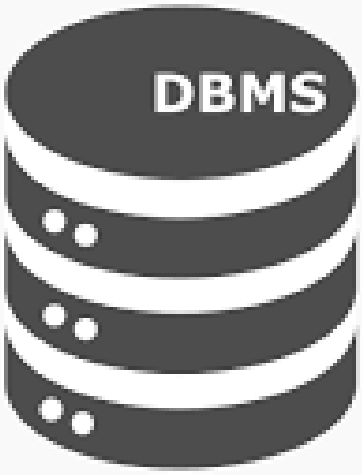


DBMS Handwritten Notes

Prepared By:

BTECH
NOTES

1. Introduction to Databases

1.1 Basic Concepts of databases

What is Database?

A database is an organized collection of structured data that is stored electronically and can be accessed, managed, and manipulated efficiently. It serves as a repository for storing and managing data for various applications.

In database, data is organized into tables i.e relations. Each table consists of rows (records) and columns (attributes) that represent different pieces of information

Data:

Data is a collection of facts and figures that can be processed to produce information.

Database Management System (DBMS)

Database management system is a collection of inter-related data and programs to access those data.

The challenges for data management:

Traditional file-based data management approache suffer from several challenges, such as data redundancy, inconsistency and isolation.

Data Redundancy: In file-based systems, data is often duplicated accross multiple files or applications. This

duplication. can lead to data inconsistency, as changes made to one copy of the data might not be reflected in other copies

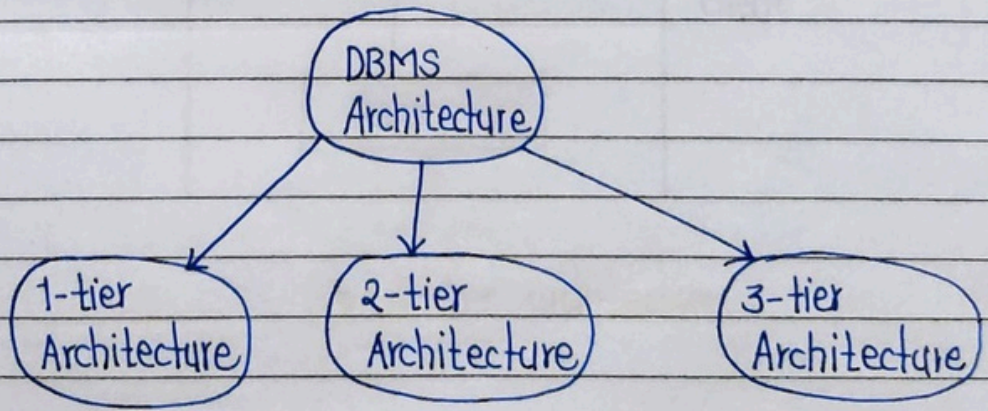
Data Inconsistency: When the same data is stored in different files or formats, it can lead to inconsistencies in the values and meanings of data.

Data isolation: File-based systems are often isolated from each other, making it difficult to share data across different applications or users.

1.2 Database Management System Architecture

The DBMS design depends upon its architecture. The basic client/servers architecture is used to deal with a large number of PCs, web servers, database servers and other components that are connected with networks.
The client/server architecture consists of many PCs and a workstation which are connected via the network.
DBMS architecture depends upon how users are connected to the database to get their request done.

Types of DBMS Architecture



Database architecture can be seen as a single tier or multi-tier But logically, database architecture is of two types like: 2-tier architecture and 3-tier architecture.

1- Tier Architecture

In this architecture, the database is directly available to the user - It means the user can directly sit on the DBMS and uses it

Any changes done here will directly be done on the database itself. It doesn't provide a handy toolennds user.

2- Tier Architecture :

The 2- Tier Architecture is same as basic client-server. In the two-tier architecture, applications on the client end can directly communicate with the database at the server side. For this interaction, API's like : ODBC, JDBC are use

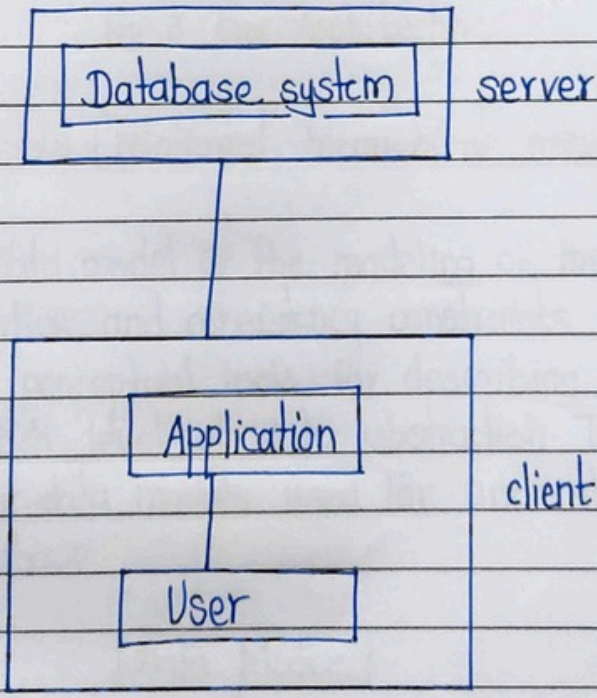


Fig : 2-tier architecture

3-Tier Architecture :

The 3-tier architecture contains layer between the client and server. In this architecture, client can't directly communicate with server.

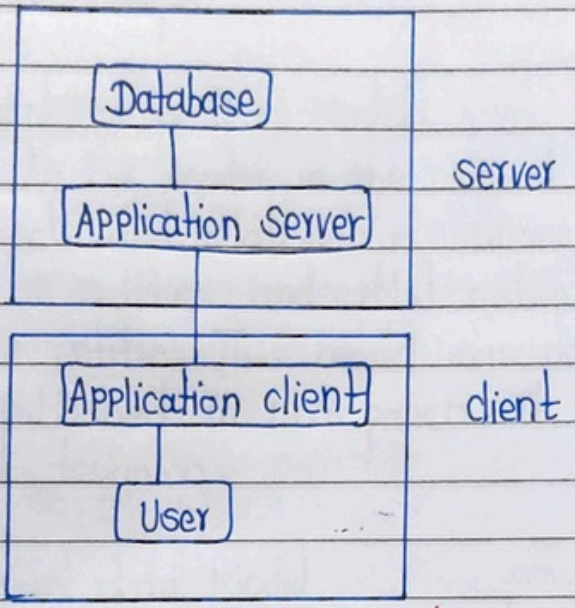
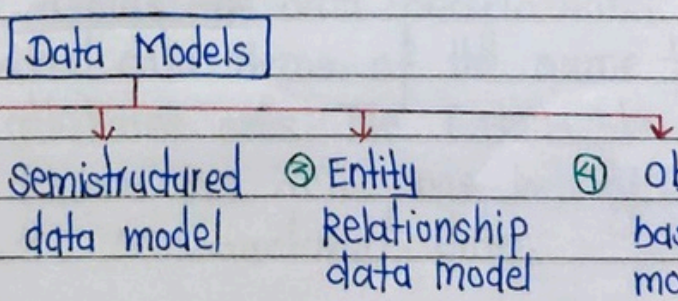


Fig 3-tier Architecture

1.3 Data Models (relational, hierarchical, network, etc)

Data model is the modeling of the data description. data semantics, and consistency constraints of the data. It provides the conceptual tools for describing the design of a database each level of data abstraction. Therefore, there are following four data models used for understanding the structure of the database.



1) Relational data Model:

This type of model designs the data in the form of rows and colums i.e table also called as relations. Relational model uses tables for representing data and in-between relationships.

2) Entity-Relationship Data Model:

An E-R model is the logical representation of data as objects and relationships among them. These objects are known as entities, and relationship is an association among these entities. This model was designed by peter chen and published in 1976 papers. It was widely used in database designing.

3) Object based data Model:

An extension of the ER model with notions of functions, encapsulation, and object identity, as well. This model supports a rich type system that includes structured and collection types. Thus, in 1980's various database systems following the object-oriented approach were developed.

4) Semistructured data Model:

This type of data Model is different from the other three data models (explained above). The semi-structured data model allows the data specifications at places where the individual data items of the same type may have different attributes sets. The Extensible Markup Language, also known as XML was initially designed for representing the semistructured data.

1.4 Advantages and disadvantages of using DBMS

- **Advantages of using DBMS:**

1. **Data integrity**: Databases enforce data integrity constraints, such as primary keys, foreign keys and unique constraints, to maintain accurate and consistent data. This ensures that data is reliable and free from inconsistencies.
2. **Data Security**: Databases offer various security features, such as access control, authentication, and encryption, to protect sensitive information from unauthorized access and data breaches.
3. **Data sharing**: Databases enables multiple users and appl^n to access and modify data concurrently. This facillitates collaboration and follows for centralized data management, leading to better data sharing across an organization.
4. **Data Independence**: Databases provide data independence, seprating the physical storage of data from the logical view. This abstraction allows changes to be made to the database structure without affecting the applications that use the data, making maintainance and scallability easier.
5. **Data Backup and Recovery**: Databases offer mechanisms for regular data backups and restore points, ensuring that data can be recovered in case of hardware failures, human errors.

