example:

```
SELECT *
FROM Students
WHERE Roll_No>=1;
```

DCL (Data Control Language)

- It is set of SQL commands used to control access to data and database. Occasionally DCL commands are grouped with DML commands.
- It provides commands like:
  - ✓ GRANT: to give access privileges to users on the database.
  - ✓ REVOKE: to withdraw access privileges given to users on the database.

TCL (Transaction Control Language)

- TCL is abbreviation of Transactional Control Language. It is used to manage different transactions occurring within a database.
  - ✓ COMMIT – Saves work done in transactions
  - ✓ ROLLBACK – Restores database to original state since the last COMMIT command in transactions
  - ✓ SAVE TRANSACTION – Sets a save point within a transaction.

Explain Transaction Control Commands. *OR* Explain commit, rollback and savepoint command.

- Transaction Control Command (TCL) is a set of SQL commands that are used to control transactional processing in a database.
- A transaction is logical unit of work that consists of one or more SQL statements, usually a group of Data Manipulation Language (DML) statements.
- Transaction Control Commands (TCL) commands include

1. **COMMIT:**
   - COMMIT command is used to permanently save any transaction into the database.
   - When we use any DML command like INSERT, UPDATE or DELETE, the changes made by these commands are not permanent, until the current session is closed, the changes made by these commands can be rolled back.
   - To avoid that, COMMIT is used to mark the changes as permanent.

     Example: Consider following table

Student		
Rollno	**Name**	**SPI**
1	Raju	8
2	Hari	9
3	Mahesh	7

```
BEGIN TRANSACTION t1
DELETE FROM STUDENT WHERE SPI <8;
COMMIT TRANSACTION t1;
```

Output:

Student		
Rollno	**Name**	**SPI**
1	Raju	8
2	Hari	9

Note:

- A transaction is a set of operations performed so that all operations are guaranteed to succeed or fail as one unit.
- If you place the BEGIN TRANSACTION before your SQL statement, the transaction will automatically turn into the explicit transaction and it will lock the table until the transaction is committed or rolled back.

2. ROLLBACK:

- The ROLLBACK command is the transactional control command used to undo transactions that have not already been saved to the database.
- Rollbacks a transaction to the beginning of the transaction.
- It is also used with SAVEPOINT command to jump to a savepoint in an ongoing transaction.
- You can use ROLLBACK TRANSACTION to erase all data modifications made from the start of the transaction or to a savepoint.

Example: Consider following table

Student		
Rollno	**Name**	**SPI**
1	Raju	8
2	Hari	9
3	Mahesh	7

```
BEGIN TRANSACTION t1
DELETE FROM STUDENT WHERE SPI <8;
ROLLBACK TRANSACTION t1;
```

Output:

Student		
Rollno	**Name**	**SPI**
1	Raju	8
2	Hari	9
3	Mahesh	7

3. SAVEPOINT:

- A SAVEPOINT is a point in a transaction when you can roll the transaction back to a certain point without rolling back the entire transaction.
- The ROLLBACK command is used to undo a group of transactions.

Example: Consider following table

Student		
Rollno	**Name**	**SPI**
1	Raju	8
2	Hari	9
3	Mahesh	7

```
BEGIN TRANSACTION t1
SAVE TRANSACTION s1
INSERT INTO Student Values (4,'Anil',6);

SAVE TRANSACTION s2
INSERT INTO Student Values (5, Gita',9);

ROLLBACK TRANSACTION s2;
```

Output:

Student		
Rollno	**Name**	**SPI**
1	Raju	8
2	Hari	9
3	Mahesh	7
4	Anil	6

Explain Security Privileged Commands. *OR* Explain Grant and Revoke command. *OR* Explain Data Control Language (DCL) Commands.

- DCL commands are used to enforce (implement) database security in a multiple user database environment.
- These commands are used to give or take back permission on any object to/from any user.
- Two types of DCL commands are GRANT and REVOKE.
- Only Database Administrator or owner of the database object can provide/remove privileges on a database object.

1. **GRANT**: GRANT is a command used to provide access or privileges or rights on the database objects to the users.