- **Disadvantages of using DBMS**

1. **Cost**: Setting up and maintaining a database system can be expensive, especially for small-scale applications. Costs include hardware, software licences, training and ongoing maintainance.

2. Complexity: Designing and managing databases require specialized knowledge and skills. The complexity of database systems may be challenging for non-experts to understand & maintain.
3. Single point of failure: A database failure can affect the entire system. If proper backup and recovery mechanisms are not in place, data loss can occur, leading to significant disruptions in operations.
4. Scalability challenges: As data volumes grow, scaling databases to handle increased loads may become complex and costly. Ensuring high performance under heavy loads can be a challenge.

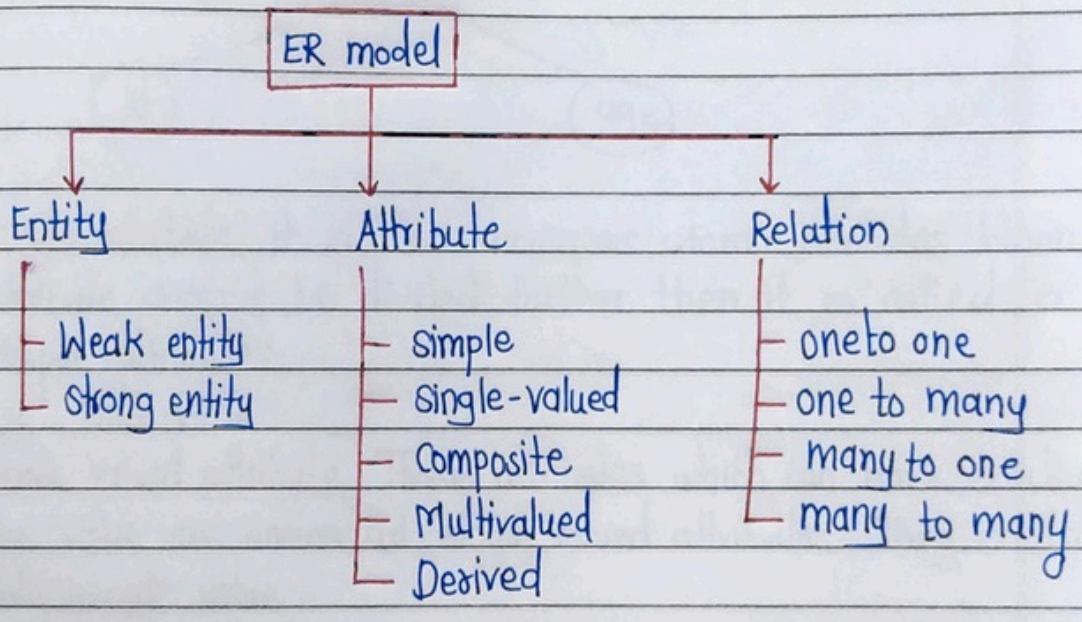
2. Entity-Relationship Model

2.1 Entities, attributes and relationships

ER model stands for an Entity-Relationship model. It is a high-level data model. This model is used to define the data elements and relationships for a specified system. It develops a conceptual design for the database. It also develops a very simple and easy design view of data.

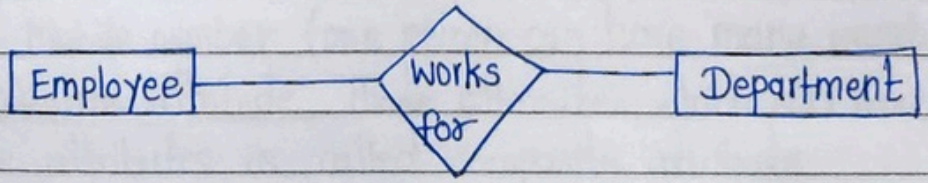
In ER modelling, the database structure is portrayed as a diagram called an entity relationship diagram.

Components of ER diagram:

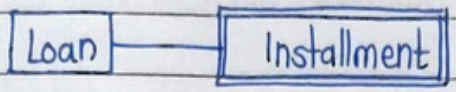


Entity: An entity is object, class, person or place. In the ER diagram, an entity can be represented as rectangles.

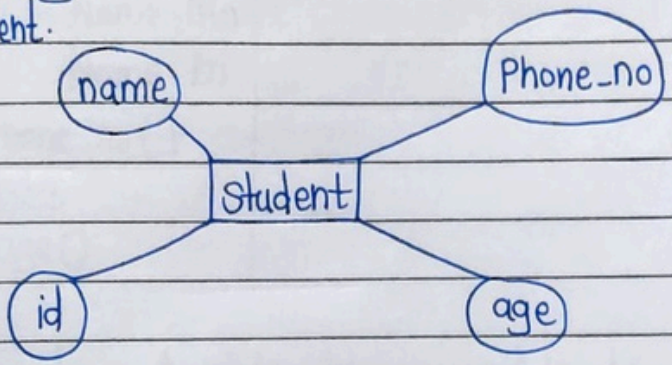
Ex:



a. Weak Entity: An entity that depends on another entity called a weak entity. The weak entity doesn't contain any key attribute of its own. The weak entity is represented by a double rectangle



2. Attribute: The attribute is used to describe the property of an entity. Eclipse is used to represent an attribute.
e.g id, age, contact numbert, name etc can be attributes of a student.



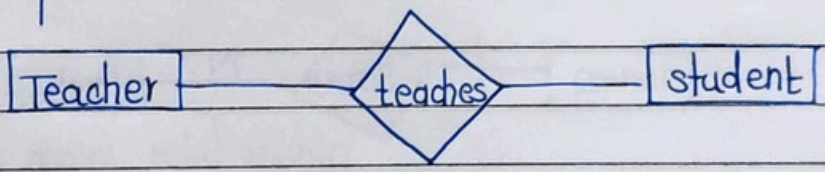
- Simple attribute: It is also known as atomic attributes. When an attribute cannot be divided further, then it is called a simple attribute:
  e.g First name
- Single-Valued attribute: Those attributes which can have exactly one value are known as single valued attributes. They contain only single value.
  e.g DOB - Birth date
- Multi-valued attribute: Those attributes which can have more than one entry or which contain more than one value are called multi-valued attributes.
  e.g Mobile number (one person can have many numbers)
- Composite attribute: Those attributes which can have more than one attributes is called composite attribute.

e.g name - First name, last name, middle name

- Derived attribute: The value of attribute derived from another attribute, those attributes are called Derived attribute.
e.g age() is derived attribute.

Student
student-id
Name
Name.Fn
Name.Mn
Name.Ln
Phone_no{}
DOB
age()

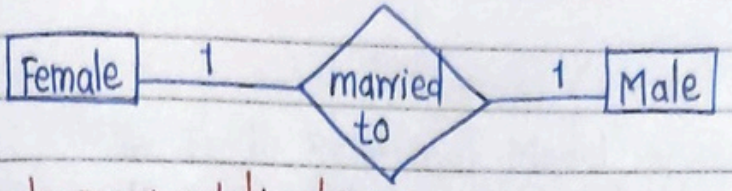
3. Relationship: A relationship is used to describe the relation between entities. Diomand or rhombus is used to represent the relationship.



Types of relationships are as follows:

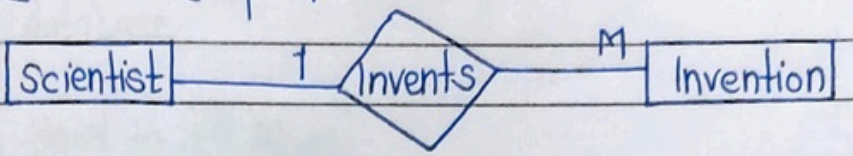
1. One to one Relationship (1-1)

When only one instance of an entity is associated with the relationship, then it is known as one to one relationship.
e.g A female can marry to one male, and a male can marry to one female.



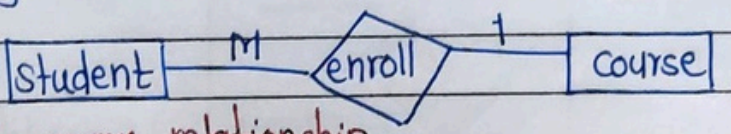
2. One to many relationship:

When one istance of the entity on the left, and more than one instance of an entity on the right associates with the relationship then its known as a one-to-many relationship.
e.g Scientist can invent many inventions, but the invention is done by the only specific scientist.



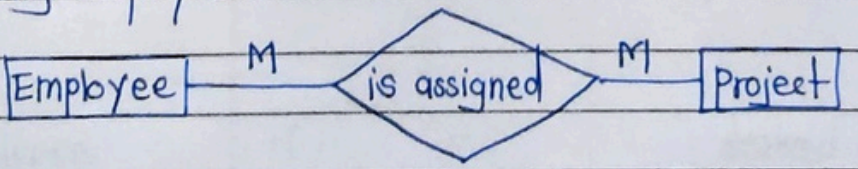
3. Many to one relationship:

When more than one instance of the entity on the left, and only one instance of an entity on the right associates with the relationship then it is known as a many-to-one relationship.
e.g Student enrolls for only one course, but a course can have many students.



4. Many to many relationship

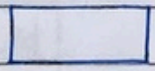
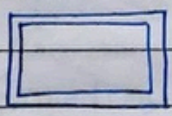
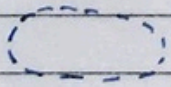
When more than one instance of the entity on the left, and more than one instance of an entity on the right associates with the relationship then its known as many-to many relationship
e.g Employee can assign by many projects and project can have many employees.



2.2 Entity - Relationship diagram (ERDs)

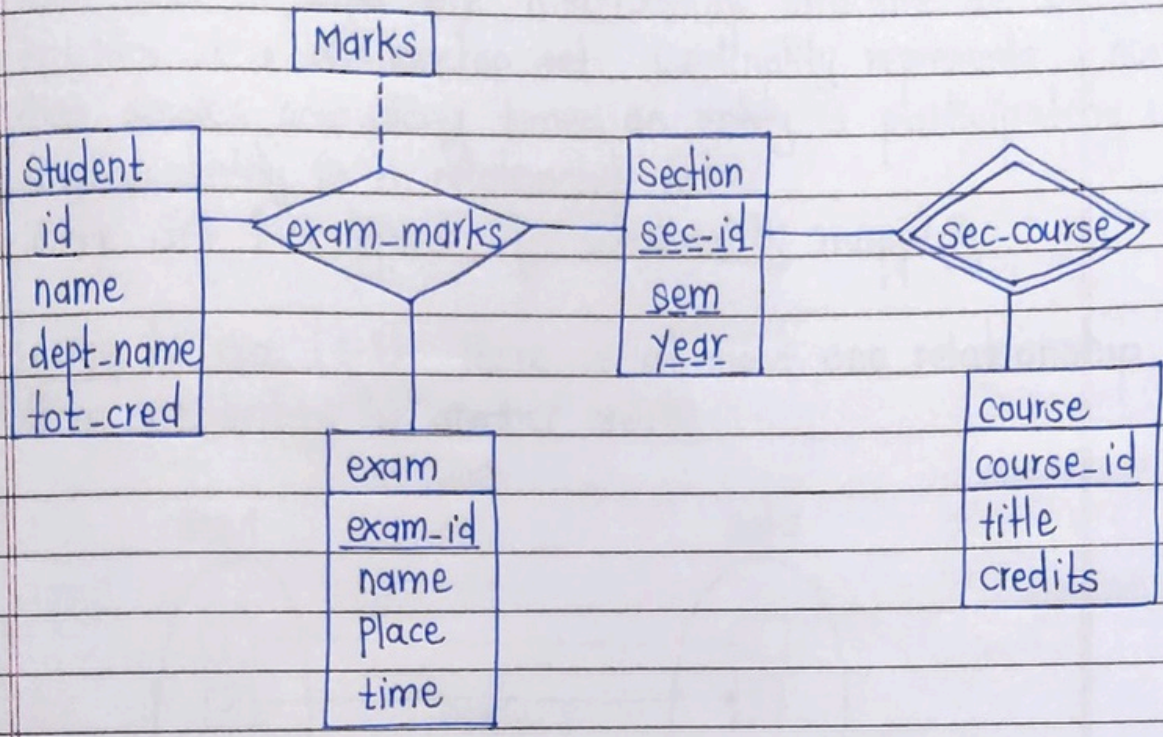
The Entity Relational Model is a model for identifying entities to be represented in the database and representation of how those entities are related. The Entity Relationship Diagram explains the relationship among the entities present in the database. ER models are used to model real-world objects like a person, a car, or a company and the relation between these real-world objects. ER diagram is the structural format of the database.

Symbols Used in ER Model :

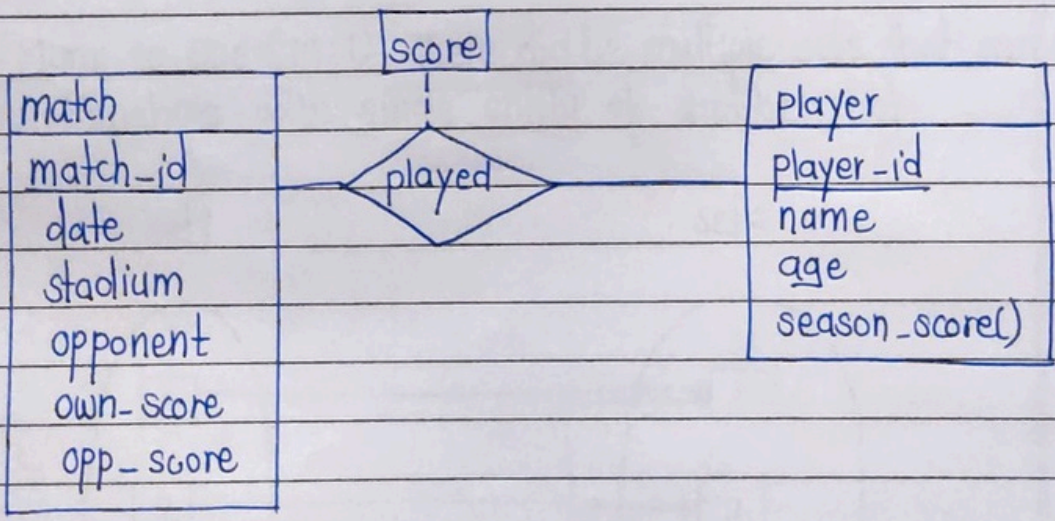
Rectangle		Entities in ER model
Ellipse		Attributes in ER model
Diamond		Relationships
Line		connection between entity and relationship
Weak entity		Double rectangle
Double Ellipse		Multi-valued attributes
underlined attribute	A	primary key
dotted ellipse		derived attribute

Some examples of ER diagram:

- ER Diagrams for marks database.



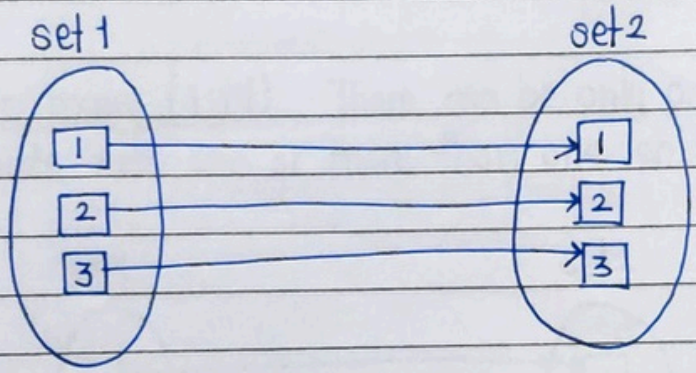
- ER diagram for Favourite team statistics



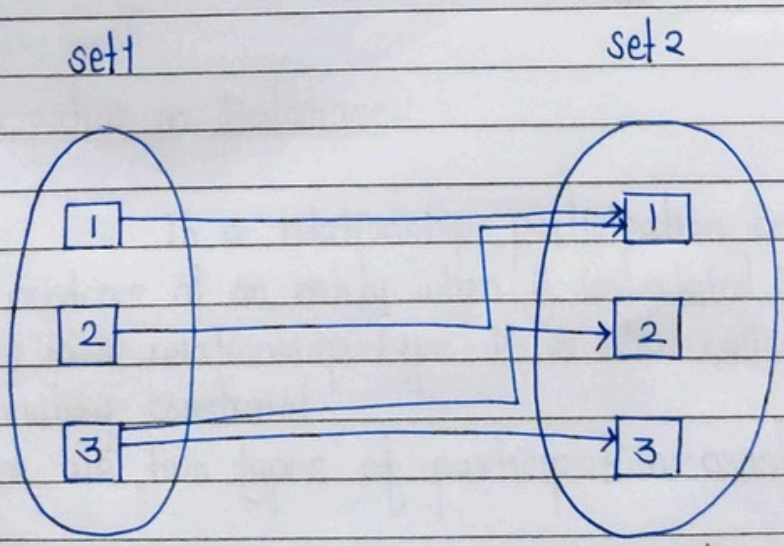
2.3 Cardinality and participation constraints.