   Syntax:

   ```
   GRANT privilege_name
   ON object_name
   TO {user_name |PUBLIC}
   [WITH GRANT OPTION];
   ```

   Explanation:

   - privilege_name is the access right or privilege granted to the user. Some of the access rights are ALL, EXECUTE, and SELECT.
   - object_name is the name of an database object like TABLE, VIEW, STORED PROC and SEQUENCE.
   - user_name is the name of the user to whom an access right is being granted.
   - PUBLIC is used to grant access rights to all users.
   - WITH GRANT OPTION - allows a user to grant (give) access rights to other users.

2. **REVOKE**: The REVOKE is a command used to take back access or privileges or rights on the database objects from the users.

   Syntax:

   ```
   REVOKE privilege_name
   ON object_name
   FROM {user_name |PUBLIC }
   ```

   Explanation:

   - privilege_name is the access right or privilege want to take back from the user. Some of the access rights are ALL, EXECUTE, and SELECT.

- object_name is the name of an database object like TABLE, VIEW, STORED PROC and SEQUENCE.
- user_name is the name of the user from whom an access right is being taken back.
- PUBLIC is used to take back access rights to all users.

Explain "on delete cascade" with example.

- A foreign key with cascade delete means that if a record in the Parent (Master) table is deleted, then the corresponding records in the Child (Foreign) table with automatically be deleted. This is called a cascade delete in Oracle.
- A foreign key with a cascade delete can be defined in either a CREATE TABLE statement or an ALTER TABLE statement.
- Syntax (Create table)

```
CREATE TABLE table_name
(
 column1 datatype null/not null,
 column2 datatype null/not null,
 ...

CONSTRAINT fk_column
    FOREIGN KEY (column1, column2, ... column_n)
    REFERENCES parent_table (column1, column2, ... column_n)
    ON DELETE CASCADE
);
```

- Syntax (Alter table)

```
ALTER TABLE table_name
ADD CONSTRAINT constraint_name
FOREIGN KEY (column1, column2, ... column_n)
    REFERENCES parent_table (column1, column2, ... column_n)
    ON DELETE CASCADE;
```

Describe the following SQL functions.

SQL Function	Description	SQL Query Example
Math function		
Abs(n)	Returns the absolute value of n.	Select Abs(-15); O/P : 15
Sign(n)	Returns the sign of x as -1,0,1	Select Sign(-15); O/P : -1

Ceiling(n)	Returns the smallest integer value that is smaller than or equal to a number.	Select ceil(24.8); O/P : 25
Floor(n)	Returns the largest integer value that is greater than or equal to a number.	Select Floor(24.8); O/P : 24
Power (m,n)	Returns m raised to n^{th} power.	Select power(3,2); O/P : 9
Round (n,m)	Returns n rounded to m places the right of decimal point.	Select round(15.91,1); O/P : 15.9
Square(n)	Returns square of n.	Select Square(4); O/P : 16
Sqrt(n)	Returns square root of n.	Select sqrt(25); O/P : 5
Exp(n)	Returns e raised to the n^{th} power, e=2.17828183.	Select exp(1); O/P : 1
X%Y	Returns reminder of X modulus Y.	Select 5%3; O/P : 2
Pi()	Returns the value of pi.	Select Pi(); O/P : 3.14159265358979
Sin(n)	Returns the sine value of n.	Select Sin(0); O/P : 0
Cos(n)	Returns the cosine value of n.	Select Cos(0); O/P : 1
Tan(n)	Returns the tangent value of n.	Select Tan(0); O/P : 0
Rand(n)	Returns a random decimal number between 0 and 1.	Select Rand(); O/P : 0.845610816728278
Log(n)	Returns the log value of n having base e.	Select Log(1); O/P : 0
Log10(n)	Returns the log value of n having base 10.	Select Log10(1000); O/P : 3
String function		
ASCII(x)	Returns ASCII value of character.	Select ASCII('A'); O/P : 65
Char(x)	Returns a character of int ASCII code.	Select CHAR(65); O/P : A
Concat()	Concatenates two strings.	Select CONCAT('great','DIET'); O/P : greatDIET
Len ()	Returns the number of character in x.	Select LEN('DIET'); O/P : 4
Lower()	Converts the string to lower case.	Select LOWER('DIET'); O/P : diet
Upper()	Converts the string to upper case.	Select UPPER('diet'); O/P : DIET
Ltrim()	Trim blanks from the left of x.	Select LTRIM(' diet '); O/P : diet