Cardinality means how the entities are arranged to each other or what the relationship structure between entities in a relationship set. Cardinality represents a number that denotes how many times an entity is participating with another entity in a relationship set.
There are four types of Cardinality mapping:

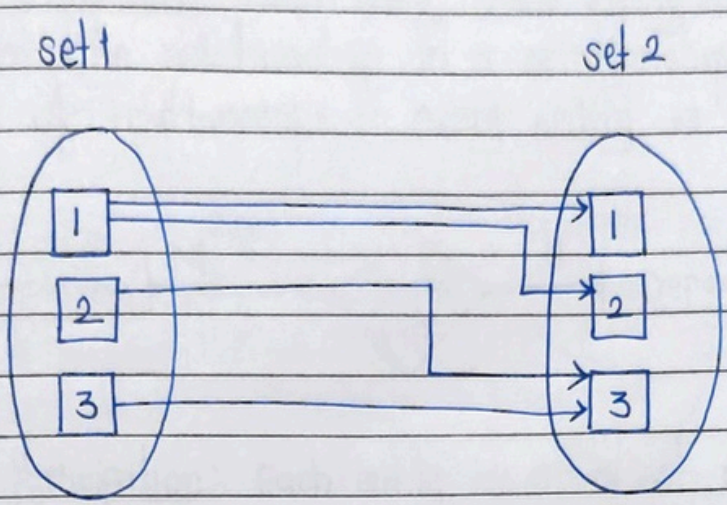
1. One to One (1:1) There is at most one relationship from one entity to another entity



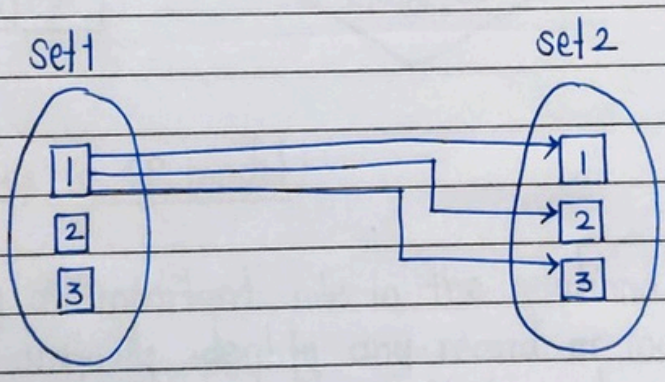
2. Many to One (M:1) There can be multiple sets that can make relationships with single entity of another set.



Many to One Many (M:M) : There can be one or more than one entity that can associate with one or more than one entity of another set.



One to many (1:M) There can be only one entity that can associate with one or more than one entity of another set.

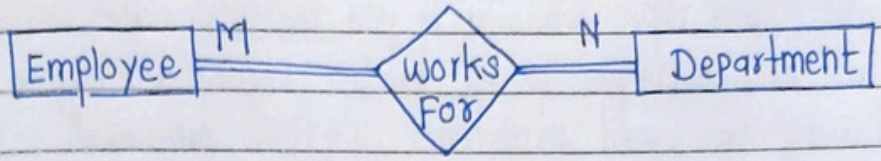


Participation in Database:

In a relationship, participation constraint specifies the existence of an entity when it is related to another entity in a relationship type. It is also called minimum cardinality constraint.
There are two types of participation constraint -

1) Total participation
2) Partial participation

Total Participation: Each entity in the entity set is involved in at least one relationship in a relationship set. i.e the number of relationships in every entity is involved is greater than 0.



Partial Participation: Each entity in entity set may or may not occur in at least one relationship in a relationship set.

2.4 keys in ER model:

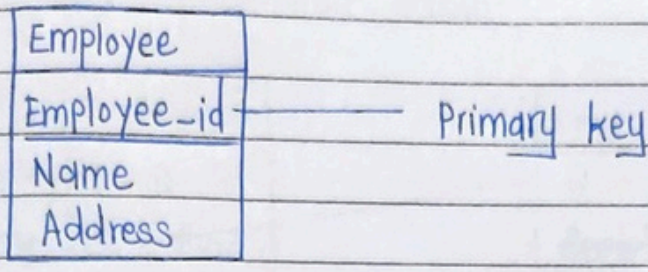
keys play an important role in the relational database. It is used to uniquely identify any record or row of data from the table. It is also used to establish and identify relationships between tables.

Primary keys Candidate key Super key Foreign keys

Primary keys: The candidate key chosen by database designer as a primary means it can be used to identify the entity in a entity set uniquely.

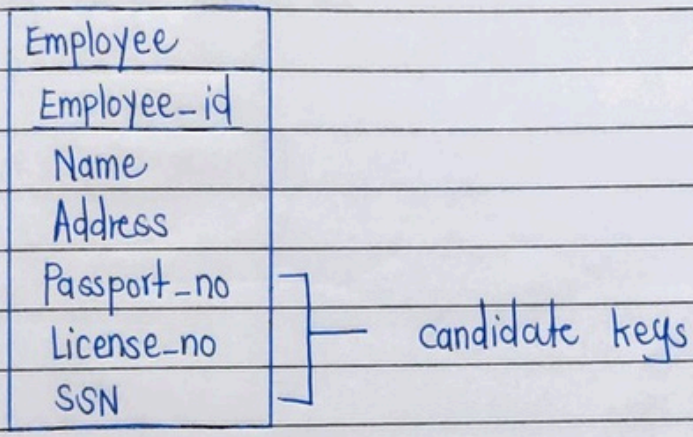
Primary keys are unique and not null.

e.g



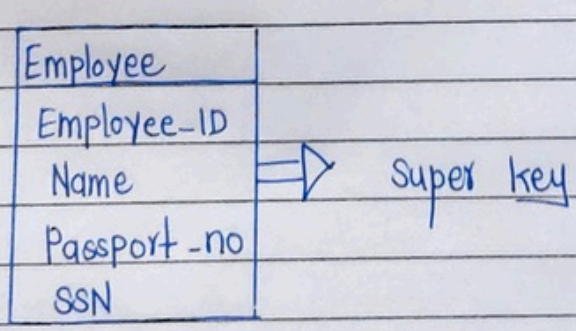
Candidate key: Candidate key is minimal super key. The super key for which no proper subset is a superkey is called candidate key. There are multiple candidate keys are possible.

e.g



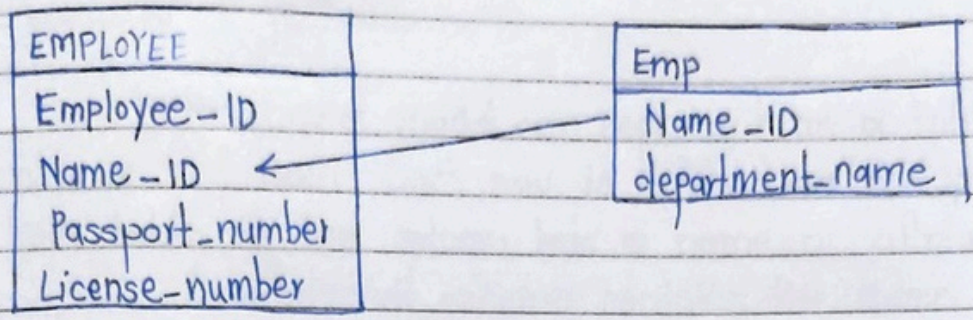
Super key: A attribute or set of attribute that uniquely identify the entity and in a entity set is called super key.

e.g



Foreign key:

Foreign keys are the column of the table used to point to the primary keys of another table.



In DBMS, super key, foreign key, candidate key and primary keys are used

3. Relational Data Model

3.1 Relational schema

Relational model can represent as a table with columns and rows. Each row is known as a tuple. Each take table of the column has a name or attribute.

. A relational schema contains the name of the relation and name of all columns or attributes.

In the relational key, each row has one or more attributes. It can identify the row in the relation uniquely.

Ex: student Relation

Name	Roll_No	Phone_No	Address	Age
Ram	14795	000007777	Noida	24
Shyam	12839	1111122222	delhi	25
Laxman	33289	222228888	delhi	23
Ganu	17282	1111111111	Mumbai	21

3.2 Relational Algebra:

Relational algebra is a theoretical query language used to manipulate data in relational databases. It consists of a set of operations that can be performed on relations (tables) to retrieve desired information. Here are some common relational algebra operations with an example:

1. selection (σ) : selects rows from a relation that satisfy a given condition.
2. Projection (π) : selects specific columns from a relation, discarding the rest.
3. Union ($\cup$) : Combines two relations to produce a new relation that contains all distinct rows from both relations.
4. Intersection ($\cap$) : Produces a new relation with rows that appear in both input relations.
5. Set difference ($-$) : Produces a new relation with rows that appear in the first relation but not in the second.

3.3 Tuple and domain relational calculus

A Tuple is a record and refers to a single row or record in a relational database table.

Ex:

Suppose we have a simple table "students" with the following attributes:

ID	Name	Age	Grade
1	John	20	A

Domain Relational Calculus (DRC):

Domain Relational Calculus is another non-procedural query language used to specify queries in relational databases. It focuses on specifying the conditions on attributes instead of specifying the tuples directly. DRC specifies the desired result by specifying conditions for each attribute.

3.4 SQL (structured Query Language) basics

SQL is structured query launguage is a powerful and widely used language for managing and manipulating relational databases. It provides a standardized way to interact with databases, enabling users to perform various operations, such as querying, inserting updating and deleting data.

Here are the SQL basics:

1) SELECT statement : Used to retreive data from a database.
syntax: ~~of~~ select col1, col2, ... from tablename;

2) WHERE clause: Used to filter data based on specific condition
syntax: SELECT col1, col2, ... From table WHERE condition;

3) INSERT INTO Statement: Used to insert new records into a database table.
syntax: INSERT INTO table(col1, col2, ...) VALUES (value1, value2...);

4) UPDATE statement: Used to modify existing records in a database table.
syntax: UPDATE table SET col1 = value1, col2 = value2, ... WHERE condition;

5) DELETE FROM statement: Used to delete existing records from a database table.
syntax: DELETE FROM table WHERE condition;

6) ORDER BY clause : Used to sort the result set based on one or more columns.

syntax: SELECT col1, col2, . . . FROM table ORDER BY col1 [ASC|DESC], column2 [ASC|DESC], ... ;

7) GROUP BY clause: Used to group rows based on the values in one or more columns

syntax: SELECT column1, aggregate_function (column2) From table_name GROUP BY column1;

4. Normalization

4.1 Functional dependencies:

Functional dependencies are a fundamental concept in database design and normalization. They define the relationships between attributes (columns) in a relation (table). A functional dependency represents a rule or constraint that states that the value of one attribute uniquely determines the value of one another attribute.

Functional dependency $A \rightarrow B$

Here A is determinant and B is determined.

Examples of functional dependecies:

1) Student_ID → Name
This dependency means that knowing the student ID uniquely determines the name of the student.

2) student_ID → Age
knowing the student_ID also uniquely determines the Age of the student.

3) student_ID, Grade → Address
This dependency indicates that the combination of student_ID and Grade uniquely determines the address of the student.

4) student_ID → Class

4.2 First, second, Third Normal Forms (1NF, 2NF, 3NF)

First Normal Form (1NF):

First Normal Form (1NF) is the fundamental level of normalization in database design. To achieve 1NF, a relation (table) must meet the following requirements:

- Each attribute (column) in the table must contain atomic values, meaning that the values are indivisible and cannot be further broken down.
- The values in each attribute must be of the same datatype
- Each attribute must have a unique name.
- The order of tuples (rows) in not significant in a relation.

1NF ensures that a relation is free from repeating groups and multivalued attributes, leading to a more organized and efficient data structure.

Second Normal Form (2NF):

Second Normal form (2NF) builds on the requirements of 1NF and introduces the concept of partial dependencies. A relation is in 2NF if it meets the following conditions:

- It is already in 1NF.
- It does not have any partial dependencies, which means that all non-key attributes must depend fully on the entire primary key.

To achieve 2NF, you need to identify composite primary keys and seprate them into multiple relations, each with its own primary key and related attributes. This step helps eliminate redundancy and improve data intogaity.

Third Normal Form (3NF)

Third Normal Form (3NF) extends the concept of 2NF and addresses transitive. A relation is in 3NF if it meets the following conditions:

- It is already in 2NF.
- It does not have any transitive dependencies, meaning that no non primary key attribute depends on another non-key attribute.

To achieve 3NF, you need to remove transitive dependencies by breaking the relation into multiple smaller relations, ensuring that each non-key attribute depends only on the primary key and not on non-key attribute.

4.3 Boyce-codd Normal Form (BCNF)

Boyce-codd Normal Form (BCNF) is an relation or extension of 3NF and addresses situations where a relation may have multiple candidate keys. A relation is in BCNF if it meets the following condition:

- It is already in 3NF.
- For every non-trivial Functional dependency ($X \rightarrow Y$), where X is a superkey (a candidate), Y is a subset of X. In simpler terms, each non-key attribute is fully fuctionally dependent on the primary key.

BCNF eliminates anomalies caused by overlapping candidate keys and ensures that all dependencies are directly related to the primary key(s).

BCNF — Boyce-codd Normal Form.

4.4 Multivalued Dependencies and Fourth Normal Form (4NF)

Fourth Normal Form (4NF) addresses multivalued dependencies, which occur when an attribute depends on multip independent sets of values. A relation is in 4NF if it meets th following conditions:

- It is already in BCNF
- It does not have any non-trivial multivalued dependencies.

To achieve 4NF, you need to split the relation into multiple smaller relations, each with its own primary key, to eliminate multivalued dependencies.

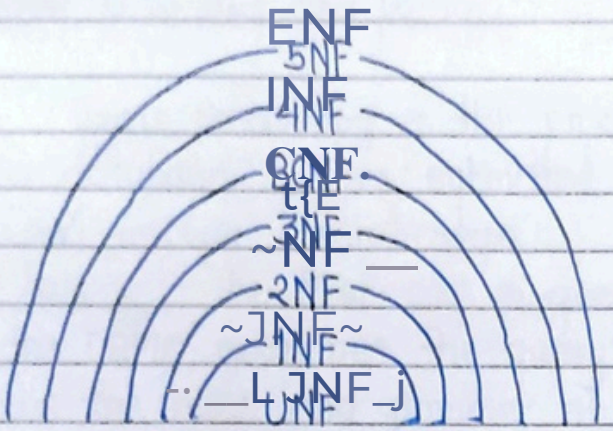
Fifth Normal Form (5NF), also known as PJNF (project-Join Normal form), addresses join dependencies which occur when a relation contains multiple candidate keys that overlap in their attributes. A relations is in 5NF if it meets the following requirements:

- It is already in 4NF.
- It does not have any non-trivial join dependencies.

To achive 5NF, you need to decompose the relation into multiple smaller relations, each representing a single join dependency. This step ensures that the relation is free from redundant information and maintains data consistency.

1NF, 2NF, 3NF, BCNF, 4NF and 5NF are different types of normal forms.

- Different types of Normal Forms



5. Query Optimization and Execution

5.1 Query processing and parsing:

Query processing is the process of interpreting and executing database queries submitted by users or applications. The process involves several steps:

1. Query Parsing :- The first step in query processing is parsing, where the DBMS examines the query's syntax to ensure it follows the rules and grammer of the query language (e.g SQL). The query is then transformed into a parse tree or an internal representation that the DBMS can understand.

2. Semantic Analysis: The DBMS performs semantic analysis to ensure that the query refers to valid database objects and attributes. It checks the query's compatibility with the database schema and resolves any ambiguities.

3. Query optimization: Once the query is parsed and validated, the DBMS generates multiple possible execution plans for the query.

4. Query Execution: After query optimization, the DBMS selects the most efficient execution plan and executes the query. The data is retrieved, processed, and returned as the final results set.

5.2 Query Optimization Techniques

Query optimization is a crucial aspect of

database management, as it directly impacts the efficiency and performance of queries. Various techniques are used to optimized queries:

1. Cost-Based Optimization: The DBMS estimates the cost of executing different query plans based on factors like the number of disk accesses, CPU usage, and memory requirements. The optimizer then selects the plan with the lowest estimated cost.

2. Join Algorithms: The choice of join algorithm (e.g nested loop join, hash join, merge join) depends on the size of the tables involved in the join operation. The optimizer selects the most appropriate join algorithm to minimize processing time.

3. Index Selection: The DBMS uses indexes to spend/speed up data retrieval. The optimizer decides which indexes to use based on the query's filtering conditions and the selectivity

4. Materiallized Views: Materiallized views are preconnected query results stored as physical tables.

5.3 Indexing and Hashing for performance Improvement

Indexing:

Indexes are data structures that allow the DBMS to locate rows quickly based on the values in one or more columns. They act as pointers to the actual data, enabling efficient data retrieval for queries involving those columns.

Hashing: Hashing is a technique that converts a value into a fixed-size hash code, which is used to locate data quickly. Hashing is useful for equality-based queries where the goal is to find a specific row based on a unique key value.

5.4 Execution plans and cost Estimation:

Execution plans represent the step-by-step process of how the DBMS will retrieve and process data to execute a query. Each query can have multiple possible execution plans and the DBMS optimizer selects the plan with the lowest estimated cost.

Cost estimation involves estimating the resources (such as CPU, memory, and disk I/O) needed to execute each step in the execution plan. The optimizer uses these cost estimates to compare different plans and choose the most efficient one.

By selecting the optimal execution plan, the DBMS ensures that queries are executed as efficiently as possible, resulting in faster response times and better overall database performance.

6. Transactions and Concurrency Control

6.1 Acid properties of Transactions:

ACID is an acronym that stands for Atomicity, Consistency, Isolation and Durability. These properties ensure that database transactions are reliable and maintain data integrity. Each transaction in a database management system must adhere to the following ACID propesties:

- Atomicity: Atomicity ensures that a transaction is treated as a single unit of work that is executed completely or not at all. If any part of the transaction fails, the entire transaction is rolled back (undone), and the database returns to its original state before the transaction started

- Consistency: Consistency ensures that a transaction brings the database from one valid state to another valid stage. The database must satisfy all integrity constraints and rules during and after the execution of a transaction.