Rtrim()	Trim blanks from the right of x.	Select RTRIM(` diet '); O/P : diet
Replace()	Looks for the string and replace the string every time it occurs.	Select Replace('this is college','is','may be'); O/P :thmay be may be college
Substring()	Returns the part of string	Select Substring('this is college',6,7); O/P : is my c
Left()	Returns the specified number of characters from the left.	Select Left('this is college',7); O/P : this is
Right()	Returns the specified number of characters from the right.	Select Right('this is college',7); O/P : college
Reverse()	Returns the reversed string.	Select Reverse('DIET'); O/P : TEID
Space()	Returns n number of spaces	Select Space(10); O/P :
Replicate()	Returns repeated string for n number of times.	Select Replicate('DIET',2); O/P :DIETDIET
Date function		
Getdate()	Returns current date and time.	Select Getdate(); O/P : 2018-09-08 10:42:02.113
Day()	Returns day of a given date.	Select Day('23/JAN/2018'); O/P : 23
Month()	Returns month of a given date.	Select Month('23/JAN/2018'); O/P : 1
Year()	Returns year of a given date.	Select Year('23/JAN/2018'); O/P : 2018
Isdate()	Returns 1 if the expression is a valid date, otherwise 0.	Select Isdate('31/FEB/2018'); O/P : 0
Datename()	Returns the specified part of a given date as varchar value.	Select Datename(month,'1-23-2018'); O/P : January
Datepart()	Returns the specified part of a given date as int value.	Select Datepart(month,'1-23-2018'); O/P : 1
Dateadd()	Returns datetime after adding n numbers of datepart to a given date.	Select Dateadd(day,5,'23/JAN/2018'); O/P : 2018-01-28 00:00:00.000
Datediff()	Returns the difference between two date values, based on the interval specified.	Select Datediff(day,5, '23/JAN/2018','23/FEB/2018'); O/P : 31
Eomonth()	Returns the last day of month.	Select Eomonth('23/FEB/2018'); O/P : 2018-01-31

Discuss aggregate functions with example(s).

- Aggregate functions perform a calculation on a set of values and return a single value.
- Aggregate functions ignore NULL values except COUNT(*).
- It is used with the GROUP BY clause of the SELECT statement.

Example: Consider following table

Student		
Rollno	**Name**	**SPI**
1	Raju	8
2	Hari	9
3	Mahesh	7
4	NULL	9
5	Anil	5

1. Avg() : It returns the average of the data values.
   Select Avg(SPI) FROM Student;
   Output: 7
2. Sum() : It returns the addition of the data values.
   Select Sum(SPI) FROM Student;
   Output: 38
3. Max() : It returns maximum value for a column.
   Select Max(SPI) FROM Student;
   Output: 9
4. Min() : It returns Minimum value for a column.
   Select Min(SPI) FROM Student;
   Output: 5
5. Count() : It returns total number of values in a given column.
   Select Count(Name) FROM Student;
   Output: 4
6. Count(*) : It returns the number of rows in a table.
   Select Count(*) FROM Student;
   Output: 5

What is join? List and explain various types of joins.

- A SQL Join statement is used to combine data or rows from two or more tables based on a common field between them.
- Different types of Joins are:
  - ✓ Inner Join
  - ✓ Outer Join
    1. Left Outer Join
    2. Right Outer Join
    3. Full Outer Join
  - ✓ Cross join
  - ✓ Self Join

INNER JOIN

- It returns records that have matching values in both tables.