- Isolation: Isolation ensures that the changes made by one transaction are isolated from the changes made by other transactions. Each transaction executes independently without interference from other concurrent transactions.

- Durability: Durability ensures that the changes made by a commited transaction are permanent and survive system failures, such as power outages or crashes. Once a transaction is committed, its effects on the database.

6.2 Schedules and serializability:

A schedule is a chronological order of transactions executed in a database system. It represents the sequence in which transactions access and modify data. A schedule is considered serializable if its outcome is equivalent to the execution of transactions in some serial (one after the other) order.

Serializability ensures that the database state remains consistent and valid, even when multiple transactions execute concurrently. A serializable schedule prevents conflicts and ensures that the final state of the database is the same as if the transactions had been executed in a strict, non-overlapping sequence.

6.3 Concurrency Control Techniques

Concurrency control Techniques are used to manage the concurrent execution of multiple transactions to prevent conflicts and maintain data consistency. Some common concurrency control techniques include:

- Locking: Lock-based Concurrency control involves using locks (e.g read locks and write locks) to control access to data items. A transaction must acquire the appropriate data locks before reading or modifying a data item. This prevents conflicts between transactions.

- Timestampping: Timestamp-based concurrency control assigns a unique timestamp to each transactions based on its start time.

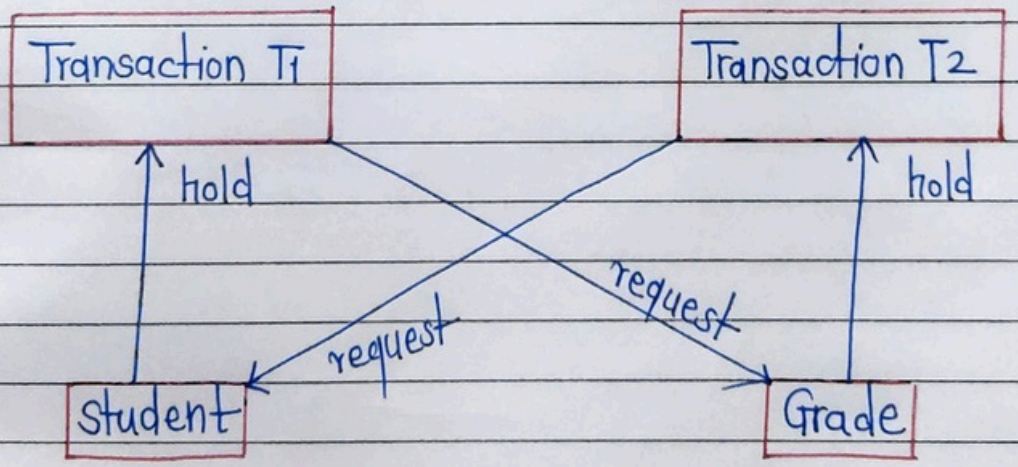
Concurrency Control Protocol

- Lock based Protocol
  - Shared lock
  - Exclusive lock
- Timestamp Based protocol
  - Basic Timestamp ordering
  - strict Timestamp ordering
  - Thoma's Write rule

6.4 Deadlocks and Their preventions.

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

e.g:



• Concept of deadlock

When a database is stuck in a deadlock, it is always better to avoid the deadlock rather than restarting or aborting the database. Deadlock avoidance method is suitable for smaller database whereas deadlock prevention method is suitable for longer database.

One method of avoiding deadlock is using application consistent logic. In the above given example, Transaction that accees students and Grades should always access the tables in the same order. In this way, in the scenario described above transtaction T1 waits for Grade.

7. Crash Recovery and Backup

7.1 Recovery Techniques (Undo, Redo, Logging)

Recovery techniques are essential for ensuring the database's durability and consistency in the event of system failures or crashes. The three primary recovery techniques used in database management systems are undo, Redo and Logging.

1. Undo: The Undo technique is used to rollback or undo the changes made by an incomplete or aborted transaction. When a transaction is rolled back, all its changes are reverted to their original state, effectively cancelling the the transactions effects.

2. Redo: The redo technique is used to reapply the changes made by a committed transaction that might have been lost due to a system crash. It ensures that the changes made by committed transactions are re-executed to bring the database back to a consistent state

3. Logging: Logging is a mechanism used to record all changes made by transactions in a log file before they are applied to the database. The log file acts as a record of transactions, including their actions (e.g updates, inserts, deletes) and the old new values of affected data items. In the event of a crash, the log file can be used to replay the transactions and apply their changes to recover the database.

7.2 Recovery Manager and Buffer Management:

The recovery Manager is a component of the Database Management System (DBMS) responsible for handling the recovery process. It ensures that the database remains in a consistent state even after system failures. The recovery manager is responsible for coordinating the Undo, Redo and Logging techniques to achieve database recovery

Buffer Management is another critical component of DBMS that deals with data storage and retrieval. It manages the buffer pool, which is a region of memory used to cache frequently accessed data pages. The Buffer manager reduces the need for physical disk I/O operations, improving the system's performance.

7.3 Backup and Restore Procedures:

Backup and restore procedures are circular for ensuring data safety and disaster recovery. Regular backups create a copy of the entire database or specific data subsets In the event of data loss or curruption, backups can be used to restore the database to a previous state.

1. Backup: Backups can be full, Incremental or differential

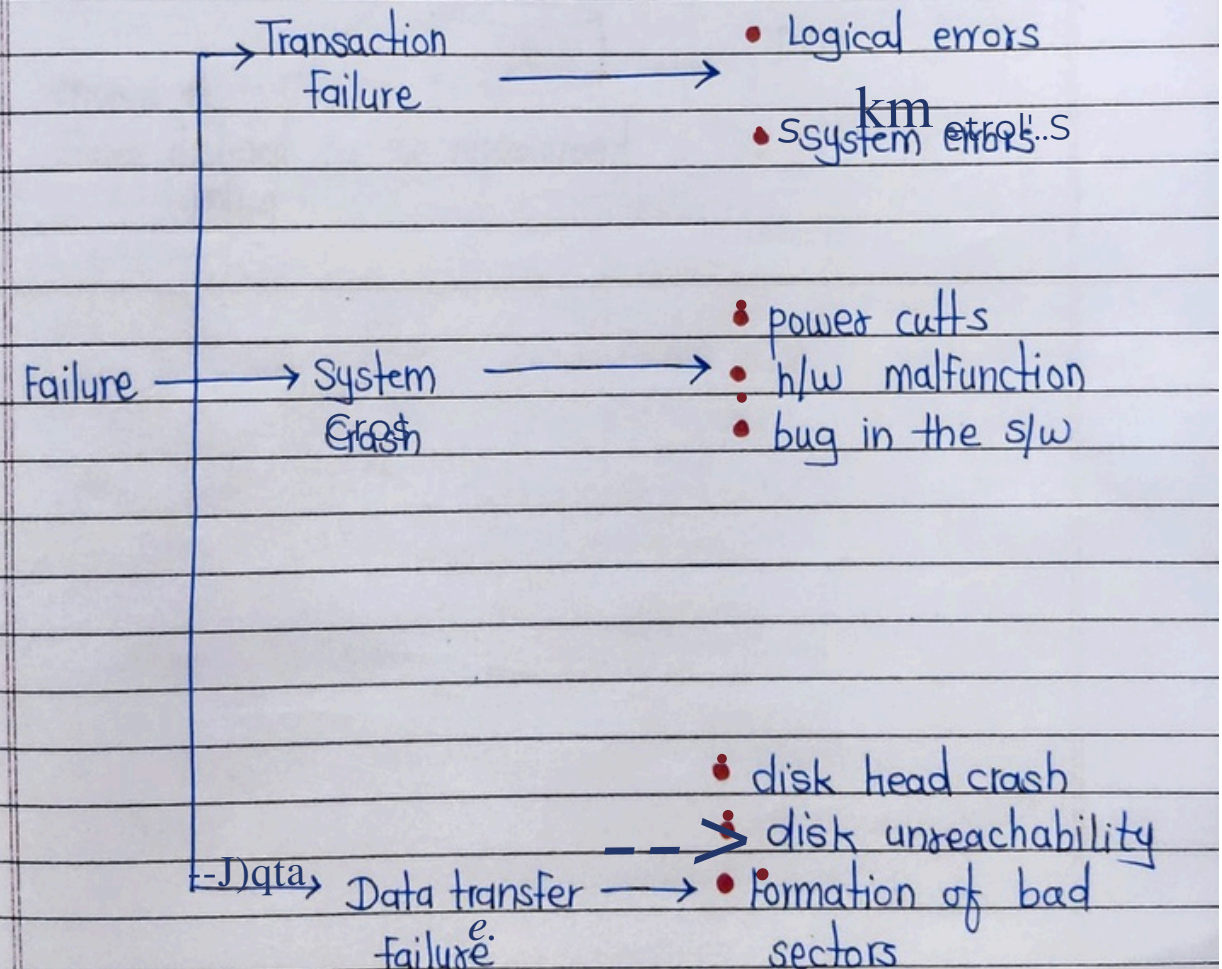
Full Backup: A full backup copies the entire database, providing a complete snapshot of the data at a specific point in time. Full Backups are time-consuming and may

schedule less frequently.

- Incremental Backup: Incremental backups only copy data that has changed since the last backup, reducing the backup time.

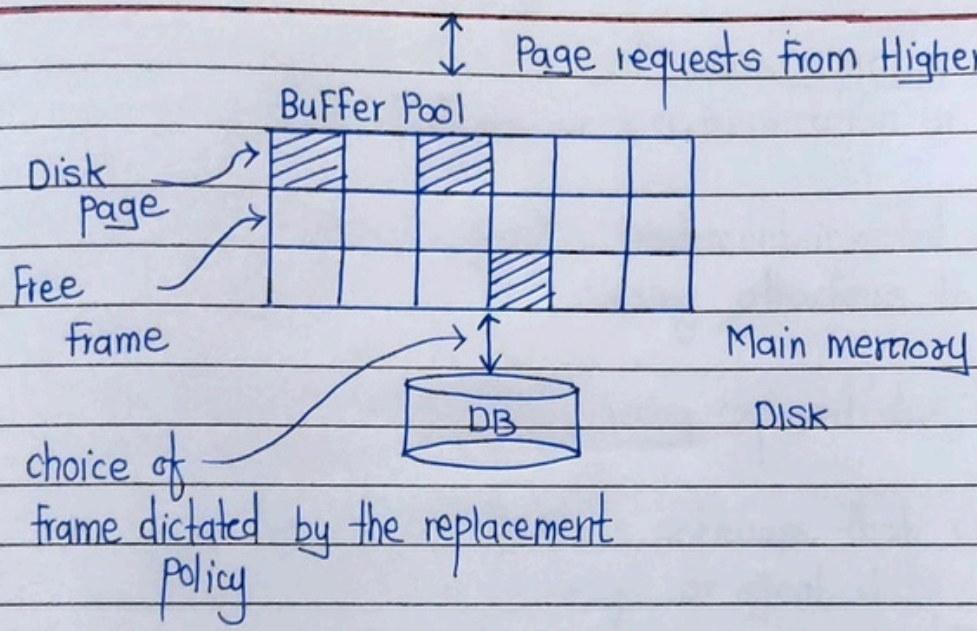
2. Restore: In case of data loss or corruption, a database can be restored using the latest available backup and transaction logs. The restore process involves applying the transaction logs. The restore using the latest available.

- Failure classifications in DBMS.



• Buffer management in DBMS •

- Data must be in RAM for DBMS to operate on its!
  - Can't keep all the DBMS pages in main memory
- Buffer management : Efficiently uses main memory
  - Memory divided into buffer frames : slots for holding disk pages.



8. Data Security and authorization

8.1 Security Threats and Countermeasures:

Security threats to a database can compromise its confidentiality, integrity and availability. Some common threats include:

Unauthorized Access: Unauthorized users gaining access to the database, either through weak authentication or exploiting vulnerabilities.
SQL injection: Malicious SQL statement injected into web forms or other input fields, allowing attackers to manipulate the database.
Data Breaches: Sensitive Data being exposed due to improper access control or data leaks.
Malware and Viruses: Malicious software that can infect the database system and corrupt or steal data.
Insider Threats: Malicious actions or unintentional mistakes by authorized users that can cause Harm to the database.

Countermeasures to these threats include implementing strong authentication mechanisms, regularly updating security patches, encrypting sensitive data, implementing intrusion detection systems and conducting security audits.

8.2 User authentication and Authorization:

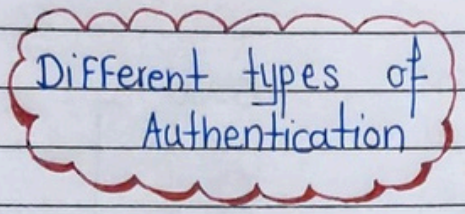
User authentication and authorization are

crucial aspects of database security:

1. User Authentication

User Authentication verifies the identity of a user attempting to access the database. It ensures that only legitimate users with valid credentials are allowed access.

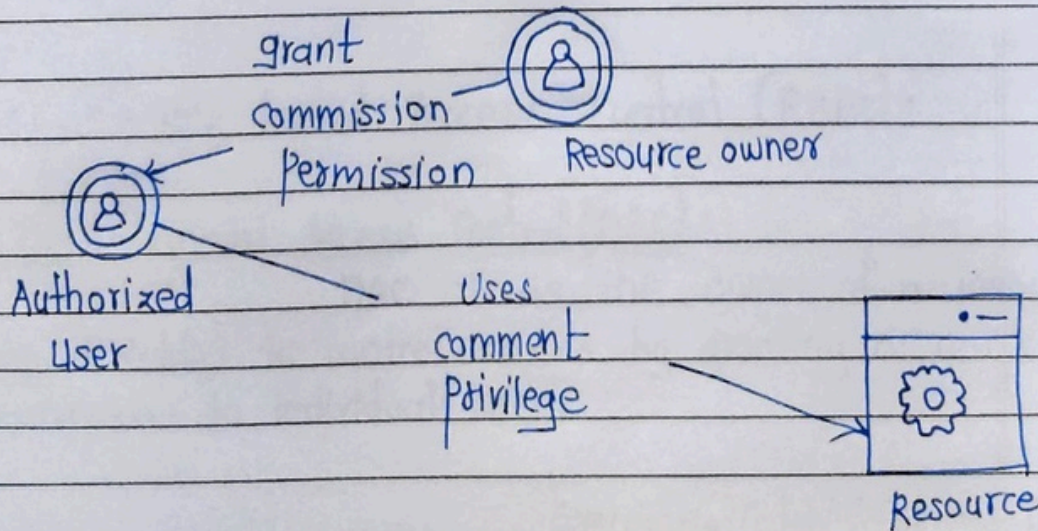
Common authentication methods include username/password, biometrics, token or MFA



① Password based ② Multi-Factor based ③ Biometric -based ④ Certificate based ⑤ Token -based

2. User Authorization:

User authorization determines the level of access and permissions granted to authenticated users Each user an role is assigned specific privileges, specifying what actions they can perform on the database.

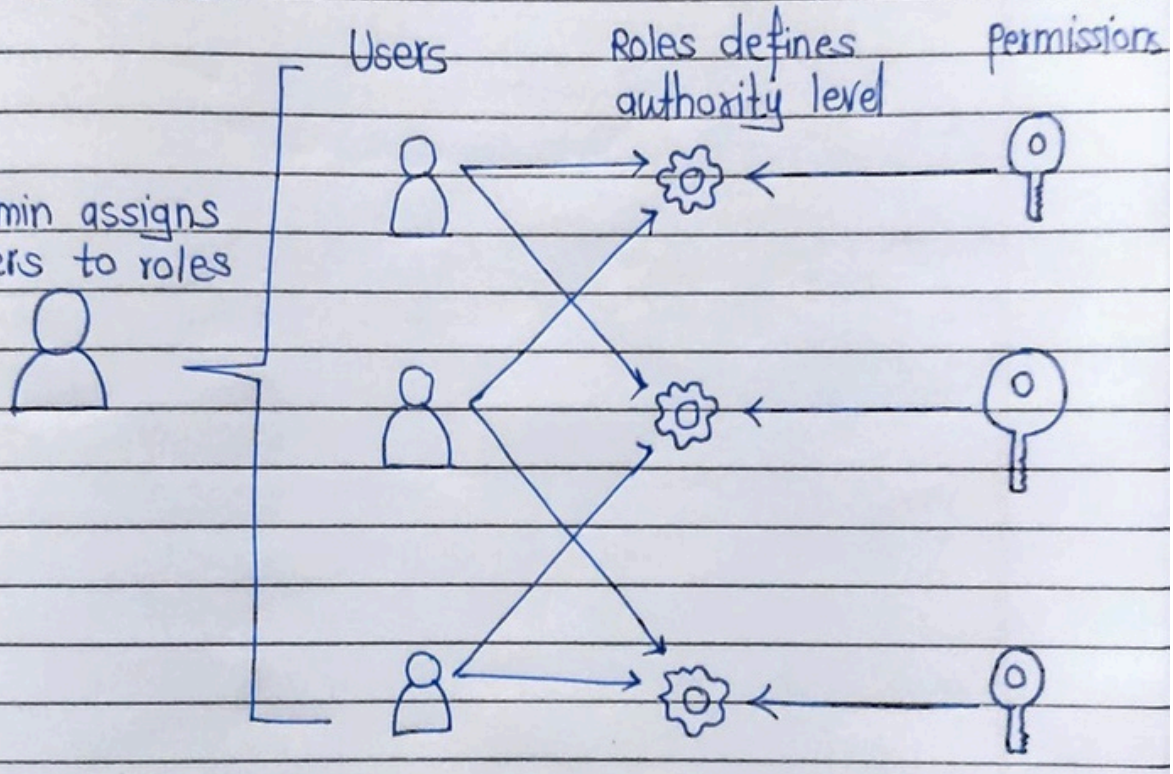


8.3 Access control and privileges

Access control mechanisms manage your user privileges to maintain data security:

1. Role Based Access Control (RBAC):

RBAC assigns roles to user based on their job responsibilities. Each role is associated with specific privileges, making it easier to manage access control for user bases.