Syntax:

```
SELECT columns
FROM table1 INNER JOIN table2
ON table1.column = table2.column;
```

Example:

Consider the following tables:

Student		
RNO	**Name**	**Branch**
101	Raju	CE
102	Amit	CE
103	Sanjay	ME
104	Neha	EC
105	Meera	EE
106	Mahesh	ME

Result	
RNO	**SPI**
101	8.8
102	9.2
104	8.2
105	7
107	8.9

```
SELECT Student.RNO, Student.Name, Student.Branch, Result.SPI
FROM Student
INNER JOIN Result
ON Student.RNO = Result.RNO;
```

Output:

Inner Join			
RNO	**Name**	**Branch**	**SPI**
101	Raju	CE	8.8
102	Amit	CE	9.2
104	Neha	EC	8.2
105	Meera	EE	7

LEFT OUTER JOIN

- The LEFT JOIN keyword returns all records from the left table (table1), and the matched records from the right table (table2).
- The result is NULL from the right side, if there is no match.

Syntax:

```
SELECT columns
FROM table1 LEFT OUTER JOIN table2
ON table1.column = table2.column;
```

Example:

Consider the Student and result tables:

```
SELECT Student.RNO, Student.Name, Student.Branch, Result.SPI
FROM Student LEFT OUTER JOIN Result
ON Student.RNO = Result.RNO;
```

Output:

Left Join			
RNO	**Name**	**Branch**	**SPI**
101	Raju	CE	8.8
102	Amit	CE	9.2
103	Sanjay	ME	NULL
104	Neha	EC	8.2
105	Meera	EE	7
106	Mahesh	ME	NULL

RIGHT OUTER JOIN

- The RIGHT JOIN keyword returns all records from the right table (table2), and the matched records from the left table (table1).
- The result is NULL from the left side, if there is no match.

Syntax:

```
SELECT columns
FROM table1 RIGHT OUTER JOIN table2
ON table1.column = table2.column;
```

Example:

Consider the Student and result tables:

```
SELECT Student.RNO, Student.Name, Student.Branch, Result.SPI
FROM Student RIGHT OUTER JOIN Result
ON Student.RNO = Result.RNO;
```

Output:

Right Join			
RNO	**Name**	**Branch**	**SPI**
101	Raju	CE	8.8
102	Amit	CE	9.2
104	Neha	EC	8.2
105	Meera	EE	7
NULL	NULL	NULL	8.9

FULL OUTER JOIN

- The FULL OUTER JOIN keyword return all records when there is a match in either left (table1) or right (table2) table records.

Syntax:

```
SELECT columns
FROM table1 FULL OUTER JOIN table2
ON table1.column = table2.column;
```

Example:

Consider the Student and result tables:

```
SELECT Student.RNO, Student.Name, Student.Branch, Result.SPI
FROM Student FULL OUTER JOIN Result
ON Student.RNO = Result.RNO;
```

Output:

Full Join			
RNO	**Name**	**Branch**	**SPI**
101	Raju	CE	8.8
102	Amit	CE	9.2
103	Sanjay	ME	NULL
104	Neha	EC	8.2
105	Meera	EE	7
106	Mahesh	ME	NULL
NULL	NULL	NULL	8.9

CROSS JOIN

- When each row of first table is combined with each row from the second table, known as Cartesian join or cross join.
- SQL CROSS JOIN returns the number of rows in first table multiplied by the number of rows in second table.

Syntax:

```
SELECT columns
FROM table1 CROSS JOIN table2;
```

Example:

Consider the following tables:

Color	
Code	**Name**
1	Red
2	Blue

Size
Amount
Small
Large

```
SELECT Color.Code, Color.Name, Size.Amount
FROM Color CROSS JOIN Size;
```

Output:

Cross Join		
Code	**Name**	**Amount**
1	Red	Small
2	Blue	Small
1	Red	Large
2	Blue	Large

SELF JOIN

- A self join is a regular join, but the table is joined with itself.
- Here, we need to use aliases for the same table to set a self join between single table.

Syntax:

```
SELECT a.column, b.column
FROM tablename a CROSS JOIN tablename b
WHERE a.column=b.column;
```

Example:

Consider the following table:

Employee		
EmpNo	**Name**	**MngrNo**
E01	Tarun	E02
E02	Rohan	E05
E03	Priya	E04
E04	Milan	NULL
E05	Jay	NULL
E06	Anjana	E03

```
SELECT e.Name as Employee, m.Name as Employee
FROM Employee e INNER JOIN Employee m
ON e.MngrNo=m.EmpNo;
```

Output:

Employee	
Employee	**Manager**
Tarun	Rohan
Rohan	Jay
Priya	Milan
Anjana	Priya

Define view. What are the types of view? Write syntax to create view of each type. Give an example of view.

- Views are virtual tables that are compiled at runtime.

- The data associated with views are not physically stored in the view, but it is stored in the base tables of the view.
- A view can be made over one or more database tables.
- Generally, we put those columns in view that we need to retrieve/query again and again.
- Once you have created the view, you can query view like as table.

TYPES OF VIEW

1. Simple View
2. Complex View

Syntax:

```
CREATE VIEW view_name
AS
SELECT column1, column2...
FROM table_name
[WHERE condition];
```

Simple View

- When we create a view on a single table, it is called simple view.
- In a simple view we can delete, update and Insert data and that changes are applied on base table.
- Insert operation are perform on simple view only if we have primary key and all not null fields in the view.

Example:

Consider following table:

Employee			
Eid	**Ename**	**Salary**	**Department**
101	Raju	5000	Admin
102	Amit	8000	HR
103	Sanjay	3000	IT
104	Neha	7000	Sales

```
--Create View
CREATE VIEW EmpSelect
AS
SELECT Eid, Ename, Department
FROM Employee;

--Display View
Select * from EmpSelect;
```

Output		
Eid	**Ename**	**Department**
101	Raju	Admin
102	Amit	HR
103	Sanjay	IT
104	Neha	Sales

Complex View

- When we create a view on more than one table, it is called complex view.
- We can only update data in complex view.
- You can't insert data in complex view.
- In particular, complex views can contain: join conditions, a group by clause, an order by clause etc.

Example:

Consider following table:

Employee			
Eid	**Ename**	**Salary**	**Department**
101	Raju	5000	Admin
102	Amit	8000	HR
103	Sanjay	3000	IT
104	Neha	7000	Sales

ContactDetails		
Eid	**City**	**Mobile**
101	Rajkot	1234567890
102	Ahmedabad	2345678901
103	Baroda	3456789120
104	Rajkot	4567891230

```
--Create View
    CREATE VIEW Empview
    AS
    SELECT Employee.Eid, Employee.Ename, ConcactDetails.City
    FROM Employee Inner Join ConcactDetails
    On Employee.Eid= ConcactDetails.Eid;

--Display View
    Select * from Empview;
```

Output		
Eid	**Ename**	**City**
101	Raju	Rajkot
102	Amit	Ahmedabad
103	Sanjay	Baroda
104	Neha	Rajkot

What is materialized view? Explain Query optimization with materialized view.

- View: View is just a named query. It doesn't store anything. When there is a query on view, it runs the query of the view definition. Actual data comes from table.
  A view uses a query to pull data from the underlying tables.
- Materialized views: Stores data physically and get updated periodically. While querying MV, it gives data from MV.
  A materialized view is a table on disk that contains the result set of a query.
- Materialized views are primarily used to increase application performance when it isn't feasible or desirable to use a standard view with indexes applied to it.
  Materialized views can be updated on a regular basis either through triggers or by using the ON COMMIT REFRESH option.

Explain the advantages of PL/SQL.

Advantages of PL/SQL

- **Block structure**: PL/SQL consist of block of code, which can be nested within each other. Each block forms a unit of a task or a logical module. PL/SQL blocks can be stored in the database and reused.
- **Procedural language capability**: PL/SQL consist of procedural constructs such as conditional statements (if, if else, nested if, else if ladder) and loops (for, while, do while).
- **Better performance**: PL/SQL engine processes multiple SQL statements simultaneously as a single block, thereby reducing network traffic.
- **Error handling**: PL/SQL handles errors or exceptions effectively during the execution of PL/SQL program. Once an exception is caught, specific action can be taken depending upon the type of the exception or it can be displayed to the user with message.