• Role based Access Control (RBAC).

2. Discretionary Access Control (DAC):

DAC allows the owner of a resource (eg a table) to control access by granting or revoking permissions to individual users.

3. Mandatory Access Control

Mac enforces security policies set by system administrators or security officers. Users do not have control over their accesses privileges.

9. Distributed Databases

9.1 Concepts of Distributed Databases:

A distributed database is a collection of multiple interconnected databases spread across different geographical locations or computer systems. It allows data to be stored, processed, and accessed in a distributed and decentralized manner. Distributed databases offer several advantages, including improved performance, fault tolerance, and scalability. The main concepts of distributed databases include:

1. Distribution:

Users and applications interact with the distributed database as if it were a single, centralized database. Distribution transparency ensures that users are unaware of underlying distribution of data.

2. Data Fragmentation:

Data Fragmentation involves dividing a database's tables into smaller, manageable pieces called fragments. These fragments are distributed across multiple nodes in the network.

3. Data Replication:

Data replication involves maintaining multiple copies of same data in different locations. Replication enhances data availability and fault tolerance. However, it introduces challenges to maintain data consistency

across replicas.

4. Data Allocation:

Data allocation determines which data fragments are stored on which nodes in the distributed system. Efficient data allocation strategies are crucial to achieve balanced load distribution and optimal query performance.

9.2 Data Fragmentation, Replication, and Allocation:

1. Data Fragmentation:

Data Fragmentation involves dividing a table or relation into smaller fragments, which can be stored on different nodes in the distributed system. Fragementation can be vertical (splitting attributes) or horizontal (spliting rows). Different fragmentation techniques such as range, hash or round-robin. are used based on the data. distribution and access patterns.

2. Data Replication:

Data replication involves maintaing multiple copies of data in different locations. Replication improves data availability and fault tolerance. However it requires mechanisms to ensure data consistency among replicas, such as using protocol like two-phase commit and three-phase commite.

9.3 Distributed Query Processing and Optimization

Distributed query processing involves executing queries that involves data from multiple nodes in a

10. NoSQL Databases

10.1 Overview of NoSQL Databases:

NoSQL (Not Only SQL) databases are a category of databases that provide a flexible and scalable alternative to traditional relational databases. Unlike relational databases, which follow a fixed schema and use SQL for querying, NoSQL databases use a variety of data models and data storage techniques. They are designed to handle large volumes of unstructured or semi-structured data and provide high performance and horizontal scalability. NoSQL databases are commonly used in modern web applications, big data processing, and real-time analytics. Some of the key characteristics of NoSQL databases include:

1. Schema Flexibility:

NoSQL databases do not require a pre-defined schema, allowing developers to add new fields or change the data structure without disrupting existing data.

2. Horizontal Scalability:

NoSQL databases are designed to scale horizontally, meaning they can handle increased data and traffic by adding more servers or nodes to the database cluster.

3. High availability:

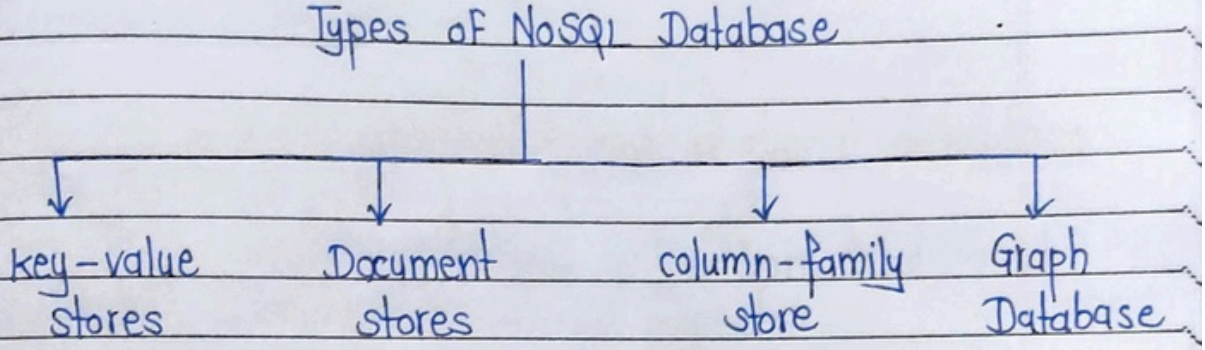
Many NOSQL databases offer automatically data replication and fault-tolerance features to ensure

high availability and data redundancy.

4. Distributed Architecture:

NoSQL databases often operate in a distributed manner across multiple nodes, enabling efficient data distribution and processing.

10.3 Types of NoSQL Databases:



key-value stores:

These databases store data as a collection of key-value pairs, where each key is unique, and the value can be any data structure, such as strings, numbers or even complex objects.

Document stores:

Document-oriented databases store data as JSON or BSON documents which can contain nested data structures. Documents within the same collection do not have to follow a consistent schema.

Column-Family stores:

These databases store data in columns

rather than rows, making them suitable for applications that require fast read and write operations on large datasets.

Graph databases:

Graph databases are designed to represent and store complex relationships between entities as nodes and edges, making them ideal for applications involving social networks, recommendation engines and complex hierarchical data.

10.3 Comparison with traditional relational databases:

NoSQL databases differ from traditional relational databases in various aspects:

1. Schema Flexibility: Traditional relational databases require a fixed schema, meaning the data structure must be predefined before data insertion. NoSQL databases offer schema flexibility, enabling easy and dynamic changes to the data structure.

2. Scalability: NoSQL databases are inherently designed for horizontal scalability, allowing them to handle massive amounts of data and high concurrent user requests more efficiently than relational databases.

3. Data Models: Relational databases use tables with rows and columns whereas NoSQL databases use various data models like key-value, document, column family, and graph to

accomodate different data types and use cases.

ACID properties: Relational databases strictly follow ACID (Atomicity, consistency, Isolation, Durability) properties to ensure data integrity and consistency.

Query Language: Traditional relational databases use SQL for querying and data manipulation, whereas NoSQL databases may use different query language.

NoSQL databases are particularly well-suited for applications that require high performance, massive scalability, and the able to handle semi-structured or unstructured data.

11. Big data and Data Warehousing

11.1 Introduction to Big data and its challenges:

Big data refers to large and complex datasets that are difficult to process and manage using traditional database management systems. These datasets typically have three defining characteristics known as the three V's

Volume: - Big data involves massive volumes of data that exceed the storage and processing capabilities of conventional databases.

Velocity : Big data is generated and collected at a high velocity often in real-time or near-real-time, requiring fast data ingestion and processing.

Variety : Big data comes in various formats and types, including structured, semi-structured, and unstructured data such as text, images, audio and video

Managing big data poses several challenges:

storage: storing and managing massive volumes of data requires scalable and distributed storage solutions.

Processing: processing large datasets in a timely manner requires powerful distributed computing and parallel processing techniques.

Analysis : Extracting valuable insights from big data

requires advanced data analytics and machine learning algorithms.

Data Quality: Ensuring data quality is challenging when dealing with diverse and heterogenous data sources.

11.2 Data warehousing Concepts and Architecture:

Data warehousing is a structured approach to storing and managing data from multiple sources in a centralized repository. A data warehouse is a large, integrated database that supports decision-making processes by providing comprehensive and historical view of the organization's data. key concepts and components of data. key concepts and components of data warehousing include:

1. ETL (Extract, Transform, Load)

The ETL process is used to extract data from various source systems, transform it into a consistent format, and load it into the data warehouse. This process ensures data integrity and consistency in the data warehouse

2. Data Marts:

Data Marts are smaller, specialized subsets of the data warehouse that cater to specific business units or departments. They provide focused and pre-aggregated data for faster query performance

3. Dimensional Modeling:

Dimensional modeling is a design

techniques used to organize data in a data warehouse.

Star and Snowflake schema:

Star and snowflake schemas are common dimensional modeling techniques

11.3 Extract, Transform, Load (ETL) processes:

ETL is a critical process in data warehousing that involves the following steps:

Extract: Data is extracted from various source systems, such as databases, applications, flat files, or APIs. Extraction methods can be full or incremental, depending on the data volume and update frequency.

Transform: Extracted data is transformed to ensure consistency and compatibility with the data warehouse's structure. Data cleansing, validation, aggregation and normalization are common transformation tasks.

Load: The transformed data is loaded into the data warehouse and datamarts. This process may involve inserting, updating, or deleting records in the data warehouse.

ETL processes are essential for ensuring the accuracy, consistency and completeness of data ~~ware~~ in the data warehouse, enabling organizations to perform effective data analysis and make informed


BTECH
NOTES


BTECH
NOTES


BTECH
NOTES