Write short note on stored procedure in PL/SQL. *OR* Explain stored procedures in PL/SQL.

- A stored procedure (proc) is a group of PL/SQL statements that performs specific task.
- A procedure has two parts, header and body.
- The header consists of the name of the procedure and the parameters passed to the procedure.
- The body consists of declaration section, execution section.
- A procedure may or may not return any value. A procedure may return more than one value.

General Syntax to create or Alter a procedure

```
CREATE [OR ALTER] PROCEDURE proc_name [list of parameters]
        Declaration section
AS

        Execution section
```

Explanation

Create:-It will create a procedure.

Alter:- It will re-create a procedure if it already exists.

We can pass parameters to the procedures in three ways.

1. IN-parameters: - These types of parameters are used to send values to stored procedures.
2. OUT-parameters: - These types of parameters are used to get values from stored procedures. This is similar to a return type in functions but procedure can return values for more than one parameters.
3. IN OUT-parameters: - This type of parameter allows us to pass values into a procedure and get output values from the procedure.

AS indicates the beginning of the body of the procedure.

The syntax within the brackets [ ] indicates that they are optional.

By using CREATE OR ALTER together the procedure is created if it does not exist and if it exists then it is replaced with the current code (The only disadvantage of CREATE OR ALTER is that it does not work in SQL Server versions prior to SQL Server 2016).

How to execute a Stored Procedure?

- There are two ways to execute a procedure.
  1) From the SQL prompt.
     Syntax: EXECUTE [or EXEC] procedure_name (parameter);
  2) Within another procedure – simply use the procedure name.
     Syntax: procedure_name (parameter);

Example 1 (Using IN)

```
CREATE PROCEDURE get_studentname_by_id
        @id int
AS
BEGIN
        SELECT studentname
        FROM stu_tbl
        WHERE studentID = @id;
END;
```

Execute:- EXECUTE get_studentname_by_id 10; **OR**
get_studentname_by_id 10;

Explanation:- Above procedure gives the name of student whose id is 10.

Example 2 (Using OUT)

```
CREATE PROCEDURE multiplication
                @mult1 int,
                @mult2 int,
                @result int OUTPUT
AS
BEGIN
                SELECT @result = @mult1 * @mult2
END
```

Execute:-
```
DECLARE @result int
EXEC multiplication 5,6,@result output
PRINT @result
```

Explanation:- Above procedure gives the multiplication of 5 and 6 that is equal to 30.

Advantages of procedure

- **Security:-** We can improve security by giving rights to selected persons only.
- **Faster Execution:-** It is precompiled so compilation of procedure is not required every time you call it.
- **Sharing of code:-** Once procedure is created and stored, it can be used by more than one user.
- **Productivity:-** Code written in procedure is shared by all programmers. This eliminates redundant coding by multiple programmers so overall improvement in productivity.

Explain database trigger with example. *OR* Write short note on database triggers in PL/SQL. *OR* What is trigger? Explain its type with their syntax.

- A trigger is a PL/SQL block structure which is triggered (executed) automatically when DML statements like Insert, Delete, and Update is executed on a table.
- There are three types of triggers.
    - **Data Definition Language (DDL) triggers**
        - In SQL Server we can create triggers on DDL statements (like CREATE, ALTER and DROP) and certain system-defined Stored Procedures that does DDL-like operations.
    - **Data Manipulation Language (DML) triggers**
        - In SQL Server we can create triggers on DML statements (like INSERT, UPDATE and DELETE) and Stored Procedures that do DML-like operations. DML Triggers are of two types.

        1. After trigger (using FOR/AFTER CLAUSE)
            - After trigger (using FOR/AFTER CLAUSE): After triggers are executed after completing the execution of DML statements.
            - Example: If you insert a record/row into a table then the trigger related/associated with the insert event on this table will executed only after inserting the record into that table.
            - If the record/row insertion fails, SQL Server will not execute the after trigger.
        - Instead of trigger (using INSTEAD OF CLAUSE)
            - Instead of Trigger (using INSTEAD OF CLAUSE) : Instead of trigger are executed before starts the execution of DML statements.
            - An instead of trigger allows us to skip an INSERT, DELETE, or UPDATE statement to a table and execute other statements defined in the trigger instead. The actual insert, delete, or update operation does not occur at all.
            - Example: If you insert a record/row into a table then the trigger related/associated with the insert event on this table will be executed before inserting the record into that table.

- If the record/row insertion fails, SQL Server will execute the Instead of Trigger.
- **Logon triggers**
- This type of trigger is executed against a LOGON event before a user session is established to the SQL Server.

- We cannot pass parameters into triggers like stored procedure.
- Triggers are normally slow.
- When triggers can be used,
  - Based on change in one table, we want to update other table.
  - Automatically update derived columns whose values change based on other columns.
  - Logging.
  - Enforce business rules.

Syntax of Trigger

```
CREATE [OR ALTER] TRIGGER trigger_name
  ON table_name
          { FOR | AFTER | INSTEAD OF }
           { [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }

  AS
  BEGIN
          Executable statements
  END;
```

CREATE [OR ALTER] TRIGGER trigger_name:- This clause creates a trigger with the given name or overwrites an existing trigger.
[ON table_name]:- This clause identifies the name of the table to which the trigger is related.
[FOR | AFTER | INSTEAD OF]:- This clause indicates at what time the trigger should be fired. FOR and AFTER are similar.
[INSERT / UPDATE / DELETE]:- This clause determines on which kind of statement the trigger should be fired. Either on insert or update or delete or combination of any or all. More than one statement can be used together separated by Comma. The trigger gets fired at all the specified triggering event.

Example 1

Trigger to display a message when we perform insert operation on student table.

```
CREATE TRIGGER student_msg
on Student
AFTER INSERT
```

```
AS
BEGIN
        print 'Record inserted successfully'
END
```

OUTPUT:- Trigger is created.

- Now when you perform insert operation on student table.

  Insert into student values (1, 'Raj', 'CE');

  It displays following message after executing insert statement.

Output:- (1 row(s) affected)
Record inserted successfully

Example 2

Trigger to insert history into Audit table when we perform insert operation on student table.

```
CREATE TRIGGER tgr_student_forinsert
ON Student
FOR INSERT
AS
BEGIN
        DECLARE @id int
        SELECT @rno= rno from INSERTED
        INSERT INTO Audit VALUES
        ('New student with rno=' + cast(@rno as varchar(10)) +
        'is added in student table ')
END
```

OUTPUT:- Trigger is created.

- **Explanation:-** Now when you perform insert operation on student table.

  Insert into student values (1, 'Raj', 'CE');

  It insert one record in Audit table as "'New student with rno = 1 is added in student table"

What is cursor? Explain with example. *OR* Write short note on cursors and its types.

- Cursors are database objects used to traverse the results of a select SQL query.
- It is a temporary work area created in the system memory when a select SQL statement is executed.
- This temporary work area is used to store the data retrieved from the database, and manipulate this data.
- It points to a certain location within a record set and allow the operator to move forward (and sometimes backward, depending upon the cursor type).

- We can process only one record at a time.
- The set of rows the cursor holds which is called the *active* set (active data set).
- Cursors are often criticized for their high overhead.
- There are two types of cursors in PL/SQL:
    1. **Implicit cursors:**
        - These are created by default by ORACLE itself when DML statements like, insert, update, and delete statements are executed.
        - They are also created when a SELECT statement that returns just one row is executed.
        - We cannot use implicit cursors for user defined work.
    2. **Explicit cursors:**
        - Explicit cursors are user defined cursors written by the developer.
        - They can be created when a SELECT statement that returns more than one row is executed.
        - Even though the cursor stores multiple records, only one record can be processed at a time, which is called as current row.
        - When you fetch a row, the current row position moves to next row.

Steps to manage explicit cursor:-

1) Declare a cursor:-

A cursor is defined in the declaration section of PL/SQL block.

Syntax:-

```
DECLARE cursorname CURSOR FOR SELECT ……
```

2) Open a cursor:-

Once cursor is declared we can open it.

When cursor is opened following operations are performed.

a. Memory is allocated to store the data.
b. Execute SELECT statement associated with cursor
c. Create active data set by retrieving data from table
d. Set the cursor row pointer to point to first record in active data set.

Syntax:-

```
OPEN cursorname;
```

3) Fetching data:-

We cannot process selected row directly. We have to fetch column values of a row into memory variables. This is done by FETCH statement.

Syntax:-

```
FETCH NEXT FROM  cursorname INTO variable1, variable2……
```

4) Processing data:-

This step involves actual processing of current row.

5) Closing cursor:-

A cursor should be closed after the processing of data completes. Once you close the cursor it will release memory allocated for that cursor.

Syntax:-

```
CLOSE cursorname;
```

6) Deallocating cursor:-

It is used to delete a cursor and releases all resources used by cursor.

Syntax:-

```
DEALLOCATE cursorname;
```

Example 1:- Cursor to insert record from student table to student1 table if branch is CE.

```
DECLARE @rno int, @name varchar(50),@branch varchar(50);
DECLARE cursor_student CURSOR
        FOR SELECT rno, name, branch FROM student;
OPEN cursor_student;
        FETCH NEXT FROM cursor_student INTO @rno, @name, @branch;
        WHILE @@FETCH_STATUS = 0
BEGIN
        IF (@branch='CE')
                INSERT INTO student1 values (@rno, @name, @branch)
        FETCH NEXT FROM cursor_student INTO @rno, @name, @branch;
END;
CLOSE cursor_student;
DEALLOCATE cursor_student;
```

Example 2:- Cursor to update SPI (SPI=SPI-7) if SPI remains greater than or equal to ZERO after update.

```
DECLARE
   @rno int, @spi decimal(8,2);
DECLARE cursor_student CURSOR
FOR SELECT rno, spi
   FROM student;
OPEN cursor_student;
FETCH NEXT FROM cursor_student INTO
   @rno, @spi;
WHILE @@FETCH_STATUS = 0
   BEGIN
        set @spi=@spi-7
        if (@spi<0)
```

```
                print 'SPI must be greater than 0'
          else
                update student
                set spi=@spi
                where rno=@rno
      FETCH NEXT FROM cursor_student INTO
      @rno, @spi;
   END;
CLOSE cursor_student;
DEALLOCATE cursor_student;
```

Write a PL/SQL block to print the sum of Numbers from 1 to 50.

```
DECLARE
      @V1 INT,
      @SUM INT
      SET @V1 = 1
      SET @SUM = 0
WHILE (@V1 <= 50)
BEGIN
      SET @SUM = @SUM + @V1
      SET @V1 = @V1 + 1
END
      PRINT @SUM
```

Write a PL/SQL block to print the given number is Odd or Even.

```
DECLARE @V1 INT
      SET @V1 = 20
BEGIN
      IF @V1%2 = 0
          PRINT 'NO IS EVEN';
      ELSE
          PRINT ' NO IS ODD';
END
```

Write a PL/SQL block to print sum of even numbers between 1 and 20.

```
DECLARE
      @V1 INT,
      @SUM INT
      SET @V1 = 1
      SET @SUM = 0
```

```
WHILE (@V1 <= 20)
BEGIN
        IF @V1%2 = 0
          SET @SUM = @SUM + @V1
          SET @V1 = @V1 + 1
END
        PRINT @SUM
```

Write a PL/SQL program for inserting even numbers in EVEN table and odd number in ODD table from number 1 to 50.

```
DECLARE
        @V1 INT
        SET @V1=1
WHILE (@V1 <= 50)
BEGIN
        IF @V1%2=0
          BEGIN
                INSERT INTO EVEN (NO) VALUES (@V1);
          END
        ELSE
          BEGIN
                INSERT INTO ODD (NO) VALUES (@V1);
          END
SET @V1 = @V1 + 1
END
```